A

Project report

On Scanning using OWASP ZAP

Generated with ZAP on Tue 5 Apr 2022, at 22:28:23

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

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.)

	User confirmed	High	Medium	Low	Total
High	0 (0.0%)	0 (0.0%)	1 (12.5%)	0 (0.0%)	1 (12.5%)
Medium	0 (0.0%)	0 (0.0%)	2 (25.0%)	0 (0.0%)	2 (25.0%)
Low	0 (0.0%)	0 (0.0%)	3 (37.5%)	0 (0.0%)	3 (37.5%)
Informational	0 (0.0%)	0 (0.0%)	0 (0.0%)	2 (25.0%)	2 (25.0%)
Total	0 (0.0%)	0 (0.0%)	6 (75.0%)	2 (25.0%)	8 (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	1 (1)	2 (3)	3 (6)	2 (8)

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	16 (200.0%)
.htaccess Information Leak	Medium	7 (87.5%)
X-Frame-Options Header Not Set	Medium	44 (550.0%)
Absence of Anti-CSRF Tokens	Low	37 (462.5%)
Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)	Low	62 (775.0%)
X-Content-Type-Options Header Missing	Low	66 (825.0%)
Charset Mismatch (Header Versus Meta Content-Type Charset)	Informational	28 (350.0%)
Information Disclosure - Suspicious Comments	Informational	1 (12.5%)
Total		8

Alerts

1. **Risk=High, Confidence=Medium (1)**
    1. **http://testphp.vulnweb.com (1)**
        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 (340 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; rv:92.0) Gecko/20100101 Firefox/92.0 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 (200 bytes) HTTP/1.1 200 OK Server: nginx/1.19.0 Date: Tue, 05 Apr 2022 16:49:31 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 Response body (5433 bytes)
Parameter	name
Attack	<scrIpt>alert(1);</scRipt>
Evidence	<scrIpt>alert(1);</scRipt>
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. **Risk=Medium, Confidence=Medium (2)**
    1. **http://testphp.vulnweb.com (2)**
        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	Request line and header section (288 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:92.0) Gecko/20100101 Firefox/92.0 Pragma: no-cache Cache-Control: no-cache Referer: http://testphp.vulnweb.com Content-Length: 0 Request body (0 bytes)
Response	Status line and header section (252 bytes) HTTP/1.1 200 OK Server: nginx/1.19.0

Date: Tue, 05 Apr 2022 16:53:55 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. X-Frame-Options Header Not Set (1)
    1. GET http://testphp.vulnweb.com

Alert tags

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

Alert description X-Frame-Options header is not included in the HTTP response to protect against 'ClickJacking' attacks.

Request

Request line and header section (205 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:92.0) Gecko/20100101 Firefox/92.0
Pragma: no-cache
Cache-Control: no-cache

Request body (0 bytes)

Response

Status line and header section (200 bytes)
HTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Tue, 05 Apr 2022 16:46:45 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

Response body (4958 bytes)

Parameter X-Frame-Options

Solution

Most modern Web browsers support the X-Frame-Options HTTP header. Ensure it's set on all web pages returned by your site (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. **Risk=Low, Confidence=Medium (3)**
    1. **http://testphp.vulnweb.com (3)**
        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 (205 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:92.0) Gecko/20100101 Firefox/92.0 Pragma: no-cache Cache-Control: no-cache Request body (0 bytes)
Response	Status line and header section (200 bytes) HTTP/1.1 200 OK Server: nginx/1.19.0 Date: Tue, 05 Apr 2022 16:46:45 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 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.

2. 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 (205 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:92.0) Gecko/20100101 Firefox/92.0

Pragma: no-cache

Cache-Control: no-cache

Request body (0 bytes)

Response

Status line and header section (200 bytes)

HTTP/1.1 200 OK

Server: nginx/1.19.0

Date: Tue, 05 Apr 2022 16:46:45 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

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.

3. 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

Other info	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. 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 (205 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:92.0) Gecko/20100101 Firefox/92.0 Pragma: no-cache Cache-Control: no-cache Request body (0 bytes)
Response	Status line and header section (200 bytes) HTTP/1.1 200 OK Server: nginx/1.19.0 Date: Tue, 05 Apr 2022 16:46:45 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 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.

4. **Risk=Informational, Confidence=Low (2)**
    1. **http://testphp.vulnweb.com (2)**
        1. Charset Mismatch (Header Versus Meta Content-Type Charset) (1)
            1. 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

Request line and header section (205 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:92.0) Gecko/20100101 Firefox/92.0

Pragma: no-cache

Cache-Control: no-cache

Request body (0 bytes)

Response

Status line and header section (200 bytes)

HTTP/1.1 200 OK

Server: nginx/1.19.0

	Date: Tue, 05 Apr 2022 16:46:45 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
	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.

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

Alert tags	• OWASP_2021_A01 • 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.
Other info	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.
Request	Request line and header section (257 bytes) Request body (0 bytes)
Response	Status line and header section (200 bytes) 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.

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

2. **.htaccess Information Leak**

Source	raised by an active scanner (.htaccess Information Leak)
CWE ID	94
WASC ID	14
Reference	0. http://www.htaccess-guide.com/

3. **X-Frame-Options Header Not Set**

Source	raised by a passive scanner (Anti-clickjacking Header)
CWE ID	1021
WASC ID	15
Reference	0. https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options

4. **Absence of Anti-CSRF Tokens**

Source	raised by a passive scanner (Absence of Anti-CSRF Tokens)
CWE ID	352

WASC ID 9

Reference
0. http://projects.webappsec.org/Cross-Site-Request-Forgery
1. http://cwe.mitre.org/data/definitions/352.html

5. **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
0. http://blogs.msdn.com/b/varunm/archive/2013/04/23/remove-unwanted-http-response-headers.aspx
1. http://www.troyhunt.com/2012/02/shhh-dont-let-your-response-headers.html

6. **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
0. http://msdn.microsoft.com/en-us/library/ie/gg622941%28v=vs.85%29.aspx
1. https://owasp.org/www-community/Security_Headers

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

Source raised by a passive scanner (Charset Mismatch)

CWE ID 436

WASC ID 15

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

8. **Information Disclosure - Suspicious Comments**

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