# Penetration Test Report Comprehensive Security Assessment of IT Services Company Network Infrastructure

Reported By –

**Swarnav Das**.

**Index**

1. Index
2. Acknowledgement.
3. Graphical Overview.
4. Executive Summary
5. Consolidated Summary of key Finding.
6. Key Recommendations
7. Introduction

1. Assessment context and Scope
2. Objectives
3. Methodology
4. Findings and Vulnerabilities.

* + 1. PHP Unsupported Version Detection.
    2. Exposed Sensitive Directories and Files.
    3. Insecure Handling of Sensitive User Data.
    4. SQL Injection and Cross-Site Scripting (XSS).
    5. Over-Privileged Database user.
    6. Insecure Cross-Domain Policies.
    7. Prediction Auto- Incrementing IDs Leading to Insecure Direct Object Reference.
    8. Vulnerable Stored Routine with Dynamic SQL Execution.
    9. Missing HTTP Security Headers.
    10. Local Host Environment Vulnerabilities.
    11. Database Engine Configuration and Transaction Integrity.
    12. Other Informational Findings.

4. Recommendations and Remediation

1. Aggregated Risk Profile.
2. Key Risk Factors.
3. Alignment with Industry Standards.

5. Recommendations And Remediation Guidance

* + 1. Prioritization of Remediation Effort.
    2. General Secure Development Best Practice.

6.Appendices

1. References to Security Standards.
2. Glossary to Technical Terms.
3. Detail of **acuart** Database Schema Information.
4. Raw Scan Output.
5. Resource Links.

**Acknowledgments**

I would like to express my sincere gratitude to the E&ICT Academy at IIT Kanpur for providing me with the opportunity to undertake this project.

My deepest thanks go to my instructor, Mr. **Rahul Gupta**, for his invaluable guidance, support, and expert insights throughout this project. His mentorship was instrumental in the successful completion of this work.

I am also grateful to all the individuals who contributed to this project, and whose support and encouragement were a great help**.**

**- Swarnav Das.**

4th August 2025

|  |  |  |  |
| --- | --- | --- | --- |
| **Critical** | **High** | **Medium** | **Low** |

**Scale Info.**

**Graphical Overview**

## 1. Executive Summary

This report presents a comprehensive overview of a penetration test conducted on two distinct yet interconnected environments: the web application **testphp.vulnweb.com** and a presumed backend **localhost environment** identified by the IP address **192.168.43.170**. The assessment employed a multi-faceted approach, integrating findings from automated scanning tools such as Nmap, Nikto, Dirb, and Nessus, with a meticulous static analysis of provided database metadata, schema definitions, and dynamic observation of active database processes.

A critical contextual detail for testphp.vulnweb.com is its explicit designation as a demonstration site for web vulnerability scanners. This fundamental characteristic is essential for interpreting the nature and severity of the identified vulnerabilities. The weaknesses discovered on this web server are not accidental flaws in a production system but rather deliberately engineered for educational and testing purposes. This understanding shifts the report's focus from merely identifying and remediating flaws to highlighting common attack vectors and underscoring the paramount importance of secure coding practices and robust configurations in real-world applications.

Based on the aggregation of findings from both environments, the combined testphp.vulnweb.com web server and 192.168.43.170 localhost environment presents a **Critical overall risk posture**. This elevated risk level is primarily attributable to a confluence of fundamental security design flaws, pervasive and actively exploited injection vulnerabilities, an over-privileged database user, and critical infrastructure misconfigurations.

### Consolidated Summary of Key Findings

|  |  |  |  |
| --- | --- | --- | --- |
| Severity | Vulnerability Title | Affected Components | Brief Description |
| **Critical** | PHP Unsupported Version Detection | testphp.vulnweb.com (Nginx/PHP) | Web server running End-of-Life PHP 5.6.40-38, lacking security patches and inherently vulnerable to RCE, SQLi, XSS. |
| **Critical** | Exposed Sensitive Directories and Files | testphp.vulnweb.com (Web Server) | Openly accessible /CVS/, /admin/, /secured/, /vendor/ directories, leading to source code, administrative interface, and sensitive data exposure. |
| **Critical** | Insecure Storage of User Credentials and Payment Information | 192.168.43.170 (acuart.users table) | Passwords stored with plaintext default ('pass') and credit card data without encryption/tokenization, enabling account takeover and fraud. |
| **Critical** | Confirmed SQL Injection and Active Exploitation Attempts | 192.168.43.170 (acuart database) | Numerous text fields susceptible to SQLi/XSS; active time-based blind SQLi and info disclosure attempts observed. |
| **Critical** | Over-Privileged Database User | 192.168.43.170 (acuart@localhost user) | acuart user possesses excessive privileges (DROP, ALTER, EXECUTE), violating least privilege and amplifying compromise impact. |
| **High** | Insecure Cross-Domain Policies | testphp.vulnweb.com (Web Server) | Client accesspolicy.xml and crossdomain.xml contain wildcard entries, bypassing Same-Origin Policy and enabling CSRF/XSS data exfiltration. |
| **Medium** | SMB Signing Not Required | 192.168.43.170 (SMB Server) | SMB server does not require message signing, making it vulnerable to Man-in-the-Middle (MitM) attacks. |
| **Medium** | Vulnerable Stored Routines with Dynamic SQL Execution | 192.168.43.170 (MySQL System Routines) | sys.execute\_prepared\_stmt and sys.create\_synonym\_db callable by over-privileged acuart user, posing privilege escalation risk. |
| **Medium** | Predictable Auto-Incrementing IDs Leading to IDOR | 192.168.43.170 (acuart tables) | Sequential primary keys (artist\_id, cat\_id, etc.) enable easy enumeration, risking unauthorized access if authorization is weak. |
| **Medium** | Missing HTTP Security Headers | testphp.vulnweb.com (Web Application) | Absence of X-Frame-Options and X-Content-Type-Options headers, weakening client-side security against clickjacking and MIME type sniffing. |
| **Low** | ICMP Timestamp Request Remote Date Disclosure | testphp.vulnweb.com, 192.168.43.170 | Hosts respond to ICMP timestamp requests, disclosing system date and potentially aiding time-based attacks. |
| **High** | Exposed Database Structures (VIEW\_TABLE\_USAGE.csv) | testphp.vulnweb.com (Database Metadata) | Extensive internal database structure details exposed, significantly aiding targeted SQL injection. |

### Key Recommendations

To significantly improve the security posture and mitigate the identified risks across both environments, immediate and decisive action is imperative. The most urgent remediation actions include:

* **Web Server:** Immediate upgrade of PHP to a currently supported and patched version. Implementation of strict access controls for sensitive directories and database metadata.
* **Localhost/Database:** Mandatory implementation of strong cryptographic hashing for passwords and secure handling (tokenization/encryption) of credit card data. Universal adoption of parameterized queries and rigorous input validation/output encoding for all user-supplied data. Strict adherence to the principle of least privilege for all database users. Enforcement of SMB message signing and disabling of SMBv1 on the localhost.

## 2. Introduction

### Assessment Context and Scope

This assessment constitutes a white-box security review, conducted through the static analysis of provided database schema files and related configuration data, complemented by the analysis of a dynamic Nessus scan report for the localhost environment, and direct observation of active database processes. The primary data sources for this review included testphp.vulnweb.com database ( **COLUMNS.csv, COLUMNS\_EXTENSIONS.csv, ENGINES.csv, CHARACTER\_SETS.csv, COLLATIONS.csv, COLLATION\_CHARACTER\_SET\_APPLICABILITY.csv,ADMINISTRABLE\_ROLE\_AUTHORIZATIONS.csv, INNODB\_TABLESPACES\_BRIEF.csv, INNODB\_DATAFILES.csv, KEY\_COLUMN\_USAGE.csv, localhost\_fnt6gc.pdf (Nessus scan report), ROUTINES.csv, PARAMETERS.csv, PROCESSLIST.csv, SCHEMA\_PRIVILEGES.csv,TABLE\_CONSTRAINTS\_EXTENSIONS.csv,TABLES.csv,TABLES\_EXTENSIONS.csv, TABLESPACES\_EXTENSIONS.csv, SCHEMATA.csv, SCHEMATA\_EXTENSIONS.csv, ST\_SPATIAL\_REFERENCE\_SYSTEMS.csv, ST\_UNITS\_OF\_MEASURE.csv, STATISTICS.csv, TABLE\_CONSTRAINTS.csv, PLUGINS.csv, INNODB\_FT\_DEFAULT\_STOPWORD.csv, INNODB\_CMPMEM\_RESET.csv, and INNODB\_FIELDS.csv )**.

It is important to emphasize that while the Nessus scan provides active findings and the process list shows live queries, the database schema analysis is based solely on the structural and configuration details inferred from the provided documents. The assessment operates under the assumption that the acuart database serves as the primary data store for a web application accessible via a web server and localhost environment.

### Objectives

The primary objectives of this simulated penetration test were meticulously defined to ensure a focused and impactful analysis:

* To identify potential vulnerabilities arising from database design choices, specific data types used for sensitive information, default values, and inferred configurations within the acuart schema.
* To identify active vulnerabilities and informational findings on the localhost environment through a simulated network scan.
* To identify and analyze active database processes for signs of exploitation or unusual activity.
* To assess the theoretical and confirmed risk associated with these potential vulnerabilities, considering how they might be exploited in a live web server environment.
* To provide actionable recommendations for mitigating these identified risks and, consequently, improving the overall security posture of the database backend and its hosting environment.

### Methodology

The methodology employed involved a systematic and in-depth static analysis of the provided database schema and metadata, combined with the interpretation of a Nessus vulnerability scan report and the MySQL process list. This process was guided by industry-standard penetration testing phases and vulnerability categories, ensuring a comprehensive and structured approach.

#### Information Gathering

This initial phase involved a meticulous examination of the provided CSV files and the Nessus scan report. Detailed analysis was conducted on database column names, their assigned data types, specified lengths, default values, and associated privileges. This granular examination was crucial for understanding the potential sensitivity of the data, identifying likely input points for malicious actors, and mapping out the overall attack surfaces within the database structure. For instance, the presence of various text-based fields and user-related tables was noted as potential interaction points for web application attacks. This phase also extended to reviewing the capabilities of different database engines, as detailed in ENGINES.csv, and understanding the implications of various character set and collation configurations. The Nessus scan report provided direct insights into open ports, running services, operating system details, and specific network-level vulnerabilities on the localhost. Crucially, the PROCESSLIST.csv provided a snapshot of active queries, revealing potential exploitation attempts. The ROUTINES.csv and PARAMETERS.csv were analyzed to understand the available stored procedures and functions, their definers, and security types. The SCHEMA\_PRIVILEGES.csv was used to confirm the exact privileges granted to the acuart database user. Additional files like TABLES.csv provided further details on table properties, including AUTO\_INCREMENT values and confirmed engine usage.

#### Vulnerability Identification

Following information gathering, the identified schema details, scan findings, and process list observations were correlated with known web application and database vulnerabilities. This involved a careful assessment for several categories of weaknesses:

* **Broken Authentication and Insecure Credential Storage:** Examining how user credentials might be stored or handled based on column definitions, especially concerning default values and data types that could imply plaintext storage.
* **Injection Flaws:** Analyzing the prevalence of text-based input fields that, if not properly validated and sanitized by the application layer, could be susceptible to SQL Injection or Cross-Site Scripting. Active SQL injection attempts observed in the process list provided direct evidence of exploitability.
* **Broken Access Control and Insecure Direct Object Reference (IDOR) Potential:** Assessing the structure of primary keys and their predictability, which could allow unauthorized access to records if application-level authorization is weak. The presence of AUTO\_INCREMENT values in TABLES.csv further reinforced this.
* **Database Engine Configuration:** Evaluating the implications of database engine choices on the reliability and consistency of data, particularly for transactional operations. The TABLES.csv confirmed the use of InnoDB for acuart tables.
* **Privilege Escalation Potential:** Inferring and confirming risks related to database user privileges based on the PRIVILEGES column in the schema data and the SCHEMA\_PRIVILEGES.csv, and considering the common misconfiguration of overly broad permissions. Analysis of stored routines (ROUTINES.csv, PARAMETERS.csv) was conducted to identify potential privilege escalation vectors.
* **Network and Host-Level Vulnerabilities:** Analyzing Nessus scan results for common misconfigurations, outdated protocols, and information disclosure on the localhost.

#### Risk Assessment

Each identified vulnerability was assigned a conceptual severity rating (Critical, High, Medium, Low, Informational). This rating was based on an evaluation of the potential impact (e.g., data breach, unauthorized access, system compromise) and the theoretical and confirmed exploitability of the vulnerability. This process drew upon common principles of the Common Vulnerability Scoring System (CVSS), as recommended by various security standards.

#### Reporting

The final phase involved meticulously documenting all findings. Each identified vulnerability was presented with clear descriptions, conceptual proofs of concept (reinforced by observed active exploitation attempts), and specific, actionable remediation steps. This structured reporting adheres to the guidelines set forth by organizations such as OWASP and SANS, ensuring that the report is comprehensive, understandable, and directly supports remediation efforts.

## 3. Findings and Vulnerabilities

This section provides a detailed breakdown of each identified potential vulnerability, its specific context within the database schema or localhost environment, the potential impact if exploited, and concrete remediation steps. Each finding is structured for clarity and actionability, aligning with established reporting standards.

To facilitate a clear understanding of the database components discussed, the following table provides an overview of the relevant acuart database schema information derived from the provided COLUMNS.csv.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Table Name** | **Column Name** | **Data Type** | **Column Type** | **IS\_NULLABLE** | **COLUMN\_DEFAULT** | **PRIVILEGES** | **Potential Vulnerabilities** |
| artists | artist\_id | int | int | NO | NULL | select,insert,update,references | Predictable ID for IDOR |
| artists | aname | varchar | varchar(50) | NO | a | select,insert,update,references | Injection (if used in queries) |
| artists | adesc | text | text | NO | NULL | select,insert,update,references | Injection (XSS/SQLi) |
| carts | cart\_id | varchar | varchar(100) | NO | 0 | select,insert,update,references | Injection (if used in queries) |
| carts | price | int | int | NO | 0 | select,insert,update,references | Data Inconsistency (if MyISAM) |
| carts | item | int | int | NO | 0 | select,insert,update,references | Data Inconsistency (if MyISAM) |
| categ | cat\_id | int | int | NO | NULL | select,insert,update,references | Predictable ID for IDOR |
| categ | cname | varchar | varchar(50) | NO | a | select,insert,update,references | Injection (if used in queries) |
| categ | cdesc | tinytext | tinytext | YES | NULL | select,insert,update,references | Injection (XSS/SQLi) |
| featured | pic\_id | int | int | NO | 0 | select,insert,update,references | Predictable ID for IDOR |
| featured | feature\_text | text | text | NO | NULL | select,insert,update,references | Injection (XSS/SQLi) |
| guest  book | sender | varchar | varchar(150) | YES | NULL | select,insert,update,references | Injection (if used in queries) |
| guest  book | mesaj | text | text | YES | NULL | select,insert,update,references | Injection (XSS/SQLi) |
| guest  book | senttime | int | int | YES | NULL | select,insert,update,references |  |
| pictures | pic\_id | int | int | NO | NULL | select,insert,update,references | Predictable ID for IDOR |
| pictures | pshort | mediumtext | mediumtext | NO | NULL | select,insert,update,references | Injection (XSS/SQLi) |
| pictures | plong | text | text | NO | NULL | select,insert,update,references | Injection (XSS/SQLi) |
| pictures | price | int | int | NO | 0 | select,insert,update,references | Data Inconsistency (if MyISAM) |
| pictures | cat\_id | int | int | NO | 0 | select,insert,update,references | Predictable ID for IDOR |
| pictures | a\_id | int | int | NO | 0 | select,insert,update,references | Predictable ID for IDOR |
| pictures | title | varchar | varchar(100) | NO | aaa | select,insert,update,references | Injection (if used in queries) |
| pictures | img | varchar | varchar(50) | NO | ./pictures/ | select,insert,update,references | Path Traversal (if not validated) |
| products | id | int | int unsigned | NO | NULL | select,insert,update,references | Predictable ID for IDOR |
| products | name | text | text | NO | NULL | select,insert,update,references | Injection (XSS/SQLi) |
| products | rewritename | text | text | NO | NULL | select,insert,update,references | Injection (XSS/SQLi) |
| products | description | text | text | NO | NULL | select,insert,update,references | Injection (XSS/SQLi) |
| products | price | int | int unsigned | NO | NULL | select,insert,update,references | Data Inconsistency (if MyISAM) |
| users | uname | varchar | varchar(100) | NO | uname | select,insert,update,references | Insecure Default Credentials |
| users | pass | varchar | varchar(100) | NO | pass | select,insert,update,references | Insecure Password Storage, Insecure Default Credentials |
| users | cc | varchar | varchar(100) | NO | no | select,insert,update,references | Insecure Credit Card Storage |
| users | address | mediumtext | mediumtext | NO | NULL | select,insert,update,references | PII Exposure, Injection (XSS/SQLi) |
| users | email | varchar | varchar(100) | NO | select,insert,update,references | PII Exposure, Injection (if used in queries) | - |
| users | name | varchar | varchar(100) | NO | a | select,insert,update,references | PII Exposure, Injection (if used in queries) |
| users | phone | varchar | varchar(100) | NO | select,insert,update,references | PII Exposure, Injection (if used in queries) | - |
| users | cart | varchar | varchar(100) | NO | 0 | select,insert,update,references | Data Inconsistency (if MyISAM) |

### 3.1. PHP Unsupported Version Detection (testphp.vulnweb.com)

The web server is currently operating with PHP version 5.6.40-38+ubuntu20.04.1+deb.sury.org+1. This particular version of PHP reached its End-of-Life (EOL) on December 31, 2018, meaning it no longer receives official security patches or updates from the vendor.1 The Nessus plugin output (ID 58987) explicitly confirmed the installed version and its EOL date. Additionally, the Nikto scan corroborated this by retrieving the same version information from theX-Powered-By HTTP header.

Potential Impact: The impact of running an unsupported PHP version is **Critical**. Such a system is inherently vulnerable to a multitude of known and potentially undiscovered security flaws, including but not limited to remote code execution (RCE), SQL injection, and cross-site scripting (XSS) vulnerabilities. The absence of ongoing security patches means that any newly discovered vulnerabilities will remain unaddressed, leaving the system perpetually exposed. Successful exploitation of an outdated PHP installation can lead to a complete compromise of the web server and, potentially, the underlying operating system. This level of compromise could allow attackers to steal sensitive data, deface the website, or establish persistent backdoors for future access.

Severity Rating: Critical (CVSS v3.0 Base Score: 10.0)

This critically outdated PHP version is not merely an isolated flaw; it represents a fundamental and pervasive security weakness that likely underpins and enables many other web application vulnerabilities observed. The OSINT data, which includes extensive discussions and tools related to SQL Injection and XSS attacks on vulnerable web applications, strongly suggests that these types of application-level vulnerabilities are present in the target.It is highly probable that the outdated PHP version is a significant contributing factor, if not the direct cause, for these application-specific weaknesses. Attackers frequently leverage publicly known exploits for EOL software to gain initial access or escalate privileges within a system. Consequently, addressing this critical vulnerability by upgrading PHP is not just a single fix; it is a foundational security improvement that will inherently mitigate a wide array of other potential web application vulnerabilities, thereby substantially reducing the overall attack surface and systemic risk.

### 3.2. Exposed Sensitive Directories and Files (testphp.vulnweb.com)

The dirb scan identified several directories and files that are openly accessible on the web server, which should ideally be protected or not exposed in a production environment.These exposures provide attackers with crucial reconnaissance information, potential access to source code, administrative interfaces, and sensitive data, significantly increasing the likelihood and impact of a successful system compromise.

##### Specific Exposures:

* **/CVS/ Directory and Contents**: The entire CVS (Concurrent Versions System) directory, along with its critical internal files such as CVS/Entries, CVS/Repository, and CVS/Root, is openly accessible. This represents a **critical information disclosure**. An attacker can download the entire source code repository, which can then be meticulously analyzed offline to identify logical flaws, hardcoded credentials (e.g., database passwords, API keys), and other vulnerabilities. This significantly streamlines the development of highly targeted exploits, potentially leading to remote code execution or a complete compromise of the application.
* **/admin/ Directory**: The administrative interface directory is directly accessible. This poses a **high risk of unauthorized administrative access**. The open accessibility of this directory suggests weak or absent authentication and authorization controls. This allows attackers to easily initiate brute-force attacks or attempt to exploit weak or default credentials to gain full control over the web application's backend and potentially the underlying server.
* **/secured/ Directory**: A directory explicitly named "secured" is also openly accessible. This is a significant **sensitive data exposure**. The naming convention strongly implies that this directory is intended to house sensitive or restricted content. Its accessibility without any apparent authentication or authorization mechanism (indicated by a 200 OK response rather than a 401 Unauthorized or 403 Forbidden) is a severe security flaw, likely leading to direct data breaches or the exposure of confidential information.
* **/vendor/ Directory**: The directory containing third-party libraries and dependencies is accessible. This constitutes **information disclosure regarding software versions**. Exposing this directory reveals the specific versions of frameworks, libraries, and other third-party components utilized by the application. This information is invaluable to attackers as it allows them to quickly identify known vulnerabilities (CVEs) associated with those specific component versions and subsequently craft highly targeted exploits.

Severity Rating: Critical (CVSS: 9.8/10), reflecting a high impact on Confidentiality, Integrity, and Availability, coupled with high exploitability.

The combination of exposed CVS, administrative, "secured," and vendor directories creates a highly permissive and vulnerable environment. Each individual exposure, while significant on its own, synergistically contributes to an attacker's ability to compromise the system. For instance, source code obtained from the /CVS/ directory might reveal default administrative credentials or specific SQL injection points, which can then be directly leveraged to exploit the /admin/ panel or extract data from the database. Knowledge of software versions from the /vendor/ directory can lead to the exploitation of known vulnerabilities in those libraries. The content of the "secured" directory, when combined with other gathered information, could enable identity theft or further system access. This interconnectedness significantly reduces the time and effort an attacker requires for a full system compromise. Organizations must adopt a "defense-in-depth" strategy, ensuring that even if one layer of defense is bypassed, subsequent layers prevent a complete system compromise. Open directory listings represent a fundamental failure in this multi-layered defense, essentially providing an attacker with a clear roadmap to critical assets.

### 3.3. Insecure Handling of Sensitive User Data (Authentication & Authorization Failures)

#### Vulnerability Title: Insecure Storage of User Credentials and Payment Information

The users table, a critical component for user management within the acuart database, contains columns designated for highly sensitive data: pass (password) and cc (credit card number). The schema explicitly defines the pass column with a default value of 'pass', and the cc column with a default value of 'no'.

Storing passwords in a varchar(100) field without any indication of strong cryptographic hashing, and particularly with a plaintext default, represents a fundamental and critical security flaw. Similarly, the storage of credit card numbers without robust encryption or tokenization is a severe vulnerability. Furthermore, other personally identifiable information (PII) such as address, email, name, and phone are stored alongside these insecure credentials, significantly amplifying the potential impact of a data breach.

Affected Database Components: The primary affected component is the acuart.users table, specifically the uname, pass, cc, address, email, name, and phone columns.

Potential Impact: The potential impact of this vulnerability is **Critical**. Immediate and widespread account compromise is possible for any users whose credentials remain at their default values. A subsequent compromise of the database could lead to mass unauthorized access to user accounts, complete theft of PII, and direct financial fraud via unencrypted credit card numbers. Such an incident could result in severe reputational damage, significant financial losses for both the organization and its customers, and substantial legal and regulatory penalties, including non-compliance with regulations like GDPR and PCI-DSS.

Severity Rating: Critical (CVSS: 9.8/10), reflecting a high impact on Confidentiality, Integrity, and Availability, coupled with high exploitability.

Conceptual Proof of Concept: An attacker, aware of common default credentials, could attempt to log in to the web application using the default username 'uname' and password 'pass'. If these default credentials are still active, the attacker gains full, unauthorized access to a default user account, potentially leading to privilege escalation if the default user possesses elevated rights within the application. Even if default credentials have been changed, the

varchar(100) data type for the pass column strongly suggests that passwords might be stored in plain text or with weak, reversible hashing. A successful database breach, perhaps through a SQL Injection attack, would immediately expose all user passwords, regardless of their complexity. Similarly, direct access to the cc (credit card) information, if stored unencrypted, would allow for immediate financial exploitation by the attacker, enabling fraudulent transactions.

The explicit default value of 'pass' for users.pass and the presence of users.cc (credit card) in a varchar(100) field are not merely weak configurations; they indicate a fundamental and severe lack of secure coding practices and security awareness during the application's development lifecycle. This situation points to a systemic issue where basic security principles, such as "never store plaintext passwords" and "never store sensitive financial data without strong encryption or tokenization," were either unknown or disregarded. The direct consequence is clear: a deficiency in secure development practices leads directly to critical vulnerabilities that expose the most sensitive user data. This type of critical vulnerability extends far beyond technical compromise. In a real-world scenario, such a security lapse can lead to devastating reputational damage, severe regulatory fines (e.g., GDPR, CCPA, PCI-DSS non-compliance), and significant financial losses. More broadly, it suggests that other parts of the application might suffer from similar fundamental security weaknesses, indicating a need for a comprehensive security review and overhaul of the entire development process.

### 3.4. SQL Injection and Cross-Site Scripting (XSS) Vulnerabilities

#### Vulnerability Title: Confirmed SQL Injection and Active Exploitation Attempts

The database schema contains multiple tables with large text fields (text, mediumtext) intended for user-supplied content.These fields are prime targets for various injection attacks, including SQL Injection (SQLi) and Cross-Site Scripting (XSS), if the web application does not implement rigorous input validation, sanitization, and context-aware output encoding. The schema itself, by allowing such broad data types, places the full burden of security on the application layer. Crucially, analysis of the

PROCESSLIST.csv reveals active queries that are characteristic of SQL injection attempts, confirming the exploitability of these vulnerabilities.

Affected Database Components: The database components primarily affected by this potential and confirmed vulnerability include: acuart.guestbook.mesaj (text), likely for user comments; acuart.pictures.pshort (mediumtext) for short picture descriptions; acuart.pictures.plong (text) for long picture descriptions; acuart.products.name (text) for product names; acuart.products.rewritename (text) for rewritten product names (e.g., for URLs); and acuart.products.description (text) for product descriptions.Active exploitation attempts were observed targetingpictures.title, pictures.pshort, users.uname, and artists.artist\_id.

Potential Impact: The potential impact of successful SQL Injection is **Critical**, as it could lead to unauthorized data access, modification, or deletion of sensitive information across the entire database. In certain configurations, it can also enable database compromise, privilege escalation, and even remote code execution on the underlying server. The observed queries indicate attempts at time-based blind SQL injection (leading to denial of service or slow data exfiltration) and information disclosure from information\_schema.For Cross-Site Scripting (XSS), the potential impact is

**Medium**, as successful attacks could lead to session hijacking (compromising user accounts), website defacement, redirection to malicious sites, phishing attacks, or the distribution of malware to end-users browsing the affected web application.

Severity Rating: Critical (CVSS: 9.8/10), reflecting high exploitability and significant impact on data integrity and confidentiality, further elevated by observed active exploitation.

Conceptual Proof of Concept (Confirmed by Active Processes):

* **Time-Based Blind SQL Injection:** The PROCESSLIST.csv shows queries like: SELECT a.\*, b.aname, b.artist\_id, c.cname FROM pictures a, artists b, categ c WHERE a.cat\_id=c.cat\_id AND a.a\_id=b.artist\_id AND a.pic\_id=(select\*from(select(sleep(20)))a) and SELECT \* FROM users WHERE uname='X'XOR(if(now()=sysdate(),(sleep(6)),0))XOR'X' AND pass='J1DMDU0\_3OO'.1 These queries demonstrate attempts to inject  
  SLEEP() functions to infer information based on response time, a common technique for blind SQL injection.
* **Information Disclosure via Union-Based SQL Injection:** A query observed in the process list is: SELECT \* FROM artists WHERE artist\_id=-1412 UNION ALL SELECT NULL,NULL,CONCAT(0x7171717671,JSON\_ARRAYAGG(CONCAT\_WS(0x766b736d707a,IFNULL(CAST(COMMAND AS CHAR),0x20),IFNULL(CAST(DB AS CHAR),0x20),IFNULL(CAST(ID AS CHAR),0x20),IFNULL(CAST(INFO AS CHAR),0x20),IFNULL(CAST(STATE AS CHAR),0x20),IFNULL(CAST(HOST AS CHAR),0x20),IFNULL(CAST(TIME AS CHAR),0x20),IFNULL(CAST(USER AS CHAR),0x20))),0x7171717171) FROM information\_schema.PROCESSLIST-- -.This query attempts to extract data from the  
  information\_schema.PROCESSLIST table using a UNION ALL attack, demonstrating an attempt to enumerate active processes and potentially other sensitive database metadata.
* **Cross-Site Scripting (XSS):** While not directly observed in the process list, the presence of large TEXT and MEDIUMTEXT fields for user input (e.g., guestbook.mesaj, pictures.pshort, pictures.plong) makes the application highly susceptible to XSS if output encoding is not properly implemented. An attacker could embed a JavaScript payload (e.g.,  
  <script>alert(document.cookie)</script>) into such fields, which would execute in a victim's browser when the content is displayed.

The widespread use of TEXT and MEDIUMTEXT data types for user-facing content, combined with the inherent lack of explicit input validation mechanisms within the database schema itself, points to a pervasive vulnerability pattern in web applications. This is a design-level concern where the database is prepared to accept arbitrary large strings, effectively placing the full burden of security on the application layer. This often leads to developers overlooking or inconsistently implementing validation and encoding. The active SQL injection attempts observed in the

PROCESSLIST.csv elevate this from a theoretical risk to a confirmed, actively exploited vulnerability, making it a critical entry point for malicious actors. A successful injection attack can serve as a gateway to further attacks, including privilege escalation, data exfiltration, or even server compromise.

### 3.5. Over-Privileged Database User

#### Vulnerability Title: Over-Privileged Database User Confirmed

The SCHEMA\_PRIVILEGES.csv explicitly confirms that the acuart@localhost database user possesses a wide array of highly permissive privileges on the acuart schema.These include

SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, REFERENCES, INDEX, ALTER, CREATE TEMPORARY TABLES, LOCK TABLES, and EXECUTE privileges. While

SELECT, INSERT, UPDATE, and DELETE are typically required for a web application, the presence of CREATE, DROP, ALTER, INDEX, and EXECUTE privileges is highly excessive and violates the principle of least privilege.

Affected Database Components: The acuart@localhost database user and, by extension, all tables within the acuart schema are directly affected.

Potential Impact: The potential impact is **Critical**. If the web application's database user account is compromised (e.g., via a successful SQL Injection attack or leaked credentials), an attacker could gain extensive and destructive control over the database. With DROP and ALTER privileges, an attacker could delete entire tables, modify schema definitions, or corrupt data. EXECUTE privilege allows the attacker to run stored procedures and functions, potentially leading to further privilege escalation or system compromise (as discussed in Section 3.8). The ability to CREATE TEMPORARY TABLES could be abused for data exfiltration or staging malicious payloads. This level of access allows for complete data compromise, denial of service, and potentially persistent backdoor creation.

Severity Rating: Critical (CVSS: 9.8/10), reflecting a high impact on confidentiality, integrity, and availability, with high exploitability due to confirmed excessive privileges.

Conceptual Proof of Concept: An attacker successfully exploiting an SQL Injection vulnerability (as confirmed in Section 3.4) could leverage the acuart user's DROP privilege to execute DROP TABLE users; to delete the entire user table, causing a severe denial of service and data loss.Alternatively, using the

ALTER privilege, the attacker could modify table schemas to exfiltrate data more easily or introduce backdoors. With the EXECUTE privilege, the attacker could call vulnerable stored procedures (e.g., sys.execute\_prepared\_stmt as discussed in Section 3.8) to perform actions beyond the application's intended scope, potentially escalating privileges to the DEFINER of those routines.

The explicit confirmation of excessive privileges for the acuart user is a critical finding. This is not an inference but a direct observation of a severe security misconfiguration. This level of privilege means that any successful compromise of the web application (e.g., via SQL injection) immediately escalates to a full database compromise, allowing an attacker to cause widespread damage, including data destruction and schema manipulation. This fundamental violation of the principle of least privilege is a common and dangerous architectural flaw that significantly amplifies the impact of other vulnerabilities. It underscores the urgent need for a comprehensive review of all database user permissions.

### 3.6. Insecure Cross-Domain Policies (testphp.vulnweb.com)

Both the clientaccesspolicy.xml (used by Microsoft Silverlight) and crossdomain.xml (used by Adobe Flash) policy files contain full wildcard entries (\*). The Nikto scan explicitly reported these full wildcard entries for both files, and Nessus also confirmed their presence and advised a careful review for improper policies. The dirb scan further confirmed the existence of crossdomain.xml.

Potential Impact: The impact of these misconfigurations is **High**. These permissive policies effectively bypass the browser's Same-Origin Policy (SOP), a fundamental security mechanism designed to prevent web pages from interacting with resources from different origins. By allowing full wildcard access, the application becomes highly vulnerable to Cross-Site Request Forgery (CSRF), Cross-Site Scripting (XSS) data exfiltration, and other cross-domain attacks originating from malicious websites. An attacker could potentially embed the vulnerable site within an iframe on their own malicious domain and programmatically interact with it, including making authenticated requests on behalf of a victim user or reading sensitive data that would otherwise be restricted. These permissive cross-domain policies act as a force multiplier for client-side vulnerabilities, significantly broadening the attack surface. If a user is authenticated to

testphp.vulnweb.com and subsequently navigates to an attacker-controlled website, the attacker's site could, due to these overly permissive policies, initiate authenticated requests to testphp.vulnweb.com on behalf of the user (CSRF). Alternatively, it could read sensitive data directly from the vulnerable domain (XSS data exfiltration), even if the XSS vulnerability itself resides on a different page or is triggered through a separate vector. This significantly lowers the barrier for successful exploitation of client-side vulnerabilities. Implementing proper and restrictive cross-domain policies is a critical component of a robust web application security posture, essential for preventing malicious interactions between untrusted origins and the application.

Severity Rating: High (CVSS: 8.5/10), reflecting a significant impact on confidentiality and integrity due to SOP bypass and high exploitability.

### 3.7. Predictable Auto-Incrementing IDs Leading to Insecure Direct Object Reference (IDOR)

#### Vulnerability Title: Predictable Auto-Incrementing IDs Leading to Insecure Direct Object Reference (IDOR)

Several key tables within the acuart database utilize auto-incrementing integer primary keys, such as artist\_id in artists, cat\_id in categ, pic\_id in pictures, and id for products. While this is a common and convenient practice for database management, this sequential numbering makes these identifiers highly predictable and easy for an attacker to guess and enumerate other records. This can be achieved by simply incrementing or decrementing the ID value in a URL parameter or API request. For instance, the

artists table has an AUTO\_INCREMENT value of 4, and the categ table has an AUTO\_INCREMENT value of 5, indicating sequential numbering.This predictability becomes a critical vulnerability if robust access controls are not strictly enforced at the application layer to verify user authorization for each requested resource.

Affected Database Components: The database components primarily affected include acuart.artists.artist\_id, acuart.categ.cat\_id, acuart.pictures.pic\_id, and acuart.products.id.

Potential Impact: The potential impact is **Medium**, allowing unauthorized access to, viewing, or manipulation of other users' or system's data that should be restricted. For example, an attacker could potentially view private artist details, modify product information, or access categories they are not authorized to manage. A **Low** impact could involve the leakage of non-sensitive but private information, thereby undermining data confidentiality.

Severity Rating: Medium (CVSS: 6.5/10), reflecting a moderate impact on confidentiality and high exploitability.

Conceptual Proof of Concept: A legitimate user accesses a resource, for instance, viewing their artist profile via a URL similar to https://example.com/artists?id=100. An attacker, observing the predictable id parameter, modifies the URL to https://example.com/artists?id=101 or https://example.com/artists?id=99. If the web application does not perform a server-side authorization check to verify that the currently authenticated user is authorized to access the data associated with artist\_id=101, the attacker gains unauthorized access to another artist's private data. This pattern applies to cat\_id, pic\_id, and products.id as well, enabling enumeration and access to other categories, pictures, or product details.

The common and convenient practice of using auto-incrementing integer IDs for primary keys is efficient for database management but inherently introduces a significant authorization risk in web applications if not coupled with strict, server-side access control. This is a recurring design pattern that frequently leads to IDOR vulnerabilities (OWASP Top 10 A01:2021 Broken Access Control). The underlying assumption is that the application layer will always handle authorization correctly, which is often a point of failure. Predictable IDs are not a vulnerability by themselves; they become a vulnerability when the application layer fails to implement proper authorization checks. This highlights a common design flaw where developers prioritize ease of use (sequential IDs) over robust security. While IDOR vulnerabilities may not always lead to critical data breaches, they consistently erode user trust, undermine the principle of least privilege, and can expose sensitive business logic or data that was intended to be private. It underscores the critical importance of a "defense-in-depth" strategy, where security is not solely reliant on a single layer (e.g., client-side checks) but is built into every layer of the application, from database design to API endpoints and user interface. If IDOR is present, it suggests a general weakness in access control mechanisms throughout the application, implying that other forms of privilege escalation might also exist.

### 3.8. Vulnerable Stored Routines with Dynamic SQL Execution

#### Vulnerability Title: Vulnerable Stored Routines with Dynamic SQL Execution

The MySQL database contains several system-level stored routines (functions and procedures) that are defined with DEFINER as mysql.sys@localhost and SECURITY\_TYPE as INVOKER. This means these routines execute with the privileges of the user calling them. While many of these routines are benign (e.g., format\_bytes, format\_time), some, like sys.execute\_prepared\_stmt and sys.create\_synonym\_db, are designed to execute dynamic SQL statements constructed from input parameters.

Given that the acuart user has EXECUTE privilege on the acuart schema, it is highly probable that this user can call these system routines.If user-controlled input is passed unsanitized to these dynamic SQL-executing routines, it could lead to privilege escalation or further compromise, even if the acuart user's direct privileges were restricted.

Affected Database Components: The affected components include MySQL system routines such as sys.execute\_prepared\_stmt and sys.create\_synonym\_db, and any application code that calls these routines with user-supplied input. The acuart database user's EXECUTE privilege is a contributing factor.

Potential Impact: The potential impact is **Medium** to **High**. If an attacker can inject malicious SQL into the parameters of sys.execute\_prepared\_stmt or sys.create\_synonym\_db via the web application, they could execute arbitrary SQL commands with the privileges of the acuart user.

Since theacuart user has extensive privileges, including CREATE, DROP, and ALTER, this could lead to full database compromise, data exfiltration, or denial of service. Even if the acuart user's privileges were reduced, the ability to execute dynamic SQL through these routines could still be a vector for bypassing other security controls or performing unauthorized actions.

Severity Rating: Medium (CVSS: 7.0/10), reflecting moderate exploitability (requires a specific call path) but high potential impact due to the acuart user's broad privileges.

Conceptual Proof of Concept: An attacker identifies a web application endpoint that calls a stored procedure or function, or constructs a query that directly calls a system routine like sys.execute\_prepared\_stmt. The attacker crafts a malicious payload (e.g., CALL sys.execute\_prepared\_stmt('DROP TABLE users;') and injects it into the application's input, which is then passed to the routine. If the application does not properly sanitize the input before passing it to sys.execute\_prepared\_stmt, the routine executes the malicious DROP TABLE command with the acuart user's privileges, leading to data loss. Similarly, sys.create\_synonym\_db could be abused to create views pointing to sensitive system tables, facilitating information gathering.

The presence of system routines that execute dynamic SQL with INVOKER security and the acuart user's EXECUTE privilege creates a dangerous combination. While these routines are part of MySQL's functionality, their misuse or insecure invocation by an over-privileged application user can lead to severe consequences. This highlights a potential avenue for privilege escalation or bypassing other security controls, even if direct table privileges were to be restricted. It emphasizes the need for a holistic security approach that includes not just table permissions but also the control over callable routines.

### 3.9. Missing HTTP Security Headers (testphp.vulnweb.com)

The web application is missing crucial HTTP security headers that are designed to instruct web browsers to enforce specific security mechanisms, thereby protecting users from common client-side attacks.The Nikto scan explicitly reported the absence of these headers. It is important to note that the Nessus scan did not explicitly report on these specific missing headers, highlighting a difference in scanner coverage and the necessity of using multiple tools.

Potential Impact: The impact of these missing headers is assessed as **Medium**. While their absence does not directly introduce a server-side vulnerability, it significantly weakens the client-side security posture. This leaves end-users more susceptible to attacks such as clickjacking and MIME type sniffing, which can lead to information disclosure or further exploitation.

Severity Rating: Medium (CVSS: 6.1/10), reflecting a moderate impact on client-side security and moderate exploitability.

* **X-Frame-Options Header:** This header is not present. Its absence allows the site to be embedded within an <iframe>, <frame>, or <object> tag on another website. This makes the site vulnerable to clickjacking attacks, where malicious content is layered transparently over the legitimate site to trick users into performing unintended actions, such as clicking a hidden button or submitting sensitive information.
* **X-Content-Type-Options Header:** This header is not set. Without this header, web browsers may attempt to "sniff" the content type of files, potentially misinterpreting non-executable files (e.g., user-uploaded images) as executable scripts. This vulnerability, known as MIME type sniffing, can lead to Cross-Site Scripting (XSS) if an attacker manages to upload a specially crafted file that the browser then executes.

These missing headers represent a failure to leverage browser-native security features, effectively shifting some security responsibilities to the client without providing proper instructions. This directly exposes end-users to attacks that could otherwise be mitigated by simple header configurations. For example, a user could be tricked into clicking a hidden button (clickjacking) or executing malicious code (MIME-sniffing XSS) even if the server-side application itself does not have a direct vulnerability. This scenario underscores that security is a multi-layered defense, and client-side protections are as crucial as server-side measures. Implementing standard security headers is a low-cost, high-impact measure that significantly enhances the overall security posture of a web application and provides a stronger defense for its users.

### 3.10. Local Host Environment Vulnerabilities

This section details vulnerabilities and informational findings identified on the localhost environment (**IP 192.168.43.170**) through Nessus and Nmap scans. The host is identified as running Microsoft Windows 10 Education.

#### 3.10.1. Medium Severity Findings

SMB Signing Not Required (Nessus Plugin ID: 57608)

Description: The remote SMB server does not require message signing. This configuration allows an unauthenticated, remote attacker to conduct man-in-the-middle (MitM) attacks against the SMB server, potentially intercepting or altering communications.

Severity Rating: Medium (CVSS: 5.9/10), reflecting a moderate impact on integrity and confidentiality due to MitM risk.

Affected Components: TCP port 445 (cifs) on 192.168.43.170.

#### 3.10.2. Low Severity Findings

ICMP Timestamp Request Remote Date Disclosure (Nessus Plugin ID: 10114)

Description: The remote host responds to ICMP timestamp requests (type 13), which allows an attacker to determine the system's exact date and time. The Nessus plugin output explicitly confirmed this, stating, "The remote clock is synchronized with the local clock".

Severity Rating: Low (CVSS: 3.1/10), reflecting a low impact on confidentiality and low exploitability.

Affected Components: ICMP protocol on 192.168.43.170

#### 3.10.3. Informational Findings and System Enumeration

The Nessus and Nmap scans also identified several informational findings that, while not direct vulnerabilities, provide valuable reconnaissance data to a potential attacker. This information can be leveraged to craft more targeted attacks.

* **SMB Protocol Version 1 Enabled (Nessus Plugin ID: 96982):** The host supports SMBv1, which Microsoft recommends disabling due to a lack of modern security features. While not rated as a direct vulnerability by Nessus in this report, it is considered an outdated and less secure protocol.
* **DCE Services Enumeration (Nessus Plugin ID: 10736):** Various Distributed Computing Environment (DCE/RPC) services were enumerated on ports 135 (epmap), 445 (cifs), and dynamic ports (49664-49671). This exposes details about running services like Security Account Manager (lsass.exe), DHCP Client Service (svchost.exe), Service Control Manager, and others, which could be used for further exploitation.
* **Operating System and Device Information:**

**OS Identification (Nessus Plugin ID: 11936, 209654):** Confirmed Microsoft Windows 10 Education.Multiple OS fingerprints were detected, including various Windows 10 editions and even Windows Server 2019.

**Device Type (Nessus Plugin ID: 54615):** Identified as a "general-purpose" computer.

**VMware Virtual Machine Detection (Nessus Plugin ID: 20094):** The host is identified as a VMware virtual machine based on its MAC address.

**Network Information Disclosure (Nessus Plugin ID: 10785, 10150, 12053, 53513):** The NetBIOS name (DESKTOP-3RRTG48), MAC address (00:00:29:06:E0:C5), SMB versions (SMBv1, SMBv2), and supported SMB2/SMB3 dialects were disclosed. FQDN resolution and LLMNR support also revealed the hostname.

* **Nessus Scan-Related Information:**

**Nessus Cannot Access Windows Registry (Nessus Plugin ID: 26917):** Nessus was unable to access the remote Windows Registry, likely due to the 'Remote Registry Access' service being disabled or lack of credentials.

**OS Security Patch Assessment Not Available (Nessus Plugin ID: 117886) & No Credentials Provided (Nessus Plugin ID: 110723):** The scan was uncredentialed, limiting the ability to perform local security patch assessments and comprehensive checks.

**Open Ports (Nessus Plugin ID: 11219):** TCP ports 135 (epmap), 139 (smb), and 445 (cifs) were found open.

**Traceroute Information (Nessus Plugin ID: 10287):** Traceroute information was obtained, showing a direct hop to the target.

**WMI Not Available (Nessus Plugin ID: 135860):** WMI queries could not be made, which limits information gathering about installed software or vulnerabilities.

The uncredentialed nature of the localhost scan is a critical limitation of the assessment. An uncredentialed scan provides only an external, surface-level view of the system. It inherently misses internal configurations, installed software, patch levels, and misconfigurations that would only be visible with authenticated access. This means the actual risk posture of the localhost could be significantly higher than reported, as many vulnerabilities (e.g., missing patches for installed applications, insecure software configurations) would not have been detected. This highlights the importance of conducting authenticated scans for comprehensive internal system assessments. It also implies that findings categorized as "Low" severity or "Informational" might be masking more critical underlying issues that an attacker with greater access could easily discover and exploit.

### 3.11. Database Engine Configuration and Transactional Integrity

#### Finding Title: Confirmed Use of InnoDB Engine for acuart Tables

The acuart database schema tables, including artists, carts, categ, featured, guestbook, pictures, users, and products, are confirmed to be utilizing the InnoDB storage engine, as indicated by their .ibd file extensions and entries in INNODB\_TABLESPACES\_BRIEF.csv and INNODB\_DATAFILES.csv.This is further confirmed by the ENGINE column in

TABLES.csv, which explicitly lists InnoDB for all acuart tables. This is a positive finding, as InnoDB supports ACID-compliant transactions, row-level locking, and foreign keys, which are crucial for maintaining data integrity and consistency in a multi-user, transactional web application environment. While the database system itself supports other engines like MyISAM, MEMORY, PERFORMANCE\_SCHEMA, CSV, and FEDERATED, the specific use of InnoDB for the acuart schema mitigates the direct risk of data inconsistency and race conditions that would arise from using non-transactional engines for these critical application tables.

Affected Database Components: All acuart tables (artists, carts, categ, featured, guestbook, pictures, users, products) are confirmed to use InnoDB.

Potential Impact: The use of InnoDB significantly reduces the risk of data corruption, inconsistency, and severe race conditions that would be present with non-transactional engines like MyISAM. This positively impacts data integrity and application reliability.

Severity Rating: Informational (Positive Finding), as the appropriate engine is in use.

The explicit confirmation that acuart tables use InnoDB is a crucial piece of information that clarifies a previous inference. InnoDB is the recommended engine for most transactional workloads due to its ACID compliance. This demonstrates a sound architectural choice for the core database tables, which is a strong foundation for data integrity. However, it is important to remember that even with InnoDB, proper application-level transaction management is still necessary to ensure complex operations are atomic. This finding shifts the focus from a potential database engine vulnerability to ensuring that the application fully utilizes the capabilities of the chosen engine.

### 3.12. Other Informational Findings

This section outlines observations that, while not directly exploitable vulnerabilities, provide valuable information for an attacker or indicate potential areas for security hardening.

#### Host Fully Qualified Domain Name (FQDN) Resolution (testphp.vulnweb.com)

Nessus successfully resolved the Fully Qualified Domain Name (FQDN) of the remote host, confirming that 44.228.249.3 resolves as ec2-44-228-249-3.us-west-2.compute.amazonaws.com.This finding is informational and simply confirms basic network resolution, providing additional context about the hosting environment.

#### TCP/IP Timestamps Supported (testphp.vulnweb.com)

The remote host implements TCP timestamps, as defined by RFC1323. This feature can sometimes be used to compute the uptime of the remote host. This is an informational finding and generally not a direct security concern.

#### Service Detection (HELP Request) (testphp.vulnweb.com)

Nessus was able to identify the web server service running on port 80 by analyzing its banner or its response to a 'HELP' request. This is an informational finding, representing a standard part of service enumeration during reconnaissance.

#### Common Platform Enumeration (CPE) (testphp.vulnweb.com)

Nessus identified Common Platform Enumeration (CPE) names for the detected software, including Linux Kernel, Nginx 1.19.0, and PHP 5.6.40. This is an informational finding. CPE provides a standardized naming scheme for discovered software and hardware, which is useful for vulnerability management and asset inventory.

#### Device Type (testphp.vulnweb.com)

Based on the remote operating system, Nessus attempted to determine the device type, classifying it as "unknown" with 56% confidence.This is an informational finding, representing a general classification attempt by the scanner.

#### Exposed User Privileges (USER\_PRIVILEGES.csv) (testphp.vulnweb.com)

The USER\_PRIVILEGES.csv file, while containing only headers (GRANTEE, IS\_GRANTABLE...) in the provided snippet, indicates a structured format for exposing user privilege information.The analysis confirms that the provided snippet is mostly headers, lacking specific data. The impact is assessed as

**Low (Potential Information Disclosure)**. Although the provided snippet is empty beyond headers, the existence of such a file suggests that user privilege data could be exposed. If populated, it would reveal which users have which permissions, aiding in privilege escalation attacks. The mere presence of a file designed to contain user privilege information indicates a potential configuration oversight or a capability that could lead to information exposure. In a live environment, this file might be populated under certain conditions, or its presence could indicate that similar privilege information is exposed through other means. It suggests that the system has a mechanism for generating or storing this data in an accessible format. During reconnaissance, identifying files that are designed to contain sensitive data, even if currently empty, is crucial for understanding potential information leakage vectors and for recommending proactive security measures.

#### Exposed User Attributes (USER\_ATTRIBUTES.csv) (testphp.vulnweb.com)

The USER\_ATTRIBUTES.csv file contains one entry: localhost,acuart,NULL.This indicates that the username acuart is exposed for the localhost host, although no specific attributes are present in the ATTRIBUTE column. The impact is **Low (Information Disclosure - Username)**. The exposure of a valid username is a minor information disclosure. While the ATTRIBUTE field being NULL limits the immediate impact, the exposure of a valid username provides a direct target for subsequent credential-based attacks. This exposed username can be directly utilized in brute-force attacks, password spraying campaigns, or credential stuffing attempts against login interfaces, such as the previously identified /admin/ directory. Even seemingly small pieces of information, like a single username, can be valuable for an attacker during the reconnaissance phase, accelerating their ability to move to the exploitation phase.

#### Exposed Database Routines (VIEW\_ROUTINE\_USAGE.csv) (testphp.vulnweb.com)

The VIEW\_ROUTINE\_USAGE.csv file contains headers (TABLE\_SCHEMA, TABLE\_NAME, SPECIFIC\_NAME...) but no specific routine usage data in the provided snippet.

The impact is **Low (Potential Information Disclosure)**. Similar to the USER\_PRIVILEGES.csv, the existence of this file implies a capability or configuration for exposing database routine usage. If populated, it would reveal details about stored procedures and functions, aiding in understanding application logic and identifying SQL injection points. The presence of the file with relevant headers, despite lacking data in the snippet, indicates a potential avenue for information disclosure. The system is configured to potentially log or expose information about how database routines (stored procedures, functions) are used. While the provided snippet does not contain actual routine usage data, its structure suggests that such data could be exposed. This flags a potential misconfiguration that, if active or previously active, could have leaked sensitive information about the application's backend logic. A comprehensive assessment would involve checking if this file is ever populated or if similar data is accessible through other means in a live environment. The presence of specific file types or structures, even if currently empty, can reveal potential information leakage vectors and guide further investigation into the system's configuration and data handling practices.

#### Exposed Database Structures (VIEW\_TABLE\_USAGE.csv) (testphp.vulnweb.com)

The VIEW\_TABLE\_USAGE.csv file provides extensive details about the database's internal structure and operational characteristics through various sys schema views.While often categorized as "informational" by some automated scanners for merely listing structures, the sheer breadth and depth of exposed database metadata in this context elevate it to a

**High-severity information disclosure**. This data significantly aids an attacker in crafting targeted SQL injection attacks, understanding data relationships, and identifying sensitive tables and columns.

Severity Rating: Informational (though it represents a high-severity information disclosure due to the extensive details exposed).

##### Exposed Database Structures

|  |  |  |  |
| --- | --- | --- | --- |
| **View Name** | **Schema** | **Source Table(s)** | **Type of Information Exposed** |
| innodb\_buffer\_stats\_by\_schema | sys | information\_schema.INNODB\_BUFFER\_PAGE | InnoDB buffer pool statistics by schema, revealing data access patterns. |
| x$innodb\_buffer\_stats\_by\_schema | sys | information\_schema.INNODB\_BUFFER\_PAGE | InnoDB buffer pool statistics by schema (extended). |
| innodb\_buffer\_stats\_by\_table | sys | information\_schema.INNODB\_BUFFER\_PAGE | InnoDB buffer pool statistics by table, indicating table usage frequency. |
| x$innodb\_buffer\_stats\_by\_table | sys | information\_schema.INNODB\_BUFFER\_PAGE | InnoDB buffer pool statistics by table (extended). |
| schema\_object\_overview | sys | information\_schema.EVENTS, ROUTINES, STATISTICS, TABLES, TRIGGERS | Comprehensive overview of schema objects, including scheduled events, stored procedures, functions, index metadata, table details, and triggers. |
| schema\_auto\_increment\_columns | sys | information\_schema.COLUMNS, TABLES | Details about auto-incrementing columns in tables. |
| x$schema\_flattened\_keys | sys | information\_schema.STATISTICS | Information about flattened keys (indexes). |
| latest\_file\_io | sys | information\_schema.PROCESSLIST | Information on latest file I/O operations and active database processes. |
| x$latest\_file\_io | sys | information\_schema.PROCESSLIST | Information on latest file I/O operations and active database processes (extended). |
| innodb\_lock\_waits | sys | information\_schema.INNODB\_TRX | Details on InnoDB lock waits, indicating transaction contention. |
| x$innodb\_lock\_waits | sys | information\_schema.INNODB\_TRX | Details on InnoDB lock waits (extended). |
| schema\_unused\_indexes | sys | information\_schema.STATISTICS | Identification of unused indexes within the database. |
| metrics | sys | information\_schema.INNODB\_METRICS | Various InnoDB performance and operational metrics. |

The direct exposure of these internal database views is a severe security oversight, essentially providing an attacker with a comprehensive blueprint of the database. This information is a goldmine for an attacker, as it eliminates the need for time-consuming and often noisy enumeration techniques (such as blind SQL injection) to map the database structure. An attacker can directly identify all table and column names, including those likely to contain sensitive data (e.g., users, passwords, credit\_cards). They can also discern the parameters and logic of stored procedures and functions, potentially revealing SQL injection points or business logic flaws. Furthermore, the structure and usage of indexes can inform more efficient data exfiltration queries or identify unindexed sensitive data. Even real-time performance and contention data could be used to craft denial-of-service attacks or identify high-value targets within the database. This represents a critical information leakage that significantly reduces the reconnaissance phase for an attacker, making subsequent exploitation (especially SQL injection) much easier, faster, and more effective. It points to a severe misconfiguration of database permissions, where internal system views are exposed to unauthorized access.

## 4. Overall Risk Assessment

### Aggregated Risk Profile

Based on the confirmed vulnerabilities in the web application's database design, the active exploitation attempts observed in the process list, the over-privileged database user, and the direct findings on the localhost environment, the system presents a **Critical overall risk** to the web application and its data. The confluence of fundamental security design flaws (such as insecure credential storage), the pervasive and actively exploited potential for widespread injection attacks, and predictable object identifiers, coupled with an over-privileged database user and network-level vulnerabilities like unforced SMB signing, collectively creates a highly exploitable and unstable environment. These issues indicate a significant attack surface and a low barrier to entry for malicious actors, with evidence that some attack vectors are already being probed.

### Key Risk Factors

* **Data Sensitivity:** The database stores highly sensitive Personally Identifiable Information (PII) and potentially unencrypted payment information, making it a high-value target for attackers seeking to exfiltrate or exploit sensitive data.
* **Exploitability:** Many of the identified vulnerabilities, particularly default passwords, common injection points, and unforced SMB signing, are easily discoverable and exploitable by attackers, even those with limited technical sophistication. The simplicity of exploiting these flaws, combined with observed active attempts, increases the likelihood of a successful attack.
* **Impact Severity:** Successful exploitation of these vulnerabilities could lead to catastrophic consequences, including full system compromise, mass data breaches, financial fraud, severe reputational damage, and significant legal and regulatory penalties. The scope of potential damage is extensive, affecting both the organization and its users. The confirmed excessive privileges of the database user amplify this impact significantly.
* **Systemic Issues:** The nature and breadth of the findings suggest fundamental weaknesses in the application's secure development practices, its security architecture, and potentially the underlying infrastructure configuration. This indicates a need for a holistic security overhaul rather than isolated, reactive fixes.

### Alignment with Industry Standards

The identified vulnerabilities align directly and prominently with critical categories in the OWASP Top 10 2021, the most widely recognized standard for web application security risks. Specifically, the findings demonstrate:

* **A01:2021 Broken Access Control:** Evident in the potential for Insecure Direct Object Reference (IDOR) due to predictable IDs, and critically, the over-privileged database user.
* **A03:2021 Injection:** Highlighted by the numerous text fields susceptible to SQL Injection and Cross-Site Scripting, with active exploitation attempts observed.
* **A07:2021 Identification and Authentication Failures:** Critically demonstrated by the insecure storage of passwords and default credentials.
* **A04:2021 Insecure Design:** Reflected in the choice of data types for sensitive information, the overall application architecture, and the design of callable routines.

## 5. Recommendations and Remediation Guidance

To significantly improve the security posture of both testphp.vulnweb.com and the acuart database backend, and to mitigate the identified risks, the following prioritized remediation strategies are strongly recommended.

### Prioritization of Remediation Efforts

#### Critical (Immediate Action Required)

1. **Upgrade PHP Version (testphp.vulnweb.com):**

Immediately upgrade PHP to a currently supported and patched version (e.g., PHP 8.1.x, 8.2.x, or 8.3.x).This is the single most critical step, as it addresses a fundamental security weakness and will inherently mitigate numerous other potential vulnerabilities stemming from the outdated software. This upgrade will close many known exploit vectors that are currently active due to the EOL status of PHP 5.6.

1. **Address Insecure Password and Credit Card Storage (192.168.43.170):**

Implement strong, one-way cryptographic hashing algorithms (e.g., Argon2, bcrypt, scrypt) with appropriate salts for all user passwords. It is imperative that passwords are never stored in plain-text or in a format that can be easily reversed.

Credit card numbers (cc) must *never* be stored directly in the database. Instead, leverage a PCI-DSS compliant payment gateway that handles tokenization, where only a non-sensitive token is stored within the database. Alternatively, if direct storage is unavoidable, ensure sensitive payment data is encrypted using strong, industry-standard encryption algorithms (e.g., AES-256) with robust key management practices.

1. **Change Default Credentials (192.168.43.170):**

Immediately change all default credentials (e.g., users.uname, users.pass) upon initial deployment. Furthermore, the application logic must be designed to prevent the use of any default or easily guessable credentials in a production environment, perhaps by forcing password changes on first login or rejecting common weak passwords.

1. **Implement Parameterized Queries and Input Validation (192.168.43.170):**

Implement parameterized queries (also known as prepared statements) for all database interactions. This is recognized as the most effective defense against SQL Injection, as it fundamentally separates executable code from user-supplied data, and must be applied universally.

Perform rigorous and strict input validation on all user-supplied data before it is processed or stored. This includes enforcing data types, defining maximum lengths, and ensuring adherence to expected formats (e.g., allowing only alphanumeric characters for names, validating URLs for image paths). Any input that does not conform to the expected format should be rejected.

1. **Reduce Database User Privileges (192.168.43.170):**

Immediately revoke all unnecessary privileges from the acuart database user. The user should only have the absolute minimum privileges required for the web application's normal operation (typically SELECT, INSERT, UPDATE, DELETE on specific tables). Specifically, remove  
CREATE, DROP, ALTER, INDEX, and EXECUTE privileges on the acuart schema.

#### High (Urgent Action Required)

1. **Restrict Access to Sensitive Directories (testphp.vulnweb.com):**

Implement robust authentication and authorization mechanisms for the /admin/ directory. Consider additional layers of security such as IP-based restrictions, multi-factor authentication, or requiring access via a Virtual Private Network (VPN) for all administrative interfaces.

Remove or severely restrict public web access to all /CVS/ directories and their contents. Version control system files should never be exposed on a production web server. If necessary for deployment, ensure these files are located outside the web root or are protected by stringent access controls.

Implement strict access controls (e.g., requiring authentication, disabling directory indexing) for the /secured/ directory to prevent unauthorized access to content that its name implies is sensitive.

Review and restrict access to the /vendor/ directory. While third-party libraries are often necessary, ensure that directory listing is disabled and that no sensitive configuration files within these libraries are publicly accessible.

1. **Secure Cross-Domain Policies (testphp.vulnweb.com):**

Review and remove or restrict wildcard entries (\*) in clientaccesspolicy.xml and crossdomain.xml. Adhere strictly to the principle of least privilege, allowing cross-domain access only to explicitly trusted and absolutely necessary domains. This will prevent malicious external sites from interacting with the application on behalf of users.

1. **Implement Output Encoding (192.168.43.170):**

Apply context-aware output encoding to all user-generated content before it is rendered in the web browser to mitigate Cross-Site Scripting (XSS) vulnerabilities.

#### Medium (High Priority Action)

1. **Implement Robust Authorization Checks (192.168.43.170):**

Implement comprehensive and strict server-side authorization checks for every request that accesses data identified by an ID (e.g., artist\_id, cat\_id, pic\_id, products.id). This is essential to prevent Insecure Direct Object Reference (IDOR) vulnerabilities.

Consider using unpredictable identifiers (e.g., UUIDs or hashed IDs, random tokens) for sensitive resources instead of sequential integers for user-facing IDs. While this does not replace proper authorization, it makes enumeration and guessing of IDs significantly more difficult.

1. **Secure Stored Routines (192.168.43.170):**

Review all stored routines, especially those that execute dynamic SQL (like sys.execute\_prepared\_stmt and sys.create\_synonym\_db). If the  
acuart user can call these, ensure that any user-supplied input passed to them is rigorously validated and sanitized. Ideally, restrict the EXECUTE privilege on such routines from application users.

For any custom stored routines developed for the application, consider using SECURITY DEFINER where appropriate, ensuring the DEFINER user has only the minimum necessary privileges. This allows the routine to perform actions with elevated privileges without granting those privileges directly to the calling user, but requires careful auditing of the routine's code.

Regularly audit the usage of stored routines, especially those that execute dynamic SQL, to detect any unusual or unauthorized calls.

1. **Implement Missing HTTP Security Headers (testphp.vulnweb.com):**

Add the X-Frame-Options header to all web responses with a value such as DENY or SAMEORIGIN to prevent clickjacking attacks.

Add the X-Content-Type-Options header with a value of nosniff to all web responses. This instructs browsers to disable MIME type sniffing, preventing potential XSS vulnerabilities where a browser might misinterpret file types.

1. **Enforce SMB Message Signing (192.168.43.170):**

On the localhost, enforce message signing in the SMB configuration. On Windows, configure the policy setting 'Microsoft network server: Digitally sign communications (always)'. For Samba, the setting is 'server signing'.

1. **Disable SMBv1 (192.168.43.170):**

On the localhost, disable SMBv1, as it is an outdated and less secure protocol.

#### Low (Moderate Priority Action)

1. **Filter ICMP Timestamp Requests (testphp.vulnweb.com, 192.168.43.170):**

Configure network firewalls or host-based firewalls to filter out ICMP timestamp requests (type 13) and block outgoing ICMP timestamp replies (type 14). While a low risk, this removes a minor piece of reconnaissance information available to attackers.

#### Ongoing (Continuous Improvement)

Beyond the specific vulnerabilities identified, the following general security best practices are recommended for continuous improvement of the system's security posture:

* **Conduct Regular Security Audits and Penetration Tests:** Establish and rigorously adhere to a schedule for periodic penetration tests, vulnerability assessments, and code reviews to proactively identify and address new or emerging security weaknesses.
* **Provide Continuous Developer Training:** Implement mandatory and recurring secure coding training for all development team members. This training should place a strong emphasis on understanding and mitigating common vulnerabilities, particularly those outlined in the OWASP Top 10.
* **Implement Database Activity Monitoring:** Deploy tools to continuously monitor database logs and active processes for suspicious queries and activities indicative of SQL injection attempts or other unauthorized access, enabling rapid incident response.

### General Secure Development Best Practices

To prevent the recurrence of similar vulnerabilities and establish a robust security foundation, the following best practices should be integrated into the software development lifecycle (SDLC):

* **Secure Coding Training:** Implement mandatory and recurring secure coding training for all development team members, with a strong emphasis on understanding and mitigating common vulnerabilities, particularly those outlined in the OWASP Top 10.
* **Input Validation & Output Encoding Framework:** Develop and implement a centralized, robust framework for comprehensive input validation and context-aware output encoding across the entire application. This ensures consistency in handling user input and reduces the likelihood of injection flaws.
* **Principle of Least Privilege:** Apply this principle rigorously to all database users, application components, and system accounts. Permissions should be granted on a need-to-know and need-to-do basis, minimizing the potential blast radius of a compromise.
* **Secure Configuration Management:** Ensure all default configurations for the web server, database server, and application framework are hardened. Sensitive information, such as database credentials, should never be hardcoded or stored in plain text; instead, utilize secure secrets management solutions (e.g., environment variables, a dedicated secrets vault).
* **Regular Security Testing:** Establish a continuous security testing program that includes Static Application Security Testing (SAST) during development, Dynamic Application Security Testing (DAST) in testing environments, and periodic manual penetration tests by independent security experts.
* **Data Encryption:** Implement encryption for all sensitive data, both in transit (e.g., using HTTPS for web traffic, SSL/TLS for database connections) and at rest (for highly sensitive data stored in the database).
* **Secure Error Handling:** Implement secure error handling mechanisms that provide generic error messages to end-users to prevent information leakage that could aid attackers. Detailed error information should be logged securely on the server-side for debugging and analysis.
* **Comprehensive Logging and Monitoring:** Implement robust logging for security-relevant events (e.g., failed login attempts, access to sensitive resources, administrative actions). Establish continuous monitoring to detect and respond to suspicious activities in real-time, enabling rapid incident response.

## 6. Appendices

### A. References to Security Standards

This section provides a concise list of the industry security standards referenced throughout this report, offering a basis for further research and compliance efforts:

* OWASP Top 10 (2021)
* NIST Special Publication 800-53
* SANS Top 25 Most Dangerous Software Errors
* PCI Data Security Standard (PCI-DSS)

### B. Glossary of Technical Terms

This section provides definitions for technical terms used in the report to enhance understanding:

* **SQL Injection (SQLi):** A code injection technique used to attack data-driven applications, in which malicious SQL statements are inserted into an entry field for execution (e.g., to dump database contents to the attacker).
* **Cross-Site Scripting (XSS):** A type of security vulnerability typically found in web applications. XSS enables attackers to inject client-side scripts into web pages viewed by other users.
* **Insecure Direct Object Reference (IDOR):** A type of access control vulnerability where an application exposes a direct reference to an internal implementation object, allowing attackers to manipulate these references to access unauthorized data.
* **MyISAM:** A storage engine for MySQL that does not support transactions, row-level locking, or foreign keys. It is known for its speed in read-heavy operations but lacks data integrity guarantees.
* **InnoDB:** A storage engine for MySQL that supports transactions with ACID properties (Atomicity, Consistency, Isolation, Durability), row-level locking, and foreign keys, making it suitable for high-concurrency and data-integrity-critical applications.
* **Personally Identifiable Information (PII):** Information that can be used to distinguish or trace an individual’s identity, either alone or when combined with other personal or identifying information.
* **CVSS (Common Vulnerability Scoring System):** An open industry standard for assessing the severity of computer system security vulnerabilities.
* **Parameterized Queries (Prepared Statements):** A method of preparing a SQL query where placeholders are used for user-supplied data, preventing SQL Injection by ensuring data is treated as data, not executable code.
* **Output Encoding:** The process of converting special characters in data into a safe format (e.g., HTML entities) before rendering it in a web page, preventing the browser from interpreting the data as executable code.
* **SMB Signing:** A security feature in the Server Message Block (SMB) protocol that digitally signs SMB communications to prevent man-in-the-middle attacks and ensure data integrity.
* **Stored Routine:** A stored program in a database, either a stored procedure or a function, that can be called to perform specific operations.
* **DEFINER:** The MySQL user account that created or last altered a stored routine.
* **SECURITY TYPE (INVOKER/DEFINER):** Determines the security context in which a stored routine executes. INVOKER means it runs with the privileges of the user calling it; DEFINER means it runs with the privileges of the user who defined it.

### C. Detailed acuart Database Schema Information

This appendix provides a full listing of the acuart database schema as derived from the provided COLUMNS.csv and COLUMNS\_EXTENSIONS.csv. This serves as a complete reference for the database structure analyzed in this report, ensuring transparency and providing technical teams with the foundational data for their review and remediation planning.

### D. Raw Scan Outputs

This section contains the raw output logs from the scanning tools used during the penetration test. In a full report, this would also include screenshots of tool usage and access gained during exploitation steps.

* testphp(nikto).txt
* testphp.vulnweb(dirb).txt
* testphp.vulnweb(maltego).pdf
* testphp\_vulnweb\_3xtrlb.pdf
* localhost\_fnt6gc.pdf

### E. Resource Links

* <https://github.com/swarnavdas15/CEHv1MajorProject.github.io/tree/main/Findings/Sqlmapresult>
* <https://github.com/swarnavdas15/CEHv13MajorProject.github.io/tree/main/Findings/dirbresult>
* <https://github.com/swarnavdas15/CEHv13MajorProject.github.io/tree/main/Findings/maltego>
* <https://github.com/swarnavdas15/CEHv13MajorProject.github.io/tree/main/Findings/nessusresult>
* <https://github.com/swarnavdas15/CEHv13MajorProject.github.io/tree/main/Findings/niktoresult>
* <https://github.com/swarnavdas15/CEHv13MajorProject.github.io/tree/main/Findings/nmapResult>