Datacon 2021 网络流量分析Writeup

作者姓名：贾锟，王志，王君楠，汪旭童，尹捷

指导教师：刘潮歌

培养单位：中国科学院信息工程研究所

2021 年 10 月

目 录

第1章　恶意流量分析

1.1　比赛题目

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。

1.2　概述

以主机端的网络流量为基础，通过流量审计和主动探测的方式，对攻击者进行追踪溯源和行为分析。

1.3　解题思路

本题的整体思路是，先从流量审计方向入手寻找恶意信息，然后再对恶意信息进行主动探测，每个环节均详细记录了团队的解题方案和历程。

1.3.1　流量审计

根据对加密流量相关的理论的了解和实践基础的积累，本团队尝试从加密流量的典型特征进行异常分析。

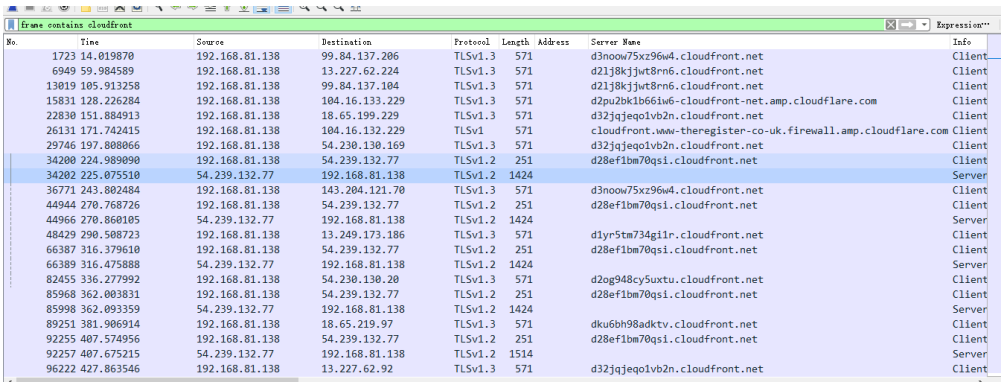
从以下几个方面定位攻击者/攻击组织的潜在攻击特征，如下：

- 证书安全性；
- 心跳报活机制；
- 数据流统计特征异常；
- 主机统计特征异常。

由于该题的数据源绝大多数均为加密流量，因此难以从payload的内容上做解析分析，但是加密流量的握手阶段会有一些比较有价值的信息，如server_name，和证书中的信任路径，有效期等。

本团队首先尝试根据证书中使用者的域名信息，过滤了alexa 100000的流量，找到了16个域名，但是对这些域名关联的流量做审计后并没有发现典型的攻击特征。于是，团队猜想，攻击者是否利用了CDN或域前置技术隐藏了C2信息，

因而攻击流量潜藏在了看似合法的流量之中。于是，团队调取了tls握手阶段访问的server_name字段，发现有cloudflare, cloudfront, alicdn等常用的CDN域名。在对cloudfront相关的流量进行分析时，我们发现了明显的心跳机制，见图1.1。


frame contains cloudfront

No.	Time	Source	Destination	Protocol	Length	Address	Server Name	Info
1723	14.019870	192.168.81.138	99.84.137.206	TLSv1.3	571		d3noow75xz96w4.cloudfront.net	Client
6949	59.984589	192.168.81.138	13.227.62.224	TLSv1.3	571		d21j8kjjwt8rn6.cloudfront.net	Client
13019	105.913258	192.168.81.138	99.84.137.104	TLSv1.3	571		d21j8kjjwt8rn6.cloudfront.net	Client
15831	128.226284	192.168.81.138	104.16.133.229	TLSv1.3	571		d2pu2bk1b66iw6-cloudfront-net.amp.cloudflare.com	Client
22830	151.884913	192.168.81.138	18.65.199.229	TLSv1.3	571		d32jqjqeqo1vb2n.cloudfront.net	Client
26131	171.742415	192.168.81.138	104.16.132.229	TLSv1	571		cloudfront.www-theregister-co-uk.firewall.amp.cloudflare.com	Client
29746	197.808066	192.168.81.138	54.230.130.169	TLSv1.3	571		d32jqjqeqo1vb2n.cloudfront.net	Client
34200	224.989090	192.168.81.138	54.239.132.77	TLSv1.2	251		d28ef1bm70qsi.cloudfront.net	Client
34202	225.075510	54.239.132.77	192.168.81.138	TLSv1.2	1424			Server
36771	243.802484	192.168.81.138	143.204.121.70	TLSv1.3	571		d3noow75xz96w4.cloudfront.net	Client
44944	270.768726	192.168.81.138	54.239.132.77	TLSv1.2	251		d28ef1bm70qsi.cloudfront.net	Client
44966	270.860105	54.239.132.77	192.168.81.138	TLSv1.2	1424			Server
48429	290.508723	192.168.81.138	13.249.173.186	TLSv1.3	571		d1yr5tm734gi1r.cloudfront.net	Client
66387	316.379610	192.168.81.138	54.239.132.77	TLSv1.2	251		d28ef1bm70qsi.cloudfront.net	Client
66389	316.475888	54.239.132.77	192.168.81.138	TLSv1.2	1424			Server
82455	336.277992	192.168.81.138	54.230.130.20	TLSv1.3	571		d2og948cy5uxtu.cloudfront.net	Client
85968	362.003831	192.168.81.138	54.239.132.77	TLSv1.2	251		d28ef1bm70qsi.cloudfront.net	Client
85998	362.093359	54.239.132.77	192.168.81.138	TLSv1.2	1424			Server
89251	381.906914	192.168.81.138	18.65.219.97	TLSv1.3	571		dku6bh98adktv.cloudfront.net	Client
92255	407.574956	192.168.81.138	54.239.132.77	TLSv1.2	251		d28ef1bm70qsi.cloudfront.net	Client
92257	407.675215	54.239.132.77	192.168.81.138	TLSv1.2	1514			Server
96222	427.863546	192.168.81.138	13.227.62.92	TLSv1.3	571		d32jqjqeqo1vb2n.cloudfront.net	Client

图 1.1 cloudfront

尤其是对域名d28ef1bm70qsi.cloudfront.net的访问明显以45s为周期。团队尝试对这些域名进行ping探测，发现其中绝大多数已经失效，仅有d28ef1bm70qsi以及d32jqjqeqo1vb2n可以连通。

1.3.2 主动探测

分析至此一度陷入瓶颈，通过端口扫描，漏洞探测等方式都没有发现这两个域名的可利用性，团队甚至一度认为可能存在尚未被定位的其他恶意域名。就在我们一筹莫展时，出题人适时地给出了hints。（1）攻击者喜欢玩CS。（2）仅凭被动流量审计不足以找到key；（3）Beacon Stager Server。由于团队成员主要涉猎流量分析和异常检测，对Cobalt Strike工具的了解程度并不算深，因此我们查阅了非常多的资料。其中，[1]以及[2]给了我们很大的启发。资料显示，Cobalt Strike采用分段投递的方式的话，会生成一个stager和stage，并把stage托管在一个web server上，最初生成的stager需要去该web server下载并加载该stage。想要成功访问该web server需要在域名后添加特殊的uri。资料显示，针对32位和64位有不同的uri。 uri会在服务器端经过一个校验计算，匹配的话就能成功下载stage。一个可用的uri是KeVV。CDN可以代理Team Server与stager的通信，从而隐藏C2。图1.2展示整体的逻辑。

我们在d28ef1bm70qsi.cloudfront.net和d32jqjqeqo1vb2n.cloudfront.net均成功地下载了可疑的stage。资料显示，已经有相应的脚本[3]可以对stage进行解析，并

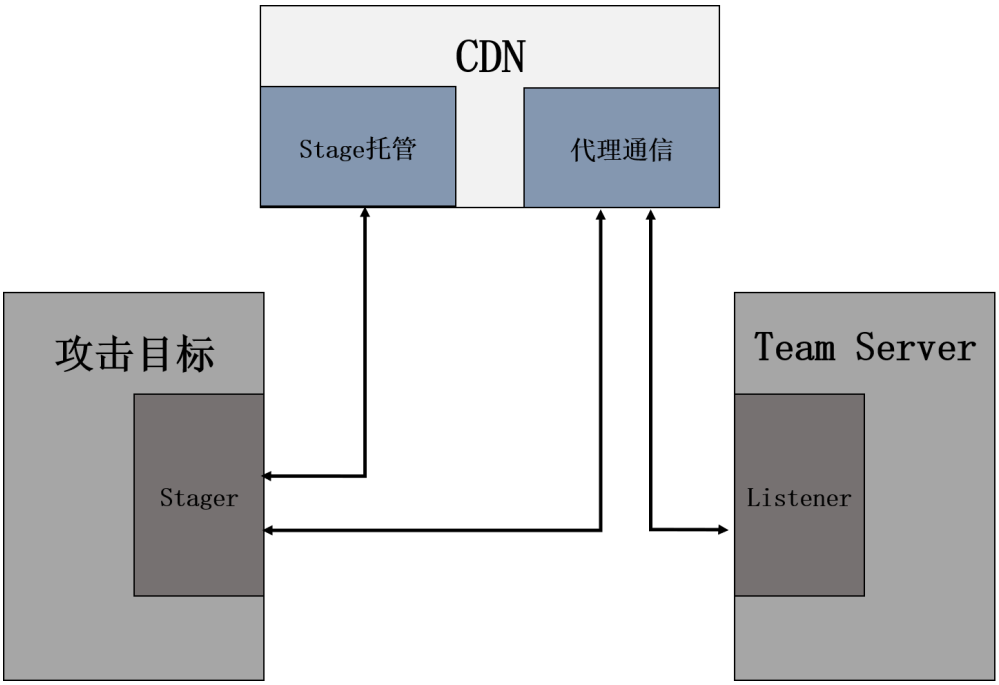


图 1.2 Cobalt Strike分段投递与CDN代理

能从中获取到Cobalt Strike的配置信息。在d28ef1bm70qsi.cloudfront.net上下载的stage中，我们成功解析出了第一题的Key。

```
MaxGetSize                      - 1403644
Jitter                          - 0
MaxDNS                          - 255
PublicKey_MD5                   - c9a494c7ef2d4e2f2aab514ba86493f8
C2Server                        - This.is.the.key.VrC5jkwa6DOUlgtN,/jquery-3.3.1.min.js
UserAgent                       - Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
HttpPostUri                     - /jquery-3.3.2.min.js
Malleable_C2_Instructions       - Remove 1522 bytes from the end
                                  Remove 84 bytes from the beginning
                                  Remove 3931 bytes from the beginning
                                  Base64 URL-safe decode
                                  XOR mask w/ random key
HttpGet_Metadata                - ConstHeaders
                                      Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
                                      Referer: http://code.jquery.com/
                                      Accept-Encoding: gzip, deflate
                                  Metadata
                                      base64url
                                      prepend "__cfduid="
                                      header "Cookie"
HttpPost_Metadata               - ConstHeaders
                                      Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
                                      Referer: http://code.jquery.com/
                                      Accept-Encoding: gzip, deflate
                                  SessionId
```

图 1.3 stage解析结果

但是d32jqjeqo1vb2n.cloudfront.net上下载的stage显示解析出错。由于沉浸在找到Key的喜悦中，团队丝毫没有顾及这个错误信息带来的巨大信息量。为了寻觅彩蛋，我们孜孜不倦地尝试从解析成功的stage中挖掘出更多的内容。一个比较合理地思路就是尝试运行该stage。这个思路阴差阳错地在后面的工作起到了巨大的作用。

由于stage并不是PE文件，直接执行断然不可，因此我们尝试在Cobalt Strike走

一遍staging的过程，看看能获得点什么。图1.4为团队创建Listener的截图：

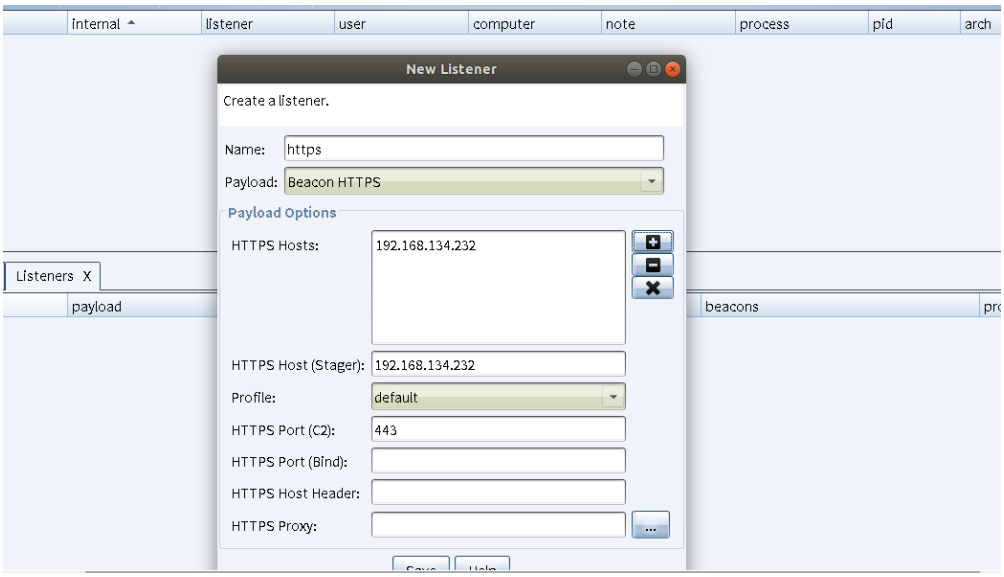


图 1.4 Cobalt Strike创建Listener

经过几天的实践，团队发现，stager运行成功后会自动去指定的web server（即上图中HTTPS Host (Stager)字段）加载stage执行，并且，我们在运行Cobalt Strike时手动配置的配置文件并不会在stager上生效。Stager对于http报文的处理会完全遵照后续加载的stage所对应的配置信息进行，即使该stage并不是原先我们listener对应的stage。这就启发我们在创建Listener时将HTTPS Host Stager域名改为d28ef1bm70qsi.cloudfront.net，这样stager就会去加载出题人设置的stage,从而让stage “跑起来”，这时候也许我们就能从流量中获取到什么。由此创建的Listener参数设置如图1.5。

团队从Wireshark捕获的流量中获得了惊人的发现，在DNS记录里面,如图1.6，我们看到了熟悉的字段：This.is.the.key.VrC5Jkwa6DOUlgtN。

我们并没有配置这个信息，但是stager却发出了这个请求，说明stage确实被成功加载运行了。团队信心满满，试图再进一步，看看能挖掘出什么更多的信息，然而，思路又一次停滞了。This.is.the.key.VrC5Jkwa6DOUlgtN并不是一个真实的域名，无论怎么尝试，连接都不可能成功。

在漫漫黑夜里求索了整整一周，光明突然刺破黑暗到来。我们还有一个stage！虽然现有的脚本无法解析出该数据，但stager很可能可以。我们只需要把创建listener时设置的托管服务器改为d32jqjeqo1vb2n.cloudfront.net，就可以看看流量里有没有什么新发现。既然Key是作为C2_Server出现，彩蛋也很可能作为C2_Server出现。

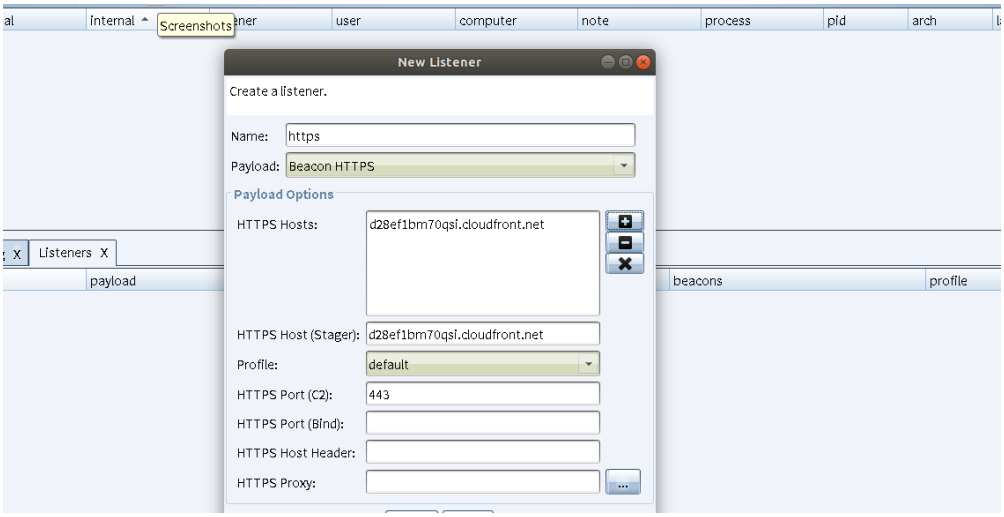


图 1.5 Cobalt Strike创建指向d28ef1bm70qsi的Listener

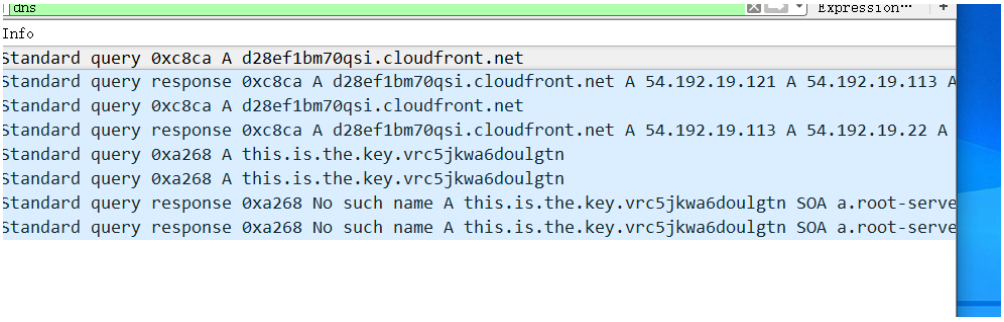


图 1.6 流量抓包发现的Key

说干就干！

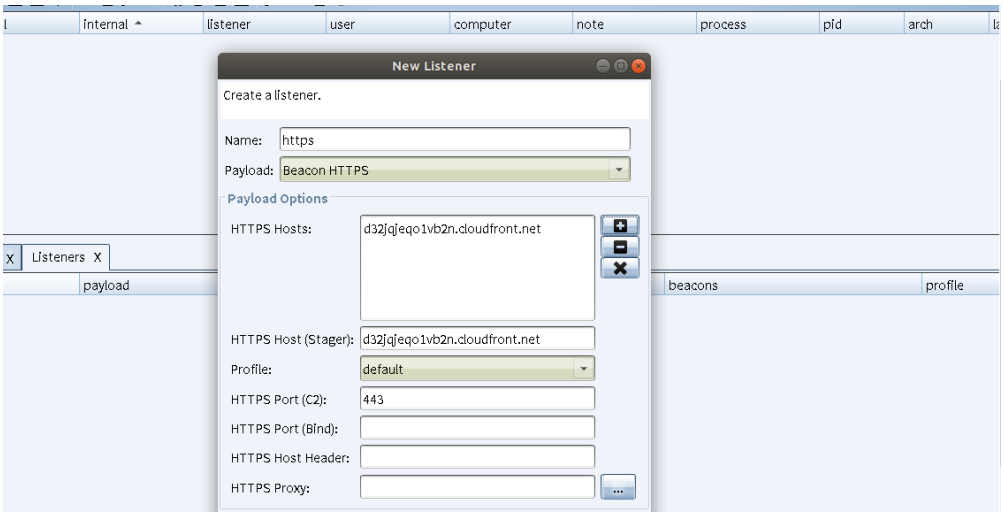


图 1.7 Cobalt Strike创建指向d32jqjeqo1vb2n的Listener

于是，我们就在抓到的流量中看到了：

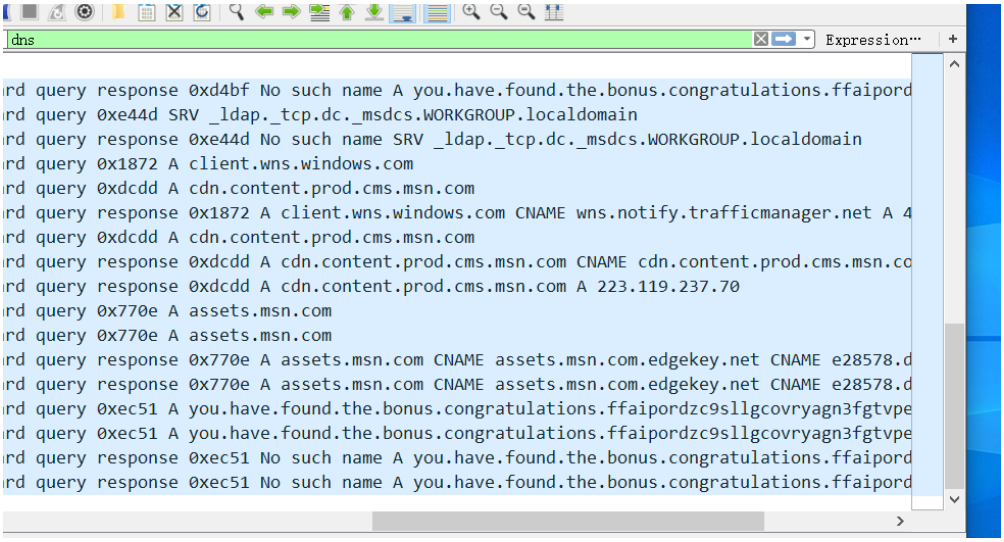


图 1.8 流量抓包发现的彩蛋

彩蛋拿下！

团队后来对彩蛋进行了深入剖析。为什么从d32jqjeqo1vb2n.cloudfront.net下载的stage无法被解析？直到查阅资料[4]后，团队判断出题人应该是更改了配置信息的默认异或值，因为异或值只有一个字节，所以只需要遍历一遍就可以找到真正的异或值。团队从CobaltStrikeParser的源码parse_beacon_config.py入手定位到解析配置文件的位置：

该函数首先根据特殊的字段定位到配置信息所在的偏移位置，特殊字段定

```
def _parse_config(self, version, quiet=False, as_json=False):
    '''
```

图 1.9 解析配置信息的函数

义在confConsts类里，由于异或值发生了改变，特殊字段不再有效，因此我们需要自己寻找偏移位置。我们尝试了一下解析成功的那个stage，发现偏移位置在196134，所以将_parse_config函数的encoded_config_offset直接定义为196134，并在_parse_config函数里添加了一个循环（测试不同的异或值）。同时，重新定义了decode_config函数。

```
    @staticmethod
    def decode_config(cfg_blob, version):
        return bytes([cfg_offset ^ confConsts.XORBYTES[version] for cfg_offset in cfg_blob])

    @staticmethod
    def decode_config1(cfg_blob, key):
        return bytes([cfg_offset ^ key for cfg_offset in cfg_blob])
```

图 1.10 重新定义decode_config函数

```
    # 196136, 196134
    encoded_config_offset = 196134
    if encoded_config_offset >= 0:
        for i in range(256):
            passbool = False
            full_config_data = cobaltstrikeConfig.decode_config1(self.data[encoded_config_offset :
            parsed_config = {}
            settings = BeaconSettings(version).settings.items()
            for conf_name, packed_conf in settings:
                parsed_setting = packed_conf.pretty_repr(full_config_data)
                parsed_config[conf_name] = parsed_setting
```

图 1.11 循环测试异或值

于是，stage文件就顺利解析出来了！如图1.12。

```
C:\Users\Jason\Desktop\找彩蛋\CobaltStrikeParser-master\CobaltStrikeParser-master>python parse_beacon_config.py "KeVV (2
)"
BeaconType                      - HTTPS
Port                            - 443
SleepTime                       - 5000
MaxGetSize                      - 1403644
Jitter                          - 0
MaxDNS                          - 255
PublicKey_MD5                   - f2d190e83dce1dbd1087f4bf57cc2e4d
C2Server                        - you.have.found.the.bonus.congratulations.ffaIpOrdZc9s1LGcOVrYagN3FGTVPewk,/jquery-3.3
.1.min.js
UserAgent                       - Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
HttpPostUri                     - /jquery-3.3.2.min.js
Malleable_C2_Instructions       - Remove 1522 bytes from the end
                                  Remove 84 bytes from the beginning
                                  Remove 3931 bytes from the beginning
                                  Base64 URL-safe decode
                                  XOR mask w/ random key
HttpGet_Metadata                - ConstHeaders
                                      Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
                                      Referer: http://code.jquery.com/
                                      Accept-Encoding: gzip, deflate
                                  Metadata
                                      base64url
                                      prepend "__cfduid="
                                      header "Cookie"
HttpPost_Metadata               - ConstHeaders
                                      Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
                                      Referer: http://code.jquery.com/
```

图 1.12 解析出来的bonus

到这儿就结束了？并没有!

实际上，除了上面两个方法，还有一种思路可以协助找到彩蛋和key。由于stage文件经过加密处理，所以我们直接读取stage文件是看不到配置信息的，但是stager肯定知道，由于key和彩蛋是作为c2域名存在，那么必然需要与之进行通信，由此可以推断存在内存里的数据有可能就包含答案，所以动态调试就一定能拿到彩蛋。嫌动态调试太麻烦的话，可以在任务管理器里创建转储文件，然后翻一翻就能找到了。

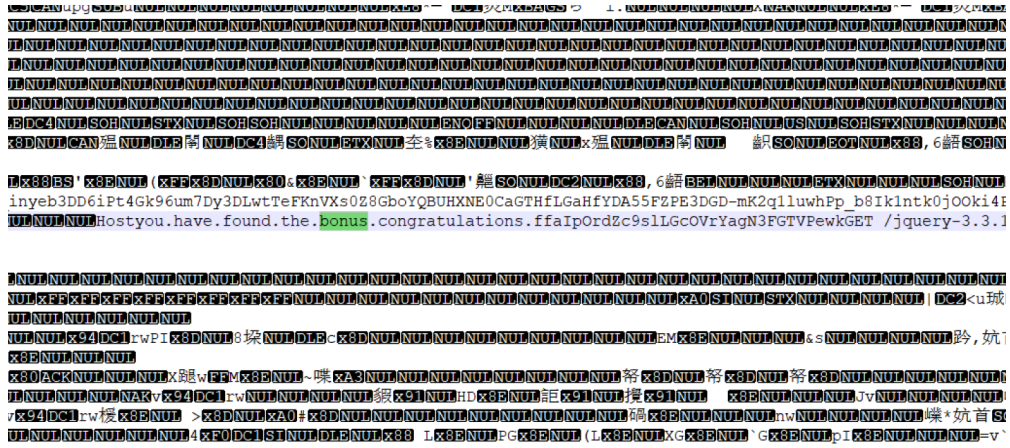


图 1.13 内存镜像中的彩蛋

1.4 总结与体会

回顾寻找Key和彩蛋的过程，团队总结了非常多的经验：1、不能放过任何一个潜在的机会，错误可能预示着重要信息；2、实践是检验真理的唯一标准，多动动手比单纯动脑要有效地多；3、不熟悉的领域一定要积极查阅资料，要仔细阅读资料提供的细节。前人栽树，后人乘凉。大牛们总结的东西，一定是重点；4、比赛也是学习的一个过程。在高强度的比赛期间，学习新知识的效率会大大提升。在学习Cobalt Strike的时候，团队翻阅了Cobalt Strike的各种教程、查阅了Java源码、研究了配置文件的书写、调研了利用CDN和域前置技术绕过检测等内容，这些信息最后大多都没能用在寻找key和彩蛋上，但是团队收获了宝贵的知识，也收获了快乐。

1.5 附加文件说明

文件说明：

parse_beacon_config-xor.py文件是团队更新的stage解析文件。

使用说明：

执行命令python parse_beacon_config-xor.py [stage]即可获取cs配置信息。

第2章　恶意流量攻击指令识别

2.1　比赛题目

本题中提供了一段时间的失陷主机的通讯流量和单个指令的流量样本的提供给参算选手，参赛选手需通过恶意软件通讯特征筛选出恶意流量，并通过提供的单一的下发指令流量作为样本，分析出通讯流量（http/dns/https）中下发的指令序列。

2.2　HTTP

2.2.1　概述

本题以主机端的 HTTP 网络流量为基础，通过流量审计的方式，结合有关 Cobalt Strike 的知识储备，对攻击者发布的指令进行分析。本题共有 50 条指令，包含5种类型，分别为 sleep、file、shell、hash 和 screen。题目提供了包含心跳包和其他各个指令的样例流量文件共 6 个，待分析的流量文件 1 个。

2.2.2　解题思路

本题的整体思路是，先从样例流量中寻找指令的特征，然后再对目标数据进行分析识别。

2.2.2.1　样例分析

在得到题目数据后，我们首先查看给出的样例文件，分析各种命令所具备的特征。命令的发布借助了域前置[5]技术，其中前置域名为 http://code.jquery.com/，Host 域名为 d32jqjeqo1vb2n.cloudfront.net。各命令以加密的形式附在 jquery-3.3.1.min.js 文件的第 3998 字节后、倒数第 1522 字节前，并有不同的长度。在收到除报活、sleep 以外的命令后，客户端会返回一个 POST 数据包，传输在宿主机获取到的数据。

各命令对应的数据流也有不同的特征。比如，screen 命令会有来自服务端和客户端双向的大流量，hash 命令有来自服务端的大流量和客户端的小流量，sleep 命令仅有 GET 数据包，没有 POST 数据包等，如图2.1所示。

通过分析各命令对应的样例数据包，我们初步确定，可以通过过滤指定域

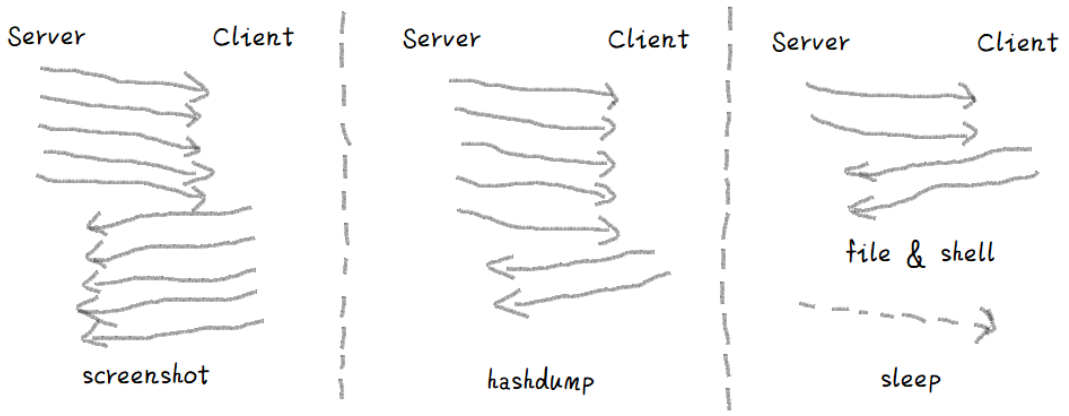


图 2.1 使用双向数据流的大小、数量等特征能初步区分不同的命令

名的方式，将包含命令的数据流从题目给出的流量文件种筛选出来，可以从数据包长度、数量、方向、HTTP 请求方式等角度来判别各种命令的内容及序列。考虑到（1）命令数量不多，（2）存在特征相似的命令（file 和 shell），（3）题目要求命令位置和顺序完全正确，我们决定手动对流量文件进行分析。分析过程中，使用到的工具主要为 WireShark。

2.2.2.2 目标数据分析

通过上述对样例的分析，我们认为这仍然是使用了域前置技术的样本。因此，通过过滤条件 http.referer == "http://code.jquery.com/" 可以把客户端和服务端的通信流量筛选出来。再结合图2.1中的粗略分析，我们大概确定有 21 条 hash 命令和 5 条 screen 命令。我们提交了一次测试数据，证实了分析的正确性。

注意到除了 sleep 命令外，其余命令都有一个 POST 数据包。因此，通过追加过滤条件 http.request.method == "POST" 能把所有 POST 数据包过滤出来，共有 45 条。由于 POST 数据包的格式比较统一，未包含干扰数据，所以我们认为一共有 5 条 sleep 命令（后来证明，此猜想是错误的）。

由于此时包含命令的关键数据已筛选出来，故我们以 HTTP 流为单位，对筛选出的数据依次进行分析，确定命令的内容和顺序。我们提取了 HTTP 流中的包长、响应时间等信息，并重点关注 POST 数据包出现前后的包长和响应时间的变化，记录了各命令数据包在数据流中的位置。

在此过程中，我们发现 HTTP 头部的 Content-Length 字段是判断命令内容的关键字段。表2.1是我们根据对数据流的分析，总结出来的 Content-Length 字段

表 2.1 Content-Length 字段和可能的命令内容对照关系

Content-Length	可能的命令	Content-Length	可能的命令
5543	beat	5713	file
5607	sleep	115580	hash
5628	shell	222076	screen

和可能的命令内容对照关系。随后，我们以 Content-Length 字段为主，结合响应时间等信息，对可能的命令进行初步排查，结果如表2.2。

根据上述方法，我们从流量文件中排查出 51 个可能的命令，包含 6 个 sleep、5 个 file、5 个 screen、13 个 shell、21 个 hash 和 1 个无法判定的命令。在判定过程中，我们无法对第 5 个命令进行判别。因为虽然它的 Content-Length 为 5607，符合 sleep 命令的特征，但该数据包后面还有一个 POST 数据包，而 sleep 命令是没有 POST 数据包的，因此这个命令便成了一大 bug。在通过其他特征无法对其判别的情况下，我们认为该命令有以下几种可能：

- 该命令是 file 或 shell；
- 该命令是 sleep；
- 该命令是偶然插入的其他命令。

针对第一、二种可能，由于命令数量已经超过 50，我们需要移除一个命令。而其余的命令都是能推导出来的，有较高的把握是正确的，不好删除或替换。我们将该命令和剩余命令一起提交，但得到的结果不符合预期。在此过程中，出题人在 QQ 群中再次确认了本题目的命令数量为 50 个。经过我们的再三研判，我们认为该命令可能是偶然插进来的其它命令，需要将其移除。在将其移除后，我们得到了符合预期的结果。

2.2.2.3　最终结果

最终提交的结果如表2.3。

2.2.3　总结与体会

本题中，我们通过 HTTP 头的 Content-Length 字段实现了对目标命令的判别。我们猜测，这类解法应该不是出题人的原意，因为碰巧所给的几种命令具备独特的长度，在命令下发时也是直接填充到 jquery-3.3.1.min.js 文件中，能够

表 2.2 依据 Content-Length 字段推测的命令

序号	位置	Content-Length	可能的命令	序号	位置	Content-Length	可能的命令
1	2167	5628	shell	27	24956	5628	shell
2	2369	5607	sleep	28	25190	5607	sleep
3	3264	115580	hash	29	25524	5628	shell
4	5684	222076	screen	30	26360	5713	file
5	6818	5607		31	26622	5628	shell
6	7154	5713	file	32	26964	5628	shell
7	7664	5628	shell	33	28610	115580	hash
8	8733	115580	hash	34	29031	115580	hash
9	10088	5713	file	35	29268	5607	sleep
10	11113	115580	hash	36	29976	5628	shell
11	12367	115580	hash	37	31144	115580	hash
12	14367	115580	hash	38	32035	115580	hash
13	14818	5628	shell	39	32854	115580	hash
14	15253	5607	sleep	40	33128	5628	shell
15	16648	115580	hash	41	33409	5628	shell
16	17747	222076	screen	42	34028	115580	hash
17	19034	5713	file	43	34851	115580	hash
18	19333	5713	file	44	35025	5628	shell
19	19819	115580	hash	45	35683	115580	hash
20	20453	115580	hash	46	35936	5607	sleep
21	21987	115580	hash	47	37913	115580	hash
22	22588	115580	hash	48	40962	115580	hash
23	22754	5607	sleep	49	42787	222076	screen
24	22972	5628	shell	50	44509	222076	screen
25	23056	5628	shell	51	47582	115580	hash
26	24233	222076	screen				

表 2.3 HTTP 指令序列最终结果

序号	命令	序号	命令	序号	命令	序号	命令
1	shell	14	hash	27	sleep	39	shell
2	sleep	15	screen	28	shell	40	shell
3	hash	16	file	29	file	41	hash
4	screen	17	file	30	shell	42	hash
5	file	18	hash	31	shell	43	shell
6	shell	19	hash	32	hash	44	hash
7	hash	20	hash	33	hash	45	sleep
8	file	21	hash	34	sleep	46	hash
9	hash	22	sleep	35	shell	47	hash
10	hash	23	shell	36	hash	48	screen
11	hash	24	shell	37	hash	49	screen
12	shell	25	screen	38	hash	50	hash
13	sleep	26	shell				

观察到很明显的不同。在实际场景中，此类技巧很可能不奏效。如果客户端和服务器之间的通信被加密，那么我们无法看到 HTTP 头的字段，也就无法从 Content-Length 字段识别命令。其次，如果客户端和服务器之间使用自定义的通信协议，对通信数据采用较高级的加密方式进行加密，那么所有加密后的数据可以是同一长度，也无法用此方法进行识别。此外，如果命令种类较多，不同的命令具备相同的长度，那么此种方法仍不能有效地区分每一种命令。在此情况下，可能要结合客户端的响应时间、响应数据包大小等来进行判别。在本题中，下发 file 和 shell 命令后，客户端返回的数据包长度不一，在一定程度上混淆了 file 和 shell 命令，同时也增加了本题的区分度。整体而言，本题属于指令识别题目的热身赛，既考察了我们对 Cobalt Strike 及其它相关知识的理解，也为后续题目的解答提供了一定的参考。

2.3　DNS

2.3.1　概述

本题以主机端的网络流量为基础，通过流量审计的方式，结合有关Cobalt Strike的知识储备对攻击者的指令进行分析和排序。

2.3.2 解题思路

本题的整体思路是，先搭建Cobalt Strike基于DNS的服务器，增加对不同指令的熟悉度；随后对指令样本进行分析提取其中的对应特征；最好对目标流量进行分析。

2.3.2.1 指令分析

通过对本赛事前面题目的作答，我们对Cobalt Strike相关机理有了基本的了解。为了更好地分析本题，我们搭建了Cobalt Strike服务器，在虚拟机中测试投递运行了DNS Beacon。然后结合题目中给的样本流量，我们得出DNS指令的几个基本特征：

（1）指令是通过查询特定域名的TXT记录下发，通过对特殊子域名（post.xx.xx）的请求完成信息回传；

（2）sleep指令修改主机的心跳周期，该指令可能携带不同的参数，如sleep 10, sleep 3；

（3）file, shell指令可能有不同的参数，因此其发送或响应的数据量有可能变化；

（4）sleep指令没有回传数据；file和shell指令通常携带一条TXT请求，多条回传数据；screen指令通常携带大量TXT请求和回传数据；hash指令通常携带大量TXT请求和相对少量的回传数据。

此外，我们在样本流量中还注意到几类域名及其解析情况。首先，主机周期性轮询2bb0a7b2.ns1.ssltestdomain.xyz和2bb0a7b2.ns2.ssltestdomain.xyz，当其中任一域名解析为74.125.196.131时，则立即查询api.0xxx.2bb0a7b2.xxx.ssltestdomain.xyz域名的A记录，如果其解析IP不为0.0.0.0，则继续查询api.1xxx...ssltestdomain.xyz域名的TXT记录，以此完成指令获取。待指令获取完毕，主机会立即向DNS服务器请求post...2bb0a7b2.xxx.ssltestdomain.xyz的A记录，从而将回传信息作为子域名反馈给服务端，具体情况如图 2.2。为了便于后文的描述，现将：

（1）api.0xxx.2bb0a7b2.xxx.ssltestdomain.xyz类型的域名，称为api类域名。

（2）post...2bb0a7b2.xxx.ssltestdomain.xyz类型的域名，称为post类域名。

2.3.2.2 样本分析及特征提取

题目所给的样本有7个文件，其中2种心跳包和5种指令，我们提取了4个特

图 2.2 域名请求情况

征，如图 2.3所示，包括：

特征1：TXT记录请求-响应包长，该特征包含了指令的信息内容。

特征2：post类域名请求-响应包长，该特征包含了回传信息的内容。

特征3：首个post类域名特征，该特征指的是在获取完指令后，第一次请求post类域名中紧跟post的字段，如域名为：post.140.09455239.2f195fb2.ns1.ssltestdomain.xyz，则提取140作为特征值，具体情况见图 2.4。

特征4：api类域名解析IP，该特征指在针对api类域名的有效解析地址（非0.0.0.0），具体情况见图 2.5。

标签	特征1：TXT记录请求-响应包长	特征2：post类域名请求-响应包长	特征3：首个post类域名特征	特征4：api类域名解析IP
sleep	104-205			74.125.196.49
shell	104-205	219-235	140	74.125.196.49
file	104-205; 104-225	277-293	1150; 1220; 1240	74.125.196.49; 74.125.196.33
hash	104-369; 105-370	277-293	1d0	74.124.134.17
screen	104-369; 105-370; 106-371	278-294; 279-295;	127230	74.124.134.17

图 2.3 DNS样本指令特征提取

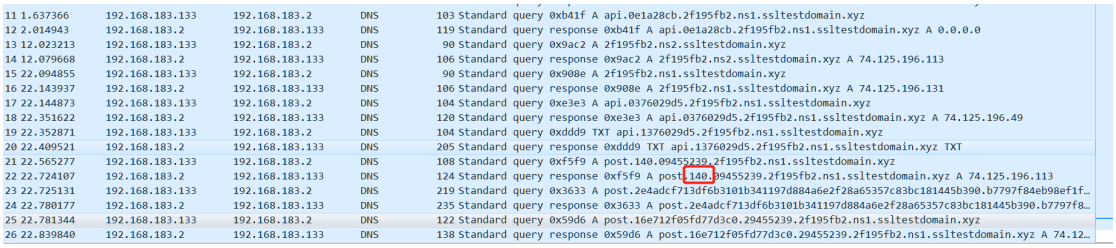


图 2.4 特征3样例

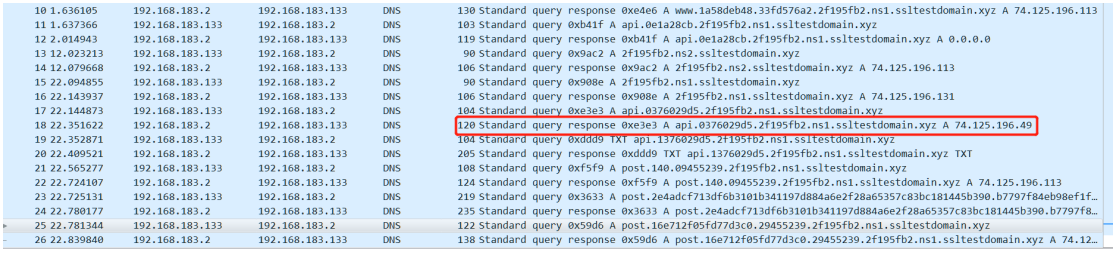


图 2.5 特征4样例

2.3.2.3 目标流量分析

题目所提供的流量包含了其他看似正常的DNS请求，结合样本流量分析，我们把ssltestdomain.xyz相关域名的数据包认为是恶意通讯流量，并通过Wireshark过滤出来（dns.qry.name contains "ssltestdomain.xyz"）,为了方便后续分析，将过滤出的恶意流量导出为.csv文件。

根据对指令的分析，我们初步认为对TXT记录的请求代表指令获取。因此，我们在流量中找到了48个疑似指令的数据块。每个数据块包含了不同数量的数据包，根据我们的初步判断，每个数据块可能是一条指令的请求与响应。但题目显示，指令应该是50个，不过无论怎么找都只有48个TXT记录，我们也是比较困惑。最后，我们决定先将这48个数据块按照样本流量所提的特征进行分析，具体特征情况如图 2.6所示。需要说明的是，在特征1和特征2提取过程中，我们注意到有很多数据包可能出现抖动和偏差，比如特征2中277-293是一个较为明显的特征值，但一些数据包长实际为279-295,276-292，比我们认为的特征值多或少几个字节，我们将这种有微小偏差的仍归类到特征277-293里。同理，特征1也做了类似的处理。

sleep指令标记。基于对特征的分析，我们首先标记了sleep的指令，总共15条，如图 2.7，可以看到除了序号1在4个特征上与其他的不同，其他的数据块的特征都是相同的，特征1为104-181，特征4为74.125.196.65。由于序列1的指令块并没有post请求数据，因此难以匹配上其他指令，这里我们暂时将其标记为sleep。接着，我们继续将104-181和74.125.196.65这两个特征作为筛选项，看看是否还存在其他数据块，发现序号2的数据块满足，如图 2.8。但是该数据块里面有post信息，不符合正常的sleep指令特征，因此暂时先不做标记。

hash&screen指令标记。然后我们对hash和screen这两个指令进行标记，由于这两个指令的数据量较多，从数量上的特征较为明显，根据样本分析阶段

序号	特征1：TXT记录请求-响应包	特征2：post类域名请求-响应包	特征3：首个post类域名特	特征4：api类域名解析IP	标签
1	104-205			74.125.196.131	
2	104-181	277-293	1180	74.125.196.65	
3	104-289	277-293	11a0	74.125.196.241	
4	104-181			74.125.196.65	
5	104-181			74.125.196.65	
6	104-353	277-293	180	74.125.196.193	
7	104-205	220-236	140	74.125.196.49	
8	104-369	279-295	1161f0	74.127.190.17	
9	104-181			74.125.196.65	
10	104-181			74.125.196.65	
11	104-181			74.125.196.65	
12	104-181			74.125.196.65	
13	104-205	220-236	140	74.125.196.49	
14	104-181			74.125.196.65	
15	104-289	277-293	1c0	74.125.196.241	
16	104-225	277-293	1260	74.125.196.33	
17	104-205	277-293	1260	74.125.196.49	
18	104-369	276-292	1d0	74.125.134.17	
19	104-181			74.125.196.65	
20	104-369	277-293	1d0	74.124.134.17	
21	104-205	277-293	1260	74.125.196.49	
22	104-289	277-293	1d0	74.125.196.241	
23	104-205	277-293	1260	74.125.196.49	
24	104-181			74.125.196.65	
25	104-369	277-293	1d0	74.124.134.17	
26	104-309	277-293	1f0	74.125.196.225	
27	104-289	277-293	1110	74.125.196.241	
28	104-289	277-293	1120	74.125.196.241	
29	104-205	220-236	140	74.125.196.49	
30	104-205	277-293	1260	74.125.196.49	
31	104-205	277-293	1260	74.125.196.49	
32	104-205	220-236	1260	74.125.196.49	
33	103-288	277-293	1140	74.125.196.241	
34	104-181			74.125.196.65	
35	104-289	277-293	1160	74.125.196.241	
36	104-205	220-236	140	74.125.196.49	
37	104-333	277-293	1170	74.125.196.209	
38	104-205	220-236	140	74.125.196.49	
39	104-289	277-293	1190	74.125.196.241	
40	104-205	277-293	1260	74.125.196.49	
41	104-289	277-293	11b0	74.125.196.241	
42	104-205	220-236	140	74.125.196.49	
43	104-181			74.125.196.65	
44	104-181			74.125.196.65	
45	104-205	277-293	1260	74.125.196.49	
46	104-181			74.125.196.65	
47	104-245	220-236	140	74.124.134.17	
48	104-181			74.125.196.65	

图 2.6 目标流量特征提取

序号	特征1：TXT记录请求-响应包长	特征2：post类域名请求-响应包长	特征3：首个post类域名特征	特征4：api类域名解析IP	标签
1	104-205			74.125.196.131	
4	104-181			74.125.196.65	
5	104-181			74.125.196.65	
9	104-181			74.125.196.65	
10	104-181			74.125.196.65	
11	104-181			74.125.196.65	
12	104-181			74.125.196.65	
14	104-181			74.125.196.65	
19	104-181			74.125.196.65	
24	104-181			74.125.196.65	
34	104-181			74.125.196.65	
43	104-181			74.125.196.65	
44	104-181			74.125.196.65	
46	104-181			74.125.196.65	
48	104-181			74.125.196.65	

图 **2.7 sleep**指令筛选

序号	特征1：TXT记录请求-响应包长	特征2：post类域名请求-响应包长	特征3：首个post类域名特征	特征4：api类域名解析IP	标签
2	104-181	277-293	1180	74.125.196.65	
4	104-181			74.125.196.65	sleep
5	104-181			74.125.196.65	sleep
9	104-181			74.125.196.65	sleep
10	104-181			74.125.196.65	sleep
11	104-181			74.125.196.65	sleep
12	104-181			74.125.196.65	sleep
14	104-181			74.125.196.65	sleep
19	104-181			74.125.196.65	sleep
24	104-181			74.125.196.65	sleep
34	104-181			74.125.196.65	sleep
43	104-181			74.125.196.65	sleep
44	104-181			74.125.196.65	sleep
46	104-181			74.125.196.65	sleep
48	104-181			74.125.196.65	sleep

图 **2.8 sleep**指令特征比较

提取的特征，我们将“特征1==104-369”的作为筛选项，得到4个数据块，如图 2.9，其中序号18、20、25四个特征相同，且与样本流量中提取的特征相同，因此标记为hash。而序号8数据块有大量的TXT请求和大量的post类域名请求，与screen的特征匹配，因此标记为screen。在这4个特征中，由于“特征2==277-293”的数据块较多，我们暂时跳过这个特征，用“特征3==1d0”的进行筛选，发现序号为22的数据块满足，但其api类域名解析IP却是74.125.196.241，与之前的均不同，因此暂时不做标记。同理使用“特征4==74.124.134.17”进行筛选，得到序号47的数据块，但其特征2与hash数据块又不相同，因此暂时不做标记。由于screen和hash的特征较为明显，且数量较少，因此在前几轮结果反馈中，我们已经确认了他们的位置和数量，当前比较难筛选的是shell和file。

序号	特征1：TXT记录请求-响应包长	特征2：post类域名请求-响应包长	特征3：首个post类域名特征	特征4：api类域名解析IP	标签
8	104-369	279-295	1161f0	74.127.190.17	screen
18	104-369	276-292	1d0	74.125.134.17	hash
20	104-369	277-293	1d0	74.124.134.17	hash
25	104-369	277-293	1d0	74.124.134.17	hash

图 2.9 screen&hash指令筛选

File&shell指令标记。我们的思路是，先标记与样本特征完全匹配的数据块，针对shell指令，我们按照“104-205、220-236、140、74.125.196.49”这一个特征组合进行筛选，总共得到6条，全部标记为shell指令，如图 2.10。针对file指令，按照“104-205，277-293，1150or1220or1240，74.125.196.49or74.125.196.33”，发现并没没有完全匹配的。

序号	特征1：TXT记录请求-响应包长	特征2：post类域名请求-响应包长	特征3：首个post类域名特征	特征4：api类域名解析IP	标签
7	104-205	220-236	140	74.125.196.49	shell
13	104-205	220-236	140	74.125.196.49	shell
29	104-205	220-236	140	74.125.196.49	shell
36	104-205	220-236	140	74.125.196.49	shell
38	104-205	220-236	140	74.125.196.49	shell
42	104-205	220-236	140	74.125.196.49	shell

图 2.10 shell指令筛选

接下来，我们将还没有标记标签的数据块进行聚类，4个特征均相同的为一类，打上同一个标签。首先按照满足“104-205，277-293，1260，74.125.196.49”

这一个特征组合的进行筛选，得到7条记录，如图 2.11。由于该组特征与file最相近，于是全部将其标记为file。剩下未标记的全部无法同时满足4个特征相同，根据在样本分析中的特征，我们发现file指令在特征3和特征4上是不固定的。因此，我们暂时不考虑3、4特征，而是通过1、2两个特征来筛选，将“104-289，277-293”作为一组，筛选出8条记录。如图 2.12,我们发现他们的特征4都是相同的，由于该组特征与file最相近，于是全部将其标记为file。

序号	特征1：TXT记录请求-响应包长	特征2：post类域名请求-响应包长	特征3：首个post类域名特征	特征4：api类域名解析IP	标签
17	104-205	277-293	1260	74.125.196.49	file
21	104-205	277-293	1260	74.125.196.49	file
23	104-205	277-293	1260	74.125.196.49	file
30	104-205	277-293	1260	74.125.196.49	file
31	104-205	277-293	1260	74.125.196.49	file
40	104-205	277-293	1260	74.125.196.49	file
45	104-205	277-293	1260	74.125.196.49	file

图 2.11 file指令筛选1

序号	特征1：TXT记录请求-响应包长	特征2：post类域名请求-响应包长	特征3：首个post类域名特征	特征4：api类域名解析IP	标签
3	104-289	277-293	11a0	74.125.196.241	file
15	104-289	277-293	1c0	74.125.196.241	file
22	104-289	277-293	1d0	74.125.196.241	file
27	104-289	277-293	1110	74.125.196.241	file
28	104-289	277-293	1120	74.125.196.241	file
35	104-289	277-293	1160	74.125.196.241	file
39	104-289	277-293	1190	74.125.196.241	file
41	104-289	277-293	11b0	74.125.196.241	file

图 2.12 file指令筛选2

剩下有8条，是通过目前的特征无法判断的，如图 2.13。因此，我们根据多数表决原则，将他们标记为更接近的那一类指令。最后得出48条记录所对应的指令。

但是，最关键的问题，总共有50条指令，而我们不管怎么数都只有48条（一度怀疑答案错了），猜测可能是其中两块数据块中隐藏了其他指令。这时，我们发现还有一个特征在前面没有考虑，那就是心跳频率，样本中给的两个心跳包频率是不同的，而sleep是可以改变心跳频率的。因此，我们大胆猜测，如果一块数据块前后心跳频率不一样，那么很有可能隐藏了sleep。

序号	特征1：TXT记录请求-响应包长	特征2：post类域名请求-响应包长	特征3：首个post类域名特征	特征4：api类域名解析IP	标签
2	104-181	277-293	1180	74.125.196.65	sleep
6	104-353	277-293	180	74.125.196.193	file
16	104-225	277-293	1260	74.125.196.33	file
26	104-309	277-293	1f0	74.125.196.225	file
32	104-205	220-236	1260	74.125.196.49	shell
33	103-288	277-293	1140	74.125.196.241	file
37	104-333	277-293	1170	74.125.196.209	file
47	104-245	220-236	140	74.124.134.17	shell

图 2.13 待确定指令筛选

为了验证猜测，我们对数据包的时间序列进行了审计，果然，在序号16和26处找到了差异。在序号16处，如图 2.14，黄色数据块上方的心跳频率是2s,而经过指令后，心跳频率变为5s。在序号26处，如图 2.15，黄色数据块上方的心跳频率是5s，而经过指令后，心跳频率变为1s。我们猜测可能是未抓到sleep数据包，也可能是一次下发了两次指令。我们暂且在这两处之后加上sleep。由于在前几轮结果反馈中，我们基本确定了hash和screen的位置，因此在17号位置和28号位置插入sleep并不影响这两个指令的位置。但是按照这个思路得分并不太理想，我们有目的的提交了几次结果，得出sleep、shell、file的个数，分别是18、17、11，而sleep还差一个，根据当前的特征，我们猜测序号2最有可能是sleep,除了带有post以外，其他均满足sleep的特征，于是将其标记为sleep，最终得到一个50个的指令序列，如图 2.16。

2.3.3　结果验证与评估

然而，根据shell和file的实际数量，当前的file过多，于是我们根据“104-205，277-293，1260，74.125.196.49”这一个特征组进行筛选，得到7条记录，并将其全部改为shell，然后得到了反馈结果比上一次要好，说明修改的一部分是正确的。但此时file仍然多出2个，shell少了2个。根据分析，我们又修改了2个file标签，将“特征3==1260”全部标记为shell。但结果还是与预想的有一点差距。

为了更好地辨析指令，我们搭建了基于DNS通信的Cobalt Strike服务器，通过自行生成指令，再利用Wireshark抓包的方式试图找到更可信的依据。测试过程我们验证了很多之前的推测，但是并没有协助我们更进一步。于是，我们转变思路，决定回溯之前的提交结果，根据排除法的原则，依次确认每个位置的

14025	1583.95567	192.168.81.2	192.168.81.139	DNS	106	Standard query response 0xbdc3 A 2bb0a7b2.ns2.ssltestdomain.xyz A 74.125.196.113
14042	1585.976918	192.168.81.139	192.168.81.2	DNS	90	Standard query 0x9cb0 A 2bb0a7b2.ns1.ssltestdomain.xyz
14044	1585.998128	192.168.81.2	192.168.81.139	DNS	106	Standard query response 0x9cb0 A 2bb0a7b2.ns1.ssltestdomain.xyz A 74.125.196.113
14140	1592.046317	192.168.81.139	192.168.81.2	DNS	90	Standard query 0xa399 A 2bb0a7b2.ns2.ssltestdomain.xyz
14141	1592.075565	192.168.81.2	192.168.81.139	DNS	106	Standard query response 0xa399 A 2bb0a7b2.ns2.ssltestdomain.xyz A 74.125.196.113
14164	1594.088702	192.168.81.139	192.168.81.2	DNS	90	Standard query 0x43b2 A 2bb0a7b2.ns1.ssltestdomain.xyz
14165	1594.122901	192.168.81.2	192.168.81.139	DNS	106	Standard query response 0x43b2 A 2bb0a7b2.ns1.ssltestdomain.xyz A 74.125.196.113
14246	1600.203194	192.168.81.139	192.168.81.2	DNS	90	Standard query 0x5216 A 2bb0a7b2.ns2.ssltestdomain.xyz
14247	1600.236224	192.168.81.2	192.168.81.139	DNS	106	Standard query response 0x5216 A 2bb0a7b2.ns2.ssltestdomain.xyz A 74.125.196.113
14259	1602.263165	192.168.81.139	192.168.81.2	DNS	90	Standard query 0x644f A 2bb0a7b2.ns1.ssltestdomain.xyz
14260	1602.286	192.168.81.2	192.168.81.139	DNS	106	Standard query response 0x644f A 2bb0a7b2.ns1.ssltestdomain.xyz A 74.125.196.131
14261	1602.287151	192.168.81.139	192.168.81.2	DNS	104	Standard query 0x50ea A api.05e6a6a45.2bb0a7b2.ns1.ssltestdomain.xyz
14262	1602.537979	192.168.81.2	192.168.81.139	DNS	120	Standard query response 0x50ea A api.05e6a6a45.2bb0a7b2.ns1.ssltestdomain.xyz A 74.125.196.33
14263	1602.53959	192.168.81.139	192.168.81.2	DNS	104	Standard query 0xcd84 TXT api.15e6a6a45.2bb0a7b2.ns1.ssltestdomain.xyz
14264	1602.77131	192.168.81.2	192.168.81.139	DNS	225	Standard query response 0xcd84 TXT api.15e6a6a45.2bb0a7b2.ns1.ssltestdomain.xyz TXT
14265	1602.815724	192.168.81.139	192.168.81.2	DNS	110	Standard query 0x79e7 A post.1260.07e9148c8.2bb0a7b2.ns1.ssltestdomain.xyz
14272	1603.83682	192.168.81.139	192.168.81.2	DNS	110	Standard query 0x79e7 A post.1260.07e9148c8.2bb0a7b2.ns1.ssltestdomain.xyz
14274	1603.881727	192.168.81.2	192.168.81.139	DNS	126	Standard query response 0x79e7 A post.1260.07e9148c8.2bb0a7b2.ns1.ssltestdomain.xyz A 74.125.1
14275	1603.883544	192.168.81.139	192.168.81.2	DNS	277	Standard query 0x2d23 A post.3174ebc4c5109bc8246b6375392dfd285c326f347c0ccad913d680c7d.
14282	1604.235698	192.168.81.2	192.168.81.139	DNS	126	Standard query response 0x79e7 A post.1260.07e9148c8.2bb0a7b2.ns1.ssltestdomain.xyz A 74.125.1
14283	1604.899623	192.168.81.139	192.168.81.2	DNS	277	Standard query 0x2d23 A post.3174ebc4c5109bc8246b6375392dfd285c326f347c0ccad913d680c7d.
14292	1605.913544	192.168.81.139	192.168.81.2	DNS	277	Standard query 0x2d23 A post.3174ebc4c5109bc8246b6375392dfd285c326f347c0ccad913d680c7d.
14293	1605.925077	192.168.81.2	192.168.81.139	DNS	293	Standard query response 0x2d23 A post.3174ebc4c5109bc8246b6375392dfd285c326f347c0ccad913
14294	1605.926642	192.168.81.139	192.168.81.2	DNS	277	Standard query 0x6c42 A post.37877c9a09c410911a967dee51d24a7e0a11bb4bd7aeecbd3392b569
14295	1605.945803	192.168.81.2	192.168.81.139	DNS	293	Standard query response 0x2d23 A post.3174ebc4c5109bc8246b6375392dfd285c326f347c0ccad913
14296	1606.126428	192.168.81.2	192.168.81.139	DNS	293	Standard query response 0x2d23 A post.3174ebc4c5109bc8246b6375392dfd285c326f347c0ccad913
14297	1606.167835	192.168.81.2	192.168.81.139	DNS	293	Standard query response 0x6c42 A post.37877c9a09c410911a967dee51d24a7e0a11bb4bd7aeecbd3
14298	1606.16996	192.168.81.139	192.168.81.2	DNS	277	Standard query 0xf95c A post.3e26eb2feb74bcfa0a297929bfb72ee213c28e556dd86627930aefd88.9
14306	1607.193114	192.168.81.139	192.168.81.2	DNS	277	Standard query 0xf95c A post.3e26eb2feb74bcfa0a297929bfb72ee213c28e556dd86627930aefd88.9
14307	1607.204993	192.168.81.2	192.168.81.139	DNS	293	Standard query response 0xf95c A post.3e26eb2feb74bcfa0a297929bfb72ee213c28e556dd8662793
14308	1607.206867	192.168.81.139	192.168.81.2	DNS	277	Standard query 0xcdc9 A post.3f181c48c9359d1c2687d54669ef4c611e773e45c284adde5d4d7a551.
14320	1608.222254	192.168.81.139	192.168.81.2	DNS	277	Standard query 0xcdc9 A post.3f181c48c9359d1c2687d54669ef4c611e773e45c284adde5d4d7a551.
14321	1608.232152	192.168.81.2	192.168.81.139	DNS	293	Standard query response 0xf95c A post.3e26eb2feb74bcfa0a297929bfb72ee213c28e556dd8662793
14322	1608.233856	192.168.81.2	192.168.81.139	DNS	293	Standard query response 0xcdc9 A post.3f181c48c9359d1c2687d54669ef4c611e773e45c284adde5c
14323	1608.235778	192.168.81.139	192.168.81.2	DNS	277	Standard query 0x6388 A post.397a0bd1f47b12f97678079f5ba4d612420b9a72159ecb3ee0d149a11
14324	1608.241669	192.168.81.2	192.168.81.139	DNS	293	Standard query response 0xcdc9 A post.3f181c48c9359d1c2687d54669ef4c611e773e45c284adde5c
14339	1609.251158	192.168.81.139	192.168.81.2	DNS	277	Standard query 0x6388 A post.397a0bd1f47b12f97678079f5ba4d612420b9a72159ecb3ee0d149a11
14364	1610.062194	192.168.81.2	192.168.81.139	DNS	293	Standard query response 0x6388 A post.397a0bd1f47b12f97678079f5ba4d612420b9a72159ecb3ee
14365	1610.064113	192.168.81.139	192.168.81.2	DNS	277	Standard query 0x0d33 A post.3a5334e1c6f2fa6801727fcaf5277471dcbf17e4105b0fa34fdf446e6.f1
14368	1610.282917	192.168.81.2	192.168.81.139	DNS	293	Standard query response 0x6388 A post.397a0bd1f47b12f97678079f5ba4d612420b9a72159ecb3ee
14369	1610.303925	192.168.81.2	192.168.81.139	DNS	293	Standard query response 0x0d33 A post.3a5334e1c6f2fa6801727fcaf5277471dcbf17e4105b0fa34fdf
14370	1610.306112	192.168.81.139	192.168.81.2	DNS	277	Standard query 0xef9c A post.3a1a3ece3e750001c7cf584d312ced74f2b37c02c7bdaeba1445a305b.
14375	1610.565736	192.168.81.2	192.168.81.139	DNS	293	Standard query response 0xef9c A post.3a1a3ece3e750001c7cf584d312ced74f2b37c02c7bdaeba14
14376	1610.568065	192.168.81.139	192.168.81.2	DNS	147	Standard query 0x4744 A post.1e1bac9db254b4c8994250ea4fccf653eeeeb52b7.87e9148c8.2bb0a7
14377	1610.798009	192.168.81.2	192.168.81.139	DNS	163	Standard query response 0x4744 A post.1e1bac9db254b4c8994250ea4fccf653eeeeb52b7.87e9148c
14428	1615.818199	192.168.81.139	192.168.81.2	DNS	90	Standard query 0x1ec8 A 2bb0a7b2.ns2.ssltestdomain.xyz
14429	1615.858902	192.168.81.2	192.168.81.139	DNS	106	Standard query response 0x1ec8 A 2bb0a7b2.ns2.ssltestdomain.xyz A 74.125.196.131
14430	1615.860076	192.168.81.139	192.168.81.2	DNS	103	Standard query 0x1f47 A api.08bd113d.2bb0a7b2.ns2.ssltestdomain.xyz
14433	1616.093014	192.168.81.2	192.168.81.139	DNS	119	Standard query response 0x1f47 A api.08bd113d.2bb0a7b2.ns2.ssltestdomain.xyz A 74.125.196.49
14434	1616.094349	192.168.81.139	192.168.81.2	DNS	103	Standard query 0x715b TXT api.18bd113d.2bb0a7b2.ns2.ssltestdomain.xyz

图 **2.14** 序号**16**数据块

25081	2469.824578	192.168.81.139	192.168.81.2	DNS	90	Standard query 0x9367 A 2bb0a7b2.ns1.ssltestdomain.xyz
25082	2469.864382	192.168.81.2	192.168.81.139	DNS	106	Standard query response 0x9367 A 2bb0a7b2.ns1.ssltestdomain.xyz A 74.125.196.113
25083	2474.881236	192.168.81.139	192.168.81.2	DNS	90	Standard query 0x3d8e A 2bb0a7b2.ns2.ssltestdomain.xyz
25084	2474.903893	192.168.81.2	192.168.81.139	DNS	106	Standard query response 0x3d8e A 2bb0a7b2.ns2.ssltestdomain.xyz A 74.125.196.113
25085	2479.906465	192.168.81.139	192.168.81.2	DNS	90	Standard query 0x0e29 A 2bb0a7b2.ns1.ssltestdomain.xyz
25086	2479.923981	192.168.81.2	192.168.81.139	DNS	106	Standard query response 0x0e29 A 2bb0a7b2.ns1.ssltestdomain.xyz A 74.125.196.113
25087	2484.94506	192.168.81.139	192.168.81.2	DNS	90	Standard query 0xb481 A 2bb0a7b2.ns2.ssltestdomain.xyz
25088	2484.99059	192.168.81.2	192.168.81.139	DNS	106	Standard query response 0xb481 A 2bb0a7b2.ns2.ssltestdomain.xyz A 74.125.196.113
25089	2490.015473	192.168.81.139	192.168.81.2	DNS	90	Standard query 0xe301 A 2bb0a7b2.ns1.ssltestdomain.xyz
25090	2490.043542	192.168.81.2	192.168.81.139	DNS	106	Standard query response 0xe301 A 2bb0a7b2.ns1.ssltestdomain.xyz A 74.125.196.113
25091	2495.053006	192.168.81.139	192.168.81.2	DNS	90	Standard query 0x47bd A 2bb0a7b2.ns2.ssltestdomain.xyz
25092	2495.086436	192.168.81.2	192.168.81.139	DNS	106	Standard query response 0x47bd A 2bb0a7b2.ns2.ssltestdomain.xyz A 74.125.196.131
25093	2495.087463	192.168.81.139	192.168.81.2	DNS	104	Standard query 0x94e9 A api.02ed67738.2bb0a7b2.ns2.ssltestdomain.xyz
25094	2495.3289	192.168.81.2	192.168.81.139	DNS	120	Standard query response 0x94e9 A api.02ed67738.2bb0a7b2.ns2.ssltestdomain.xyz A 74.125.196.225
25095	2495.330631	192.168.81.139	192.168.81.2	DNS	104	Standard query 0x1126 TXT api.12ed67738.2bb0a7b2.ns2.ssltestdomain.xyz
25096	2495.585435	192.168.81.2	192.168.81.139	DNS	309	Standard query response 0x1126 TXT api.12ed67738.2bb0a7b2.ns2.ssltestdomain.xyz TXT
25097	2495.588769	192.168.81.139	192.168.81.2	DNS	108	Standard query 0x4dcb A post.1f0.0b6f7be8.2bb0a7b2.ns2.ssltestdomain.xyz
25098	2495.828411	192.168.81.2	192.168.81.139	DNS	124	Standard query response 0x4dcb A post.1f0.0b6f7be8.2bb0a7b2.ns2.ssltestdomain.xyz A 74.125.196.
25099	2495.82979	192.168.81.139	192.168.81.2	DNS	276	Standard query 0x3122 A post.3fd277cea80d1d815314fe430f39fdbcbd06b6e3b476ba6d8567f7e8d.19
25100	2496.072389	192.168.81.2	192.168.81.139	DNS	292	Standard query response 0x3122 A post.3fd277cea80d1d815314fe430f39fdbcbd06b6e3b476ba6d856
25101	2496.073639	192.168.81.139	192.168.81.2	DNS	276	Standard query 0x1090 A post.3217912a9d82aebf2124aec06d02282f14af7185f1ab11988897657c6.284
25102	2496.315436	192.168.81.2	192.168.81.139	DNS	292	Standard query response 0x1090 A post.3217912a9d82aebf2124aec06d02282f14af7185f1ab1198889
25103	2496.316629	192.168.81.139	192.168.81.2	DNS	219	Standard query 0xc85f A post.2ca3bb8022c5c3aae94dcc2e7555281822e8906b30f710d18d34e306e.aaf
25104	2496.562429	192.168.81.2	192.168.81.139	DNS	235	Standard query response 0xc85f A post.2ca3bb8022c5c3aae94dcc2e7555281822e8906b30f710d18d34
25105	2496.563843	192.168.81.139	192.168.81.2	DNS	138	Standard query 0x3bac A post.12aee0e15b729cd4e657fad676afbb5b6.4b6f7be8.2bb0a7b2.ns2.ssltest
25106	2496.804163	192.168.81.2	192.168.81.139	DNS	154	Standard query response 0x3bac A post.12aee0e15b729cd4e657fad676afbb5b6.4b6f7be8.2bb0a7b2.
25107	2497.830005	192.168.81.139	192.168.81.2	DNS	90	Standard query 0xd2e7 A 2bb0a7b2.ns1.ssltestdomain.xyz
25108	2497.85449	192.168.81.2	192.168.81.139	DNS	106	Standard query response 0xd2e7 A 2bb0a7b2.ns1.ssltestdomain.xyz A 74.125.196.131
25109	2497.855727	192.168.81.139	192.168.81.2	DNS	103	Standard query 0x4266 A api.03937ffb.2bb0a7b2.ns1.ssltestdomain.xyz
25110	2498.09614	192.168.81.2	192.168.81.139	DNS	119	Standard query response 0x4266 A api.03937ffb.2bb0a7b2.ns1.ssltestdomain.xyz A 0.0.0.0
25111	2499.125303	192.168.81.139	192.168.81.2	DNS	104	Standard query 0x7839 A api.0347b2e03.2bb0a7b2.ns2.ssltestdomain.xyz
25112	2499.36268	192.168.81.2	192.168.81.139	DNS	120	Standard query response 0x7839 A api.0347b2e03.2bb0a7b2.ns2.ssltestdomain.xyz A 0.0.0.0
25113	2500.38883	192.168.81.139	192.168.81.2	DNS	104	Standard query 0xf2e7 A api.07fbd054c.2bb0a7b2.ns1.ssltestdomain.xyz
25114	2500.662637	192.168.81.2	192.168.81.139	DNS	120	Standard query response 0xf2e7 A api.07fbd054c.2bb0a7b2.ns1.ssltestdomain.xyz A 0.0.0.0
25115	2501.66851	192.168.81.139	192.168.81.2	DNS	90	Standard query 0xf8b5 A 2bb0a7b2.ns2.ssltestdomain.xyz
25116	2501.708087	192.168.81.2	192.168.81.139	DNS	106	Standard query response 0xf8b5 A 2bb0a7b2.ns2.ssltestdomain.xyz A 74.125.196.113
25117	2502.715937	192.168.81.139	192.168.81.2	DNS	90	Standard query 0x71ac A 2bb0a7b2.ns1.ssltestdomain.xyz

图 **2.15** 序号**26**数据块

序号	特征1：TXT记录请求-响应包	特征2：post类域名请求-响应包	特征3：首个post类域名特	特征4：api类域名解析II	标签
1	104-205			74.125.196.131	sleep
2	104-181	277-293	1180	74.125.196.65	sleep
3	104-289	277-293	11a0	74.125.196.241	file
4	104-181			74.125.196.65	sleep
5	104-181			74.125.196.65	sleep
6	104-353	277-293	180	74.125.196.193	file
7	104-205	220-236	140	74.125.196.49	shell
8	104-369	279-295	1161f0	74.127.190.17	screen
9	104-181			74.125.196.65	sleep
10	104-181			74.125.196.65	sleep
11	104-181			74.125.196.65	sleep
12	104-181			74.125.196.65	sleep
13	104-205	220-236	140	74.125.196.49	shell
14	104-181			74.125.196.65	sleep
15	104-289	277-293	1c0	74.125.196.241	file
16	104-225	277-293	1260	74.125.196.33	file
					sleep
17	104-205	277-293	1260	74.125.196.49	file
18	104-369	276-292	1d0	74.125.134.17	hash
19	104-181			74.125.196.65	sleep
20	104-369	277-293	1d0	74.124.134.17	hash
21	104-205	277-293	1260	74.125.196.49	file
22	104-289	277-293	1d0	74.125.196.241	file
23	104-205	277-293	1260	74.125.196.49	file
24	104-181			74.125.196.65	sleep
25	104-369	277-293	1d0	74.124.134.17	hash
26	104-309	277-293	1f0	74.125.196.225	file
					sleep
27	104-289	277-293	1110	74.125.196.241	file
28	104-289	277-293	1120	74.125.196.241	file
29	104-205	220-236	140	74.125.196.49	shell
30	104-205	277-293	1260	74.125.196.49	file
31	104-205	277-293	1260	74.125.196.49	file
32	104-205	220-236	1260	74.125.196.49	shell
33	103-288	277-293	1140	74.125.196.241	file
34	104-181			74.125.196.65	sleep
35	104-289	277-293	1160	74.125.196.241	file
36	104-205	220-236	140	74.125.196.49	shell
37	104-333	277-293	1170	74.125.196.209	file
38	104-205	220-236	140	74.125.196.49	shell
39	104-289	277-293	1190	74.125.196.241	file
40	104-205	277-293	1260	74.125.196.49	file
41	104-289	277-293	11b0	74.125.196.241	file
42	104-205	220-236	140	74.125.196.49	shell
43	104-181			74.125.196.65	sleep
44	104-181			74.125.196.65	sleep
45	104-205	277-293	1260	74.125.196.49	file
46	104-181			74.125.196.65	sleep
47	104-245	220-236	140	74.124.134.17	shell
48	104-181			74.125.196.65	sleep

图 2.16 50个指令序列

指令，最终确定了满分答案。虽然通过“奇技淫巧”获得了满分答案，但团队在对照指令与数据包时发现仍然有许多困惑的地方。困惑的地方主要在于序号38-序号47的数据块，根据满分答案，序号47数据块是有两个shell指令，如果其他数据块不动，那么整个指令序列就有51个。但显然，之后额外加上的两个sleep是合理的，同时也符合满分答案，前35个序列根据我们的分析思路来看也是正确的。如果按照序号47数据块存在2个指令的规则，即（1）TXT响应包长比原有认定的特征值大，（2）特征3可能存在2个。那么按照这个思路去分析，又有很多个数据块存在2条指令的可能，这就使得总指令数多于50个了。所以无论用哪种规则去判定，都存在矛盾的地方，以我们现有的分析思路和特征无法解释序号38以后的序列规则，这也是我们在拿到满分答案后百思不得其解的地方。最后，团队根据总结后的解题思路，再次提交了一版答案，获得了42分，如图2.17，标黄处是与满分答案不同的地方。

2.3.4 总结与体会

通过本题，我们进一步了解了CS DNS Beacon的原理，并尝试搭建了一套自己的CS服务端和客户端，是一次宝贵的学习经验和机会。而对于最后的结果，我们也是存在一些困惑，或许是我们整个解题思路哪里存在问题，而且在整个解题过程中，我们主要还是依靠人工特征提取和筛选，并没有找到自动化的方法或者其他更快速准确的方法去解决问题，很期待在之后的交流中能够在解题思路和方法方面有所收获。

序号	特征1：TXT记录请求-响应包	特征2：post类域名请求-响应包	特征3：首个post类域名特	特征4：api类域名解析II	标签
1	104-205			74.125.196.131	sleep
2	104-181	277-293	1180	74.125.196.65	sleep
3	104-289	277-293	11a0	74.125.196.241	file
4	104-181			74.125.196.65	sleep
5	104-181			74.125.196.65	sleep
6	104-353	277-293	180	74.125.196.193	file
7	104-205	220-236	140	74.125.196.49	shell
8	104-369	279-295	1161f0	74.127.190.17	screen
9	104-181			74.125.196.65	sleep
10	104-181			74.125.196.65	sleep
11	104-181			74.125.196.65	sleep
12	104-181			74.125.196.65	sleep
13	104-205	220-236	140	74.125.196.49	shell
14	104-181			74.125.196.65	sleep
15	104-289	277-293	1c0	74.125.196.241	file
16	104-225	277-293	1260	74.125.196.33	shell
					sleep
17	104-205	277-293	1260	74.125.196.49	shell
18	104-369	276-292	1d0	74.125.134.17	hash
19	104-181			74.125.196.65	sleep
20	104-369	277-293	1d0	74.124.134.17	hash
21	104-205	277-293	1260	74.125.196.49	shell
22	104-289	277-293	1d0	74.125.196.241	file
23	104-205	277-293	1260	74.125.196.49	shell
24	104-181			74.125.196.65	sleep
25	104-369	277-293	1d0	74.124.134.17	hash
26	104-309	277-293	1f0	74.125.196.225	file
					sleep
27	104-289	277-293	1110	74.125.196.241	file
28	104-289	277-293	1120	74.125.196.241	file
29	104-205	220-236	140	74.125.196.49	shell
30	104-205	277-293	1260	74.125.196.49	shell
31	104-205	277-293	1260	74.125.196.49	shell
32	104-205	220-236	1260	74.125.196.49	shell
33	103-288	277-293	1140	74.125.196.241	file
34	104-181			74.125.196.65	sleep
35	104-289	277-293	1160	74.125.196.241	file
36	104-205	220-236	140	74.125.196.49	shell
37	104-333	277-293	1170	74.125.196.209	shell
38	104-205	220-236	140	74.125.196.49	shell
39	104-289	277-293	1190	74.125.196.241	file
40	104-205	277-293	1260	74.125.196.49	shell
41	104-289	277-293	11b0	74.125.196.241	file
42	104-205	220-236	140	74.125.196.49	shell
43	104-181			74.125.196.65	sleep
44	104-181			74.125.196.65	sleep
45	104-205	277-293	1260	74.125.196.49	shell
46	104-181			74.125.196.65	sleep
47	104-245	220-236	140	74.124.134.17	shell
48	104-181			74.125.196.65	sleep

图 2.17 50个指令序列最终提交版

2.4 HTTPS

2.4.1 概述

以主机端的网络流量为基础，通过流量审计的方式，结合有关Cobalt Strike的知识储备对攻击者的指令进行分析。

2.4.2 解题思路

本题的整体思路是，先从熟悉Cobalt Strike指令方向入手，随后开展流量审计寻找指令的特征，然后再对目标文件进行分析识别。

2.4.2.1 指令分析

由于无法从加密流量中提取payload的有效信息，团队需要具备对Cobalt Strike指令的熟悉度。为此，团队尝试实践搭建了Cobalt Strike服务器，虚拟机中测试投递运行beacon。实践发现几个重大特征：

- sleep指令修改肉鸡的心跳周期，该指令可能携带不同的参数，如sleep 10, sleep 3；
- 所有的指令都是在服务器响应心跳时下发；
- 有些指令，如hash, screen, file ,shell会在指令下发后不久由肉鸡端发起一个新的连接，传输指令运行后需要回传的数据；
- file, shell可能有不同的参数，因此他们发送或响应的数据量有可能变化；
- 由于每次心跳、指令执行后的响应都会发起一个新的连接（源端口变化），因此分析指令序列需要基于session。

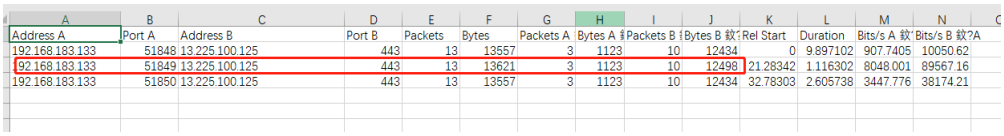
2.4.2.2 流量审计

Sample有6个文件，分别对应5种指令和1个心跳。实践的经验告诉我们，心跳是伴随所有指令一直存在的，因而首先判断心跳的特点是非常重要的。

从心跳包中，团队得到了如下特征：1、客户端到服务端的数据包和字节数一直非常稳定为3，1123；2、心跳具有周期性；3、样本流量删除了所有tcp.len==0的数据包。

由于sleep指令会改变心跳的周期，定位sleep指令会对理解目标流量和辨析指令的相对位置具有重大作用，因此团队随后审计了sleep的流量。

从sleep 10的样本中，团队得到了如下特征：1、sleep指令不会在心跳之后再建立连接传输响应信息；2、如果sleep指令的参数与原周期不同，那么sleep指令会改变周期；3、sleep指令由服务端到客户端的数据会比心跳时多64字节。

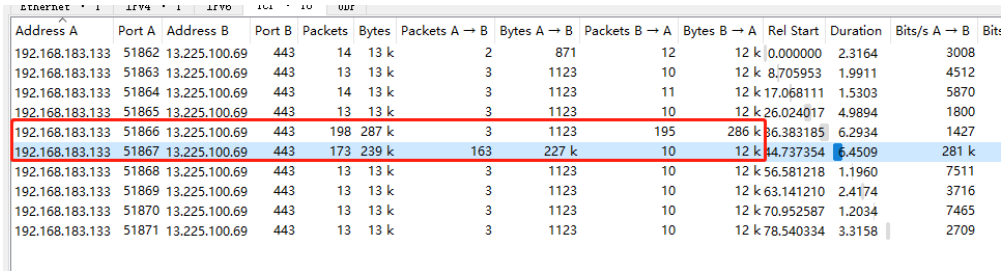


Address A	Port A	Address B	Port B	Packets	Bytes	Packets A	Bytes A	Packets B	Bytes B	Rel Start	Duration	Bits/s A	Bits/s B
192.168.183.133	51848	13.225.100.125	443	13	13557	3	1123	10	12434	0	9.897102	907.7405	10050.62
192.168.183.133	51849	13.225.100.125	443	13	13621	3	1123	10	12498	21.28342	1.116302	8048.001	89567.16
192.168.183.133	51850	13.225.100.125	443	13	13557	3	1123	10	12434	32.78303	2.605738	3447.776	38174.21

图 2.18 sleep样本特征

随后，团队依次分析了，screen和hash样本流量。

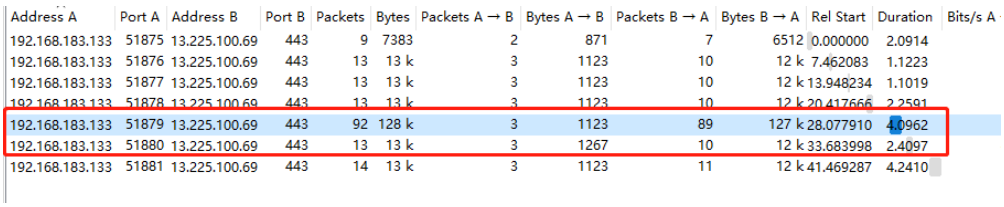
其中，screen的流量特点为：1、服务端和客户端双向的流量都会忽然变大，和其他指令的数据量是明显不同的。2、服务端到客户端的大流量是在心跳期间传输，而客户端到服务端的截屏数据是在心跳之后新建一个连接进行传输。



Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B
192.168.183.133	51862	13.225.100.69	443	14	13 k	2	871	12	12 k	0.000000	2.3164	3008
192.168.183.133	51863	13.225.100.69	443	13	13 k	3	1123	10	12 k	8.705953	1.9911	4512
192.168.183.133	51864	13.225.100.69	443	14	13 k	3	1123	11	12 k	17.068111	1.5303	5870
192.168.183.133	51865	13.225.100.69	443	13	13 k	3	1123	10	12 k	26.024017	4.9894	1800
192.168.183.133	51866	13.225.100.69	443	198	287 k	3	1123	195	286 k	36.383185	6.2934	1427
192.168.183.133	51867	13.225.100.69	443	173	239 k	163	227 k	10	12 k	44.737354	6.4509	281 k
192.168.183.133	51868	13.225.100.69	443	13	13 k	3	1123	10	12 k	56.581218	1.1960	7511
192.168.183.133	51869	13.225.100.69	443	13	13 k	3	1123	10	12 k	63.141210	2.4174	3716
192.168.183.133	51870	13.225.100.69	443	13	13 k	3	1123	10	12 k	70.952587	1.2034	7465
192.168.183.133	51871	13.225.100.69	443	13	13 k	3	1123	10	12 k	78.540334	3.3158	2709

图 2.19 screen样本特征

其中，hash的流量特点为：1、在心跳连接中，服务端到客户端的流量会急剧变大。2、随后的响应连接中，客户端到服务端的流量会稍微变大。3、响应连接与心跳连接结束的时间差与正常的心跳周期不同。



Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A
192.168.183.133	51875	13.225.100.69	443	9	7383	2	871	7	6512	0.000000	2.0914	
192.168.183.133	51876	13.225.100.69	443	13	13 k	3	1123	10	12 k	7.462083	1.1223	
192.168.183.133	51877	13.225.100.69	443	13	13 k	3	1123	10	12 k	13.948234	1.1019	
192.168.183.133	51878	13.225.100.69	443	13	13 k	3	1123	10	12 k	20.417666	2.2591	
192.168.183.133	51879	13.225.100.69	443	92	128 k	3	1123	89	127 k	28.077910	4.0962	
192.168.183.133	51880	13.225.100.69	443	13	13 k	3	1267	10	12 k	33.683998	2.4097	
192.168.183.133	51881	13.225.100.69	443	14	13 k	3	1123	11	12 k	41.469287	4.2410	

图 2.20 hash样本特征

File和shell指令同样会在心跳之后新建一个连接返回指令执行后的数据，且该连接与上次心跳结束的连接间隔时间会比较短，但是两个指令相互之间的区分非常困难，无法定性地考虑，此外，这两个指令的参数还可能发生变化，进

而导致数据量也可能发生变化。综上，为了分辨这两个指令，团队采取了定量的思路。

在样本流量中，团队找到了四个file指令，并记录了从客户端到服务端的字节数分别为1507，1838，1059和1870：

Ethernet · 1 | IPv4 · 1 | IPv6 | TCP · 15 | UDP

Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bit
192.168.183.133	51891	13.225.100.69	443	6	6361	0	0	6	6361	0.000000	0.4114	0	
192.168.183.133	51892	13.225.100.69	443	13	13 k	3	1123	10	12 k	6.772881	3.0112	2983	
192.168.183.133	51893	13.225.100.69	443	13	13 k	3	1123	10	12 k	15.149964	1.9009	4726	
192.168.183.133	51894	13.225.100.69	443	13	13 k	3	1123	10	12 k	23.433196	1.5987	5619	
192.168.183.133	51895	13.225.100.69	443	13	13 k	3	1507	10	12 k	25.394163	1.9623	6143	
192.168.183.133	51896	13.225.100.69	443	13	13 k	3	1123	10	12 k	32.716998	3.8890	2310	
192.168.183.133	51897	13.225.100.69	443	15	14 k	4	1838	11	12 k	36.977804	13.3640	1100	
192.168.183.133	51898	13.225.100.69	443	13	13 k	3	1123	10	12 k	56.722824	3.1588	2844	
192.168.183.133	51899	13.225.100.69	443	13	13 k	3	1059	10	12 k	60.240178	2.5550	3315	
192.168.183.133	51900	13.225.100.69	443	13	13 k	3	1123	10	12 k	68.166849	6.0367	1488	
192.168.183.133	51901	13.225.100.69	443	13	13 k	3	1123	10	12 k	79.580252	1.6336	5499	
192.168.183.133	51902	13.225.100.69	443	13	13 k	3	1123	10	12 k	86.576881	1.1017	8154	
192.168.183.133	51903	13.225.100.69	443	13	13 k	3	1123	10	12 k	93.038813	5.3811	1669	
192.168.183.133	51904	13.225.100.69	443	14	14 k	4	1870	10	12 k	98.770678	1.8727	7988	
192.168.183.133	51905	13.225.100.69	443	14	13 k	3	1123	11	12 k	106.025024	4.3114	2083	

图 2.21 file样本特征

在shell类样本流量中，团队找到了一个shell指令，客户端到服务端的字节数为1075字节：

Ethernet · 1 | IPv4 · 1 | IPv6 | TCP · 4 | UDP

Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B
192.168.183.133	51855	13.225.100.125	443	13	13 k	3	1123	10	12 k	0.000000	1.6030	560
192.168.183.133	51856	13.225.100.125	443	13	13 k	3	1123	10	12 k	11.996709	2.9544	304
192.168.183.133	51857	13.225.100.125	443	13	13 k	3	1075	10	12 k	15.504096	2.6582	323
192.168.183.133	51858	13.225.100.125	443	6	6454	2	472	4	5982	30.554153	0.7016	538

图 2.22 shell样本特征

2.4.2.3 目标流量分析

样本范例中获得的经验告诉我们，划分指令应该以流为单元进行，因此，团队使用Wireshark Statistics的Conversations功能转储会话数据到Excel，便于我们的分析和标注。还有一个关键细节，目标流量一定要排除掉tcp.len==0的流量，不然会大大增加分析的困难。下图为团队处理数据的范例，其中interval属性是统计下一个流的开始时间与上一个流的结尾时间的时间间隔。

Label	Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B	Bytes A→B	Packets B→A	Bytes B→A	Rel Start	Duration	Bits/s A→B	Bits/s B→A	Feature	Interval
&	192.168.81.139	54599	205.251.252	443	14	13595	3	1123	11	12472	0	0.927149	9689.9204	107615.93		
&	192.168.81.139	54600	205.251.252	443	14	13648	3	1123	11	12525	6.098764	0.709872	12655.803	141152.21	6.098764	5.171615
sleep	192.168.81.139	54601	205.251.252	443	14	13728	3	1123	11	12605	11.988543	0.920754	9757.2207	109518.94	5.889779	5.179907
&	192.168.81.139	54602	205.251.252	443	14	13664	3	1123	11	12541	15.078983	0.69728	12884.351	143884.81	3.09044	2.169686
&	192.168.81.139	54603	205.251.252	443	14	13664	3	1123	11	12541	17.9715	0.701261	12811.207	143067.99	2.892517	2.195237
?	192.168.81.139	54604	205.251.252	443	15	13867	3	1123	12	12744	20.866149	0.688979	13039.585	147975.48	2.894649	2.193388
shell	192.168.81.139	54605	205.251.252	443	14	13252	3	1075	11	12177	21.777175	1.053519	8163.1181	92467.245	0.911026	0.222047

图 2.23 目标流量会话概览

和对样本的分析一样，我们沿着心跳->sleep->screen->hash的方向进行分析，由于这三个指令的特征非常明显，因此团队轻松地找到了6个sleep，17个hash还有5个screen。剩下的22个指令，可以断言为file和shell。由于样本中流量和目标流量的性质发生了细微的变化，因此团队需要重新考虑一些信息，如从目标流量中可以看到正常心跳流从服务端到客户端的字节长度不再是稳定的，而是有多种可能，如12541，12472，12592等十余种。

团队最初是依据客户端响应指令服务器的数据量来区分shell和file，结果证明正确率并不高，于是转变思路开始从字节数目入手。针对file在样本流量中存在1507，1838，1059，1870四种字节数目的情况，团队分别进行了检索。其中符合1507的检索到了1个，符合1838的检索到了1个。根据shell在样本流量中存在1075字节的特征，团队检索到了10个shell。于是，最终仅有10个指令有待确认。从依据判断出来的shell指令中，发现，服务器到客户端的指令信息长度很多都是12568，12675和12621字节（12621比12675恰好少一个tcp包头的长度），因此，额外再找出来了4个shell。剩下的就是非常繁琐复杂的相似度分类了，如：

19	&	192.168.81.139	54816	205.251.25	443	14	13744	3	1123	11	12621	551.9175	0.868655	10342.43	116234.9	1.918019	1.182913
20	shell	192.168.81.139	54817	205.251.25	443	13	12574	4	2030	9	10544	552.9827	0.859024	18905.18	98195.16	1.065194	0.196539
21	&	192.168.81.139	54818	205.251.25	443	13	13541	3	1123	10	12418	555.0231	0.693018	12963.59	143349.8	2.040383	1.181359
22	&	192.168.81.139	54819	205.251.25	443	14	13595	3	1123	11	12472	556.8822	0.700872	12818.32	142359.8	1.859121	1.166103
23	?	192.168.81.139	54820	205.251.25	443	14	13744	3	1123	11	12621	558.7602	0.664429	13521.38	151962.1	1.877954	1.177082
24	shell	192.168.81.139	54821	205.251.25	443	15	14624	4	2030	11	12594	559.6243	0.808574	20084.74	124604.6	0.864116	0.199687
25	&	192.168.81.139	54822	205.251.25	443	13	13610	3	1123	10	12487	561.6174	0.659382	13624.88	151499.4	1.993098	1.184524
26	&	192.168.81.139	54823	205.251.25	443	15	13787	3	1123	12	12664	563.4628	0.690648	13008.07	146691.2	1.845428	1.186046

72	?	192.168.81.139	54869	205.251.25	443	15	13798	3	1123	12	12675	710.5824	0.676361	13282.85	149919.9	30.84538	30.17081
73	shell	192.168.81.139	54870	205.251.25	443	16	14625	4	2030	12	12595	711.4568	0.664907	24424.47	151540	0.874403	0.198042

最初团队并不能判定这3个指令属于什么分类，但是根据其特性的相似，团队可以大胆猜测这3个指令属于同种类型，又因为其中一个指令从服务端到客户端的字节数为12621，因此符合前面判定shell的依据，进而可以断言这三个指令均为shell。由于其中一个已经标签为shell，因此只算是新增了2个答案。

同样具有相似性地还有：

30	?	192.168.81.139	54627	205.251.25	443	13	13786	3	1123	10	12663	77.16805	0.704675	12749.14	143759.9	2.864622	2.176449
31	file	192.168.81.139	54628	205.251.25	443	12	13736	3	1571	9	12165	78.04172	0.721144	17427.86	134952.2	0.873674	0.168999
32	&	192.168.81.139	54629	205.251.25	443	13	13610	3	1123	10	12487	80.92607	0.784972	11444.99	127260.6	2.884348	2.163204
33	&	192.168.81.139	54630	205.251.25	443	14	13664	3	1123	11	12541	83.91282	0.918435	9781.857	109238	2.986745	2.201773
34	?	192.168.81.139	54631	205.251.25	443	14	13909	3	1123	11	12786	86.99881	0.793366	11323.9	128929.1	3.085989	2.167554
35	file	192.168.81.139	54632	205.251.25	443	14	14165	3	1571	11	12594	87.95689	0.795112	15806.58	126714.2	0.958084	0.164718

由此，只剩下2个指令，根据客户端到服务端的字节数量，实际上团队可以定位到该指令所在的地址，但是已无法从数据特征中识别该指令。于是，团队只能从之前上传的答案中寻找与当前判断匹配的信息，最终确定了剩下2个指令的位置。

2.4.3 总结与体会

由于HTTPS流量不具有可读性，因此分析指令只能从统计信息和指令的特点入手，为了增加对指令的熟悉度，动手实践必不可缺，不然很容易被误导。团队最初一直以为sleep指令是让肉鸡短暂休眠一段时间，在beacon中执行后才发现该指令是修改肉鸡的报活周期；此外file指令和shell指令也是在实践过程中才发现可能会有不同参数的情况。

第3章　加密代理流量分析

3.1　比赛题目-阶段一

本题中，管理员获取了内网中由数个不明用户构建的加密代理节点，这些软件使用的通信协议不完全一致，管理员只分别提取了一定数量的样本。参赛选手需要通过不同的加密代理特征，按照样本对目标流量进行分类。

3.1.1　概述

本题共有 11 个加密代理，对应 11 个类别，其中 11 个类别共包含带标签样本 51 个，无标签样本 1000 个，需要我们通过分析 51 个带有标签样本，找到 11 个类别所对应的具体的特征，从而对 1000 个测试样本进行分类。

这需要我们对不同的加密代理流量包进行分析，了解加密流量的特点，同时找到不同加密代理流量的不同特点。

3.1.2　解题思路

3.1.2.1　实验框架

由于此题类别数目较少，只有 11 个类别，且每个类别对应的带标签样本数目也很少，因此我们采取规则匹配的方法来实现分类。我们的实验框架如图3.1所示。

首先通过分析不同的类别特征制定每个类别对应的规则，从 1000 个测试样本的 pcap 包里根据规则从提取相应的数据，继而基于这些数据进行匹配分类。

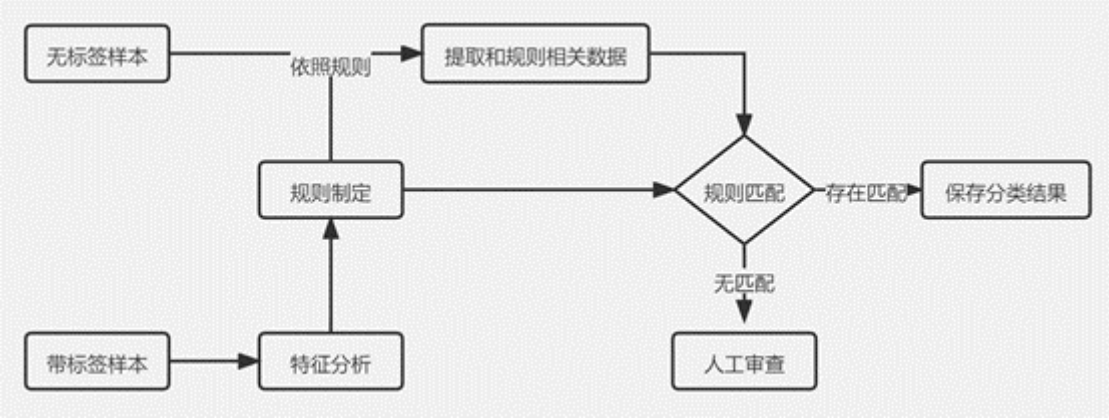


图 3.1 实验框架

最终对于极少数没有匹配结果的样本，采用人工审查的方式进行分类。值得注意的是，由于同一个样本可能满足多个类别的特点，所以不同类别的规则匹配顺序会对分类结果有一定影响。

3.1.2.2 加密代理特点分析

加密流量的特征一般包含协议特征、数据元统计特征。尤其应该注意TLS协议的流量特点。

对于恶意加密流量和良性加密流量的区别，Cisco 的研究表明[6]显示，对于 TLS 协议特征，在 ClientHello 数据包中，恶意软件提供于普通客户端完全不同的一组密码套件，此外，这些密码套件通常很脆弱或已经过时。相比之下，几乎所有良性应用程序都提供相同的密码套件。除此之外，恶意软件通常提供很少的扩展，而正常用户最多有9个拓展。此外客户端的公钥长度也存在很大的差异。在 ServerHello 和 Certificate 数据包中，恶意软件查询的服务器会选择不常见的密码套件，因为优先提供的密码套件的大小受到限制。此外，证书的有效期和SAN条目数量也存在区别，而且恶意服务器发送自签名证书的比例也比普通服务器高一个数量级。

对于数据元统计特征，恶意流量于良性流量的特征差异主要表现在数据信息、数据包的大小、到达时间序列和字节分布等。

3.1.2.3 特征分析与规则制定

基于上述加密代理的特点，我们分析 51 个带标签的样本，并着重关注了 TLS 特征。我们对协议信息、TLS 应用数据、TLS 握手信息、数据信息、数据包大小等特征进行了分析，找到了有效区分的方法。

其中有三个类别有明显的区别，只关注协议信息即可区分开来，如类别 0 对应的 pcap 包所有的通信流量只包含 UDP 协议，而在 11 个类别中，只有类别 0 具有此特点。在类别 6 中，发现只有类别 6 的流量中同时出现了 TCP 和 UDP 协议，此外，只有类别 6 中出现了 OCSP 协议。只有在类别 8 中，我们发现了 WireGuard 协议，WireGuard 是一种 VPN 协议，它的大量出现使得类别 8 的特征变得明显。通过以上方法，我们可以通过协议信息将类别 0、类别 6、类别 8 区分出来。

剩下的 8 类需要我们做进一步区分，我们发现类别 4、类别 5、类别 7、类别

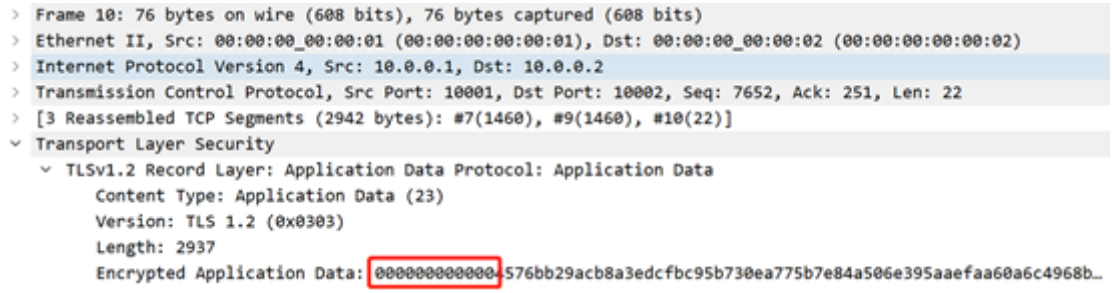


图 3.2 类别 1 特征

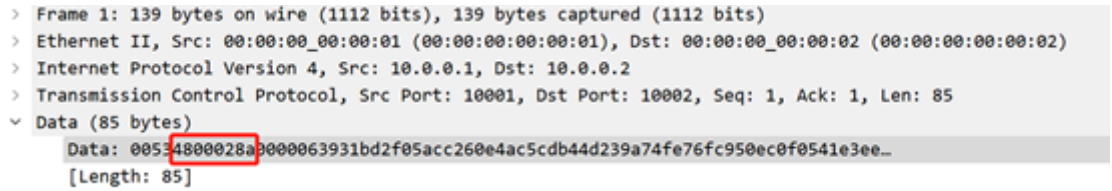


图 3.3 类别 5 特征

9 的通信流量中只包含 TCP 协议，类别 1、类别 2、类别 3、类别 10 的通信流量由 TCP+TLS 协议组成。到了此时，我们的进度开始变得缓慢，因为剩下的 8 类没有很明显的区分特征。但是在我们的不懈努力下，我们逐渐有了突破，我们发现，比如只有在类别 1 中，存在 tls.app_data 字段的前 12 位为 00:00:00:00:00:00 的流量，如图3.2所示；在类别 7 中，存在 data.data 字段的前 6 位为 00:00:00 的流量；在类别 5 中，存在 data.data 字段的第 5 位到第 12 位为 48:00:02:8a 的流量，如图3.3所示；在类别 3 中，存在 tls.handshake.extensions_length 字段值为 156 的流量；在类别 10 中，存在 tls.handshake.ciphersuite 值为 0x1301 并且 tls.handshake.type 值为 2 的流量，如图3.4所示；在类别 2 中，存在 tls.handshake.ciphersuite 值为 0x1303 并且 tls.handshake.type 值为 2 的流量；在类别 4 中，存在 frame.len 值为 1514 和 71 的流量，值得注意的是，当出现 frame.len 为 1514 时，通常下一条流量的 frame.len 为 71，如图3.5所示；在类别 9 中，存在 data.len 的值为 1424 的流量。

综上所述，我们可以通过以上方法将 11 个类别区分出来，文字描述略显繁杂，表3.1展示了我们制定的匹配规则。

3.1.2.4　实验过程

基于上述选择的特征，我们使用 tshark 工具对 pcap 包进行解析，tshark 是网络分析工具WireShark 下的一个命令行工具，主要用于命令行环境下的抓包、分

```
Handshake Protocol: Server Hello
    Handshake Type: Server Hello (2)
    Length: 124
    Version: TLS 1.2 (0x0303)
    Random: 67fc922a47f2e1233433b88804e56665c84add53d597e6f6f49ad771675bb8ef
    Session ID Length: 32
    Session ID: 223c7fac565d23c1f7a595dae559a998b45c73915eed13967d3ac04e709ae678
    Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)
    Compression Method: null (0)
```

图 **3.4** 类别 **10** 特征

```
10.0.0.2    420 0.729818  10.0.0.1  TCP   1514 10002 → 10021 [ACK] Seq=270 Ack=1271 Win=61568 Len=1460
10.0.0.2    421 0.729860  10.0.0.1  TCP     71 10002 → 10021 [PSH, ACK] Seq=1730 Ack=1271 Win=61568 Len=17
10.0.0.1    422 0.729872  10.0.0.2  TCP     54 10021 → 10002 [ACK] Seq=1271 Ack=1747 Win=131328 Len=0
10.0.0.2    423 0.729895  10.0.0.1  TCP   1514 10002 → 10021 [ACK] Seq=1747 Ack=1271 Win=61568 Len=1460
10.0.0.2    424 0.729905  10.0.0.1  TCP     71 10002 → 10021 [PSH, ACK] Seq=3207 Ack=1271 Win=61568 Len=17
10.0.0.1    425 0.729911  10.0.0.2  TCP     54 10021 → 10002 [ACK] Seq=1271 Ack=3224 Win=131328 Len=0
10.0.0.2    426 0.730062  10.0.0.1  TCP   1298 10002 → 10021 [PSH, ACK] Seq=3224 Ack=1271 Win=61568 Len=1244
10.0.0.2    427 0.732340  10.0.0.1  TCP   1514 10002 → 10016 [ACK] Seq=270 Ack=1265 Win=61568 Len=1460
10.0.0.2    428 0.732406  10.0.0.1  TCP     71 10002 → 10016 [PSH, ACK] Seq=1730 Ack=1265 Win=61568 Len=17
10.0.0.1    429 0.732421  10.0.0.2  TCP     54 10016 → 10002 [ACK] Seq=1265 Ack=1747 Win=131328 Len=0
10.0.0.2    430 0.732441  10.0.0.1  TCP   1514 10002 → 10016 [ACK] Seq=1747 Ack=1265 Win=61568 Len=1460
10.0.0.2    431 0.733239  10.0.0.1  TCP   1514 10002 → 10021 [ACK] Seq=4468 Ack=1271 Win=61568 Len=1460
10.0.0.1    432 0.733300  10.0.0.2  TCP     54 10021 → 10002 [ACK] Seq=1271 Ack=5928 Win=131328 Len=0
10.0.0.2    433 0.733333  10.0.0.1  TCP   1315 10002 → 10016 [PSH, ACK] Seq=3207 Ack=1265 Win=61568 Len=1261
10.0.0.1    434 0.733344  10.0.0.2  TCP     54 10016 → 10002 [ACK] Seq=1265 Ack=4468 Win=131328 Len=0
10.0.0.2    435 0.733352  10.0.0.1  TCP     71 10002 → 10021 [PSH, ACK] Seq=5928 Ack=1271 Win=61568 Len=17
10.0.0.2    436 0.733367  10.0.0.1  TCP   1514 10002 → 10016 [ACK] Seq=4468 Ack=1265 Win=61568 Len=1460
10.0.0.2    437 0.733377  10.0.0.1  TCP     71 10002 → 10016 [PSH, ACK] Seq=5928 Ack=1265 Win=61568 Len=17
```

图 **3.5** 类别 **4** 特征

表 **3.1** 加密代理流量分析 **Stage1** 匹配规则

Label	**Protocol**	**Rule**
0	UDP	∀ protocol = UDP
1	TCP+TLS	∃ tls.app_data[0:12] = 00:00:00:00:00:00
2	TCP+TLS	∃ (tls.handshake.ciphersuite = 0x1303 and tls.handshake.type = 2)
3	TCP+TLS	∃ tls.handshake.extensions_length=156
4	TCP	∃ (frame.len = 1514 and next frame.len = 71)
5	TCP	∃ data.data[4:12] = 48:00:02:8a
6	(TCP+UDP)/OCSP	(∃ protocol = TCP and ∃ protocol = UDP) or ∃ protocol = OCSP
7	TCP	∃ data.data[0:6] = 00:00:00
8	WireGuard	∃ protocol = WireGuard
9	TCP	∃ data.len = 1424
10	TCP+TLS	∃ (tls.handshake.ciphersuite = 0x1301 and tls.handshake.type = 2)

析工作。我们使用tshark提取了protocol、tls.app_data、tls.handshake.extensions_length、tls.handshake.type、tls.handshake.ciphersuite、data.data、data.len 和 frame.len 特征。

我们将第3.1.2.3节中选择的特征作为每个类别的匹配规则，我们的规则匹配算法如算法1如下所示。

算法 1

Require: Samples

Ensure: Result

```
Set Rules = [ r0, r6, r8, r1, r5, r7, r10, r3, r2, r4, r9 ]
for sample in Samples do
    sample = tshark(sample)
    for rule in Rules do
        if sample match rule then
            Label(sample) = Label(rule)
            Result[sample] = Label(sample)
            break
        end if
    end for
    Result[sample] = manual_check(sample)
end for
```

在上述算法中，r0 表示类别 0 的规则，相应的 r9 表示类别 9 的规则。我们首先定义 11 个类别的规则匹配顺序，然后使用 tshark 提取样本所需的数据，将样本按照规则匹配顺序依次对 11 个规则进行匹配，如果命中某一规则，则中止匹配后续其他规则，同时给该样本标注分类结果，如果 11 个规则都没有命中，则需要借助人工审查。需要注意的是，在算法 1 中，Rules 列表的顺序是可调整的，这是因为在 1000 个测试样本中，有的样本可能会存在同时满足多个类别规则的情况，比如 sample1 的实际标签是类别 2，但同时满足类别 1 和类别 2 的规则，而先命中规则 1 后，将不在匹配规则 2，这就导致了错误的分类，因此我们需要调整 Rules 列表中规则的顺序来达到更好的分类效果。

我们根据类别规则的鲁棒性来进行排序，我们首先确定了 r0、r6、r8、r1、

表 3.2 分类结果数量统计

Label	0	1	2	3	4	5	6	7	8	9	10	?
Number	53	98	95	106	92	98	97	94	53	109	99	6

r5、r7 和 r4 的顺序，因为这些规则特征尤为明显，对于剩下的规则顺序我们根据得分反馈不断进行调整，最终确定了如上述算法中的规则匹配顺序。

3.1.3 结果验证与评估

通过实验我们得到了分类结果，其中 994 个样本命中了匹配规则，只有 6 个样本没有命中匹配规则，分类统计表3.2所示。

对于剩余的 6 个样本，我们采用人工审查的方法对这 6 个样本进行分工，最终我们得到了 100 分的结果，这说明了我们通过规则匹配命中的 994 个样本全部分类正确，规则匹配的准确率是 99.4%，辅以极小成本的人工审查，我们可以达到 100% 的准确率，充分说明了我们选取的特征及规则的有效性。

有效的规则制定和匹配顺序极大程度降低了人工的成本，即便在没有人工审查的阶段，我们依然可以得到 99.4% 的准确率。值得注意的是，在所有参赛队伍中我们是唯一在该题中获得满分的队伍，这意味着我们将 1000 个样本全部分类正确。

3.1.4 总结与体会

通过解决该题我们意识到特征选择的重要性，特征选择的质量直接影响到规则的质量和分类的结果。而特征如何选择，需要我们积极查阅资料，借鉴前人的工作成果，了解加密代理流量的特点，为我们选择特征奠定基础。

同时，在实验过程中要不断尝试不断探索，以这道题为例，我们需要发掘区分的特征，需要尝试不同的匹配顺序，总之坚持就会带来希望。

3.1.5 附加文件说明

文件说明：

real_data 目录下应为官方提供的 1000 个待测样本；

stage1.py 的功能是利用 tshark 提取样本数据并制定规则完成匹配，生成分类结果；

使用说明：

在相应目录下运行 python3 stage1.py，即可生成分类结果文件 result.txt。

3.2　比赛题目-阶段二

身份不明的用户或恶意软件可能使用未授权的加密代理进行通信，访问恶意网站或正常网站。确认这些行为的详细信息有利于对可疑用户行为和恶意软件进行分析，但截获的加密流量无法进行破解。

本题中，管理员使用机器自动产生了恶意软件与多个恶意网站（或正常网站）通信产生的，经过加密代理的流量。根据已知的信息，管理员提取了一部分可以确定类别的样本信息。参赛选手需要利用管理员生成的样本，标记每个流量包可能访问站点的标签。

3.2.1　概述

题目主要考察加密流量识别能力，需要选手根据提供的已知类别样本，识别测试数据可能可能访问何种恶意站点。经过对题目和数据的初步分析，我们发现所提供加密流量数据均由同一加密代理软件产生，且数据包中并未观测到特殊应用层协议可供分析。因此，我们确定了从统计特征和行为特征两个角度角度对加密流量样本进行审计分析，深入挖掘标识访问不同恶意通信流量的有效特征，从而实现对恶意流量的自动化分类的基本思路。

图3.6展示了我们方案的整体流程。具体来讲，我们分别提取训练样本的多维统计特征及标识行为特征的包长序列特征，分别构建分类器，基于投票机制结合多维分类结果来获得最终判定结果，从而避免异构数据混淆和权重问题带来的误分类情况。接下来我们将分别介绍我们所采用的特征及对应的分类器、实现和性能以及展望和总结。

表 3.3 数据分析

文件夹	说明	文件数量
Train_Data	样本流量，文件名格式为 label_n.pcap（共计100类）,其中label为样本的标签(代表与n网站通信产生的流量)，n为序号	100类，共1401个流量样本
Test_Data	目标流量，由不同的网站经过同一加密代理软件产生	8153个流量样本

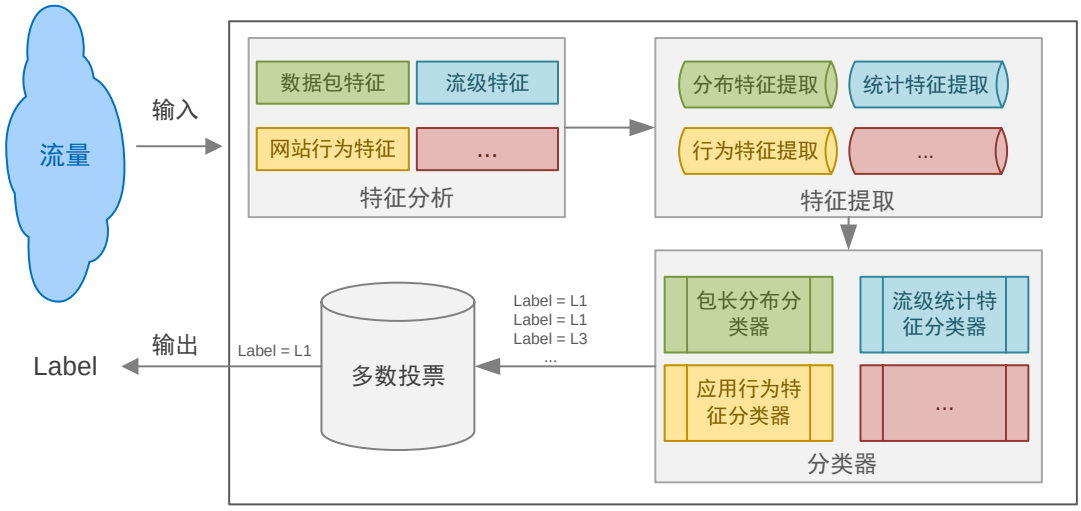


图 3.6 整体流程图

3.2.2 解题思路

数据分析

拿到数据后我们首先对提供的训练样本和测试样本进行初步分整理分析，情况如表3.3：

*train_data*文件夹中，共包含1401个流量样本，除第85类只有2个样本，29、52、94只有6个样本，平均每个家族都有13-15个样本。

下面我们将详细介绍六个参与投票的子模型中的每一个模型所采用的特征及对应的分类器，以及设计的动机和意义。

3.2.2.1 数据包特征

动机

由于网络中不同的服务商提供的服务不同，导致数据流中的数据分组大小存在一定的差异，如流媒体的数据分组较小以提高播放流畅度而文件下载通常是以最大的负载进行传输[7]。由于数据分组大小与网络服务有关且不受加密技术的影响，因此可以根据数据分组的分布对加密流量进行识别。

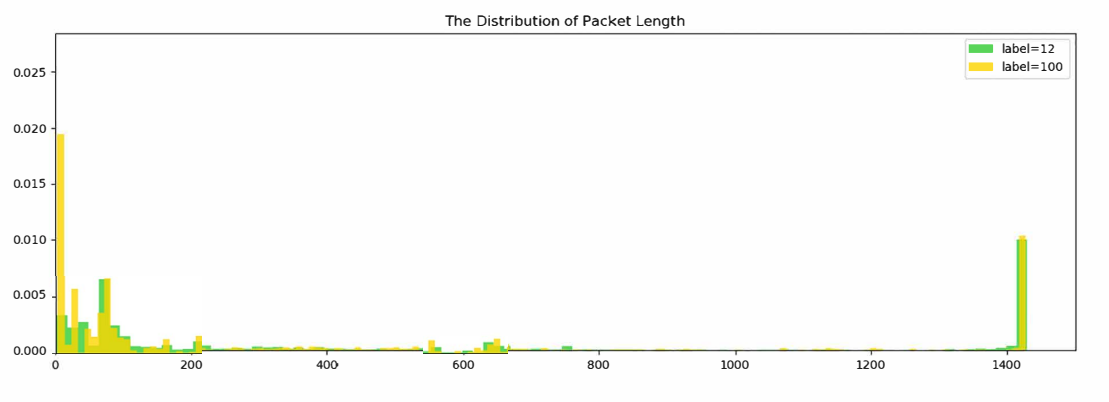


图 3.7 不同类别数据包长分布的区别

特征提取与分类器

我们根据方向的不同，首先针对每个pcap文件分别提取前向包有效负载长度序列和后向有效负载长度序列。然后每个序列分别提取方差、均值、四分位值、众数、平均数、最大值等特征从而描述分布特点。

由于包长分布的特征本质上是一个离散概率分布，特征值之间的差值可以很好的描述不同分布之间的差异，因此我们选择基于欧拉距离度量的KNN分类算法。KNN通过测量不同特征值之间的距离来讲待检测样本分类到训练样本之中，因此并不适用于特征维度过高的数据，我们所提取的12维特征可以较好地应用于此。

3.2.2.2　流级特征

动机

根据五元组（源IP、目的IP、源端口，目的端口，协议）划分得到的会话数据能够最大程度的保留客户端和服务端之间的通信特征信息，因此成为了流量分析领域常用的分析对象[8]。通信双方的会话级别的统计特征能够很好的反映不同类型应用的属性。如恶意通信往往具有持续时间短、传输速率快、出入站比例大等特点，而正常流量则往往具有持续时间长、包长分布均匀、数据包到达间隔稳定、出入站比例较小等特点。

本题中提供的加密流量只是对数据包的载荷进行加密，对流的特征属性的影响较小，因此可以根据流量的属性如间隔时间，报文大小，流持续时间等提取相应的流量特征，根据流量特征准确的识别出访问不同网站的加密流量的类

别。如图所示，为题目提供训练样本中不同类别的加密流量在流级统计特征上的分布差异。

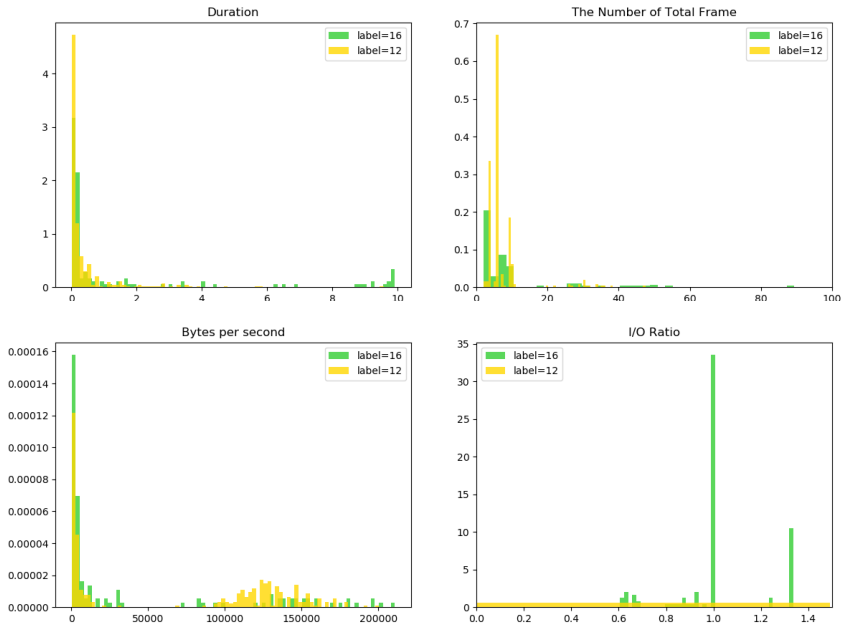


图 3.8 不同类别流级特征的区别

特征提取与分类器

我们利用SplitCap工具按照（源IP、目的IP、源端口，目的端口，协议）五元组将提供的训练和测试数据分别拆分成单独的会话，并过滤掉未完成三次握手的会话。我们认为未完成三次握手的会话可能是由于抓包或是题目设定原因出现了截断，而这种不完整的会话若何混合在其他完整的会话中一同提取统计特征，将会使特征变得混淆而不可用。因此我们从全部切分得到的46,667个会话文件中过滤后得到可用于特征提取和模型训练的22,211条完整会话。

在确定并获得训练数据后，我们直接实现了从原始报文数据中的流量特征提取，没有依赖tshark等第三方工具。具体来讲，我们基于python和dpkt库，直接将pcap文件作为二进制流读取并依据协议解析每个字段，从而计算80+维的会话统计特征。具体使用特征如表3.4所示：

特征提取后，我们选择XGBoost作为流级特征的分类算法。XGBoost（eXtreme Gradient Boosting）是基于Boosting框架的一个算法工具包（包括工程实现），在并行计算效率、缺失值处理、预测性能上都非常强大。同时基于树的方法可以直接对特征重要性进行评分，这对于后续挑选重要特征、降低特征维度、删除

表 3.4 特征选取

特征	说明
持续时间	会话从建立连接到最好一个数据包的间隔时间
数据包达到间隔	包的时间间隔、正向包的时间间隔、反向包的时间间隔等
包数特征	总包数、前向包数、反向包数
包长度特征	正向包数量、反向包数量
字节长度特征	正向字节数量、反向字节数量
传输速率	字节速率、数据包速率、正向和反向数据包速率、正向和反向字节速率
Flag特征	SYN、ACK、PSH、RST、URG等
窗口尺寸	TCP Window_Size
上下行比例	上下行数据包比例、上下行字节数比例

冗余特征十分方便，同时还可以对*max_depth*参数进行限制防止特征过于细化和线性相关带来的过拟合风险。

3.2.2.3　网站行为特征

动机

由于本题数据说明提到所有样本均由同一加密代理生成，差异在于访问了不同的目标网站。而恶意网站由于其目的和功能的不同，在通信过程中往往存在特定的通信模式和规律，通过分析比较有效负载长度序列的异同，可有效定位不同类别的访问流量[9]，如图3.9。与流级特征中提取的包长统计特征不同，该部分重点关注的是序列特征，通过数据包长度的时序性交错变化来反映不同网站的应用和业务特征。在加密代理不变且网站业务功能稳定的情况下，利用基于包长序列的网站应用行为特征进行加密网站流量分类具备相当的稳定性和精确度。

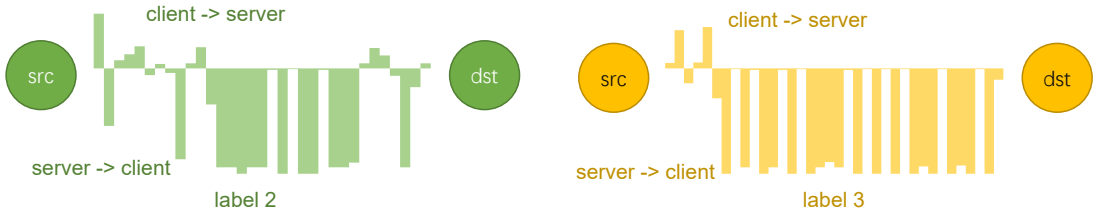


图 3.9 不同类别包长度序列的区别

特征提取与分类器

在这样的想法推动下，我们提取所有包含三次握手的完整会话，并提取其有效负载长度序列。如图3.10所示，可以明显观察到不同网站访问流量的包长序列之间的差异。

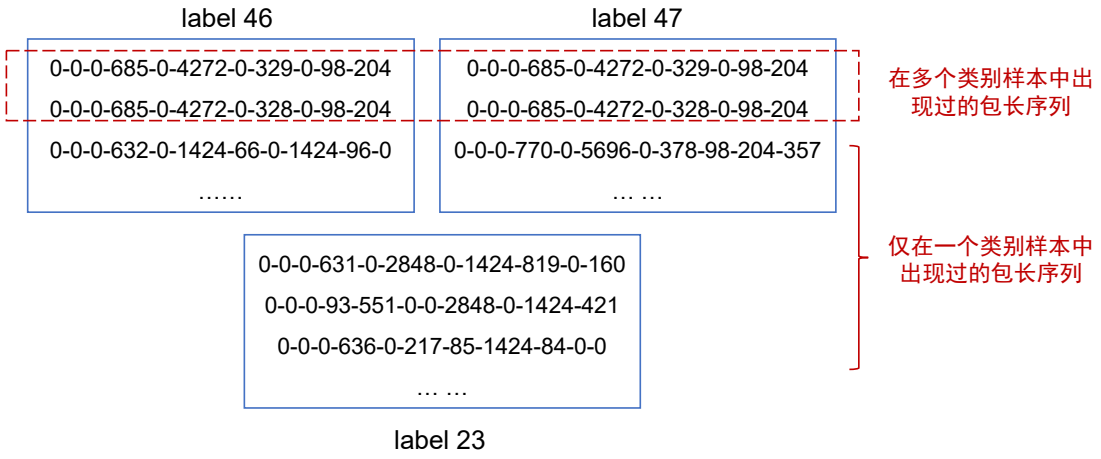


图 3.10 不同类别包长序列的区别

为证明该特征的有效性，我们先将序列长度窗口设定为11，在1401个训练样本中获得了1396个样本所包含的独特包长序列，并基于匹配的强规则确定4734个样本的标签。此次提交获得了55.18分，即提交内容获得了94%左右的正确率，从根本上证明该特征的有效性。

在获得以上的验证结果后，我们开始思考如何将以上的匹配规则“软化”成机器学习可以处理的形式。我们希望能够兼顾包长序列在莫以类别的出现频率和“特异程度”，结合以往课程和信息检索相关知识，我们想到了TF-IDF(term frequency–inverse document frequency)技术。即将包长序列视为item，拼接后形成对应该类别的“文档”，随后基于tf-idf思想，计算每个item的出现频率tf和特异性程度idf，两者结合后用来代表该类别的特征向量。在本题提供的数据中，我们共得到了1306维的特征，也就是说1396的训练样本和8109的测试样本，共出现不同的包长序列1306个。

在获得特征向量后，我们依然选择了在前面就已经取得良好表现的梯度提升树算法——XGBoost作为该部分的子分类器。主要是考虑到该部分特征向量维度较高，且容易出现过拟合的问题。因此，我们在设定树模型的超参数时，集中调整了如max_depth、$colsample\_bytree$、$subsample$以及eta、$lambda$、$alpha$等控制数据采样和正则化程度的超参数。

3.2.3　结果验证与评估

我们基于python的sklearn和dpkt库实现了相应的机器学习模型和特征提取。由于所提供数据包规模不大，dpkt并为遇到内存爆炸无法打开的情况，sklearn中所包含的KNN模型和XGBoost模型则均采用之前多次试验得到的较好的超参数设置。而且在流级特征分类时，我们基于XGBoost树模型特有的特征重要度评分功能，进行了一定程度的特征选择，最终得到了纬度较低且精确度较高的模型作为最终使用的判定模型。

给予我们的多级别特征提取和分类器投票结果，提交多次后发现并未达到理想的分数。因此后期我们又基于KNN算法和tf-idf特征进行进一步分类，将多个模型判定结果均相同的2270个测试样本提取出来补充相对较少的训练样本集重新训练各部分子分类器，最终达到了87分的结果。

3.2.4　总结与体会

随着近来互联网的加速发展和人民网络安全意识的增强，网络加密技术开始大范围的应用，加密流量爆发式增长，这为流量识别和分类带来了极大挑战。在加密流量识别技术也伴随着机器学习技术的发展而得到了长足的进步，但在常规应用场景下一般来讲第一想法就是通过对TLS协议特征进行分析或是基于图像纹理来识别有效负载的数据分布特征，但这些常规的思路在该题目的设定下都很难发挥作用，因为本题的所有加密流量都是由同一加密代理加密，仅访问目标站点存在差异。所以在仔细分析题目和样本后，我们跳出常规思路，从行为和统计特征两个角度全面刻画访问不同站点所生成的流量样本之间的差异，结合多种机器学习算法，构建多维分类识别系统，在不解密有效负载的前提下，实现了更加精确地加密流量分类。

在解题过程中，由于训练样本数量少、不同特征之间差异较大且容易模糊，确实为识别工作带来很大的干扰。常规特征的提取和多分类器多数加权投票的判决机制确实能够取得一定程度上的精确率，但在面对真实环境中更加广泛和多样化的加密流量数据时，现有的特征和常规模型是否仍然能够有较好的表现，如何深度挖掘不同应用场景下流量样本的差异，全面刻画样本特征，仍然需要研究人员对加密机制、通信机制等根本性原理进行探索和学习，从而进一步推动加密流量检测技术的持续发展。

3.2.5 附加文件说明

pkt_len_analyze.py，用于提取数据包长度分布特征，并生成相应的csv文件。

flow_feature_analyze.py，用于提取流级统计特征，并生成相应的csv文件。

clf_XGBoost.py，用于读取特征文件，训练XGBoost模型，并保存训练好的模型以便后期预测。

tfidf.py，用于读取包长序列，并拼接形成“文档”，基于tf-idf算法生成高维特征向量，从而训练XGBoost或KNN模型。

参考文献

[1] 大海捞“帧”：Cobalt Strike服务器识别与staging beacon扫描 [EB/OL]. Ocotober. 2021. https://www.freebuf.com/articles/network/273480.html.

[2] 魔改CobaltStrike [EB/OL]. Ocotober. 2021. https://bbs.pediy.com/thread-267208.htm.

[3] Cobaltstrikeparser [EB/OL]. Ocotober. 2021. https://github.com/Sentinel-One/CobaltStrikeParser.

[4] Bypass cobaltstrike beacon config scan [EB/OL]. Ocotober. 2021. https://cloud.tencent.com/developer/article/1764340.

[5] Fifield D, Lan C, Hynes R, et al. Blocking-resistant communication through domain fronting [J/OL]. Proc. Priv. Enhancing Technol., 2015, 2015(2): 46-64. https://doi.org/10.1515/popets-2015-0009.

[6] Roques O. Detecting malware in tls traffic [D]. Imperial College London, 2019.

[7] Qin T, Wang L, Liu Z, et al. Robust application identification methods for p2p and voip traffic classification in backbone networks [J]. Knowledge-Based Systems, 2015, 82: 152-162.

[8] 陈良臣, 高曙, 刘宝旭, 等. 网络加密流量识别研究进展及发展趋势 [J]. 信息网络安全, 2019, 3.

[9] Dong C, Lu Z, Cui Z, et al. Mbtree: Detecting encryption rats communication using malicious behavior tree [J/OL]. IEEE Transactions on Information Forensics and Security, 2021, 16: 3589-3603. DOI: 10.1109/TIFS.2021.3071595.