Website Vulnerability Report: <http://testphp.vulnweb.com/>

High

Overall risk level:

Risk ratings:

High

Low

Medium

Scan information:

Start time: Nov 13, 2023 /13:53:03

Finish time: Nov 13, 2023 /13:53:19

Scan duration: 16 sec

Tests performed: 19/19

Submitted by :

Pollos

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Risk Level | CVSS | CVE | Summary | Exploit | Affected software |
| High | 7.5 | https://nvd.nist.gov/vuln/detail/CVE-2017-8923 | The zend\_string\_extend function in Zend/zend\_string.h in PHP through 7.1.5 does not prevent changes to string objects that result in a negative length, which allows remote attackers to cause a denial of service (application crash) or possibly have unspecified other impact by leveraging a script's use of .= with a long string. | N/A | php 5.6.40 |
| High | 7.5 | https://nvd.nist.gov/vuln/detail/CVE-2019-9641 | An issue was discovered in the EXIF component in PHP before 7.1.27, 7.2.x before 7.2.16, and 7.3.x before 7.3.3. There is an uninitialized read in exif\_process\_IFD\_in\_TIFF. | N/A | php 5.6.40 |
| High | 6.8 | https://nvd.nist.gov/vuln/detail/CVE-2015-9253 | An issue was discovered in PHP 7.3.x before 7.3.0alpha3, 7.2.x before 7.2.8, and before 7.1.20. The php-fpm master process restarts a child process in an endless loop when using program execution functions (e.g., passthru, exec, shell\_exec, or system) with a non-blocking STDIN stream, causing this master process to consume 100% of the CPU, and consume disk space with a large volume of error logs, as demonstrated by an attack by a customer of a shared-hosting facility. | N/A | php 5.6.40 |
| High | 6.5 | https://nvd.nist.gov/vuln/detail/CVE-2022-31629 | In PHP versions before 7.4.31, 8.0.24 and 8.1.11, the vulnerability enables network and same-site attackers to set a standard insecure cookie in the victim's browser which is treated as a `\_\_Host-` or `\_\_Secure-` cookie by PHP applications. | N/A | php 5.6.40 |
| High | 5.8 | https://nvd.nist.gov/vuln/detail/CVE-2017-7272 | PHP through 7.1.11 enables potential SSRF in applications that accept an fsockopen or pfsockopen hostname argument with an expectation that the port number is constrained. Because a :port syntax is recognized, fsockopen will use the port number that is specified in the hostname argument, instead of the port number in the second argument of the function. | N/A | php 5.6.40 |

Risk description: These vulnerabilities expose the affected applications to the risk of unauthorized access to confidential data and possibly to denial of service attacks. An attacker could search for an appropriate exploit (or create one himself) for any of these vulnerabilities and use it to attack the system.

Recommendation: We recommend you to upgrade the affected software to the latest version in order to eliminate the risk of these vulnerabilities.

Classification:

CWE : [CWE - CWE-1026: Weaknesses in OWASP Top Ten (2017) (4.13) (mitre.org)](https://cwe.mitre.org/data/definitions/1026.html)

OWASP Top 10 - 2013 : [OWASP Top 10 - 2013](https://owasp.org/www-pdf-archive/OWASP_Top_10_-_2013.pdf#page=15)

OWASP Top 10 - 2017 : [Search | OWASP Foundation](https://owasp.org/search/?searchString=A9%20-%20Using%20Components%20with%20Known%20Vulnerabilities%202017)

Communication is not secure

|  |  |
| --- | --- |
| URL | Evidence |
| [Home of Acunetix Art (vulnweb.com)](http://testphp.vulnweb.com/) | Communication is made over unsecure, unencrypted HTTP. |

Risk description: The communication between the web browser and the server is done using the HTTP protocol, which transmits data unencrypted over the network. Thus, an attacker who manages to intercept the communication at the network level is able to read and modify the data transmitted (including passwords, secret tokens, credit card information and other sensitive data).

Recommendation: We recommend you to reconfigure the web server to use HTTPS - which encrypts the communication between the web browser and the server.

Classification:

CWE : CWE-311

OWASP Top 10 - 2013 : A6 - Sensitive Data Exposure

OWASP Top 10 - 2017 : A3 - Sensitive Data Exposure

Issue detail

The name of an arbitrarily supplied URL parameter is copied into the HTML document as plain test between tags. The payload <script>alert(1)</script> was submitted in the name of an arbitrarily supplied URL parameter. This input was echoed unmodified in the application's response.

This behavior demonstrates that it is possible to inject new HTML tags into the returned document. An attempt was made to identify a full proof-of-concept attack for injecting arbitrary JavaScript but this was not successful.. You should manually examine the application's behavior and attempt to identify any unusual input validation or other obstacles that may be in place.

Issue background

Reflected cross-site scripting vulnerabilities arise when data is copied from a request and echoed into the application's immediate response in an unsafe way. An attacker can use the vulnerability to construct a request that, if issued by another application user, will cause JavaScript code supplied by the attacker to execute within the user's browser in the context of that user's session with the application.

The attacker-supplied code can perform a wide variety of actions, such as stealing the victim's session token or login credentials, performing arbitrary actions on the victim's behalf, and logging their keystrokes.

Users can be induced to issue the attacker's crafted request in various ways. For example, the attacker can send a victim a link containing a malicious URL in an email or instant message. They can submit the link to popular web sites that allow content authoring, for example in blog comments. And they can create an innocuous looking web site that causes anyone viewing it to make arbitrary cross-domain requests to the vulnerable application (using either the GET or the POST method).

The security impact of cross-site scripting vulnerabilities is dependent upon the nature of the vulnerable application, the kinds of data and functionality that it contains, and the other applications that belong to the same domain and organization. If the application is used only to display non-sensitive public content, with no authentication or access control functionality, then a cross-site scripting flaw may be considered low risk However, if the same application resides on a domain that can access cookies for other more security-critical applications, then the vulnerability could be used to attack those other applications, and so may be considered high risk Similarly, if the organization that owns the application is a likely target for phishing attacks, then the vulnerability could be leveraged to lend credibility to such attacks, by injecting Trojan functionality into the vulnerable application and exploiting users' trust in the organization in order to capture credentials for other

applications that it owns. In many kinds of application, such as those providing online banking functionality.

cross-site scripting should always be considered high risk

Issue remediation

In most situations where user-controllable data is copied into application responses, cross-site scripting attacks

can be prevented using two layers of defenses

Input should be validated as strictly as possible on arrival, given the kind of content that it is expected

to contain. For example, personal names should consist of alphabetical and a small range of

typographical characters, and be relatively short, a year of birth should consist of exactly four

numerals; email addresses should match a well-defined regular expression. Input which fails the

validation should be rejected, not sanitized

• User input should be HTML-encoded at any point where it is copied into application responses. All HTML metacharacters, including <>and = should be replaced with the corresponding HTML entities (&lt;&gt; etc)

in cases where the application's functionality allows users to author content using a restricted subset of HTML

tags and attributes (for example, blog comments which allow limited formatting and linking), it is necessary to

parse the supplied HTML to validate that it does not use any dangerous syntax: this is a non-trivial task.

References

1)Cross-site scripting

2)Reflected cross-site scripting

3)Using Burp to Find XSS issues

Vulnerability classifications

CWE-79: Improper Neutralization of Input During Web Page Generation (Cross-site Scripting")

CWE-80: Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)

CWE-116: Improper Encoding or Escaping of Output

CWE-159: Failure to Sanitize Special Element