攻守道—流量分析的刀光剑影（下）

原创 队员编号023 酒仙桥六号部队 6月23日

这是 **酒仙桥六号部队** 的第 **24** 篇文章。

全文共计3449个字，预计阅读时长11分钟。

大家好，我又回来了，上一期我们对那一起攻击事件流量的分析只进行了一半，事情还远未结束，这期，我们继续来进行探究。

第五问 数据库

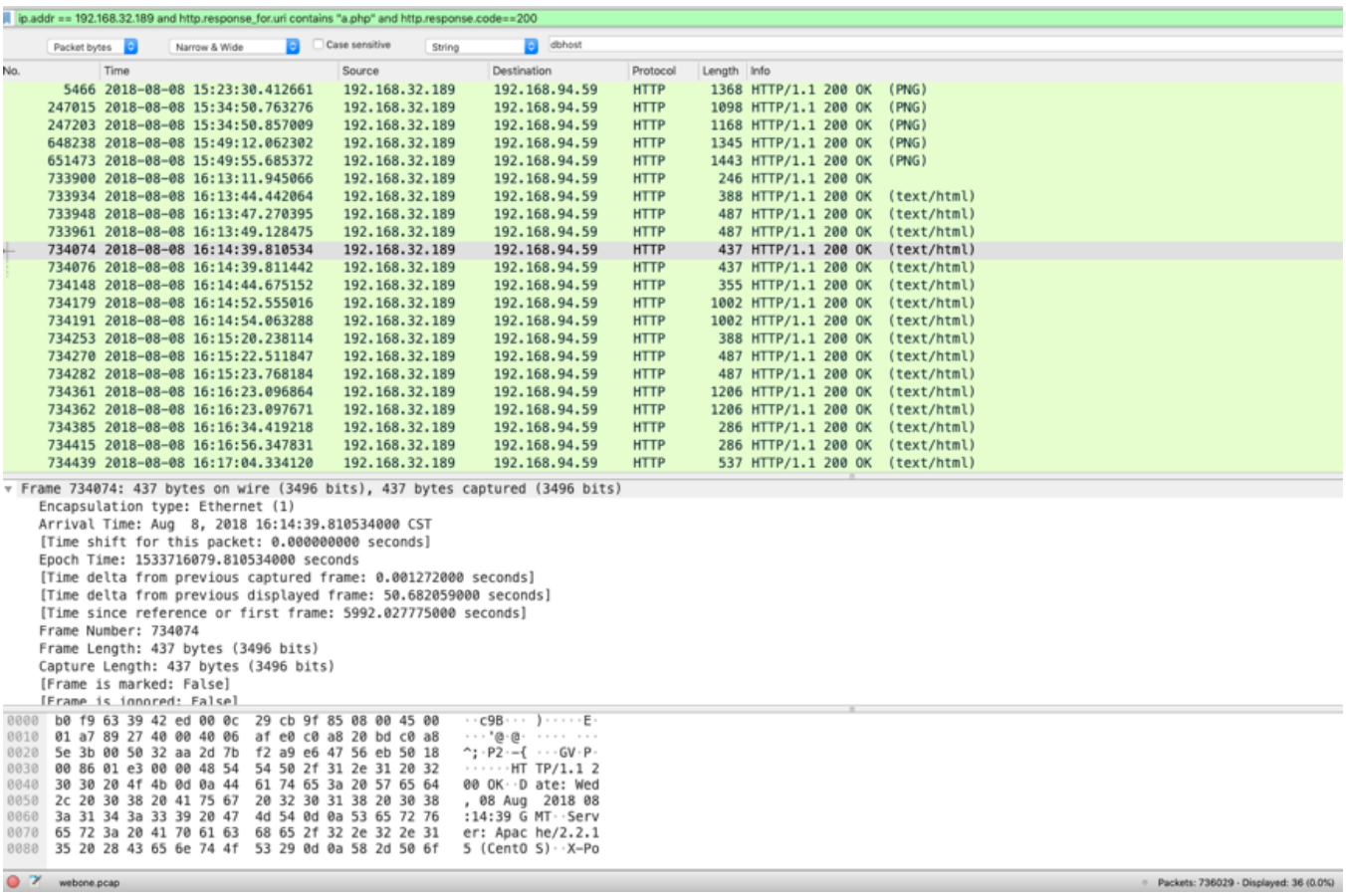
攻击者拿到的数据库配置信息是什么。

前面攻击者已经拿下了一个网站，并上传了webshell，通常都会读取网站的配置文件来获取数据库的账号密码，然后连接数据库进行数据脱取等操作，那么，我们就来从流量中找找攻击者读取的数据库文件。

通常，在php语言的网站中，数据库的配置会写在config.php、database.php等名称的文件中，Java语言的网站中，当使用一些框架时，数据库的配置可能会写在datasource.xml、config.xml、config.properties等名称的文件中，一般情况下，在这些文件中写数据库的连接信息时，都会有hostname、dbname、username、password这类的关键字，在java中还会有jdbc的字段。大多数情况下，这些配置信息在配置文件中都是明文存储的，当存在任意文件读取、下载、获取webshell之后都可以通过读取这些文件直接获取数据库信息。

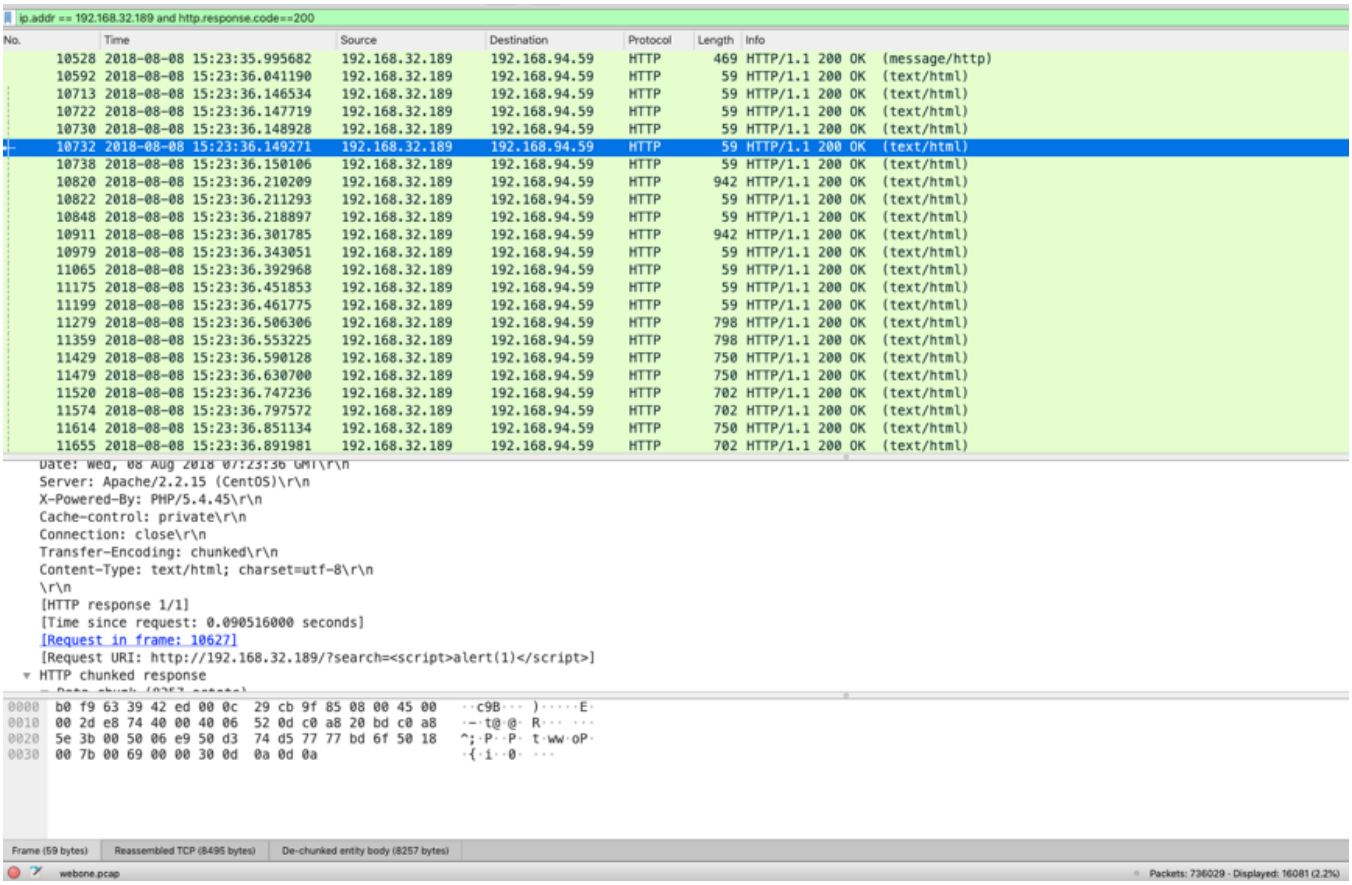
现在我们就来看看攻击者的流量，看看他读到的数据库信息是什么。首先我们能想到，攻击者读取到的数据库信息肯定是在返回包中显示，再通过我们之前分析出来的攻击者使用的名称为a.php的webshell来筛选一下。先筛选攻击者IP发出的请求，并且返回包是由a.php请求来的：ip.addr == 192.168.32.189 and http.response_for.uri contains "a.php"

筛选出来90条数据包，再把返回包200的请求筛选出来，就只剩下36条数据包，这样找起来就更容易了。

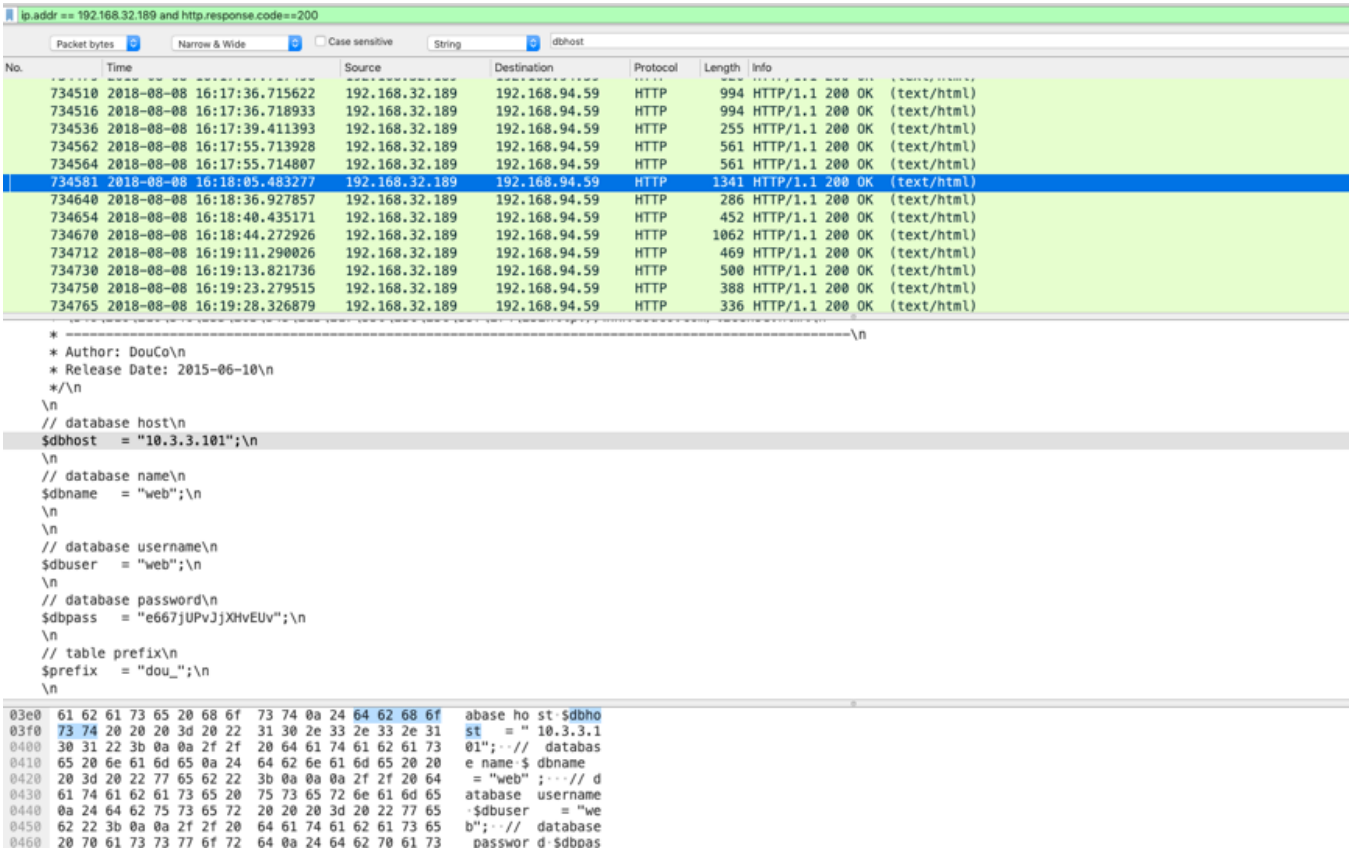


我们直接搜索数据包中的字符串，搜一些跟数据库配置文件相关的关键字，由于是搜的返回包内容，所以我们直接搜host、dbname、password、dbpwd等。搜索了一番，怎么都没找到读取数据库配置文件的数据包，难道没有读数据库配置文件吗？难道筛选逻辑有问题？不禁心生疑问。此时想到之前遇到过的一个情况，流量包里只有返回包，

没有记录到请求包时，就无法通过特定的请求路径筛选出对应的返回数据包。难不成读取数据库配置文件的请求包没有被记录下来吗？带着这个疑问，我又来翻查一遍数据包，这次不再筛选数据包的路径，直接将所有攻击者发起的请求的返回包，并且状态是200的数据包筛选出来，再去查找。



筛选出的数据包有1万多个，使用wireshark的搜索字符串的功能，搜索host发现有大量的噪声包，就直接搜索dbhost字段，正中目标，发现攻击者只读取了一次配置文件，只有这一个数据包。



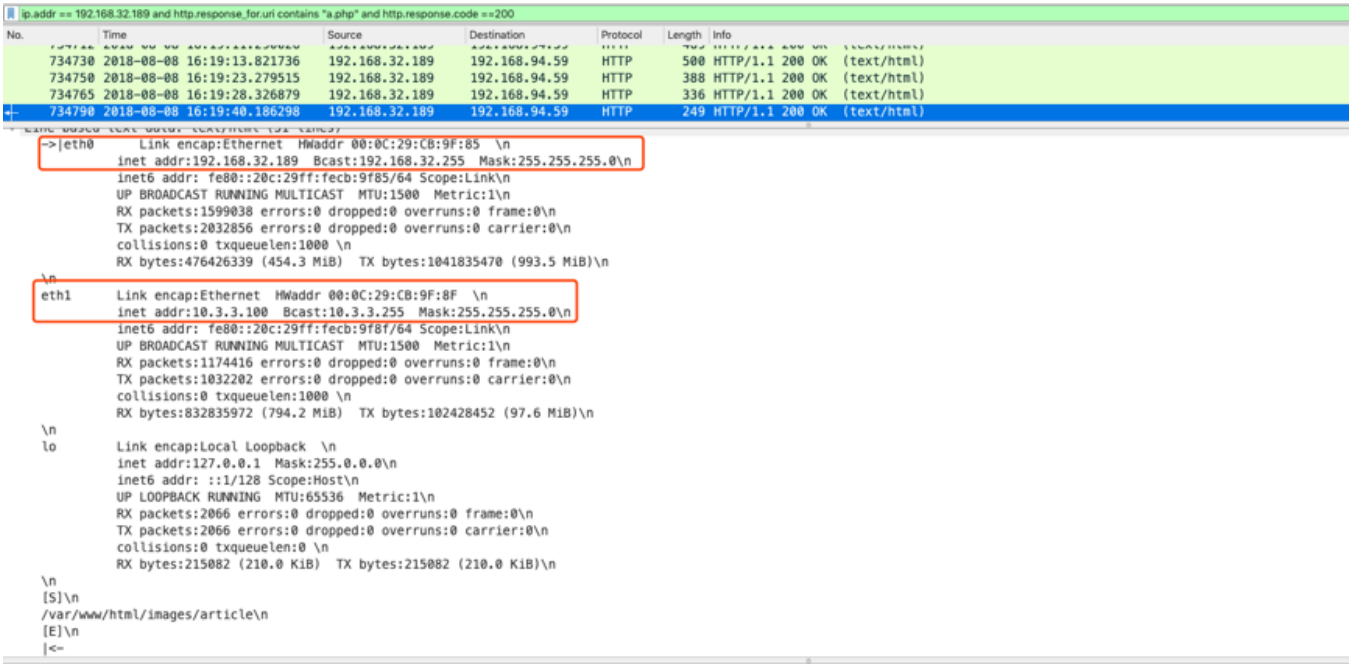
跟踪了这个数据包之后发现，整个数据流只有返回包的内容，没有请求包，似乎就可以印证之前的猜想了，流量包中并没有记录到这个操作的请求包，只记录到了返回包。

第六问 邮箱

攻击者获得的邮箱账号密码是什么？

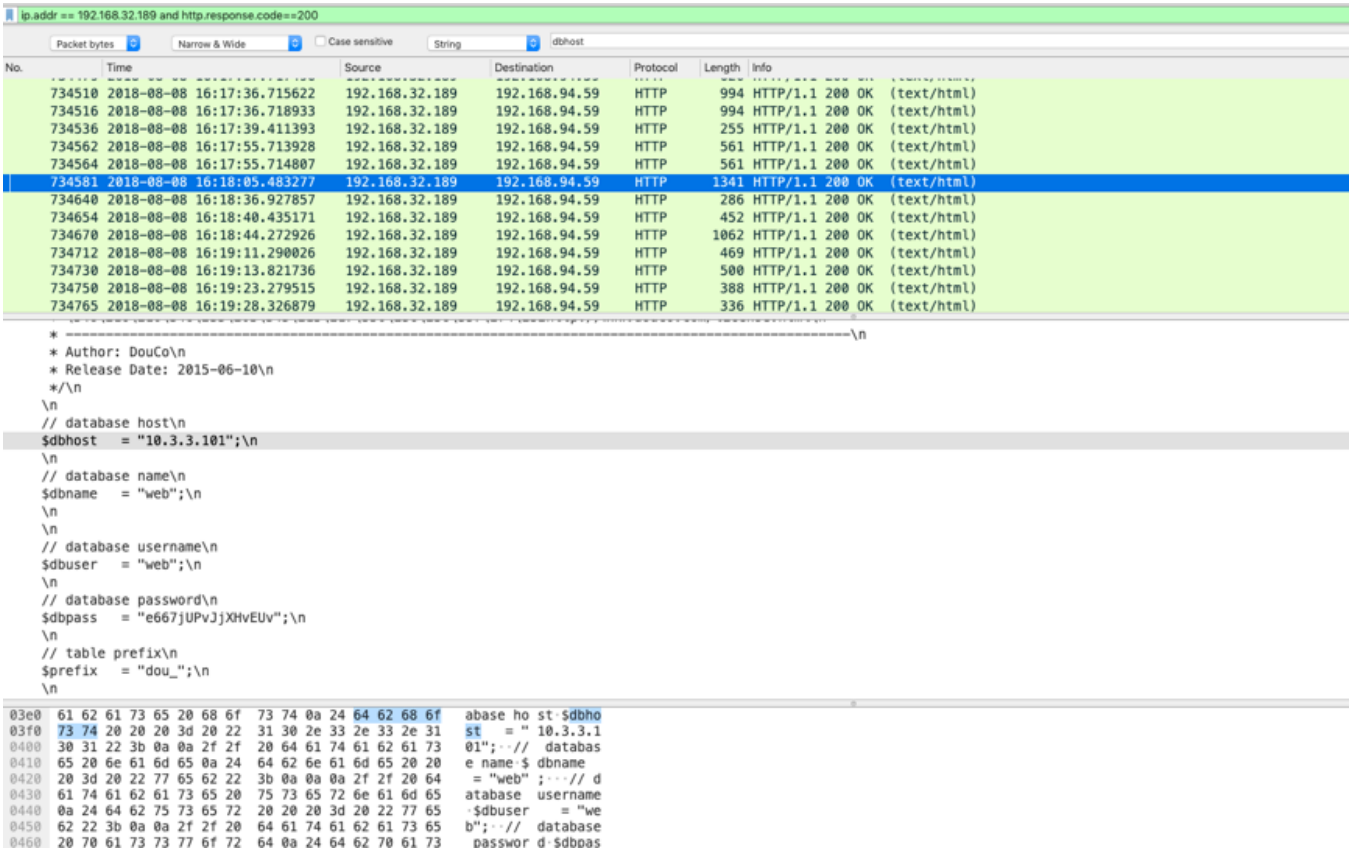
前面攻击者已经拿到了数据库的账号密码，我们猜测攻击者可能通过哪些方式去查询的数据库：第一种，通过webshell连接数据库，执行sql语句查询数据库，这种方式的流量为http流量；第二种，通过数据库连接工具或命令行连接数据库，执行SQL语句查询数据库，这种方式就取决于连接工具本身，mysql一般情况下都是tcp流量。

从上一步获取到的数据库配置信息中看到，数据库的地址是10.3.3.101，我们先来看看被攻击者攻击的网站的网卡信息。



可以看到被攻击的web服务器存在两张网卡，一个对外，一个与数据库通信。继续分析寻找攻击者查询数据库获取邮箱账号信息，查找http流量，没有发现任何基于http流量的查询数据库操作，在记录到的数据包流量中，有mysql流量，直接筛选出来mysql的流量来分析。

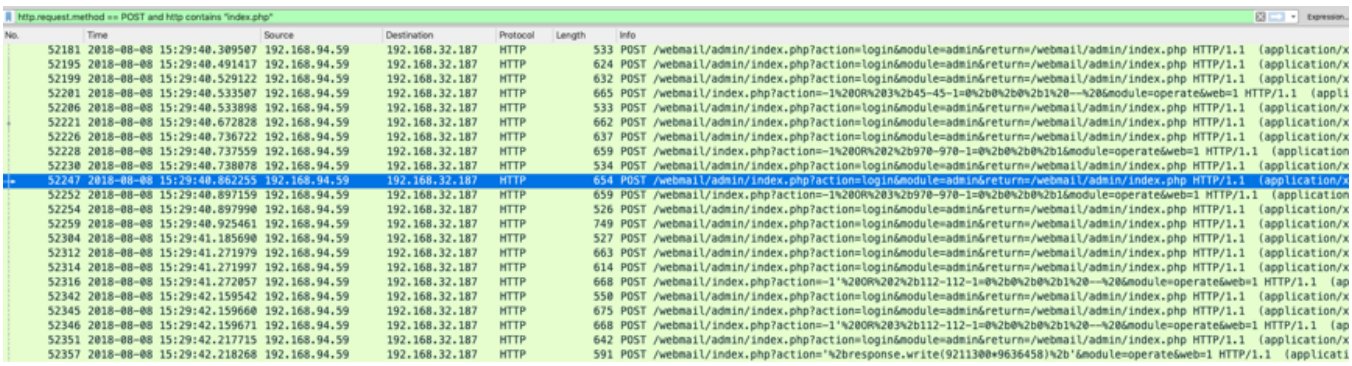
筛选之后发现只有web服务器与数据库的通信流量，猜测可能攻击者使用web服务器做流量代理连接数据库或者直接使用web服务器连接数据库。流量包中的mysql流量共有40多万条，整体逐条筛选肯定会很困难，此时，我们可以回到攻击者发起攻击的行为来分析。攻击者发起读取数据库行为，肯定发生在读取数据库配置文件之后，由此而来，我们可以看看数据包中记录到的读取数据库配置文件的时间，然后寻找mysql流量中对应的时间之后的流量。



可以看到读取配置文件的时间是8月8日16点18分05秒，再回到mysql流量里，按照时间排序流量包，然后来分析。查看了这个时间之后的mysql流量包，发现并没有预期的结果，难道没有通过连接数据库来读数据库里的信息吗？难道是通过注入拿到的信息？现在我们也无法直接确定，我们就来反推一下这个过程，先来找找攻击者登录的邮箱。

与之前一样，先来找找与邮箱登录相关的流量：

```
1 http.request.method == POST and http contains "index.php"
```



存在大量192.168.94.59发出的攻击流量，这个应该就是攻击者的IP了。再来看一下正常的邮箱登录是什么样的返回包，过滤出来与邮箱有关的http数据包后，发现有一个logout退出登录的数据包，在这个数据包中的cookie参数中发现有个login_name参数，那么这个肯定就是正常登录的用户了。

继续看看这个用户名的正常登录是什么样的，会返回来什么样的数据。

```
<!-- Version: 1.6.11 -->GET /webmail/index.php?module=view&action=login&mode=browser&width=1366&height=768 HTTP/1.1
Host: 192.168.32.187
Connection: keep-alive
Accept: */*
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/68.0.3440.84 Safari/
537.36
Referer: http://192.168.32.187/webmail/index.php?module=view&action=login
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: login_domain=test.com; PHPSESSID=csm2kh9f3kjqsft1nl7ft7dk95; SL_G_WPT_TO=zh-CN; SL_GWPT_Show_Hide_tmp=1;
SL_wptGlobTipTmp=1; login_name=wenwenni

HTTP/1.1 200 OK
Server: nginx/1.6.0
Date: Wed, 08 Aug 2018 06:34:40 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 16
Connection: keep-alive
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache

{"success":true}GET /webmail/index.php?module=view&action=index&mode=welcome HTTP/1.1
Host: 192.168.32.187
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/68.0.3440.84 Safari/
537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Referer: http://192.168.32.187/webmail/index.php
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: login_domain=test.com; PHPSESSID=csm2kh9f3kjqsft1nl7ft7dk95; SL_G_WPT_TO=zh-CN; SL_GWPT_Show_Hide_tmp=1;
SL_wptGlobTipTmp=1; login_name=wenwenni
```

这里登录之后返回了一串json格式的字符串，然后又请求welcome模块跳到正常登录后访问的页面。所以登录成功后会返回{"success":true}这样的json字符串。根据这个特征，来找找攻击者登录的邮箱账号是什么。

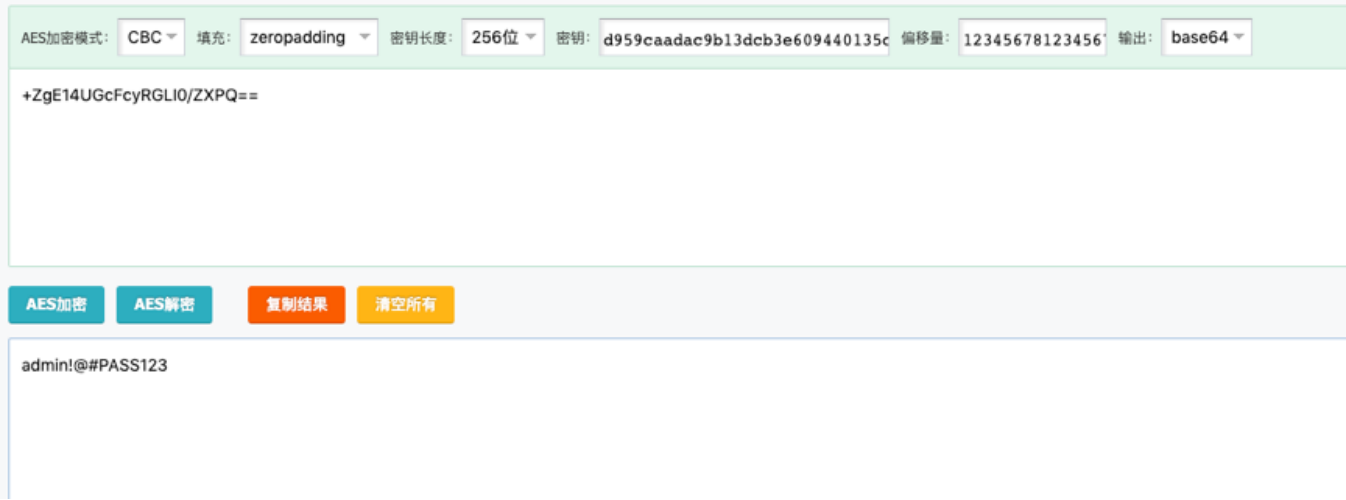
```
1 (http contains "{\"success\":true}" or http.request.method=="POST") and i
```

```
</html>POST /webmail/index.php?module=operate&action=login&web=1 HTTP/1.1
Host: 192.168.32.187
Connection: keep-alive
Content-Length: 111
Cache-Control: max-age=0
Origin: http://192.168.32.187
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/67.0.3396.62
Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Referer: http://192.168.32.187/webmail/index.php?module=view&action=login
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: PHPSESSID=ss2u262l0mkb57coh6hb4hl7j2

username=admin&domain=test.com&password=%2BZgE14UGcFcyRGLI0%2FZXPQ%3D%3D&captcha=OC87&language=zh_CN&enter=trueHTTP/
1.1 200 OK
Server: nginx/1.6.0
Date: Wed, 08 Aug 2018 08:23:12 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 16
Connection: keep-alive
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache

{"success":true}
```

在搜寻的过程中发现还存在大量的爆破的痕迹，不排除攻击者根本没有通过之前拿到的数据库获得密码的可能，而是通过爆破获得的账号。在找到攻击者登录邮箱使用的账号密码后，发现密码是一串加密的字符串，我们还需要解密出密码的明文，才有可能判断出攻击者是否是通过数据库获得的密码。再返回来找一下登录页面的数据包，看看返回的页面中有没有加密方法。

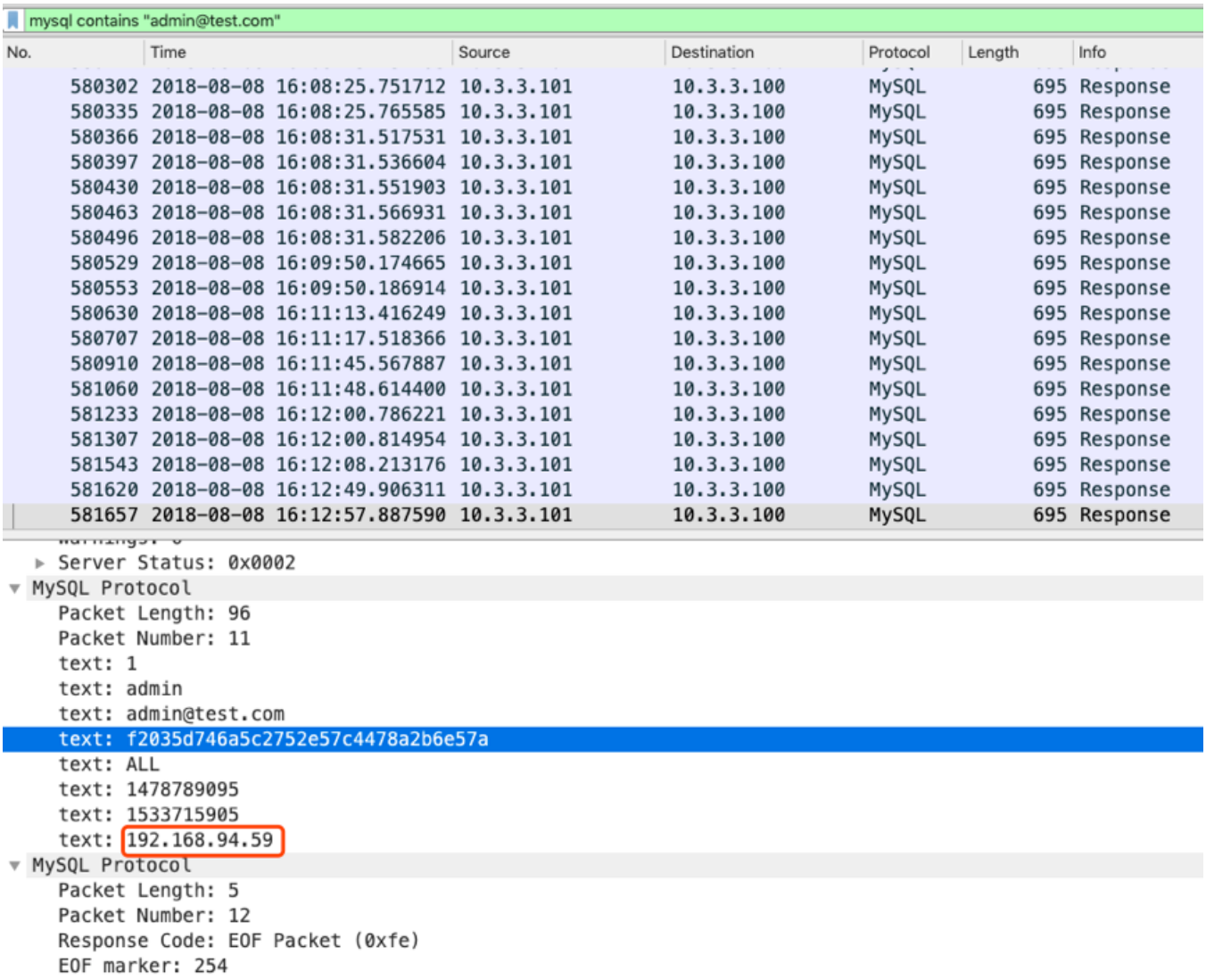
```
28 2018-08-08 14:34:39.386203 192.168.94.131    192.168.32.187   HTTP          676 GET /webmail/index.php?module=view&action=login HTTP/1.1
35 2018-08-08 14:34:39.432072 192.168.32.187    192.168.94.131   HTTP           59 HTTP/1.1 200 OK  (text/html)

    };\n
\n
    var loginCheck = function(form) {\n
        if(form.username.value == "") {\n
            alert("\351\202\256\344\273\266\345\270\220\345\217\267\344\270\215\350\203\275\344\270\272\347\251\272\357\274\201");\n
            form.username.focus();\n
            return false;\n
        }\n
        if(form.password.value == "") {\n
            alert("\347\231\273\345\275\225\345\257\206\347\240\201\344\270\215\350\203\275\344\270\272\347\251\272\357\274\201");\n
            form.password.focus();\n
            return false;\n
        }\n
\t\telse{\n
\t\t\tvar key_hash = CryptoJS.MD5('1234567812345678');\n
\t\t\tvar key = CryptoJS.enc.Utf8.parse(key_hash);\n
\t\t\tvar iv  = CryptoJS.enc.Utf8.parse('1234567812345678');\n
\t\t\tform.password.value = CryptoJS.AES.encrypt(form.password.value, key, { iv: iv,mode:CryptoJS.mode.CBC,padding:CryptoJS.pad.ZeroPadding});\n
\t\t}\n
\t\t\t\n
        if(form.domain.value == '') {\n
            alert("\350\257\267\351\200\211\346\213\251\351\202\256\344\273\266\345\237\237\345\220\215\357\274\201");\n
            return false;\n
        }\n
\t\t\n
\t\tif(jQuery('#ssllgoin').prop('checked') == true){\n
\t\t\tjQuery('#loginForm').prop('action','https://192.168.32.187/webmail/index.php?module=operate&action=login&web=1');\n
```

终于在这里找到了登录的密码加密方法，是AES加密，key是1234567812345678的MD5的hash值，iv偏移量是1234567812345678，后面还说了mode和padding的类型，到这里，我们就可以借助解密工具对密码解密了。



到这里我们再回过头从数据库里看看有没有查询admin@test.com密码的操作。

```
mysql contains "admin@test.com"
```



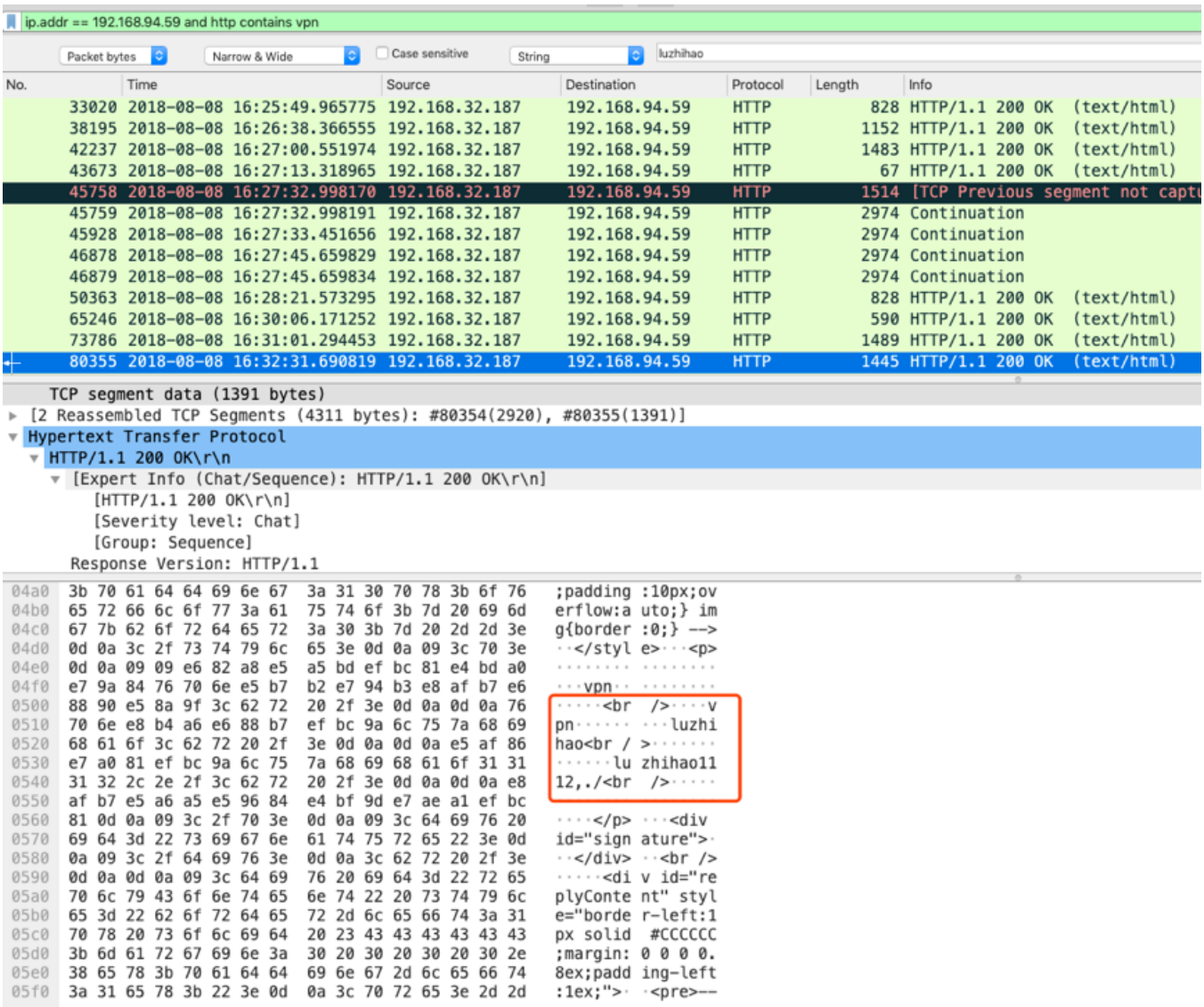
查看mysql里包含admin@test.com的流量，发现时间均在攻击者通过webshell读取数据库配置文件之前。并且数据库中记录的最后登录IP就是攻击者的ip。看来攻击者在读取数据库操作之前就已经拿到了账号密码。

第七问 VPN

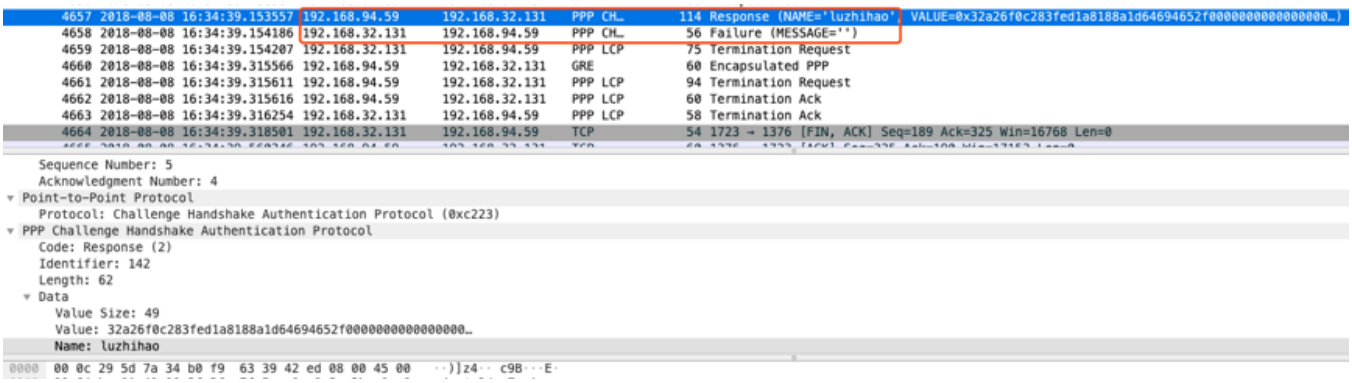
攻击者登录vpn后的ip是多少?

前面攻击者已经拿到了admin权限的邮箱账号，应该所有用户的邮件都可以看到了，我们来找找攻击者有没有从其中拿到vpn的账号密码。

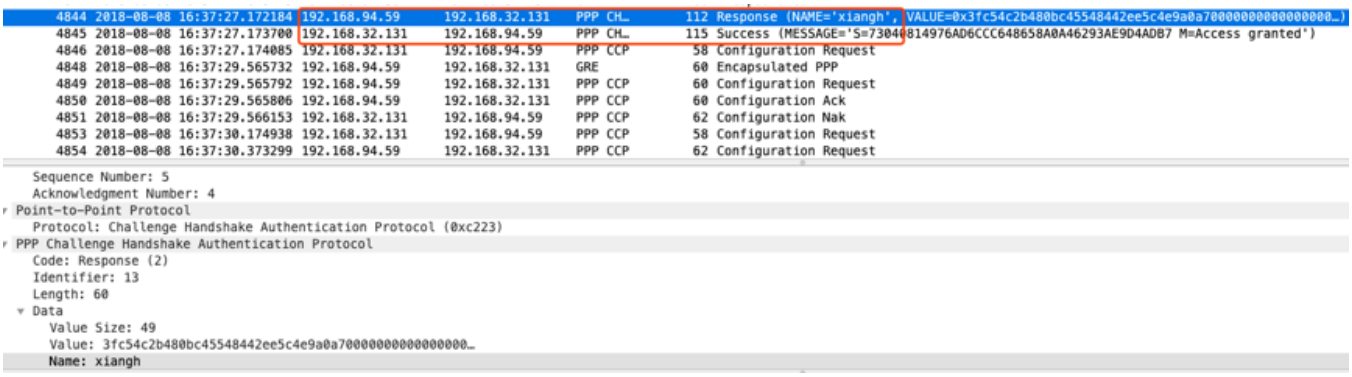
```
ip.addr == 192.168.94.59 and http contains vpn
```



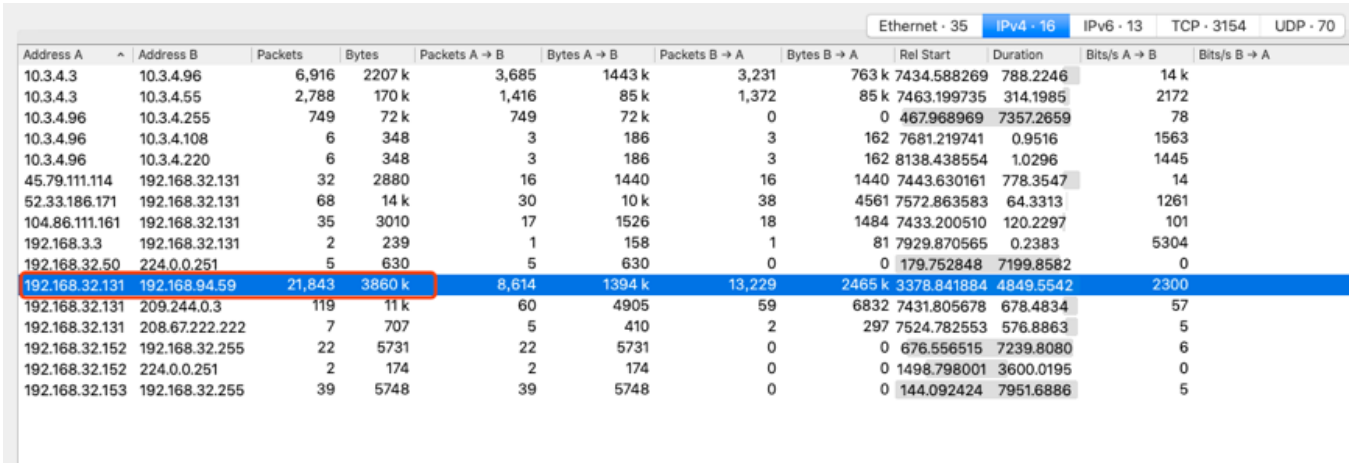
过滤之后发现攻击者所有请求的数据包中包含vpn信息的用户名只有luzhihao一个，再到vpn的流量记录里看看。



这里使用luzhihao的账号进行登录，但并未登录成功。这里我们可以知道192.168.32.131是vpn服务器。

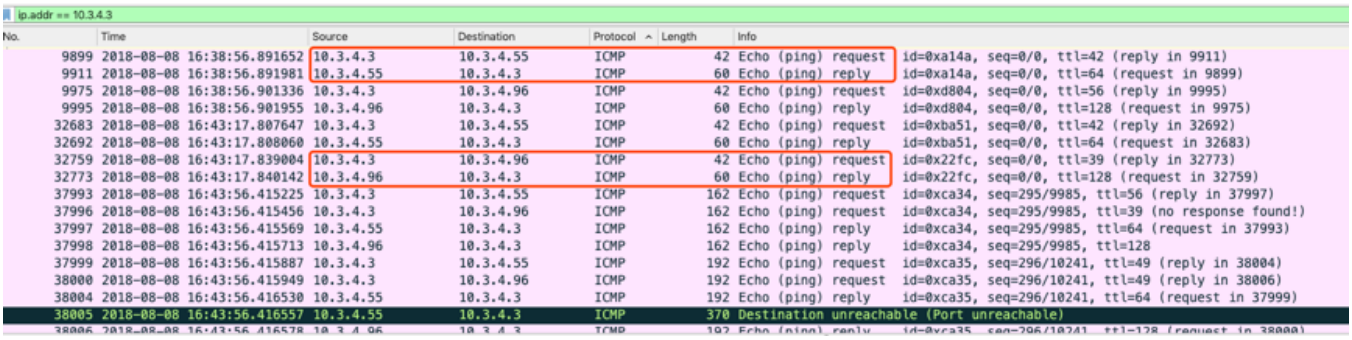


在后面的流量包中发现有使用xingh的账号连接vpn成功的流量包。来使用wireshark自带的流量统计功能看看每个ip之间的流量请求情况。



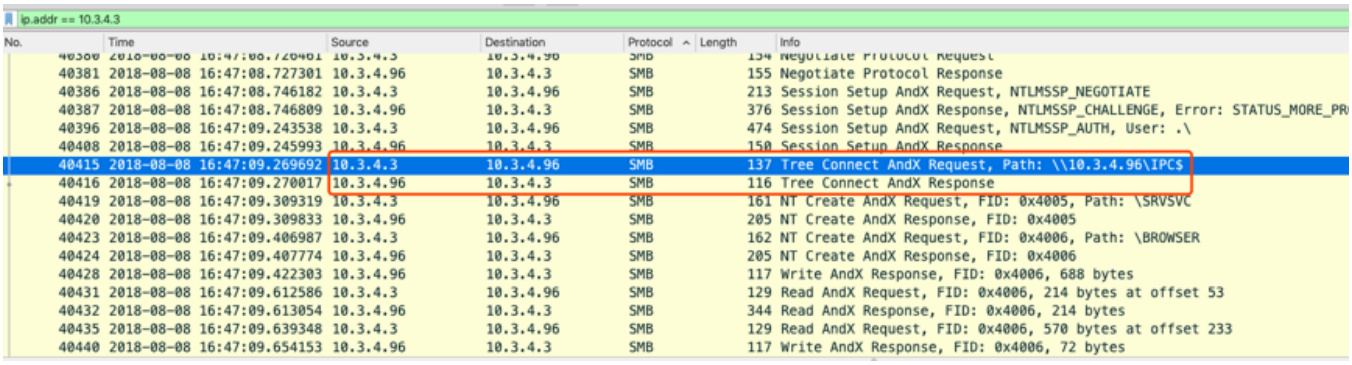
Address A	Address B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
10.3.4.3	10.3.4.96	6,916	2207 k	3,685	1443 k	3,231	763 k	7434.588269	788.2246	14 k	
10.3.4.3	10.3.4.55	2,788	170 k	1,416	85 k	1,372	85 k	7463.199735	314.1985	2172	
10.3.4.96	10.3.4.255	749	72 k	749	72 k	0	0	467.968969	7357.2659	78	
10.3.4.96	10.3.4.108	6	348	3	186	3	162	7681.219741	0.9516	1563	
10.3.4.96	10.3.4.220	6	348	3	186	3	162	8138.438554	1.0296	1445	
45.79.111.114	192.168.32.131	32	2880	16	1440	16	1440	7443.630161	778.3547	14	
52.33.186.171	192.168.32.131	68	14 k	30	10 k	38	4561	7572.863583	64.3313	1261	
104.86.111.161	192.168.32.131	35	3010	17	1526	18	1484	7433.200510	120.2297	101	
192.168.3.3	192.168.32.131	2	239	1	158	1	81	7929.870565	0.2383	5304	
192.168.32.50	224.0.0.251	5	630	5	630	0	0	179.752848	7199.8582	0	
192.168.32.131	192.168.94.59	21,843	3860 k	8,614	1394 k	13,229	2465 k	3378.841884	4849.5542	2300	
192.168.32.131	209.244.0.3	119	11 k	60	4905	59	6832	7431.805678	678.4834	57	
192.168.32.131	208.67.222.222	7	707	5	410	2	297	7524.782553	576.8863	5	
192.168.32.152	192.168.32.255	22	5731	22	5731	0	0	676.556515	7239.8080	6	
192.168.32.152	224.0.0.251	2	174	2	174	0	0	1498.798001	3600.0195	0	
192.168.32.153	192.168.32.255	39	5748	39	5748	0	0	144.092424	7951.6886	5	

从统计出来的流量上看到vpn服务器返回给攻击者的ip大量数据包，其中有一部分是前期攻击者登录的流量包，其余的就是攻击者连接上vpn之后的操作。再看其他的统计数据，10.3.4.3向10.2.4.96发送了大量数据包，那么这两个ip中应该有一个是攻击者连接vpn之后分配的ip，再来看看哪个是攻击者的ip。



No.	Time	Source	Destination	Protocol	Length	Info
9899	2018-08-08 16:38:56.891652	10.3.4.3	10.3.4.55	ICMP	42	Echo (ping) request id=0xa14a, seq=0/0, ttl=42 (reply in 9911)
9911	2018-08-08 16:38:56.891981	10.3.4.55	10.3.4.3	ICMP	60	Echo (ping) reply id=0xa14a, seq=0/0, ttl=64 (request in 9899)
9975	2018-08-08 16:38:56.901336	10.3.4.3	10.3.4.96	ICMP	42	Echo (ping) request id=0xd804, seq=0/0, ttl=56 (reply in 9995)
9995	2018-08-08 16:38:56.901955	10.3.4.96	10.3.4.3	ICMP	60	Echo (ping) reply id=0xd804, seq=0/0, ttl=128 (request in 9975)
32683	2018-08-08 16:43:17.807647	10.3.4.3	10.3.4.55	ICMP	42	Echo (ping) request id=0xba51, seq=0/0, ttl=42 (reply in 32692)
32692	2018-08-08 16:43:17.808060	10.3.4.55	10.3.4.3	ICMP	60	Echo (ping) reply id=0xba51, seq=0/0, ttl=64 (request in 32683)
32759	2018-08-08 16:43:17.839004	10.3.4.3	10.3.4.96	ICMP	42	Echo (ping) request id=0x22fc, seq=0/0, ttl=39 (reply in 32773)
32773	2018-08-08 16:43:17.840142	10.3.4.96	10.3.4.3	ICMP	60	Echo (ping) reply id=0x22fc, seq=0/0, ttl=128 (request in 32759)
37993	2018-08-08 16:43:56.415225	10.3.4.3	10.3.4.55	ICMP	162	Echo (ping) request id=0xca34, seq=295/9985, ttl=56 (reply in 37997)
37996	2018-08-08 16:43:56.415456	10.3.4.3	10.3.4.96	ICMP	162	Echo (ping) request id=0xca34, seq=295/9985, ttl=39 (no response found!)
37997	2018-08-08 16:43:56.415569	10.3.4.55	10.3.4.3	ICMP	162	Echo (ping) reply id=0xca34, seq=295/9985, ttl=64 (request in 37993)
37998	2018-08-08 16:43:56.415713	10.3.4.96	10.3.4.3	ICMP	162	Echo (ping) reply id=0xca34, seq=295/9985, ttl=128
37999	2018-08-08 16:43:56.415887	10.3.4.3	10.3.4.55	ICMP	192	Echo (ping) request id=0xca35, seq=296/10241, ttl=49 (reply in 38004)
38000	2018-08-08 16:43:56.415949	10.3.4.3	10.3.4.96	ICMP	192	Echo (ping) request id=0xca35, seq=296/10241, ttl=49 (reply in 38006)
38004	2018-08-08 16:43:56.416530	10.3.4.55	10.3.4.3	ICMP	192	Echo (ping) reply id=0xca35, seq=296/10241, ttl=64 (request in 37999)
38005	2018-08-08 16:43:56.416557	10.3.4.55	10.3.4.3	ICMP	370	Destination unreachable (Port unreachable)

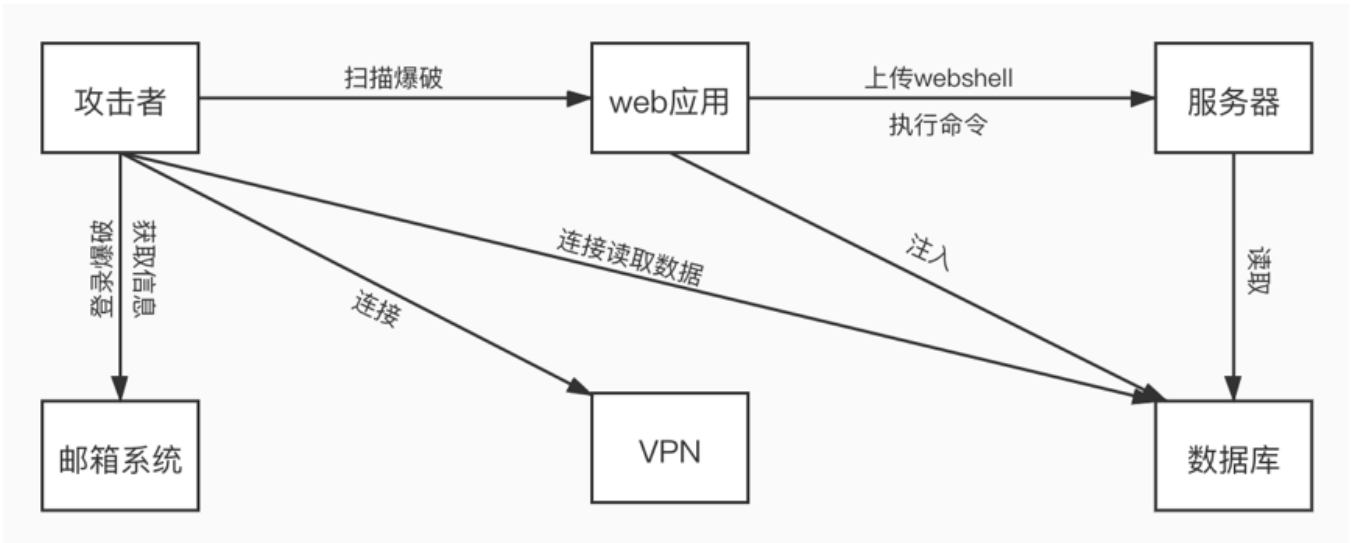
筛选10.3.4.3的流量后，按照协议类型排序，在ICMP类型的流量中，发现10.3.4.3主动向10.3.4.55和10.3.4.96发起了ping请求，通常情况下可以判定10.3.4.3是人为控制的机器。



No.	Time	Source	Destination	Protocol	Length	Info
40381	2018-08-08 16:47:08.727301	10.3.4.96	10.3.4.3	SMB	155	Negotiate Protocol Response
40386	2018-08-08 16:47:08.746182	10.3.4.3	10.3.4.96	SMB	213	Session Setup AndX Request, NTLMSSP_NEGOTIATE
40387	2018-08-08 16:47:08.746809	10.3.4.96	10.3.4.3	SMB	376	Session Setup AndX Response, NTLMSSP_CHALLENGE, Error: STATUS_MORE_PR
40396	2018-08-08 16:47:09.243538	10.3.4.3	10.3.4.96	SMB	474	Session Setup AndX Request, NTLMSSP_AUTH, User: .\
40408	2018-08-08 16:47:09.245993	10.3.4.96	10.3.4.3	SMB	150	Session Setup AndX Response
40415	2018-08-08 16:47:09.269692	10.3.4.3	10.3.4.96	SMB	137	Tree Connect AndX Request, Path: \\10.3.4.96\IPC$
40416	2018-08-08 16:47:09.270017	10.3.4.96	10.3.4.3	SMB	116	Tree Connect AndX Response
40419	2018-08-08 16:47:09.309319	10.3.4.3	10.3.4.96	SMB	161	NT Create AndX Request, FID: 0x4005, Path: \SRVSVC
40420	2018-08-08 16:47:09.309833	10.3.4.96	10.3.4.3	SMB	205	NT Create AndX Response, FID: 0x4005
40423	2018-08-08 16:47:09.406987	10.3.4.3	10.3.4.96	SMB	162	NT Create AndX Request, FID: 0x4006, Path: \BROWSER
40424	2018-08-08 16:47:09.407774	10.3.4.96	10.3.4.3	SMB	205	NT Create AndX Response, FID: 0x4006
40428	2018-08-08 16:47:09.422303	10.3.4.96	10.3.4.3	SMB	117	Write AndX Response, FID: 0x4006, 688 bytes
40431	2018-08-08 16:47:09.612586	10.3.4.3	10.3.4.96	SMB	129	Read AndX Request, FID: 0x4006, 214 bytes at offset 53
40432	2018-08-08 16:47:09.613054	10.3.4.96	10.3.4.3	SMB	344	Read AndX Response, FID: 0x4006, 214 bytes
40435	2018-08-08 16:47:09.639348	10.3.4.3	10.3.4.96	SMB	129	Read AndX Request, FID: 0x4006, 570 bytes at offset 233
40440	2018-08-08 16:47:09.654153	10.3.4.96	10.3.4.3	SMB	117	Write AndX Response, FID: 0x4006, 72 bytes

继续查看，发现10.3.4.3向10.3.4.96主动发起共享请求，10.3.4.96返回回应，这就可以判断10.3.4.3是攻击者控制的机器，所以攻击者接入vpn后的ip是10.3.4.3。

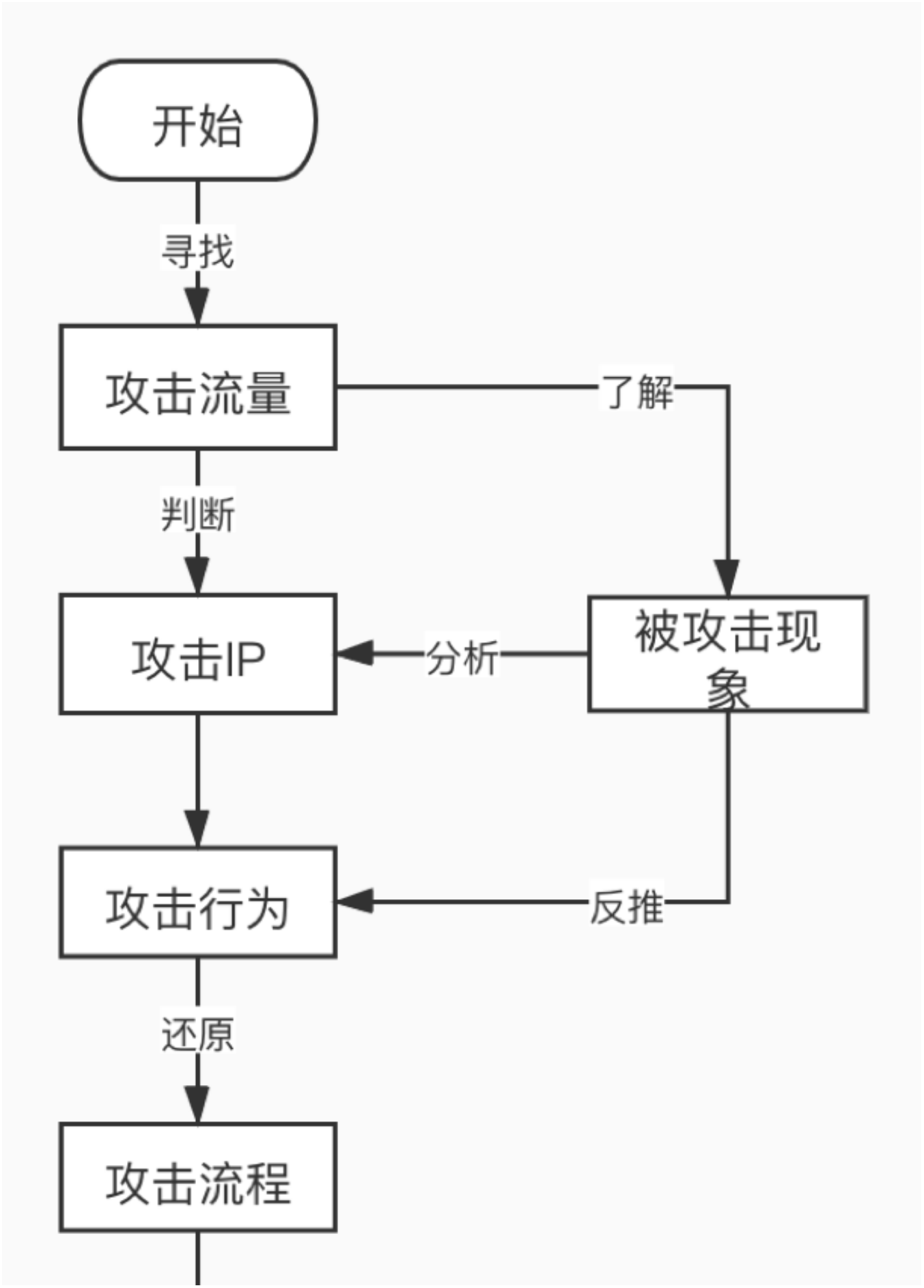
至此，我们整个的分析就已完成。攻击者的整个攻击流程大致如下：

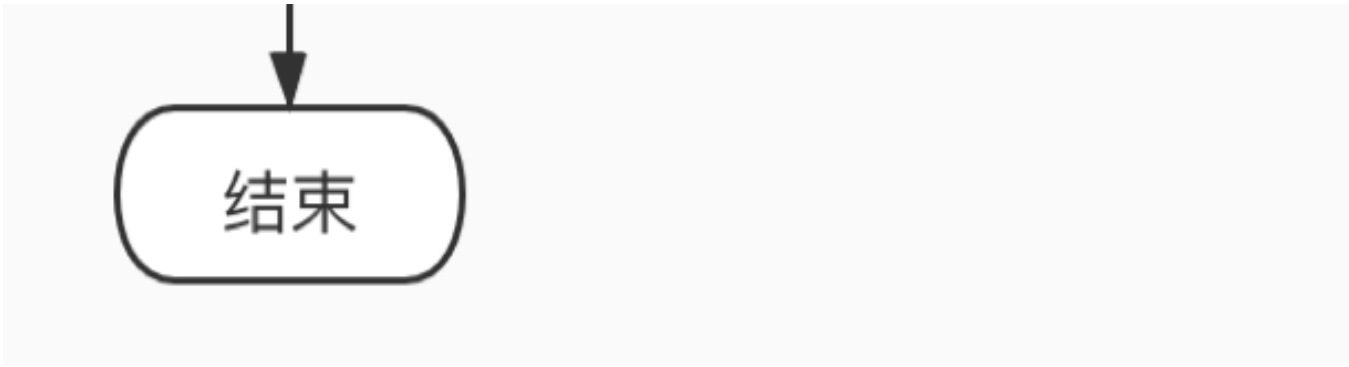


总结

流量分析，防守之道。站在攻击者的角度，思考攻击者的整体流程，他们的最终目的无外乎拿数据、得权限，正向走不通就反向推演，万变不离其宗。

对整个分析过程做一个回顾，大致可以把整个思路汇总成为下面这一个图：



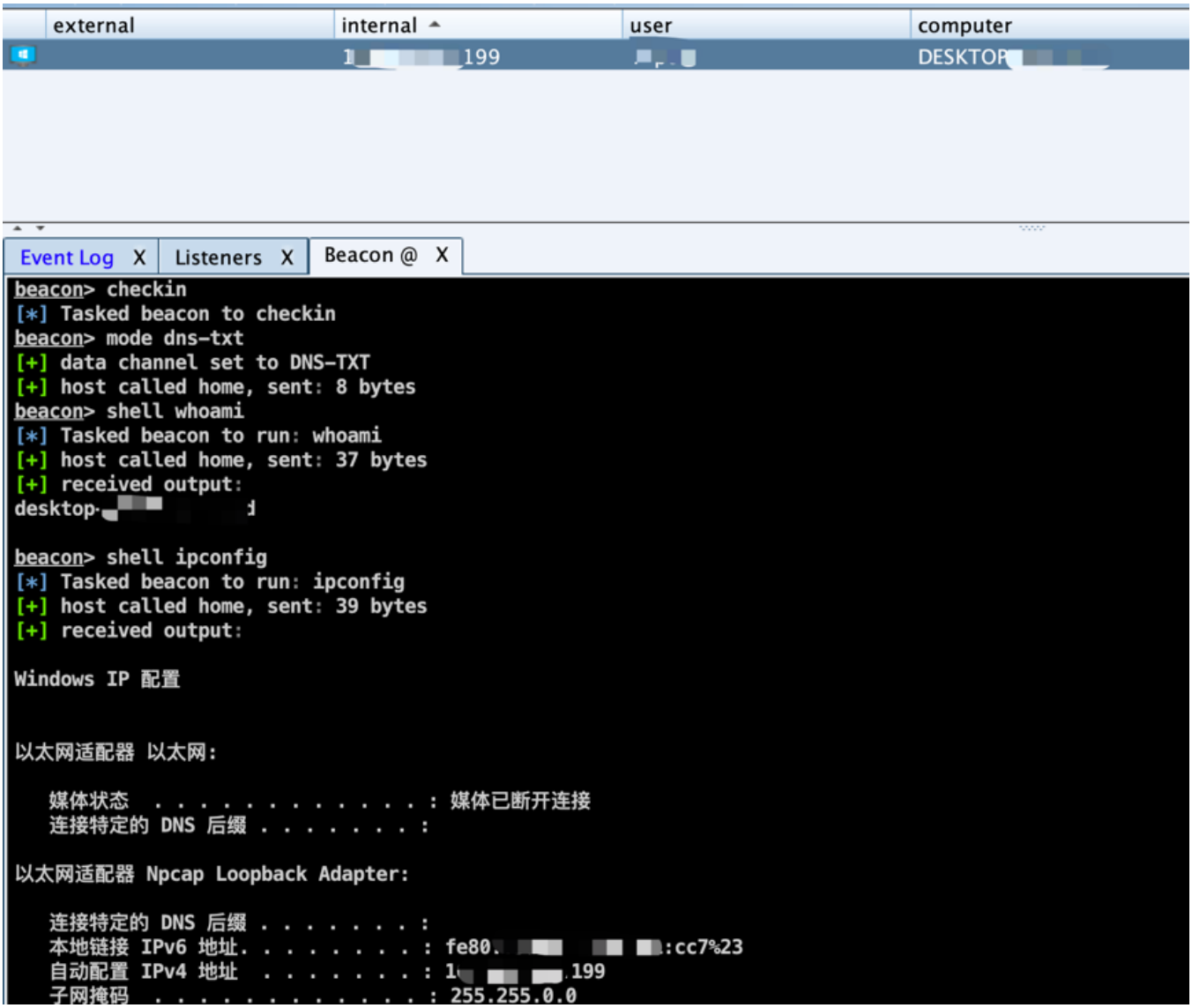


拓展

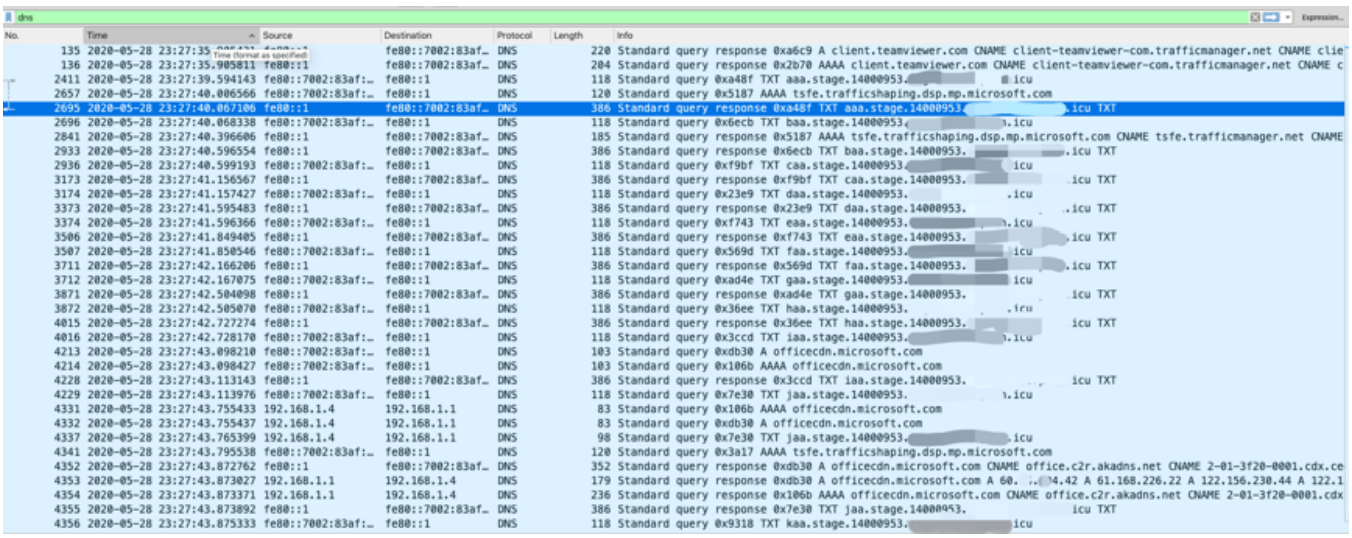
我们继续来说说cobalt strike的几种不同类型的beacon模式的远控流量所存在的一些特征。

DNS隧道

配置好服务器，做好域名解析后，在cobalt strike上创建一个监听器，并使用cs生成一个木马文件（这里使用的是cobalt strike 3.13版本）。将木马放在windows下执行，同时我们使用wireshark来抓取流量包，查看整个执行远控的过程中的通信情况。



执行木马主机上线后，这里我执行了ipconfig、net user、dir命令，来看一下cobaltstrike远程控制主机执行这些命令后，teamserver与远控主机之间是如何进行通信的。



我们使用的是dns隧道实现的远程控制，在流量中会出现大量的dns数据包，从这些dns数据包的解析地址来看，发现异常的域名地址还是相对比较容易。全局搜索执行的命令，无论是搜字符串还是hex值，都没有任何信息。相对从数据包内容上来说，dns隧道还是比较隐蔽，很多攻击者在使用dns隧道来进行远控时，都会使用隐蔽性更强的域名，会使用一些与常用域名非常相似的域名，如a1iyun.com这样的，不容易被认出来，这时候就需要借助一些第三方的威胁情报库去辅助判断。