# Automating Post Exploitation with Metasploit

Meterpreter Scripts and Post Modules

## 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.

# The Post Mixin

- To aid in the creation of Post Exploitation module and Meterpreter scripts the Post Exploitation Mixin was created
- There are currently mixins for:
  - Common Mixin all platforms
  - File Mixin all Platform
  - Windows Mixin both Shell and Meterpreter(not all calls only registry for now)
  - Linux Mixin
  - Solaris Mixin
  - OSX Mixin

# The Post Mixin - Common

- The common mixin has the calls
  - **`cmd_exec(cmd, opts, time_out)`** - this call will execute a system shell command and return a multi line string value. On Meterpreter the command will be hidden and channelized.
  - The Options are the command to execute, the options for the command and a timeout in seconds to wait, default is 15 seconds.

# The Post Mixin - File

- The File mixin has the calls
  - **`file_local_digestmd5(file2md5)`** - Returns a string with MD5 checksum of a given local file
  - **`file_local_digestsha1(file2sha1)`** - Returns a string with SHA1 checksum of a given local file
  - **`file_local_digestsha2(file2sha2)`** - Returns a string with SHA256 checksum of a given local file
  - **`file_local_write(file2wrt, data2wrt)`** - Writes a given string to a specified file
  - **`read_file(file_name)`** - Retuns the content of file in the target host as a string
  - **`write_file(file_name, data)`** - Platform-agnostic file write. Writes given object content to a remote file.Returns Boolean true if successful
  - **`append_file(file_name, data)`** - Platform-agnostic file append. Appends given object content to a remote file. Returns Boolean true if successful

# The Post Mixin - Windows Accounts

- The Windows Account mixin has the call
  - **`delete_user(username,server)`** - Deletes a user account from the given server (or local if none given)
    - Returns a hash with the following keys:
      - **`:success`** - Everything went as planned
      - **`:invalid_server`** - The server name provided was invalid
      - **`:not_on_primary`** - Operation allowed only on domain controller
      - **`:user_not_found`** - User specified does not exist on the given server
      - **`:access_denied`** - You do not have permission to delete the given user
    - Returns nil if not successful

  - **`resolve_sid(sid, system_name)-`** Retrieves the name, domain, and type of account for the given sid
    - Returns a hash with the following keys
      - **`:name`** - account name (e.g. "SYSTEM")
      - **`:domain`** - domain where the account name was found. May have values such as the work station's name, BUILTIN, NT AUTHORITY, or an empty string
      - **`:type`** - one of :user, :group, :domain, :alias, :well_known_group,deleted_account, :invalid, :unknown, :computer
      - **`:mapped`** - There was a mapping found for the SID

# The Post Mixin - Windows Priv

- The Windows Privilege(priv) mixin has the calls
  - **`is_admin?() -`** Returns true if user is admin and false if not.
  - **`is_system?() -`** Returns true if running as Local System
  - **`is_uac_enabled?()-`** Returns true if UAC is enabled,Returns false if the session is running as system, if uac is disabled or if running on a system that does not have UAC

# The Post Mixin - Windows Eventlog

- The Windows Eventlog mixin has the calls
  - **`eventlog_clear(evt)-`** clears a given eventlog or all eventlogs if none is given. Returns an array of eventlogs that where cleared.
  - **`eventlog_list()-`** enumerate eventlogs returning an array with the names of the eventlog on the system

# The Post Mixin - Windows Registry

- The Windows Registry mixin has the calls
  - **`registry_createkey(key) -`** Create the given registry key
  - **`registry_deletekey(key) -`** Delete a given registry key, returns true if successful
  - **`registry_deleteval(key, valname) -`** Deletes a registry value given the key and value name, returns true if successful
  - **`registry_enumkeys(key) -`** Return an array of subkeys for the given registry key
  - **`registry_enumvals(key) -`** Return an array of value names for the given registry key
  - **`registry_getvaldata(key, valname) -`** Return the data of a given registry key and value
  - **`registry_getvalinfo(key,valname)-`** Return the data and type of a given registry key and value
  - **`registry_setvaldata(key, valname, data, type) -`** Sets the data for a given value and type of data on the target registry, returns true if successful
  - **`registry_loadkey(key,file)-`** Will load a specified hive file and return the path of where it was loaded
  - **`registry_unloadkey(key)-`** Will unload a loaded registry hive file given its path

# The Post Mixin - Windows Profile

- This mixin is best used when running and SYSTEM and access to the profiles of other users on the target is needed
- The Windows Profile mixin has the call
    - **`grab_user_profiles()`** – Returns an array of hashes for each user profile found. the keys are:
        - **`UserName`** **-** User Username
        - **`SID`** **-** User SID
        - **`ProfileDir`** **-** User profile directory
        - **`AppData`** **-** User Application Data directory
        - **`LocalAppData`** **-** User Local Application Data directory
        - **`LocalSettings`** **-** User Local Settings directory
        - **`Desktop`** **-** User Desktop directory
        - **`MyDocs`** **-** User My Documents directory
        - **`Favorites`** **-** User IE Favorites directory
        - **`History`** **-** User IE History directory
        - **`Cookies`** **-** User IE Cookies directory

# The Post Mixin - Linux System

- The Linux System mixin has the calls
  - **`get_groups()`** - Returns an array of hashes each hash representing a user group Keys are name, gid and users
  - **`get_sysinfo()`** - Returns a Hash containing Distribution Name, Version and Kernel Information
  - **`get_users()`** - Returns an array of hashes each representing a user Keys are name, uid, gid, info, dir and shell

## The Post Mixin - Linux Priv

- The Linux Priv mixin has the call
  - **`is_root?()`** - Returns true if running as root, false if not

## The Post Mixin - OSX Sys

- The OSX Sys mixin has the calls
  - **`get_groups()`** - Returns an array of hashes each representing user group on the system Keys are name, guid and users
  - **`get_nonsystem_accounts()`** - Returns an array of hashes each representing non system accounts on the system Keys are name, gid, uid, dir and shell
  - **`get_sysinfo()`** - Return a hash with system Information
  - **`get_system_accounts()`** - Returns an array of hashes each representing a system accounts on the system Keys are name, gid, uid, dir and shell
  - **`get_users()`** - Returns an array of hashes each representing a user on the system Keys are name, gid, uid, dir and shell

## The Post Mixin - Solaris Priv

- The Solaris Priv mixin has the calls
  - **`is_root?()`** - Returns true if running as root, false if not

# The Post Mixin - Solaris System

- The Solaris System mixin has the calls
  - **`get_groups()`** - Returns an array of hashes each hash representing a user group Keys are name, gid and users
  - **`get_sysinfo()`** - Returns a Hash containing Distribution Name, Version and Kernel Information
  - **`get_users()`** - Returns an array of hashes each representing a user Keys are name, uid, gid, info, dir and shell

## The Post Mixin - Solaris System

- Loading the mixins in post modules
  - Files are required before declaring the post module subclass
  - Includes are after the declaration of the subclass

-
  ```
  require 'msf/core'
  require 'rex'

  # Multi platform requiere
  require 'msf/core/post/common'
  require 'msf/core/post/file'

  class Metasploit3 < Msf::Post

      include Msf::Post::Common
      include Msf::Post::File
  ```

## The Post Mixin

- Loading the mixins in post modules
  - Files are required before declaring the post module subclass
  - Includes are after the declaration of the subclass

-
  ```
  require 'msf/core'
  require 'rex'

  # Multi platform requiere
  require 'msf/core/post/common'
  require 'msf/core/post/file'

  class Metasploit3 < Msf::Post

      include Msf::Post::Common
      include Msf::Post::File
  ```

## The Post Mixin

- Windows Mixin

-
  ```
  require 'msf/core/post/windows/eventlog'
  require 'msf/core/post/windows/priv'
  require 'msf/core/post/windows/registry'
  require 'msf/core/post/windows/accounts'

  include Msf::Post::Windows::Eventlog
  include Msf::Post::Windows::Priv
  include Msf::Post::Windows::Registry
  include Msf::Post::Windows::Accounts
  ```
- Linux Mixin

-
  ```
  require 'msf/core/post/linux/system'
  require 'msf/core/post/linux/priv'

  include Msf::Post::Linux::System
  include Msf::Post::Linux::Priv
  ```

## The Post Mixin

- OSX Mixin

-
  ```
  require 'msf/core/post/osx/system'
  require 'msf/core/post/osx/priv'

  include Msf::Post::OSX::System
  include Msf::Post::OSX::Priv
  ```
- Solaris Mixin

-
  ```
  require 'msf/core/post/solaris/system'
  require 'msf/core/post/solaris/priv'

  include Msf::Post::Solaris::System
  include Msf::Post::Solaris::Priv
  ```

## Meterpreter Scripts

# Meterpreter Scripts

- For script the best practice is to set the proper SVN Keywords, Variables and Declare the Options
- If the session object is not set as an instance variable it will need to be passed as an option to all methods or running it outside meterpreter shell will fail

-
  ```
  # $Id$
  # $Revision$
  # Author:
  #-------------------------------------------------------------------------------
  ################## Variable Declarations ##################

  @client = client
  sample_option_var = nil
  meter_type = client.platform
  @exec_opts = Rex::Parser::Arguments.new(
      "-h" => [ false, "Help menu." ],
      "-o" => [ true , "Option that requires a value"]
      )
  ```

# Meterpreter Scripts

- Ensure that the options that require values are set to true and those that do not are set to false
- Make sure you declare the variables that will be used with the options with the declaration of the options so as to make it easier to manage
- Try to avoid using `nil` as an option for boolean options
- The use of method is recommended since it will make the code easier to re-use and debug
- The use of methods will make the porting to post modules simpler

## Meterpreter Scripts

- Meterpreter scripts called from the Meterpreter Shell will have as the session object name client when using methods and executing the scripts from AutoRunScript and from outside the shell make sure that you made this object an instance variable or pass the object to the method as an option if not the script will error

-
  ```
  #### Variables ####
  @client = client
  #### Methods #####
  def get_sysinfo()
      sysinfo = @client.sys.config.sysinfo
      print_status("Running on #{sysinfo['Computer']}")
  end
  ```

## Meterpreter Scripts

- Meterpreter scripts are read and processed every time they are executed, so they can be modified and run without having to reload the framework

# Meterpreter Scripts

- All Scripts need to show a help message when given -h option
- All script must exit using by raising Rex::Script::Completed
-
```
# Usage Message Function
#-------------------------------------------------------------------------------
def usage
    print_line "Meterpreter Script for INSERT PURPOSE."
    print_line(@exec_opts.usage)
    raise Rex::Script::Completed
end
```

# Meterpreter Scripts

- Since there are several flavors of Meterpreter and many of the features that are on the windows version are not present in the checking for proper platform is important

-
  ```
  # Wrong Meterpreter Version Message Function
  #-------------------------------------------------------------------------------
  def wrong_meter_version(meter = meter_type)
      print_error("#{meter} version of Meterpreter is not supported with this Script!")
      raise Rex::Script::Completed
  end

  ################## Main ##################
  @exec_opts.parse(args) { |opt, idx, val|
      case opt
      when "-h"
          usage
      when "-o"
          sample_option_var = val
      end
  }

  # Check for Version of Meterpreter
  wrong_meter_version(meter_type) if meter_type !~ /win32|win64|java|php|linux/i # Remove none supported
  versions
  ```

## Lab

- Use the template provided and write a script called get_privs that will print true or false for:
  - –Running with Admin Token
  - –Running As System
  - –UAC Enabled or not
- Write a script that will execute a list of commands saved in an array and save the output of each command in to a file under loot

## Post Modules

## Scripts and Post Modules

- The most popular ways to perform post exploitation tasks against systems
- Post Modules will replace Meterpreter scripts in the near future
- Scripts will be supported for quite some time do to the amount of scripts left for migration and scripts in private collections

## Scripts and Post Modules

- Scripts are a great way to prototype and test ideas for post modules based on Meterpreter
- Meterpreter scripts automatically load the entire Windows Post Mixins by default so no require and includes are necessary
- Scripts are based on command options that are set by switches while modules can use global options or set options.
- Meterpreter scripts are limited by the platforms where Meterpreter is supported and the API by the version of Meterpreter

## Scripts and Post Modules

- In Post modules there is no need to do **`include Msf::Auxiliary::Report`** for reporting like in Auxiliary and Exploit modules since this is already done in the module initialization.

# Post Module

```
require 'msf/core'
require 'rex'
# Post Mixin requires go here
class Metasploit3 < Msf::Post
    # Post Mixin includes go here
    def initialize(info={})
        super( update_info( info,
                'Name'            => 'Module Name',
                'Description'     => %q{ Module Description },
                'License'         => MSF_LICENSE,
                'Author'          => [ 'Author Name <Email>'],
                'Version'         => '$Revision$', # SVN Keyword DO NOT Change
                'Platform'        => ['windows', 'linux', 'bsd', 'ios', 'solaris'],# OS to Target
                'SessionTypes'    => ['meterpreter','shell' ] # Type of Session
            ))
        # Define Module Options
        register_options(
            [

            ], self.class)
    end

    # Run Method for when run command is issued
    def run
        print_status("Running module against #{sysinfo['Computer']}")
    end
end
```

# Post Modules

- SESSION Option is set by default and always required
- Registered Options is optional
- Recommend the use of Methods and not piling everything in to the run Methods
- You can use @ Instance Variables but never @@ Class Variables in post modules

## Post Modules

- All post modules must be subclass of Msf::Post
- Keep the modules you are working on in **`~/.msf4/modules/post`**
- Modules are keeps as **`<platform>/<function>/<sub-function>`**
- Do not mix Platform Post Mixin’s

# Post Modules - Options

- The types of options that can be set on a module are:
    - OptBool - Boolean Option (true or false)
    - OptString - String value
    - OptAddress - IP Address
    - OptAddressRange - IP Address Range or CIDR
    - OptPath - Path to a File or Folder
    - OptInt - Integer Value
    - OptEnum - Selection of of Options

# Post Modules - Options

$$\underbrace{\texttt{OptEnum.new(}}_{\text{Option Type}}\underbrace{\texttt{'NAME'}}_{\text{Name}}\texttt{, }\underbrace{\texttt{[false}}_{\text{Required}}\texttt{, }\underbrace{\texttt{'Description'}}_{\text{Description}}\texttt{, }\underbrace{\texttt{nil}}_{\text{Default Value}}\texttt{, }\underbrace{\texttt{[opt1, opt2]}}_{\text{Options}}$$

$$\underbrace{\texttt{OptString.new(}}_{\text{Option Type}}\underbrace{\texttt{'NAME'}}_{\text{Name}}\texttt{, }\underbrace{\texttt{[false}}_{\text{Required}}\texttt{, }\underbrace{\texttt{'Description'}}_{\text{Description}}\texttt{, }\underbrace{\texttt{nil}}_{\text{Default Value}}\texttt{])}$$

## Post Modules - Options

```
# Standard Options, Shown win ‘show options’ console command
register_options(
            [
                OptBool.new('OPT1',   [false, 'First Option', true]),
                OptString.new('OPT2', [false, 'Second Option']),
                ........

            ], self.class)
```

## Post Modules - Options

```
# Advanced Options, Shown win 'show advanced' console command
advanced_options(
            [
                OptBool.new('ADVOPT1',   [false, 'First Option', true]),
                OptString.new('ADVOPT2', [false, 'Second Option']),
                ........

            ], self.class)
```

# Post Module

- It is important to specify the correct type of option for the data that will be required since they will be validated before running the run method
- Options are part of the Initialize Method
- Default value is not required and if none it should not be specified as nil, for OptEnum an empty string('') is recommended
- All values are stored on the datastore hash and are called as datastore['OptionName']
- The Name of options should be all uppercase letters

# Post Modules

Post Module Default Option

```
msf  post(samplemod) > show options

Module options (post/windows/gather/checkvm):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   SESSION                   yes       The session to run this ...
```

## Post Modules

Post Module Default Advanced Options

```
msf  post(samplemod) > show advanced

Module advanced options:

   Name           : VERBOSE
   Current Setting: false
   Description    : Enable detailed status messages

   Name           : WORKSPACE
   Current Setting:
   Description    : Specify the workspace for this module
```

# Post Modules

- The **`run`** method inside a Post Module will execute the main logic and code block of a module
- You can define other methods in addition to the **`run`** method that are executed by it so as to modularize and make the code simpler to debug and reusable
- If any changes to a module are made the module needs to be reloaded via the **`reload`** command if inside the msfconsole shell, the **`run`** command inside the Meterpreter shell will read and load the module on each execution

## Lab

- Migrate the scripts created in the previous lab in to post modules

Questions?