《量子加密》writeup by @xmcp

本题有跑脚本的环节，下文所述的运行时间在一台 i7-10510U (25W) 笔记本电脑上测出。

解题过程分为三个部分：1) 解密 ZIP，2) 分析流量，3) 破解口令。

解密 ZIP

首先打开压缩包，发现有密码保护，看到注释：

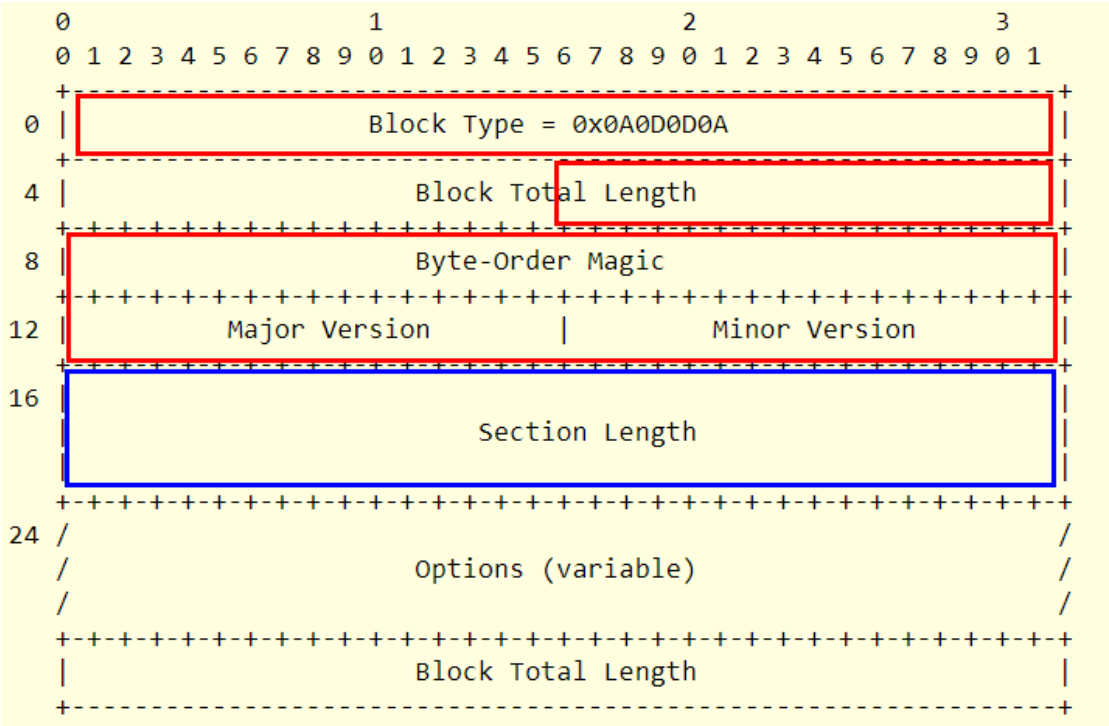
> Password is longer than 16 bytes, and includes at least one number and one uppercase letter.
> Feel free to crack it if you can, probably with your high-performance quantum computer :)

说明难以暴力破解。

压缩包里面有“capture.pcapng”和“hint_for_capture.txt”，压缩算法都是 Store，这很奇怪，因为 pcapng 并非十分紧密的格式，压缩软件没有理由不压缩它。

因此我们考虑 ZIP 明文攻击，只需要知道 pcapng 中至少 12 个字节（其中至少 8 个字节连续）的内容即可破解。

上网搜索 pcapng 的文件头格式：



红色部分是固定的：Block Type 始终为 0A0D0D0A；Block Total Length 是小端存储的 Header 长度，显然不会超过 64KB，所以高两位都是 00；Byte-Order Magic 在小端机器上始终为 4D3C2B1A；Major Version 目前只有 0100；Minor Version 目前只有 0000。

这样可以知道文件中的 4+10 字节内容，满足明文攻击的要求。

其实蓝色部分也是可以猜出来的：Section Length 是可选字段，大多数软件（比如 WireShark）在保存 pcapng 时会写入-1（即 8 个字节的 FF）。

因此，把蓝色部分也算上的话，我们知道文件中的 4+18 字节内容，对明文攻击绰绰有余了。为了严谨起见，我们只使用十分确定的红色部分进行明文攻击。

注意到 AZPR 不支持明文分段的 ZIP 明文攻击，我们使用 bkcrack。

在压缩包中提取出密文 capture.pcapng.enc（888832 字节）和 hint_for_capture.txt.enc（48 字节），把已知的 10 字节明文保存为 capture.pcapng.plain。运行下面的命令进行攻击：

```
root@xmcp-kali-vm:~/zipp# time bkcrack/src/bkcrack -c capture.pcapng.enc -p capture.pcapng.plain -o 6 -x 0 0a0d0d0a
Generated 4194304 Z values.
[17:16:02] Z reduction using 2 bytes of known plaintext
100.0 % (2 / 2)
1827840 values remaining.
[17:16:02] Attack on 1827840 Z values at index 13
79.0 % (1444343 / 1827840)
[17:45:39] Keys
e33a580c c0c96a81 1246d892

real    29m37.637s
user    231m23.620s
sys     1m10.138s
```

运行 29 分钟后得到秘钥。运行下面的命令解密文件：

```
root@xmcp-kali-vm:~/zipp# bkcrack/src/bkcrack -c capture.pcapng.enc -k e33a580c c0c96a81 1246d892 -d capture.pcapng
Wrote deciphered text.
root@xmcp-kali-vm:~/zipp# bkcrack/src/bkcrack -c hint_for_capture.txt.enc -k e33a580c c0c96a81 1246d892 -d hint_for_capture.txt
Wrote deciphered text.
```

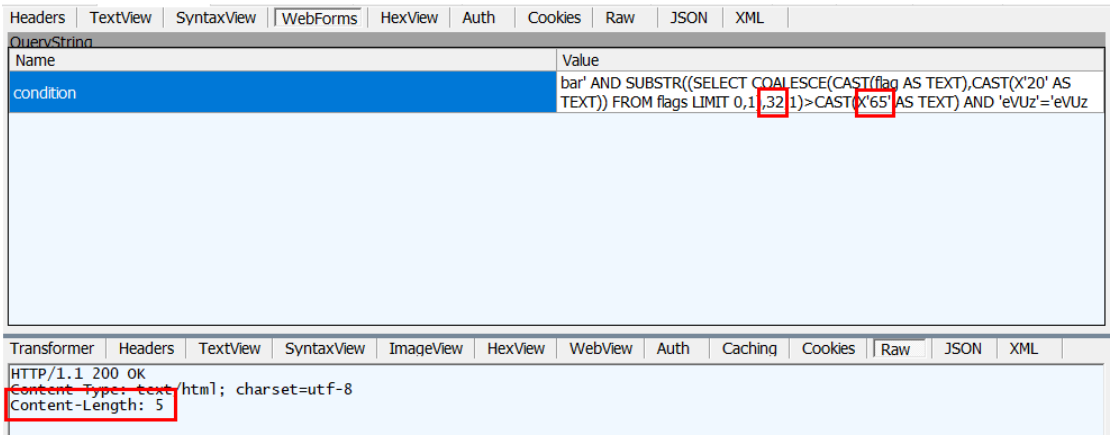
分析流量

打开 pcap 看到上千个 HTTP 流量，用 Fiddler 查看发现全都是对 192.168.142.138:8080 的请求，User-Agent 为 sqlmap/1.3.5.105#dev，可以判断是 sqlmap 自动对服务器进行 SQL 注入的过程。

#	Result	Prot...	Host	URL
2	200	HTTP	192.168.142.138...	/?condition=bar
3	200	HTTP	192.168.142.138...	/?condition=bar&ibLV=5634%20AND%201%3D1%20UNIO...
4	200	HTTP	192.168.142.138...	/?condition=bar
5	200	HTTP	192.168.142.138...	/?condition=bar
6	200	HTTP	192.168.142.138...	/?condition=6924
7	200	HTTP	192.168.142.138...	/?condition=bar%27%28%28%28.%22..%2C.
8	200	HTTP	192.168.142.138...	/?condition=bar%27bWQlWv%3C%27%22%3EMOLSSx
9	200	HTTP	192.168.142.138...	/?condition=bar%27%29%20AND%206689%3D8081%20...
10	200	HTTP	192.168.142.138...	/?condition=bar%27%29%20AND%208287%3D8287%20...
11	200	HTTP	192.168.142.138...	/?condition=bar%27%20AND%206598%3D1238%20AND...
12	200	HTTP	192.168.142.138...	/?condition=bar%27%20AND%208287%3D8287%20AND...
13	200	HTTP	192.168.142.138...	/?condition=bar%27%20AND%203152%3D6834%20AND...
14	200	HTTP	192.168.142.138...	/?condition=bar%27%20ORDER%20BY%201--%20WNlD
15	200	HTTP	192.168.142.138...	/?condition=bar%27%20ORDER%20BY%203077--%20mZ...
16	200	HTTP	192.168.142.138...	/?condition=bar%27%20ORDER%20BY%2010--%20pRYb
17	200	HTTP	192.168.142.138...	/?condition=bar%27%20ORDER%20BY%206--%20tNPl
18	200	HTTP	192.168.142.138...	/?condition=bar%27%20ORDER%20BY%204--%20GAwa
19	200	HTTP	192.168.142.138...	/?condition=bar%27%20ORDER%20BY%203--%20StvY
20	200	HTTP	192.168.142.138...	/?condition=bar%27%20ORDER%20BY%202--%20Ptkz

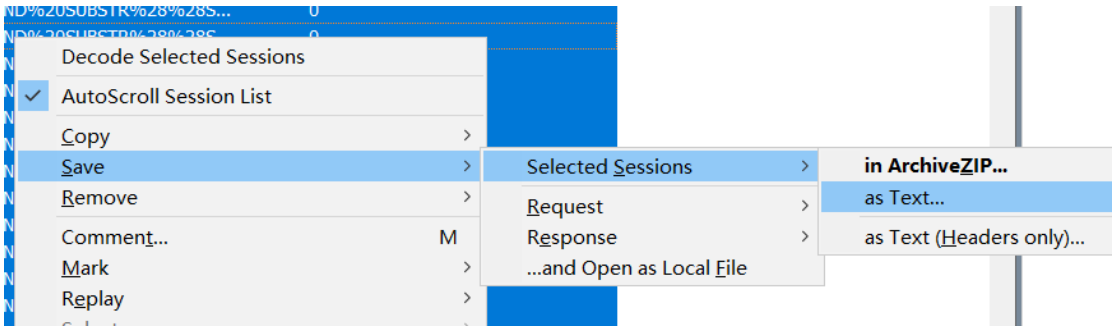
具体观察请求参数和响应，发现 sqlmap 在尝试盲注 flags 表的 flag 字段，请求使用了 HEAD 方法（对应 sqlmap 的--null-connection 参数），而且请求顺序是乱的（对应 sqlmap 的--threads 参数）。

根据前几个 GET 的响应可以判断出响应头 Content-Length 为 5 表示“error”，为 2 表示“ok”。



例如上述请求在判断 flag 的第 32 位字符是否大于 0x65，返回 5 表明结果是否定的。

将所有 HTTP 请求导出成文本文件：

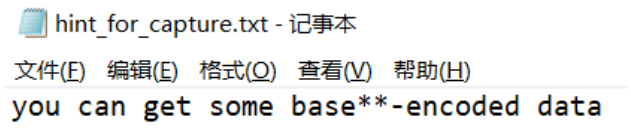


然后编写脚本获得 flag 内容：

analyze_pcap.py

```python
import re
import urllib.parse
inj_re=re.compile(r"bar' AND SUBSTR\(\(SELECT COALESCE\(CAST\(flag AS TEXT\),CAST\(X'20' AS TEXT\)\) FROM flags LIMIT 0,1\),(\d+),1\)>CAST\(X'([0-9a-f]+)' AS TEXT\) AND 'eVUz'='eVUz")

content=[666 for _ in range(200)]

reqs=open('sqlmap_full.txt').read().split(
'---------------------------------------------------------------')

for req in reqs:
    req=urllib.parse.unquote(req)
    match=inj_re.search(req)
    if match and 'Content-Length: 5' in req:
        ind,hextxt=match.groups()
        txt=eval('0x'+hextxt)
        old=content[int(ind)-1]
        if txt<old:
            content[int(ind)-1]=txt

print(''.join([chr(x) for x in content if x!=666]))
```

根据 hint_for_capture.txt 的提示，得到的内容经过了 base85 编码。



将脚本输出 base85 解码得到：

flag is md5("Sq1it3"+压缩包密码)

Hint: 密码是一个身份证号，且出生年份恰有两个质因子

破解口令

我们需要破解压缩包口令，考虑 John the Ripper。首先用 zip2john 转换一下：

```
C:\Users\xmcp\Desktop\zipp>john-1.9.0-jumbo-1-win64\run\zip2john.exe -a hint_for_capture.txt quantum_encryption.zip > jo
hn.txt
Using file hint_for_capture.txt as an 'ASCII' quick check file
ver 1.0 quantum_encryption.zip/capture.pcapng PKZIP Encr: cmplen=886328, decmplen=886316, crc=2C1486C4
ver 1.0 quantum_encryption.zip/hint_for_capture.txt PKZIP Encr: cmplen=48, decmplen=36, crc=E66A9C5
NOTE: It is assumed that all files in each archive have the same password.
If that is not the case, the hash may be uncrackable. To avoid this, use
option -o to pick a file at a time.
```

其中 hint_for_capture.txt 是压缩包中的纯 ASCII 文件而且比较小，用来加速破解过程。

已经知道压缩包密码是合法的身份证号。由于压缩包注释提到密码包括大写字母，可以推出身份证尾号为 X。通过分解质因数得到可行的出生年份列表：

```
In [6]: [x for x in range(2020,1920,-1) if len(list(primefactors(x)))==2]
Out[6]:
[2019,
 2018,
 2012,
 2009,
 2008,
 2007,
 2005,
 2000,
 1996,
 1994,
 1991,
 1985,
 1984,
 1983,
 1982,
 1981,
 1977,
 1975,
 1971,
 1969,
 1967,
 1966,
 1964,
 1963,
 1961,
 1959,
 1957,
 1954,
 1952,
```

我们分析一下身份证的格式：

- 前 6 位是区号，上网搜一下可以得到区号列表，全国的合法区号不超过 4000 个。
- 然后 4 位是出生年份，我们从 2020 往下枚举。
- 然后 4 位是出生月日，不超过 366 个。
- 然后 3 位是当天的序号，只需要枚举两位，最后一位可以利用校验码唯一确定。
- 最后 1 位是校验码，固定为 X。

这样一来，密码的值域降低到了 10^{10} 量级，可以进行破解了。

首先我们在网上搞一份身份证区号列表，存为 id_prefix.txt，然后用 C++写一个枚举合法身份证号的代码（编译选项-Ofast -march=native，不用 Python 因为太慢了）。

gen_id.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;

//        0     6 8 101214
char s[32]="12345619001231002X\n";
int weight[32]={7,9,10,5,8,4,2,1,6,3,7,9,10,5,8,4,2};
int remain[16]={0,5,-1,4,9,3,8,2,7,1,6};
int days[16]={-1,31,29,31,30,31,30,31,31,30,31,30,31};

bool calc_lastbit() {
    int tot=9;
    for(int i=0;i<16;i++)
        tot+=weight[i]*(s[i]-'0');
    int rem=remain[tot%11];
    if(rem>=0) {
        s[16]='0'+rem;
        return true;
    } else
        return false;
}

void insert_int(int pos,int num) {
    s[pos]='0'+num/10;
    s[pos+1]='0'+num%10;
}

int main(int argv,char **argc) {
    FILE *fin=fopen("id_prefix.txt","r");
    FILE *fout=fopen("id_list.txt","w");
    s[7]=argc[1][1];
    s[8]=argc[1][2];
    s[9]=argc[1][3];
    while(true) {
        if(fscanf(fin,"%s",s)<=0) return 0;
        s[6]=argc[1][0];
        for(int m=1;m<=12;m++) {
            insert_int(10,m);
            for(int d=days[m];d>0;d--) {
                insert_int(12,d);
                for(int rem=0;rem<100;rem++) {
                    insert_int(14,rem);
                    if(calc_lastbit())
                        fwrite(s,1,19,fout);
                }
            }
        }
    }
}
```

上面这个代码的命令行参数是年份，将把该年的所有尾号为 X 的合法身份证号输出到 stdout。调用该程序生成字典，然后用 john 破解。脚本如下：

crack.bat

```bat
@echo off
for %%i in (2019 2018 2012 2009 2008 2007 2005 2000 1996 1994 1991 1985 1984 1983 1982 1981) do (
    echo cracking %%i
    gen_id %%i
    john-1.9.0-jumbo-1-win64\run\john --wordlist=id_list.txt --fork=8 john.txt
)
```

运行 4 分钟后得到密码：

```
cracking 1984
Using default input encoding: UTF-8
Loaded 1 password hash (PKZIP [32/64])
Node numbers 1-8 of 8 (fork)
Press 'q' or Ctrl-C to abort, almost any other key for status
32070119840810108X (quantum_encryption.zip)
1 1g 0:00:00:02 DONE (2020-07-28 17:13) 0.4943g/s 2018Kp/s 2018Kc/s 2018KC/s 32070119840814563X..32070119840808112X
```

计算 md5("Sq1it32070119840810108X")即得到 flag。