Web Application Security Assessment Report

Table of Contents

Description of the vulnerability

1. SQL Injection (SQLi)

Description:

SQL Injection is a critical vulnerability that allows an attacker to interfere with the queries that an application makes to its database. It occurs when user input is improperly sanitized and embedded directly into SQL statements. By injecting malicious SQL payloads, attackers can manipulate the query logic, access unauthorized data, perform administrative operations on the database, and, in severe cases, gain complete control over the application's backend.

At http://testphp.vulnweb.com/listproducts.php?cat=1, the cat parameter is directly used in an SQL query without proper input validation or parameterization. By injecting payloads such as ' OR '1'='1 --, the attacker can bypass the intended logic and retrieve all product records.

RiskLevel:Critical
CWE-ID: CWE-89 – Improper Neutralization of Special Elements used in an SQL Command('SQLInjection')
OWASP Category: A1:2021 – Broken Access Control / Injection

The URL in which the vulnerability has been found

.SQL Injection (SQLi)

- Vulnerability: SQL Injection
- URL Affected:

bash

CopyEdit

http://testphp.vulnweb.com/listproducts.php?cat=1

- Vulnerable Parameter: cat

The parameters which are vulnerable (if any)

Here are the **vulnerable parameters** identified during the assessment of http://testphp.vulnweb.com/, along with the associated vulnerabilities:

🔍 1. SQL Injection (SQLi)

- **Affected URL:**

bash

CopyEdit

http://testphp.vulnweb.com/listproducts.php?cat=1

- **Vulnerable Parameter:**

bash

CopyEdit

cat

- **Explanation:**
  This parameter is directly used in backend SQL queries without sanitization, allowing injection of SQL payloads such as ' OR '1'='1 --.

Payload used to trigger the vulnerability

SQL Injection (SQLi)

- **Affected URL:**

bash

CopyEdit

http://testphp.vulnweb.com/listproducts.php?cat=1

- **Vulnerable Parameter:**

bash

CopyEdit

cat

- **Explanation:**
  This parameter is directly used in backend SQL queries without sanitization, allowing injection of SQL payloads such as ' OR '1'='1 --.

Proof of Concept (PoC): SQL Injection

TargetURL:http://testphp.vulnweb.com/listproducts.php?cat=1
Vulnerability: SQL Injection via cat parameter

Objective:

Demonstrate exploitation of an SQL injection vulnerability in the cat parameter to bypass logic and retrieve unauthorized data.

Step-by-Step Instructions:

Step 1: Access the Original Endpoint

- Open your browser or Burp Suite.
- Visit:

bash

CopyEdit

http://testphp.vulnweb.com/listproducts.php?cat=1

- Observe: It displays products belonging to category 1.

Step 2: Inject Malicious Payload

- Modify the cat parameter with an SQLi payload:

bash

CopyEdit

http://testphp.vulnweb.com/listproducts.php?cat=' OR '1'='1' -- -

- Press Enter or forward the request via Burp.

Step 3: Analyze the Result

- All products from all categories are displayed.
- Expected behavior was to show products from only cat=1.
- This confirms the SQL logic was bypassed and the query is vulnerable.

Step 4: Automate with sqlmap (optional)

If allowed, use Kali Linux to test deeper extraction:

bash

CopyEdit

```
sqlmap -u "http://testphp.vulnweb.com/listproducts.php?cat=1" --risk=3 --level=5 --dump
```

- This command checks and extracts database data using SQLi exploitation.

Observation:

- The application does not properly sanitize user input.
- The payload ' OR '1'='1' -- - terminates the original SQL query and appends always-true logic.

Result:

- **Vulnerability Confirmed**
- Injection leads to **unauthorized data disclosure** and potentially **full DB compromise** depending on backend configuration.

Business Impact of the Vulnerability

SQL Injection (SQLi) vulnerabilities pose one of the most critical risks to web applications and can lead to devastating consequences for an organization. By exploiting SQLi, an attacker can bypass authentication mechanisms, access or manipulate sensitive information such as usernames, passwords, financial records, and even customer data. In severe cases, SQLi can enable attackers to perform administrative operations on the database server, delete critical data, or pivot to internal systems.

The business ramifications include data breaches that violate data privacy laws such as GDPR, HIPAA, or CCPA, leading to heavy fines and regulatory scrutiny. Furthermore, such attacks can severely damage customer trust, erode the organization’s brand reputation, and result in the loss of business partnerships. Competitors or malicious actors could use stolen data for competitive intelligence, ransomware operations, or social engineering attacks against clients.

Additionally, if database integrity is compromised, the organization could face operational disruptions, data loss, and increased costs for incident response, forensic investigation, and remediation. Given the high profile of SQLi in OWASP’s Top 10 vulnerabilities, the presence of such a flaw signals significant weaknesses in the organization's secure coding practices and application security posture.

Remediations / Countermeasures / Recommendations

To effectively mitigate SQL Injection and related vulnerabilities, organizations must adopt a multi-layered defense strategy. First and foremost, use parameterized queries (also known as prepared statements). These ensure that user-supplied data is treated strictly as input, not as part of the executable SQL command. Modern frameworks and database libraries support these constructs and should be used consistently throughout the codebase.

Input validation should also be enforced rigorously. Validate all user inputs against an allow-list of acceptable characters or formats (e.g., numeric only for ID fields). Avoid relying solely on blacklists or client-side validation.

Additionally, implement a Web Application Firewall (WAF) to filter and block suspicious traffic patterns, especially when patching is delayed. Configure detailed logging and monitoring to detect and alert on anomalies related to query behavior.

On the infrastructure side, enforce least privilege database access controls, ensuring web applications connect with restricted accounts. Disable verbose error messages to prevent leakage of database structure or stack traces.

Regular code reviews, automated static analysis, and penetration testing should be conducted as part of the SDLC. Follow secure coding guidelines such as those from OWASP, and integrate security into CI/CD pipelines for continuous protection.

Reputed References for the Vulnerabilities

1. **OWASPSQLinjectionCheatSheet**
   https://owasp.org/www-community/attacks/SQL_Injection
   A comprehensive overview of SQL Injection types, examples, and defense strategies.
2. **OWASPTopTen2021-Injection**
   https://owasp.org/Top10/A03_2021-Injection/
   Describes injection attacks as one of the top application security risks.
3. **NIST SP 800-115 – Technical Guide to Information Security Testing**
   https://csrc.nist.gov/publications/detail/sp/800-115/final
   Defines methodology for security testing, including SQL injection testing procedures.
4. **CWE-89: Improper Neutralization of Special Elements in SQL Commands**
   https://cwe.mitre.org/data/definitions/89.html
   Official MITRE entry on SQLi vulnerability, classification, and examples.
5. **PortSwiggerWebSecurity-Academy**
   https://portswigger.net/web-security/sql-injection
   Hands-on labs and educational material for SQLi, XSS, and other web security flaws.