Cross-site Scripting (XSS)

Affecting org.webjars.npm:materialize-css package, versions **[0,]**

INTRODUCED: 23 DEC 2021 CVE-2022-25349
CWE-79

Share

How to fix?

There is no fixed version for `org.webjars.npm:materialize-css`.

Overview

org.webjars.npm:materialize-css is a CSS Framework based on Material Design.

Affected versions of this package are vulnerable to Cross-site Scripting (XSS) due to improper escape of user input (such as `<not-a-tag />`) that is being parsed as HTML/JavaScript, and inserted into the Document Object Model (DOM). This vulnerability can be exploited when the user-input is provided to the autocomplete component.

PoC

```
$('input.autocomplete').autocomplete({data: {
"Apple": null, "Orange": null, "XSS <img src=x
onerror=alert('bad!')>": null, }});
```

5.4
MEDIUM

Snyk CVSS

Attack Complexity **Low**

User Interaction **Required**

See more

NVD 5.4 MEDIUM

Do your applications use this vulnerable package?

In a few clicks we can analyze your entire application and see what components are vulnerable in your application, and suggest you quick fixes.

Test your applications

Details

A cross-site scripting attack occurs when the attacker tricks a legitimate web-based application or site to accept a request as originating from a trusted source.

This is done by escaping the context of the web application; the web application then delivers that data to its users along with other trusted dynamic content, without validating it. The browser unknowingly executes malicious script on the client side (through client-side languages; usually JavaScript or HTML) in order to perform actions that are otherwise typically blocked by the browser's Same Origin Policy.

Injecting malicious code is the most prevalent manner by which XSS is exploited; for this reason, escaping characters in order to prevent this manipulation is the top method for securing code against this vulnerability.

Escaping means that the application is coded to mark key characters, and particularly key characters included in user input, to prevent those characters from being interpreted in a dangerous context. For example, in HTML, `<` can be coded as `<` and `>` can be coded as `>`; in order to be interpreted and displayed as themselves in text, while within the code itself, they are used for HTML tags. If malicious content is injected into an application that escapes special characters and that malicious content uses `<` and `>` as HTML tags, those characters are nonetheless not interpreted as HTML tags by the browser if they've been correctly escaped in the application code and in this way the attempted attack is diverted.

The most prominent use of XSS is to steal cookies (source: OWASP HttpOnly) and hijack user sessions, but XSS exploits have been used to expose sensitive information, enable access to privileged services and functionality and deliver malware.

Types of attacks

There are a few methods by which XSS can be manipulated:

Type	Origin	Description
Stored	Server	The malicious code is inserted in the application (usually as a link) by the attacker. The code is activated every time a user clicks the link.

Snyk ID **SNYK-JAVA-ORGWEBJARSNPM-2766498**

Published **18 Apr 2022**

Disclosed **23 Dec 2021**

Credit **Paul Wheeler**

Type	Origin	Description
Reflected	Server	The attacker delivers a malicious link externally from the vulnerable web site application to a user. When clicked, malicious code is sent to the vulnerable web site, which reflects the attack back to the user's browser.
DOM-based	Client	The attacker forces the user's browser to render a malicious page. The data in the page itself delivers the cross-site scripting data.
Mutated		The attacker injects code that appears safe, but is then rewritten and modified by the browser, while parsing the markup. An example is rebalancing unclosed quotation marks or even adding quotation marks to unquoted parameters.

Affected environments

The following environments are susceptible to an XSS attack:

- Web servers
- Application servers
- Web application environments

How to prevent

This section describes the top best practices designed to specifically protect your code:

- Sanitize data input in an HTTP request before reflecting it back, ensuring all data is validated, filtered or escaped before echoing anything back to the user, such as the values of query parameters during searches.
- Convert special characters such as `?`, `&`, `/`, `<`, `>` and spaces to their respective HTML or URL encoded equivalents.
- Give users the option to disable client-side scripts.
- Redirect invalid requests.
- Detect simultaneous logins, including those from two separate IP addresses, and invalidate those sessions.
- Use and enforce a Content Security Policy (source: Wikipedia) to disable any features that might be manipulated for an XSS attack.

- Read the documentation for any of the libraries referenced in your code to understand which elements allow for embedded HTML.

References

- Vulnerable Code

Events

FIND US ONLINE

TRACK OUR DEVELOPMENT

Join the >> community

Registered in England and Wales. Company number: 09677925

Registered address: Highlands House, Basingstoke Road, Spencers Wood, Reading, Berkshire, RG7 1NT.