

Introduction

This report presents a vulnerability assessment which had been done on a publicly accessible website <http://testphp.vulnweb.com/>. The purpose of this assessment was to identify common web application vulnerabilities using ethical testing techniques. It was performed using manual browser based analysis and passive scanning tools such as OWASP ZAP.

Task 1

Checking JavaScript errors or warnings

The screenshot shows a web browser window with two tabs: 'Pulse - Puppy67/Future-interns' and 'Home of Acunetix Art'. The active tab is 'Home of Acunetix Art' at testphp.vulnweb.com. The page content includes a header with the Acunetix logo and a navigation menu with links like 'home', 'categories', 'artists', 'disclaimer', 'your cart', 'guestbook', and 'AJAX Demo'. A sidebar on the left contains links for 'search art', 'Browse categories', 'Browse artists', 'Your cart', 'Signup', 'Your profile', 'Our guestbook', and 'AJAX Demo'. Another sidebar lists 'Links' including 'Security art', 'PHP scanner', 'PHP vuln help', and 'Fractal Explorer'. At the bottom, there's a 'Warning' box stating that the site is intentionally vulnerable for testing purposes. On the right side of the browser, the developer tools are open, specifically the 'Console' tab, which is currently selected. The console output shows a single message: '> (ctrl) [i] to turn on code suggestions. Don't show again'. The rest of the console is empty.

The browser developer console had been reviewed during page load to identify any errors or warnings, no errors or warnings were observed which shows stable client side execution.

Checking network requests and security headers

The screenshot shows the Acunetix Web Vulnerability Scanner interface. The main window displays a list of network requests for the URL `testphp.vulnweb.com`. The 'Headers' tab is selected for the first request, which is for the root domain. The response headers shown are:

| Name | Value |
|-------------------|---|
| Request URL | http://testphp.vulnweb.com/ |
| Request Method | GET |
| Status Code | 200 OK |
| Remote Address | 44.228.249.3:80 |
| Referrer Policy | strict-origin-when-cross-origin |
| Connection | keep-alive |
| Content-Encoding | gzip |
| Content-Type | text/html; charset=UTF-8 |
| Date | Mon, 19 Jan 2026 22:34:19 GMT |
| Server | nginx/1.19.0 |
| Transfer-Encoding | chunked |

Below the header table, there is a warning message: "Warning: This is not intended to help let someone break Look for potential S".

This site lacks multiple security headers such as content security policy, x frame options, x content type options and strict transport security. This exposes the website to multiple attacks such as cross site scripting and man in the middle attacks.

The request method used on this site is: Request Method: GET this means that data is passed via URL and this makes it easier to test for SQLi.

Automated Passive Vulnerability Assessment (OWASP ZAP)

The screenshot shows the OWASP ZAP interface in 'ATTACK Mode'. The main window displays an 'Automated Scan' configuration panel with the URL `http://testphp.vulnweb.com` entered. The 'Attack' button is highlighted. The progress bar indicates 'Actively scanning (attacking) the URLs discovered by the spider(s)'. On the left, a sidebar lists various alerts found during the scan, including Cross Site Scripting (Reflected), Absence of Anti-CSRF Tokens (Systemic), Content Security Policy (CSP) Header Not Set (Systemic), Missing Anti-clickjacking Header (Systemic), In Page Banner Information Leak, Server Leaks Information via "X-Powered-By" HTTP Response Header, Server Leaks Version Information via "Server" HTTP Response Header, X-Content-Type-Options Header Missing (Systemic), Authentication Request Identified, Charset Mismatch (Header Versus Meta Content-Type Charset), Modern Web Application (Systemic), and User Controllable HTML Element Attribute (Potential XSS). The bottom status bar shows the current proxy is running on localhost:8080.

This screenshot shows a detailed view of a 'Cross Site Scripting (Reflected)' alert. The alert information includes:

- URL: `http://testphp.vulnweb.com/guestbook.php`
- Risk: High
- Confidence: Medium
- Parameter: name
- Attack: `<script>alert(1)</scRipt>`
- Evidence: `<script>alert(1)</scRipt>`
- CWE ID: 79
- WASC ID: 8
- Source: Active (40012 - Cross Site Scripting (Reflected))
- Input Vector: Form Query

The description notes that Cross-site Scripting (XSS) is an attack technique involving echoing attacker-supplied code into a user's browser instance. It provides a link to the OWASP XSS reference page (<https://owasp.org/www-community/attacks/xss/>). The bottom status bar shows the current proxy is running on localhost:8080.

The screenshot shows the ZAP (Zed Attack Proxy) interface. The top menu bar includes File, Edit, View, Analyse, Report, Tools, Import, Export, Online, Help, and Untitled Session - 202601.. The toolbar contains icons for ATTACK Mode, Sites, Contexts, Requests, Responses, Requester, and other analysis tools.

In the left sidebar, under 'Contexts', there is a 'Default Context' and a 'Sites' section. The main workspace displays a POST request to `http://testphp.vulnweb.com/guestbook.php` with the following headers:

```
POST http://testphp.vulnweb.com/guestbook.php HTTP/1.1
host: testphp.vulnweb.com
user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/141.0.0.0 Safari/537.36
pragma: no-cache
cache-control: no-cache
```

The request body contains the payload: `name=ZAP&text=&submit=add+message%27+AND+%271%27%3D%271%27+--`.

The navigation bar at the bottom includes History, Search, Alerts (selected), Output, Active Scan, Spider, AJAX Spider, and a plus sign icon.

The 'Alerts' panel on the left lists various findings, with 'SQL Injection' selected. Other alerts include Cross Site Scripting (Reflected), SQL Injection - MySQL, Absence of Anti-CSRF Tokens, Content Security Policy Header Not Set, HTTP Only Site, Missing Anti-clickjacking Header, XSLT Injection, In Page Banner Information Leak, Server Leaks Information via "X-Powered-By" HTTP Response Header, Server Leaks Version Information via "Server" HTTP Response Header, X-Content-Type-Options Header Missing, Authentication Request Identified, Charset Mismatch, GET for POST, Modern Web Application, User Agent Fuzzer, and User Controllable HTML Element Attribute (Potential XSS).

The right panel provides detailed information about the selected 'SQL Injection' alert:

- SQL Injection**
- URL:** `http://testphp.vulnweb.com/guestbook.php`
- Risk:** High
- Confidence:** Medium
- Parameter:** submit
- Attack:** add message' OR '1'='1' --
- Evidence:**
- CWE ID:** 89
- WASC ID:** 19
- Source:** Active (40018 - SQL Injection)
- Input Vector:** Form Query
- Description:** SQL injection may be possible.
- Other Info:** The page results were successfully manipulated using the boolean conditions [add message' AND '1'='1' --] and [add message' OR '1'='1' --].
- Solution:** Do not trust client side input, even if there is client side validation in place. In general, type check all data on the server side.
- Reference:** https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection.html

At the bottom, the status bar shows 'Alerts 4', 'Flags 5', 'Pages 4', 'Main Proxy: localhost:8080', 'Current Status: 0/0', and various network activity indicators.

Recording Vulnerability Assessment

Cross site scripting (Risk Level: High) - attack technique that involves echoing attacker-supplied code into a user's browser instance.

SQL Injection (Risk Level: High) – cyber attack where malicious SQL code is inserted into database queries.

Absence of anti CSRF token (Risk Level: Medium) – No anti CSRF tokens were found in HTML submission form.

Content security policy (CSP) Header not set (Risk Level: Medium) – This is an added layer of security that helps to detect and mitigate certain types of attacks such as cross site scripting and data injection.

HTTP only site (Risk Level: Medium) – The site is only served under HTTP and not HTTPS.

XSLT Injection (Risk Level: Medium) - Injection using XSL transformations may be possible and may allow an attacker to read system information.

In page banner information leak (Risk Level: Medium) - The server returned a version banner string in the response content. Such information leaks may allow attackers to further target specific issues impacting the product and version in use.

Remediation Recommendations

Cross site scripting – encode output before rendering in HTML, validate all user input on server side, implement a strong content security policy.

Eg : use content security policy : Content-Security-Policy: default-src 'self';

Security Benefits: Prevents malicious scripts from executing in users browsers.

SQL Injection – use parameterized queries, apply strict server side input validations, use least privilege database accounts.

Eg: SELECT * FROM users WHERE id = ?

Security Benefits: Prevents attackers from manipulating database queries.

Absence of anti CSRF token – Implement unique CSRF tokens in all state changing forms, validate tokens on the server.

Eg: Add hidden CSRF token field to forms.

Security Benefits: Prevents cross site request forgery attacks.

Content security policy(CSP) header not set – Define a strict CSP header, limit scripts, styles and media to trusted domains.

Eg: Content-Security-Policy: default-src 'self';

Security Benefits: reduces impact of data injection attacks.

HTTP only site – implement HTTPS using valid TLS certificate, redirect all HTTP traffic to HTTPS.

Eg: Strict-Transport-Security: max-age=31536000; includeSubDomains

Security Benefits: Protects data confidentiality and integrity.

XSLT Injection – Disable external entity resolution in XSLT processors, apply strict input validation.

Security Benefits: Prevents abuse of XML processing engines

In page banner information leak – Disable detailed error messages, configure server to hide version information.

Eg:

Nginx: server_tokens off;

PHP: expose_php = Off

Security Benefits: Reduces information available to attackers.

Conclusion

The vulnerability assessment identified multiple security weaknesses including high risk issues such as cross site scripting and SQL injection. These vulnerabilities could potentially be exploited if not resolved. Implementing the recommended remediation measures would improve the security of the application.