Analyzing Swift Apps with Swift-Frida and r2

Malte Kraus

September 8, 2018

Swift

Swift

- created by Apple
- compiled to native code
- interop with Objective C
- no stable ABI (coming soon?)
- ABI documentated in C++ (and Markdown, but ...)

First look at Swift bins

Frida, frida-swift, swift-frida, r2frida

Can we get Reflection Data in Frida?

Available Type Metadata

- Type Metadata
  - Variables of Protocol/Class types carry pointers to Metadata of their dynamic type
  - Describe type definitions + memory layout for use by Swift runtime, LLDB Debugger
  - need to run code from the binary to initialize $\Rightarrow$ need dynamic analysis $\Rightarrow$ Frida
- Reflection Metadata
  - Seems mostly redundant to type metadata
  - Available fully statically (`swift-reflection-dump`)
  - no info about memory layout (?)
  - Not really looked into it too deep (project started out with Frida)

JavaScript Access to Type Metadata in Frida

- Each kind of type (class, struct, tuple, ...) stores different data: can recover memory layout, nearly complete type declaration
- Manually translated C++ class definitions for type metadata to 1600 lines of JavaScript
- higher level JavaScript objects to wrap these classes: JavaScript strings/numbers/arrays/functions instead of pointers

Determine Type Names

try runtime function `swift_getTypeName`, fallbacks:

1. tuples: concat names of tuple element types
2. functions: concat names of argument/return types, plus convention
3. ...
4. finally: use symbol information, if available (demangling via runtime function)

JavaScript Access to Swift Variables

- Value Witness Table: function pointers for copying, destroying, creating, ...values of a type
- Enum Value Witness Table: also r/w access to the tag and payload
- Compound types: member names, types, offsets are known ⇒ transparent access by their names, just like for normal JS objects
- Enums: `$enumCase`, `$enumPayloadCopy()`, `$setTo(case, payloadvalue)`
- primitive types: automatic conversion to JS equivalents
- Use type info object as constructor on `NativePointer` to get JavaScript wrapper for that Swift value

toString

- Swift stdlib has `dump(, to:)` function: recursively descends into members and prints member name and debug representation of value to output stream
- for `toString`, we call dump with the Swift variable and a `String` as output stream, then convert that to a C string, then that to a JavaScript string
- opposite direction: runtime function to initialize Swift `String` from C string

TODO

- only Swift 4.0.* on iOS supported for now
- no easy way to read/write function parameters for function hooks/calls
- can't locate type metadata for some private types
- no function calls
- member functions not associated with their types

- not yet part of `https://github.com/frida` project
- needs wider testing: try it and report bugs! `https://github.com/maltek/swift-frida`

End

Questions?

mailto:swift@maltekraus.de

Freenode: drivel

https://github.com/maltek

mailto:jobs@maltekraus.de

Swift Calling Convention

- based on C and C++ calling conventions for the platform. C++ calling convention for ARMv8 iOS: `this`, indirect results are passed in registers `x20`, `x8`
- error handling: no “zero-cost” exceptions – register `x21` is 0 on success or pointer to heap-allocated error value
- compound types are converted to use as few register space as possible – not yet implemented in Swift-Frida

Finding Type Metadata in Frida

- public types are referenced from `__swift2_types`
  `Process.enumerateModulesSync() -> getsectiondata()`
- types conforming to a protocol are referenced from `__swift2_proto` section
- libraries export symbols for type descriptors
- recursive search on members referenced from known type metadata
- automatically instantiate generic types with 0 type parameters
- special cases: `Any`, `AnyObject`, `AnyClass`, `Void` from runtime functions `swift_get*TypeMetadata`