Web Application Security

Student number: AB0197

Name: Veeti Hakala

Group: TIC21S

Time management: Approximately 10 hours

1 Week 02

1.1 JWT Tokens:

1.1.1 Old Wasdat - Curl - Change password using curl

Title: Change password feature doesn't require old password. Password can be changed via `CURL`.

Description: Web application has a vulnerable change password feature. Authentication with old password is not implemented in the application.

Steps to produce:

1. Navigate to `https://wasdat.fi/8080`.
2. Create a user & login in to the website.
3. Open developer tools and head to user settings.
4. Change password in a browser and copy http request cURL. (Adding something in to `bio` field is required first.)
5. Modify curl statement:

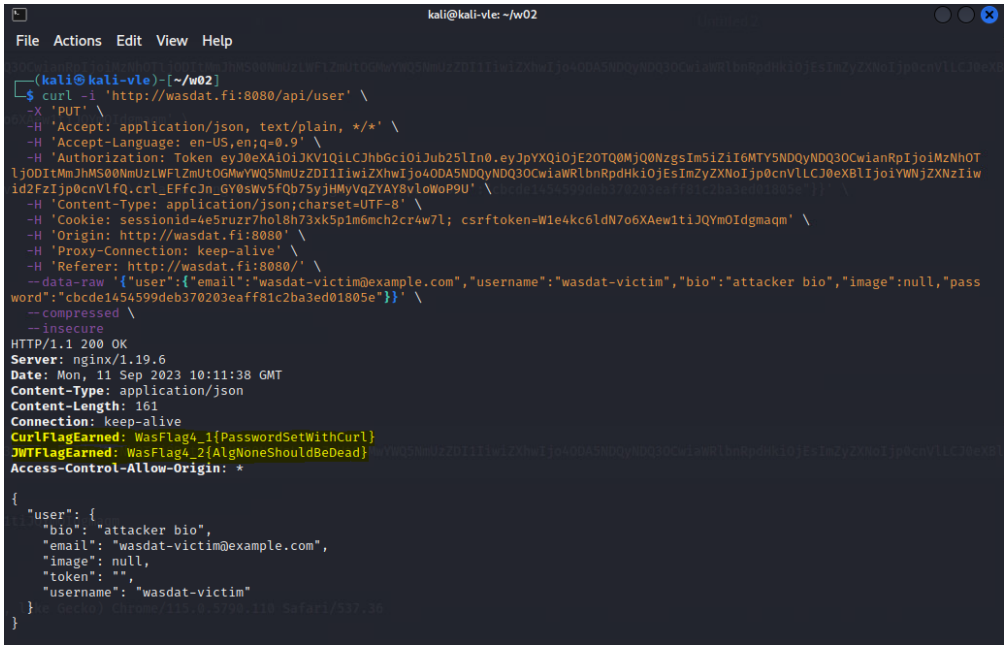
   Remove browser header and set password to `JAMK2023` -> `cbcde1454599deb370203eaff81c2ba3ed01805e`

```
curl -i 'http://wasdat.fi:8080/api/user' ¥
-X 'PUT' ¥
-H 'Accept: application/json, text/plain, */*' ¥
-H 'Accept-Language: en-US,en;q=0.9' ¥
-H 'Authorization: Token
eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJpYXQiOjE2OTQ0MjQ0NzgsIm5iZiI6MTY5
NDQyNDQ3OCwianRpIjoiMzNhOTljODItMmJhMS00NmUzLWFlZmUtOGMwYWQ5NmUzZDI1Iiwi
ZXhwIjo4ODA5NDQyNDQ3OCwiaWRlbnRpdHkiOjEsImZyZXNoIjp0cnVlLCJ0eXBlIjoiYWNj
ZXNzIiwid2FzIjp0cnVlfQ.crl_EFfcJn_GY0sWv5fQb75yjHMyVqZYAY8vloWoP9U' ¥
-H 'Content-Type: application/json;charset=UTF-8' ¥
-H 'Cookie: sessionid=4e5ruzr7hol8h73xk5p1m6mch2cr4w7l;
csrftoken=W1e4kc6ldN7o6XAew1tiJQYmOIdgmaqm' ¥
-H 'Origin: http://wasdat.fi:8080' ¥
-H 'Proxy-Connection: keep-alive' ¥
-H 'Referer: http://wasdat.fi:8080/' ¥
--data-raw
'{"user":{"email":"attacker¥@example.com","username":"attacker","bio":"a
ttacker
bio","image":null,"password":"cbcde1454599deb370203eaff81c2ba3ed01805e"}
}' ¥
```

```
--compressed ¥
--insecure
```

6 Run command from Kali CLI and password is set to `JAMK2023`.

7 You also get the flag: CurlFlagEarned: WasFlag4_1{PasswordSetWithCurl}`.



- Impact estimation: **High Severity**
  1 Unauthorized Account Access: The vulnerability enables malicious actors to modify the password of any user account without knowing the original password. This can lead to unauthorized access to sensitive account information, manipulation of account settings, or misuse of the account in malicious activities.
    - See: https://cwe.mitre.org/data/definitions/620.html
  2 Potential for Mass Account Compromises: Given the ease with which this vulnerability can be exploited (using CURL), there is a risk of automated attacks, potentially compromising a large number of user accounts.
  3 Data Leakage: Post unauthorized access, sensitive personal information related to the user might be accessed, leading to privacy breaches.
- Mitigation:
  - Implement Old Password Check: Before allowing a user to change their password, they should be prompted to enter their current (old) password. This will ensure that only the legitimate owner of the account (or someone who has the current password) can change the password.
    - ✧ See: https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html#change-password-feature
  - Multi-Factor Authentication (MFA): Encourage or enforce users to set up multi-factor authentication. Even if an attacker changes the password, they will be unable to login without the second factor.

1.1.2 Old wasdat - Craft JWT token with alg=none and change user's password

Title: Encoding algorithm can be bypassed with setting alg=none `JWT Token`.

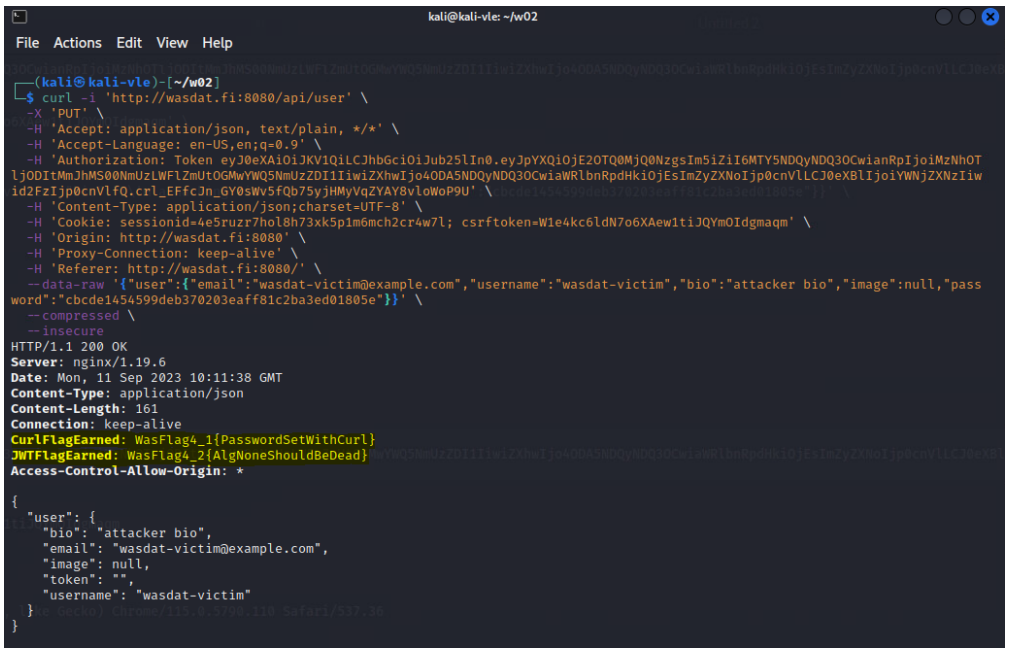
Description: Web Application has misconfiguration in JWT Token authentication. Application has secret_key but it can be bypassed with setting encryption alg=none. Basically it sends and empty signature and application accepts it as valid signature.

Steps to produce:

1 Navigate to `https://wasdat.fi/8080`.
2 Create a user & login in to the website.
3 Open developer tools and head to user settings.
4 Change password in a browser and copy http request cURL. (Adding something in to `bio` field is required first.)
5 Modify curl statement:

Remove browser header and set password to `JAMK2023`. Take first part of the `JWT Token` which is `eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9`. Decode in `https://jwt.io`.

6 You will see that payloads is: `{"typ":"JWT","alg":"HS256"}` change it to: `{"typ":"JWT","alg":"none"}`
7 Decode it again and replace old value in the Authentication Token field.
8 Run curl command from kali cli.
9 You'll get flag: `JWTFagEarned: WasFlag4_2{AlgNoneShouldBeDead}`



- Impact estimation: **Critical**
  - Unauthorized Account Access: The vulnerability allows attackers to modify JWT tokens and change user passwords without knowing the original ones. Such unauthorized access can lead to data breaches, account misuse, and unauthorized changes to user settings.

- Impersonation: By manipulating the JWT token, an attacker might be able to impersonate any user, potentially gaining access to privileged accounts.
- Potential for Mass Account Takeovers: Given the nature of JWT and its widespread usage, attackers can script attacks to potentially compromise numerous accounts.
- Data Leakage: Successful exploitation could lead to unauthorized access and exfiltration of sensitive user data.

- Mitigation:
  - Disallow "none" Algorithm: Update the JWT processing logic to reject tokens that specify the "none" algorithm unless they are expected and strictly required for a specific reason.
    - ✧ See: https://auth0.com/blog/a-look-at-the-latest-draft-for-jwt-bcp/
  - Implement Stronger Signature Verification: Ensure that all JWT tokens are verified using a robust and secure method before being processed.
  - Auditing and Monitoring: Introduce logging mechanisms for authentication and token processing. Monitor these logs for anomalies, such as repeated failed login attempts or unexpected use of the "none" algorithm.
  - Update Libraries: Ensure that you are using the latest and most secure version of JWT processing libraries. Many libraries have addressed vulnerabilities related to the "none" algorithm.
- Related OWASP CWE:
  - CWE-347: Improper Verification of Cryptographic Signature
  - CWE-306: Missing Authentication for Critical Function
  - CWE-287: Improper Authentication

1.1.3 Old wasdat - Craft JWT token with known secret and impersonate to be the victim :

Title: JWT Token `secret` has leaked, which is librarys default value.

Description: Web application has vulnerability in the JWT Token `secret`. It is using default value. Any user can impersonate other users because authentication is always validated if secret is known and is used in the `http payload`.

Steps to produce:

1 Navigate to `https://wasdat.fi/8080`.

2 Create a user & login in to the website.

3 Open developer tools and head to user settings.

4 Change password in a browser and copy http request cURL. (Adding something in to `bio` field is required first.)

5 Modify curl statement:

Remove browser header and set password to `JAMK2023`. Copy the Authentication `JWT Token`. Decode in `https://jwt.io`.

6 You will see that payloads is:

```
HMACSHA256(
```

```
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
)
```

7 Add `secret-key` to signature:

```
  HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret-key
  )
```

8 Decode it again and replace old value in the Authentication Token field.

9 Run curl command from kali cli.

10 You'll get flag: `JWTFagEarned: WasFlag4_3{AchievementUnlocked_MasterOfTokens}.`

- Impact estimation: **Critical**
  - Unauthorized Account Access: Attackers who know the JWT secret can generate or modify JWT tokens, potentially leading to unauthorized access to any account, including admin or privileged ones.
  - Impersonation: With the ability to manipulate the JWT token, an attacker can impersonate any user, gaining unauthorized access to potentially sensitive user data and operations.
- Mitigation:
  1 Change the JWT Secret: Immediately rotate the JWT secret to a complex, unique value that is not based on any default or easily guessable value.
  2 Secure Secret Management: Ensure that secrets are stored securely, such as in environment variables, secret management tools, or encrypted databases. They should never be hardcoded in the application or leaked in repositories.
  3 Implement Stronger Signature Verification: Ensure that all JWT tokens are verified against the secret before being processed. Ensure the use of strong cryptographic algorithms for signing.
  4 Periodic Secret Rotation: Establish a policy to rotate secrets periodically, thus reducing the window of exposure should they ever be compromised.

5 Auditing and Monitoring: Implement strict logging and monitoring mechanisms. Be alert for suspicious activities, such as unexpected spikes in login attempts or unauthorized data access patterns.

6 User Notification: Notify affected users about the vulnerability. Encourage password resets and review of account activities.

7 Update Libraries and Dependencies: Ensure the usage of updated, secure versions of JWT processing libraries and other dependencies.

- Related OWASP CWE:
  - CWE-326: Inadequate Encryption Strength
  - CWE-347: Improper Verification of Cryptographic Signature.
  - CWE-798: Use of Hard-coded Credentials
  - CWE-359: Exposure of Private Information ('Privacy Violation')