ZAP Scanning Report

Generated with ZAP on Tue 14 Feb 2023, at 13:24:51

Contents

About this report

Report parameters

Contexts

No contexts were selected, so all contexts were included by default.

Sites

The following sites were included:

http://testphp.vulnweb.com

(If no sites were selected, all sites were included by default.)

An included site must also be within one of the included contexts for its data to be included in the report.

Risk levels

Included: High, Medium, Low, Informational

Excluded: None

Confidence levels

Included: User Confirmed, High, Medium, Low

Excluded: User Confirmed, High, Medium, Low, False Positive

Summaries

Alert counts by risk and confidence

This table shows the number of alerts for each level of risk and confidence included in the report.

(The percentages in brackets represent the count as a percentage of the total number of alerts included in the report, rounded to one decimal place.)

		Confidence				
		User Confirmed	High	Medium	Low	Total
Risk	High	0 (0.0%)	0 (0.0%)	3 (16.7%)	1 (5.6%)	4 (22.2%)
	Medium	0 (0.0%)	1 (5.6%)	3 (16.7%)	1 (5.6%)	5 (27.8%)
	Low	0 (0.0%)	1 (5.6%)	2 (11.1%)	0 (0.0%)	3 (16.7%)
	Informational	0 (0.0%)	1 (5.6%)	2 (11.1%)	3 (16.7%)	6 (33.3%)
	Total	0 (0.0%)	3 (16.7%)	10 (55.6%)	5 (27.8%)	18 (100%)

Alert counts by site and risk

This table shows, for each site for which one or more alerts were raised, the number of alerts raised at each risk level.

Alerts with a confidence level of "False Positive" have been excluded from these counts.

(The numbers in brackets are the number of alerts raised for the site at or above that risk level.)

		Risk			
		High (= High)	Medium (>= Medium)	Low (>= Low)	Informational (>= Informational)
Site	http://testphp.vulnweb.com	4 (4)	5 (9)	3 (12)	6 (18)

Alert counts by alert type

This table shows the number of alerts of each alert type, together with the alert type's risk level.

(The percentages in brackets represent each count as a percentage, rounded to one decimal place, of the total number of alerts included in this report.)

Alert type	Risk	Count
Cross Site Scripting (Reflected)	High	3 (16.7%)
Path Traversal	High	1 (5.6%)
SQL Injection	High	3 (16.7%)
SQL Injection - MySQL	High	7 (38.9%)
.htaccess Information Leak	Medium	7 (38.9%)
Absence of Anti-CSRF Tokens	Medium	41 (227.8%)
Content Security Policy (CSP) Header Not Set	Medium	49 (272.2%)
Missing Anti-clickjacking Header	Medium	45 (250.0%)
XSLT Injection	Medium	2 (11.1%)

This table shows the number of alerts of each alert type, together with the alert type's risk level.

(The percentages in brackets represent each count as a percentage, rounded to one decimal place, of the total number of alerts included in this report.)

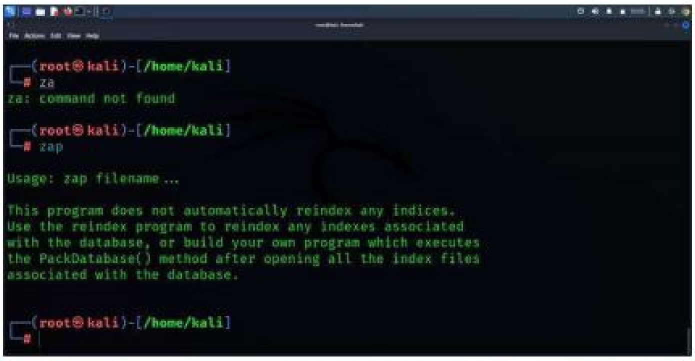
Alert type	Risk	Count
Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)	Low	63 (350.0%)
Server Leaks Version Information via "Server" HTTP Response Header Field	Low	75 (416.7%)
X-Content-Type-Options Header Missing	Low	69 (383.3%)
Charset Mismatch (Header Versus Meta Content-Type Charset)	Informational	32 (177.8%)
GET for POST	Informational	1 (5.6%)
Information Disclosure - Suspicious Comments	Informational	1 (5.6%)
Modern Web Application	Informational	9 (50.0%)
User Agent Fuzzer	Informational	197 (1,094.4%)
User Controllable HTML Element Attribute (Potential XSS)	Informational	3 (16.7%)
Total		18

STEPS:

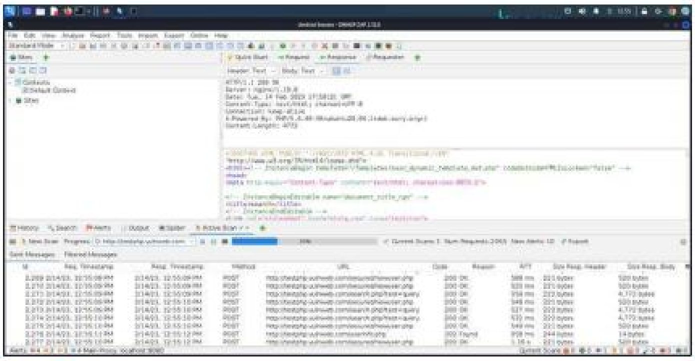
Scanning by zaproxy

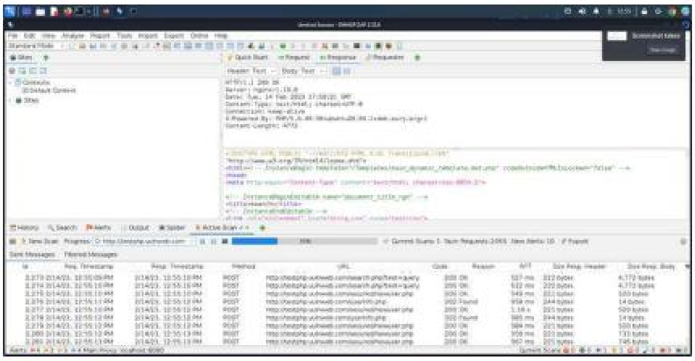
1. **open website vulnweb on which we will test**
2. **open zap**
3. **then type the site that you want to scan**
4. **click on automated scan**
5. **once scan is complete click on genrate report in html format**

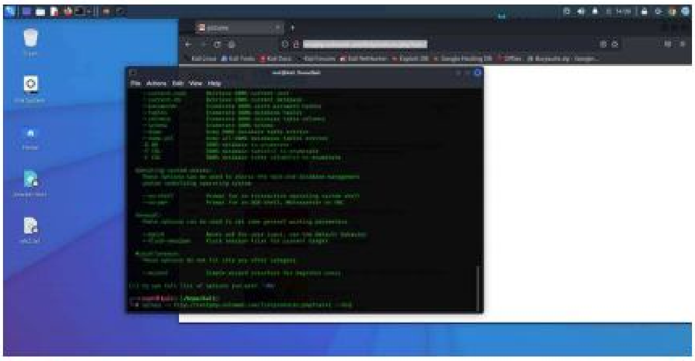
SCREENSHOTS OF SCANNING USING OWASP ZAP AND ENTERING INTO DATABASE USING SQLMAP

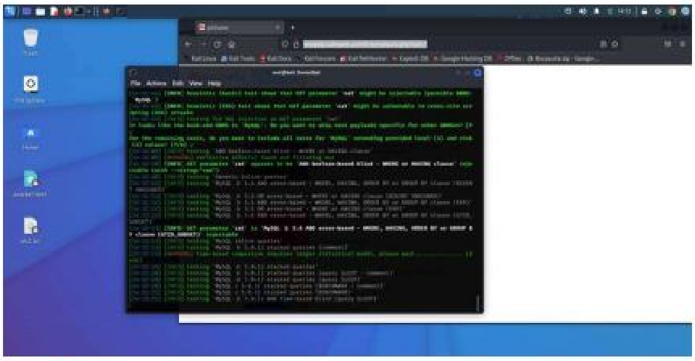


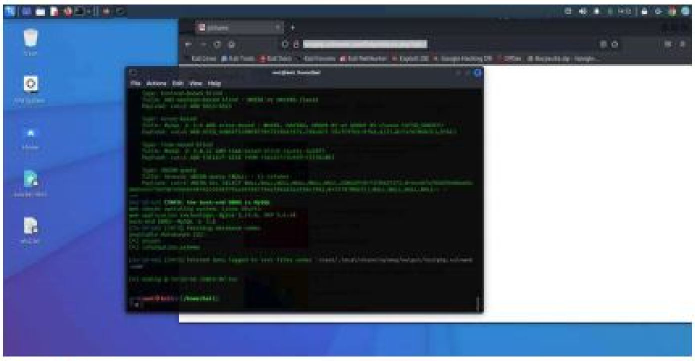
```
┌──(root㉿kali)-[/home/kali]
└─# za
za: command not found

┌──(root㉿kali)-[/home/kali]
└─# zap

Usage: zap filename ...

This program does not automatically reindex any indices.
Use the reindex program to reindex any indexes associated
with the database, or build your own program which executes
the PackDatabase() method after opening all the index files
associated with the database.

┌──(root㉿kali)-[/home/kali]
└─#
```

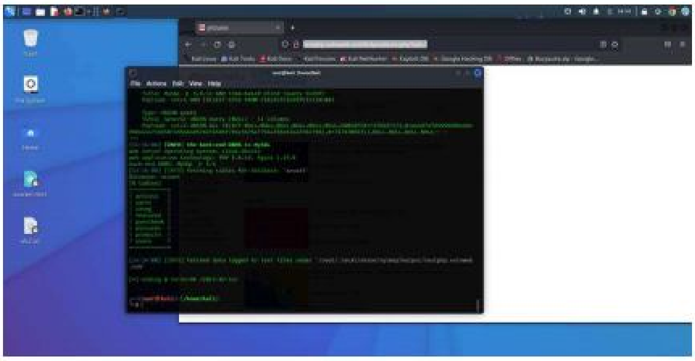


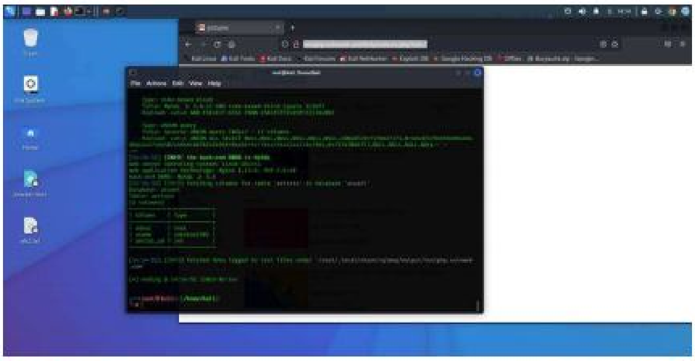


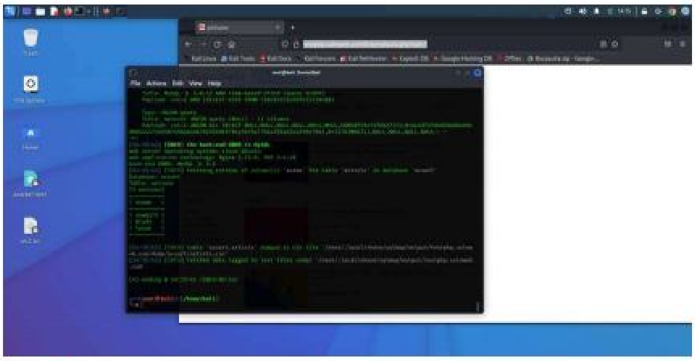


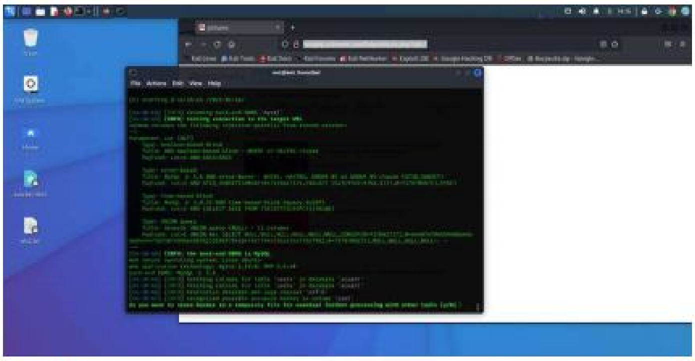


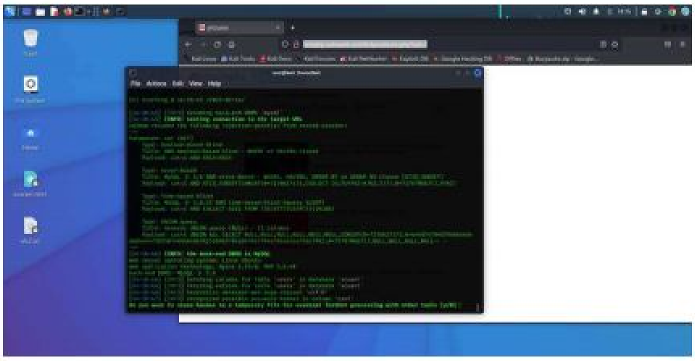


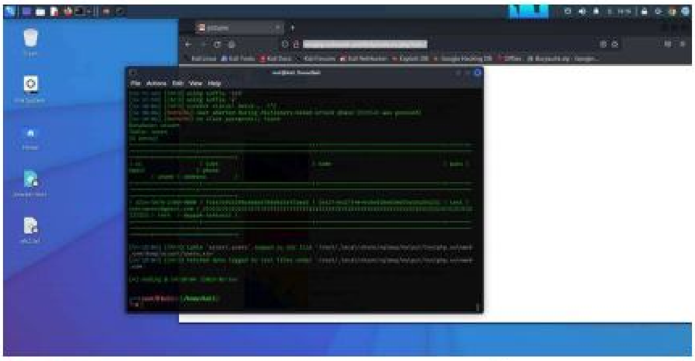


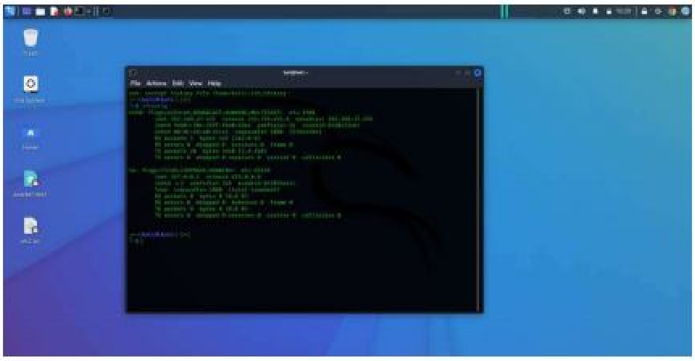












Alerts

1. **Risk=High, Confidence=Medium (3)**

    1. **http://testphp.vulnweb.com (3)**

        1. **Cross Site Scripting (Reflected) (1)**

            1.

            ▶ GET http://testphp.vulnweb.com/hpp/?pp=javascript%3Aalert%281%29%3B

gs
- OWASP_2021_A03
- WSTG-v42-INPV-01
- OWASP_2017_A07

ption 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 a

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.

t ▼
Request line and header section (286 bytes)

```
GET http://testphp.vulnweb.com/hpp/?pp=javascript%3Aalert%281%29%3B
HTTP/1.1
Host: testphp.vulnweb.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0)
Gecko/20100101 Firefox/105.0
Pragma: no-cache
Cache-Control: no-cache
Referer: http://testphp.vulnweb.com/hpp/
```

▼
Request body (0 bytes)

ce ▼
Status line and header section (221 bytes)

```
HTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Tue, 14 Feb 2023 16:42:58 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
X-Powered-By: PHP/5.6.40-38+ubuntu20.04.1+deb.sury.org+1
Content-Length: 445
```

▼
Response body (445 bytes)

```
<title>HTTP Parameter Pollution Example</title>

<a href="?pp=12">check</a><br/>
<a href="params.php?p=valid&pp=javascript%3Aalert%281%29%3B">link1</
a><br/><a href="params.php?p=valid&pp=javascript:alert(1);">link2</a><br
/><form action="params.php?p=valid&pp=javascript:alert(1);"><input
type=submit name=aaaa/></form><br/>
<hr>
<a href='http://blog.mindedsecurity.com/2009/05/client-side-http-
parameter-pollution.html'>Original article</a>
```

```
pp
javascript:alert(1);
javascript:alert(1);
```

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.

2. SQL Injection (1)

1.

▶ POST http://testphp.vulnweb.com/cart.php

:s
- OWASP_2021_A03
- WSTG-v42-INPV-05
- OWASP_2017_A01

ption SQL injection may be possible

fo The page results were successfully manipulated using the boolean conditions [10000' AND '1'='1] and

[10000' AND '1'='2]

Data was returned for the original parameter.

The vulnerability was detected by successfully restricting the data originally returned, by manipulating the

parameter

▼
Request line and header section (341 bytes)

```
POST http://testphp.vulnweb.com/cart.php HTTP/1.1
Host: testphp.vulnweb.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0)
Gecko/20100101 Firefox/105.0
Pragma: no-cache
Cache-Control: no-cache
Content-Type: application/x-www-form-urlencoded
Referer: http://testphp.vulnweb.com/product.php?pic=6
Content-Length: 43
```

▼
Request body (43 bytes)

```
price=10000%27+AND+%271%27%3D%271&addcart=6
```

e ▼
Status line and header section (222 bytes)

```
HTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Tue, 14 Feb 2023 16:44:58 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
X-Powered-By: PHP/5.6.40-38+ubuntu20.04.1+deb.sury.org+1
Content-Length: 4903
```

▶
Response body (4903 bytes)

```
price
10000' AND '1'='1
```

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.

If the application uses JDBC, use PreparedStatement or CallableStatement, with parameters passed by '?'

If the application uses ASP, use ADO Command Objects with strong type checking and parameterized queries.

If database Stored Procedures can be used, use them.

Do *not* concatenate strings into queries in the stored procedure, or use 'exec', 'exec immediate', or equivalent functionality!

Do not create dynamic SQL queries using simple string concatenation.

Escape all data received from the client.

Apply an 'allow list' of allowed characters, or a 'deny list' of disallowed characters in user input.

Apply the privilege of least privilege by using the least privileged database user possible.

In particular, avoid using the 'sa' or 'db-owner' database users. This does not eliminate SQL injection, but minimizes its impact.

Grant the minimum database access that is necessary for the application.

3. SQL Injection - MySQL (1)

1.

▶ POST http://testphp.vulnweb.com/secured/newuser.php

gs
- OWASP_2021_A03
- WSTG-v42-INPV-05
- OWASP_2017_A01

ption SQL injection may be possible

fo RDBMS [MySQL] likely, given UNION-specific message fragment [\QThe used SELECT statements have a different number of columns\E] in HTML results

t ▼

Request line and header section (346 bytes)

```
POST http://testphp.vulnweb.com/secured/newuser.php HTTP/1.1
Host: testphp.vulnweb.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0)
Gecko/20100101 Firefox/105.0
Pragma: no-cache
Cache-Control: no-cache
Content-Type: application/x-www-form-urlencoded
Referer: http://testphp.vulnweb.com/signup.php
Content-Length: 125
```

▼

Request body (125 bytes)

```
uuname=ZAP%27+UNION+ALL+select+NULL+--+&upass=ZAP&upass2=ZAP&urname=ZAP&
ucc=ZAP&uemail=ZAP&uphone=ZAP&uaddress
=&signup=signup
```

ie ▼

Status line and header section (221 bytes)

```
HTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Tue, 14 Feb 2023 17:50:37 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
X-Powered-By: PHP/5.6.40-38+ubuntu20.04.1+deb.sury.org+1
Content-Length: 482
```

▼

Response body (482 bytes)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>add new user</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<link href="style.css" rel="stylesheet" type="text/css">
</head>
<body>
<div id="masthead">
  <h1 id="siteName">ACUNETIX ART</h1>
</div>
<div id="content">
        Unable to access user database: The used SELECT statements have a
different number of columns
```

er
```
uuname
```

```
ZAP' UNION ALL select NULL --
```

e
```
The used SELECT statements have a different number of columns
```

1 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.

If the application uses JDBC, use PreparedStatement or CallableStatement, with parameters passed by '?'

If the application uses ASP, use ADO Command Objects with strong type checking and parameterized queries.

If database Stored Procedures can be used, use them.

Do *not* concatenate strings into queries in the stored procedure, or use 'exec', 'exec immediate', or equivalent functionality!

Do not create dynamic SQL queries using simple string concatenation.

Escape all data received from the client.

Apply an 'allow list' of allowed characters, or a 'deny list' of disallowed characters in user input.

Apply the privilege of least privilege by using the least privileged database user possible.

In particular, avoid using the 'sa' or 'db-owner' database users. This does not eliminate SQL injection,

but minimizes its impact.

Grant the minimum database access that is necessary for the application.

2. Risk=`High`, Confidence=`Low` (1)

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

1. Path Traversal (1)

1.

▶ POST http://testphp.vulnweb.com/search.php?test=query

gs
- OWASP_2021_A01
- WSTG-v42-ATHZ-01
- OWASP_2017_A05

ption The Path Traversal attack technique allows an attacker access to files, directories, and commands that potentially reside outside the web document root directory. An attacker may manipulate a URL in such a way that the web site will execute or reveal the contents of arbitrary files anywhere on the web server. Any device that exposes an HTTP-based interface is potentially vulnerable to Path Traversal.

Most web sites restrict user access to a specific portion of the file-system, typically called the "web document root" or "CGI root" directory. These directories contain the files intended for user access and the executable necessary to drive web application functionality. To access files or execute commands anywhere on the file-system, Path Traversal attacks will utilize the ability of special-characters sequences.

The most basic Path Traversal attack uses the "../" special-character sequence to alter the resource location requested in the URL. Although most popular web servers will prevent this technique from escaping the web document root, alternate encodings of the "../" sequence may help bypass the security filters. These method variations include valid and invalid Unicode-encoding ("..%u2216" or "..%c0%af") of the forward slash character, backslash characters ("..\") on Windows-based servers, URL encoded characters "%2e%2e%2f"), and double URL encoding ("..%255c") of the backslash character.

Even if the web server properly restricts Path Traversal attempts in the URL path, a web application itself may still be vulnerable due to improper handling of user-supplied input. This is a common problem of web applications that use template mechanisms or load static text from files. In variations of the attack, the original URL parameter value is substituted with the file name of one of the web application's dynamic scripts. Consequently, the results can reveal source code because the file is interpreted as text instead of an executable script. These techniques often employ additional special characters such as the dot (".") to reveal the listing of the current working directory, or "%00" NULL characters in order to bypass rudimentary file extension checks.

fo Check 5

t ▼

Request line and header section (336 bytes)

```
POST http://testphp.vulnweb.com/search.php?test=query HTTP/1.1
Host: testphp.vulnweb.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0) Gecko/
20100101 Firefox/105.0
Pragma: no-cache
Cache-Control: no-cache
Content-Type: application/x-www-form-urlencoded
Referer: http://testphp.vulnweb.com
Content-Length: 32
```

▼

Request body (32 bytes)

```
searchFor=search.php&goButton=go
```

ic ▼

Status line and header section (221 bytes)

```
HTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Tue, 14 Feb 2023 16:31:01 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
X-Powered-By: PHP/5.6.40-38+ubuntu20.04.1+deb.sury.org+1
Content-Length: 170
```

▼

Response body (170 bytes)

```
Warning: mysql_connect(): Connection refused in /hj/var/www/database_
connect.php on line 2
Website is out of order. Please visit back later. Thank you for
understanding.
```

er `searchFor`

`search.php`

1 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."

For filenames, use stringent allow lists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses, and exclude directory separators such as "/". Use an allow list of allowable file extensions.

Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A sanitizing mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the sanitizing mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the sanitizing mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass allow list schemes by introducing dangerous inputs after they have been checked.

Use a built-in path canonicalization function (such as realpath() in C) that produces the canonical version of the pathname, which effectively removes ".." sequences and symbolic links.

Run your code using the lowest privileges that are required to accomplish the necessary tasks. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment.

For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.

When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs.

Run your code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by your software.

OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows you to specify restrictions on file operations.

This may not be a feasible solution, and it only limits the impact to the operating system; the rest of your application may still be subject to compromise.

3. **Risk=Medium, Confidence=High (1)**

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

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

1.

▶ GET http://testphp.vulnweb.com/

gs
- OWASP_2021_A05
- OWASP_2017_A06

ption 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.

t ▼
Request line and header section (208 bytes)

```
GET http://testphp.vulnweb.com/ HTTP/1.1
Host: testphp.vulnweb.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0) Gecko/
20100101 Firefox/105.0
Pragma: no-cache
Cache-Control: no-cache
```

▼
Request body (0 bytes)

e ▼
Status line and header section (221 bytes)

```
HTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Tue, 14 Feb 2023 16:29:05 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
X-Powered-By: PHP/5.6.40-38+ubuntu20.04.1+deb.sury.org+1
Content-Length: 170
```

▼
Response body (170 bytes)

```
Warning: mysql_connect(): Connection refused in /hj/var/www/database_
connect.php on line 2
Website is out of order. Please visit back later. Thank you for
understanding.
```

n Ensure that your web server, application server, load balancer, etc. is configured to set the Content-Security-Policy header, to achieve optimal browser support: "Content-Security-Policy" for Chrome 25+, Firefox 23+ and Safari 7+, "X-Content-Security-Policy" for Firefox 4.0+ and Internet Explorer 10+, and "X-WebKit-CSP" for Chrome 14+ and Safari 6+.

4. Risk=Medium, Confidence=Medium (3)

1. http://testphp.vulnweb.com (3)

1. .htaccess Information Leak (1)

1.

▶ GET http://testphp.vulnweb.com/Mod_Rewrite_Shop/.htaccess

gs
- OWASP_2021_A05
- WSTG-v42-CONF-05
- OWASP_2017_A06

ption htaccess files can be used to alter the configuration of the Apache Web Server software to enable/disable additional functionality and features that the Apache Web Server software has to offer.

t ▼
Request line and header section (271 bytes)

```
GET http://testphp.vulnweb.com/Mod_Rewrite_Shop/.htaccess HTTP/1.1
Host: testphp.vulnweb.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0)
Gecko/20100101 Firefox/105.0
Pragma: no-cache
Cache-Control: no-cache
Referer: http://testphp.vulnweb.com
```

▼
Request body (0 bytes)

ie ▼
Status line and header section (252 bytes)

```
HTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Tue, 14 Feb 2023 18:14:05 GMT
Content-Type: application/octet-stream
Content-Length: 176
Last-Modified: Wed, 15 Feb 2012 10:32:40 GMT
Connection: keep-alive
ETag: "4f3b89c8-b0"
Accept-Ranges: bytes
```

▼
Response body (176 bytes)

```
RewriteEngine on
RewriteRule Details/.*/(.*?)/ details.php?id=$1 [L]
RewriteRule BuyProduct-(.*?)/ buy.php?id=$1 [L]
RewriteRule RateProduct-(.*?)\.html rate.php?id=$1 [L]
```

e `HTTP/1.1 200 OK`

1 Ensure the .htaccess file is not accessible.

2. Missing Anti-clickjacking Header (1)

1.

▶ GET http://testphp.vulnweb.com/

gs
- OWASP_2021_A05
- WSTG-v42-CLNT-09
- OWASP_2017_A06

ption The response does not include either Content-Security-Policy with 'frame-ancestors' directive or X-Frame-Options to protect against 'ClickJacking' attacks.

t ▼
Request line and header section (208 bytes)

```
GET http://testphp.vulnweb.com/ HTTP/1.1
Host: testphp.vulnweb.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0)
Gecko/20100101 Firefox/105.0
Pragma: no-cache
Cache-Control: no-cache
```

▼
Request body (0 bytes)

ie ▼
Status line and header section (221 bytes)

```
HTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Tue, 14 Feb 2023 16:29:05 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
X-Powered-By: PHP/5.6.40-38+ubuntu20.04.1+deb.sury.org+1
Content-Length: 170
```

▼
Response body (170 bytes)

```
Warning: mysql_connect(): Connection refused in /hj/var/www/database_
connect.php on line 2
Website is out of order. Please visit back later. Thank you for
understanding.
```

er `X-Frame-Options`

n 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.

3. XSLT Injection (1)

1.

▶ GET http://testphp.vulnweb.com/showimage.php?file=%3Cxsl%3Avalue-of+select%3D%22document%28%27http%3A%2F%2Ftestphp.vulnweb.com%3A22%27%29%22%2F%3E

gs
- OWASP_2021_A03
- OWASP_2017_A01

iption Injection using XSL transformations may be possible, and may allow an attacker to read system information, read and write files, or execute arbitrary code.

nfo Port scanning may be possible.

st ▼

Request line and header section (383 bytes)

```
GET http://testphp.vulnweb.com/showimage.php?file=%3Cxsl%3Avalue-of+select%3D%22document%28%27htt
29%22%2F%3E HTTP/1.1
Host: testphp.vulnweb.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0) Gecko/
20100101 Firefox/105.0
Pragma: no-cache
Cache-Control: no-cache
Referer: http://testphp.vulnweb.com/listproducts.php?cat=2
```

▼

Request body (0 bytes)

ise ▼

Status line and header section (207 bytes)

```
HTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Tue, 14 Feb 2023 18:15:29 GMT
Content-Type: image/jpeg
Connection: keep-alive
X-Powered-By: PHP/5.6.40-38+ubuntu20.04.1+deb.sury.org+1
Content-Length: 286
```

▼

Response body (286 bytes)

```
Warning: fopen(<xsl:value-of select="document('http://testphp.vulnweb.
com:22')"/>): failed to open stream: No such file or directory in /hj/
var/www/showimage.php on line 13

Warning: fpassthru() expects parameter 1 to be resource, boolean given
in /hj/var/www/showimage.php on line 19
```

ter `file`

k `<xsl:value-of select="document('http://testphp.vulnweb.com:22')"/>`

ce `failed to open stream`

on Sanitize and analyze every user input coming from any client-side.

5. Risk=Medium, Confidence=Low (1)

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

1. Absence of Anti-CSRF Tokens (1)

1.

▶ GET http://testphp.vulnweb.com

igs
- OWASP_2021_A01
- WSTG-v42-SESS-05
- OWASP_2017_A05

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

A cross-site request forgery is an attack that involves forcing a victim to send an HTTP request to a target destination without their knowledge or intent in order to perform an action as the victim. The underlying cause is application functionality using predictable URL/form actions in a repeatable way. The nature of the attack is that CSRF exploits the trust that a web site has for a user. By contrast, cross-site scripting (XSS) exploits the trust that a user has for a web site. Like XSS, CSRF attacks are not necessarily cross-site, but they can be. Cross-s

CSRF attacks are effective in a number of situations, including:

* The victim has an active session on the target site.

* The victim is authenticated via HTTP auth on the target site.

* The victim is on the same local network as the target site.

CSRF has primarily been used to perform an action against a target site using the victim's privileges, but recent techniques have been discovered to disclose information by gaining access to the response. The risk of information disclosure is dramatically increased when the target site is vulnerable to XSS, because XSS can be used as a platform for CSRF, allowing the attack to operate within the bounds of the same-origin policy.

nfo No known Anti-CSRF token [anticsrf, CSRFToken, __RequestVerificationToken, csrfmiddlewaretoken, authenticity_token, OWASP_CSRFTOKEN, anoncsrf, csrf_token, _csrf, _csrfSecret, __csrf_magic, CSRF, _token, _csrf_token] was found in the following HTML form: [Form 1: "goButton" "searchFor"].

st ▼
Request line and header section (207 bytes)

```
GET http://testphp.vulnweb.com HTTP/1.1
Host: testphp.vulnweb.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0) Gecko/
20100101 Firefox/105.0
Pragma: no-cache
Cache-Control: no-cache
```

▼
Request body (0 bytes)

ise ▼
Status line and header section (222 bytes)

```
HTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Tue, 14 Feb 2023 16:29:06 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
X-Powered-By: PHP/5.6.40-38+ubuntu20.04.1+deb.sury.org+1
Content-Length: 4958
```

▶
Response body (4958 bytes)

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

m 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.

For example, use anti-CSRF packages such as the OWASP CSRFGuard.

Phase: Implementation

Ensure that your application is free of cross-site scripting issues, because most CSRF defenses can be bypassed using attacker-controlled script.

Phase: Architecture and Design

Generate a unique nonce for each form, place the nonce into the form, and verify the nonce upon receipt of the form. Be sure that the nonce is not predictable (CWE-330).

Note that this can be bypassed using XSS.

Identify especially dangerous operations. When the user performs a dangerous operation, send a separate confirmation request to ensure that the user intended to perform that operation.

Note that this can be bypassed using XSS.

Use the ESAPI Session Management control.

This control includes a component for CSRF.

Do not use the GET method for any request that triggers a state change.

Phase: Implementation

Check the HTTP Referer header to see if the request originated from an expected page. This could break legitimate functionality, because users or proxies may have disabled sending the Referer for privacy reasons.

6. Risk=Low, Confidence=High (1)

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

1. Server Leaks Version Information via "Server" HTTP Response Header Field (1)

1.

▶ GET http://testphp.vulnweb.com/

ıgs
- OWASP_2021_A05
- OWASP_2017_A06
- WSTG-v42-INFO-02

iption The web/application server is leaking version information via the "Server" HTTP response header.

Access to such information may facilitate attackers identifying other vulnerabilities your web/application server is subject to.

st ▼

Request line and header section (208 bytes)

```
GET http://testphp.vulnweb.com/ HTTP/1.1
Host: testphp.vulnweb.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0)
Gecko/20100101 Firefox/105.0
Pragma: no-cache
Cache-Control: no-cache
```

▼

Request body (0 bytes)

ıse ▼

Status line and header section (221 bytes)

```
HTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Tue, 14 Feb 2023 16:29:05 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
X-Powered-By: PHP/5.6.40-38+ubuntu20.04.1+deb.sury.org+1
Content-Length: 170
```

▼

Response body (170 bytes)

```
Warning: mysql_connect(): Connection refused in /hj/var/www/database_
connect.php on line 2
Website is out of order. Please visit back later. Thank you for
understanding.
```

ce `nginx/1.19.0`

ın Ensure that your web server, application server, load balancer, etc. is configured to suppress the "Server" header or provide generic details.

7. Risk=Low, Confidence=Medium (2)

1. http://testphp.vulnweb.com (2)

1. Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s) (1)

1.

▶ GET http://testphp.vulnweb.com/

ıgs
- OWASP_2021_A01
- WSTG-v42-INFO-08
- OWASP_2017_A03

ription The web/application server is leaking information via one or more "X-Powered-By" HTTP response headers. Access to such information may facilitate attackers identifying other frameworks/components your web application is reliant upon and the vulnerabilities such components may be subject to.

st ▼

Request line and header section (208 bytes)

```
GET http://testphp.vulnweb.com/ HTTP/1.1
Host: testphp.vulnweb.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0)
Gecko/20100101 Firefox/105.0
Pragma: no-cache
Cache-Control: no-cache
```

▼

Request body (0 bytes)

ıse ▼

Status line and header section (221 bytes)

```
HTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Tue, 14 Feb 2023 16:29:05 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
X-Powered-By: PHP/5.6.40-38+ubuntu20.04.1+deb.sury.org+1
Content-Length: 170
```

▼

Response body (170 bytes)

```
Warning: mysql_connect(): Connection refused in /hj/var/www/
database_connect.php on line 2
Website is out of order. Please visit back later. Thank you for
understanding.
```

ce `X-Powered-By: PHP/5.6.40-38+ubuntu20.04.1+deb.sury.org+1`

ın Ensure that your web server, application server, load balancer, etc. is configured to suppress "X-Powered-By" headers.

2. X-Content-Type-Options Header Missing (1)

1.

▶ GET http://testphp.vulnweb.com/

igs
- OWASP_2021_A05
- OWASP_2017_A06

iption The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing.

nfo This issue still applies to error type pages (401, 403, 500, etc.) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type.

At "High" threshold this scan rule will not alert on client or server error responses.

st ▼
Request line and header section (208 bytes)

```
GET http://testphp.vulnweb.com/ HTTP/1.1
Host: testphp.vulnweb.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0)
Gecko/20100101 Firefox/105.0
Pragma: no-cache
Cache-Control: no-cache
```

▼
Request body (0 bytes)

ise ▼
Status line and header section (221 bytes)

```
HTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Tue, 14 Feb 2023 16:29:05 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
X-Powered-By: PHP/5.6.40-38+ubuntu20.04.1+deb.sury.org+1
Content-Length: 170
```

▼
Response body (170 bytes)

```
Warning: mysql_connect(): Connection refused in /hj/var/www/database_
connect.php on line 2
Website is out of order. Please visit back later. Thank you for
understanding.
```

ter `X-Content-Type-Options`

in Ensure that the application/web server sets the Content-Type header appropriately, and that it sets the X-Content-Type-Options header to 'nosniff' for all web pages.

If possible, ensure that the end user uses a standards-compliant and modern web browser that does not perform MIME-sniffing at all, or that can be directed by the web application/web server to not perform MIME-sniffing.

8. Risk=`Informational`, Confidence=`High` (1)

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

1. GET for POST (1)

1.

▶ GET http://testphp.vulnweb.com/cart.php

gs
- OWASP_2021_A04
- WSTG-v42-CONF-06
- OWASP_2017_A06

ption A request that was originally observed as a POST was also accepted as a GET. This issue does not represent a security weakness unto itself, however, it may facilitate simplification of other attacks. For example if the original POST is subject to Cross-Site Scripting (XSS), then this finding may indicate that a simplified (GET based) XSS may also be possible.

t ▼
Request line and header section (342 bytes)

```
GET http://testphp.vulnweb.com/cart.php?addcart=6&price=10000 HTTP/1.1
Host: testphp.vulnweb.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0)
Gecko/20100101 Firefox/105.0
Pragma: no-cache
Cache-Control: no-cache
Content-Type: application/x-www-form-urlencoded
Referer: http://testphp.vulnweb.com/product.php?pic=6
```

▼
Request body (0 bytes)

se ▼
Status line and header section (222 bytes)

```
HTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Tue, 14 Feb 2023 18:15:37 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
X-Powered-By: PHP/5.6.40-38+ubuntu20.04.1+deb.sury.org+1
Content-Length: 4903
```

▶
Response body (4903 bytes)

```
GET http://testphp.vulnweb.com/cart.php?addcart=6&price=10000 HTTP/1.1
```

Ensure that only POST is accepted where POST is expected.

9. **Risk=`Informational`, Confidence=`Medium` (2)**

1. **`http://testphp.vulnweb.com` (2)**

   1. **Modern Web Application (1)**

      1\.

      ▶ GET http://testphp.vulnweb.com/artists.php

gs

ption The application appears to be a modern web application. If you need to explore it automatically then the Ajax Spider may well be more effective than the standard one.

fo Links have been found that do not have traditional href attributes, which is an indication that this is a modern web application.

t ▼
Request line and header section (256 bytes)

```
GET http://testphp.vulnweb.com/artists.php HTTP/1.1
Host: testphp.vulnweb.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0)
Gecko/20100101 Firefox/105.0
Pragma: no-cache
Cache-Control: no-cache
Referer: http://testphp.vulnweb.com
```

▼
Request body (0 bytes)

e ▼
Status line and header section (222 bytes)

```
HTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Tue, 14 Feb 2023 16:29:08 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
X-Powered-By: PHP/5.6.40-38+ubuntu20.04.1+deb.sury.org+1
Content-Length: 5328
```

▶
Response body (5328 bytes)

e
```
<a href='#' onClick="window.open('./comment.php?aid=1','comment',
'width=500,height=400')">comment on this artist</a>
```

n This is an informational alert and so no changes are required.

2. User Agent Fuzzer (1)

1.

▶ POST http://testphp.vulnweb.com/guestbook.php

Check for differences in response based on fuzzed User Agent (eg. mobile sites, access as a Search Engine Crawler). Compares the response statuscode and the hashcode of the response body with the original response.

▼
Request line and header section (312 bytes)

```
POST http://testphp.vulnweb.com/guestbook.php HTTP/1.1
Host: testphp.vulnweb.com
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1)
Pragma: no-cache
Cache-Control: no-cache
Content-Type: application/x-www-form-urlencoded
Referer: http://testphp.vulnweb.com/guestbook.php
Content-Length: 33
```

▼
Request body (33 bytes)

```
name=ZAP&text=&submit=add+message
```

▼
Status line and header section (222 bytes)

```
HTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Tue, 14 Feb 2023 18:15:46 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
X-Powered-By: PHP/5.6.40-38+ubuntu20.04.1+deb.sury.org+1
Content-Length: 5393
```

▶
Response body (5393 bytes)

```
Header User-Agent
Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1)
```

10. **Risk=Informational, Confidence=Low (3)**

1. **http://testphp.vulnweb.com (3)**

1. **Charset Mismatch (Header Versus Meta Content-Type Charset) (1)**

1.

▶ GET http://testphp.vulnweb.com

gs

ption This check identifies responses where the HTTP Content-Type header declares a charset different from the charset defined by the body of the HTML or XML. When there's a charset mismatch between the HTTP header and content body Web browsers can be forced into an undesirable content-sniffing mode to determine the content's correct character set.

An attacker could manipulate content on the page to be interpreted in an encoding of their choice. For example, if an attacker can control content at the beginning of the page, they could inject script using UTF-7 encoded text and manipulate some browsers into interpreting that text.

fo There was a charset mismatch between the HTTP Header and the META content-type encoding declarations: [UTF-8] and [iso-8859-2] do not match.

t ▼
Request line and header section (207 bytes)

```
GET http://testphp.vulnweb.com HTTP/1.1
Host: testphp.vulnweb.com

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0)

Gecko/20100101 Firefox/105.0
Pragma: no-cache
Cache-Control: no-cache
```

▼
Request body (0 bytes)

re ▼
Status line and header section (222 bytes)

```
HTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Tue, 14 Feb 2023 16:29:06 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
X-Powered-By: PHP/5.6.40-38+ubuntu20.04.1+deb.sury.org+1
Content-Length: 4958
```

▶
Response body (4958 bytes)

- Force UTF-8 for all text content in both the HTTP header and meta tags in HTML or encoding declarations in XML.

2. Information Disclosure - Suspicious Comments (1)

1.

▶ GET http://testphp.vulnweb.com/AJAX/index.php

gs
- OWASP_2021_A01
- OWASP_2017_A03

ption The response appears to contain suspicious comments which may help an attacker. Note: Matches made within script blocks or files are against the entire content not only comments.

fo The following pattern was used: \bWHERE\b and was detected in the element starting with: "

<script type="text/javascript">

var httpreq = null;

function SetContent(XML) {

var items = XML.getElementsByTagName('i", see evidence field for the suspicious comment/snippet.

t ▼
Request line and header section (259 bytes)

```
GET http://testphp.vulnweb.com/AJAX/index.php HTTP/1.1
Host: testphp.vulnweb.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0)
Gecko/20100101 Firefox/105.0
Pragma: no-cache
Cache-Control: no-cache
Referer: http://testphp.vulnweb.com
```

▼
Request body (0 bytes)

ce ▼
Status line and header section (222 bytes)

```
HTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Tue, 14 Feb 2023 16:29:09 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
X-Powered-By: PHP/5.6.40-38+ubuntu20.04.1+deb.sury.org+1
Content-Length: 4236
```

▶
Response body (4236 bytes)

e where

n Remove all comments that return information that may help an attacker and fix any underlying problems they refer to.

3. User Controllable HTML Element Attribute (Potential XSS) (1)

1.

▶ POST http://testphp.vulnweb.com/search.php?test=query

gs
- OWASP_2021_A03
- OWASP_2017_A01

ption This check looks at user-supplied input in query string parameters and POST data to identify where certain HTML attribute values might be controlled. This provides hot-spot detection for XSS (cross-site scripting) that will require further review by a security analyst to determine exploitability.

fo User-controlled HTML attribute values were found. Try injecting special characters to see if XSS might be possible. The page at the following URL:

http://testphp.vulnweb.com/search.php?test=query

appears to include user input in:

a(n) [input] tag [name] attribute

The user input found was:

goButton=go

The user-controlled value was:

gobutton

t ▼
Request line and header section (336 bytes)

```
POST http://testphp.vulnweb.com/search.php?test=query HTTP/1.1
Host: testphp.vulnweb.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0) Gecko
/20100101 Firefox/105.0
Pragma: no-cache
Cache-Control: no-cache
Content-Type: application/x-www-form-urlencoded
Referer: http://testphp.vulnweb.com
Content-Length: 25
```

▼
Request body (25 bytes)

```
searchFor=ZAP&goButton=go
```

ie ▼
Status line and header section (222 bytes)

```
HTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Tue, 14 Feb 2023 16:29:16 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
X-Powered-By: PHP/5.6.40-38+ubuntu20.04.1+deb.sury.org+1
Content-Length: 4772
```

▶
Response body (4772 bytes)

er	goButton
1	Validate all input and sanitize output it before writing to any HTML attributes.

Appendix

Alert types

This section contains additional information on the types of alerts in the report.

1. Cross Site Scripting (Reflected)

Source	raised by an active scanner (Cross Site Scripting (Reflected))
CWE ID	79
WASC ID	8
Reference	▪ http://projects.webappsec.org/Cross-Site-Scripting ▪ http://cwe.mitre.org/data/definitions/79.html

2. Path Traversal

Source	raised by an active scanner (Path Traversal)
CWE ID	22
WASC ID	33
Reference	▪ http://projects.webappsec.org/Path-Traversal ▪ http://cwe.mitre.org/data/definitions/22.html

3. SQL Injection

Source	raised by an active scanner (SQL Injection)
CWE ID	89
WASC ID	19
Reference	▪ https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

4. SQL Injection - MySQL

Source	raised by an active scanner (SQL Injection)
CWE ID	89
WASC ID	19
Reference	▪ https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

5. .htaccess Information Leak

Source	raised by an active scanner (.htaccess Information Leak)
CWE ID	94
WASC ID	14
Reference	▪ http://www.htaccess-guide.com/

6. Absence of Anti-CSRF Tokens

Source	raised by a passive scanner (Absence of Anti-CSRF Tokens)
CWE ID	352
WASC ID	9
Reference	▪ http://projects.webappsec.org/Cross-Site-Request-Forgery ▪ http://cwe.mitre.org/data/definitions/352.html

7. Content Security Policy (CSP) Header Not Set

Source	raised by a passive scanner (Content Security Policy (CSP) Header Not Set)
CWE ID	693
WASC ID	15
Reference	▪ https://developer.mozilla.org/en-US/docs/Web/Security/CSP/Introducing_Content_Security_Policy ▪ https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html ▪ http://www.w3.org/TR/CSP/ ▪ http://w3c.github.io/webappsec/specs/content-security-policy/csp-specification.dev.html ▪ http://www.html5rocks.com/en/tutorials/security/content-security-policy/ ▪ http://caniuse.com/#feat=contentsecuritypolicy ▪ http://content-security-policy.com/

8. Missing Anti-clickjacking Header

Source	raised by a passive scanner (Anti-clickjacking Header)
CWE ID	1021
WASC ID	15
Reference	▪ https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options

9. XSLT Injection

Source	raised by an active scanner (XSLT Injection)
CWE ID	91
WASC ID	23
Reference	▪ https://www.contextis.com/blog/xslt-server-side-injection-attacks

10. Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)

Source	raised by a passive scanner (Server Leaks Information via "X-Powered-By" HTTP Response Header Fie
CWE ID	200
WASC ID	13
Reference	▪ http://blogs.msdn.com/b/varunm/archive/2013/04/23/remove-unwanted-http-response-headers.aspx ▪ http://www.troyhunt.com/2012/02/shhh-dont-let-your-response-headers.html

11. Server Leaks Version Information via "Server" HTTP Response Header Field

Source	raised by a passive scanner (HTTP Server Response Header)
CWE ID	200
WASC ID	13
Reference	▪ http://httpd.apache.org/docs/current/mod/core.html#servertokens ▪ http://msdn.microsoft.com/en-us/library/ff648552.aspx#ht_urlscan_007 ▪ http://blogs.msdn.com/b/varunm/archive/2013/04/23/remove-unwanted-http-response-headers.aspx ▪ http://www.troyhunt.com/2012/02/shhh-dont-let-your-response-headers.html

12. X-Content-Type-Options Header Missing

Source	raised by a passive scanner (X-Content-Type-Options Header Missing)
CWE ID	693
WASC ID	15
Reference	▪ http://msdn.microsoft.com/en-us/library/ie/gg622941%28v=vs.85%29.aspx ▪ https://owasp.org/www-community/Security_Headers

13. Charset Mismatch (Header Versus Meta Content-Type Charset)

Source	raised by a passive scanner (Charset Mismatch)
CWE ID	436
WASC ID	15
Reference	▪ http://code.google.com/p/browsersec/wiki/Part2#Character_set_handling_and_detection

14. GET for POST

Source	raised by an active scanner (GET for POST)
CWE ID	16
WASC ID	20

15. Information Disclosure - Suspicious Comments

Source	raised by a passive scanner (Information Disclosure - Suspicious Comments)
CWE ID	200
WASC ID	13

16. Modern Web Application

Source	raised by a passive scanner (Modern Web Application)

17. User Agent Fuzzer

Source	raised by an active scanner (User Agent Fuzzer)
Reference	▪ https://owasp.org/wstg

18. User Controllable HTML Element Attribute (Potential XSS)

Source	raised by a passive scanner (User Controllable HTML Element Attribute (Potential XSS))
CWE ID	20
WASC ID	20
Reference	▪ http://websecuritytool.codeplex.com/wikipage?title=Checks#user-controlled-html-attribute

CONCUSION:

High (= High)	Medium (>= Medium)	Low (>= Low)	Informational (>= Informational)
4 (4)	5 (9)	3 (12)	6 (18)