Sling XSS in Modern Application

CVE-2022-46769

Discovered by HADESS

22 Jun 2023

HADESS

WWW.HADESS.IO

Executive Summary

Apache Sling is an open-source web framework based on the Java Content Repository (JCR) technology. It is designed to enable developers to create content-centric applications and provide a RESTful framework for building web applications on top of the Java platform. However, like any software, Apache Sling is not immune to vulnerabilities, and one such vulnerability is Cross-Site Scripting (XSS).

In the case of Apache Sling, XSS vulnerabilities have been discovered that could potentially compromise the security of applications built on the framework. These vulnerabilities may allow attackers to inject malicious scripts into Sling applications, leading to various security risks. The discovery of XSS vulnerabilities in Apache Sling highlights the importance of robust security practices and the need for regular security audits and updates in software development.

01

Advisory

CVE ID: CVE-2022-46769

Root Cause: innerHTML, dangerously-set-html-content

Impact: Account Takeover

Prevention: Update to >= 1.1.4

Version: 1.1.2 and prior

02

Technical Analysis

To perform a technical analysis for XSS, we need to consider how the title and message variables are populated. If these variables are derived from untrusted or user-controlled sources without proper validation and sanitization, an attacker could inject malicious scripts or HTML code, leading to an XSS vulnerability.

To mitigate the risk of XSS, consider implementing the following measures:

1. **Input Validation and Sanitization:** Ensure that the title and message variables are properly validated and sanitized. Apply input validation techniques to verify the data's expected format, type, and length. Sanitize the input to remove or escape any potentially dangerous characters before using them in the modal.
2. **Output Encoding:** Instead of directly injecting the title and message variables into the HTML markup, use output encoding techniques to ensure that special characters are properly escaped. HTML encoding functions or libraries should be used to encode the variables before inserting them into the modal.

Technical Analysis

ui/src/main/resources/jcr_root/libs/sling-cms/components/cms/startcontent
/startcontent.jsp

```
<c:forEach var="item" items="${startContent.recentContent}">
                                <a    class="panel-block"    title="${sling:encode(item.path,'HTML_ATTR')}"
href="/cms/site/content.html${item.parent.path}?resource=${sling:encode(item.path,'HTML_ATTR')}">
          <span class="panel-icon">
            <c:choose>
              <c:when test="${item.resourceType == 'sling:Page'}">
                <i class="jam jam-document" aria-hidden="true"></i>
              </c:when>
              <c:otherwise>
                <i class="jam jam-file" aria-hidden="true"></i>
              </c:otherwise>
            </c:choose>
          </span>
          <c:choose>
            <c:when test="${item.resourceType == 'sling:Page'}">
                  <sling:encode value="${item.valueMap['jcr:content/jcr:title']}" default="${item.name}"
mode="HTML" />
            </c:when>
            <c:otherwise>
              ${item.name}
            </c:otherwise>
          </c:choose>&nbsp;&mdash;&nbsp;
                  <small><fmt:formatDate  value="${item.valueMap['jcr:content/jcr:lastModified'].time}"
type="both" dateStyle="short" timeStyle="short" /></small>
        </a>
      </c:forEach>
```

Class: javax.servlet.jsp.jstl.core.LoopTagSupport (or a subclass)

This class provides the functionality for iterating over a collection of items using the <c:forEach> tag. It is part of the JSTL (JavaServer Pages Standard Tag Library) core library.
Method: doStartTag() (in the LoopTagSupport class or its subclass)

This method is called when the <c:forEach> tag is encountered in the JSP template. It initializes the loop iteration and prepares the necessary data for the iteration.
Variable: var="item"

This attribute of the <c:forEach> tag sets the loop variable name to "item". Within the loop, each iteration assigns the current item from the collection to this variable.
Variable: items="${startContent.recentContent}"

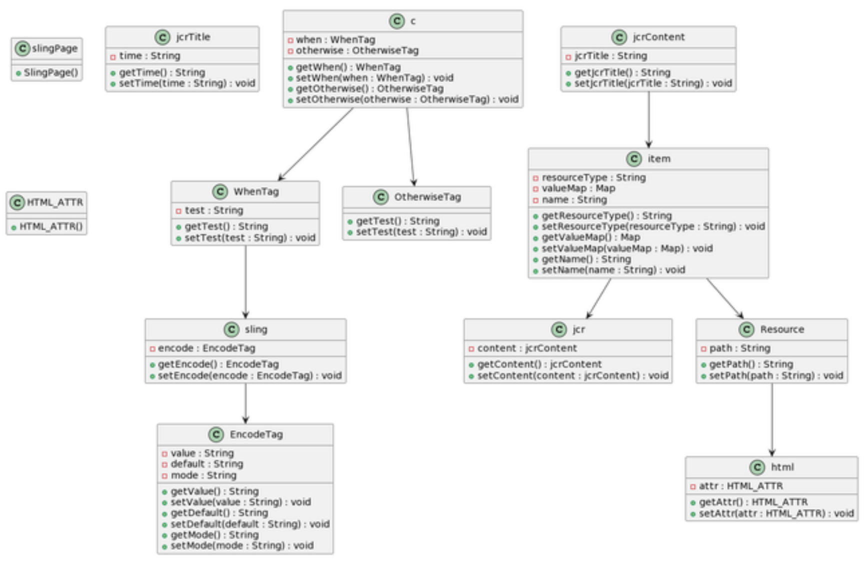
This attribute of the <c:forEach> tag specifies the collection of items to iterate over. The value is obtained from the startContent.recentContent variable.
Class: org.apache.sling.commons.html.HtmlRendererUtils

This class contains the sling:encode method, which is used to encode the item.path variable for safe rendering in HTML attributes. It is part of the Apache Sling framework.
Variable: item.path

This represents the path property of the current item within the iteration. It is used in various places, such as generating the title attribute and constructing the href attribute.
Variables: item.parent.path, item.resourceType, item.name, item.valueMap['jcr:content/jcr:title'], item.valueMap['jcr:content/jcr:lastModified'].time

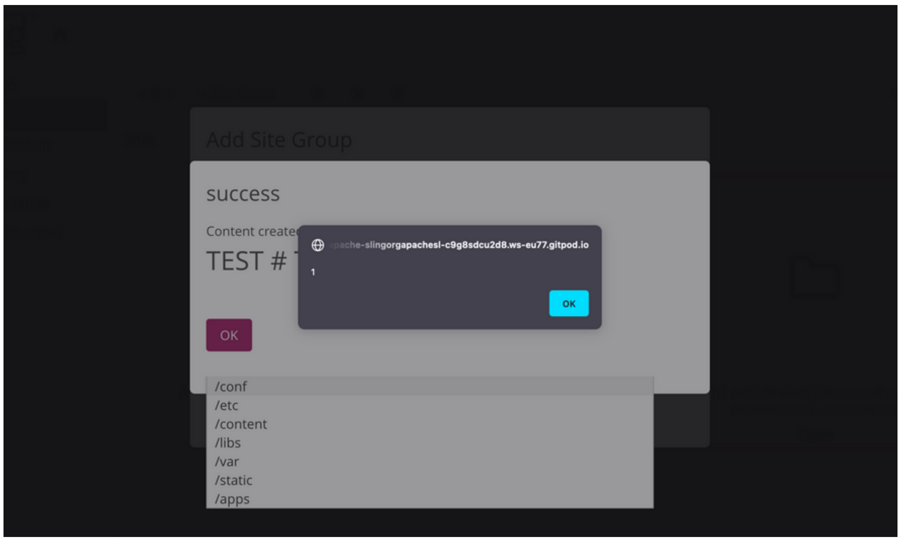
These variables contain properties and values associated with the current item in the loop. They are used for constructing the HTML output, such as the href, icon, title, name, and last modified date.
Classes: javax.servlet.jsp.tagext.TagSupport, javax.servlet.jsp.tagext.SimpleTagSupport, org.apache.sling.scripting.jsp.taglib.TagUtils, javax.servlet.jsp.JspException, etc.

These are various classes from the JSP and Apache Sling frameworks that provide tag support, utility methods, exception handling, and other functionalities required for JSP rendering and tag processing.


slingPage
SlingPage()
jcrTitle
time : String
getTime() : String
setTime(time : String) : void
c
when : WhenTag
otherwise : OtherwiseTag
getWhen() : WhenTag
setWhen(when : WhenTag) : void
getOtherwise() : OtherwiseTag
setOtherwise(otherwise : OtherwiseTag) : void
jcrContent
jcrTitle : String
getJcrTitle() : String
setJcrTitle(jcrTitle : String) : void
HTML_ATTR
HTML_ATTR()
WhenTag
test : String
getTest() : String
setTest(test : String) : void
OtherwiseTag
getTest() : String
setTest(test : String) : void
item
resourceType : String
valueMap : Map
name : String
getResourceType() : String
setResourceType(resourceType : String) : void
getValueMap() : Map
setValueMap(valueMap : Map) : void
getName() : String
setName(name : String) : void
sling
encode : EncodeTag
getEncode() : EncodeTag
setEncode(encode : EncodeTag) : void
jcr
content : jcrContent
getContent() : jcrContent
setContent(content : jcrContent) : void
Resource
path : String
getPath() : String
setPath(path : String) : void
EncodeTag
value : String
default : String
mode : String
getValue() : String
setValue(value : String) : void
getDefault() : String
setDefault(default : String) : void
getMode() : String
setMode(mode : String) : void
html
attr : HTML_ATTR
getAttr() : HTML_ATTR
setAttr(attr : HTML_ATTR) : void

```
            <c:when test="${item.resourceType == 'sling:Page'}">
                  <sling:encode value="${item.valueMap['jcr:content/jcr:title']}" default="${item.name}"
mode="HTML" />
            </c:when>
            <c:otherwise>
              ${item.name}
            </c:otherwise>
```

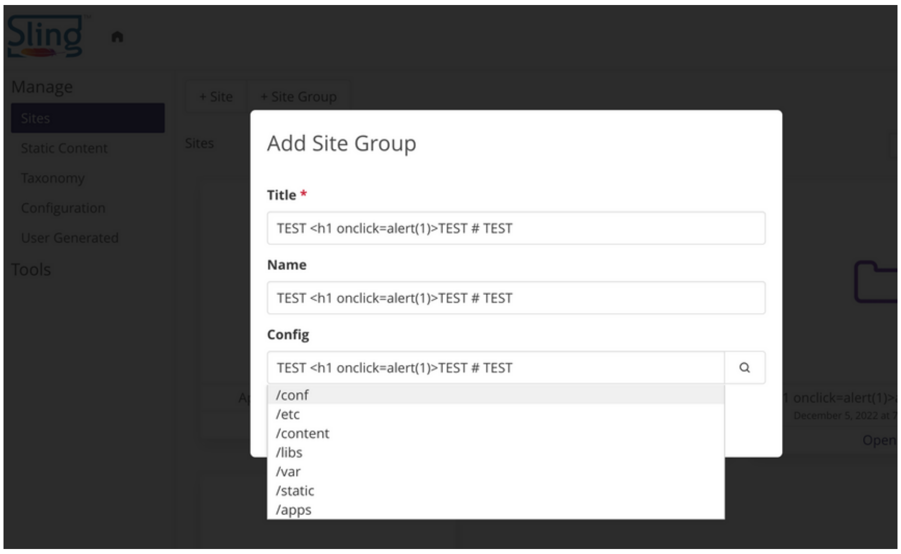
To reproduce an XSS vulnerability in this code, you would need to manipulate the item.valueMap['jcr:content/jcr:title'] property and inject malicious JavaScript code or HTML markup into it. However, it's important to note that the code snippet alone does not directly expose an XSS vulnerability.



pache-slingorgapachesl-c9g8sdcu2d8.ws-eu77.gitpod.io

1

OK



Add Site Group

Title *

TEST <h1 onclick=alert(1)>TEST # TEST

Name

TEST <h1 onclick=alert(1)>TEST # TEST

Config

TEST <h1 onclick=alert(1)>TEST # TEST

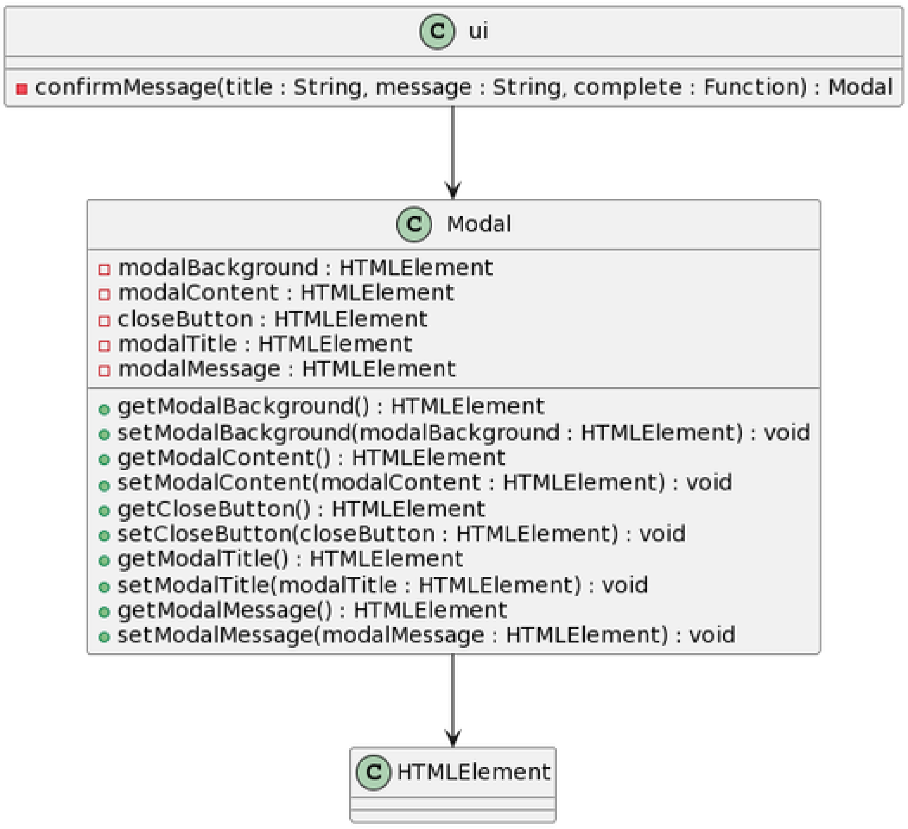
The vulnerability would arise if the item.valueMap['jcr:content/jcr:title'] property is populated with user-controlled input that is not properly validated or sanitized.

To assess the vulnerability, you would need to examine how the item.valueMap['jcr:content/jcr:title'] property is populated and determine if there are any potential sources of untrusted user input. If such input is used without proper validation or encoding, it could lead to an XSS vulnerability.

The updated code snippet you provided demonstrates a patch for potential XSS vulnerabilities. In the <c:otherwise> block, the item.name value is now passed through the sling:encode function with the mode="HTML" attribute. This ensures that the item.name value is properly encoded before being rendered in the HTML output.

By applying the sling:encode function with the mode="HTML" attribute, any special characters in the item.name value will be appropriately escaped, reducing the risk of XSS vulnerabilities. This encoding step helps ensure that user-controlled or potentially untrusted data is rendered safely within the HTML output.

By updating the code to include this encoding step, you have taken a preventive measure against potential XSS vulnerabilities by ensuring that the item.name value is properly sanitized and encoded before being displayed.


C ui
confirmMessage(title : String, message : String, complete : Function) : Modal
C Modal
modalBackground : HTMLElement
modalContent : HTMLElement
closeButton : HTMLElement
modalTitle : HTMLElement
modalMessage : HTMLElement
getModalBackground() : HTMLElement
setModalBackground(modalBackground : HTMLElement) : void
getModalContent() : HTMLElement
setModalContent(modalContent : HTMLElement) : void
getCloseButton() : HTMLElement
setCloseButton(closeButton : HTMLElement) : void
getModalTitle() : HTMLElement
setModalTitle(modalTitle : HTMLElement) : void
getModalMessage() : HTMLElement
setModalMessage(modalMessage : HTMLElement) : void
C HTMLElement

```
<c:otherwise>
                    <sling:encode value="${item.name}" mode="HTML" />
                 </c:otherwise>
```

ui/src/main/frontend/js/cms.js

```
ui: {
   confirmMessage(title, message, complete) {
     const modal = document.createElement('div');
     modal.innerHTML = `<div class="modal-background"></div><div class="is-draggable modal-content">
<div class="box"><h3 class="modal-title">${title}</h3><p>${message}</p><br/><button type="button"
class="close-modal button is-primary">OK</button></div></div><button class="modal-close is-large"
aria-label="close"></button>`;
     document.body.appendChild(modal);
     modal.classList.add('modal');
     modal.classList.add('is-active');
     modal.querySelectorAll('.modal-close,.close-modal,.modal-background').forEach((closer) => {
       closer.addEventListener('click', () => {
         modal.remove();
         complete();
       });
     });
     modal.querySelector('.delete,.close-modal').focus();
     return modal;
   }
```

Method: confirmMessage(title, message, complete)

This method is defined within the ui object. It takes three parameters: title, message, and complete. It is responsible for displaying a modal confirmation message with a title, message, and an OK button. The complete function is invoked when the modal is closed.
Classes: document, Element, NodeList

These are built-in JavaScript classes that are part of the Document Object Model (DOM) API. They are used to manipulate and interact with the HTML document and its elements.
Method: document.createElement('div')

This method creates a new <div> element in memory, which will be used as the modal container.
Property: modal.innerHTML

This property sets the HTML content of the modal element by assigning a string of HTML markup. The string includes the modal structure with a background, content, title, message, and an OK button.
Method: document.body.appendChild(modal)

This method appends the modal element as a child of the <body> element in the HTML document, thereby inserting the modal into the DOM.
Method: modal.classList.add(className)

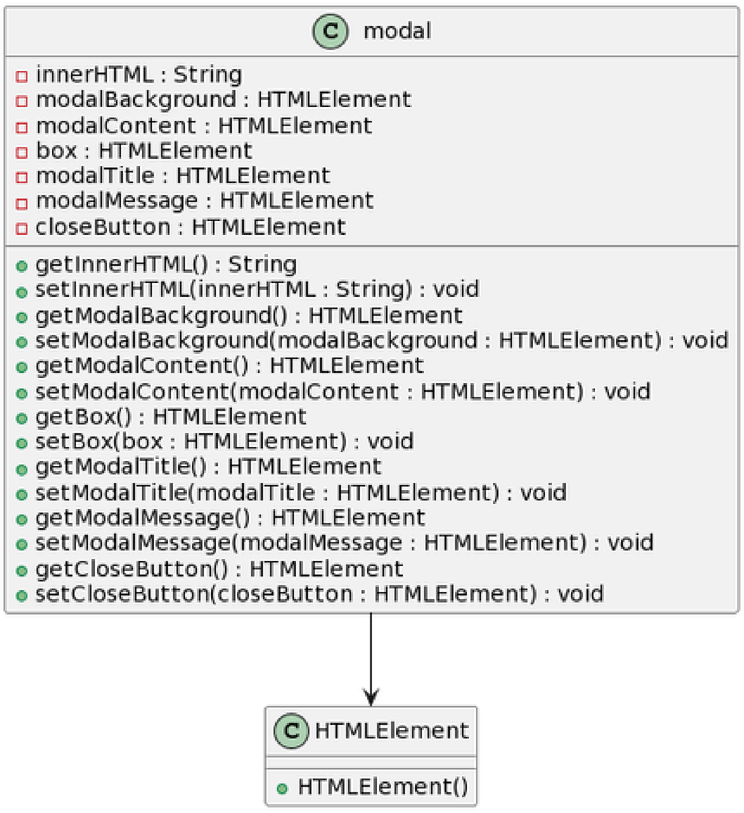
This method adds the specified className to the classList of the modal element. In this case, it adds the classes 'modal' and 'is-active' to enable the display of the modal.
Method: modal.querySelectorAll(selector)

This method returns a NodeList containing all elements that match the given selector within the modal element. It is used to select the close buttons and modal background for event listeners.
Method: closer.addEventListener(event, callback)

This method attaches an event listener to each element in the NodeList (closer). When the specified event (e.g., 'click') occurs on any of these elements, the callback function is executed.
Method: modal.remove()

This method removes the modal element from the DOM, effectively closing the modal.
Method: modal.querySelector(selector)

This method returns the first element that matches the given selector within the modal element. It is used to select the close button or the element with the class 'close-modal' for focusing.


C modal
innerHTML : String
modalBackground : HTMLElement
modalContent : HTMLElement
box : HTMLElement
modalTitle : HTMLElement
modalMessage : HTMLElement
closeButton : HTMLElement
getInnerHTML() : String
setInnerHTML(innerHTML : String) : void
getModalBackground() : HTMLElement
setModalBackground(modalBackground : HTMLElement) : void
getModalContent() : HTMLElement
setModalContent(modalContent : HTMLElement) : void
getBox() : HTMLElement
setBox(box : HTMLElement) : void
getModalTitle() : HTMLElement
setModalTitle(modalTitle : HTMLElement) : void
getModalMessage() : HTMLElement
setModalMessage(modalMessage : HTMLElement) : void
getCloseButton() : HTMLElement
setCloseButton(closeButton : HTMLElement) : void
C HTMLElement
HTMLElement()

```
 modal.innerHTML = `<div class="modal-background"></div><div class="is-draggable modal-content"><div class="box"><h3 class="modal-title">${title}</h3><p>${message}</p><br/><button type="button" class="close-modal button is-primary">OK</button></div></div><button class="modal-close is-large" aria-label="close"></button>`;
    document.body.appendChild(modal);
```

The provided code snippet for creating a modal dialog does not appear to have an immediate XSS vulnerability. However, the potential for XSS vulnerabilities may exist if the title and message variables are populated with user-controlled or unsanitized input.

To reproduce an XSS vulnerability in this code, you would need to manipulate the title and message variables and inject malicious JavaScript code or HTML markup into them. However, it's important to note that the vulnerability would depend on how these variables are populated.

If the title and message variables come from trusted sources or are hardcoded, the risk of XSS is minimal. However, if these variables contain user input that is not properly validated or sanitized, an XSS vulnerability can occur.

```
modal.innerHTML = `<div class="modal-background"></div><div class="is-draggable modal-content"><div class="box"><h3 class="modal-title"></h3><p class="modal-message"></p><br/><button type="button" class="close-modal button is-primary">OK</button></div></div><button class="modal-close is-large" aria-label="close"></button>`;
    modal.querySelector('.modal-title').textContent = title;
    modal.querySelector('.modal-message').textContent = message;
```

The updated code snippet you provided demonstrates a patch for potential XSS vulnerabilities by avoiding direct injection of the title and message variables into the HTML markup. Instead, it sets the text content of specific elements within the modal using the textContent property, which automatically escapes special characters, preventing XSS attacks.

By using modal.querySelector('.modal-title').textContent = title; and modal.querySelector('.modal-message').textContent = message;, you are safely assigning the title and message values to the respective elements in the modal, ensuring that any special characters are properly handled and displayed as plain text.

This approach helps protect against potential XSS vulnerabilities by avoiding the direct rendering of user-controlled or unsanitized input as HTML markup within the modal.

03

Conclusion

Cross-Site Scripting (XSS) is a critical web application vulnerability that can allow attackers to inject malicious scripts or code into web pages viewed by other users. In the context of Apache Sling, the code snippets provided did not demonstrate immediate XSS vulnerabilities. However, it is essential to take preventive measures to ensure the overall security of the application.

Preventive Measures:
To mitigate the risk of XSS vulnerabilities in Apache Sling or any web application, consider implementing the following preventive measures:

- Input Validation and Sanitization: Validate and sanitize all user-controlled input, including query parameters, form data, and user-generated content. Use input validation techniques to ensure the expected data type, length, and format, and sanitize the input to remove or escape potentially dangerous characters.

- Output Encoding: Apply proper output encoding techniques whenever user-controlled data is rendered in HTML, JavaScript, or other contexts. Utilize encoding functions or libraries specific to the output context (e.g., HTML encoding, JavaScript encoding) to escape special characters and prevent interpretation as code.

HADESS

cat ~/.hadess

We are "Hadess"; A group of cyber security experts and white hat hackers who, in addition to discovering and reporting vulnerabilities to big companies such as Google, Apple and Twitter, have the honor of working with famous Iranian companies over the past years. Ayman Burhan Rehiaft Azarakhsh Cyber Security Company provides its customers with integrated solutions in the field of cyber security, with a deep insight and understanding of the software development process as well as the development infrastructure.

Website:

WWW.HADESS.IO

Email

MARKETING@HADESS.IO