

# 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 - Duppy67/Future-interns' and 'Home of Acunetix Art'. The address bar shows 'testphp.vulnweb.com' with a 'Not secure' warning. The website content includes the 'acunetix art' logo, a search bar, and a list of links: 'home', 'categories', 'artists', 'disclaimer', 'your cart', 'guestbook', and 'AJAX Demo'. The main content area says 'welcome to our page' and 'Test site for Acunetix WVS.' The footer contains a warning: 'Warning: This is not a real shop. This is an example PHP application, which is intentionally vulnerable to web attacks. It is intended to help you test Acunetix. It also helps you understand how developer errors and bad configuration may let someone break into your website. You can use it to test other tools and your manual hacking skills as well. Tip: Look for potential SQL Injections, Cross-site Scripting (XSS), and Cross-site Request Forgery (CSRF), and more.'

The browser's developer console is open, showing a message: '<ctrl> to turn on code suggestions. Don't show again'. The console tabs include 'Elements', 'Console', 'Sources', 'Network', and 'Performance'. The 'Console' tab is active, showing the message. The bottom of the browser window shows the 'Console' tab selected, with 'What's new', 'AI assistance', and 'Issues' options.

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 a web browser window with the address bar displaying "testphp.vulnweb.com". The page content includes a navigation menu with links like "home", "categories", "search art", "Browse categories", "Browse artists", "Your cart", "Signup", "Your profile", "Our guestbook", "AJAX Demo", "Links", "Security art", "PHP scanner", "PHP vuln help", and "Fractal Explorer". A warning message is visible at the bottom: "Warning: This is not... It is intended to help... let someone break... Look for potential S...".

The browser's developer tools are open, showing the "Network" tab. The "Headers" sub-tab is selected, displaying the following information:

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														
Response Headers	<table border="1"><thead><tr><th>Header</th><th>Value</th></tr></thead><tbody><tr><td>Connection</td><td>keep-alive</td></tr><tr><td>Content-Encoding</td><td>gzip</td></tr><tr><td>Content-Type</td><td>text/html; charset=UTF-8</td></tr><tr><td>Date</td><td>Mon, 19 Jan 2026 22:34:19 GMT</td></tr><tr><td>Server</td><td>nginx/1.19.0</td></tr><tr><td>Transfer-Encoding</td><td>chunked</td></tr></tbody></table>	Header	Value	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
Header	Value														
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														

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 2.17.0 interface. The top menu bar includes File, Edit, View, Analyse, Report, Tools, Import, Export, Online, and Help. The main window is titled "Untitled Session - 20260120-174914 - ZAP 2.17.0". The left sidebar shows a tree view with "Sites", "Contexts", and "Default Context". The main area is titled "Automated Scan" and contains the following text:

This screen allows you to launch an automated scan against an application - just enter its URL below and press 'Attack'. Please be aware that you should only attack applications that you have been specifically given permission to test.

URL to attack:  Select...

Use traditional spider: ☒

Use ajax spider: ☐ If Modern ☐ with

Attack

Progress: Actively scanning (attacking) the URLs discovered by the spider(s)

The bottom status bar shows "Alerts 12", "Main Proxy: localhost:8090", and "Current Status" with various icons. The system tray at the bottom shows the date and time as 18:01 on 2026/01/20.

The screenshot shows the OWASP ZAP 2.17.0 interface with the "Alerts" tab selected. The left sidebar shows a tree view with "Alerts (14)". The main area displays the details of a "Cross Site Scripting (Reflected)" alert.

**Cross Site Scripting (Reflected)**

URL: <http://testphp.vulnweb.com/guestbook.php>

Risk: High

Confidence: Medium

Parameter: name

Attack: `</strong><script>alert(1);</script></strong>`

Evidence: `</strong><script>alert(1);</script></strong>`

CWE ID: 79

WASC ID: 8

Source: Active (40012 - Cross Site Scripting (Reflected))

Input Vector: Form Query

Description:

Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a browser object embedded in a software product such as the browser within WinAmp, an RSS reader, or an email client. The code itself is usually written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology.

Other Info:

Solution:

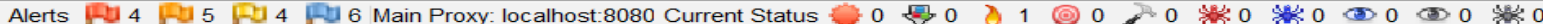
Phase: Architecture and Design

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. Examples of libraries and frameworks that make it easier to generate properly encoded output include Microsoft's Anti-XSS library, the OWASP ESAPI Encoding module, and Apache Wicket.

Reference:

<https://nwsn.nl/www-community/attacks/xss/>

The bottom status bar shows "Alerts 14", "Main Proxy: localhost:8090", and "Current Status" with various icons. The system tray at the bottom shows the date and time as 18:03 on 2026/01/20.



ATTACK Mode

Sites

Contexts

- Default Context

Sites

Quick StartRequestResponseRequester

Header: TextBody: Text

HTTP/1.1 200 OK

Server: nginx/1.19.0

Date: Tue, 20 Jan 2026 15:56:19 GMT

Content-Type: text/html; charset=UTF-8

Connection: keep-alive

X-Powered-By: PHP/5.6.40-38ubuntu20.04.1deb-cvnr-001

<!--end content -->

<div id="navBar">

<div id="search">

<form action="search.php?test=query" method="post">

HistorySearchAlertsOutputActive ScanSpiderAJAX Spider

Alerts (19)

- Cross Site Scripting (Reflected) (19)
- SQL Injection
- SQL Injection - MySQL (9)
- SQL Injection - MySQL (Time Based) (2)
- Absence of Anti-CSRF Tokens (Systemic)
- Content Security Policy (CSP) Header Not Set (Systemic)
- HTTP Only Site
- Missing Anti-clickjacking Header (Systemic)
- XSLT Injection (2)
- In Page Banner Information Leak (3)
- 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) (3)
- GET for POST (3)
- Modern Web Application (Systemic)
- User Agent Fuzzer (Systemic)
- User Controllable HTML Element Attribute (Potential XSS) (2)

Absence of Anti-CSRF Tokens

URL: http://testphp.vulnweb.com

Risk: Medium

Confidence: Low

Parameter:

Attack:

Evidence: <form action="search.php?test=query" method="post">

CWE ID: 352

WASC ID: 9

Source: Passive (10202 - Absence of Anti-CSRF Tokens)

Input Vector:

Description:

No Anti-CSRF tokens were found in a HTML submission form.

A cross-site request forgery is an attack that involves forcing

Other Info:

No known Anti-CSRF token [anticsrf, CSRFToken, \_\_RequestVerificationToken, csrfmiddlewaretoken, authenticity\_token, OWASP\_CSRFTOKEN, anoncsrf,

Solution:

Phase: Architecture and Design

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this

Reference:

Alerts 4 5 4 6 Main Proxy: localhost:8080 Current Status 0 0 1 0 0 0 0 0 0 0 0 0

## 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 a 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.