Quasar: Compromising Electron Apps

The rare app mass ejection. Here we see Teams getting yeeted at the speed of light. 7 minute read Published: 2022-09-06

This is the story of how I used Microsoft Teams's own design against itself.

We all *kinda* know that Electron apps are dangerous—at least to our RAM, am I right??

But seriously, these cross-platform apps, because of how they get installed, present a tasty spot for attackers to take up residence and even inject malicious code into trusted applications, with the poor user being none the wiser.

Here's how it works.

Update: 9/7/22. Samuel Attard from the Electron project informed me that Electron currently has integrity checking as an experimental feature for macOS, and other platforms will hopefully be supported soon. I missed this in my research into the issue, and I apologize for any mischaracterization of the Electron team's work.

Research Objectives

In truth, my initial plan for this bit of research was not in fact Electron apps in general—rather, my target was Microsoft Teams, looking for any soft spots in this ubiquitous application that could present a risk to an organization like mine. Not all of what I discovered will be disclosed here, but the Electron-centric aspects of the research ended up being broadly applicable to all Electron apps.

The vulnerabilities discussed here result from 3 design decisions in Electron and apps developed with it: installation to user-writeable folders, easily unpackable application files, and lack of integrity checking of those application files.

ASAR Files

So you're running an Electron app. Things are humming along smoothly, but do you know what's going on under the hood?

Generally speaking, Electron apps are ad-hoc Chromium browsers loading web content to produce the user interface. But these are not static files, oh no. Electron includes an implementation of the Node runtime, meaning that each Electron app has a server/client relationship between the app runner and the interface. This usage of Node will come into play later, but for now let's focus on those user-facing files.

You might think these files are kept flat in some normal folder like `C:\Program Files\Microsoft Teams` or somesuch, right?

Insanely, Electron apps are installed to user-writeable directories. On Windows, this means `%LOCALAPPDATA%`, better known as `C:\Users\<username>\Appdata\Local`. The exact location will differ from app to app, but eventually you'll land on a file called `app.asar`.

`asar` is Electron's special file format to package up application files. What we're dealing with here is a glorified 0-compression tar file. During the run of an Electron app, this file is queried *constantly* for all the resources the app needs to function.

Process Monitor showing constant `ReadFile` events to the `app.asar` file from `Teams.exe`

Reading ASAR files is a lot easier than I thought it was going to be. In addition to Electron's first-party `npm` module linked above, there exists a plugin for 7-Zip! By installing this plugin it's possible to pack/unpack ASAR files, but you can directly edit the internal contents.

This made for some fun testing, in which I discovered multiple areas to explore. Perhaps the easiest, of course, is `main.bundle.js`.

Extracted ASAR file showing the app JavaScript

A quick note about ASAR and its weirdness. You might have noticed an `app.asar.unpacked` folder in addition to the `app.asar` file in the earlier screenshot. ASAR files have sort of a split identity between the components that are packed in the archive and the components that are not. Mostly this appears to be to accommodate the `node_modules` which are installed after packaging and not included in the `app.asar` proper. Nevertheless, both the file and the accompanying unpacked folder are required in the same directory for working with the ASAR format.

Electron JavaScript

So what can we do with that `main.bundle.js`? Here's the thing about this file: it's "server-side," which means it has access to Node APIs. Including things like, I dunno, `child_process`. Just as a for instance.

Adding calc

Popping calc

Other Files

Now of course this is an easy way in for local code execution and persistence. However, wouldn't it be neat if we could compromise the UI as well?

We can!

Teams specifically has a number of HTML files readily available in the source that are editable straight away. Here's what happens when I add a little `alert()` action to `oops.html`, along with a modified `Content-Security-Policy` via the file's `<meta>` tag:

XSS in Teams!

What an attacker might do with the ability to guide user input via custom HTML/JS, I leave to your imagination.

QuASAR: An ASAR Manipulation Tool

Now this research focused on Teams, but as I discovered that *all* Electron apps utilize this ASAR format. To make demonstrating the risk a bit easier, I wrote up a lil JS and called it QuASAR.

QuASAR is a simple utility that will analyze ASAR file discover injectable files, and allow you to inject whatever code execution you like via `child_process`. This is **NOT** intended to be a red team tool that you use on engagements. I ain't out here making weapons. Instead, this is designed to demonstrate the risk and help Defenders examine this behavior to better detect and prevent it.

Usage

```
quasar [options]

Options:
  -i, --input <inputFile>  asar file to mutate (default: "app.asar")
  -c, --command <command>  command to insert (default: "calc.exe")
  -w --write               write evil files directly to application dir
  -h, --help               display help for command
```

`quasar` requires a `.asar` file as a target. It can either be located elsewhere on the filesystem or, as is default, an `app.asar` file local to the current directory.

You will be presented with a list of injectable `.js` files in the archive. Select one by number, and the command provided by `-c` will be injected.

Without `-w`, the resulting `app.asar` and `app.asar.unpacked` will be created in a new `evil` directory within the current directory. However, if `-w` is provided, the ASAR files will be written back to the original path, and the original files will have `.bak` appended to their filenames.

quASAR in Action

Mitigation/Detection

A long-standing-issue on the Electron project indicates that one mitigation—cryptopgraphic signing of the ASAR files—is not being considered.

Similarly, there seems to be little interest in the types of integrity checking that Chromium-based browsers implement to protect extensions.

For detections, I would consider adding detections for `cmd.exe` and `powershell.exe` of common Electron extensions in your environment. Similarly, you may wish to identify what updaters (like `Squirrel.exe` for Teams) actually *should* be making changes to your respective `app.asars`, and alert on anything else doing so. These detections are, in my experience, quite high fidelity given the insular nature of Electron app code.

Prior work

Beemka uses a similar technique, albeit a little less modularly and focuses on front-end code injection.

Responsible Disclosure

The findings in this writeup were reported to Microsoft and marked as "intended functionality." Put another way, this is how Electron apps are supposed to work. There is plenty of folk wisdom about this potential attack path, but neither the Electron project nor developers using it seem keen to add any mitigations against it. Therefore, this research is published in the hope that it will motivate developers to take this issue seriously—and if not, at least defenders are armed with the knowledge of how to detect this attack path.

---

in writeups and tagged redteam , webapp and microsoft