

BSidesSF 2019 部分writeup

原创

TuudOp 于 2019-03-06 19:26:05 发布 2092 收藏

分类专栏: [ctf](#) 文章标签: [BSidesSF 2019](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: https://blog.csdn.net/qq_39850969/article/details/88248406

版权



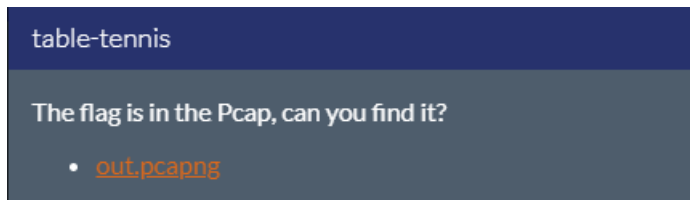
[ctf](#) 专栏收录该内容

7 篇文章 1 订阅

订阅专栏

forensics:

table-tennis



给了一个流量包, 使用wireshark打开, 通过查找字符串和跟踪TCP数据流并没有发现什么, 以我的水平, 基本上到这里就结束了。

后面查看了其他人写的writeup, 跟着复现了一下。

这个流量包中除了大量的加密的TLS和TCP数据流, 还有一些ICMP数据包, 发现里面有HTML数据。

out.pcapng

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

icmp

No.	Time	Source	Destination	Protocol	Length	Info
1393	5.0212704...	216.58.195.68	192.168.10.212	ICMP	98	Echo (ping) reply id=0x1350, seq=1/256, ttl=55 (request in 1390)
1375	4.9378033...	172.217.5.100	192.168.10.212	ICMP	98	Echo (ping) reply id=0x134e, seq=1/256, ttl=55 (request in 1372)
1361	4.8721926...	172.217.5.100	192.168.10.212	ICMP	98	Echo (ping) reply id=0x134c, seq=1/256, ttl=55 (request in 1358)
1333	4.7869271...	172.217.5.100	192.168.10.212	ICMP	98	Echo (ping) reply id=0x1340, seq=1/256, ttl=55 (request in 1326)
1311	4.7010758...	172.217.6.68	192.168.10.212	ICMP	98	Echo (ping) reply id=0x133e, seq=1/256, ttl=55 (request in 1306)
1282	4.6323198...	172.217.0.36	192.168.10.212	ICMP	98	Echo (ping) reply id=0x133c, seq=1/256, ttl=55 (request in 1277)
1259	4.5443314...	172.217.5.100	192.168.10.212	ICMP	98	Echo (ping) reply id=0x133a, seq=1/256, ttl=55 (request in 1255)
1219	4.4413748...	172.217.0.36	192.168.10.212	ICMP	98	Echo (ping) reply id=0x1338, seq=1/256, ttl=55 (request in 1214)
1185	4.3368259...	172.217.5.100	192.168.10.212	ICMP	98	Echo (ping) reply id=0x1336, seq=1/256, ttl=55 (request in 1180)
1140	4.2339898...	172.217.5.100	192.168.10.212	ICMP	98	Echo (ping) reply id=0x1334, seq=1/256, ttl=55 (request in 1135)
1112	4.1460119...	172.217.5.100	192.168.10.212	ICMP	98	Echo (ping) reply id=0x1332, seq=1/256, ttl=55 (request in 1109)
1071	4.0431413...	172.217.6.36	192.168.10.212	ICMP	98	Echo (ping) reply id=0x1330, seq=1/256, ttl=55 (request in 1066)
1031	3.9577349...	172.217.5.100	192.168.10.212	ICMP	98	Echo (ping) reply id=0x132e, seq=1/256, ttl=55 (request in 1028)
1003	3.8926075...	172.217.5.100	192.168.10.212	ICMP	98	Echo (ping) reply id=0x132c, seq=1/256, ttl=55 (request in 1000)
970	3.7906337...	172.217.0.36	192.168.10.212	ICMP	98	Echo (ping) reply id=0x132a, seq=1/256, ttl=55 (request in 954)
927	3.7044915...	172.217.5.100	192.168.10.212	ICMP	98	Echo (ping) reply id=0x1328, seq=1/256, ttl=55 (request in 922)
889	3.6004801...	172.217.5.100	192.168.10.212	ICMP	98	Echo (ping) reply id=0x1326, seq=1/256, ttl=55 (request in 871)
844	3.4975648...	172.217.6.36	192.168.10.212	ICMP	98	Echo (ping) reply id=0x1324, seq=1/256, ttl=55 (request in 839)
812	3.4117503...	172.217.5.100	192.168.10.212	ICMP	98	Echo (ping) reply id=0x1322, seq=1/256, ttl=55 (request in 810)

..0. = More fragments: Not set
 ...0 0000 0000 0000 = Fragment offset: 0
 Time to live: 55
 Protocol: ICMP (1)
 Header checksum: 0x0530 [validation disabled]
 [Header checksum status: Unverified]
 Source: 172.217.6.36
 Destination: 192.168.10.212

Internet Control Message Protocol
 Type: 0 (Echo (ping) reply)

```

0000 e4 b3 18 4b f0 c3 b0 fa eb 62 ee ff 08 00 45 00 ...K...-b...E-
0010 00 54 00 00 00 00 37 01 05 30 ac d9 06 24 c0 a8 ..T...7..0...$..
0020 0a d4 00 00 f6 9f 13 24 00 01 d0 dc 74 5c 00 00 ...$...L...
0030 00 00 5e ac 02 00 00 00 00 00 3c 68 65 61 64 3e ..^.....<head>
0040 0a 09 3c 68 65 61 64 3e 0a 09 3c 68 65 61 64 3e ..<head> ..<head>
0050 0a 09 3c 68 65 61 64 3e 0a 09 3c 68 65 61 64 3e ..<head> ..<head>
0060 0a 09
  
```

https://blog.csdn.net/qq_39850969

这里要将这些HTML数据提取出来，照着大佬使用python scapy模块写的脚本提取数据，我也依样画瓢谢了一下并且理解了脚本的意思，脚本比较简单：

```

from scapy.all import *

packets = rdpcap('out.pcapng') #rdpcap()读取pcapng文件

for packet in packets: #遍历每一个数据包
    if packet.haslayer(ICMP): #haslayer()判断数据包的类型，此处为ICMP
        if packet[ICMP].type == 0: #每一个ICMP的type值为0的包
            print packet[ICMP].load[-8:], #打印每个数据包的最后8位，因为前面数据是重复的
  
```

提取出来的数据：

```
root@kali:~/桌面# python pcap.py
<html>
  <head>
    <title> I <3 Cor gi </tit le>
    <s cript>
document.write(at ob('Q1RG e0p1c3RB UzBuZ0Fi MHV0UDFu Z1Awbmd9 "));
    < /script>

  </hea d>

  <bo dy>

    < h1> Woof !! </h1>

  </bod y>

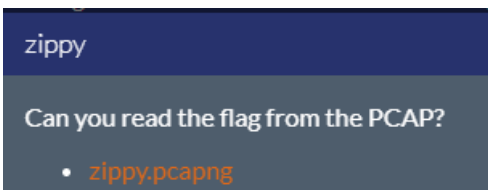
</ht
```

https://blog.csdn.net/qq_39850969

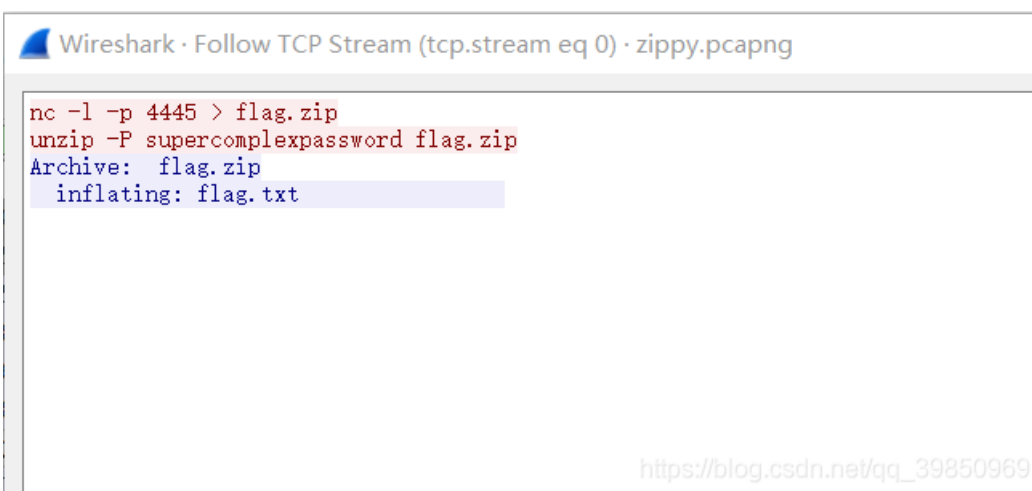
将这段base64加密的数据空格合并 并解密:

```
root@kali:~/桌面# echo Q1RG e0p1c3RB UzBuZ0Fi MHV0UDFu Z1Awbmd9 | base64 -d
CTF{JustAS0ngAb0utP1ngP0ng}root@kali:~/桌面#
```

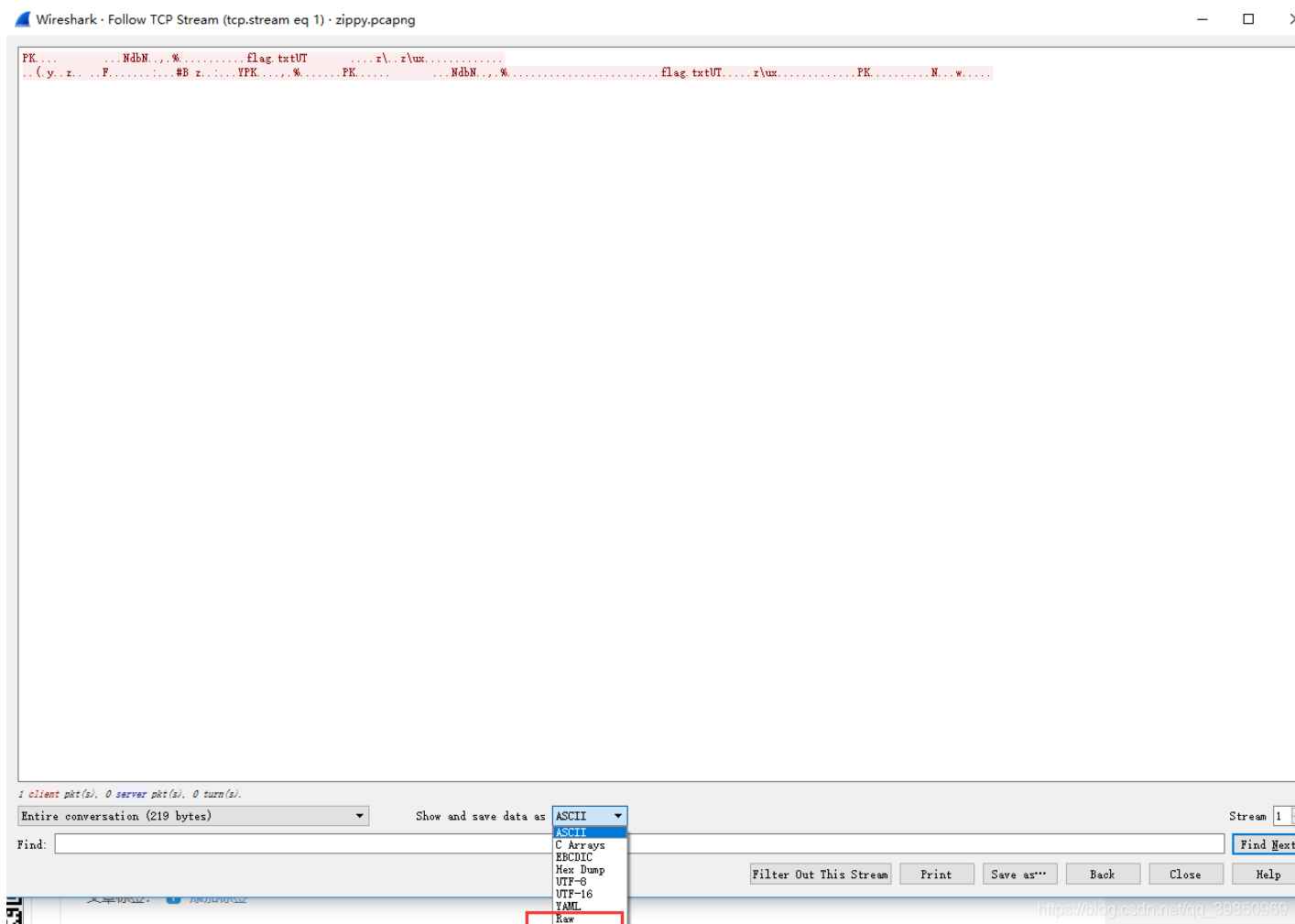
Zippy



也是给了一个pcapng文件，打开之后，跟踪一个TCP数据流



应该是正在发送一个加了密的zip文件，找到这个加密文件的数据流



然后以原始数据保存为zip文件

使用密码解压就可以得到flag

```
root@kali:~/桌面# unzip -P supercomplexpassword flag.zip
Archive:  flag.zip
  inflating: flag.txt
```

```
root@kali:~/桌面# cat flag.txt
CTF{this_flag_is_your_flag}
```

SlashSlash

给了一个flag.zip文件，解压得到flag.aes128cbc文件，同时还有一个字符串，应该是密码

```
root@kali:~/桌面# unzip flag.zip
Archive:  flag.zip
SevenPinLock0123456
  extracting: flag.aes128cbc
```

使用 openssl 即可解密

```
openssl enc -d -aes-128-cbc -pass pass:SevenPinLock0123456 -in flag.aes128cbc
```

```
root@kali:~/桌面# openssl enc -d -aes-128-cbc -pass pass:SevenPinLock0123456 -in flag.aes128cbc  
CTF{always_add_comments}  
root@kali:~/桌面# openssl enc -d -aes-128-cbc -in flag.aes128cbc  
enter aes-128-cbc decryption password:  
CTF{always_add_comments}
```