

Challenge Information

- ## Solution Summary

- 1 Extracting the JWT token from a test login
- 2 Cracking the weak JWT secret key
- 3 Forging a new JWT with elevated privileges
- 4 Accessing the protected vault endpoint

Step 1: Initial Reconnaissance

- ## Step 2: JWT Token Extraction

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6InRlc3QiLCJybzIxIjoiYmVudGF0eSIsInN1YXJlLnR5bWVudFIjoiWDBZR1VBd1FQScjRjaFk5TFBjSWNNSThhUVk4emVsSTkiLCJpYXQiOjE3Njk4MTEwOTR9.wrJ0OzSVmoUu0f0MwRG4reeY7o0jzbHyv4k5yIHmftk

Step 3: JWT Decoding

Header:

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

Payload:

```
{
  "username": "test",
  "role": "hostage",
  "secret_plan": "X0YGUAWQPJ4chY9LPcIcgI8aQY8zelI9",
  "iat": 1769811194
}
```

Step 4: Secret Key Cracking

Performed brute force attack using a wordlist based on the La Casa de Papel theme.

Secret Key Found: dali (referencing Salvador Dalí, whose mask is iconic in the show)

Step 5: JWT Forgery

Created a forged JWT token with modified claims:

- **Username:** professor
- **Role:** Professor (capital P is critical!)
- **Secret:** dali

Forged JWT Token:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzZWNyZXRFcGxhbiI6IlgwWUdVQXNdRUe0Y2hzOUxvY0ljZDk4YVFZOHplbkE5IiwiaWF0IjoxNzY5ODExMTk0LCJlcn2VybmFtZSI6ImBib2Zlc3NvciiIsInJvbGUOiJQcm9mZXRnbnZ3IiwiaWF0IjoxNzY5ODExMTk0LCJlcn2VybmFtZSI6ImBib2Zlc3NvciiIsInJvbGUOiJQcm9mZXRnbnZ3IiwiaWF0IjoxNzY5ODExMTk0LCJlcn2VybmFtZSI6ImBib2Zlc3NvciiIsInR5cCI6IkpXVCJ9.gbb0c13qlMQ58cCeJTfmdi8R1OibfgPCT9fO21Hu26E

Step 6: Vault Access

Using the forged JWT token, successfully accessed the /vault endpoint.

FLAG

```
SECE{M0n3y_H31st_15_0n!}
```

Key Findings

Critical Vulnerabilities

- 5 **Weak JWT Secret Key:** The secret key dali is only 4 bytes long and easily crackable
- 6 **Case-Sensitive Role Validation:** The backend checks for role Professor (capital P), not professor
- 7 **Username/Role Combination:** Required username professor with role Professor

Attack Vector

- **Type:** JWT Secret Key Brute Force + Token Forgery
- **Difficulty:** Medium
- **Tools Used:** Python, PyJWT library, custom wordlist

Successful Payload

```
{  
  
  "username": "professor",  
  "role": "Professor",  
  "secret_plan": "X0YGUAwQPJ4chY9LPcIcgI8aQY8zelI9",  
  "iat": 1769811194  
  
}
```

Additional Information

The vault page also contained an encrypted secret plan hint:

- **Encrypted:** X0YGUAwQPJ4chY9LPcIcgI8aQY8zeII9
 - **Hint:** "First rotation (ROT3), then encoding (Base64)"
 - **Note:** This appears to be a bonus challenge within the main challenge
-

Recommendations for Remediation

- 8 **Use Strong Secret Keys:** Implement cryptographically secure random keys (minimum 32 bytes)
 - 9 **Key Rotation:** Regularly rotate JWT signing keys
 - 10 **Add Expiration:** Implement short-lived tokens with exp claim
 - 11 **Rate Limiting:** Prevent brute force attacks on authentication endpoints
 - 12 **Additional Validation:** Implement IP-based or session-based validation
 - 13 **Use RS256:** Consider using asymmetric algorithms (RS256) instead of HS256
-

Tools and Scripts Used

- 14 jwt_decode.py - JWT token analysis
 - 15 jwt_forge.py - Token generation with various payloads
 - 16 crack_jwt_secret.py - Secret key brute force
 - 17 exhaustive_test.py - Systematic testing of username/role combinations
-

Challenge Completed Successfully! 🎉