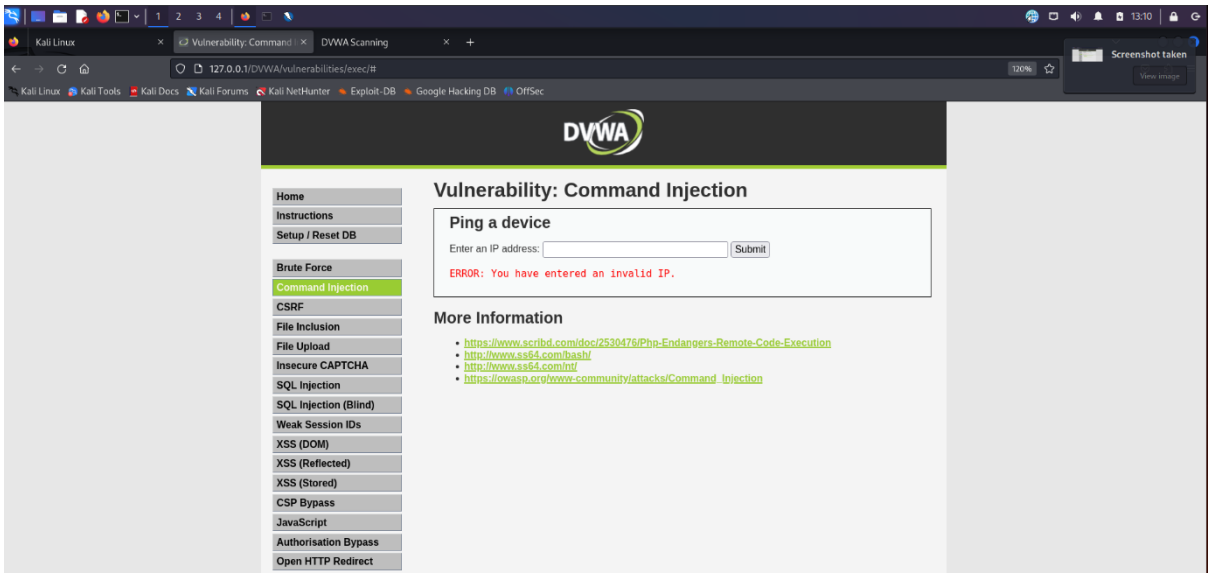
Web Application Security Assessment Report

Testing Website :- 127.0.0.1/DVWA/

The **DVWA** Vulnerable Web Application (DVWA) is a extremely insecure web application created for security professionals to practice their skills and learn about web vulnerabilities. It provides a safe environment to test various types of attacks, such as SQL injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF). DVWA is an excellent tool for learning and improving web application security assessment techniques.



1. Content Security Policy (CSP) Header Not Set

- **Description:** The Content Security Policy (CSP) header is a security feature that helps prevent cross-site scripting (XSS) attacks, clickjacking, and other code injection attacks. By not setting a CSP header, the application is more vulnerable to these attacks, as it allows the execution of malicious scripts or content from untrusted sources.

- **Solution:** Implement a CSP header in your application's HTTP response headers. For example:

```
Content-Security-Policy: default-src 'self'; script-src 'self'
https://trustedscripts.example.com; object-src 'none';
```

This policy will restrict the sources of content to trusted origins.

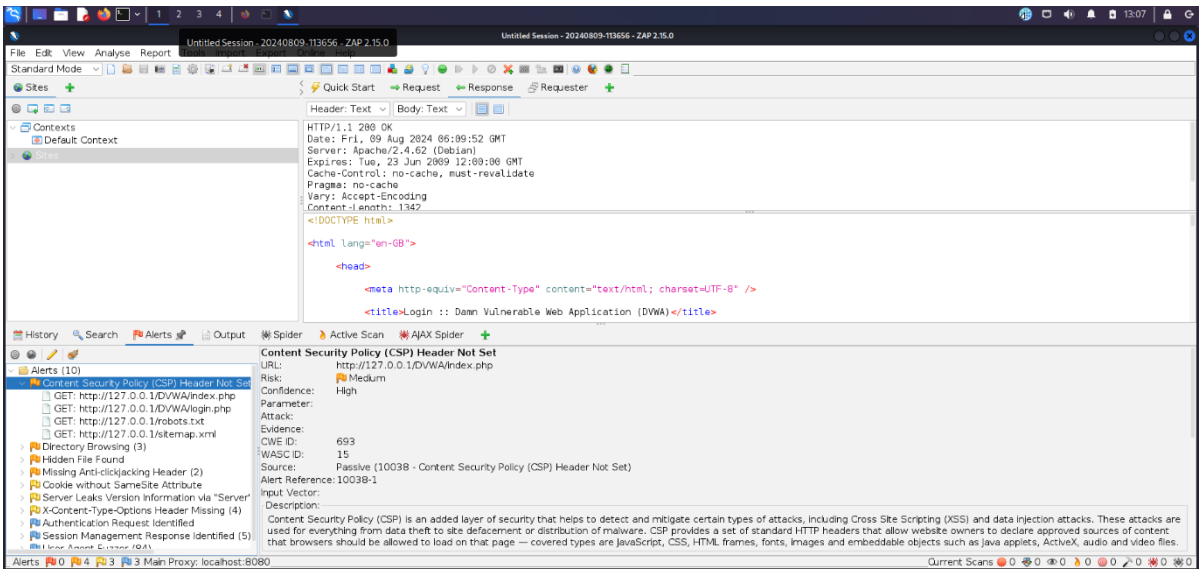
2. Directory Browsing

- **Description:** Directory browsing allows attackers to view the contents of directories on your web server. This can expose sensitive files, configuration files, or other data that could be used to exploit the application.

- **Solution:** Disable directory browsing by configuring your web server settings. For Apache, you can disable directory browsing by adding:

```
Options -Indexes
```

to your .htaccess file or server configuration.



3. Hidden File Found

- **Description**: Hidden files are files that are not meant to be publicly accessible but can be discovered by attackers. These files may contain sensitive information such as configuration settings, backup data, or other internal resources.

- **Solution**: Ensure that hidden files are not accessible via the web. Move them to a secure location outside the web root directory or use access control mechanisms to restrict access.

4. Missing Anti-clickjacking Header

- **Description:** The absence of anti-clickjacking headers, such as X-Frame-Options, makes the application vulnerable to clickjacking attacks. In a clickjacking attack, a malicious site can trick users into clicking on something different from what they perceive, leading to unintended actions on your website.

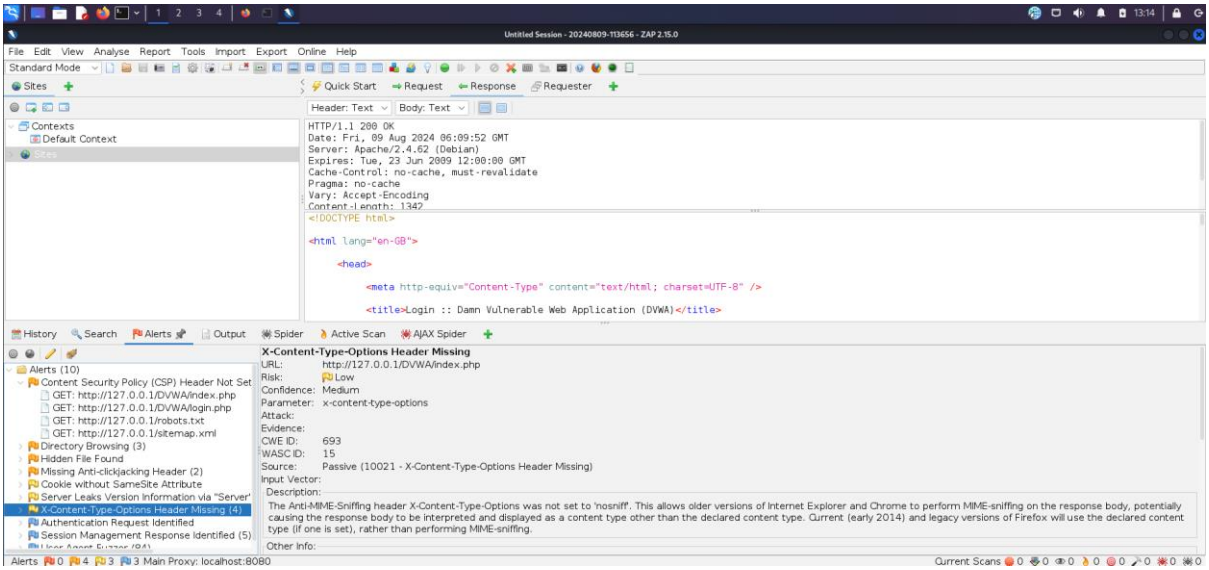
- **Solution:** Add the X-Frame-Options header to your HTTP responses. For example:

```
X-Frame-Options: DENY
```

This will prevent your website from being embedded in iframes, protecting against clickjacking.

5. Cookie without SameSite Attribute

- **Description:** The SameSite attribute on cookies helps mitigate the risk of cross-site request forgery (CSRF) attacks by restricting how cookies are sent with cross-site requests. Without this attribute, cookies may be sent inappropriately, increasing the risk of CSRF.



- **Solution:** Set the SameSite attribute for cookies in your application. For example:

```
Set-Cookie: sessionId=abc123; SameSite=Lax;
```

This will ensure that cookies are not sent with cross-site requests, reducing the risk of CSRF.

6. Server Leaks Version Information via "Server" HTTP Response Header Field

- **Description:** The Server HTTP header in responses reveals the version of the web server software in use. This information can help attackers target specific vulnerabilities associated with that version.

- **Solution:** Configure your web server to hide or modify the Server header. For Apache, you can set:

```
ServerSignature Off
ServerTokens Prod
```

This will prevent the server from revealing its version in the response headers.

7. X-Content-Type-Options Header Missing

- **Description:** The X-Content-Type-Options header is used to prevent MIME type sniffing, which can lead to security vulnerabilities such as XSS attacks. Without this header, browsers may incorrectly interpret files as executable scripts, leading to potential security risks.

- **Solution:** Add the X-Content-Type-Options header to your HTTP responses. For example:

  ```
  X-Content-Type-Options: nosniff
  ```

  This will instruct browsers not to guess the MIME type and only execute files with the specified Content-Type.

8. Authentication Request Identified

- **Description:** An authentication request was detected during the scan. This may indicate a potential vulnerability if the authentication mechanism is not properly secured. Weak authentication can lead to unauthorized access and data breaches.

- **Solution:** Ensure that the authentication mechanism is robust by implementing strong password policies, using multi-factor authentication, and securing the transmission of credentials using HTTPS.

9. Session Management Response Identified

- **Description:** Session management issues can lead to security vulnerabilities such as session fixation or hijacking. Proper session management is crucial to ensure that user sessions are secure and cannot be exploited by attackers.

- **Solution:** Implement secure session management practices, such as regenerating session IDs after login, setting secure and HttpOnly flags on session cookies, and using short session expiration times.

User Agent Fuzzer

- **Description:** A user agent fuzzer is a tool that tests the application by sending various user-agent strings to identify vulnerabilities. This may indicate a potential vulnerability if the application does not properly handle unexpected or malformed user-agent strings.

- **Solution:** Ensure that your application properly handles all input, including user-agent strings, by validating and sanitizing them to prevent injection attacks or other vulnerabilities.