Security Final Project

6432106821 Punyaphat Surakiatkamjorn

1. Brute Force

- **URL Endpoint:**
  `http://localhost:4280/vulnerabilities/brute/#`

- **Method:**
  Using the assumption that most websites have an `admin` role, the username `admin` was tested with multiple common passwords. The password list was sourced from the provided reference. The correct password was identified as `password`.

- **Description:**
  Brute Force is a technique that involves trying numerous combinations of passwords until one works. Attackers might gain unauthorized access if successful.

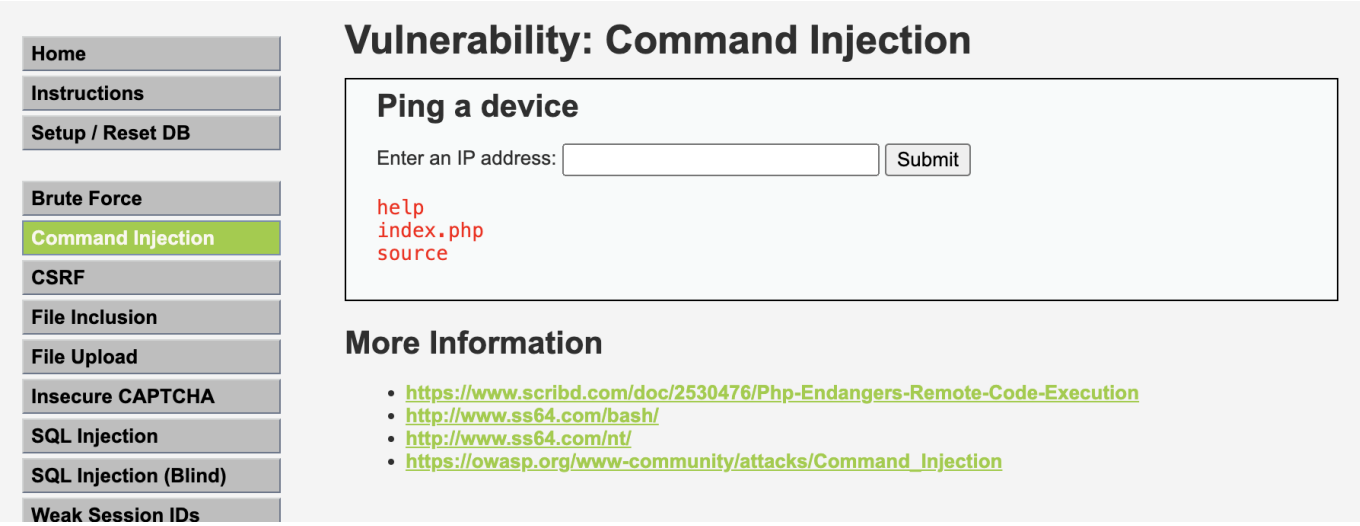
- **Risk Score:**

Exploitability	Weakness Prevalence	Weakness Detectability	Technical Impacts
3	2	3	3

- **Fix/Mitigation Method:**
  - Use stronger passwords combining uppercase, lowercase, numbers, and special characters.
  - Avoid common words found online.
  - Limit login attempts and introduce waiting periods after several failed attempts.
  - Switch to authentication methods like Google OAuth.

---

2. Command Injection



- **URL Endpoint:**
  `http://localhost:4280/vulnerabilities/exec/#`

- **Method:**
  The `security` level was set to `low` in the browser cookies. The `IP address` field was modified to `127.0.0.1 | ls`, which executed the `ls` command via the server's shell script.

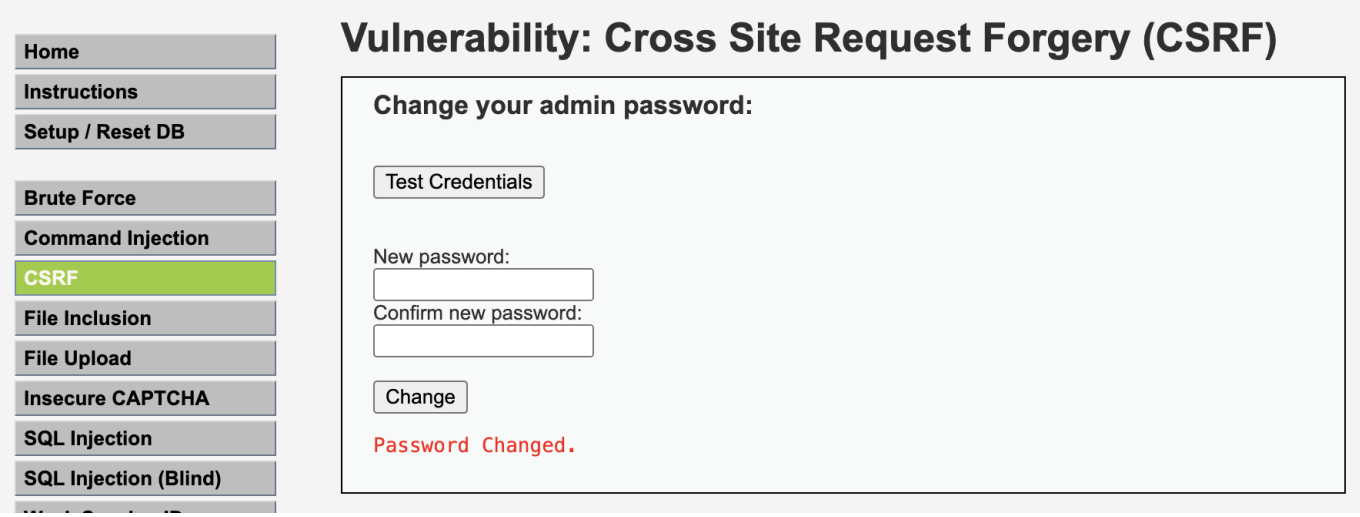
- **Description:**
  Command Injection involves injecting malicious commands via inputs processed by the system. If the input is not validated, these commands might execute and harm the system.

- **Risk Score:**

Exploitability	Weakness Prevalence	Weakness Detectability	Technical Impacts
3	2	2	3

- **Fix/Mitigation Method:**
  - Validate inputs to ensure they match the expected format (e.g., only IPv4 addresses like `127.0.0.1`).

3. Cross Site Request Forgery (CSRF)



- **URL Endpoint:**
  `http://localhost:4280/vulnerabilities/csrf/?password_new=password&password_conf=password&Change=Change#`

- **Method:**
  Observed that submitting the form to change the password exposed the `new password` and `confirmation password` directly in the URL. This URL was then shared with a victim who was logged in. Once they accessed it, their password was changed.

- **Description:**
  CSRF tricks an authenticated user into making unwanted requests to a web application, potentially causing harm, like changing account passwords without consent.

- **Risk Score:**

Exploitability	Weakness Prevalence	Weakness Detectability	Technical Impacts
3	3	3	3

- **Fix/Mitigation Method:**
  - Do not expose sensitive data (e.g., passwords) in URLs.
  - Implement unique, hard-to-guess Anti-CSRF tokens for user sessions.
  - Validate tokens on the server side.

4. File Inclusion

- **URL Endpoint:**
  `http://localhost:4280/vulnerabilities/fi/?page=../../../../../../etc/passwd`

- **Method:**
  The application allowed file viewing through a `page` parameter in the URL. By modifying this parameter to traverse directories (`../../../../../../etc/passwd`), sensitive system files were accessed.

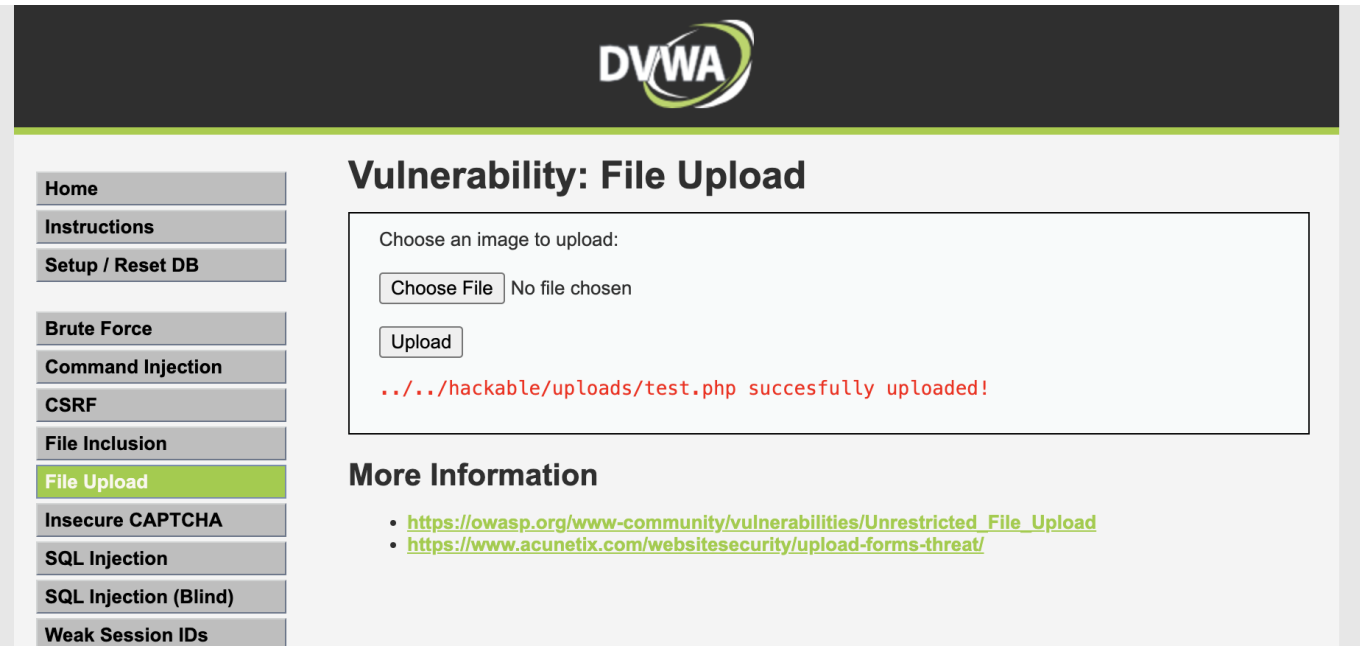
- **Description:**
  Local File Inclusion (LFI) allows attackers to include files from the server via manipulated inputs, potentially accessing sensitive data.

- **Risk Score:**

Exploitability	Weakness Prevalence	Weakness Detectability	Technical Impacts
2	2	2	3

- **Fix/Mitigation Method:**
  - Restrict file access to authorized users.
  - Avoid using file names as URL parameters.
  - Validate and sanitize input.

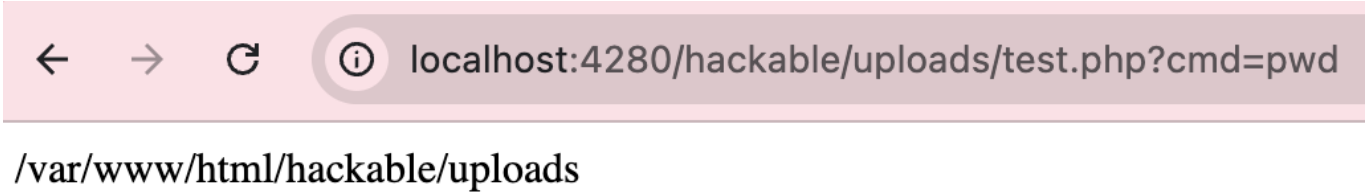
5. File Upload



- **URL Endpoint:**
  `http://localhost:4280/vulnerabilities/upload/#`

- **Method:**
  Uploaded a `.php` file containing malicious code (`system($_REQUEST["cmd"])`) to execute commands via a `cmd` parameter in the URL.

Accessing the uploaded file executed the script.



- **Description:**
  Improperly validated file uploads can lead to malicious scripts being executed on the server, causing significant harm.

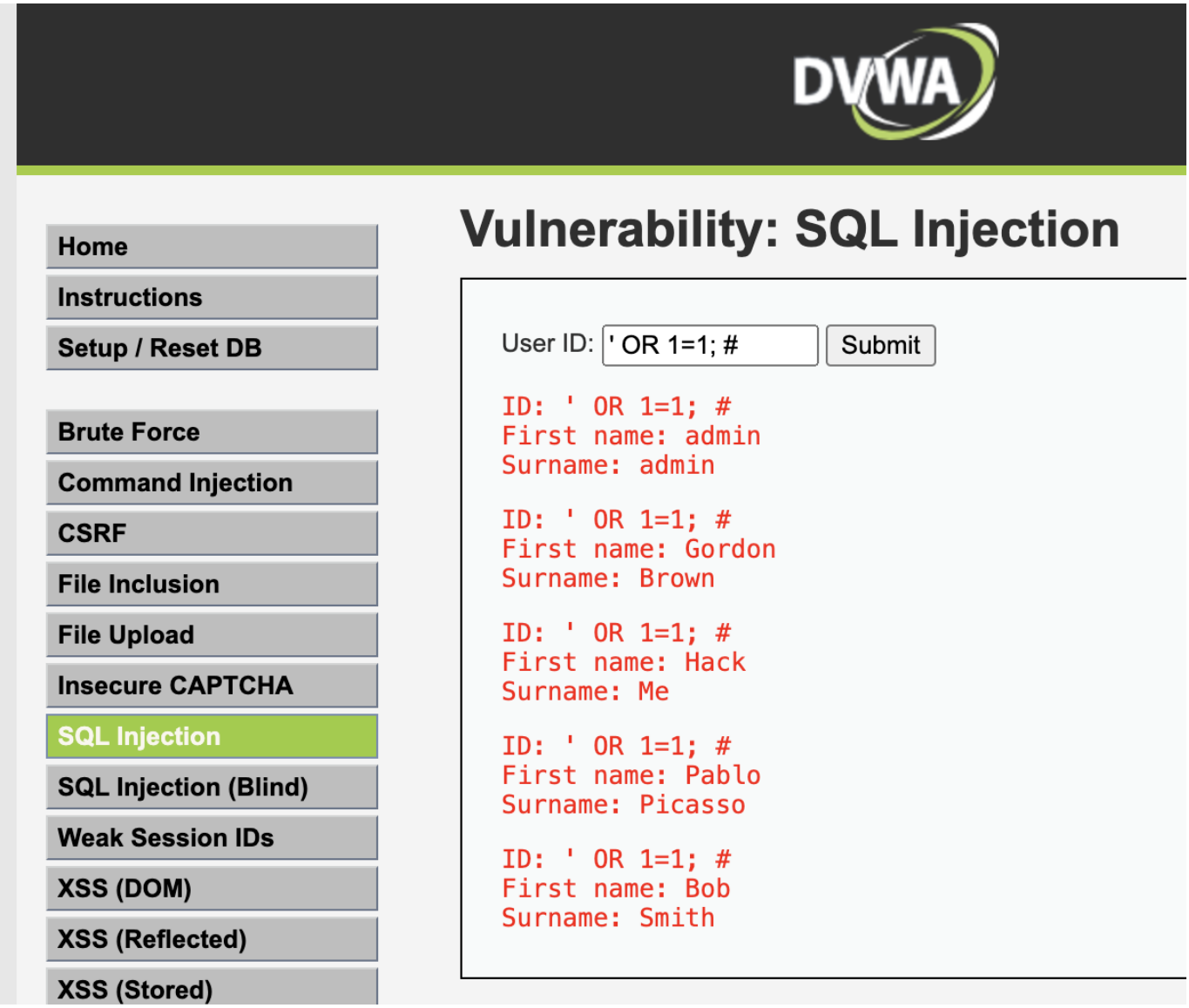
- **Risk Score:**

Exploitability	Weakness Prevalence	Weakness Detectability	Technical Impacts
3	3	3	3

- **Fix/Mitigation Method:**
  - Validate file content and limit file types.
  - Randomize stored file names to obscure their identities.
  - Avoid revealing the file's server location in responses.

---

6. SQL Injection



- **URL Endpoint:**
  `http://localhost:4280/vulnerabilities/sqli/?id=%27+OR+1%3D1%3B+--+&Submit=Submit#`

- **Method:**
  Entered `' OR 1=1; #` in the input field, bypassing authentication and retrieving all user data.

- **Description:**
  SQL Injection occurs when attackers manipulate SQL queries to gain unauthorized data access or modify the database.

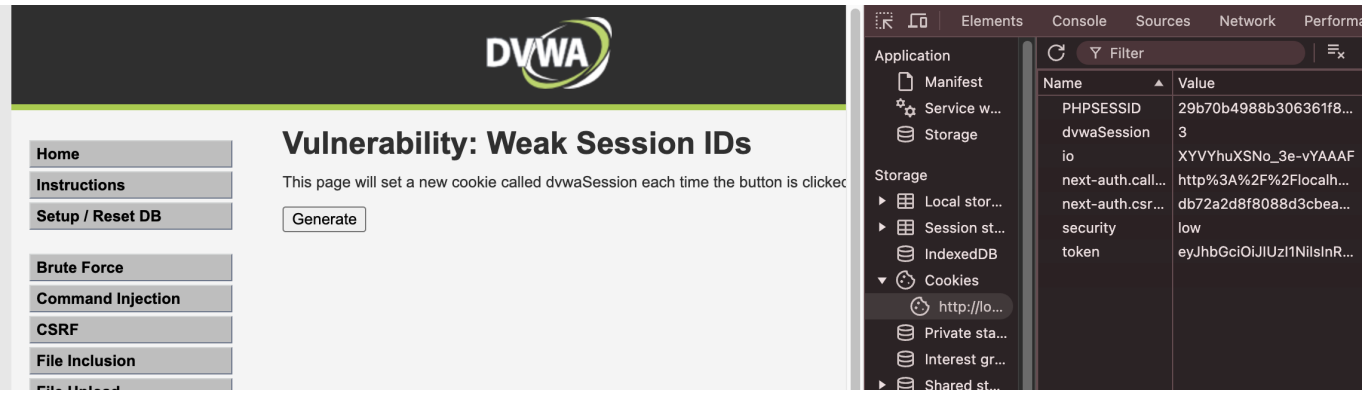
- **Risk Score:**

Exploitability	Weakness Prevalence	Weakness Detectability	Technical Impacts
2	3	2	3

- **Fix/Mitigation Method:**
  - Use prepared statements and parameterized queries.
  - Limit database user privileges.

---

7. Weak Session IDs



- **URL Endpoint:**
  `http://localhost:4280/vulnerabilities/weak_id/`

- **Method:**
  Observed that generating a new session incremented the session ID (`dvwaSession`) sequentially by 1, making it predictable.

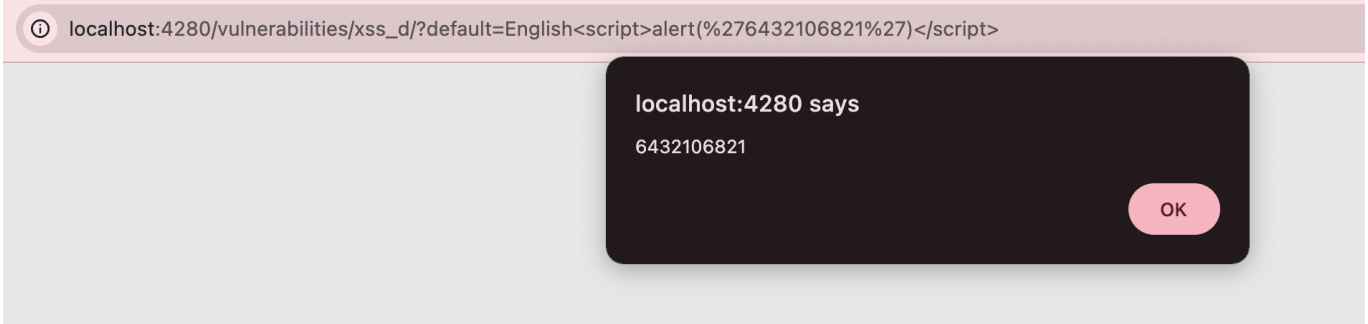
- **Description:**
  Predictable session IDs increase the risk of session hijacking by attackers. If the pattern is known, attackers can guess or brute force session IDs to impersonate users.

- **Risk Score:**

Exploitability	Weakness Prevalence	Weakness Detectability	Technical Impacts
3	2	2	2

- **Fix/Mitigation Method:**
  - Use long, random, and unpredictable session IDs.

8. DOM-Based Cross Site Scripting (XSS)



- **URL Endpoint:**
  `http://localhost:4280/vulnerabilities/xss_d/?default=default=Endlish%3Cscript%3Ealert(6330203521)%3C/script%3E`

- **Method:**
  Injected a `<script>` tag into the `default` query parameter. This resulted in the script being executed within the browser's DOM, displaying an alert box.

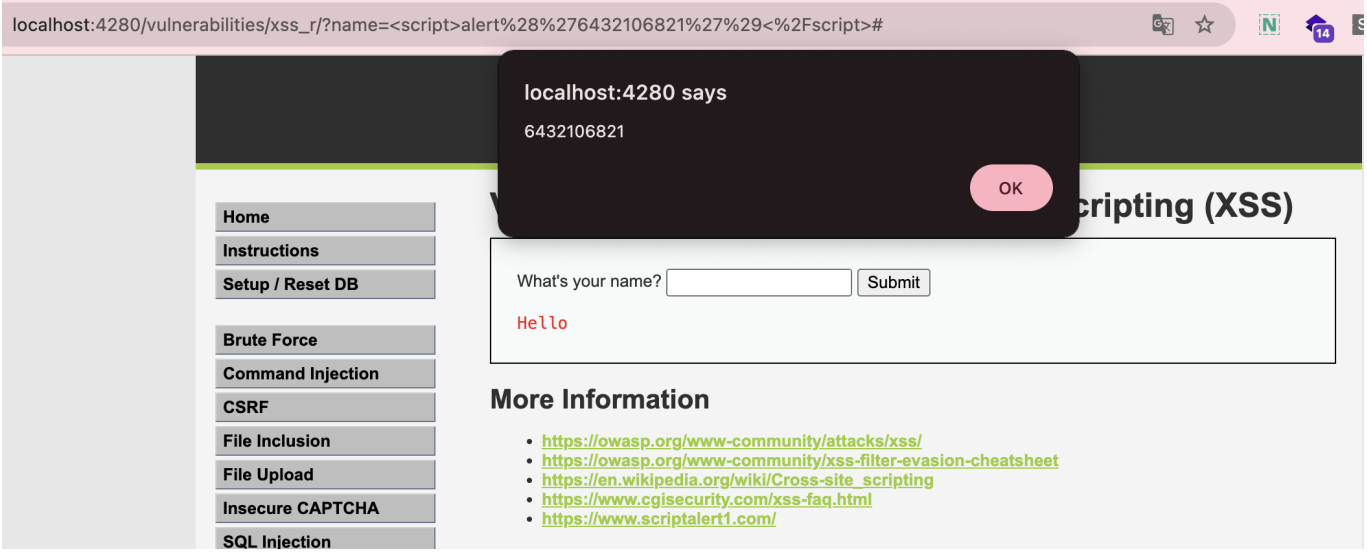
- **Description:**
  DOM-based XSS involves injecting scripts into a web page's DOM, causing them to execute in the user's browser. It can lead to sensitive information theft or unauthorized actions.

- **Risk Score:**

Exploitability	Weakness Prevalence	Weakness Detectability	Technical Impacts
3	3	3	2

- **Fix/Mitigation Method:**
  - Validate and sanitize inputs.
  - Use output encoding to neutralize malicious code.

9. Reflected Cross Site Scripting (XSS)



- **URL Endpoint:**
  `http://localhost:4280/vulnerabilities/xss_r/?name=Hack%21%3Cscript%3Ealert%28%22Get+Hack+na%22%29%3B%3C%2Fscript%3E#`

- **Method:**
  Submitted the input `<script>alert('6432106821')</script>` in a form field. The server reflected the malicious script in the response, which executed when rendered by the browser.

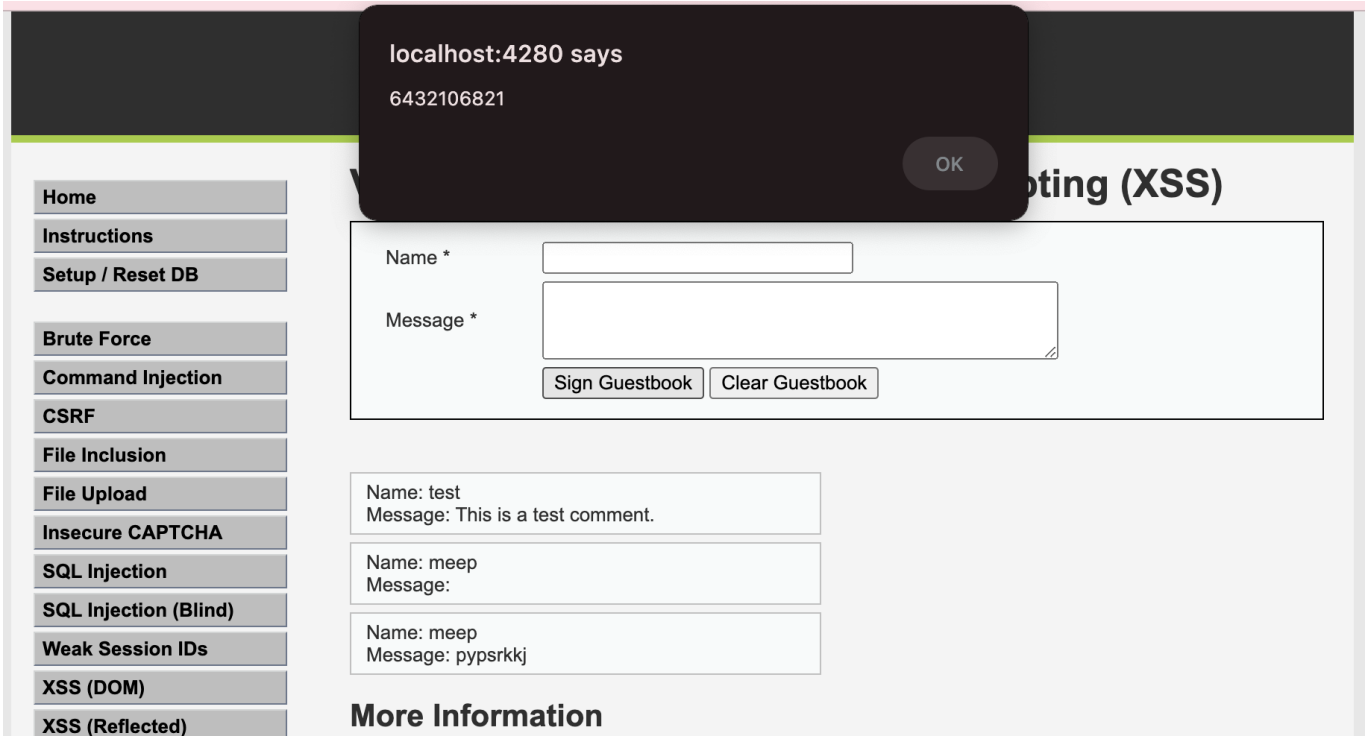
- **Description:**
  Reflected XSS occurs when a server includes unsanitized input in its response. Attackers often use reflected XSS to deliver payloads via malicious links.

- **Risk Score:**

Exploitability	Weakness Prevalence	Weakness Detectability	Technical Impacts
3	3	3	3

- **Fix/Mitigation Method:**
  - Input validation.
  - Use browser security headers like `X-XSS-Protection`.

10. Stored Cross Site Scripting (XSS)



- **URL Endpoint:**
  `http://localhost:4280/vulnerabilities/xss_s/`

- **Method:**
  Injected the script `<script>alert('6432106821')</script>` into a message field. The script was saved on the server and executed every time the page was loaded, displaying an alert box.

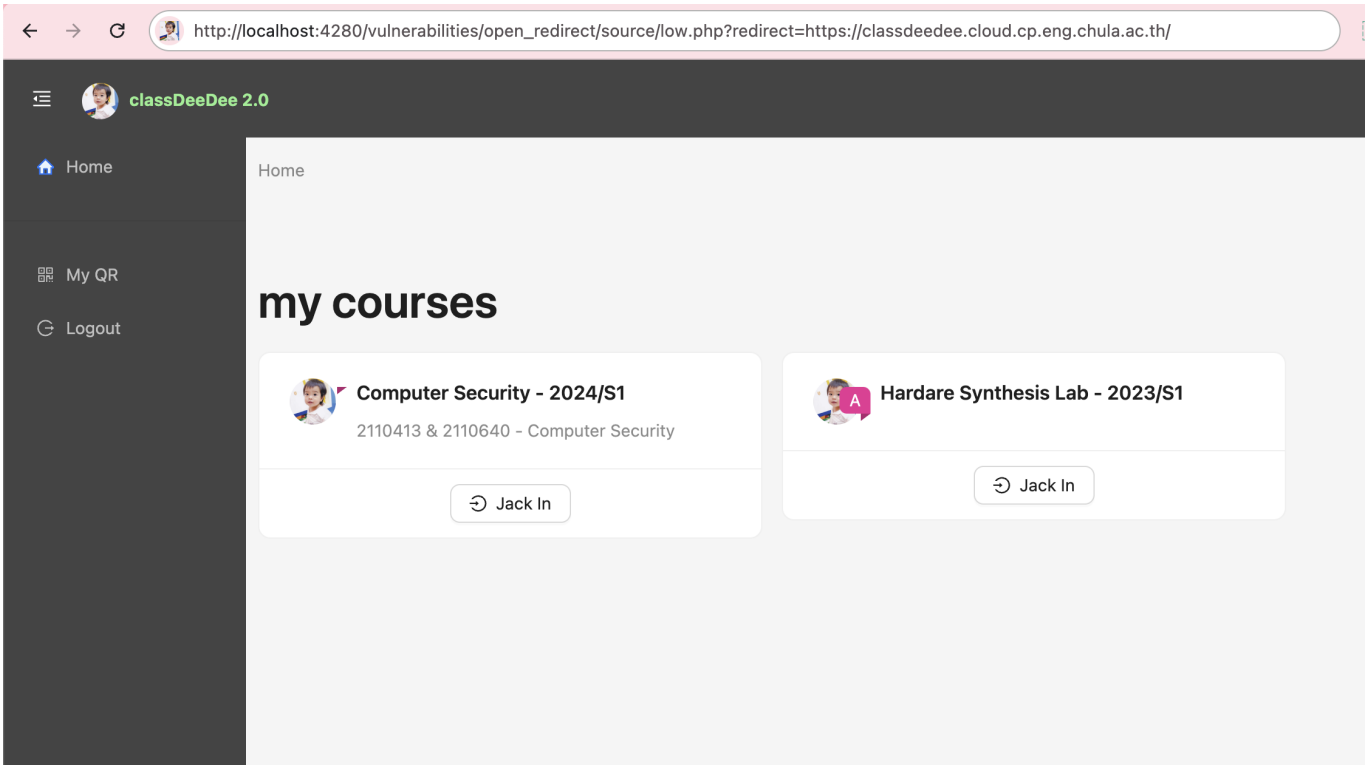
- **Description:**
  Stored XSS occurs when malicious scripts are saved on the server and executed whenever a user accesses the affected page, making it persistent and potentially more harmful.

- **Risk Score:**

Exploitability	Weakness Prevalence	Weakness Detectability	Technical Impacts
3	3	3	3

- **Fix/Mitigation Method:**
  - Validate inputs at both client and server sides.
  - Use HTTP-only cookies.

11. Open HTTP Redirect



- **URL Endpoint:**
  `http://localhost:4280/vulnerabilities/open_redirect/source/low.php?redirect=https://classdeedee.cloud.cp.eng.chula.ac.th/`

- **Method:**
  Modified the `redirect` parameter in the URL to point to an external site (`https://classdeedee.cloud.cp.eng.chula.ac.th/`). The application redirected users to this malicious site.

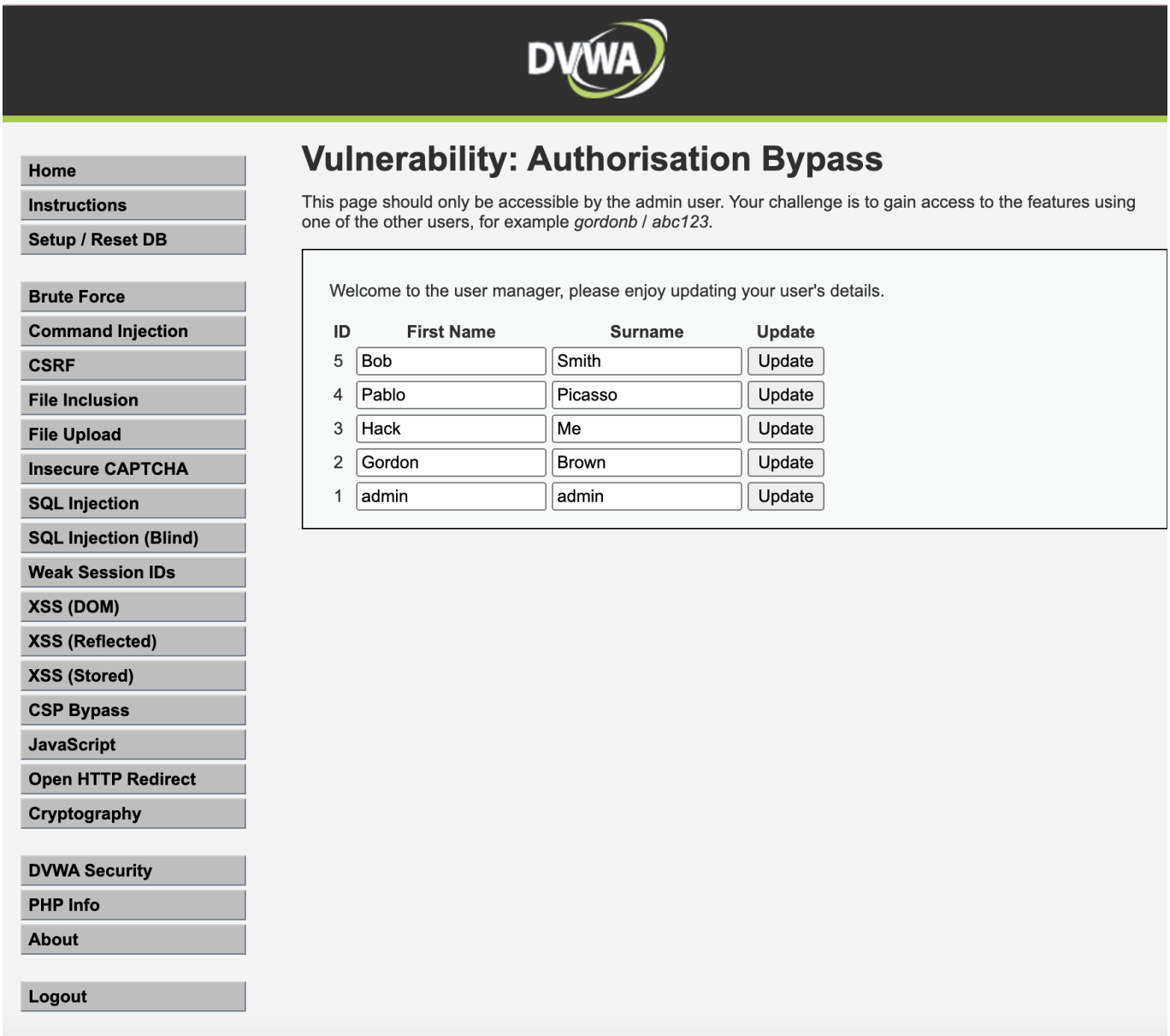
- **Description:**
  Open redirects allow attackers to manipulate URLs, tricking users into visiting malicious websites while believing they are accessing a trusted site.

- **Risk Score:**

Exploitability	Weakness Prevalence	Weakness Detectability	Technical Impacts
2	1	2	2

- **Fix/Mitigation Method:**
  - Validate redirect parameters to ensure they point only to trusted URLs.
  - Use built-in framework functions for secure redirection.
  - Limit user control over redirect parameters.

12. Authorization Bypass



- **URL Endpoint:**
  `http://localhost:4280/vulnerabilities/authbypass/`

- **Method:**
  Logged in as a regular user (`username = gordonb, password = abc123`). Manually navigated to the admin page (`/authbypass/`) by modifying the URL. The system failed to enforce access controls, granting admin privileges.

- **Description:**
  Authorization bypass occurs when access control mechanisms are improperly implemented, allowing unauthorized users to access restricted resources.

- **Risk Score:**

Exploitability	Weakness Prevalence	Weakness Detectability	Technical Impacts
1	2	2	2

- **Fix/Mitigation Method:**
    - Implement strict access control mechanisms using the Principle of Least Privilege (PoLP).
    - Verify user permissions before granting access to restricted resources.
    - Secure backend APIs with proper authentication and authorization checks.