Why do I see an "Electron Security Warning" after updating my Electron project to the latest version?

By Un1

Published: 2018-02-18 · Archived: 2026-04-05 22:27:49 UTC

You're having this:

> Electron Security Warning This renderer process has Node.js integration enabled and attempted to load remote content. This exposes users of this app to severe security risks.

Because from the 2nd Security Recommendations from Electron Documentation

2) Disable Node.js Integration for Remote Content

It is paramount that you disable Node.js integration in any renderer (BrowserWindow, BrowserView, or WebView) that loads remote content. The goal is to limit the powers you grant to remote content, thus making it dramatically more difficult for an attacker to harm your users should they gain the ability to execute JavaScript on your website.

After this, you can grant additional permissions for specific hosts. For example, if you are opening a BrowserWindow pointed at "https://my-website.com/", you can give that website exactly the abilities it needs, but no more.

Why?

A cross-site-scripting (XSS) attack is more dangerous if an attacker can jump out of the renderer process and execute code on the user's computer. Cross-site-scripting attacks are fairly common - and while an issue, their power is usually limited to messing with the website that they are executed on. Disabling Node.js integration helps prevent an XSS from being escalated into a so-called "Remote Code Execution" (RCE) attack.

How?

```
 // Bad
const mainWindow = new BrowserWindow()
mainWindow.loadURL('https://my-website.com')

// Good
const mainWindow = new BrowserWindow({
  webPreferences: {
    nodeIntegration: false,
    preload: './preload.js'
  }
```

```
})

mainWindow.loadURL('https://my-website.com')
```

```
 <!-- Bad -->
<webview nodeIntegration src="page.html"></webview>

<!-- Good -->
<webview src="page.html"></webview>
```

When disabling Node.js integration, you can still expose APIs to your website that do consume Node.js modules or features. Preload scripts continue to have access to require and other Node.js features, allowing developers to expose a custom API to remotely loaded content.

In the following example preload script, the later loaded website will have access to a `window.readConfig()` method, but no Node.js features.

```
 const { readFileSync } = require('fs')

window.readConfig = function () {
  const data = readFileSync('./config.json')
  return data
}
```

Therefore you're been warned so that you can **Disable Node.js Integration for Remote Content**.

I hope this helps answer your question.
