前言

想了很多题目，感觉都不合适，比如，初探External C2、小白学External C2、通过端口复用让无法主动出网内网机器在CS上线、菜鸡玩Cobalt Strike等等。

标题不能起的太长，又不想当标题党，所以就在开头把题目和测试环境先讲清楚。目标将内网一台web服务器的80端口映射出来，但是此web服务器是不能出网的。目标不能出网，但你又想用CS（内网直接connect别的主机，多舒服）。

本文是通过实现这样的功能的一个小Demo来介绍下external C2这个功能。

Demo代码地址：https://github.com/hl0rey/Web_ExternalC2_Demo

externalC2介绍

因为我这篇文章是面向新手的（尤其是external C2介绍部分，大佬可以略过），希望不熟悉externalC2的朋友能够在看完这篇文章之后，也能自己尝试着手自定义C2通信来适应不同的场景。

什么是externalC2

就如这个单词的字面意思一样：额外的C2。externalC2是cobalt strike开放给用户的接口。利用这个特性，用户让自己编写的程序在cobalt strike的teamserver和beacon之间处理C2通信的过程，也就是充当“翻译”的角色。

externalC2的基本架构

完成一次externalC2通信逻辑上需要五个部分参与：

- 团队服务器（Teamserver）

  cobalt strike的核心部分，一切功能的基础。

- Exterlnal C2 服务器（External C2 server）

  依托于Teamserver启动的External C2服务，将开启一个端口，接收传入的请求。

- 第三方控制器（Third-party Controller）

自己编写的与External C2 server 直接交互的程序，一般用来把第三方客户端传回的信息，翻译成External C2 server能懂的格式。

- 第三方客户端（Third-party Client）

  用来获取被控端的基本信息和执行CS下发的payload的程序。

- Beacon（SMB Beacon）

  CS功能的基础，一切的由CS下发的命令，最终都交由它来执行。

由于官方的架构图是英文的，看起来没有亲切感，所以我再画一遍。

从图中可以看出，使用External C2对我们最基本的要求是

- 编写一个第三方控制器，该控制器可以通过TCP连接向EternalC2服务器发送数据和从ExternalC2服务器接收数据，并且能与第三方客户端进行通信。
- 编写一个第三方客户端，可以启动Beacon，并且能通过命名管道与Beacon进行交互，还要能与第三方控制器进行通信。

ExternalC2的通信协议

看代码是最能熟悉它的通信协议的方式，这里只介绍几个关键的点。

- 无认证的帧格式。
- 先是一个长度为四字节的小端字节序的表示长度的值（高级语言可能会使用大端字节序序列化数字为流，所以得保证这个数字得按照小端字节序序列化）。
- 根据长度读取真正的数据部分，也就是说要完成一次对结果的读取，要先读长度，在读内容。
- 进行写入操作时也是需要符合该格式。
- 与external C2服务器的通信，以及与命名管道的通信都遵循这个格式。

External C2组件

External C2 服务器

从客户端加载contana脚本即可。

```
externalc2_start("0.0.0.0",2222);
```

第三方控制器

想让被控端上线的时候，就先向external C2服务器请求建立一个会话，发送被控端的相关配置，服务端会返还需要执行的Payload。每一个链接对应一个会话。

具体细节是：

首先设置当前会话，就是向External C2 服务器发送被控端的相关信息，发送一个或多个数据包中包含key=value格式的键值对。具体格式如下：

选项	默认值	描述
arch	x86	payload的位数，可接受的值：x86、x64
pipename		命名管道的名字
block	100	以毫秒为单位的时间，没有操作的时候，external C2应该阻塞多长时间，应该相当于sleep

设置发送完毕之后，向external C2 服务器发送一个字符串 go，然后等external C2 服务器返回payload，第三方控制器需要把payload中继到第三方客户端，并且有第三方客户端执行它。当第三方控制器从external C2 服务器断开连接时，teamserver就会把当前连接对应的会话标记为失效会话，目前没有办法恢复已死的会话。（经过我对其协议的学习，感觉官方文档这句还是有有点绝对的，应该是存在断线重连的方法的。）

第三方客户端

第三方客户端负责执行从第三方控制端接收到的payload。该payload是一个头部经过修改的可以自我引导、反射加载的DLL（关于这部分内容可以参考参考链接中的pe_to_shellcode）。用通常的进程注入技术就可以执行它。当payload处于运行状态时，第三方客户端即可以通过对前一阶段会话配置时设置的命名管道名称（

```
\\.\pipe\[pipe name]
```

）的读写操作来与Beacon交互。

会话生命周期

Teamserver	External C2服务器	第三方控制器	第三方客户端	SMB Beacon
			向控制器请求建立一个新的会话	
		连接到External C2服务器		
		向External C2发送会话配置		
		向External C2请求 stage（payload）		
	把stage发给第三方控制器			
		中继stage给第三方客户端		
			把stage注入某进程	
				启动命名管道服务器
			连接到命名管道服务器	
				向第三方客户端发送metadata

Teamserver	External C2服务器	第三方控制器	第三方客户端	SMB Beacon
			中继metadata给第三方控制器	
		中继metadata给external C2服务器		
	处理metadata			
Coabalt strike中出现了一个新的会话				
	用户下发任务或者不下发			
	向第三方控制器发送任务			
		中继任务		
			中继任务	
				处理任务
				向第三方客户端发送任务结果
			中继任务结果	
		中继任务结果		
	处理任务结果			
显示结果				
当会话存活的时候，重复从出现会话到显示结果的过程				
				会话退出
			读写命名管道出错	

Teamserver	External C2服务器	第三方控制器	第三方客户端	SMB Beacon
			通知第三方控制器退出会话	
		与External C2 服务器断开连接	退出	

推荐看下官方的示例代码，我顺便一起放到了Demo代码中。

实现目的的基本思路

实现目的主要有以下需要考虑的点：

- 问题：当payload成功执行之后，需要由第三方客户端完全接管与Beacon的交互。

  解决方案：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。

- 问题：因为目标不能出网，所以就无法向第三方客户端主动请求建立会话，也就是说按照官方生命周期的描述是无法建立会话的。

  解决方案：其实这一步不是必要，可以省略，或者将其变为第三方控制器先向External C2服务器请求payload，然后将payload直接发送给第三方客户端，让它执行即可。

- 问题：Beacon返回执行结果时，无法主动向外发送数据。

  解决方案：可以在第三方控制器对第三方控制器进行轮询解决该问题。会话退出的消息同理。

这个需求的难点在于把所有第三方控制器、Beacon主动对外的访问都转化为第三方控制器对第三方客户端的请求，变主动为被动。

具体实现

主要有3个模块参与：

第三方控制器：保存payload和轮询第三方客户端；

第三方客户端A：payload执行与管道中继；

第三方客户端B：第三方控制器请求中继）。

官方的会话生命周期描述不符合当前需求，符合当前的需求的会话生命周期如下：

- 在webshell里确定目标服务器的类型，第三方控制器向External C2服务器请求payload，并将返回的paylaod保存起来。
- 根据保存起来的payload生成可加载payload的可执行文件，也就是为了用第三方客户端执行payload而做准备。
- 通过webshell把第三方客户端A和payload传送到目标机器上，并且执行。
- 将第三方客户端B（符合目标服务器环境的web服务端脚本）传送到目标web路径下，后续的与第三方客户端A的交互，由它完成。
- 告知第三方控制器上一步上传的服务端脚本（第三方客户端B）的路径。
- 第三方控制器命令第三方客户端B从第三方客户端A获取metadata，第三方客户端A获取metadata并且回传。
- 第三方控制器将获取到的metadata发送给External C2服务器，此时Cobalt strike中已经出现上线的机器。

- 第三方控制器将任务下发给第三方客户端，并开始轮询是否有结果返回（时间间隔可以长一些，频繁的访问不是好事）。
- 第三方控制器轮询第三方客户端时，发现其返回来命名管道读写失败的信息，第三方控制器断开与External C2服务器的连接。

第三方控制器python编写，因为涉及到HTTP请求，可以直接使用requests库。第三方客户端分为两个部分，分别用C和PHP编写，C的部分负责保持与Beacon命名管道的持久连接（为了不让beacon认为自己掉线了），并且创建管道供PHP部分读写，PHP部分负责将数据中继出来，返回给第三方控制器。

功能测试

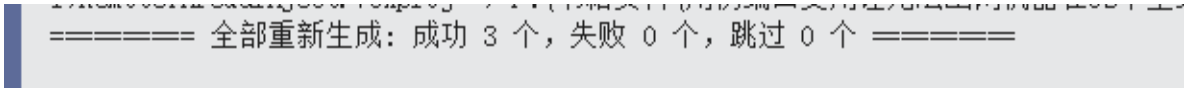
环境准备：

一台win7，普通客户机（虚拟机）

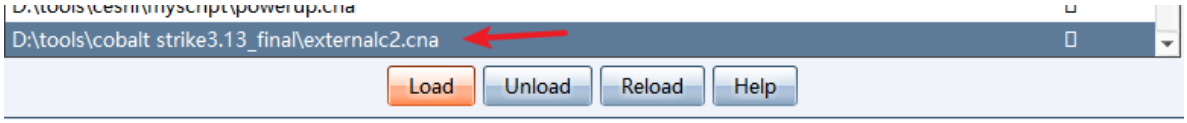
一台win10，web服务器，有php运行环境（我的物理机）

一台kali，攻击机，cobalt strike3.13运行External C2服务（虚拟机）

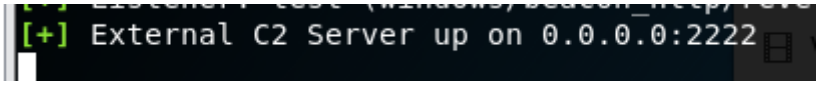
前期先把C程序编译好，我使用VS2019编写并编译可以正常使用。



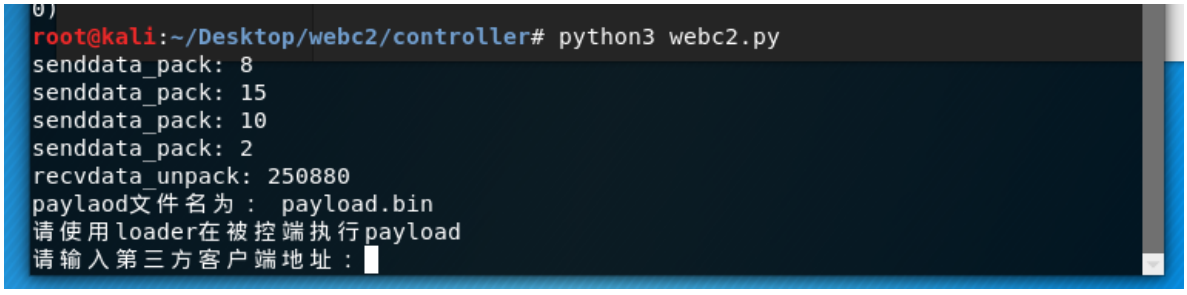
启动TeamServer，使用客户端连接TeamServer，并且加载externalc2.cna。



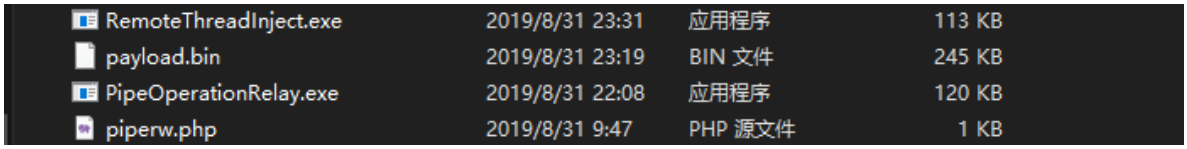
服务端出现这个就成功了。



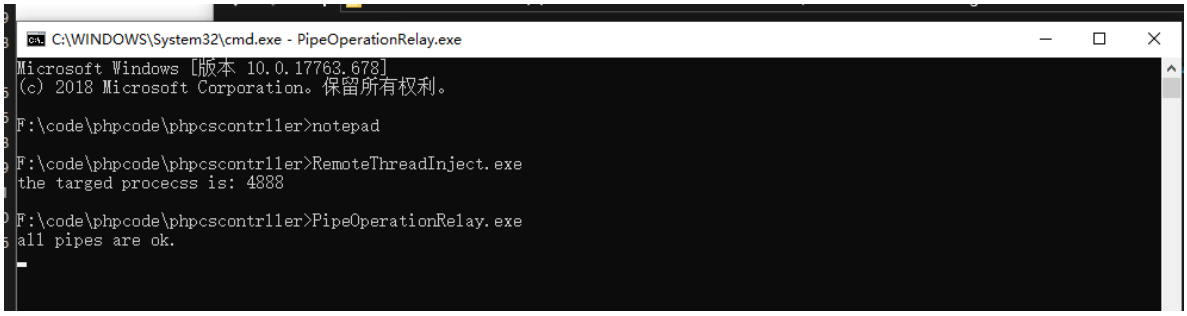
执行第三方控制器，事先需在第三方控制器脚本中配置好External C2服务器的地址。脚本将payload保存在脚本当前路径下的payload目录中，名为payload.bin。



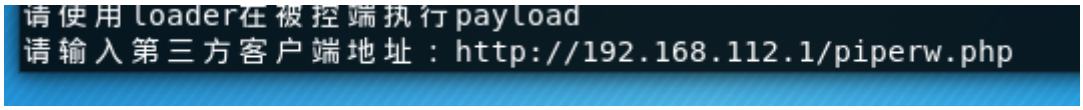
接下来，将第三方客户端以及payload上传至目标服务器，也就是win10上。



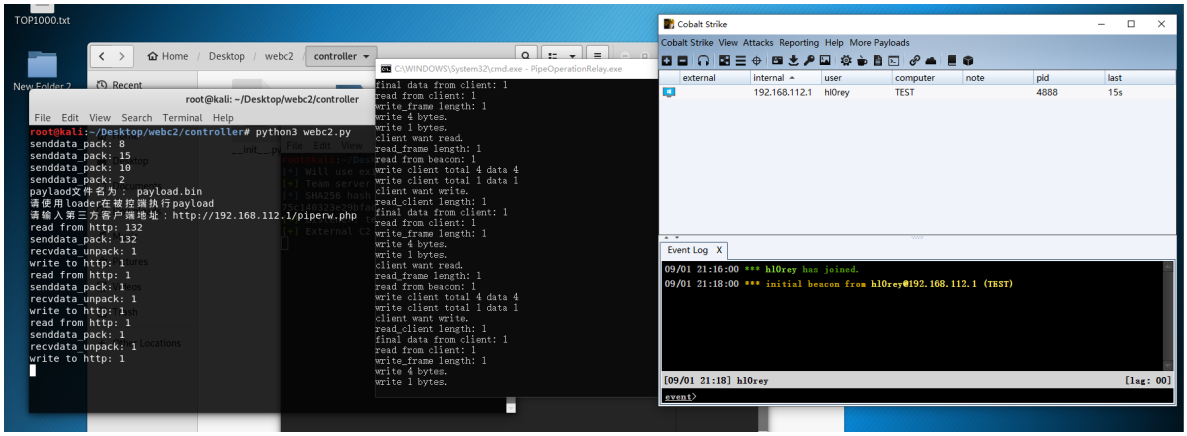
先启动一个notepad，因为默认情况下会把payload注入进notepad进程。然后先后运行RemoteThreadInject.exe和PipeOperationRelay.exe。当看到all pipe are ok。就说明程序运行所需要的所有管道都建立好了。



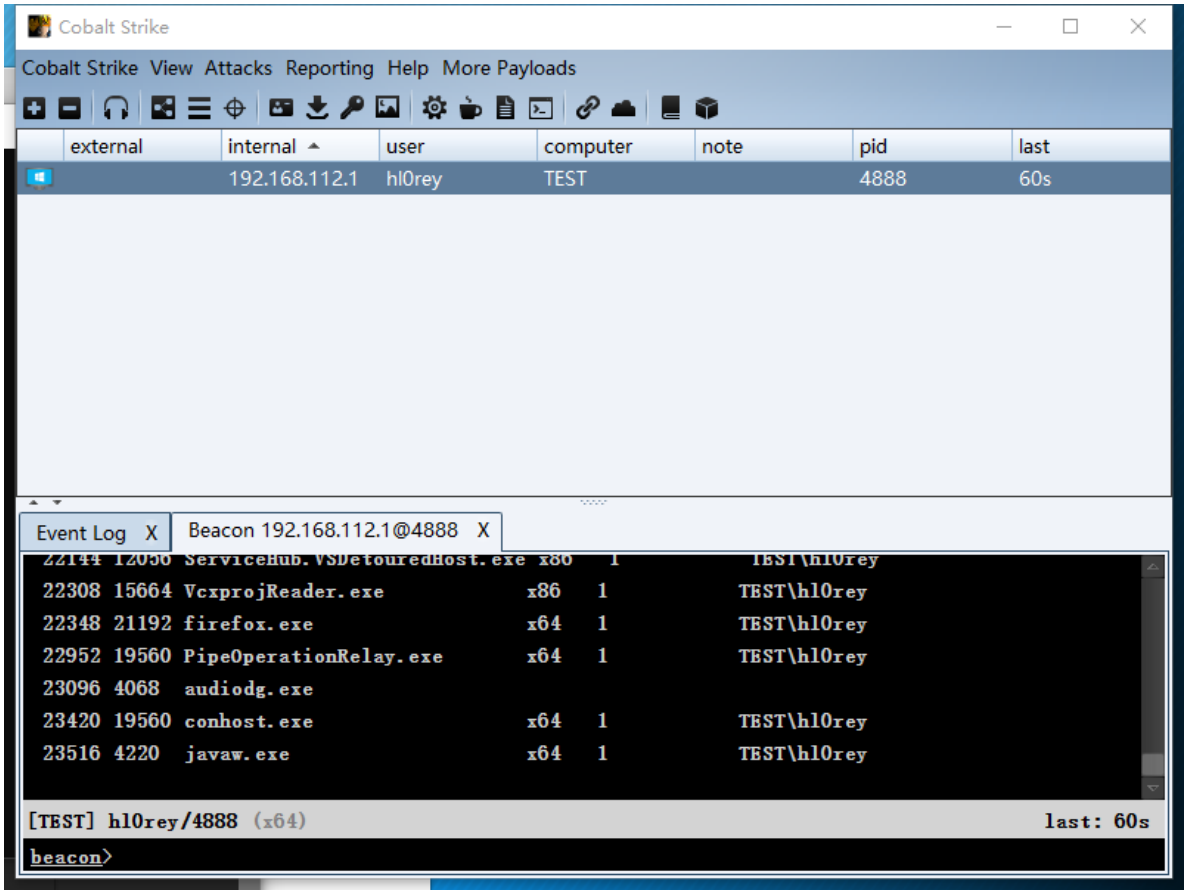
然后去第三方控制器里配置上传的piperw.php的url。



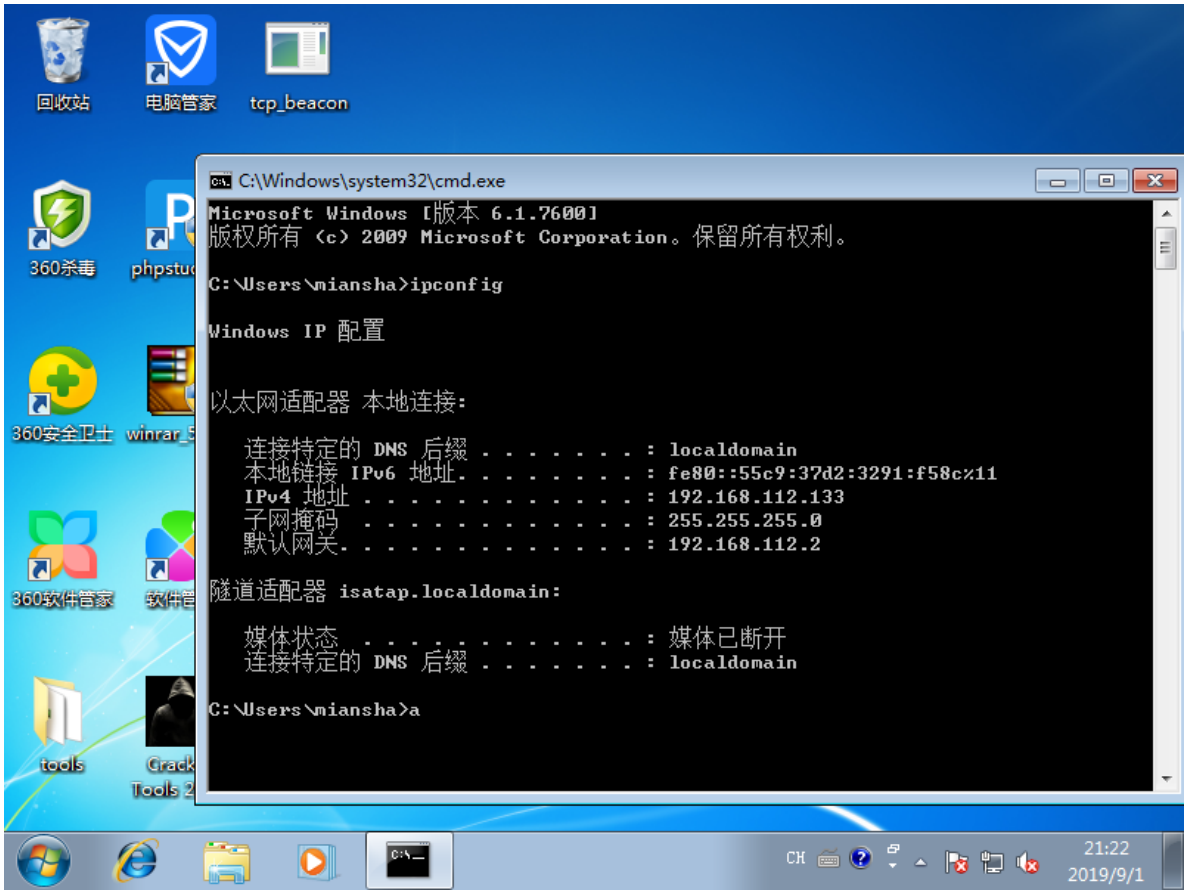
回车之后，上线成功。



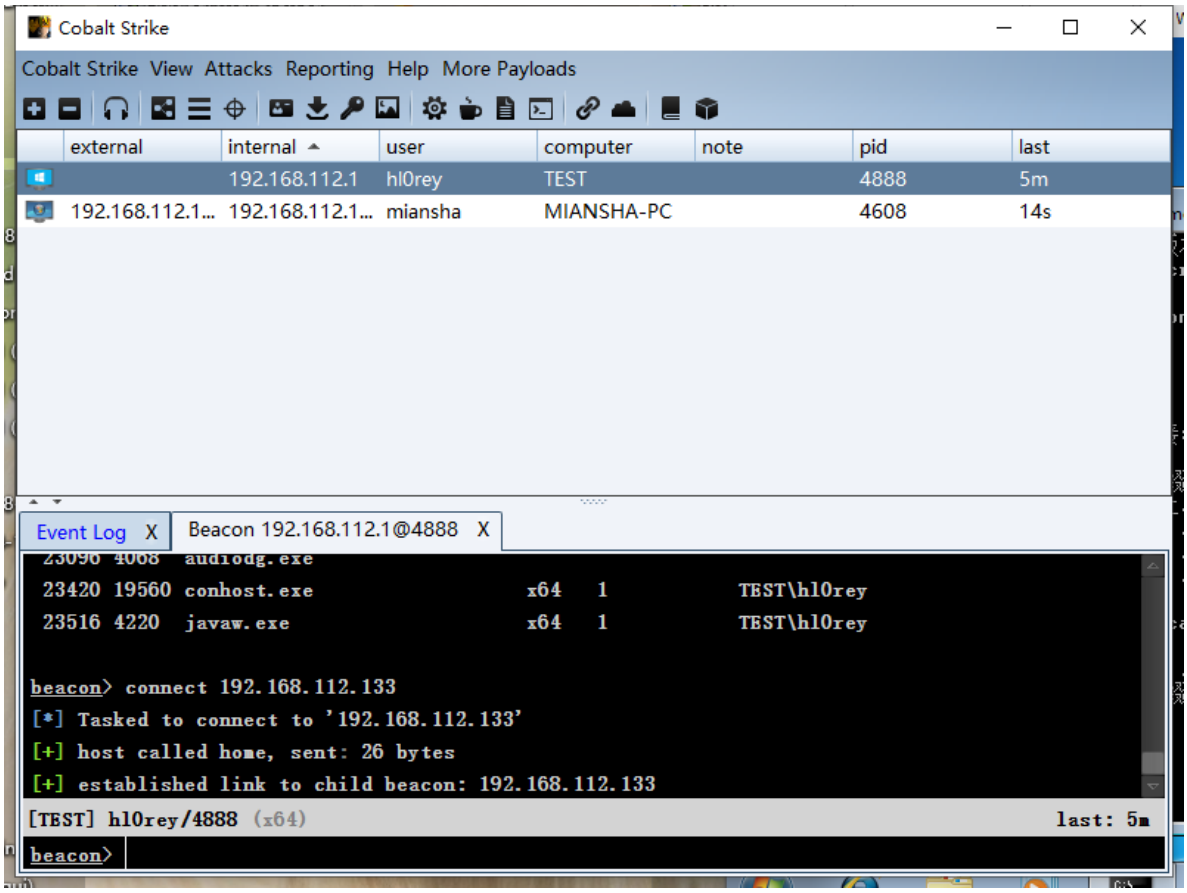
测试下功能是否正常，查看进程。



测试TCP Beacon，在另一台主机上执行TCP Beacon。

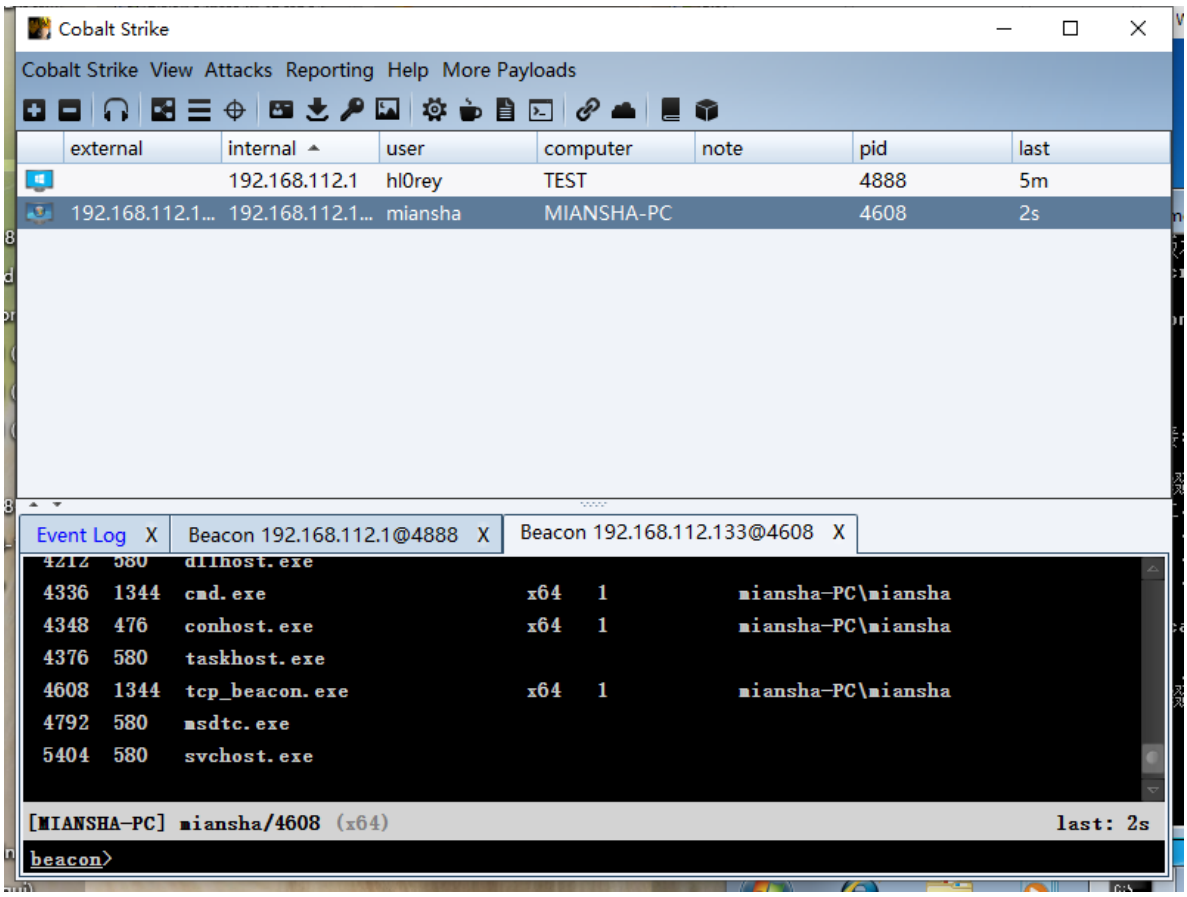


可以正常进行链接。

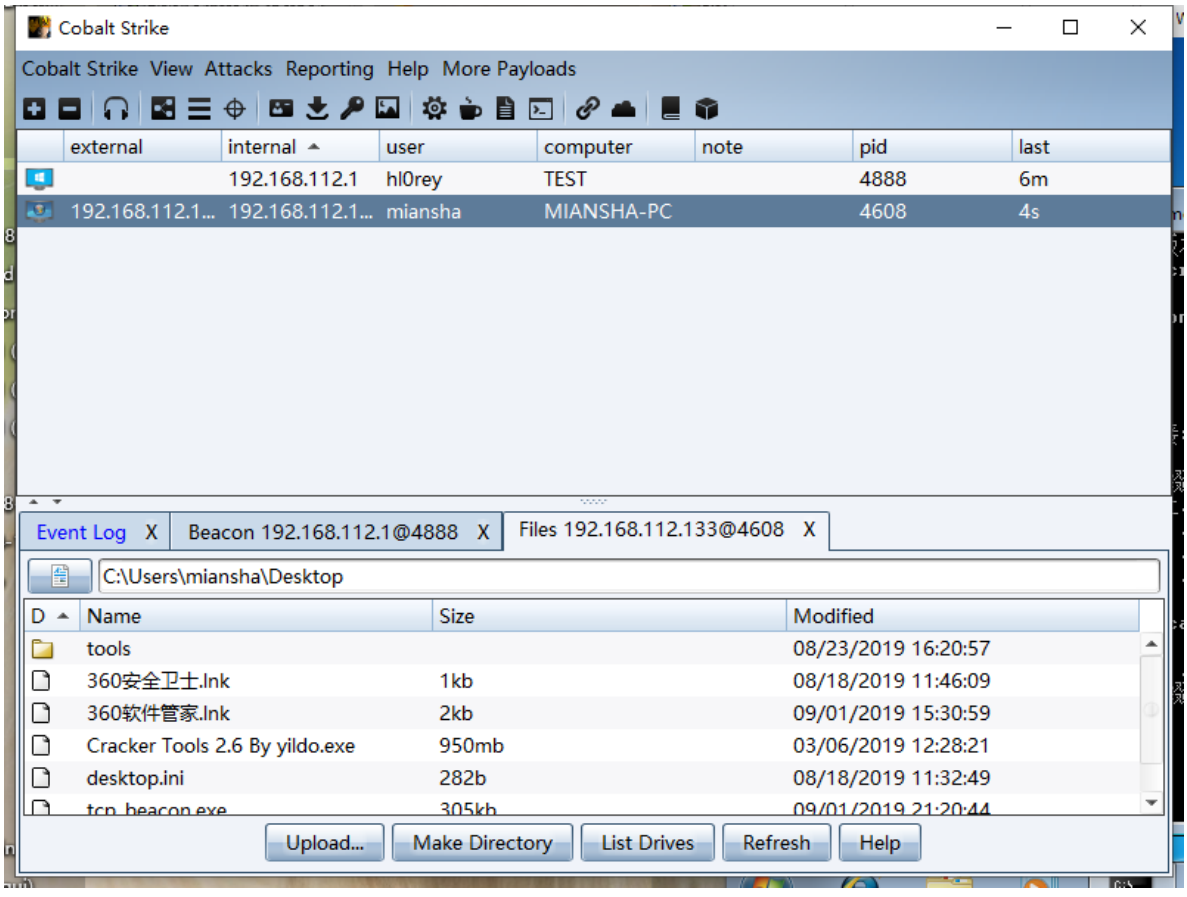


测试TCP Beacon的功能。

查看进程：



查看文件：



后话

- 每次断开与命名管道的连接，再次打开时，必须重新发送上线包，在这上边栽了三天，才发现这个问题。最终通过写一个管道访问中继程序解决了这个问题，断开命名管道的连接，beacon就认为自己掉线了。

- php的recourse指针无法放到session中。但这个不能叫坑，正常可以理解的特性。
- 有些东西做出来之后，才知道有没有用。External C2表现并不好，在一些操作上表现出了极差的稳定性，看到官网的文档是2016年，也许作者并没有把重心放在这上边，因为不是开源的，别人也没法帮他，只能等着了。
- 因为是测试代码，所以保留了很多调试输出。
- 我用win10做web服务器是因为想下周干点别的，不想解决兼容性问题了。

参考连接

https://github.com/hasherezade/pe_to_shellcode

https://docs.microsoft.com/zh-cn/windows/win32/ipc/pipes

https://blog.xpnsec.com/exploring-cobalt-strikes-externalc2-framework/

https://github.com/Und3rf10w/external_c2_framework

https://www.cobaltstrike.com/help-externalc2