0x01 赛题要求

本题中赛事方提供了某一失陷主机在某段时间内的通信流量包，要求选手通过流量审计、追踪溯源等方式找出控制该失陷主机的C2服务器

恶意流量分析(阶段一)

设计说明

恶意软件通常会对其通讯进行加密传输甚至模拟一些正常软件的特征从而避免直接被检测，而正常软件在使用过程中也同样会产生大量的加密流量，二者混在一起便会导致检测准确率降低，使得真正的恶意流量无法有效被检测

本题文件中提供了一段时间的失陷主机的通讯流量，参赛选手需审计该流量包，进行 **追踪溯源**，找到真正的恶意服务器，最终获取key，此外本题还设有可以获得额外分数的彩蛋题目

数据说明

文件：stage1

密码：datacon2021stage1

md5：5858bfd9f63db6d33facef6ef6f99bba

提交形式和文件格式

直接提交获取到的key和彩蛋key。

key为16位随机字符串，该字符串前有显著提示，告知选手该字符串为key

彩蛋为32位随机字符串，该字符串前有显著提示，告知选手该字符串为彩蛋。

提交规则

选手在本题的持续时间内，提交次数不限。

本阶段包含一个彩蛋，彩蛋提供额外分数

0x02 题目背景及思路

根据给出的hint，得知与失陷主机通信的是**传说中的红队利器Cobalt Strike**的team server. CS的特点是Malleable C2，即“可定制”的C2流量，可以通过配置文件，来修改CS beacon和C2的流量和行为特征，使C2流量混合在目标环境流量中，伪装为正常应用流量，达到欺骗的作用。

根据题目描述，溯源到C2服务器的目的是拿到flag；另一方面，hint中提及“仅靠被动审计是不行的”。所以我们初步认为需要通过主动探测的方式扫描流量中出现的潜在C2主机，而不是针对流量本身进行识别分类。

根据这些关键词，结合搜索到的资料、文章，我们了解到，team server上的Beacon Listener (Beacon是指CS运行在目标主机上的payload)**存在被主动探测手段发现的可能**，因为Beacon Listener所在的Staging Server会提供Payload Staging功能，分阶段进行payload投递，使得任何人都可以通过正确的stage的Url下载stage。只要存储着Beacon配置和payload的stage服务器暴露在公网上，就可以就可以通过主动扫描发现它。

0x03 具体流程

1.主动扫描

将pcap文件中所有tls会话的handshake阶段的server name提取出来(数字是出现次数)：

```
1440 code.jquery.com
372 www.baidu.com
360 jquery.com
186 www.w3school.com.cn
180 www.runoob.com
...
1 12www-mirror-co-uk.amp.cloudflare.com
1 11-www-mirror-co-uk.amp.cloudflare.com
1 0-gravatar-com.amp.cloudflare.com
```

首先常规思路筛选出Alexa top1M以外的域名，**企图缩小范围。**

```
186 www.w3school.com.cn
10 mat1.gtimg.com
8 wordpress.com
...
1 bizup.cc.danuoyi.tbcache.com
1 ag.innovid.com
1 adm.leju.com
```

根据stage url的生成规则，扫描时需要**额外拼接上url的checksum8校验码**，否则会返回404。根据大佬们公开的逆向出的stage url生成规则，可以得到校验码的生成方法，并最终生成32位或64位payload的4位校验码。

最终发现并没有什么用，无法下载到任何文件。这说明**攻击者确实使用了CS的Malleable C2功能进行了域名伪造，不能简单粗暴地进行Alexa筛选。**

对所有域名进行扫描，发现如下几个域名可以得到stage文件：

```
27 d28ef1bm7Oqsi.cloudfront.net
13 d1yr5tm734gi1r.cloudfront.net
10 dku6bh98adktv.cloudfront.net
9 d2lj8kjjwt8rn6.cloudfront.net
8 d3noow75xz96w4.cloudfront.net
8 d2og948cy5uxtu.cloudfront.net
6 d32jqjeqo1vb2n.cloudfront.net
```

2.解析stage配置文件

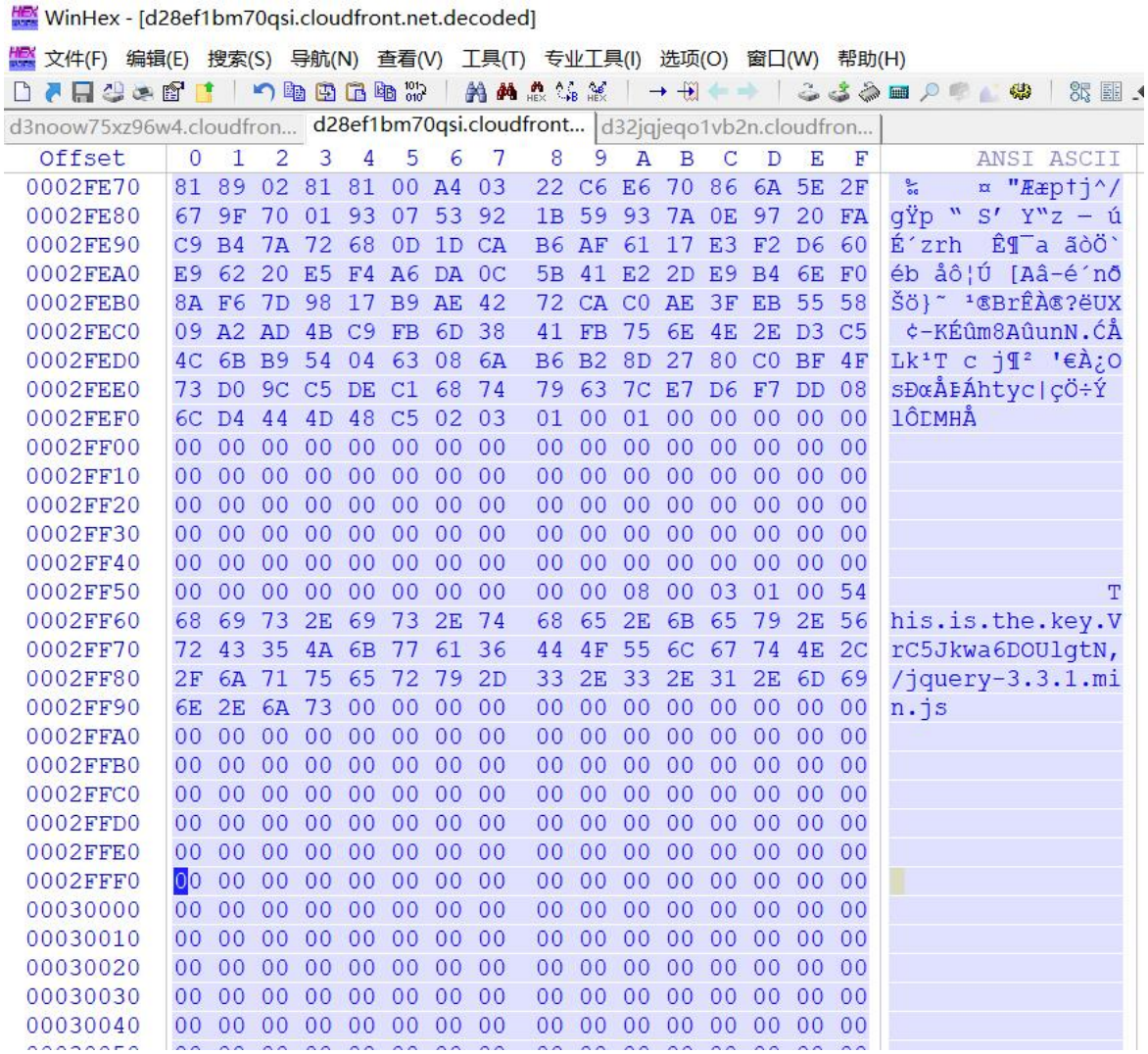
获取配置文件后，使用如下脚本进行解密：

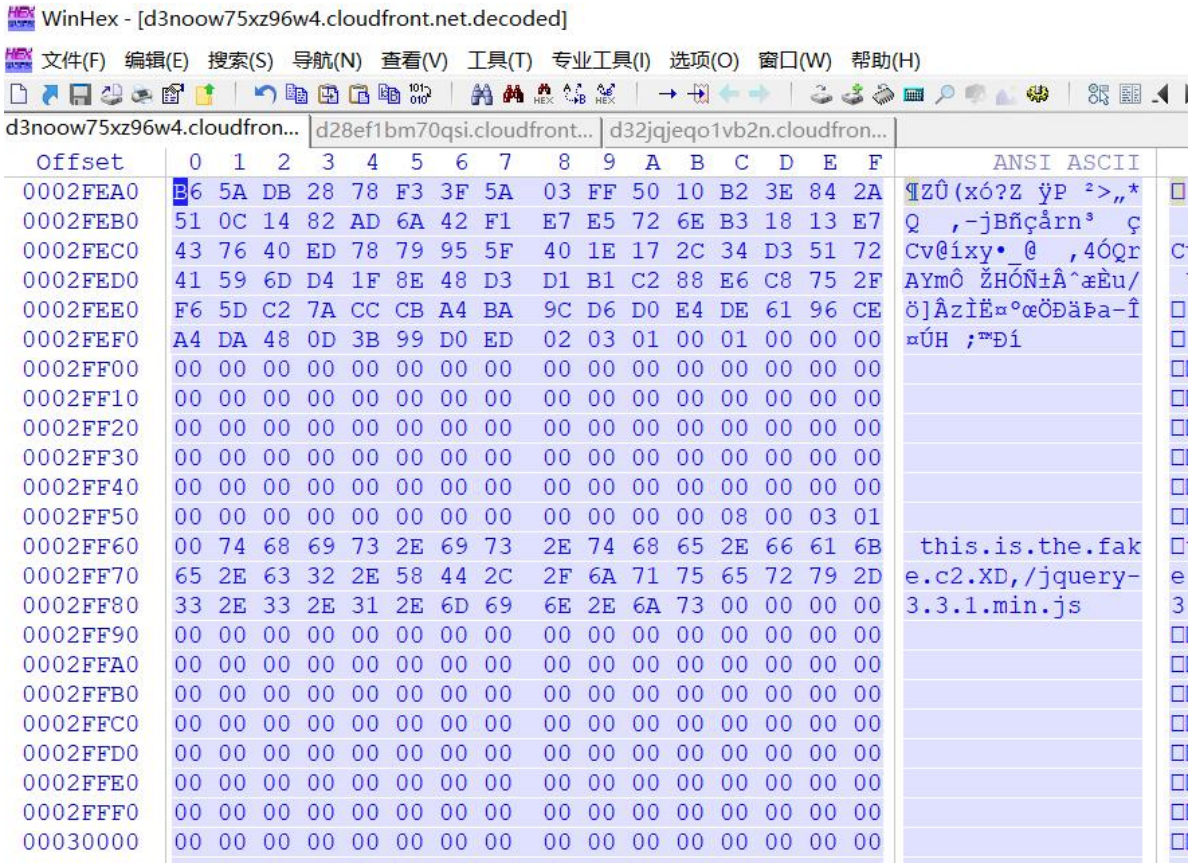
```
import sys
import struct

filename = sys.argv[1]
data = open(filename, 'rb').read()
t = bytearray(data[0x45:])
(a,b) = struct.unpack_from('<II', t)
key = a
```

```
t2 = t[8:]
out = ""
with open(filename+'_decoded', 'wb') as f:
    for i in range(len(t2)//4):
        temp = struct.unpack_from('<I', t2[i*4:])[0]
        temp ^= key
        out = struct.pack('<I', temp)
        print(out)
        key ^= temp
        f.write(out)
```

但解密后打开还是乱码，这是因为**CS对配置信息还进行了异或加密。**根据多篇文章中给出的解析，CS 3.x/4.x的异或密钥分别是0x69、0x2E，经过我们尝试之后发现是0x2E。

最终只有通过d28ef1bm70qsi.cloudfront.net下载的配置文件中含有正确的key，而d32jqjeqo1vb2n.cloudfront.net的配置文件则明显不是使用这两个密钥中的任何一个加密的，因为没有找到任何可读明文。其他域名得到的文件则是用于干扰选手的fake文件：

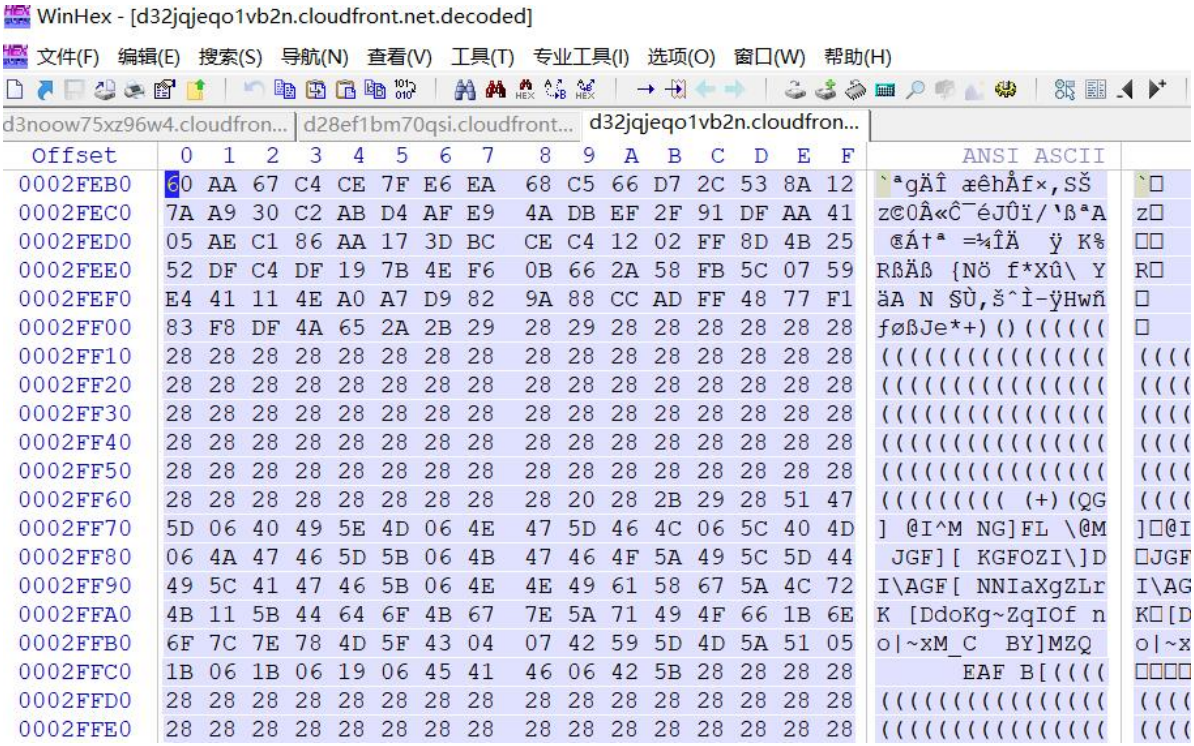


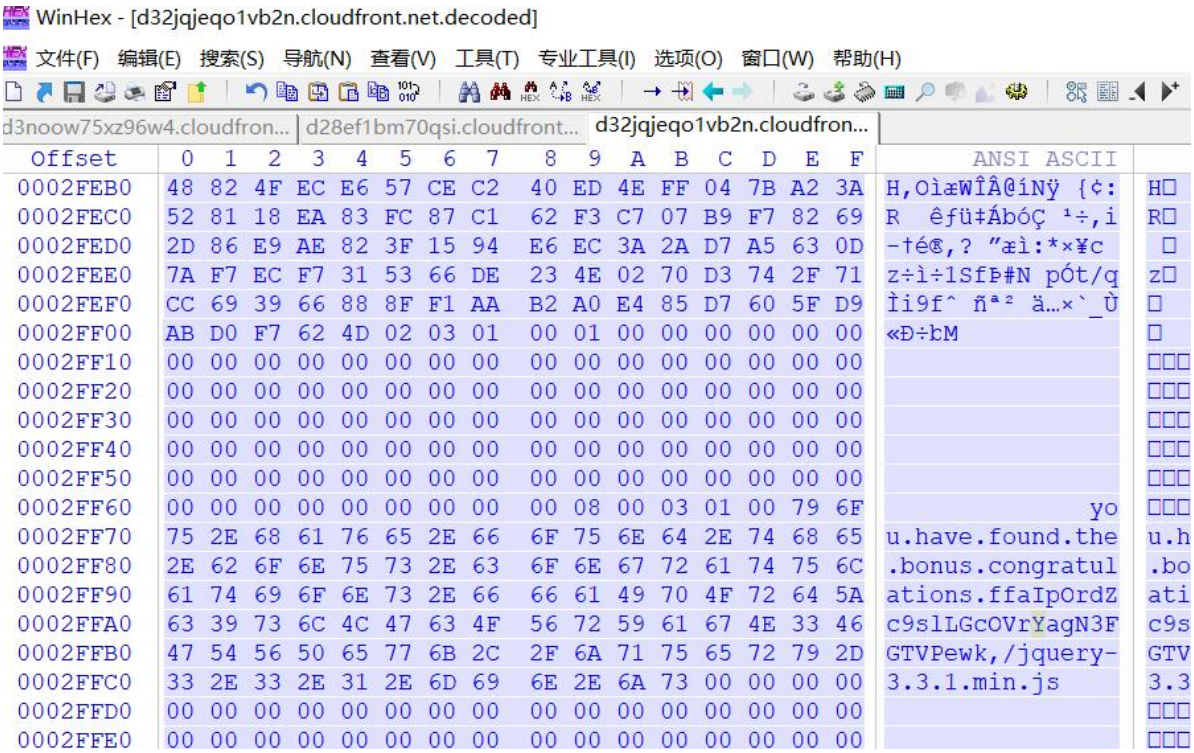


0x04彩蛋

关于彩蛋，唯一的线索就在未知异或密钥的那个文件中，可以观察到，不论是fake文件还是有key的文件，解密后的明文位置都是0x2FF60，说明出题人还是手下留情了。如果参照相关的博客，在最后一步利用脚本从配置文件解析明文信息，而不是用winhex进行异或运算，就无法看出这一点。

可以发现，所有文件在解密为明文后，0x2FF60位置周围都是用于填充的00，因此异或加密之后，相应的位置显示的就是异或密钥本身(00 xor A ==A). 而这一系列配置文件的格式大概率都是相同的，所以在d32jqjeqo1vb2n.cloudfront.net的配置文件中，相应位置的明文也必然是用于填充的00，观察decode后的文件，可以得出异或密钥为0x28的结论：





References

https://www.sohu.com/a/435908844_750628

https://www.freebuf.com/articles/network/273480.html

https://cloud.tencent.com/developer/article/1764340

https://www.52pojie.cn/thread-1396671-1-1.html

https://xz.aliyun.com/t/2796