Assignment 03
Vulnerability Report

December 10 - December 11

Prepared by: Danil Vilmont & Jason Mou
Course: ITAS 268 - Assignment 03
Date: December 10, 2025

Table of Contents

Next Page →

Team Information:

- **Team Member 1:** Jason Mou
- **Team Member 2:** Danil Vilmont
- **Report Date:** 12/10/2025
- **Target Application:** bWAPP (Bee-Soft Web Application Platform) 2.2
- **Testing Period:** 12/05/2025 - 10/09/2025

Vulnerability #1:

SQL Injection (Login Form -Post)

- **Summary:** The login form in bWAPP is vulnerable to SQL Injection through unsanitized POST parameters. By injecting SQL into the `login` field, an attacker can bypass authentication and access the system without a valid password.

Severity & Classification

Metric	Value
Severity Rating	Critical
CVSS v3.1 Score	8.6 (High)
CVSS Vector String	AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:N
CWE Classification	CWE-89: Improper Neutralization of Special Elements used in SQL Commands ('SQL Injection')

Next Page →

Affected Component

Field	Details
URL/Endpoint	http://localhost/bwapp/sqli_3.php
HTTP Method	POST
Vulnerable Parameter(s)	Login, Password
bWAPP Security Level Tested	Low

Description

- **Description:** The login form does not sanitize user input such as ‘ or ‘1’=’1’ - -, allowing attackers to bypass authentication using SQL logical operators.

Proof of Concept

- **Prerequisites**
  - *None(no login required)*
  - *Access to the bWAPP application*
- **Steps to Reproduce**
  1. Navigate to: **SQL Injection (Login Form/Hero)** (sqli_3.php)
  2. In the **Login** field, enter the SQL payload.
  3. Leave the **Password** field blank.
  4. Click **Login**.
  5. The application logs you in as “Neo” without requiring valid credentials.

Next Page →

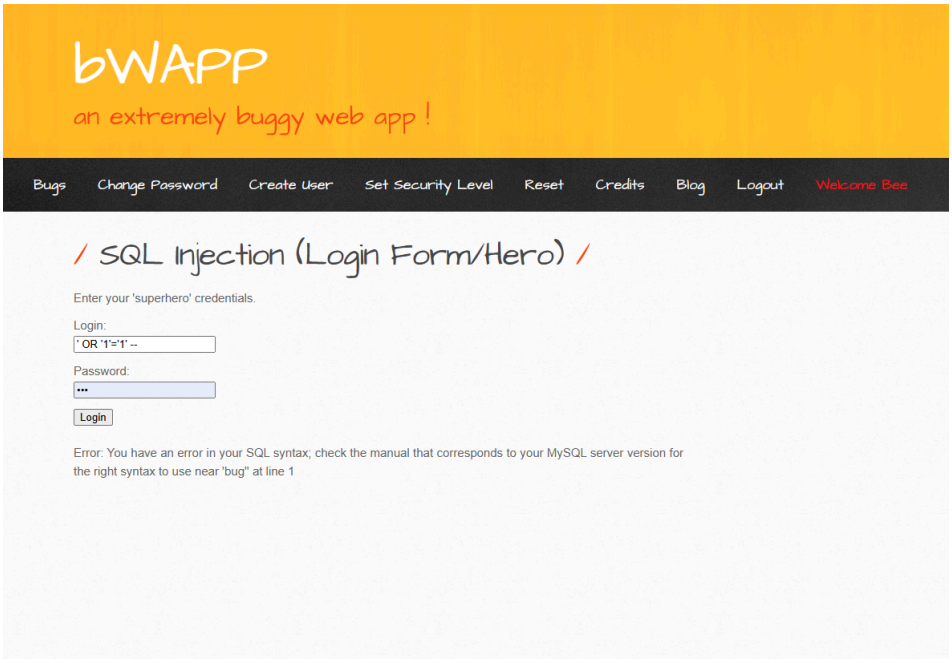
Payload Used

- ' OR '1'='1' --

Expected vs. Actual Behavior

Expected Behavior	Actual Behavior
Application validates credentials and denies incorrect logins.	The application authenticates the attacker and logs them in as “Neo”.

Evidence

- **Figure 1: Successful login after SQL Injection**
  - 

Next Page →

Impact Analysis

CIA Triad Assessment

Impact Category	Rating	Justification
Confidentiality	(None, Low, High) High	Attackers can gain full account access and potentially dump database contents.
Integrity	(None, Low, High)High	Attackers can manipulate or modify database data.
Avaliability	(None, Low, High)Low	No direct DoS, but chained attacks may lead to service disruption.

Real-World Attack Scenario

- *An attacker could use this vulnerability to log in without a real password and access accounts they shouldn't. Once inside, they could see private user information, steal data, or even get admin access. They might then dump the whole database or use the account to launch more attacks*.

Remediation Recommendations

Immediate Actions

1. Check and clean user inputs on the server to block harmful characters.
2. Escape or filter inputs before using them in SQL queries.

Next Page →

Long-Term Fixes

1. Use parameterized queries (prepared statements) instead of directly inserting input.
2. Use a Web Application Firewall (WAF) to catch attacks.
3. Give the web app only the minimum database permissions it needs.

References

- **OWASP:**https://owasp.org/www-community/attacks/SQL_Injection
- **CWE:** https://cwe.mitre.org/data/definitions/89.html

Vulnerability #2:

XSS - Stored (blog)

- **Summary:** Stored Cross-Site Scripting (XSS) in Blog Comments

Severity & Classification

Metric	Value
Severity Rating	Critical
CVSS v3.1 Score	7.6 (High)
CVSS Vector String	CVSS:3.1/AV:N/AC:L/PR:L/UI:R/S:C/C:L/I:H/A:N
CWE Classification	CWE-79: Improper Neutralization of Input During Web Page Generation

Next Page →

Affected Component

Field	Details
URL/Endpoint	/bWAPP/xss_stored_1.php
HTTP Method	POST
Vulnerable Parameter(s)	Blog comment fields
bWAPP Security Level Tested	Low

Description

Description: The blog feature of bWAPP fails to sanitize or encode input before storing it in the backend and rendering it to other users. Because the application directly inserts user data into the HTML output without validation, an attacker can inject JavaScript that executes automatically whenever a visitor loads the page.

This vulnerability is dangerous because it persists across sessions and affects every user who views the blog. An attacker could use this vector to steal session cookies, perform account actions on behalf of victims, or display fraudulent content.

Proof of Concept

- **Prerequisites**
  - Standard authenticated user session (bee/bug)
  - Security level set to Low
- **Steps to Reproduce**
  1. Go to /bWAPP/xss_stored.php
  2. In the comment field, submit any js payload: e.g.
     a. “<script>alert(‘TEST”);</script>”
  3. Submit
  4. Refresh page
  5. Profit

Next Page →

Payload Used

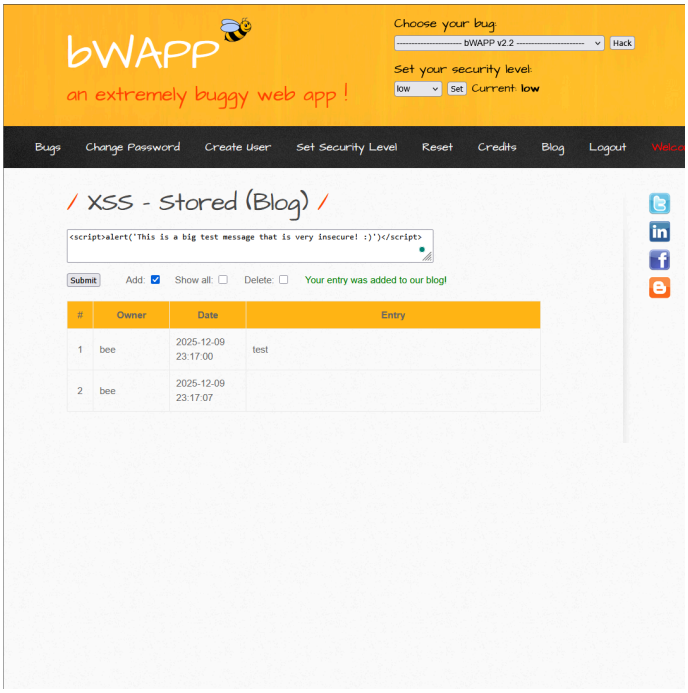
- <script>alert(‘TEST”);</script>

Expected vs. Actual Behavior

Expected Behavior	Actual Behavior
User input should be encoded or rejected.	User input is executed as JavaScript when the page loads.

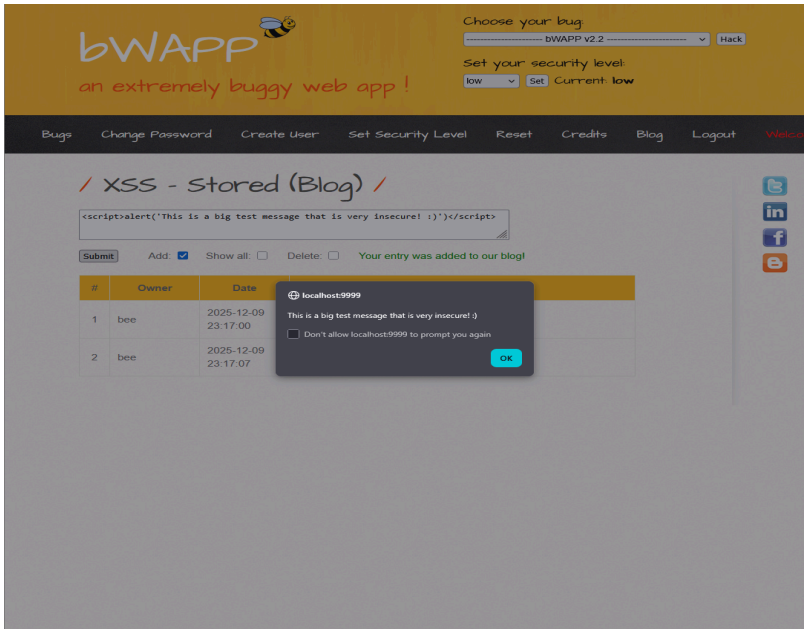
Evidence

- **Figure 1:** Stored XSS payload entered
  - xss_stored_01_form.png



Next Page →

- **Figure 2:** Persistent payload displayed in comments
  - xss_stored_02_alert.png



Impact Analysis

CIA Triad Assessment

Impact Category	Rating	Justification
Confidentiality	High	Attackers can steal session cookies or sensitive data.
Integrity	High	Attackers can alter page content.
Avaliability	Low	Does not directly affect anything outside of blog.

Real-World Attack Scenario

- An attacker posts a malicious comment to the blog. Every visitor who loads the page unknowingly executes the attacker's JavaScript. This could steal their session token, allowing the attacker to hijack their account or escalate privileges.

Next Page →

Remediation Recommendations

Immediate Actions

1. Validate and sanitize all user input.
2. Encode output with “htmlspecialchars()”.

Long-Term Fixes

1. Implement a universal input validation system.
2. Have a plan for handling future XSS attacks.

Code Example (if applicable)

- **Vulnerable Code Pattern:**
  - echo $_POST['txtMessage'];
- **Secure Code Pattern:**
  - echo htmlspecialchars($_POST['txtMessage'], ENT_QUOTES, 'UTF-8');

References

- **OWASP:** https://owasp.org/www-community/attacks/xss/
- **CWE:** https://cwe.mitre.org/data/definitions/79.html

Next Page →

Vulnerability #3:

Directory Traversal

- **Summary:** Directory Traversal via Page Parameter

Severity & Classification

Metric	Value
Severity Rating	High
CVSS v3.1 Score	7.4 (High)
CVSS Vector String	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:L/I:L/A:L
CWE Classification	CWE-22: Improper Limitation of a Pathname to a Restricted Directory

---

Affected Component

Field	Details
URL/Endpoint	/bWAPP/directory_traversal_1.php
HTTP Method	GET & POST
Vulnerable Parameter(s)	?page=""
bWAPP Security Level Tested	Low

Next Page →

Description

- **Description:** The application's file viewer allows anything to be given as input. This can allow an attacker to access any file on the filesystem, as there is no validation or sanitization.

  This flaw allows unauthorized exposure of sensitive server files, including configuration, passwords, and log files. File exposure may further enable secondary attacks such as credential theft.

Proof of Concept

- **Prerequisites**
  - Standard authenticated user session (bee/bug)
  - Security level set to Low
- **Steps to Reproduce**
  1. Go to /bWAPP/Directory_Traversal_1.php
  2. In the "?page='" field, enter a payload, such as "../../../../etc/passwd"
  3. Submit
  4. The server now returns the content of /etc/passwd.
  5. profit

Payload Used

- ?page="../../../../etc/passwd"

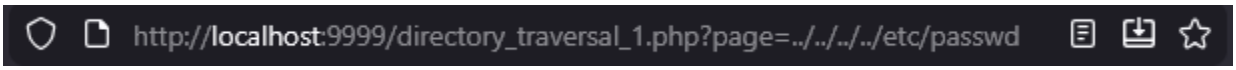
Expected vs. Actual Behavior

Expected Behavior	Actual Behavior
Application restricts access to safe directories.	The application reads and writes arbitrary system files.

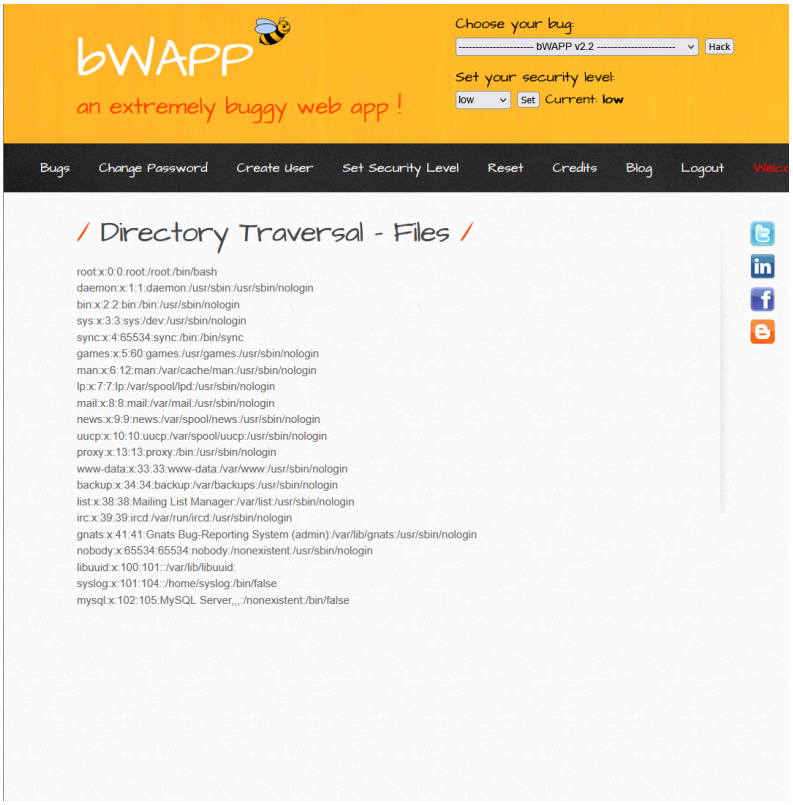
Next Page →

Evidence

- **Figure 1:** Payload entered into the file parameter
  - traversal_01_input.png



- **Figure 2:** Returned content from '/etc/passwd'
  - traversal_02_output.png



Next Page →

Impact Analysis

CIA Triad Assessment

Impact Category	Rating	Justification
Confidentiality	High	Attackers can access sensitive system files.
Integrity	Low	Read-Only impact but may enable further attacks.
Avaliability	Low	The entire backend is now readable; this means you can access admin data.

Real-World Attack Scenario

- An attacker can read configuration files containing database credentials. They then use those credentials to connect to the production database and extract user data.

Remediation Recommendations

Immediate Actions

1. Validate the ‘page’ parameter against an allow-list of files.
2. Reject any kind of “../” travel.

Long-Term Fixes

1. Implement a secure file access system that doesn’t rely on user input.

Next Page →

Code Example (if applicable)

- **Vulnerable Code Pattern:**
  - include($_GET[‘page’]);
- **Secure Code Pattern:**
  - $allowedFiles = [‘about.html’, ‘contact.html’];
    If (in_array($_GET[‘page’], $allowedFiles)) {
    include($_GET[‘page’]);
    }

References

- **OWASP:** https://owasp.org/www-community/attacks/Path_Traversal
- **CWE:** https://cwe.mitre.org/data/definitions/22.html
