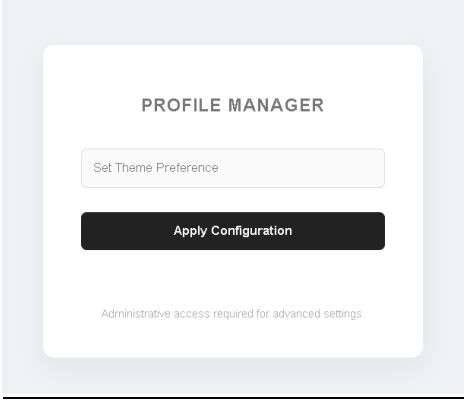
Double Blind

Category**:** Web Exploitation

1. Discovery:



- The application features a user dashboard with endpoints to view (/api/me) and update (/api/v1/update) profiles.
- Accessing the admin panel at /dashboard returns **403 Forbidden**.
- Source code inspection (app.py) shows access is controlled by the user's role.

2. Code Analysis:

- **The Filter:** The application blocks direct updates to sensitive keys like role or admin.
- **The Flaw:** The input validation occurs **before** Unicode Normalization (NFKC).
- **The Chain:**
    1. User sends a payload.
    2. App checks for restricted words (e.g., "role").
    3. App normalizes keys (converts Unicode equivalents like $\mathscr{o}$ to o).
    4. App saves data to the database, overwriting the protected role.

3. The Logic:

- **The Trap (Developer Mode):** Setting the role to "developer" grants partial access but reveals a decoy flag.
- **The Real Exploit (Admin Mode):** Setting the role to "admin" grants full access.
- **SSTI Vector:** The admin dashboard renders the user's theme setting using render_template_string, allowing for **Server-Side Template Injection** to read the server config.

4. Solving:

- **Step 1: Privilege Escalation (Unicode Bypass)**
    - Send the blocked key role using the Unicode character ℴ (U+2134).
    - **Payload:** {"r\u2134le": "admin"}
    - *Result:* Server normalizes this to role: admin.
- **Step 2: SSTI Injection**
    - Inject Jinja2 code into the theme field to read the hidden key.
    - **Payload:** {"profile": {"theme": "{{ config['sys_master_key'] }}"}}
- **Step 3: Flag Retrieval**
    - **Trap Response (Developer):** LNMHACKS{n1c3_try_but_d3v_m0d3_is_us3l3ss}
    - **Real Response (Admin):** LNMHACKS{un1c0d3_byp4ss_int0_sst1_rce_m4st3r}