**WEB APPLICATION PENTESTING**

The domain of the Project

Cybersecurity - Web Application Security

Under the guidance of

Mr. Nishchay Gaba (Penetration Tester)

By

Mr. G Sai Ujwal (B. Tech)

Period of the project

November 2024 to July 2025

SURE TRUST

PUTTAPARTHI, ANDHRA PRADESH

**DECLARATION**

The project titled **“*Web Application Pentesting*”** has been mentored by **Mr. Nishchay Gaba** and organized by SURE Trust from September 2024 to February 2025**.** This initiative aims to benefit educated unemployed rural youth by providing hands-on experience in industry-relevant projects, thereby enhancing employability.

I, **Mr. Gattu Sai Ujwal,** hereby declare that I have solely worked on this project under the guidance of my mentor. This project has significantly enhanced my practical knowledge and skills in the domain.

### **Name** **Signature**

Mr. Gattu Sai Ujwal



### Mr. Nishchay Gaba

### **Mentor** **Signature**

### 

### **Seal & Signature**

Prof.Radhakumari

Executive Director & Founder

SURE Trust

# **Table of Contents**

1. Disclaimer
2. Contact Information
3. Overview
4. Vulnerability Risk Ratings  
   4.1 Finding Severity Rating  
   4.2 Risk Factors (Likelihood & Security Implications)
5. Scope
6. Methodology
7. Executive Summary
8. Technical Findings  
   8.1 SQL Injection  
   8.2 Login Bypass via SQL Injection  
   8.3 Local File Inclusion (LFI)  
   8.4 Sensitive Information Disclosure  
   8.5 Credentials in HTTP Stream  
   8.6 End-of-Life (EOL) Software  
   8.7 Open Admin Page  
   8.8 Session ID in URL & Fixation  
   8.9 File Upload to Remote Shell  
   8.10 OS Command Injection  
   8.11 Remote File Inclusion (RFI)  
   8.12 Insecure Direct Object Reference (IDOR)  
   8.13 Improper Logout Functionality  
   8.14 Business Logic Flaw (Price Tampering)  
   8.15 Cross-Site Scripting (XSS)  
   8.16 Cross-Site Request Forgery (CSRF)  
   8.17 Misconfigured crossdomain.xml  
   8.18 No Rate Limiting  
   8.19 HTTP TRACE Enabled (XST)  
   8.20 Credentials Visible on Login Page  
   8.21 Server-Side Request Forgery (SSRF)  
   8.22 TLS/SSL Not Configured  
   8.23 HTML Injection
9. Conclusion
10. **Responsibility Statement:**

This *Web Application Penetration Testing Report* was prepared as part of an internship project at Sure Trust and is based on the assessment of the provided web applications. The findings, analysis, and recommendations are derived from the testing conducted within the approved scope.

While all efforts have been made to identify vulnerabilities accurately, this report does not guarantee the complete security of the tested systems. The author and Sure Trust are not responsible for any misuse, unauthorized actions, or unintended consequences arising from this report.

Use this report responsibly and implement necessary security measures as appropriate.

1. **Get in Touch:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Title** | **Contact Information** |
| G Sai Ujwal | Intern | gattuujwal037@gmail.com |

1. **Overview:**

This Web Application Penetration Testing report outlines the security assessment performed on the target application to identify vulnerabilities, assess risks, and provide remediation guidance. The testing followed standard methodologies including the **OWASP Top 10**, focusing on real-world attack vectors.

The engagement involved reconnaissance, scanning, vulnerability analysis, and manual exploitation. A total of **23 issues** were identified, including **critical vulnerabilities** such as SQL Injection, Command Injection, and SSRF, as well as several medium and low-risk findings.

Each finding includes a risk rating, technical details, and recommended fixes to help strengthen the application’s overall security posture.

1. **Security Concern Levels:**
   1. **Finding severity rating:**

Vulnerabilities in this report are classified into different severity levels based on their Security Implications and exploitability. The classification follows industry-standard frameworks such as CVSS (Common Vulnerability Scoring System) and OWASP risk rating methodology.

**4.2 Risk Factors & Likelihood**

The risk of each vulnerability was assessed based on:

* **Security Implications**: Potential damage to confidentiality, integrity, or availability.
* **Likelihood**: Ease of exploitation and attacker access level.
* **Exploitability**: Whether the issue can be exploited remotely, without authentication, or with minimal user interaction.

**Likelihood Levels**

* **High**: Easily exploitable, no authentication needed.
* **Medium**: Requires some conditions (e.g., valid session, user interaction).
* **Low**: Complex to exploit or requires privileged access.

Each issue’s final severity is rated using **CVSS v3.1**.

1. **Scope:**

Web application URL: <http://testphp.vulnweb.com/>

Virtual machines: Damn Vulnerable Web Application (DVWA) , Buggy Web application (BWapp)

• Authentication Mechanisms: Testing login and session management.

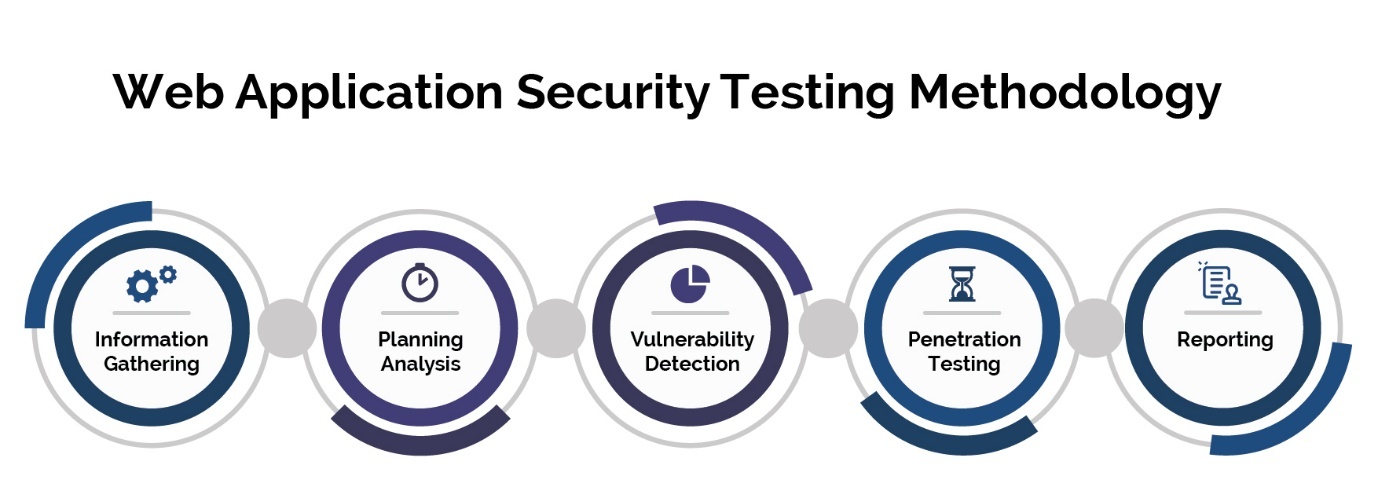
• User Input Validation: Assessing forms, parameters, and API requests for injection vulnerabilities.

• File Uploads and Downloads: Ensuring secure handling of uploaded files and preventing directory traversal attacks.

• Session Management: Evaluating session expiration, logout functionality, Session Fixation, Session Hijacking

• Injection Methods: SQL injection, HTML Injection, XSS scripting, OS command injection

• File Uploads and Downloads: Ensuring secure handling of uploaded files and preventing directory traversal attacks.

1. **Methodology:**

### **Information Gathering**

Identify publicly accessible data, technologies in use, and potential attack surfaces using tools like WHOIS, Shodan, and recon scripts.

### 2. **Planning & Analysis**

Define testing scope, objectives, and approach. Prioritize areas based on business logic and threat models.

### 3. **Vulnerability Detection**

Use automated scanners and manual analysis to identify known security flaws (e.g., OWASP Top 10).

### 4. **Penetration Testing**

Manually exploit discovered vulnerabilities to assess real-world Security Implications and confirm risk severity.

### 5. **Reporting**

Document findings with CVSS scores, proof-of-concept (Validation Snapshot), business Security Implications, and remediation steps.

**EXECUTIVE**

**SUMMARY**

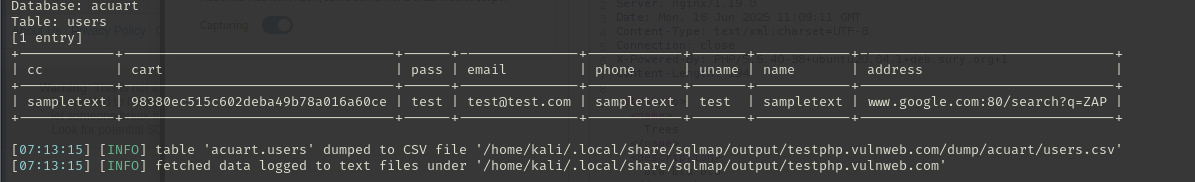
CRITICAL

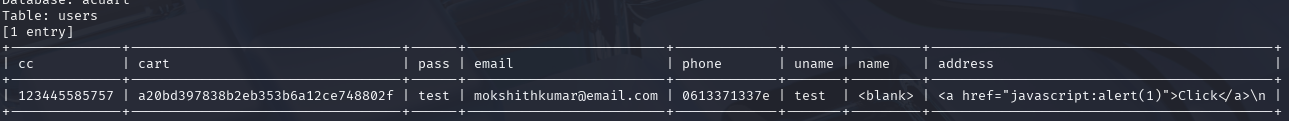
1. **Vulnerability Name: SQL INJECTION**

* **Security Insight:** SQL injection, also known as SQLI, is a common attack vector that uses malicious SQL code for backend database manipulation to access information that was not intended to be displayed. This information may include any number of items, including sensitive company data, user lists or private customer details.
* **CVSS:** 3.1/AV: N/AC: L/PR: N/UI: N/S:C/C:H/I:H/A:H
* **CWE-ID:** 
  + **CWE-89:** Improper Neutralization of Special Elements used in an SQL Command
  + **CVE-2023-39344**
* **Security Implications:** A successful SQL injection attack can result in unauthorized access to sensitive data, such as:
  + **Passwords**
  + **Credit card details**
  + **Personal user information**

SQL injection attacks have been used in many high-profile data breaches over the years. These have caused reputational damage and regulatory fines. In some cases, an attacker can obtain a persistent backdoor into an organizations system, leading to a long-term compromise that can go unnoticed for an extended period.

* **Security Recommendations:** 
  + we canprevent most instances of SQL injection using parameterized queries instead of string concatenation within the query. These parameterized queries are also known as "prepared statements".
  + Use trusted Object-Relational Mappers (like Sequelize, SQLAlchemy, Hibernate, Django ORM).
  + Reject unexpected input types, especially for IDs and numeric values.
  + Allow only expected values.
  + Use parameterized stored procedures.
  + Avoid dynamic SQL inside stored procedures.
* **Reference:** 
  + [**https://github.com/fobybus/social-media-skeleton/security/advisories/GHSA-857x-p6fq-mgfh**](https://github.com/fobybus/social-media-skeleton/security/advisories/GHSA-857x-p6fq-mgfh)
  + [**https://exchange.xforce.ibmcloud.com/vulnerabilities/15651**](https://exchange.xforce.ibmcloud.com/vulnerabilities/15651)
* **Impacted URLs:** 
  + http://testphp.vulnweb.com/userinfo.php
  + http://testphp.vulnweb.com/guestbook.php
  + http://testphp.vulnweb.com/AJAX/showxml.php
  + http://testphp.vulnweb.com/secured/newuser.php
  + http://testphp.vulnweb.com/artists.php?artist=3
  + http://testphp.vulnweb.com/listproducts.php?artist=3
  + http://testphp.vulnweb.com/comment.php?aid=3
  + http://testphp.vulnweb.com/listproducts.php?cat=1
  + http://testphp.vulnweb.com/hpp/?pp=12
  + http://testphp.vulnweb.com/showimage.php?file=
  + http://testphp.vulnweb.com/hpp/params.php?p=valid&pp=12
  + http://testphp.vulnweb.com/product.php?pic=6
  + http://testphp.vulnweb.com/showimage.php?file=./pictures/6.jpg&size=160
  + http://testphp.vulnweb.com/comment.php?pid=6
* **VALIDATION SNAPSHOT:**

****





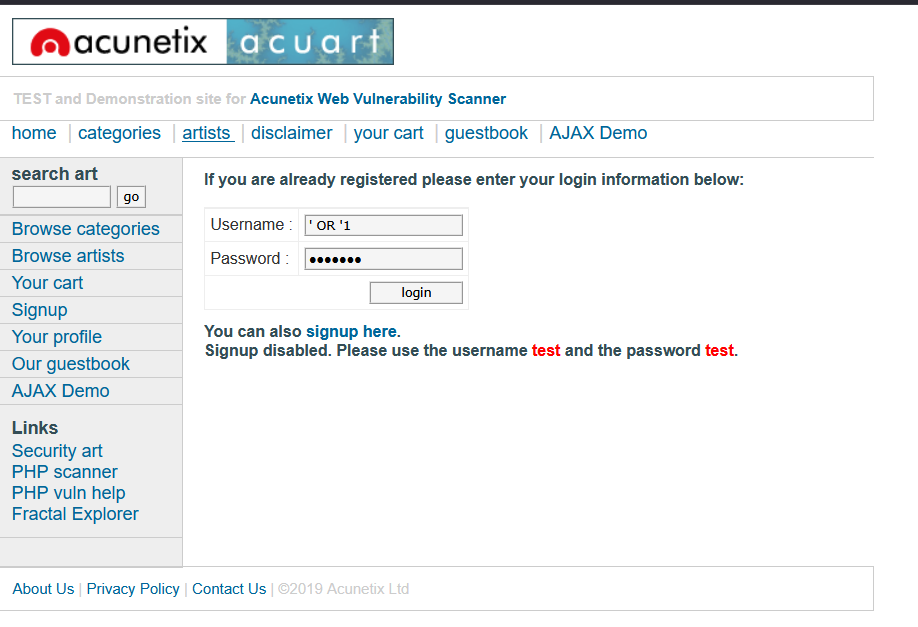
1. **Vulnerability Name: Login Bypass using SQL INJECTION**

* **Security Insight:** Login bypass via SQL Injection occurs when user input in a login form (usually the username or password field) is unsafely embedded in a SQL query without proper sanitization. This allows attackers to manipulate the SQL logic and authenticate without valid credentials.
* **CVSS:** 3.1/AV: N/AC: L/PR: N/UI: N/S:C/C:H/I:H/A:H
* **CWE-ID:** 
  + **CWE-89:** Improper Neutralization of Special Elements used in an SQL Command
  + **CVE-2017-8917, CVE-2019-11043**
* **Security Implications:**

|  |  |
| --- | --- |

* **Unauthorized Data Access:** Attackers can view, steal, modify, or delete sensitive data stored in the database, such as user information, financial records, or proprietary data.
* **Privilege Escalation:** Once inside, attackers might be able to escalate their privileges, gaining administrative rights and further control over the system.
* **System Compromise:** In some cases, attackers can use SQL injection to execute commands on the server, potentially leading to a full system compromise and the installation of malware or backdoors.
* **Reputation Damage:** A data breach or system compromise due to SQL injection can severely damage an organization's reputation, leading to loss of customer trust and potential legal ramifications.
* **Financial Loss:** The costs associated with investigating and remediating a breach, potential fines, legal fees, and lost business can be substantial.
* **Service Disruption:** Attackers could disrupt the availability of the application or service by altering or deleting critical data, or by crashing the database server.
* **Security Recommendations:**

* **Keep Software Updated:** Ensure that the database system, operating system, and web application framework are kept up-to-date with security patches**.**
* **Web Application Firewalls (WAFs):** A WAF can provide an additional layer of defense by inspecting incoming traffic and blocking requests that appear to contain malicious SQL injection payloads. While not a replacement for secure coding practices, a WAF can help mitigate known attack patterns.
* **Principle of Least Privilege**: The database user account used by the application should have only the necessary permissions to perform its functions. It should not have administrative privileges or the ability to modify database schema. This limits the potential damage an attacker can cause even if they manage to inject some SQL.
* **Reference:** 
  + [**https://developer.joomla.org/security-centre/692-20170501-core-sql-injection.html**](https://developer.joomla.org/security-centre/692-20170501-core-sql-injection.html)
  + [**https://www.exploit-db.com/exploits/42033/**](https://www.exploit-db.com/exploits/42033/)
* **Impacted URLs:**
  + [**http://testphp.vulnweb.com/admin/**](http://testphp.vulnweb.com/admin/)
* **VALIDATION SNAPSHOT:**

****

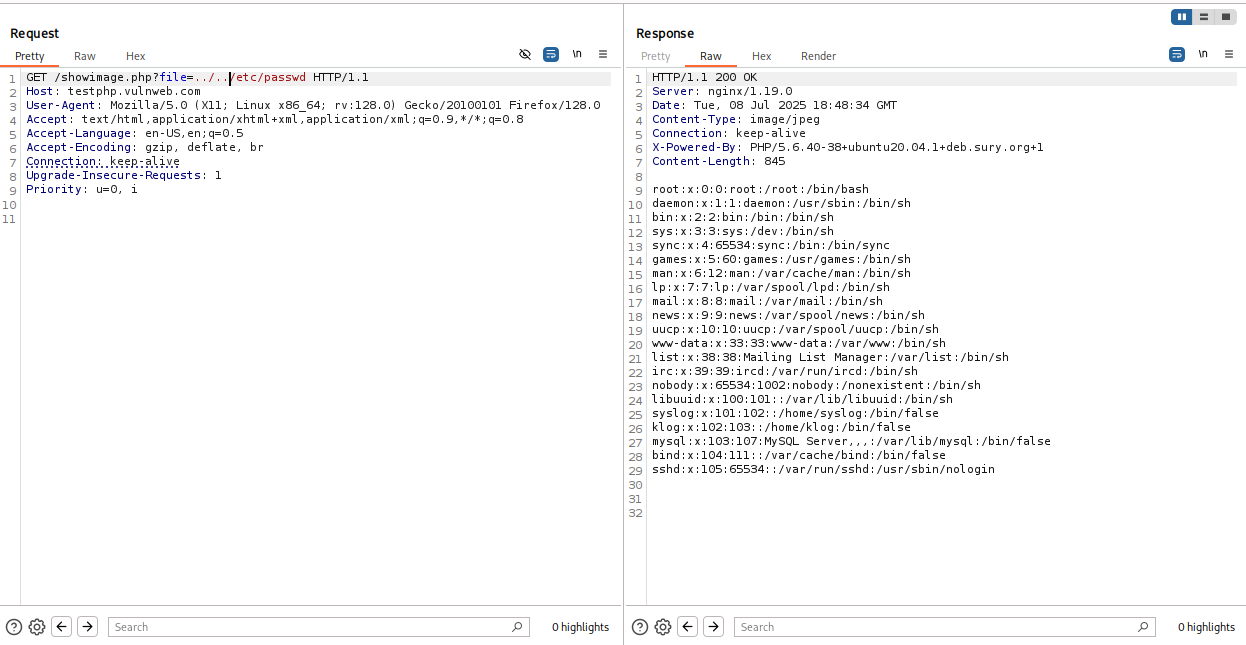
1. **Vulnerability Name: Local File Inclusion (LFI)**

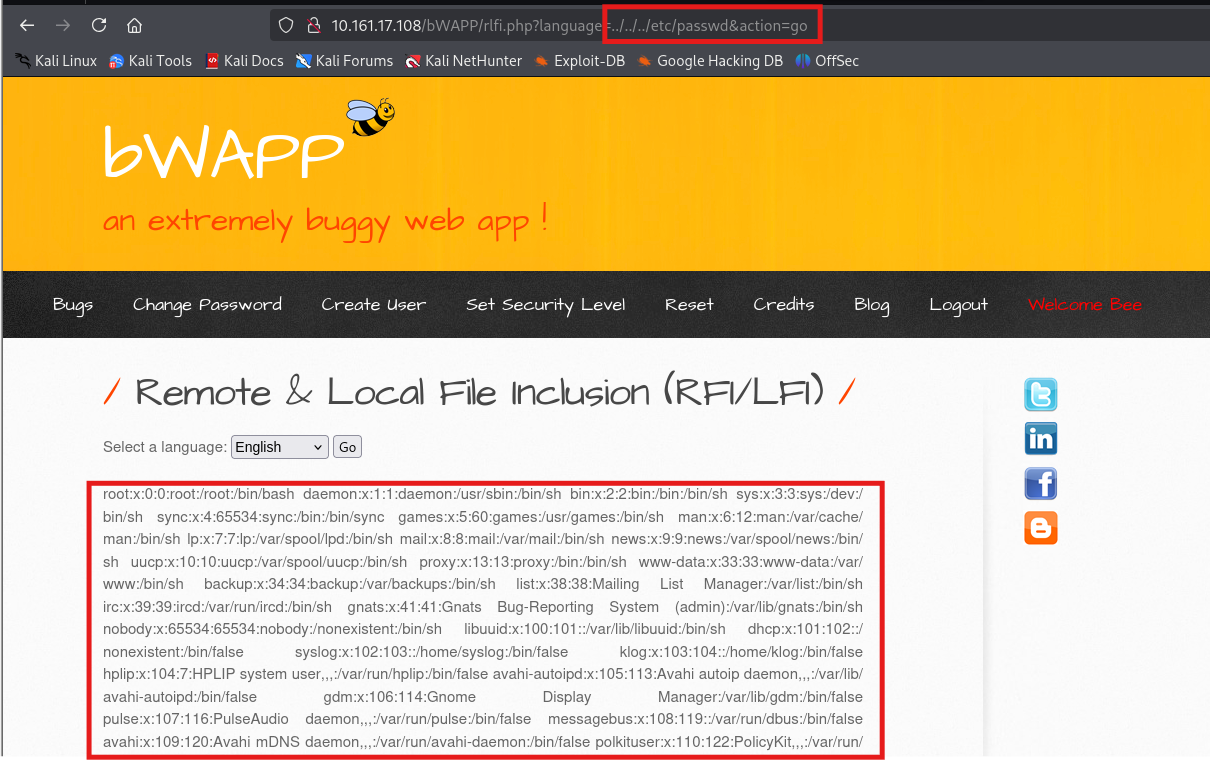
* **Security Insight: Local File Inclusion (LFI)** is a vulnerability where a web application allows user input to **specify and include files** from the local server's filesystem.
* **CVSS:** 3.1/AV: N/AC: L/PR: N/UI: N/S:C/C:H/I:H/A:H
* **CWE-ID:** 
  + **CWE-98:** Improper Control of Filename for Include
  + CVE-2016-10033, CVE-2020-11022
* **Security Implications:**

|  |  |
| --- | --- |

* **Information Disclosure:** Attackers can read sensitive files on the server, such as configuration files (containing database credentials, API keys), source code files, system files (like /etc/passwd on Linux), or log files.
* **Code Execution:** If combined with other vulnerabilities (like file uploads or directory traversal), LFI can sometimes be used to execute arbitrary code on the server. This is often achieved by including a file that an attacker has managed to upload containing malicious code or by including log files that the attacker has injected malicious code into via another vector.
* **Denial of Service (DoS**): Including large or critical system files repeatedly can potentially consume server resources and lead to a denial of service.
* **Access to Internal Systems:** By reading configuration files or other internal documents, attackers might gain information that allows them to access internal networks or systems**.**
* **Further Exploitation:** Information gathered through LFI can be used to identify other vulnerabilities or weaknesses in the application or server environment, facilitating further attacks.
* **Security Recommendations:**

* **Avoid Passing User-Supplied Input Directly to File Inclusion Functions**: The most effective mitigation is to avoid using user input (from URL parameters, form data, etc.) directly in file inclusion functions like include(), require(), include\_once(), or require\_once() in PHP, or similar functions in other languages.
* **Sanitization:** Remove or encode potentially malicious characters and path traversal sequences (../, ..\). However, relying solely on this is risky due to encoding variations and sophisticated bypass techniques**.**
* **Restrict File Permissions:** Ensure that sensitive files on the server have strict file permissions, preventing the web server process from reading them even if an LFI vulnerability exists.
* **Error Handling:** Configure the application to not display verbose error messages that might reveal file paths or internal information.
* **Impacted URLs:**
* [**http://testphp.vulnweb.com/showimage.php?file=./pictures/1.jpg**](http://testphp.vulnweb.com/showimage.php?file=./pictures/1.jpg)
* [**http://testphp.vulnweb.com/showimage.php?file=./pictures/2.jpg**](http://testphp.vulnweb.com/showimage.php?file=./pictures/2.jpg)
* [**http://testphp.vulnweb.com/showimage.php?file=./pictures/3.jpg**](http://testphp.vulnweb.com/showimage.php?file=./pictures/3.jpg)
* [**http://testphp.vulnweb.com/showimage.php?file=./pictures/4.jpg**](http://testphp.vulnweb.com/showimage.php?file=./pictures/4.jpg)
* [**http://testphp.vulnweb.com/showimage.php?file=./pictures/5.jpg**](http://testphp.vulnweb.com/showimage.php?file=./pictures/5.jpg)
* [**http://testphp.vulnweb.com/showimage.php?file=./pictures/6.jpg**](http://testphp.vulnweb.com/showimage.php?file=./pictures/6.jpg)
* [**http://testphp.vulnweb.com/showimage.php?file=./pictures/7.jpg**](http://testphp.vulnweb.com/showimage.php?file=./pictures/7.jpg)
* **VALIDATION SNAPSHOT:**

****



1. **Vulnerability Name: Sensitive Information Disclosure**

* **Security Insight:** The web server is configured to allow directory listing for the /pictures/ path, enabling unauthenticated users to view all files within that directory. Directory listing is a misconfiguration that exposes the internal file structure and may unintentionally reveal sensitive or critical files.
* **CVSS:** 3.1/AV: N/AC: L/PR: N/UI: N/S:C/C:H/I:H/A:H
* **CWE-ID:** 
  + CWE-54
  + CWE-200: Exposure of Sensitive Information
  + CWE-522: Insufficiently Protected Credentials
* **Security Implications:**

|  |  |
| --- | --- |

* **1.Complete Server Compromise (Code Execution)**

**System Commands**: From the web shell, the attacker can execute system commands, browse the server's file system, and manage processes. They effectively "own" the web server process.

**Web Shell Installation**: The most common payload is a "web shell," which is a script that gives the attacker a command-line interface to the server through their browser.

**System Commands**: From the web shell, the attacker can execute system commands, browse the server's file system, and manage processes. They effectively "own" the web server process.

**2. Data Theft and Information Disclosure**

Once an attacker has code execution, they can access and exfiltrate any data the web server process has permission to read**.**

* **Sensitive Files:**Reading configuration files (wp-config.php, env) to steal database credentials.
* **Application Source Code:**Stealing the entire application's source code to find other vulnerabilities or intellectual property.
* **Customer Data:**Accessing databases to steal user credentials, personal information, credit card numbers, etc.

**3. Pivoting and Lateral Movement**

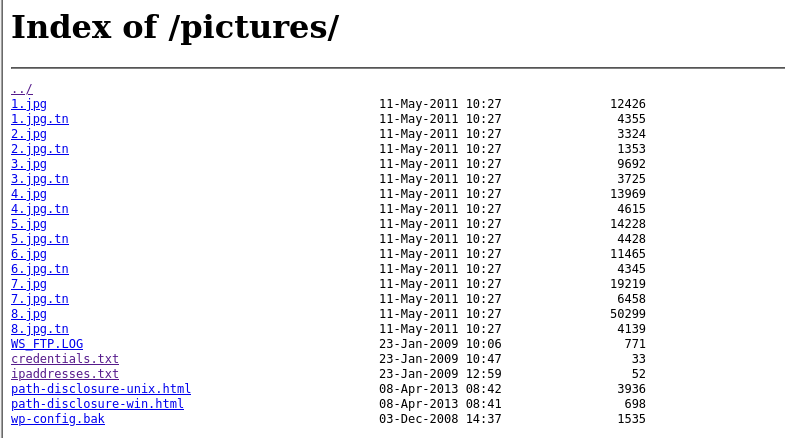
* The attacker can use the server to scan for and attack other servers (like database servers or internal admin panels) that are not exposed to the public internet**.**
* **Security Recommendations:**

Never trust user input. Even with remote inclusion disabled, a similar vulnerability called Local File Inclusion (LFI) can still be exploited.

* **Use a Whitelist**: The best approach is to not use the user's input directly as a filename. Instead, map the input to a pre-approved, hardcoded list of safe files.
* Avoid Using User Input in File Paths Altogether The most secure design is one where user input is never used to determine which file to include. Use a router or a switch-case statement based on user actions rather than direct file paths.
* **Principle of Least Privilege:** Run your web server and application with the minimum permissions necessary. If an RFI vulnerability is somehow exploited, this will limit the attacker's ability to read/write sensitive system files or access other parts of the system.
* **Impacted URLs:**

<http://10.161.17.108/bWAPP/rlfi.php?language=http://10.161.17.91:8000/php-reverse-shell.php&action=go>

* **VALIDATION SNAPSHOT:**

****

1. **Vulnerability Name: Username and Password Visible in HTTP stream**

* **Security Insight:** The application transmits **user credentials (username and password)** over an **unencrypted HTTP connection**. This exposes sensitive information to interception via **network sniffing**, especially over public or untrusted networks (e.g., open Wi-Fi or LAN).

During testing, credentials were found in **plaintext** within the HTTP request/response payload (e.g., in GET/POST parameters or HTTP Basic Auth headers), which violates secure transmission principles.

* **CVSS:** 3.1/AV: N/AC: L/PR: N/UI: N/S:C/C:H/I:H/A:H
* **CWE-ID:** 
  + CWE-319: Cleartext Transmission of Sensitive Information
  + CWE-522: Insufficiently Protected Credentials
  + CWE-598: Use of GET Method with Sensitive Data
* **Security Implications:**

|  |  |
| --- | --- |

* **1.Complete Server Compromise (Code Execution)**

This is the most direct and obvious Security Implications. An attacker who intercepts the traffic gets the exact username and password for the user's account on that specific website. They can then log in, impersonate the user, access private data, and perform any action the user could.

**Reputation Damage (for the Website Owner):**

**Loss of Trust:** Users who see their browser warn them that a page is "Not Secure" will lose trust in the service.

**Legal and Compliance Failures:** For services handling personal data, this can violate data protection regulations like GDPR or CCPA, leading to massive fines.**Brand Damage:** A publicized breach resulting from this vulnerability can permanently tarnish a company's reputation.

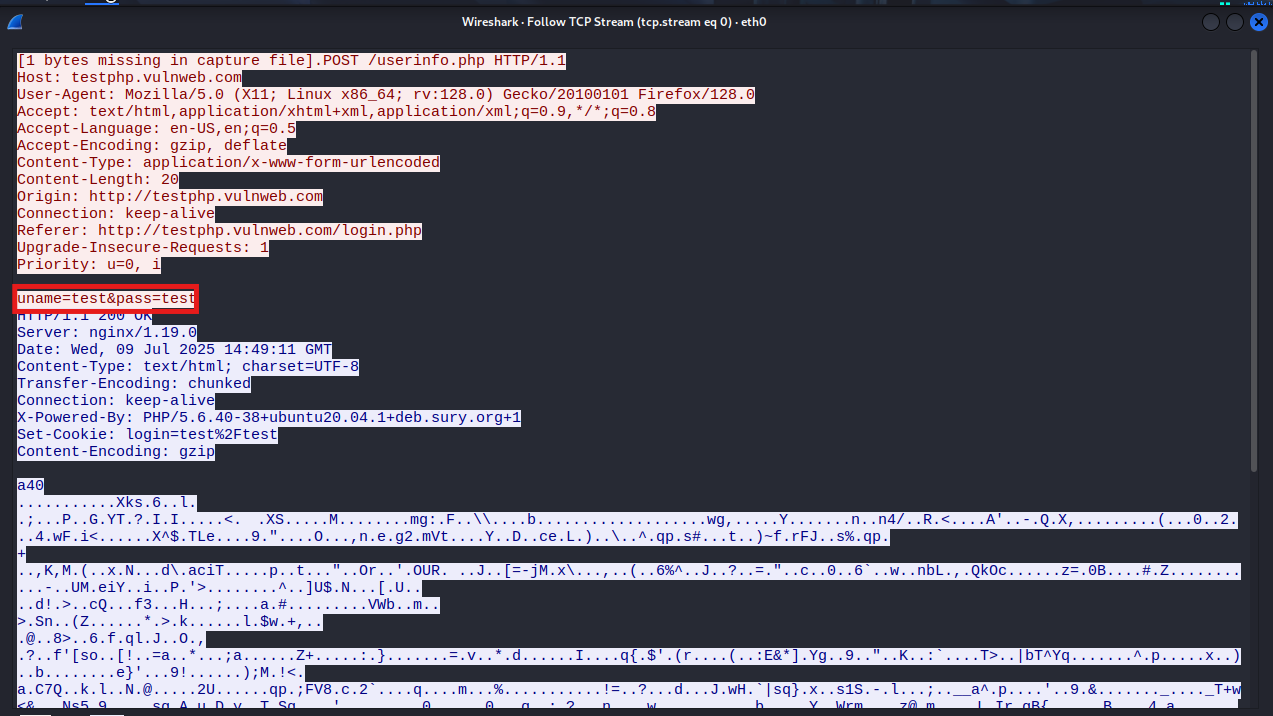
* **Security Recommendations:**

**Use HTTPS Everywhere (The Non-Negotiable Solution)**

* Action: Obtain a TLS/SSL certificate (many are free, e.g., via Let's Encrypt) and configure your web server to use HTTPS for *all* pages, not just the login page.
* Why it Works: HTTPS encrypts the *entire* session. The user's browser and the server establish a secure, private channel *before* any sensitive data like a username or password is sent.

**Enforce HTTPS with 301 Redirects**

* **Action:** Configure your server to automatically and permanently redirect any user who tries to access an http:// URL to the https:// version.
* **Why it Works:** This ensures that even if a user types the old address or clicks an old link, they are immediately moved to the secure version of the site before they can submit any data.
* **VALIDATION SNAPSHOT:**

****

1. **Vulnerability Name: EOF (End of Life)**

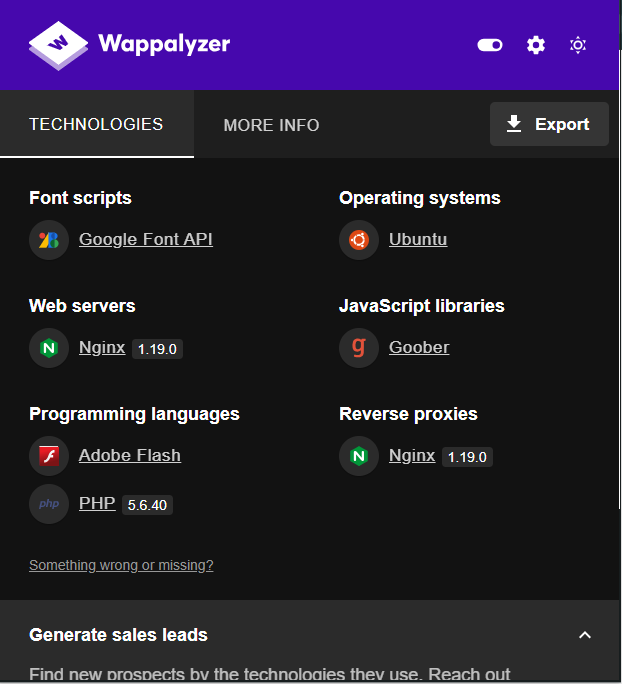
* **Security Insight: End-of-Life (EOL)** software refers to products that are **no longer supported** by their vendors. This includes:
  + No more security patches
  + No vulnerability fixes
  + No official updates or documentation
  + Running EOL software in production **introduces significant security risks**, as attackers may exploit **known unpatched vulnerabilities** that are public and widely weaponized.
* **CVSS:** 3.1/AV: N/AC: L/PR: N/UI: N/S:C/C:H/I:H/A:H
* **CWE-ID:** 
  + CWE-1104 - Use of Unmaintained Third-Party Components
  + CWE-1035 Configuration Management Errors
  + CWE-937 OWASP Top 10 2021: A06 – Vulnerable and Outdated Components
* **Security Implications:**
* **No Security Patches:** Vendors stop providing patches for newly discovered vulnerabilities.
* **Exposure to Known Exploits:** Public CVEs remain unpatched, increasing exploitability**.**
* **Higher Attack Surface:** Legacy software may have multiple unpatched vulnerabilities**.**
* **Compliance Violations:** Non-compliant with standards like **PCI-DSS, HIPAA, ISO 27001**.
* **May trigger audit failures or penalties.**
* **Operational Instability: EOL** software may lack compatibility with modern systems, tools, and protocols.
* **Integration Bottlenecks**: Difficult to integrate with modern secure APIs and toolchains**.**
* **Legal & Liability Exposure:** Breach incidents tied to unsupported software can result in legal action or customer lawsuits.
* **Security Recommendations:**

**Upgrade or Migrate**:Replace the EoL software/hardware with a currently supported version (e.g., upgrade from Windows Server 2012 to Windows Server 2022; upgrade from PHP 7.4 to a supported version). If a direct upgrade isn't possible, migrate the application or service to a new, supported platform.This is the only strategy that actually removes the vulnerability.

**Decommission or Retire:**If the system or application is no longer needed, take it offline completely. Shut down the server, uninstall the software, and remove it from the network.

**Air-Gapping:**

* + **Action:** Disconnect the system from all networks entirely. It can only be accessed by a physically present operator. Data is transferred via physical media like a USB drive (which itself must be scanned for malware).
  + **Use Case:** This is only practical for highly specialized systems, like industrial controllers or scientific equipment that cannot be upgraded and do not need network connectivity to function.
* **VALIDATION SNAPSHOT:**

****

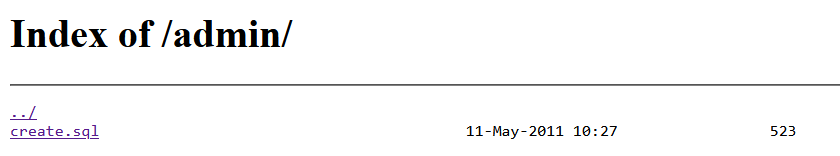
1. **Vulnerability Name: Open Admin Page**

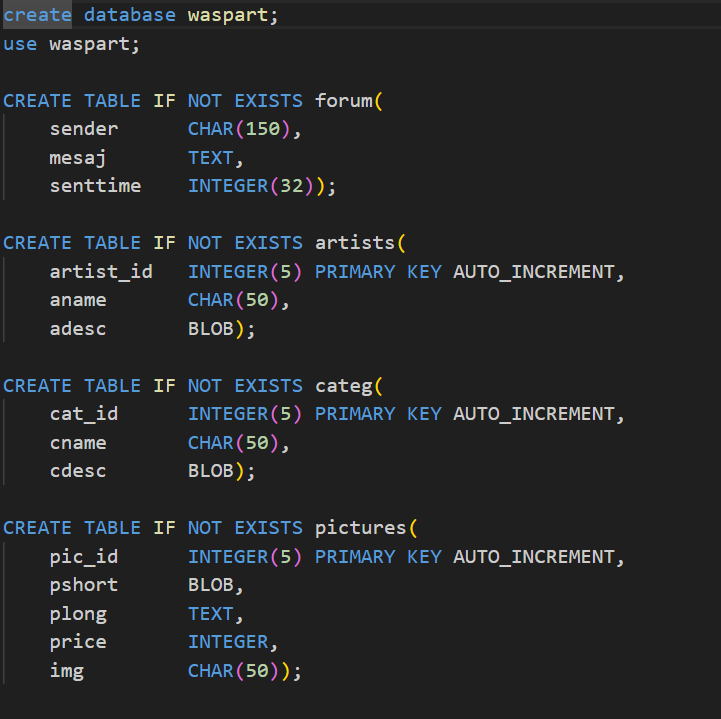
* **Security Insight**: An open admin page vulnerability occurs when administrative or backend panels (e.g., /admin, /wp-admin, /cpanel, /dashboard, etc.) are accessible without proper authentication or access controls. Attackers can:
  + **Access sensitive functions like user management, billing, and settings.**
  + **Attempt brute-force login.**
  + **Exploit unauthenticated vulnerabilities in the admin interface.**
* **CVSS:** 3.1/AV: N/AC: L/PR: N/UI: N/S:C/C:H/I:H/A:H
* **CWE-ID:** 
  + CWE-284 Improper Access Control
  + CWE-306 Missing Authentication for Critical Function
  + CWE-200 Exposure of Sensitive Information
  + CWE-732 Incorrect Permission Assignment for Critical Resource
* **Security Implications:**

**1. Complete Data Compromise**

* **Data Theft:**Attackers can view and exfiltrate all data managed by the application. This includes:
  + Personally Identifiable Information (PII): Customer names, addresses, phone numbers, email addresses.
  + Financial Data: Credit card details, transaction histories, pricing information.
  + Intellectual Property: Proprietary business logic, secret formulas, internal documents.
* **Data Manipulation**: An attacker can alter, corrupt, or modify any data.
  + Change user credentials, locking out legitimate users.
  + Modify product prices to $0 to buy them for free.
  + Alter financial records to commit fraud.
  + Change shipping addresses to reroute products.
* **Data Deletion:**An attacker can delete entire databases, user accounts, or critical records, causing irreparable damage and business disruption.

**2. Full System and Infrastructure Compromise**

* **Server Takeover:**Many admin panels have features to upload files (e.g., images, plugins, themes). An attacker can upload a malicious script (a "web shell") to gain command-line access to the server itself.
* **Malware and Ransomware Deployment:**Once they control the server, they can install malware to attack users or encrypt all server files and demand a ransom (ransomware).
* **Resource Hijacking:**
  + **Becoming a Botnet Zombie:**Using your server to participate in Distributed Denial-of-Service (DDoS) attacks against other targets.
* **Security Recommendations:**
* **Implement Mandatory Authentication (The Core Fix)**Immediately place a login form in front of the admin page. No one should be able to see or interact with any admin functionality without first authenticating.
* Use a well-vetted authentication library or framework feature (e.g., Spring Security, Django Auth, Devise for Rails). Do not try to write your own authentication from scratch.
* **Enforce Strong Password Policies**Ensure that all admin accounts are protected by strong, unique passwords. Minimum Length: Require at least 12-16 characters.Complexity: Require a mix of uppercase letters, lowercase letters, numbers, and symbols.Password Hashing: Store passwords using a modern, slow hashing algorithm like Argon2 or bcrypt with a salt. Never store passwords in plain text**.**
* **Implement Multi-Factor Authentication (MFA/2FA)**Require a second form of verification in addition to a password. This is one of the single most effective security controls.Use authenticator apps (like Google Authenticator or Authy), hardware tokens (like a YubiKey), or SMS codes (less secure but better than nothing).
* **VALIDATION SNAPSHOT:**



1. **Vulnerability Name: Session ID Visible in URL & Session Fixation/Prediction**

* **Security Insight**: The application includes the **session identifier (PHPSESSID, JSESSIONID, etc.) in the URL** (e.g., https://example.com/dashboard?sid=abcd1234). An attacker can:
  + View, share, or intercept session IDs through browser history, logs, or referrers.
  + Modify the session ID in the URL and gain **unauthorized access** to another user's account if session validation is weak.
  + In this case, you were able to **manually change the session ID** and gain access to another user’s session, confirming a **Session Fixation/Session Prediction vulnerability**.
* **CVSS:** 3.1/AV: N/AC: L/PR: N/UI: N/S:C/C:H/I:H/A:H
* **CWE-ID:** 
  + CWE-384 Session Fixation
  + CWE-598 Exposure of Session ID in URL
  + CWE-640 Weak Session Management
* **Security Implications:**
  + When a session ID is in the URL (e.g.,

https://example.com/account?session\_id=aBcDeFg12345),

it becomes highly exposed and can be easily stolen through several vectors:

**Referer Headers:**

* + When a user clicks a link on your site that goes to a different domain (e.g., a link to a partner site or an ad), the browser sends the full URL of the page they came from in the Referer HTTP header.
  + The external site receives the user's session ID, and its administrators (or anyone who can view their logs) can use it to hijack the session on your site.
  + **Browser History:**

The URL, including the session ID, is stored in the user's local browser history.

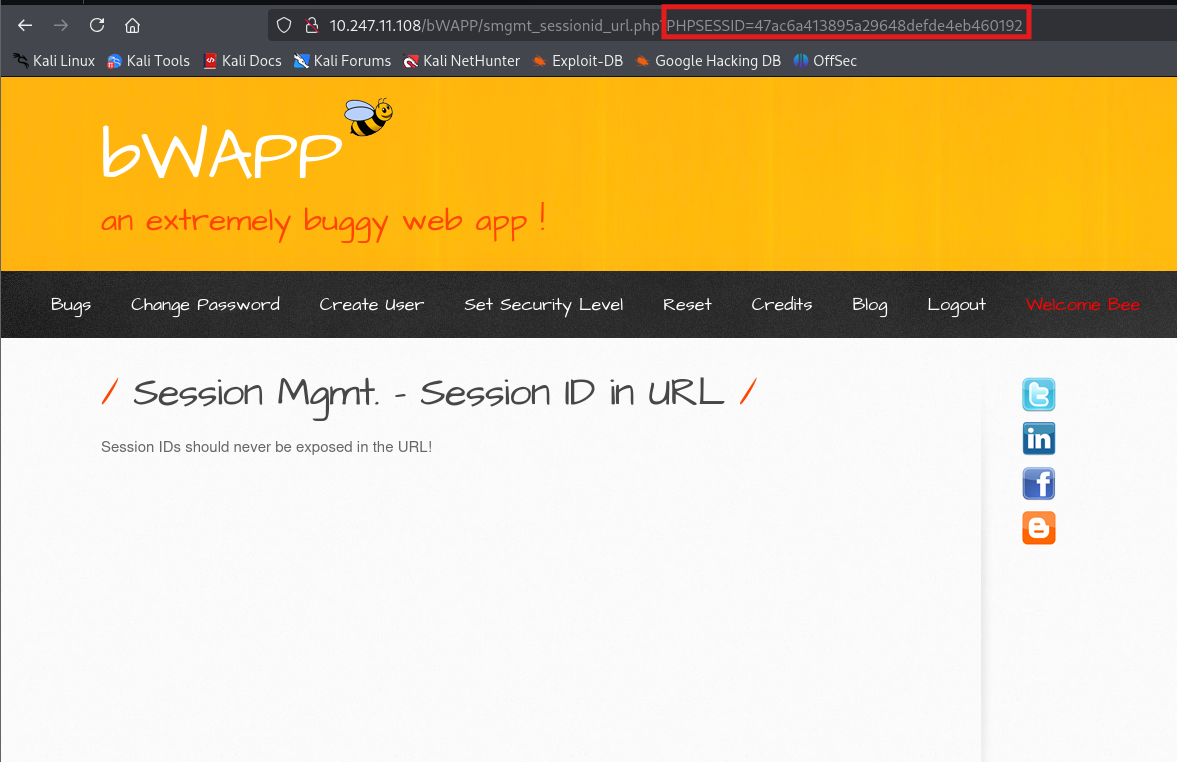
**Security Implications:** Anyone with physical access to the user's computer (a family member, a coworker, or a thief) can simply look through the browser history, copy the URL, and gain full access to the user's account.

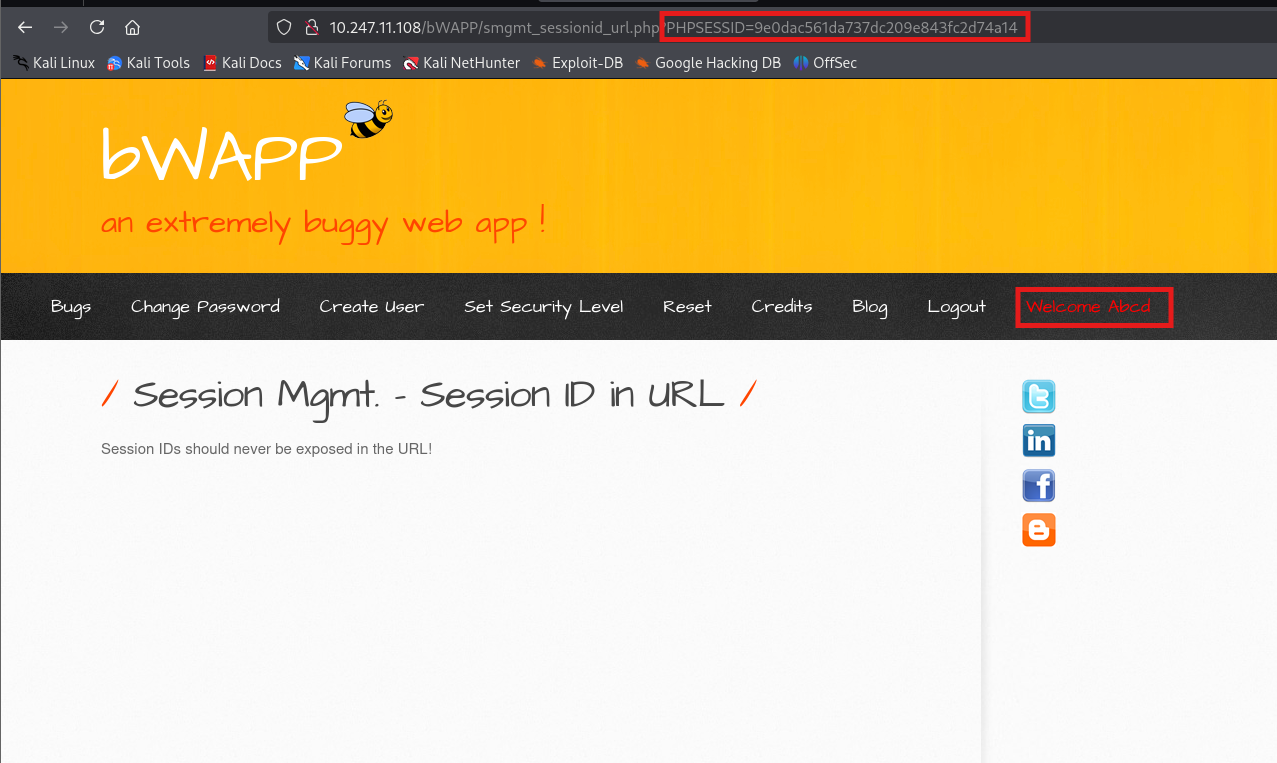
* **Attacker obtains a valid session ID**: The attacker visits the website's login page and the server assigns them a session ID (e.g., SID\_known\_by\_attacker).
* Attacker "fixes" the ID onto the victim's browser: The attacker tricks the victim into using this ID. A common way is to send them a crafted link:
* [**https://example.com/login?session\_id=SID\_known\_by\_attacker**](https://example.com/login?session_id=SID_known_by_attacker)**.**
* Victim logs in: The victim clicks the link, lands on the login page, and enters their username and password. The server sees the session\_id from the URL, accepts it, and ties the victim's

newly authenticated session to SID\_known\_by\_attacker.

**Attacker takes over:** The attacker, who already has SID\_known\_by\_attacker, can now refresh their browser and will be logged in as the victim**.**

* **Security Recommendations:**
* Use Cookies for Session IDs, Not URLs (Mitigates URL Exposure)
* **Set Secure Cookie Flags (Defense-in-Depth for Cookies) :Even when using cookies, you must secure them.HttpOnly Flag:** Prevents the cookie from being accessed by client-side scripts (JavaScript). This is a crucial defense against session theft via Cross-Site Scripting (XSS).**Secure Flag:**Ensures the cookie is only sent over an encrypted HTTPS connection. This prevents it from being intercepted in a man-in-the-middle (MITM) attack on an unsecured Wi-Fi network.**SameSite Flag**: Set to Lax or Strict to protect against SameSite Flag: Set to Lax or Strict to protect against Cross-Site Request Forgery (CSRF), where an attacker tricks a user into making an unintended request on your site.
* Implement Session Timeouts
* Regenerate the Session ID on Authentication (Mitigates Session Fixation)
* **VALIDATION SNAPSHOT:**

****



1. **Vulnerability Name: Session ID Visible in URL & Session Fixation/Prediction**

* **Security Insight** A file upload vulnerability occurs when an application allows users to upload files without properly validating the file type, content, or destination. If an attacker is able to upload a **malicious script** (e.g., .php, .jsp, .asp) and execute it on the server, it leads to **Remote Code Execution (RCE)**.
* **CVSS:** 3.1/AV: N/AC: L/PR: N/UI: N/S:C/C:H/I:H/A:H
* **CWE-ID:** 
  + CWE-434 Unrestricted Upload of File with Dangerous Type
  + CWE-22 Path Traversal (if used in upload path)
  + CVE-2019-6340 RCE via file upload in Drupal Core
* **Security Implications:**
* Full command execution on server, leading to full takeover.
* Persistent backdoor on the server via web shell.
* Use compromised server to pivot deeper into internal network.
* Access sensitive files, credentials, or databases.
* Upload of cron jobs or malware for continuous access.
* Defacement or Destruction Replace web pages or delete site content.
* **Security Recommendations**:

1. Restrict File Types Strictly

* + Allow only necessary file extensions (e.g., .jpg, .png, .pdf).
  + Validate both file extension and MIME type server-side.

2. Deep File Inspection

* + Check file magic bytes (e.g., using file command or libraries like mime\_content\_type().

Don't rely on client-side JavaScript alone.

3. Rename and Store Files Securely

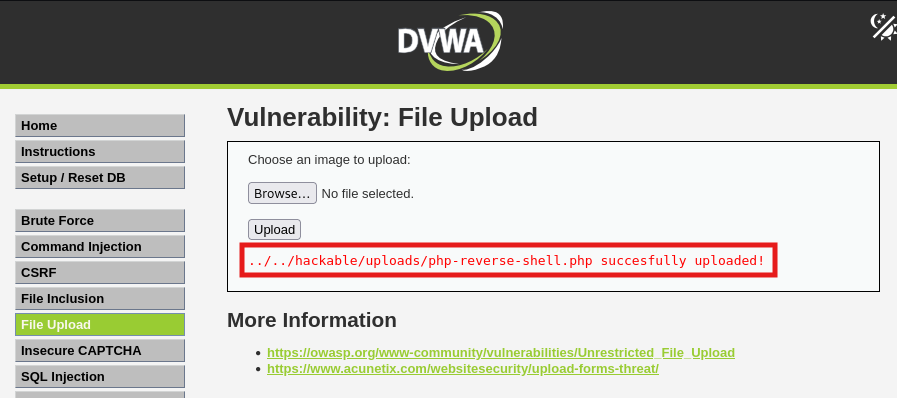
* + Rename uploaded files to random names (e.g., UUID).
  + Store them outside the web root (/var/uploads/, not /var/www/html/uploads/).

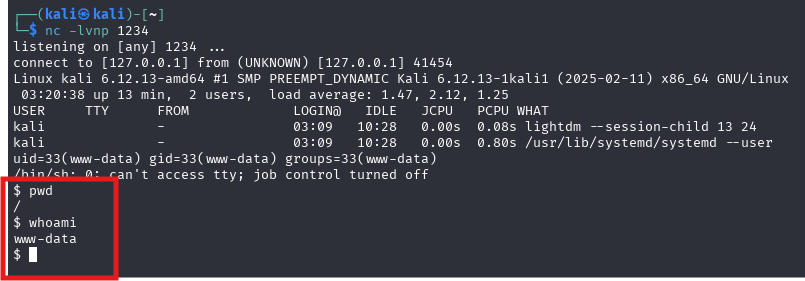
4. Disallow Executable Scripts

* + Prevent .php, .asp, .jsp, .exe, etc., from being uploaded or executed.

5. Web Application Firewall (WAF)

* + Use WAFs like ModSecurity to detect/stop suspicious uploads.
* **VALIDATION SNAPSHOT:**

****

****

1. **Vulnerability Name: OS command injection**

* **Security Insight** **OS Command Injection** occurs when untrusted input is directly passed to a system shell or command interpreter.

An attacker can manipulate the input to **execute arbitrary operating system commands**.

* **CVSS:** 3.1/AV: N/AC: L/PR: N/UI: N/S:C/C:H/I:H/A:H
* **CWE-ID:**
* CWE-78 Improper Neutralization of OS Command
* CVE-2014-6271 Shellshock – Classic OS Command Injection via Bash
* CVE-2019-11043 PHP-FPM remote code execution via crafted query string
* **Security Implications:**
  + Attacker can execute arbitrary commands like file deletion, system access, persistence
  + Can read system files (/etc/passwd, .env, config) or application source code
  + May lead to root if combined with misconfigurations or vulnerable binaries
  + Use compromised server as pivot to attack internal systems
  + Delete or overwrite files, bring down services, or deface web content
* **Security Recommendations**:

1. Avoid System Calls Altogether

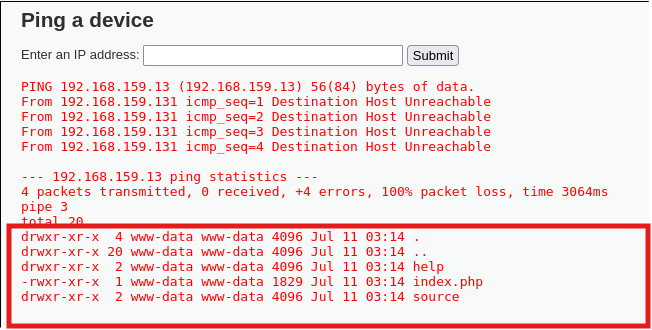
* + Use native language functions instead of system calls.
  + Example: Use scandir() in PHP instead of system('ls ...').

2. Input Validation and Sanitization

* + Strictly validate input against a whitelist (e.g., allowed file names or IDs).
  + Reject unexpected or malformed input (e.g.,;, &&, |).

3. Use Escaping Functions

* + If system commands must be used, use proper escaping:
  + In PHP: escapeshellarg () or escapeshellcmd ()
  + In Python: subprocess.run ([...], shell=False)
  + In Node.js: child\_process. execFile () instead of exec ()
* **VALIDATION SNAPSHOT:**

****

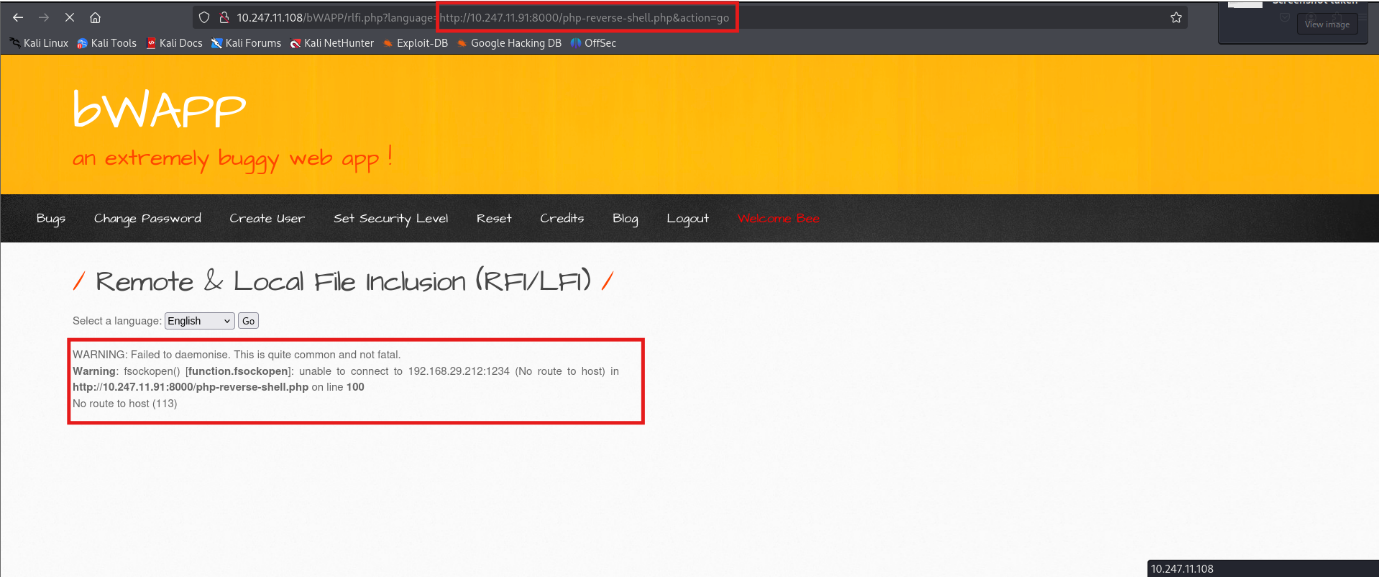
HIGH

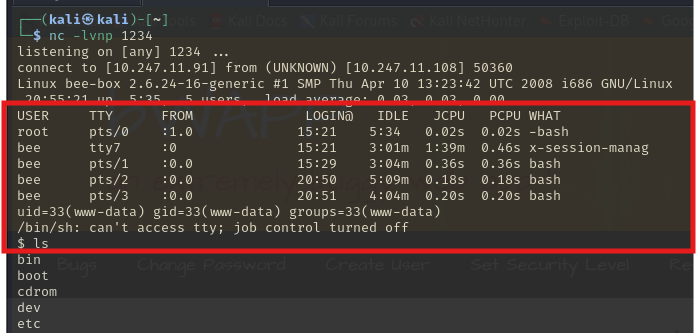
1. **Vulnerability Name: Remote File Inclusion (RFI)**

* **Security Insight Remote File Inclusion (RFI)** is a vulnerability that allows an attacker to include a remote file (typically a malicious script) through the web application. It usually occurs in applications written in PHP where user input is used directly in file-related functions like include (), require (), etc., without proper sanitization or validation.
* **CVSS:** AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
* **CWE-ID:** 
  + CWE-94 Improper Control of Generation of Code ('Code Injection'
  + CWE-98 Improper Control of Filename for Include/Require
  + CWE-20 Improper Input Validation
* **Security Implications:**
* Complete Server Compromise (Remote Code Execution - RCE)
* Sensitive Data Exposure and Theft
* **Server as an Attack Platform**

**The compromised server becomes a powerful tool for the attacker, who can use its resources to:**

* Launch DDoS Attacks: Use the server's bandwidth to attack other targets.
* Send Spam/Phishing Emails: Use the server's reputation and mail services to conduct spam or phishing campaigns.
* Pivot to Internal Networks: If the web server is in a corporate network, attackers can use it as a foothold to attack other internal, more sensitive systems that are not exposed to the internet.
* Website Defacement and Reputational Damage
* **Security Recommendations:**
* **Never Trust User Input for File Paths (The Golden Rule)**The root cause of RFI is trusting user-supplied input to determine which file to include. Never use user input directly in file inclusion functions (include, require, include\_once, require\_once in PHP).
* **Use an Allow-list (Whitelisting)**If you must dynamically include files based on a parameter, use a strict allow-list of acceptable file names. This is the most effective mitigation.
* **Disable Remote File Inclusion in Server Configuration (Crucial)**For PHP, the php.ini configuration file has directives that can disable RFI at the server level. This is a critical safety net.
* **VALIDATION SNAPSHOT:**





1. **Vulnerability Name: Insecure Direct Object Response (IDOR)**

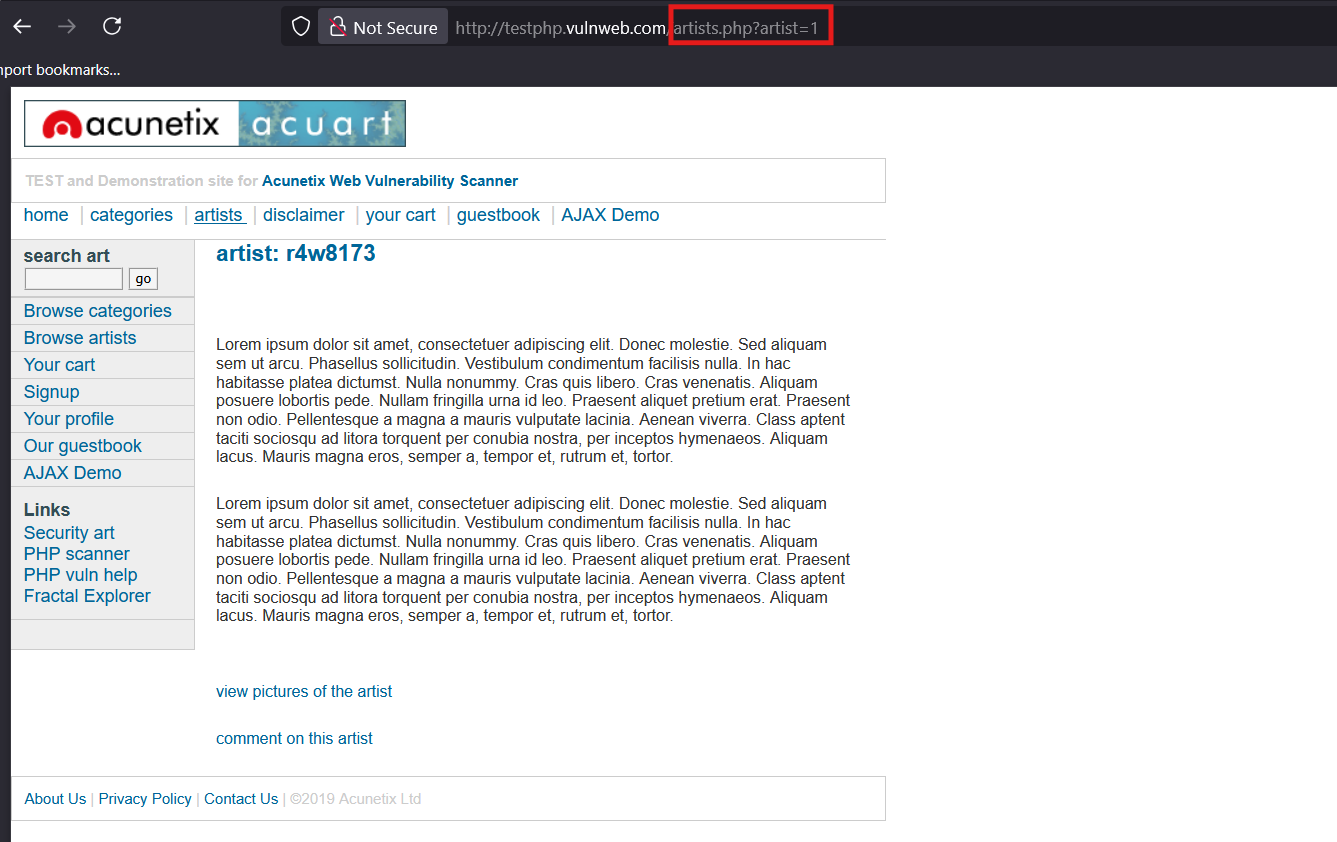
* **Security Insight:** Insecure Direct Object Reference (IDOR) is an access control vulnerability that occurs when an application exposes a reference to internal implementation objects, such as database keys, filenames, user IDs, or other internal data, and fails to validate the user's authorization to access them.
* **CVSS:** 7.5
* **CWE-ID:** 
  + CWE-639 Authorization Bypass Through User-Controlled Key
  + CWE-862 Missing Authorization
  + CWE-285 Improper Authorization
* **Security Implications:**

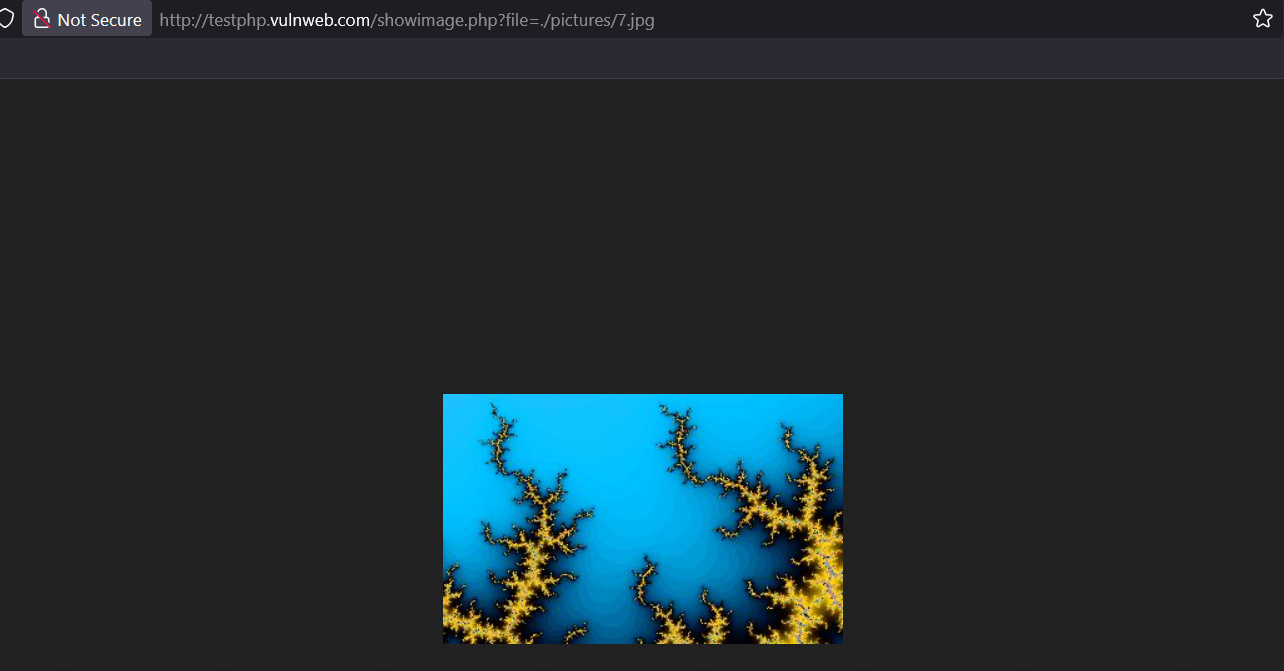
**Unauthorized Data Disclosure (Confidentiality Breach)**

* **Personally Identifiable Information (PII):** Names, addresses, phone numbers, email addresses.
* **Financial Information**: Bank account details, transaction histories, credit card numbers.
* **Private Communications**: Direct messages, support tickets, internal
* **Health Information:** Medical records, prescriptions (a major HIPAA violation).

**Unauthorized Data Modification (Integrity Breach)**

Attackers can alter or tamper with data that isn't theirs. This can lead to:

* **Account Takeover:** By changing another user's password, email, or security questions.
* **Data Manipulation:** Modifying order details, changing shipping addresses, or deleting records.
* **Privilege Escalation:** If an attacker can modify their own user role from "user" to "admin".
* **Security Recommendations:**
* **Implement Robust Access Control Checks (The Primary Defense)**
* This is the most important and effective mitigation. For every single request that accesses a private object, the server-side code must verify that the currently authenticated user has the necessary permissions to perform the requested action on that specific object.
* **Avoid Direct, Predictable References (Defense-in-Depth)**While not a substitute for access control, using less predictable identifiers can make it much harder for attackers to guess or enumerate the IDs of other objects. This is a form of "security by obscurity."
* **VALIDATION SNAPSHOT:**



**13. Vulnerability Name: Improper Logout Functionality**

* **Security Insight:** The application fails to properly **invalidate the session on logout**, allowing users to regain access to **authenticated content** by:
  + Clicking the browser **Back** button
  + Accessing cached pages (no-cache headers missing)
  + Session ID still being valid on the server
  + This violates standard session handling practices and can lead to **unauthorized access after logout**, especially on shared or public systems.
* **CVSS:** 7.5
* **CWE-ID:** 
  + CWE-613 Insufficient Session Expiration
  + CWE-306 Missing Authentication for Critical Function
  + CWE-525 Use of Web Browser Cache Containing Sensitive Information
* **Security Implications:**

**Prolonged Exposure Window:**

The core problem is that the session's "time-to-live" is not cut short by the logout action. If a session is configured to last for 8 hours, it remains valid for the full 8 hours, even if the user logged out after 5 minutes. This gives an attacker a much larger window of opportunity to find and exploit a stolen session token.

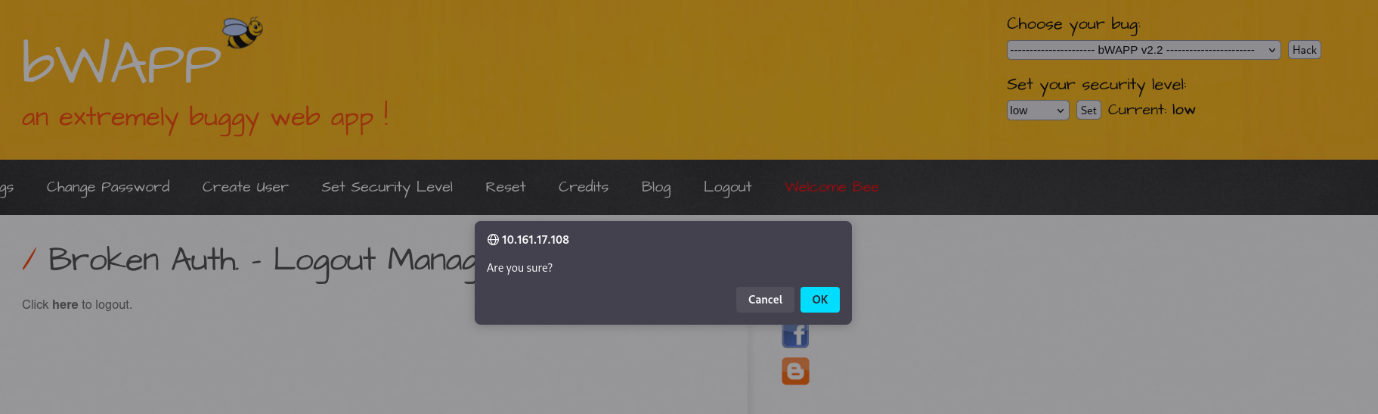
**Account Takeover via Stolen Session Cookies**

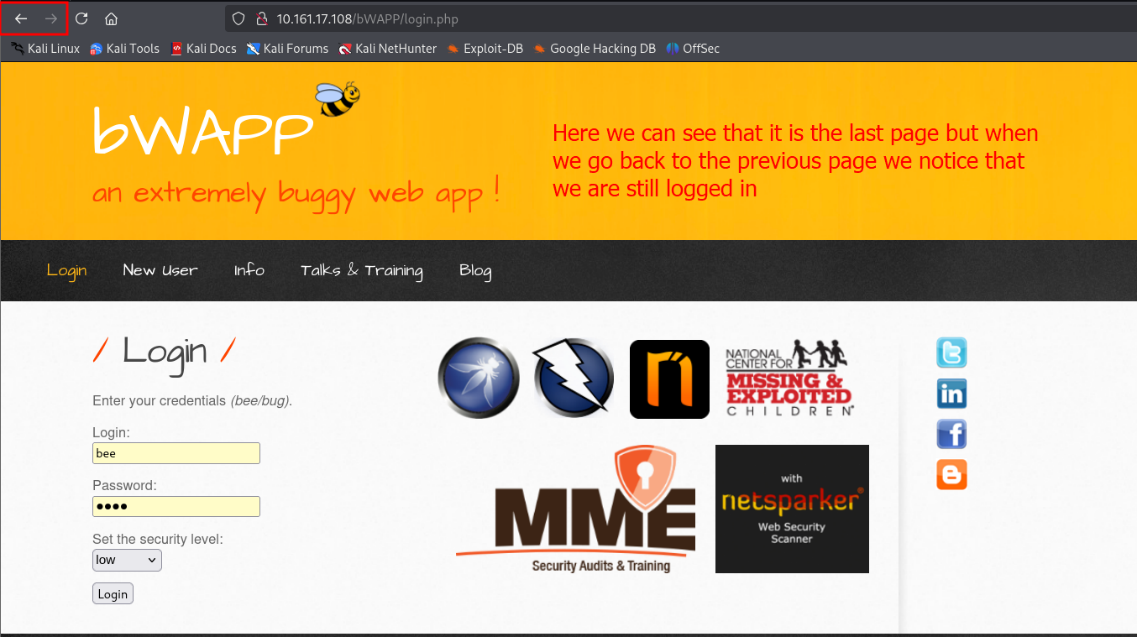
An active, non-invalidated session token is a valuable target for attackers.

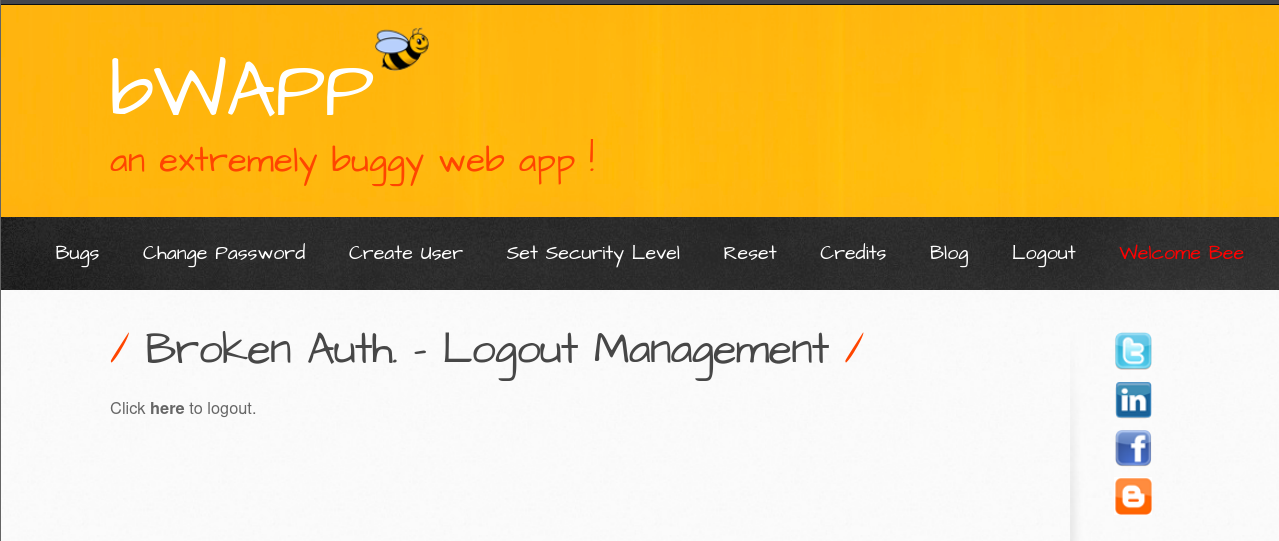
**Account Takeover via Shared or Public Computers**

This is the most classic and realistic attack scenario.

* **Security Recommendations:**
* **Always Invalidate the Session on the Server-Side (The Golden Rule)**This is the single most important mitigation. When a user clicks the logout button, the server-side code must explicitly destroy the session.
* **Clear Session Cookies on the Client-Side (Good Hygiene)**After invalidating the session on the server, you should also instruct the browser to clear the session cookie. This prevents the cookie from being resent in subsequent requests and removes it as a target for theft from the browser's storage.
* **VALIDATION SNAPSHOT:**





****

**14. Vulnerability Name: Business Logic Flaw (BLF)**

* **Security Insight**: A business logic vulnerability exists when an attacker is able to manipulate the price of a product during the "add to cart" or checkout process — by intercepting/modifying client-side data (e.g., using a proxy like Burp Suite) or crafting custom requests.
* **CVSS:** 8.1
* **CWE-ID:** 
  + CWE-640 Weak Password Recovery Mechanism for Forgotten Password
  + CWE-345 Insufficient Verification of Data Authenticity
  + CWE-441 Unintended Proxy or Intermediary (Confused Deputy)
  + CWE-472 External Control of Assumed-Immutable Web Parameter
  + CWE-285 Improper Authorization
  + CWE-840 Business Logic Errors
* **Security Implications:**

**Financial Logic Flaws**

These abuse the rules related to pricing, payments, and transactions.

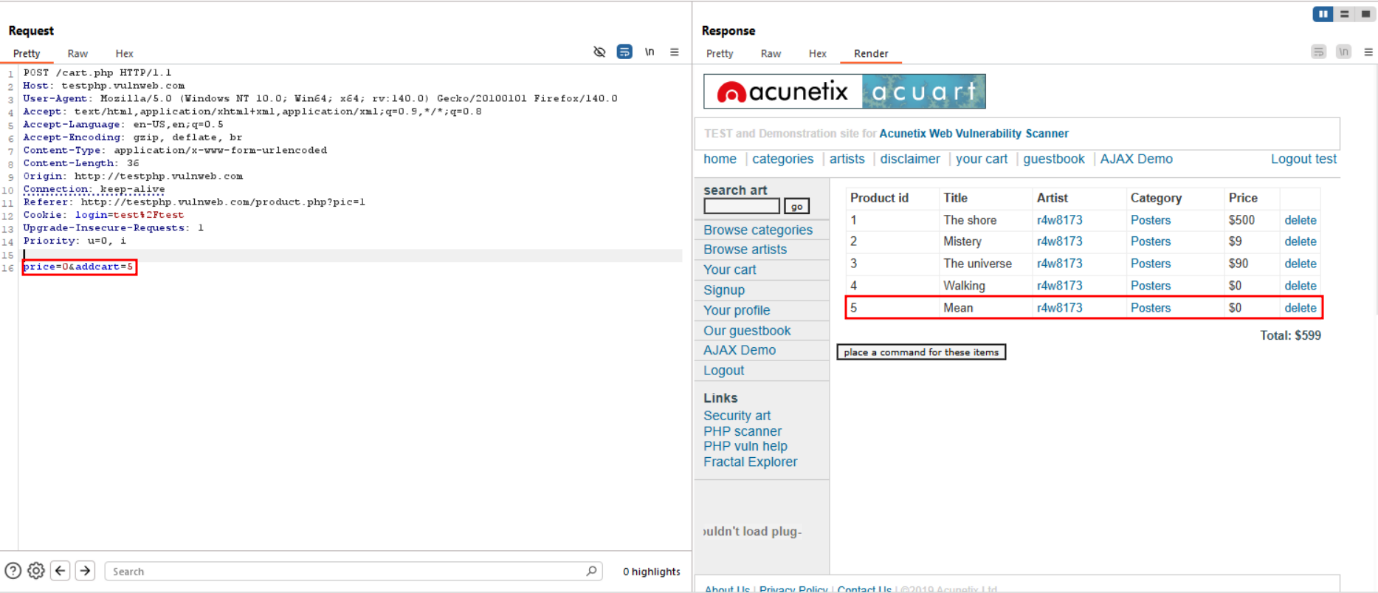
**Workflow and State Management Flaws**

These abuse multi-step processes by skipping steps, re-doing steps, or performing them out of order.

**Privilege Escalation Flaws**

These allow a user to perform actions they are not authorized to do.

* Complete System Compromise.
* Unauthorized Data Disclosure (accessing all user data).
* Ability to Modify or Delete Data for All Users.
* **Security Recommendations:**
* **Enforce Workflows on the Server-Side**
  + Never trust that the user followed the intended path. Use a server-side state machine to manage user progression through a multi-step process.Clear Session Cookies on the Client-Side (Good Hygiene)
* **Rigorous Manual Testing and Code Review**
  + Manual Security Testing: This is where human intelligence shines. A skilled penetration tester will try to understand the business and purposefully break its rules. This is essential for finding logic flaws.
  + Peer Code Reviews: Have another developer who understands the business context review the code for a feature. They can often spot assumptions or logical gaps that the original developer missed.
* **VALIDATION SNAPSHOT:**

****

**15. Vulnerability Name: Cross-Site Scripting (XSS)**

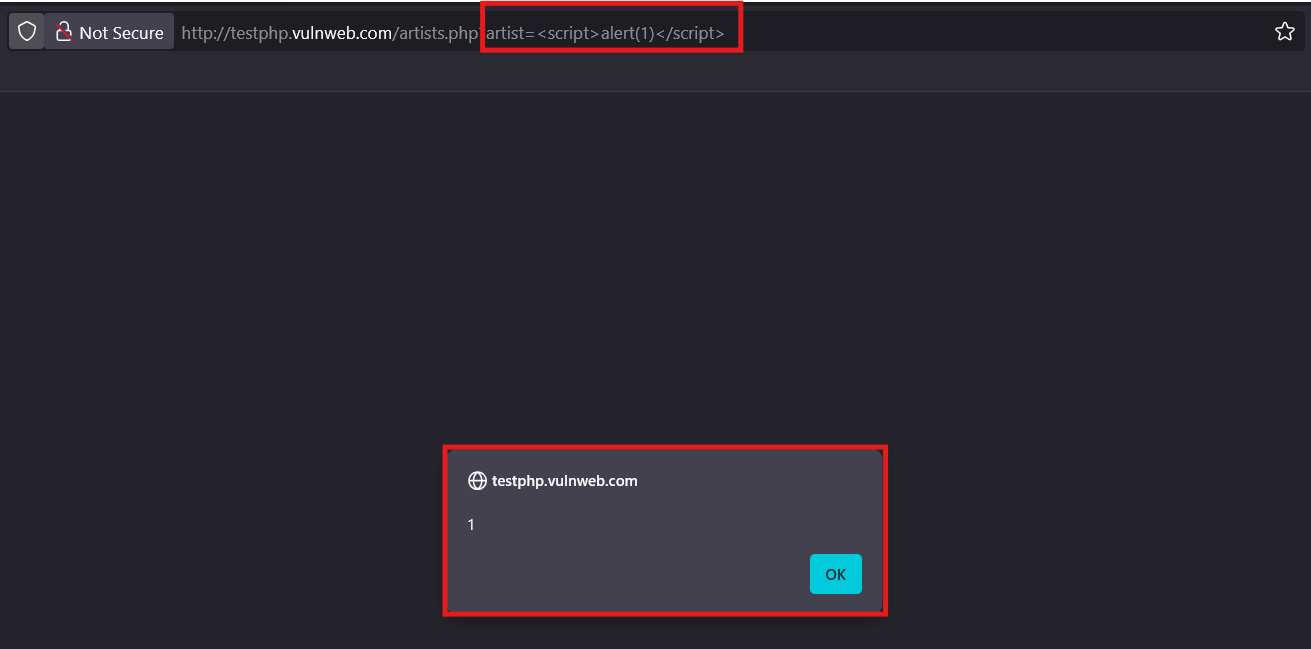
* **Security Insight:** 
  + **Untrusted Input Rendered as HTML/Script**: When web applications include user-supplied input (e.g., form fields, query parameters, or JSON data) in the HTML or JavaScript output without proper encoding or sanitization, attackers can inject malicious scripts.
  + **Browser Execution Context Exploitation**: The browser trusts the application to render content safely. An attacker abuses this trust by injecting scripts that execute in the context of a legitimate user session — leading to cookie theft, defacement, or account compromise.
* **CVSS:** 7.4
* **CWE-ID:** 
  + CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
  + CWE-80 Improper Neutralization of Script-Related HTML Tags in a Web Page
  + CWE-116 Improper Encoding or Escaping of Output
* **Security Implications:**
  + Attacker can steal session cookies and impersonate a user.
  + Inject fake login forms (phishing), capture input fields.
  + Alter or overwrite page content for all users (stored XSS).
  + Drop drive-by downloads or redirect users to malicious domains.
  + Use browser APIs to scan internal networks (e.g., in internal apps).
* **Security Recommendations:**

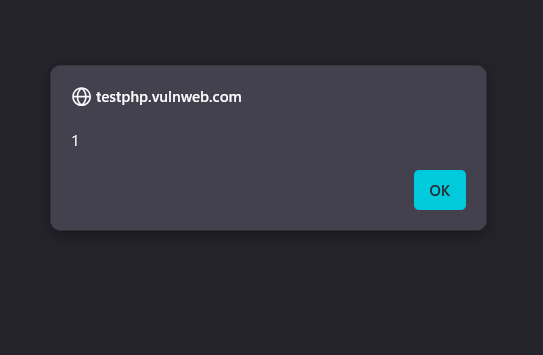
### Rigorous Manual Testing and Code Review

* **Manual Security Testing**: Human testers simulate real-world attacks to identify vulnerable input fields, improper context rendering, or unescaped output. Especially effective for DOM-based XSS.
* **Peer Code Reviews**: Developers reviewing each other’s code can catch trust boundary violations or unsafe JavaScript rendering patterns early.

**Input Validation and Sanitization**

* **Allowlist Validation**: Accept only known-good input (e.g., alphanumeric usernames) instead of blocking known-bad input.
* **Sanitize Input**: Use libraries like DOM Purify to strip unsafe tags in rich text fields (e.g., blog posts, comments).
* **VALIDATION SNAPSHOT:**

****



**16. Vulnerability Name: Cross-Site Request Forgery (CSRF) – Reusable Validation POC for Password Change**

* **Security Insight:**

**Lack of Anti-CSRF Controls on Sensitive Endpoints**

* **Unauthenticated Request Execution via Trusted User Context:** CSRF allows an attacker to craft a malicious HTML form or JavaScript that, when visited by an authenticated user (like a logged-in admin), automatically submits a request on their behalf, such as changing their account password — without their knowledge.
* **Reusable CSRF Payload Without Anti-Replay Checks:** In this case, the attacker can replay the same CSRF Validation Snapshot multiple times, because the endpoint lacks anti-CSRF tokens, session binding, or request uniqueness (e.g., nonce).
* **CVSS:** 8.8
* **CWE-ID:**
* CVE-2020-8184
* CVE-2018-1000525
* CVE-2020-15227
* **Security Implications:**
* The attacker changes the victim's password, locking them out.
* Validation Snapshot can be reused multiple times because the request lacks nonces or tokens.
* If the CSRF affects admin or elevated users, full control of the application may be obtained.
* Violates OWASP Top 10 (A01:2021 – Broken Access Control) and common privacy laws if users lose access to their accounts.
* **Security Recommendations:**

**Rigorous Manual Testing and Code Review**

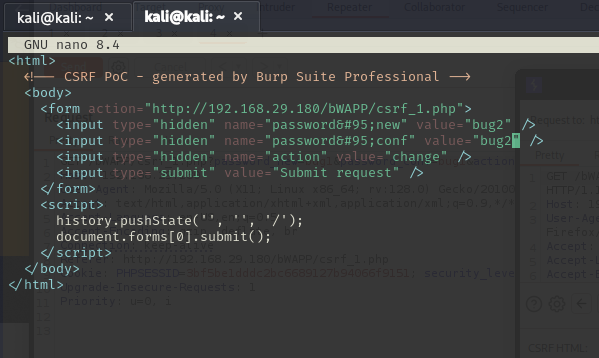
* **Manual Testing with Burp Suite or Postman**: Security testers try to submit forged POST requests to sensitive endpoints like /change-password and observe behavior without CSRF tokens.
* **Peer Code Reviews**: Developers should inspect forms and endpoint logic to verify anti-CSRF mechanisms (e.g., synchronizer token pattern, SameSite cookies).

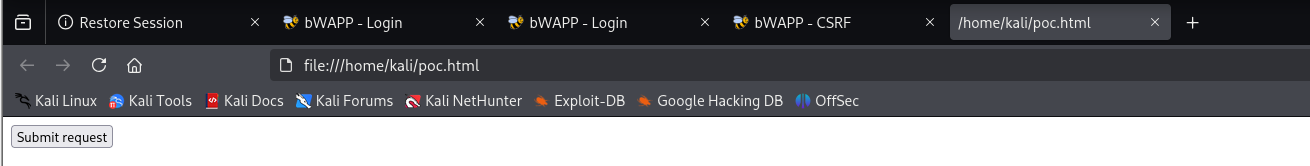
**Implement CSRF Tokens (Synchronizer Token Pattern)**

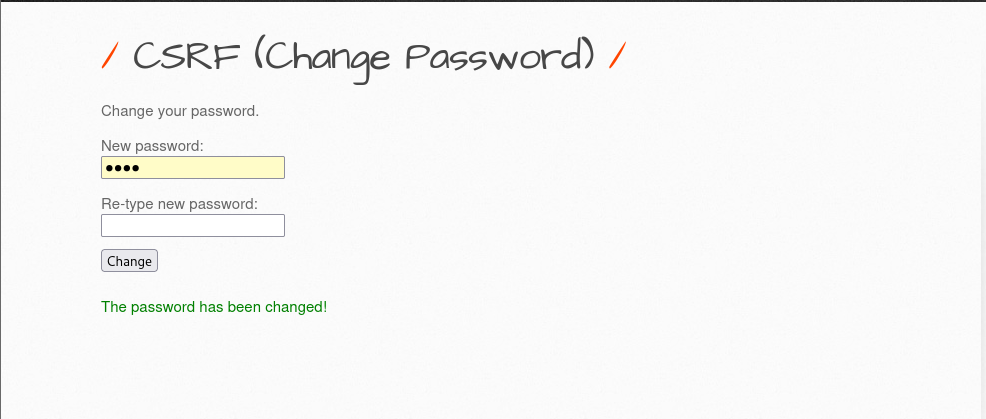
* **Generate a Unique CSRF Token per Session/Form**: Include it in all state-changing requests (POST, PUT, DELETE).

**Enforce Reauthentication for Sensitive Actions**

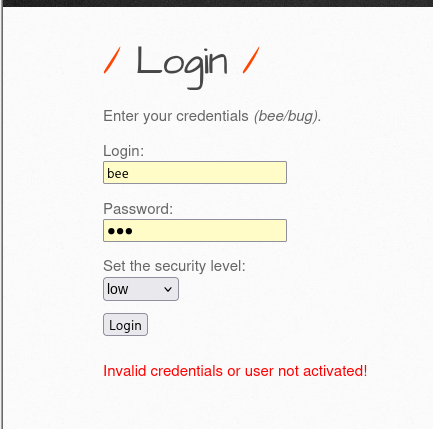
* Require the user to enter the **current password or OTP** before changing the account password — even if they are logged in.
* **VALIDATION SNAPSHOT:**

****





* Here we can see that the password is changed to bug2
* For clarification let’s use the bee/bug credentials and see if the login is possible.



* Here we can see that the password: bug is not valid.

**17. Vulnerability Name: Misconfigured crossdomain.xml (Wildcard Domain & Ports)**

* **Security Insight:**

**Overly Permissive Cross-Domain Policy in crossdomain.xml**

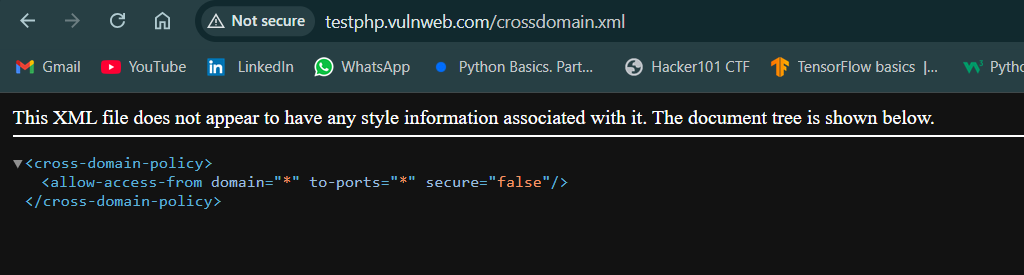
* **Legacy Cross-Domain Trust Model:** The crossdomain.xml file allows Flash, Silverlight, and older applets to request resources from your domain. When domain="\*" and to-ports="\*" are used, any external domain can make unrestricted cross-origin requests to all application ports, even to sensitive internal APIs or services.
* **No Restrictions + No TLS Requirement**: The secure="false" setting means even unencrypted HTTP connections are permitted — exposing data and functionality to any untrusted origin**.**
* **CVSS:** 8.6
* **CWE-ID:** 
  + CWE-942 Overly Permissive Cross-domain Policy
  + CWE-284 Improper Access Control
  + CWE-200 Exposure of Sensitive Information to Unauthorized Actor
* **Security Implications:**
* Any site can send authenticated or anonymous Flash-based requests to your app and access sensitive resources.
* Could allow unauthorized access to internal APIs, admin panels, or backend services through legacy vectors.
* Flash apps or malware can be used to interact with internal apps over cross-origin requests (Client-Side Request Forgery / CSRF over Flash).
* Violates OWASP Secure Configuration and modern secure cross-origin resource sharing practices.
* **Security Recommendations:**
* Decommission Flash/Legacy Dependencies
  + Audit and remove legacy Flash/Silverlight apps from your infrastructure.
  + Modern applications should use CORS headers, not crossdomain.xml.

### Web Application Firewall (WAF) Filtering

* Block .xml files exposing wildcard cross-domain access using WAF or reverse proxy filtering.

**Server-Side Validation**

* Ensure backend services do not trust origin headers blindly — implement strict session and auth checks on all endpoints.
* **VALIDATION SNAPSHOT:**

****

**18. Vulnerability Name: No Rate Limiting Implemented**

* **Security Insight:**

**Unrestricted Repeated Requests to Critical Endpoints**

* Missing Brute-Force and Abuse Protection: When a web application does not implement rate limiting, an attacker can send unlimited requests to endpoints like login, password reset, 2FA verification, or form submissions.
* No Throttling on High-Risk Actions: Without request throttling, APIs and endpoints become vulnerable to automated attacks, such as credential stuffing, OTP brute-forcing, enumeration, and service degradation.
* **CVSS:** 8.8
* **CWE-ID:** 
  + CWE-770 Allocation of Resources Without Limits or Throttling
  + CWE-307 Improper Restriction of Excessive Authentication Attempts
  + CWE-400 Uncontrolled Resource Consumption (Resource Exhaustion)
* **Security Implications:**
* Attackers use automated scripts to test millions of credentials on login forms without being blocked.
* Lack of rate limiting on MFA endpoints enables brute-force guessing of short codes.
* Error-based or timing-based responses can be harvested rapidly to identify valid usernames/emails.
* Mass form submissions or resource-intensive API hits can exhaust system resources.
* APIs or checkout pages may be spammed to abuse coupons, offers, or overload inventory.
* **Security Recommendations:**

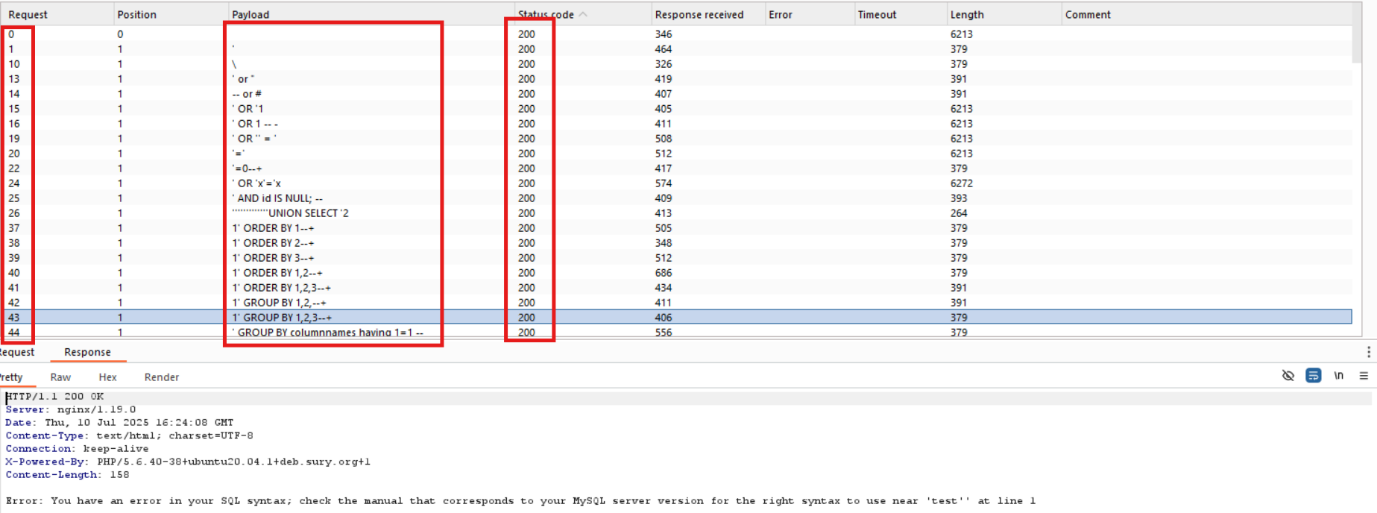
**Server-Side Rate Limiting and Throttling**

* **Apply per-IP, per-account, and per-endpoint limits** using server rules or middleware.
  + Example: Limit login attempts to 5 per minute per IP.
* Use tools like:
  + **Fail2Ban** (Linux)
  + **express-rate-limit** (Node.js)
  + **ModSecurity** or **NGINX rate-limiting modules**

**CAPTCHA After Threshold**

* **Introduce CAPTCHA** or other human verification after a set number of failed attempts.
  + Example: After 5 login attempts, show CAPTCHA or re-authenticate via email/OTP.

**Progressive Delays**

* **Increase wait times** (e.g., 2s, 4s, 8s…) after repeated failed attempts, delaying brute-force tools.
* **VALIDATION SNAPSHOT:**

**19. Vulnerability Name: HTTP TRACE Method Enabled – Vulnerable to Cross-Site Tracing (XST)**

* **Security Insight:**

**Server Echoes Back Request Headers Including Cookies**

* XST (Cross-Site Tracing) is an attack technique that combines HTTP TRACE with XSS to steal sensitive headers, especially session cookies — even if they are marked HttpOnly.
* The TRACE method is used for diagnostic purposes and echoes back all headers, including cookies and custom headers, making it exploitable if an attacker can trigger a TRACE request via XSS.
* **CVSS:** 7.5
* **CWE-ID:**
* CWE-201 Exposure of Sensitive Information Through Sent Data
* CWE-598 Information Exposure Through Query Strings in GET Request
* CWE-693 Protection Mechanism Failure
* **Security Implications:**
* Even HttpOnly cookies can be exfiltrated using XSS + TRACE combo.
* Exposes all HTTP headers, including authentication tokens, cookies, user agents, and custom headers.
* Allows an attacker to bypass security controls like HttpOnly or logging restrictions.
* **Security Recommendations:**

**Harden Web Server Configuration**

* Disable unnecessary HTTP methods like TRACE, TRACK, OPTIONS if not required.
* Use HTTP method whitelisting (e.g., only allow GET, POST, DELETE).

**Rigorous Manual Testing and Code Review**

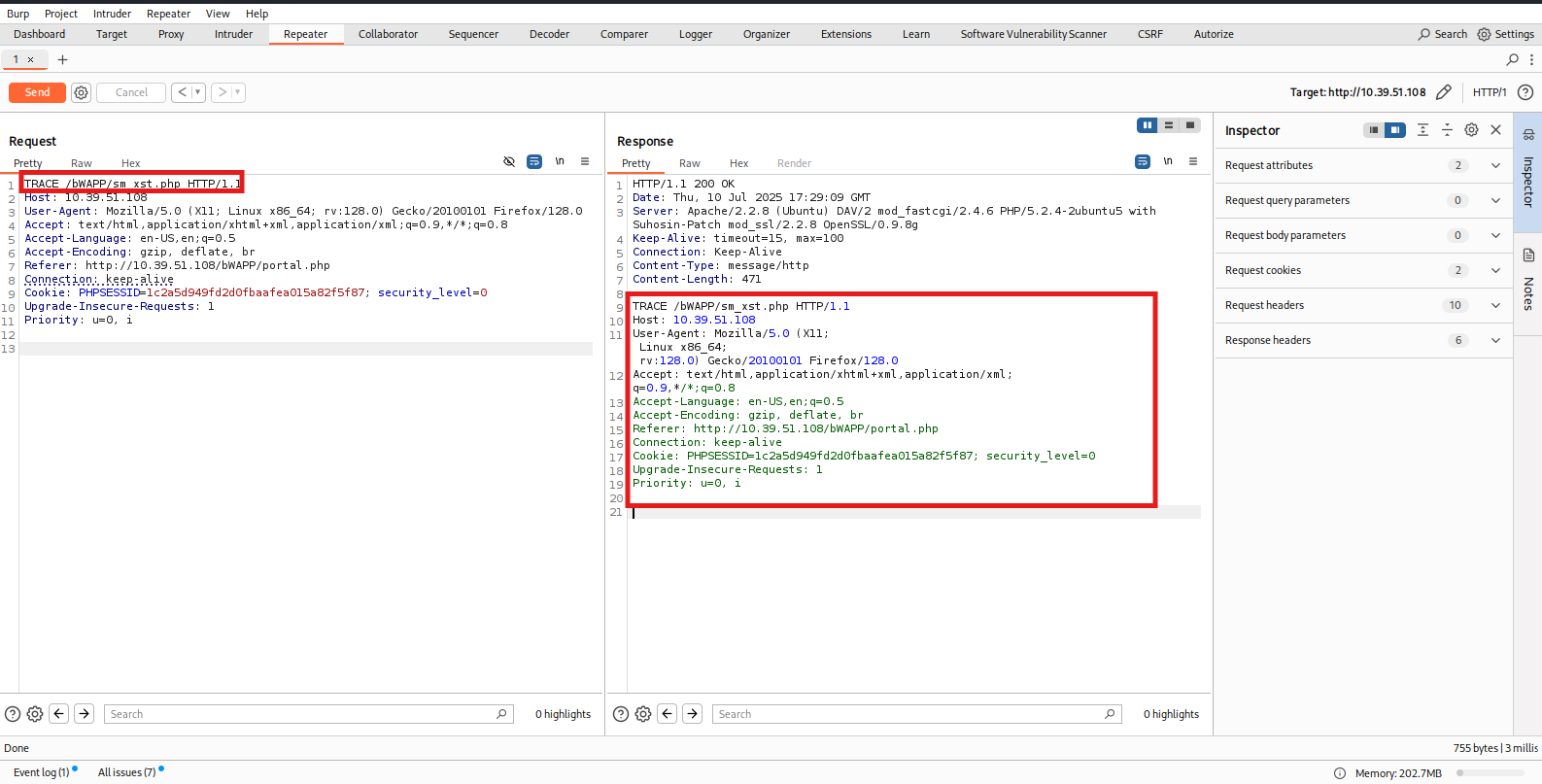
* **Manual Header Inspection**: Security testers use Burp Suite or curl to send TRACE requests and check if headers (including cookies) are reflected.
* **Review Server Configs**: Ensure TRACE is explicitly disabled even in dev/test environments.

**Implement Strong XSS Protections**

* **XST relies on XSS** — so mitigating XSS (input validation, output encoding, CSP) indirectly protects against XST.

**Content Security Policy (CSP)**

* Deploy CSP headers to restrict script sources and prevent JavaScript from sending arbitrary requests.
* **VALIDATION SNAPSHOT:**



**20. Vulnerability Name: Credentials Visible on Login Page**

* **Security Insight:**

This vulnerability occurs when valid credentials (username and/or password) are hardcoded, pre-filled, or displayed directly in the HTML source, browser console, JavaScript, or autocomplete fields of a login page. It happens due to:

* Developer oversight (debug/test accounts left behind)
* Misconfigured frontend logic
* Hardcoded values in client-side JavaScript
* Browser autofill caching sensitive values
* **CVSS:** 7.5
* **CWE-ID:**

* + CWE-200 Exposure of Sensitive Information to Unauthorized Actor
  + CWE-522 Insufficiently Protected Credentials
  + CWE-359 Exposure of Private Personal Information
  + CWE-922 Insecure Storage of Sensitive Information
* **Security Implications:**
  + Attackers can view and use pre-filled or hardcoded credentials for unauthorized access.
  + If credentials grant admin or elevated access, it can lead to full account takeover.
  + Encourages credential stuffing, privilege escalation, or lateral movement within the app.
  + May breach data protection laws (e.g., GDPR, HIPAA) if credentials are exposed unintentionally.
* **Security Recommendations:**

**Remove Hardcoded Demo Credentials**

* Delete prefilled value attributes in login form inputs (e.g., value="admin").
* Avoid any default usernames/passwords in production systems — use environment-based configuration if needed for testing.

**Do Not Embed Sensitive Data in Client-Side Code**

* Avoid placing credentials, API keys, or access tokens in:
  + HTML comments
  + JavaScript variables
  + Inline event handlers

**Implement Rigorous Code Review**

* Conduct manual code reviews or use automated tools to detect credentials or secrets exposed in frontend templates, JavaScript, or initial DOM load.
* **VALIDATION SNAPSHOT:**

****

**21. Vulnerability Name: Server-Side Request Forgery (SSRF)**

* **Security Insight:**

**SSRF** occurs when an attacker manipulates a server-side application to make HTTP requests to arbitrary destinations, including **internal services**, **metadata endpoints**, or **external malicious hosts**.

1. **RFI-based SSRF** – where attacker supplies a malicious remote file to a vulnerable inclusion point (include($\_GET['page']))
2. **XXE-based SSRF** – using XML payloads that instruct the server to fetch internal URLs via <!ENTITY>
3. **Legacy Attack (CVE-2013-4890)** – SSRF through XXE targeting Samsung SmartTVs over the LAN

* **CVSS:** 7.5
* **CWE-ID:**

* + CWE-918 Server-Side Request Forgery (SSRF)
  + CWE-611 Improper Restriction of XML External Entity Reference ('XXE')
  + CWE-829 Inclusion of Functionality from Untrusted Control Sphere
* **Security Implications:**
* Attacker discovers internal services (e.g., localhost:3306, 127.0.0.1:8000) using timing, response content, or error messages.

Cloud exploitation:

* Example: AWS metadata API → http://169.254.169.254/latest/meta-data/ → credentials theft.
* Targeting IoT or legacy systems (as in CVE-2013-4890) with malformed SSRF payloads can crash devices.
* Exploit misconfigured APIs (like internal admin panels, debug endpoints, or Redis) which trust localhost.
* **Security Recommendations:**
  + Disable Remote File Inclusion
  + Whitelist Only Trusted Internal Files
  + Allow only specific known file paths or templates.
  + Disallow URLs in Parameters
  + Validate input strictly — disallow http://, IPs, or domain names in user-controlled parameters.
  + Block Internal IP Ranges in SSRF-Prone Functions
  + Firewall Egress Rules (Server Outbound Protection)
  + Block backend servers from making arbitrary outbound HTTP requests.
* **VALIDATION SNAPSHOT:**

****

MEDIUM

**22. Vulnerability Name: TLS/SSL Not configured**

* **Security Insight:**

A web application without a **configured TLS/SSL certificate** does not encrypt data transmitted between the client and the server. This exposes sensitive information (e.g., login credentials, session tokens, personal data) to interception via **Man-in-the-Middle (MitM)** attacks.

Instead of using https://, the application uses http://, meaning:

* All traffic is sent in **plain text**.
* Attackers on the same network (e.g., public Wi-Fi) can easily sniff and modify traffic.
* **CVSS:** 7.5
* **CWE-ID:**

* CWE-311 Missing Encryption of Sensitive Data
* CWE-319 Cleartext Transmission of Sensitive Information
* **Security Implications:**
* Attackers can intercept login credentials and tokens
* Session cookies sent without Secure flag can be stolen
* Any sensitive information (e.g., emails, card numbers) sent via HTTP is readable
* Users can be tricked into visiting non-secure versions
* Violates PCI-DSS, GDPR, HIPAA, etc.
* **Security Recommendations:**
* Install a Valid SSL/TLS Certificate

Obtain certificates from trusted Certificate Authorities (CAs) like:

* Let’s Encrypt (free)
* DigiCert, Sectigo, etc.
* Redirect All HTTP to HTTPS

Force secure connections:

*RewriteEngine On*

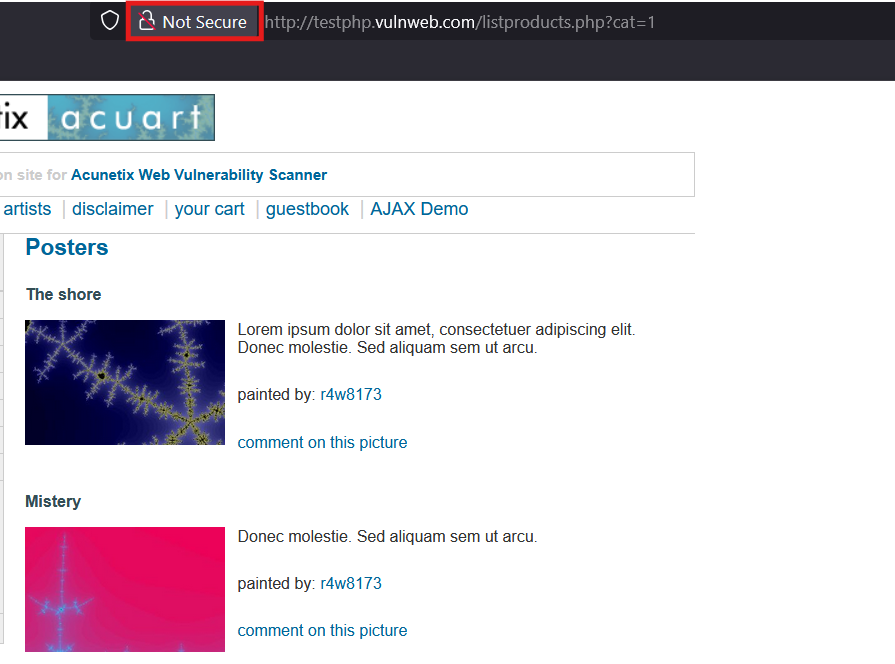
*RewriteCond %{HTTPS} off*

*RewriteRule ^ (. \*) $ https://% {HTTP\_HOST} %{REQUEST\_URI} [L, R=301]*

* Block HTTP Access

If possible, close port 80 entirely in production environments.

* **VALIDATION SNAPSHOT:**



LOW

**23. Vulnerability Name: HTML Injection**

* **Security Insight:** HTML Injection occurs when untrusted input is embedded directly into a web page’s HTML structure without proper sanitization or encoding.

**Unlike Cross-Site Scripting (XSS), which executes scripts, HTML Injection may:**

* Insert or modify page content (e.g., adding links, images, forms)
* Deface pages
* Phish users via crafted input (e.g., fake login forms)
* Interfere with UI or application logic (clickjacking, fake buttons, etc.)
* This often results from failing to escape input in the HTML body, attributes, or form elements.
* **CVSS:** 4.0
* **CWE-ID:**

* CWE-80 Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)
* CWE-79 Improper Neutralization of Input During Web Page Generation
* **Security Implications:**
  + - Injected HTML modifies appearance or content of the page
    - Fake forms or overlays may be inserted to steal user data
    - Users may be tricked into interacting with malicious elements
    - HTML can alter the structure of existing forms or fields
* **Security Recommendations:**

**Encode User Input Before Rendering:**

* + - Always encode characters like <, >, ", ', and & before rendering input in HTML.
    - Use HTML entity encoding: e.g., < → &lt;, > → &gt;

**Use Trusted Templating Engines:**

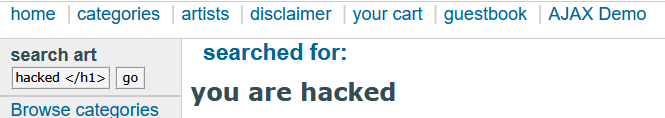
* + - Templating engines like **Handlebars**, **Mustache**, **Thymeleaf**, etc., auto-escape HTML by default.

**Validate Input Strictly:**

* + Whitelist expected inputs (e.g., only alphanumeric or specific format).
  + Reject inputs containing suspicious characters.

**Do Not Reflect Unescaped Input in the DOM:**

* + Never embed user input directly inside tags or attributes.
* **VALIDATION SNAPSHOT:**

****

* 1. **Conclusion:**

**Summary of Findings:**

The web application contains multiple vulnerabilities, including SQL Injection, XSS, authentication flaws, and missing security controls, which could lead to data breaches, unauthorized access, and system compromise.

**Overall Risk Level:** High

**Next Steps:** Address the identified vulnerabilities, implement recommended security measures, and conduct a retest to ensure mitigation.