@quasar/tauri - v1.0.0-alpha.0

https://github.com/quasarframework/tauri

TAURI

Create tiny, fast and secure cross-platform native apps with the ease of VueJS and the power of the Quasar Framework

Secure by Design

This guide seeks to explain the high level concepts and Security Features at the core of Tauri's design that make you, your apps and your users safer by default.

Please Note
While we take every opportunity to help you harden your application - there are always underlying threats like BIOS attacks, memory rowhammering and other operating system vulnerabilities that are constantly being discovered and (in the best cases) responsibly disclosed.

Furthermore, there are many ways that development teams can cut corners and either leak sensitive information or leave doors wide open to any of a range of attacks. Security is a never-ending quest, and your users count on you to keep them safe.

Therefore, we highly recommend that you take some time to consider the security ramifications of everything that your application does, especially in the context of running on the semi-hostile platform of end-user devices.

If you need help or want a review, you are welcome to contact the Quasar team for security consultation.

Security Researchers
If you feel that there is a security concern or issue with anything in Quasar or Tauri, please do not publicly comment on your findings. Instead, reach out directly to our security team:

security@quasar.dev

Although we do not currently have a budget for Security Bounties, in some cases we will consider rewarding responsible disclosure.

dAoT

dynamic Ahead of Time Compilation

This process of compilation happens several times during the build phase of a Tauri app. By using a dynamic Ahead of Time compiler, you can generate code references that are unique for every session and still technically static code units.

OTP

One Time Pad Tokenization and Hashing

Hashing important messages with a OTP salt, you are able to encrypt messages between the UI and the Rust backend.

fASLR

functional Address Space Layout Randomization

fASLR techniques randomize function names at runtime and implement optional OTP hashing so no two sessions are ever the same. We propose a novel type of function naming at boot time and optionally after every execution (KFI). Using a UID for each function prevents static attacks.

KFI

Kamikaze Function Injection

This advanced type of fASLR is a function (with randomized handle) that Rust inserts at runtime into the Webview, where its interface is locked within the promise resolution handler and is nulled after execution.

API Whitelisting

Shake Dangerous Functions out of the Codebase

You have the ability to pick and choose which API functions are available to the UI and to Rust. If they are not enabled, the code will not be shipped with your app, which reduces binary size and attack surface.

Bridge, don't Serve

Message Passing instead of serving via localhost

Instead of passing potentially unsafe functions, the bridge is used to pass messages and commands to named brokers at each respective side of the bridge. Most of the time you don't NEED a local server, and its inclusion opens security gaps in the final application.

CSP

Content Security Policy Management

Preventing unauthorized code execution for websites has long since been "resolved" by using CSPs. Tauri can inject CSPs into the index.html of the user interface, and when using a localhost server it will also send these headers to the UI or any other clients that connect with it.

Tauri-Frida

Post-Binary Analysis

Use industrial-grade pentester-tooling to discover and fix security weaknesses in your final binaries - before you ship.

AntiVuln Integration

Realtime Security Auditing Made Easy

With the forthcoming AntiVuln integration, you will have realtime threat notification and in-place patching for Node Modules and Rust Crates optionally baked into your development and build workflows.

Binary Format

Bundled Tauri Apps are true Binaries

This means that your apps cannot be easily decompiled as is the case with Electron ASAR files, which makes the process of reverse engineering your project much more time intensive and requires specialist training.