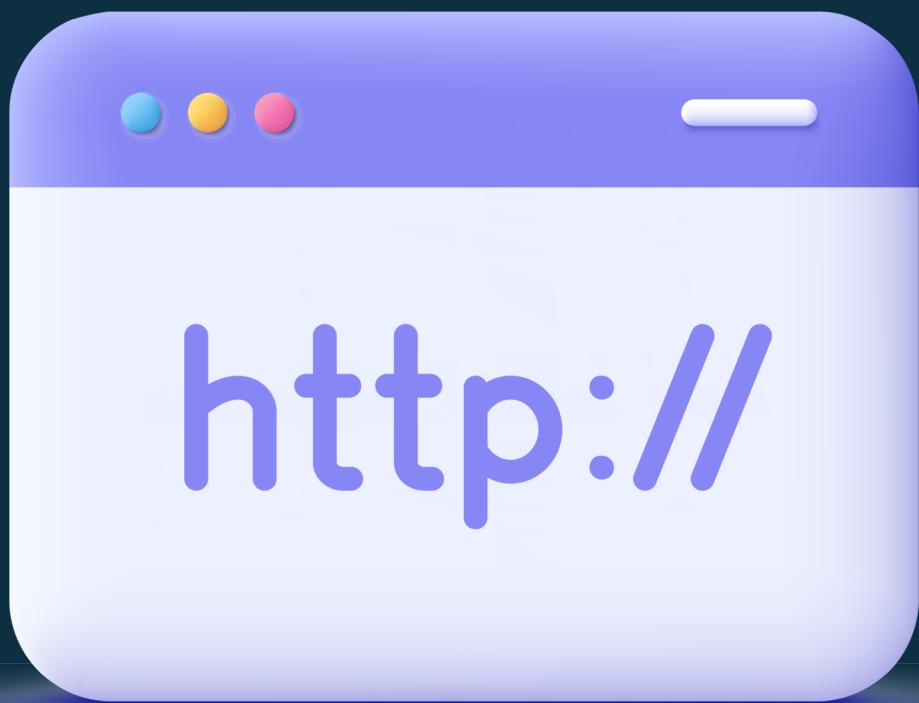


Electron Security



allowRunningInsecureContent

NOP Team



allowRunningInsecureContent

Ox01 简介

大家好，今天和大家讨论的是 `allowRunningInsecureContent` 安全配置选项，这个选项非常容易理解，就是是否允许在 `HTTPS` 的网站加载或执行 `HTTP` 协议的 `JavaScript`、`CSS`、插件等

从 `Electron 2.0.0` 开始默认为 `false`，即不允许在 `HTTPS` 网站中加载或执行 `HTTP` 协议的内容

当 `webSecurity` 被设置为 `false` 时，会自动将 `allowRunningInsecureContent` 设置为 `true`

Ox02 扩展探索

既然不允许 `HTTPS` 的网站加载 `HTTP`，是否会允许 `loadFile` 加载本地文件创建窗口加载 `HTTP` 的资源呢？

1. 正常功能测试

我们得找一个 `HTTPS` 的网站，并且还在网页内引用了 `HTTP` 的 `JavaScript` 等，搭建起来会比较麻烦，因为要有有效证书，还有配置同源策略等，我也不希望自签名证书通过自定义配置通过后再影响实验，因此通过 `fofa` 平台进行搜索

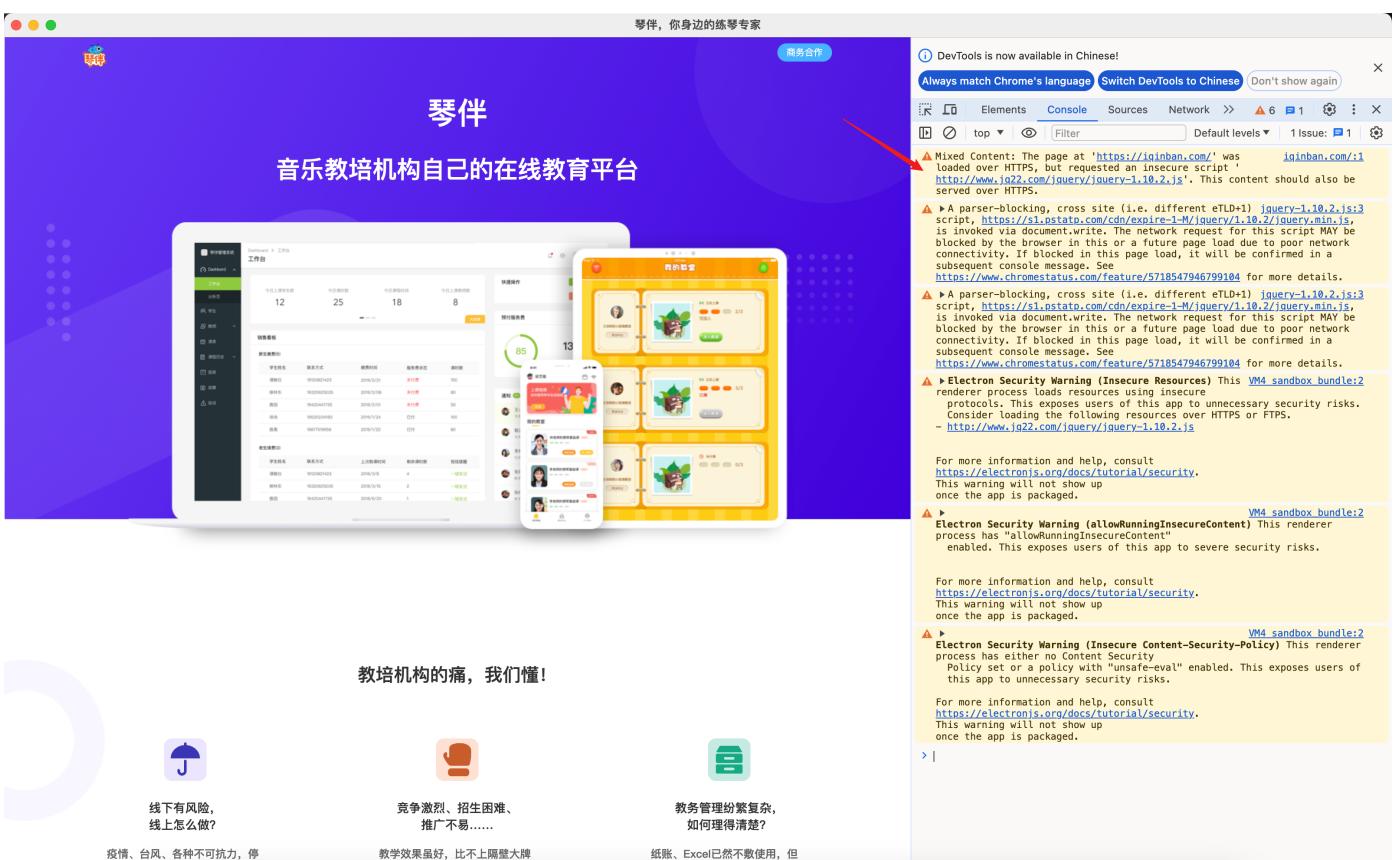
```
body=<script src=\"http://\" && port=\"443\" && country=\"CN\"
```

FQFA

body="



在 Electron 31.0.0.alpha.2 中是会拦截的，设置 allowRunningInsecureContent 为 true 后，再次加载



这回就可以加载了，我们尝试在各个版本的 Electron 中进行尝试，在 Electron 5.0、10.0、20.0、30.0 版本中执行结果保持一致

2. 本地加载文件测试

尝试在通过加载本地文件创建窗口情况下加载 HTTP 资源

搭建 XSS 服务器

```
[~/D/t/21 >>> cat remote.js
alert('remote xss is running')
[~/D/t/21 >>> sudo python3 -m http.server 80
>Password:
Serving HTTP on :: port 80 (http://[::]:80/) ...
```

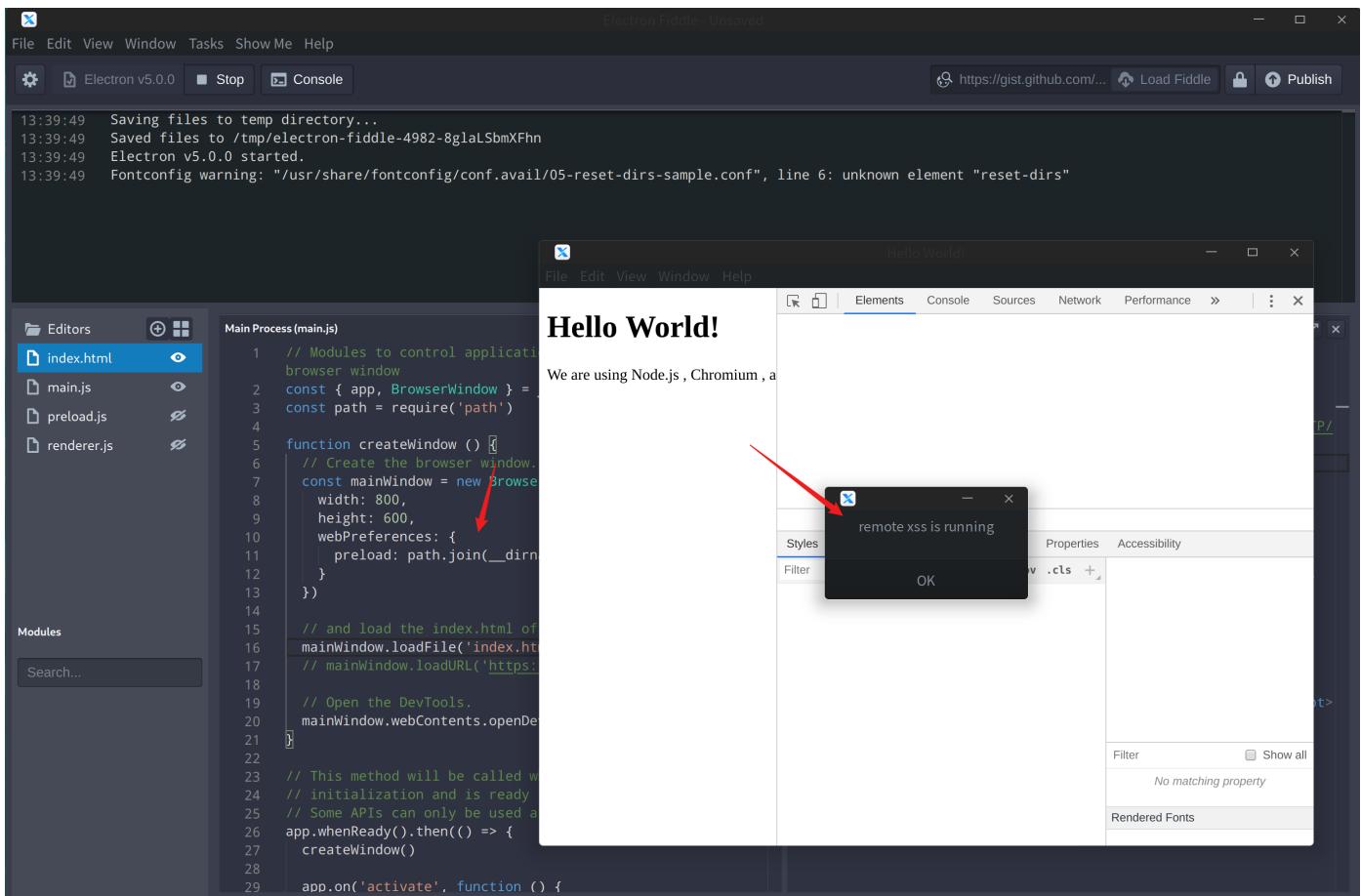
关闭 `csp`，在 `index.html` 中远程加载 `remote.js`

The screenshot shows the Electron Fiddle interface. On the left, the 'Main Process (main.js)' editor displays the following code:

```
1 // Modules to control application life and create native
2 // browser window
3 const { app, BrowserWindow } = require('electron')
4 const path = require('path')
5
6 function createWindow () {
7   // Create the browser window.
8   const mainWindow = new BrowserWindow({
9     width: 800,
10    height: 600,
11    webPreferences: {
12      preload: path.join(__dirname, 'preload.js')
13    }
14  })
15
16  // and load the index.html of the app.
17  mainWindow.loadFile('index.html')
18  // mainWindow.loadURL('https://iqinban.com/')
19
20  // Open the DevTools.
21  mainWindow.webContents.openDevTools()
22
23 // This method will be called when Electron has finished
24 // initialization and is ready to create browser windows.
25 // Some APIs can only be used after this event occurs.
26 app.whenReady().then(() => {
27   createWindow()
28
29   app.on('activate', function () {
30     // On macOS it's common to re-create a window in the
31     // app when the
32     // dock icon is clicked and there are no other windows
33     // open.
34     if (BrowserWindow.getAllWindows().length === 0)
35       createWindow()
36   })
37
38 // Quit when all windows are closed, except on macOS.
39 // There it's common
```

On the right, the 'HTML(index.html)' editor displays the generated HTML content:

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="UTF-8">
5     <!-- https://developer.mozilla.org/en-US/docs/Web/HTTP/
CSP -->
6     <!-- <meta http-equiv="Content-Security-Policy"
contents="default-src 'self'; script-src 'self'" -->
7     <title>Hello World!</title>
8   </head>
9   <body>
10    <h1>Hello World!</h1>
11    We are using Node.js <span id="node-version"></span>,
12    Chromium <span id="chrome-version"></span>,
13    and Electron <span id="electron-version"></span>.
14
15    <!-- You can also require other files to run in this
process -->
16    <script src="./renderer.js"></script>
17    <script src="http://192.168.31.216/remote.js"></script>
18  </body>
19</html>
```



发现即使在默认关闭了 `allowRunningInsecureContent` 的情况下，还是可以远程加载 `HTTP` 的 `JavaScript` 的

所以这方面还是得看 `CSP` 的，`CSP` 更牛一些

0x03 总结

`allowRunningInsecureContent` 仅在通过 `loadURL` 等远程加载网站创建窗口的时候有意义，对于通过 `loadFile` 加载本地文件的场景是没有作用的，同时 `Electron` 也没有变态到默认所有的远程加载内容（包括页面内 `img` 等元素的 `src` 属性指定的内容）必须都是 `HTTPS`

