Pentest-Tools.com

XSS Scanner Report (Light)

✓ **http://testphp.vulnweb.com/**

Summary

Overall risk level:

High

Risk ratings:

High:	1
Medium:	0
Low:	0
Info:	1

Scan information:

Start time:	2020-10-11 06:36:37 UTC+03
Finish time:	2020-10-11 06:37:08 UTC+03
Scan duration:	31 sec
Tests performed:	2/2
Scan status:	Finished

Findings

🚩 Cross-Site Scripting

Vulnerable Page	Vulnerable Parameter	Method	Attack Vector	
/guestbook. php	name	POST	http://testphp.vulnweb.com/guestbook.php **POST Data:** name=<script>alert(1);</script><strong>	🚀
/guestbook. php	text	POST	http://testphp.vulnweb.com/guestbook.php **POST Data:** text=</td><script>alert(1);</script><td>	🚀
/listproducts .php	cat	GET	http://testphp.vulnweb.com/listproducts.php?cat=%3C%2Fdiv%3E%3Cscript%3Ealert%281%29%3B%3C%2Fscript%3E%3Cdiv%3E	🚀
/showimage .php	file	GET	http://testphp.vulnweb.com/showimage.php?file=%3Cscript%3Ealert%281%29%3B%3C%2Fscript%3E	🚀
/search.php	searchFor	POST	http://testphp.vulnweb.com/search.php?test=query **POST Data:** searchFor= <script>alert(1);</script>	🚀

⌄ Details

Risk 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.

Recommendation:
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 a whitelist 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 blacklist). However, blacklists 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.

⚑ Light spider results: 11 dynamic URLs of total 25 URLs crawled

METHOD	URL	PARAMS
GET	/listproducts.php	cat=2

GET	/listproducts.php	cat=3
GET	/listproducts.php	cat=4
POST	/userinfo.php	uname=ZAP&pass=ZAP
POST	/search.php?test=query	searchFor=ZAP&goButton=go
GET	/showimage.php	file='%20+%20pict.item(0).firstChild.nodeValue%20+%20'
POST	/guestbook.php	name=anonymous+user&text=&submit=add+message
GET	/artists.php	artist=3
GET	/artists.php	artist=2
GET	/listproducts.php	cat=1
GET	/artists.php	artist=1

Scan coverage information

List of tests performed (2/2)

- ✔ Spidering target
- ✔ Scanning for XSS vulnerabilities

Scan parameters

Website URL: http://testphp.vulnweb.com/
Scan type: Light
Authentication: False