O'REILLY®

Second Edition

Zero Trust Networks

Building Secure Systems in Untrusted Networks

Razi Rais, Christina Morillo,
Evan Gilman & Doug Barth

O'REILLY®

Zero Trust Networks

This practical book provides a detailed explanation of the zero trust security model. Zero trust is a security paradigm shift that eliminates the concept of traditional perimeter-based security and requires you to "always assume breach" and "never trust but always verify." The updated edition offers more scenarios, real-world examples, and in-depth explanations of key concepts to help you fully comprehend the zero trust security architecture.

- **Examine fundamental concepts of zero trust security model, including trust engine, policy engine, and context aware agents**
- **Understand how this model embeds security within the system's operation, with guided scenarios at the end of each chapter**
- **Migrate from a perimeter-based network to a zero trust network in production**
- **Explore case studies that provide insights into organizations' zero trust journeys**
- **Learn about the various zero trust architectures, standards, and frameworks developed by NIST, CISA, DoD, and others**

Razi Rais is a cybersecurity expert at Microsoft with over two decades of experience in building products.

Christina Morillo is an enterprise cybersecurity and technology leader with over two decades of experience.

Evan Gilman has built and operated systems in hostile environments throughout his career.

Doug Barth is a software engineer who loves to learn and share his knowledge with others.

"Zero trust is not just a strategy; it is a mindset that challenges assumptions, scrutinizes every interaction, and guards our digital systems against unseen foes. This book offers practical guidance for chief technology officers, engineers, and information technology professionals embarking on their zero trust journey."

—**Ann Johnson**
Corporate Vice President, Microsoft Security

"This book packages essential concepts of zero trust security in an easy to understand language. A definitive read for beginners and professionals alike."

—**Karan Dwivedi**
Security Engineering Manager at Google

NETWORK SECURITY

US $65.99 CAN $82.99

ISBN: 978-1-492-09659-7

Twitter: @oreillymedia
linkedin.com/company/oreilly-media
youtube.com/oreillymedia

Praise for *Zero Trust Networks*, 2nd edition

Zero trust is not just a strategy; it is a mindset that challenges assumptions, scrutinizes every interaction, and guards our digital systems against unseen foes. This book offers practical guidance for chief technology officers, engineers, and information technology professionals embarking on their zero trust journey.

—Ann Johnson, Corporate Vice President, Microsoft Security

This book packages essential concepts of zero trust security in an easy to understand language. A definitive read for beginners and professionals alike.

—Karan Dwivedi, Security Engineering Manager at Google

This book does an excellent job of synthesizing the zero-trust security model. It explains the key pillars of zero trust security while also covering the zero trust frameworks developed by NIST, DoD, CISA, and other organizations, making it a valuable resource for anyone seeking to understand how to implement the zero-trust security model.

—Andrew Cameron, Automotive Industry Technical Fellow in Identity

We may not realize this, but our lives depend on computers. When you are in an airplane, or in a hospital, or in a train, or even turning a light bulb on at home, it's all computers. A breach can cause pandemonium, and securing this infrastructure is paramount. As such, zero trust networks provide you with the fundamentals and mindset you need to understand to secure your investments. This book is a great resource for developers, infrastructure engineers, and managers alike, as it thoroughly explains the whys and hows of zero trust.

—Sahil Malik, Security Engineer, IT Industry

With the rapid adoption of cloud networks, bring-your-own-device, and work-from-home policies, implementing zero trust security in today's enterprise networks is an absolute must. It's a lot more complicated than it sounds. But Razi Rais and Christina Morillo make all of the technicalities understandable for readers with general IT backgrounds. Their book is a must read for all people who administrate computer networks for business.

—*Kim Crawley, cybersecurity researcher and author of*
Hacker Culture: A to Z *and* The Pentester Blueprint

2ND EDITION

Zero Trust Networks

Building Secure Systems in Untrusted Networks

Razi Rais, Christina Morillo,
Evan Gilman, and Doug Barth

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Zero Trust Networks
by Razi Rais, Christina Morillo, Evan Gilman, and Doug Barth

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*https://oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Acquisitions Editor: Simina Calin
Development Editor: Michele Cronin
Production Editor: Ashley Stussy
Copyeditor: Liz Wheeler
Proofreader: Sonia Saruba

Indexer: WordCo Indexing Services, Inc.
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Kate Dullea

June 2017: First Edition
March 2024: Second Edition

Revision History for the First Edition
2024-02-23: First Release

See *https://oreilly.com/catalog/errata.csp?isbn=9781492096597* for release details.

978-1-492-09659-7

[LSI]

Table of Contents

Preface

Thank you for choosing to read *Zero Trust Networks, 2E*! Building trusted systems in hostile networks has been a passion of ours for many years. In building and designing such systems, we have found frustration in the pace of progress toward solving some of the more fundamental security problems plaguing our industry. We'd very much like to see the industry move more aggressively toward building systems that strive to solve these problems.

To that end, we propose that the world take a new stance toward building and maintaining secure computer networks. Rather than being something that is layered on top, only considered after some value has been built, security must be fundamentally infused with the operation of the system itself. It must be ever-present, enabling operation rather than restricting it. As such, this book sets forth a collection of design patterns and considerations which, when heeded, can produce systems that are resilient to the vast majority of modern-day attack vectors.

This collection, when taken as a whole, is known as the zero trust model. In this model, nothing is taken for granted, and every single access request—whether it's made by a client in a coffee shop or a server in the datacenter—is rigorously checked and proven to be authorized. Adopting this model practically eliminates lateral movement, VPN headaches, and centralized firewall management overhead. It is a very different model; one that we believe represents the future of network and infrastructure security design.

In the second edition, we broaden the scope to include recent developments in zero trust. We have added two entirely new chapters and additional real-world scenario walkthroughs to the current chapters. The chapter on zero trust architectural standards, frameworks, and guidelines will help you better grasp the zero trust perspective from leading organizations, such as NIST, CISA, DoD, and others. Since zero trust initiatives are not easy, we added a chapter dedicated to discussing challenges and practical advice to deal with them. This chapter finishes with an examination of more recent technical advancements, including artificial intelligence, quantum computing,

and privacy-preserving technologies, all of which are highly relevant to zero trust and cybersecurity in general.

Who Should Read This Book

Have you found the overhead of centralized firewalls to be restrictive? Perhaps you've even found their operation to be ineffective. Have you struggled with VPN headaches, TLS configuration across a myriad of applications and languages, or compliance and auditing hardships? These problems represent a small subset of those addressed by the zero trust model. If you find yourself thinking that there just has to be a better way, then you're in luck—this book is for you.

Network engineers, security engineers, CTOs, and everyone in between can benefit from zero trust learnings. Even without a specialized skill set, many of the principles included in this book can be clearly understood, helping leaders make decisions that implement a zero trust model, improving their overall security posture incrementally.

Additionally, readers with experience using configuration management systems will see the opportunity to use those same ideas to build a more secure and operable networked system—one in which resources are secure by default. They will be interested in how automation systems can enable a new network design that is able to apply fine-grained security controls more easily. Finally, this book explores a mature zero trust design, enabling those who have already incorporated the basic philosophies to further the robustness of their security systems.

Why We Wrote This Book

We started speaking about our approach to system and network design at industry conferences in 2014. At the time, we were using configuration management systems to rigorously define the system state, applying changes programmatically as a reaction to topological changes. As a result of leveraging automation tools for this purpose, we naturally found ourselves programmatically calculating the network enforcement details instead of managing the configuration by hand. We found that using automation to capture the system design in this way allowed us to deploy and manage security features, including access control and encryption, much more easily than in systems past. Even better, doing so allowed us to place much less trust in the network than other systems might normally do, which is a key security consideration when operating in and across public clouds.

While writing this book, we spoke to individuals from dozens of companies to understand their perspective on network security designs. We found that many of those companies were reducing the trust of their internal networks. While each organization took a slightly different approach in their own system, it was clear that

they were all working under the same threat model and were, as a result, building solutions that shared many properties.

Our goal with this book isn't to present one or two particular solutions to building these types of systems, but rather to define a system model that places no trust in its communication network. Therefore, this book won't be focused on using specific vendor software or implementations, but rather it will explore the concepts and philosophies that are used to build a zero trust network. We hope you will find it useful to have a clear mental model for how to construct this type of system when building your own system or, even better, reusable solutions for the problems described herein.

Navigating This Book

This book is organized as follows:

- Chapters 1 and 2 discuss the fundamental concepts at play in a zero trust security model.
- Chapters 3 and 4 explore the new concepts typically seen in mature zero trust networks: context-aware network agents and trust engines.
- Chapters 5 through 8 detail how trust is established among the various actors in a network, with focus on devices, identities, applications, and network traffic. Most of this content is focused on existing technology that could be useful in a traditional network security model. The scenario walkthroughs at the end of each chapter will help you understand how the core principles of zero trust are used in a real-world setting.
- Chapter 9 brings all this content together to discuss how you could begin building your own zero trust network and includes two case studies.
- Chapter 10 looks at the zero trust security model from an adversarial view. It explores potential weaknesses, discussing which are well mitigated and which are not.
- Chapter 11 explores zero trust architectures, standards, and frameworks from NIST, CISA, DoD, and others. The goal is to help you understand the zero trust security model from the perspective of leading organizations in the industry.
- Chapter 12 outlines various functional and technical obstacles that organizations experience when implementing zero initiatives. It also provides high-level considerations that may assist you in effectively dealing with these challenges. Additionally, it examines the impact of artificial intelligence (AI), quantum computing, and privacy-enhancing technologies on zero trust security models, which are extremely important advancements to understand. The potential impact of AI, quantum computation, and privacy-enhancing technologies on zero trust

security model is also examined. Comprehending these advancements is of the utmost importance, given their pivotal role in cybersecurity strategy.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

This element signifies a general note.

This element indicates a warning or caution.

O'Reilly Online Learning

O'REILLY®

For more than 40 years, *O'Reilly Media* has provided technology and business training, knowledge, and insight to help companies succeed.

Our unique network of experts and innovators share their knowledge and expertise through books, articles, and our online learning platform. O'Reilly's online learning platform gives you on-demand access to live training courses, in-depth learning paths, interactive coding environments, and a vast collection of text and video from O'Reilly and 200+ other publishers. For more information, visit *https://oreilly.com*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-889-8969 (in the United States or Canada)
707-827-7019 (international or local)
707-829-0104 (fax)
support@oreilly.com
https://www.oreilly.com/about/contact.html

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *https://oreil.ly/zero-trust-networks-2e*.

For news and information about our books and courses, visit *https://oreilly.com*.

Find us on LinkedIn: *https://linkedin.com/company/oreilly-media*

Follow us on Twitter: *https://twitter.com/oreillymedia*

Watch us on YouTube: *https://youtube.com/oreillymedia*

Acknowledgments from the First Edition

We would like to thank our editor, Courtney Allen, for her help and guidance during the writing process. Thanks also to Virginia Wilson, Nan Barber, and Maureen Spencer for their help during the review.

We had the opportunity to meet with many people during the writing of this content, and we appreciate their willingness to speak with us and provide intros to other folks working in this space. Thanks to Rory Ward, Junaid Islam, Stephen Woodrow, John Kindervag, Arup Chakrabarti, Julia Evans, Ed Bellis, Andrew Dunham, Bryan Berg, Richo Healey, Cedric Staub, Jesse Endahl, Andrew Miklas, Peter Smith, Dimitri Stiliadis, Jason Chan, and David Cheney.

A special thanks to Betsy Beyer for writing the Google BeyondCorp case study included in the book. We really appreciate your willingness to work on getting that content included. Thanks!

Thanks to our technical reviewers, Ryan Huber, Kevin Babcock, and Pat Cable. We found your comments invaluable and appreciate the time you took to read through the initial drafts.

Doug would like to thank his wife, Erin, and daughters, Persephone and Daphne, for being so very understanding of the time it took to write this book.

Evan thanks his partner, Kristen, for all of her support through the writing of this book. He would also like to thank Kareem Ali and Kenrick Thomas—without them, none of this would have been possible.

Acknowledgments from the Second Edition

We are especially grateful to Michele Cronin, our development editor, for her assistance and direction throughout the process. Thanks also to Simina Calin, our acquisitions editor, for helping us in establishing a successful path for this book.

Our heartfelt thanks goes out to our technical reviewers, including Kim Crawley, Steve Winterfeld, and Karan Dwivedi, whose extensive feedback and recommendations have enhanced every facet of this book. Many thanks!

Razi would like to thank his wonderful wife, Javeria, as well as his mother and sister, Zahida and Khaizran, for their unwavering support throughout the writing of this book.

Christina would like to thank her husband and children for their steadfast support and patience during the book-writing journey.

CHAPTER 1
Zero Trust Fundamentals

In an age when network surveillance is ubiquitous, we find it difficult to trust anyone, and defining what trust is itself is equally difficult. Can we trust that our internet traffic will be safe from eavesdropping? Certainly not! What about that provider you leased your fiber from? Or that contracted technician who was in your datacenter yesterday working on the cabling?

Whistleblowers like Edward Snowden and Mark Klein have revealed the tenacity of government-backed spy rings. The world was shocked at the revelation that they had managed to get inside the datacenters of large organizations. But why? Isn't it exactly what you would do in their position? Especially if you knew that traffic there would not be encrypted?

The assumption that systems and traffic within a datacenter can be trusted is flawed. Modern networks and usage patterns no longer echo those that made perimeter defense make sense many years ago. As a result, moving freely within a "secure" infrastructure frequently has a low barrier to entry once a single host or link there has been compromised.

You may think that the idea of using a cyberattack as a weapon to disrupt critical infrastructure like a nuclear plant or a power grid is far-fetched, but cyberattacks on the Colonial Pipeline in the United States and the Kudankulam Nuclear Power Plant in India serve as a stark reminder that critical infrastructure will continue to be a high-value target for attackers. So, what was common between the two attacks?

Well, in both cases, security was abysmal. Attackers took advantage of the fact that the VPN (virtual private network) connection to the Colonial Pipeline network was possible using a plain-text password without any multifactor authentication (MFA) in place. In the other example, malware was discovered on an Indian nuclear power plant employee's computer that was connected to the administrative network's internet servers. Once the attackers gained access, they were able to roam within the network due to the "trust" that comes with being inside the network.

Zero trust aims to solve the inherent problems in placing our trust in the network. Instead, it is possible to secure network communication and access so effectively that the physical security of the transport layer can be reasonably disregarded. It goes without saying that this is a lofty goal. The good news is that we've got pretty powerful cryptographic algorithms these days, and given the right automation systems, this vision is actually attainable.

What Is a Zero Trust Network?

A zero trust network is built upon five fundamental assertions:

- The network is always assumed to be hostile.
- External and internal threats exist on the network at all times.
- Network locality alone is not sufficient for deciding trust in a network.
- Every device, user, and network flow is authenticated and authorized.
- Policies must be dynamic and calculated from as many sources of data as possible.

Traditional network security architecture breaks different networks (or pieces of a single network) into zones, contained by one or more firewalls. Each zone is granted some level of trust, which determines the network resources it is permitted to reach. This model provides very strong defense-in-depth. For example, resources deemed more risky, such as web servers that face the public internet, are placed in an exclusion zone (often termed a "DMZ"), where traffic can be tightly monitored and controlled. Such an approach gives rise to an architecture that is similar to some you might have seen before, such as the one shown in Figure 1-1.

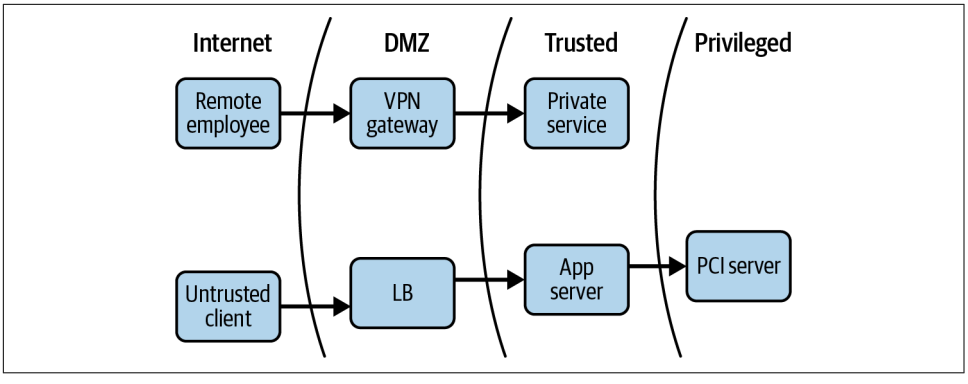

Figure 1-1. Traditional network security architecture

The zero trust model turns this diagram inside out. Placing stopgaps in the network is a solid step forward from the designs of yesteryear, but it is significantly lacking in the modern cyberattack landscape. There are many disadvantages:

- Lack of intra-zone traffic inspection
- Lack of flexibility in host placement (both physical and logical)
- Single points of failure

It should be noted that, should network locality requirements be removed, the need for VPNs is also removed. A virtual private network (VPN) allows a user to authenticate in order to receive an IP address on a remote network. The traffic is then tunneled from the device to the remote network, where it is decapsulated and routed. It's the greatest backdoor that no one ever suspected. If we instead declare that network location has no value, VPN is suddenly rendered obsolete, along with several other modern network constructs. Of course, this mandate necessitates pushing enforcement as far toward the network edge as possible, but at the same time it relieves the core from such responsibility. Additionally, stateful firewalls exist in all major operating systems, and advances in switching and routing have opened an opportunity to install advanced capabilities at the edge. All of these gains come together to form one conclusion: the time is right for a paradigm shift. By leveraging distributed policy enforcement and applying zero trust principles, we can produce a design similar to the one shown in Figure 1-2.

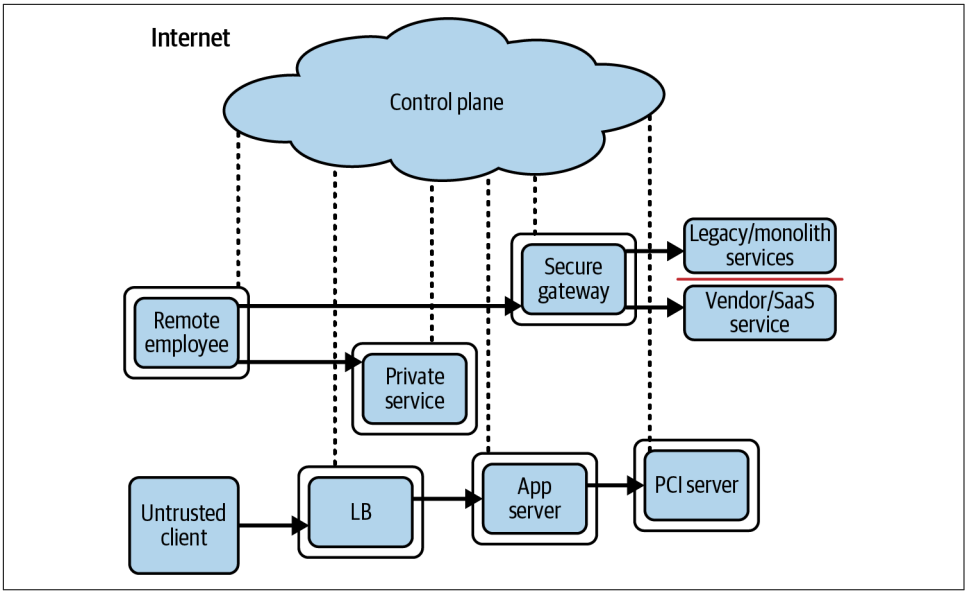

Figure 1-2. Zero trust architecture

Introducing the Zero Trust Control Plane

The supporting system is known as the control plane, while most everything else is referred to as the data plane, which the control plane coordinates and configures. Requests for access to protected resources are first made through the control plane, where both the device and user must be authenticated and authorized. Fine-grained policy can be applied at this layer, perhaps based on role in the organization, time of day, geo-location, or type of device. Access to more secure resources can additionally mandate stronger authentication.

Once the control plane has decided that the request will be allowed, it dynamically configures the data plane to accept traffic from that client (and that client only). In addition, it can coordinate the details of an encrypted tunnel between the requestor and the resource. This can include temporary one-time-use credentials, keys, and ephemeral port numbers.

It should be noted that the control plane decision to allow a request is time-bound rather than permanent. This means that if and when the factors that led the control plane decision to allow the request in the first place have changed, it may coordinate with the data plane to revoke the requested access to the resource.

While some compromises can be made on the strength of these measures, the basic idea is that an authoritative source, or trusted third party, is granted the ability to authenticate, authorize, and coordinate access in real time, based on a variety of inputs. We'll discuss the control and data planes more in Chapter 2.

Evolution of the Perimeter Model

The traditional architecture described in this book is often referred to as the perimeter model, after the castle-wall approach used in physical security. This approach protects sensitive items by building lines of defenses that an intruder must penetrate before gaining access. Unfortunately, this approach is fundamentally flawed in the context of computer networks and no longer suffices. To fully understand the failure, it is useful to recall how the current model was arrived at.

Managing the Global IP Address Space

The journey that led to the perimeter model began with address assignment. Networks were being connected at an ever-increasing rate during the days of the early internet. If a network wasn't being connected to the internet (remember, the internet wasn't ubiquitous at the time), it was being connected to another business unit, another company, or perhaps a research network. Of course, IP addresses must be unique in any given IP network, and if the network operators were unlucky enough to have overlapping ranges, they would have a lot of work to do in changing them all. If the network you are connecting to happens to be the internet, then your addresses must be globally unique. So clearly some coordination is required here.

The Internet Assigned Numbers Authority (IANA), formally established in 1998, is the body that today provides that coordination. Prior to the establishment of the IANA, this responsibility was handled by Jon Postel, who created the internet map shown in Figure 1-3. He was the authoritative source for IP address ownership records, and if you wanted to guarantee that your IP addresses were globally unique, you would register with him. At this time, everybody was encouraged to register for IP address space, even if the network being registered was not going to be connected to the internet. The assumption was that even if a network was not connected now, it would probably be connected to another network at some point.

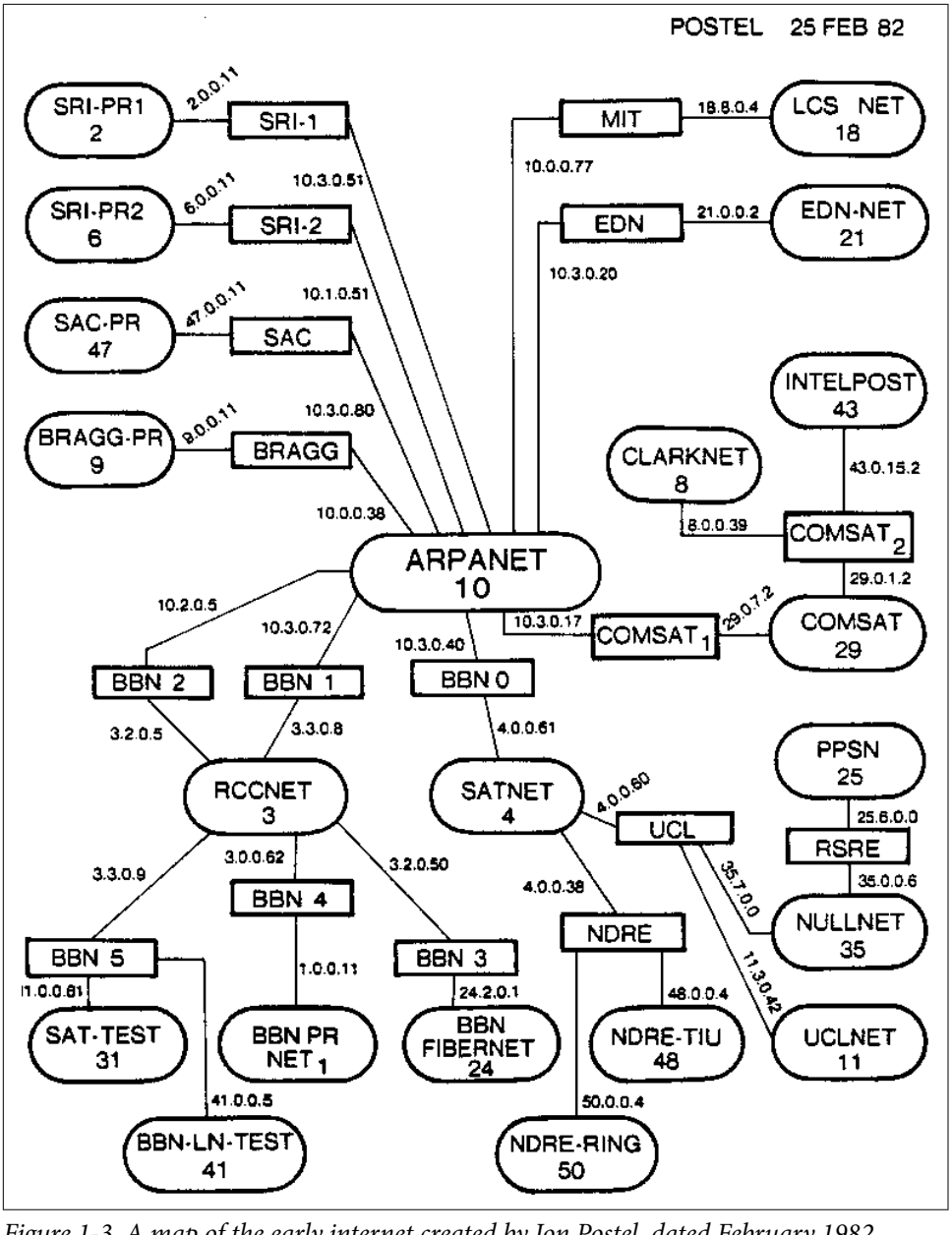

Figure 1-3. A map of the early internet created by Jon Postel, dated February 1982

Birth of Private IP Address Space

As IP adoption grew through the late 1980s and early 1990s, frivolous use of address space became a serious concern. Numerous cases of truly isolated networks with large IP address space requirements began to emerge. Networks connecting ATMs and arrival/departure displays at large airports were touted as prime examples. These networks were considered truly isolated for various reasons. Some devices might be isolated to meet security or privacy requirements (e.g., networks meant for ATMs). Some might be isolated because the scope of their function was so limited that having broader network access was seen as exceedingly unlikely (e.g., airport arrival and departure displays). RFC 1597, Address Allocation for Private Internets, was introduced to address this wasted public address space issue.

In March of 1994, RFC 1597 announced that three IP network ranges had been reserved with IANA for general use in private networks: 10.0.0.0/8, 172.16.0.0/12, and 192.168.0.0/16. This had the effect of slowing address depletion by ensuring that the address space of large private networks never grew beyond those allocations. It also enabled network operators to use nonglobally unique addresses where and when they saw fit. It had another interesting effect, which lingers with us today: networks using private addresses were more secure, because they were fundamentally incapable of joining other networks, particularly the internet.

At the time, very few organizations (relatively speaking) had an internet connection or presence, and as such, internal networks were frequently numbered with the reserved ranges. Additionally, security measures were weak to nonexistent because these networks were typically confined by the walls of a single organization.

Private Networks Connect to Public Networks

The number of interesting things on the internet grew fairly quickly, and soon most organizations wanted at least some sort of presence. Email was one of the earliest examples of this. People wanted to be able to send and receive email, but that meant they needed a publicly accessible mail server, which of course meant that they needed to connect to the internet somehow. With established private networks, it was often the case that this mail server would be the only server with an internet connection. It would have one network interface facing the internet, and one facing the internal network. With that, systems and people on the internal private network got the ability to send and receive internet email via their connected mail server.

It was quickly realized that these servers had opened up a physical internet path into their otherwise secure and private network. If one was compromised, an attacker might be able to work their way into the private network, since hosts there can communicate with it. This realization prompted strict scrutiny of these hosts and their network connections. Network operators placed firewalls on both sides of them

to restrict communication and thwart potential attackers attempting to access internal systems from the internet, as shown in Figure 1-4. With this step, the perimeter model was born. The internal network became the "secure" network, and the tightly controlled pocket that the external hosts lay in became the DMZ, or the demilitarized zone.

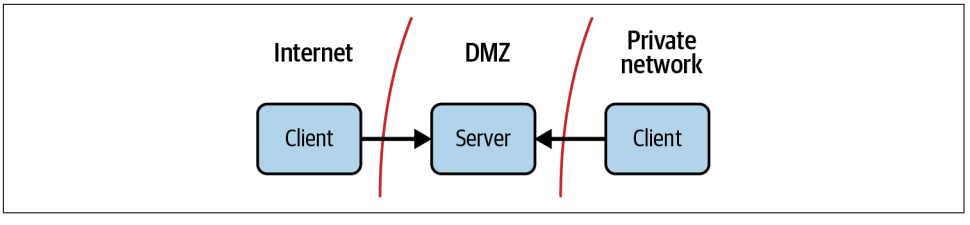

Figure 1-4. Both internet and private resources can access hosts in the DMZ; private resources, however, cannot reach beyond the DMZ, and thus do not gain direct internet access

Birth of NAT

The number of internet resources being accessed from internal networks was growing rapidly, and it quickly became easier to grant general internet access to internal resources than it was to maintain intermediary hosts for every application desired. NAT, or network address translation, solved that problem nicely.

RFC 1631, The IP Network Address Translator, defines a standard for a network device that is capable of performing IP address translation at organizational boundaries. By maintaining a table that maps public IPs and ports to private ones, it enabled devices on private networks to access arbitrary internet resources. This lightweight mapping is application agnostic, which meant that network operators no longer needed to support internet connectivity for particular applications; they needed only to support internet connectivity in general.

These NAT devices had an interesting property: because the IP mapping was many-to-one, it was not possible for incoming connections from the internet to access internal private IPs without specifically configuring the NAT to handle this special case. In this way, the devices exhibited the same properties as a stateful firewall. Actual firewalls began integrating NAT features almost instantaneously, and the two became a single function, largely indistinguishable. Supporting both network compatibility and tight security controls meant that eventually you could find one of these devices at practically every organizational boundary, as shown in Figure 1-5.

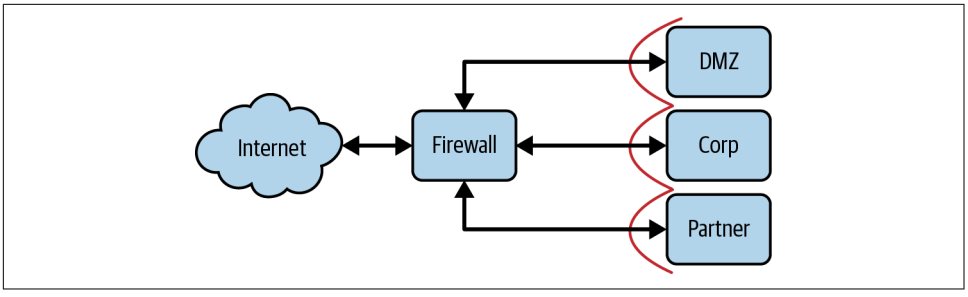

Figure 1-5. Typical (and simplified) perimeter firewall design

The Contemporary Perimeter Model

With a firewall/NAT device between the internal network and the internet, the security zones are clearly forming. There is the internal "secure" zone, the DMZ (demilitarized zone), and the untrusted zone (aka the internet). If at some point in the future, this organization needed to interconnect with another, a device would be placed on that boundary in a similar manner. The neighboring organization would likely become a new security zone, with particular rules about what kind of traffic can go from one to the other, just like the DMZ or the secure zone.

Looking back, we see the progression. We went from offline/private networks with just one or two hosts with internet access to highly interconnected networks with security devices around the perimeter. It is not hard to understand: network operators couldn't afford to sacrifice the perfect security of their offline network because they had to open doors for various business purposes. Tight security controls at each door minimized the risk.

Evolution of the Threat Landscape

Even before the public internet, communicating with a remote computer system was highly desirable. This was commonly done over the public telephone system. Users and computer systems could dial in and, by encoding data into audible tones, gain connectivity to the remote machine. These dial-in interfaces were the most common attack vector of the day, since gaining physical access was much more difficult.

Once organizations had internet-connected hosts, attacks shifted from occurring over the telephone network to being launched over the dial-up internet. This triggered a change in most attack dynamics. Incoming calls to dial-in interfaces tied up a phone line, and were a notable occurrence when compared to a TCP connection coming from the internet. It was much easier to have a covert presence on an IP network than it was on a system that needed to be dialed into. Exploitation and brute force attempts could be carried out over long periods of time without raising too much

suspicion...though an additional and more impactful capability arose from this shift: malicious code could then listen for internet traffic.

By the late 1990s, the world's first (software) Trojan horses began to make their rounds. Typically, a user would be tricked into installing the malware, which would then open a port and wait for incoming connections. The attacker could then connect to the open port and remotely control the target machine.

It wasn't long before that people realized it would be a good idea to protect those internet-facing hosts. Hardware firewalls were the best way to do it (most operating systems had no concept of a host-based firewall at the time). They provided policy enforcement, ensuring that only whitelisted/allowed-listed "safe" traffic was allowed in from the internet. If an administrator inadvertently installed something that exposed an open port (like a Trojan horse), the firewall would physically block connections to that port until explicitly configured to allow it. Likewise, traffic to the internet-facing servers from inside the network could be controlled, ensuring that internal users could speak to them, but not vice versa. This helped prevent movement into the internal network by a potentially compromised DMZ host.

DMZ hosts were of course a prime target (due to their connectivity), though such tight controls on both inbound and outbound traffic made it hard to reach an internal network through a DMZ. An attacker would first have to compromise the firewalled server, then abuse the application in such a way that it could be used for covert communication (they need to get data out of that network, after all). Dial-in interfaces remained the lowest-hanging fruit if one was determined to gain access to an internal network.

This is where things took an interesting turn. NAT was introduced to grant internet access to clients on internal networks. Due in some part to NAT mechanics and in some part to real security concerns, there was still tight control on inbound traffic, though internal resources wishing to consume external resources might freely do so. There's an important distinction to be made when considering a network with NAT'd internet access against a network without it: the former has a relaxed (if any) outbound network policy.

This significantly transformed the network security model. Hosts on the "trusted" internal networks could then communicate directly with untrusted internet hosts, and the untrusted host was suddenly in a position to abuse the client attempting to speak with it. Even worse, malicious code could then send messages to internet hosts from within the internal network. Today, we know this as "phoning home."

Phoning home is a critical component of most modern attacks. It allows data to be exfiltrated from otherwise-protected networks; but more importantly, since TCP is bidirectional, it allows data to be injected as well. A typical attack involves several steps, as shown in Figure 1-6. First, the attacker will compromise a single computer

on the internal network by exploiting the user's browser when they visit a particular page, by sending them an email with an attachment that exploits some local software, for example. The exploit carries a very small payload, just enough code to make a connection out to a remote internet host and execute the code it receives in the response. This payload is sometimes referred to as a dialer.

The dialer downloads and installs the real malware, which more often than not will attempt to make an additional connection to a remote internet host controlled by the attacker. The attacker will use this connection to send commands to the malware, exfiltrate sensitive data, or even to obtain an interactive session. This "patient zero" can act as a stepping stone, giving the attacker a host on the internal network from which to launch additional attacks.

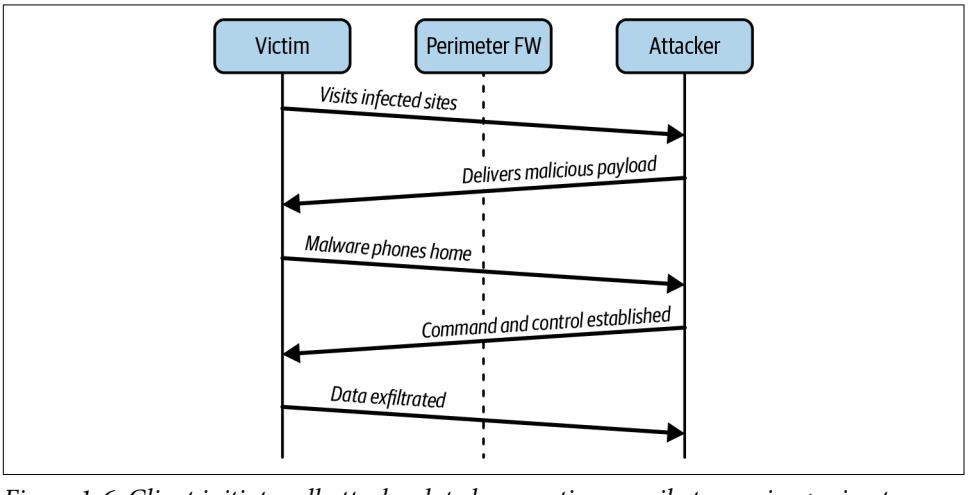

Figure 1-6. Client initiates all attack-related connections, easily traversing perimeter firewalls with relaxed outbound security

Outbound Security

Outbound network security is a very effective mitigation measure against dialer-based attacks, as the phone home can be detected and/or blocked. Oftentimes, however, the phone home is disguised as regular web traffic, possibly even to networks that are seemingly benign or "normal." Outbound security tight enough to stop these attacks will oftentimes cripple web usability for users. This is a more realistic prospect for back-office systems.

The ability to launch attacks from hosts within an internal network is a very powerful one. These hosts almost certainly have permission to talk to other hosts in the same security zone (lateral movement) and might even have access to talk to hosts in zones more secure than their own. To this effect, by first compromising a low-security

zone on the internal network, an attacker can move through the network, eventually gaining access to the high-security zones.

Taking a step back for a moment, it can be seen that this pattern very effectively undermines the perimeter security model. The critical flaw enabling attack progression is subtle, yet clear: security policies are defined by network zones, enforced only at zone boundaries, using nothing more than the source and destination details.

Other threats have risen as the world has become more ubiquitous over the years. Companies nowadays allow their workers to use their own devices for work in addition to the devices provided by the company, thanks to the popularity of Bring Your Own Device (BYOD). Employees can be more productive as a result of this, as they work from home more than ever before. During COVID-19, we discovered the advantages of BYOD when employees were no longer able to enter the workplace for extended periods of time. However, the attack surface area has grown because patching numerous devices with the most recent security fixes is significantly more difficult than patching a single device. One type of attack, among others, is the zero-click attack, which does not even require user interaction (more about it in the note below). Attackers deliberately look for devices that haven't had their security patches updated in order to exploit vulnerabilities and obtain unauthorized access to them. In Chapter 5, we'll look at the role of security patches and how to automate them to improve device trust.

Zero-Click Attack

A zero-click attack is a highly sophisticated attack that infects the user's device without the user's involvement. Zero-click attacks frequently take advantage of unpatched arbitrary code execution and buffer overflow security flaws. Because these attacks are conducted without user interaction, they can be incredibly effective. Popular apps like WhatsApp and Apple's iMessage have been reported to be vulnerable to zero-click attacks. In 2021, Google provided a comprehensive investigation of the iMessage zero-click vulnerability, which describes the attack's far-reaching ramifications. Patching all devices that have access to company resources and services is critical at all times.

Perimeter Shortcomings

Even though the perimeter security model still stands as the most prevalent model by far, it is increasingly obvious that the way we rely on it is flawed. Complex (and successful) attacks against networks with perfectly good perimeter security occur every day. An attacker drops a remote access tool (or RAT) into your network through one of a myriad of methods, gains remote access, and begins moving laterally. Perimeter

firewalls have become the functional equivalent of building a wall around a city to keep out the spies.

The problem comes when architecting security zones into the network itself. Imagine the following scenario: you run a small ecommerce company. You have some employees, some internal systems (payroll, inventory, etc.), and some servers to power your website. It is natural to begin classifying the kind of access these groups might need: employees need access to internal systems, web servers need access to database servers, database servers don't need internet access but employees do, and so on. Traditional network security would codify these groups as zones and then define which zone can access what, as shown in Figure 1-7. Of course, you need to actually enforce these policies; and since they are defined on a zone-by-zone basis, it makes sense to enforce them wherever one zone can route traffic into another.

As you might imagine, there are always exceptions to these generalized rules...they are, in fact, colloquially known as firewall exceptions. These exceptions are typically as tightly scoped as possible. For instance, your web developer might want SSH access to the production web servers, or your HR representative might need access to the HR software's database in order to perform audits. In these cases, an acceptable approach is to configure a firewall exception permitting traffic from that individual's IP address to the particular server(s) in question.

Now let's imagine that your archnemesis has hired a team of hackers. They want to have a peek at your inventory and sales numbers. The hackers send emails to all the employee email addresses they can find on the internet, masquerading as a discount code for a restaurant near the office. Sure enough, one of them clicks the link, allowing the attackers to install malware. The malware phones home and provides the attackers with a session on the now-compromised employee's machine. Luckily, it's only an intern, and the level of access they gain is limited.

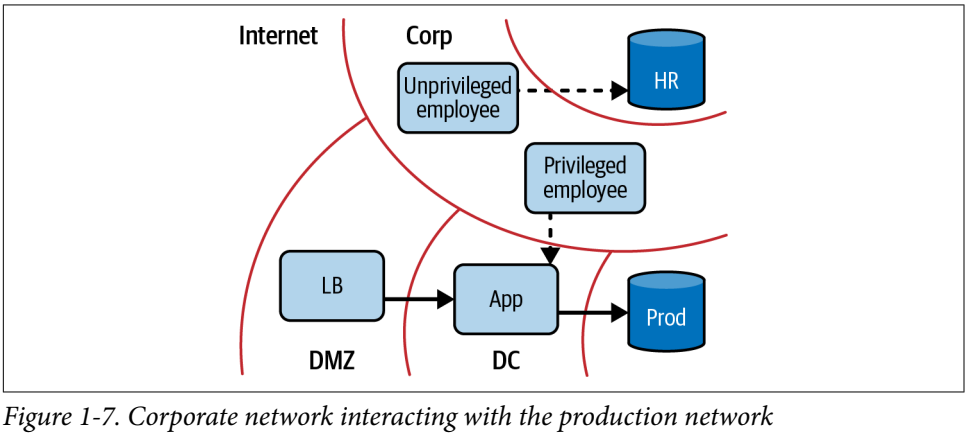

Figure 1-7. Corporate network interacting with the production network

They begin searching the network and find that the company is using file sharing software on its network. Out of all the employee computers on the network, none of them have the latest version and are vulnerable to an attack that was recently publicized.

One by one, the hackers begin searching for a computer with elevated access (this process of course can be more targeted if the attacker has advanced knowledge). Eventually they come across your web developer's machine. A keylogger they install there recovers the credentials to log in to the web server. They SSH to the server using the credentials they gathered, and using the sudo rights of the web developer, they read the database password from disk and connect to the database. They dump the contents of the database, download it, and delete all the log files. If you're lucky, you might actually discover that this breach occurred. They accomplished their mission, as shown in Figure 1-8.

Wait, what? As you can see, many failures at many levels led to this breach, and while you might think that this is a particularly contrived case, successful attacks just like this one are staggeringly common. The most surprising part, however, goes unnoticed all too often: what happened to all that network security? Firewalls were meticulously placed, policies and exceptions were tightly scoped and very limited, everything was done right from a network security perspective. So what gives?

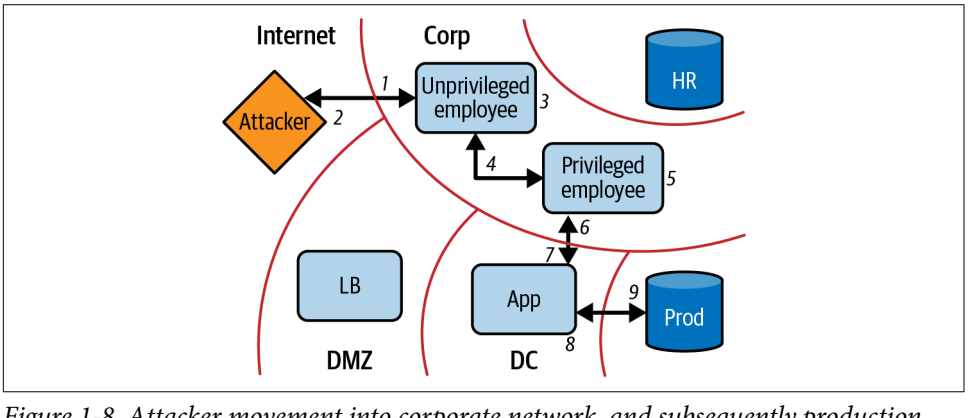

Figure 1-8. Attacker movement into corporate network, and subsequently production into network

When carefully examined, it is overwhelmingly obvious that this network security model is not enough. Bypassing perimeter security is trivial with malware that phones home, and firewalls between zones consider nothing more than source and destination when making enforcement decisions. While perimeters can still provide some value in network security, their role as the primary mechanism by which a network's security stance is defined needs to be reconsidered.

Example Attack Progression

1. Employees targeted via phishing email
2. Corporate machine compromised, shell shoveled
3. Lateral movement through corporate network
4. Privileged workstation located
5. Local privilege escalation on workstation—keylogger installed
6. Developer password stolen
7. Compromised prod app host from privileged workstation
8. Developer password used to elevate privileges on prod app host
9. Database credentials stolen from app
10. Database contents exfiltrated via compromised app host

The first step, of course, is to search for existing solutions. Sure, the perimeter model is the accepted approach to securing a network, but that doesn't mean we haven't learned better elsewhere. What is the worst possible scenario network security-wise? It turns out that there is actually a level of absoluteness to this question, and the crux of it lies in trust.

Where the Trust Lies

When considering options beyond the perimeter model, one must have a firm understanding of what is trusted and what isn't. The level of trust defines a lower limit on the robustness of the security protocols required. Unfortunately, it is rare for robustness to exceed what is required, so it is wise to trust as little as possible. Once trust is built into a system, it can be very hard to remove.

A zero trust network is just as it sounds. It is a network that is completely untrusted. Lucky for us, we interact with such a network very frequently: the internet. The internet has taught us some valuable security lessons. Certainly, an operator will secure an internet-facing server much differently than it secures its locally accessible counterpart. Why is that? And if the pains associated with such rigor were cured (or even just lessened), would the security sacrifice still be worth it?

The zero trust model dictates that all hosts be treated as if they're internet facing. The networks they reside in must be considered compromised and hostile. Only with this consideration can you begin to build secure communication. With most operators having built or maintained internet-facing systems in the past, we have at least some idea of how to secure IP in a way that is difficult to intercept or tamper with (and, of course, how to secure those hosts). Automation enables us to extend this level of security to all of the systems in our infrastructure.

Automation as an Enabler

Zero trust networks do not require new protocols or libraries. They do, however, use existing technologies in novel ways. Automation systems are what allow a zero trust network to be built and operated.

Interactions between the control plane and the data plane are the most critical points requiring automation. If policy enforcement cannot be dynamically updated, zero trust will be unattainable; therefore, it is critical that this process be automatic and rapid.

There are many ways that this automation can be realized. Purpose-built systems are most ideal, though more mundane systems like traditional configuration management can fit here as well. Widespread adoption of configuration management represents an important stepping stone for a zero trust network, as these systems often maintain device inventories and are capable of automating network enforcement configuration in the data plane.

Due to the fact that modern configuration management systems can both maintain a device inventory and automate the data plane configuration, they are well positioned to be a first step toward a mature zero trust network.

Perimeter Versus Zero Trust

The perimeter and zero trust models are fundamentally different from each other. The perimeter model attempts to build a wall between trusted and untrusted resources (i.e., the local network and the internet). On the other hand, the zero trust model basically throws the towel in and accepts the reality that the "bad guys" are everywhere. Rather than build walls to protect the soft bodies inside, it turns the entire population into a militia.

The current approaches to perimeter networks assign some level of trust to the protected networks. This notion violates the zero trust model and leads to some bad behavior. Operators tend to let their guard down a bit when the network is "trusted" (they are human). Rarely are hosts that share a trust zone protected from themselves. Sharing a trust zone, after all, seems to imply that they are equally trusted. Over time, we have come to learn that this assumption is false, and it is not only necessary to protect your hosts from the outside, but it is also necessary to protect them from each other.

Since the zero trust model assumes the network is fully compromised, you must also assume that an attacker can communicate using any arbitrary IP address. Thus, protecting resources by using IP addresses or physical location as an identifier is not enough. All hosts, even those that share "trust zones," must provide proper identification. Attackers are not limited to active attacks, though. They can still perform

passive attacks in which they sniff your traffic for sensitive information. In this case, even host identification is not enough—strong encryption is also required.

There are three key components in a zero trust network: user/application authentication and authorization, device authentication and authorization, and trust. The first component has some duality in it due to the fact that not all actions are taken by users. So in the case of automated action (inside the datacenter, for instance), we look at qualities of the application in the same way that we would normally look at qualities of the user.

Authenticating and authorizing the device is just as important as doing so for the user/application. This is a feature rarely seen in services and resources protected by perimeter networks. It is often deployed using VPN or NAC technology, especially in more mature networks, but finding it between endpoints (as opposed to network intermediaries) is uncommon.

NAC as a Perimeter Technology

NAC, or Network Access Control, represents a set of technologies designed to strongly authenticate devices in order to gain access to a sensitive network. These technologies, which include protocols like 802.1X and the Trusted Network Connect (TNC) family, focus on admittance to a network rather than admittance to a service and as such are independent of the zero trust model. An approach more consistent with the zero trust model would involve similar checks located as close to the service being accessed as possible (something which TNC can address—more on this in Chapter 5). While NAC can still be employed in a zero trust network, it does not fulfill the zero trust device authentication requirement due to its distance from the remote endpoint.

Finally, a "trust score" is computed, and the application, device, and score are bonded to form an agent. Policy is then applied against the agent in order to authorize the request. The richness of information contained within the agent allows very flexible yet fine-grained access control, which can adapt to varying conditions by inclusion of the score component in your policies.

If the request is authorized, the control plane signals the data plane to accept the incoming request. This action can configure encryption details as well. Encryption can be applied at the device level, application level, or both. At least one is required for confidentiality.

With these authentication/authorization components, and the aid of the control plane in coordinating encrypted channels, we can assert that every single flow on the network is authenticated and expected. Hosts and network devices drop traffic that has not had all of these components applied to it, ensuring sensitive data can

never leak out. Additionally, by logging each of the control plane events and actions, network traffic can be easily audited on a flow-by-flow or request-by-request basis.

Perimeter networks can be found that have similar capability, though these capabilities are enforced at the perimeter only. VPNs famously attempt to provide these qualities in order to secure access to an internal network, but the security ends as soon as your traffic reaches a VPN concentrator. It is apparent that operators know what internet-strength security is supposed to look like; they just fail to implement those strong measures throughout.

If one can imagine a network that applies these measures homogeneously, a brief thought experiment can shed a lot of light on this new paradigm. Identity can be proven cryptographically, meaning it no longer matters what IP address any given connection is originating from (technically, you can still associate risk with it—more on that later). With automation removing the technical barriers, the VPN is essentially obsolete. "Private" networks no longer mean anything special: the hosts there are just as hardened as the ones on the internet. Thinking critically about NAT and private address space, perhaps zero trust makes it more obvious that the security arguments for it are null and void.

Ultimately, the perimeter model flaw is its lack of universal protection and enforcement. Secure cells with soft bodies inside. What we're really looking for is hard bodies, bodies that know how to check IDs and speak in a way they can't be overheard. Having hard bodies doesn't necessarily preclude you from also maintaining the security cells. In very sensitive installations, this would still be encouraged. It does, however, raise the security bar high enough that it wouldn't be unreasonable to lessen or remove those cells. Combined with the fact that the majority of the zero trust function can be done with transparency to the end user, the model almost seems to violate the security/convenience trade-off: stronger security, more convenience. Perhaps the convenience problem (or lack thereof) has been pushed onto the operators.

Applied in the Cloud

There are many challenges in deploying infrastructure into the cloud, one of the larger being security. Zero trust is a perfect fit for cloud deployments for an obvious reason: you can't trust the network in a public cloud! The ability to authenticate and secure communication without relying on IP addresses or the security of the network connecting them means that compute resources can be nearly commoditized. Since zero trust advocates that every packet be encrypted, even within the same datacenter, operators need not worry about which packets traverse the internet and which don't. This advantage is often understated. Cognitive load associated with when, where, and how to encrypt traffic can be quite large, particularly for developers who may not fully understand the underlying system. By eliminating special cases, we can also eliminate the human error associated with them.

Some might argue that intra-datacenter encryption is overkill, even with the reduction in cognitive load. History has proven otherwise. At large cloud providers like AWS, a single "region" consists of many datacenters, with fiber links between them. To the end user, this subtlety is often obfuscated. The NSA was targeting precisely links like these in rooms like the one shown in Figure 1-9.

Figure 1-9. Room 641A—NSA interception facility inside an AT&T datacenter in San Francisco

There are additional risks in the network implementation of the provider itself. It is not impossible to think that a vulnerability might exist in which neighbors can see your traffic. A more likely case is network operators inspecting traffic while troubleshooting. Perhaps the operator is honest, but how about the person who stole their laptop a few hours later with your captures on the disk? The unfortunate reality is that we can no longer assume that our traffic is protected from snooping or modification while in the datacenter.

Role of Zero Trust in National Cybersecurity

In 2021, the United States White House released Executive Order (EO) 14028, calling out the need to improve national cybersecurity on an urgent basis. The backdrop of

this EO was ever increasingly sophisticated cyberattacks over the span of many years, predominantly from foreign adversaries, putting national security at risk. EO 14028 specifically calls out advancement toward zero trust architecture as a critical step in improving national cybersecurity:

> The Federal Government must adopt security best practices; **advance toward Zero Trust Architecture**; …..
>
> —Excerpt from EO 14028

Adoption of zero trust is not just exclusive to the United States government by any means. Governments across the globe have been embracing it to improve the security posture. Another example is United Kingdom's National Cyber Security Centre zero trust architecture design principles.

In later chapters, we'll cover efforts from various governmental and non-governmental organizations like the National Institute of Standards and Technology (NIST), the Cybersecurity & Infrastructure Security Agency (CISA), The Open Group, etc., in publishing zero trust architecture, principles, and guidelines.

Summary

This chapter explored the high-level concepts that have led us toward the zero trust model. The zero trust model does away with the perimeter model, which attempts to ensure that bad actors stay out of the trusted internal network. Instead, the zero trust system recognizes that this approach is doomed to failure, and as a result, starts with the assumption that malicious actors are within the internal network and builds up security mechanisms to guard against this threat.

To better understand why the perimeter model is failing us, we reviewed how the perimeter model came into being. Back at the internet's beginning, the network was fully routable. As the system evolved, some users identified areas of the network that didn't have a credible reason to be routable on the internet, and thus the concept of a private network was born. Over time, this idea took hold, and organizations modeled their security around protecting the trusted private network. Unfortunately, these private networks aren't nearly as isolated as the original private networks were. The end result is a very porous perimeter, which is frequently breached in regular security incidents.

With the shared understanding of perimeter networks, we are able to contrast that design against the zero trust design. The zero trust model carefully manages trust in the system. These types of networks lean on automation to realistically manage the security control systems that allow us to create a more dynamic and hardened system. We introduced some key concepts like the authentication of users, devices, and applications, and the authorization of the combination of those components. We will discuss these concepts in greater detail throughout the rest of this book.

Finally, we talked about how the move to public cloud environments and the pervasiveness of internet connectivity have fundamentally changed the threat landscape. "Internal" networks are now increasingly shared and sufficiently abstracted away in such a way that end users don't have as clear an understanding of when their data is transiting more vulnerable long-distance network links. The end result of this change is that data security is more important than ever when constructing new systems. The next chapter will discuss the high-level concepts that need to be understood in order to build systems that can safely manage trust.

CHAPTER 2
Managing Trust

Trust management is perhaps the most important component of a zero trust network. We are all familiar with trust to some degree—you probably trust members of your family, but not a stranger on the street, and certainly not a stranger who looks threatening or menacing. Why is that?

For starters, you actually know your family members. You know what they look like, where they live; perhaps you've even known them your whole life. There is no question of who they are, and you are more likely to trust them with important matters than others.

A stranger, on the other hand, is someone completely unknown. You might see their face, and be able to tell some basic things about them, but you don't know where they live, and you don't know their history. They might appear perfectly cromulent, but you likely wouldn't rely on them for important matters. Watch your stuff for you while you run to the bathroom? Sure. Make a quick run to the ATM for you? Definitely not.

In the end, you are simply taking in all the information you can tell about the situation, a person, and all you may know about them, and deciding how trustworthy they are. The ATM errand requires a very high level of trust, whereas watching your stuff needs much less, but not zero.

You may not even trust yourself completely, but you can definitely trust that actions taken by you were taken by you. In this way, trust in a zero trust network always originates with the operator. Trust in a zero trust network seems contradictory, though it is important to understand that when you have no inherent trust, you must source it from somewhere and manage it carefully. There's a small wrinkle though: the operator won't always be available to authorize and grant trust! Plus, the operator

just doesn't scale :). Luckily, we know how to solve that problem—we delegate trust as shown in Figure 2-1.

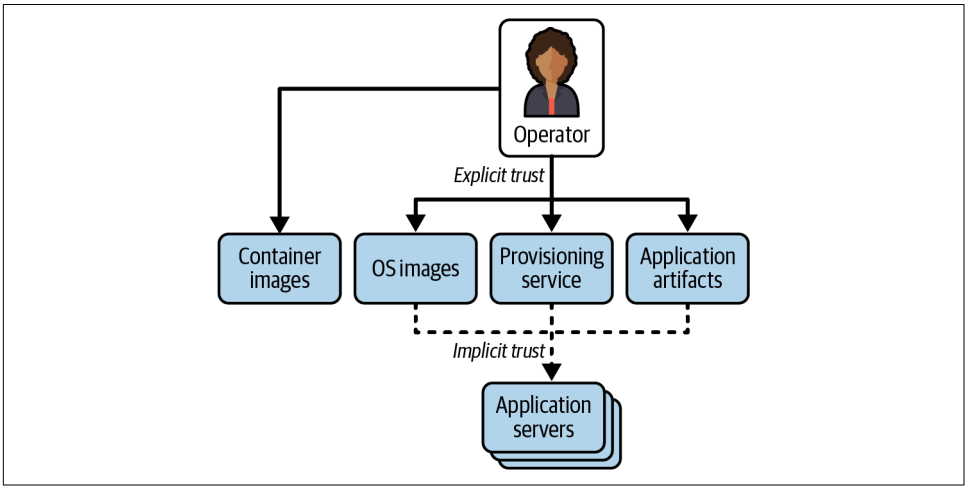

Figure 2-1. An operator declares trust in a particular system, which can in turn trust another, forming a trust chain

Trust delegation is important because it allows us to build automated systems that can grow to a large scale and operate in a secure and trusted way with minimal human intervention. The trusted operator must assign some level of trust to a system, enabling it to take actions on behalf of the operator. A simple example of this is autoscaling. You want your servers to provision themselves as needed, but how do you know a new server is one of yours and not some other random server? The operator must delegate the responsibility to a provisioning system, granting it the ability to assign trust to, and create, new hosts. In this way, we can say that we trust the new server is indeed our own, because the provisioning system has validated that it has taken the action to create it, and the provisioning system can prove that the operator has granted it the ability to do so. This flow of trust back to the operator is often referred to as a trust chain, and the operator can be referred to as a trust anchor.

Threat Models

Defining threat models is an important first step when designing a security architecture. A threat model enumerates the potential attackers, their capabilities and resources, and their intended targets. Threat models will normally define which attackers are in scope, rationally choosing to mitigate attacks from weaker adversaries before moving on to more difficult adversaries.

A well-defined threat model can be a useful tool to focus security mitigation efforts. When building security systems, like most engineering exercises, there is a tendency

to focus on the fancier aspects of the engineering problem to the detriment of the more boring but still important parts. This tendency is especially worrisome in a security system, since the weakest link in the system is where attackers will quickly focus their attention. Therefore, the threat model serves as a mechanism for focusing our attention on a single threat and fully mitigating their attacks. Threat models can also be useful when prioritizing security initiatives. Fighting state-level actors is pointless if a system's security measures are insufficient to defend against a simple brute-force attack on a user's poor password. As such, it is important to start first with simpler personas when building a threat model.

Common Threat Models

There are many different techniques for threat modeling in the security field. Here are some of the more popular ones:

- STRIDE
- DREAD
- PASTA
- Trike
- VAST
- MITRE ATT&CK

The varying threat modeling techniques provide different frameworks for exploring the threat space. Each of them is after the same goal: to enumerate threats to the system and further enumerate the mitigating systems and processes for those threats. Different threat models approach the problem from different angles. Some modeling systems might focus on the assets that an attacker would be targeting. Others might look at each software component in isolation and enumerate all the attacks that could be applied to that system. Finally, some models might look at the system as a whole from the attacker's perspective: as an attacker, how might I approach penetrating this system? Each of these approaches has pros and cons. For a well-diversified mitigating strategy, a blend of the three approaches is ideal.

If we look at the attacker-based threat modeling methodology, we are able to categorize attackers into a list of increasing capabilities (ordered from least to most threatening):

Opportunistic attackers
: So-called script kiddies, who are unsophisticated attackers taking advantage of well-known vulnerabilities with no predetermined target.

Targeted attackers
: Attackers who craft specialized attacks against a particular target. Spear phishing and corporate espionage might fall into this bucket.

Insider threats
: A credentialed but everyday user of a system. Contractors and unprivileged employees generally fall into this bucket.

Trusted insider
: A highly trusted administrator of a system.

State-level actor
: Attackers backed by foreign or domestic governments and assumed to have vast resources and positioning capabilities to attack a target.

Categorizing threats like this is a useful exercise to focus discussion around a particular level to mitigate against. We will discuss which level zero trust targets in the next section.

Threats Versus Vulnerabilities?

While the terms threat and vulnerability may appear to be synonymous, they are fundamentally different in the context of security. A threat is an event (which can be based on hardware, software, a person, a process, or even nature) that has the potential to negatively impact a valuable resource. A vulnerability, on the other hand, is a flaw in a resource or its surroundings that allows a threat to be realized.

It is critical to document and manage vulnerabilities. As a result, standard bodies such as NIST and nonprofit organizations such as MITRE offer repositories and programs for vulnerability assessment and management:

Common Vulnerability Scoring System (CVSS)
: This is a vulnerability management system by NIST that is widely used. CVSS ranks the severity of an information security vulnerability and is used in many vulnerability scanning tools.

Common Vulnerabilities and Exposures (CVE)
: This is an MITRE-maintained list of publicly disclosed vulnerabilities and exposures.

National Vulnerability Database (NVD)
: This is a NIST database that is fully synchronized with the MITRE CVE list.

Zero Trust's Threat Model

RFC 3552 describes the internet threat model. Zero trust networks generally follow the internet threat model to plan their security stance. While reading the entire RFC is recommended, here is a relevant excerpt:

> The Internet environment has a fairly well understood threat model. In general, we assume that the end-systems engaging in a protocol exchange have not themselves been compromised. Protecting against an attack when one of the end-systems has been compromised is extraordinarily difficult. It is, however, possible to design protocols which minimize the extent of the damage done under these circumstances.
>
> By contrast, we assume that the attacker has nearly complete control of the communications channel over which the end-systems communicate. This means that the attacker can read any PDU (Protocol Data Unit) on the network and undetectably remove, change, or inject forged packets onto the wire. This includes being able to generate packets that appear to be from a trusted machine. Thus, even if the end-system with which you wish to communicate is itself secure, the Internet environment provides no assurance that packets which claim to be from that system in fact are.

Zero trust networks, as a result of their control over endpoints in the network, expand upon the internet threat model to consider compromises at the endpoints.

The response to these threats is generally to first harden the systems proactively against compromised peers, and then facilitate detection of those compromises. Detection is aided by scanning of devices and behavioral analysis of the activity from each device. Additionally, mitigation of endpoint compromise is achieved by frequent upgrades to software on devices, frequent and automated credential rotation, and in some cases, frequent rotation of the devices themselves.

An attacker with unlimited resources is essentially impossible to defend against, and zero trust networks recognize that. The goal of a zero trust network isn't to defend against all adversaries, but rather the types of adversaries that are commonly seen in a hostile network.

From our earlier discussion of attacker capabilities, a zero trust network is generally attempting to mitigate attacks up to and including attacks originating from a "trusted insider" level of access. Most organizations do not experience attacks that exceed this level of sophistication. Developing mitigations against these attackers will defend against the vast majority of compromises and would be a dramatic improvement for the industry's security stance.

Zero trust networks generally do not try to mitigate all state-level actors, though they do attempt to mitigate those attempting to compromise their systems remotely. State-level actors are assumed to have vast amounts of money, so many attacks that would be infeasible for lesser organizations are available to them. Additionally, local

governments have physical and legal access to many of the systems that organizations depend upon for securing their networks.

Defending against these localized threats is exceedingly expensive, requiring dedicated physical hardware, and most zero trust networks consider the more extreme forms of attacks (say, a vulnerability being inserted into a hypervisor that copies memory pages out of a VM) out of scope in their threat models. We should be clear that while security best practices are still very much encouraged, the zero trust model only requires the safety of information used to authenticate and authorize actions, such as on-disk credentials. Further requirements on endpoints, say full disk encryption, can be applied via additional policy.

Strong Authentication

Knowing how much to trust someone is useless without being able to associate a real-life person with that identity you know to trust. Humans have many senses to determine if the person in front of them is who they think they are. Turns out, combinations of senses are hard to fool.

Computer systems, however, are not so lucky. It's more like talking to someone on the phone. You can listen to their voice, read their caller ID, ask them questions...but you can't see them. Thus, we are left with a challenge: how can one be reasonably assured that the person (or system) on the other end of the line is in fact who they say they are?

Typically, operators examine the IP address of the remote system and ask for a password. Unfortunately, these methods alone are insufficient for a zero trust network, where attackers can communicate from any IP they please and insert themselves between you and a trusted remote host. Therefore, it is very important to employ strong authentication on every flow in a zero trust network. The most widely accepted method to accomplish this is a standard named X.509, which most engineers are familiar with. It defines a certificate standard that allows identity to be verified through a chain of trust. It's popularly deployed as the primary mechanism for authenticating Transport Layer Security (TLS), formerly Secure Sockets Layer (SSL) connections.

Certificates utilize two cryptographic keys: a public key and a private key. The public key is distributed, and the private key is held as a secret. The public key can encrypt data that the private key can decrypt, and vice versa, as shown in Figure 2-2. This allows one to prove they are in the presence of the private key by correctly decrypting a piece of data that was encrypted by the well-known (and verifiable) public key. In this way, identity can be validated without ever exposing the secret.

TLS Is Anonymous

The most widely consumed TLS configuration validates that the client is speaking to a trusted resource, but not that the resource is speaking to a trusted client. This poses an obvious problem for zero trust networks.

TLS additionally supports mutual authentication, in which the resource also validates the client. This is an important step in securing private resources; otherwise, the client device will go unauthenticated. More on zero trust TLS configuration in "Mutually Authenticated TLS (mTLS)" on page 180.

Certificate-based authentication lets us be certain that the person on the other end of the line has the private key, and also lets us be certain that someone listening in can't steal the key and reuse it in the future. It does, however, still rely on a secret, something that can be stolen. Not necessarily by listening in, but perhaps by a malware infection or physical theft.

So while we can validate that credentials are legitimate, we might not trust that they have been kept a secret. For this reason, it is desirable to use multiple secrets, stored in different places, which in combination grant access. With this approach, a potential attacker must steal multiple components.

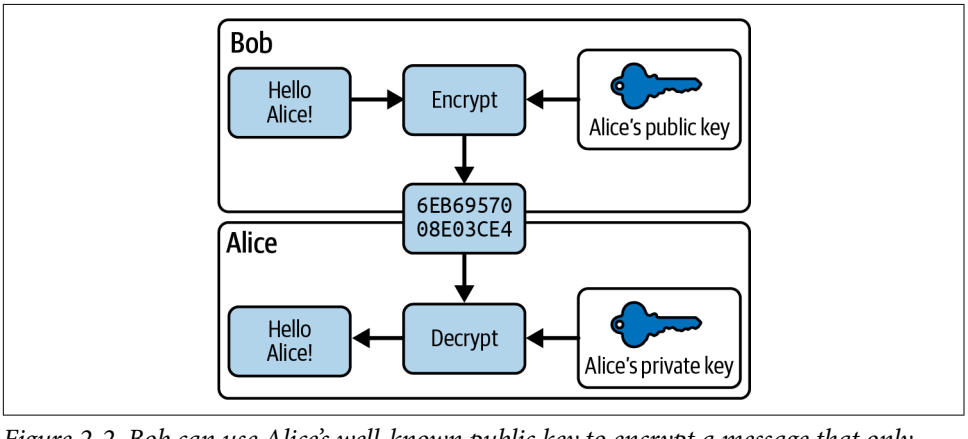

Figure 2-2. Bob can use Alice's well-known public key to encrypt a message that only Alice is able to decrypt

While having multiple components goes a long way in preventing unauthorized access, it is still conceivable that all these components can be stolen. Therefore, it is critical that all authentication credentials be time-boxed. Setting an expiration on credentials helps to minimize the blast radius of leaked or stolen keys and gives the operator an opportunity to reassert trust. The act of changing, or renewing, keys/passwords is known as credential rotation.

Credential rotation is essential for validating that no secrets have been stolen, and they should be revoked when required. Systems utilizing keys/passwords that are hard or impossible to rotate should be avoided at all costs, and when building new systems this fact should be taken into account early on in the design process. The rotation frequency of a particular credential is often inversely proportional to the cost of rotation.

Examples of Secrets Expensive to Rotate

- Certificates requiring external coordination
- Hand-configured service accounts
- Database passwords requiring downtime to reset
- A site-specific salt that cannot be changed without invalidating all stored hashes

Authenticating Trust

We spoke a little bit about certificates and public key cryptography. However, certificates alone don't solve the authentication issue. For instance, you can be assured that a remote entity is in possession of a private key by making an assertion using its public key. But how do you obtain the public key to begin with? Sure, public keys don't need to be secret, but you must still have a way to know that you have the right public key. Public key infrastructure, or PKI, defines a set of roles and responsibilities that are used to securely distribute and validate public keys in untrusted networks.

The goal of a PKI system is to allow unprivileged participants to validate the authenticity of their peers through an existing trust relationship with a mutual third party. A PKI system leverages what is known as a registration authority (RA) in order to bind an identity to a public key. This binding is embedded in the certificate, which is cryptographically signed by the trusted third party. The signed certificate can then be presented in order to "prove" identity, so long as the recipient trusts the same third party.

There are many types of PKI providers. The most popular two are certificate authorities (CAs) and webs of trust (WoTs). The former relies on a signature chain that is ultimately rooted in the mutually trusted party. The latter allows systems to assert the validity of their peers, forming a web of endorsements rather than a chain. Trust is then asserted by traversing the web until a trusted certificate is found. While this approach is in relatively wide use with Pretty Good Privacy (PGP) encryption, this book will focus on PKI systems that employ a CA, the popularity of which overshadows the WoT provider.

What Is a Certificate Authority?

Certificate authorities act as the trust anchor of a certificate chain. They sign and publish public keys and their bound identities, allowing unprivileged entities to assert the validity of the binding through the signature.

CA certificates are used to represent the identity of the CA itself, and it is the private key of the CA certificate that is used to sign client certificates. The CA certificate is well known, and is used by the authenticating entity to validate the signature of the presented client certificate. It is here that the trusted third-party relationship exists, issuing and asserting the validity of digital certificates on behalf of the clients. The trusted third-party position is very privileged. The CA must be protected at all costs, since its subversion would be catastrophic. Digital certificate standards like X.509 allow for chaining of certificates, which enables the root CA to be kept offline. This is considered standard practice in CA-based PKI security. We'll talk more about X.509 security in Chapter 5.

Importance of PKI in Zero Trust

All zero trust networks rely on PKI to prove identity throughout the network. As such, it acts as the bedrock of identity authentication for the majority of operations. Entities that might be authenticated with a digital certificate include:

- Devices
- Users
- Applications

Binding Keys to Entities

PKI can bind an identity to a public key, but what about a private key to the entity it is meant to identify? After all, it is the private key that we are really authenticating. It is important to keep the private key as close to the entity it was meant to represent as possible. The method by which this is done varies by the type of entity. For instance, a user might store a private key on a smart card in their pocket, whereas a device might store a private key in an onboard security chip. We'll discuss which methods best fit which entities in Chapters 5, 6, and 7.

Given the sheer number of certificates that a zero trust network will issue, it is important to recognize the need for automation. If humans are required in order to process certificate signing requests, the procedure will be applied sparingly, weakening the

overall system. That being said, you will likely wish to retain a human-based approval process for certificates deemed highly sensitive.

Private Versus Public PKI

PKI is perhaps most popularly deployed as a public trust system, backing X.509 certificates in use on the public internet. In this mode, the trusted third party is publicly trusted, allowing clients to authenticate resources that belong to other organizations.

While public PKI is trusted by the internet at large, it is not recommended for use in a zero trust network.

Some might wonder why this is. After all, public PKI has some defensible strengths. Factors like existing utilities/tooling, peer-reviewed security practices, and the promise of a better time to market are all attractive. There are, however, several drawbacks to public PKI that work against it. The first is cost.

The public PKI system relies on publicly trusted authorities to validate digital certificates. These authorities are businesses of their own, and usually charge a fee for signing certificates. Since a zero trust network has many certificates, the signing costs associated with public authorities can be prohibitive, especially when considering rotation policies.

Another significant drawback to public PKI is the fact that it's hard to fully trust the public authorities. There are lots of publicly trusted CAs, operating in many countries. In a zero trust network leveraging public PKI, any one of these CAs can cut certificates that your network trusts. Do you trust the laws and the governments associated with all of those CAs too? Probably not. While there are some mitigation methods here, like certificate pinning or installing trust in a single public CA, it remains challenging to retain trust in a disjointed organization.

Finally, flexibility and programmability can suffer when leveraging public CAs. Public CAs are generally interested in retaining the public's trust, so they do employ good security measures. This might include policies about how certificates are formed, and what information can be placed where. This can adversely affect zero trust authentication in that it is often desirable to store site-specific metadata in the certificate, like a role or a user ID. Additionally, not all public CAs provide programmable interfaces, making automation a challenge.

Public PKI Is Better than None

While the drawbacks associated with public PKI are significant, and the authors heavily discourage its use within a zero trust network, it remains superior to no PKI at all. A well-automated PKI system is the first step, and work will be required in this area no matter which PKI approach you choose. The good news is that if you choose to leverage public PKI initially, there is a clear path to switch to private PKI once the

risk becomes too great. It begs the question, however, if it is even worth the effort, since automation of those resources will still be required.

Least Privilege

The principle of least privilege is the idea that an entity should be granted only the privileges it needs to get its work done. By granting only the permissions that are always required, as opposed to sometimes desired, the potential for abuse or misuse by a user or application is greatly reduced.

In the case of an application, that usually means running it under a service account, in a container or jail, etc. In the case of a human, it commonly manifests itself as policies like "only engineers are allowed access to the source code." Devices must also be considered in this regard, though they often assume the same policies as the user or application they were originally assigned to.

Privacy as Least Privilege

The application of encryption in the name of privacy is an often overlooked application of least privilege. Who really needs access to the packet payload?

Another effect of this principle is that if you do need elevated access, you retain those access privileges for only as long as you need them. It is important to understand what actions require which privileges so that they may be granted only when appropriate. This goes one step beyond simple access control reviews.

This means that human users should spend most of their time executing actions using a nonprivileged user account. When elevated privileges are needed, the user needs to execute those actions under a separate account with higher privileges. On a single machine, elevating one's privileges is usually accomplished by taking an action that requires the user to authenticate themselves. For example, on a Unix system, invoking a command using the sudo command will prompt the user to enter their password before running that command as a different role. In GUI environments, a dialog box might appear requiring the user's password before performing the risky operation. By requiring interaction with the user, the potential for malicious software to take action on behalf of the user is (potentially) mitigated.

In a zero trust network, users should similarly operate in a reduced privilege mode on the network most of the time, only elevating their permissions when needed to perform some sensitive operation. For example, an authenticated user might freely access the company's directory or interact with project planning software. Accessing a critical production system, however, should require additional confirmation that the user or the user's system is not compromised. For relatively low-risk actions,

this privilege elevation could be as simple as re-prompting for the user's password, requesting a second factor token, or sending a push notification to the user's phone. For high-risk access, one might choose to require active confirmation from a peer via an out-of-band request.

Human-Driven Authentication

For particularly sensitive operations, an operator may rely on the coordination of multiple humans, requiring a number of people to be actively engaged in order to authenticate a particular action. Forcing authentication actions into the real world is a good way to ensure a compromised system can't interfere with them. Be careful, however—these methods are expensive and will become ineffective if employed too frequently.

Like users, applications should also be configured to have the fewest privileges necessary to operate on the network. Sadly, applications deployed in a corporate setting are often given fairly wide access on the network. Either due to the difficulty of defining policies to rein in applications, or the assumption that compromised users are the more likely target, it's now become commonplace for the first step in setting up a machine to be disabling the application security frameworks that are meant to secure the infrastructure.

Beyond the traditional consideration of privilege for users and applications, zero trust networks also consider the privilege of the device on the network. It is the combination of user or application and the device being used that determines the privilege level granted. By joining the privilege of a user to the device being used to access a resource, zero trust networks are able to mitigate the effects of lost or compromised credentials. Chapter 3 will explore how this marriage of devices and users works in practice.

Privilege in a zero trust network is more dynamic than in traditional networks. Traditional networks eventually converge on policies that stay relatively static. If new use cases appear that require greater privilege, either the requestor must lobby for a change in policy; or, perhaps more frequently, ask someone with greater privilege (a sysadmin, for example) to perform the operation for them. This static definition of policy presents two problems. First, in more permissive organizations, privilege will grow over time, lessening the benefit of least privilege. Second, in both permissive and restrictive organizations, admins are given greater access, which has resulted in malicious actors purposefully targeting sysadmins for phishing attacks.

A zero trust network, by contrast, will use many attributes of activity on the network to determine a riskiness factor for the access being requested currently. These attributes could be temporal (access outside of the normal window activity for that user is more suspicious), geographical (access from a different location than the user

was last seen), or even behavioral (access to resources the user does not normally access). By considering all the details of an access attempt, the determination of whether the action is authorized or not can be more granular than a simple binary answer. For example, access to a database by a given user from their normal location during typical working hours would be granted, but access from a new location at different working hours might require the user to authenticate using an additional factor.

The ability to actively adjust access based on the riskiness of activity on a network is one of the several features that make zero trust networks more secure. By dynamically adjusting policies and access, these networks are able to respond autonomously to known and unknown attacks by malicious actors.

Dynamic Trust

Managing trust is perhaps the most difficult aspect of running a secure network. Choosing which privileges people and devices are allowed on the network is time-consuming, constantly changing, and directly affects the security posture the network presents. Given the importance of trust management, it's surprising how under-deployed network trust management systems are today.

Definition of trust policies is typically left as a manual effort for security engineers. Cloud systems might have managed policies, but those policies provide only basic isolation (e.g., super user, admin, regular user), which advanced users typically outgrow. Perhaps in part due to the difficulty of defining and maintaining them, requests to change existing policies can be met with resistance. Determining the impact of a policy change can be difficult, so prudence pushes the administrators toward the status quo, which can frustrate end users and overwhelm system administrators with change requests.

Policy assignment is also typically a manual effort. Users are granted policies based on their responsibilities in the organization. This role-based policy system tends to produce large pools of trust in the administrators of the network, weakening the overall security posture of the network. These pools of trust have created a market for hackers to hunt for system admin accounts, like the Conti ransomware group, which seeks out and compromises system administrators as the final step in their ransomware attack. Cybercriminal organizations, such as LAPSUS$, actively recruit company insiders to assist them in gaining access to corporate networks via VPN or Citrix, as shown in Figure 2-3. Their goal is to gain access to the network, preferably using employee credentials that grant them privileged access. LAPSUS$ has used this method successfully against Samsung, NVIDIA, Vodafone, Microsoft, and Okta in recent years. Perhaps the gold standard for a secure network is one that does not have access to highly privileged system administrators.

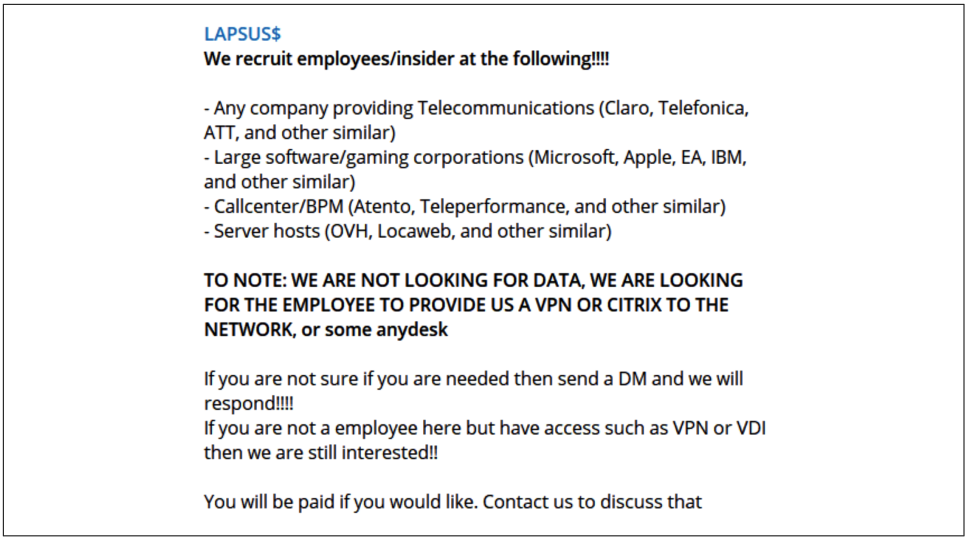

Figure 2-3. Cybercrime group LAPSUS$'s message on the Telegram channel for recruiting employees/insiders

These pools of trust underscore the fundamental issue with how trust is managed in traditional networks: policies are not nearly dynamic enough to respond to the threats being leveled against the network. Mature organizations will have some sort of auditing process in place for activity on their network, but audits can be done too infrequently, and are frankly so tedious that doing them well is difficult for humans. How much damage could a rogue sysadmin do on a network before an audit discovered their behavior and mitigated it? A more fruitful path might be to rethink the actor/trust relationship, recognizing that trust in a network is ever-evolving and based on the previous and current actions of an actor within the network.

This model of trust, considering all the actions of an actor and determining their trustworthiness, is not novel. Credit agencies have been performing this service for many years. Instead of requiring organizations like retailers, financial institutions, or even an employer to independently define and determine one's trustworthiness, a credit agency can use actions in the real world to score and gauge the trustworthiness of an individual. The consuming organizations can then use their credit score to decide how much trust to grant that person. In the case of a mortgage application, an individual with a higher credit score will receive a better interest rate, which mitigates the risk to the lender. In the case of an employer, one's credit score might be used as a signal for a hiring decision. On a case-by-case basis, these factors can feel arbitrary and opaque, but they serve a useful purpose; providing a mechanism for defending a system against arbitrary threats by defining policy based not only on specifics, but also on an ever-changing and evolving score, which we call a "trust score."

Trust Score

A zero trust network utilizes trust scores to define trust within the network, as shown in Figure 2-4. Instead of defining binary policy decisions assigned to specific actors in the network, a zero trust network will continuously monitor the actions of an actor on the network to update their trust score. This score can then be used to define policy in the network based on the severity of breach of that trust (Figure 2-5). A user viewing their calendar from an untrusted network might require a relatively low trust score. However, if that same user attempted to change system settings, they would require a much higher score and their request would be denied or flagged for immediate review. Even in this simple example, one can see the benefit of a score: we can make fine-grained determinations on the checks and balances needed to ensure trust is maintained.

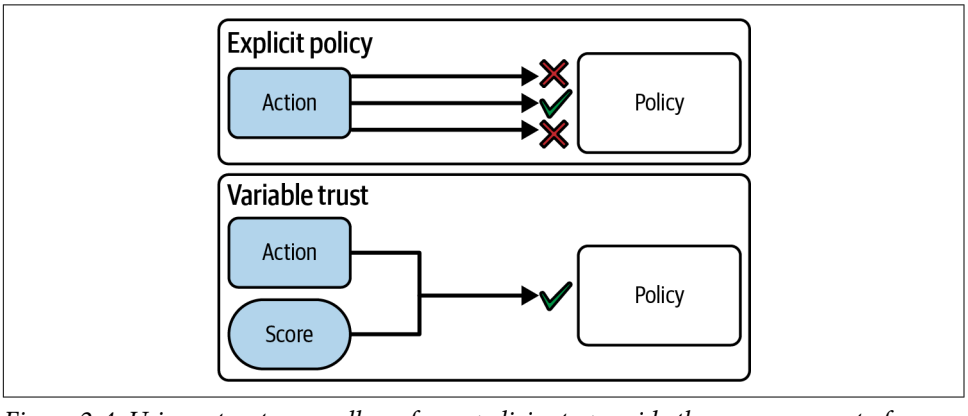

Figure 2-4. Using a trust score allows fewer policies to provide the same amount of access

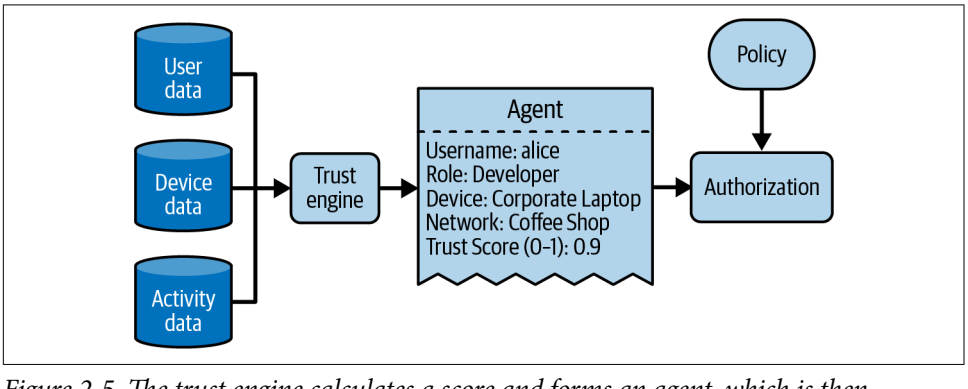

Figure 2-5. The trust engine calculates a score and forms an agent, which is then compared against policy in order to authorize a request. We'll talk more about agents in Chapter 3.

Monitoring Encrypted Traffic

Since practically all flows in a zero trust network are encrypted, traditional traffic inspection methods don't work as well as intended. Instead, we are limited to inspecting what we can see, which in most cases is the IP header and perhaps the next protocol header (like TCP in the case of TLS). If a load balancer or proxy is in the request path, however, there is an opportunity for deeper inspection and authorization, since the application data will be exposed for examination.

Clients begin sessions as untrusted. They must accumulate trust through various mechanisms, eventually accruing enough to gain access to the service they're requesting. Strong authentication proving that a device is company owned, for instance, might accumulate a good bit of trust, but not enough to allow access to the billing system. Providing the correct RSA token might give you a good bit more trust, enough to access the billing system when combined with the trust inferred from successful device authentication.

Strong Policy as a Trust Booster

Things like score-based policies, which can affect the outcome of an authorization request based on a number of variables like historical activity, drastically improve a network's security stance when compared to static policy. Sessions that have been approved by these mechanisms can be trusted more than those that haven't. In turn, we can rely (a little bit) less on user-based authentication methods to accrue the trust necessary to access a resource, improving the overall user experience.

Challenges with Trust Scores

Switching to a trust score model for policies, which we introduced in Chapter 1, is not without drawbacks. The first hurdle is whether a single score is sufficient for securing all sensitive resources. In a system where a trust score can decrease based on user activity, a user's score can also increase based on a history of trustworthy activity. Could it be possible for a persistent attacker to slowly build their credibility in a system to gain more access?

Perhaps slowing an attacker's progress by requiring an extended period of "normal" behavior would be sufficient to mitigate that concern, given that an external audit would have more opportunity to discover the intruder. Another way to mitigate that concern is to expose multiple pieces of information to the control plane so that sensitive operations can require access from trusted locations and persons. Binding a trust score to device and application metadata allows for flexible policies that can

declare absolute requirements yet still capture the unknown unknowns through the computed trust score.

Loosening the coupling between security policy and a user's organizational role can cause confusion and frustration for end users. How can the system communicate to users that they are denied access to some sensitive resource from a coffee shop, but not from their home network? Perhaps we present them with increasingly rigorous authentication requirements? Should new members be required to live with lower access for a time before their score indicates that they can be trusted with higher access? Maybe we can accrue additional trust by having the user visit a technical support office with the device in question. All of these are important points to consider. The route one takes will vary from deployment to deployment.

Control Plane Versus Data Plane

The roles of the zero trust control plane and data plane are introduced in Chapter 1. The distinction between the control plane and the data plane is a concept that is commonly referenced in network systems. The basic idea is that a network device has two logical domains with a clear interface between those domains. The data plane is the relatively dumb layer that manages traffic on the network. Since that layer is handling high rates of traffic, its logic is kept simple and often pushed to specialized hardware. The control plane, conversely, could be considered the brains of the network device. It is the layer that system administrators apply configuration to, and as a result is more frequently changed as policy evolves.

Since the control plane is so malleable, it is unable to handle a high rate of traffic on the network. Therefore, the interface between the control plane and the data plane needs to be defined in such a way that nearly any policy behavior can be implemented at the data layer, with infrequent requests being made to the control plane (relative to the rate of traffic).

A zero trust network also defines a clear separation between the control plane and data plane. The data plane in such a network is made up of the applications, firewalls, proxies, and routers that directly process all traffic on the network. These systems, being in the path of all connections, need to quickly make determinations of whether traffic should be allowed. When viewing the data plane as a whole, it has broad access and exposure throughout the system, so it is important that the services on the data plane cannot be used to gain privilege in the control plane and thereby move laterally within the network. We'll discuss control plane security in Chapter 4.

The control plane in a zero trust network is made up of components that receive and process requests from data plane devices that wish to access (or grant access to) network resources, as shown in Figure 2-6. These components will inspect data about the requesting system to make a determination on how risky the action is, and examine relevant policy to determine how much trust is required. Once a determination

is made, the data plane systems are signaled or reconfigured to grant the requested access.

The mechanism by which the control plane affects change in the data plane is of critical importance. Since the data plane systems are often the entry point for attackers into a network, the interface between it and the control plane must be clear, helping to ensure that it cannot be subverted to move laterally within the network. Requests between the data plane and control plane systems must be encrypted and authenticated using a nonpublic PKI system to ensure that the receiving system is trustworthy. The control/data plane interface should resemble the user/kernel space interface, where interactions between those two systems are heavily isolated to prevent privilege escalation.

This concern with the interface between the control plane and the data plane belies another fundamental property of the control plane: the control plane is the trust grantor for the entire network. Due to its far-reaching control of the network's behavior, the control plane's trustworthiness is critical. This need to have an actor on the network with a highly privileged role presents a number of interesting design requirements.

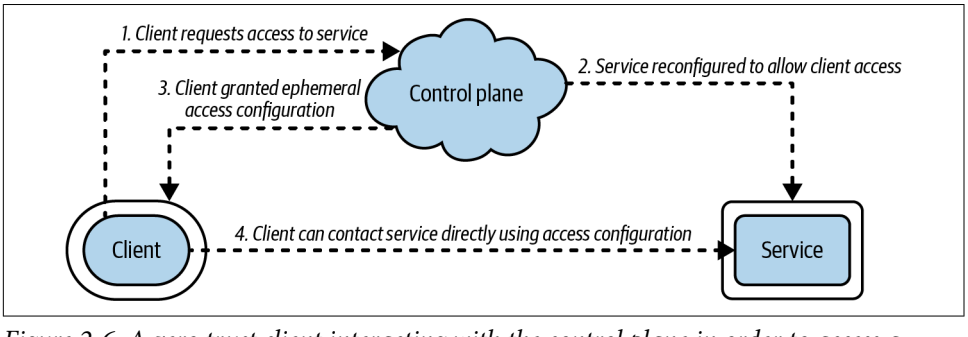

Figure 2-6. A zero trust client interacting with the control plane in order to access a resource

The first requirement is that the trust granted by the control plane to another actor in the data plane should have limited real-time value. Trust should be temporary, requiring regular check-ins between the truster and trustee to ensure that the continued trust is reasonable. When implementing this tenet, leased access tokens or short-lifetime certificates are the most appropriate solution. These leased access tokens should be validated not just within the data plane (e.g., when the control plane grants a token to an agent to move through the data plane), but also between the interaction between the data plane and the control plane. The control plane decides whether or not to allow a request by considering all of its factors. Because trust is temporary and time bound, if and when the factors that led the control plane decision to allow the request in the first place have changed, it may coordinate with the data plane

to revoke the request access to the resource. By limiting the window during which the data plane and control plane can interact with a particular set of credentials, the possibility for physical attacks against the network is mitigated.

Summary

This chapter discussed the critical systems and concepts that are needed to manage trust in a zero trust network. Many of these ideas are common in traditional network security architectures, but it is important to lay the foundation of how trust is managed in a network without any.

Trust originates from humans and flows into other systems via trust mechanisms that a computer can operate against. This approach makes logical sense: a system can't be considered trusted unless the humans who use it feel confident that it is faithfully executing their wishes.

Security has frequently been viewed as a set of best practices, which are passed down from one generation of engineers to the next. Breaking out of this cycle is important, since each system is unique, so we discussed the idea of threat models. Threat models attempt to define the security posture of a system by enumerating the threats against the system and then defining the mitigating systems and processes that anticipate those threats. While a zero trust network assumes a hostile environment, it is still fundamentally grounded in the threat model, which makes sense for the system. We enumerated several present-day threat-modeling techniques so that readers can dig deeper. We also discussed how the zero trust model is based on the internet threat model and expands its scope to the endpoints that are under the control of zero trust system administrators.

Having trust in a system requires the use of strong authentication throughout the system. We discussed the importance of this type of authentication in a zero trust network. We also briefly talked about how strong authentication can be achieved with today's technology. We will discuss these concepts more in later chapters. To effectively manage trust in a network, you must be able to positively identify trusted information, particularly in the case of authentication and identity. Public key infrastructure (or PKI) provides the best methods we have today for asserting validity and trust in a presented identity. We discussed why PKI is important in a zero trust network, the role of a certificate authority, and why private PKI is preferred over public PKI.

Least privilege is one of the key ideas in these types of networks. Instead of constructing a supposedly safe network over which applications can freely communicate, the zero trust model assumes that the network is untrustworthy, and as a result, components on the network should have minimal privileges when communicating.

We explained what the concept of least privilege is in this context, and how it is similar to and different from least privilege in standalone systems.

One of the most exciting ideas of zero trust networks is the idea of variable trust. Network policy has traditionally focused on which systems are allowed to communicate in what manner. This binary policy framework can result in policy that is either too rigidly defined (creating human toil to continually adjust) or too loosely defined (resulting in security systems that assert very little). Additionally, policy that is defined based on concrete details of interactions will invariably be stuck in a cat-and-mouse game of adjusting policy based on past threats. The zero trust model leans on the idea of variable trust, a numeric value representing the level of trust in a component. Policy can then be written against this number, effectively capturing a number of conditions without complicating the policy with edge cases. By defining policy in less-concrete details, and considering the trust score while making an authorization decision, the authorization systems are able to adjust to novel threats.

CHAPTER 3
Context-Aware Agents

Imagine you're in a security-conscious organization. Each employee is given a highly credentialed laptop to do their work. With today's blending of work and personal life, some also want to view their email and calendar on their phone. In this hypothetical organization, the security team applies fine-grained policy decisions based on which device the user is using to access a particular resource.

For example, perhaps it is permissible to commit code from the employee's company-issued laptop, but doing so from their phone would be quite a strange thing. Since source code access from a mobile device is decidedly riskier than from an enrolled laptop, the organization blocks such access. That said, an employee accessing corporate email from a personal device may be permitted. As you will learn throughout this chapter, context is critical when making decisions in a zero trust environment.

The story described here is a fairly typical application of zero trust, in that multiple factors of authentication and authorization take place, concerning both the user and the device. In this example, however, it is clear that one factor has influenced the other—a user who might "normally" have source code access won't enjoy such access from their mobile device. Additionally, this organization does not want authenticated users to commit code from just any trusted device—it expects users to use their organization's device.

This marriage of user and device is a new concept that zero trust introduces, which we are calling an agent. In a zero trust network, it is insufficient to treat the user and device separately, because policy often needs to consider the two together to accurately enforce desired behavior. By defining an agent formally in the system, we are able to capture this relationship and use it to drive policy decisions.

This chapter will define what an agent is and how it is used. In doing that, we will discuss the types of data that are included in an agent, some of which is potentially

sensitive. Given the nature of that data, we will discuss when and how an agent should be exposed to data plane systems. An agent, being a new concept, could benefit from standardization. We will explore the benefits of standardizing this agent.

What Is an Agent?

An agent is a combination of data known about the actors in a request. This typically consists of a user (also known as the subject), a device (an asset used by the subject to make the request), and an application (web app, mobile app, API endpoint, etc.). Traditionally, these entities have been authorized separately, but zero trust networks recognize that policy is best captured as a combination of all participants in a request. By authorizing the entire context of a request, the impact of credential theft is greatly mitigated.

It's best to think of an agent as an ephemeral entity that is formed on demand to evaluate a policy. The data that is used to form an agent—user and device information—will typically be stored in persistent storage and queried to form an agent.

When this data is queried, the union of the data at that point in time is what we call an agent. The benefit of this approach is that any changes to the data used to form an agent will change the agent itself. For example, if a user's role changes, the agent used to evaluate policy for that user will also change. If a device is unenrolled from the system, any agents associated with that device will no longer be valid.

What Is a Subject?

The term "user" is commonly used to refer to user identity, but it is important to understand that the term "subject" is also used, particularly by standard bodies, such as the National Institute of Standards and Technology (NIST) and others, to define both human and nonhuman users (like headless or machine identities). This distinction will be emphasized further in subsequent chapters when we examine users in Chapter 6 and devices in Chapter 5. When we talk about the user in the rest of this chapter, we mean both human and machine identities.

Agent Volatility

Some fields in the agent are made available specifically to mitigate against active attacks, and are therefore expected to change rapidly, relative to the infrequent changes that IT organizations normally expect. Trust scores are an example of this type of dynamic data. Trust score systems can evaluate each request in the network, using that activity feed to update the trust scores of users, applications, and devices. Therefore, in order for a trust score to mitigate a novel attack, it needs to be updated as close to real time as possible. Chapter 4 goes into greater detail about trust scores.

In addition to rapidly changing data, agents will frequently have sparse data. A device undergoing bootstrapping is an example scenario where the agent will have less data when compared to a mature device. During the bootstrapping process, little is known about the device, yet it must still interact with corporate infrastructure to perform tasks like device enrollment and software installation. In this case, the bootstrapping device is not yet assigned to a user and can run into problems if policy expects an assigned user to be present in the agent. This scenario should be expected and reflected in the authorization policy.

Sparse data isn't just found in bootstrapping scenarios. Autonomous systems in a zero trust network will frequently have sparse data when compared to human-operated systems. These systems, for example, will likely not authenticate the user account the application runs under, relying instead on the security of the configuration management system that created that user.

What's in an Agent?

The granularity of data contained within an agent can vary based on needs and maturity. It can be as high level as a user's name or a device's manufacturer, or as low level as serial numbers and place of residence or issue. Note that the more detailed data is more likely to have data cleanliness issues, which must be dealt with.

Agent Data Fields

The type of data stored in an agent can greatly vary in both presence and granularity. Here are some examples of data that one might find in an agent:

- Agent trust score
- User trust score
- User role or entitlements
- User groups
- User location
- User authentication method (MFA, password, etc.)
- Device trust score
- Device manufacturer
- Host operating system manufacturer and version
- Hardware security module (HSM) manufacturer and version
- Trusted platform module (TPM) manufacturer and version
- Current device location
- IP address

Another point of consideration is if the data contained in the agent is trusted or not. For instance, device data populated during the procurement process is more trusted than device data that is reported back from an agent running on it. This difference in trust arises from difficulties in ensuring the accuracy and integrity of the reported information if the device is compromised.

How Is an Agent Used?

When making an authorization decision in a zero trust network, it is the agent that is in fact authorized. While it is tempting to authorize the device and user separately, this approach is not recommended. Since the agent is the entity that is authorized, it is also the thing against which policy is written.

As noted in the previous section, the agent carries many pieces of information. So while more "traditional" authorization information like IP addresses can still be used, leveraging the agent also unlocks the use of "nontraditional" authorization information like device type or city of residence. As such, zero trust network policy is written considering the agent as a whole, as opposed to crafting disjointed user and device policies. Using an agent to drive authorization policy encourages authors to consider the totality of the communication context. The marriage of user and device is very important in zero trust authorization decisions, and co-locating the data in an agent makes it difficult to ignore one or the other. As with other portions of the zero trust architecture, lowering the barrier to entry is key, and co-locating the data to make device/user comparisons easier is no different.

Data Co-Location

When user and device data are combined or co-located in a request, they then form an agent. Thus, the overall context of the request becomes much clearer.

Consider the following scenario: Adam requests access to a high-business-impact quarterly sales report via his iPhone, running the iOS operating system, which he uses as part of Bring Your Own Device (BYOD). If his iPhone does not have any mobile device management solution installed on it that can perform policy enforcement, his request may be denied because the device used is not considered trustworthy. The combination of user and device attributes is critical here; otherwise, Adam has access to the report as a user.

An agent, being the primary actor in the network, plays an additional role in the calculation of trust scores. The trust engine can use recorded actions, in addition to data contained within the agent itself, to score agents for their trustworthiness. This trust score will then be exposed as an additional attribute of the agent against which

most policy should be defined. We'll talk more about how the trust score is calculated in Chapter 4.

Agents Are Not for Authentication

It is important to understand the difference between authentication and authorization in the context of an agent. Agents serve solely as authorization components and do not play any part in authentication. In fact, authentication is a precursor to agent formation and is generally performed separately for user and device. For example, devices could be authenticated with X.509 certificates, while users might be authenticated through a traditional multifactor approach.

Following successful authentication, the canonical identifiers for users and devices can be used to form an agent and its details. A device-specific certificate might be used as the canonical identifier for the device and therefore be used to populate information like device type or device owner. Similarly, a username might serve as the lookup key to populate user information like their role in the company.

Typically, authentication is session oriented, but in the case of authorization, it is best to be request oriented. As a result, caching the outcome of an authentication request is permissible, but caching an agent or the result of an authorization request is ill-advised. This is because details in the agent, which are used to make authorization decisions, can change rapidly based on a number of factors, and it is desirable to make authorization decisions using the latest data. This is in contrast to authentication materials, which change much less often and don't directly affect authorization itself.

Finally, the act of generating an agent should be as lightweight as possible. If agent generation is expensive, it will discourage frequent authorization requests due to performance reasons. We will talk more about how performance affects authorization in Chapter 4.

Revoke Authorization First, Credentials Second

Successful authentication is the act of proving one's identity to a remote system. That verified identity is then used to determine if the user actually has rights to access the resource in question (the authorization). If access must be revoked, updating authorization is more effective than changing authentication credentials. This is doubly so when considering that authentication results are typically cached and assigned to a session identifier. The act of validating an authenticated session is really an authorization decision.

How to Expose an Agent?

The data contained in an agent is potentially sensitive. Personally identifiable user information (e.g., name, address, phone number) will usually be present in the agent to facilitate detailed authorization decisions. This data should be treated with care to protect the privacy of users.

The sensitive nature of the data extends beyond users, however. Device details can also be sensitive data when they fall into the hands of a determined attacker. An attacker with detailed knowledge of a user's device could use that data to craft a targeted remote attack, or even learn a pattern of that user's physical location to steal the device.

To adequately secure the sensitive agent details, the entirety of the agent lifecycle should be contained to trusted control plane systems, which themselves are heavily secured. These systems should be logically and physically separated from the data plane systems, have clear boundaries, and change infrequently.

Most policy decisions will be made in the control plane systems, since the agent data is needed to make those decisions. However, it will often be the case that the authorization engine in the control plane is not in the best position to enforce application-centric policy, despite its ability to enforce authorization on a request-by-request basis. This is especially so in user-facing systems. As a result, some agent details will need to be exposed to data plane systems.

Let's look at an example. An administrative application stores details on all the customers of a particular company. This system exposes that data to employees based on their role within the company. A search feature allows employees to search within the subset of data that they are allowed to access. The application needs to implement this logic, and it needs access to the roles of the users to do so.

To allow applications to implement their own fine-grained authorization logic, agent details can be exposed to applications via a trusted communication channel. This could be as simple as injecting headers into network requests that flow through a reverse proxy. The proxy, being a zero trust control plane system, can view the agent to enforce its own authorization decisions and expose a subset of the data to the downstream application for further authorization.

Exposing agent details to the downstream application can also be useful to enable compatibility with preexisting applications that have a rich authorization system. This compatibility goal highlights that agent details should be exposed to the application in a format that is preferred by the application. For third-party applications, the format of the agent data will vary. For first-party applications, a common structure for the agent data will ease management of the system.

Rigidity and Fluidity, at the Same Time

Knowing the format of an agent, and where to find particular pieces of data within it, is very important when considering how and by what it will be consumed. The "coordinates" of certain pieces of data must be fixed and well known in order to ensure consistency across control plane systems. A good analogy here is the schema of a relational database, which applications accessing the data must have knowledge of in order to extract the right pieces of information.

This data compatibility is extremely important when it comes to implementing and maintaining zero trust control plane systems. Zero trust networks, particularly more mature ones, are likely to construct an agent from multiple systems and data sources. Without a schema of sorts, not only will it be difficult to surface the data in a consistent manner, but it will also contribute negatively to the amount of effort required to introduce new control plane systems or agent data, something that is considered critical for a maturing zero trust network.

One thing to keep in mind, however, is that agent data is likely to be fairly sparse, thanks to the practically unavoidable data cleanliness issues encountered in source systems like device inventories. The result is a "best-effort" agent, where many fields may be unpopulated for one reason or another. Rather than seeking data cleanliness (a problem that only gets harder with scale), it is best to accept reality and craft policy that understands that not all data may be present. So while one may still require a particular piece of data to be present in the agent, it is a useful thought exercise to consider alternative pieces of data that are appropriate replacements in its absence.

Standardization Desirable

One might wonder how it would be possible to standardize a data format that is so seemingly inextricably tied to the organization consuming it. After all, an agent is likely to contain information types that relate to business logic or other proprietary/local information. Is standardization even feasible in such a case?

Luckily, there are already some standards out there defining data formats that behave in such a way. One of the best examples is the Simple Network Management Protocol (SNMP) and its associated management information base (MIB).

The SNMP is a protocol frequently used for network device management, allowing devices to expose data to operators and management systems in a standard yet flexible way. The MIB component describes the format of the data itself, which is a collection of OIDs, or object identifiers. Each OID describes (and is reserved for) a particular piece of data and is registered with the ISO, or International Organization for Standardization. This lends itself well to widely accepted "coordinates" for certain pieces of data. Let's look at an example, shown in Figure 3-1, of a simplified set of nodes in an OID tree.

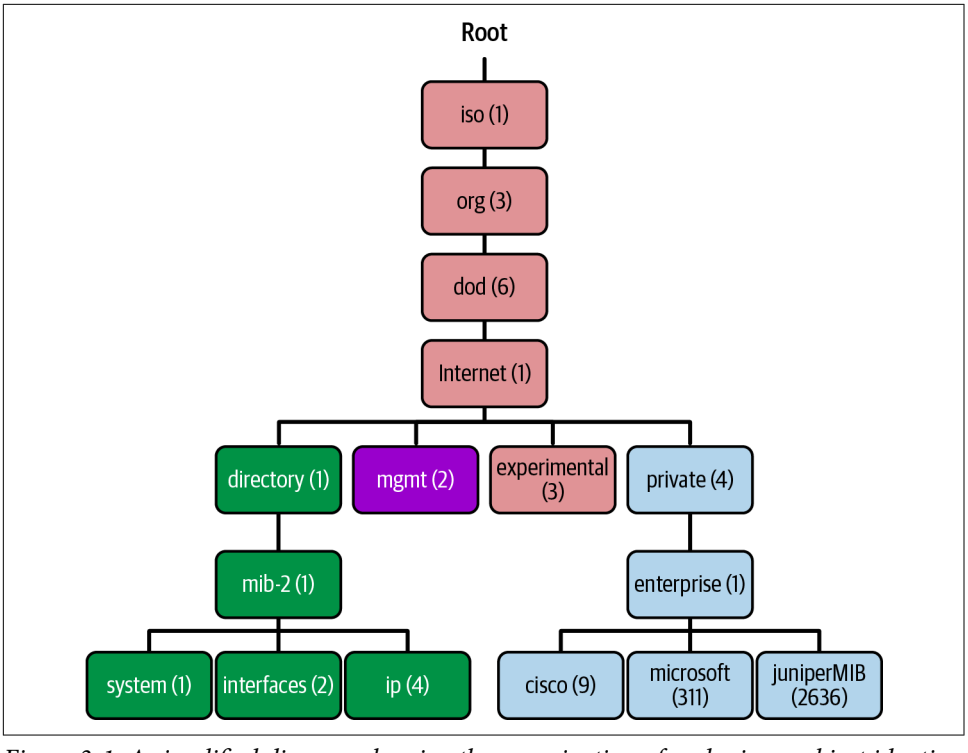

Figure 3-1. A simplified diagram showing the organization of nodes in an object identifier (OID) tree

In this example, the "IP" node and associated data would be addressed as 1.3.6.1.1.1.4. An MIB arranges and gives color to a set of OIDs. For example, a Cisco MIB might provide definitions for all OIDs under the 1.3.6.1.4.1.9 portion of the tree, including human-readable descriptions.

Of course, this registered list can be extended, and oftentimes chunks of OID space are carved out for organizations or manufacturers. In this way, an OID can be compared to an IP address, where an IP address globally identifies a computer system, and an OID globally identifies a piece of data.

Unfortunately, there is no good OID equivalent of private IP address space, which would be useful for ad hoc or site-specific data. The best available compromise is to register for a Private Enterprise Number with IANA, which will give you a dedicated OID prefix for private use. Luckily, such registration is free and with few questions asked. There have been some efforts to create a private range similar to that found in IP. However, such efforts have been unsuccessful. Despite the lack of a truly free/private OID space for experimental or internal use, the SNMP remains a useful analogy to make when considering the standardization of an agent. It describes

the format and packaging of a set of data—data that is easily found and identified using unique OIDs—and how that data can be transmitted and understood from one system to another.

In the Meantime?

While there have been several developments in zero trust networks in recent years, and standard bodies such as NIST and others have issued architecture guidance, agent standardization remains primarily an implementation task. In the meantime, agents take the form of least resistance, given the needs of the implementor. Whether it be a JSON blob, signed and encrypted JSON Web Token (JWT), Protocol Buffers, FlatBuffers, or any other custom binary format, it is recommended to ensure that the data contained within it is flexible and easily extensible. Loose typing or no typing should be preferred over strong typing, as the latter will make introducing new data and systems more difficult. Pluggable design patterns may help in moving to a standardized agent in the future. However, this is far from required, and should not be pursued if they impede the adoption of agent authorization in your network.

The following is an illustration of a possible JSON script that carries an agent's information:

```
{
  "iss": "Wayne Corporation ",
  "iat": 1676938201,
  "exp": 1708474201,
  "aud": "APP01091",
  "sub": "bob@waynecorp.com",
  "given_name": "Bob",
  "email": "bob@waynecorp.com",
  "assigned_roles": [
    "Manager",
    "Project Administrator"
  ],
  "device_id": "D98VCVQ3JMBH",
  "device_patched": "yes",
  "device_os_version": "13.2 (22D49)",
  "device_os_type": "MacOS",
  "user_id": "UID1233",
  "user_location": "Dallas,TX",
  "user_ip_address": "1.2.3.4",
  "user_auth_method": "X.509",
  "trust_score": "7"
}
```

Once signed and encoded, the JSON yields a JWT that looks like the one below. Any common JWT decoder will work to decode it. For instance, by pasting the encoded token into a website like *https://jwt.io*, you can decode its content online. Note that you can also encrypt the token, though the example omits this step for clarity.

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJXYXluZSBDb3Jwb3JhdGlvbiAiLCJpYX-
QiOjE2NzY5MzgyMDEsImV4cCI6MTcwODQ3NDIwMSwiYXVkIjoiQVBQMDEwOTEiLCJzdWIiOi-
Jib2JAd2F5bmVjb3JwLmNvbSIsImdpdmVuX25hbWUiOiJCb2IiLCJlbWFpbCI6ImJvYkB3YXluZWNvcn
AuY29tIiwicm9sZSI6WyJNYW5hZ2VyIiwiUHJ-
vamVjdCBBZG1pbmlzdHJhdG9yIl0sImRldmljZV9pZCI6IkQ5OFZDVlEzSk1CSCI-
sImRldmljZV9wYXRjaGVkIjoieWVzIiwiZGV2aWNlX29zX3ZlcnNpb24iOiIxMy4yICgyMkQ0OSkiLCJ
kZXZpY2Vfb3NfdHlwZSI6Ik1hY09TIiwidXNlcl9pZCI6IlVJRDEyMz-
MiLCJ1c2VyX2xvY2F0aW9uIjoiRGFsbGFzLFRYIiwidXNlcl9pcF9hZGRyZXNzIjoiMS4yLjMuNCI-
sInVzZXJfYXV0aF9tZXRob2QiOiJYLjUwOSIsInRydXN0X3Njb3JlIjoiNyJ9.eSE9Onzyb3s_WGGi5k
E2eaH5h8KVCSlOcCoNeWMWwOE
```

Sharing Agent Data Fields Using JWT

A JSON Web Token (JWT), as defined by RFC 7519, is a compact way for two parties to exchange claims. JWTs are encoded as JSON objects that can carry the data fields needed to represent an agent. Additionally, you can also digitally sign and encrypt the JWT when sharing information about an agent to ensure high integrity and confidentiality.

Summary

This chapter introduced the concept of an agent, a new entity in a zero trust network against which authorization decisions are made. Adding this concept is critical to realizing the benefits of a zero trust network.

We explored what goes into creating an agent. Agents contain rapidly changing data and frequently have data that is unavailable or inconsistent. Accepting that reality is important for success when introducing the agent concept.

Agents are used purely for making authorization decisions. Authentication is a separate concern, and the current authentication status is reflected in the properties of an agent. Control plane systems use the agent to authorize requests. These systems are the primary enforcers of authorization in a zero trust network, but sometimes they must expose agent details to applications that are better positioned to implement fine-grained authorization decisions. We explored how to expose this data to applications while maintaining privacy.

While standard bodies such as NIST have recently developed guidance around zero trust, the administration side of it is still very new, and as a result, no proven standard for agents exists. Defining a standard would allow for better reuse and interoperability of zero trust systems, aiding the adoption of this technology. We discussed a possible approach for standardizing the definition of an agent.

The next chapter will focus on the systems that are responsible for authorizing all requests in a zero trust network.

CHAPTER 4

Making Authorization Decisions

Authorization is arguably the most important process occurring within a zero trust network, and as such, making an authorization decision should not be taken lightly. Every flow and/or request will ultimately require that a decision be made.

The databases and supporting systems we will discuss here are the key systems that come together to make and affect those decisions. Together, they are authoritative for access control and thus need to be rigorously isolated from each other. Careful distinction should be made between these responsibilities, particularly when deciding whether to collapse them into a single system, which should generally be avoided if possible.

Taking reality into account, this chapter will focus on the high-level architectural arrangement of the components required to make zero trust authorization decisions, as well as how they fit together and enforce said decisions.

Authorization Architecture

The zero trust authorization architecture comprises four main components, as shown in Figure 4-1:

- Enforcement
- Policy engine
- Trust engine
- Data stores

These four components are distinct in their responsibilities, and as a result, we treat them as separate systems. From a security standpoint, it is highly desirable that these components be isolated from each other. These systems represent the

practical crown jewels of the zero trust security model, so special care should be taken in their maintenance and security posture. It is critical from an implementation standpoint that isolation exists among these components so that a breach of one does not automatically lead to a breach of the entire system, both from a security and availability standpoint. This is typically handled by cloud-based systems, where SaaS-based services allow isolation based on various factors while services remain available under a single vendor's umbrella. Another common pattern is the use of microservices, in which various services are distributed across providers and are exposed via well-defined APIs. Because software systems are typically heavily distributed these days, planning for isolation should be prioritized early on.

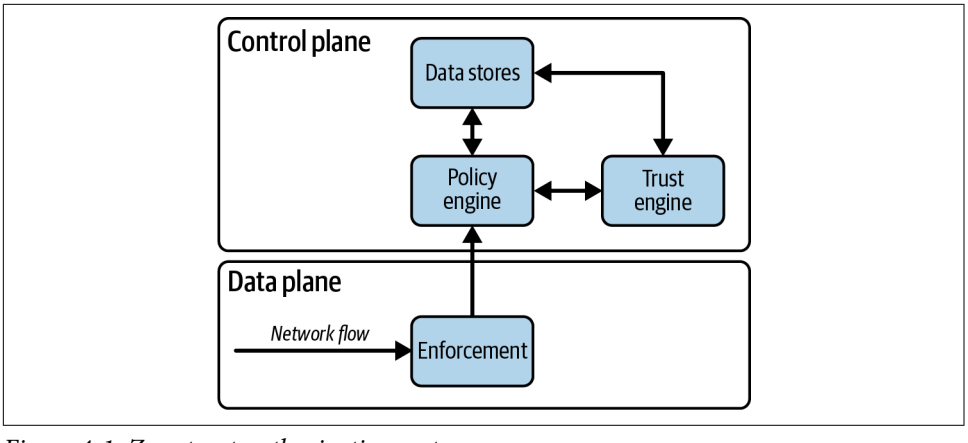

Figure 4-1. Zero trust authorization system

Enforcement is the component that actually affects the outcome of the authorization decision. It is typically manifested as a load balancer, proxy, or even a firewall. This component interacts with the policy engine, which is the piece that we use to make the actual decision. The enforcement component ensures that clients are authenticated, and passes the context of each flow/request to the policy engine. The policy engine compares the request and its context to policy, and informs the enforcer whether the request will be permitted or not. The enforcement components should exist in large numbers throughout the system and should be as close to the workload as possible.

A trust engine leverages multiple data sources in order to compute a risk score, similar to a credit score. This score can be used to protect against unknown unknowns, and helps keep policy strong and robust without complicating it with edge cases and signatures. It is used by the policy engine as an additional component by which authorization decisions can be made. Google's BeyondCorp is widely recognized as having pioneered this technology. The trust engine is leveraged by the policy engine for risk analysis purposes.

Finally, the various data stores represent the source of truth for the data being used to inform authorization. This data is used to paint a full contextual picture of a particular flow/request, using small authenticated bits of data as the primary lookup keys (i.e., a username or a device's serial number). These data stores, be they user data, device data, or otherwise, are heavily leveraged by both the policy engine and trust engine, and represent the backing against which all decisions are measured.

Enforcement

The enforcement component (depicted in Figure 4-2) is a natural place to start. It sits on the "front line" of the authorization flow and is responsible for carrying out decisions made by the rest of the authorization system.

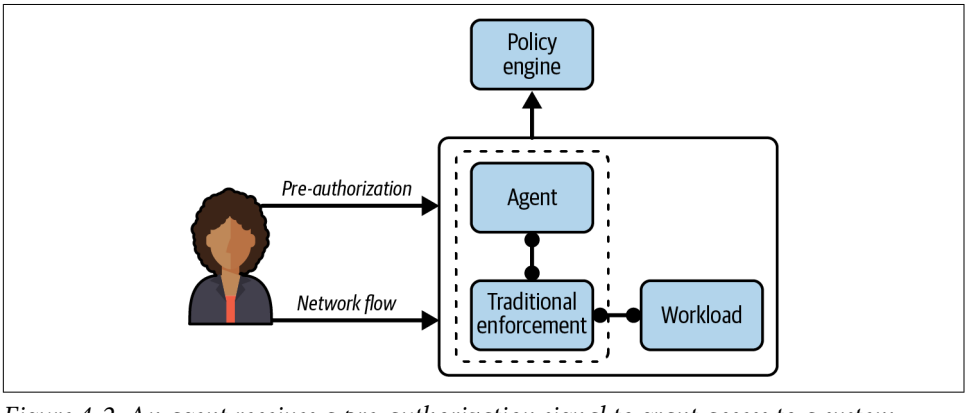

Figure 4-2. An agent receives a pre-authorization signal to grant access to a system using traditional enforcement mechanisms. These systems together form the enforcement component.

Enforcement can be broken down into two primary responsibilities. First, an interaction with the policy engine must occur. This is generally the authorization request itself (e.g., a load balancer has received a request and needs to know whether it is authorized or not). The second is the actual installation and ongoing enforcement of the decision. While these two responsibilities represent a single component in the zero trust authorization architecture, you can choose whether they are fulfilled together or separately.

The way you choose to handle this will likely depend on your use case. For instance, an identity-aware proxy can call the policy engine to actively authorize a request it has received, and in the same step use the response to either service or reject the request. This is an example of treating the concerns as unified. Alternatively, perhaps a pre-authorization daemon receives a request for access to a particular service, which then calls the policy engine for authorization. Upon successful authorization, the daemon can manipulate local firewall rules to allow the specific request. With

this approach, we rely on "standard" enforcement mechanisms that are informed/programmed by the zero trust control plane. It should be noted, however, that this approach requires a client-side hook in order to notify the control plane of the authorization request. This may or may not be acceptable, depending on the level of control you need over your devices and applications.

Placement of the enforcement component is very important. Since it represents our control point within the data plane, we must ensure that enforcement components are placed as close to the endpoints as possible. Otherwise, trust can pool "behind" the enforcement component, undermining zero trust security. Luckily, the enforcement component can be modeled as a client of sorts and applied liberally throughout the system. This is in contrast to the rest of the authorization components, which are modeled as services.

Policy Engine

The policy engine is the component that has the power to make a decision. It compares the request coming from the enforcement component against policy in order to determine whether the request is authorized or not. Once determined, the result is returned to the enforcement piece for actual realization.

The arrangement of the enforcement layer and policy engine allows for dynamic, point-in-time decisions to be made, allowing revocation to occur rapidly. As such, it is important that these components be considered separately and independently. That is not to say, however, that they cannot be co-located.

Depending on a number of factors, a policy engine may be found hosted side by side with the enforcement mechanism. An example of this might be a load balancer that authorizes requests through inter-process communication (IPC) instead of a remote call. The most attractive benefit of this architecture is the lower latency to authorize the request. A low-latency authorization system enables fine-grained and comprehensive authorization of network activity; for example, individual HTTP requests could be authorized instead of the session-level authorization that commonly is deployed. It should be noted that it is best to maintain process-level isolation between the policy engine and enforcement layer. The enforcement layer, being in the user's data path, is more exposed; therefore, integrating the policy engine in the same process could expose it to unwanted risk. Deploying the policy engine as its own process goes a long way to ensure that bugs in the enforcement layer don't result in a policy engine compromise.

Policy Storage

The rules referenced by the policy engine need to be stored. These policy rules are ultimately loaded into the policy engine, but it is strongly recommended that the rules are captured outside of the policy engine itself. Storing the policy rules in a version control system is ideal and provides several benefits:

- Changes to policy can be tracked over time.
- The rationale for changing policy is tracked in the version control system.
- The expected current policy state can be validated against the actual enforcement mechanisms.

Many of these benefits have historically been implemented using rigorous change management procedures. In that system, changes to the system's configuration are requested and approved before ultimately being applied. The resulting change management log can be used to determine why the system is in the current state.

Moving policy definitions into version control is the logical conclusion of change management procedures when the system can be configured programmatically. Instead of relying on human system administrators to load desired policy into the system, we can instead capture the policy as data that a program can read and apply. In many ways, loading policy is then similar to deployable software. As a result, system administrators can use standard software development procedures (namely code review and promotion pipelines) to manage the changes in policy.

What Makes Good Policy?

Policy in a zero trust network is in some ways similar to traditional network security, and in other ways substantially different.

Zero Trust Policy Is Still Not Standardized

The reality today is that zero trust policy is still not standardized in the same way as a network-oriented policy. As a result, defining the standard policy language used in a zero trust network is a great opportunity.

Let's look at what's similar first. Good policy in a zero trust network is fine-grained. The level of granularity will vary based on the maturity of the network, but the desired goal is policy that is scoped to the individual resource being secured. This is not very different from a traditional network security model that aims to segment the network to decrease attack surface area.

The zero trust model starts to diverge from traditional network security in the control mechanisms that are used to define policy. Instead of defining policy in terms of network implementation details (such as IP addresses and ranges), policy is best defined in terms of logical components in the network. These components will generally consist of:

- Network services
- Device endpoint classes
- User roles

Scale in the Age of the Cloud: Pets Versus Cattle

As more businesses migrate to the cloud as part of what is commonly referred to as the "digital transformation," they are reducing their reliance on on-premise datacenters. The cloud brings with it a level of scale that businesses have never experienced. By utilizing lightweight containers and server-less technologies, applications can scale in seconds to support very high throughput and then scale back, removing containers or server-less instances when they are no longer required; thus, the term "cattle" is commonly used to refer to them. In contrast, in an on-premises datacenter, physical servers are named and maintained for years by the businesses, so they are referred to as "pets." In the context of zero trust, policies must be dynamic enough to work consistently as scaling occurs, and this applies to both scaling up and scaling down.

Defining policy from logical components that exist in the network allows the policy engine to calculate the enforcement decisions based on its knowledge of the current state of the network. To put this in concrete terms, a web service running on one server today might be on a different server tomorrow, or might even move between servers automatically as directed by a workload scheduler. The policy that we define needs to be divorced from these implementation details to adapt to this reality. An example of this style of policy from the Kubernetes project is shown in Figure 4-3.

```
metadata:
 name: test-network-policy
 namespace: default
spec:
 podSelector:
  matchLabels:
    role: db
 ingress:
  - from:
      - namespaceSelector:
         matchLabels:
          project: myproject
      - podSelector:
         matchLabels:
          role: frontend
```

Figure 4-3. A snippet from a Kubernetes network policy. These policies use workload labels, computing the underlying IP-based enforcement rules when and where necessary.

Although there is no single method or standard for defining policies, they are typically configured in JSON or YAML format and are easy to understand semantically. Consider Google's custom access-level cloud policy, which defines conditions for known devices, such as corporate-owned and admin-approved devices running a known operating system:

```
{
   "name": "example_custom_level",
   "title": "Example custom level",
   "description": "An example custom access level.",
   "custom":  {
     "expr": {
       "expression": "device.is_corp_owned == true || (device.os_type !=
OsType.OS_UNSPECIFIED && device.is_admin_approved_device == true)",
       "title": "Check for known devices",
       "description": "Permits requests from corp-owned devices and admin-
approved devices with a known OS."
     }
   }
 }
```

Policy in a zero trust network also leans on trust scores to anticipate unknown attack vectors. By defining policy with a trust score component, administrators are able to mitigate risk that otherwise can't be captured with a specific policy. Therefore, most policies should include a trust score component. Check out the following example of a conditional access policy in Microsoft's Azure cloud that requires any user with a risk score of "medium" or "high" to perform mandatory multifactor authentication when signing in to an HR application. The trust score is covered in detail later in this chapter:

```
{
   "displayName": "Require MFA For High/Medium Sign-in Risk",
   "state": "enabled",
   "conditions": {
       "signInRiskLevels": ["high", "medium"],
       "clientAppTypes": [
           "all"
       ],
        "users": {
           "includeUsers": ["*"]
       }
   },
   "grantControls": {
       "operator": "OR",
       "builtInControls": [
           "mfa"
       ]
   }
}
```

Lack of Policy Standards

At the time of writing, there is no industry-wide standard for defining policies; however, efforts are being made toward this by organizations such as the National Cybersecurity Center of Excellence (NCCoE). It has created a publicly available description of the practical steps required to implement the cybersecurity reference designs for zero trust, as well as various components of zero trust, including policies. You can read more about this by visiting the NCCoE website.

Policy should not rely on trust scores alone. Specific characteristics of the request being authorized can also be part of the policy definition. An example of this might be that certain user roles should only have access to a particular service.

Who Defines Policy?

Zero trust network policy should be fine-grained, which can place an extraordinary burden on system administrators to keep the policy up to date. To help spread the load of this configuration burden, most organizations decide to distribute policy definition across teams so they can help maintain policy for the services they own. Opening up policy definition to an entire organization can present certain risks, like well-meaning users who create overly broad policies, thereby increasing the attack surface area of the system they intended to constrain. Zero trust systems lean on two organizational workflows to counteract this exposure.

Policy Reviews

First, since policy is typically stored under version control, having another person review changes to the policy helps ensure that changes are well considered. Security teams can additionally review the changes and ask probing questions to ensure that the policy being defined is as tightly scoped as possible. Since the policy is defined using logical intent instead of physical components, the policy will change less rapidly than if it was defined in physical terms.

The second organizational measure used is to layer broad infrastructure policy on top of fine-grained policy. For example, an infrastructure group might rightly require that only a certain set of roles be allowed to accept traffic from the internet. The infrastructure team will therefore define policy that enforces that restriction, and no user-defined policy will be allowed to circumvent it. Enforcing this constraint could take several forms: an automated test of proposed policy, or perhaps a policy engine that will simply refuse overly broad policy assertions from untrusted sources. Such enforcement can also be useful for compliance and regulatory requirements.

While there is no standard zero trust process to define the policy, the Kipling Method provides a good guideline for defining zero trust policies. This method helps explain the Who, What, When, Where, Why, and How of resource access policy succinctly:

- *Who* should be allowed to access a resource? This is essentially the identity (which can be human or machine) that is allowed to initiate the flow.
- *What* application/API/service is allowed to access the resource?
- *When* is the identity permitted to access the resource? This is primarily concerned with time frames such as office hours, etc.
- *Where* is the resource located? This can be anywhere, including the cloud, on-premises datacenters, etc.

- *Why* is the identity's access to the resource permitted? This is the primary justification or rationale for the access and is crucial for compliance and regulatory purposes.
- *How* should traffic be processed as it accesses a resource?

Trust Engine

The trust engine is the system in a zero trust network that performs risk analysis against a particular request or action. This system's responsibility is to produce a numeric assessment of the riskiness of allowing a particular request/action, which the policy engine uses to make an ultimate authorization decision.

The trust engine will frequently pull from data contained in authoritative inventory systems to check the attributes of an entity when computing its score. A device inventory, for example, could provide the trust engine with information like the last time a device was audited or scanned for compliance, or whether it has a particular hardware security feature.

Creating a numeric assessment of risk is a difficult task. A simple approach would be to define a set of ad hoc rules that score an entity's riskiness. For example, a device that is missing the latest software patches could have its score reduced. Similarly, a user who is continually failing to authenticate could have their trust score reduced. While ad hoc trust scoring might be simple to get started with, a set of statically defined rules will be insufficient to meet the desired goal of defending against unexpected attacks. As a result, in addition to using static rules, mature trust engines use machine learning techniques to derive a scoring function.

Machine learning derives a scoring function by calculating observable facts from a subset of activity data known as training data. The training data is raw observations that have been associated with trusted or untrusted entities. From this data, features are extracted and used to derive a computer-generated scoring function. This scoring function, or model in machine learning terms, is then run against a set of data that is in the same format as the training data. The resulting scores are compared against human-defined risk assessments, and the model's quality can then be refined based on its ability to correctly predict the risk associated with the data being analyzed. A model that has sufficient accuracy can then be said to be predictive of the riskiness of yet unseen requests in the network.

Machine learning models can learn from a variety of attributes, like the user's IP address, geo-location, device, and so on, to evaluate whether a current user request is anomalous or typical in the current context. Keep in mind that "false positives" can occur anytime. This is because there are legitimate situations where the user activity in question is normal, but the prediction tends to be anomalous. In real life, an example of a false positive can be seen when a user travels to a new location, perhaps

for a vacation, and makes an access request. In this case, the machine learning model has not yet been trained against this new user's location, so it will most likely identify this as an anomalous pattern. Dealing with false positives is a hot topic in machine learning, and it's usually improved by adjusting factors such as learning period, precision, and recall, among others.

What Factors Should You Consider for Machine Learning?

Although it is impossible to compile an exhaustive list, consider the following factors as a starting point while working toward the machine learning model and training set:

IP address, Autonomous System Number (ASN), and geo-location
: These are important attributes that can assist in determining anomalous patterns by a user/device over time.

User activity
: This includes regular user requests that fall under day-to-day productivity, like access to different applications/API endpoints, and so on.

Privileged activity
: This includes activity that typically falls under the administrator role, but also includes practically any activity that is categorized as privileged, such as deleting user accounts, etc.

Dormant accounts
: Accounts that have been inactive for an extended period must be labeled as such. This aids in the detection of unusual activity. Fraudulent account access can be detected by identifying dormant accounts that suddenly become active.

While machine learning is increasingly used to solve hard computational problems, it does not obviate the need for more explicit rules in the trust engine. Whether due to a limitation of the derived scoring models or a desire for customization of the scoring function, trust engines will typically use a mixture of ad hoc and machine learning scoring methods.

What Entities Are Scored?

Deciding which components of a zero trust network should be scored is an interesting consideration. Should scores be calculated for each individual entity (user, device, and application), for the network agent as a whole, or for both? Let's look at some scenarios.

Using network agents for scoring

Imagine a user's credentials are being brute-forced by a malicious third party. Some systems will mitigate this threat by locking the user's account, which can present a denial-of-service attack against that particular user. If we were to score a user negatively based on that activity, a zero trust network would suffer the same problem. A better approach is to realize that we're authenticating the network agent, and so the attacker's network agent is counteracted, leaving the legitimate user's network agent unharmed. This scenario makes a case that the network agent is the entity that should be scored.

Using devices for scoring

But just scoring the network agent can be insufficient against other attack vectors. Consider a device that has been associated with malicious activity. A user's network agent on that device may be showing no signs of malicious behavior, but the fact that the agent is being formed with a suspected device should clearly have an impact on the trust score for all requests originating from that device. This scenario strongly suggests that the device should be scored.

Finally, consider a malicious human user (the infamous internal threat) who is using multiple kiosk devices to exfiltrate trade secrets. We'd like the trust engine to recognize this behavior as the user hops across devices and to reflect the decreasing level of trust in their trust score for all future authorization decisions. Here again, we see that scoring the network agent alone is insufficient for mitigating common threats. Taken as a whole, it seems like the right solution is to score both the network agent itself and the underlying entities that make up the agent. These scores can be exposed to the policy engine, which can choose the correct component(s) to authorize based on the policy being written.

Presenting so many scores for consideration when writing policy, however, can make the task of crafting policy more difficult and error prone. In an ideal world, a single score would be sufficient, but that approach presents extra availability requirements to the trust engine. A system that tries to create a single score would likely need to move to an online model, where the trust engine is interactively queried during the policy decision making. The engine would be given some context about the request being authorized so it could choose the best scoring function for that particular request. This design is clearly more complex to build and operate. Additionally, for policy where a system administrator specifically wishes to target a particular component (say, only allow deployments from devices with a score above X), it seems rather roundabout.

Exposing Scores Considered Risky

While the scores assigned to entities in a zero trust network are not considered confidential, exposing the scores to end users of the system should be avoided. Seeing one's score could be a signal to would-be attackers that they are increasing or decreasing their trustworthiness in the system. This desire to withhold information should be balanced against the frustration provoked by end users' inability to understand how their actions are affecting their own trustworthiness in the system. A good compromise from the fraud industry is to show users their scores infrequently, and to highlight contributing factors to their score determination.

Data Stores

The data stores used to make authorization decisions are, very simply, the sources of truth for the current and past states of the system. Information from these data stores flows through the control plane systems, providing a large portion of the basis on which authorization decisions are made, as demonstrated in Figure 4-4.

We previously spoke about the trust engine leveraging these data stores in order to produce a trust score, which in turn is considered by the policy engine. In this way, information from control plane data stores has flowed through the authorization system, finally reaching the policy engine where the decision was made. These data stores are used by the policy engine, both directly and indirectly, but they can be useful to other systems that need authoritative data about the state of the network.

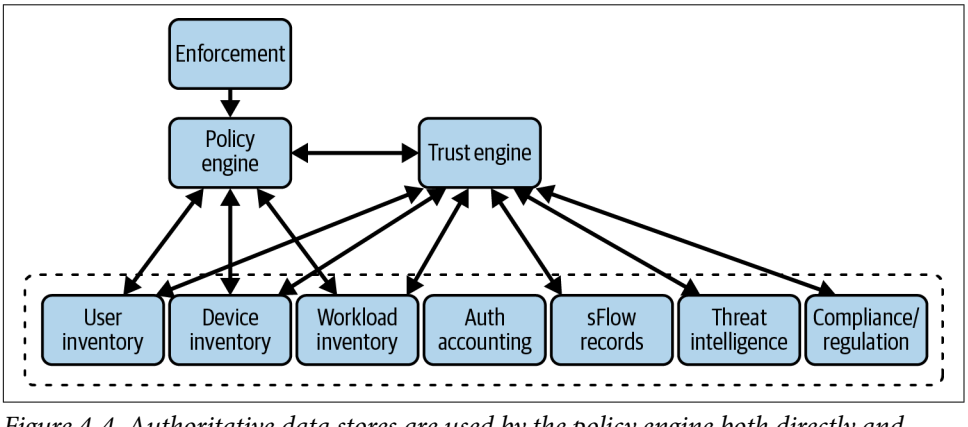

Figure 4-4. Authoritative data stores are used by the policy engine both directly and indirectly through the trust engine

Zero trust networks tend to have many data stores, organized by function. There are two primary types: inventory and historical. An inventory is a single consistent source of truth, recording the current state of the resource(s) it represents. An example is a user inventory that stores all user information, or a device inventory that records information about devices known to the company.

In an inventory, a primary key exists that uniquely represents the tracked entity. In the case of a user, the likely choice is the username; for a device, perhaps it's a serial number. When a zero trust agent undergoes authentication, it is authenticating its identity against this primary key in the inventory. Think about it like this: a user authenticates against a given username. The policy engine gets to know the username, and that the user was successfully authenticated. The username is then used as the primary key for lookup against the user inventory. Keeping this flow and purpose in mind will help you choose the right primary keys, depending on your particular implementation and authentication choices.

A historical data store is a little bit different. Historical data stores are kept primarily for risk analysis purposes. They are useful for examining recent/past behavior and patterns in order to assess risk as it relates to a particular request or action. Trust engine components are most likely to be consuming this data, as trust/risk determinations are the engine's primary responsibility.

One can imagine many types of historical data stores, and when it comes to risk analysis, the sky's the limit. Some common examples include user accounting records and sFlow[1] data. Regardless of the data being stored, it must be queryable using the primary key from one of the inventory systems. We will talk about various inventory and historical data stores as we introduce related concepts throughout this book.

Threat intelligence gathered from both internal and external third-party sources, such as Open Source Intelligence (OSINT), provides valuable insights that trust engines can use to determine a trust score. Consider a scenario in which a user's credentials were leaked on the dark web as a result of a recent data breach. In this case, the trust engine can use threat intelligence to calculate the trust score against the user, which may lead to the policy engine denying the request or granting it limited access.

Compliance and regulatory standards like the General Data Protection Regulation (GDPR), Federal Risk and Authorization Management Program (FedRAMP), etc., have an impact on the policy engine's decision-making process when analyzing a request. Organizations typically maintain a versioned system for maintaining compliance and regulatory requirements that can be used to create policies, ideally entirely automated, but most likely requiring final human review before release. The end

1 sFlow, short for "sampled flow," is an industry standard for packet export at Layer 2 of the OSI model.

result is a robust system in which the policy engine can query the compliance and regulatory store to determine if a request should be granted or rejected.

Scenario Walkthrough

Before we wrap up this chapter, let's consider a simple but real-world scenario that will help you understand the various components discussed in this and earlier chapters, plus how they interact with one another. Later chapters will expand on the scenario as we delve deeper into various aspects of zero trust, such as users, devices, applications, and traffic.

Let's look at a typical workflow for a user named Bob, who works as a business manager for Wayne Corporation and is attempting to access a resource, such as a printer. Figure 4-5 depicts a high-level breakdown of the zero trust components in this scenario.

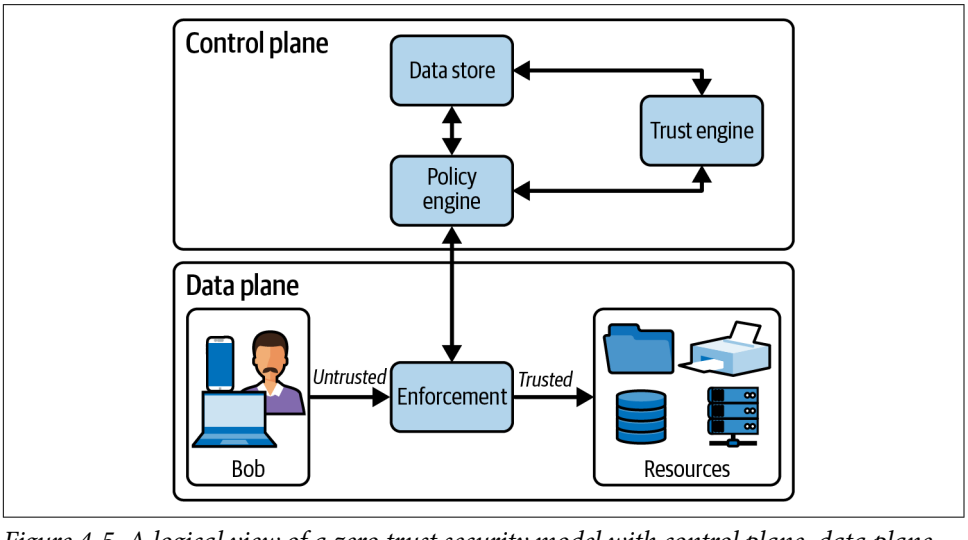

Figure 4-5. A logical view of a zero trust security model with control plane, data plane, user, and resources

First, examine the components in the control plane, as shown in Figure 4-6. Bob's personal information, such as his name, IP address, and location, is stored in the user store. The device data includes details such as the operating system and whether or not Bob's devices have received the most recent security patch. Finally, activity logs record every interaction he has, including the timestamp (in Unix format), IP address, and location.

The trust engine employs a machine learning model to dynamically calculate the trust score by looking for anomalous behavior in Bob's activity logs. Its primary responsibility is to calculate and communicate the trust score to the policy engine.

The policy engine, which is at the heart of the control plane, uses trust score and compliance policies to determine whether Bob's request should be granted or denied.

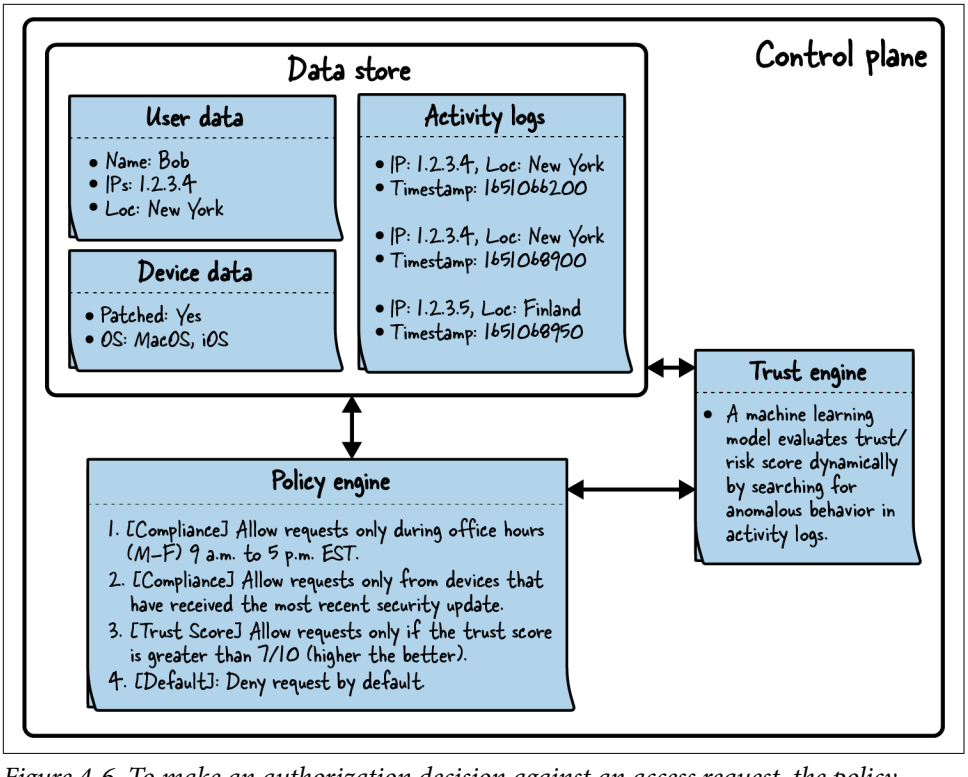

Figure 4-6. To make an authorization decision against an access request, the policy engine utilizes a trust score as well as compliance rules

We'll now take a closer look at the policy rules that govern the policy engine's behavior. The first two are compliance related, ensuring that the system always adheres to regulatory and operational business requirements. The third adds a trust score as a dynamic input to the policy, ensuring that requests are only granted if the score exceeds a certain threshold. Finally, if no other policy rule applies, the default behavior is set to deny the authorization request, ensuring that access must be denied unless a policy rule explicitly grants it:

1. *Compliance*
   Allow requests only during office hours, Monday through Friday between 9 a.m. and 5 p.m. Eastern Time Zone (EST).

2. *Compliance*
   Allow requests only from devices that have received the most recent security update. The goal is to ensure that devices are patched and less vulnerable to exploits.

3. *Trust Score*
   Allow requests only if the trust score is greater than 7/10. A higher trust score inspires more confidence in this case, so a value of 7 is used. Typically, trust score values in policies are configurable and adjusted over time to ensure a balance; a low score threshold allows malicious requests to slip through the cracks, while a high score may negatively impact genuine access requests.

4. *Default*
   If no other policy rule is applied, this is the catch-all (thus default) rule that takes effect. This rule is important because it is recommended to deny by default rather than allow by default. This is useful because there is no inherent trust in a zero trust system, so each request is evaluated on its own merits and is treated as equally malicious.

Next, consider the data plane, which includes enforcement, resources (printers, file shares, and so on), and the user Bob, who requests access to a resource (file share in this case). Figure 4-7 depicts the control plane as well as the data plane.

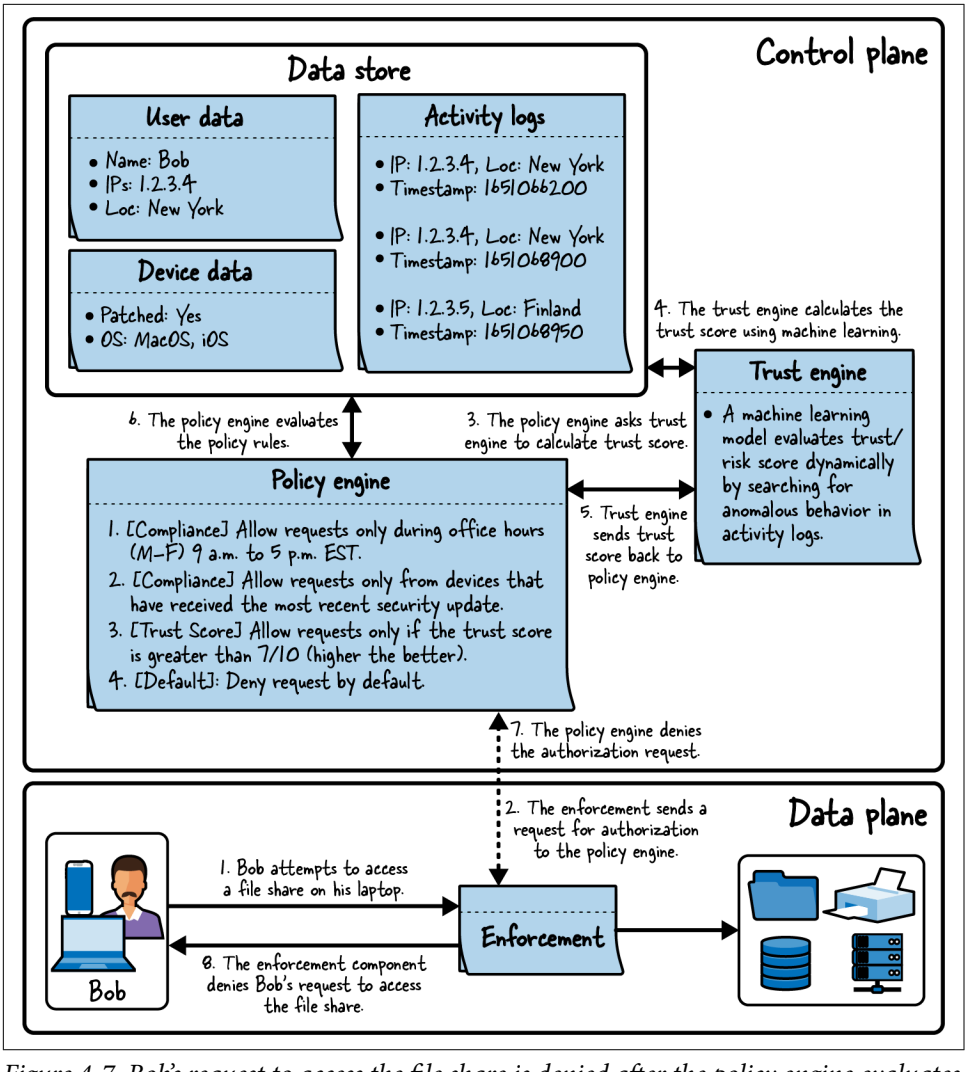

Figure 4-7. Bob's request to access the file share is denied after the policy engine evaluates the request using the trust score and other policy rules

Here's a step-by-step analysis of Bob's request:

1. On Monday, at 9.30 a.m. Eastern Time Zone (EST), Bob requests access to the file share from his laptop. The laptop is fully patched and runs MacOS.
2. The enforcement component intercepts the request and sends it to the policy engine for authorization.

3. The policy engine receives the request and consults with the trust engine to determine the request's trust score.
4. The trust engine uses a machine learning model to calculate the trust score based on the activity logs. Anomalies are detected because Bob's IP address of 1.2.3.5 and location in Finland are unusual. Moreover, given that the requests were made from New York and Finland and are only a few seconds apart, the timestamps between the last two activities appear impossible for a human to match. The machine learning model decides that the request should be assigned a trust score of 3, indicating a low level of trust, and sends the score to the policy engine.
5. The policy engine receives the trust score of 3 from the trust engine.
6. For authorization, the policy engine compares the request to all policy rules:

   This first rule results in a grant (or allow) action because the request is made during the permissible hours on Monday.

   The second rule results in grant (or allow) action because the request is made using a device that has been fully patched with the most recent security update.

   The third rule results in a deny action because the request received a trust score of 3, whereas the policy requires a trust score of 7 or higher to grant access. Because deny action is a final action, the policy engine does not process any additional rules.
7. The policy engine sends a deny action to the enforcement component. It also sends additional information about the result, which can aid in understanding the reason for the denial of the requested action.
8. The enforcement component receives the policy engine's result and denies Bob's request, preventing him from accessing the file share. It also sends Bob a helpful message about how to improve his chances of gaining access to the resource if he decides to do so in a future request.

While basic in nature, the scenario walkthrough in this section provides a functional understanding of various components in the control plane and data plane working together to deny Bob's request to access the file share. The key takeaway is that the system in place does not make authorization decisions based on ad hoc basics, but rather takes the overall context of the access request into account when making decisions.

Summary

This chapter focused on the systems that are responsible for making the ultimate decision of whether a particular request should be authorized in a zero trust network. This decision is a critical component of such a network, and therefore should be carefully designed and isolated to ensure it is trustworthy.

We broke this responsibility down into four key systems: enforcement, policy engine, trust engine, and data stores. These components are logical areas of responsibility. While they could be collapsed into fewer physical systems, the authors prefer an isolated design.

The enforcement system is responsible for ensuring that the policy engine's authorization decision takes effect. This system, being in the data path of user traffic, is best implemented in a manner where the policy decision is referenced and then enforced. Depending on the architecture chosen, the policy engine might be notified before a request occurs, or during the processing of that same request.

The policy engine is the key system that computes the authorization decision based on data available to it and the policy definitions that have been crafted by the system administrators. This system should be heavily isolated. The policy that is defined should ideally be stored separately from the engine and should use good software development practices to ensure that changes are understood, reviewed, and not lost as the policy moves from being proposed to being implemented. Furthermore, since zero trust networks expect to have much finer-grained policy, mature organizations choose to distribute the responsibility of defining that policy into the organization with security teams reviewing the proposed changes.

The trust engine is a new concept in security systems. This engine is responsible for calculating a trust score of components of the system using static and inferred algorithms derived from past behavior. The trust score is a numerical determination of the trustworthiness of a component and allows the policy writers to focus on the level of trust required to access some resource instead of the particular details of what actions might reduce that trust.

The final component of this part of the system is the authoritative data sources that capture current and historical data that can be used to make the authorization decision. These data stores should focus on being sources of truth. The policy engine, the trust engine, and perhaps third-party systems can leverage this data, so the collection of this data will have a decent return on the investment of capturing it.

The scenario walkthrough demonstrated how various control and data plane components interact to make the system work. In our scenario, the request from user Bob to access a file share was evaluated based on the overall context of the request, which included both a dynamic trust score calculation and various policies put in place by the business to make a final authorization decision. This scenario walkthrough will be expanded upon in later chapters.

The next chapter will dig into how devices gain and maintain trust.

CHAPTER 5
Trusting Devices

Trusting devices in a zero trust network is extremely critical; it's also an exceedingly difficult problem. Devices are the battlegrounds upon which security is won or lost. Most compromises involve a malicious actor gaining access to a trusted device; once that access is obtained, the device cannot be trusted to attest to its own security. This chapter will discuss the many systems and processes that need to be put in place to have sufficient trust in devices deployed in the network. We will focus on the role that each of these systems plays in the larger goal of truly trusting a device. Each technology is complicated in its own right. While we can't go into exhaustive detail on each protocol or system, we will endeavor to give enough details to help you understand the technology and avoid any potential pitfalls when using it.

We start with learning how devices gain trust in the first place.

Bootstrapping Trust

When a new device arrives, it is typically assigned an equal level of trust as that of the manufacturer and distributor. For most people, that is a fairly high level of trust (whether warranted or not). This inherited trust exists purely in meatspace, though, and it is necessary to "inject" this trust into the device itself.

There are a number of ways to inject (and keep) this trust in hardware. Of course, the device ecosystem is massive, and the exact approach will differ on a case-by-case basis, but there are some basic principles that apply across the board. These principles reduce most differences to implementation details.

The first of those principles has been known for a long time: golden images. No matter how you receive your devices, you should always load a known good image on them. Software can be hard to vet; rather than doing it many times hastily (or not at all), it makes good sense to do it once and certify an image for distribution.

Loading a "clean" image onto a device grants it a great deal of trust. You can be reasonably sure that the software running there is validated by you, and secure. For this reason, recording the last time a device was imaged is a great way to determine how much trust it gets on the network.

Secure Boot

There are, of course, ways to subvert devices so that they retain the implant across reimaging and other low-level operations, as the implants in these cases are usually themselves fairly low level.

A *secure boot* is one way to help fend against these kinds of attacks. It involves loading a public key into the device's firmware, which is used to validate driver and OS loader signatures to ensure that nothing has been slipped in between. While effective, support is limited to certain devices and operating systems. More on this later.

Being able to certify the software running on a device is only the first step. The device still needs to be able to identify itself to the resources that it is attempting to access. This is typically done by generating a unique device certificate that is signed by your private certificate authority. When communicating with network resources, the device presents its signed certificate. This certificate proves not only that it is a known device, but it also provides an identification method. Using details embedded in the certificate, the device can be matched with data from the device inventory, which can be used for further decision making.

Generating and Securing Identity

When providing a signed certificate by which a device may be identified, it is necessary to store the associated private key in a secure manner. This is not an easy task. Theft of the private key would enable an attacker to masquerade as a trusted device. This is the worst possible scenario for device authentication.

A simple yet insecure way to do this is to configure access rights to the key in such a way that only the most privileged user (root or administrator) can access it. This is the least desirable storage method, as an attacker who gains elevated access can exfiltrate the unprotected key.

Another way to do this is to encrypt the private key. This is better than relying on simple permissions, though it presents usability issues because a password (or other secret material) must be furnished in order to decrypt and use the key. This may not pose a problem for an end-user device, as the user can be prompted to enter the password, though this is usually not feasible for server deployments; human interaction is required for every software restart.

The best way by far to store device keys is through the use of secure cryptoprocessors. These devices, commonly referred to as hardware security modules (HSMs) or trusted platform modules (TPMs), provide a secure area in which cryptographic operations can be performed. They provide a limited API that can be used to generate asymmetric encryption keys, where the private key never leaves the security module. Since not even the operating system can directly access a private key stored by a security module, they are very difficult to steal.

Identity Security in Static and Dynamic Systems

In relatively static systems, it is common for an operator to be involved when new hosts are provisioned. This makes the injection story easy—the trusted human can directly cut the new keys on behalf of the hosts. Of course, as the infrastructure grows, this overhead will become problematic.

In automating the provisioning and signing process, there is an important decision to make: should a human be involved when signing new certificates? The answer to this largely depends on your sensitivity.

A signed device certificate carries quite a bit of power, and serves to identify anything with the private key as an authentic and trusted device. Just as we go through measures to protect their theft locally, we must also protect against their frivolous generation. If your installation is particularly sensitive, you might choose to involve a human every time a new certificate is signed.

Laws and Certificate Authorities

Modern browsers widely support certificates issued by well-known, trusted certificate authorities from many countries, including the United States and many others in the European Union, but this trust is vulnerable to geopolitical tensions. For example, during the Russo-Ukrainian war in 2022, Russia began offering its own trusted certificate authority to replace certificates that needed to be renewed by foreign countries. Without this action, Russian websites would have been unable to renew their certificates because sanctions prevent many countries' signing authorities from accepting payments from Russia. This serves as a stark reminder that, because issuing authorities are bound by the laws of the land, they can pose their own difficulties.

If provisioning is automated, but still human driven, it makes a lot of sense to allow the human driving that action to also authorize the associated signing request. Having a human involved every time is the best way to prevent unauthorized requests from being approved. Humans are not perfect, though. They are susceptible to fatigue and other shortcomings. For this reason, it is recommended that they be responsible for approving only requests that they themselves have initiated.

It is possible to accomplish provisioning and signature authorization in a single step through the use of a time-based one-time password (TOTP). The TOTP can be provided along with the provisioning request and passed through to the signing service for verification, as shown in Figure 5-1. This simple yet strong mechanism allows for human control over the signing of new certificates while imposing only minimal administrative overhead. Since a TOTP can only be used once, a TOTP verification failure is an important security event.

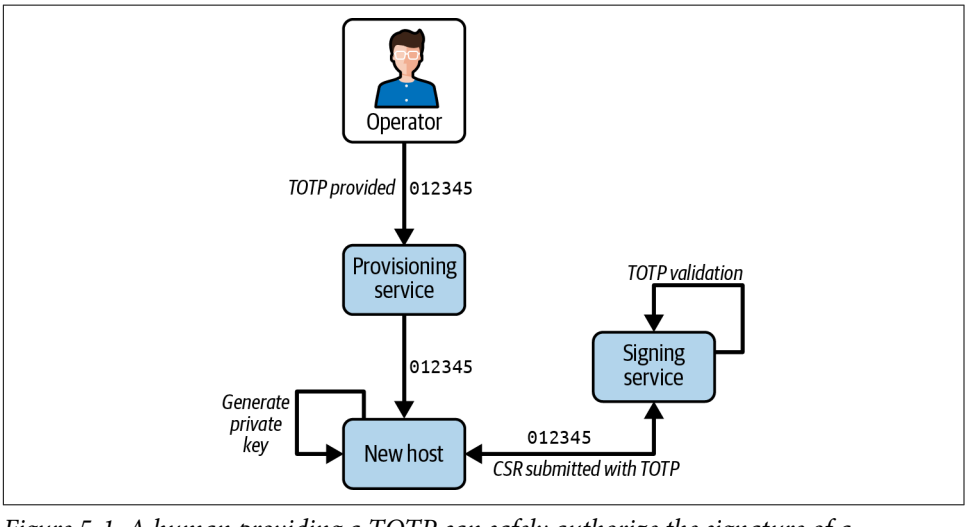

Figure 5-1. A human providing a TOTP can safely authorize the signature of a certificate

It goes without saying that none of this applies if you want to fully automate the provisioning of new hosts. Frequently referred to as "auto-scaling," systems that can grow and shrink themselves are commonly found in large, highly automated installations.

Allowing a system to scale itself decreases the amount of care and feeding required, significantly reducing administrative overhead and cost. Signing a certificate is an operation that requires a great deal of trust, and just as with other zero trust components, this trust must be sourced from somewhere. There are three common choices:

- A human
- The resource manager
- The image or device

The human is an easy and secure choice for relatively static infrastructure or end-user devices, but is an obvious nonstarter for automated infrastructure. In this case, you must choose the resource manager or the image...or both.

The resource manager is in a privileged position. It has the ability to both grow and shrink the infrastructure, and is likely able to influence its availability. It provides a good analog to a human in a more static system. It is in a position to assert, "Yes, I turned this new host on, and here is everything I know about it." It can use this position to either directly or indirectly authorize the signing of a new certificate.

To make the job of resource manager easier, many cloud vendors such as Microsoft, Google, and others provide built-in support for identities that do not require credentials and can be used to authenticate against specific services in a well-defined manner. Learn more about Google's service accounts and Microsoft's managed identities.

Depending on your needs, it might be desirable to not grant this ability wholly to the resource manager. In this case, credentials can be baked into an image. This is generally not advised as a primary mechanism, as it places too much responsibility on the image store; and protecting and rotating images can be fraught with peril. In a similar way, HSMs or TPMs can be leveraged to provide a device certificate that is tied to the hardware. This is better than baking material into the image, though requiring a TPM-backed device to sign a new certificate is still not ideal, especially when considering cloud-based deployments.

One good way to mitigate these concerns is to require both the resource manager and a trusted image/device. Generic authentication material baked into the image (or a registered TPM key) can be used to secure communication with the signing service and can serve as a component in a multifaceted authorization. The following are examples of components for authorization consideration:

- Registered TPM key or image key
- Correct IP address
- Valid TOTP (generated by resource manager)
- Expected certificate properties (i.e., expected common name)

By validating all of these points, the certificate signing service can be relatively certain that the request is legitimate. The resource manager alone cannot request a certificate, and since it does not have access to the hosts it provisions, the most an attacker could do is impact availability. Similarly, a stolen image alone cannot request a certificate,

as it requires the resource manager to validate that it has provisioned the host and expects the request.

By splitting these responsibilities and requiring multiple systems to assert validity, we can safely (well, as safely as is possible) remove humans from the loop.

Resource Managers and Containers

Sometimes it all comes down to terminology. In host-centric systems, resource managers create auto-scaling systems, making decisions about when and where capacity is needed. In containerized environments, the same decisions are made and executed by a resource scheduler. For the purposes of zero trust application, these components are practically identical, and the principles apply equally to host-centric and container-centric environments.

Authenticating Devices with the Control Plane

Now that we know how to store identity in a new device or host, we have to figure out how to validate that identity over the network. Luckily, there are a number of open standards and technologies available through which to accomplish this. Here, we'll discuss two of those technologies and why they are so important to device authentication: first we'll cover X.509 before moving on to look at TPMs.

These technologies enjoy widespread deployment and support, though this was not always the case. While we discuss real-world approaches to securing legacy devices in Chapter 8, we'll additionally explore here what the future might hold for zero trust support in legacy hardware.

X.509

X.509 is perhaps the most important standard we have when it comes to device identity and authentication. It defines the format for public key certificates, revocation lists, and methods through which to validate certification chains. The framework it puts forth aids in the formation of identity used for secure device authentication in nearly every protocol we'll discuss in this book.

One of the coolest things about X.509 is that the public/private key pairs it uses to prove identity can also be used to bootstrap encrypted communication. This is just one of many reasons that X.509 is so valuable for internet security.

Please refer to RFC 5280, RFC 4519, and the ITU-X.509 documentation for more information on the X.509 certificate format and supported attributes.

Certificate chains and certification authorities

For a certificate to mean anything, it has to be trusted. A certificate can be created by anyone, so just having one with the right name on it does not mean much. A trusted party must endorse the validity of the certificate by digitally signing it. A certificate without a "real" signature is known as a self-signed certificate and is typically only used for development/testing purposes.

It is the responsibility of the registration authority (a role commonly filled by the certificate authority) to ensure that the details of the certificate are accurate before allowing it to be signed. In signing the certificate, a verifiable link is created from the signed certificate to the parent. If the signed certificate has the right properties, it can sign further certificates, resulting in a chain. The certificate authority lies at the root of this chain.

By trusting a certificate authority (CA), you are trusting the validity of all the certificates signed by it. This is quite a convenience, because it allows us to distribute only a small number of public keys in advance—the CA public keys, namely. All certificates furnished from there on can be linked back to the known trusted CA, and therefore also be trusted. We spoke more about the CA concept and PKI in general in Chapter 2.

Device identity and X.509

The primary capability of an X.509 certificate is to prove identity. It leverages two keys instead of one: a public key and a private key. The public key is distributed, and the private key is held by the owner of the certificate. The owner can prove they are in the presence of the private key by encrypting a small piece of data, which can only be decrypted by the public key. This is known as public key cryptography, or asymmetric cryptography.

The X.509 certificate itself contains a wealth of configurable information. It has a set of standard fields, along with a relatively healthy ecosystem of extensions, which allow it to carry metadata that can be used for authorization purposes. Here is a small sample of typical information found within an X.509 certificate:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 16210155439472130208 (0xe0f60a7cb39a38a0)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer:C=US, ST=TX, L=Houston, O=Contoso Corp
        Validity
            Not Before: Aug 18 22:54:43 2022 GMT
            Not After : Aug 18 22:54:43 2025 GMT
        Subject: C=US, ST=TX, L=Dallas, O=Contoso Corp, CN=mgmt011134.con-
toso.corp
        Subject Public Key Info:
```

```
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:af:ff:04:2e:69:96:40:eb:62:20:a8:db:61:06:
                    ……
                    3f:bd:b1:49:50:26:07:ac:72:c7:9b:81:5d:54:19:
                    88:8b
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Key Usage:
                Digital Signature, Key Encipherment
            X509v3 Subject Alternative Name:
                DNS:mgmt011134.contoso.corp,
                DNS:10.69.54.169,
                DNS:fdc0:8a12:793b:adf7:8da6:12cd:e34c:daf6
            X509v3 Extended Key Usage:
                TLS Web Client Authentication,
                TLS Web Server Authentication
    Signature Algorithm: sha256WithRSAEncryption
         1e:e1:ed:8a:40:85:ac:fb:85:78:9c:88:ee:75:30:76:14:79:
         ……
         8d:9f:44:ea
```

The Issuer field specifies who issued the certificate, and the Subject field specifies for whom this certificate is intended. Both the Issuer and Subject fields contain information such as Country (C), State (S), Locality Name (L), Organization (O), and Common Name (CN), among other fields.

The sample certificate above could be issued to a device and includes the following information:

- Country (C): US. This is the device's assigned country.
- State (S): TX, short for Texas. This is the device's assigned state.
- Locality Name (L): Dallas. This is the device's assigned city.
- Organization (O): Contoso Corp. This is the device's assigned organization.
- Common Name (CN): This is the device name or unique identifier assigned to the device.

Additionally, the Subject Alternate Field contains additional information such as the device's IPv4 and IPv6 addresses.

Since the certificate is signed and trusted, we can use this information to make authorization decisions. Leveraging the X.509 fields in this way means that device access may be authorized without a call to an external service, so long as the server knows who/what it should be expecting.

Depreciation of the Organization Unit

You may have seen the use of the Organization Unit (OU) field in an X.509 certificate. However, in June 2021, the CA/Browser Forum passed ballot SC47 to deprecate the use of the Organization Unit (OU) field from all public trust TLS/SSL certificates. This change was made because the OU represents a much smaller unit within an Organization (O), making it difficult for CA to assert its identity consistently. The proposed change went into effect on September 1, 2022.

Public and private components

As mentioned earlier, X.509 deals with key pairs rather than a single key. While it is overwhelmingly common that these are RSA key pairs, they don't necessarily have to be. X.509 supports many types of key pairs, and we have recently begun to see the popularization of other key types (such as ECDSA).

Private key storage challenges

X.509 is incredibly useful for device authentication, but it doesn't solve all the problems. It still has a private key, and that private key must be protected. If the private key is compromised, the device's identity and privacy will be vulnerable as well. While other zero trust measures help guard against the damage this might cause (like user/application authentication or authorization risk analysis), this is considered a worst case scenario and should be avoided at all costs.

Private keys can be encrypted when they are stored, requiring a password to decrypt. This is a good practice because it would require more than just disk access to successfully steal, but is only practical for user-facing devices. In the datacenter, encrypting the private key doesn't solve the problem because you still have to store the password, or somehow transmit it to the server, at which point the password becomes just as cumbersome as the private key itself.

Hardware security modules (HSMs) go a good distance in attempting to protect the private key. They contain hardware that can generate a public/private key pair and store the private key in secure memory. It is not possible to read the private key from the HSM. It is only possible to ask the HSM to do an operation with it on your behalf. In this way, the private key cannot be stolen, as it is protected in hardware. We'll talk more about TPMs, a type of HSM, in the next section.

X.509 for device authentication

The application of X.509 to device authentication in a zero trust network is immense. It is a foundational cornerstone in proving device identity for just about every

protocol we have and is instrumental in enabling end-to-end encryption. Every single device in a zero trust network should have an X.509 certificate.

There is one important consideration to make, however. We are using X.509 to authenticate a device, yet the heart of the whole scheme—the private key—is decidedly software based. If the private key is stolen, the whole device authentication thing is a sham!

These certificates are often used as a proxy for true device authentication because the keys are so long and unwieldy that you would never write one down or memorize one. They are something that would be downloaded and installed, and because of that, they don't tend to follow users around—they more typically follow devices. While it might be determined that the risk associated with the private key problem is acceptable, it still stands as a serious issue, particularly for zero trust. Fortunately, we can see some paths forward, and by leveraging TPMs it is possible to inextricably marry a private key to its hardware.

TPMs

A trusted platform module (TPM) is a special chip that is embedded in a compute device called a cryptoprocessor. These chips are dedicated to performing cryptographic operations in a trusted and secure way. They include their own firmware and are often thought of as a computer on a chip.

This design enables a small and lean hardware API that is easily audited and analyzed for vulnerability. By providing facilities for cryptographic operations, and excluding interfaces for retrieving private keys, we get the security we need without ever exposing secret keys to the operating system. Instead, they are bound to the hardware.

This is a very important property and the reason that TPMs are so important for device authentication in zero trust networks. Great software frameworks for identity and authentication (like X.509) do a lot for device authentication. But without a way to bind the software key to the hardware device it is attempting to identify, we cannot really call it device identity. TPMs solve this problem, providing the necessary binding.

Encrypting data using a TPM

TPMs generate and store what is known as a storage root key, or an SRK. This key pair represents the trust root for the TPM device. Data encrypted using its public key can be decrypted by the originating TPM only.

The astute reader might question the usefulness of this function in the application of bulk data encryption. We know asymmetric cryptographic operations to be very expensive, and thus not suitable for the encryption of relatively large pieces of data.

Thus, in order to leverage the TPM for bulk data encryption, we must reduce the amount of data that the SRK is responsible for securing.

An easy way to do this is to generate a random encryption key, encrypt the bulk data using known performant symmetric encryption (i.e., AES, or Advanced Encryption Standard), and then use the SRK to encrypt the resulting AES key. This strategy, shown in Figure 5-2, ensures that the encryption key cannot be recovered, unless in the presence of the TPM that originally protected it.

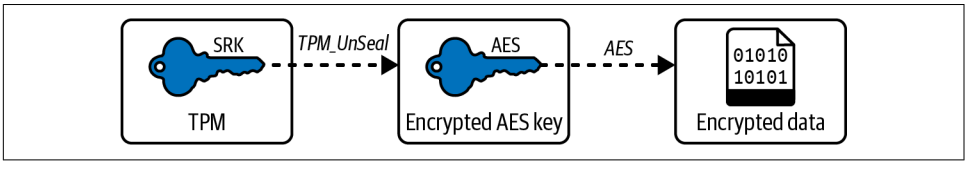

Figure 5-2. The data is encrypted with an AES key, which in turn is encrypted by the TPM

Most TPM libraries available for open consumption perform these steps for you, through the use of helper methods. It is recommended to inspect the internal operation of such methods before using them.

Intermediary keys and passphrases

Next, let's consider intermediary keys. Many TPM libraries (such as TrouSerS) create intermediary keys when encrypting data using the TPM. That is, they ask the TPM to create a new asymmetric key pair, use the public key to encrypt the AES key, and finally use the SRK to encrypt the private key. When decrypting the data, you must first decrypt the intermediate private key, use it to decrypt the AES key, then decrypt the original data.

This implementation seems strange, but there are some relatively sane reasons for it. One reason is that the additional level of indirection allows for more flexibility in the distribution of secured data. Both the SRK and intermediate keys support passphrases, so the use of an intermediary key enables the use of an additional, perhaps more widely known, passphrase.

This may or may not make sense for your particular deployment. For the purposes of "this key should only be decryptable on this device only," it is OK (and more performant) to bypass the use of an intermediary key, if desired.

The most important application of TPM-backed secure storage is in protecting the device's X.509 private key. This secret key serves to authoritatively prove device identity, and if stolen, so is the identity. Encrypting the private key using TPM means that while the key might still be taken from disk, it will not be recoverable without the original hardware.

Key Theft Is Still Possible

Encrypting the device's private key and wrapping the key with the SRK does not solve all of the theft vectors. It protects the key from being directly read from disk, though an attacker with elevated privileges might still be able to read it from memory or simply ask the TPM to perform the operation for them. The following "Platform configuration registers" and "Remote attestation" sections provide additional information on how to further validate hardware identity (beyond X.509 identity).

Platform configuration registers

Platform configuration registers (PCRs) are an important TPM feature. They provide storage slots into which hashes of running software are stored. It starts with the hash of the BIOS, then the boot record, its configuration, and so on. This sequence of hashes can then be used to attest that the system is in an approved configuration or state. Here is a truncated example of the first few registers stored in the TPM:

```
PCR-00: A8 5A 84 B7 38 FC ...        # BIOS
PCR-01: 11 40 C1 7D 0D 25 ...        # BIOS Configuration
PCR-02: A3 82 9A 64 61 85 ...        # Option ROM
PCR-03: B2 A8 3B 0E BF 2F ...        # Option ROM Configuration
PCR-04: 78 93 CF 58 0E E1 ...        # MBR
PCR-05: 72 A7 A9 6C 96 39 ...        # MBR Configuration
```

This is useful in a number of ways, including in ensuring that only authorized software configurations are allowed to decrypt data. This can be done by passing in a set of known good PCR values when using the TPM to encrypt some data. This is known as "sealing" the data. Sealed data can only be decrypted by the TPM that sealed it, and only while the PCR values match.

Since PCR values cannot be modified or rolled back, we can use TPM sealing to ensure that our secret data is not only locked to the device, but also locked to a specific software configuration and version. This helps to prevent attackers from using device access to obtain the private key, since only the unmodified and approved software can unlock it.

Remote attestation

We have learned many ways we can use embedded device security to protect private keys and other sensitive device-related data. The unfortunate truth is that so long as a private key is stored outside of a physical TPM, it is still vulnerable to theft. This fact remains because all it takes to recover the private key is to convince the TPM to unlock it once. This action discloses the actual private key—something that is not possible when it is stored on the TPM.

Luckily, the TPM provides a way for us to uniquely identify it. It's another key pair called the endorsement key (EK), and each TPM has a unique one. The private component of an EK only ever exists on the TPM itself, and thus remains completely inaccessible to the operating system.

Remote attestation is a method by which the TPM generates something called a "quote," which is then securely transmitted to a remote party. The quote includes a list of current PCR values, signed using the EK. A remote party can use this to assert both host identity (since the EK is unique to the TPM) and software state/configuration (since PCRs cannot be modified). We'll talk more about how the quote can be transmitted in Chapter 8.

Why Not Just TPM?

You may find yourself wondering: why not use the TPM exclusively for device identity and authentication, and why include X.509 at all?

Currently, TPM access is cumbersome and non-performant. It can provide an X.509 certificate to confirm its identity, but it is limited in its interaction with the private key. For instance, the key used for attestation is only capable of signing data that originates in the TPM. For a protocol like TLS, this is a deal-breaker.

There have been some attempts to coerce the TPM attestation protocols into a more flexible form (like the IETF draft, which defines a TLS extension for device authentication via TPM), though none of them have gained widespread adoption at the time of this writing.

There are a few open source implementations of remote attestation, including one in the popular IKE daemon strongSwan. This opens the doors for leveraging TPM data to not only authenticate an IPsec (Internet Protocol Security) connection, but also authorize it by using PCR data to validate that the host is running authentic and unmodified software.

TPMs for Device Authentication

It is clear that TPMs present the best option for strong device authentication in mature zero trust networks. They provide the linchpin between software identity and physical hardware. There are, however, a couple of limitations.

Many datacenter workloads are heterogeneous and isolated, like virtual machines or containers, both of which need to resort to TPM virtualization to allow the isolated workload to accomplish similar goals. While there are implementations available (such as vTPM for Xen), trust must still be rooted in a hardware TPM, and designing a secure TPM-based system that is capable of live migration is challenging.

Additionally, TPM support is still sparse despite its many uses and strengths. While TPM use would be expected in the context of device authentication in mature zero trust networks, it should not be considered a requirement. Adopting TPM support is no small feat, and there are many lower-hanging fruits in terms of zero trust adoption and migration.

HSM and TPM Attack Vectors

HSM and TPM attack vectors have been in the news lately with the discovery of new attacks that can be used to bypass the security features of these devices. These attacks are based on the way HSMs and TPMs are typically implemented; they each use a shared secret key to encrypt and decrypt data. This shared secret key is known as the "endorsement key" (EK).

The EK is used to encrypt and decrypt a second key, known as the "storage root key" (SRK). The SRK is used to encrypt and decrypt the data that is stored on the HSM or TPM. The problem is that the EK is usually generated by the HSM or TPM manufacturer and is not kept secret, which means that if an attacker can obtain the EK, they can use it to decrypt the SRK and then access the data that is encrypted with the SRK.

Several attacks have been developed that can be used to obtain the EK from an HSM or TPM:

1. The first attack, known as the "ROCA" attack, was discovered in 2017. The ROCA attack is a mathematical attack that can be used to calculate the EK if the attacker has access to a small amount of data encrypted with the HSM or TPM.
2. The second attack, known as the "Side-Channel" attack, was discovered in 2018. The Side-Channel attack is a physical attack that can be used to obtain the EK by measuring the electrical characteristics of the HSM or TPM while encrypting or decrypting data.
3. The third attack, known as the "Fault Injection" attack, was discovered in 2019. The Fault Injection attack is a physical attack that can be used to introduce faults into the HSM or TPM while it is encrypting or decrypting data. These faults can then be used to obtain the EK.

These attacks have raised concerns about the security of HSMs and TPMs. In response to these concerns, several companies have been working on solutions to protect HSMs and TPMs from these attack vectors.

One solution that has been proposed is "confidential computing." Confidential computing is a security technique that can be used to protect data that is stored or processed on an HSM or TPM. This method uses encryption to protect the data while

it is being stored or processed, which means that even if an attacker obtains the EK, they will not be able to decrypt the data.

We have also noted another proposed solution in "Bootstrapping Trust" on page 75 known as "secure boot." Secure boot is a security technique that can be used to ensure that only trusted software can be run on an HSM or TPM. Secure boot uses cryptographic signatures to verify the software's identity on the HSM or TPM so that even if an attacker obtains the EK, they will not be able to run malicious software on the HSM or TPM.

As attacks against HSMs and TPMs become more sophisticated, it is important to ensure that your HSMs and TPMs are protected against these attack vectors. In addition to confidential computing and secure boot technologies, make sure you:

- Keep your HSMs and TPMs up to date with the latest security patches.
- Use HSMs and TPMs from reputable vendors.
- Use HSMs and TPMs that have been independently audited.
- Use physical and logical security measures to protect your HSMs and TPMs.
- Keep your confidential data offline.
- Destroy your confidential data when you no longer need it.

Hardware-Based Zero Trust Supplicant?

The most common approach for supporting legacy devices in a zero trust network is to use an authentication proxy. The authentication proxy terminates the zero trust relationship and forwards the connection to the legacy host.

While it is possible to enforce policy between the authentication proxy and the legacy backend, this mode of operation is less than ideal and shares a handful of attack vectors with traditional perimeter networks. When dealing with legacy devices, it is desirable to push the zero trust termination point as close to the device as possible.

When possible, it is preferable to use a dedicated hardware device rather than an application proxy. This device can act as a zero trust supplicant, carrying a TPM chip, and plug directly into a legacy device's Ethernet port. Pairing the two in your inventory management system can allow for seamless integration between legacy devices and a zero trust network.

There are many applications that would significantly benefit from such a device. SCADA (supervisory control and data acquisition) and HVAC (heating, ventilating, and air conditioning) systems, for instance, come to mind.

Inventory Management

Authenticating a device's identity and integrity goes a long way in providing strong zero trust security, but being able to identify a device as belonging to the organization is only part of the challenge. There are lots of other pieces of information we need in order to calculate policy and make enforcement decisions.

Inventory management involves the cataloging of devices and their properties. Maintaining these records is equally important for both servers and client devices. It is sometimes more helpful to think of these as network entities rather than physical devices. While they indeed are commonly physical devices, they might also be logical entities on the network.

For instance, it is conceivable that a virtual machine or a container could be considered a "device," depending on your needs. They have lots of the same descriptive properties that a real server might have, after all. Lumping all of the virtual machine traffic from a single host into one policy gets us right back to the perimeter model. Instead, the zero trust model advocates that the workloads be tracked in order to drive the network policies they require. This inventory (or workload) database in this case can be specialized in order to accommodate the high rates of change that virtualized/containerized environments experience. So, while the traditional inventory management system and the workload scheduler might be different systems, they can still work together; for the purposes of this book, the scheduler service may act as an inventory management system of sorts, as shown in Figure 5-3.

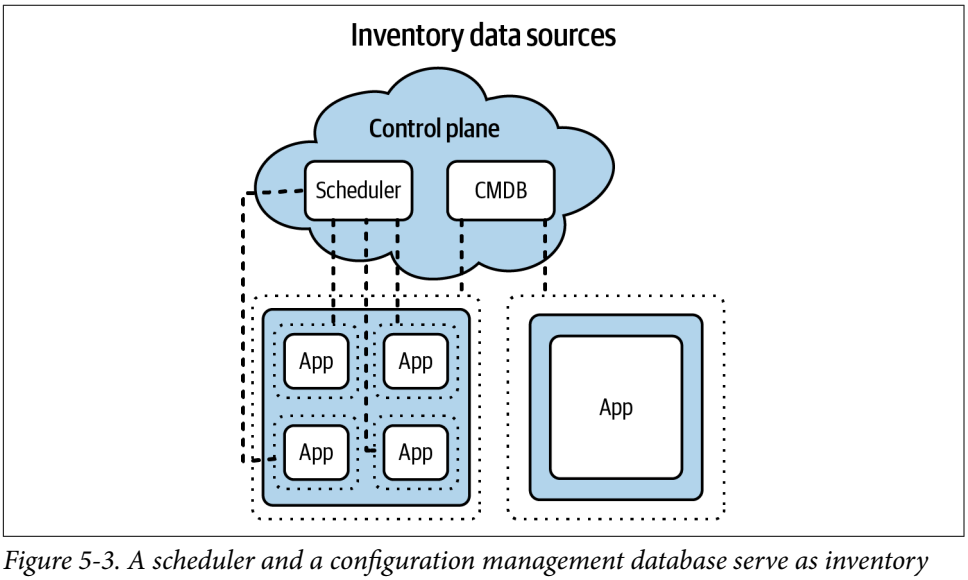

Figure 5-3. A scheduler and a configuration management database serve as inventory stores for the control plane

It is not uncommon to have more than one inventory management system. As an example, many companies have both asset management and configuration management software. Both of these store device metadata that is useful to us; they just store different sets, collected in different ways.

Configuration Management as an Inventory Database

Many configuration management (CM) systems, such as Chef or Puppet, offer modes in which data about the nodes they run on get persisted into a centralized database. Name, IP address, and the "kind" of server are examples of the type of information typically found in a CM-backed database. Using configuration management in this way is an easy first step toward developing an inventory database if you don't have one already.

Knowing What to Expect

One of the great powers of a zero trust network is that it knows what to expect. Trusted entities can push expectations into the system, allowing all levels of access to be denied by default—only expected actions/requests are permitted.

An inventory database is a major component in realizing this capability. A huge amount of information about what to expect can be generated from this data; things like which user or application should be running on it, what locations we might expect it to be in, or even the kind of operating system are all pieces of information that can be used to set expectations.

In the datacenter, these expectations can be very strong. For instance, when provisioning a new server, we often know what IP address it will be assigned and what purpose it will serve. We can use that information to drive network ACLs (access control lists) and/or host-based firewalls, poking holes for that specific IP address only where necessary. In this way, we can have all traffic denied, allowing only the very specific flows we are expecting. The more properties that can be expected, the better.

This is not such an easy prospect for client-facing systems, however. Clients operate in new and unexpected ways all the time, and knowing exactly what to expect from them and when is very difficult. Servers in the datacenter often have relatively static and long-lived connections to a well-defined set of hosts or services. By contrast, clients tend to make many short-lived connections to a variety of services—the timing, frequency, and patterns of which can vary organically.

To address the wild nature of client-facing systems, we need a slightly different approach. One way to do this is to simply allow global access to the service and protect it with mutually authenticated TLS, forcing the client to provide a device certificate before it can communicate with it. The device certificate can be used

to look the device up in the inventory database and determine whether or not to authorize it. The advantage is that lots of systems support mutually authenticated TLS already, and specialized client software is not strictly required. One can provide reasonably strong security without too badly hindering accessibility or usability.

A significant drawback to this approach, however, is that the service is globally reachable. Requiring client certificates is a great way to mitigate this danger. However, we have seen from vulnerabilities like Heartbleed that the attack surface of a TLS server is relatively large. Additionally, the existence of the resources can be discovered by simply scanning for them, since we get to speak TCP (Transmission Control Protocol) to the resource before we authenticate with it.

How can we ensure that we don't engage clients that are not trusted? There has to be some untrusted communication, after all. What comes before the authentication?

Secure Introduction

The very first connection from a new device is a precarious one. After all, these packets must be admitted somewhere, and if they are not strongly authenticated, then there is a risk. Therefore, the first system that a new device contacts needs a mechanism by which it can authenticate this initial contact.

This arrangement is commonly known as secure introduction. It is the process through which a new entity is introduced to an existing one in a way that trust is transferred to it. There are many ways in which this can be effected; the method through which an operator passes a TOTP code to a provisioner in order to authorize a certificate request is a form of secure introduction.

The best (and perhaps only) way to do a secure introduction is by setting an expectation. Secure introduction practically always involves a trusted third party. This is a system that is already introduced, and it holds the ability to introduce new systems. This trusted third party is the system that then coordinates/validates the specifics of the system to be introduced and sets the appropriate expectations.

Secure Introduction for Client Systems

Secure introduction of client-facing systems can be difficult due to the hard-to-predict nature of wild clients. When publicly exposing a client-facing endpoint is considered too risky, it is necessary to turn to more complicated schemes. The currently accepted approach is to use a form of signaling called pre-authentication, which announces a client's intentions just prior to taking action. We'll talk more about pre-authentication in Chapter 8.

What makes a good, secure introduction system? There are three main criteria to consider:

Single-use
: Credentials and privileges associated with the introduction should be single-use, preventing an attacker from compromising and reusing the key.

Short-lived
: Credentials and privileges associated with the introduction should be short-lived, preventing the accumulation of valid but unused keys.

Third party
: Leveraging a third party for introduction allows for separation of duty, prevents the introduction of poor security practices, and alleviates operational headaches.

While these requirements might at first seem rigorous, they can be met through fairly simple means. A great example can be found in the way Chef implements host introduction. Originally, there was a single secret (deemed the "validation certificate") which was qualified to admit any host that possessed it as a new node. Thus, the introduction would involve copying this secret to the target machine (or baking it into the image), using it to register the new node, then deleting it.

This approach is neither single-use nor short-lived. Should the secret be recovered, it could be used by a malicious actor to steer application traffic to attacker-controlled hosts, or even trigger a denial of service.

Chef takes a different approach in later versions. Instead of having a static validation certificate, the provisioning system (via the Chef client utility "knife") communicates with the Chef server and creates a new client and associated client certificate. It then creates the new host and passes in its client certificate. In this way, an expectation for the new client has been set. While these credentials are not short-lived, it remains a superior approach.

Renewing and Measuring Device Trust

It is important to accept the fact that no level of security is perfect—not even yours. Once this fact is acknowledged, we can begin to mitigate its consequences. The natural progression is that the longer a device is operating, the greater its chances of being compromised. This is why device age is a heavily weighted trust signal.

For this reason, rotation is very important. We earlier spoke at length about the importance of rotation, and devices are no different. Of course, this "rotation" is manifested in different ways, depending on your definition of "device." If your infrastructure is run in a cloud, perhaps a "device" is a host instance. In this case, rotation is easy: just tear down the instance and build a new one (you are using configuration management, right?). If you're running physical hardware, however, this prospect is a little more difficult.

Reimaging is a good way to logically rotate a device. It is a fairly low-level operation, and will succeed in removing the majority of persistent threats seen in the wild today. One can trust a freshly reimaged device more than one that has been running for a year. While reimaging does not address hardware attacks or other low-level attacks like those shown in Figure 5-4, it serves as a reasonable compromise in places where physical rotation is more difficult. Datacenter and supply chain security partially mitigate this concern.

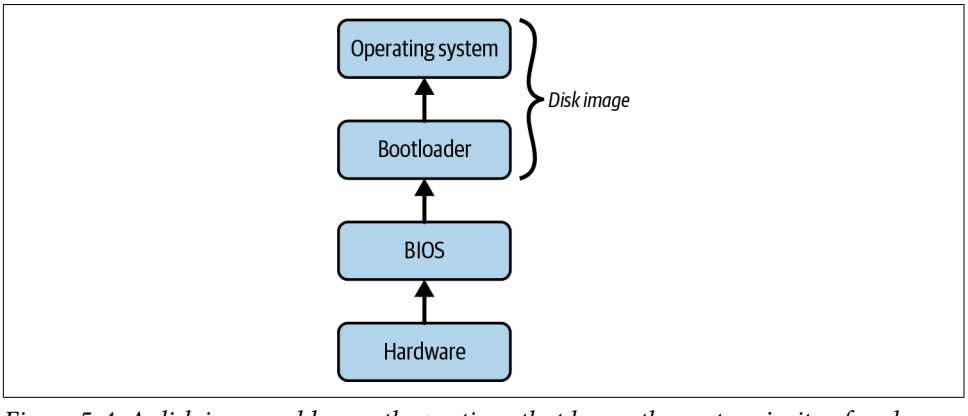

Figure 5-4. A disk image addresses the portions that house the vast majority of malware, but it's certainly not the whole picture

When it comes to managing client devices, the story changes quite a bit. Reimaging a client device is extraordinarily inconvenient for users. They customize the device (and its contents) over time in ways that are difficult to effectively or securely preserve. Oftentimes, when given a new device, they want to transfer the old image! This is not great news for people trying to secure client devices.

The solution largely depends on your use case. The trade-off between security and convenience will be very clear in this area. Everyone agrees that client devices should be rotated and/or reimaged every so often, but the frequency is up to you. There is one important relationship to keep in mind: the less often a device is rotated or reimaged, the more rigorous your endpoint security must be.

Without the relatively strong assurances of device security that we get with rotation, we must look for other methods to renew trust in a device that has been operating for a long time. There are two general methods through which this can be done: local measurement or remote measurement.

Local Measurement

Local measurement can be one of two types: hardware backed or software backed. Hardware-backed measurement is more secure and reliable, but limited in capability.

Software-backed measurement is much less secure and reliable, but practically unlimited in its measurement capabilities.

One good option for hardware-backed local measurement is leveraging the TPM for remote attestation. Remote attestation uses a hardware device to provide a signed response outlining the hashes of the software currently running on that machine. The response is highly reliable and very difficult to reproduce. However, it generally only gives a picture of the low-level software or specifically targeted software. If an attacker has managed to get an unauthorized process running in user space, the TPM will not be very useful in its detection; thus, it has limited capability. See "Remote attestation" on page 86 for more information.

Software-backed local measurement involves some sort of agent installed on the endpoint which is used to report health and state measurements. This could be anything from a managed antivirus client to policy enforcement agents. These agents go to great lengths in order to attest and prove the validity of the measurements they report, but even cursory thought quickly reaches the conclusion that these efforts are generally futile. Software-backed measurements lack the protection provided by hardware measurements, and an attacker with sufficient privilege can subvert systems like this.

Remote Measurement

Remote measurement is the best of the two options for one simple reason: it benefits from separation of duty. A compromised host can report whatever it wants to, possibly falsifying information in order to conceal the attacker. This is not possible with remote or passive measurement, since a completely different system is determining the health of the host in question.

Traditionally, remote measurement is performed as a simple vulnerability scan. The system in question will be periodically probed by a scanning device, which observes the response. The response gives some information away, like what operating system might be running on that device, what services might be active there, and maybe even what version of those services.

The scan results can be cross-referenced with known bad signatures, like malicious software or vulnerable versions of legitimate software. Detection of known bad signatures can then influence the trust of the device appropriately.

There are a number of open source and commercial options available in the vulnerability scanning arena, including OpenVAS, Nessus, and Metasploit. These projects are all fairly mature and relied on by many organizations.

Unfortunately, vulnerability scanning comes with the same fundamental problem as local measurement: it relies on interrogation of the endpoint. It's the difference between asking someone if they robbed a bank, and watching them rob a bank. Sure,

sometimes you can get the robber to admit that they did it, but a professional would never fall for that. Catching them in the act is much more effective. See "Network Communication Patterns" on page 102 for more about how to solve this dilemma.

Unified Endpoint Management (UEM)

Endpoint management is an example of software-based remote management. UEM systems allow an administrator to manage the security posture of all devices in an organization from a centralized console and play a critical role in achieving and maintaining device trust. Microsoft Intune, VMware AirWatch, MobileIron, ClearPass, and FreeIPA are all examples of endpoint management systems. While UEM systems were not designed with security as their primary focus, they have become an essential part of the security ecosystem because they are often used to enforce security policy on devices. For example, they can ensure that devices have a minimum level of security before they are allowed to connect. These systems can also monitor devices for compliance with security policy, push updates out centrally, and be configured to alert the security team if the system is no longer managing a device.

Continuous monitoring is a key part of trusting devices. Security teams must constantly be on the lookout for changes in device behavior that could indicate a compromise. For example, changes in network traffic patterns might suggest that a device has been infected with malware and is now communicating with a malicious server, while changes in file access patterns might indicate that an unauthorized user is trying to access sensitive data.

Security teams must have visibility into all device changes to properly assess the risk posed by those changes. However, it is not enough to monitor devices; security teams also need to be able to take action when a compromise is detected. UEM systems provide the ability to remotely lock or wipe a device if it is determined to be compromised, ensuring that its data will remain confidential even if a device is lost or stolen.

It is also essential to have a process in place for renewing trust in devices that have been compromised. All too often, organizations wipe away a device that has been compromised and start over with a new one. While this may be the safest option, it's usually not practical or cost-effective. Instead, it's much better to have a process for thoroughly cleaning and verifying a device before putting it back into production.

Device trust is a critical part of zero trust security. By understanding the various technologies and processes used to achieve device trust, you will be in a much better position to defend your organization against attacks.

Device Compliance Change Signals

The OpenID Foundation is currently working on a new standard called the Shared Signals and Events (SSE) Framework, which aims to standardize the exchange of status signals about changes in device, user, and machine identities, as well as application and session status, between cooperating parties. The OpenID Continuous Access Evaluation Profile, in particular, provides specific semantics to signal device compliance change in the form of JSON, as shown in this example:

```
        "iss": "https://idp.example.com/123456789/",
        "jti": "24c63fb56e5a2d77a6b512616ca9fa24",
        "iat": 1615305159,
        "aud": "https://sp.example.com/caep",
        "events": {
            "https://schemas.openid.net/secevent/caep/event-type/\
            device-compliance-change": {
                "subject": {
                    "device": {
                        "format": "iss_sub",
                        "iss": "https://idp.example.com/123456789/",
                        "sub": "e9297990-14d2-42ec-a4a9-4036db86509a"
                    },
                    "tenant": {
                        "format": "opaque",
                        "id": "123456789"
                    }
                },
                "current_status": "not-compliant",
                "previous_status": "compliant",
                "initiating_entity": "policy",
                "reason_admin": {
                    "en": "Location Policy Violation: C076E8A3"
                },
                "reason_user": {
                    "en": "Device is no longer in a trusted location."
                },
                "event_timestamp": 1615304991643
            }
        }
    }
```

Software Configuration Management

Configuration management is the process of tightly controlling and documenting all software changes. The desired configurations are typically defined as code or data, and checked into a revision control system, allowing changes to be audited, rolled back, and so on. There are many commercial and open source options available, the most popular of which being Chef, Puppet, Ansible, and CFEngine.

Configuration management software is useful in both datacenter and client deployments, and simply becomes required beyond a certain scale. Leveraging such software comes with many security wins, such as the ability to quickly upgrade packages after vulnerability announcements or to similarly assert that there are no vulnerable packages in the wild.

Beyond auditing and strict change control, configuration management can also be used as an agent for dynamic policy configuration. If a node can get a reliable and trusted view of the world (or part of it, at least), it can use it to calculate policy and install it locally. This functionality is practically limited to the datacenter, though, since dynamic, datacenter-hosted systems are decidedly more static and predictable than client systems. We'll talk more about this mode of zero trust operation later on.

The main difference between endpoint management and software configuration management is that endpoint management is focused on the security of individual devices, whereas software configuration management is focused on the security of the software that runs on those devices.

CM-Based Inventory

We have mentioned several times the idea of using a configuration management database for inventory management purposes. This is a great first step toward a mature inventory management system and can provide a rich source of information about the various hosts and software running in your infrastructure.

We like to think that CM-based inventory management is a "freebie" in that configuration management is typically leveraged for the bevy of other benefits it brings.

Using it as an inventory database most often comes about out of convenience. Maintaining this view is important: configuration management systems aren't designed to act as inventory management systems...they're designed to act as configuration management systems! Using it as such will surely bring a few rough edges, and you will eventually outgrow it. This is not to say don't do it. It is better to actually realize a zero trust network by leveraging as much existing technology as possible than it is to never get there due to the high barrier to entry. Once we accept this fact, we can begin to leverage the wealth of data provided to us by the CM agents.

Searchable Inventory

Some CM systems centrally store the data generated by their agents. Typically, this data store is searchable, which opens lots of possibilities for young zero trust networks. For instance, the agent can perform a search to retrieve the IP address of all web servers in datacenter A and use the results to configure a host-based firewall. Audits and report generation are greatly enhanced through searchable inventory as well. This applies not only to datacenter hosts, but also to clients. By storing the agent

data and making it searchable, you can ensure that you changed the CM code to upgrade that vulnerable package, and that the package did indeed update where it said it did.

Secure Source of Truth

One important thing to remember when using CM systems in the zero trust control plane is that the vast majority of the data available to CM systems is self-reported. This is critical to understand, since a compromised machine could potentially misrepresent itself. This can lead to complete compromise of the zero trust network if these facts are not considered during its design.

Thinking back to trust management, the trusted system in this case is the provisioner. Whether it be a human or some automated system, it is in the best position to assert the critical aspects of a device, which include the following:

- Device type
- Role
- IP address (in datacenter systems)
- Public key

These attributes are considered critical because they are often used in making authorization or authentication decisions. If an attacker can update the device role, for instance, perhaps they can coerce the network to expose protected services. For this reason, restricting write access to these attributes is important. Of course, you can still use self-reported attributes for making decisions, but they should not be considered fact under any circumstance. It's useful to think of self-reported attributes as hints rather than truth.

Using Device Data for User Authorization

The zero trust model mandates authentication and authorization of both the device and the user or application. Since device authentication typically comes before user authentication, it must be done without information gained through user authentication. This is not the case for user authentication.

When user authentication occurs, device authentication has already succeeded, and the network has knowledge of the device identity. This position can be leveraged for all kinds of useful contextual knowledge, enabling us to do much stronger user authentication than was previously attainable.

One of the more common lookups one might make is to check whether we would expect this user, given the type of device or place of issue. For instance, you are unlikely to see an engineer's credentials being used from a mobile device that was

issued to HR. So while the HR employee can freely access a particular resource using their own credentials, user authentication attempts using other credentials might be blocked.

Another good signal is user authentication frequency. If you have not seen a user log in from one of their devices in over a year, and all of a sudden there is a request from that device furnishing the user's credentials—well, I think it's fair to be a bit skeptical. Could it have been stolen?

Of course, there is also a good chance that the request is legitimate. In a case like this, we lower the trust score to indicate that things are a little fishy. The lower score can then manifest itself in many ways, like still being trusted enough to read parts of the internal wiki, but not enough to log in to financial systems.

Being able to make decisions like this is a big part of the zero trust architecture and underscores the importance of a robust inventory management database. While inventory management is strictly required for device authentication reasons, the contextual advantage given to user authentication is invaluable.

Trust Signals

This section serves as a reference for various trust signals that are useful in calculating device trust score and writing policy.

Time Since Image

Over time, the likelihood that a device has been compromised increases dramatically. Endpoint security practices aim to decrease the risk associated with long-lived or long-running devices. Still, these practices are far from perfect.

Imaging a device ensures that the contents of the hard drive match a known good. While not effective against some lower-level attacks, it provides a reasonably strong assurance of trust. In the moments immediately following the image restore, a tremendous amount of trust exists in the device, as only the hardware or the restore system itself would be able to taint the process. Over time, though, that trust wears off as the system goes through prolonged exposure.

Historical Access

Device authentication patterns, similar to user authentication patterns, are important in understanding risk and act as a nice proxy for behavioral filtering. Devices that have not been seen in a while are more suspicious than ones that come and go frequently. Maybe suspicious is the wrong word, but it's certainly unusual to see one.

The request in question can also be tied to a resource, and it is wise to consider the device and the resource together in this context. For instance, a months-old device requesting access to a new resource is more suspicious than a request to a resource it has been accessing weekly for some time. This stands to say that the "first few" access attempts to a particular resource will be viewed with more skepticism than subsequent attempts.

Similarly, frequency can be analyzed to understand if a resource is being suspiciously overutilized. A request from a device that has made 100 requests in the last day, but only 104 over the last month, is certainly more suspicious than one with 0 requests on the last day and 4 in the last month.

Location

While network location, including attributes like geo-location, IP address, etc., are typically something we aim to not make strong decisions on with regard to the zero trust model, they still provide reliable trust signaling in many cases.

One such case might be a sudden location change. Since we are talking about device authentication, we can set some reasonable expectations about the way that a device should move around. For instance, a device authentication attempt from Europe might be pretty suspicious if we have authorized that same device in the US office just a couple of hours prior.

It should be noted that this is a bit of a slippery slope when it comes to the zero trust model. Zero trust aims to eliminate positions of advantage within the network, so using network location to determine access rights can be considered a little contradictory. The authors recognize this and acknowledge that location-related data can be valuable while making authorization decisions. That said, it is important that this consideration not be binary. One should look for patterns in locations, and never make an absolute decision based solely on location. For instance, a policy that dictates that an application can only be accessed from the office is a direct violation of the zero trust model.

Network Communication Patterns

For devices that are connected to networks owned by the operator, there is an opportunity to measure communication patterns to develop a norm. Sudden changes from this norm are suspicious and can affect how much the system trusts such a device. Network instrumentation and flow collection can quickly detect intrusions by observing them on the network. Making authorization decisions informed by this detection is very powerful. One example might be shutting down database access to a particular web server because that web server began making DNS queries for hosting providers on another continent.

The same applies to client devices. Consider a desktop that has never before initiated an SSH connection but is now frequently SSHing to internet hosts. It is fair to say that this change in behavior is suspicious and should result in the device being less trusted than it was previously.

Machine Learning

Machine learning assists in calculating trust scores by considering the entire context of the access request, including the user, device, and resource requested, as well as historical activity to identify anomalous requests. Machine learning models are typically trained over time to distinguish between anomalous and legitimate access patterns. As a result, machine learning can assist in reducing any potential blind spots in identifying malicious requests. However, as with other aspects of zero trust, relying solely on machine learning is not recommended; rather, using it in conjunction with other trust signals yields the best results.

Scenario Walkthrough

We'll conclude this chapter by expanding on the previous chapter's scenario walkthrough and learning about the role of device trust in the evaluation of Bob's authorization request.

Let's start with a close look at the device data store, as shown in Figure 5-5. The device data contains details specific to the device that you expect, such as the device model, operating system details, firmware-related information, and, most importantly, whether the device is in compliance with the organization's policy and when the last compliance check was reported. Also, take note of the use of universally unique identifiers (UUIDs) to uniquely identify the machine and the use of International Mobile Equipment Identity (IMEI) to uniquely identify a mobile device.

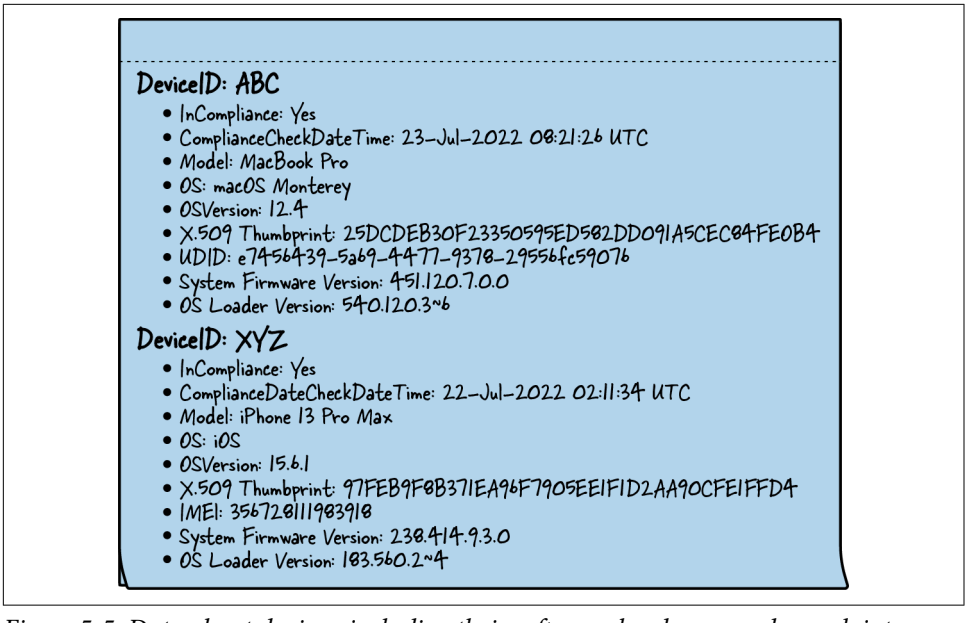

Figure 5-5. Data about devices, including their software, hardware, and complaint status, is recorded in a data store

The attribute "InCompliance" may need some explanation because it is critical in driving device trust. It is used to indicate whether a device adheres to an organization's compliance standards, and this is primarily driven by the current state of device encryption, the installation of the most recent security patch, the firmware version, and the execution of any other necessary software agents on the device, including but not limited to anti-malware software, etc. Organizations typically store device compliance status and the last time it was checked in the data store as part of device data. Furthermore, devices tend to fall out of compliance if a compliance check fails or if the device is not available for compliance checks after a certain period of time, such as 48 hours, but this duration is typically determined by the organization's compliance policy.

Following that, we examine activity logs, as shown in Figure 5-6, which depict activity from two of Bob's registered devices. By logging device activities, the trust engine can examine them for anomalous behavior. It is common for attackers to infect devices first, then use them to perform network scans, and then target critical resources using the information gathered. The example activity logs show only a few basic attributes, such as device ID, IP address, and geo-location, but this can easily be expanded to include a richer set of attributes, such as application or API being accessed, result of the activity, and so on.

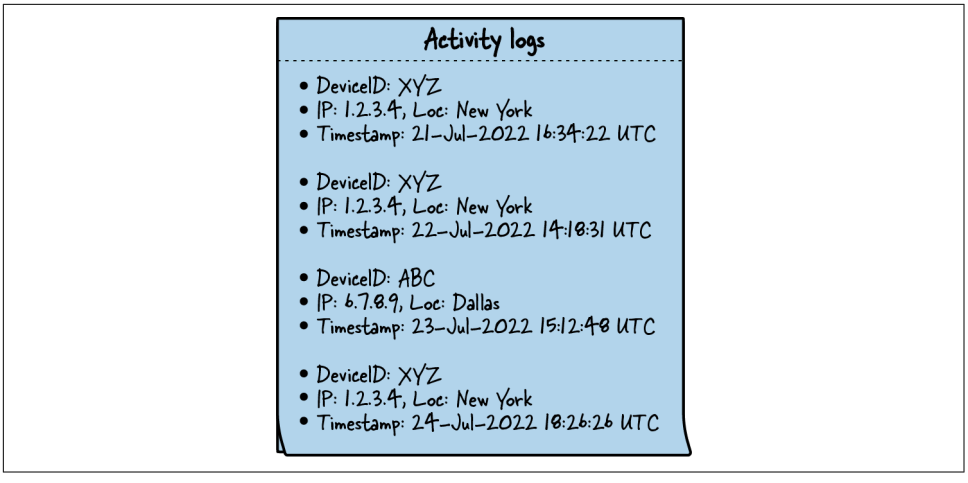

Figure 5-6. Activity logs record device activity and serve as an audit trail, which is useful for determining anomalous behavior from devices and calculating trust scores

The user store, as shown in Figure 5-7, contains Bob's user (identity) data, which includes his name, registered authentication methods, device ID, geo-location, IP address, and name. One thing to keep in mind is that certain user attributes change less frequently than others. For example, usernames tend to stay the same for the most part, whereas device IDs may change every few years as part of a typical organization's device refresh or in the case of Bring Your Own Device (BYOD), where the user leverages a noncorporation-issued device. There is always room for adding more user attributes, as well as challenges with maintaining user data, which we will discuss in Chapter 7 as part of the user trust discussion. Although the scenario focuses primarily on user identities, the points discussed also apply to machine identities.

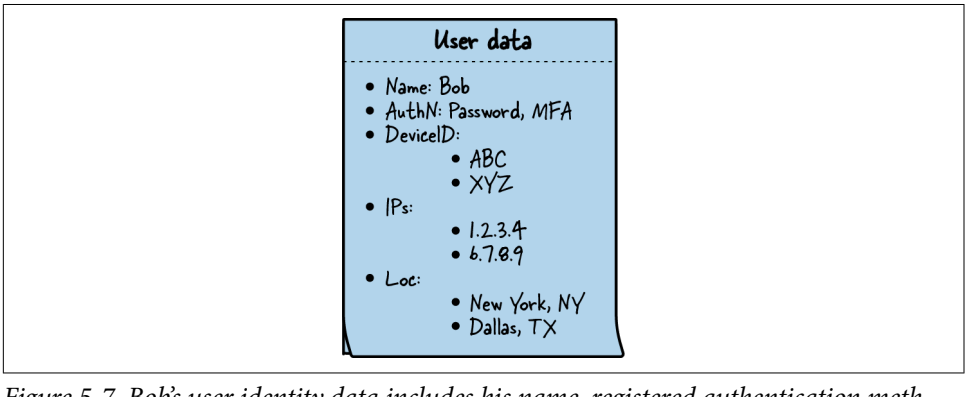

Figure 5-7. Bob's user identity data includes his name, registered authentication methods, device ID, geo-location, and IP address

The trust engine, as shown in Figure 5-8, evaluates and assigns a trust score to access requests from Bob using both dynamic and static rules. It actively uses data from various entities within the data store and deploys machine learning to ensure that any blind spots are identified, as well as using static rules for specific conditions. In this case, the trust score is calculated dynamically using machine learning to detect anomalous behavior using activity logs, which store historical user and device activity logs. The machine learning model classifies the request as highly anomalous, with a trust score of 1 or 2; moderately anomalous, with a trust score of 3 or 4; or low anomalous, with a trust score of 5 or 6.

The trust engine also takes into account the user's authentication method, specifically whether or not MFA is used to verify their identity. Requests with only a single factor receive a low trust score of 3, whereas requests with MFA receive a high trust score of 5. The final trust score is calculated by averaging all of the scores assigned to a request. Please keep in mind that how a trust score is calculated in real-world zero trust implementations is heavily dependent on the software calculating the score, but every request must receive a final trust score that the policy engine can use for decision making.

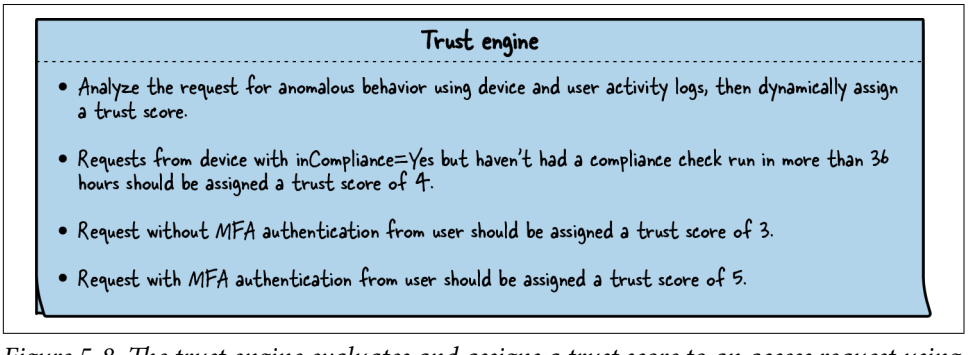

Figure 5-8. The trust engine evaluates and assigns a trust score to an access request using both dynamic and static rules

Finally, as illustrated in Figure 5-9, we have a policy engine that defines rules based on the overall context of the authorization request, which includes the user, application, device, regulatory requirements, and trust score. The policy engine also takes the "deny all" approach. Essentially, unless the request is explicitly permitted by one of the rules, it will be denied access to the resource.

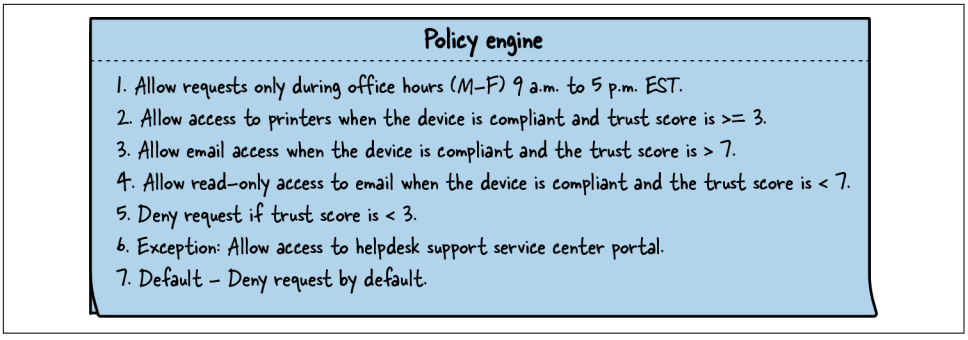

Figure 5-9. The policy engine is ultimately responsible for granting or denying access requests

Let's go through a few use cases.

Use Case: Bob Wants to Send a Document for Printing

Here is what we know about Bob's request:

- Bob is requesting access to send a document to an organization's printer.
- Bob is using his laptop with the device ID "ABC."
- Bob has used MFA and also used a password as the first factor for authentication.
- Bob is making the request during office hours.

Request Analysis

1. Bob's access request (action: print document, device-id: ABC, authentication: pwd/mfa, location: New York, IP: 1.2.3.4, datetime: 24-july-2022-10:00am-est-timezone) reaches the enforcement component.
2. The enforcement component forwards the access request to the policy engine for approval.
3. The policy engine receives the request and consults with the trust engine to determine the request's trust score.
4. The trust engine evaluates the request:
   - It finds no anomalies because the device access request pattern, as well as the IP address and location, appear to be consistent with historical data. It gives a high trust score of 6.
   - Bob has also completed MFA, so a trust score of 5 is assigned.
   - The device is in compliance and had its most recent compliance check less than 36 hours ago.

- Finally, the trust engine computes the average of the trust scores, which is 5.5, and returns it to the policy engine.

5. The policy engine receives the trust score of 5.5 from the trust engine.
6. For authorization, the policy engine compares the request to all policy rules:
   - This first rule results in a grant (or allow) action because the request is made during the permissible office hours.
   - The second rule results in a grant (or allow access to printer) action because the request has a trust score greater than 3.
   - Rules 3 and 4 do not apply to the current access request because it is specifically for a printer.
   - The fifth rule does not apply to the current request as the trust score is greater than 3.
   - The sixth rule does not apply to the current request as the request is not for the help desk.
   - The seventh rule, which is also a default rule, will not be applicable. This rule is only executed when no other rules are executed.
   - The policy engine stops processing and makes the final decision to allow printer access.
7. The policy engine sends an *allow* action to the enforcement component.
8. The enforcement component receives the policy engine's result and allows Bob's request to print the document.

Use Case: Bob Wants to Delete an Email

Here is what we know about Bob's request:

- Bob wants to delete an email from his inbox.
- Bob is using his mobile phone with the device ID "XYZ."
- Bob has used MFA and also used his password as the first factor for authentication.
- Bob is making the request during office hours.

Request Analysis

1. Bob's access request (action: delete email, device-id: XYZ, authentication: pwd/mfa, location: Dallas, IP: 6.7.8.9, datetime: 24-july-2022-9:45am-est-timezone) reaches the enforcement component.

2. The enforcement component forwards the access request to the policy engine for approval.
3. The policy engine receives the request and consults with the trust engine to determine the request's trust score.
4. The trust engine evaluates the request:
   - It finds no anomalies because the device access request pattern, as well as the IP address and location, appear to be consistent with historical data. It gives a high trust score of 6.
   - Bob has also completed MFA, so a trust score of 5 is assigned.
   - The device is in compliance, but its most recent compliance check was performed more than 36 hours ago, so a trust score of 4 is assigned.
   - Finally, the trust engine computes the average of trust scores, which is 5, and returns it to the policy engine.
5. The policy engine receives the trust score of 5 from the trust engine.
6. For authorization, the policy engine compares the request to all policy rules:
   - This first rule results in a grant (or allow) action because the request is made during the permissible office hours.
   - The second rule does not apply to the current access request since the request is not for the printer.
   - The third rule does not apply to the current access request because the trust score is lower than 7.
   - The fourth rule does apply to the current access request, as the trust score is less than 7, which restricts email access to read-only (no deletion or sending of email is allowed).
   - The fifth rule does not apply to the current request as the trust score is greater than 3.
   - The sixth rule does not apply to the current request as the request is not for the help desk.
   - The seventh rule, which is also a default rule, will not be applicable. This rule is only executed when no other rules are executed.
   - The policy engine stops processing and makes the final decision to only allow Bob read-only access to the email inbox, with no delete permissions.
7. The policy engine's decision is received by the enforcement component, which grants Bob read-only access to his email inbox but denies him the ability to delete emails. This is a good way to ensure that user Bob's ability to be productive is not completely hampered, but privileged tasks such as email deletion are limited.

Summary

This chapter focused on how a system can trust a device. This is a surprisingly hard problem, so a lot of different technologies and practices need to be applied to ensure that trust in a device is warranted.

We started with looking at how trust is injected into a device from human operators. For relatively static systems, we can have a person involved in providing the critical credentials, but for dynamic infrastructure, that process needs to be delegated. Those credentials are incredibly valuable, and we discussed how to safely manage them.

Devices eventually need to participate in the network, and understanding how they authenticate themselves is important. We covered several technologies, such as X.509 and TPMs, which can be used to authenticate a device on the network. Using these technologies along with databases of expected inventory can go a long way toward providing the checks and balances that give devices trustworthiness.

Trust is fleeting and degrades over time, so we talked about the mechanisms for renewing trust. Additionally, we discussed the many signals that can be continually used to gauge the trustworthiness of a device over time and the mechanisms used to manage devices. Perhaps the most important lesson is that a device starts out in a trusted state and only gets worse from there. The rate at which its trust declines is what we'd like to keep a handle on.

The scenario walkthrough revisits Bob from the previous chapter, but this time the focus is on device trust and how the policy engine, along with other components such as the trust engine and data store, handle various use cases.

The next chapter looks at how we can establish trust in the users of the system.

CHAPTER 6

Trusting Identities

It's tempting to conflate user trust with device trust. Security-conscious organizations might deploy X.509 certificates to users' devices to gain stronger credentials than passwords provide. One could say that the device certificate strongly identifies the user, but does it? How do we know that the intended user is actually at the keyboard? Perhaps they left their device unlocked and unattended?

Conflating user identity with device identity also runs into problems when users have multiple devices, which is increasingly becoming the norm. Credentials need to be copied between several devices, putting them at increased risk of exposure. Devices might need different credentials based on their capabilities. In networks that have kiosks, this problem becomes even more difficult.

Zero trust networks identify and trust users separately from devices. Sometimes identification of a user will use the same technology that is used to identify devices, but we must be clear that these are two separate credentials.

This chapter will explore what it means to identify a user and store their identity. We will discuss when and how to authenticate users. User trust is often stronger when multiple people are involved, so we will discuss how to create group trust and how to build a culture of security.

Identity Authority

Every user has an identity, which represents how they are known in a larger community. In the case of a networked system, the identity of a user is how they are recognized in that system.

Given the large number of individuals in the world, identifying a user can be a surprisingly hard problem. Let's explore two types of identity:

- Informal identity
- Authoritative identity

Informal identity is how groups self-assemble identity. Consider a real-world situation where you meet someone. Based on how they look and act, you can build up an identity for that person. When you meet them later, you can reasonably assume that they are the same person based on these physical characteristics. You might even be able to identify them remotely—for example, by hearing their voice.

Informal identity is used in computer systems. Pseudonymous accounts—accounts that are not associated with one's real-world name—are common in online communities. While the actual identity of an individual is not necessarily known in these communities, through repeated interactions, an informal identity is created.

Informal identity works in small groups, where trust between individuals is high and the risks are relatively low. This type of identity has clear weaknesses when the stakes are higher:

- One can manufacture a fictitious identity.
- One can claim the identity of another person.
- One can create several identities.
- Multiple individuals can share a single identity.

When a stronger form of identity is required, an authority needs to create authoritative identity credentials for individuals. In the real world, this authority often falls to governments. Government-issued IDs (e.g., a driver's license or passport) are distributed to individuals to represent their identity to others. For low-risk situations, these IDs alone are sufficient proof of one's identity. However, for higher-risk situations, cross-checking the credentials against the government database provides a better guarantee.

Computer systems often need a centralized authority for user identity as well. Like in the real world, users are granted credentials (of varying strength) that identify them in the system. Based on the degree of risk, cross-checking the credentials against a centralized database may be desired. We will discuss how these systems should function later.

Credentials can be lost or stolen, so it is important that an identity authority have mechanisms for individuals to regain control of their identity. In the case of government-issued identification, a person often needs to present other identifying information (e.g., a birth certificate or fingerprint) to a government authority to have

their ID reissued. Computer systems similarly need mechanisms for a user to regain control of their identity in the case of lost or stolen credentials. These systems often require presenting another form of verification, say a recovery code or alternative authentication credential. The choice of required material to reassert one's identity can have security implications, which we will discuss later.

Bootstrapping Identity in a Private System

Storing and authenticating user identity is one thing, but how do you generate the identity to begin with? Humans interacting with computer systems need a way to digitally represent their identity, and we seek to bind that digital representation as tightly to the real-world human as possible.

The genesis of a digital identity, and its initial pairing to a human, is a very sensitive operation. Controls to authenticate the human outside of your digital system must be strong in order to prevent an attacker from masquerading as a new employee, for instance. Similar controls might also be exercised for account recovery procedures where the user is unable to provide their current credentials.

Attacking Identity Recovery Systems

Users occasionally misplace or forget authentication material such as passwords or smart cards. To recover the factor (i.e., reset the password), the user must be authenticated by alternative and sometimes untraditional means. Attacks on such systems are frequent and successful. For example, in 2012, a popular journalist's Amazon account was broken into, and the attacker was able to recover the last four digits of the most recent credit card used. With this information, the attacker called Apple support and "proved" their identity using the recovered number. Be sure to carefully evaluate such reset processes—"secret" information is often less secret than it appears.

Given the sensitivity of this operation, it is important to put good thought and strong policy around how it is managed. It is essentially a secure introduction for humans, and the good news is, we know how to do that pretty well!

Government-Issued Identification

It probably comes as no surprise that one of the primary recommendations for accomplishing human authentication is through the use of government-issued identification. After all, human authentication is precisely what they were designed for in the first place!

In some implementations, it may even be desirable to request multiple forms of ID, raising the bar for potential forgers/imposters. It goes without saying that staff must be properly trained in validating these IDs, lest the controls be easily circumvented.

Nothing Beats Meatspace

Despite our best efforts, human-based authentication schemes remain stronger than their digital counterparts. It's always a good idea to bootstrap a human's new digital identity in person. Email or other "blind" introductions are heavily discouraged. For instance, shipping a device configured to trust the user on first use (sometimes referred to as TOFU) is not uncommon. However, this method suffers from physical weakness since the package is vulnerable to interception or redirection.

Oftentimes, the creation of the digital identity is preceded by a lengthy human process, such as a series of interviews or the completion of a business contract. The result is that the individual has been previously exposed to already-trusted individuals who have learned some of their qualities along the way. This knowledge can be leveraged for further human-based authentication, as shown in Figure 6-1.

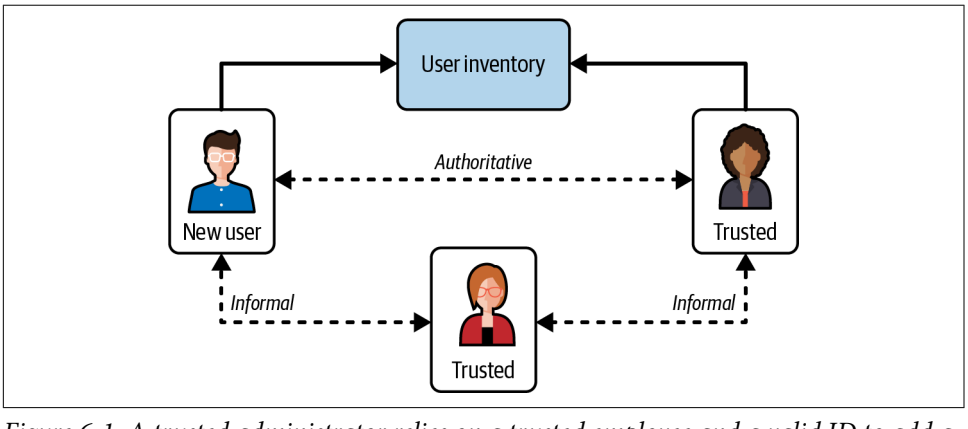

Figure 6-1. A trusted administrator relies on a trusted employee and a valid ID to add a new user to an inventory system

For instance, a hiring manager is in a good position to escort a new hire to the help desk for human authentication, since the hiring manager is presumably already familiar with the individual and can attest to their identity. While this would be a strong signal of trust, just like anything else in a zero trust network, it should not be the only method of authentication.

Expectations and Stars

There are usually many pieces of information available prior to bootstrapping a digital identity. It is desirable to use as many pieces of information as is reasonable to

assert that all of the stars line up as expected. These expectations are similar to ones set in a typical zero trust network; they are simply accrued and enforced by humans. These expectations can range from the language(s) they speak to the home address printed on their ID, with many other creative examples in between. A thorough company may choose to even use information learned through a background check to set real-world expectations. Humans use methods like this every day to authenticate each other (both casually and officially), and as a result, these methods are mature and reliable.

Storing Identity

Since we need to bridge identity from the physical world to the virtual world, identity must be transformed into bits. These bits are highly sensitive and oftentimes need to be stored permanently. Therefore, we will discuss how to store this data to ensure its safety.

User Directories

To trust users, systems typically need centralized records of those users. One's presence in such a directory is the basis by which all future authentication will occur. Having all this highly sensitive data stored centrally is a challenge that, unfortunately, cannot be avoided.

A zero trust network makes use of rich user data to make better authentication decisions. Directories will store traditional information like usernames, phone numbers, and organization role, and also extended information like expected user location or the public key of an X.509 certificate they have been issued.

Given the sensitive nature of the data being stored on users, it's best to not store all information together in a single database. Information about users isn't typically considered secret, but becomes sensitive when such data is used to make authorization decisions. Additionally, having broad knowledge of all users in a system can be a privacy risk. For example, a system that stores the last known location of all users could be used to spy on users. Stored user data can also be a security risk, if that data can be leveraged to attack another system. Consider systems that ask users fact-based information as a means to further validate their identity.

Instead of storing all user information in a single database, consider splitting the data into several isolated databases. These databases should ideally only be exposed via a constrained API, which limits the information divulged. In the best case, raw data is never divulged, but rather assertions can be made about a user by the application that has access to the data. For example, a system that stores a user's previous known location could expose the following APIs:

- Is the user currently or likely to be near these coordinates?
- How frequently does the user change locations?

Directory Maintenance

Keeping user directories accurate is critical for the safety of a zero trust network. Users are expected to come and go over the lifetime of a network system, so good onboarding and offboarding procedures should be created to keep the system accurate. As much as possible, it's best to integrate technical identity systems (LDAP—Lightweight Directory Access Protocol—or local user accounts) into organizational systems. For example, a company might have human resource systems to track employees that are joining or leaving the company. It is expected that these two sources of data are consistent with each other, but unless there is a system that has integrated the two or is checking their contents, the sets of data will quickly diverge. Creating automated processes for connecting these systems is an effort that will quickly pay dividends.

The case of two divergent identity systems raises an important point—which system is authoritative? Clearly one system must be the system of record for identity, but that choice should be made based on the needs of the organization. It doesn't much matter which system is chosen, only that one is authoritative and all other identity systems derive their data from the system of record.

Minimizing Data Stored Can Be Helpful

A system of record for identity does not need to contain all identity information. Based on our earlier discussion, it can be better to purposefully segment user data. The system of record needs to only store the information that is critical for identifying an individual. This could be as simple as storing a username and some personal information for the user to recover their identity should they forget it. Derivative systems can use this authoritative ID to store additional user information.

When to Authenticate Identity

Even though authentication is mandatory in a zero trust network, it can be applied in clever ways to significantly bolster security while at the same time working to minimize user inconvenience.

While it might be tempting (and even logical) to adopt a position of "It's not supposed to be easy; it's supposed to be secure," user convenience is among one of the most important factors in designing a zero trust network. Security technologies that present a poor user experience are often systematically weakened and undermined by

their own users. A poor experience will be a disincentive for the user from engaging with the technology, and shortcuts to sidestep enforcement will be taken more often.

Authenticating for Trust

The act of authenticating a user is, essentially, the system seeking to validate that the user is indeed who they say they are. As you'll learn in the next section, different authentication methods have different levels of strength, and some are strongest when combined with others. Due to the fact that these authentication mechanisms are never absolute, we can assign some level of trust to the outcome of the operation.

For instance, you may need only a password to log in to a subscription music service, but your investment account probably requires a password and an additional code. This is because investing is a sensitive operation: the system must trust that the user is authentic. The music service, on the other hand, is not as sensitive and chooses to not require an additional code, because doing so would be a nuisance.

By extension, a user may pass additional forms of authentication in order to raise their level of trust. This can be done specifically in a time of need. A user whose trust score has eroded below the requirements for a particular request can be asked for additional proof, which if passed will raise the trust to acceptable levels.

This is far from a foreign concept; it can be seen in common use today. Requiring users to enter their password again before performing a sensitive operation is a prime example of this concept in action. It should be noted, however, that the amount of trust one can gain through authentication mechanisms alone should not be unbound. Without it, consequences of poor device security and other undesirable signals can be washed out.

Trust as the Authentication Driver

Since authentication derives trust, and it is our primary goal to not frivolously drag users through challenges, it makes sense to use the trust score as the mechanism that mandates authentication requirements. This means that a user should not be asked to further authenticate if their trust score is sufficiently high and, conversely, that a user should be asked to authenticate when their score is too low. This is to say that, rather than selecting particular actions that require additional authentication, one should assign a required score and allow the trust score itself to drive the authentication flow and requirements. This gives the system the opportunity to choose a combination of methods in order to meet the goal, possibly reducing the invasiveness by having context about the level of sensitivity and knowledge of how much each method is trusted.

This approach is fundamentally different from traditional authentication design approaches, which seek to designate the most sensitive areas and actions, and

authenticate them the heaviest, perhaps despite previous authentication and trust accumulation. In some ways, the traditional approach can be likened to perimeter security, in which sensitive actions must pass a particular test, after which no further protections are present. Instead, leveraging the trust score to drive these decisions removes arbitrary authentication requirements and installs adaptive authentication and authorization that is only encountered when necessary.

The Use of Multiple Channels

When authenticating and authorizing a request, using multiple channels to reach the requestor can be very effective. One-time codes provide an additional factor, especially when the code-generating system is on a separate device. Push notifications provide a similar capability by using an active connection to a mobile device. There are many applications of this idea, and they can take different forms.

Depending on the use case, one might choose to leverage multiple channels as an integral part of a digital authentication scheme. Alternatively, those channels might be used purely as an authorization component, where a requestor might be prompted to approve a risky operation. Both uses are effective in their own right, though user experience should (as always) be kept in mind when deciding when and where to apply them.

Channel Security

Communication channels are constructed with varying degrees of authentication and trust. When leveraging multiple channels, it is important to understand how much trust should be placed on the channel itself. This will dictate which channels are selected for use and when. For instance, physical rotating code devices are only as secure as the system used to distribute them or the identification check required to physically obtain one from your administrator. Similarly, a prompt via a corporate chat system is only as strong as the credentials required to sign in to it. Be sure to use a different channel than the one you are trying to authenticate/authorize in the first place.

Leveraging multiple channels is effective not because compromising a channel is hard, but because compromising many is hard. We will talk more about these points in the next section.

Caching Identity and Trust

Session caching is a relatively mature technology that is well documented, so we won't spend too much time talking about it, but it is important to highlight some design choices that are important for secure operation in a zero trust network.

Frequent validation of the client's authorization is critical. This is one of the only mechanisms allowing the control plane to effect changes in data plane applications as a result of changes in trust. The more frequently this can be done, the better. Some implementations authorize every request with the control plane. While this is ideal, it may not be a realistic prospect, depending on your situation.

Many applications validate SSO tokens only at the beginning of a session and set their own tokens after that. This mode of operation removes session control from the control plane and is generally undesirable. Authorizing requests with control plane tokens rather than application tokens allows us to easily revoke when trust levels fluctuate or erode.

How to Authenticate Identity

Now that we know when to authenticate, let's dig into how to authenticate a user. The common wisdom, which is also applicable in zero trust networks, is that there are four ways to identify a user:

Something they know
: Knowledge the user alone has (e.g., a password, a personal identification number (PIN)).

Something they have
: A physical credential that the user can provide (e.g., a hardware token, key fob, smart card, USB key, or access badge).

Something they are
: An inherent/biometric characteristic that uniquely identifies the user (e.g., a fingerprint, iris scan, voiceprint, or facial recognition).

Behavioral patterns
: The use of machine learning to analyze unique behavioral patterns to verify the user's identity (e.g., how they hold their device or how they type).

We can authenticate a user using one or more of these methods. Which method or methods chosen will depend on the level of trust required. For high-risk operations, which request multiple authentication factors, it's best to choose methods that are not in the same grouping of something you know, something you have, or something you are. This is because the attack vectors are generally similar within a particular grouping. For example, a hardware token (something you have) can be stolen and subsequently used by anyone. If we pair that token with a second token, it's highly likely that both devices will be near each other and stolen together.

Which factors to use together will vary based on the device that the user is using. For example, on a desktop computer, a password (something you know) and a hardware token (something you have) is a strong combination that should generally

be preferred. For a mobile device, however, a fingerprint (something you are) and passphrase (something you know) might be preferred.

Physical Safety Is a Requirement for Trusting Users

This section focuses on technological means to authenticate the identity of a user, but it's important to recognize that users can be coerced to thwart those mechanisms. A user can be threatened with physical harm to force them to divulge their credentials or to grant someone access under a trusted account. Behavioral analysis and historical trending can help to mitigate such attempts, though they remain an effective attack vector.

Something You Know: Passwords

Passwords are the most common form of authentication used in computer systems today. While often maligned due to users' tendency to choose poor passwords, this authentication mechanism provides one very valuable benefit: when done well, it is an effective method for asserting that a user's mind is present.

A good password has the following characteristics:

It's long
: A recent NIST password standard states a minimum of 8 characters, but 20+ character passwords are common among security-conscious individuals. Passphrases are often encouraged to help users remember a longer password.

It is difficult to guess
: Users tend to overestimate their ability to pick truly random passwords, so generating passwords from random number generators can be a good mechanism for choosing a strong password, though convenience is affected if it cannot be easily committed to memory.

It is not reused
: Passwords need to be validated against some stored data in a service. When passwords are reused, the confidentiality of that password is only as strong as the weakest storage in use.

Choosing long, difficult-to-guess passwords for every service or application a user interacts with is a high bar for users to meet. As a result, users are well served to make use of a password manager to store their passwords. Using this tool will allow users to pick much harder-to-guess passwords and thereby limit the damage of a data breach.

When building a service that authenticates passwords, it's important to follow best practices. Passwords should never be directly stored or logged. Instead, a cryptographic hash of the password should be stored. The cost to brute force a password

(usually expressed in time and/or memory requirements) is determined by the strength of the hashing algorithm. NIST periodically releases standards documents that include recommended password procedures. As computers become more powerful, the current recommendations change, so it's best to consult industry best practices when choosing algorithms.

Something You Have: TOTP

The time-based one-time password, or TOTP, is an authentication standard wherein a constantly changing code is provided by the user. RFC 6238 defines the standard implemented in hardware devices and software applications. Mobile applications are often used to generate the code, which works well, since users tend to have their phones close by.

Whether using an application or hardware device, a TOTP requires sharing a random secret value between the user and the service. This secret and the current time are passed through a cryptographic hash and then truncated to produce the code to be entered. As long as the device and the server roughly agree on the current time, a matching code confirms that the user is in possession of the shared key.The storage of the shared key is critical, both on the device and on the authenticating server. Losing control of that secret will permanently break this authentication mechanism. The RFC recommends encrypting the key using a hardware device like a TPM, and then limiting access to the encrypted data.

Exposing the shared key to a mobile device places it in greater danger than if it is on a server. The device could connect to a malicious endpoint that might be able to extract the key. To mitigate this vector, an alternative to a TOTP is to send to the user's mobile phone a random code over an encrypted channel. This code is then entered on another device to authenticate that the user is in possession of their mobile phone.

SMS Is Not a Secure Communication Channel

Sending the user a random code for authentication requires that the authentication code is reliably delivered to the intended device and is not exposed during transit. Systems have previously sent random codes as SMS messages, but the SMS system does not make sufficient guarantees to protect the random code in transit. Using SMS for this system is therefore not recommended.

Something You Have: Certificates

Another method to authenticate users is to generate per-user X.509 certificates. The certificate is derived from a strong private key and then signed using the private key of the organization that provided the certificate. The certificate cannot be modified without invalidating the organization's signature, so the certificate can be used

as a credential with any service that is configured to trust the signature of the organization.

Since an X.509 certificate is meant for consumption by a computer, not by humans, it can provide much richer details when presented to a service for authentication. As an example, a system could encode metadata about the user in the certificate and then trust that data since it has been signed by a trusted organization. This can alleviate the need to create a trusted user directory in less mature networks.

Using certificates to identify users relies heavily on those certificates being securely stored. It is strongly preferred to both generate and store the private key component on dedicated hardware so as to prevent digital theft. We'll talk more about that in the next section.

Something You Have: Security Tokens

Security tokens are hardware devices that are used primarily for user authentication, but they have additional applications. These devices are not mass storage devices storing a credential that was provisioned elsewhere. Instead, the hardware itself generates a private key. This credential information never leaves the token. The user's device interacts with the hardware's APIs to perform cryptographic operations on behalf of the user, proving that they are in possession of the hardware.

As the security industry progresses, organizations are increasingly turning toward hardware mechanisms for authenticating user identity. Devices like smart cards or Yubikeys can provide a 1:1 assertion of a particular identity. By tying identity to hardware, the risk that a particular user's credentials can be duplicated and stolen without their knowledge is greatly mitigated, as physical theft would be required.

Storing a private key in hardware is by far the most secure storage method we have today. The stored private key can then be used as the backing for many different types of authentication schemes. Traditionally, they are used in conjunction with X.509, but a new protocol called Universal 2nd Factor (U2F) is gaining rapid adoption. U2F provides an alternative to full-blown PKI, offering a lightweight challenge-response protocol that is designed for use by web services. Regardless of which authentication scheme you choose, if it relies on asymmetric cryptography, you should probably be using a security token.

While these hardware tokens can provide strong protection against credential theft, they cannot guarantee that the token itself isn't stolen or misused. Therefore, it's important to recognize that while these tokens are great tools in building a secure system, they cannot be a complete replacement for a user asserting their identity. If we want the strongest guarantee that a particular user is who they claim to be, using a security key with additional authentication factors (e.g., a password or biometric sensor) is still strongly recommended.

Something You Are: Biometrics

Asserting identity by recognizing physical characteristics of the user is called biometrics. Biometrics is becoming more common as advanced sensors are making their way into devices we use every day. This authentication system offers better convenience and potentially a more secure system if biometric signals, such as the following, are used wisely:

- Fingerprints
- Handprints
- Retina scans
- Voice analysis
- Face recognition

Using biometrics might seem like the ideal authentication method. After all, authenticating a user is validating that they are who they say they are. What could be better than measuring physical characteristics of a user? While biometrics is a useful addition to system security, there are some downsides that should not be forgotten.

Authenticating via biometrics relies on accurate measurement of a physical characteristic. If an attacker is able to trick the scanner, they are able to gain entry. Fingerprints, being a common biometric, are left on everything a person touches. Attacks against fingerprint readers have been demonstrated—attackers obtain pictures of a latent fingerprint and then 3D print a fake one, which the scanner accepts. Additionally, biometric credentials cannot be rotated, since they're a physical characteristic. They can also present an accessibility issue if, for example, an individual is born without fingerprints (a condition known as adermatoglyphia) or if they lost their fingers in an accident.

Finally, biometrics can present surprising legal challenges when compared against other authentication mechanisms. In the United States, for example, a citizen can be compelled by a court to provide their fingerprint to authenticate to a device, but they cannot be compelled to divulge their password, owing to their Fifth Amendment right against self-incrimination.

Behavioral Patterns

Behavioral authentication is a method of identity verification that uses machine learning to analyze a person's unique behavioral patterns (such as how they type, how they hold their device, etc.) and identify individual characteristics that can then be used to verify their identity. It is effective because it can quickly adapt to changes in a person's behavior, making it difficult for someone to replicate another person's behavioral patterns.

This form of authentication is often used in conjunction with other forms of authentication, such as passwords or PINs, to provide an additional layer of security because, as we know, no single form of authentication is 100% effective.

There are a couple of drawbacks of behavioral authentication:

- It can be more intrusive than other forms of authentication, such as passwords or PINs, because behavioral authentication requires users to provide more personal information, such as their fingerprints or iris scans.
- Behavioral authentication can be less reliable than other forms since behavioral patterns can change over time, making it difficult for the system to accurately verify a person's identity.

Despite these drawbacks, behavioral authentication is another powerful tool to protect sensitive information and prevent identity theft.

Out-of-Band Authentication

Out-of-band authentication purposefully uses a separate communication channel from the original channel the user used to authenticate that request. For example, a user logging in to a website for the first time on a device might receive a phone call to validate the request. By using an out-of-band check, a service is able to raise the difficulty of breaking into an account, since the attacker would need control of the out-of-band communication channel as well.

Out-of-band checks can come in many forms. These forms should be chosen based on the desired level of strength needed for each interaction:

- A passive email can inform users of potentially sensitive actions that have recently taken place.
- A confirmation can be required before a request is completed. Confirmation could be a simple "yes," or it could involve entering a TOTP code.
- A third party could be contacted to confirm the requested action.

When used well, out-of-band authentication can be a useful tool to increase the security of the system. As with all authentication mechanisms, some level of taste is required to choose the right authentication mechanism and frequency, based on the request taking place.

Single Sign-On

Given the large number of services users interact with, the industry would prefer to decouple authentication from end services. Having authentication decoupled provides benefits to both the service and the user:

- Users only need to authenticate with a single service.
- Authentication material is stored in a dedicated service, which can have more stringent security standards.
- Security credentials in fewer locations means less risk and eased rotations.

Single sign-on (SSO) is a fairly mature concept. Under SSO, users authenticate with a centralized authority, after which they will typically be granted a token of sorts. This token is then used in further communication with secured services. When the service receives a request, it contacts the authentication authority over a secure channel to validate the token provided by the client.

This is in contrast to decentralized authentication. A zero trust network employing decentralized authentication will use the control plane to push credentials and access policy into the data plane. This empowers the data plane to carry out authentication on its own, whenever and wherever necessary, while still being backed by control plane policy and concern. This approach is sometimes favored over a more mature SSO-based approach since it does not require running an additional service, though it introduces enough complexity that it is not recommended.

SSO tokens should be validated against the centralized authority as often as possible. Every call to the control plane to authorize an SSO token provides an opportunity to revoke access or alter the trust level (as known to the caller).

A popular mode of operation involves the service performing its own sign-in, backed by SSO authentication. The primary drawback of this approach is that it allows the control plane to authorize the request only once, and leaves the application to make all further decisions. Trust variance and invalidation is a key aspect of a zero trust network, so decisions to follow this pattern should not be taken lightly.

Existing SSO Options

SSO has been around for a long time, and as such, there are many mature protocols/technologies to support it, including these popular ones:

SAML
: Security Assertion Markup Language is an XML-based standard for securely exchanging authentication and authorization data.

WS-Federation (WS-Fed)
: WS-Fed is a protocol for negotiating the issuance of a token. It can be used by applications (relying parties) as well as identity providers (IdPs) for SSO. WS-Fed's credentials are carried in claims, and the most common claim type is, ironically, a SAML assertion.

Kerberos
: A mature protocol that is widely used in enterprise environments. It's very scalable and can be used to support SSO for many users. However, it can be complex to set up and configure.

OAuth
: A popular protocol for authorization that can also be used for SSO. While more straightforward to set up and configure than SAML or Kerberos, it doesn't work as well with mobile devices.

OpenID Connect (OIDC)
: An identity layer built on top of the OAuth 2.0 protocol. It allows clients to verify the identity of the end user based on the authentication performed by an authorization server, as well as to obtain basic profile information about the end user in an interoperable and RESTlike manner.

CAS
: The Central Authentication Service (CAS) is an open source protocol for SSO. CAS provides enterprise SSO for web and mobile applications. It also supports authentication and authorization for RESTful web services.

It is critical that authentication remains a control plane concern in a zero trust network. As such, when designing authentication systems in a zero trust network, aim for as much control plane responsibility as possible, and validate authorization with the control plane as often as is reasonably possible.

Workload Identities

A workload identity is a unique identifier assigned to a software or service workload (such as an application, script, cron job, container, and so on) that allows it to authenticate and access other services and resources. These identities are inherently distinct from user identities due to their different lifecycles and usage scenarios. As more organizations adopt DevSecOps, the lifecycle of the workload identities needs to be fully automated, and their usage should also be constantly monitored. A significant number of cloud providers already support workload identities, making it simpler for everyone in the ecosystem to utilize them. Following is a list of a few of these:

- Amazon (AWS)
- Microsoft (Azure)
- Google (GKE)

Secure Production Identity Framework For Everyone (SPIFFE)

SPIFFE is a collection of open source specifications for a framework that facilitates the provisioning and issuance of identities (including workload identities) to services across heterogeneous environments and organizational boundaries. The SPIFFE core specification includes three main tenants:

SPIFFE ID
: A standard that specifies how services identify themselves to one another. These are used as Uniform Resource Identifiers (URIs).

SPIFFE Verifiable Identity Document (SVID)
: There is a standard for encoding SPIFFE IDs in a cryptographically verifiable document known as a SPIFFE Verifiable Identity Document (SVID).

Workload API
: An API specification for issuing and/or retrieving SVIDs.

While SPIFFE primarily focuses on specification and framework, the SPIFFE Runtime Environment (SPIRE) implements the SPIFFE APIs in a production-ready manner and performs node and workload attestation in order to securely issue SVIDs to workloads and verify the SVIDs of other workloads based on a predefined set of conditions. You can learn more about SPIRE architecture and implementation and real-world case studies.

Both SPIFFE and SPIRE are graduated projects of the Cloud Native Computing Foundation (CNCF).

Moving Toward a Local Auth Solution

Local authentication that is extended out into remote services is another authentication mechanism that is increasingly becoming a possibility. In this system, users authenticate their presence with a trusted device, and then the device is able to attest to that identity with a remote service. Open standards like the FIDO Alliance's UAF standard use asymmetric cryptography and local device authentication systems (e.g., passwords and biometrics) to move trust away from a large number of services to relatively few user-controlled endpoints.

UAF, in a way, looks a lot like a password manager. However, instead of storing passwords, it stores private keys. The authenticating service is then given the user's public key and is thereby able to confirm that the user is in possession of the private key. By moving authentication into a smart local device, a number of benefits emerge:

- Replay attacks can be mitigated via a challenge-and-response system.
- Man-in-the-Middle attacks can be thwarted by having the authentication service refuse to sign the challenge unless it originated from the same domain the user is visiting.
- Credential reuse is nonexistent, since per-service credentials can be trivially generated.

Authenticating and Authorizing a Group

Nearly every system has a small set of actions or requests that must be closely guarded. The amount of risk one is willing to tolerate in this area will vary from application to application, though there is practically no lower limit.

One risk you encounter as you approach zero is the limited amount of trust you should place in any single human being. Just like in real life, there are many times in which it is desirable to gain the consent of multiple individuals in order to authorize a particularly sensitive action. There are a couple of ways that this can be achieved in the digital realm, and the cool part is, we can cryptographically guarantee it!

Shamir's Secret Sharing

Shamir's Secret Sharing is a scheme for distributing a single secret among a group of individuals. The algorithm breaks the original secret into *n* parts, which can then be distributed (Figure 6-2). Depending on how the algorithm was configured when the parts were generated, *k* parts are needed to recalculate the original secret value. When protecting large amounts of data using Shamir's Secret Sharing, a symmetric encryption key is usually split and distributed instead of using the algorithm directly on data. This is because the size of the secret that is being split needs to be smaller than some of the data used in the secret-sharing algorithm.

```
~ $ echo 'this is a secret' | ssss-split -n 5 -t 2
Generating shares using a (2,5) scheme with dynamic security level.
Enter the secret, at most 128 ASCII characters: Using a 128 bit security level.
1-4054162f42f328c2ecbff990e9e1996f
2-93285deac4d6406cde841b05b350f61f
3-22039b5646ca98093092ba897ac02cb0
4-35d0ca61c89c9130baf3de2f06322866
5-84fb0cdd4a80495554e57fa3cfa2f2c9
~ $ ssss-combine -t 2
Enter 2 shares separated by newlines:
Share [1/2]: 5-84fb0cdd4a80495554e57fa3cfa2f2c9
Share [2/2]: 4-35d0ca61c89c9130baf3de2f06322866
Resulting secret: this is a secret
```

Figure 6-2. An example ssss session

A Unix/Linux version of this algorithm is called ssss. Similar applications and libraries exist for other operating systems or programming languages.

Red October

Cloudflare's Red October project is another approach to implementing group authentication to access shared data. This web service uses layered asymmetric cryptography to encrypt data such that a certain number of users need to come together to decrypt the data. Encrypted data isn't actually stored on the server. Instead, only user public/private key pairs (encrypted with a user-chosen password) are stored.

When data is submitted to be encrypted, a random encryption key is generated to encrypt the data. This encryption key is then itself encrypted using unique combinations of user-specific encryption keys, based on an unlock policy that the user requests. In the simplest case, a user might encrypt some data such that two people in a larger group need to collaborate to decrypt the data. In this scenario, the original encrypted data's encryption key is therefore doubly encrypted with each unique pair of user encryption keys.

About DNS Root Zone Signing

The DNS Root Zone Signing Ceremony is an interesting example of a group authentication procedure. This ceremony is used to generate the root keys upon which all DNSSEC trust is based. If the root key is compromised, the entire DNSSEC system's trustworthiness would be compromised, so the root key ceremony is built specifically to mitigate that risk.

The first ceremony occurred on June 16, 2010, and a new ceremony occurs every quarter. The ceremony utilizes seven actors, each with a different role. The ceremony mitigates the risk of compromise to a one-in-a-million chance, assuming a dishonesty rate of 5% among the actors in the ceremony. A strict procedural document is generated in order to organize the ceremony. HSMs, biometric scanners, and air-gapped systems are used to protect the digital key. In the end, a new public/private key pair is generated and signed, continuing the internet's trust anchor's status for another quarter. You can read more about the signing ceremony on Cloudflare's website, or you can view the materials for each ceremony on IANA's website.

See Something, Say Something

Users in a zero trust network, like devices, need to be active participants in the security of the system. Organizations have traditionally formed dedicated teams to focus on the security of the system. Those teams, more often than not, took that mandate to mean that they were solely responsible for the system's security. Changes needed

to be vetted by them to ensure that the system's security was not compromised. This approach produces an antagonistic relationship between the security team and the rest of the organization, and as result, reduces security.

A better approach is to build a culture of collaboration toward the security of the system. Users should be encouraged to speak up if something they do or witness looks odd or dangerous, even if it's small. This sharing of knowledge will give much better context on the threats that the security team is working to defend against. Reporting phishing emails, even when users did not interact with them, can let the security team know if a determined attacker is attempting to infiltrate the network. Devices that are lost or stolen should be reported immediately. Security teams might consider providing ways for users to alert them day or night in the event that their device has gone missing.

When responding to tips or alerts from users, security teams should be mindful of how their response to the incident affects the organization more broadly. A user who is shamed for losing a device will be less willing to report the loss in a timely manner in the future. Similarly, a late-night false alarm should be met with thanks to ensure that reporters don't second-guess themselves. As much as possible, try to bias the organization toward overreporting.

Trust Signals

Historical user activity is a rich source of data for determining the trustworthiness of a user's current actions. A system can be built that mines user activity to build up a model of expected behavior. This system will then compare current behavior against that model as a method for calculating a trust score of a user.

Humans tend to have predictable access patterns. Most people will not try to authenticate multiple times a second. They also are unlikely to try to authenticate hundreds of times. These types of access patterns are extremely suspicious and are often mitigated via active methods like CAPTCHAs (automated challenges that only a human is able to answer) or locked accounts. Reducing false positives requires setting fairly high bars to be actively banned. Including this activity in an overall threat assessment score can help catch suspicious, but not obviously bad, behavior.

Looking at access patterns doesn't need to be restricted to authentication attempts. Users' application usage patterns can also reveal malicious intent. Most users tend to have fairly limited roles in an organization and therefore might only need to access a subset of data that is available to them. In an attempt to increase security, organizations may begin removing access rights from employees unless they definitely need the access to do their job. However, this type of restrictive access control can impact the ability of the organization to respond quickly to unique events. System administrators are a class of users that are given broad access, thereby weakening this

approach as a defense mechanism. Instead of choosing between these two extremes, we can score the user's activity in aggregate and then use their score to determine if they are still trusted to access a particularly sensitive resource. Having hard stops in the system is still important—it's the less-clear cases where the system should trust users, but verify their trustworthiness via logged activity.

Lists of known bad traffic sources, like the one provided by Spamhaus, can be another useful signal for the trustworthiness of a user. Traffic that is originating from these addresses and is attempting to use a particular user's identity can point toward a potentially compromised user.

Geo-location can be another useful signal for determining the trustworthiness of a user. We can compare the user's current location against previously visited locations to determine if it is out of the ordinary. Has the user's device suddenly appeared in a new location in a timeframe that they couldn't reasonably travel? If the user has multiple devices, are they reporting conflicting locations? Geo-location can be wrong or misleading, so systems shouldn't weigh it too strongly. Sometimes users forget devices at home, or geo-location databases are simply incorrect.

Scenario Walkthrough

Let's run through a scenario walkthrough where Bob is making a request to a high-impact resource. The key components are shown in Figures 6-3 through 6-6 and request analysis is done next.

Use Case: Bob Wants to View a Sensitive Financial Report

Here is what we know about Bob's request:

- Bob is requesting access to sensitive, high-impact business financial reports.
- Bob is using his work laptop with device ID "ABC," which is fully compliant with organization policy.
- Bob has used a password for authentication along with SMS as an MFA method.
- Bob is making the request during office hours.

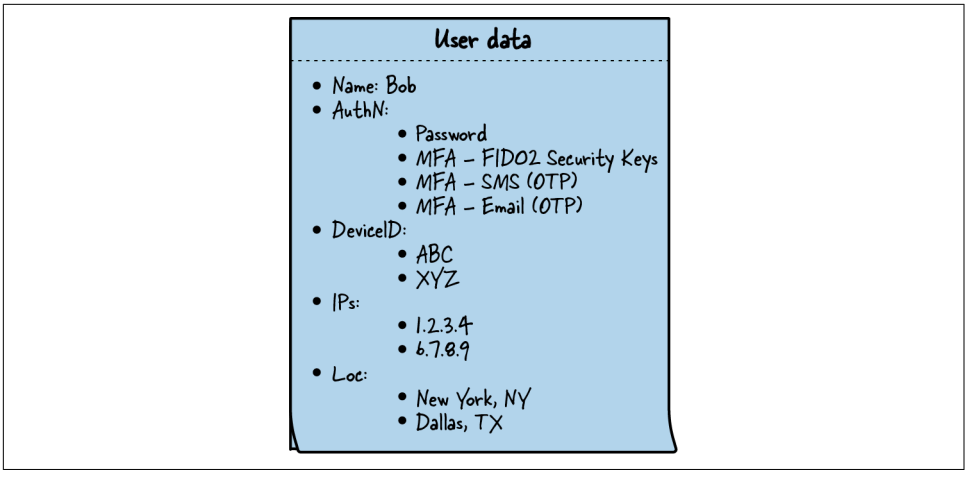

Figure 6-3. Bob's user identity data includes his name, authentication methods such as passwords, and other MFA methods such as FIDO2[1] security keys, phone, and email

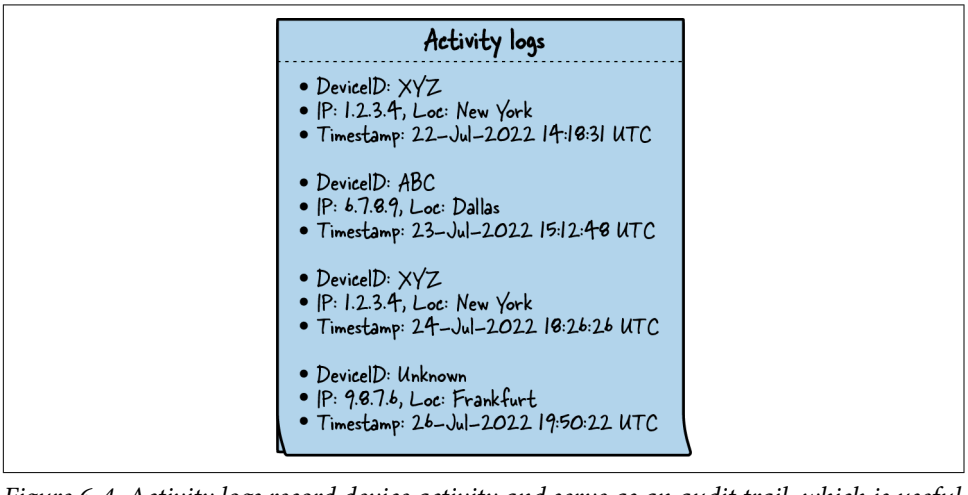

Figure 6-4. Activity logs record device activity and serve as an audit trail, which is useful for determining anomalous and suspicious behavior from devices, and calculating trust scores

1 FIDO (Fast IDentity Online) is a set of open standard authentication protocols that promote strong authentication and the elimination of passwords.

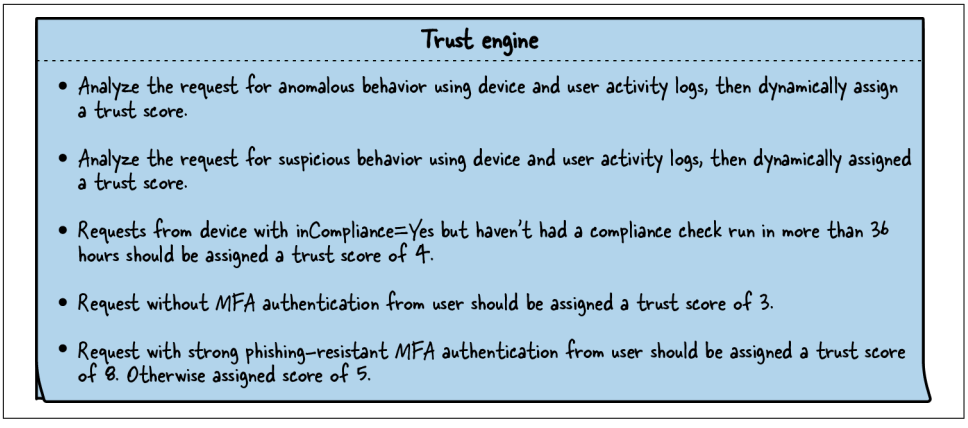

Figure 6-5. The trust engine evaluates and assigns a trust score to an access request using both dynamic and static rules

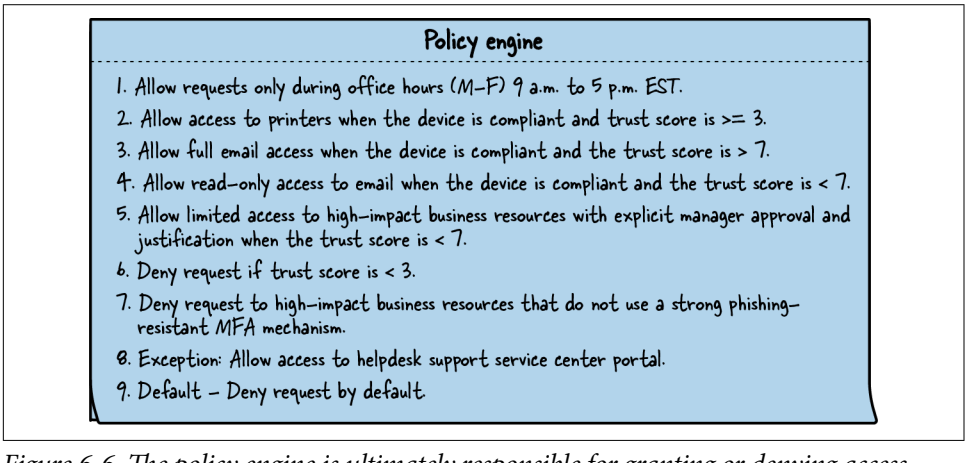

Figure 6-6. The policy engine is ultimately responsible for granting or denying access requests

Request Analysis

1. Bob's access request (action: print document, device-id: ABC, authentication: pwd/mfa-SMS, location: Dallas, IP: 6.7.8.9, datetime: 28-july-2022-11:00am-est-timezone) reaches the enforcement component.
2. The enforcement component forwards the access request to the policy engine for approval.
3. The policy engine receives the request and consults with the trust engine to determine the request's trust score.

4. The trust engine evaluates the request:
   - It detects suspicious activities in the user activity log because the IP address looks to be a common anonymous proxy exit node address, and the user device is also unknown. It gives a low trust score of 3.
   - Bob has also completed SMS-based MFA, so a trust score of 5 is assigned.
   - The device is in compliance and had its most recent compliance check less than 36 hours ago.
   - Finally, the trust engine computes the average of trust scores, which is 4, and returns it to the policy engine.
5. The policy engine receives the trust score of 4 from the trust engine.
6. For authorization, the policy engine compares the request to all policy rules:
   - This first rule results in a grant (or allow) action because the request is made during the permissible office hours.
   - The rules 2–4 do not apply to the current access request because it is specifically for a high-impact business resource, i.e., financial reports.
   - The fifth rule applies to the current request as the trust score is less than 7 and it is to gain access to financial reports. Manager approval will be required.
   - The sixth rule does not apply to the current request as the request trust score is not less than 3.
   - The seventh rule applies because the request is for high-impact business resources and SMS was used for MFA instead of robust, phishing-resistant MFA methods (e.g., a FIDO2 security key). The result of this rule's application is to deny access to the request.
   - The eighth rule will not be applicable since the request is not for the help desk. This rule is only executed when no other rules are executed.
   - The ninth rule, which is also a default rule, will not be applicable. This rule is only executed when no other rules are executed.
   - The policy engine sends a *deny* action to the enforcement component along with a suggested action to the user to utilize phishing-resistant MFA methods (such as a FIDO2 security key) rather than SMS as an MFA method. In addition, manager approval will be necessary for access to high-impact business resources because the trust score is very low.
7. The enforcement component receives the policy engine's result and denies Bob's request to access the financial report. It also provides user-friendly information about the decision for Bob to act and gain limited access to the resource by using phishing-resistant MFA methods (such as a FIDO2 security key) and asking the manager for approval of the access.

Summary

This chapter focused on how to establish trust in users in a system. We talked about how identity is defined and the importance of having an authority to reference when checking the identity of a user in the system. Users need to be entered into a system to have an identity, so we talked about some ideal ways to bootstrap their identity. Identity needs to be stored somewhere, and that system is a very valuable target for attackers. We talked about how to store the data safely, the importance of limiting the breadth of data being stored in a single location, and how to keep stored identity up to date as users come and go.

With authoritative identity defined and stored, we turned our attention to authenticating users that claim to have a particular identity. Authentication can be an annoyance for users, so we discussed when to authenticate users. We don't want users to be inundated with authentication requests, since that will increase the likelihood that they accidentally authenticate against a malicious service. Therefore, finding the right balance is critical.

There are many ways that users can be authenticated, so we dug into the fundamental concepts. We discussed several authentication mechanisms that are in use today. We also looked at some authentication mechanisms that are on the horizon as system security practices are responding to threats.

Oftentimes, increasing trust in a system of users involves creating procedures where multiple users play a role to accomplish a goal. We discussed group authentication and authorization systems like "two-person rules," which can be used to secure extremely sensitive data. We also talked about building a culture of awareness in an organization by encouraging users to report any suspicious activity.

Finally, zero trust networks can leverage user activity logs to build a profile of users to compare against when evaluating new actions. We enumerated some useful signals which can be used to build that profile.

The next chapter looks at how trust in applications can be built.

CHAPTER 7
Trusting Applications

Marc Andreessen, a notable Silicon Valley investor, famously declared that "software is eating the world." In many ways, this statement has never been truer. It is the software running in your datacenter that makes all of the magic happen, and as such, it is no secret that we wish to trust its execution.

Code, running on a trusted device, will be faithfully executed. A trusted device is a prerequisite for trusting code, which we covered in Chapter 5. However, even with our execution environment secured, we still have more work to do to trust that the code that's running on a device is trustworthy.

As such, trusting the device is just half of the story. One must also trust the code and the programmers who wrote it. With the goal being to ensure the integrity of a running application, we must find ways to extend this human trust from the code itself all the way to execution.

Trusting code refers to ensuring that the code used in software applications is free from vulnerabilities, is produced by trusted sources, and has not been tampered with.

To establish trust in code, there are a few minimum requirements that need to be met:

- The people producing the code are themselves trusted and follow secure coding practices.
- The code was scanned for vulnerabilities, signed, and accurately processed to produce a trustworthy application.
- Trusted applications are properly deployed to the infrastructure to be run.
- Trusted applications are continually monitored for updates to components, dependencies, and any attempts to coerce the application with malicious actions.

This chapter will discuss approaches to securing each of these steps, with a focus on the inheritance of trust from human to production application.

Understanding the Application Pipeline

The creation, delivery, and execution of code within a computer system is a very sensitive chain of events. These systems are an attractive target for adversaries due to the ability they offer to gain greater access. Attack vectors exist at every step, and subversion at these stages can be very difficult to detect.

Understanding the entire application pipeline, including the development, build, and distribution process, is essential. The pipeline starts with development, where applications are created and tested. The next step is the build process, where the application is compiled and packaged for distribution. Finally, the application is distributed to end users or deployed in a production environment. Therefore, we must work to ensure that every link of this chain (shown in Figure 7-1) is secured in a way that makes subversion detectable.

This process is similar to supply chain security, and the collective efforts of governments around the world to enhance security. Ensuring that military equipment is securely built/sourced is critical in ensuring the effectiveness of the fighting force, and software creation and delivery are no different.

Supply Chain Security

Supply chain security refers to the steps taken to protect the integrity and safety of the supply chain. It includes ensuring that products are not tampered with or contaminated during production, transportation, storage, and distribution. It also involves verifying that the products meet all safety and quality standards. Here are a few examples of supply chain security being compromised.

In December 2020, a malicious attack on SolarWinds' software supply chain was exposed. As one of the leading providers of IT management and network monitoring solutions to organizations worldwide, this posed an immense threat to all those who relied upon their services. By compromising SolarWinds' software build process, the attacker was able to insert malicious code into a software update, which was then distributed to thousands of customers. The incident had a widespread impact, with several government agencies and large corporations reporting that their networks had been compromised due to the attack.

In March 2023, 3CX, a well-known provider of VoIP software with over 12 million users, disclosed that malicious code had infected its desktop applications for both Windows and macOS. This attack was carried out by the North Korean hacker group UNC4736, which has been reported to be associated with the APT (Advanced Persistent Threat) group Lazarus. This incident was also the first reported case of a double supply chain attack where later attacks capitalize on the earlier attack; it involved compromised software chains of both 3CX and X_TRADER. Additional information may be found here.

These incidents reiterate the value of secure supply chains and emphasize the need for organizations to prioritize supply chain security as a critical component of their overall cybersecurity strategy. Not only do these events remind us of the dangers associated with supply chain attacks, but they also urge companies to integrate more secure measures into their cybersecurity plans. By taking action today, we can safeguard against future threats.

An application build pipeline typically begins with source code stored in a repository such as Git. The code is then automatically built, tested, and packaged through continuous integration (CI) and continuous delivery (CD) processes. Next, code signing by a trusted third party verifies the integrity of the code and ensures that it is free from any malicious code. The signed code is then distributed to the appropriate servers or devices where it is executed. Throughout the entire process, continuous monitoring is performed to identify and remediate any security incidents.

Defending Against Software Supply Chain Attacks

According to the Cybersecurity & Infrastructure Security Agency (CISA), a Software Bill of Materials (SBOM) has "emerged as a key building block in software security and supply chain management." The Software Bill of Materials is an inventory of the components and dependencies used in a software application, which plays a crucial role in ensuring the trustworthiness of applications.

Also, CISA and the National Institute of Standards and Technology (NIST) have jointly released recommendations on how software customers and vendors can use the NIST Cybersecurity Supply Chain Risk Management (C-SCRM) program and the Secure Software Development Framework (SSDF) to identify, assess, and mitigate software supply chain risks. Further details are included here.

In a secure software delivery chain, every step of the process should be fully auditable, with cryptographic validation occurring at each critical point. Generally speaking, these steps can be broken down into four distinct phases (Figure 7-1):

- Source code
- Build/compilation
- Distribution
- Execution

Let's start with trusting the source code itself.

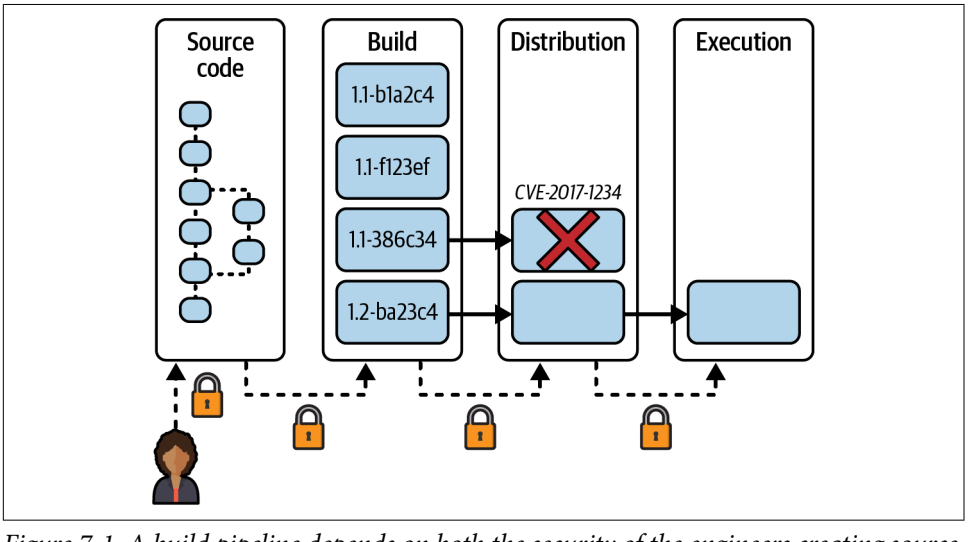

Figure 7-1. A build pipeline depends on both the security of the engineers creating source code and configuring the system, and the security of the components of the pipeline

Trusting Source Code

Source code is the first step in running any piece of software. To put it very simply, it's difficult to trust source code that is written by an untrusted human. Even with careful code auditing, it is still possible for a malicious developer to purposefully encode (and hide) a vulnerability in plain sight. While even well-meaning developers can inadvertently add weakness to an application, a zero trust network will focus on identifying malicious use instead of removing trust from those users. Setting the trusted developer problem aside for a minute, we still face the problem of securely storing and distributing the source code itself. Typically, source code is stored in a centralized code repository, against which many developers interact and commit work. These repositories must also fall under tight control, particularly if they are being used directly by systems that build/compile the code in question.

Securing the Repository

Maintaining traditional security approaches when it comes to securing a software repository is still effective, and does not prohibit the addition of more advanced security features. These include basic principles such as the principle of least access, whereby users are only given as much access to the repository as is required to complete the task at hand. In practice, this usually manifests itself as heavily limited/restricted write access.

While this approach is still valid and recommended, the story has changed a little bit with the introduction of distributed source control. With the code repository living in multiple places, it is not always possible to secure a single, centralized entity. In this circumstance, however, there remains an analog for this centralized repository—the system storing the code from which the build system reads.

In this case, it is still highly desirable to protect this system through traditional means; however, the problem becomes more difficult, since code can enter the distributed repository in any number of ways. The logical extension, then, is that securing the build source repository alone is not enough.

Authentic Code and the Audit Trail

Many version control systems (VCSs), particularly those that are distributed, store source history using cryptographic techniques. This approach, called content-addressable storage, uses the cryptographic hash of the content being stored as the identifier of that object in a database, rather than its location or coordinates. It's possible to see how a source file could be hashed and stored in such a database, thereby ensuring that any change in the source file results in a new hash. This property means that files are stored immutably: it's impossible to change the contents of the files once stored.

Some VCSs take this storage mechanism a step further by storing the history itself as an object in the content-addressable database. Git, a popular distributed VCS project, stores the history of commits to the repository as a directed acyclic graph (DAG). The commits are objects in the database, storing details like the commit time, author, and identifiers of ancestor commits. By storing the cryptographic hashes of ancestor commits on each commit itself, we form a Merkle tree, which allows one to cryptographically validate that the chain of commits are unmodified (Figure 7-2).

If a commit in the DAG is modified, its update will affect all the descendant commits in the graph, changing each commit's content, and by extension, its identifier. With the source history distributed to many contributors, the system gains another beneficial property: it's impossible to change the history without other contributors noticing.

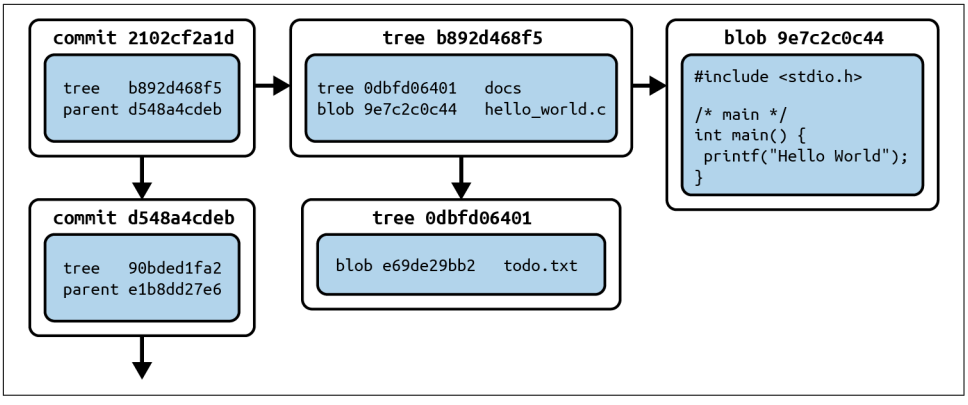

Figure 7-2. Git's database makes unwanted changes difficult, since objects are referenced using a hash of their contents

Storing the DAG in this manner gives us tamper-proof history: it's impossible to change the history subversively. However, this storage does nothing to ensure that new commits in the history are authorized and authentic. Imagine for a moment that a trusted developer is persuaded to pull a malicious commit into their local repository before pushing it to the official repository. This commit has now been added to the repository by leaning on the trusted developer's push access. Even more concerning, the authorship metadata is just plain text: a malicious committer can put whatever details they want in that field (a fact that was used amusingly to make commits appear to be authored by Linus Torvalds on GitHub).

To guard against this attack vector, Git has the ability for commits and tags to be signed using the Gnu Privacy Guard (GnuPG) key of a trusted developer. Tags, which point to the head commit in a particular history, can be signed using a GnuPG key to ensure the authenticity of a release. Signed commits allow one to go a step further and authenticate the entire Git history, making it impossible for an attacker to impersonate another committer without first stealing that committer's GnuPG key.

Signed source code clearly provides significant benefits and should be used wherever possible. It provides robust code authentication not only to humans, but machines too. This is especially important if CI/CD systems build and deploy the code automatically. A fully signed history allows build systems to cryptographically authenticate the code as trusted before compiling it for deployment.

In the Beginning, There Was Nothing

Many repositories begin with unsigned commits, transitioning to signed commits later on. In this brownfield case, the first commit to be signed is essentially endorsing all commits that came before it. This is important to understand, as you may wish to perform an audit at this time. Having said that, the overhead or difficulty of performing such an audit should not dissuade or delay the transition to signed code; the audit, if you choose to do one, can be performed in due time.

Code Reviews

As we learned in Chapter 6, it can be dangerous to concentrate powerful capabilities onto a single user. This is no different when considering source code contributions. Signed contributions enable us to authenticate the developer committing the code, but do not ensure that the code being committed is correct or safe. Of course, we do place a nontrivial amount of trust in developers, though this does not mean that said developer should unilaterally commit code to sensitive projects.

To mitigate this risk, most mature organizations implement a code review process. Under code review, all contributions must be approved by one or more additional developers. This simple process drastically improves not just the quality of the software, but also reduces the rate at which vulnerabilities are introduced, whether they be intentional or accidental.

Trusting Builds

Build servers play a critical role in the software development lifecycle as they automate the process of compiling, testing, and packaging code into a deployable software product. In addition, build servers are also responsible for transforming code into executable code and ensuring that it meets quality and security standards.

Because these servers have elevated access and produce code that is executed directly into production, they are also frequently targeted by persistent threats.

Detecting artifacts that have been compromised during the build stage can be very difficult, so it is important to apply strong protections to these services.

Software Bill of Materials (SBOM): The Risk

In trusting a build system, there are generally three things that we want to assert:

- The source code it built is the code we intended to build.
- The build process/configuration is what we intended.
- The build itself was performed faithfully, without manipulation.

Build systems can ingest signed code and produce a signed output, but the function(s) applied in between (i.e., the build itself) is generally not protected cryptographically—this is where the most significant attack vector lies.

This particular vector is a powerful one, as shown in Figure 7-3. Without the right processes and validation, subversion of this kind can be difficult or impossible to detect. For instance, imagine a compromised CI/CD system that ingests signed C code, and compiles it into a signed binary, which is then distributed and run in production. Production systems can validate that the binary is signed, but would have no way of knowing if additional malicious code has been compiled in during the build process. In this way, a seemingly secure system can successfully run malicious code in production without detection. Perhaps even worse, the consumers are fooled into thinking the output is safe. This break in the chain poses a great threat, and is a powerful attack vector.

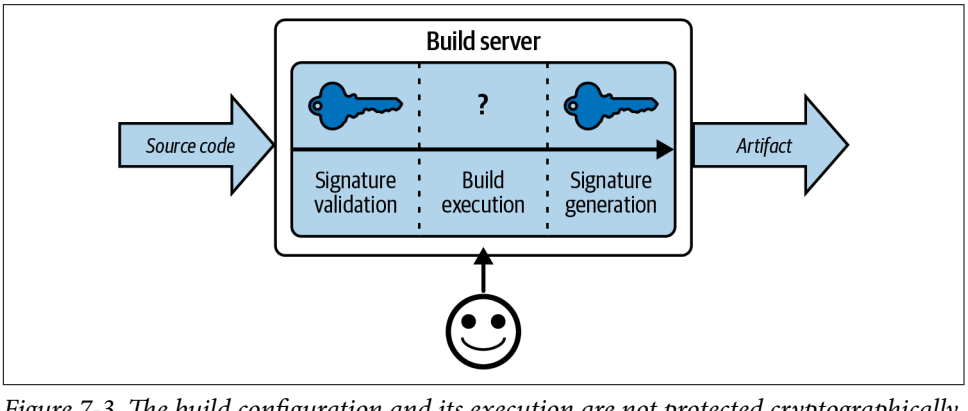

Figure 7-3. The build configuration and its execution are not protected cryptographically, in contrast to the source code and the generated artifact

Due to the sensitive nature of the build process, outsourcing the responsibility should be carefully evaluated. Things like reproducible builds can help identify compromises in this area (more on that in a bit), but can't always prevent their distribution. Is this really something you want a third-party provider to do for you? How much do you

trust them? Their security posture should be weighed against your own chance of being a high-value target.

Host Security Is Still Important

This section focuses on securing various steps of the software build process, but it is important to note that the security of the build servers themselves is still important. We can secure the input, output, and configuration of the build, but if the build server is compromised, then it can no longer be trusted to faithfully perform its duties. Reproducible builds, immutable hosts, and the zero trust model itself can help in this regard.

Trusted Input, Trusted Output

If we think of the build system as a trusted operation, it's clear that we need to trust the input of that operation in order to produce trusted output.

Let's start with trusting the input to the build system. Earlier, we discussed mechanisms for trusting the source control systems. The build system, as a consumer of the version control system, is responsible for validating the trustworthiness of the source. The version control system should be accessed over an authenticated channel, commonly TLS. Additionally, for extra security guarantees, tags and/or commits should be signed, and the build system should validate those signatures—or chain of signatures—before starting a build.

The build configuration is another important input to the build system. Attacking the build configuration could allow an attacker to direct the build system to link against a malicious library. Even seemingly safe optimization flags can be malicious in security-critical code, where timing attack mitigation code can be accidentally optimized away. Putting this configuration under source control, where it can be versioned and attested to via signed commits, helps to ensure that the build configuration is also a trusted input.

With the input sufficiently secured, we can turn our attention to the output of the build process. The build system needs to sign the generated artifacts so downstream systems can validate their authenticity. Build systems typically also generate cryptographic hashes of the build artifacts to guard against corruption or malicious attempts to replace the binaries, once produced. Securing the build artifacts and hashes, and then distributing them to downstream consumers, completes the trusted output of the build system.

Reproducible Builds

Reproducible builds are the best tool we have in guarding against subversion of the build pipeline. In short, software supporting reproducible builds is compiled in a deterministic way, ensuring that the resulting binary is exactly the same for a given source code, no matter who built it. This is a very powerful property, as it allows multiple parties to examine the source code and produce identical builds, thus gaining confidence that the build process used to generate a particular binary was not tampered with.

This can be done in a number of ways, but it generally involves a codified build process, and enables developers to set up their own build environment to produce binaries that match the distributed versions bit for bit. With reproducible builds, one can "watch" the output of a CI/CD system and compare its output to results compiled locally. In this way, malicious interference or code injection during the build process can be easily detected. When combined with signed source code, we arrive at a fairly robust process that is able to authenticate both the source code and the binary produced by it.

Virtualized Build Environments Enable Reproducible Builds

Having reproducible builds sounds easy on paper, but reproducing a built binary so it's byte-for-byte identical is a very hard problem. Distributions have historically built packages inside a virtual filesystem (a chroot jail) to ensure that all dependencies of the build are captured in the build configuration. Virtual machines or containers can be useful tools to ensure that the build environment is fully insulated from the host running the build.

Decoupling Release and Artifact Versions

Immutable builds are critical in ensuring the security of a build and release system. Without them, replacing a known good version is possible, opening up the door for attacks that target the underlying build artifact. This would enable an attacker to masquerade a "bad" version as a "good" version. For this reason, artifacts generated by build systems should have Write Once Read Many semantics.

Given the immutable artifact requirement, a natural tension arises with the versioning of those artifacts. Many projects prefer to use meaningful version numbers (e.g., semantic versioning) in their releases to communicate the potential impact to downstream consumers with an upgrade of their software. This desire to attach meaning to the version number can be difficult to incorporate into a build system that needs to ensure that every version is immutable.

For example, when working toward a major release, a project might have a misconfigured build that causes the build system to produce incorrect output. The maintainers now face a choice. They could republish the release using a patch-level bump, or they might decide to bend the rules and republish the same version using a new build artifact. Many projects choose the latter option, preferring the benefit of a clearer marketing story than the more correct reversion. This is a bad habit to get into when considering the masquerade just described.

It's clear from this example that in either case, two separate build artifacts were produced, and the version number associated with the build artifact is a separate choice for the project. Therefore, when creating a build system, it's better to have the build system produce immutable versions independent of the publicly communicated version. A later system (the distribution system) can manage the mapping of release versions to build artifact versions. This approach enables us to maintain immutable build artifacts without sacrificing usability or introducing bad security practices.

Trusting Distribution

The process of choosing which build artifacts to deliver to downstream consumers is called distribution. The build system produces many artifacts, some of which are meant for downstream consumption. Therefore, we need to ensure that the distribution system maintains control over which artifacts are ultimately delivered.

Promoting an Artifact

Based on our earlier discussion on immutable build artifacts, promotion is the act of designating a build artifact as the authoritative version without changing the contents of that artifact. This act itself should be immutable: once a version is assigned and released, it cannot be changed. Instead, a new artifact needs to be produced and released under an incrementally higher version number.

This constraint presents a chicken-and-egg scenario. Software typically includes a way to report its version number to the user, but if the version number isn't assigned until later in the build process, how does one add that version information without changing the build artifact?

A naive approach would be to subtly change the artifact during the promotion process, for example, by having the version number stored in a trivially modified location in the build artifact. This approach, however, is not preferred. Instead, release engineers should make a clear separation between the publicly released version number and the build number, which is an extra component of the release information. With this model, many build artifacts are produced that use the same public release version, but each build is additionally tagged with a unique build number (Figure 7-4). The act of releasing that version is therefore choosing the build

artifact that will be signed and distributed. Once such a version is released, all new builds should be configured to use the next target version number.

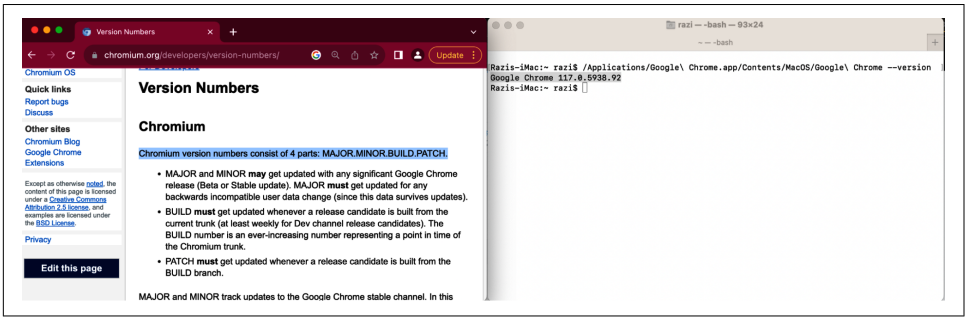

Figure 7-4. This Chromium public release version is 117.0.5938.92, using the versioning format MAJOR.MINOR.BUILD.PATCH

Of course, this promotion must be communicated to the consumer in a way that they can validate they are in possession of the promoted build, and not some intermediary and potentially flawed build. There are a number of ways to do this, and it is largely a solved problem. One solution is to sign the promoted artifacts with a release-only key, thus communicating to the consumers that they have a promoted build. Another way to do this is to publish a signed manifest, outlining the released versions and their cryptographic hashes. Many popular package distribution systems, such as APT, use this method to validate builds obtained from their distribution systems.

Distribution Security

Software distribution is similar to electricity distribution, where electricity is generated by a centralized source, and carried over a distribution network in order to be delivered to a wide consumer base. Unlike electricity, however, the integrity of the produced software must be protected while it transits the distribution system, allowing the consumer to independently validate its integrity. There are a number of widely adopted package distribution and management systems, practically all of which have implemented protections around the distribution process and allow consumers to validate the authenticity of packages received through them. Throughout this section, we will use the popular package management software Advanced Packaging Tool (APT) as an example of how certain concepts are implemented in real life, though it is important to keep in mind that there are many options available to you—APT is merely one.

Integrity and Authenticity

There are two primary mechanisms used to assert integrity and authenticity in software distribution systems: hashing and signing. Hashing a software release involves

computing and distributing a cryptographic hash representing the binary released, which the consumer can validate to ensure that the binary has not been changed since it left the hands of the developer. Signing a release involves the author encrypting the hash of the release with their private key, allowing consumers to validate that the software was released by an authorized party. Both methods are effective, and are not necessarily mutually exclusive. To better understand how these methods can be applied in a distribution system, it is useful to look at the structure and security of an APT repository.

An APT repository contains three types of files: a Release file, a Packages file, and the packages themselves. The Packages file acts as an index for all of the packages in the repository. It stores a bit of metadata on every package the repository contains, such as filenames, descriptions, and checksums. The checksum from this index is used to validate the integrity of the downloaded package before it is installed. This provides integrity, assuring us that the contents have not changed in flight. It is, however, mostly only effective against corruption, since an attacker can simply modify the index hashes if the goal is to deliver modified software. This is where the Release file comes in.

The Release file contains metadata about the repo itself (as opposed to the Packages file, which stores metadata about the packages contained within it). This includes things like the name and version of the OS distribution the repo is meant for. It also includes a checksum of the Packages file, allowing the consumer to validate the integrity of the index, which in turn can validate the integrity of the packages we download. That's great, except an attacker can simply modify the Release file with the updated hash of the Packages file and be on their way.

So, we introduce cryptographic signatures (Figure 7-5). A signature provides not only integrity for the contents of the signed file (since a hash is included in the signature), but also authenticity, since successful decryption of the signature proves that the generating party was in the presence of the private key.

Using this principle, the maintainer of the software repo signs the Release file with a private key, to which there is a well-known and well-distributed public key. Any time the repo is updated, package file hashes are updated in the index, and the index's final hash is updated in the Release file, which is then signed. This chain of hashes, the root of which is signed, provides the consumer with the ability to authenticate the software they are about to install.

In the event that you're unable to sign a software release in some way, it is essential to fall back to standard security practices. You will need to ensure that all communication is mutually authenticated—this means traffic to, from, and in between any distribution repositories. Additionally, you'll need to ensure that the storage the repository leverages is adequately secured, be it Amazon Simple Storage (Amazon S3), or otherwise.

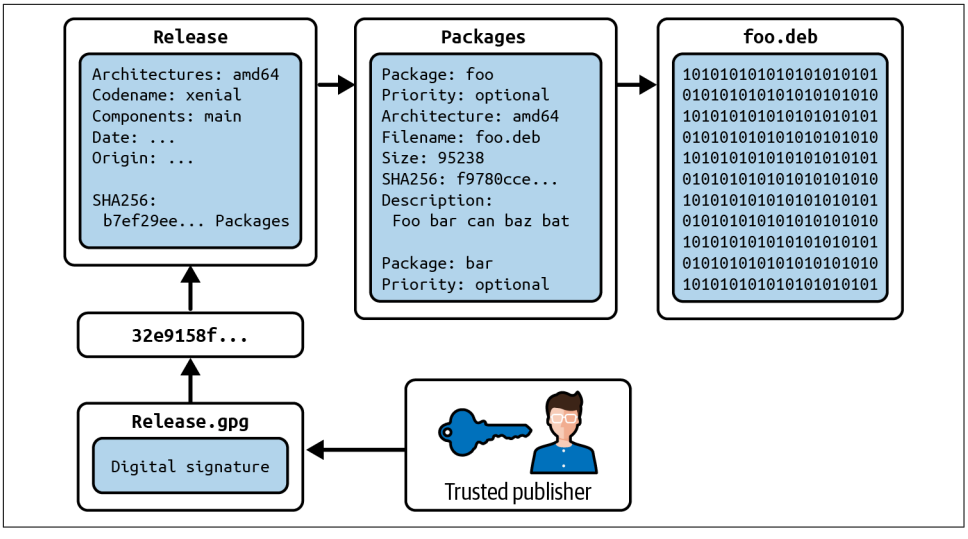

Figure 7-5. The maintainer signs the Release file, which contains a hash of the Packages index, which contains hashes of the packages themselves

Trusting a Distribution Network

When distributing software with a large or geographically disparate consumer base, it is common to copy the software to multiple locations or repositories in order to meet scaling, availability, or performance challenges. These copies are often referred to as mirrors. In some cases, particularly when dealing with publicly consumed software, the servers hosting the mirrors are not under the control of the organization producing the software. This is obviously a concern, and underscores the requirement of a software repo to be authenticated against the author (and not the repo owner).

Referring back to APT's hashing and signing scheme, it can be seen that we can, in fact, authenticate the Release file against the author using its signature. This means that for every mirror we access, we can check the Release signature to validate that the mirror is in fact a faithful copy of the original release.

One might think that by signing the Release file, software can be distributed through untrusted mirrors safely. Additionally, repositories are often hosted without TLS under the assertion that the signing of the release is sufficient for protecting the distribution network. Unfortunately, both of these assertions are incorrect.

There are several classes of attacks that open up when connecting to an untrusted mirror, despite the fact that the artifact you're obtaining is ultimately signed. For instance, a downgrade to an older (signed) version can be forced, as the artifact served will still be legitimate. Other attack vectors can include targeting the package

management client itself. In the interest of protecting your clients, always make sure they are connecting to a trusted distribution mirror.

The dearth of TLS-protected repositories presents another vulnerability to the distribution of software. Attackers that are in a position to modify the unprotected response could perform the same attacks that an untrusted mirror could. Therefore, the best solution to this problem is moving package distribution to TLS-protected mechanisms. By adding TLS, clients can validate that they are in fact connecting to a trusted repository and that no tampering of the communication can occur.

Improving the Software Supply Chain Integrity

Cyberattacks like those on SolarWinds, Codecov, etc., have exposed the severe consequences of supply chain integrity flaws, which have caused significant disruption. In addition, they have demonstrated that there are inherent risks not only in the code itself, but also at multiple locations in the complex process of incorporating that code into software systems, or the software supply chain. This is where frameworks like Supply-chain Levels for Software Artifacts (SLSA) can assist with automation that tracks code handling from source to binary, safeguarding against tampering regardless of the software supply chain's complexity. This also instills confidence that the analysis and review conducted on the source code will continue to apply to the binary after the build and distribution process.

Humans in the Loop

With a secure pipeline crafted, we can make considered decisions on where humans are involved in that pipeline. By limiting human involvement to only a few key points, the release pipeline stays secure while also ensuring that attackers are not able to leverage automation in the pipeline to deliver malicious software. The ability to commit code to the version control system is a clear spot where humans are involved. Depending on the sensitivity of the project, requiring humans to only check in signed commits provides strong confidence that the commit is authentic.

Once committed, humans needn't be involved in the building of software artifacts. Those artifacts should ideally be produced automatically in a secured system. Humans should, however, be involved in the process of choosing which artifact is ultimately distributed. This involvement could be implemented using various mechanisms: copying an artifact from the build database to the release database or tagging a particular commit in source control, for example. The mechanism by which humans certify a releasable binary doesn't matter much, as long as that mechanism is secured.

It's tempting when building secure systems to apply extreme measures to mitigate any conceivable threat, but the burden placed on humans should be balanced against the potential risk. In the case of software that is widely distributed, the private signing key should be well guarded, since the effort of rotating a compromised key would be extreme. Organizations that release software like this will commonly use "code signing ceremonies," where the signing key is stored on a hardware security module (HSM) and unlocked using authorization from multiple parties, as a mitigation against the theft of this highly sensitive key. For internal use-only software, the effort to rotate a key might be reasonably less, so more lax security practices are reasonable. An organization might still prefer a code signing ceremony for particularly sensitive internal applications—a system that stores credit card details, for example.

Consequences of Insecure Code Signing System

As mentioned earlier in the chapter, in 2020, SolarWinds, a leading IT management and monitoring software provider, fell victim to a highly sophisticated cyberattack. The attackers compromised the company's software development environment and inserted malicious code into its Orion platform, distributed to numerous government agencies, Fortune 500 companies, and other organizations around the world. The attackers used a valid code signing certificate to sign the compromised software update, allowing it to bypass security measures and infiltrate targeted networks. This incident highlights the importance of securing code signing keys. Organizations with higher risk should consider implementing robust security measures, like code signing ceremonies, to protect their digital assets from potential attacks.

Trusting an Instance

Understanding what is running in your infrastructure is important when designing a zero trust network. After all, how can you know what to expect on your network if you don't know what to expect on your hosts? A solid understanding of the software (and versions) running in your datacenter will go a long way in both breach detection and vulnerability mitigation.

Upgrade-Only Policy

Software versions are important constructs in determining exactly which version of the code you have and how old it is. Perhaps most importantly, they are used heavily in order to determine what vulnerabilities one might be exposed to, given the version they are running.

Vulnerability announcements/discoveries are typically associated with a version number (online service vulnerabilities being the exception), and generally include

the version numbers in which the vulnerability was fixed. With this in mind, we can see that it might be desirable to induce a version downgrade in order to expose a known vulnerability. This is an effective attack vector, as the software being coerced to run is frequently authorized and trusted, since it is a perfectly valid release, albeit an older one.

If the software is built for internal distribution, perhaps the distribution system serves only the latest copy. Doing this prevents a compromised or misconfigured system from pulling down an old version that may contain a known vulnerability. It is also possible to enforce this roll-forward mentality in hardware. Apple iOS famously uses a hardware security chip to validate software updates and to ensure that only signed software built after the currently installed software can be loaded.

Authorized Instances

The importance of knowing what's running is more nuanced than simply understanding what is the latest deployed version. There are many edge cases that arise, such as a host that has fallen out of the deployment system—one that has been previously authorized but is now "rogue" by dint of no longer receiving updates. To guard against cases like this, it's critical that running instances be individually authorized.

It is possible to use techniques described in Chapter 4 to build dynamic network policy in an effort to authorize application instances, but network policy is often host/device oriented rather than application oriented. Instead, we can leverage something much more application-centric in the pursuit of authorizing a running instance: secrets.

Most running applications require some sort of secret in order to do their job. This secret can manifest itself in many ways: an API key, an X.509 certificate, or even credentials to a message queue are common examples. Applications must obtain the secret(s) in order to run, and furthermore, the secret must be valid. The validity of a secret (as obvious as it sounds) is the key to authorizing a running application, as with validation comes invalidation.

Attaching a lifetime to a secret is extremely effective in limiting its abuse. By creating a new secret for every deployed instance and attaching a lifetime to the secret, we can assert that we know precisely what is running, since we know precisely how many secrets we have generated, who we gave them to, and their lifetimes. Allowing secrets to expire mitigates the impact of "rogue" instances by ensuring they will not operate indefinitely.

Of course, someone must be responsible for generating and injecting these secrets at runtime, and this is no small responsibility. The system carrying this responsibility is ultimately the system that is authorizing the instance to run. As such, it makes sense

for this responsibility to fall in the hands of the deployment system, since it already carries similar responsibility.

Trusted Third Parties in Instance Authorization

Rather than giving your deployment system direct access to secrets, it is possible to leverage a trusted third party, allowing the deployment system to instead assign policy dictating which secrets the running instance can access. HashiCorp's Vault, for instance, has a feature called response wrapping in which an authorized party can request a secret to be generated and stored for later retrieval. In the context of a deployment system, the deploy itself could contact Vault and direct the creation of secrets on behalf of the authorized instances, injecting a one-time token into the runtime, which the application can use to retrieve the generated secrets, as shown in Figure 7-6.

In such a system, the deployment service notifies the secret management service of the impending changes, authorizing the new application instances. During the deploy itself, the deployment service injects key(s), which the new instances use to identify themselves to the secret management system, which is expecting their request. The secret management system then provisions unique time-bound credentials, returns them to the application, and further continues to manage their lifecycle.

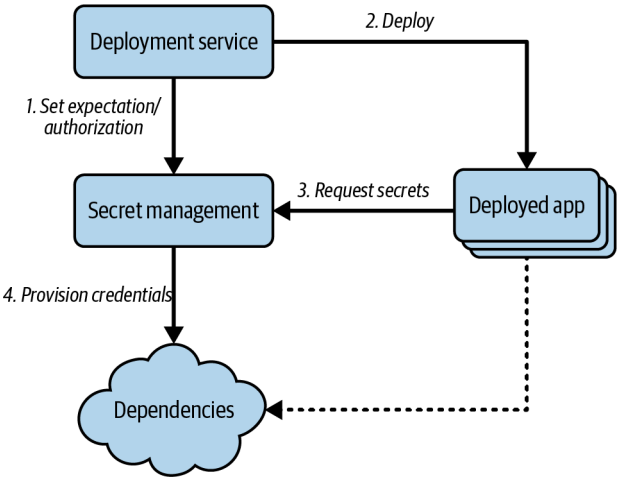

Figure 7-6. Example flow of a system that provisions per-deployment credentials

It doesn't take much thought to realize the power of a system that can create and (potentially) retrieve secrets. With great power comes great responsibility. If allowing an autonomous system to generate and distribute secrets comes with too much risk for your organization, you might consider including a human at this step. Ideally,

this would manifest as a human-approved deployment in which a TOTP or other authenticating code is provided. This code will, in turn, be used to authorize the creation/retrieval of the secrets by the deployment system.

Runtime Security

Trusting that an application instance is authorized/sanctioned is only half of the concern. There is also the need to validate that it can run safely and securely through its lifecycle. We know how to deploy an application securely, and validate that its deployment is authorized, but will it remain an authorized and trustworthy deployment for the entirety of its life?

There are many vectors that can compromise perfectly authorized application instances, and it might be no surprise to learn that these are the most commonly used vectors. For instance, it is typically easier to corrupt an existing government agent than it is to masquerade as one or attempt to become one. For this reason, individuals with outstanding debt are commonly denied security clearance. They might be fully trusted at the time they are granted clearance, but how susceptible are they to bribery if they are in debt? Can they be trusted in this case?

Secure Coding Practices

Most (all?) application-level vulnerabilities start with a latent bug, which an attacker can leverage to coerce the trusted application to perform an undesirable action. Fixing each bug in isolation will result in a game of whack-a-mole, where developers fix one security-impacting bug only to find two more. Truly mitigating this exposure requires a shift in the mindset of the application developers to secure coding practices. Injection attacks, where user-supplied data is crafted to exploit a weakness in an application or related system, commonly occur when user data is not properly validated before being processed. This type of attack is mitigated by introducing several layers of defenses. Application libraries will carefully construct APIs that avoid trusting user-supplied data. Database querying libraries, for example, will provide APIs to allow the programmer to separate the static query from variables that are provided by the user. By instituting a clear separation between logic and data, the potential for injection attacks is greatly reduced.

Having clear APIs can also support automated scans of application software. Security-aware organizations are increasingly running automated analysis tools against their source code to detect and warn application developers of insecure coding practices. These systems warn about using insecure APIs, for example, by highlighting database queries that are constructed using string concatenation instead of the API discussed earlier. Beyond warning about insecure APIs, application logic can be traced to identify missing checks. For example, these tools might confirm that every system transaction includes some authorization check, which mitigates vulnerabilities that

allow attackers to reference data that they should not be allowed to access. These examples represent only a handful of the capabilities possessed by code analysis tools.

Proactively identifying known vulnerabilities is useful, but some vulnerabilities are too subtle to deterministically detect. As a result, another mitigation technique in use is fuzzing. This practice sends random data to running applications to detect unexpected errors. These errors, when exposed, are often the sort of weaknesses that attackers use to gain a foothold in the system. Fuzzing can be executed as part of a functional testing suite early in the build pipeline, or even continuously against production infrastructure.

There are entire books written on secure coding practices, some of which are dependent on the type of application being created. Programmers should familiarize themselves with the appropriate practices to improve the security of their applications.

Many organizations choose to have security consultants inspect their applications and development practices to identify problems.

Isolation

Isolating deployed applications by constraining the set of resources they can access is important in a zero trust network. Applications have traditionally been executed inside a shared environment, where a user's applications are running in an execution environment with very few constraints on how those applications can interact. This shared environment creates a large amount of risk should an application be compromised, and presents challenges similar to the perimeter model.

Application isolation seeks to constrain the damage of a potentially compromised application by clearly defining the resources that are available to the application. Isolation will constrain capabilities and resources that the operating system provides:

- CPU time
- Memory access
- Network access
- Filesystem access
- System calls

When implemented at its best, every application is given the least amount of access necessary to complete its work. A well-constrained application that becomes compromised will quickly find that no additional leverage in the larger system is gained. As a result, by isolating applications, the potential damage from a compromised application is greatly reduced. In a multiprocess environment (e.g., a server running several services), other still-safe services are protected from attempts to move laterally on that system.

Application isolation can be accomplished using a number of different technologies:

- SELinux, AppArmor FreeBSD jails
- Virtualization/containerization
- Apple's App Sandbox
- Windows Isolated Applications
- Docker
- Kubernetes
- Firejail
- Google's gVisor

Isolation is generally seen as breaking down into two types: virtualization and shared kernel environments. Virtualization is often considered more secure, since the application is contained inside a virtual hardware environment, which is serviced by a hypervisor outside the virtual machine's (VM) execution environment. Having a clear boundary between the hypervisor and the VM creates the smallest surface area of the two.

Shared kernel environments, like those used in containerized or application policy systems, provide some isolation guarantees, but not to the same degree as a fully virtualized system. A shared kernel execution environment uses fewer resources to run the same set of applications, and is therefore gaining favor in cost-conscious organizations.

As virtualization tries to address the resource efficiency problem by providing more direct access to the underlying hardware, the security benefits of the virtualized environment begin to look more like the shared kernel environment. Depending on your threat model, you may choose to not share hardware at all.

Active Monitoring

As with any production system, careful monitoring and logging is of the utmost importance, and is particularly critical in the context of security. Traditional security models focus their attention on external attack vectors. Zero trust networks encourage the same level of rigor for internal activity. Early detection of an attack could be the difference between complete compromise and prevention altogether.

Apart from the general logging of security events throughout the infrastructure, such as failed or successful logins, which is considered passive monitoring, there exists an entire class of active monitoring as well. For instance, the fuzzing scans we previously discussed can take time to turn up new vulnerabilities—perhaps more time than

you're willing to spend early on in the release pipeline. An active monitoring strategy advocates that the scans also be run continuously against production.

Don't Do That in Production!

Occasionally, the desire to take certain actions in production can be met with resistance for fear of impacting the availability or stability of the overall system. Security scans frequently fall into this bucket. In reality, if a security scan can destabilize your system, then there is a greater underlying problem, which might even be a vulnerability in and of itself. Rather than avoiding potentially dangerous scans in production, ask why they might be risky, and work to ensure that they can be run safely by resolving any system deficiencies contributing to the concern.

Of course, fuzzing is just one example. Automated scanning can be a useful tool for ensuring consistent behavior in a system. For example, a database of anticipated listening services could be compared against an automated scan of actual listening services so deviations can be addressed. Not all scanning will result in such clear action, however. Scanning of installed software, for example, will typically be used to drive prioritization of upgrades based on the threats a network is exposed to or expects to see.

Effective system scanning requires multiple types of scanners, each of which inspects the system in a slightly different manner:

- Fuzzing (i.e., afl-fuzz)
- Injection scanning (i.e., sqlmap)
- Network port scanning (i.e., nmap)
- Common vulnerability scanning (i.e., nessus)

So, what to do when all this monitoring actually discovers something? The answer typically depends on the strength of the signal. Traditionally, suspicious (but not critical) events get dumped into reports and periodically reviewed. This practice is by far the least effective, as it can lead to report fatigue, with reports going unnoticed for weeks at a time. Alternatively, important events can page a human for active investigation. These events have a strong enough signal to warrant waking someone up. In most cases, this is the strongest line of defense.

Applications Monitoring Applications

Application security monitoring is the idea that applications participating in a single cluster or service can actively monitor the health of their peers, and gain consensus with others on their sanity. This might manifest itself as TPM quotes, behavioral analysis, and everything in between. By allowing applications to monitor each other, you gain a high signal-to-noise ratio while at the same time distributing the responsibility throughout the infrastructure. This approach most effectively guards against side-channel attacks, or attacks enabled through multitenancy, since these vectors are less likely to be shared across the entire cluster.

In highly automated environments, however, a third option opens up: active response. Strong signals that "something is wrong" can trigger automated actions in the infrastructure. This could mean revoking keys belonging to the suspicious instance, booting it out of cluster membership, or even signaling to datacenter management software that the instance should be moved offline and isolated for forensics.

Of course, as with any high-level automation, one can do a lot of damage very quickly when utilizing active responses. It is possible to introduce denial-of-service attacks with such mechanisms, or perhaps more likely, shut down a service as a result of operator error. When designing active response systems, it is important to put a number of fail-safes in place. For instance, an active response that ejects a host from a cluster should not fire if the cluster size is dangerously low. Being thoughtful about building active response limitations such as this goes a long way in ensuring the sanity of the active response process itself.

Secure Software Development Lifecycle (SDLC)

Integrating security best practices and tools throughout the entire software development lifecycle is critical in a zero trust environment. By doing so, organizations can identify and remediate vulnerabilities before they reach production environments, reducing the risk of security breaches and promoting a more secure application landscape. Let's cover the key aspects of a secure SDLC next.

Requirements and Design

During the initial stages of software development, security requirements should be defined, and potential risks should be assessed. This involves considering the application's architecture, data flows, and potential attack vectors. Security measures like threat modeling can help identify and address potential vulnerabilities at the design stage. Incorporating privacy by design principles also protects sensitive data throughout the application's lifecycle.

Coding and Implementation

Developers should adhere to secure coding practices, such as input validation, output encoding, and proper error handling, to minimize the likelihood of introducing vulnerabilities in the code. Organizations should establish coding standards and guidelines to ensure consistency across the development team. Using secure programming languages and frameworks with built-in security features can reduce the risk of introducing vulnerabilities.

Static and Dynamic Code Analysis

Implementing static application security testing (SAST) and dynamic application security testing (DAST) tools can help identify vulnerabilities and coding flaws during development. SAST analyzes source code for potential security issues, while DAST tests running applications for vulnerabilities that may only be apparent during execution. Integrating these tools with continuous integration and continuous deployment (CI/CD) pipelines allows for automated and ongoing security assessments.

Peer Reviews and Code Audits

Regular peer reviews and code audits can help identify security issues that automated tools may have missed. This process encourages knowledge sharing among the development team, fostering a security-conscious mindset. In addition to internal reviews, external audits conducted by independent security experts can provide an unbiased assessment of the application's security posture.

Quality Assurance and Testing

Security testing should be an integral part of the quality assurance process, with test cases designed to evaluate the application's resilience to various attack scenarios. Penetration testing, performed by internal or external security experts, can provide further insight into the application's security posture. Automated security tests should

be integrated with the CI/CD pipeline to ensure continuous application security validation.

Deployment and Maintenance

Once the application is deployed, keeping it up to date with the latest security patches and updates is essential. Monitoring and logging mechanisms should be implemented to detect potential security incidents, allowing for timely response and mitigation. In addition, configuration management tools and processes should be in place to ensure consistent and secure deployment of applications and updates.

Continuous Improvement

Organizations should learn from security incidents and incorporate the lessons learned into their SDLC processes. Regularly reviewing and updating security policies, procedures, and practices ensures that the organization remains agile and responsive to emerging threats. Conducting post-mortem analyses of security incidents and sharing the findings with the development team can lead to better security awareness and improved practices.

By incorporating these practices, organizations can foster a security-centric culture and promote trust in their applications.

Protecting Application and Data Privacy

You have learned in previous sections the importance of securing various stages of the software development lifecycle in the context of zero trust networks; this section briefly examines the role of confidential computing. It is a technology that guarantees both code/data integrity and confidentiality while allowing applications to be deployed and operate, particularly in a public cloud, as well as in any other hostile environment (think client devices such as mobile phones).

When You Host Applications in a Public Cloud, How Can You Trust It?

Isn't it convenient to lift-and-shift existing applications to the public cloud? Of course, and if you are reading this, it is likely that you are already using the public cloud in some shape and form, as it provides comprehensive DevSecOps, monitoring, scalability, and resiliency. But there is a caveat—can you trust the cloud operator hosting the application not to tamper with the code and data? Well, in the context of zero trust, cloud operators cannot be implicitly trusted. Why? There can be a variety of reasons, ranging from regulatory (e.g., government-issued subpoenas) to malicious (e.g., disgruntled employees, espionage), resulting in data as well as code getting into the hands of unauthorized resources. While data is typically protected at rest (using disk-based encryption such as FIPS) and in transit (using TLS) when security

best practices are followed, this still leaves data as well as code accessible to cloud operators while computation is performed on it (e.g., while in memory). To mitigate this risk, you need technical assurance backed by cryptographic proof from the cloud operator that the code and data cannot be accessed by an unauthorized party (including the cloud operator) while in use, as well as proof that code operating on the data is never tampered with. This is an important step toward building confidence while deploying and running applications on a modern cloud platform—never trust, always verify.

Confidential computing offers this assurance.

Confidential Computing

The term confidential computing is defined by the Confidential Computing Consortium (CCC) as "*the protection of data in-use by performing computation in a hardware-based, attested Trusted Execution Environment (TEE).*" A TEE is a secure computing environment that consists of software and hardware component(s) and ensures the execution of only authorized applications. Data stored within the TEE is impervious to unauthorized access or manipulation by external code. The objective of the confidential computing threat model is to mitigate or minimize the potential for cloud provider operators and other entities inside the environment to gain unauthorized access to code and data during execution.

What Makes Confidential Computing Different from Others?

Please note that there are several other privacy-enhancing technologies with varying degrees of maturity for protecting data while it is in use during computation, such as homomorphic encryption (HE), secure multiparty computation (SMPC), and zero-knowledge proof (ZKP), etc. The distinguishing characteristic of confidential computing is its ability to verify that the code operating on the data has not been tampered with in any way and is precisely as expected. Chapter 12 also covers privacy-enhancing technologies and provides additional references for interested readers to examine.

Understanding Hardware-Based Root-of-Trust (RoT)

Hardware-based root-of-trust (RoT) is an immutable hardware component, often a silicon chip or a cluster of chips, that is designed to be exceptionally tamper-resistant from a wide range of attacks. With RoT, the cryptographic keys are embedded in the hardware and are inseparable from it; no one can access them, not even the cloud provider.

The RoT is used within a TEE to enable hardware isolation, memory protection, encryption, secure storage, and attestation capabilities. This is essentially the lowest

(closest to silicon) level of protection available for protecting applications, including code and data.

Role of Attestation

Attestation is a vital step in building confidence in hardware and software components. First, attestation ensures the integrity and dependability of the TEE hardware, verifying that the TEE is based on the expected hardware (manufacturer, version, firmware, etc.) and that the memory protection functions associated with that hardware are enabled. Second, the software running within the TEE is authentic (version, runtime properties, etc.) and has not been tampered with.

Attestation in the Public Cloud?

Most large-scale hardware chip manufacturers provide attestation as a built-in feature that cloud operators enable so that organizations may use it to achieve a high level of confidence while running applications in the cloud. For further information, refer to: Intel® Software Guard Extensions remote attestation, AMD SEV-SNP Attestation, NVIDIA Hopper Confidential Computing Attestation Verifier, AWS cryptographic attestation, Microsoft Azure Attestation, Google attestation policies, and IBM Secure Execution attestation.

One thing to keep in mind is that, despite the fact that confidential computing provides strong protection, it may still have threats (as does any other technology) that must be evaluated and go through threat modeling. Readers are encouraged to examine the confidential computing threat model outlined by the Confidential Computing Consortium for more details on this topic.

Scenario Walkthrough

Let's run through another scenario walkthrough. The key components are shown in Figures 7-7 and 7-8 while request analysis is performed next.

Use Case: Bob Sends Highly Sensitive Data to Financial Application for Computation

Here is what we know about Bob's request:

- Bob is sending highly sensitive data for computation to the financial application FinApp.
- Bob has encrypted the sensitive data using a public key provided by FinApp.

- Bob is using his work laptop with device ID "ABC," which is fully compliant with organization policy.
- Bob has used a password for authentication along with SMS as an MFA method.
- Bob is making the request during office hours.

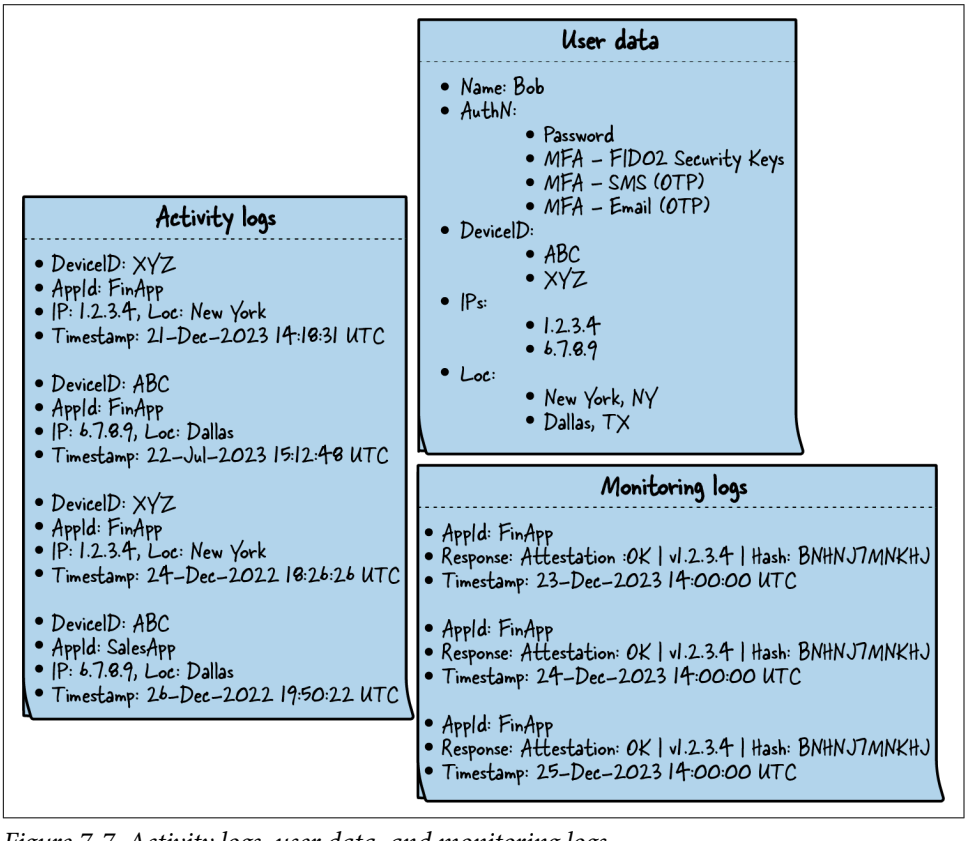

Figure 7-7. Activity logs, user data, and monitoring logs

Here's what we know about FinApp:

- FinApp runs in a trusted execution environment/secure hardware enclave.
- All communications to FinApp are through an encrypted TLS channel.
- Continuous monitoring runs attestation checks on a regular basis to ensure FinApp is running the expected version and has not been tampered with.

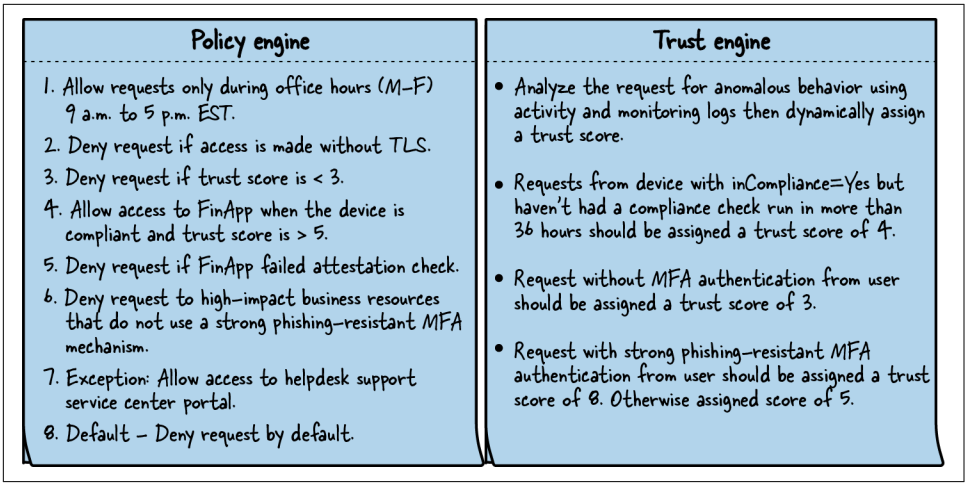

Figure 7-8. Policy engine and trust engine

Request Analysis

1. Bob's access request (action: compute-score, app-id: FinApp, device-id: ABC, authentication: FIDO2, location: Dallas, IP:6.7.8.9, datetime: 23-Dec-2023-4:00pm-est-timezone) reaches the enforcement component.
2. The enforcement component forwards the access request to the policy engine for approval.
3. The policy engine receives the request and consults with the trust engine to determine the request's trust score.
4. The trust engine evaluates the request:
   - No anomalous behavior is detected based on the user activity logs, and monitoring logs also show the application properly responding to attestation requests.
   - The device is in compliance and had its most recent compliance check less than 36 hours ago, so add 4 points to the trust score.
   - Bob has also used FIDO2 as an MFA method, which is phishing resistant, so add another 8 points to the trust score.
   - Finally, the trust engine computes the average of trust scores, which is 6, and returns it to the policy engine.
5. The policy engine receives the trust score of 6 from the trust engine.

6. For authorization, the policy engine compares the request to all policy rules:
   - This first rule results in a grant (or allow) action because the request is made during the permissible office hours.
   - The second rule does not apply because the access request is made over TLS.
   - The third rule does not not apply as the trust score received from the trust engine was 6, which is higher than 3.
   - The fourth rule does apply to the current request as the request's trust score is greater than 5. The request is set to grant.
   - The fifth rule does not apply because the FinApp request does not fail any attestation checks, as monitoring logs show.
   - The sixth rule does not apply since the request is using a strong, phishing-resistant MFA mechanism.
   - The seventh rule does not apply since the request is not for the help desk support service center portal.
   - The eighth rule does not apply since it is a default and only applicable if any other prior rule is not applied. In this case, the fourth rule was applied to allow the request.
   - The policy engine sends an *allow* action to the enforcement component.
7. The enforcement component receives the result from the policy engine and grants Bob access to the FinApp.

Summary

This chapter dove into how applications in a zero trust network are secured. It might seem counterintuitive that a zero trust network needs to be concerned with application security. After all, the network is untrusted, so untrustworthy applications existing on the network should be expected. However, while the network works to detect and identify malicious application activity, that goal is made impossible if deployed applications are not properly vetted before being authorized to run. As a result, most of this chapter focused on how to securely develop, build, and deploy applications in a zero trust network, and then monitor the running instances to ensure that they stay trustworthy.

The chapter introduced the concept of a trusted application pipeline, which is the mechanism by which software written by trusted developers is transformed into built applications that are then deployed into infrastructure. This pipeline is a highly valuable target for would-be attackers, and so it deserves special attention. We dug into secure source code-hosting practices, sound practices for turning source code into trusted artifacts, and securely selecting and distributing those artifacts to downstream consumers. The application pipeline can be visualized as a series of immutable

transformations of inputs from earlier in the pipeline, so we explored how to meet the goals of that pipeline without introducing too much friction in the process.

Human attention is a scarce but important resource in a secure system. With the rate of software releases ever increasing, it's important to mindfully consider when humans are best introduced in the process. We discussed where to put humans in the loop to ensure that the pipeline remains secure.

Once applications are built, the process of securing their continued execution in a production environment shifts a bit. Old, trusted applications may in the future become untrusted as vulnerabilities are discovered, so we discussed the importance of an upgrade-only policy when running applications. Secrets management is often a difficult task for security engineers, where changing credentials is often very burdensome.

With a smooth credential provisioning process, however, a new opportunity emerges to frequently rotate credentials, using the credentialing process itself as a mechanism for ensuring only authorized applications continue to run in a production environment.

We ended the chapter with a section discussing good application security hygiene. Learning secure coding practices, deploying applications in isolated environments, and then monitoring them aggressively is the final aspect of a trustworthy production environment.

Finally, we ended the chapter with a discussion about secure software development.

With all the components of a zero trust network explored, the next chapter focuses on how network communication itself is secured.

CHAPTER 8

Trusting the Traffic

Authenticating and authorizing network flows is a critical aspect of a zero trust network. In this chapter, we're going to discuss how encryption fits into the picture, how to bootstrap flow trust by way of secure introduction, and where in your network these security protocols best fit.

Zero trust is not a complete departure from everything we know. Traditional network filtering still plays a significant role in zero trust networks, though its application is nontraditional. We'll explore the role filtering plays in these networks toward the end of this chapter.

Encryption Versus Authentication

Encryption and authenticity often go hand in hand, yet serve distinctly separate purposes. Encryption ensures confidentiality—the promise that only the receiver can read the data you send. Authentication enables a receiver to validate that the message was sent by the thing it is claiming to be.

Authentication comes with another interesting property. To ensure that a message is in fact authentic, you must be able to validate the sender and that the message is unaltered. Referred to as integrity, this is an essential property of message authentication.

Encryption is possible without authentication, though this is considered a poor security practice. Without validation of the sender, an attacker is free to forge messages, possibly replaying previous "good" messages. An attacker could change the ciphertext, and the receiver would have no way of knowing. A number of vectors are opened by the omission of authentication, so the recommendation is pretty much the same across the board: use it.

Additionally, it is important to consider the following aspects when discussing encryption and authentication in the context of a zero trust network:

Secure key management
: In modern encryption practices, secure key management plays a crucial role. It involves the secure generation, storage, and distribution of encryption keys. Techniques such as the use of hardware security modules (HSMs) or key management services (KMSs) ensure the protection of encryption keys from unauthorized access or compromise.

Forward secrecy
: Forward secrecy is a critical property of encryption protocols, such as TLS. It ensures that the compromise of a single encryption key does not compromise the confidentiality of past or future communications. Forward secrecy relies on using ephemeral keys discarded after a single session, making it harder for attackers to decrypt previously recorded encrypted traffic.

Multifactor authentication (MFA)
: In the context of trusting traffic in a zero trust network, incorporating multifactor authentication adds a layer of security, since MFA requires users to provide multiple forms of authentication, such as a password, a fingerprint scan, or a security token, before gaining access to resources or transmitting data. Implementing MFA at the authentication level strengthens the trust in the network.

Post-quantum cryptography
: With the rise of quantum computers, traditional cryptographic algorithms currently considered secure may become vulnerable to attacks. Post-quantum cryptography focuses on developing encryption algorithms that can withstand attacks from quantum computers. Research and standardization efforts are underway to identify and deploy post-quantum cryptographic algorithms that can replace or augment existing algorithms to ensure long-term security in the face of quantum computing advancements.

Authenticity Without Encryption?

Message authenticity is a stated requirement of a zero trust network, and it is not possible to build one without it. But what about encryption?

Encryption brings confidentiality, but it can also be an occasional nuisance. Troubleshooting becomes harder when you can't read packet captures without complicated decryption processes. Intrusion detection becomes difficult to impossible if the network traffic can't be inspected. There are, in fact, some legitimate reasons to avoid encryption.

That said, be absolutely certain that you do not care about data confidentiality if you choose to not use encryption. While keeping data unencrypted is convenient for administrators, it is never legitimate if the data actually requires confidentiality. For instance, consider the scenario shown in Figure 8-1.

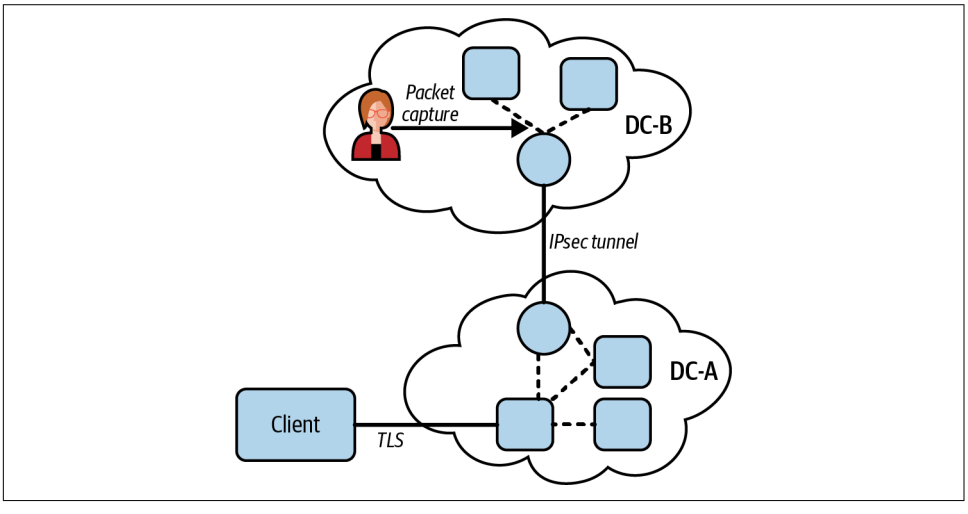

Figure 8-1. Confidentiality within the datacenter is just as important as outside the datacenter

This is an exceedingly common architecture. Note that it only encrypts traffic in certain areas, leaving the rest open (perhaps for the benefit of system administrators). Clearly, however, this data requires confidentiality, as it is encrypted in transit between sites.

This is a direct contradiction of the zero trust architecture, as it creates privileged zones in the network. Thus, citing good reasons to not encrypt traffic is a very slippery slope. In practice, systems that truly do not require confidentiality are rare.

In addition to all of this, authentication is still required. There are few network protocols that provide strong authentication but not encryption, and all of the transport protocols we discuss in this book provide authentication as well as encryption. If you look at it this way, encryption is attained "for free," leaving few good reasons to exclude it.

Bootstrapping Trust: The First Packet

The first packet in a flow is oftentimes an onerous one. Depending on the type of connection, or point in the device lifecycle, this packet can carry with it very little trust.

We generally know what flows to expect inside the datacenter, but in client-facing systems, it's anyone's guess. These systems must be widely reachable, which greatly increases risk. We can use protocols like mutually authenticated TLS to authenticate the device before it is allowed to access the service; however, the attack surface in this scenario is still considerable, and the resources are also publicly discoverable.

So how do you allow only trusted connections, silently dropping all others, without answering a single unauthenticated packet? This is known as the first-packet problem, and it is mitigated through a method called pre-authentication (Figure 8-2).

Pre-authentication can be thought of as the authorizing of an authentication request by setting an expectation for it. It is often accomplished by encrypting and/or signing a small piece of data and sending it to the resource as a UDP (User Datagram Protocol) packet. The use of UDP for pre-authentication is important because UDP packets do not receive a response by default. This property allows us to "hide," exposing ourselves only once we passively receive a packet encrypted with the right key.

Upon the passive receipt of a properly encrypted pre-authentication packet, we know we can expect the sender to begin authentication with us, and we can poke granular firewall holes allowing only the sender the ability to speak with our TLS server. This mode of pre-authentication operation is also known as single-packet authorization (SPA).

SPA is not a fully suited device authentication protocol. It merely helps to mitigate the first-packet problem. Without downplaying the importance of the properties we gain by using pre-authentication, it must not be substituted for a more robust mutually authenticating protocol like TLS or IKE (Internet Key Exchange).

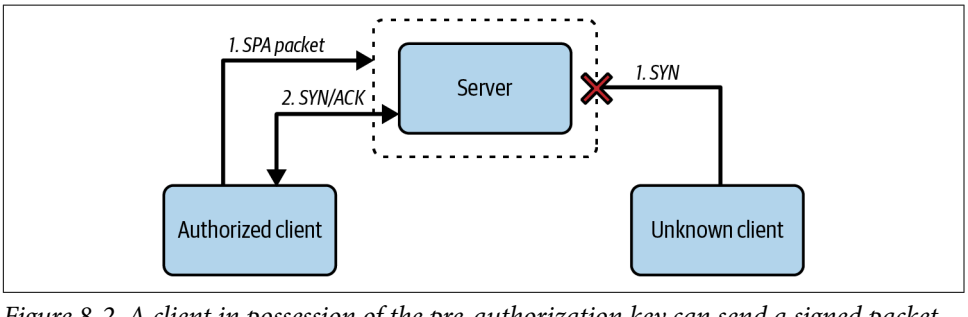

Figure 8-2. A client in possession of the pre-authorization key can send a signed packet in order to set an expectation for a TCP connection. Without it, no acknowledgments are sent.

FireWall KNock OPerator (fwknop)

fwknop is an open source tool that stands for "FireWall KNock OPerator" and uses SPA for authorization. It is compatible with multiple operating systems and directly integrates with host firewalls to create temporary exceptions tightly scoped to specific needs.

Short-Lived Exceptions

When fwknop receives a valid SPA packet, its contents are decrypted and inspected. The decrypted payload includes the protocol and port numbers that the sender is requesting access to. fwknop uses this to create firewall rules permitting traffic from the sender to those particular ports—rules that are removed after a configurable period of time. The default value is 30 seconds, but in practice, you may only need a few seconds.

As mentioned, the rule that fwknop creates is tightly scoped. It permits only the sender's IP address and only the destination ports requested by the sender. The destination ports that may be requested can be restricted via policy on a user-by-user basis. Additionally, it is possible for the sender to specify a source port, restricting the scope of the rule even further.

SPA Payload

The fwknop SPA implementation has seven mandatory fields and three optional fields included in its payload. Among these are a username, the access request itself (which port, etc.), a timestamp, and a checksum:

- 16 bytes of random data
- Local username
- Local timestamp
- fwknop version
- SPA message type
- Access request
- SPA message digest (SHA-256 by default)

Once the client has generated the payload, it is encrypted, an optional HMAC (hashed message authentication code) is added, and the SPA packet is formed and transmitted.

Payload Encryption

Two modes of encryption are supported: AES (Advanced Encryption Standard) and GnuPG (Gnu Privacy Guard). The former being symmetric and the latter being asymmetric, two options are provided in order to cater to multiple use cases and preferences.

Personal applications or small installations might prefer AES since it does not require any GnuPG tooling. AES is also more performant with regard to data volume and computational overhead. It does have some downsides, though, practically all of which originate from the fact that it is a symmetric algorithm.

Symmetric encryption comes with difficult key distribution problems, and beyond a certain scale, these challenges can grow to be untenable. Leveraging the GnuPG encryption mode solves most of these problems and is the recommended mode of operation, despite it being less performant than its counterpart.

HMAC

fwknop can be configured to add an HMAC to the end of its payload. The HMAC prevents tampering by guaranteeing that the message is authentic. This is important because otherwise an attacker could arbitrarily modify the ciphertext, and the receiver would be forced to process it.

You may have noticed that there is a message digest that is calculated and stored along with the plain text. This digest helps to mitigate attacks in which the ciphertext is modified, but it is also less than ideal, as this method (known as authenticate-then-encrypt, or AtE) is vulnerable to a few niche classes of attacks. Adding an HMAC to the encrypted payload prevents these attacks from being effective.

In addition, decryption routines are generally much more complex than HMAC routines, meaning they are more likely to suffer from a vulnerability. Applying an HMAC to the ciphertext allows the receiver to perform a lightweight integrity check, helping to ensure that we are only sending trusted data to the decryption routines. It is strongly recommended to configure fwknop to use HMAC.

For more information about networking protocols, please refer to the Appendix.

Where Should Zero Trust Be in the Network Model?

With a better understanding of network layer models, we can now take a look at where to best apply zero trust controls in the network stack.

There are two predominant network security suites: TLS and IPsec. TLS (to which SSL is a predecessor) is the most common of the two. Many application layer

protocols support TLS to secure traffic. IPsec, or Internet Protocol Security, is an alternative protocol, more commonly used to secure things like VPNs.

Despite having "transport" in its name, TLS does not live in the transport layer of the TCP/IP (internet protocol suite) model. It is found in the application layer (somewhere between layer 5 and 6 in the OSI, or Open Systems Interconnection, model), and as such is largely an application concern.

TLS as an Infrastructure Concern

Perimeter networks frequently abstract TLS away from applications, shifting the responsibility from the application to the infrastructure. In this mode, TLS is "terminated" by a dedicated device at the perimeter, forwarding the decrypted traffic to a backend service. While this mode of operation is not possible in a zero trust network, there remain a handful of strategies for deploying TLS as an infrastructure concern while still conforming to the zero trust model. More on that later.

IPsec, by contrast, is generally considered part of the internet layer in the TCP/IP model (layer 3 or 4 in the OSI model, depending on interpretation). Being further down the stack, IPsec is usually implemented in a host's kernel. IPsec was developed for the IPv6 specification. It was originally a requirement for IPv6, but was eventually downgraded to a recommended status.

With two alternatives to secure network transit, the question becomes, is one preferred over the other? Zero trust's goal is secure communication for all traffic. The best way to accomplish this goal is to build systems that provide secure communication by default. IPsec, being a low-level service, is well positioned to provide this service.

Using IPsec, host-to-host communication can be definitively secured. Being integrated deep in the network stack, IPsec can be configured to only allow packet transmission once a secure communication channel has been established. Furthermore, the receiving side can be configured to only process packets that have been sent securely. In this system, we have essentially created a "secure virtual wire" between two hosts over which only secured traffic can flow. This is a huge benefit over traditional security initiatives that add secure communication one application at a time. Simply securing communications between two devices is not sufficient to build a zero trust network. We need to ensure that each individual network flow is authorized. There are several options for meeting this need:

- IPsec can use a unique security association (SA) per application (see RFC 4301, section 4.4.1.1). Only authorized flows are then allowed to construct these security policies.

- Filtering systems (software firewalls) can be layered on top of IPsec. We will discuss the role of filtering in zero trust later in this chapter.
- Application-level authorization should be used to ensure that communications are authorized. This could involve standard authorization techniques, such as access tokens or X.509 certificates, while delegating strong encryption and authentication responsibilities to the IPsec stack.
- For a truly "belt and suspenders" system, mutually authenticated TLS could be layered on top of the existing IPsec layer. This defense-in-depth approach provides two layers of encryption (mTLS, or mutual TLS, and IPsec), protecting communication should one of them become compromised, at the expense of complexity and increased overhead.

Client and Server Split

While IPsec has a number of beneficial properties, its lack of popularity presents real-world obstacles for its use in systems today. The issues one will see can be broken down into three areas:

- Network support
- Device support
- Application support

Network Support Issues

Network support can hamper the use of IPsec in the wild. IPsec introduces several new protocols, two of which (ESP, or Encapsulating Security Payload and AH, or Authentication Header) are new IP protocols. While these protocols are fully supported in simple LANs (local area networks), on some networks, getting these packets transmitted can be quite a challenge. This could be due to misconfigured firewalls, NAT (network address translation) traversal, or routers being purposefully configured to not allow traffic to flow. For example, Amazon Web Services (AWS), a large public cloud provider, does not allow ESP or AH traffic to be transmitted on its networks. Public hotspots like those found at businesses or libraries also often have spotty support for IPsec traffic. To mitigate these issues, IPsec includes support for encapsulating traffic in a UDP frame (depicted in Figure 8-3). This encapsulation allows an inhospitable network to transmit the traffic, but it adds extra complexity to the system.

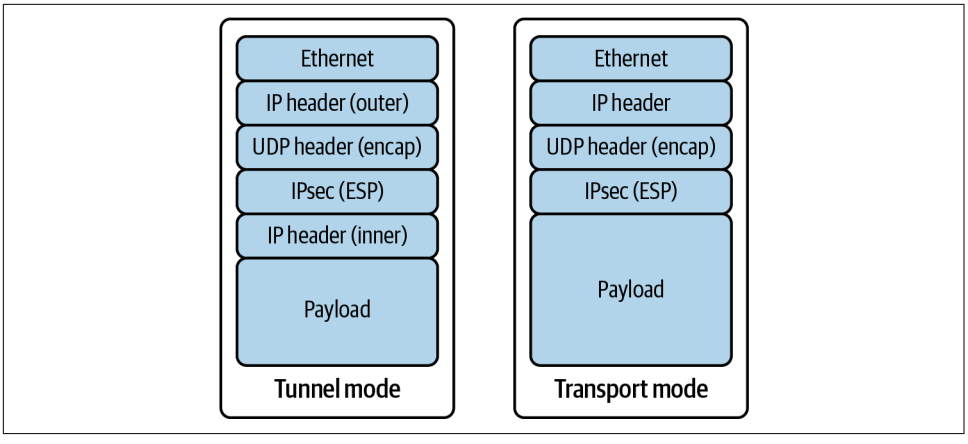

Figure 8-3. IPsec supports encapsulating ESP packets in a UDP packet, making it look like normal UDP traffic

Device Support Issues

Device support can also be a major factor in rolling out an IPsec-protected network. The IPsec standard is complex, with many configuration options and cipher suites. Both hosts in the relationship need to agree to a common protocol and cipher suite before communication can flow. Cipher suites in particular frequently need to be adjusted as compromises are revealed. Finding that a stronger cipher suite has not been implemented is a real issue in IPsec systems. To be fair, TLS needs to handle these same issues, but due to the nature of having IPsec implemented in the system's kernel, progress on newer protocols and cipher suites is naturally slower.

IPsec also requires active configuration of the devices in the relationship. In a client/server system with varying device capabilities, configuring the client devices can be rather challenging. Desktop operating systems can usually be configured to support the less popular protocol. Mobile operating systems, however, are less likely to fully support IPsec in a way that conforms to the zero trust model.

Application Support Issues

IPsec places additional requirements on the system configuration versus typical TLS-based security. A system wanting to make use of IPsec needs to configure IPsec policy, enable kernel support for the desired cipher suites, and run an IKE daemon to facilitate the negotiation of IPsec security associations. When compared to a library-based approach for TLS, this extra complexity can be daunting. This is doubly so when many applications already come with built-in TLS support, which seemingly offers a turnkey solution for network security.

It should be noted that while the library approach seems more attractive at first glance, in practice it presents quite a bit of hidden complexity. Applications frequently support the more common server TLS, but neglect to expose configuration for presenting a client certificate that is required to create a mutually authenticated TLS connection. Additionally, system administrators may need to adjust configuration in reaction to recently exposed vulnerability. With a large set of applications, finding the application-specific configuration that needs to be adjusted can hamper the rollout of a critical fix.

The web browser is frequently the common access point into organizational systems. Its support for modern TLS is generally very good (assuming organizations stay up to date on the latest browser versions). This common access point mitigates the issue of configuration, as there is a small set of target applications that need to be adjusted. On the server side, many organizations are turning toward a model where network communication is secured via a local daemon. This approach centralizes configuration in a single application and allows for a base layer of network security to be supplied by the system administrator. In a way, it looks very similar to the IPsec model, but implemented using TLS instead.

Given all the pluses and minuses of the two approaches, a pragmatic solution seems available to system administrators.

A Pragmatic Approach

For client/server interactions, mutually authenticated TLS seems to be the most reasonable approach to network security. This approach would typically involve configuring a browser to present client certificates to server-side access proxies, which will ensure that the connection is authenticated and authorized. Of course, this restricts the use of zero trust to browser-based applications.

For server/server interactions, IPsec seems more approachable. The server fleet is generally under more controlled configuration, and the network environment is more well-known. For networks that don't support IPsec, UDP encapsulation can be used to avoid network transit issues.

Microsoft Server Isolation

For environments that fully employ Microsoft Windows with Active Directory, a feature called server isolation is particularly attractive. By leveraging Windows Firewall, Network Policy, and Group Policy, server isolation provides a framework through which IPsec configuration can be automated. Furthermore, server isolation can be tied to Active Directory security groups, providing fine-grained access control which is backed by strong IPsec authentication. While complications surrounding IPsec transit over public networks still exist, server isolation is perhaps the most pragmatic approach for obtaining zero trust semantics in a Windows-based environment.

Since the IPv6 standard includes IPsec, the authors hope that it will become a more viable solution for both types of network communication as network adoption progresses.

The Protocols

In this section, we will go over mutually authenticated TLS (mTLS) in more detail, as well as briefly discuss IPsec, as both are crucial protocols.

IKE and IPsec

Internet Key Exchange (IKE) is a protocol that performs the authentication and key exchange components of IPsec. It is typically implemented as a daemon and uses a pre-shared key or an X.509 certificate to authenticate a peer and create a secure session. Inside this secure session, another key exchange is made. The results of this second key exchange are then used to set up an IPsec security association, the parameters of which are leveraged for bulk data transfer. Let's take a closer look.

IKEv1 Versus IKEv2

There are two versions of IKE, and most software suites support both. For all new deployments, it is strongly recommended to use IKEv2. It is both more flexible and more reliable than its predecessor, which was overly complicated and less performant. For the purposes of this book, we will be talking about IKEv2 exclusively.

There is frequent confusion around the relationship between IKE and IPsec. The reality is that IPsec is not a single protocol; it is a collection of protocols. IKE is often considered part of the IPsec protocol suite, though its design makes it feel complimentary, as opposed to a core component. IKE can be thought of as the control plane of IPsec. It handles session negotiation and authentication, using the results of the negotiation to configure the endpoints with session keys and encryption algorithms.

Since the core IPsec protocols are embedded in the IP stack, IPsec implementations are typically found in the kernel. With key exchange being a relatively complex mechanism, IKE is implemented as a user space daemon. The kernel holds state-defining active IPsec security associations, and traffic selectors defining which packets IPsec policy should be applied to. The IKE daemon handles everything else, including the negotiation of the IPsec security association (SA) itself (which is subsequently installed into the kernel for use).

For a more detailed understanding of IKE and IPSec, please review RFC 6071: IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap.

Mutually Authenticated TLS (mTLS)

Commonly referred to by the name of its predecessor, SSL, TLS is the protocol most commonly used to secure web traffic. It is a mature and well-understood protocol, is widely deployed and supported, and is already trusted with some of the most sensitive tasks, like banking transactions. It is the "S" in HTTPS.

When TLS is used to secure web sessions, the client validates that the server certificate is valid, but the server rarely validates the client. In fact, the client rarely presents a certificate at all! The "mutual" prefix for TLS (mTLS) is meant to denote a TLS configuration in which client certificate validation is required (and thus, mutually authenticated). While a lack of client authentication may be acceptable for services that are being published to the general public, it is not acceptable for any other use case. Mutual authentication is a requirement for security protocols conforming to the zero trust model, and TLS is no exception.

The basics of a TLS handshake are fairly straightforward, as shown in Figure 8-4. A client initiates the session with a Client-Hello message sent to the server, which includes a compatibility list for things like cipher suites and compression methods. The server chooses parameters from the compatibility list and replies with a Server-Hello defining the selections it made, followed by the server's X.509 certificate. It also requests the client's certificate at this time.

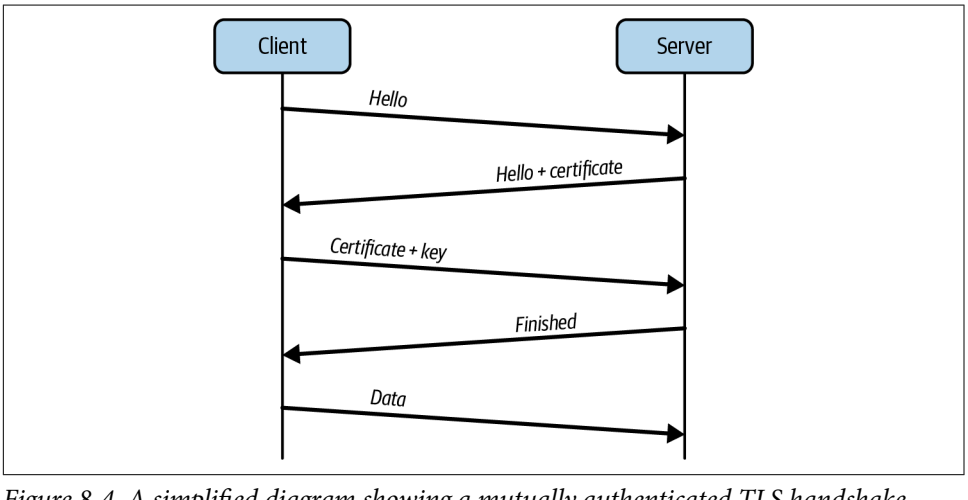

Figure 8-4. A simplified diagram showing a mutually authenticated TLS handshake using RSA key exchange (RSA stands for Rivest-Shamir-Adleman, based on the surnames of those who developed the cryptosystem)

The client then generates a secret key and uses the server's public key to encrypt it. It sends the server this encrypted secret key, as well as its client certificate, and a small bit of proof that it is in fact the owner of that certificate. The secret key generated by

the client is ultimately used to derive several additional keys, including one that acts as a symmetric session key. So, once the client sends these details off, it has enough information to set up its side of the encrypted session. It signals the server that it is switching to session encryption; the server then validates the client and sends a similar message in return, and the session is fully upgraded.

Separation of duty

For the purposes of a zero trust network, it is a good idea to separate the encryption duties from the application itself (Figure 8-5). The resource we are securing in this case is the device, and as such, it makes a lot of sense for this piece to be independent of the workload itself.

Doing this also alleviates a number of pain points, including zero-day mitigation, performance penalties, and auditing. For protocols like IPsec, this separation of duty is part of the design, but this is not the case for TLS. Historically, applications speak TLS directly, loading and configuring shared TLS libraries for remote communication. We have seen this pattern's rough spots time and time again. Shared libraries become littered throughout the infrastructure, being consumed by a multitude of projects, all with independent versions and configurations. Some languages have more flexible libraries than others, limiting your ability to enforce the latest and greatest. Above all, it is very difficult to ensure that all these applications are indeed consuming TLS the right way, and remain up to date with regard to known vulnerabilities.

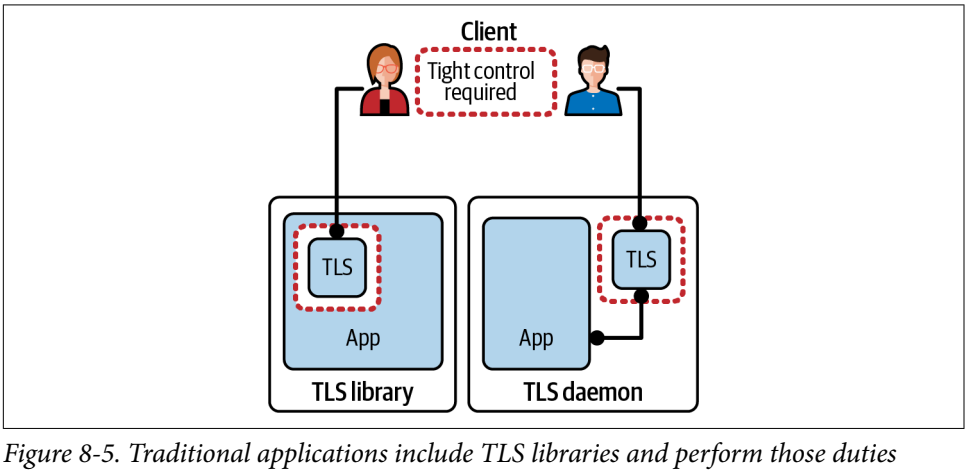

Figure 8-5. Traditional applications include TLS libraries and perform those duties themselves. Using a local TLS daemon instead means better control and consistent performance.

To address the problem, it is useful to move the handling of TLS configuration to the control plane. Connections to the service are brokered by the TLS daemon, then locally forwarded to the application. The TLS daemon is configured with system certificates, trust authorities, and endpoint information—that's about it.

In this way, we can ensure that all software receives device authentication and security with TLS, regardless of its support for it. Additionally, since zero trust networks whitelist flows, we can ensure that application traffic is protected by limiting whitelisted flows to known TLS endpoints.

Bulk encryption

All the TLS intricacies and components discussed up to this point apply primarily to the initial TLS handshake. The TLS handshake serves two primary purposes: authentication and the creation of session keys.

TLS handshakes are computationally expensive due to the mathematical operations required to make and validate them. This is a distinct trade-off between security and performance. While we strongly desire this level of security, the performance impact is prohibitively expensive if we apply these operations to all communications.

Asymmetric cryptography is extraordinarily important in the process of secure introduction and authentication, but its strength can be matched by symmetric cryptography so long as identity or authentication is not a concern. Symmetric encryption uses a single secret key instead of a public/private key pair, and is less computationally expensive than asymmetric cryptography by orders of magnitude. This is where the concept of a TLS handshake and session keys comes in.

Some very smart mathematicians and cryptographers realized that we can use the strong yet expensive operations to securely generate a single secret—one that can be shared between the parties (Figure 8-6). The key exchange component of TLS generates this shared key and ensures that both parties have knowledge of it.

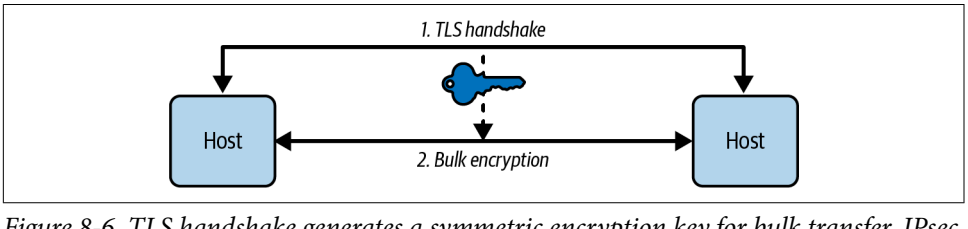

Figure 8-6. TLS handshake generates a symmetric encryption key for bulk transfer. IPsec uses a similar mechanism.

This shared key is then used as the input for a symmetric encryption algorithm, which is applied to all session traffic following the handshake. This methodology ensures that the entire session benefits from the strength of asymmetric cryptography

without inheriting any of the performance implications associated with asymmetric encryption schemes.

When it comes to choices for bulk encryption algorithms, TLS supports many, but the recommendation is pretty well aligned across the board: just use AES. It checks all the desirable boxes, including the fact that it is unpatented, widely implemented in hardware, and practically universally implemented in software. It is very performant, heavily vetted/scrutinized, and remains unbroken to the best of public knowledge. Many people say "AES is good enough," and while that might be a tough pill to swallow when it comes to security protocols, such a statement has never been so close to the truth.

Message authenticity

When communicating securely, message authenticity is an important if not required property. Encryption provides confidentiality, but without message authenticity, how do you ensure the integrity of that message? Without an error during decryption, it is difficult or impossible to distinguish a tampered message from an authentic one. Some encryption modes (such as AES-GCM; GCM stands for Galois/Counter Mode) provide message confidentiality and authenticity guarantees simultaneously. However, these guarantees are only applicable during bulk encryption; there are several TLS exchanges that are not protected by the bulk transfer specifications, and the message authenticity scheme protects those as well.

Explicit Authenticity Sometimes Required

Since some bulk encryption algorithms provide message integrity assurances, it is not always necessary to perform explicit authenticity checks on every packet. Instead, TLS will prefer built-in assurances for bulk transfers and rely on explicit authenticity checks for all packets not associated with the bulk transfer (for instance, TLS control messages).

As far as choice goes, the options are limited to MD5 (message digest 5) and the SHA (Secure Hash Algorithm) family of hashes. The former has been cryptographically broken for quite some time now, leaving the SHA family as the only reasonable choice for ensuring message integrity under TLS. There are even concerns when using the weaker SHA variant, SHA-1, as it is now considered vulnerable in the face of ever-increasing compute power. As such, it is recommended to choose the strongest SHA hash that can be reasonably deployed, given hardware and software constraints.

It is additionally recommended to use bulk encryption with built-in authenticity wherever possible, as it is generally more performant and secure than relying on a

disjointed authenticity mechanism. TLS version 1.3 mandates the use of authenticated encryption.

Mutually authenticated TLS for device authentication

Just like any other protocol used for device authentication, TLS comes with its ups and downs.

The first is that, due to its position in the network stack, TLS is protocol dependent. It is most commonly implemented as a TCP-based protocol, though a UDP-based variant dubbed DTLS (the D stands for Datagram) is also available. The presence of DTLS highlights the deficiency of the position of TLS in the stack. With this, TLS suffers diminishing returns when used to secure IP protocols other than those that it natively supports, like TCP or UDP.

Another thing to consider is the automation requirement. TLS is commonly deployed as an infrastructure service in perimeter networks by leveraging intermediaries, which are typically positioned at the perimeter. This mode of operation, however, is unsuitable for a zero trust network as long as the intermediary and the upstream endpoint are separated by a computer network. In a zero trust network, applications leveraging a TLS-speaking intermediary must be on the same host as the intermediary itself. As a result, protecting datacenter zero trust networks with TLS requires additional automation to configure applications to speak through this layer of external security. It does not come "for free" like other protocols, such as IPsec.

All of that said, it remains today's best choice for protecting client-facing zero trust networks. TLS is very widely supported in both software and transit (i.e., intermediary networks worldwide), and can be relied upon for straightforward and trustworthy operation. Most web browsers support mutually authenticated TLS natively, which means that resources can be protected using zero trust principles without the immediate need for specialized client-side software.

For more detailed technical discussion on mTLS, please review RFC 8120: Mutual Authentication Protocol for HTTP.

Trusting Cloud Traffic: Challenges and Considerations

The cloud has opened up a world of possibilities for organizations, allowing businesses to scale quickly and access unlimited resources. However, this newfound freedom also comes with challenges, particularly around trusting traffic from the cloud environments. There are several challenges when transitioning to cloud systems, which include maintaining compliance and security standards, securing against cyber threats, ensuring network visibility, navigating the variety of cloud providers available, and keeping costs reasonable:

Compliance and security
: One of the primary concerns of trusting cloud traffic is ensuring compliance with regulations and security best practices. Organizations moving their infrastructure to the cloud must ensure their data is secure and not shared with unauthorized parties, which can be a challenge in multitenant public cloud environments, as the data of multiple organizations is often stored on the same physical server. Keeping up to date with the latest regulations and security protocols is crucial to avoid data leakage, compliance violations, and other severe consequences.

Cyber threats
: Cloud environments are vulnerable to a range of cyber threats, including malicious actors targeting the platform itself (e.g., distributed denial-of-service attacks) and threats that target individual users and their data (e.g., phishing scams or ransomware). Additionally, cloud providers have become increasingly attractive targets for state-sponsored actors due to the large amounts of data stored in these environments, making it essential to have robust security measures.

Network visibility
: Traditional on-premises solutions are often limited by physical constraints, whereas cloud networks are often more flexible and expansive. Organizations must know that they will lose some visibility into their cloud-based networks and must be mindful of network monitoring capabilities to ensure sufficient visibility into their systems.

Diversity of cloud providers
: There are a variety of cloud providers out there, each with its unique offerings and security protocols. Understanding the differences among cloud providers is important because it allows organizations to make informed decisions about which ones to trust. Different providers provide different levels of security and services, and understanding these differences is key to making the right decision for an organization's specific needs. Switching between providers can be expensive and time-consuming, so choosing the right provider from the start is important.

Cost
: Cost is often a significant factor as organizations must be willing to invest in the appropriate security controls and monitoring tools to ensure data security. Switching between providers can also be an expensive and time-consuming process. Those opting for a multi-cloud or hybrid cloud solution must be aware of the additional costs of managing multiple providers, considering the complexity of managing a multi-cloud or hybrid environment and the potential for increased latency due to data traveling across different regions and providers.

Despite these challenges, organizations can still take various measures to ensure the trustworthiness of their traffic, both in the cloud and otherwise. These can include, but are not limited to, the following:

- Ensuring that compliance and security standards are being adhered to
- Monitoring cloud-based networks for suspicious activity, maintaining network visibility, and ensuring that data is encrypted at rest and in transit
- Using the latest security protocols, such as mutual TLS (mTLS)
- Using multifactor authentication (MFA) for added security
- Conducting regular vulnerability scans and penetration tests
- The use of intrusion detection/prevention systems (IDS/IPS); training staff on best security practices
- Implementing access control policies
- Regularly patching and updating systems
- Monitoring for unauthorized cloud access
- Implementing a security incident management plan

The zero trust network paradigm is especially useful for determining the trustworthiness of traffic originating from cloud environments. By leveraging mutual authentication, access control, logging, and monitoring of all inbound/outbound traffic, organizations can ensure that only verified endpoints are sending and receiving data across systems. In addition, encryption techniques such as TLS and IPsec can protect data integrity as it traverses external networks, further helping to ensure a secure transport medium.

Cloud Access Security Brokers (CASBs) and Identity Federation

Cloud access security brokers (CASBs) provide an additional layer of security when accessing cloud resources. CASBs are usually deployed as a cloud-based proxy between the organization's network and the provider, providing visibility and monitoring of all traffic transiting to and from the cloud. Some of the capabilities of a cloud access security broker include, but are not limited to, the following:

Data loss prevention (DLP)
: Preventing data leakage via the cloud by detecting and blocking sensitive information

Threat detection and remediation
: Monitoring for malicious activity such as malware, ransomware, phishing attacks, etc., and taking action to prevent the threat from spreading

Encryption and data integrity
: Ensuring that internal data is encrypted in transit and at rest, and that data integrity is maintained

Enforcing authentication policies
: Such as multifactor authentication (MFA) and role-based access control (RBAC), making it harder for attackers to gain unauthorized access to an organization's resources

In addition to the discussed capabilities, CASBs can also apply and maintain network security policies like access control lists (ACLs) and network segmentation. They also offer monitoring and logging functions that enable organizations to detect suspicious or potentially harmful activity.

Identity federation is yet another key component of establishing trust for cloud traffic. Federated identity services like SAML and OAuth are important tools for establishing trust, as they allow organizations to establish single sign-on (SSO) capabilities across multiple cloud applications.

By limiting resources to verified users, the risk of unauthorized access can be significantly reduced.

Filtering

Filtering is the process by which packets are admitted or rejected by systems on a network. When most people think of filtering, they typically envision a firewall, a service or device that sits between the network and application to filter traffic going to or coming from that device. Firewalls do provide filtering, but they can provide other services like network address translation (NAT), traffic shaping, and VPN tunnel services. Filtering can be provided by other systems not traditionally considered, like routers or managed switches. It's important to remember that filtering is a simple service that can be applied at many points in a networked system.

Filtering can be quite frustrating for users without a security mindset since it blocks desired network communication. Wouldn't it be better to get rid of that nuisance and assume the user knows what they want? Unfortunately, well-meaning users can trivially expose services that on further inspection they would rather not expose. During the early days of always-on internet connections, users' computers routinely accidentally exposed file sharing and chat services on the public internet. Filtering provides a type of checks and balances for network communication, forcing users to consider whether a particular connection should really cross a sensitive boundary.

Many of the zero trust concepts so far have focused on advanced encryption and authentication systems. This is because these aspects of network security are not nearly as pervasive in network designs as they should be. However, we should not downplay the importance of network filtering. It is still a critical component of a zero trust architecture, and so we will explore it in three parts:

Host filtering
: Filtering traffic at the host

Bookended filtering
: Filtering traffic by a peer host in the network

Intermediary filtering
: Filtering traffic by devices in between two hosts

Host Filtering

Host filtering deputizes a network endpoint to be an active participant in its own security. The goal is to ensure that every host is configured to filter its own network traffic. This is different from traditional network design, where filtering is delegated to a centralized system away from the host.

Centralized filtering is most often implemented using a hardware firewall. These firewalls make use of application-specific integrated circuits (ASICs) to efficiently process packets flowing through the device. Since the device is often a shared resource for many backend systems, these ASICs are critical for it to accomplish the task of filtering the aggregate traffic of all those systems. Using ASICs brings raw performance at the expense of flexibility.

Software firewalls, like those found in modern operating systems, are much more flexible than their hardware counterparts. They offer a rich set of services like defining policies based on the time of day and arbitrary offset values. Many of these software firewalls can be further extended with new modules to offer additional services. Unlike the early days of the internet, all modern desktop and server operating systems now offer some form of network filtering via a host-based firewall:

Linux
: IPtables

BSD systems
: Berkeley Packet Filter (BPF)

macOS
: Application firewall and additional host firewalls available via the command line

Windows
: Windows Firewall service

Perhaps surprisingly, neither iOS nor Android ships with a host-based firewall. Apple's iOS security guide, *Apple Platform Security,* notes that it considers a firewall unnecessary since the attack surface area is reduced on iOS "by limiting listening ports and removing unnecessary network utilities such as telnet, shells, or a web server." Google does not publish an official security guide. Android, perhaps owing to its ability to run non-Play Store-approved software, does have third-party firewalls available to install if a user chooses to do so.

Zero trust systems assume the network is hostile. As a result, they filter network traffic at every point possible, often using on-host firewalls. Adding an on-host firewall reduces the attack surface of a host by filtering out undesirable network traffic. While software-based firewalls don't have the same throughput capabilities as hardware-based systems, the fact that the filtering is distributed across the system (and therefore sees only a portion of the aggregate traffic) often results in little performance degradation in practice.

Using on-host filtering is simple to get started with. Configuration management systems have very good support for managing on-host firewalls. When writing the logic to install services, it's easiest to capture the allowed connections right alongside their installation and configuration routines. Filtering in a remote system, conversely, is more difficult since the exceptions are separated from the application that needs them.

On-host firewalls also offer opportunities for novel uses of programmable filtering. Single-packet authorization (SPA), which we discussed earlier in this chapter, is a great example of this idea. SPA programmatically manages the on-host firewall to reduce the attack surface of a service on a host. This is advantageous because on occasion, carefully crafted malicious packets can be constructed to exploit a weakness in network services. For example, a service might require authentication and authorization before processing a request, but the authentication logic could have a buffer overflow error which an attacker can use to implement a remote code execution vulnerability. By introducing a filtering layer, we can hide the more complex service interface behind a simpler system that manages firewall rules.

There are, of course, issues when using on-host firewalls exclusively for network filtering. One such issue is the chance for a co-located firewall to be rendered meaningless should a host become compromised. An attacker who is able to gain access to a host and elevate their privilege could remove the on-host firewall or adjust its configuration. Needless to say, this is a big deal, as it removes a layer of defense in the system. This concern is why filtering has traditionally been handled by a separate device, away from potentially risky hosts.

This approach highlights the benefits of isolation in security design, which on-host filtering could benefit from. As the industry moves toward isolation techniques like virtualization and containerization, it becomes clear that these technologies present

an opportunity to further isolate on-host filtering. Without these technologies, the only form of isolation that is available is local user privilege. On a Unix-based system, for example, only the root user is able to make adjustments to the firewall configuration. In a virtualized system, however, one could implement filtering outside the virtual machine, which provides strong guarantees against attacks on the filtering system. In fact, this is how Amazon's security group feature is implemented, as shown in Figure 8-7.

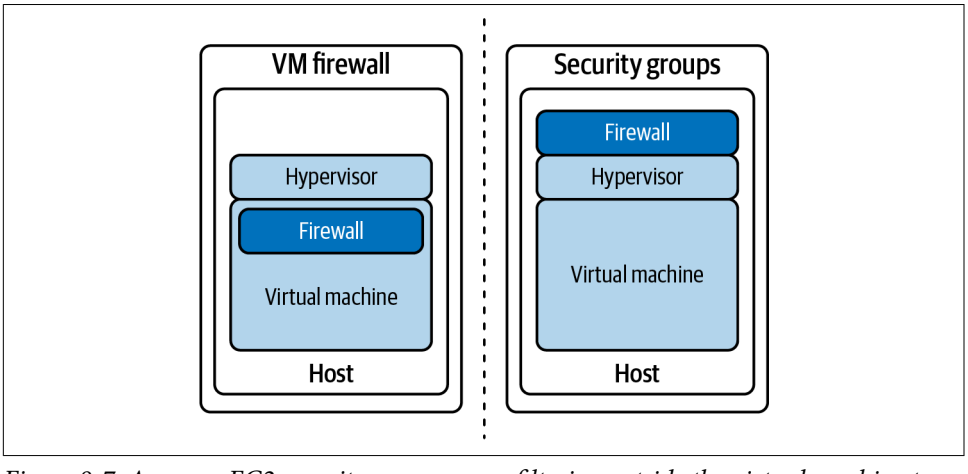

Figure 8-7. Amazon EC2 security groups move filtering outside the virtual machine to improve isolation

Another issue with on-host filtering is the cost associated with pushing filtering deep into the network. Imagine a scenario where a large percentage of traffic is filtered away by on-host filtering. By applying filtering nearest to the destination system, the network incurs extra cost to transmit those packets, only for them to be ultimately thrown away. This situation also raises the possibility of a denial-of-service attack forcing internal network infrastructure to route large volumes of useless traffic, as well as overwhelming the comparatively weaker software firewalls. For this reason, while on-host firewalls are the best place to start thinking about filtering, they present a risk if they are the only place filtering occurs. We will discuss ways to push filtering out into the network in "Intermediary Filtering" on page 192.

Bookended Filtering

Bookended filtering is the act of applying policy not just on the receipt of a packet, but while sending them too. This mode of filtering is not commonly found in traditional networks. It brings some interesting advantages to network design, which we will now explore.

Egress (the opposite of ingress) is a term used to describe network traffic that is leaving a host. This type of filtering is commonly used to manage communication from a private network out to public networks, but it is rarely used within a private network. There are a few reasons this is the case:

- Ingress filtering is easier to reason about, since listening services can be enumerated when building firewall rules. Egress filtering requires more bookkeeping to capture how hosts intend to communicate.
- Ingress filtering is generally considered good enough to stop undesirable communication in the network.
- Egress filtering requires knowledge of every expected flow, something not usually found in traditional networks.

Bookended filtering uses egress filtering within the zero trust network to further harden the system. We can see how this hardening is beneficial with the example shown in Figure 8-8. Let's consider a system where a database server has ingress filter rules set up to allow access from application servers. A well-meaning administrator is investigating some network connectivity issues. In the process of their investigation, the admin loosens the database's ingress filtering to rule out the possibility that it was causing the issue. Crucially, this administrator forgets to revert the change after disproving that theory. This error removes a layer of defense in the system for some time. Worse yet, discovering this lost defense can be difficult because the expected communication (from the app servers to the database server) continues to work.

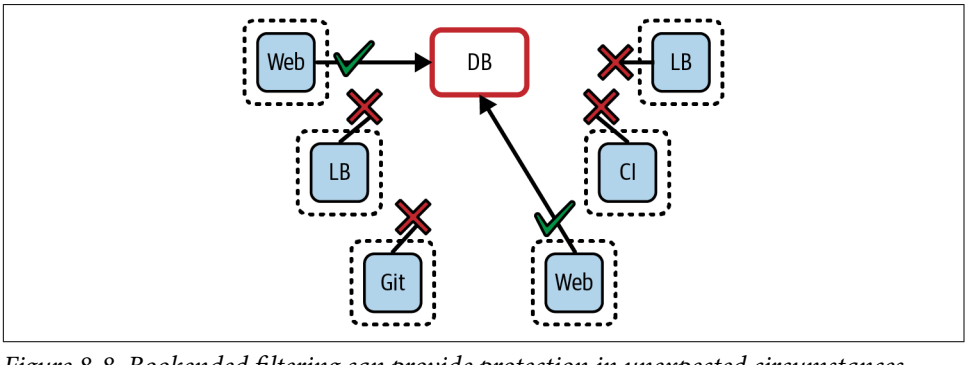

Figure 8-8. Bookended filtering can provide protection in unexpected circumstances

In this scenario, a network that has pervasive bookended filtering is protected even when this critical misconfiguration is in the system. In a way, it's similar to herd immunity—the collective benefit that a community provides to unvaccinated members when the vast majority of members are vaccinated against a disease. Instead of preventing illness, bookended filtering protects misconfigured systems from the potential impact of that misconfiguration.

Building bookended filtering into a system isn't as hard as it might seem, given the right conditions. Communication flows need to be captured in a way that can be consumed programmatically. The best way to capture these flows is by defining fine-grained ingress rules. These ingress rules should allow access to a service based on each client's server role instead of broadly opening access to a service. By capturing this detail, we have constructed a dependency graph from which egress rules can be calculated and applied throughout the system.

Like we discussed in host filtering, egress filtering is best applied when it is isolated from the applications running within the system. The same insights apply here: it's preferable to implement filtering on the other side of a virtualized or containerized environment to have the most robust filtering mechanisms. Looking beyond the filtering implementation, it's important to consider the isolation of the data used to build egress filtering rules. It might seem attractive to calculate that data from a dynamic data source such as a service discovery system, but bookended filtering is most effective when the flow database is isolated from the running system. Instead, use a slowly changing database, especially one that requires a human to review changes.

Project Calico

Project Calico is a virtual network system for dynamically scheduled workloads. A workload is a generic term that applies to any application that needs to be run in a datacenter. This application could be inside a container or a virtual machine. Calico takes the lessons learned in operating the internet and brings them into the datacenter to create a simpler network that can scale efficiently as the size of one's network grows.

Calico is not a full zero trust solution, but it does echo some of the ideas of zero trust networks. Calico distributes filtering throughout the network, which is enforced on the host machines. These hosts are dynamically reconfigured based on changes in a database that describes the entire network. This design looks very similar to the host filtering we discussed earlier.

Calico also includes the bookended filtering concepts we discussed. This means that hosts on both ends of a connection are filtering traffic based on their knowledge of which connections should be allowed. This double enforcement of network communication is seen as a secondary defense in the network fabric.

Intermediary Filtering

Intermediary filtering is the idea that devices other than the sender or receiver can and should participate in filtering traffic in a zero trust network. This, at minimum,

means perimeter filtering can play a role in a zero trust network, and at maximum, intermediary devices within the network's fabric do.

As we discussed in "Host Filtering" on page 188, filtering traffic only at the destination incurs an extra cost on the network when the ratio of undesirable traffic is very high. High-throughput filtered traffic will most often originate from internet ingress traffic. Ideally, we want to filter traffic as soon as possible to reduce the impact and the cost of filtering. For this application, filtering at the perimeter systems that sit between the zero trust network and the internet is ideal. These devices typically need to be hardware-based to efficiently filter the packets coming into the system. Perimeter filters can also be an important check and balance in a zero trust network. The perimeter filters should be a combination of global rules and coarse-grained host policy. By keeping global rules separate from host policy, invariants about the external network configuration are defined.

Exceptions to this policy should be traceable back to host infrastructure that relies on those exceptions, and the actions taken to instantiate them. The best implementation derives these exceptions from the host policies themselves. By tying the host policy to the exception policy, the system will be more consistent as hosts come and go from the network. These exceptions, however, must be verified to be as narrowly scoped as possible. A review process should be exercised for all policy changes in order to guard against overly broad exceptions, which can compromise the system's security.

UPnP Considered Harmful

Deriving perimeter policies from host policies should not be conflated with UPnP (Universal Plug and Play), a technology used to reconfigure consumer firewalls. UPnP is rightly criticized because any application on the network can reconfigure the perimeter. In the zero trust model, there is a chain of trust between the host policies and the exceptions that are created at the perimeter.

It might seem odd that we're discussing perimeter filtering in such a positive light, given the failings of the perimeter model. The key detail to understand here is that zero trust networks don't throw out all perimeter concepts. Instead, they encourage administrators to start at the host and work their way outward. Perimeter devices eventually play a role in this way, with denial-of-service mitigation being by far the most notable application.

An exciting idea in zero trust networks is to use the host policy database to dynamically program the network fabric itself. This would result in a software-defined network (SDN) that does not blindly route packets to the destination, but actively manages switching and routing policy based on which flows are expected and allowed. This results in a couple of benefits:

- Potentially malicious traffic is kept away from hosts, reducing the attack surface.
- Software firewalls on the hosts are augmented by the network itself, adding additional layers of defense in the network.

Like the perimeter filtering discussed earlier, filtering in the network fabric should be seen as an enhancement to the base layer of host-based filtering. It must not act as a replacement for it.

Forwarding and Routing Authorization

As we discuss filtering, there is a theme that arises—zero trust networks leverage relatively slowly changing details of the network to distribute enforcement, resulting in a network that is more secure. This observation opens up an interesting opportunity: can we propagate enforcement into the network infrastructure, effectively elevating those pipes from a simple packet transmitting system to a smart network fabric? Imagine an SDN controller that only installs flow instructions based on the result of a strong authentication and authorization process. A client wishing to access a network resource can signal the control plane, providing the network access request along with the appropriate credentials. After successful request authorization, the network is installed and available, but only for the specific flow that was authorized.

Scenario Walkthrough

In this scenario, Bob is using his browser to access email while connected to a public anonymous network (e.g., Tor or I2P). In this situation, it is critical to ensure that Bob remains productive without reducing the security posture. This is accomplished by granting Bob read-only access to the email while closely monitoring his activities. Again, an essential element in zero trust is to remember that policies like these are not static, but rather dynamic, and should be reviewed on a frequent basis. For example, a report may be generated specifically to assess how many users are actually using public anonymous networks and their access patterns across applications. This will aid in the continuous improvement of policy.

Take a look at Figure 8-9, which shows control plane components: user data, activity logs, and monitoring logs. Details of the trust engine and policy engine are shown in Figure 8-10.

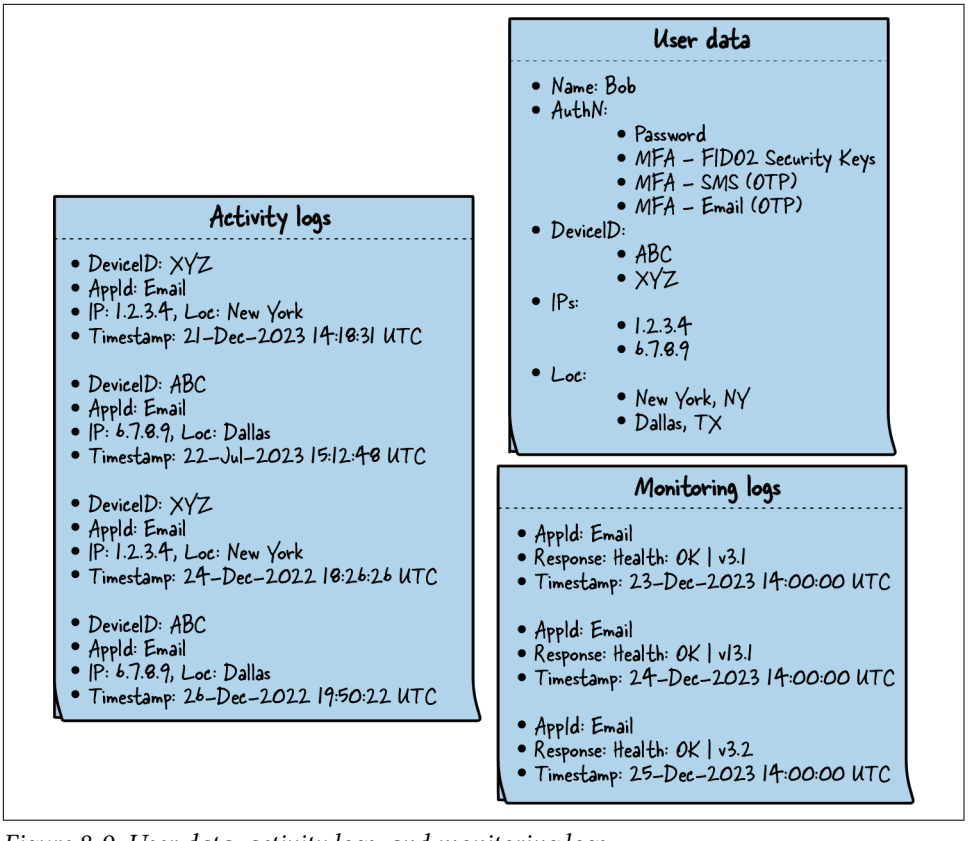

Figure 8-9. User data, activity logs, and monitoring logs

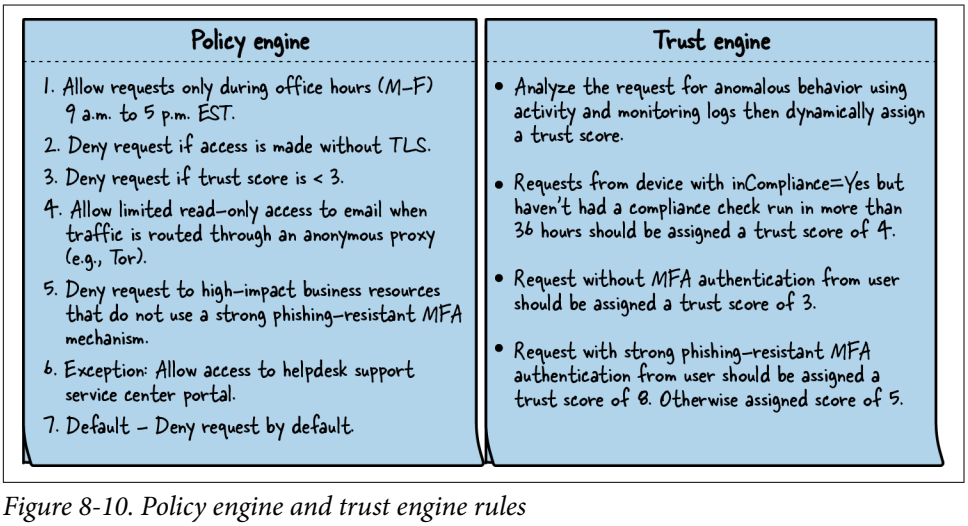

Figure 8-10. Policy engine and trust engine rules

Use Case: Bob Requests Access to an Email Service Over an Anonymous Proxy Network

Here is what we know about Bob's request:

- Bob is sending a request to access an email service via a web application while using an anonymous proxy network while at the airport.
- Bob's request is using the latest version of TLS while accessing email using a browser.
- Bob is using his work laptop with device ID "ABC," which is fully compliant with organization policy.
- Bob has used a strong phishing-resistant MFA method for authentication.
- Bob is making the request during office hours.

Here's what we know about the email service:

- Email is hosted by a public cloud vendor and offered as a SaaS service.
- All access to email requires an encrypted TLS channel.
- Continuous monitoring runs health probes on a regular basis from various regions globally to ensure the email service is online.

Request Analysis

1. Bob's access request (action: access-email, app-id: Browser, device-id: ABC, authentication: FIDO2, location: Dallas, IP:6.7.8.9, datetime: 23-Dec-2023-4:00pm-est-timezone) reaches the enforcement component.
2. The enforcement component forwards the access request to the policy engine for approval.
3. The policy engine receives the request and consults with the trust engine to determine the request's trust score.
4. The trust engine evaluates the request:
   - No anomalous behavior is detected based on the user activity logs, and monitoring logs also show the application responding to attestation requests properly.
   - The device is in compliance and had its most recent compliance check less than 36 hours ago, so 4 points are added to the trust score.

- Bob has also used FIDO2 as an MFA method, which is phishing resistant, so another 8 points are added to the trust score.
- Finally, the trust engine computes the average of the trust scores, which is 6, and returns it to the policy engine.

5. The policy engine receives the trust score of 6 from the trust engine.
6. For authorization, the policy engine compares the request to all policy rules:
   - This first rule results in a grant (or allow) action because the request is made during the permissible office hours.
   - The second rule does not apply because the access request is made over TLS.
   - The third rule does not apply, as the trust score received from the trust engine was 6, which is higher than 3
   - The fourth rule does apply to the current request as the request is for email service and is made over a public anonymous proxy network. Only read-only access to email is allowed.
   - The fifth rule does not apply since the request is using a strong phishing-resistant MFA mechanism.
   - The sixth rule does not apply since the request is not for the help desk support service center portal.
   - The seventh rule does not apply since it is a default and only applicable if any other prior rule is not applied. In this case, the fifth rule was applied to allow the request.
7. The policy engine sends an *allow* action request to the enforcement component.
8. The enforcement component receives the result from the policy engine and grants read-only access to the email service to Bob.

Summary

This chapter focused on how traffic gains trust in a zero trust network. We teased apart the distinctions between encryption and authenticity—two concepts that are related but distinct. Zero trust networks require authenticity in communication, and most networks also gain value in having their traffic encrypted.

We explored the first-packet problem in network communications. Modern authentication systems are fairly complicated, which results in a large surface area for attacks. We talked about hiding those services behind a single-packet authorization system, which is a relatively simple service that can be used to hide a more complex authentication system like TLS.

We then talked about two protocols for encryption and authentication of network traffic: TLS and IPsec. We provided guidance that mutually authenticated TLS

(mTLS) is best suited for client/server interactions or in heterogeneous environments, while IPsec seems well suited inside the datacenter (particularly so when network address translation is not present).

We also covered cloud traffic and the fact that cloud-based services present additional challenges to determining the trustworthiness of traffic. We talked about some of the challenges, leveraging authentication and authorization mechanisms for cloud-based flows, and cloud-native security tools to apply access control policies and enforce the principle of least privilege.

Zero trust networks still need packet-filtering capabilities, which they deploy throughout the network. We described three types of filtering that can be deployed in such a network: host, bookended, and intermediary filtering. Each type of filtering adds additional robustness to the network and can be deployed in the network using system automation and a shared database of expected network communication.

Finally, the chapter concludes with a user scenario walkthrough in which traffic is routed through a public anonymous proxy, and this affects how users access the email system. This is a real-world scenario in which a balance must be struck between productivity and maintaining a secure posture.

The next chapter takes all the concepts we have learned thus far and lays out a plan for creating your own zero trust network.

CHAPTER 9

Realizing a Zero Trust Network

This chapter will help readers develop a strategy for taking the knowledge in previous chapters and applying it to their system. Zero trust networks are very likely to be built around existing systems, so this chapter will focus on how to make that transition successfully.

It's important to remember that zero trust is not a product or even a single service that can be bolted onto the network. It is a set of architectural principles that are applied based on the needs and constraints of the network. Therefore, this chapter cannot provide a checklist of changes to be made, but rather a framework for how to approach realizing a zero trust network in your own system.

The First Steps Toward a Zero Trust Network: Understanding Your Current Network

Thoroughly assessing your network infrastructure is the bedrock of a robust zero trust strategy. Begin by mapping out all network elements, including devices, software, and data flows, to identify security gaps and areas ripe for enhancement. This comprehensive view of your network's current state is pivotal, providing insights into potential vulnerabilities and informing where and how to apply zero trust principles effectively. This foundational understanding is necessary for any security measures to be aligned with your organization's specific needs and vulnerabilities that could potentially lead to ineffective defenses and susceptibility to security breaches. The end goal is a clear blueprint of your existing network that will guide the seamless integration of zero trust in your environment.

Choosing Scope

Before setting out to build a zero trust network, it is important to choose the proper scope for the effort. A very mature zero trust network will have many interacting systems. For a large organization, constructing these systems might be feasible, but for smaller organizations, the number and complexity of those systems may make a zero trust network seem out of reach.

It's important to remember that the zero trust architecture is an ideal to work toward instead of a list of requirements that must be met completely from day one. This is no different from perimeter-based networks. Less mature networks may initially choose a simple network design to reduce the complexity of administration. As the network matures and the risk of a breach increases, the network will need to be redesigned to further isolate systems.

While the zero trust network design is an ideal, not all features of the design have equal value. Determining which components are required and which are nice to have will go a long way in ensuring the success of a zero trust implementation.

Assessment and Planning

Building a zero trust network starts with a strong foundation, so once you understand scope, the next step will be the assessment and planning phase. Think of this phase as laying the groundwork for the principles of "never trust, always verify" to thrive. During this phase, the current network setup is evaluated, assets are pinpointed, and a gap analysis is conducted to determine the steps needed to transition to a zero trust architecture.

Network assessments help evaluate the existing network structure and identify strengths, weaknesses, and security measures aligned with zero trust principles. Next comes asset identification and gap analysis. Asset identification involves enumerating and categorizing all assets in the network, such as devices, applications, data repositories, and cloud resources. Finally, the gap analysis reveals the differences between the current network structure and a zero trust architecture, highlighting the necessary technological and policy changes.

Embracing the Essence of Zero Trust

Embracing the essence of zero trust requires a fundamental paradigm shift in your approach to security. It's about understanding that zero trust is not a tangible product you can plug into your network but a comprehensive mindset that permeates every aspect of your organization's security posture.

Start by ingraining the "never trust, always verify" principle at all levels, from policy creation to daily operations. Educate stakeholders about the importance

of continuous verification and least-privilege access. The goal is to foster a proactive security culture where trust is earned, not assumed, leading to a significantly enhanced defense against evolving cyber threats.

Remember, zero trust is a mindset, not a product. The outcome should be a shift in organizational culture toward constant vigilance and security mindfulness.

Requirements: What Is Actually Required?

Limiting the scope of a zero trust network necessarily requires prioritizing the set of properties that are presented earlier in this book. This RFC-style prioritization list is the authors' opinion on how that work should be prioritized:

- All network flows MUST undergo authentication before processing.
- All network flows SHOULD be encrypted before transmission.
- Authentication and encryption MUST be performed by the endpoints in the network.
- System access SHOULD be enforced by enumerating all network flows.
- The strongest authentication and encryption suites SHOULD be used within the network.
- Authentication SHOULD NOT rely on public PKI providers. Private PKI systems should be used instead.
- Devices SHOULD be regularly scanned, patched, and rotated.

RFC-Style Prioritized Lists

Request for Comments (RFC) documents are the lingua franca of proposed changes to internet infrastructure. In these documents, language and structure are clearly defined to allow readers to more quickly understand the proposed changes. One aspect of that language that is very useful in prioritization discussions is the standard terms defined in RFC 2119. This RFC defines a set of terms (MUST/MUST NOT, SHOULD/SHOULD NOT, MAY/MAY NOT) which, when used, carry greater weight than their normal usage in common literature.

This book's prioritized list uses these terms with a similar intention to their definitions in RFC 2119. While architectural characteristics don't have quite the same requirements as protocol designs, the use of these standard terms is intended to echo the usage presented in that RFC. For completeness, here are the intended definitions of these standard terms when used in this book:

MUST
: This term is used for a requirement that is required for the implemented system to be considered compatible with the zero trust design.

MUST NOT
: This is the opposite of MUST. A system intending to implement the zero trust design is required to not have this characteristic.

SHOULD
: This term denotes an architectural characteristic that is desired in a zero trust network, but given cost constraints can be deprioritized. When deprioritizing this feature, system administrators should be aware that they are trading the security of their systems for reduced cost in implementing them. When at all possible, system administrators should avoid compromising on these characteristics because the benefit of not compromising on them is considered worth the up-front cost of their implementation.

SHOULD NOT
: This is the opposite of SHOULD.

MAY
: This term is used for architectural characteristics of a zero trust network that bring value, but are considered nice to have. System administrators should plan on implementing these aspects once they have built a system that satisfies the MUST and SHOULD definitions. It is important to note that these additional features bring additional value to the network by hardening it, so they should not be considered a net loss.

With this prioritized list of design requirements for building a zero trust network, let's dig into why particular requirements are categorized the way they are.

All Network Flows MUST Undergo Authentication Before Processing

In a zero trust network, all packets received by the system are immediately suspicious. As such, they must be rigorously inspected before allowing the data within them to be processed. Strong authentication is the primary mechanism by which we accomplish this.

Authentication is absolutely required in order to gain confidence about the provenance of network data. It is, perhaps, the single most important component of a zero trust network. Without it we have nothing, and are forced to place trust in the network.

All network flows SHOULD be encrypted before transmission

A key lesson of this book is that a network link cannot be trusted to reliably convey data or signals from one system to another. The physical accessibility of a network link to unsafe actors makes it trivial for that network to be compromised. Moreover, even in a physically secure network, bad actors can digitally infiltrate a system and passively probe the network for valuable data.

By encrypting data on a device before transmitting it on the network, we reduce the attack surface of that communication to the trustworthiness of the device itself, namely application and physical device security.

Authentication and encryption MUST be performed by the application-layer endpoints

Since zero trust networks recognize the threat that trusting network links poses to the security of a system, it is important that secure communications be established between application-layer endpoints. Adding middleware components that handle these responsibilities (like VPN concentrators or TLS-terminating load balancers) can leave upstream network communications exposed to physical and virtual threats. As a result, a system that claims to be zero trust is required to implement encryption and authentication at every application-layer endpoint on the network.

System access SHOULD be enforced by enumerating all network flows

Zero trust networks depend on data that defines the expected characteristics of the network. Therefore, defining every expected network flow is critical to safeguarding the network.

We should be careful to note that enumerating flows does not require onerous change management controls to provide value. A simple process for defining expected flows brings enormous value in terms of network enforcement and change auditing. Without the list of expected network flows, zero trust systems are unable to highlight unexpected communications that need attention from administrators or should be denied.

It is the strongly held opinion of the authors that deferring the effort to enumerate flows will ultimately result in a task list that is considered infeasible. The authors feel that the best way to keep this database of expected flows up to date is to distribute the responsibility of defining those flows throughout the organization. When distributing this responsibility, organizations should take caution to educate teams on best practices for change management to guard against internal threats to the system. One such threat is when a single person is allowed to update the flow database without any oversight. A simple review system can mitigate this threat.

Flow Data as the Source of Truth

Building a database of expected flows is best accomplished by making the flow database the data source for allowing that access. By setting up this dependency (and disallowing external modification), the flow database will be consistent with the actual allowed access.

When capturing flows, following these rules will improve the quality of the data:

- Capture the intended use of a flow along with the policy details (e.g., load balancer access—from load balancing hosts to web application).
- Prefer narrowly defined flows over broad access.

The strongest authentication and encryption suites available SHOULD be used within the network

Zero trust networks assume a hostile network environment, so strong authentication and encryption suites are an important component in the security of a zero trust network. Which suites offer strong security, unfortunately, changes, so this book cannot offer specific choices that will stand the test of time. Readers should refer to security standards like the NIST encryption guidelines to pick strong cipher suites.

System administrators should always aim for the strongest suites possible, but device and application capabilities might limit the types of suites that are available. In these cases, administrators should be aware that by reducing the strength of these suites, security is being compromised in their network.

Authentication SHOULD NOT rely on public PKI providers—private PKI systems should be used instead

Public PKI systems provide trust assurances to unmanaged endpoints in a secure communication. A certificate authority (CA) signs certificates used in establishing secure communications. The endpoint receiving that signed certificate is able to verify its authenticity by comparing the signed material against the list of trusted certificate authorities already present on the system. By seeding systems with a list of trusted public certificate authorities, endpoints can establish secure communication channels with systems they have not previously communicated with.

Given the benefit that the public PKI system provides to build secure communication channels, why do zero trust networks prefer private PKI systems? The reason, perhaps unsurprisingly given zero trust's focus on managing trust, is that trusting a third party places the system at increased risk. There are several risks that the public PKI system brings to a zero trust network.

One concern is the number of public CAs that are considered trusted. As internet traffic has grown, the number of trusted public CAs has grown with it. Each one of those trusted CAs has the ability to sign a fraudulent certificate that incorrectly asserts the trustworthiness of a malicious system. Certificate pinning can help with this risk by giving an endpoint the knowledge of which certificate to expect for a given endpoint, but certificate pinning requires that the endpoint have prior knowledge of the expected certificate, which presents a new challenge.

Using a public CA also presents another threat. State actors have become more aggressive in using judicial powers to force organizations to act against the trust guarantees that they provide to their customers. These requests have increasingly used laws that prohibit involved parties from disclosing their actions. Given this aggressive stance, allowing state actors into the trust mechanisms of a zero trust network should give system administrators pause.

Based on these concerns, zero trust networks should prefer privately held PKI systems. Endpoints should be configured to only allow certificates signed by the private PKI system. We discussed PKI in greater detail in Chapter 2.

Devices SHOULD be regularly scanned, patched, and rotated

We learned in Chapter 5 that the security of devices is critical for building a zero trust network. Administrators need to build with the assumption that trusted devices on the network are compromised, and therefore build defenses into device management to mitigate this threat.

To that end, devices should be regularly scanned to capture the software that's running or installed on the device at a given point in time. Scanning can be used to discover and prevent known malicious software from running on the device, but administrators should operate under the assumption that malware prevention software (e.g., antivirus software) will always be imperfect. Rather than focusing all energy on stopping malicious software from running, administrators should focus on building forensics capabilities so they can analyze the impact of an inevitable malware attack.

Keeping devices fresh is also very important. System administrators should have a plan for regularly installing the latest security patches. Additionally, a regular device rotation policy will help ensure that devices don't accrue cruft, which can compromise the security of the system.

Prefer Reimaging over Long-Term Scanning and Patching

Device trustworthiness degrades over time due to the increased risk that a device could have been compromised. Regularly reimaging devices, while disruptive, ensures that the trust in the fleet remains high. Aim to reimage servers once a quarter, and personal devices every two years.

Building a System Diagram

Building a system diagram is another important step toward realizing a zero trust network. Having a clear picture of how both internal and external network communication is occurring will be useful when designing system communication channels.

System diagrams, such as the one shown in Figure 9-1, are often maligned for being horribly out of date. These diagrams are typically built by hand, which requires a large amount of human effort. Given the speed at which the diagrams fall out of date, there is a commonly held opinion that system diagrams simply aren't worth the investment. This viewpoint, however, misses the benefit of having a human-focused view of how the system should be constructed. While an engineer could read code or interrogate existing systems to determine how the system is constructed, this doesn't give any insight into whether that state was desired or accidental.

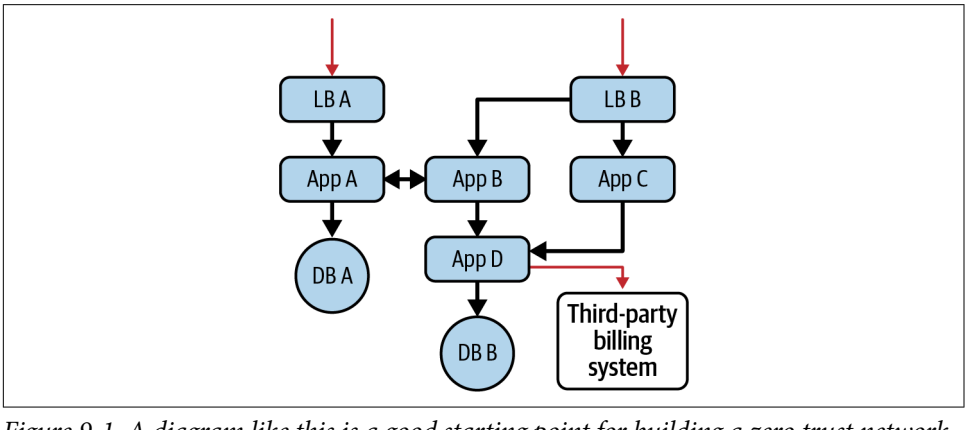

Figure 9-1. A diagram like this is a good starting point for building a zero trust network. Directionality is important.

So if system diagrams are useful, but often out of date, the natural question is how much time and effort should be put into their creation. A good path forward for an existing network is to first observe the communication that is flowing through the network. You can capture this communication using tools that log flows. Once flow information is captured, producing a system diagram will be an exercise in categorizing classes of communication.

In the next section, we talk about tools for capturing and categorizing network flows, as well as a strategy for breaking down this large effort into smaller chunks of work.

Understanding Your Flows

A network flow is a time-bound communication between a source system and a destination. A single flow could be directly mapped to an entire conversation when using a bidirectional transport protocol (e.g., TCP). For unidirectional transport protocols (e.g., UDP), a single flow might only capture half of a network conversation. This is because while two UDP flows might be logically related, an observer on the network may be unable to make that association without a deep understanding of the application data.

Capturing all the flow activity in an existing production network is a logical first step for a system that wants to move to a zero trust model. Logging flows in a network over a long period of time is a noninvasive way to discover what network connections exist and should be considered in the new security model. Without this up-front information gathering, efforts to move to a zero trust model will result in frequent network communication issues, causing the project to be deemed too invasive and disruptive.

Ways to Discover Flows

There are many different mechanisms for logging and analyzing network flows. Which system is used will largely depend on the type of network being run (physical or virtual) and the level of access that an administrator has over the endpoints.

As of the writing of this book, here are a few of the popular and widely regarded tools for capturing flows:

Wireshark
: Wireshark remains a powerful and widely used network protocol analyzer. It allows you to capture and interactively browse the traffic running on a computer network.

Cisco Secure Network Analytics (formerly Stealthwatch)
: This is known for its network telemetry and sophisticated analytics. It utilizes enterprise telemetry from the existing network infrastructure for advanced threat detection, behavioral modeling, and secure network visibility.

ManageEngine NetFlow Analyzer
: This is a real-time bandwidth monitoring tool that relies on flow technologies like NetFlow, sFlow, and Cflow to create comprehensive system diagrams and traffic analytics.

Plixer Scrutinizer
: This provides flow analysis for enhanced visibility into network traffic. It helps in creating detailed system diagrams by analyzing data from various flow protocols.

Datadog Network Performance Monitoring
: Datadog's solution allows you to visualize network flows in real time, making it easier to map out system diagrams and understand the interactions between various network components.

SolarWinds Network Performance Monitor
: SolarWinds provides comprehensive network flow monitoring and analysis. It can help with creating system diagrams by tracking flow data using tools such as NetFlow, Jflow, sFlow, and others.

These tools can be instrumental in logging flows and creating system diagrams, which are critical for implementing and managing zero trust networks. They provide the necessary visibility and analytics to understand network behavior and ensure secure operations.

Physical networks have rich capabilities for accessing the raw packets that are flowing over the network. Business-class switches will generally have the ability to mirror packets to a second port on the switch (known as a SPAN—Switched Port Analyzer—or mirror port). This approach is relatively safe to enable on a lightly loaded switch, but it will mask some types of errors in the network. TAP (test access point) devices, which are placed inline in the network link, will guarantee that all data is transmitted to a monitoring device. For the purposes of discovering logical flows in the network, either approach will work.

Virtualized networks might have the ability to inspect network traffic, but they generally operate on a coarser level. Amazon Web Services, for example, has a feature that logs every flow in a network, which can be used to analyze traffic on its systems (Figure 9-2).

While discovering flows via the network fabric gives perfect visibility into the traffic that is flowing, tying that analysis back to individual applications is difficult without an endpoint monitoring system. In a case where control of endpoints is feasible, discovering network flows on the endpoints themselves can provide a more detailed view of the source of traffic in the system. Software firewalls operating in log-only mode can be a useful tool to discover flows in the system without impacting communication. On Linux endpoints, there are several approaches to discovering and cataloging network flows, which Harald Welte's paper "Flow-based network accounting with Linux" captures.

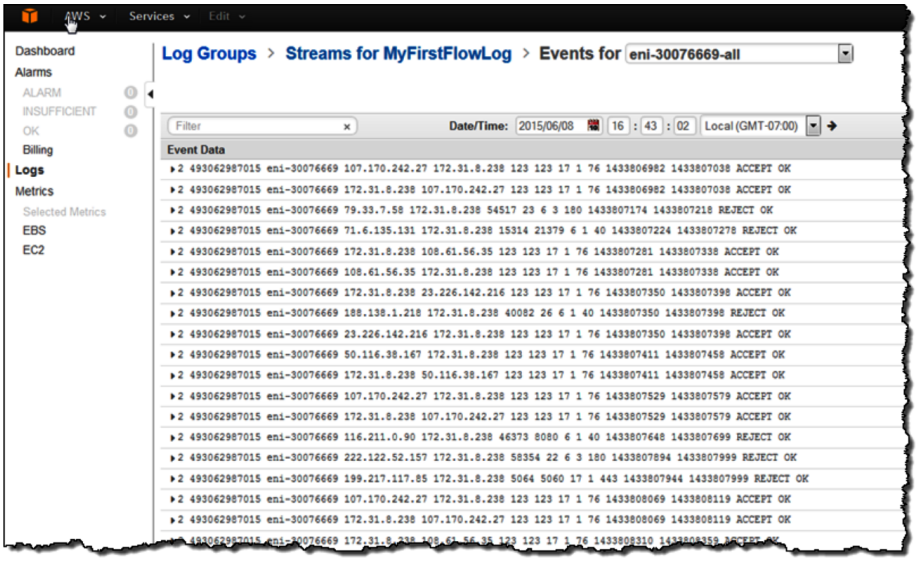

Figure 9-2. Some cloud providers have flow logging features built in; this is a screenshot of the AWS Flow Log feature

With all network flows logged, the next goal is to categorize flows based on higher-level system connections. These connections should be defined at the logical systems level instead of the individual IP/port level. The connections being defined with this exercise are very valuable data. With the definitions in hand, one is able to better enforce known connections and gain awareness of changes to the communication patterns within a network. Since many operations of a secure network can be derived from this database of connections, it's clear that capturing this mapping is very useful.

For a very large network, capturing and categorizing all network flows could be an enormous undertaking. The natural question is whether capturing all network connections is a requirement for transitioning to a zero trust network. Fortunately, a zero trust network can be incrementally realized within an existing perimeter-based system. One can leverage the existing perimeter or network boundaries to build a zero trust network on either side of the boundary. The zero trust model can then spread from zone to zone as in Figure 9-3, enhancing the network security of the existing system while maintaining the operational security measures already in place.

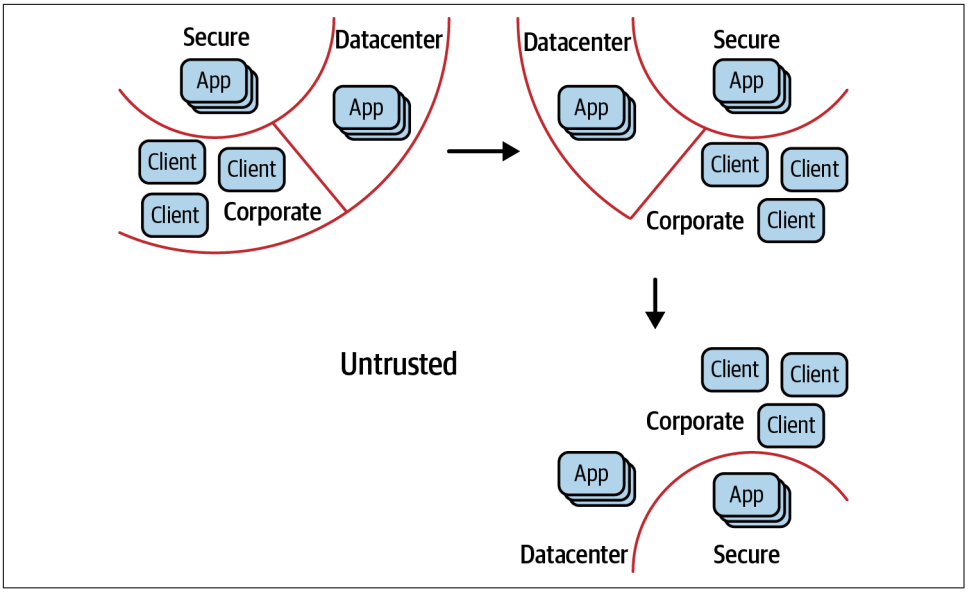

Figure 9-3. Zero trust adoption can move zone by zone, providing an easy migration path away from the traditional perimeter architecture

Micro-Segmentation

Micro-segmentation is a fundamental cornerstone in implementing a zero trust network and involves dividing the network into smaller, more manageable, and secure zones, allowing organizations to have precise control over interactions and data flows between different sectors of the network.

This level of control is crucial for embodying the principle of zero trust, which emphasizes verification over blind trust. It ensures that each network segment follows strict access and security policies, regardless of the overall network environment.

As per the National Institute of Standards and Technology (NIST), capabilities enabled by micro-segmentation include:

- Segments being isolated and relatively small enables close monitoring of the traffic because of better visibility.
- The consequence of the above capability is that granular access control is possible by defining associated policies.

In today's ever-growing threat landscape, micro-segmentation is key in mitigating the risk of threats spreading within the network. By isolating breaches to specific segments, it prevents them from spreading and causing broader compromise. As organizations increasingly adopt hybrid and multi-cloud architectures, micro-segmentation

provides a structured approach to managing and securing diverse network environments within a unified framework.

Software-Defined Perimeter

The goal of software-defined perimeter (SDP) architecture is to provide logically air gapped, dynamically provisioned, on-demand networks that are isolated from unsecured networks and resistant to common network-based attacks. By enabling a drop-all firewall by default, an SDP enhances security by requiring authentication and authorization before allowing access to assets concealed by the SDP system by users or devices. Furthermore, by mandating connection pre-vetting, a SDP will restrict all connections into the trusted zone depending on who may connect, from which devices, to which services and infrastructure, along with considering other factors.

An SDP can be a useful technique for implementing zero trust architecture and has been a focus of active research in recent years. For more comprehensive guidance on the topic, refer to "Integrating SDP and DNS: Enhanced Zero Trust Policy Enforcement," published by the Cloud Security Alliance.

Controller-Less Architecture

A fully mature zero trust network will have at its core several control plane systems that provide critical security services. While having these systems is ideal, it is possible to iterate toward the idealized deployment while using common infrastructure systems initially. We will explore some of these systems now.

"Cheating" with Configuration Management

Many operationally mature organizations use configuration management tools to manage their infrastructure. When using these systems, the desired configuration state is captured and version controlled. After examining the current state of the system, the configuration management system uses this desired configuration to calculate modifications that will bring the system to the desired state. Using a configuration management tool brings a number of benefits over planned changes executed by humans:

- Changes to the system are applied consistently across the entire fleet.
- The configuration data can be stored in a version control system, which provides a useful record of what changes were made and why.
- Configuration drift is less likely to occur, since its state is policed by the configuration management system.

The first way that configuration management is often deployed is to manage the configuration of individual computers. The systems are started from a known blank slate (usually just the initial installation of the operation system) and then reconfigured to the desired state based on that machine's role in the infrastructure. Having this process automated makes it easy to replace infrastructure.

While using configuration management for this task brings a lot of value, these tools can also be used as a general-purpose automation framework. For instance, they can be used to configure cryptographic primitives between infrastructure hosts, or to poke tightly scoped holes in host-based firewalls. In this way, configuration management (or CM) systems can be used to drive a subset of the functions that are normally offered by a mature zero trust control plane.

Similarly, CM systems can also be used to build up useful abstractions in the network. Most CM tools support mechanisms for extending a set of available resources or actions. Using this extension point, it's possible to build more complex resources into the system. For example, one could define the concept of a service resource that would capture all the standard infrastructure that should be used to make the service available on the network.

CM Is a Temporary Stepping Stone

Configuration management systems are best deployed in a manner wherein the system reaches a stable configuration. With this ideal in mind, using a configuration management system to make frequent changes to the system would seem counterproductive. We shouldn't dismiss this concern, as it has some validity to it. Instead, we should be mindful that leveraging a configuration management system to build a zero trust network is just a stepping stone to the ideal solution, which would move those responsibilities to a dedicated controller.

Implementation Phase: Application Authentication and Authorization

A typical organization makes use of many services, the client-side delivery of which is increasingly browser based. Since a zero trust network does not infer trust based on the network address of a connection, every service needs to handle authentication and authorization.

A simple solution is to store usernames and passwords in each application. This approach, however, is heavily discouraged, primarily due to management complexity. Instead of having each application implement its own authentication systems, it is far better to have applications integrate with an identity provider system that can

provide centralized authentication and authorization checks. SAML (Security Assertion Markup Language) is one technology that can be used to integrate an application with an identity provider. OAuth2 is another.

This is not to say that an application should have no authorization responsibilities at all. To the contrary, it is expected that some application-level authorization exists, particularly when considering things like varying user permissions. The overhead of account management, user authentication, and high-level authorization/access can be offloaded while still allowing room for application-centric authorization. When authenticating with an identity provider, multifactor authentication must be used to ensure that the user credentials cannot be easily stolen. We discuss multifactor authentication in Chapter 6.

Authenticating Load Balancers and Proxies

Many service architectures call for the use of a load balancer to distribute requests to a set of backend hosts. Oftentimes these load balancers represent the boundary between a client-facing system and a datacenter system. This can create confusion around how to properly apply zero trust controls in such a system, since client-facing zero trust semantics can be fairly different from those of server-side systems.

In Chapter 7, we write about how to manage application authentication and authorization as an analog to user authentication and authorization. In backend systems, the best way to authorize an application is to inject ephemeral credentials at runtime, whether that be an API key, short-lived certificate, or otherwise. Each credential uniquely represents a running application instance.

In a load-balanced system, the load-balancing software itself can be viewed as a server-side application. Each software instance is started with ephemeral credentials identifying the instance to upstream hosts. This is in addition to device authentication, which occurs between the load balancer and upstream system using techniques discussed in Chapter 5.

With this architecture, the load balancer can then handle user and client device authentication and authorization responsibilities, leveraging identity providers if desired. Information from the resulting authentication and authorization process (such as username) can then be sent along with the original request to the backend hosts. In this way, the zero trust architecture can be preserved as data crosses client/server boundaries and enters the datacenter.

Prefer Security Tokens over TOTP

When multifactor authentication was first deployed in organizations, users were given simple devices that continuously generated time-based tokens. With the prevalence of today's smartphones, most users prefer to use a multifactor application on their smartphone to generate codes.

Protocols that use security tokens, like U2F, are increasingly preferred over time-based token systems due to their protection against phishing attacks. It's a bonus that these systems are generally also easier for users to work with. When possible, prefer security tokens over TOTP systems. We discussed these technologies in Chapter 6.

Relationship-Oriented Policy

Zero trust advocates for a control plane that injects the results of authorization decisions into the network to allow trusted communication to occur. In that model, each network flow is individually authenticated and authorized. Enforcement is obtained by reconfiguring or signaling the network fabric to allow authorized communication. In a scaled-down zero trust network, which lacks these control plane systems, we are forced to scale back that ambition. Instead of building a network that uses dynamic injection and signaling, we can build a system that defines policies at the relationship level.

In the relationship-oriented network policy, communication between two devices is defined and controlled via traditional network filtering mechanisms like firewalls and required TLS connections. These policy enforcement mechanisms can seem very similar to a perimeter-based model. The key difference in the relationship-oriented model is that the policy is tightly scoped to communicating devices instead of communicating network segments. This approach is sometimes referred to as microperimeterization. By capturing and enforcing which devices should be communicating with each other, we build a database of expected communication, which will be of great value in the future when dynamic policy systems are deciding whether to allow a network flow.

Policy Distribution

Distributing policy (as opposed to just enforcement) throughout the network is a common characteristic of a scaled-down version of zero trust. Given the fine-grained policy decisions we expect in the network, automation is critical to making the network operable.

In a mature zero trust network, policy interpretation is fully handled by control plane systems, which can dynamically reconfigure network infrastructure and devices, or give authorization responses to signaling enforcement components. In a

controller-less deployment, however, we must use a different mechanism. Configuration management systems can be used to fill this void in the network control plane.

Devices can be dynamically configured to implement their own enforcement of expected network communication. Configuring an on-host software firewall that is calculated from the relationship policy database can provide per-host enforcement that is less difficult to operate than a centralized, physical firewall. Communications can be similarly authorized by hosts via mechanisms like mutually authenticated TLS, again controlled by configuration management software.

The key realization here is that by using existing configuration management systems, we are able to build a virtual control plane, which can distribute enforcement responsibilities into the network fabric. While this approach is pragmatic, it isn't without its downsides:

- Requiring hosts to enforce policy risks having that policy removed or altered should the host be compromised. In compatible environments, pushing this responsibility across an isolation boundary (e.g., a hypervisor, the host OS in containerized systems, or network security groups) provides better protection.
- Changes via configuration management systems often have a longer period of inconsistency while policy is being rolled out into the system.

Defining and Implementing Security Policies

Security policies need to be captured in a format that's separate from the individual devices that are used to implement those policies. There are a few reasons for storing this data outside the implementing systems:

- Having the policy captured separately allows for auditing of the implementation against the desired policy.
- The policy definitions can be reused when switching underlying enforcement systems. For example, configuring a new vendor's system is made easier if the policy is captured in a non-vendor-specific format.

A separate database that captures intended policy can quickly fall out of date unless mechanisms are put in place to ensure that it is consistent with the implementation. The best way to ensure this happens is to generate implementation configuration from this policy database using configuration management systems.

Some system administrators may choose to capture policy directly in configuration management code. In less mature networks, this approach is considered sufficient, since the configuration management system will consistently apply the policies defined on the target devices. As the network matures, administrators may find that moving the definitions out to data allows for them to be used in more locations.

For example, host-based and managed network firewalls could be configured from a shared policy database if that data is extracted from configuration management code. Defining variable trust policies is too difficult to attempt in less mature networks. System administrators should instead focus on defining and capturing known policies.

When building up policies, especially in an existing network, it is helpful to have mechanisms for testing proposed policies. The gold standard is a system that can take proposed policy changes and report on traffic that would be denied by the enforcement of those policy changes. Building up this policy preview system requires quite a few components: a database of logged production flows, a policy simulator, and a system to identify differences in current production policy and proposed policy. For many organizations, that level of sophisticated policy simulation is simply out of reach.

A simpler approach to safely introducing policy changes can be achieved using the following rollout procedure:

1. Take a subset of the desired policy, which we will call the proposed policy.
2. Deploy the proposed policy in a logging-only fashion.
3. Collect production traffic over a sufficient period of time.
4. Investigate traffic that would be rejected should the proposed policy be enforced.
5. Enforce the proposed policy.
6. Repeat this process until all desired policy has been deployed.
7. When all the desired policy is in place, enable a policy that rejects traffic by default.

This "log then enforce" procedure will provide ample time to discover unforeseen issues in the production environment. In addition to this approach, a phased rollout, wherein policy is enforced over a subset of the production footprint, can also help identify issues without affecting the entire production system.

Zero Trust Proxies

Zero trust proxies are application-level proxy servers that can be used to secure a zero trust network. Proxies are deployed as infrastructure to handle authentication, authorization, and encryption responsibilities. The manner in which these proxies are deployed is critical to ensure the safety of a zero trust network.

Zero trust proxies can operate in two different modes: reverse proxy or forward proxy. Depending on the situation, one or both of these proxy modes may be used, as shown in Figure 9-4.

In reverse proxy mode, the proxy is receiving connection requests from zero trust-enabled clients. The proxy receives the initial connection, validates that the connection should be allowed, and then passes the request to the application for processing. In forward proxy mode, a nonzero trust-aware component needs to make a network request to another zero trust system on the network. Since the nonzero trust-aware component is unable to work with the control plane to initiate the request properly, it communicates through the authentication proxy to handle that responsibility.

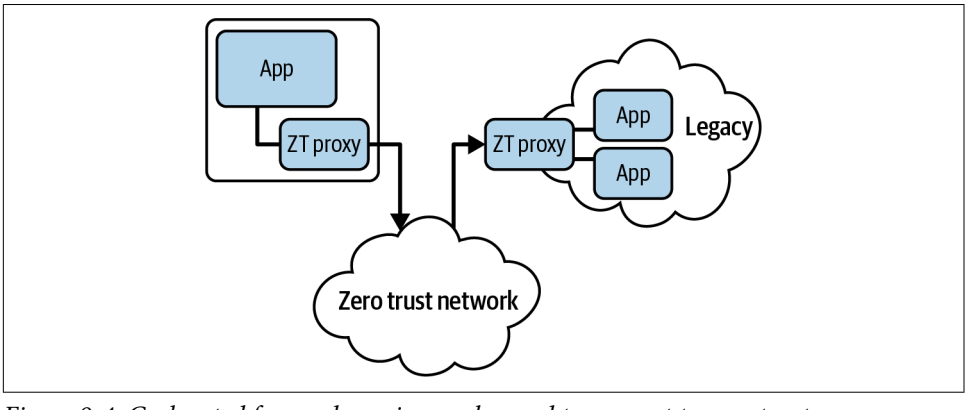

Figure 9-4. Co-located forward proxies can be used to connect to zero trust resources from legacy systems, while co-located or centralized reverse proxies can allow access to legacy services by zero trust clients

Proxies can be used to build a zero trust network, but the proxies should be deployed on the same device that the workload is running on. When a zero trust network is built in this manner, all workload communication is forcibly routed through the proxy before being emitted on the network. Isolating this responsibility in a proxy brings advantages over incorporating it in individual applications, which we cover in Chapter 8.

Placing proxies on dedicated devices is not recommended for building a zero trust network. Trying to isolate zero trust responsibilities in an external proxy goes against the model, which seeks to secure all traffic, including traffic between proxies/load balancers and backend services.

Building a zero trust network can be especially difficult for system administrators who do not have complete control of all devices or services on the network. For example, a network might have vendor-supplied components that need to be secured without changing the device itself.

Zero trust proxies can help bridge the gap in this situation. Placing such a proxy between the unmodifiable component and the zero trust network can allow that component to participate in the network, though with a lesser guarantee of its security. It is critical that the nonzero trust-aware component be completely isolated. This

isolation must ensure that all network communication to and from that component can only occur through its authentication proxy. If possible, direct mechanical connection should be preferred.

Client-Side Versus Server-Side Migrations

When realizing a zero trust network, deciding on whether client-to-server interactions or server-to-server interactions should be undertaken first is ultimately dependent on the needs of the organization and the level of effort required to meet the goal. Client-to-server interactions are usually the first to be focused on. Oftentimes, the clients are physically mobile and accessing services from uncontrolled networks. Additionally, with these devices being mobile, the physical security of the device is reasonably called into question. Building zero trust capabilities at this access point, therefore, brings a lot of value.

There are, however, real hurdles to building zero trust at the client/server layer. Organizations don't necessarily have existing automation systems installed on client machines to allow the zero trust network to be built. Additionally, the types of devices in use on the client side can be much more diverse, which means that the required automation has to be compatible with more systems.

Server-to-server interactions can be an easier initial target for zero trust networks. These systems frequently have existing automation tools installed. They also tend to have a less diverse set of providers in use. Finally, they are often the systems that are housing sensitive data, and so are an attractive target for would-be attackers. Ultimately, the decision of where to start should focus on which target is the weakest link in the system's network defenses. Building a threat model can help determine which systems are the most exposed. With that knowledge, choosing where to invest time and resources is easier.

Given these considerations, the following steps outline a structured approach to determining the focus of initial efforts in transitioning to a zero trust network, ensuring that organizational resources are utilized effectively to mitigate identified risks:

Identify priority areas
: Evaluate the risks associated with both client-side and server-side interactions to ascertain where the initial efforts would be most beneficial.

Conduct threat modeling
: Undertake a threat modeling exercise to gauge the potential threats facing client-to-server and server-to-server interactions, aiding in informed decision making.

Allocate resources
: Based on the assessment and threat modeling, allocate resources strategically to address the identified priorities.

Iterative implementation
: Begin with the prioritized layer, apply zero trust principles, assess the effectiveness, and progressively extend the implementation to the other layers.

Monitor and adapt
: Establish a continuous monitoring mechanism to evaluate the effectiveness of the zero trust controls and adapt the strategy in response to evolving threat landscapes and organizational requirements.

By methodically evaluating the risks and benefits associated with each layer, organizations can make informed decisions on where to focus initial efforts. The structured approach outlined above provides a roadmap to navigate this complex terrain, ensuring the transition to a zero trust network is strategic, manageable, and aligned with organizational objectives.

The journey toward zero trust may begin with a focus on either client-side or server-side interactions; however, a holistic and continuous approach to implementing and refining zero trust principles across all network interactions is fundamental for achieving a robust security posture in the long term.

Endpoint Security

In the context of realizing a zero trust network, endpoint security is paramount. It involves securing every device that connects to the network, as each endpoint can potentially serve as an entry point for threats. To reinforce endpoint security, ensure that all endpoints comply with your organization's security policies. This includes implementing strong authentication, regular patching, and endpoint detection and response (EDR) solutions. Additionally, enforce the principle of least privilege to minimize the access and permissions of each endpoint. One example of enforcing the principle of least privilege could be a company policy wherein employees are not given local admin rights on their company-issued laptops. Instead, they have standard user accounts for daily tasks. If they need to install software or perform actions that require admin rights, they must go through a controlled process, such as submitting a request to the IT department. This procedure is reviewed, and if justified, the IT team performs the action or grants temporary admin rights. This minimizes the risk of unauthorized changes to the system and reduces the attack surface for potential exploits. The expected outcome is a network where trust is continuously earned and validated, drastically reducing the attack surface and enhancing the overall security posture.

Case Studies

Since the exact architecture of a zero trust network is dependent on the details of a particular organization's network, it can be hard to see how all the pieces fit together. To help visualize how these principles manifest themselves in different situations, we are going to explore the experiences of a couple of organizations that have successfully transitioned to a zero trust model.

Google's BeyondCorp effort focused on bringing zero trust architecture to the client-to-server interactions that their highly distributed and mobile workforce uses every day.

PagerDuty's cloud-agnostic network focuses on server-to-server and cross-cloud interactions that needed to be secured from both external and internal threats.

Case Study: Google BeyondCorp

Betsy Beyer

Starting in November 2014, Google published a series of articles in *;login:* describing a new and groundbreaking security model it was deploying to its entire corporate network. The following case study is based on excerpts from those three articles, with permission from Google and *:login;*.

We encourage you to read the original source material to learn more details:

- "BeyondCorp: A New Approach to Enterprise Security" by Rory Ward and Betsy (Adrienne Elizabeth) Beyer
- "BeyondCorp: Design to Deployment at Google" by Barclay Osborn, Justin McWilliams, Betsy Beyer, and Max Saltonstall
- "Beyond Corp: The Access Proxy" by Batz Spear, Betsy Beyer, Luca Cittadini, and Max Saltonstall

By the early 2010s, Google was increasingly uncomfortable with the perimeter model of network defense. Creating high, impregnable "castle walls" was not going to protect us when tens of thousands of our employees performed much of their work while physically outside our offices, while on any given day we invited thousands of people inside. At the same time, as the critical role Google plays in the lives of billions of users continued to increase, so did the almost incalculable value we place on the user data entrusted to us.

In light of the scope and scale of our employee base and our corporate network, and the variety of ways in which our employees interact with corporate resources (as a mobile workforce using cloud services and a variety of client devices), it became obvious that the castle-wall metaphor was unsustainable. We needed a strategy much

more akin to a modern city than a medieval castle: a system that mediates access to applications, data, and services according to who you are, not which network you use.

With this security imperative in mind, Google revisited the state of the enterprise with a fresh set of eyes. We knew that we could do better than any of the conventional network security models deployed across the industry, so we took the radical step of redesigning our entire approach.

Starting from square one in re-envisioning internal network security, we invested over four years of design and iteration in creating a robust implementation of the zero trust model. While most enterprises assume that the internal network is a safe environment in which to expose corporate applications, we assume that an internal network is as fraught with danger as the public internet.

This new model dispenses with a privileged corporate network entirely. Instead, access depends solely on device and user credentials, regardless of a user's network location—be it an enterprise location, a home network, or a hotel or coffee shop. All access to enterprise resources is fully authenticated, fully authorized, and fully encrypted based upon device state and user credentials. We can enforce fine-grained access to different parts of enterprise resources. As a result, all Google employees can work successfully from any network, and without the need for a traditional VPN connection into the privileged network. The user experience between local and remote access to enterprise resources is effectively identical, apart from potential differences in latency.

When reading the following case study, keep in mind that we're well aware that Google is unique both in terms of its scale and in the amount of resources we were able to devote to this problem space. Because we weren't constrained by resources, we could act more or less purely motivated by ambitious goals that did away with the conventional network security paradigm.

Fast-forward from BeyondCorp's inception to 2017: hacking tools have advanced in sophistication and dropped massively in cost. Malicious efforts that might once have been worthwhile only when turned against Google-scale targets are now applicable to much smaller enterprises. While the risk profile of small- to medium-sized organizations has increased, so too have their options to protect themselves; the commercial network security industry has likewise matured. While Google had to build its security infrastructure from scratch, today there actually are enterprise network security offerings your organization can employ in moving away from the perimeter model. Regardless of individual components you're considering in this space, keep the core design principles and objectives that motivated Google in mind as you develop a strategy.

While technical and implementation details of BeyondCorp may have varying degrees of direct applicability to your enterprise or organization, many of the risk

factors we designed to protect against are widely germane, and the fundamental design principles we employed should be directly relevant to all.

The Major Components of BeyondCorp

As shown in Figure 9-5, BeyondCorp consists of many cooperating components to ensure that only appropriately authenticated devices and users are authorized to access the requisite enterprise applications. The following sections describe individual components of BeyondCorp.

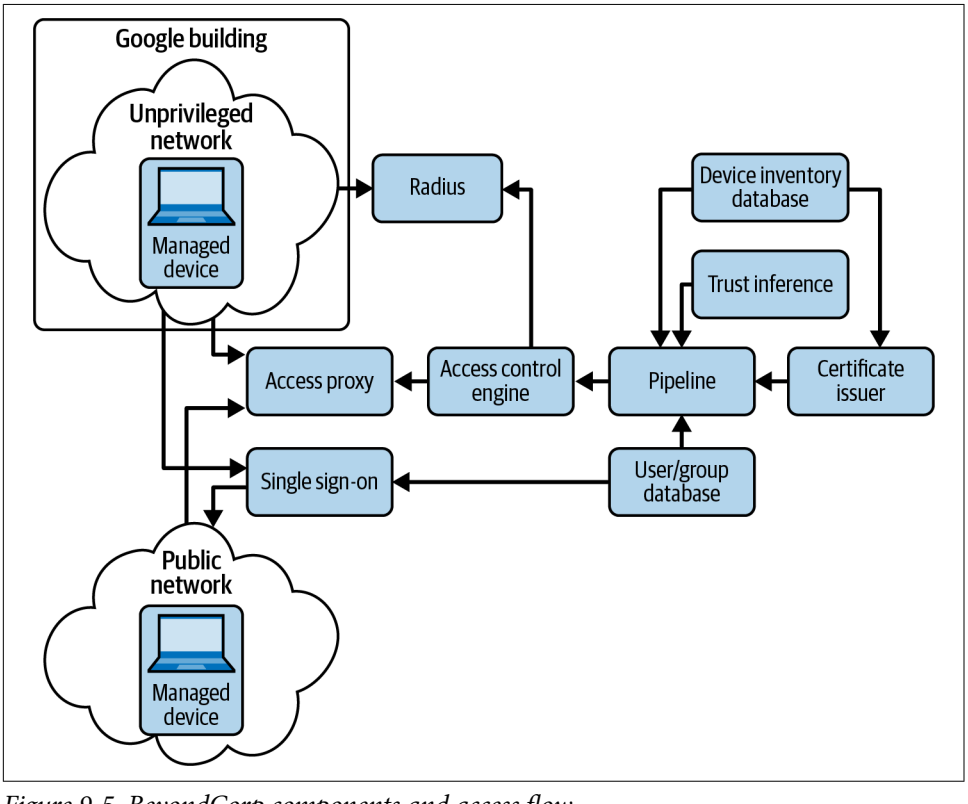

Figure 9-5. BeyondCorp components and access flow

Securely identifying the device

BeyondCorp securely identifies and tracks all managed devices using a master Device Inventory Database and device certificates.

Device Inventory Database

BeyondCorp uses the concept of a "managed device," which is a device that is procured and actively managed by the enterprise. Only managed devices can access

corporate applications. A device tracking and procurement process revolving around our Device Inventory Database is one cornerstone of this model.

As a device progresses through its lifecycle, Google keeps track of changes made to the device. This information is monitored, analyzed, and made available to other parts of BeyondCorp. Because Google has multiple inventory databases, we use a meta-inventory database to amalgamate and normalize device information from these multiple sources, and to make the information available to downstream components of BeyondCorp. With this meta-inventory in place, we have knowledge of all devices that need to access our enterprise.

Device identity

All managed devices need to be uniquely identified in a way that references the record in the Device Inventory Database. One way to accomplish this unique identification is to use a device certificate that is specific to each device. To receive a certificate, a device must be both present and correct in the Device Inventory Database. We store the certificate on a hardware or software trusted platform module (TPM) or a qualified certificate store. A device qualification process validates the effectiveness of the certificate store, and only a device deemed sufficiently secure can be classed as a managed device. These checks are also enforced as certificates and are renewed periodically. Once installed, the certificate is used in all communications to enterprise services. While the certificate uniquely identifies the device, it does not single-handedly grant access privileges. Instead, it is used as a key to a set of information regarding the device.

Securely identifying the user

BeyondCorp also tracks and manages all users in a User Database and a Group Database. This database system tightly integrates with Google's HR processes that manage job categorization, usernames, and group memberships for all users.

An externalized, single sign-on (SSO) system is a centralized user authentication portal that validates primary and second-factor credentials for users requesting access to our enterprise resources. After validating against the User Database and Group Database, the SSO system generates short-lived tokens that can be used as part of the authorization process for specific resources.

Externalizing applications and workflows: the Access Proxy

All enterprise applications at Google are exposed to external and internal clients via an internet-facing Access Proxy (AP) that enforces encryption between the client and the application. The AP is configured for each application and provides common features such as global reachability, load balancing, access control checks, application health checks, and denial-of-service protection. This proxy delegates requests as

appropriate to the backend application after the access control checks (described in the next section) complete. See "Leveraging and Extending the GFE" on page 225 for more details about AP features.

Implementing inventory-based access control

The level of access given to a single user and/or a single device can change over time. By interrogating multiple data sources, we are able to dynamically infer the level of trust to assign to a device or user. The Access Control Engine (described in more detail next) can then use this trust level as part of its decision process, as in the following examples:

- A device that has not been updated with a recent OS patch might be relegated to a reduced level of trust.
- A particular class of device, such as a specific model of phone or tablet, might be assigned a particular trust level.
- A user accessing applications from a new location might be assigned a different trust level.

We use both static rules and heuristics to ascertain these levels of trust. An Access Control Engine within the Access Proxy provides service-level authorization to enterprise applications on a per-request basis. The authorization decision takes several factors into account:

- Information about the user, the groups to which the user belongs, the device certificate, and artifacts of the device, as reported by the Device Inventory Database
- The inferred level of trust in the user and the device
- If necessary, the Access Control Engine can also enforce location-based access control

For example, the following policies are possible with the Access Control Engine:

- Restrict access to Google's bug tracking system to full-time engineers using an engineering device.
- Restrict access to a finance application to full-time and part-time employees in the finance operations group using managed nonengineering devices.

The Access Control Engine can also restrict parts of an application in different ways. For example, viewing an entry in our bug tracking system might require less strict access control than updating or searching the same bug tracking system.

Leveraging and Extending the GFE

A conventional approach might integrate each backend with the device trust inference service in order to evaluate applicable policies; however, this approach would significantly slow the rate at which we're able to launch and change products. Instead, Google implemented a centralized policy enforcement frontend AP to handle coarse-grained company policies.

BeyondCorp leverages the existing Google Front End (GFE) infrastructure as a logically centralized point of access for policy enforcement. Funneling requests in this manner led us to naturally extend the GFE to provide other features, including self-service provisioning, authentication, authorization, and centralized logging. The resulting extended GFE is called the AP. The following section details the features of the AP that are particularly pertinent to this case study. For details about its other features, see "Beyond Corp: The Access Proxy".

The GFE provides some built-in benefits, such as load balancing for the backends and TLS management, that weren't designed specifically for BeyondCorp. The AP extends the GFE by introducing authentication and authorization policies.

User authentication

In order to properly authorize a request, the AP needs to identify the user and the device making the request. Authenticating the device poses a number of challenges in a multiplatform context, which we address in "Challenges with Multiplatform Authentication" on page 227.

The AP verifies user identities by integrating with Google's identity provider (IdP). Because it isn't scalable to require backend services to change their authentication mechanisms in order to use the AP mechanism, the AP needs to support a range of authentication options: OpenID Connect, OAuth, and some custom protocols. The AP also needs to handle requests without user credentials, for example, a software management system attempting to download the latest security updates. In these cases, the AP can disable user authentication.

When the AP authenticates the user, it strips the credential before sending the request to the backend. Doing so is essential for two reasons:

- The backend can't replay the request (or the credential) through the Access Proxy.
- The proxy is transparent to the backends. As a result, the backends can implement their own authentication flows on top of the Access Proxy's flow, and won't observe any unexpected cookies or credentials.

Authorization

Two design choices drove our implementation of the authorization mechanism:

- A centralized access control list (ACL) engine queryable via remote procedure calls (RPCs)
- A domain-specific language to express the ACL that is readable and extensible

Providing ACL evaluation as a service enables us to guarantee consistency across multiple frontend gateways (e.g., AP or the RADIUS network access control infrastructure, or Remote Authentication Dial-In User Service, and SSH, or Secure Shell, proxies). We chose to combine coarse-grained, centralized authorization at the AP with fine-grained authorization at the backend.

Mutual authentication between the proxy and the backend

Because the backend delegates access control to the frontend, it's imperative that the backend can trust that the traffic it receives has been authenticated and authorized by the frontend. This is especially important since the AP terminates the TLS handshake, and the backend receives an HTTP request over an encrypted channel.

Meeting this condition requires a mutual authentication scheme capable of establishing encrypted channels—for example, you might implement mutually authenticated TLS authentication and a corporate public key infrastructure. Our solution is an internally developed authentication and encryption framework called LOAS (Low Overhead Authentication System) that bidirectionally authenticates and encrypts all communication from the proxy to the backends.

One benefit of mutual authentication and encryption between the frontend and backend is that the backend can trust any additional metadata inserted by the AP (usually in the form of extra HTTP headers). While adding metadata and using a custom protocol between the reverse proxy and the backends isn't a novel approach (for example, see the Apache JServ Protocol), the AP's mutual authentication scheme ensures that the metadata is not spoofable.

As an added benefit, we can also incrementally deploy new features at the AP, which means that consenting backends can opt in by simply parsing the corresponding headers. We use this functionality to propagate the device trust level to the backends, which can then adjust the level of detail served in the response.

Challenges with Multiplatform Authentication

At minimum, performing proper device identification requires two components:

- Some form of device identifier
- An inventory database tracking the latest known state of any given device

Because BeyondCorp replaces trust in the network with an appropriate level of trust in the device, each device must have a consistent, non-cloneable identifier, while information about the software, users, and location of the device must be integrated in the inventory database.

Desktops and laptops

Desktops and laptops use an X.509 machine certificate and a corresponding private key stored in the system certificate store. Key storage, a standard feature of modern operating systems, ensures that command-line tools (and daemons) that communicate with servers via the AP can be consistently matched against the correct device identifier. Since TLS requires the client to present a cryptographic proof of private key possession, this implementation makes the identifier non-spoofable and non-cloneable, assuming it's stored in secure hardware such as a trusted platform module (TPM).

Mobile devices

Instead of relying on certificates, we use a strong device identifier natively provided by the mobile operating systems. For iOS devices, we use the identifierForVendor, while Android devices use the device ID reported by the Enterprise Mobility Management application.

Migrating to BeyondCorp

Like virtually every other enterprise in the world, Google maintained a privileged network for its clients and applications for many years. This paradigm gave rise to significant infrastructure that is critical to the day-to-day workings of the company. While all components of the company will migrate to BeyondCorp, moving every network user and every application to the BeyondCorp environment in one fell swoop would be incredibly risky to business continuity. For that reason, Google has invested heavily in a phased migration that has successfully moved large groups of network users to BeyondCorp with zero effect on their productivity.

Deploying an unprivileged network

To equate local and remote access, BeyondCorp defines and deploys an unprivileged network that very closely resembles an external network, although within a private address space. The unprivileged network only connects to the internet, limited infrastructure services (e.g., DNS, Domain Name Service, DHCP, Dynamic Host Configuration Protocol, and NTP, Network Time Protocol), and configuration management systems such as Puppet. All client devices are assigned to this network while physically located in a Google building. There is a strictly managed access control list between this network and other parts of Google's network.

Workflow qualification

All the applications used at Google are required to work through the AP. The BeyondCorp initiative examined and qualified all applications, which accomplish tasks ranging from the simple (e.g., supporting HTTPS traffic) to the more difficult (e.g., SSO integration). Each application required an AP configuration and, in many cases, a specific stanza in the Access Control Engine. Each application went through the following phases:

1. Available directly from the privileged network and via a VPN connection externally.
2. Available directly from the privileged network and via the AP from external and unprivileged networks. In this case, we used split DNS. The internal name server pointed directly at the application, and the external name pointed at the AP.
3. Available via the AP from external, privileged, and unprivileged networks.

Cutting back on VPN usage

As more and more applications became available via the Access Proxy, we started actively discouraging users from using the VPN, employing the following strategy:

1. We restricted VPN access to users with a proven need.
2. We monitored use of the VPN and removed access rights from users who did not use the VPN over a well-defined period.
3. We monitored the VPN usage for active VPN users. If all of their workflows were available through the AP, we strongly encouraged users to give up their VPN access rights.

Traffic analysis pipeline

It was very important that we moved users to the unprivileged network only when we were certain (or very close to certain) that all of their workflows were available

from this network. To establish a relative degree of certainty, we built a traffic analysis pipeline. Our analysis proceeded as follows:

1. As input to this pipeline, we captured sampled netflow data from every switch in the company.
2. We analyzed this data against the canonical ACL between the unprivileged network and the rest of the company's network. This analysis allowed us to identify the total traffic that would have passed the ACL, plus an ordered list of traffic that would not have passed the ACL.
3. We could now attach the non-passing traffic to specific workflows and/or specific users and/or specific devices.

We progressively worked through the list of non-passing traffic to make it function in the BeyondCorp environment.

Unprivileged network simulation

To augment the traffic analysis pipeline, we also simulated unprivileged network behavior across the company via a traffic monitor that we installed on all user devices attached to Google's network. The traffic monitor examined all incoming and outgoing traffic on a per-device basis, validated this traffic against the canonical ACL between the unprivileged network and the rest of the company's network, and logged the traffic that did not pass the validations. The monitor had two modes:

Logging mode
: Captured the ineligible traffic, but still permitted said traffic to leave the device

Enforcement mode
: Captured and dropped the ineligible traffic

Migration strategy

With the traffic analysis pipeline and the unprivileged simulation in place, we defined and began implementing a phased migration strategy that entails the following:

1. Identifying potential sets of candidates by job function and/or workflow and/or location.
2. Operating the simulator in logging mode, identifying users and devices that have >99.9% eligible traffic for a contiguous 30-day period.
3. Activating simulator enforcement mode for users and devices that have >99.99% eligible traffic for that period. If necessary, users can revert the simulator to logging mode.

4. After operating the simulator in enforcement mode successfully for 30 days, recording this fact in the device inventory.
5. Along with inclusion in the candidate set, successful operation in the simulator's enforcement mode for 30 days provides a very strong signal that the device should be assigned to the unprivileged network.

Exemption handling

In addition to automating the migration of users and devices from our privileged to our new unprivileged network as much as possible, we also implemented a simple process for users to request temporary exemptions from this migration:

- We maintained a known list of workflows that were not yet qualified for BeyondCorp. Users could search through these workflows, and with the correct approval levels, mark themselves and their devices as active users of a certain workflow.
- When the workflow was eventually qualified, its users were notified and were again eligible to be selected for migration.

Lessons Learned

The migration to BeyondCorp came with a set of challenges and kinks to be ironed out along the way. Hopefully the following lessons can save some time and headaches for other organizations seeking to implement a similar model.

Communication

Fundamental changes to the security infrastructure can potentially adversely affect the productivity of the entire company's workforce. It's important to communicate the impact, symptoms, and available remediation options to users, but it can be difficult to find the balance between over-communication and under-communication.

Under-communication results in the following problems:

- Surprised and confused users
- Inefficient remediation
- Untenable operational load on the IT support staff

Over-communication is also problematic:

- Change-resistant users tend to overestimate the impact of changes and attempt to seek unnecessary exemptions.
- Users can become inured to potentially impactful changes.

- As Google's corporate infrastructure is evolving in many unrelated ways, it's easy for users to conflate access issues with other ongoing efforts, which also slows remediation efforts and increases the operational load on support staff.

Engineers need support

Transitioning to a new network security paradigm doesn't happen overnight, and requires coordination and interaction among multiple teams. At a large enterprise scale, it's impossible to delegate the entire transition to a single team. The migration will likely involve some backward-incompatible changes that need sufficient management support.

In our experience, the success of the transition largely depended on how easy it was for teams to successfully set up their service behind the Access Proxy. Making the lives of developers easier should be a primary goal, so keep the number of surprises to a minimum. Provide sane defaults, create walkthrough guides for the most common use cases, and invest in documentation. Provide sandboxes for the more advanced and complicated changes—for example, you can set up separate instances of the Access Proxy that the load balancer intentionally ignores but that developers can reach (e.g., temporarily overriding their DNS configuration). Sandboxes have proven extremely useful in numerous cases, like when we needed to make sure that clients would be able to handle TLS connections after major changes to the X.509 certificates or to the underlying TLS library.

Data quality and correlation

Poor data quality in asset management can cause devices to unintentionally lose access to corporate resources. Typos, transposed identifiers, and missing information are common. Such mistakes may happen when procurement teams receive asset shipments and add the assets to our systems, or may be due to errors in a manufacturer's workflow. Data quality problems also originate quite frequently during device repairs, when physical parts or components of a device are replaced or moved between devices. Such issues can corrupt device records in ways that are difficult to fix without manually inspecting the device.

The most effective solutions in this arena have been to find local workflow improvements and automated input validation that can catch or mitigate human error at input time. Double-entry accounting helps, but doesn't catch all cases. However, the need for highly accurate inventory data in order to make correct trust evaluations forces a renewed focus on inventory data quality. The accuracy of our data is at previously unseen levels, and this precision has had secondary security benefits. For example, the percentage of our fleet that is updated with the latest security patches has increased.

Sparse data sets

Upstream data sources don't necessarily share overlapping device identifiers. To enumerate a few potential scenarios:

- New devices might have asset tags but no hostnames.
- The hard drive serial number might be associated with different motherboard serials at different stages in the device lifecycle.
- MAC addresses might collide.

A reasonably small set of heuristics can correlate the majority of deltas from a subset of data sources. However, in order to drive accuracy closer to 100%, you need an extremely complex set of heuristics to account for a seemingly endless number of edge cases. A tiny fraction of devices with mismatched data can potentially lock hundreds or even thousands of employees out of applications that they need to be productive.

Conclusion

Fortunately, an organization seeking to implement a zero trust network strategy today does have resources at hand to bootstrap this process. While this journey will by no means be trivial, there are a number of enterprise and commercial solutions available in this arena, and we hope that the rough blueprint outlined in this case study is helpful as you contemplate potential approaches. Keep the core motivations and design principles outlined here in mind while weighing your options and choosing the optimal security strategy for your needs.

Case Study: PagerDuty's Cloud-Agnostic Network

Evan Gilman and Doug Barth

PagerDuty began building a zero trust network in 2013, and completed it in 2014. It has continued to evolve, and remains in production as of this writing. The authors would like to thank PagerDuty for its permission to use its name and describe some of the details behind its zero trust implementation. All opinions are those of the authors, and PagerDuty is not at fault for errors or inaccuracies contained herein.

PagerDuty is a platform that organizations use to power their incident response. Users are able to integrate their existing tools like monitoring, ticketing, and reporting systems using PagerDuty's API. Most users first configure their monitoring systems to route alerts through PagerDuty so PagerDuty can manage on-call rotations and escalations. Given the critical nature of the service being provided, a zero trust network was ideal to meet both the reliability and data privacy requirements of that system.

PagerDuty's zero trust network primarily deals with server-to-server interactions purely within a multiprovider public cloud environment. Cloud providers have varying network control plane capabilities. Some providers give none of the controls that are normally required for a traditional perimeter system like a stateful firewall, private addressing, and network ACLs. In the most extreme case, hosts are placed onto the public internet and the host needs to secure itself. This disparity in provider capabilities makes running a provider-agnostic network exceptionally difficult when using traditional perimeter concepts.

PagerDuty's system also makes heavy use of WAN (wide area network) communication in its normal operation. Business-critical systems are deployed across three separate regions with the goal of surviving the loss of an entire region without impacting normal business operations. Relying on the WAN for normal application operation places some heavy requirements on the system. The internet is generally a challenging network environment with the potential for unexpected high latency and packet loss. In addition, communications need to be encrypted and authenticated to ensure data privacy and integrity. By deploying a perimeterless zero trust network, failure isolation is achieved since each node in the cluster is responsible for just its own communication.

Configuration Management as an Automation Platform

The key asset used to construct PagerDuty's zero trust network is its configuration management tool, Chef. Chef was already being used to configure every virtual machine in the system, and so it is a readily available automation layer which could be leveraged to build a zero trust network. With configuration management, policy can be centrally managed in code while distributing the enforcement into the entire fleet.

This approach has a number of benefits:

- Network compute power scales as the number of instances increases. This scaling property removes the need to buy ever larger shared hardware as the network grows.
- Failures tend to be more isolated. Instead of having "the firewall," the system ends up having many smaller firewalls. A failure of a single firewall affects a much smaller set of traffic and oftentimes can be routed around.

Distributing policy throughout the network isn't without its downsides:

- Constant validation of the expected policy state is required to ensure that all nodes are correctly enforcing the expected policy.

- Ensuring that changes to policies are consistent across the fleet. This can be a bit jarring if a system administrator expects to be able to make a change and see it take effect immediately.

While configuration management was an ideal place to quickly iterate on the zero trust ideas, it is not an ideal long-term solution. As these systems have become more mature, they have graduated out of Chef and into their own systems, which can be deployed and tuned for optimal performance.

Dynamically Calculated Local Firewalls

Without a consistent provider-supplied firewall solution, PagerDuty found it needed to ensure that each host was secured without relying on provider systems. To meet that need, Chef was taught how to generate IPtables configuration based on its existing knowledge of the system.

Servers in the system are categorized by their role, which captures the set of services and expected communication patterns that should exist for that role. Each server of a given role is identical in its configuration.

IPtables chains are constructed on each individual host that enumerates the IP addresses for servers of a particular role. These chains are then used to define the rules which allow expected access by role. If a flow does not match the whitelisted rules, its packets are dropped.

Here's an example of an IPtables configuration representing this arrangement:

```
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
target prot in out source destination
ACCEPT all lo * 0.0.0.0/0 0.0.0.0/0
ACCEPT all * * 0.0.0.0/0 0.0.0.0/0 state RELATED,ESTABLISHED
bastion tcp * * 0.0.0.0/0 0.0.0.0/0 tcp dpt:22
lb tcp * * 0.0.0.0/0 0.0.0.0/0 tcp dpt:80
lb tcp * * 0.0.0.0/0 0.0.0.0/0 tcp dpt:443
LOG all * * 0.0.0.0/0 0.0.0.0/0 limit: avg 10/min burst 5...
DROP all * * 0.0.0.0/0 0.0.0.0/0
Chain bastion (1 references)
target prot in out source destination
ACCEPT all * * 192.168.0.55 0.0.0.0/0
ACCEPT all * * 192.168.5.4 0.0.0.0/0
ACCEPT all * * 10.0.2.78 0.0.0.0/0
ACCEPT all * * 172.16.0.132 0.0.0.0/0
Chain lb (2 references)
target prot in out source destination
ACCEPT all * * 192.168.1.221 0.0.0.0/0
ACCEPT all * * 192.688.1.222 0.0.0.0/0=
```

Distributed Traffic Encryption

For network encryption and authentication, PagerDuty decided to implement an IPsec host-to-host mesh network. This network architecture has a number of benefits:

- All packets are encrypted and authenticated by every node in the system.
- Since encryption and authentication are distributed throughout the system, as the number of hosts grows, the capacity to provide these critical functions grows as well.

Network encryption and authentication are normally viewed as an application-level concern, but requiring every application to provide these safety controls results in a less secure or less operable system. Application encryption can have issues with correct implementation of the encryption specifications, lack the configuration controls to respond to security vulnerabilities, or introduce performance regressions into the system. For these reasons, PagerDuty decided to rely on the kernel's IPsec stack to provide this bit of critical infrastructure.

A system utilizing mutually authenticated TLS could provide similar benefits to an IPsec-based network. In order to provide the same guarantees, system administrators should separate the TLS infrastructure from the application.

Out-of-Process Encryption Is Increasingly Becoming the Standard

In many systems, encryption and authentication are considered an application concern, and applications usually provide this functionality using standard libraries. As the number of applications in systems has grown, systems are increasingly using out-of-process mechanisms for securing network communication.

By moving the encryption logic into a separate process, administrators gain a standard set of controls to use to respond to security vulnerabilities. In addition, having a separate process controlling the sensitive encryption process reduces the surface area for attacks that might expose secret data.

PagerDuty's network uses IPsec in transport mode. The phase 1 and phase 2 cipher suites use the strongest possible configuration available. When choosing the cipher suites, RFC 6379 was referenced to ensure that the algorithms chosen were recommended to be used together. IPsec communication is normally transmitted using ESP packets. Since some cloud providers' networks do not route ESP packets, all IPsec traffic is encapsulated in UDP packets.

PagerDuty's experience with operating an IPsec mesh network in production has been a bit mixed. The network has handled production throughput, and has grown with the fleet. During the initial rollout, communication failures did occur, often due to inconsistent state on either side of the IPsec relationship. Having metrics and logging to surface these issues was critical to operating the network. While having these failures was certainly frustrating, with a mesh network, these failures were isolated to pairs of hosts, which often reduced the impact of the failure.

PagerDuty's initial rollout of the IPsec network utilized Chef and some simple scripts to configure preexisting IPsec packages. As the network has grown, the configuration of the system has moved out of Chef and into a dedicated service that can handle the sole responsibility of configuring this aspect of the system. Moving the logic into its own system was done to lessen the convergence time for deploying a change to the network. The Chef-based system required running an entire Chef convergence run to update all relevant hosts in the network—a heavyweight operation that handles more than just the network configuration.

Decentralized User Management

PagerDuty's user access control is deployed in a centralized fashion, much like the networking systems previously discussed. Instead of relying on a centralized LDAP system, local users and groups are programmatically constructed on each host in the network. This approach removes a dependency on the network, which helps the system continue to operate even during challenging periods.

While the enforcement of user access control is distributed into the network, the definitions of which users and groups should be created is centralized. This information could be captured in an LDAP server or some other database. In PagerDuty's case, it used Chef data bags to define users and groups. Server roles are marked with the set of groups that should be created on that role. Chef uses this data to only create the users and groups on a particular server that need access to that infrastructure.

Rollout

PagerDuty's network, like most networks, is an ever-evolving system. The network transitioned from a traditional design to a zero trust network over time, while production traffic was flowing.

Changing a network architecture while critical production traffic is flowing can be difficult, so it was important that the rollout was planned to reduce risk. PagerDuty followed a slow rollout pattern:

1. New policies are defined.
2. Policies are deployed in a manner that does not affect the production system, but instead collects useful metrics or logs.

3. The metrics/logs are inspected over a long period of time to ensure that the behavior is desired.
4. The policy is slowly enabled across the fleet, growing from a small percentage to 100% coverage.

This simple procedure can be used to reduce the risk of most production changes. It is much better than the common approach of using a scheduled maintenance window.

The slow rollout pattern is used to deploy most changes in PagerDuty's systems. For the distributed firewall project, all hosts were initially configured to log packets, which would be dropped at a later date. Firewall rules were created to classify traffic flows, which could be deployed without the risk of blocking any production traffic. With the rules deployed, the logged traffic was reduced; and once enough time had passed, the system was reconfigured to drop all non-whitelisted traffic.

The distributed traffic encryption followed the same rollout procedure. IPsec policies were first deployed into the fleet in a no-op configuration. These policies control whether a particular traffic flow should use IPsec for communication. IPsec supports three different states:

None
: IPsec will not be used.

Use
: IPsec will be optimistically used if a relationship can be negotiated.

Required
: IPsec must be used for traffic to be processed.

The initial set of policies were deployed in the none state. The end goal was to get the entire system to the required state by stepping through the use state. Based on testing of the failure modes of the use state, it was determined that intermediate stateful firewalls would block communication if the IPsec relationships were broken, as packets would fall back to a none policy. These packets would not be associated with an expected flow (remember that previously they were encrypted and wrapped in a UDP encapsulation packet) and so would be dropped.

Instead of configuring the entire network to a use state, smaller portions of the network were transitioned to a use state and then reconfigured to a required state. This phased approach minimized the amount of time the network was in the potentially risky use state while still allowing hosts to communicate as they reconfigured themselves. Chef calculated the minimum policy between a pair of hosts based on their preferred state.

Value of a Provider-Agnostic System

It goes without saying that building a provider-agnostic system requires significant engineering effort. For many systems, this effort may not be justified. In PagerDuty's case, the business requirements determined that the effort was justified. Having this provider-agnostic network in place provided a significant return on investment when PagerDuty decided to move off one of its cloud providers. Normally an effort like this would be a several-month effort with many high-risk change windows.

In PagerDuty's case, this change was relatively straightforward. It took roughly six weeks from making the decision to having all production traffic moved over. The bulk of that time was spent researching new providers, testing the new provider's systems, and reworking the Chef automation. The actual changes were deployed to production in one week during normal business hours without any customer impact.

Summary

This chapter focused on the considerations that an organization that wants to move to a zero trust network needs to decide on. Where possible, it gave real-world recommendations to help readers through making these decisions.

We spent time discussing the importance of understanding the state of the system using system diagrams and capturing network flows from real production traffic. Building all the zero trust control plane systems as standalone services can be a large up-front investment, so practical alternatives were explored.

The most important detail to remember is that zero trust is an architectural ideal, so this chapter discussed how to get started down the path by defining and capturing policy in a manner that can be later reused. It explored putting in place authentication proxies that can incorporate systems that aren't directly compatible with zero trust. It also explored whether organizations should start with client/server interactions or server/server interactions.

Finally, to help readers see how this type of endeavor played out in other organizations' systems, this chapter explored two concrete case studies. These case studies explored the particular approaches and trade-offs that were made to make zero trust a reality in existing production networks.

The next chapter focuses on how a hypothetical attacker might try to thwart a zero trust network.

CHAPTER 10

The Adversarial View

The adversarial view assumes that all digital systems are susceptible to compromise and that malicious attackers will persistently attempt to breach them. By understanding this approach, we can assess the probability and ramifications of potential attacks, identify potential vulnerabilities, and ultimately build robust, resilient, and secure systems.

To effectively defend against potential breaches in a zero trust network, organizations must understand how attackers may attempt to bypass security measures. They must also proactively identify weak points to minimize the risk of a successful attack by identifying the entry points most likely to be targeted.

With increasingly sophisticated cyberattacks on the rise, organizations have turned to the zero trust model to protect their systems from malicious activities. While this approach provides greater protection against data breaches, organizations must be aware of potential pitfalls, risks, and attack vectors associated with this model.

In this chapter, we will explore the potential challenges that can arise related to the zero trust model in greater detail. If you were trying to penetrate a zero trust network, how might you do it?

Potential Pitfalls and Dangers

Implementing a zero trust model can present challenges in complexity, time, and cost.

Insufficiently secure authentication measures within organizations can be exploited by attackers, circumventing their effectiveness. Poorly configured authentication systems and policies may hinder user experiences and business productivity. Misconfigurations can introduce security vulnerabilities, such as unauthorized access due to

access protocol and authentication criteria misconfigurations or lack of auditing. Another concern is the limited visibility into user activity when relying solely on authentication measures, potentially leading to a false sense of security. Additionally, adversaries continue to employ social engineering and phishing attacks to bypass authentication protocols and gain unauthorized system access.

The zero trust model is not a silver bullet; attackers are constantly looking for ways to bypass implemented security controls.

One of the most significant risks associated with zero trust networks is the potential for exploiting software vulnerabilities or abusing system privileges to establish unauthorized access. We call these attack vectors.

Attack Vectors

There are a multitude of attack vectors that can be used to bypass a zero trust network. These include, but are not limited to, social engineering and credential theft, privilege escalation and lateral movement, distributed denial of service (DDoS) attacks, zero-day vulnerabilities, and application exploitation techniques. Table 10-1 summarizes many of these attacks, including countermeasures.

These attack vectors can potentially disrupt an organization's operations and ultimately compromise security. To better understand these threats, let's explore each attack vector in greater detail in the next section.

Table 10-1. Common attack vectors and suggested countermeasures

Category	Attack vector	Description	Countermeasure
Identity, Access, & Authentication	Insider Threats	Attacks by individuals with legitimate network access, exploiting privileges.	Continuous monitoring, behavioral analytics, access controls.
Identity, Access, & Authentication	Credential Theft and Reuse	Accessing the system using stolen or reused credentials.	Multifactor authentication, regular password updates, password complexity requirements.
Identity, Access, & Authentication	Credential Theft and Reuse	Using known username-password combinations.	Employ multifactor authentication and monitor for failed login attempts.
Cloud Security	Cloud Security	Challenges in securing cloud environments with zero trust.	Implement proper cloud configuration, identity and access management (IAM) policies, and use native cloud security tools.
Cloud Security	Misconfigurations	Incorrect configurations exposing data or granting excessive access.	Security audits, configuration management tools, automated vulnerability scanning.
Data Handling	Data Exfiltration	Techniques to steal sensitive data from the network.	Employ data loss prevention (DLP) tools and monitor outbound traffic.

Category	Attack vector	Description	Countermeasure
Device Trust	Untrusted Computing Platform	Foundational platform vulnerabilities.	Ensure devices meet security standards before granting access.
Device Vulnerabilities	Internet of Things Vulnerabilities	Many IoT devices lack robust security.	Regularly update IoT device firmware and change default credentials.
Exploits & Vulnerabilities	Zero-Day Vulnerabilities	Unknown vulnerabilities without available patches being exploited.	Regular updates, intrusion detection and prevention systems, network segmentation.
Infrastructure & Network Security	DDoS Attacks	Overwhelming a network or site with traffic.	Use advanced DDoS protection services and maintain diversified network resources.
Infrastructure & Network Security	Man-in-the-Middle (MitM Attacks)	Intercepting communications can compromise data.	Use end-to-end encryption and secure communication protocols.
Infrastructure & Network Security	Control Plane Security	Traffic manipulation by compromising control routing.	Secure and monitor control plane and utilize network monitoring tools.
Infrastructure & Network Security	Endpoint Enumeration	Identifying devices or users within a network.	Implement network cloaking, segment networks, and restrict responses to unidentified queries.
Social Engineering & Human Factors	Physical Coercion	Forcing individuals to perform unwanted actions through physical threats.	Physical security measures, awareness training, and duress codes.
Session & Data Handling	Invalidation	Bypassing security measures by invalidating tokens or sessions.	Time-limited sessions, prompt for re-authentication for critical actions.
Session & Data Handling	Session Hijacking	Attackers take over a valid user session.	Use encrypted sessions, regularly renew session tokens, and prompt for re-authentication for critical actions.
Social Engineering & Human Factors	Phishing and Social Engineering	Deceptive tactics tricking individuals into revealing sensitive data or actions.	Security awareness training, multifactor authentication, email security.
Third-Party Risks	Supply Chain Attacks	Compromising third-party vendors or software to gain access.	Vendor risk assessments, security audits, application-allowed listings.

Identity and Access

Within identity, there are two main types of threat: credential theft and privilege escalation.

Credential Theft

Practically all of the decisions and operations performed within a zero trust network are made on the basis of authenticated identity. In Chapter 6, we discussed the difference between informal and authoritative identity, such as the difference between your "human" identity and your government identity. Computer systems implement authoritative identity similarly to the way governments do—and similarly to the way

your government identity can be stolen, so can your identity within a computer system.

If your identity is stolen or compromised, it might be possible for an attacker to masquerade their way through the zero trust authentication and authorization checks. This is, of course, extremely undesirable. Since identity in a computer system is typically tied to some sort of "secret" that is used to prove said identity, it is extraordinarily important to protect those secrets as well as we can.

These secrets can be protected in different ways, based on the type of component the identity belongs to. Careful consideration should go into choosing which methods to use for which components. We discussed different ways to approach this problem in previous chapters.

Since a zero trust network authenticates both the device and the user/application, it is necessary for an attacker to steal at least two identities in order to gain access to resources within it, raising the bar compared to traditional approaches in use today. These concerns can be additionally mitigated through the use of trust engine behavioral analysis.

While securing identity is a widespread industry concern, and is not specific to zero trust, its importance is large enough to justify calling it out as something that should be carefully handled, despite the fact that the zero trust model works to naturally mitigate this threat.

Privilege Escalation and Lateral Movement

Privilege escalation is a common attack vector and refers to an attacker using various techniques, such as hijacking user sessions, abusing system privileges, malicious code injection, or exploiting misconfigurations in authentication systems, to elevate access to a user account with a higher level of privilege. One example could be an attacker gaining access to an admin account, or a standard user account with administrative access. The attacker could then leverage this privileged account to laterally move about and across the network. By gaining access via this attack vector, attackers can, as you can already imagine, cause serious damage and wreak havoc.

Scenario: Privilege Escalation and Lateral Movement—Simulated Attack

This scenario demonstrates a realistic attack vector wherein an attacker initially gains access to a cloud environment through a phishing attack and subsequently leverages compromised administrative credentials to pivot laterally into the organization's on-premises network, escalating privileges along the way.

Let's walk through this scenario using a fictional company called Bases Loaded, Inc. (BSL):

1. *Initial compromise:* The attacker, using social engineering tactics, sends a phishing email to an employee, Alex, at BSL. The email appears to be from the company's IT department, claiming an urgent security update and providing a link to a seemingly legitimate login page.
2. *Phishing attack:* Alex, unaware of the phishing attempt, clicks on the link and is directed to a convincing login page that mirrors BSL's official login portal. Believing it to be genuine, Alex enters their credentials, unwittingly providing the attacker with their username and password.
3. *Cloud account access:* Armed with Alex's credentials, the attacker gains access to Alex's cloud account. In this case, the attacker discovers that Alex has administrative privileges within BSL's cloud environment.
4. *Privilege escalation:* The attacker leverages administrative privileges to escalate their access. They identify an administrative cloud account linked to BSL's Azure Active Directory (Azure AD/Entra ID). By exploiting security misconfigurations or vulnerabilities within the cloud environment, the attacker successfully breaches the administrative cloud account belonging to a user named Jane, who has elevated privileges.
5. *Lateral movement to the on-premises network:* With control over Jane's administrative cloud account, the attacker explores the organization's resources, discovering that BSL uses a hybrid setup with on-premises Active Directory and cloud services.
6. *Exploiting the hybrid setup:* The attacker also discovers that BSL synchronizes its on-premises AD to its Azure AD/Entra ID. Using the administrative privileges acquired from Jane's cloud account, the attacker abuses the trust relationship to access and move laterally into BSL's on-premises network.
7. *Compromising the Active Directory:* Once inside the on-premises network, the attacker finds that they can compromise BSL's Active Directory, potentially gaining access to sensitive user data, corporate applications, and critical infrastructure.

BSL may have prevented this attack through security awareness training that would help the employees of BSL recognize phishing attempts by implementing phish-resistant multifactor authentication (MFA) for all accounts, using cloud-only accounts (accounts sourced only in the cloud and not synced back on premises) for cloud administration, conducting regular security audits, and/or maintaining up-to-date security configurations for both its cloud and on-premises environments.

Infrastructure and Networks

Infrastructure and networks are foundational in information technology and serve as the backbone for data flow and communication. These elements encompass the physical and virtual resources that facilitate connectivity, data transfer, and management across various platforms and devices. While crucial for seamless operations and connectivity, they also present unique security challenges.

Control Plane Security

Control plane security involves the protection of systems and processes that control the routing and forwarding of data, and plays a pivotal role in ensuring the safe and efficient functioning of network operations and safeguarding against unauthorized access and malicious attacks.

We discuss many control plane services throughout this book, responsible for things like policy authorization and tracking inventory. Depending on needs, a zero trust control plane can comprise a nontrivial number of services, all of which play a crucial role in ensuring authorization security throughout the network. A natural question follows: how can you protect your zero trust control plane systems, and what happens if one is compromised?

Well, it's not good, that's for sure! It is possible to completely undermine the zero trust architecture if a control plane compromise is pervasive enough. As such, it is absolutely critical to ensure the security of these systems. This is not a weakness unique to the zero trust model—it exists today even in perimeter networks. If your perimeter firewall is compromised, what is the impact? As such, the concern is great enough to warrant a discussion.

Scenario: Control Plane Security and the Implications of an Attack

Imagine our technology environment as a bustling airport. The control tower represents our control plane—the nerve center that manages and directs the movement of planes on the runways and in the airspace. This control tower holds immense power, determining which planes can take off, land, or taxi.

In our digital landscape, the control plane serves a similar purpose. It manages the access, permissions, and configurations of our systems and networks. When our control plane is secure, we can trust that only authorized individuals and processes have the permissions they need. It's like giving the right pilots access to the right planes at the right time.

However, consider the chaos that would ensue if the control tower at an airport were compromised. Unauthorized personnel could grant takeoff clearance to an aircraft without proper checks. Similarly, if our control plane is not adequately secured, it

becomes a potential target for attackers. They could exploit vulnerabilities to manipulate permissions, grant unauthorized access, or redirect critical processes.

The implications are far-reaching:

Data breaches
: An insecure control plane could lead to unauthorized access to sensitive data, putting customer information, financial data, and intellectual property at risk.

Service disruption
: Just as a compromised control tower could disrupt flights, an attacker manipulating the control plane might disrupt our services, causing downtime and financial losses.

Lateral movement
: Without proper control plane security, attackers could pivot from one area of our environment to another, escalating their access privileges and infiltrating deeper into our network.

Regulatory non-compliance
: Depending on our industry, insufficient control plane security could lead to compliance violations, resulting in hefty fines and reputational damage.

By prioritizing control plane security, we are fortifying our organization's control tower by ensuring that only authorized processes and individuals can access our critical systems and data. We are also minimizing the risk of unauthorized changes and manipulations that could lead to substantial breaches.

Control plane security can begin through traditional means, providing very limited network connectivity and strict access control. Some control plane systems are more sensitive than others. For instance, compromising a data store housing historical access data is, strictly, less useful to an attacker than compromising the policy engine. In the former case, an attacker may be able to artificially raise their level of trust by falsifying access patterns, whereas the latter leads to a complete compromise of zero trust authorization, allowing the attacker to authorize anything they please.

For the most sensitive systems (i.e., the policy engine), rigorous controls should be applied from the beginning. Requiring group authentication and authorization in order to make changes to these systems is a real option and should be heavily considered. Changes should be infrequent and should generate broadly seen messages or alerts. It should not be possible for a control plane change to go unnoticed.

Another good practice is to keep the control plane systems isolated from an administrative standpoint. Perhaps that means they live in a dedicated cloud provider account or are kept in a part of the datacenter that has more rigorous access control. Doing this allows access to be more carefully audited and minimizes the risk presented

to control plane systems by their administrative facilities. Isolating these systems administratively does not mean that they are logically isolated from the rest of the network.

Despite administrative isolation, it is important that control plane systems participate in the network just as any other service does. Attempts to isolate them can quickly lead back to a perimeterized design, which can be considered the worst-case scenario for zero trust control plane security.

As the network matures, zero trust enforcement can be slowly applied to the control plane systems themselves. Kind of like rewriting the C compiler in C, backing zero trust enforcement into the control plane ensures that tight security is applied homogeneously throughout the network and that there are no special cases. The propensity to introduce a chicken-and-egg problem should not deter you from this approach. Such problems are manageable and can usually be worked through if sufficient thought is put into them. The alternative (putting control plane systems in a perimeter network) would leave these systems the least protected of all, and is generally unacceptable in the context of a zero trust network.

Endpoint Enumeration

The zero trust model lends itself naturally to perimeterless networks since a perimeter makes much less sense when the internal network is untrusted. The peer-to-peer nature of perimeterless networks make them generally easier to maintain than perimeter networks, which frequently include network gateways and tunnels like VPNs which pose scaling, performance, and availability challenges.

As a result of this architecture, it is possible for an adversary to build a system diagram by observing which systems talk to which endpoints. This is in contrast to architectures that leverage network gateways like VPNs, since an adversary observing VPN traffic can't see conversations with endpoints beyond the VPN gateway. It should be noted that this advantage is lost as soon as the traffic crosses the gateway—a classic property of the perimeter model.

It is here that we make a distinction between privacy and confidentiality. The zero trust model guarantees network confidentiality, but not privacy. That is, ongoing conversations can be observed and asserted to exist; however, the contents of the conversation are protected. Systems that provide network privacy attempt to obscure the fact that the conversation happened at all. Tor is a popular example of a system that provides network privacy. This is a wholly different problem space and is considered out of scope for the zero trust model.

If a limited form of privacy over public networks is desired, tunneling traffic through site-to-site tunnels is still an option in zero trust networks. This deployment will make it more difficult to see which individual hosts are communicating on either side

of the tunnel. We should be clear that this additional privacy protection should not be considered critical to the network's security. In fact, in some ways it undermines the zero trust model itself, as hiding information in one part of the network and not another suggests that one is more trusted than the other.

Untrusted Computing Platform

We covered this in Chapter 5, but it's important to reiterate that zero trust networks require the underlying computing platform to be a trustworthy system. There's a distinction to be made here between the computing platform itself (think cloud hardware, virtual machine hypervisor) being trusted and the "device" being trusted. Oftentimes these two systems are conflated, but the attacks against each are subtly different due to their differing privilege levels.

Totally defending against untrustworthy computing platforms is practically impossible. Consider a system that uses hardware that purposefully generates weak random numbers (which encryption systems depend on). Defending against that type of attacker would first involve detecting the problem, though this alone might be impossible if the attacker hides their capability most of the time.

Despite our inability to guard against a truly malicious, untrusted computing platform, zero trust systems can still guard against simpler attacks against the platform. Encrypting persistent data and swapped-out memory pages will mitigate simpler attacks by malicious peers on the computing platform. It will also obviate the need for trust in the platform's operators and, therefore, is recommended.

Distributed Denial of Service (DDoS) Attacks

A zero trust network is primarily concerned with authentication, authorization, and confidentiality, all generally affected by tight control of access to all network resources. While the architecture strives to authenticate and authorize just about everything on the network, it does not provide good mitigation against denial-of-service (DoS) attacks on its own. Distributed DoS (DDoS) attacks that are volumetric in nature can be particularly troublesome.

Just about any system that can receive packets is vulnerable to volumetric DDoS, even those employing zero trust architecture. Some implementations "darken" internet-facing endpoints through the use of pre-authentication protocols. We spoke a little about these in "Bootstrapping Trust: The First Packet" on page 171, the basic premise being to hide those endpoints behind a deny-all rule, adding narrow exceptions based only on signaling. While this method goes a long way in helping to keep the endpoint addresses obscured, it does not fundamentally mitigate DDoS attacks.

A zero trust network, by nature, retains a great deal of information about what to expect on the network. This information can be used to calculate policy for

more traditional traffic filtering defenses far upstream. For instance, perhaps only a few systems in the network actually communicate with the internet. In this case, we can use the policy to calculate coarse enforcement rules from the perspective of an upstream device, applying very broad enforcement with few exceptions. The advantages of this approach over the typical approach are twofold:

- The configuration is fully automated.
- The traffic filtering mechanisms can remain stateless.

The second advantage is quite a large one, since it obviates the need for expensive hardware and complicated state replication schemes. In this way, these filtering devices act more like scrubbers than firewalls. Of course, this only makes sense if you operate a large network. If you have a few racks in a co-location facility, or are cloud native, you might prefer to leverage an online DDoS-prevention service.

The short of it is, DDoS is still a problem in the zero trust world, and while we might have a few new clever ways to address it, it will still require special attention.

Man-in-the-Middle (MitM) Attacks

Man-in-the-Middle (MitM) attacks pose a significant challenge in cybersecurity. While zero trust networks prioritize strong authentication, careful authorization, and strict control over network resource access, they may not offer solid protection against MitM attacks.

MitM attacks exploit vulnerabilities in communication channels, allowing attackers to intercept and manipulate data between two parties without detection. Even zero trust environments can be compromised, which means extra precautions are required when using VPNs, cloud services, and remote access tools, as these technologies are prone to MitM attacks. While zero trust networks do their best to hide endpoints and closely manage network actions, these measures might only partially stop a sophisticated attack.

The good news is that the insights gathered from these networks can both make our defenses stronger and be used to find efficient ways of sorting through incoming traffic. Think of it this way: networks that follow the zero trust approach show patterns in how they communicate, so we can then leverage these patterns to set clear rules on device communication, minimizing the risks of MitM attacks.

This approach has two benefits: automatic rule configuration and the ability to use stateless traffic filtering. This is important because it means that we don't need to keep track of everything or use expensive hardware. The one caveat is that this solution is better suited to large networks. If the network environment involves limited infrastructure or cloud-based operations, an online MitM solution might be better.

While innovative approaches can potentially mitigate MitM attacks, unique strategies and vigilance are also necessary. Organizations must continue to invest in technologies that detect suspicious activity, implement strong authentication protocols, and regularly evaluate endpoints for vulnerabilities to minimize the risk of malicious actors accessing restricted information.

More information on MitM may be found here.

Invalidation

Invalidation is a hard problem in computer science. In the context of a zero trust network, invalidation applies chiefly to long-running actions that were previously authorized but are no longer.

The definition of an action is largely dependent on your chosen authorization processes. For instance, if you authorize access on a request-by-request basis, an action would be considered to be a single application-level request/operation. If, on the other hand, you authorize network flows (like a TCP session) instead of application requests, an action would be considered to be a single network session.

How quickly and effectively ongoing actions can be invalidated deeply affects security responses. It is important to gauge how much risk you're willing to tolerate in this area as you design your zero trust network, since the answer has the potential to significantly affect how you might approach certain problems. For instance, if a new TCP session is the action being authorized, and some services maintain TCP sessions for multiple days on end, is it acceptable to say that an entity with revoked credentials might retain access for that long? Maybe not.

Luckily, we have some tools in our chest to address this problem. First, and perhaps most obvious, is to perform more granular authorizations on actions that are short-lived. Perhaps this means that the enforcement component authorizes application-level requests instead of new network sessions. While it is still possible to have long-running application requests, they are in practice less frequent than long-running network sessions.

Another approach, although it is somewhat naive, is to periodically reset network sessions, enforcing a maximum lifetime. When the application/client reconnects, it will be forced back through the authorization process.

The best approach, though, is to teach the enforcement component to track ongoing actions, and rather than reset them after a period of time, send another authorization request to the policy engine. If the policy engine decides that the action is now unauthorized, the enforcement component can forcibly reset it.

As you can see, these mechanisms still rely on a "pull" model, in which the enforcement component is forced to periodically reauthorize. As a result, sessions can only be invalidated as fast as the longest polling period configured in the enforcement component. While invalidation is best done as a push or event-based model, those approaches come with additional complexities and challenges that perhaps outweigh the benefits. Regardless, it can be seen that the problem is (at the very least) addressable.

Phishing

Social engineering attacks, which trick trusted humans into taking action on a trusted device, are still very much a concern in zero trust networks. Whether they are phishing attacks, which craft written communication that is not obviously malicious, or they take place via face-to-face communications like those that customer service departments have had to deal with, a zero trust network can only do so much to defend against attacks enabled by an unwitting participant.

For less sensitive resources, behavioral analysis of internal activity is the mechanism used to guard against this threat. That analysis is coupled with end-user training that teaches users to think like an adversary and be suspicious of requests that are out of the ordinary.

For more sensitive resources, group authentication/authorization schemes like Shamir's Secret Sharing can help mitigate the effects of a single member of the group causing unintended actions to occur. This scheme can be very burdensome on a day-to-day basis, so the best plan is to save it for the truly critical assets. Chapter 6 has more details on these mechanisms for defending against social engineering attacks.

Physical Coercion

Zero trust networks effectively mitigate many threats in the virtual world, but threats in the real world are another beast entirely. Valid users and devices can be effectively coerced to aid an attacker to gain access to a system that they shouldn't have access to. Border crossing can often be a place where government entities have substantial power over an individual who just wants to get to their destination. And someone with a blunt instrument can force even the most honest individuals to aid them (as demonstrated in Figure 10-1).

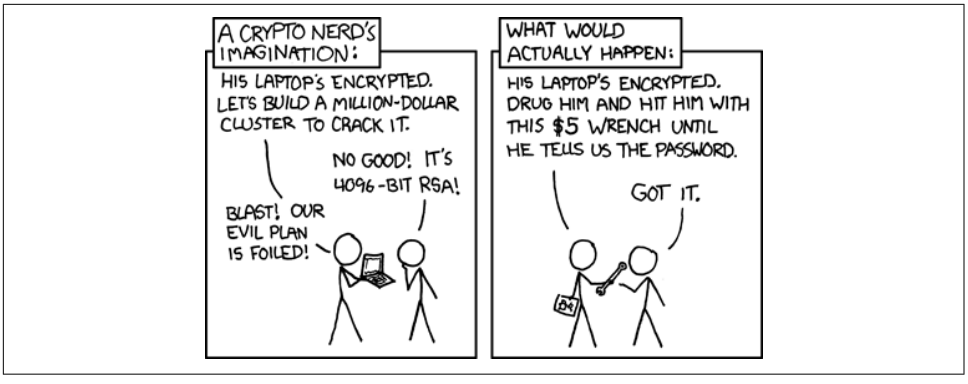

Figure 10-1. The reality of threats in a system (cartoon by XKCD)

The reality is that relying on individuals defending themselves against these types of compromises in the moment is ill-advised. No security professional would ever tell someone in this situation to risk their physical well-being to protect the information that they have access to. Therefore, the best we can work toward as an industry is to keep only the least sensitive data and systems vulnerable to the compromise of a single individual. For higher-value targets, group authorization is an effective mitigation against these threats.

Subtler physical attacks against individuals (say someone is able to insert a USB device into an unguarded laptop) are best mitigated by a consistent process of cycling both devices and credentials. Scanning of unrotated devices can also help to mitigate these types of attacks.

If someone has physical access to your device, they can do a lot of damage. However, that statement should not be license to throw our hands up in the air and not at least try to mitigate these threats, particularly when it comes to securing data used for zero trust authentication/authorization. There are clear steps that can be taken to lessen the impact and duration of compromise, even if someone has physical access to a device, and zero trust networks add those steps. You can read more about physical device security in Chapter 5.

Role of Cyber Insurance

Cybersecurity insurance, often known as cyber liability insurance, is a contract that a company can buy to help decrease the financial risks of conducting business online. From a risk perspective, cyber insurance may be used to mitigate risk when implementing zero trust or any other security initiative. The insurance policy transfers some of the risk to the insurer in exchange for a monthly or quarterly charge. Cyber insurance evolved from errors and omissions (E&O) insurance, a separate type of

insurance that protects against flaws and deficiencies in a company's services. In a nutshell, cyber insurance provides relief in the following circumstances:

- When an incident occurs, it acts as a financial safety net for the company.
- It provides customers with assurance that a cyber incident will not force the company out of business.
- It protects against regulatory penalties and third-party legal consequences, within reason.

The insured entity's annual revenue and industry determine cyber insurance pricing. The insurance company usually requires a security audit or paperwork using an approved evaluation tool, such as that of the Federal Financial Institutions Examination Council (FFIEC). Many cybersecurity insurance rules exclude human-caused security vulnerabilities, such as poor configuration management or negligent handling of digital assets. For more information on cyber insurance and its market, please read the report on the cyber insurance market, published by the National Association of Insurance Commissioners (NAIC).

Summary

This chapter attempts to approach the zero trust network from the opposite perspective of the administrators of the system. By putting ourselves into the mindset of a would-be attacker, we can evaluate the system as an adversary who has vast knowledge of how it is put together.

Some of the attacks against zero trust networks are well mitigated, whereas for others we are only able to detect the attack, at best. Even a zero trust network can be compromised by a determined adversary, as the inconvenience of defending against any theoretical attack is simply too high a price to pay in the day-to-day operation of such a network.

The reality is that every system is susceptible to an attacker with sufficient resources. When faced with the most advanced attacks, the best we can hope for is efficient and accurate detection. Starting from the assertion that a system has been compromised and working our way backward toward limiting the damage is a sage strategy that might allow us to sleep soundly.

While the zero trust model certainly introduces some new consideration points with regard to networked system security, at the same time, it resolves many more. By applying the power of automation to tried-and-true security primitives and protocols, the authors are confident that the zero trust model will rise to replace the perimeter model as a more effective, scalable, and secure solution to the computer network security problem.

CHAPTER 11

Zero Trust Architecture Standards, Frameworks, and Guidelines

Stephen Paul Marsh originally coined the term "zero trust" in his April 1994 Ph.D. dissertation on computer security, where he mathematically defined trust and also claimed that the idea of trust transcends a variety of human traits like morality, ethics, etc. However, it was not until the Forrester report published in November 2010 that the term zero trust was defined and articulated within the context of the zero trust security paradigm that we are familiar with today. Since the publication of that report, a lot has changed in the digital world: we've seen widespread adoption of cloud computing, a massive shift toward digitization accelerated by a surge in remote work during and after the COVID-19 pandemic, and the ubiquitous presence of mobile phones and social media in our daily lives. Also, artificial intelligence has evolved from a distant promise to a reality and has become an enormous disruptor to both businesses and individuals.

As a result of technological advances, we are becoming more interconnected, which has many advantages such as improved communications and quicker access to resources, but it has the unintended security consequence of increasing the attack surface area for malicious actors to exploit. This is evident from the FBI Internet Crime Report, which shows that security and data-related crimes in 2022 caused losses in excess of 10 billion USD in the United States alone. As a result, organizations have to pivot away from conventional security models such as perimeter-based security and adopt a security philosophy and principles based on zero trust to strengthen their security posture. Gartner, a prominent research and advisory firm, has estimated that by 2026, 10% of large enterprises will have a mature and measurable zero trust program.

This chapter will discuss a variety of zero trust frameworks, standards, and guidelines published by government organizations, standardization bodies, and private, public, and nonprofit organizations. The goal is to provide foundational knowledge and awareness of the prevalent zero trust architectures, designs, and vocabulary that you will likely encounter while working on zero trust initiatives.

Finally, while a concerted effort to ensure that coverage of zero trust publications remains as thorough and current as possible at the time of writing, it is virtually impossible to include every publication due to the rapid pace of development in the area of zero trust.

Governments

This section is focused on various zero trust publications released by government organizations across the globe. These publications not only play a vital role in the national policy for cybersecurity as a whole, but they also have an influence on organizations in the private and public sectors. Table 11-1 shows a list of artifacts published on zero trust by various governmental organizations worldwide.

Please keep in mind that this collection represents a broad overview of a dynamic and growing field. Consider it a starting point, and use the resources listed below to continue to learn about ongoing research.

Table 11-1. Zero trust publications from various government organizations worldwide

Government organization	Country	Publication
National Institute of Standards and Technology (NIST)	United States	"Zero Trust Architecture" (NIST SP 800-207)
National Cybersecurity Center of Excellence (NCCoE)	United States	"NIST Cybersecurity Practice Guide SP 1800-35 Vol C–D (Implementing a Zero Trust Architecture)" - Draft
Department of Defense (DoD)	United States	"Department of Defense (DoD) Zero Trust Reference Architecture"
National Security Agency	United States	"Embracing a Zero Trust Security Model"
Cybersecurity & Infrastructure Security Agency (CISA)	United States	"Zero Trust Maturity Model"
National Institute of Standards and Technology (NIST)	United States	"Planning for a Zero Trust Architecture: A Planning Guide for Federal Administrators"
National Cyber Security Centre	United Kingdom	"Zero trust architecture design principles"
Agence nationale de la sécurité des systèmes d'information (ANSSI)	France	"Le modèle Zero Trust"
Singapore Government Developer Portal	Singapore	"Government Zero Trust Architecture" (GovZTA)
Canadian Centre for Cyber Security	Canada	"Zero Trust security model"—ITSAP.10.008
Government Communications Security Bureau	New Zealand	"Information Security Manual-Zero trust"

United States

The United States has taken a leading role in the release of an array of zero trust artifacts, including architecture guidelines, capability maturity models, and other strategic reports.

Executive Order (EO) 14028—Improving the Nation's Cybersecurity

On May 12, 2021, the president of the United States issued the major executive order "Executive Order (EO) 14028—Improving the Nation's Cybersecurity", highlighting the imperative need to overhaul the security practices of all US government agencies with zero trust architecture as the foundation. The order specifically calls for the planning and implementation of a zero trust architecture in accordance with NIST and DoD guidelines; the relevant portion of the order is shown here:

> "[E]ach agency shall develop a plan to implement Zero Trust Architecture, which shall incorporate, as appropriate, the migration steps that the National Institute of Standards and Technology (NIST) within the Department of Commerce has outlined in standards and guidance, describe any such steps that have already been completed, identify activities that will have the most immediate security impact, and include a schedule to implement them."

The executive order was later followed by a memorandum titled "Moving the U.S. Government Toward Zero Trust Cybersecurity Principles", which was released on January 26, 2022. This memorandum reinforced the move toward the zero trust architecture but also outlined the strategy for government agencies to meet specific cybersecurity standards and goals by the end of fiscal year 2024 (which ends on September 30th, 2024). Both the executive order and the memorandum make significant references to NIST, CISA, and the DoD's publications on zero trust, which are covered in detail in the subsequent sections. However, here are the key business scenarios mentioned in the memorandum that must be supported by the zero trust strategy:

- Federal staff have enterprise-managed accounts, allowing them to access everything they need to do their job while remaining reliably protected from even targeted, sophisticated phishing attacks.
- The devices that federal staff use to do their jobs are consistently tracked and monitored, and the security posture of those devices is taken into account when granting access to internal resources.
- Agency systems are isolated from each other, and the network traffic flowing between and within them is reliably encrypted.
- Enterprise applications are tested internally and externally, and can be made available to staff securely over the internet.

- Federal security teams and data teams work together to develop data categories and security rules to automatically detect and ultimately block unauthorized access to sensitive information.

How to Navigate the US Government?

You can find a high-level U.S. government chart by visiting the United States organizational chart and a full list of US government departments and agencies at the A-Z index of US departments and agencies. Some readers may find it intriguing that the US legislative branch also has the United States Botanic Garden as a federal agency directly underneath it, mostly for historic reasons.

National Institute of Standards and Technology (NIST)

NIST released a special publication, SP 800-207, on zero trust architecture (ZTA) in August 2020. It primarily targets architects as an audience, offering an in-depth understanding of zero trust architecture, definitions, use cases, threats, and a roadmap for migrating toward zero trust architecture. It is important to highlight that the publication is vendor agnostic and makes a conscious choice not to provide vendor-specific implementation guidance. NIST primarily relies on the National Cybersecurity Center of Excellence (NCCoE) to provide blueprints for ZTA implementations with the assistance of vendors active in that domain, as ZTA implementations invariably involve vendor participation in real-world scenarios. NCCoE's zero trust publications are discussed later in this chapter.

In this section, you will learn the fundamental concepts of zero trust architecture as laid out by the NIST, including its logical components, deployment variations, the role of the trust algorithm, and threats to ZTA. Our goal is to cover topics from the publication in adequate detail to provide you with a firm grasp of ZTA, but we do encourage you to thoroughly study the publication for additional information, because covering every single topic in detail from the publication is beyond the scope of this book.

Zero trust/zero trust architecture definition

NIST defines zero trust (ZT) as *"a cybersecurity paradigm focused on resource protection and the premise that trust is never granted implicitly but must be continually evaluated,"* and it further elaborates that zero trust operationally is *"a collection of concepts and ideas designed to minimize uncertainty in enforcing accurate, least privilege per-request access decisions in information systems and services in the face of a network viewed as compromised."*

Finally, NIST goes on to define zero trust architecture as *"an enterprise's cybersecurity plan that utilizes zero trust concepts and encompasses component relationships, workflow planning, and access policies."*

The key tenets of zero trust can be summed up as follows:

- Always assume breach/compromise
- Always enforce least privilege access
- Always enforce per-request/session access
- Always enforce precise/just enough access
- Always apply access policy that is dynamic in nature
- Always include all computing services in the system, including the data sources, as resources that need protection
- Always monitor all resources on a continuous basis to evaluate their security posture and adjust access policies dynamically based on the threats
- Never grant implicit access based on network location alone

Thinking About Resource Access Holistically

The NIST guidance stresses the need to think more broadly about securing resource access, which should include securing access to devices, services, identity, etc., and that zero trust should not be limited to data access alone.

Zero trust architecture—logical components

ZTA logical architecture is divided into two basic building blocks: core components and data sources. The core components appear at the center of Figure 11-1, within the border, and communicate via a combination of the control plane and the data plane. It is important to note that this model may be translated into physical infrastructure in a variety of ways, as these components may be situated on premises, in the cloud, or both, depending on enterprise infrastructure requirements.

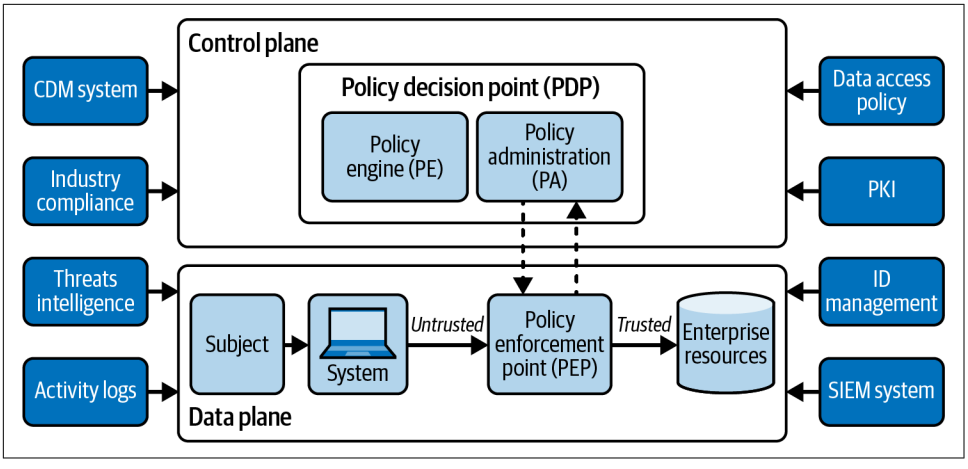

Figure 11-1. Logical components of ZTA

The ZTA's core components are defined in Table 11-2, along with a brief description of their purpose and interaction with other components.

Table 11-2. ZTA core components

ZTA core component	Description
Policy engine (PE)	The policy engine (PE) resides in the control plane and acts as the brain, deciding whether to allow, deny, or revoke access to a resource for a specific subject (human or nonhuman identities). The PE relies heavily on various external inputs (see Table 11-3) and the trust engine for decision making. The PE engages with the policy administrator, which is in charge of ensuring that the PE's decisions are actually enforced.
Policy administrator (PA)	The policy administrator (PA) resides in the control plane and is responsible for establishing or terminating the communication route between a subject and a resource. It depends on the PE to make decisions (grant, refuse, or revoke) and then communicate with the policy enforcement point to enforce the decision.
Policy decision point (PDP)	The PE and PA are logically combined into the policy decision point (PDP). However, the decision to combine PE and PA into a single component or to maintain their separation into two logical components is really a matter of implementation.
Policy enforcement point (PEP)	This policy enforcement point (PEP) is responsible for establishing, terminating, and monitoring connections between a subject and a resource. The PEP communicates with the PA to evaluate the access requests, as they are received from the subject, and to obtain and enforce the policy decisions that come back from the PA. The PEP can be a single logical component, or it can be divided into two distinct components: a client-side component, such as an agent running on a device, and a resource-side component, such as a gateway component that resides in front of the resource and enforces access. See "Zero trust architecture—deployment variations" on page 260 for more information on PEP placement.
Subject	The subject may be a human or nonhuman entity, such as an end user, service, application, API, etc., that requests access to resources.
System	The subject interacts with the system, and the system must verify the subject's identity as well as carry out authentication and authorization. The system can be a device such as a laptop, mobile phone, virtual machine, container, etc., and the subject may be a human using a laptop or a nonhuman such as a cron job. The system depends on PEP to allow communication with the resource.

ZTA core component	Description
Enterprise resources	Enterprise resources comprise a wide range of resources that enterprises want to secure—applications, services, databases/data lakes, processes, printers, networks, APIs, and so on. These enterprise resources can be on premises, in the cloud, or a hybrid of both.
Untrusted/ trusted	Untrusted means there is no implicit trust, and the system must verify the subject and ensure that the access request is valid. The PDP makes the necessary decision to enable or deny the subject's access to the resource, and the PEP enforces it. Trusted implies that all traffic beyond the PEP shares the same degree of trust. Also, the PEP, at the direction of the PA, may revoke the already established communication between the subject and the resource at any moment, as trust is not perpetual and is contingent on a variety of factors (e.g., external threat intelligence, etc.).

Aside from the core components, we have a variety of data sources that help provide a range of input for the policy engine and assist it during decision making while evaluating the access request. Table 11-3 describes these data sources, along with an explanation of their role in the ZTA.

Table 11-3. Data sources used by the ZTA

ZTA sources	Description
Continuous diagnostics and mitigation (CDM) system	The CDM system is in charge of collecting data on the current state of corporate assets as well as changes to the system's software and hardware setup. The PA uses CDM to get information about the asset requesting access, such as its hardware, OS version and security patch level, etc.
Industry compliance system	This system consists of policy rules to ensure that all necessary enterprise regulations and compliance standards are met.
Threat intelligence	Threat intelligence is a feed compiled from internal or external sources that helps the PE make access decisions. The dark web is an example of an external threat intelligence source, as it may contain information about compromised user accounts. Another example may be the information from third-party intelligence sources, which may include newly discovered software vulnerabilities (such as zero-days) that affect enterprise assets such as operating systems, software, etc.
Activity logs	This system aggregates enterprise-wide networking and activity logs, capturing a vast multitude of events in real time or near-real time, and can generate comprehensive reports on the security posture of enterprise information systems.
Data access policy	This is a wide collection of attributes and rules concerning access to the enterprise resources. These policies are the starting point for authorizing access to a resource, as they stipulate the fundamental access privileges for enterprise accounts and resources.
Enterprise public key infrastructure (PKI)	This system is responsible for issuing and tracking certificates that the enterprise issues to resources, subjects, services, and applications. The most prevalent certificate format used in the industry is X.509; however, other certificate formats are also used.
Identity (ID) management system	This system is in charge of the identity lifecycle management of enterprise identities. For example, it may contain all of the relevant information, such as the user's legal name, email address, certificates, devices, and extra information needed for entitlement management, such as role assignments, access attributes, etc. This system also makes use of other components, such as enterprise PKI, to associate certificates with the identities.

ZTA sources	Description
Security information and event management (SIEM)	This system collects security-related information for further analysis of enterprise-wide activities. The data is then used to enhance existing policies and to generate alerts regarding potential threats to corporate assets. For instance, it can assist in identifying a pattern or an anomalous behavior pattern exhibited by a user or malicious actor who is potentially conducting lateral movement in the network.

Zero trust architecture—deployment variations

While logical components aid in conceptualizing ZTA components and interactions, the actual deployment of ZTA may take different variations. This section briefly describes different deployment variations, as outlined in the NIST publication, along with their pros and cons. It is also important to note that within a single enterprise, several variants can exist at the same time; they are not mutually exclusive, and their usage is primarily dependent on maturity and business needs.

Device agent/gateway-based deployment. In this deployment model, as shown in Figure 11-2, the PEP is divided into two components—an agent running on a device and a resource gateway, which is placed on or in front of the enterprise resource that needs to be protected (e.g. APIs, databases, etc.).

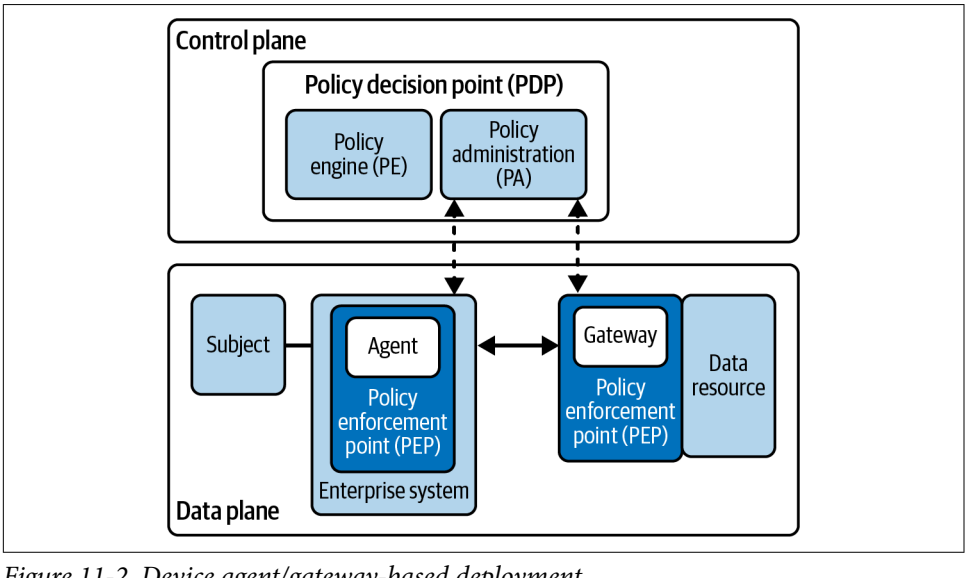

Figure 11-2. Device agent/gateway-based deployment

Pros

Ideal for enterprises with robust asset/device management, as agents need to be installed and maintained on the devices.

Cons

Supporting Bring Your Own Device (BYOD) scenarios may be challenging as there is a high barrier to entry since agent installation and maintenance on personal devices can be difficult and costly.

Enclave gateway model. This deployment model, as shown in Figure 11-3, is a variant of the previous device agent/gateway deployment model. The difference is that the resource gateway may not exist on or in front of individual resources, but rather at the boundary of a resource enclave (for example, the entire datacenter is located behind the resource gateway).

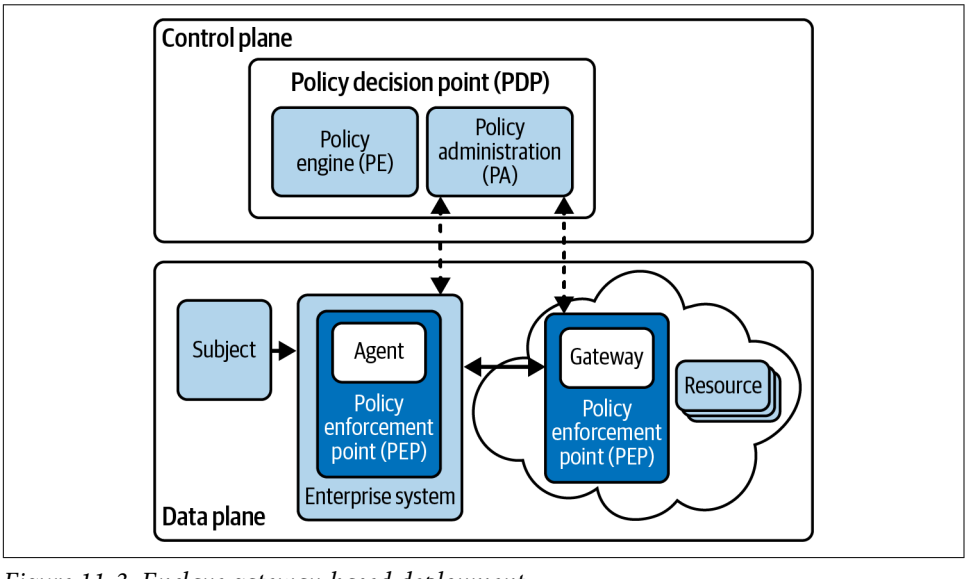

Figure 11-3. Enclave gateway-based deployment

Pros

Suitable for enterprises with legacy applications or those with on-premises datacenters that cannot accommodate using separate gateways due to various technical and functional reasons.

Cons

Since a resource gateway is placed in front of a collection of resources, it is possible for subjects to view and perform reconnaissance on resources to which they do not have access.

Resource portal-based deployment. In this deployment model, as shown in Figure 11-4, the PEP is a singular component that functions as a gateway for handling access requests. This model may be used for a single resource or for multiple resources that are bundled together to serve a specific business function.

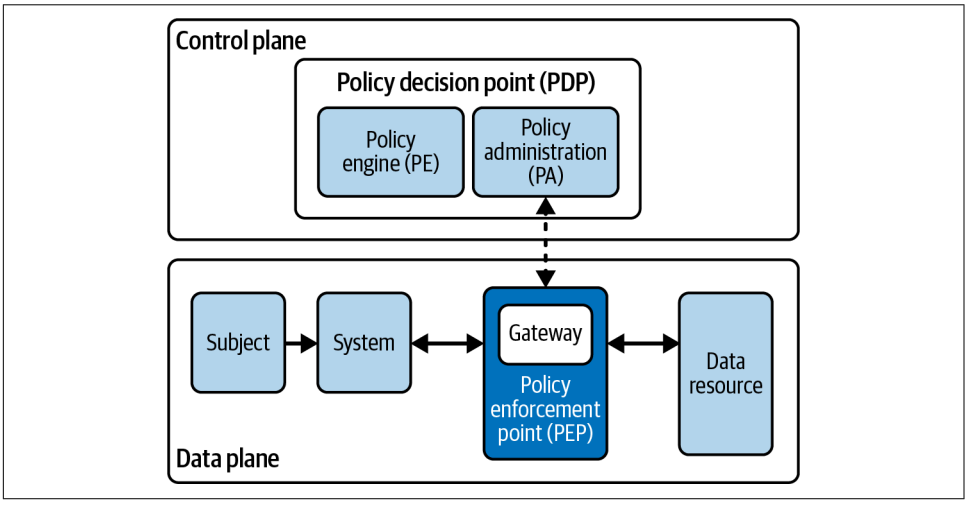

Figure 11-4. Resource portal-based deployment

Pros

Does not require agents to be running on all client devices, which is a key benefit over other models. This approach eliminates the cost associated with agent installation and management and is also more conducive to BYOD scenarios.

Cons

As no agent is present on the devices, this approach does not provide the enterprise with complete visibility or control over its assets, and as a result, device vulnerabilities may lead to a lower security posture and expose the resources to various attacks such as denial-of-service (DoS), ransomware, etc.

Device application sandboxing. This is another variation of the device agent/gateway-based deployment model. In this model, as shown in Figure 11-5, the subject's device runs enterprise-approved applications in an isolated sandbox environment (such as a virtual machine, containers, trusted platform module, etc.). Only the approved applications can communicate with the PEP to request resource access, and the PEP will deny requests from other applications on the asset.

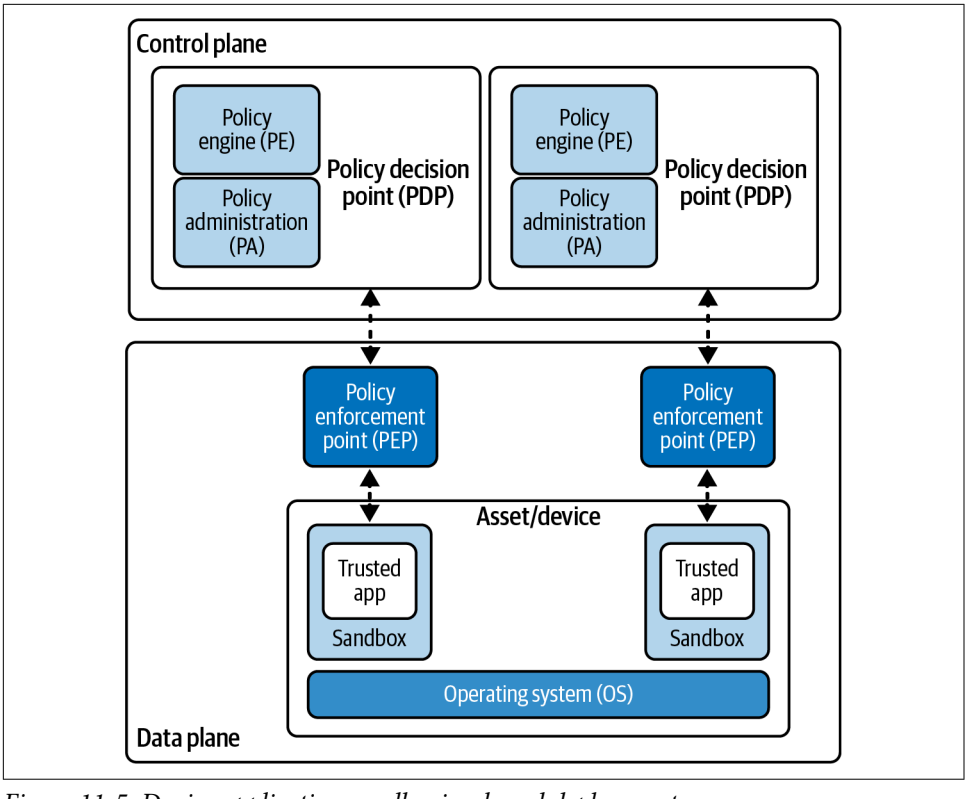

Figure 11-5. Device application sandboxing-based deployment

Pros

The separation of individual applications, due to the sandbox, from the rest of the asset is a primary benefit. Individual sandboxed applications may be protected against a possible malware infection in the host asset if the asset cannot be scanned for vulnerabilities.

Cons

Enterprises are required to maintain sandboxed applications for all client assets. Additionally, the organization must ensure that every sandboxed application is secure, which may require more effort than merely monitoring devices.

Trust algorithm

The trust algorithm (TA) is the fundamental process upon which the policy engine relies when making the resource access decision (grant, deny, revoke). The TA consumes a variety of data sources, as shown in Figure 11-6, which include information about subjects, resources, entitlements, activity logs, and so on. The TA makes the

decision, which is captured by PE and eventually relayed to the PA, which is in charge of carrying it out through the PEP.

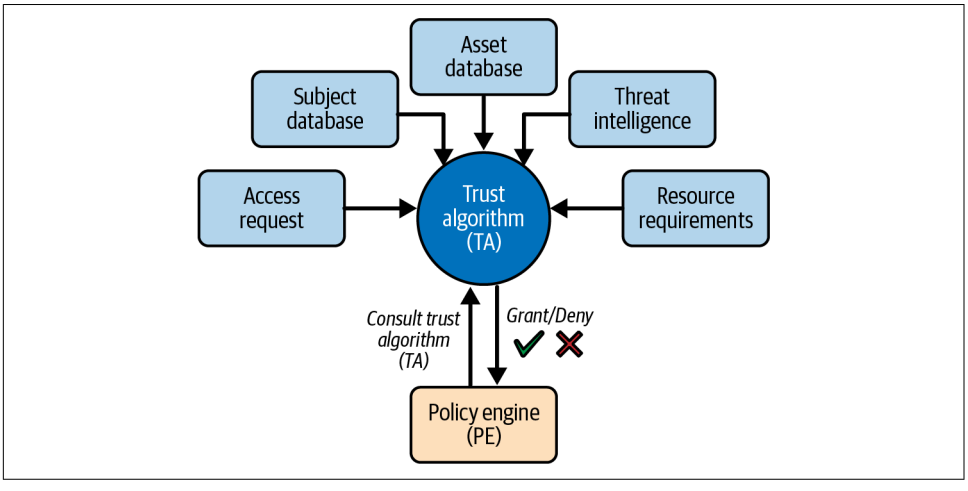

Figure 11-6. Sources used by ZTA trust algorithm

Access request
: This is the access request from the subject. The primary information utilized is the resource requested, although other information related to the subject and network agent may also be leveraged (for example, security patch version, device compliance status, etc.)

Subject database
: Database containing attributes related to the identity of the subject, including but not limited to personally identifiable information (PII), geo-location, entitlements, etc. This information is typically stored in the identity management system.

Asset database
: This is the asset inventory database, which contains the known state of every enterprise-owned and/or nonenterprise/BYOD of assets, including but not limited to devices, virtual machines, etc. The state can include attributes like OS version, firmware version, etc.

Resource requirements
: This includes enterprise rules/policies aligned with business processes and compliance requirements, such as restricting access based on time/day, geo-location, and requiring higher authentication (e.g., MFA) based on the data sensitivity/criticality of the resource.

Threat intelligence
: This includes feeds on the most recent threats compiled from various sources, such as the dark web, third-party sources such as Common Vulnerabilities and Exposures (CVE), operated by the MITRE corporation, etc.

When implementing the TA, organizations can take a variety of approaches. At a high level, there are two key factors to consider, as described by the NIST. First, how should the TA evaluate various input sources? Second, how is the access request evaluated by the TA?

Let's examine each of these separately.

Evaluation of input sources by the trust algorithm. When evaluating various input sources, the TA can utilize two distinct approaches. A *criteria-based* approach is one in which the TA maintains a list of qualifying criteria or attributes for resource access. The TA evaluates each factor as a binary (such as yes/no) decision based on a set of criteria. When all criteria are met, only then is access granted. Another approach is to calculate a *score/confidence level* based on enterprise-specified data source values and weights against each factor. In this approach, the TA performs an evaluation and grants access only if the confidence level or score exceeds the resource's predetermined threshold (e.g., high, medium, or low; or it can be a discrete value within a range like 1–10). If not, the request is either denied or access is restricted.

The benefit of a criteria-based approach is that enterprises may be able to leverage existing access policies that have well-defined criteria for resource access based on various factors. Rather than starting from scratch, this gives a low barrier to entry and a starting point for TA implementation. However, on the flip side, because policies are based on predefined criteria and are typically static in nature, organizations may find it difficult to change them dynamically and quickly enough to address security issues as they arise. In contrast, with a score/confidence level approach, weights/thresholds can be changed dynamically, hence adapting to security challenges more quickly. However, determining the initial weights/thresholds requires maturity as well as time for tuning because the values may be less optimal at first and require testing and adjustments, resulting in a suboptimal experience initially (e.g., users being asked to perform MFA every time due to weights that haven't yet been adjusted).

Evaluation of access request by the trust algorithm. Regarding how the TA should evaluate access requests, there are two possible approaches highlighted by the NIST: s*ingular* and c*ontextual*. With the singular approach, each access request is handled independently, and the evaluation of the request does not factor in the subject's and network agent's prior access requests and outcomes. The contextual approach, in contrast, takes them into account.

The singular approach does provide speedier request evaluations; however, there is a risk that an attack may go unnoticed as long as it remains within the boundaries of a subject's/network agent's approved role/entitlements, since any other context information based on historic patterns of prior requests is out of scope and not considered during the request evaluation. On the other hand, the contextual approach provides a far more robust security posture because trends (such as lateral movement, for example) can be discovered because the subject's and network agent's history is available during the access request evaluation, and broader context is available for the TA. However, the TA is required to maintain the subject's/network agent's history and state information. This may incur additional costs, and until the history is established, performance may not be optimal.

Threats

Enterprises that adopt and implement ZTA should be aware of the various types of threats to ZTA and potential methods for addressing them, as published by the NIST. These threats are briefly summarized in Table 11-4, along with a list of the ZTA components they affect and the proposed mitigation strategies.

Table 11-4. Summary of threats to ZTA with proposed mitigation strategies

Potential threat to ZTA	Description of threat	Impacted components	Mitigation strategies
Subversion of ZTA decision process	Inadequate configurations of the PE and PA may lead to disruption of enterprise operations as a whole. In ZTA, enterprise components will be unable to communicate without the PE and PA.	Policy engine, policy administrator	Monitoring, logging and auditing, configuration management
Denial-of-service or network disruption	A DoS attack or lack of network availability can disrupt or deny access to the PEP, PA, and PE, and negatively affect enterprise operations.	Policy engine, policy administrator, policy enforcement point	Resiliency
Stolen credentials/ insider threat	Malicious actors utilizing social engineering or other attacks on company employees can obtain and utilize credentials and then use them to access certificates, secrets, data, etc. Similar risks arise from insider threats when an employee goes rogue.	Identity (ID) Management System, PKI, data access policy , continuous diagnostics and mitigation (CDM) system	Contextual trust algorithm (TA) for detecting and stopping anomalous behavior, logging and auditing, monitoring
Visibility on the network	Due to network traffic that may not be appropriately logged and detected (e.g., at layer 3), malicious actor behaviors on the network, such as lateral movement, may go undetected.	Activity logs, security information and event management (SIEM) system	Metadata analysis for detecting malware communication patterns, machine learning, deep packet inspection

Potential threat to ZTA	Description of threat	Impacted components	Mitigation strategies
Storage of system and network information	An attacker may target the crucial networking data and related assets kept by SIEM and CDM systems, data access policy, and other components. PE policy rules can potentially reveal useful information about the nature of PE policies to attackers.	CDM system, industry compliance, activity logs, data access policy, PKI, identity and access management, SIEM system, policy engine, policy administrator	Strict access policies enforcement
Reliance on proprietary data formats or solutions	When it comes to processing and storing information related to subjects, assets, and threat intelligence, enterprises may become reliant on proprietary standards from a handful of vendors. This inadvertently leads to interoperability challenges, higher operational costs, and even service disruption if the enterprise decides to switch vendors for any reason.	CDM system, industry compliance, activity logs, data access policy, PKI, identity management system, SIEM system, policy engine, policy administrator	Uses industry or open standards, avoids proprietary standards, analyzes cost/time impact of switching vendors, and conducts supply chain risks analysis before selecting vendors
Use of non-person entities (NPEs) in ZTA administration	When using a non-person entity (NPE) like a machine/device, there is a possibility of false positives as well as false negatives owing to access patterns. Furthermore, NPEs often cannot conduct MFA, which introduces further identity-related risks like spoofing, etc.	Policy engine, policy administrator	Performs continuous activity analysis for NPEs and catches and fixes errors (e.g., false positives)

National Cybersecurity Center of Excellence (NCCoE)

The NCCoE is a nonregulatory federal organization in the United States and part of the NIST's information technology laboratory. The NCCoE collaborates across multiple sectors, including private industry, government agencies, and academia, in order to publish security solutions based on industry standards for organizations of all sizes.

At the time of this writing, the NCCoE is working on a project called "Implementing a Zero Trust Architecture" that intends to give practical examples of the zero trust principles defined in NIST's SP 800-207 "Zero Trust Architecture" publication. As part of the project, the NCCoE has published a new guide titled "NIST SP 1800—Implementing a Zero Trust Architecture" in collaboration with a variety of commercial vendors that offer zero trust products and services. This comprehensive guide has been divided into five volumes to accommodate a variety of audiences. Table 11-5 provides an overview of all five volumes as well as a brief description of each guide along with its intended audience.

Table 11-5. Listing of various volumes published as part of NIST's "SP-1800 Implementing a Zero Trust Architecture" project

Volume	Publication	Description	Audience
35A	"Executive Summary" (2nd Preliminary Draft)—SP-1800	This document summarizes the objectives behind the SP-1800 series of publications, the key challenges addressed by the NCCoE, the approach adopted to solve these challenges, and the relevance of this guide to enterprises. Since the focus of the SP-1800 series is on the implementation of zero trust, it also describes how the NCCoE and its collaborators are utilizing commercially available technology to develop interoperable, open standards-based ZTA implementations that adhere to the key principles specified in the NIST zero trust architecture publication.	CISOs, CTOs, IT professionals, or anyone in a decision-making role.
35B	"Approach, Architecture, and Security Characteristics" (2nd Preliminary Draft)—SP-1800	This is a very comprehensive document with detailed examples of ZTA approaches based on various commercial vendors. It begins by demonstrating the mapping of various vendor products/services to a logical ZTA before describing the physical architecture required for implementation. The guidance provided is primarily geared toward large and medium-sized enterprises. Although it covers a range of zero trust scenarios relevant to an enterprise, industrial control systems (ICSs) and operational technology (OT) environments are expressly excluded.	IT professionals, architects, and security managers, as well as anyone in a technical management or architecture role.
35C	"How-To Guides" (2nd Preliminary Draft)—SP-1800	This document provides instructions on how to construct the reference implementations specified in NIST SP 1800-35B by a variety of commercial vendors. In addition, it provides links to the extensive vendor documentation required to establish a functioning environment based on the proposed vendor product or service.	IT professionals, or anyone in an operations and infrastructure management role.
35D	"Functional Demonstrations" (2nd Preliminary Draft)—SP-1800	This document describes a variety of ZTA use cases and scenarios. It also discusses the results of use case implementations based on several commercial vendors' products/services.	IT professionals, architects, security managers, or anyone in an operations and infrastructure management role.
35E	"Risk and Compliance Management" (Preliminary Draft)—SP-1800	This document provides mappings between logical components of the ZTA reference design (especially section 4.1 of NIST SP 1800-35B) and security characteristics mentioned in a variety of NIST cybersecurity publications, with a particular focus on: SP 800-53r5: "Security and Privacy Controls for Information Systems and Organizations" NIST CSF 1.1: "Framework for Improving Critical Infrastructure Cybersecurity" "Security Measures for 'EO-Critical Software' Use"	IT professionals, architects, and security managers, as well as anyone in a technical management, risk management, or architecture role.

Cybersecurity and Infrastructure Security Agency (CISA)

The CISA Zero Trust Maturity Model has its foundation in the zero trust pillars concept from the DoD and NSA zero trust architectures (both are covered later in this chapter). The purpose of CISA's maturity model is to aid government agencies in developing zero trust implementation plans in response to Executive Order 14028.

The zero trust pillars of CISA, as depicted in Figure 11-7, match the first five DoD/NSA architectural pillars, with the exception of the renaming of the first pillar from User to Identity.

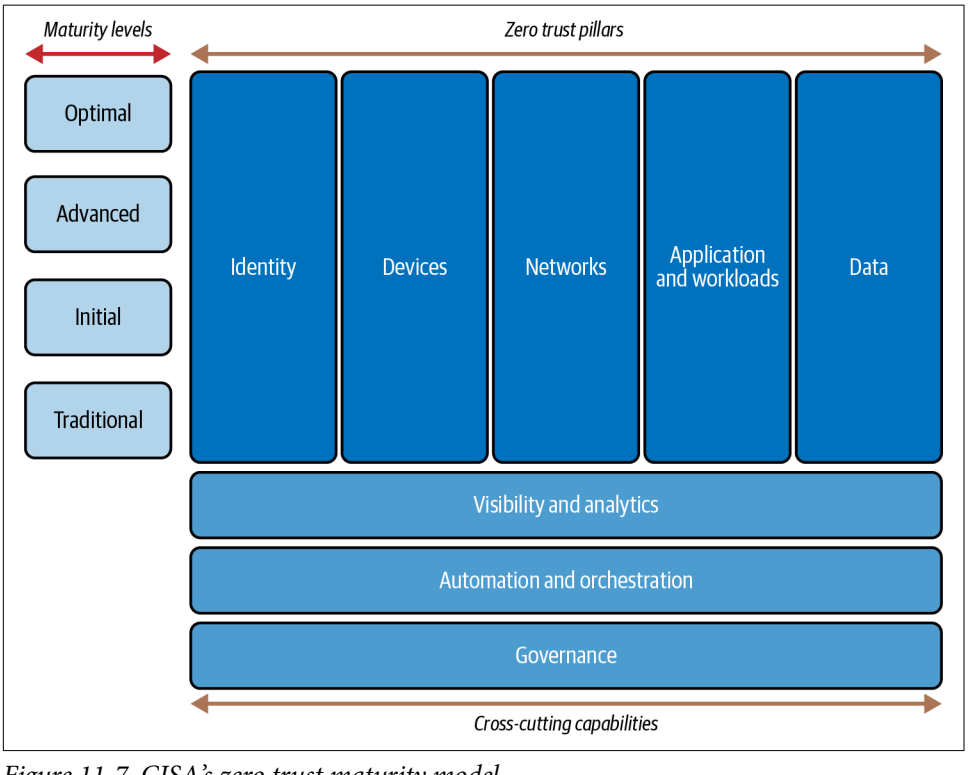

Figure 11-7. CISA's zero trust maturity model

The following are high-level descriptions of the pillars in this model:

Identity
: An identity is an attribute or group of qualities that uniquely describes a user or entity (which can be a non-person entity, known as an NPE).

Devices
: A device is any hardware asset that can connect to a network. This may include an internet of things (IoT) device, smart watch, mobile phone, laptop, server, virtual machine, etc.

Networks
: A network is an open communications channel that transports messages over agency internal networks, wireless networks, and the internet.

Applications and workloads
: These include computer programs, apps, APIs, and services that run on premises or in the cloud.

Data
: Includes any information a business requires to conduct its key operations; it can reside on premises or in the cloud.

In addition to the five pillars, there are three key cross-cutting capabilities that facilitate the interoperability across the pillars:

Visibility and analytics
: The term *visibility* refers to the observable artifacts produced by the events occurring throughout the enterprise. The primary focus is on security-related data analysis that aids in enhancing policies, facilitating responses, and developing a risk profile in order to implement proactive security measures that can be used to prevent potential attacks.

Automation and orchestration
: Zero trust is highly reliant on automated tools, processes, and workflows to enable the security response function across all pillars.

Governance
: This is the process of defining and enforcing the cybersecurity policies, government standards, procedures, and processes across the zero trust pillars.

According to CISA, achieving zero trust maturity involves beginning at a traditional level as a starting point, then progressing through initial, advanced, and optimal levels as zero trust architecture is implemented. Each successive maturity level necessitates higher levels of security, interoperability among pillars, automation, and overall awareness of the organization's security posture and ability to react quickly against attacks:

Traditional
: This is the most basic maturity level, with little to no automation for the lifecycle management of identities, assets, and resources. Security policies are static and fragmented, and the least privilege principle is applied only when establishing

new accounts and resources; it is not maintained beyond that. The organization lacks fundamental security information and event management (SIEM) and security orchestration, automation, and response (SOAR) capabilities, leaving it largely reliant on manual and ad hoc processes to detect and respond to attacks.

Initial
: At this maturity level, the lifecycle management of identities, assets, and resources has a basic level of automation. The majority of security rules are static, with little cross-pillar integration. After initial account and resource provisioning, the least privilege principle is only partly tracked and applied. Security information and event management capabilities are limited, and security orchestration, automation, and response capabilities are only partially available in a siloed fashion among the pillars.

Advanced
: At this level of maturity, the controls for lifespan and configuration assignment for identities, assets, and resources are automated. Security policies are coordinated across multiple pillars. The least privilege principle is used throughout the ecosystem, and risk is considered while evaluating changes in privilege. SIEM and SOAR capabilities are available centrally, with information flowing in from across all the pillars.

Optimal
: This is the highest level of maturity, and at this level, controls for lifecycle and configuration assignment for identities, assets, and resources are just-in-time (JIT) and fully automated. Security policies are adaptive and able to adjust automatically, reflecting changes in the security posture across the pillars. The least privilege principle is dynamic and applies privileges to assets and resources across the pillars using just-enough access (JEA). SIEM and SOAR capabilities are centrally available and provide a comprehensive view of the security posture across the zero trust pillars.

Department of Defense (DoD)

The United States Department of Defense (DoD) published the "Zero Trust Reference Architecture", which divides zero trust principles and technologies into seven pillars: user, device, network/environment, application and workload, data, visibility and analytics, and automation and orchestration.

Each pillar is a critical focus area for zero trust control implementation, and all are needed to protect the data, which is at the center, as shown in Figure 11-8. This reference architecture's primary stakeholders are the DoD's mission owners (MOs), who are defined as "*individuals/organizations responsible for the overall mission environment, ensuring that the functional and cybersecurity requirements of the system are being met.*"

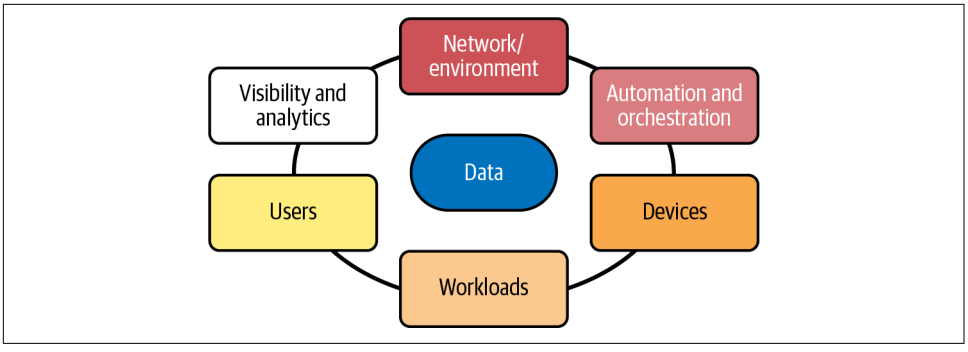

Figure 11-8. DoD zero trust pillars

The following is a brief description of each pillar:

Users
: Personal and nonperson entity access must be secured, constrained, and enforced using identity capabilities such as multifactor authentication (MFA) and privileged access management (PAM) for privileged functions. To control user access and privileges, organizations must authenticate, authorize, and monitor user activity patterns on a continuous basis.

Devices
: A zero trust system requires the capacity to identify, authenticate, enumerate, authorize, isolate, secure, remediate, and control all devices. Also, devices must be authenticated, inspected, appraised, and patched on a regular basis.

Application and workload
: The security and management of applications and workloads such as containers, virtual machines, and so on are critical to zero trust adoption. To secure applications from the start, source code and shared libraries must adhere to strict secure development lifecycle (SDLC) and DevSecOps guidelines.

Data
: As part of their overall zero trust strategy, organizations must classify their data, assets, applications, and services (DAAS) based on mission criticality, and use this information to develop a comprehensive data management plan. This involves validating data, classifying data, creating schemas, and encrypting data at rest and in transit.

Visibility and analytics
: Visibility facilitates the detection of anomalous behavior and provides insights that facilitate changes to the security policy dynamically. A zero trust enterprise will record and examine traffic. This requires going beyond network telemetry and inspecting packets using techniques such as deep packet inspection.

Network/environment
: Use segmentation to isolate and control off-premises (e.g., cloud) and on-premises (e.g., private datacenter) networks/environments with granular access control and policies. Use of micro-segmentation is highly recommended as it enables tighter and fine-grained control over access flows, which may help limit the lateral movement.

Automation and orchestration
: Automation of manual security procedures to enable policy-based decision making across the enterprise is necessary. The use of SIEM and SOAR to detect, react, and respond to threats more swiftly is important.

The DoD reference architecture uses the following guiding principles when establishing a zero trust security approach:

- Assume no implicit trusted zone in networks.
- Identity-based authentication and authorization are strictly enforced for all connections and access to infrastructure, data, and services.
- Machine to machine (M2M) authentication and authorization are strictly enforced for communication between servers and applications.
- Risk profiles, generated in near-real time from monitoring and assessment of both user and device behaviors, are used in authorizing users and devices to access the resources.
- All sensitive data is encrypted both in transit and at rest.
- All events are to be continuously monitored, collected, stored, and analyzed to assess compliance with security policies.
- Policy management and distribution is centralized.

The reference architecture is comprehensive. You can acquire a deeper understanding of specific topics covered in the reference architecture document by perusing the relevant sections:

- "Vision and Goals" (refer to section 1.4)
- "Pillars and Principles" (refer to section 2)
- "Capabilities" (refer to section 3)
- "Use Cases" (refer to section 4)
- "Technical Positions" (refer to section 5)
- "Security Assessment" (refer to section 6)

- "Architecture Patterns" (refer to section 7)
- "Transition Architecture Planning" (refer to section 8)

DoD Resources Library

Visit the DoD online digital library to gain access to all the zero trust publications mentioned in this book, as well as other relevant cybersecurity and technology-related publications.

Further publications from the DoD on zero trust are provided here. These can be used to learn more about the DoD's zero trust vision, strategy, timetables, and methodology:

"Zero Trust Strategy"
: Establishes intended outcomes for various components in order to achieve minimum zero trust target-level capabilities for data, assets, applications, and services (DAAS) at all classification levels throughout the DoD Information Enterprise (IE).

"Zero Trust Reference Architecture"
: Establishes a zero trust reference architecture, provides direction via architectural pillars and principles, and identifies broader strategic goals and objectives.

"Zero Trust Capability Execution Roadmap"
: This is the roadmap that depicts how zero trust capabilities will progress across the seven pillars of zero trust.

Zero Trust strategy placemat
: Provides a concise picture of the DoD Information Enterprise (IE)—Zero Trust Framework, including zero trust culture adoption, technology, and zero trust enablement.

National Security Agency (NSA)

The National Security Agency has published "Embracing a Zero Trust Security Model", which provides guidance that highlights how zero trust security principles can better position cybersecurity professionals to protect enterprise networks and sensitive data. To provide NSA customers with a foundational understanding of zero trust, this publication discusses its benefits, outlines a maturity model along with the potential challenges, and makes recommendations for implementing zero trust within their networks.

The NSA publication stresses taking a fresh perspective to address cyber threats, and advocates for the following guidelines:

- Coordinated and aggressive system monitoring, system management, and defensive operations capabilities.
- Assuming all requests for critical resources and all network traffic may be malicious.
- Assuming all devices and infrastructure may be compromised.
- Accepting that all access approvals to critical resources incur risk, and being prepared to perform rapid damage assessment, control, and recovery operations.

It also emphasizes the need to adhere to the fundamental principles of zero trust: never trust, always verify, assume breach, and verify explicitly. It also suggests the importance of acknowledging the difficulties and maturity required for implementing zero trust, as well as the fact that such an endeavor requires time and deliberate planning.

It highlights that transitioning to a mature zero trust architecture all at once, in a big bang fashion, is not necessary and may even be detrimental. The transition should instead be gradual and iterative. When examining how to integrate zero trust principles into an environment, the NSA advises beginning with early planning, and progressing through basic, intermediate, and advanced levels of maturity over time. The NSA's approach toward zero trust mature implementation is shown in Figure 11-9.

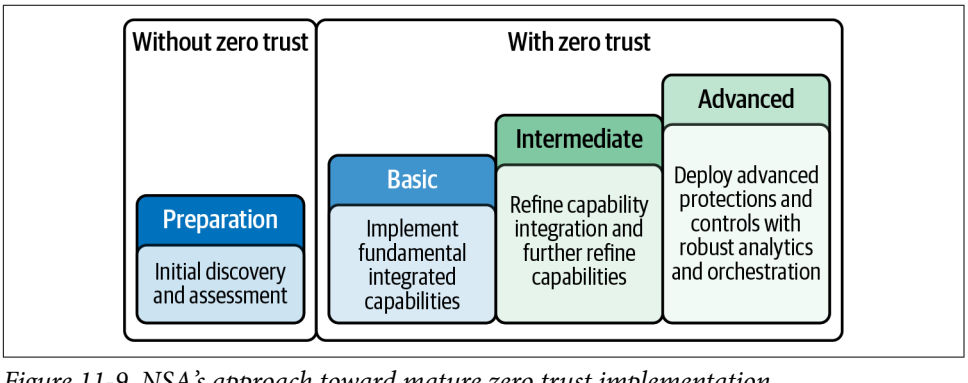

Figure 11-9. NSA's approach toward mature zero trust implementation

The NSA also identified potential roadblocks on the path towards zero trust. These can be summed up as follows:

- Lack of support/buy-in from leadership, administrators, and users
- Infrastructure improvements and scalability are needed to enable the zero trust initiative
- Implementation fatigue can result from persistent technical and functional challenges

The "Advancing Zero Trust Maturity Throughout the User Pillar" is a related publication by the NSA that describes how to improve the maturity of the user pillar.

United Kingdom

The National Cyber Security Centre of the United Kingdom has published "Zero trust architecture design principles," which provide guidance for designing and evaluating a zero trust architecture in accordance with an organization's specific needs. These guidelines are intended to assist organizations implementing a zero trust architecture in an enterprise environment, including both the public and private sectors. These principles are high-level guidelines focused on knowing the key tenets of zero trust, then applying policies to implement zero trust, and then ensuring that trust is not granted implicitly at any point. Figure 11-10 visually explains these principles.

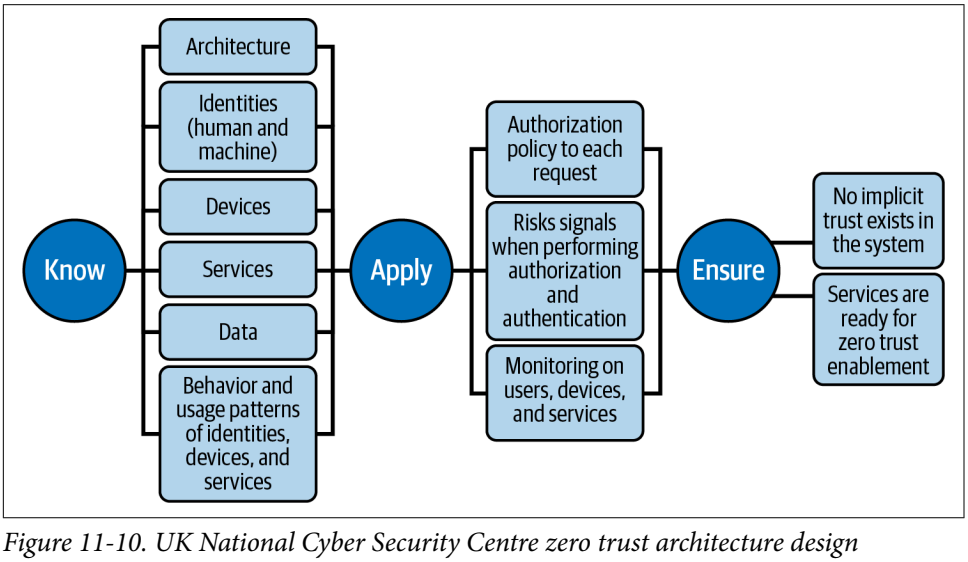

Figure 11-10. UK National Cyber Security Centre zero trust architecture design principles

European Union

The EU Directive 2022/2555 advocates for the implementation of zero trust principles. The following excerpt highlights the call for the adoption of zero trust principles along with other cybersecurity practices:

> "Essential and important entities should adopt a wide range of basic cyber hygiene practices, such as zero-trust principles, software updates, device configuration, network segmentation, identity and access management or user awareness, organise training for their staff and raise awareness concerning cyber threats, phishing or social engineering techniques."

Although the directive does not provide a formal zero trust reference architecture or model, it does emphasize zero trust principles as part of overall cybersecurity.

The Network and Information Security (NIS2) directive, published in August 2023, is another relevant piece of EU-wide cybersecurity legislation with the stated goal of attaining a high common standard of cybersecurity across all European Union member states.

Private and Public Organizations

This section covers several publications that are pertinent to the zero trust and are widely referenced within the security industry. These publications have been generated by public or private organizations, including working groups, consulting and research firms, etc.

Cloud Security Alliance (CSA)

The Cloud Security Alliance (CSA) is a nonprofit organization dedicated to developing and advocating best practices for cloud computing security. Following is a list of various publications by the CSA, including reports, specifications, and guidance on zero trust. These publications have broad audiences, including senior leaders, executives in security roles, architects, engineers, and technical product managers.

"CISO Perspectives and Progress in Deploying Zero Trust"
: This is survey-based research that covers topics such as where zero trust stands in organizations, the percentage of people who have completed related implementations, top business issues, and top technical challenges.

"Integrating SDP and DNS: Enhanced Zero Trust Policy Enforcement"
: The goal of this article is to show how an enterprise-managed DNS, DHCP, and IPAM (IP address management)—their combination is referred to as DDI—system can be integrated with a Software-Defined Perimeter (SDP) to improve security visibility, resiliency, and responsiveness.

"Software-Defined Perimeter (SDP) and Zero Trust"
: This paper demonstrates how an SDP can be used to construct zero trust networks (ZTNs) and why the SDP architecture is the best for achieving zero trust. It explains how applying an SDP enhances the security posture of enterprises facing the challenge of continuously adjusting to expanding and increasingly malicious security threats.

"Software-Defined Perimeter (SDP) Specification v2.0"
: This specification document covers the definition of an SDP and its operation, an explanation of the three SDP architecture components (Controller, Initiating Hosts, and Accepting Hosts), and provides a preliminary look at six distinct SDP deployment models.

"Toward a Zero Trust Architecture"
: This paper explores the impacts of evolving and diversified solutions as well as difficulties that an organization may have in delivering a zero trust architecture.

"Zero Trust as a Security Philosophy"
: This paper examines zero trust from a vendor-neutral and technology/solution-neutral perspective, and offers ideas for developing a strategy and supporting architecture that support the company and its workflows while aligning IT to business goals and outcomes.

The Open Group

The Open Group is a global consortium that facilitates the achievement of business objectives through the implementation of technology standards. Following is a list of The Open Group's zero trust publications. Their target audiences include senior leaders, executives in security roles, architects, and others.

"Zero Trust Commandments"
: This document is intended for business, security, and information technology administrators. The commandments are derived and expanded from the "Zero Trust Core Principles" whitepaper that is also released by The Open Group.

"Zero Trust Core Principles"
: This document presents zero trust to business, security, and IT leaders. It outlines the key motivations for zero trust, its effects, and the core components of zero trust.

Gartner

Gartner, Inc. is a prominent technology research and consulting organization that performs technology research and offers its findings through private consultation, executive programs, conferences, and other reports. According to Gartner, zero trust

network access (ZTNA) is a maturing technology, with several vendors offering various ZTNA products and services. These services, however, are often simply the first step in using technology as part of a zero trust approach. Typically, organizations begin by analyzing ZTNA vendor capabilities, ignoring the broader alignment to strategy and use cases. To address this gap, Gartner has introduced SASE (pronounced "sassy," and stands for secure access service edge), which refers to a framework (not a technology) that includes comprehensive WAN and security services functions. It also calls for alignment of security requirements with ZTNA's primary use cases, as shown in Figure 11-11.

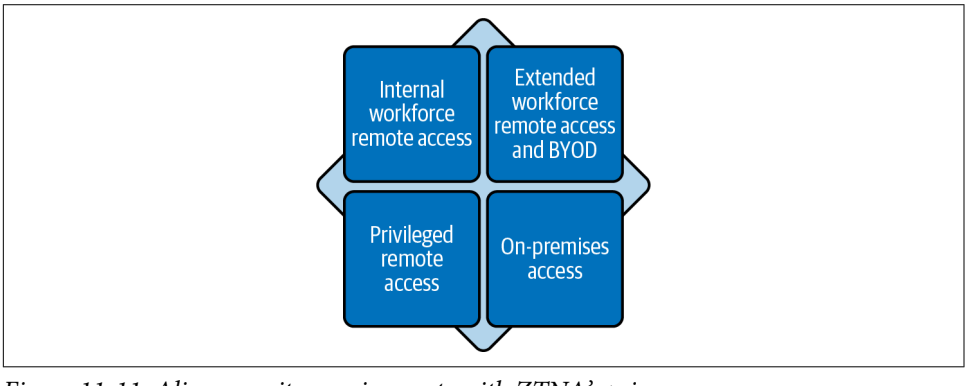

Figure 11-11. Align security requirements with ZTNA's primary use cases

SASE offers a converged network and security as a service capabilities, including an SD-WAN (software-defined wide area network), SWG (secure web gateway), CASB (cloud access security broker), NGFW (next-generation firewall), and ZTNA (zero trust network access). SASE enables secure access use cases for branch offices, remote workers, and on-premises. SASE is largely supplied as a service and offers zero trust access based on the device or entity's identity, in conjunction with real-time context and security and compliance standards.

Gartner chose not to release its Magic Quadrant, an access point for suitable vendors, for the SASE, but instead released it for security service edge (SSE), which is described as a convergence of network security services supplied from a purpose-built cloud platform. In a nutshell, SSE is a subset of the SASE framework, with its architecture centered on security services. SSE is made up of three key services: an SWG, a CASB, and a framework for ZTNA.

Forrester

Forrester is a UK research and advisory firm that provides research, consulting, and events, among other services. In November 2010, it published a report on zero trust titled "No More Chewy Centers: The Zero Trust Model Of Information Security" that defined the key aspects of zero trust security as they are known today. In January

2022, it published "The Definition Of Modern Zero Trust", which defines zero trust as:

> "Zero Trust is an information security model that denies access to applications and data by default. Threat prevention is achieved by only granting access to networks and workloads utilizing policy informed by continuous, contextual, risk-based verification across users and their associated devices. Zero Trust advocates these three core principles: All entities are untrusted by default; least privilege access is enforced; and comprehensive security monitoring is implemented."

Forrester's Zero Trust Model of information security is a conceptual and architectural model for how security teams should redesign networks to have secure microperimeters, strengthen data security using obfuscation techniques, limit the risks associated with excessive user privileges and access, and dramatically improve security detection and response with analytics and automation.

Forrester also releases the "Zero Trust eXtended (ZTX) Ecosystem", which is a framework for controlling the evolution of the zero trust ecosystem. This more in-depth look into zero trust gives security professionals a complete reference point for identifying which tools and technologies are available in this domain, and which they should use for their security operations needs. It also periodically releases updates to this framework and lists prominent vendors in the zero trust ecosystem.

International Organization for Standardization (ISO)

While the International Organization for Standardization (ISO) does not provide specific guidance on zero trust design and implementation, the core elements of the ISO 27001 standard, such as risk assessment, access control, and continuous improvement, align very well with the key tenets of zero trust: no implicit trust, always verify requests, use of least privilege permissions, and continuous logging and monitoring.

Least privilege-based access control
: Limiting a user's access permissions to the bare minimum is in line with the ISO 27001 access restriction policy (Annex A.9). As a result of this policy, organizations may decrease their attack surface and mitigate the risk of unauthorized access.

Network security with no implicit trust
: The ISO 27001 communications security policy (Annex A.13) recommends network segmentation to contain potential breaches and prevent unauthorized access. This is consistent with the zero trust core principle of no implicit trust (just because you are inside the network, you are not automatically trustworthy), and is typically implemented through schemes such as microsegmentation.

Continuous logging and monitoring

The ISO 27001 logging and monitoring policy (Annex A.12.4) is intended to guarantee that organizations keep track of security and audit events in a non-refutable fashion. This is also a crucial requirement for zero trust, which demands constant logging to keep track of and monitor activities in the organization, as well as the ability to respond to them in a timely manner to guarantee that the security posture does not deteriorate over time.

Commercial Vendors

In recent years, there has been a significant influx of zero trust security products and services from cloud vendors and traditional networking product vendors alike. This is due to a combination of factors, including an increase in cyberattacks, a shift toward cloud computing, and regulatory agencies imposing harsh fines in the event of a data breach. In general, CISOs and executives are increasingly seeking zero trust solutions from vendors already present in their system, as they are typically hesitant to engage a heterogeneous mix of vendors for security needs.

The Table 11-6 list selects vendors based on two sources: those classified as leaders or challengers in Gartner's SSE Magic Quadrant (for the year 2023, which is the most recent at the time of writing) and cloud vendors based on their market share as published by Statista.

Table 11-6. Zero trust offerings from commercial vendors

Vendor	Description
Microsoft	• Zero Trust model • "Evolving Zero Trust" (whitepaper) • "Zero Trust Essentials" (ebook) • Microsoft Zero Trust Maturity Assessment Quiz
Google	BeyondCorp is Google's zero trust model implementation and can be enabled in any organization through BeyondCorp Enterprise
Amazon	• Amazon's Zero Trust approach: "Zero Trust on AWS" • "Zero Trust architectures: An AWS perspective"
IBM	• "Zero trust security solutions" • "Getting started with zero trust security" • "Protect the Hybrid Cloud with Zero Trust"
Alibaba	• "Alibaba Cloud Service Mesh: Overview of zero trust security" • "Zero-Trust Security – Part 1: How Is Zero-Trust Security Helpful for the Cloud?" • "Zero-Trust Security – Part 2: Getting Started with Zero-Trust Security" • "Zero-Trust Security – Part 3: Zero-Trust Security with Cloud-Native Microservices and Containers"
Salesforce	• About Salesforce security • "Zero Trust: Securing Your Business for Tomorrow"
Oracle	• "Zero-trust security model" • "Approaching Zero Trust Security with Oracle Cloud Infrastructure"

Vendor	Description
Netskope	• "Netskope Reference Architecture for Zero Trust" • "What is Zero Trust?"
ZScaler	"What Is Zero Trust?"
Palo Alto Networks	"Zero Trust with Zero Exceptions"
Cisco	"Zero trust security"

Summary

In this chapter, we discussed a variety of publications that provide zero trust reference architectures, maturity models, principles, guidelines, and implementation guidance from a variety of institutions ranging from governments to public and private organizations. Zero trust is a highly dynamic and continuously growing field, and it is recommended that you use the links and references supplied throughout this chapter to look deeper into particular aspects of zero trust.

In the next and final chapter of this book, you will learn about common challenges faced by organizations implementing the zero trust initiative. It also discusses the potential impact of technological advancements such as artificial intelligence, quantum computation, and privacy-enhancing technologies that are useful for zero trust security.

CHAPTER 12

Challenges and the Road Ahead

In this concluding chapter, you will learn about the challenges commonly encountered when implementing zero trust initiatives, as well as the impact of new technological developments, such as quantum computing, artificial intelligence, and privacy-enhancing technologies. The goal is to provide a high-level view of the technical and functional challenges associated with zero trust initiatives in a vendor-neutral manner, as well as to discuss emerging technologies that will ultimately impact both zero trust and organizations' broader cybersecurity efforts.

Challenges

This section discusses the common challenges encountered during the implementation of a zero trust initiative and provides a few recommendations for dealing with them. As you delve into the subject matter, it becomes evident that the challenges include not only technical factors but also functional issues, requiring potential cultural and process-level improvements.

Mindset Shift

Implementing a zero trust initiative requires a paradigm shift in security philosophy. It necessitates a shift away from traditional perimeter-based security to an "always assume breach" mindset. However, when dealing with a large organization with a decade of legacy IT architecture and practices, the implementation of zero trust principles turns out to be challenging.

For most organizations, the zero trust initiative is a new endeavor that requires full backing and buy-in from the executive leadership. To effectively handle the challenges that are encountered when executing a zero trust initiative, all key stakeholders

within the organization must examine their core objectives and incorporate zero trust related activities into their team or group specific objectives and key results (OKRs).

Furthermore, it is recommended to use an iterative approach while implementing a zero trust initiative within an organization. While there may be consensus among the key stakeholders that embracing a zero trust approach can enhance security posture, it may be necessary to prioritize the implementation process for businesses or groups within organizations that exhibit a higher level of receptivity to change and fresh ideas. You may want to refer to the maturity models discussed in Chapter 11 to gain a more thorough understanding of the various zero trust implementation modalities.

Shadow IT

Shadow IT is the use of IT-related hardware, software, and cloud services that are not owned and governed by an IT organization. Shadow IT means that the configuration management database (CMDB) won't have a complete record of devices, software, services, etc., that are used within an organization and are required to be monitored and updated with patches. This also leads to situations where data is created, managed, and shared internally or even externally without any checks and balances from the organization. In a nutshell, organizations with shadow IT face severe regularity, compliance, operational, and security repercussions.

Shadow IT poses a significant obstacle to the successful implementation of a zero trust initiative. This is primarily due to the difficulty of verifying vital information such as device, application, and agent details, as there is no central location where this information is stored. These pieces of information are also necessary for zero trust components such as the policy engine and trust engine to effectively evaluate the access request and make a determination. Moreover, ensuring continuous monitoring and implementation of policy enforcement poses a significant challenge.

Additionally, users are typically unaware of the effects of shadow IT and may encounter challenges. For instance, in order to improve security posture, organizations may ask users to perform step-up or adoptive authentication, such as multifactor authentication (MFA), when signing in from devices or when accessing endpoints that are not part of the CMDB. This may have a negative impact on their productivity, as they may be prompted for MFA more frequently until the CMDB is correctly updated.

In order to tackle shadow IT, organizations may want to put in place a cloud access security broker (CASB), which aids in the management of shadow IT. As per Gartner:

> Cloud access security brokers (CASBs) are on-premises, or cloud-based security policy enforcement points, placed between cloud service consumers and cloud service providers to combine and interject enterprise security policies as the cloud-based resources are accessed.

With CASB, organizations gain visibility into traffic and application utilization patterns, including endpoints, which enables them to comprehend the scale of their shadow IT footprint.

Siloed Organizations

Zero trust endeavors require a high degree of collaboration among the teams across the organization, such as engineering, operations, support/customer care, and compliance. Large organizations develop siloed processes over time for managing teams, resulting in muddled communication regarding roles, responsibilities, and ownership. The more siloed an organization is, the more difficult it is to carry out an initiative like zero trust without being sidetracked by day-to-day politics. Ultimately, the final outcomes are constrained by the organizational structure, as indicated by Conway's law. It is critical that communication channels exist among teams without red tape.

When dealing with siloed organizational structures, there are two areas to pay special attention to. First, relevant leaders from various divisions, such as engineering, infrastructure, customer service, compliance, and so on, must be on the same page and agree to rally behind the initiative at the highest level. Second, it is also critical that an ongoing cadence be established to review obstacles and any roadblocks, such as technical issues, budgetary challenges, upskill/training requirements, and so on. If these issues are not tracked, they will never be addressed.

Another consideration is to prevent the "too many chefs in the room" situation. While it is absolutely critical for leadership and necessary teams to be aligned, just allocating more resources to the problem is unlikely to produce the intended results. This generally happens when organizational culture is such that, while under pressure, teams capitulate and, rather than finding a resolution based on active collaboration among existing teams and applying cerebral thinking to the problem, leaders are inclined toward adding more resources. This may actually have a negative impact on overall progress as it will increase management time and overhead. A better approach is to define the goals clearly and create ownership with accountability based on the subject matter expertise of the teams, and if there is a contentious issue, employ approaches like disagree and commit to get over that hurdle.

Lack of Cohesive Zero Trust Products

As the zero trust model gains traction in the security industry, a plethora of products and services targeting zero trust initiatives have entered the market.

Organizations undertaking zero trust initiatives have to deal with a wide array of vendors offering competing products and services with little to no cohesion and harmony among them. This creates both functional and technical challenges, such as usage of proprietary data formats, lack of well-defined APIs for backend connectivity,

and wildly different UX. Operationally, having a broad range of products and services involves dealing with distinct service-level agreements (SLAs) and licensing, which makes administration complex. Additionally, this also frustrates and creates friction for end users since they have to frequently switch between various products, which eventually makes it more difficult for the organization to implement the zero trust initiative.

Organizations should examine frameworks such as Gartner's security service edge (SSE) to gain a more thorough comprehension of the vendor ecosystem and its maturity. For more information, you should review the SSE Magic Quadrant and SSE vendor platform and service reviews.

Scalability and Performance

It is common for IT teams to support architecture that is conducive to the various workstyles of employees as well as infrastructure required to support essential business services. Zero trust implementation requires highly scalable and performant control and data planes. For instance, the trust engine and policy engine are critical decision-making components that must be able to scale and perform within the quality of service (QoS) parameters defined by the organization.

This is why it is necessary to carry out adequate testing to ensure that zero trust architecture can withstand scalability and performance requirements, can dynamically scale up or down as required, and meets or exceeds the SLA. For example, running a simulation based on the anticipated workload and calibrating the components is critical from an operational aspect and will also instill confidence in the organization's early adopters of the zero trust architecture.

Key Takeaways

While it is difficult to provide prescriptive guidelines and a list of tasks to pursue when embarking on a zero trust initiative, a number of high-level considerations can help set the ship on its proper course:

- Identify the organization's key drivers and use cases, and also outline business case scenarios and policies.
- Define a zero trust implementation roadmap that clearly defines all stages as well as the risks associated with them.
- Perform an audit of existing infrastructure and technologies to determine technical debt.
- Analyze legacy technologies, outdated protocols, products, and so on that may need to be replaced or updated as part of the zero trust implementation.

- Conduct a security posture analysis to determine where the organization's risks are. This encompasses penetration testing, threat modeling, auditing, and other similar activities.

Technological Advancements

There are numerous technological advances that already have or will soon have a significant impact on security. In this section, you'll learn about the effects that quantum computing, artificial intelligence, and privacy-enhancing techniques are likely to have your enterprise's security posture and, ultimately, the zero trust initiative.

Quantum Computing

Modern IT infrastructure relies heavily on public key infrastructure (PKI) to secure and protect data, assets, communications, etc. For example, the use of X.509 certificates to verify identities and secure data, both in transit, is just one example of how prevalent and important PKI is in today's security infrastructure. The security of the underlying cryptographic algorithms used by PKI is based on the fundamental assumption that certain mathematical problems are computationally very hard and complex to solve in a reasonable amount of time, even for the most powerful supercomputers. For example, the security of the asymmetric algorithm keys such as the Rivest-Shamir-Adleman (RSA) algorithm, one of the most widely used algorithms today, depends on the difficulty of factoring the product of two very large prime numbers. The underlying theme is that it will take an inordinate amount of time for adversaries to crack these keys using today's standard computing technology.

However, as quantum computing gains traction and becomes more imminent with each passing year, using computationally hard mathematical problems as a method to secure keys is called into question. For instance, Shor's algorithm, a quantum algorithm, has the capability to solve large integer factorization problems in polynomial time. It means that quantum computers will be able to solve difficult mathematical problems such as prime number factorization in a fraction of the time required by the most advanced computational machines that are available today. Essentially, quantum computing is an imminent threat to the security of RSA and other widely used asymmetric cryptographic algorithms employed to secure infrastructure today.

Quantum Computing's Impact on Data Confidentiality

Consider a scenario in which an adversary gains access to data encrypted with an algorithm that is not quantum resistant (such as RSA). Since the data is encrypted, it cannot be decrypted and read instantaneously. However, an adversary with access to a quantum computer can decrypt the data at a later time. Since many organizations, such as critical government agencies, financial institutions, and health care providers, etc., have regulatory and compliance requirements to store data for extended periods of time and keep it confidential, quantum computers represent a threat to data even before they have been built.

Active research has been conducted to address the quantum computing threats to current cryptographic algorithms. Post-quantum cryptography (PQC) and quantum key distribution (QKD) are two prevalent approaches to quantum-resistant cryptography. The PQC takes an approach to the development of novel cryptographic schemes that does not require the factorization of large prime numbers, can be executed on modern computers, and is resistant to cryptanalytic attacks from both classical and quantum computers. QKD, on the other hand, creates a cryptographic key utilizing the fundamental principles of quantum mechanics, making it the theoretically most secure technique for protecting against threats from both regular computers and quantum computers.

Participants in the development of these two approaches believe they will coexist for some time, primarily for practical reasons. PQC, for example, has a lower barrier to entry because it is primarily software based, does not require quantum mechanics, and can leverage existing infrastructure, whereas QKD does require specialized hardware as it relies on quantum mechanics. Nonetheless, depending on the use case, QKD may be preferred to PQC for highly sensitive applications.

Organizations working toward implementing zero trust architecture need to be cognizant of the fact that cryptographic algorithms that are considered relatively secure today may become insecure when quantum computing becomes accessible. Thus, it is crucial to remain up to date on the risks and benefits of quantum computing. The following publications from various leading institutions will help you understand the landscape of quantum computing and its impact on cybersecurity:

- European Union Agency for Cybersecurity (ENISA)—"Post-Quantum Cryptography: Anticipating Threats and Preparing the Future"
- National Institute of Standards and Technology (NIST)—"Post-Quantum Cryptography"
- United States White House—"Executive Order on Enhancing the National Quantum Initiative Advisory Committee"

- United States Cybersecurity & Infrastructure Security Agency (CISA)—"Quantum-Readiness: Migration to Post-Quantum Cryptography"
- United States Department of Homeland Security (DHS)—"Post-Quantum Cryptography"
- United States National Security Agency (NSA)—"The Commercial National Security Algorithm Suite 2.0 and Quantum Computing FAQ"

Artificial Intelligence

The significance of artificial intelligence (AI) in today's world cannot be overstated. It affects every industry and business segment, and its utilization and adoption will only increase over time. When it comes to cybersecurity, AI can be a double-edged sword for organizations implementing zero trust initiatives. It can be a significant productivity accelerator as well as an enabler to improve security posture. However, it may also provide attackers with more sophisticated tools for their attack, leaving organizations vulnerable to a broadened attack surface area if AI-based security threats are not addressed properly.

The following are some of the key AI-enabled capabilities that organizations implementing zero trust initiatives should evaluate and plan to capitalize on, as these are directly related to the strengthening of security posture. It is recommended to use AI to perform the following functions:

- Enable the detection of and protection from zero-day attacks and anomalous/malicious behavior patterns, and provide complementary heuristic and signature-based approaches.
- Assist organizations in improving their privacy and data classification processes by automatically discovering and identifying sensitive data and enterprise digital assets.
- Enable security teams to reduce time spent on hunting for threats and respond quickly to critical security incidents while also helping with root cause analysis (RCA). Generative AI can help with this by providing assistance through a prompt-based natural language interface.
- Simplify configuration management and access management policies to make them faster and more optimal. Moreover, use AI to detect blind spots and security weaknesses in policies that might otherwise be difficult to discover.

AI also poses a range of security threats that must be carefully evaluated. Here is a nonexhaustive list:

- Deepfakes use AI to supplant the likeness of one person in audio or video with the likeness of another person. They can be an extremely effective tool for social engineering-based attacks.
- AI models are trained on massive datasets, raising issues about data privacy and security because they may include the processing of sensitive personally identifiable information (PII), which must be treated with prudence in order to comply with data privacy regulations.
- Organizations must implement AI responsibly. The use of AI in the context of cybersecurity must be dependable, secure, and ethical. To ensure that the results generated by AI are reliable and consistent, it is necessary to comprehend how AI comes to those conclusions in the first place. However, AI models can be quite complex and difficult to interpret, making it difficult for security teams to comprehend how the AI arrives at its conclusions, and thereby making it difficult to explain them to broader stakeholders. Responsible AI is a field where these issues are discussed, and leaders must stay current with it in order to keep up with the most recent developments in that discipline.
- AI models take time to mature and may occasionally mistake legitimate/safe requests for threats or anomalies. This results in false positives, which leads to wasted effort from security teams, as they need to review and then mark these requests as false positives. Moreover, this creates friction with end users, who are on the receiving end of these false positives. Having an AI model with a feedback loop typically decreases the probability of false positives over time as the model can learn from the feedback.
- AI models are susceptible to attacks that can generate adversarial examples, causing the AI model to misclassify inputs and diminishing its efficacy. This risk is amplified in the era of generative AI with large language models (LLMs) such as GPT, LLaMA, etc., which can generate highly realistic synthetic data that may be difficult for AI security systems to distinguish from authentic data. Another rising threat to AI models is data poisoning, which involves tampering with the data used to train the models, resulting in back doors exploited by adversaries.

AI is a rapidly evolving field, and these resources will help you learn more about AI-based security risks and ethical responsibilities:

- "NIST AI Risk Management Framework"
- European Commission—"High-level expert group on artificial intelligence," ethics, policy, and investment guidelines and recommendations

- European Union Agency for Cybersecurity—"Multilayer Framework for Good Cybersecurity Practices for AI"
- Republic of China—interim administrative measures for generative artificial intelligence
- The Open Worldwide Application Security Project—"OWASP Top 10 for LLMs"

Privacy-Enhancing Technologies

Zero trust places great emphasis on ensuring data is protected at all times from unauthorized access. However, in today's world of Bring Your Own Device (BYOD), remote work, and the wide array of applications through which users share their data, this poses a significant challenge: how to preserve the privacy of data and ensure it is protected at all times—at rest, in transit, and while in use. Although tools and technologies for protecting data at rest and in transit have matured significantly, protecting data while in use (such as when performing a computation) and when shared with multiple parties for processing remains a significant challenge. This is where privacy-enhancing technologies (PETs) come into play. PETs assist organizations in obtaining cryptographic assurance that their data is protected at all times, even when in use, and cannot be accessed by the cloud provider or any other unauthorized entity. PETs strive for data protection by design and ensure this protection at all times, both when it is being computed and when it is being used by multiple parties.

PET Terminology

PETs are a rapidly developing area, so definitions and terminologies are still evolving, and terms like privacy-preserving and privacy-enhancing are sometimes used interchangeably. If you want to learn more about PETs and their use cases, review a report from the White House titled "National Strategy to Advance Privacy-Preserving Data Sharing and Analytics", which provides an accessible summary of various privacy-preserving technologies. Also, publications available from NIST and EU ENISA can be very useful in exploring PETs.

Many organizations transition to the public cloud as an important part of their digitization efforts, but are also in the process of implementing zero trust architecture. Organizations utilizing the public cloud must rely on their cloud provider's assurances that it will not access their data. For instance, PETs like homomorphic encryption ensure that data is protected via encryption even when computation is carried out within the public cloud infrastructure (e.g., virtual machines, containers, etc.). Another example is confidential computing, which provides attestation backed by cryptographic proof that both data and code reside in a highly secure Trusted Execution Environment (TEE, a distinct chip within the CPU/GPU), and that they

are always encrypted and protected from any unauthorized access. Confidential computing is also very useful in establishing both secure boot operations and ensuring via attestation capability that applications operate within the expected parameters on specific hardware and software platforms.

The following is a short list of PETs that organizations implementing zero trust should review in order to improve data privacy:

Homomorphic encryption
: This allows you to compute encrypted data without first decrypting it. The computations themselves are encrypted as well. Once decrypted, the output is identical to what would have been produced if the computation had been conducted on the original plain-text data.

Confidential computing
: This involves the protection of data in use by conducting computation in an attested hardware-based Trusted Execution Environment.

Secure multiparty computation (SMPC)
: This technique allows at least two different parties to jointly process their combined information without requiring each party to share all of its information with the other parties.

Zero-knowledge proof (ZKP)
: This is any procedure in which a prover (typically a person) can prove to another party (verifier) that they have a secret (something they know but the verifier does not).

Summary

This chapter began with a discussion about the obstacles that organizations face when implementing zero initiatives, such as the shift in mindset required by leadership in order to embrace zero trust security. It also addressed shadow IT, which is prevalent in large organizations, conceals the actual use of IT infrastructure and applications, and serves as an impediment to the implementation of zero trust initiatives.

It also addressed the obstacles posed by siloed organizations and their ineffective cross-organization collaboration patterns, which impede the implementation of zero trust initiatives, as zero trust requires that teams collaborate in harmony. In the absence of a consolidated and cohesive set of zero trust tools, organizations need to evaluate an excessive number of vendors, which presents its own set of challenges, such as a lack of standard API offerings and incompatible UX, as well as disparate licensing models.

As components of zero trust architecture in both the control plane and the data plane necessitate communication patterns that may not be typical of existing systems, scalability and performance issues may also be encountered by organizations.

Finally, you learned about cutting-edge technological innovations such as quantum computing, which will have a significant impact on public key infrastructure in particular, as well as artificial intelligence and privacy-enhancing technologies, which play a crucial role in enhancing data protection. All of these areas are crucial for any organization to closely monitor and remain current on, both from a zero trust perspective and from a more general cybersecurity standpoint.

As we conclude this book, we would like to congratulate you on reading it and sincerely hope that you now have a firm understanding of zero trust security. We encourage you to put these concepts into practice!

We will leave you with a memorable line from the film *The Matrix*:

> I can only show you the door, you're the one that has to walk through it.
>
> —Morpheus

APPENDIX

A Brief Introduction to Network Models

Networking stacks have many different responsibilities in transmitting data over a network. As such, it would be easy for a networking stack to become a jumbled mess of code. Therefore, the industry long ago decided to spend the effort to clearly define a set of standardized layers in a networking stack. Each layer is responsible for some portion of the job of transmitting data over the wire. Lower layers deliver functionality and guarantees to higher layers in the stack.

Building up these layers isn't just useful for organizing code. These layer definitions are often used to describe where new technology operates in the stack. For example, you might have heard of a layer-7 or layer-4 load balancer. A load balancer distributes traffic load across a set of backend machines, but the layer at which it operates greatly determines its capabilities. A layer-7 load balancer, for example, can make decisions about where to route traffic based on details in an HTTP request, like the requested path or a particular header. HTTP operates at layer 7, so this data is available to inspect. A layer-4 load balancer, by contrast, does not consider layer-7 data, and therefore can only pass traffic based on simpler connection details, like the source IP and port.

There are many different network models. Most of these models can be roughly mapped to equivalents in other network models, but sometimes the boundaries can be a bit fuzzy. For this book, we will only focus on two network models: the Open Systems Interconnection (OSI) network model and the TCP/IP (internet protocol suite) network model. Understanding the boundaries of these two models will help in later discussions about where zero trust responsibilities should be handled in the network model.

Network Layers, Visually

The idea of a layer might be strange at first, though a simplistic way to understand the concept is by comparing them to Russian nesting dolls. Each layer typically contains the next, encapsulated by it in a section known as the payload (Figure A-1).

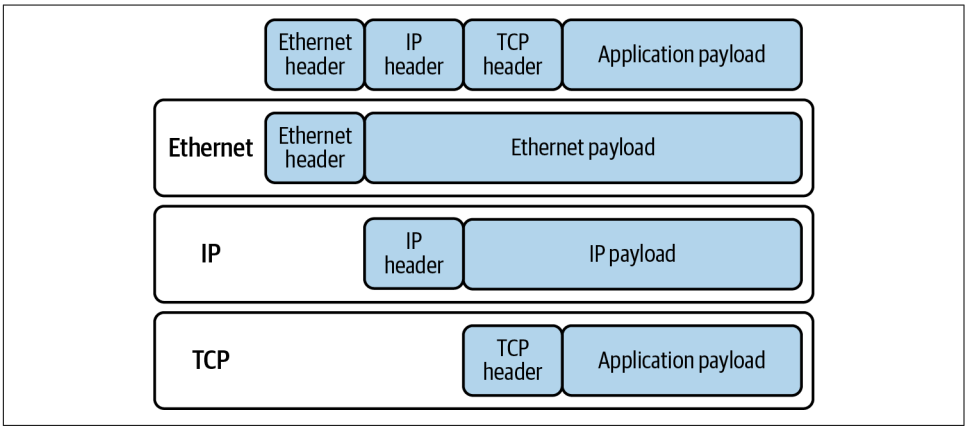

Figure A-1. Lower network layers transport higher-layer traffic in their payload fields, creating a nested structure inside a single packet

OSI Network Model

The OSI network model was published in 1984 after being merged from two separate documents started several years earlier. The model has been published by two separate standards bodies: the International Organization for Standardization (ISO) published ISO 7498, while the Telecommunication Standardization Sector of the International Telecommunication Union (ITU-T) published X.200.

The model itself is extracted from experiences of building several networks at the time, ARPANET (the Advanced Research Projects Agency Network) being the most well known. The model defines seven distinct layers (explained in the following sections), each of which owns a portion of the responsibilities for transmitting data.

Layer 1—Physical Layer

The physical layer is defined as the interface between a network device and the physical medium over which network transmission occurs. This can include things like pin layout, line impedance, voltage, and frequency. The parameters of the physical layer (sometimes referred to as a PHY) depend on the kind of medium used. Twisted pair, coaxial cabling, and radio waves are examples of media in common use today.

Layer 2—Data Link Layer

The data link layer is responsible for the transmission of data over the physical layer. This layer only considers data transmission between directly connected nodes. There is no concept of transmission between interconnected networks. Ethernet (802.3) is the most well-known protocol operating at this layer.

Layer 3—Network Layer

The network layer is responsible for transmitting data packets between two interconnected nodes. At this layer, packets might need to transverse multiple layer-2 segments to reach their destination, so this includes concepts to allow routing data to its destination by inspecting a destination address. IP is often said to operate at this layer, but the boundaries can be a bit fuzzy, as we will explore later.

Layer 4—Transport Layer

The transport layer builds upon the simple packet transmission capabilities of layer 3, usually as an intermediary protocol designed to augment layer 3 with many desirable services:

- Stateful connections
- Multiplexing
- Ordered delivery
- Flow control
- Retransmission

These services might look similar to the services that a protocol like TCP provides. In fact, TCP is a layer 4 protocol; however, in a way similar to IP, this association can be a bit awkward.

Not all of these services need to be provided by a protocol operating at this level. UDP, for example, is a layer 4 protocol that offers only one of these services (multiplexing). It remains a layer 4 protocol because it is an intermediary protocol that is directly encapsulated by layer 3.

Layer 5—Session Layer

The session layer isn't commonly discussed in most networks. This layer provides an additional layer of state over connections, allowing for a communication resumption and communication through an intermediary. Several VPNs (PPTP, L2TP) and proxy protocols (SOCKS) operate at this layer.

Layer 6—Presentation Layer

The presentation layer is the layer that application developers will most commonly interact with. This layer is responsible for handling the translation between application data (often represented as structural data) and transmittable data streams. In addition to this serialization responsibility, this layer is often responsible for cross-cutting concerns like encryption and compression. TLS is a well-known protocol operating at this layer, though it operates at layer 6 only after the session is established (which happens at layer 5—the process of changing from a lower layer to a higher layer is sometimes referred to as an upgrade).

Layer 7—Application Layer

The application layer is the highest layer in the OSI model. This layer provides the high-level communication protocols that an application uses to communicate on the network. Some common protocols at this layer are DNS, HTTP, and SSH.

TCP/IP Network Model

The TCP/IP network model is another important network model. This model deals with the protocols most often found on the internet today.

Unlike the OSI model, the TCP/IP model does not try to define strict layers with clear boundaries. In fact, RFC 3439 (*https://www.ietf.org/rfc/rfc3439.txt*), which documents the "philosophical guidelines" that internet architects use, has a section entitled "Layering Considered Harmful." Still, the model is said to define the following rough layers, from lowest to highest:

- Link layer
- Internet layer
- Transport layer
- Application layer

These layers can be roughly mapped to the OSI model, but the mappings are not perfect. The application layer roughly covers layers 5–7 in the OSI model. The transport layer roughly maps to layer 4, though its introduction of the concept of a port gives it some layer 5 characteristics. The internet layer is similarly generally associated with layer 3. The abstraction is leaky, however, as higher-level protocols like ICMP, or Internet Control Message Protocol (which are transmitted via IP), concern themselves with details of how traffic is routed around the internet.

Index

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

About the Authors

Razi Rais is a cybersecurity leader with more than 20 years of expertise in building and running secure and resilient systems. He has been working with Microsoft for over a decade, holding positions such as software engineer, architect, and product manager. His current focus at Microsoft is on building cutting-edge cybersecurity products and services. Razi is also a lead author of several books, including *Azure Confidential Computing and Zero Trust* (O'Reilly), *Microsoft Identity and Access Administrator* (Microsoft Press), and *Programming Microsoft's Clouds* (Wrox Press). In addition to being an active member of the GIAC Advisory Board, he speaks frequently at international conferences like RSA and conducts workshops and training sessions on platforms such as O'Reilly and LinkedIn. You can contact him on LinkedIn or visit his website.

Christina Morillo is an accomplished enterprise information security and technology leader with over two decades of practical experience building and leading comprehensive information security and technology programs. Her skill and expertise have landed her roles at organizations such as Microsoft and Morgan Stanley, and she currently leads information security for an NFL sports team. Christina's impact extends beyond her enterprise security work. She is a speaker and the author of *97 Things Every Information Security Professional Should Know* and *The Future of Security* (both published by O'Reilly). Christina has also contributed to and been featured in a variety of industry publications. In addition, she serves as a Fellow and Advisor at New America for the #ShareTheMicInCyber Initiative, showcasing her commitment to the broader security community. For more on her professional journey and insights, visit *https://bio.site/christinamorillo* and *https://www.christinamorillo.com*.

Evan Gilman is the cofounder and CEO of SPIRL, the workload identity company. With roots in academia and a background in operations engineering and computer networks, he has been building and operating systems in hostile environments his entire professional career. An open source contributor, speaker, and author, Evan is passionate about designing systems that strike a balance with the networks they run on.

Doug Barth is a software engineer who loves to learn and shares his knowledge with others. In his over 20 years of professional experience, he has worked as both an infrastructure and product engineer at companies like SPIRL, Stripe, PagerDuty and Orbitz. He has built and spoken about monitoring systems, mesh networks, and failure injection practices.

Colophon

The animal on the cover of *Zero Trust Networks* is a squat lobster, a type of crustacean found in the *Galatheoidea* and *Chirostyloidea* superfamilies; there are over 1,000 species, most of which spend their lives on the sea floor. Despite their name, squat lobsters are more closely related to hermit crabs than lobsters.

The squat lobster does not have a shell on its back, and protects itself by squeezing into crevices or under rocks. Its claws remain out, ready to fend off predators, defend its territory, and scavenge for food floating by or buried in the sand. A squat lobster's arms can grow to be many times longer than its body. These crustaceans do appear similar to lobsters, with a segmented thorax and large claws, but are generally flatter and smaller.

The meat of squat lobsters is known as langostino (derived from the Spanish word for lobster, *langosta*). It is often used in seafood dishes as a less expensive alternative to lobster.

Many of the animals on O'Reilly covers are endangered; all of them are important to the world.

The cover illustration is by Jose Marzan, based on an antique line engraving from *Pictorial Museum of Animated Nature*. The series design is by Edie Freedman, Ellie Volckhausen, and Karen Montgomery. The cover fonts are Gilroy Semibold and Guardian Sans. The text font is Adobe Minion Pro; the heading font is Adobe Myriad Condensed; and the code font is Dalton Maag's Ubuntu Mono.