**Metasploit Exploit Development Cheatsheet**

**1) Exploit Skeleton**

**class MetasploitModule < Msf::Exploit::Remote**

**Rank = <rank>**

**include <mixin>**

**def initialize(info = {})**

**super(update\_info(info,**

**'License' => MSF\_LICENSE**

**'Name' => '<name>',**

**'Description' => %q{**

**},**

**'Author' => [<authors>],**

**'Reverences' => [[<name>, <reference>], ...],**

**'Privileged' => 'false',**

**'Platform' => 'unix',**

**'Arch' => ARCH\_CMD,**

**'DefaultOptions' => {**

**'<name>' => '<value>'**

**}**

**'DisclosureDate' => '<date>'))**

**register\_options([**

**Opt::RPORT(80),**

**])**

**def exploit**

**print\_status("#{rhost}:#{rport} Launching exploit...")**

**end**

**2) Exploit Ranking**

Table 2.1 – Exploit Rankings

| **Ranking** | **Description** |
| --- | --- |
| ExcellentRankig | The exploit will never crash the service. This is the case for SQL Injection, CMD execution, RFI, LFI, etc. No typical memory corruption exploits should be given this ranking unless there are extraordinary circumstances ([WMF Escape()](https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/windows/browser/ms06_001_wmf_setabortproc.rb)). |
| GreatRanking | The exploit has a default target AND either auto-detects the appropriate target or uses an application-specific return address AFTER a version check. |
| GoodRanking | The exploit has a default target and it is the "common case" for this type of software (English, Windows 7 for a desktop app, 2012 for server, etc). |
| NormalRanking | The exploit is otherwise reliable, but depends on a specific version and can't (or doesn't) reliably autodetect. |
| AverageRanking | The exploit is generally unreliable or difficult to exploit. |
| LowRanking | The exploit is nearly impossible to exploit (or under 50% success rate) for common platforms. |
| ManualRanking | The exploit is unstable or difficult to exploit and is basically a DoS. This ranking is also used when the module has no use unless specifically configured by the user (e.g.: [exploit/unix/webapp/php\_eval](https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/unix/webapp/php_eval.rb)). |

**3) Reference Identifiers**

Table 3.1 – Reference Identifiers

| **ID** | **Source** | **Code Example** |
| --- | --- | --- |
| CVE | cvedetails.com | ['CVE', '2014-9999'] |
| CWE | cwe.mitre.org | ['CWE', '90'] |
| BID | securityfocus.com | ['BID', '1234'] |
| MSB | technet.microsoft.com | ['MSB', 'MS13-055'] |
| EDB | exploit-db.com | ['EDB', '1337'] |
| US-CERT-VU | kb.cert.org | ['US-CERT-VU', '800113'] |
| ZDI | zerodayinitiative.com | ['ZDI', '10-123'] |
| WPVDB | wpvulndb.com | ['WPVDB', '7615'] |
| PACKETSTOM | packetstormsecurity.com | ['PACKETSTORM', '132721'] |
| URL | anything | ['URL', 'http://example.com/blog.php?id=123'] |

**4) In-Module Logging**

**def <log\_function>(msg, src = 'core', level = 0, from = caller)**

| **Parameter** | **Description** |
| --- | --- |
| msg | Mesage to print |
| src | The source of the error (default is core, as in Metasploit core) |
| level | The log level |
| from | The current execution stack |

Table 4.1 – Log levels

| **Log Level** | **Description** |
| --- | --- |
| LEV\_0 (Default) | The default log level if none is specified. It should be used when a log message should always be displayed when logging is enabled. Very few log messages should occur at this level aside from necessary information logging and error/warning logging. Debug logging at level zero is not advised. |
| LEV\_1 (Extra) | This log level should be used when extra information may be needed to understand the cause of an error or warning message or to get debugging information that might give clues as to why something is happening. This log level should be used only when information may be useful to understanding the behavior of something at a basic level. This log level should not be used in an exhaustively verbose fashion. |
| LEV\_2 (Verbose) | This log level should be used when verbose information may be needed to analyze the behavior of the framework. This should be the default log level for all detailed information not falling into LEV\_0 or LEV\_1. It is recommended that this log level be used by default if you are unsure. |
| LEV\_3 (Insanity) | This log level should contain very verbose information about the behavior of the framework, such as detailed information about variable states at certain phases including, but not limited to, loop iterations, function calls, and so on. This log level will rarely be displayed, but when it is the information provided should make it easy to analyze any problem. |

Table 4.2 – Log functions

| **Parameter** | **Description** |
| --- | --- |
| dlog() | Log DEBUG info |
| elog() | Log ERROR info |
| wlog() | Log WARNING info |
| ilog() | Log INFO info |
| rlog() | Log RAW info |

**5) Command Stagers (Msf::Exploit::CmdStager)**

***def execute\_cmdstager(flavor: “auto”, delay: 0.25, linemax: 2047)***

Should be run inside “exploit” function

| **Parameter** | **Description** |
| --- | --- |
| flavor | Cmd stager type to use (passed as “flavor: :<flavor\_name>”) |
| delay | Delay in seconds between executing each command |
| linemax | Maximum number of characters per command |

***def execute\_command(cmd, opts={})***

Should be included within the exploit body

| **Parameter** | **Description** |
| --- | --- |
| cmd | Command that is executed |
| opts | Additional options |

Table 5.1 – List of command stagers

| **Name** | **Platform** | **Description** |
| --- | --- | --- |
| vbs | Windows | Encodes payload with Base64, saves it on the target machine, also writes a VBS script using the echo command, and then lets the VBS script to decode the Base64 payload, and execute it. |
| certutil | Windows | Saves the payload in Base64 fake cert, decodes and executes it. |
| debug\_write | Windows | Uses debug.exe to write a small .Net binary, and that binary will take a hex-ascii file created by the echo command, decode the binary, and finally execute. |
| debug\_asm | Windows | Assembles a COM file, and then COM file will decode our hex-ascii payload, and execute it. |
| tftp | Windows | Uses tftp.exe to download and execute the payload. |
| bourne | UNIX | Similar to VBS, but has a richer set of commands to choose from. |
| echo | UNIX | Echo, chmod and execute |
| printf | UNIX | Printf, chmod and execute |
| wget, curl, fetch | Multi | Remote, command-based stagers |

Table 5.2 – List of options included in this mixin

| **Name** | **Value** | **Description** |
| --- | --- | --- |
| CMDSTAGER::FLAVOR | “auto” | Name of the flavour to use |
| CMDSTAGER::TEMP | None | Directory to store staged files |
| CMDSTAGER::SSL | “false” | Use SSL for supported stagers |

**6) File Cleanup (Msf::Exploit::FileDropper)**

***def register\_file\_for\_cleanup(\*file)***

Removes specific files on cleanup

| **Parameter** | **Description** |
| --- | --- |
| \*file | Files to delete after cleanup |

***def register\_dir\_for\_cleanup(\*dir)***

Removes specific directories on cleanup

| **Parameter** | **Description** |
| --- | --- |
| \*dir | Directories to delete after cleanup |

***def file\_dropper\_win\_path(path)***

Converts path to Windows path (separator replacement)

| **Parameter** | **Description** |
| --- | --- |
| path | Path to convert |

**7) Datastore Options**

***def <opt\_type>.new(name, [required, description, value])***

Adds a new option of give type

| **Parameter** | **Description** |
| --- | --- |
| name | Name of the option |
| required | Whether or not this option is mandatory |
| description | Option’s description |
| value | Option’s default value |

***def register\_[advanced\_]options([\*opt], self.class)***

Adds option to the module’s datastore

| **Parameter** | **Description** |
| --- | --- |
| \*opt | New option type |

***def deregister\_options(\*options)***

Remove options that are registered by default

| **Parameter** | **Description** |
| --- | --- |
| options | Option names to remove from datastore |

Table 7.1 – Option types

| **Name** | **Description** |
| --- | --- |
| OptString | String-type option |
| OptInt | Int-type option |
| OptBool | Boolean-type option |
| OptEnum | Choice-limited option. Choices are passed as a list. |
| OptPort | Port option (value ranges from 0 to 65535) |
| OptAddress | IPv4 address |
| OptAddressRange | Address/CIDR range/file with addresses/“rand:<n>” |
| OptPath | Path to a local file |
| OptRegexp | Regex expression |

**8) Check Method**

Table 8.1 – Check Method Logging Functions

| **Method** | **Description** |
| --- | --- |
| vprint\_line() | verbose version of print\_line |
| vprint\_status() | verbose version of print\_status that begins with "[\*]" |
| vprint\_error() | verbose version of print\_error that begins with "[x]" |
| vprint\_warning() | verbose version of print\_warning that begins with "[!]", in yellow |
| vprint\_debug() | verbose versino of print\_debug that begins with "[!]", in blue |

Table 8.2 – Check Codes

| **Checkcode** | **Description** |
| --- | --- |
| Exploit::CheckCode::Unknown | Used if the module fails to retrieve enough information from the target machine, such as due to a timeout. |
| Exploit::CheckCode::Safe | Used if the check fails to trigger the vulnerability, or even detect the service. |
| Exploit::CheckCode::Detected | The target is running the service in question, but the check fails to determine whether the target is vulnerable or not. |
| Exploit::CheckCode::Appears | This is used if the vulnerability is determined based on passive reconnaissance. For example: version, banner grabbing, or simply having the resource that's known to be vulnearble. |
| Exploit::CheckCode::Vulnerable | Only used if the check is able to actually take advantage of the bug, and obtain some sort of hard evidence. For example: for a command execution type bug, get a command output from the target system. For a directory traversal, read a file from the target, etc. Since this level of check is pretty aggressive in nature, you should not try to DoS the host as a way to prove the vulnerability. |
| Exploit::CheckCode::Unsupported | The exploit does not support the check method. If this is the case, then you don't really have to add the check method. |

**9) Powershell (Msf::Exploit::Powershell)**

***def compress\_script(script\_in, eof)***

Returns a GZIP compressed script with decompression stub

| **Parameter** | **Description** |
| --- | --- |
| script\_in | Contents of the script to compress |
| eof | Marker to indicate the end of file appended to script |

***def cmd\_psh\_payload(pay, payload\_arch)***

Returns a GZIP compressed script with decompression stub

| **Parameter** | **Description** |
| --- | --- |
| pay | Payload instance to wrap with Powershell, usually *payload.encoded* |
| payload\_arch | Payloads architecture, usually *payload\_instance.arch.first* |

Table 9.1 – List of options included in this mixin (POWERSHELL::<opt\_name>)

| **Name** | **Value** | **Description** |
| --- | --- | --- |
| PERSIST | false | Run the payload in a loop |
| PREPEND\_SLEEP | None | Prepend seconds of sleep |
| PREPEND\_PROTECTIONS\_BYPASS | false | Prepend AMSI/SBL bypass |
| STRIP\_COMMENTS | false | Strip comments |
| STRIP\_WHITESPACE | false | Strip whitespace |
| SUB\_VARS | false | Substitute variables names |
| SUB\_FUNCS | false | Substitute functions names |
| EXEC\_IN\_PLACE | false | Produce PSH without executable wrapper |
| REMOVE\_COMSPEC | false | Produce script calling powershell directly, not using the interpreter defined in %COMSPEC% |
| NONINTERACTIVE | true | Execute without interaction |
| ENCODE\_FINAL\_PAYLOAD | false | Encode final payload passed to *EncodedCommand* flag |
| ENCODE\_INNER\_PAYLOAD | false | Encode inner payload passed to *EncodedCommand* flag |
| WRAP\_DOUBLE\_QUOTES | false | Wraps *Command* flag argument in single quotes |
| NO\_EQUALS | false | Pad base64 until no "=" remains |
| METHOD | “reflection” | Payload delivery method |