**BWAPP VULNERABLITY ASSESSMENT REPORT**

# TABLE OF CONTENT

# 1.Executive summary

**2. Introduction, Scope, and Objectives**

 2.1. Objectives

 2.2. Scope

**3. Methodology**

**4. Risk Rating System**

**5. Detailed Findings and Mitigation**

 A01: Broken Authentication – Weak Passwords

A02: Cryptographic Failure (Base64 Encoding Secret)

 A03: SQL Injection (GET/Search)

A04: Insecure Design (Insecure Direct Object Reference - IDOR)

 A05: Security Misconfiguration

A06: Missing HTTPS / TLS (Outdated Component)

**6. Appendix: Technical Component Analysis**

  6.1. Component and Banner Enumeration

 6.2. Nmap Scan Summary

**7. Conclusion and Remediation Plan**

*Prioritized Remediation Schedule*

1. Executive Summary

This assessment was conducted on the bWAPP application to identify, analyze, and report security vulnerabilities. The engagement identified **six major vulnerabilities** across key security domains, including authentication, cryptographic protection, injection, design, and component health.

The assessment concluded that the application currently maintains a **Critical-Risk Posture**. The findings include

**three Critical-Severity** vulnerabilities (SQL Injection and Security Misconfiguration) and **three High-Severity** vulnerabilities (Broken Authentication, Cryptographic Failure, and Insecure Design). The most pressing risks include the ability to easily bypass authentication , exposure of database details , and public access to credentials and configuration files. Immediate remediation is mandatory to prevent unauthorized access, sensitive data disclosure, and full system compromise

**2. Introduction, Scope, and Objectives**

**2.1. Objectives**

The primary objective of this vulnerability assessment was to:

1. Identify and document security weaknesses within the bWAPP application using industry-standard testing methodologies.
2. Provide clear evidence and Proof of Concepts (PoCs) for all discovered vulnerabilities.
3. Assign a risk severity rating to each finding based on potential impact and exploitability.
4. Deliver detailed, actionable remediation recommendations to improve the application’s security posture.

**2.2. Scope**

The assessment was limited to the publicly exposed web application endpoints and underlying infrastructure components of the bWAPP target environment.

* **Target:** bWAPP (Buggy Web Application)
* **Testing Type:** External, Black-Box Assessment (no source code review)
* **IP/Hostname:** 127.0.0.1 (or equivalent test environment IP)
* **Timeframe:** Assessment conducted on a defined date (as indicated by timestamps in the Nmap scan).

**3. Methodology**

The vulnerability assessment was executed using a

**hybrid manual and automated black-box approach**. This methodology simulated the actions of an external, unauthenticated attacker with no prior knowledge of the system's internal structure.

1. **Information Gathering & Enumeration:** Initial scanning was performed using tools like **Nmap** to identify open ports, services (80/tcp for Apache, 39121/tcp for Golang HTTP) , and technology banners (

curl) to discover outdated components (Apache 2.4.7, PHP 5.5.9).

1. **Vulnerability Identification:** The core testing involved following OWASP Top 10 categories. This included testing for

**Injection** (A03) ,

**Broken Authentication** (A01) ,

**Insecure Design/IDOR** (A04) , and

**Cryptographic Failures** (A02).

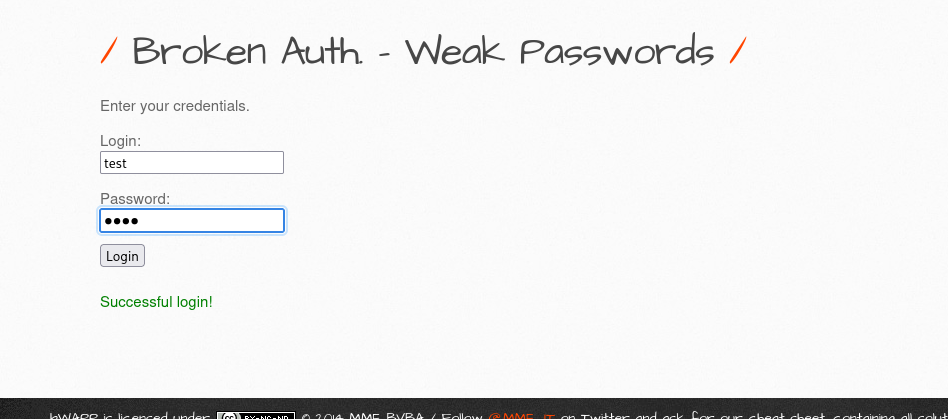
1. **Configuration Analysis:** Direct access attempts were made to common administrative and configuration paths (e.g., /passwords/, /admin/) to test for **Security Misconfiguration** (A05).
2. **Validation and Proof of Concept (PoC):** Every potential finding was manually validated to confirm exploitability and impact. Evidence, such as successful logins with weak credentials or decoded secrets, was captured for documentation.
3. **Risk Analysis and Reporting:** Findings were assigned a severity rating based on a CVSS-like score and detailed mitigation steps were provided

**OWASP TOP VULNERABILITES SHORT NOTES :**

| ID | Vulnerability Title | Severity | CVSS-like Score | Primary Risk |
| --- | --- | --- | --- | --- |
| **A03** | SQL Injection (GET/Search) | **CRITICAL** | 9.8 | Database Compromise |
| **A05** | Security Misconfiguration (File Exposure) | **CRITICAL** | 9.1 | Credential/Data Leakage |
| **A01** | Broken Authentication (Weak Passwords) | **HIGH** | 9.8 | Account Takeover |
| **A04** | Insecure Design (IDOR) | **HIGH** | 8.7 | Unauthorized Data Tampering |
| **A02** | Cryptographic Failure (Base64 Secret) | **HIGH** | 8.6 | Secret Disclosure |
| **A06** | Missing HTTPS / TLS (Outdated Component) | **MEDIUM** | ~5.0 | Traffic Sniffing (MITM) |

**VULNERABLITIES WITH POC:**

**A01: Broken Authentication – week password**



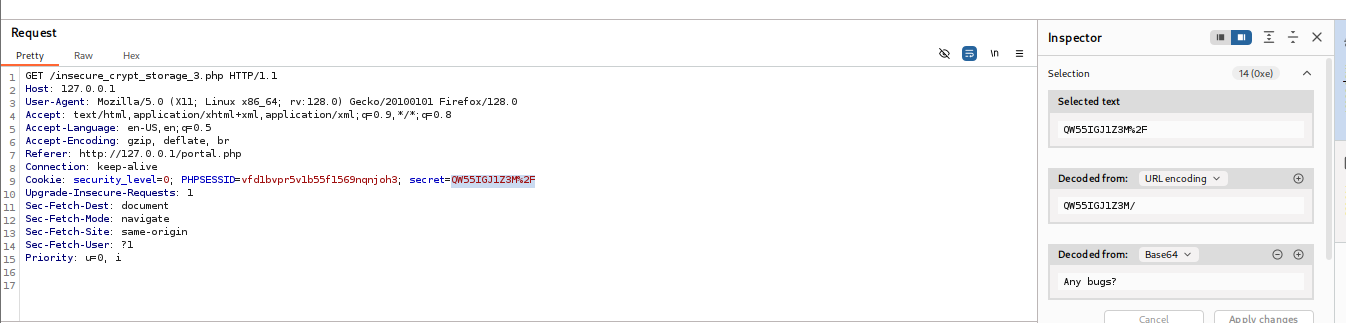
1. The application allows login with a **weak password** (test), enabling easy unauthorized access.

2. This broken authentication exposes user accounts and sensitive data, increasing risk of account takeover and lateral movement.

3. Lack of password complexity requirements makes the system highly vulnerable to brute-force and dictionary attacks.

|  |  |  |
| --- | --- | --- |
| SEVERITY | SCORE | MITIGATION |
| HIGH | 9.8 | 1. Enforce strong password policies and prevent the use of default or commonly used passwords. 2. Implement multi-factor authentication (MFA) and account lockout/rate limiting to block brute-force attacks. 3. Store all credentials securely using modern hashing algorithms such as bcrypt or Argon2. |

**A02: Cryptographic Failure (Base 64 encoding secret)**



1. Base64 is encoding, not encryption — it’s reversible, so decoding the captured value returns the original secret.

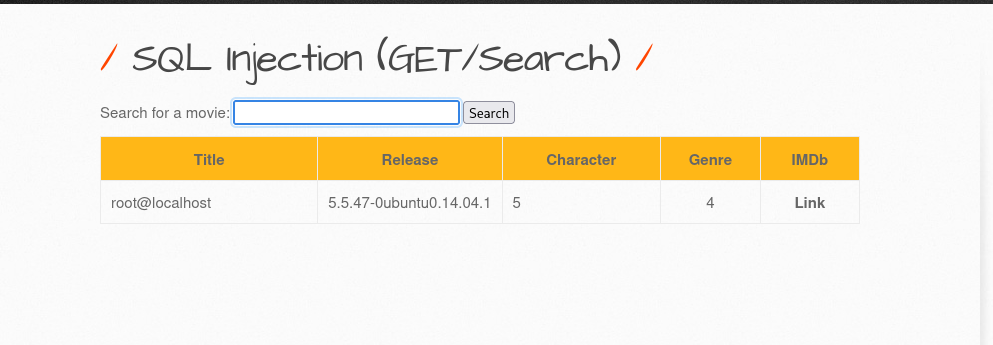
2. Capturing the request with Burp and decoding the Base64 proves exposure: the secret is retrievable from network traffic or logs.

3. This is a **cryptographic failure** (sensitive data handled improperly) because secrets were stored/transmitted in a reversible form, increasing risk of unauthorized access.

|  |  |  |
| --- | --- | --- |
| SEVERITY | SCORE | MITIGATION |
| HIGH | 8.6 | 1. Rotate & revoke any exposed values immediately — treat decoded Base64 tokens as compromised, revoke them, issue new credentials with least‑privilege scopes, and audit recent usage for abuse.  2. Protect in transit & at rest — never rely on Base64; use TLS for all transport and strong encryption (e.g., AES‑256 or platform KMS) for stored secrets; store only one‑way hashed values for passwords.  3. Use a secrets manager & remove secrets from logs/URLs — centralize secrets in a vault (HashiCorp Vault, AWS/Azure/GCP secret manager), issue short‑lived/rotating tokens, apply RBAC/audit logging, and ensure the app never writes secrets to logs or places them in query strings. |

**A03: SQL Injection**

**Sql injection (Get/Search)**



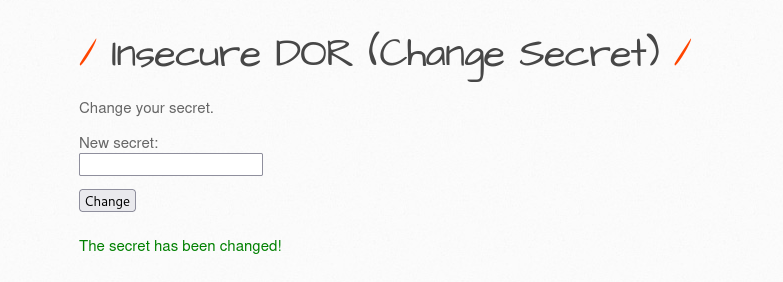
**1. Vulnerability** — The search (GET) parameter is vulnerable to **UNION-based SQL Injection** due to improper input handling; attacker-supplied SQL fragments are concatenated into the query.

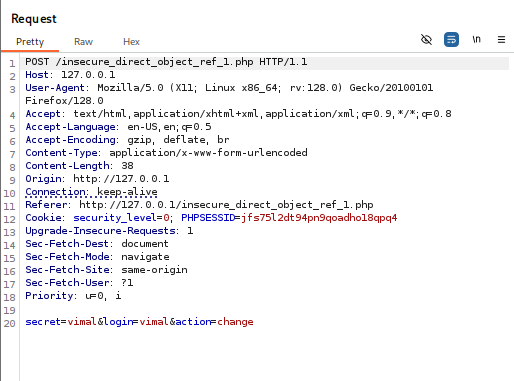
**2.** **Proof / Evidence** — PoC payload used:  
iron' UNION SELECT 1, user(), @@version, 4, 5, 6, 7 #  
Response contained database metadata (e.g. root@localhost from user() and 5.5.47-0ubuntu0.14.04.1 from @@version), confirming successful exploitation.

**3. Impact** — Successful exploitation allows an attacker to disclose sensitive database information, enumerate schema/users, and potentially escalate to full database compromise.

|  |  |  |
| --- | --- | --- |
| SEVERITY | SCORE | MITIGATION |
| CRITICAL | 9.8 | 1. Use Parameterized Queries / Prepared Statements – Ensure all database queries use placeholders instead of directly concatenating user input into SQL statements.  2. Implement Input Validation & Escaping – Enforce strict whitelisting on the search field (e.g., only allow alphanumeric and limited special characters) and properly escape any dynamic input.  3. Apply Least Privilege & Secure Error Handling – Use non-root database accounts with minimum privileges for the web application and suppress detailed database error messages from being displayed to users. |

**A04: Insecure Design**





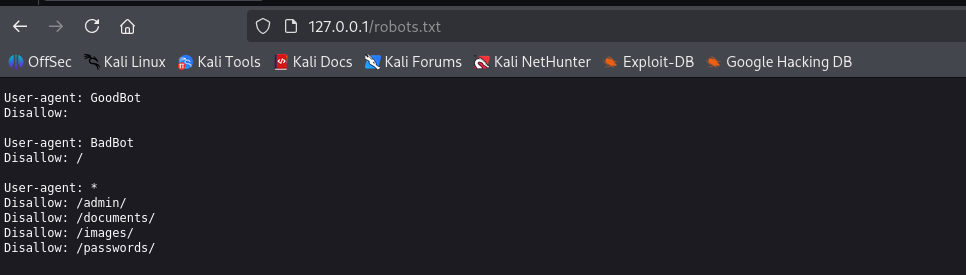
1. **Parameter Manipulation Exploitation** – By modifying the secret parameter value, the application exposed unauthorized data belonging to other users, confirming an IDOR vulnerability.

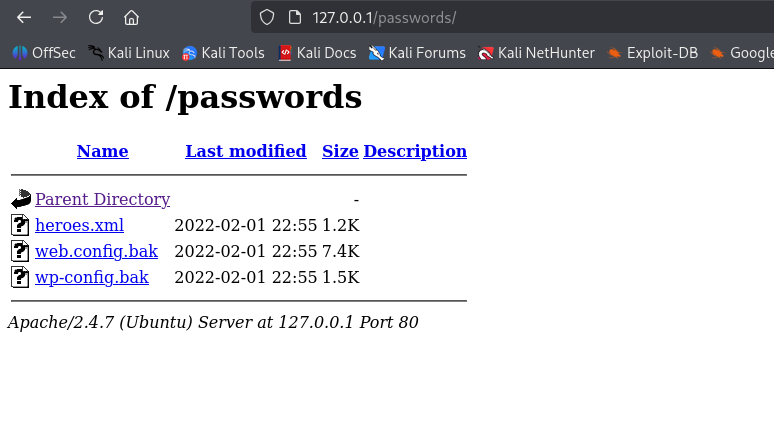
2. **Broken Authorization** **Control** – The backend did not validate user ownership of the requested secret, allowing attackers to directly access objects that do not belong to them.

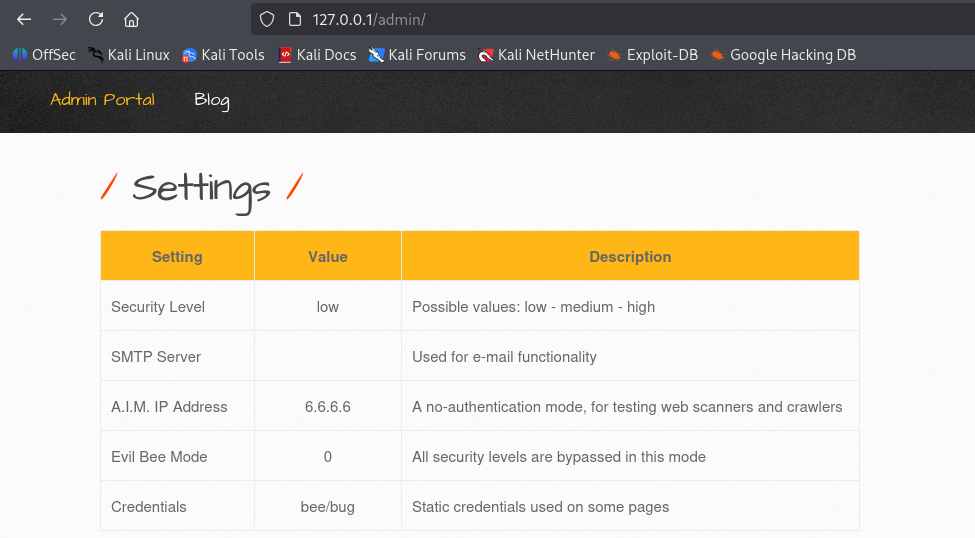
3. **Impact on Data Security** – Successful exploitation could lead to sensitive information disclosure, privilege abuse, or tampering with another user’s resources.

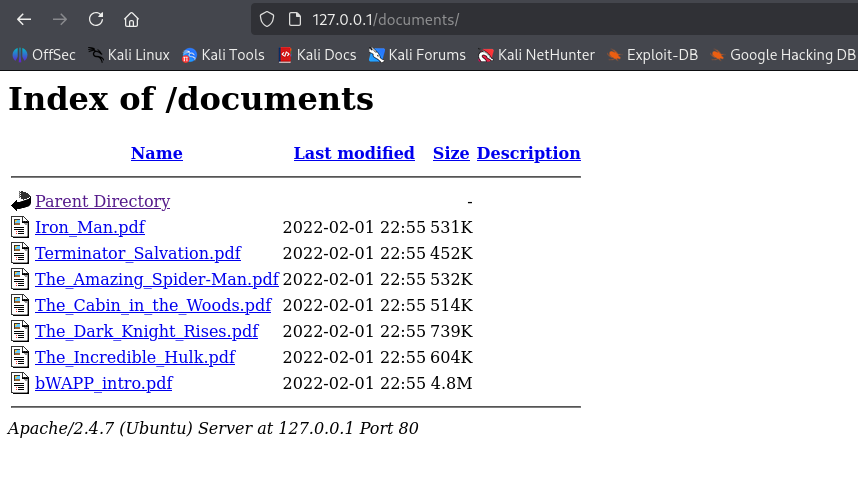
|  |  |  |
| --- | --- | --- |
| SEVERITY | SCORE | MITIGATION |
| HIGH | 8.7 | 1. Implement Proper Authorization Checks – Validate user ownership of objects (like secret) on the server side before granting access.  2. Use Indirect Object References – Replace direct identifiers (e.g., numeric IDs) with randomized or hashed tokens that cannot be easily guessed.  3. Enforce Role-Based Access Control (RBAC) – Ensure users can only access resources based on their roles and permissions, preventing privilege abuse. |

**A05: Security Misconfiguration**









**1.** **Exposed Sensitive Files** – The application allows access to administrative passwords, documents, and images without proper access restrictions.

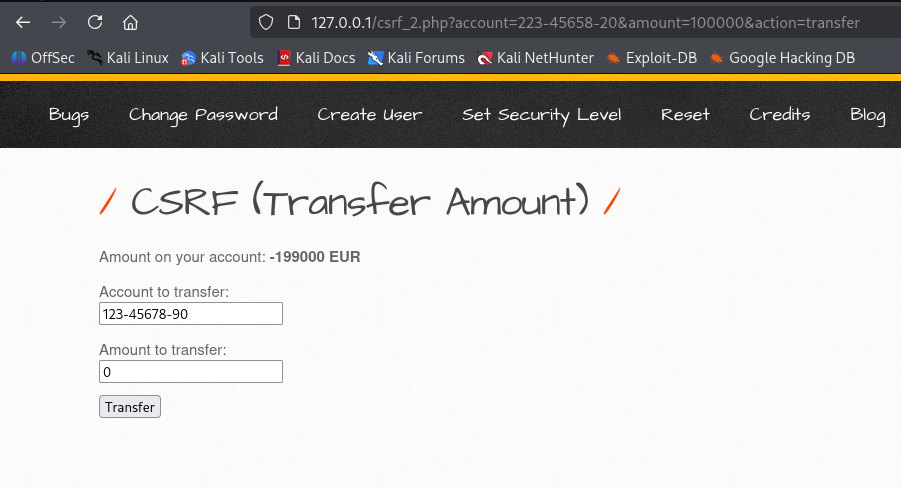
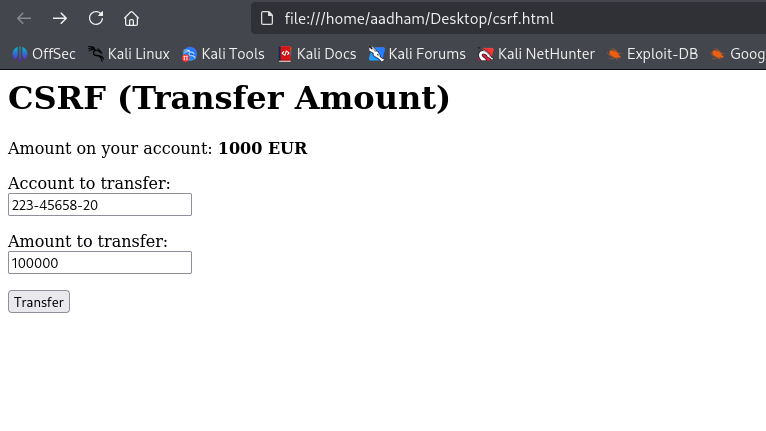
**2.** **Improper Security Settings** – Default configurations or missing access controls enable attackers to view or download sensitive data directly.

**3.** **Risk of Data Compromise** – Unauthorized access could lead to credential theft, data leakage, and further exploitation of the system.

|  |  |  |
| --- | --- | --- |
| SEVERITY | SCORE | MITIGATION |
| CRITICAL | 9.1 | 1. Restrict Access to Sensitive Files – Ensure that administrative files, passwords, and documents are not publicly accessible; apply proper file permissions and authentication checks.  2. Harden Server and Application Configurations – Disable directory listing, remove default accounts/files, and ensure secure default settings are applied.  3. Implement Strong Access Controls – Enforce role-based access controls (RBAC) and validate user permissions for all resources to prevent unauthorized access. |

# A05: CSRF: Unauthorized Funds Transfer (Transfer Amount)

**Finding ID:** VULN-2025-CSRF-001  
**Title:** Cross-Site Request Forgery (CSRF) — Transfer Amount  
**Affected Component / URL:** http://127.0.0.1/csrf\_2.php (Transfer Amount form)

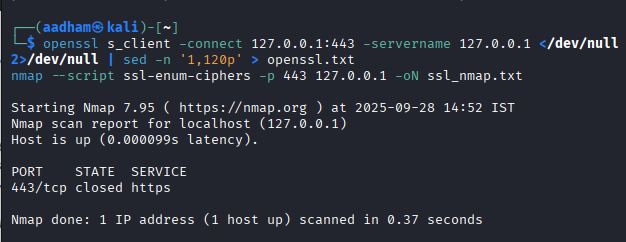


**Impact**

An attacker can cause an authenticated user’s browser to submit an unintended transfer request, resulting in unauthorized funds transfers and financial loss. Because the form uses GET, the attack surface includes simple techniques that require minimal attacker skill and can be triggered by embedding resources or links on attacker-controlled pages.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **SEVERITY** | | **SCORE** | | | **MITIGATION** |
| HIGH | | 7.1 | 1. **Use per-request anti-CSRF tokens:** Include and validate a unique token for each state-changing request. 2. **Use POST for all state-changing actions:** Do not allow GET requests to perform critical actions. 3. **Validate Origin/Referer headers and secure cookies:** Ensure requests originate from trusted domains and set Secure, HttpOnly, and SameSite attributes on session cookies. | | |

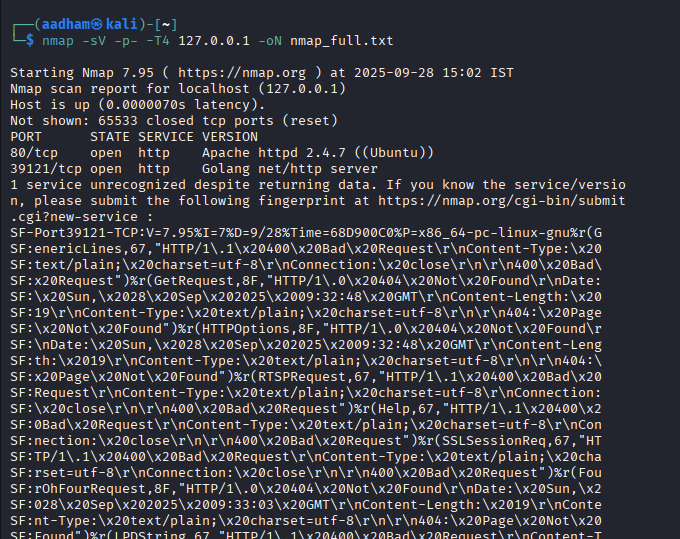
**TLS CERTIFICATION HEALTH :**



**Immediate security implication (prioritized)**

1. **Missing HTTPS / TLS (Observed)** — **Severity: Medium (CVSS-like ~ 5.0)**  
   *Risk:* Traffic to the application may be sent in plaintext (HTTP) — passwords, session cookies and other sensitive data can be intercepted or modified (MITM).  
   *Short-term mitigation (quick win):* Do not use bWAPP instance on a public network without TLS. If testing locally, ensure tests run in an isolated lab or tunnel (ngrok) with TLS.
2. *Long-term mitigation (recommended):*
   1. Enable TLS (bind a certificate) and listen on 443. Use a CA-signed cert for public exposure (Let's Encrypt is fine for labs) or a valid self-signed cert trusted by your test machine if local.
   2. Redirect HTTP → HTTPS (301) and enable Strict-Transport-Security header.
   3. Configure modern TLS: disable TLS 1.0/1.1, enable TLS 1.2+ (1.3 preferred), prefer AEAD ciphers, enable forward secrecy.
   4. Set cookies: Secure; HttpOnly; SameSite=strict where applicable.

**Nmap Scan :**



**Scan summary:**

| **Port** | **State** | **Service** | **Version** | **Notes** |
| --- | --- | --- | --- | --- |
| 80/tcp | open | http | Apache httpd 2.4.7 (Ubuntu) | Default bWAPP HTTP port |
| 39121/tcp | open | http (unknown) | Golang net/http server | Likely a test or auxiliary service, version not fully fingerprinted |
| 443/tcp | closed | https | N/A | No TLS service; no SSL/TLS available |

1. **Other ports:** 65533 closed.
2. **Latency:** negligible (local scan).
3. **Unknown service on 39121/tcp:** Nmap could not fully identify it; shows generic HTTP responses (400/404) — may be a lab/test service.

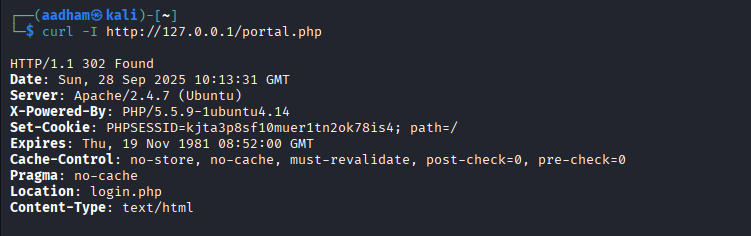
**Observation**:

1. **Apache 2.4.7 on port 80** is vulnerable to **old CVEs** if unpatched (Ubuntu 14.04 or older). Check for CVEs like:
   * CVE-2017-3169 (DoS)
   * CVE-2017-3167 (mod\_http2)
   * Note: CVEs depend on installed patches; version suggests outdated software.
2. **Port 39121 — Golang HTTP server** may be a custom or test service. Default HTTP responses indicate it’s live but may expose debug or error messages.
3. **Port 443 is closed** — TLS/HTTPS is not configured. This is a **medium security risk**, as traffic is unencrypted.

**Mitigation:**

1. **No TLS** (port 443 closed) — all HTTP traffic is plaintext. Sensitive data (credentials, cookies) is exposed.
2. **Outdated Apache** — potential vulnerabilities, especially if this is an older Ubuntu image.
3. **Exposed server banners** — Nmap fingerprints show server versions; attackers could look for known exploits.

**Banner Enumeration:**



**Prioritized findings (what this output shows + risk)**

1. **Missing HTTPS / TLS listener (Observed earlier: 443 closed)** — *Severity: Medium → High if publicly accessible*
   1. **Why:** Traffic is served over HTTP (plain). Credentials and session cookies travel in plaintext and can be intercepted (MITM).
   2. **Mitigation:** Enable HTTPS (bind a certificate and listen on 443), redirect HTTP → HTTPS, enable HSTS (Strict-Transport-Security), and set Secure on cookies.
2. **Outdated Apache (Apache/2.4.7) — possible known CVEs / backported patches uncertain** — *Severity: Medium→High depending on kernel/patch state*.
   1. **Why:** 2.4.7 is an old release; many fixes were made in later 2.4.x versions. Even if a distro backports fixes, running that release is a red flag and increases risk. See Apache 2.4 vulnerabilities listing. [Apache HTTP Server+1](https://httpd.apache.org/security/vulnerabilities_24.html?utm_source=chatgpt.com)
   2. **Mitigation:** Patch/upgrade Apache to a supported version or ensure OS packages are up-to-date (apt-get update && apt-get upgrade on Ubuntu). Verify vendor patches/backports are applied.
3. **Outdated PHP (X-Powered-By: PHP/5.5.9) — known vulnerabilities** — *Severity: High*
   1. **Why:** PHP 5.5.x is EOL and several critical CVEs affect older PHP 5.5 releases (deserialization/use-after-free/exec-related CVEs). Running this version increases risk of RCE or other remote exploit. Examples: CVE-2015-6832 (SPL unserialize use-after-free), other 5.5.x CVEs. [NVD+1](https://nvd.nist.gov/vuln/detail/cve-2015-6832?utm_source=chatgpt.com)
   2. **Mitigation:** Upgrade PHP to a supported release (PHP 7.4/8.x as appropriate) or apply vendor security updates. Remove X-Powered-By header (see below) if upgrade is delayed.
4. **Session cookie missing Secure and HttpOnly flags; missing SameSite** — *Severity: Medium*
   1. **Why:** Set-Cookie: PHPSESSID=...; path=/ has no Secure or HttpOnly attributes (and PHP 5.5 may not support SameSite via ini). Risk: session theft via XSS (no HttpOnly) or exposure over HTTP (no Secure).
   2. **Mitigation:**
      1. When HTTPS enabled, set session.cookie\_secure = On and session.cookie\_httponly = On in php.ini or set cookies manually:
      2. session\_set\_cookie\_params(['secure' => true, 'httponly' => true, 'samesite' => 'Strict']);
      3. session\_start();
      4. For PHP 5.5 you may need to set HttpOnly via session.cookie\_httponly = 1 and implement SameSite by custom cookie logic or upgrade PHP.
5. **Server and technology banners exposed (Server / X-Powered-By)** — *Severity: Low → Medium*
   1. **Why:** Server: Apache/2.4.7 (Ubuntu) and X-Powered-By: PHP/5.5.9 help attackers fingerprint software and target known CVEs.
   2. **Mitigation:** Disable or minimize headers: in Apache ServerSignature Off and ServerTokens Prod; remove X-Powered-By with expose\_php = Off in php.ini.
6. **302 redirect to login.php (normal for bWAPP) — but could leak sensitive pages or allow open redirects if used improperly** — *Severity: Low*
   1. **Why:** Redirect itself is fine; check login page for issues (brute force, lack of CSRF, weak password controls).
   2. **Mitigation:** Harden authentication: strong password policy, account lockout, CSRF on login-sensitive actions, rate limiting.