ANVIL SECURE

2025-09-03

ENGAGEMENT REPORT

Security Advisories for jSuites Components

FOR: Jsuites

Anvil Secure
2125 Western Ave Suite 208
Seattle, WA 98121
United States of America
+1 206.753.7649
info@anvilsecure.com

ANVIL SECURE

Table of Contents

Summary

A number of Cross-Site Scripting (XSS) vulnerabilities were identified in different jSuites components. While not all of the occurrences are guaranteed to become exploitable (as this depends on how the components are integrated), at a minimum, it is recommended to explicitly warn about such risks in the public documentation, or name the fields that allow raw HTML in a way that they contain "HTML" in the name of these properties (although this would break their compatibility with previous versions).

Note that this was a best-effort manual review, only focusing on XSS vulnerabilities and not aiming a full coverage, so other vulnerabilities may exist. In addition, other potential occurrences were identified but were not reported because of the nature of the components or affected fields. For example, the jSuites JavaScript Picker explicitly states in the documentation that the `content` field is expected to contain HTML:

Property	Description
data: string[]	The picker options.
value: int	The position of the initial picker option.
content: string	The html or material-design icon that should be in front of the picker.
width: number	The picker width.
render: function	How each option should be shown. `function(string option) => string`

Figure 1: Public documentation of the jSuites JavaScript Picker component

In other components, like the jSuites JavaScript Toast, a developer might assume that the notification messages could contain HTML contents, although this was not explicitly stated in the public documentation or in the official examples.

Findings

JSA-01 - Reflected Cross-Site Scripting in jSuites HTML Editor through Drag and Drop

Severity	Medium
Category	Validation and Sanitization
Impact	High
Likelihood	Low
UniqueId	6697db6f-f5c7-4cd3-a50e-44b34ccdca69

Details:

The following function of the jSuites HTML editor is vulnerable to Cross-Site Scripting (XSS) when the `html` parameter is under an attacker's control:

jsuites/src/plugins/editor.js

```
var extractImageFromHtml = function(html) {
    // Create temp element
    var div = document.createElement('div');
    div.innerHTML = html;

    // Extract images
    var img = div.querySelectorAll('img');
    …
}
```

This function (`extractImageFromHtml`) is invoked from `editorDrop`, which is set as an event handler for the `drop` event:

jsuites/src/plugins/editor.js

```
 var editorDrop = function(e) {
     if (editorAction || obj.options.dropZone == false) {
         // Do nothing
     } else {
         ...
         var html = (e.originalEvent || e).dataTransfer.getData('text/html');
         var text = (e.originalEvent || e).dataTransfer.getData('text/plain');
         var file = (e.originalEvent || e).dataTransfer.files;

         if (file.length) {
             obj.addFile(file);
         } else if (text) {
             extractImageFromHtml(html);
         }

```

```
            el.classList.remove('jeditor-dragging');
            e.preventDefault();
        }
    }

...
obj.editor.addEventListener('drop', editorDrop);
...
```

To trigger the vulnerable function with potentially dangerous input, the dragged and dropped content must:

- Not contain any files: This ensures that the `file.length` condition evaluates to false. This can be easily achieved by dropping selected text instead of files.
- Contain a MIME type of `text/plain` with any contents: Just to make sure that the `text` variable is not undefined.
- Contain a MIME type of `text/html` with a valid payload: This will be placed in the `html` variable, which triggers the vulnerability.

Steps to Reproduce:

One of the simplest proof-of-concept code to meet all the above conditions would be:

Basic PoC with HTML and text

```
<html>
  <img src="does-not-exist.png" onerror="alert('XSS in ' + document.location)" />
  text
</html>
```

To exploit the vulnerability, the victim must be tricked into visiting a website hosting this HTML file, selecting the contents (which contain both HTML and plaintext MIME types), and then dragging and dropping them into the HTML editor.

A more elaborate version of the same proof-of-concept code is shown below. This version automatically selects the contents, minimizing the actions required by the victim. It also prevents them from deselecting the content and defuses the payload on the attacker-controlled HTML page by including a Content Security Policy (CSP):

Drag-PoC.html

```
<html>
    <head>
        <!-- Prevent the "onerror" handler from executing when rendering this PoC -->
        <meta http-equiv="Content-Security-Policy" content="default-src 'self'; script-src 'self' 'nonce-x';">
    </head>
    <body>
        <div id="selectMe">
            <img src="does-not-exist.png" onerror="javascript:alert('XSS in ' + document.location)" /><br />
            Make sure this text is also selected
        </div>

        <script nonce="x">
            // This is to automatically select the contents after the page loads
            function selectContent() {
                const range = document.createRange();
```

```
                const selection = window.getSelection();
                const node = document.getElementById("selectMe");

                range.selectNodeContents(node);
                selection.removeAllRanges();
                selection.addRange(range);
            }

            window.addEventListener('DOMContentLoaded', selectContent);

            // And to prevent the user from deselecting it, but allowing to drag and drop the
             ↪ contents
            document.addEventListener('selectionchange', () => {
                const selection = window.getSelection();
                if (selection && selection.isCollapsed) {
                    selectContent();
                }
            });
        </script>
    </body>
</html>
```

The relative `does-not-exist.png` image path must not exist in the victim's website. However, it can be hosted on the attacker's site to make the attack appear more convincing. For example, like the following:

Figure 2: Example of a `does-not-exist.png` image file.

The vulnerability could be exploited in any of the examples hosted on the official website (https://jsuites.net/docs/javascript-html-editor) by dragging and dropping the contents from another tab, web browser, or application that supports displaying HTML contents (e.g., email clients, rich text editors, etc.).

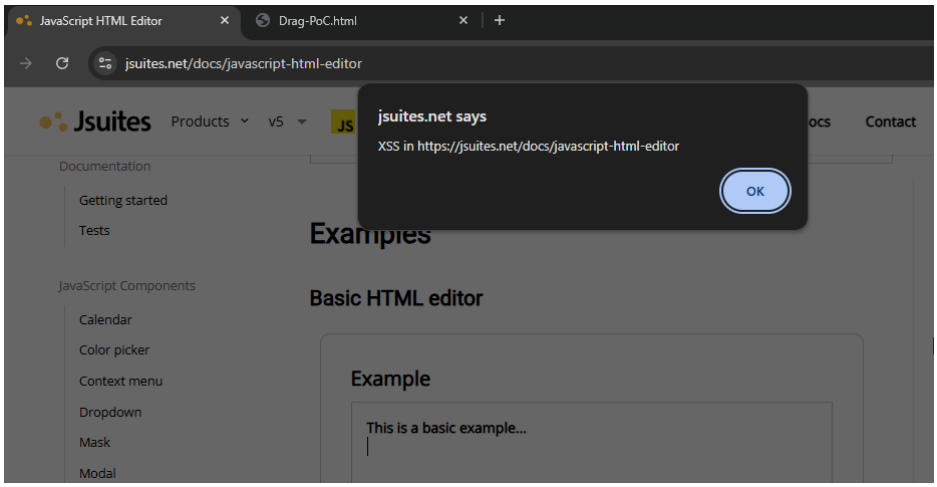


Figure 3: XSS triggered via jSuites HTML editor.

Impact:

A Cross-Site Scripting (XSS) vulnerability may allow arbitrary client-side code to be executed in the victim's web browser, leading to unintended actions on behalf of a logged-in user or even account takeover. The final severity of the impact depends on where the HTML editor component is used and the functionality supported by the website hosting the vulnerable jSuites component.

While the likelihood of exploitation is lower than more common XSS scenarios, which are easier to trigger, this vulnerability requires a more specific action, as the victim must drag and drop specially crafted content. However, depending on the context, social engineering tactics (e.g., "drag and drop this to get a free discount") could deceive victims, making this a potentially serious vulnerability.

Mitigation:

Creating temporary HTML elements and setting their `innerHTML` property is a well-known vulnerable pattern when the main `document` is used. The following alternatives can be used to mitigate this vulnerability.

- Use `DOMParser` to parse the HTML contents from the string.

Safe alternative using DOMParser

```
var extractImageFromHtml = function(html) {
    var parser = new DOMParser();
    var doc = parser.parseFromString(html, 'text/html');

    // Extract images
    var img = doc.querySelectorAll('img');
    ...
}
```

- Use `createHTMLDocument` to create an isolated document where scripts would not run.

Safe alternative using createHTMLDocument

```
var extractImageFromHtml = function(html) {
    var doc = document.implementation.createHTMLDocument('');

```

```
    var div = doc.createElement('div');
    div.innerHTML = html;

    // Extract images
    var img = div.querySelectorAll('img');
    ...
}
```

JSA-02 - Reflected Cross-Site Scripting in jSuites Image Cropper through Drag and Drop

Severity	Medium
Category	Validation and Sanitization
Impact	High
Likelihood	Low
UniqueId	7f891876-bd86-40b2-a506-c4e7794eeae2

Details:

Similar to JSA-01 - Reflected Cross-Site Scripting in jSuites HTML Editor through Drag and Drop, the jSuites Image Cropper component is also vulnerable to reflected Cross-Site Scripting (XSS) when dragging and dropping malicious contents, as it relies on the same vulnerable code:

jsuites/packages/cropper/cropper.js

```
el.addEventListener('drop', function(e) {
    e.preventDefault();
    e.stopPropagation();


    var html = (e.originalEvent || e).dataTransfer.getData('text/html');
    var file = (e.originalEvent || e).dataTransfer.files;

    if (file.length) {
        for (var i = 0; i < e.dataTransfer.files.length; i++) {
            obj.addFromFile(e.dataTransfer.files[i]);
        }
    } else if (html) {
        // Create temp element
        var div = document.createElement('div');
        div.innerHTML = html;

        // Extract images
        var img = div.querySelector('img');
        ...
```

Steps to Reproduce:

In this case, there is no need to have any plaintext selected, as it is enough with the `text/html` MIME type, but the same proof-of-concept code used in JSA-01 - Reflected Cross-Site Scripting in jSuites HTML Editor through Drag and Drop can also be used to exploit this vulnerability.

The vulnerability could be exploited in the example hosted on the official website (https://jsuites.net/docs/image-cropper) by dragging and dropping the contents from another tab, web browser,

or application that supports displaying HTML contents (e.g., email clients, rich text editors, etc.).

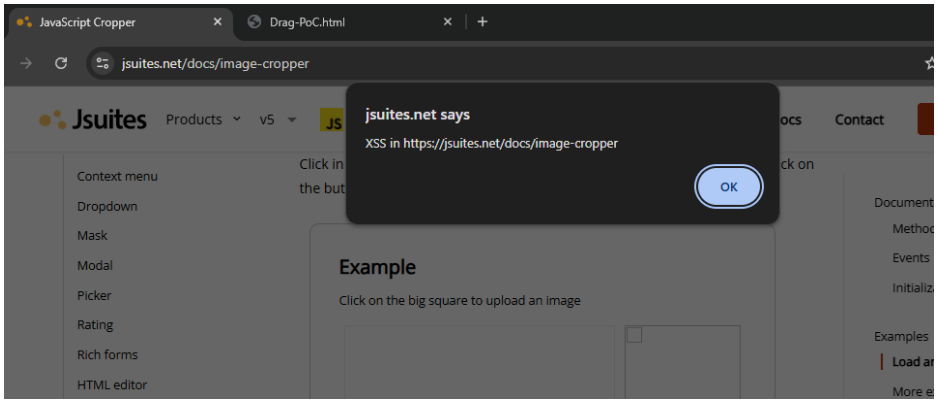


Figure 4: XSS triggered via jSuites Image Cropper.

Impact:

A Cross-Site Scripting (XSS) vulnerability may allow arbitrary client-side code to be executed in the victim's web browser, leading to unintended actions on behalf of a logged-in user or even account takeover. The final severity of the impact depends on where the HTML editor component is used and the functionality supported by the website hosting the vulnerable component.

While the likelihood of exploitation is lower than more common XSS scenarios, which are easier to trigger, this vulnerability requires a more specific action, as the victim must drag and drop specially crafted content. However, depending on the context, social engineering tactics (e.g., "drag and drop this to get a free discount") could deceive victims, making this a potentially serious vulnerability.

Mitigation:

The same mitigation explained in JSA-01 - Reflected Cross-Site Scripting in jSuites HTML Editor through Drag and Drop would help mitigate the vulnerability.

JSA-03 - Reflected Cross-Site Scripting in jSuites Upload Plugin through Drag and Drop

Severity	Medium
Category	Validation and Sanitization
Impact	High
Likelihood	Low
UniqueId	af47a7cf-8e81-4384-9fd8-fdb66c1a37c7

Details:

Similar to JSA-01 - Reflected Cross-Site Scripting in jSuites HTML Editor through Drag and Drop, the jSuites Upload plugin is also vulnerable to reflected Cross-Site Scripting (XSS) when dragging and dropping malicious contents, as it relies on the same vulnerable code:

jsuites/src/plugins/upload.js

```
el.addEventListener('drop', function(e) {
    e.preventDefault();
    e.stopPropagation();

    var html = (e.originalEvent || e).dataTransfer.getData('text/html');
    var file = (e.originalEvent || e).dataTransfer.files;

    if (file.length) {
        for (var i = 0; i < e.dataTransfer.files.length; i++) {
            obj.addFromFile(e.dataTransfer.files[i]);
        }
    } else if (html) {
        if (obj.options.multiple == false) {
            el.innerText = '';
        }

        // Create temp element
        var div = document.createElement('div');
        div.innerHTML = html;

        // Extract images
        var img = div.querySelectorAll('img');

        if (img.length) {
            for (var i = 0; i < img.length; i++) {
                obj.addFromUrl(img[i].src);
            }
        }
    }
```

Steps to Reproduce:

The same proof-of-concept code used in JSA-01 - Reflected Cross-Site Scripting in jSuites HTML Editor through Drag and Drop can also be used to exploit this vulnerability.

The vulnerability could be exploited in the example hosted on the official website of the jSpreadsheet component (https://bossanova.uk/jspreadsheet/docs/images), which relies on the vulnerable Upload plugin. By double-clicking on an empty cell of the "Image" column and dragging and dropping the contents from another tab, web browser, or application that supports displaying HTML contents (e.g., email clients, rich text editors, etc.).

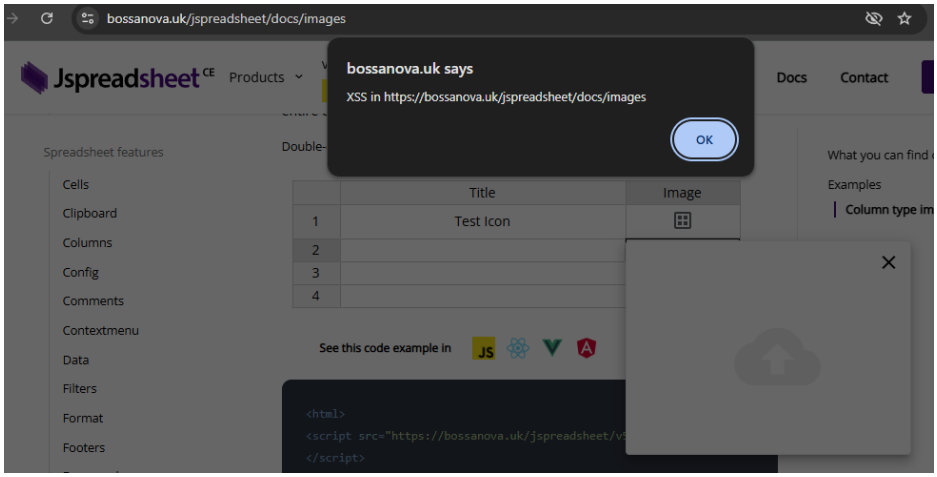


Figure 5: XSS triggered via jSuites Upload used in jSpreadsheet.

Impact:

A Cross-Site Scripting (XSS) vulnerability may allow arbitrary client-side code to be executed in the victim's web browser, leading to unintended actions on behalf of a logged-in user or even account takeover. The final severity of the impact depends on where the HTML editor component is used and the functionality supported by the website hosting the vulnerable component.

While the likelihood of exploitation is lower than more common XSS scenarios, which are easier to trigger, this vulnerability requires a more specific action, as the victim must drag and drop specially crafted content. However, depending on the context, social engineering tactics (e.g., "drag and drop this to get a free discount") could deceive victims, making this a potentially serious vulnerability.

Mitigation:

The same mitigation explained in JSA-01 - Reflected Cross-Site Scripting in jSuites HTML Editor through Drag and Drop would help mitigate the vulnerability.

JSA-06 - Potential Cross-Site Scripting in jSuites Tabs through Titles and Icons

Severity	Medium
Category	Validation and Sanitization
Impact	High
Likelihood	Low
UniqueId	cb3a06dd-2482-4f46-8017-5367891e1db9

Details:

The jSuites Tabs component is vulnerable to persistent Cross-Site Scripting (XSS) when a tab's title or custom icon contains a malicious payload. The public documentation does not have any reference explaining these properties, or explicitly warning the integrators about the dangers of allowing unsanitized input.

Data property

The data property define the content of the component, and have the following properties

Property	Description
title	Header title
width	Header width
icon	Header icon
content	Content

Figure 6: Public documentation of the jSuites Tabs component.

In addition, when the title is not programmatically set, the user will be prompted to enter an arbitrary value, as can be observed in the following code, making it impossible for the integrator developers to validate or sanitize, at least against self-XSS attacks:

jsuites/src/plugins/tabs.js

```
1  obj.appendElement = function(title, cb, openTab, position) {
2      if (! title) {
3          var title = prompt('Title?', '');
4      }
5
6      if (title) {
7          let headerId = Helpers.guid();
8          let contentId = Helpers.guid();
9          // Add content
10         var div = document.createElement('div');
11         div.setAttribute('id', contentId);
12         div.setAttribute('role', 'tabpanel');
```

```
        div.setAttribute('aria-labelledby', headerId);

        // Add headers
        var h = document.createElement('div');
        h.setAttribute('id', headerId);
        h.setAttribute('role', 'tab');
        h.setAttribute('aria-controls', contentId);

        h.innerHTML = title;
        h.content = div;
```

The custom icons are also vulnerable because of the use of the `innerHTML` property:

jsuites/src/plugins/tabs.js

```
// Icon
if (obj.options.data[i].icon) {
    var iconContainer = document.createElement('div');
    var icon = document.createElement('i');
    icon.classList.add('material-icons');
    icon.innerHTML = obj.options.data[i].icon;
    iconContainer.appendChild(icon);
    headerItem.appendChild(iconContainer);
}
```

Steps to Reproduce:

There is a public example hosted on the official website (https://jsuites.net/docs/javascript-tabs) that can be used to demonstrate the vulnerability. In the basic example that allows adding tabs, click on the "+" button to add one and enter the following title:

```
<img src=x onerror="alert('XSS in title')">
```

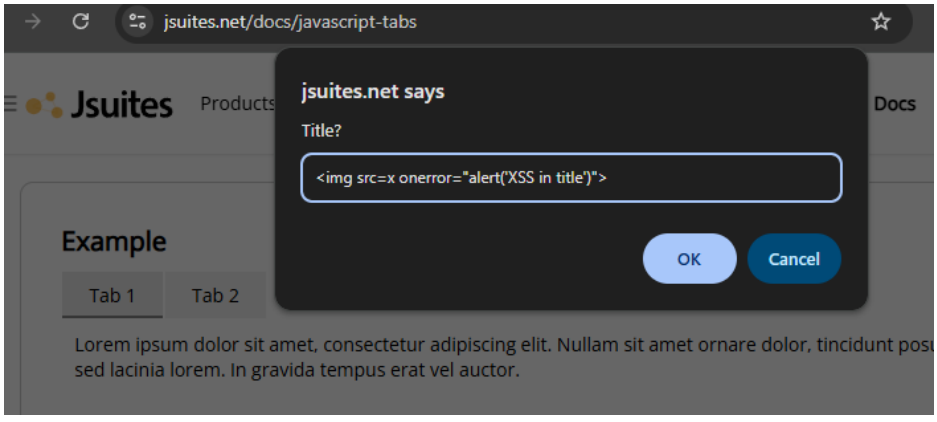


Figure 7: Entering a malicious payload in the component's dialog prompt.

Once the title is set, observe the alert dialog box, confirming the execution of the JavaScript payload:

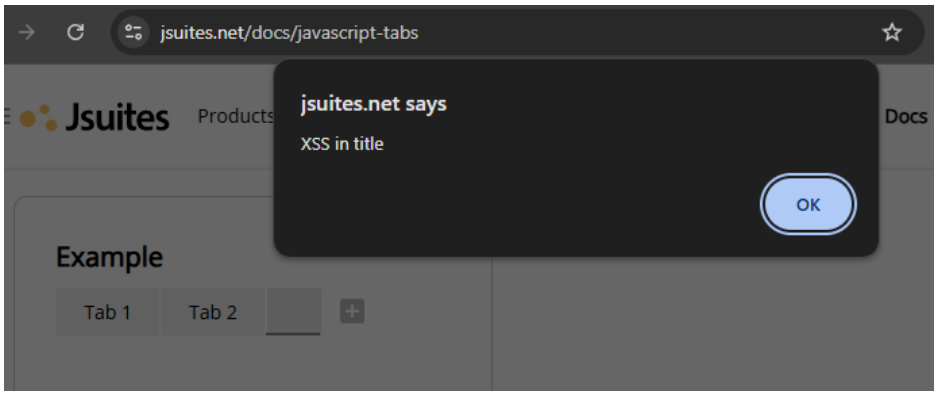


Figure 8: XSS in tab's title triggered via jSuites Tabs.

Impact:
A Cross-Site Scripting (XSS) vulnerability may allow arbitrary client-side code to be executed in the victim's web browser, leading to unintended actions on behalf of a logged-in user or even account takeover. The final severity of the impact depends on where the HTML editor component is used and the functionality supported by the website hosting the vulnerable component.

For the vulnerability to become exploitable, an attacker must be in control of the title of a new tab, which depends on how the component is integrated as part of a larger application.

Mitigation:
Replace the unsafe uses of the `innerHTML` property with `innerText` or `textContent`:

```
h.textContent = title;

icon.textContent = obj.options.data[i].icon;
```

JSA-07 - Potential Cross-Site Scripting in jSuites Organogram through Multiple Attributes

Severity	Medium
Category	Validation and Sanitization
Impact	High
Likelihood	Low
UniqueId	c79fd4b2-8d95-4bbe-b600-2384ee7449a2

Details:

The jSuites Organogram component is vulnerable to persistent Cross-Site Scripting (XSS) when a person's name, role, image path, or custom color contain a malicious payload.

The function responsible for building each block uses unsafe string interpolation, which literal contents are set through the `innerHTML` property:

jsuites/packages/organogram/organogram.js

```
var getContent = function(node) {
    var role = node.role;
    var color = node.color || 'lightgreen';
    if (obj.options.roles && node.role >= 0) {
        var o = getRoleById(node.role);
        if (o) {
            role = o.name;
            var color = o.color;
        }
    }

    return `<div class="jorg-user-status" style="background:${color}"></div>
        <div class="jorg-user-info">
            <div class='jorg-user-img'><img src="${node.img ? node.img : '#'}" ondragstart="return
              ↪  false" /></div>
            <div class='jorg-user-content'><span>${node.name}</span><span>${role}</span></div>
        </div>`;
}

// Creates the shape of a node to be added to the organogram chart tree
var mountNodes = function(node, container) {
    var li = document.createElement('li');
    var span = document.createElement('span');
    span.className = 'jorg-tf-nc';
    span.innerHTML = getContent(node);
    ...
```

Steps to Reproduce:

The following HTML page, which is based on a public example (https://jsuites.net/docs/organogram), can be used to demonstrate the vulnerability, by simulating contents that could come from user-controlled data:

organogram.html

```
<html>
<script src="https://jsuites.net/v5/jsuites.js"></script>
<link rel="stylesheet" href="https://jsuites.net/v5/jsuites.css" type="text/css" />
<script src="https://cdn.jsdelivr.net/npm/@jsuites/organogram/organogram.min.js"></script>
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/@jsuites/organogram/organogram.min.css"
 ↪  type="text/css" />

<div id='organogram'></div>

<script>
var orgData = [{
    "id": 1,
    "parent": 0,
    "name": "<img src=x onerror=\"alert('XSS in name')\">",
    "role": "<img src=x onerror=\"alert('XSS in role')\">",
    "img": "x\" onerror=\"alert('XSS in image')",
    "color": "red\"><img src=x onerror=\"alert('XSS in color')"
}]

jSuites.organogram(document.getElementById('organogram'), {data: orgData});
</script>
</html>
```

Open this file in any web browser and observe how several alert boxes will be displayed.

Figure 9: XSS in person's name triggered via jSuites Organogram.

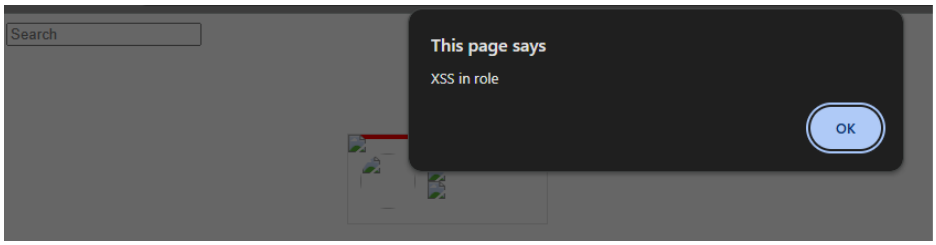


Figure 10: XSS in person's role triggered via jSuites Organogram.

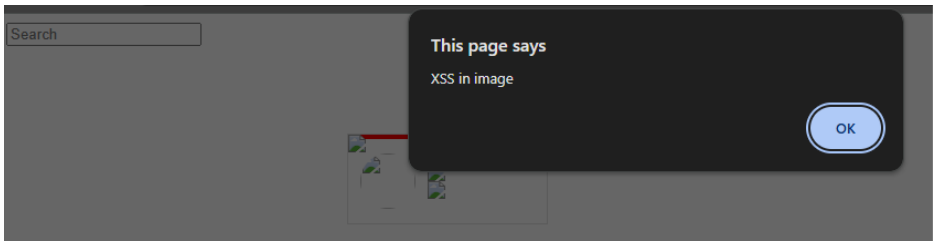


Figure 11: XSS in person's image triggered via jSuites Organogram.

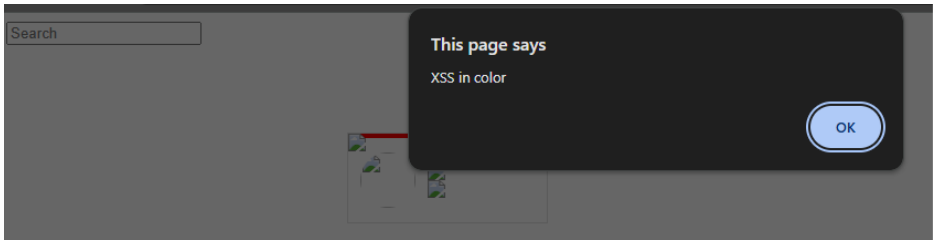


Figure 12: XSS in header's color triggered via jSuites Organogram.

Impact:
A Cross-Site Scripting (XSS) vulnerability may allow arbitrary client-side code to be executed in the victim's web browser, leading to unintended actions on behalf of a logged-in user or even account takeover. The final severity of the impact depends on where the HTML editor component is used and the functionality supported by the website hosting the vulnerable component.

For the vulnerability to become exploitable, an attacker must be in control of any of the affected input parameters, which depends on how the component is integrated as part of a larger application.

Mitigation:
Avoid using string concatenation or interpolation for contents ending on a `innerHTML` property. Replace the affected code with alternate ways to programmatically build HTML elements. For example:

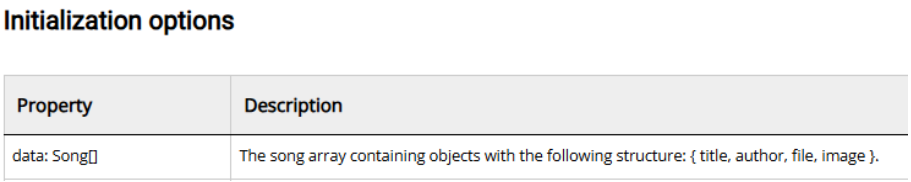
```
var userStatus = document.createElement('div');
userStatus.classList.add('jorg-user-status');
userStatus.style.backgroundColor = color;

...

var userContent = document.createElement('div');
userContent.classList.add('jorg-user-content');

var nameSpan = document.createElement('span');
nameSpan.textContent = node.name;

var roleSpan = document.createElement('span');
roleSpan.textContent = role;

```

```
16 userContent.appendChild(nameSpan);
17 userContent.appendChild(roleSpan);
```

JSA-04 - Potential Cross-Site Scripting in jSuites Player through Song Titles and Authors

Severity	Medium
Category	Validation and Sanitization
Impact	Medium
Likelihood	Low
UniqueId	ef3d8c2c-c180-4e91-a63b-0e52c55d06bd

Details:
The jSuites Player component is vulnerable to persistent Cross-Site Scripting (XSS) when a song to be reproduced contains a malicious payload in the song's title or author. The public documentation does not have any reference explaining these properties, or explicitly warning the integrators about the dangers of allowing unsanitized input.



Initialization options

Property	Description
data: Song[]	The song array containing objects with the following structure: { title, author, file, image }.

Figure 13: Public documentation of the jSuites Player component.

However, both fields are internally treated as HTML code instead of plaintext strings, making them dangerous.

jsuites/packages/player/player.js

```
obj.loadSong = function() {
    const audioObj = getCurrentAudio();
    ...

    songImage.src = audioObj.image;
    songTitle.href = '/songs/' + audioObj.id;
    songTitle.innerHTML = audioObj.title;
    songArtist.innerHTML = audioObj.author;

    queue.href = obj.options.queueRedirect;
}
```

Steps to Reproduce:
There is a public example hosted on the official website (https://jsuites.net/docs/player) that can be used to demonstrate the vulnerability. In the example that allows adding songs programmatically from a JSON string,

add the following entry:

```
{"title": "<img src=x onerror=\"alert('XSS in title')\">",
"author": "<img src=x onerror=\"alert('XSS in author')\">"}
```

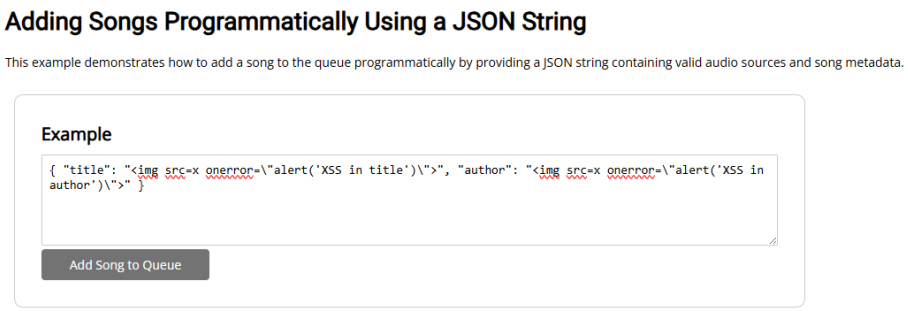


Figure 14: Public example allowing to add songs programmatically.

Once the song is added, observe the two consecutive alert dialog boxes, confirming the execution of both JavaScript payloads:

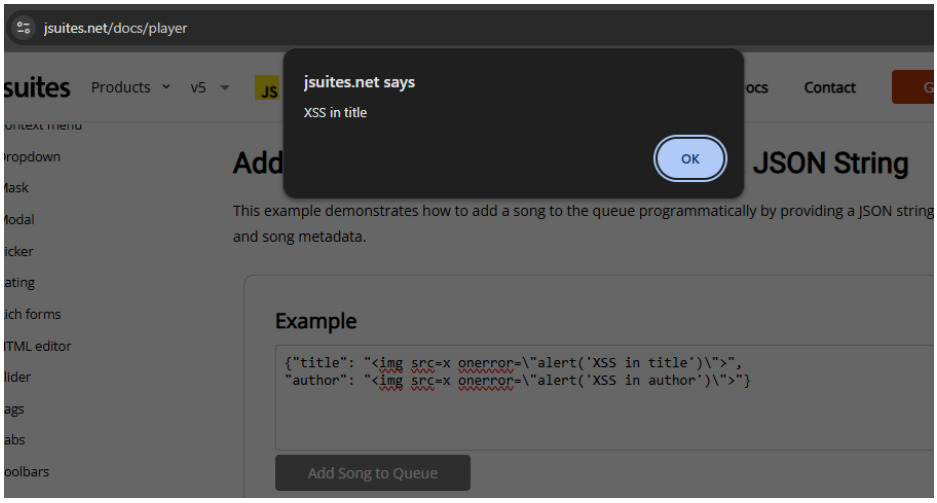


Figure 15: XSS in song's title triggered via jSuites Player.

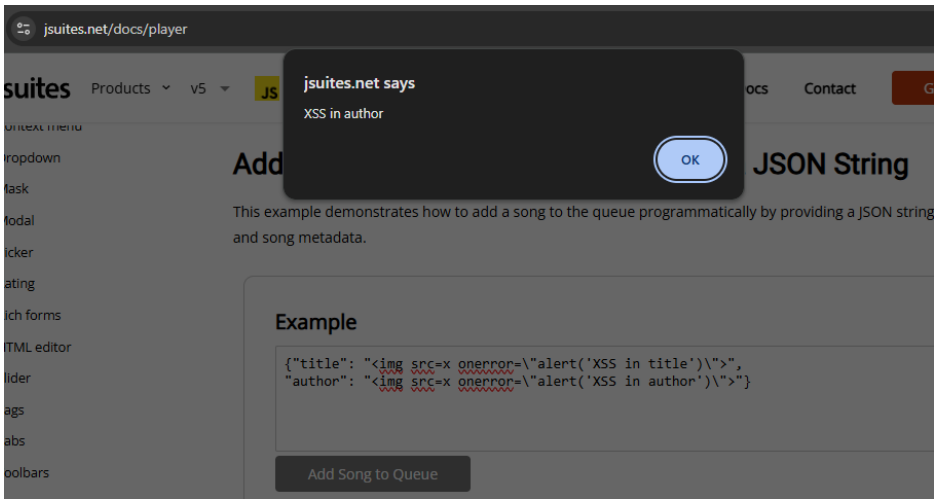


Figure 16: XSS in song's author triggered via jSuites Player.

Impact:
A Cross-Site Scripting (XSS) vulnerability may allow arbitrary client-side code to be executed in the victim's web browser, leading to unintended actions on behalf of a logged-in user or even account takeover. The final severity of the impact depends on where the HTML editor component is used and the functionality supported by the website hosting the vulnerable component.

For the vulnerability to become exploitable, an attacker must be in control of the song title or artist name, which depends on how the component is integrated as part of a larger application.

Mitigation:
Song titles and authors should not be expected to contain HTML contents, especially when the component already supports other fields like the song's image. Anvil recommends replacing the unsafe `innerHTML` property with `innerText` or `textContent`:

```
songTitle.innerText = audioObj.title;
songArtist.innerText = audioObj.author;
```

JSA-05 - Potential Cross-Site Scripting in jSuites Heatmap through Title and Colors

Severity	Medium
Category	Validation and Sanitization
Impact	Medium
Likelihood	Low
UniqueId	f9a477e9-d798-4970-a6a8-fa1ea87d42f0

Details:

The jSuites Heatmap component is vulnerable to persistent Cross-Site Scripting (XSS) when its title or any custom colors contains a malicious payload. The public documentation does not have any reference explaining these properties, or explicitly warning the integrators about the dangers of allowing unsanitized input.

Observe how the code unsafely concatenates these input variables in strings that end in an `innerHTML` properties:

jsuites/packages/heatmap/heatmap.js

```
// Initializes the plugin
var init = (function() {
    ...
    // Apply the plugin header if it was passed as an argument
    if (obj.options.title !== '') {
        var pluginHeader = '<div class="jheatmap-header">' + obj.options.title + '</div>';

        el.innerHTML = pluginHeader;
    }
```

jsuites/packages/heatmap/heatmap.js

```
    // Apply the plugin tooltip if it was passed as an argument
    if (obj.options.tooltip) {
        var pluginFooter = '<div class="jheatmap-footer"><div>Less</div><table><tr><td
        ↪ style="background-color:' + obj.options.colors[0] + '"></td><td
        ↪ style="background-color:' + obj.options.colors[1] + '"></td><td
        ↪ style="background-color:' + obj.options.colors[2] + '"></td><td
        ↪ style="background-color:' + obj.options.colors[3] + '"></td><td
        ↪ style="background-color:' + obj.options.colors[4] +
        ↪ '"></td></tr></table><div>More</div></div>';

        el.innerHTML += pluginFooter;
    }

...

// Create and apply the plugin body
var createBody = function() {
```

```
    ...
    // If currentDay exists, a TD referring to it is added with a color resulting from its value
    if (currentDay) {
        ...
        setOfDays[date.getDay()].push('<td style="background-color:' +
        ↪  obj.options.colors[colorPosition] + '">' + '</td>');
        ...
        // Apply the plugin body to the tag passed as an argument
        el.getElementsByClassName('jheatmap-body')[0].innerHTML = pluginBody;
    }
```

Steps to Reproduce:

Taking one of the examples from the public documentation (https://jsuites.net/docs/heatmap) as a starting point, set XSS payloads in the `title` variable and `colors` array, to simulate user-controlled input:

heatmap-xss.html

```
<html>
<script src="https://jsuites.net/v5/jsuites.js"></script>
<link rel="stylesheet" href="https://jsuites.net/v5/jsuites.css" type="text/css" />

<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/@jsuites/heatmap/heatmap.min.css"
↪  type="text/css" />
<script type="text/javascript"
↪  src="https://cdn.jsdelivr.net/npm/@jsuites/heatmap/heatmap.min.js"></script>

<div id='root' style="padding: 40px;"></div>

<script>
var initialDate = '2021-01-01';

var year = [];
var date = new Date(initialDate);

while (year.length < 5) {
    year.push({
        date: date.toISOString().slice(0, 10),
        value: year.length,
    });

    date.setDate(date.getDate() + 1);
}

jSuites.heatmap(document.getElementById('root'), {
    title: '<img src=x onerror="alert(\'XSS in title\')">',
    data: year,
    date: initialDate,
    colors: ['x"><img src=x onerror="alert(\'XSS in color\')"><x ', '#4DB6AC', '#009688',
    ↪  '#00796B', '#004D40'],
    tooltip: true,
});
</script>
</html>
```

Note that other minor changes were also made to the original example, to reduce the number of displayed

pop-up alerts and to guarantee that the color containing the injected payload will be present in the rendered heatmap, instead of being randomly generated.

Open this file in any web browser and observe how several alert boxes will be displayed.

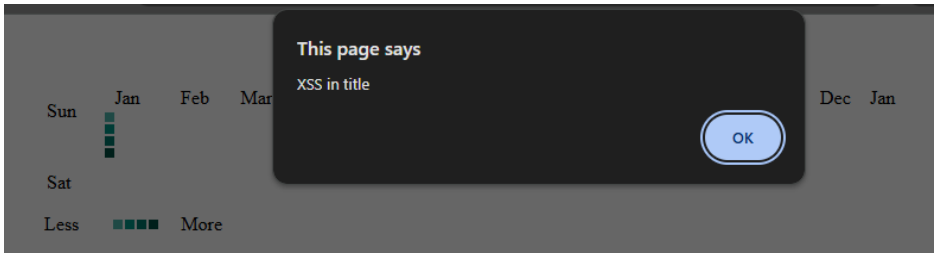


Figure 17: XSS in title triggered via jSuites Heatmap.

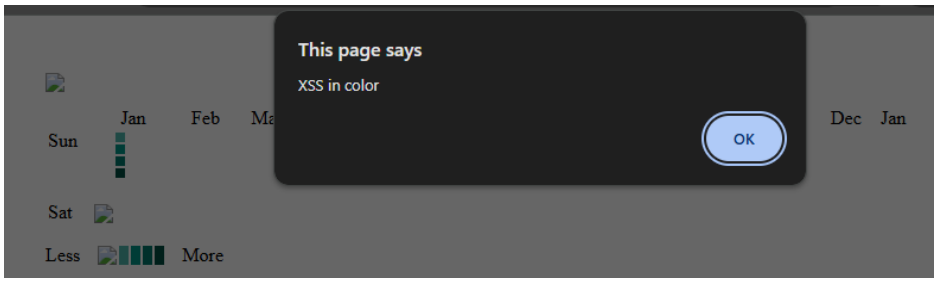


Figure 18: XSS in one of the colors triggered via jSuites Heatmap.

Mitigation:

Depending on the context, the affected input parameters are not necessarily exposed and under the end user's control, as this is a software component meant to be integrated as part of a larger application. However, the unsafe way they are rendered as HTML can be significantly improved, mitigating risks such as injection vulnerabilities.

The recommended way to generate HTML elements is by doing it programmatically. For example, to ensure that the title does not allow any injections:

```
var div = document.createElement('div');
div.className = 'jheatmap-header';
div.textContent = obj.options.title;
```

Similarly, for the background color of a cell, a safer alternative would be:

```
var td = document.createElement('td');
td.style.backgroundColor = obj.options.colors[colorPosition];
```

About Anvil Secure

Anvil was founded in 2016 with a vision to make a change in the information security consulting services industry. Anvil has since grown to be an industry recognized information security partner to some of the largest tech and Fortune 500 companies across the globe.

Anvil was founded with the principles of creating an information consulting services company that is honest, transparent, professional, and that will consistently deliver quality services to our clients. Anvil continues to hold these principles to this day and is now known for delivering consistently quality service, and for our transparent and inclusive approach.

Anvil's team is filled with dedicated industry veterans that are experts in fields such as:

- embedded/hardware security
- industrial control systems
- cloud security
- web application security
- mobile security
- network security
- operating system security

Several team members come from National labs and other industry recognized consulting firms. The team's technical backgrounds and consulting experiences position Anvil to effectively improve the security posture of leading technology companies and security groups.

Anvil is headquartered in Seattle with an office in Amsterdam as well as several employees located around the globe including France, Spain, and Argentina.

For more information about Anvil and to stay up to date on our latest research and progress, visit our website and social media pages:

- Website
- LinkedIn
- Twitter

For any questions or further information, please reach out via Email or Phone:

- Email: info@anvilsecure.com
- Phone: 206-753-7649