Week 1: security Assessment report

Application Setup

I set up the mock website by importing it from the github, and downloaded it as zip file.

• Installing necessary packages(e.g: npm install)

• Running the server locally

• Accessing the application on http://localhost:3000

• Exploring key pages such as Signup, Login, and Profile.

The website is now running properly on local host for security testing.

Vulnerability Assessment

- **OWASP zap vulnerability scanning**

10 risks were identified using OWASP zap

1. Absence of Anti-CSRF Tokens

   Risk level: Medium
   <form action="/login" method="POST" class="card-body">

   Description
   No Anti-CSRF tokens were found in a HTML submission form.

2. Content Security Policy (CSP) Header Not Set
   Risk Level: Medium

   Description
   Passive (10038 - Content Security Policy (CSP) Header Not Set)

3. Missing Anti-clickjacking Header
   Risk Level: Medium

Description

The response does not protect against 'ClickJacking' attacks. It should include either Content-Security-Policy with 'frame-ancestors' directive or X-Frame-Options.

4. Cookie without SameSite Attribute
   Risk Level: Low

   Description

   A cookie has been set without the SameSite attribute, which means that the cookie can be sent as a result of a 'cross-site' request.

5. Cross-Domain JavaScript Source File Inclusion
   Risk Level: Low

   Description

   The page includes one or more script files from a third-party domain.

6. Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)
   Risk Level: Low

   Description

   The web/application server is leaking information via one or more "X-Powered-By" HTTP response headers. Access to such information may facilitate attackers identifying other frameworks/components your web application is reliant upon and the vulnerabilities such components may be subject to.

7. X-Content-Type-Options Header Missing
   Risk Level: Low

   Description

   The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type.

8. Authentication Request Identified
   Risk Level: High

Description

The given request has been identified as an authentication request. The 'Other Info' field contains a set of key=value lines which identify any relevant fields. If the request is in a context which has an Authentication Method set to "Auto-Detect" then this rule will change the authentication to match the request identified.

9. Information Disclosure - Suspicious Comments
   Risk Level: High

   Description

   The response appears to contain suspicious comments which may help an attacker.

10. Session Management Response Identified
    Risk Level: High

    Description

    The given response has been identified as containing a session management token. The 'Other Info' field contains a set of header tokens that can be used in the Header Based Session Management Method. If the request is in a context which has a Session Management Method set to "Auto-Detect" then this rule will change the session management to use the tokens identified.

Areas For Improvement

1. Secure Authentication Flow

- Issue: Authentication requests are in the open and may be intercepted.
- Improvement:
  - Utilize HTTPS (TLS) to encrypt all login credentials in transit.
  - Use rate limiting and CAPTCHA to defend against brute-force attacks.
  - Have all authentication tokens signed and validated on the server side.

2. Eliminate Suspicious or Debug Comments

- Issue: Sensitive or debug data remain in comments, potentially providing attackers with an understanding of app logic or internal structure.
- Improvement:

o Audit all backend, JS, and HTML code for residual comments revealed.

o Destroy or obfuscate any comments that contain credentials, structure, API keys, or debug information prior to deployment.

3. Secure Session Management

- Issue: Session tokens are leaked in responses, making it easier to hijack sessions.
- Improvement:

o Use secure, HTTP-only cookies for session tokens.

o Set the SameSite=Strict or Lax and Secure flags on session cookies.

o Enable session timeout and token rotation

to minimize the effect of session theft.