

VAPT ASSESSMENT

report

Name : Prashant kumar Yadav

Report creation date : 09-02-2025

Table of Contents

Summary	3
1.Cross-Site Scripting (Self)	4
2. Flash cross-domain policy	7
3. Unencrypted communications	9
4. Cross-domain Referrer leakage	11
5. Frameable response (potential Clickjacking)	14
6. Email addresses disclosed	16
7.Blind SQL	17
8.LFI (Local File Inclusion)	19
9. Description and Risk Rating	21

Executive Summary

To understand security risk and system vulnerabilities testphp.vulnweb.com approached us to give assessment report with consultation service.

Security risk analysis, otherwise known as risk assessment, is fundamental to the security of any organization. It is essential in ensuring that controls and expenditure are fully commensurate with the risks to which the organization is exposed.

This report provides the risk assessment of all public and private portal of services provided by testphp.vulnweb.com. This will include authenticated as well as unauthenticated penetration testing of portals. This will cover backend infrastructure scanning for possible attack surface like brute force, unauthorized access, data leakage and vulnerability exploitation.

Purpose

The purpose of this Assessment Summary Report is to provide the Certifier and the Designated Approving Authority with a more holistic view of risk regarding the system. It documents the security assessment activities that were performed on the system and the results of those activities.

This report provides the system's stakeholders with an assessment of the adequacy of the management, operational, and technical controls used to protect the confidentiality, integrity, and availability of the system and the data it stores, transmits or processes.

1. Cross-site scripting (self)

Summary

Severity:	Medium
Confidence:	Firm
Host:	http://testphp.vulnweb.com/
Path:	/robots.txt

Issue detail

The name of an arbitrarily supplied URL parameter is copied into the HTML document as plain text 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 (< > 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

- Cross-site scripting
- Reflected cross-site scripting • Using Burp to Find XSS issues

Vulnerability classifications

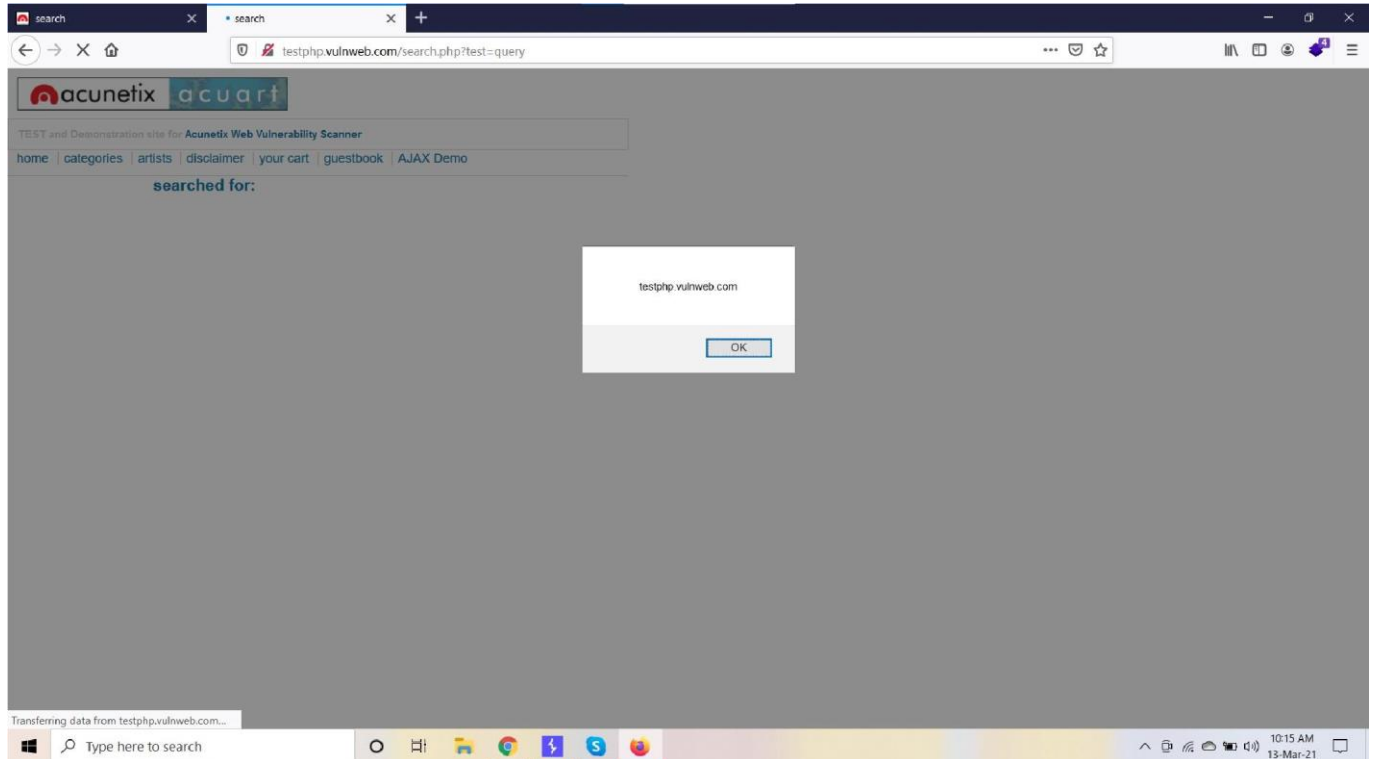
- CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
- CWE-80: Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)
- CWE-116: Improper Encoding or Escaping of Output
- CWE-159: Failure to Sanitize Special Element

Request

```
POST /search.php?test=query HTTP/1.1
Host: testphp.vulnweb.com
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en-US,en-GB;q=0.9,en;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/88.0.4324.150 Safari/537.36
Connection: close
Cache-Control: max-age=0
Referer: http://testphp.vulnweb.com/index.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 61

searchFor=<script>alert (document.domain)</script>&goButton=go
```

Response



2. Flash cross-domain policy

Summary

Summary

	Severity:	High
	Confidence:	Certain
	Host:	http://testphp.vulnweb.com
	Path:	/crossdomain.xml

Issue detail

The application publishes a Flash cross-domain policy which allows access from any domain.

Allowing access from all domains means that any domain can perform two-way interaction with this application. Unless the application consists entirely of unprotected public content, this policy is likely to present a significant security risk.

Issue background

The Flash cross-domain policy controls whether Flash client components running on other domains can perform two-way interaction with the domain that publishes the policy. If another domain is allowed by the policy, then that domain can potentially attack users of the application. If a user is logged in to the application, and visits a domain allowed by the policy, then any malicious content running on that domain can potentially gain full access to the application within the security context of the logged in user.

Even if an allowed domain is not overtly malicious in itself, security vulnerabilities within that domain could potentially be leveraged by a third-party attacker to exploit the trust relationship and attack the application that allows access. Any domains that are allowed by the Flash crossdomain policy should be reviewed to determine whether it is appropriate for the application to fully trust both their intentions and security posture.

Issue remediation

Any inappropriate entries in the Flash cross-domain policy file should be removed.

Vulnerability classifications

- CWE-942: Overly Permissive Cross-domain Whitelist
-

Request

```
GET /crossdomain.xml HTTP/1.1
Host: testphp.vulnweb.com
Connection: close
```


Response

```
HTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Sat, 13 Mar 2021 04:21:46 GMT
Content-Type: text/xml
Content-Length: 224
Last-Modified: Tue, 11 Sep 2012 10:30:22 GMT
Connection: close
ETag: "504f12be-e0"
Accept-Ranges: bytes

<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy SYSTEM "http://www.adobe.com/xml/dtds/cross-domain-policy.dtd">
<cross-domain-policy>
<allow-access-from domain="*" to-ports="*" secure="false"/>
...[SNIP]...
```


3. Unencrypted communications

Summary

	Severity:	Low
	Confidence:	Certain
	Host:	http://testphp.vulnweb.com
	Path:	/

Issue description

The application allows users to connect to it over unencrypted connections. An attacker suitably positioned to view a legitimate user's network traffic could record and monitor their interactions with the application and obtain any information the user supplies. Furthermore, an attacker able to modify traffic could use the application as a platform for attacks against its users and thirdparty websites. Unencrypted connections have been exploited by ISPs and governments to track users, and to inject adverts and malicious JavaScript. Due to these concerns, web browser vendors are planning to visually flag unencrypted connections as hazardous.

To exploit this vulnerability, an attacker must be suitably positioned to eavesdrop on the victim's network traffic. This scenario typically occurs when a client communicates with the server over an insecure connection such as public Wi-Fi, or a corporate or home network that is shared with a compromised computer. Common defenses such as switched networks are not sufficient to prevent this. An attacker situated in the user's ISP or the application's hosting infrastructure could also perform this attack. Note that an advanced adversary could potentially target any connection made over the Internet's core infrastructure.

Please note that using a mixture of encrypted and unencrypted communications is an ineffective defense against active attackers, because they can easily remove references to encrypted resources when these references are transmitted over an unencrypted connection.

Issue remediation

Applications should use transport-level encryption (SSL/TLS) to protect all communications passing between the client and the server. The Strict-Transport-Security HTTP header should be used to ensure that clients refuse to access the server over an insecure connection.

References

- Marking HTTP as non-secure
- Configuring Server-Side SSL/TLS
- HTTP Strict Transport Security **Vulnerability**

classifications

- CWE-326: Inadequate Encryption Strength

4. Cross-domain Referrer leakage

Summary

	Severity:	Information
	Confidence:	Certain
	Host:	http://testphp.vulnweb.com
	Path:	/listproducts.php

Issue detail

The page was loaded from a URL containing a query string:

- <http://testphp.vulnweb.com/listproducts.php>

The response contains the following links to other domains:

- <http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab>
- <http://www.acunetix.com/>
- <https://www.acunetix.com/>
- <https://www.acunetix.com/blog/articles/prevent-sql-injection-vulnerabilities-inphp-applications/>
- <https://www.acunetix.com/vulnerability-scanner/>
- <https://www.acunetix.com/vulnerability-scanner/php-security-scanner/>
- <http://www.electasy.com/Fractal-Explorer/index.html>

Issue background

When a web browser makes a request for a resource, it typically adds an HTTP header, called the "Referer" header, indicating the URL of the resource from which the request originated. This occurs in numerous situations, for example when a web page loads an image or script, or when a user clicks on a link or submits a form.

If the resource being requested resides on a different domain, then the Referer header is still generally included in the cross-domain request. If the originating URL contains any sensitive information within its query string, such as a session token, then this information will be transmitted to the other domain. If the other domain is not fully trusted by the application, then this may lead to a security compromise.

You should review the contents of the information being transmitted to other domains, and also determine whether those domains are fully trusted by the originating application.

Today's browsers may withhold the Referer header in some situations (for example, when loading a non-HTTPS resource from a page that was loaded over HTTPS, or when a Refresh directive is issued), but this behavior should not be relied upon to protect the originating URL from disclosure.

Note also that if users can author content within the application then an attacker may be able to inject links referring to a domain they control in order to capture data from URLs used within the application.

Issue remediation

Applications should never transmit any sensitive information within the URL query string. In addition to being leaked in the Referer header, such information may be logged in various locations and may be visible on-screen to untrusted parties. If placing sensitive information in the URL is unavoidable, consider using the Referer-Policy HTTP header to reduce the chance of it being disclosed to third parties.

References

- Referer Policy

Vulnerability classifications

- CWE-200: Information Exposure

Request

```
GET /listproducts.php?cat=1 HTTP/1.1
Host: testphp.vulnweb.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:86.0) Gecko/20100101 Firefox/86.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Referer: http://testphp.vulnweb.com/categories.php
Upgrade-Insecure-Requests: 1
```

Response

```
HTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Sat, 13 Mar 2021 04:34:24 GMT
Content-Type: text/html; charset=UTF-8
Connection: close
X-Powered-By: PHP/5.6.40-38+ubuntu20.04.1+deb.sury.org+1
Content-Length: 7880

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html><!-- InstanceBegin template="/Templates/main_dynamic_template.dwt.php" codeOutsideHTMIsLoc
...[SNIP]...
<p>
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000" codebase="http://download.macromedia.com/pub/shockwave/cabs/flash
/swflash.cab#version=6,0,29,0" width="107" height="66">
<param name="movie" value="Flash/add.swf">
...[SNIP]...
<div id="siteInfo"> <a href="http://www.acunetix.com">About Us</a>
...[SNIP]...
```

5. Frameable response (potential Clickjacking)

There are 5 instances of this issue:

- /
- /categories.php
- /comment.php
- /guestbook.php • /listproducts.php

Issue description

If a page fails to set an appropriate X-Frame-Options or Content-Security-Policy HTTP header, it might be possible for a page controlled by an attacker to load it within an iframe. This may enable a clickjacking attack, in which the attacker's page overlays the target application's interface with a different interface provided by the attacker. By inducing victim users to perform actions such as mouse clicks and keystrokes, the attacker can cause them to unwittingly carry out actions within the application that is being targeted. This technique allows the attacker to circumvent defenses against cross-site request forgery, and may result in unauthorized actions.

Note that some applications attempt to prevent these attacks from within the HTML page itself, using "framebusting" code. However, this type of defense is normally ineffective and can usually be circumvented by a skilled attacker.

You should determine whether any functions accessible within frameable pages can be used by application users to perform any sensitive actions within the application.

Issue remediation

To effectively prevent framing attacks, the application should return a response header with the name **X-Frame-Options** and the value **DENY** to prevent framing altogether, or the value **SAMEORIGIN** to allow framing only by pages on the same origin as the response itself. Note that the SAMEORIGIN header can be partially bypassed if the application itself can be made to frame untrusted websites.

References

- X-Frame-Options

Vulnerability classifications

- CWE-693: Protection Mechanism Failure

6. Email addresses disclosed

There are 4 instances of this issue:

- /
- /categories.php
- /guestbook.php • /listproducts.php

Issue background

The presence of email addresses within application responses does not necessarily constitute a security vulnerability. Email addresses may appear intentionally within contact information, and many applications (such as web mail) include arbitrary third-party email addresses within their core content.

However, email addresses of developers and other individuals (whether appearing on-screen or hidden within page source) may disclose information that is useful to an attacker; for example, they may represent usernames that can be used at the application's login, and they may be used in social engineering attacks against the organization's personnel. Unnecessary or excessive disclosure of email addresses may also lead to an increase in the volume of spam email received.

Issue remediation

Consider removing any email addresses that are unnecessary, or replacing personal addresses with anonymous mailbox addresses (such as `helpdesk@example.com`).

To reduce the quantity of spam sent to anonymous mailbox addresses, consider hiding the email address and instead providing a form that generates the email server-side, protected by a CAPTCHA if necessary.

Vulnerability classifications

- CWE-200: Information Exposure

7.Blind SQL

Issue Background

SQL injection is a code injection technique, used to attack data-driven applications, in which malicious SQL statements are inserted into an entry field for execution (e.g. to dump the database contents to the attacker).[1] SQL injection must exploit a security vulnerability in an application's software, for example, when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and unexpectedly executed. SQL injection is mostly known as an attack vector for websites but can be used to attack any type of SQL database.

Request

```
GET /product.php?pic=1' HTTP/1.1
Host: testphp.vulnweb.com
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en-US,en-GB;q=0.9,en;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/88.0.4324.150 Safari/537.36
Connection: close
Cache-Control: max-age=0
Referer: http://testphp.vulnweb.com/listproducts.php?cat=1
```


Response



TEST and Demonstration site for **Acunetix Web Vulnerability Scanner**

[home](#) | [categories](#) | [artists](#) | [disclaimer](#) | [your cart](#) | [guestbook](#) | [AJAX Demo](#)

search art

[Browse categories](#)

[Browse artists](#)

[Your cart](#)

[Signup](#)

[Your profile](#)

[Our guestbook](#)

[AJAX Demo](#)

Links

[Security art](#)

[PHP scanner](#)

[PHP vuln help](#)

[Fractal Explorer](#)

couldn't load plugi

Warning: mysql_fetch_array() expects parameter 1 to be resou
boolean given in /hj/var/www/product.php on line 70

[About Us](#) | [Privacy Policy](#) | [Contact Us](#) | ©2019 Acunetix Ltd

8.LFI (Local File Inclusion)

A file inclusion vulnerability is a type of web vulnerability that is most commonly found to affect web applications that rely on a scripting run time. This issue is caused when an application builds a path to executable code using an attacker-controlled variable in a way that allows the attacker to control which file is executed at run time. A file include

vulnerability is distinct from a generic directory traversal attack, in that directory traversal is a way of gaining unauthorized file system access, and a file inclusion vulnerability subverts how an application loads code for execution. Successful exploitation of a file inclusion vulnerability will result in remote code execution on the web server that runs the affected web application. An attacker can use remote code execution to create a web shell on the web server, which can be used for website defacement.

Request

```
GET /showimage.php?file=%2e%2e%2f%2e%2e%2fetcc%2fpasswd HTTP/1.1
Host: testphp.vulnweb.com
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en-US,en-GB;q=0.9,en;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/88.0.4324.150 Safari/537.36
Connection: close
Cache-Control: max-age=0
Referer: http://testphp.vulnweb.com/listproducts.php?cat=1
```

Response

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
nobody:x:65534:1002:nobody:/nonexistent:/bin/sh
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
syslog:x:101:102::/home/syslog:/bin/false
klog:x:102:103::/home/klog:/bin/false
mysql:x:103:107:MySQL Server,,,:/var/lib/mysql:/bin/false
bind:x:104:111::/var/cache/bind:/bin/false
sshd:x:105:65534::/var/run/sshd:/usr/sbin/nologin
```

9. Description and Risk Rating

S.No.	Description	Risk Rating	Status
1	Cross-Site Scripting (Self)	Medium	Found
2	Flash cross-domain policy	High	Found
3	Unencrypted communications	Low	Found
4	Cross-domain Referrer leakage	Low	Found
5	Frameable response (potential Clickjacking)	Low	Found
6	Email addresses disclosed	Low	Found
7	Blind sqli	High	Found

8	LFI (Local File Inclusion)	High	Found
9	HTML Injection	Medium	Not Found
10	Parameter Tampering	High	Not Found
11	Server-Side Request Forgery	Medium	Not Found
12	Client-Side Request Forgery	Medium	Not Found
13	Command Injection	Medium	Not Found
14	Host Header Attack	Low	Not Found
15	XML Entity Attack	Medium	Not Found