Bug Bounty Report — Time-based Blind SQL Injection

Test Date (UTC): 2025-10-16

Target: http://testphp.vulnweb.com
Reported by: Masud Rana

Executive Summary
During authorized testing of the target application, a high severity time-based blind SQL injection was identified in the `artist` parameter of `artists.php`. The vulnerability allows an attacker to cause the database to execute arbitrary SQL logic (demonstrated via the MySQL *SLEEP()* function). This can be used to exfiltrate data, enumerate schema, and in some setups, modify or destroy data.

Vulnerability Details

Vulnerability Type: Time-based Blind SQL Injection (CWE-89)
Severity: High (CVSSv3.1 estimate: 8.6)
Affected Endpoint: /artists.php?artist=<value>
Parameter: artist

The application reflects database errors and executes injected SQL fragments from the `artist` parameter. Error-based testing (single quote) returned a MySQL syntax error. Time-based testing using a payload that invoked SLEEP(5) produced a consistent ~5 second increase in response time versus baseline, confirming execution.

Proof of Concept (PoC) — Non-destructive (authorized testing only)

Baseline timing (example):

```
curl -s -w 'Total time: %{time_total}s\n' -o /dev/null
"http://testphp.vulnweb.com/artists.php?artist=1"
Average output observed: ~0.6s
```

Time-based payload (URL-encoded):

```
curl -s -w 'Total time: %{time_total}s\n' -o /dev/null "http://testphp.vulnweb.com/artists.php
?artist=1%27%20AND%20IF(1%3D1%2C%20SLEEP(5)%2C%200)%20--%20"
Average output observed: ~5.6s (approx. baseline + 5s)
```

Note: The above PoC is for authorized testing only. Do not execute against systems you do not own or have permission to test.

Impact

Successful exploitation allows an attacker to perform blind or error-based extraction of database contents, potentially exposing PII, credentials, or other sensitive data. An attacker could also alter or delete data, and combine the issue with other vulnerabilities to escalate access.

Root Cause Analysis

The application appears to construct SQL queries by concatenating user-supplied input (the artist parameter) directly into SQL statements without using parameterized queries or sufficient input validation. Additionally, the application discloses detailed SQL error messages to end users which increases the information available to attackers.

Remediation & Mitigation

1. Use Parameterized Queries / Prepared Statements
Replace any string concatenation with prepared statements (PDO or mysqli). Bind user inputs as parameters.
2. Input Validation
Validate and coerce types server-side (e.g., require integer IDs via FILTER_VALIDATE_INT or casting).
3. Suppress Detailed DB Errors
Disable display of database errors to end users. Log details internally.

4. Least Privilege
Ensure the DB user used by the web application only has necessary privileges (avoid DROP/CREATE if not needed).
5. WAF / Rate Limiting
Add WAF rules to detect and block time-based SQLi patterns and implement request throttling.
6. Additional Security Controls
Add Content-Security-Policy, set cookie flags (HttpOnly, Secure, SameSite), and sanitize output to prevent XSS.

After applying the fixes above, perform an internal retest. Consider a full code audit for other parameters and endpoints.

Evidence & Timeline

Event	Details
Initial discovery	Single-quote in `artist` parameter returned MySQL syntax error (error-based indicator).
Timing test	Baseline ~0.6s; payload with SLEEP(5) produced ~5.6s (test date 2025-10-16).
Report created	2025-10-16 (UTC)

Appendix — Helpful Commands & Checks

Quick grep checks:
```
grep -R --line-number -E "\$_(GET|POST|REQUEST)\['?[A-Za-z0-9_]+\'?\]" .
grep -R --line-number -E "(\.\s*\$_(GET|POST)|\".*\$_(GET|POST)|'\s*\.\s*\$_(GET|POST))" .
```
Disable display_errors in production: `ini_set('display_errors', 0);`

End of report
Prepared by: Masud Rana