Markdown's XSS Vulnerability (and how to mitigate it)

Jump to bottom

Estevão Soares dos Santos edited this page on May 11, 2018 · 4 revisions

Introduction

Cross-Site Scripting (XSS) is a well known technique to gain access to private information of the users of a website. The attacker injects spurious HTML content (a script) on the web page which will read the user's cookies and do something bad with it (like steal credentials). As a countermeasure, you should filter any suspicious content coming from user input. Showdown doesn't include an XSS filter, so you must provide your own. But be careful in how you do it...

Markdown is inherently unsafe

Markdown syntax allows for arbitrary HTML to be included. For instance, this is perfectly valid markdown:

```
This is a regular paragraph.

<table>
    <tr><td>Foo</td></tr>
</table>

This is another regular paragraph.
```

This means a malicious user could do something like this:

```
This is a regular paragraph.

<script>alert('xss');</script>

This is another regular paragraph.
```

While `alert('xss');` is hardly problematic (maybe just annoying) a real scenario might be a lot worse. Obviously, this kind of straightforward attack can be easily prevented. For instance, Showdown could provide some kind of whitelist where only certain HTML tags are allowed. However, this can be easily circumvented...

Whitelist / Blacklist can't prevent XSS

Consider the following markdown content:

```
hello <a href="www.google.com">*you*</a>
```

As you see, it's a link, nothing really malicious about this. And `<a>` tags are pretty innocuous right? Showdown should definitely allow `<a>` tags. What if the content is altered slightly, like this:

```
hello <a name="n" href="javascript:alert('xss')">*you*</a>
```

Now this is a lot more problematic. Once again, it's not that hard to filter Showdown's input to expunge problematic attributes (such as `href` in `<a>` tags) of scripting attacks. In fact, a regular HTML XSS prevention library will probably catch this kind of straightforward attack.

At this point you're probably thinking that the best way is to follow Stackoverflow's cue and simply disallow embedded HTML in markdown. Well, unfortunately it's not enough.

Striping HTML tags is not enough

Consider the following markdown input:

```
[some text](javascript:alert('xss'))
```

Showdown will correctly parse this piece of markdown input as:

```
<a href="javascript:alert('xss')">some text</a>
```

In this case, it was Markdown's syntax itself to create the dangerous link. No HTML XSS filter can catch this. And unless you start striping dangerous words like *javascript* (which would make this article extremely hard to write), there's nothing you can really do to filter XSS attacks from your input. Things get even harder when you tightly mix HTML with Markdown.

Mixed HTML/Markdown XSS attack

Consider the following piece of markdown:

```
> hello <a name="n"
> href="javascript:alert('xss')">*you*</a>
```

If we apply a XSS filter to this Markdown input to filter bad HTML, the XSS filter, expecting HTML, will likely think the `<a>` tag ends with the first character on the second line and will leave the text snippet untouched. It will probably fail to see that the `href="javascript:…"` thing is part of the `<a>` element and leave it alone. But when Markdown converts this to HTML, you get this:

```
<blockquote>
 <p>hello <a name="n"
 href="javascript:alert('xss')"><em>you</em></a></p>
</blockquote>
```

After parsing with Markdown, the first > on the second line disappears because it is used as the blockquote marker in the Markdown blockquote syntax, and now you've got a link containing an XSS attack!

Did Markdown generate the HTML? No, the HTML was already in plain sight in the input. The XSS filter couldn't catch it because the input doesn't follow HTML's rules: it's a mix of Markdown and HTML and the filter doesn't know a dime about Markdown.

Mitigating XSS

So, is it all lost? Not really. The answer is not to filter the *input*, but rather the *output*. After the *input* text is converted into full fledged HTML, you can then reliably apply the correct XSS filters to remove any dangerous or malicious content.

Also, client-side validations are not reliable. This should be a given, but in case you're wondering, you should (almost) never trust data sent by the client. If there's some critical operation you must perform to the data (such as XSS filtering), it should be done *SERVER SIDE* not client side.

HTML XSS filtering libraries are useful here, since they prevent most of the attacks. However, you should not use them blindly: a library can't predict all the contexts and situation your application may face.

Conclusion

Showdown tries to convert the input text as closely as possible, without any concerns for XSS attacks or malicious intent. So, the basic rules are:

- **removing HTML entities from markdown does not prevent XSS**. Markdown syntax can generate XSS attacks.
- **XSS filtering should be done AFTER Showdown has processed any input, not before or during**. If you filter before, it'll break some of Markdown's features and will leave security holes.
- **perform the necessary filtering server-side, not client side**. XSS filtering libraries are useful but shouldn't be used blindly.

Disclaimer

This wiki page is based on "Markdown and XSS" excellent article by Michel Fortin

- Showdown Options
- Showdown's Markdown syntax
- Tutorial: Markdown editor using Showdown

Clone this wiki locally

```
https://github.com/showdownjs/showdown.wiki.git
```