VULNERABILITY ASSESSMENT REPORT

testphp.vulnweb.com

P. CHARAN TEJA

Date : January 8, 2026

This report details a security assessment of the target website. We identified 14 total vulnerabilities during our scan. The primary concerns involve outdated software versions and lack of modern browser protections, which could lead to data theft if not addressed

OVERALL RISK RATING: HIGH

Summary of Findings:
We performed a passive security scan of testphp.vulnweb.com using OWASP ZAP. The scan identified 14 total alerts.
Critical Issue: We discovered a DOM-Based Cross-Site Scripting (XSS) vulnerability. This could allow an attacker to execute malicious code in a user's browser.
System Weakness: The server is running outdated software (PHP 5.6.40) that is no longer receiving security updates.
Conclusion: Immediate remediation is required to protect user data and server integrity.

Finding 1: Cross-Site Scripting (DOM-Based)

Risk Level : HIGH

Risk=High, Confidence=High (1)

http://testphp.vulnweb.com (1)

Cross Site Scripting (DOM Based) (1)

▼ GET http://testphp.vulnweb.com/#jaVasCript:/*-/*`/*\`/*'/*"/**/(/*
*/oNcliCk=alert(5397))//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/-
-!>\x3csVg/<sVg/oNloAd=alert(5397)//>\x3e

Alert tags

- WSTG-v42-CLNT-01
- CWE-79
- POLICY_SEQUENCE =
- OWASP_2021_A03
- POLICY_DEV_FULL =
- POLICY_QA_STD =
- POLICY_QA_FULL =
- OWASP_2017_A07

Alert 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.

When an attacker gets a user's browser to execute his/her code, the code will run within the security context (or zone) of the hosting web site. With this level of privilege, the code has the ability to read, modify and transmit any sensitive data accessible by the browser. A Cross-site Scripted user could have his/her account hijacked (cookie theft), their browser redirected to another location, or possibly shown fraudulent content delivered by the web site they are visiting. Cross-site Scripting

attacks essentially compromise the trust relationship between a user and the web site. Applications utilizing browser object instances which load content from the file system may execute code under the local machine zone allowing for system compromise.

There are three types of Cross-site Scripting attacks: non-persistent, persistent and DOM-based.

Non-persistent attacks and DOM-based attacks require a user to either visit a specially crafted link laced with malicious code, or visit a malicious web page containing a web form, which when posted to the vulnerable site, will mount the attack. Using a malicious form will oftentimes take place when the vulnerable resource only accepts HTTP POST requests. In such a case, the form can be submitted automatically, without the victim's knowledge (e.g. by using JavaScript). Upon clicking on the malicious link or submitting the malicious form, the XSS payload will get echoed back and will get interpreted by the user's browser and execute. Another technique to send almost arbitrary requests (GET and POST) is by using an embedded client, such as Adobe Flash.

Persistent attacks occur when the malicious code is submitted to a web site where it's stored for a period of time. Examples of an attacker's favorite targets often include message board posts, web mail messages, and web chat software. The unsuspecting user is not required to interact with any additional site/link (e.g. an attacker site or a malicious link sent via email), just simply view the web page containing the code.

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. Phases: Implementation; Architecture and Design Understand the context in which your data will be used and the encoding that will be expected. This is especially important when transmitting data between different components, or when generating outputs that can contain multiple encodings at the same time, such as web pages or multi-part mail messages. Study all expected communication protocols and data representations to determine the required encoding strategies. For any data that will be output to another web page, especially any data that was received from external inputs, use the appropriate encoding on all non-alphanumeric characters. Consult the XSS Prevention Cheat Sheet for more details on the types of encoding and escaping that are needed. Phase: Architecture and Design For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated.

Phase: Implementation

For every web page that is generated, use and specify a character encoding such as ISO-8859-1 or UTF-8. When an encoding is not specified, the web browser may choose a different encoding by guessing which encoding is actually being used by the web page. This can cause the web browser to treat certain sequences as special, opening up the client to subtle XSS attacks. See CWE-116 for more mitigations related to encoding/escaping.

To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set.

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use an allow list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a deny list). However, deny lists can be useful for detecting potential attacks or determining which inputs are so malformed that they should

be rejected outright.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue."

Ensure that you perform input validation at well-defined interfaces within the application. This will help protect the application even if a component is reused or moved elsewhere.

What is it?

- This flaw occurs when the website's code (JavaScript) handles data from a user in an unsafe way.
- An attacker can 'inject' their own code into the page to run malicious actions.

Impact :

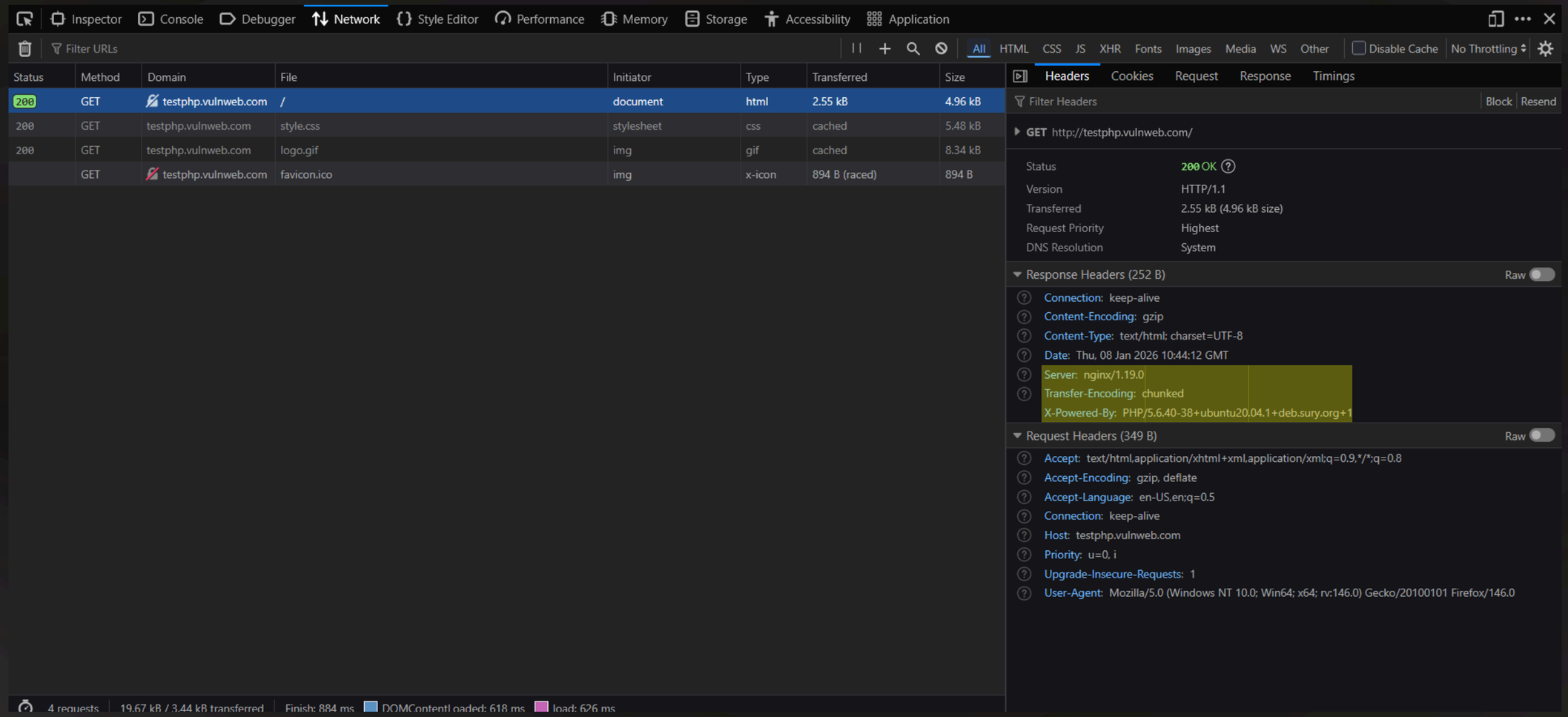
- A hacker could steal your customers' session cookies or login details.
- They could also redirect your visitors to a fraudulent website without them knowing.

Solution :

- Apply strict input validation to ensure only safe data is accepte.
- Use modern security libraries (like OWASP ESAPI) to 'clean' all data before it is shown on the screen.

Finding 2: Outdated Server Software & Information Leak

Risk Level : HIGH



What is it?

- This vulnerability occurs when the web server reveals the specific version of software it is running (Nginx 1.19.0 and PHP 5.6.40).
- It provides a "blueprint" for attackers to target the site with specific hacks.

Impact :

- Attackers can search for known security flaws (CVEs) tied to these exact versions to take control of the server.
- PHP 5.6.40 is no longer supported, meaning it contains unpatched security holes.

Solution :

- Upgrade the server to a modern version, such as PHP 8.2 or higher, immediately.
- Configure the server to hide the "Server" and "X-Powered-By" headers so attackers cannot see the software details.

Finding 3: Content Security Policy (CSP)

Risk Level : MEDIUM

Risk=Medium, Confidence=High (1)

http://testphp.vulnweb.com (1)

Content Security Policy (CSP) Header Not Set (1)

▼ GET http://testphp.vulnweb.com/

Alert tags	▪ OWASP_2021_A05 ▪ POLICY_QA_STD = ▪ POLICY_PENTEST = ▪ SYSTEMIC ▪ CWE-693 ▪ OWASP_2017_A06
Alert description	Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware. CSP provides a set of standard HTTP headers that allow website owners to declare approved sources of content that browsers should be allowed to load on that page — covered types are JavaScript, CSS, HTML frames, fonts, images and embeddable objects such as Java applets, ActiveX, audio and video files.
Solution	Ensure that your web server, application server, load balancer, etc. is configured to set the Content-Security-Policy header.

What is it?

- CSP is an added layer of security that helps detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks

Impact :

- Without it, attackers can use the site for data theft, site defacement, or the distribution of malware.

Solution :

- Configure your web server, application server, or load balancer to set the Content-Security-Policy header.

Finding 4: Missing Anti-clickjacking Header

Risk Level : MEDIUM

Risk=Medium, Confidence=Medium (1)

http://testphp.vulnweb.com (1)

Missing Anti-clickjacking Header (1)

▼ GET http://testphp.vulnweb.com/

Alert tags	▪ OWASP_2021_A05 ▪ POLICY_QA_STD = ▪ POLICY_PENTEST = ▪ CWE-1021 ▪ SYSTEMIC ▪ WSTG-v42-CLNT-09 ▪ OWASP_2017_A06
Alert description	The response does not protect against 'ClickJacking' attacks. It should include either Content-Security-Policy with 'frame-ancestors' directive or X-Frame-Options.
Solution	Modern Web browsers support the Content-Security-Policy and X-Frame-Options HTTP headers. Ensure one of them is set on all web pages returned by your site/app. If you expect the page to be framed only by pages on your server (e.g. it's part of a FRAMESET) then you'll want to use SAMEORIGIN, otherwise if you never expect the page to be framed, you should use DENY. Alternatively consider implementing Content Security Policy's "frame-ancestors" directive.

What is it?

- The website does not include security headers (like X-Frame-Options) that prevent the page from being loaded inside an iframe on another site.

Impact :

- Attackers can use this to perform 'Clickjacking'. They can trick a user into clicking something on your site (like a 'Delete Account' button) by overlaying it with a fake, invisible layer

Solution :

- Configure the web server to include the X-Frame-Options header. Use the value SAMEORIGIN to allow your own site to frame the page, or DENY to block all framing.

Conclusion & Remediation Roadmap :

1. Executive Summary

- The security assessment of testphp.vulnweb.com revealed 14 total vulnerabilities. The most critical risks involve the use of unsupported server software (PHP 5.6.40) and a DOM-based Cross-Site Scripting (XSS) flaw.
- If left unpatched, these vulnerabilities could lead to full server compromise or the theft of user session data.

2. Next Steps (Action Plan) ✅

Use a numbered list to show priority:

- High Priority: Upgrade the server to PHP 8.2 or higher and update the Nginx web server.
- High Priority: Implement strict Input Validation and context-aware encoding to fix the XSS vulnerability.


Thank
you