Engineer Software

Wrote a parser verifying every FFI call

Wrote a parser that reads the real 7,308-line C header and verifies every call site, every enum constant and that every pointer-owning class is final, after a hand-written placeholder header let calls to three removed functions compile, link and crash.

September 2026

7,308

lines of C header, read rather than assumed

Situation. Early in the project the C interface was represented by a hand-written header describing the functions the application expected. That header compiled, the application linked, and calls to three functions that no longer existed in the core reached the point of being called and crashed. The compiler and the linker had both been satisfied by a description of the library rather than the library, and the gap only appeared at runtime.

Task. The real header — 7,308 lines and 268 declarations — had to become the authority, and every use of it in the Swift code had to be verified against it automatically rather than by review.

Action. The check is a parser that reads the actual upstream header and builds the set of functions, enum constants and types it declares, then reads the Swift source and resolves every call site and every constant reference against that set. A call to a function the header does not declare fails the build. A reference to an enum constant that has been renamed fails the build. The current count is 132 of the 268 declarations referenced, and knowing which 136 are unused is itself useful, because it says exactly how much of the core the client has not reached. The parser also enforces a rule the compiler cannot: every Swift class that owns a pointer into the core must be final. A non-final pointer-owning class can be subclassed, and a subclass that overrides deinitialisation or adds its own lifetime changes when the pointer is freed — a use-after-free with no unsafe keyword anywhere near it. The rule is checked by name across the whole tree.

Result. The class of defect that produced the original crash cannot recur, because a stale reference is now a build failure rather than a runtime one. The limitation is that the parser understands the header's declarations and not its semantics: it proves a function exists with a matching name, and a function whose meaning or ownership rule changed upstream while keeping its signature passes without comment.

Bring this experience to your project

Use "Wrote a parser verifying every FFI call" as a starting point for a similar challenge in your organisation. The services behind this work are available for your project. The part users never see: where the data lives, what the rules are, and how other systems are allowed to talk to it.

Explore our services Get in touch →

Look behind the result

Read the situation, decisions and result in this example. We can discuss which parts transfer to your constraints and which need another approach. Wrote a parser verifying every FFI call — 7,308 lines of C header, read rather than assumed

Explore our services Get in touch →

Would your build catch a call to a function that is gone?

Talk through your project with the engineer responsible for delivery. We agree the scope, price basis and handover in writing before work begins.

Role context: macOS Application Engineer. It draws on Backend Engineering, APIs & Integration, Automation & CI/CD, Security, Testing & QA and Documentation. And while you are here, we are rather good at Backend & API Development, DevOps & CI/CD Automation and Technical Documentation.

Get in touch Explore our services

One piece of **a longer record** — and it is all on this site.

Every entry is written the same way: the situation, the task, what we did and what changed.

See the whole record

Define the improvement you need

Describe the workflow, the existing interfaces and the users who need the change. We can define a testable delivery and a handover that lets your team maintain it.

See what it costs Get in touch →

Would your build catch a call to a function that is gone?

Ask about the decisions behind “Wrote a parser verifying every FFI call”. You will speak with the people who can explain the work and assess your own project.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://software.engineer.company/portfolio/wrote-a-parser-verifying-every-ffi-call/