Development Security

Whether you like it or not, today's applications live in operating systems that can be (and regularly are) compromised by any number of attacks. When your insecure application is a gateway for such lateral movement into the operating system, you are contributing to the tools that professional hackers have at their disposal. Don't be a tool.

This is why we have taken every opportunity to help you secure your application, prevent undesired access to system-level interfaces, and manufacture bullet-proof applications. Your users assume you are following best practices. We make that easy, but you should still be aware of the information below.

Security Is A Community Responsibility

It is important to remember that the security of your Tauri application is the sum of the overall security of Tauri itself, all Rust and npm dependencies, your code, and the devices that run the final application. The Tauri Team does their best to do their part, the security community does its part, and you too should follow a few important best practices.

Keep Your Application Up-To-Date

When releasing your app into the wild, you are also shipping a bundle that has Tauri in it. Vulnerabilities affecting Tauri may impact the security of your application. By updating Tauri to the latest version, you ensure that critical vulnerabilities are already patched and cannot be exploited in your application. Also be sure to keep your compiler (rustc) and transpilers (nodejs) up to date, because there are often security issues that are resolved.

Evaluate Your Dependencies

While npm and Crates.io provide many convenient packages, it is your responsibility to choose trustworthy third-party libraries - or rewrite them in Rust. If you do use outdated libraries which are affected by known vulnerabilities or are unmaintained, your application security and good night's sleep could be in jeopardy. Use tooling like npm audit and cargo audit to automate this process, and lean on the security community's important work.

Adopt More Secure Coding Practices

The first line of defense for your application is your own code. Although Tauri can protect you from common web vulnerabilities, such as Cross-Site Scripting based Remote Code Execution, improper configurations can have a security impact. Even if this were not the case, it is highly recommended to adopt secure software development best practices and perform security testing. We detail what this means in the next section.

Educate Your Users

True security means that unexpected behaviour cannot happen. So in a sense, being more secure means having the peace of mind of knowing that ONLY those things that you want to happen can happen. In the real world, though, this is a utopian "dream". However, by removing as many vectors as possible and building on a solid foundation, your choice of Tauri is a signal to your users that you care about them, their safety, and their devices.

Threat Models

Tauri applications are composed of many pieces at different points of the lifecycle. Here we describe classical threats and what you SHOULD do about them.

Upstream Threats

Tauri is a direct dependency on your project, and we maintain strict authorial control of commits, reviews, pull requests, and releases. We do our best to maintain up-to-date dependencies and take action to either update or fork and fix. Other projects may not be so well maintained, and may not even have ever been audited. Please consider their health when integrating them, otherwise, you may have adopted architectural debt without even knowing it.

Development Threats

We assume that you, the developer, care for your development environment. It is on you to make sure that your operating system, build toolchains, and associated dependencies are kept up to date.

A genuine risk all of us face is what is known as "supply-chain attacks", which are usually considered to be attacks on direct dependencies of your project. However, a growing class of attacks in the wild directly target development machines, and you would be well off to address this head-on.

One practice that we highly recommend, is to only ever consume critical dependencies from git using hash revisions at best or named tags as second best. This holds for Rust as well as the Node ecosystem. Also, consider requiring all contributors to sign their commits and protect Git branches and pipelines.

Buildtime Threats

Modern organisations use CI/CD to manufacture binary artifacts. At Tauri, we even provide a GitHub Workflow for building on multiple platforms. If you create your own CI/CD and depend on third-party tooling, be wary of actions whose versions you have not explicitly pinned.

You should sign your binaries for the platform you are shipping to, and while this can be complicated and somewhat costly to set up, end users expect that your app is verifiably from you.

Runtime Threats

We assume the webview is insecure, which has led Tauri to implement several protections regarding webview access to system APIs in the context of loading untrusted userland content.

You can read more in detail below, but using the CSP will lockdown types of communication that the Webview can undertake. Furthermore, Context Isolation prevents untrusted content or scripts from accessing the API within the Webview.

And please, whatever you do, **DO NOT** trust the results of cryptography using private keys in the Webview. Rust is there for a reason.

Updater Threats

We have done our best to make shipping hot updates to the app as straightforward and secure as possible. However, all bets are off if you lose control of the manifest server, the build server, or the binary hosting service. If you build your own system, consult a professional OPS architect and build it properly.

Secure Content Loading

Tauri restricts the Content Security Policy (CSP) of your HTML pages. Local scripts are hashed, styles and external scripts are referenced using a cryptographic nonce, which prevents unallowed content from being loaded.

> **DANGER**
>
> Avoid loading remote content such as scripts served over a CDN as they introduce an attack vector. But any untrusted file can introduce new and subtle attack vectors.

The CSP protection is only enabled if `[tauri > security > csp]` is set on the Tauri configuration file. You should make it as restricted as possible, only allowing the webview to load assets from hosts you trust, and preferably own. At compile time, Tauri appends its nonces and hashes to the relevant CSP attributes automatically, so you only need to worry about what is unique to your application.

See `script-src`, `style-src` and CSP Sources for more information about this protection.