Security Policy

Workflow for Resolving Critical Vulnerabilities

The maintainers of MapLibre Native are committed to a fast and efficient resolution of critical security vulnerabilities. We aim to get back to you within 24 hours of creating the report. However, we cannot guarantee this. Luckily, vulnerabilities can be reported, fixed, merged, and released by anyone with write access (committers).

1. `[Reporter]` To report a critical security vulnerability in MapLibre Native, create a security advisory.
2. `[Reporter or Maintainer]` A private fork will be created. You can add collaborators that you believe will be able to help work on a fix.
3. `[Reporter, Collaborator or Maintainer]` The `main` branch may contain breaking changes or other changes that need to be tested before they can be released. That is why you should create a branch for each affected platform, where you check out the tag of the latest affected version. For example:

   ```
   git checkout ios-v5.13.0
   git branch -b ios-fix
   git checkout android-v10.2.0
   git branch -b android-fix
   ```

4. `[Committer]` Once a fix has been implemented for a platform, a branch on `maplibre/maplibre-native` will be created in preparation of a release. For example, `ios-v5.13.0` (tag) -> `ios-5.13.1` (new branch).
5. `[Reporter, Collaborator or Maintainer]` A PR is created from the fork that targets the new branch on `maplibre/maplibre-native`. It is reviewed and merged.
6. `[Committer]` A new release is pushed out from the branch that now contains the fix. The release process is automated and documented and can be done by anyone with write access.
   - Android: platform/android/RELEASE.md
   - iOS: platform/ios/RELEASE.md
7. `[Maintainer]` The security advisory is published.
8. `[Reporter or Maintainer]` Another PR is created that applies the changes to the `main` branch. This step is important so new releases don't inadvertently re-introduce the same security vulnerability.

Mitigation

We are actively trying to prevent security incidents using the following methods:

- Static analysis with the use of clang-tidy.
- C++ code scanning with CodeQL.

MapLibre Native relies on several external open-source libraries. We currently do not monitor these dependencies automatically for vulnerabilities.

If you have any suggestions how to improve our security mitigation strategies, feel free to open an issue or start a Discussion. Be sure to check out the open issues tagged with the security label.