🤩

API2:2019 Broken User Authentication

Introduction

API2:2019 Broken User Authentication

Threat agents/Attack vectors	Security weakness	Impacts
It's not easy to implement authentication correctly. A lot of developers and architects get confused by what to include in the authentication control because software these days often is made up of complex multi-layered systems. Usually authentication mechanisms are also an often targeted system due to it being publicly available.	One of the issues that can arise is that we implement the authentication control incorrectly. API endpoints that handle authentication need to be designed differently from other endpoints and this is often overlooked. Another issue that might arise is that the authentication is implemented incorrectly. For example we might have an endpoint for mobile app authentication that is being used for web application authentication as well.	This vulnerability can easily lead to an attacker taking over the complete account of the victim. They can then imporsonate the victim and steal their privata data that is saved on the site.
Untitled		

What is Broken User Authentication?

Broken User Authentication can manifest in several issues. Whenever we come across an API endpoint that handles authentication we need to be extra careful since these endpoints will often determine how a user can flow through the application and what data they see. Whenever one of the following conditions is true, we can speak of a "Broken User Authentication

- If your API allows for credential stuffing, This is an attack where a hacker will try to brute force credentials by using a list with known account names and password from other websites. This works because people often re-use their old usernames and passwords so if their credentials leak one time, a lot of accounts are vulnerable. Attackers will take these old credentials and try them on our API endpoints rapidly to see if any of the accounts work on our application. Rate limiting should exits.
- If the API endoint does not verify the request with a captcha when needed
- When the application allows for weak passwords. The best policy is to use a passphrase instead of a password but anything is better than a weak password.
- When the endpoint uses GET parameters (parameters in the URL) to send sensitive data such as passwords or tokens. This could allow for a MiTM attack.
- When our endpoint issues a token such as JWT but does not validate it.
- When we use JWT, we should be aware that we should not accept unsigned or weakly signed tokens.
- When the endpoint does not validate the expiration date of authentication tokens such as session tokens or JWT
- When the endpoint does not encrypt or hash the password or uses weak encryption algorithms.
- When it uses weak encryption keys that are easy to guess (Like a birthday) or easy to crack (like md5)

Example Attack Scenarios

password recovery

The attacker might start the workflow to reset a password by triggering the /api/v1/reset-password endpoint.

```
POST /api/v1/reset-password
{
  userID=123
}
```

This will trigger a password reset for user "rat" and the user will receive a password reset token in their mailbox which is a 4 digit number. Since there is no rate limiting on the endpoint, the attacker can try to send all 4 digit numbers in rapid succession and simply brute force it.

```
POST /api/v1/reset-password-token
{
  userID=123
  tokenID=xxxx
  newPass=test
}
```

the attacker can then guess the token and reset the password for the user.

JWT validation endpoint accepts "None" algorithm

A JWT endpoint should always validate the token with the proper algortihm, most JWT frameworks have the None algorith enabled by default and this very bad, to know why we shoud have a look at how JWT for work first.

A JWT token is Json Web Token. They are tokens containing information about users for example and the beauty is that we can always easily decode these and view the information. If we want to change anything though we have to sign it with an algorith.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjIzNCIsInN1YiI6IjEyMzQ1Njc4OTAiLCJuYW1lIjoiS
m9obiBEb2UiLCJpYXQiOjE1MTYyMzkwMjJ9.lt2GhI6wX0D46cGiKk7wSiqUXdGYZXtHXZIXrKQThNI

This is an example of a JWT token, when we decode it we get the following:
{
  "alg": "HS256",
  "typ": "JWT"
}
{
  "userID": "123",
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}

If we want to edit this JWT token we will have to know the HS256 key.
```

Now that you know how JWT's work, you can see why it's bad if the verification mechanismins accepts the None algorith. We can simply change something in the JWT and encode it again when we change the algorith to None.

```
{
  "alg": "None",
  "typ": "JWT"
}
{
  "userID": "567",
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjU2NyIsInN1YiI6IjEyMzQ1Njc4OTAiLCJuYW1lIjoiS
m9obiBEb2UiLCJpYXQiOjE1MTYyMzkwMjJ9.NhHRUCQw5Wtuc7Jn2ImmC8URY0UmcuEkukmNA9Frccs
```

If we now replace this token in our headers when we make a request, we should see an error because our token should not be valid, however if the server is still configured to accept requests signed with the None algorithm, they will be vulnerable to Broken User Authentication.

Preventive measures against Broken User Authentication

- It's really important to map all the flows that are related to authentication on your API's. Make sure you list all of the flows, including mobile, deeplinks, one-click login etc...
- Understand exactly how your authentication mechanisms work, don't just blindly implement something like oauth 2, a lot of developers implement this incorrectly.

- Do not implement your own authentication mechanisms but use well known authentication solutions
- All authentication endpoints (Including forgot password) should be protected by rate limiting it and implementing lockout mechanismsm. These mechanisms have to be stricted than on other endpoints.
- OWASP Authentication Cheatsheet is a great reference for implementing authentication
- If possible, implement multi-factor authentication such as SMS or authenticators
- Implement captcha mechanisms to prevent attackers from brute forcing your service on top of your rate limiting and lockout mechanisms
- Do not use API keys for authentication of the user, these should be used for client/app authentication

Conclusion

When dealing with authentication endpoints we need to implement much stricted security mechanisms than when dealing with normal endpoints. We need to make sure we have good rate limiting, lockout and captcha mechanisms to prevent attackers from brute forcing or credential stuffing our API's. Make sure you implement safe authentication mechanisms and if you are unsure you can always refer to the OWASP Authentication Cheatsheet.