API2:2023 Broken Authentication

Threat agents/Attack vectors	Security Weakness	Impacts
API Specific : Exploitability **Easy**	Prevalence **Common** : Detectability **Easy**	Technical **Severe** : Business Specific
The authentication mechanism is an easy target for attackers since it's exposed to everyone. Although more advanced technical skills may be required to exploit some authentication issues, exploitation tools are generally available.	Software and security engineers’ misconceptions regarding authentication boundaries and inherent implementation complexity make authentication issues prevalent. Methodologies of detecting broken authentication are available and easy to create.	Attackers can gain complete control of other users’ accounts in the system, read their personal data, and perform sensitive actions on their behalf. Systems are unlikely to be able to distinguish attackers’ actions from legitimate user ones.

Is the API Vulnerable?

Authentication endpoints and flows are assets that need to be protected. Additionally, "Forgot password / reset password" should be treated the same way as authentication mechanisms.

An API is vulnerable if it:

- Permits credential stuffing where the attacker uses brute force with a list of valid usernames and passwords.
- Permits attackers to perform a brute force attack on the same user account, without presenting captcha/account lockout mechanism.
- Permits weak passwords.
- Sends sensitive authentication details, such as auth tokens and passwords in the URL.

- Allows users to change their email address, current password, or do any other sensitive operations without asking for password confirmation.
- Doesn't validate the authenticity of tokens.
- Accepts unsigned/weakly signed JWT tokens (`{"alg":"none"}`)
- Doesn't validate the JWT expiration date.
- Uses plain text, non-encrypted, or weakly hashed passwords.
- Uses weak encryption keys.

On top of that, a microservice is vulnerable if:

- Other microservices can access it without authentication
- Uses weak or predictable tokens to enforce authentication

Example Attack Scenarios

Scenario #1

In order to perform user authentication the client has to issue an API request like the one below with the user credentials:

```
POST /graphql
{
  "query":"mutation {
    login (username:\"<username>\",password:\"<password>\") {
      token
    }
   }"
}
```

If credentials are valid, then an auth token is returned which should be provided in subsequent requests to identify the user. Login attempts are subject to restrictive rate limiting: only three requests are allowed per minute.

To brute force log in with a victim's account, bad actors leverage GraphQL query batching to bypass the request rate limiting, speeding up the attack:

```
POST /graphql
[
  {"query":"mutation{login(username:\"victim\",password:\"password\"){toke
n}}"},
  {"query":"mutation{login(username:\"victim\",password:\"123456\"){toke
n}}"},
  {"query":"mutation{login(username:\"victim\",password:\"qwerty\"){toke
n}}"},
  ...
  {"query":"mutation{login(username:\"victim\",password:\"123\"){token}}"},
]
```

Scenario #2

In order to update the email address associated with a user's account, clients should issue an API request like the one below:

```
PUT /account
Authorization: Bearer <token>

{ "email": "<new_email_address>" }
```

Because the API does not require users to confirm their identity by providing their current password, bad actors able to put themselves in a position to steal the auth token might be able to take over the victim's account by starting the reset password workflow after updating the email address of the victim's account.

How To Prevent

- Make sure you know all the possible flows to authenticate to the API (mobile/web/deep links that implement one-click authentication/etc.). Ask your engineers what flows you missed.
- Read about your authentication mechanisms. Make sure you understand what and how they are used. OAuth is not authentication, and neither are API keys.

- Don't reinvent the wheel in authentication, token generation, or password storage. Use the standards.
- Credential recovery/forgot password endpoints should be treated as login endpoints in terms of brute force, rate limiting, and lockout protections.
- Require re-authentication for sensitive operations (e.g. changing the account owner email address/2FA phone number).
- Use the OWASP Authentication Cheatsheet.
- Where possible, implement multi-factor authentication.
- Implement anti-brute force mechanisms to mitigate credential stuffing, dictionary attacks, and brute force attacks on your authentication endpoints. This mechanism should be stricter than the regular rate limiting mechanisms on your APIs.
- Implement account lockout/captcha mechanisms to prevent brute force attacks against specific users. Implement weak-password checks.
- API keys should not be used for user authentication. They should only be used for API clients authentication.