Swift and Security

(not @SwiftOnSecurity)

OWASP Boston Chapter: December 3, 2014

Ming Chow
mchow@cs.tufts.edu
Twitter: @0xmchow

Disclaimer

We are all new to Apple’s Swift Programming Language.

Why Apple's Swift Programming Language

- Friendlier and gentler than Objective-C…
- ...therefore, I predict there will be an influx of new developers…
- ...who may be programming for the first time…
- ...who will be clueless about the idea of security

What's the Point of this Presentation?

Whether you are a first time developer or seasoned mobile developer, we need to start the conversation about security issues related to a programming language or framework early on. You don't want history to repeat itself.

What Does Swift Give You in Terms of Security?

- “Hides” pointers
- Type safety and type checking
- Type inference
- “Values are never implicitly converted to another type” [1]
- Array bounds checking
- Playground for debugging

What Hasn’t Changed from Objective-C

- The usual gang: operators, operands, conditionals, syntax
- Automatic Reference Counting (ARC)
- No garbage collection

The Potentially Interesting

- Legacy support: seamless Cocoa and Objective-C compatibility including interaction with pointers
- A “compiled” programming language and an “interpreted” scripting language. [4]
- “`Int` can store any value between -2,147,483,648 and 2,147,483,647, and is large enough for many integer ranges.” [1]
- “Every string is composed of encoding-independent Unicode character” (including Emoji)

Access Control

1. `private` entities (e.g., classes) are available only from within the source file where they are defined.
2. `internal` (DEFAULT) entities are available to the entire module that includes the definition (e.g. an app or framework target).
3. `public` entities are intended for use as API, and can be accessed by any file that imports the module, e.g. as a framework used in several of your projects.

Source: [10]

Pointers in Swift

- Yes, you can interact with pointers in Swift!
  - In C: `const Type *`
    - In Swift: `UnsafePointer<Type>`
  - In C: `Type *`
    - In Swift: `UnsafeMutablePointer<Type>`
- Documentation: [11]

About Working with Pointers in Swift

Swift works hard to make interaction with C pointers convenient, because of their pervasiveness within Cocoa, while providing some level of safety. However, interaction with C pointers is inherently unsafe compared to your other Swift code, so care must be taken. In particular:

- *These conversions cannot safely be used if the callee saves the pointer value for use after it returns. The pointer that results from these conversions is only guaranteed to be valid for the duration of a call. Even if you pass the same variable, array, or string as multiple pointer arguments, you could receive a different pointer each time. An exception to this is global or static stored variables. You can safely use the address of a global variable as a persistent unique pointer value, e.g.: as a KVO context parameter.*
- ***Bounds checking is not enforced when a pointer to an Array or String is passed.** A C-based API can't grow the array or string, so you must ensure that the array or string is of the correct size before passing it over to the C-based API.* [12]

Format Strings

- The good news: `printf()` is no longer allowed (it was allowed in early summer). `println()` and `print()` with string interpolation are now used
- However, you can still have format strings in Swift via `NSString(format: …)`

What Swift Does NOT Offer --yet [2][10]

1. Taint checking for user inputs
2. Query parameterization (e.g., for Core Data)
3. Lint

What to Do Next?

- Generally speaking, the risks are a lot lower with Swift
- Refer to Apple’s “Introduction to Secure Coding Guide” [13]

References

1. “The Swift Programming Language” https://developer.apple.com/library/mac/documentation/Swift/Conceptual/Swift_Programming_Language/#//apple_ref/doc/uid/TP40014097-CH3-XID_0
2. http://www.safelightsecurity.com/swift-pros-and-cons
3. http://www.drdobbs.com/security/security-issues-in-swift-what-the-new-la/240168882
4. http://blog.erratasec.com/2014/06/why-it-had-to-be-swift.html
5. http://itsecurity.co.uk/2014/06/apples-new-swift-is-it-secure/
6. https://www.checkmarx.com/2014/08/20/swift-security-issues/
7. https://news.ycombinator.com/item?id=7837627
8. https://www.reddit.com/r/swift/comments/2efjdv/security_issues_in_swift_what_the_new_language/
9. http://teks.co.in/site/blog/14-reasons-apple-swift-quite-ideal-programming-language-yet/
10. https://developer.apple.com/swift/blog/?id=5
11. https://developer.apple.com/library/ios/documentation/Swift/Conceptual/BuildingCocoaApps/InteractingWithCAPIs.html
12. https://developer.apple.com/swift/blog/?id=6
13. https://developer.apple.com/library/mac/documentation/Security/Conceptual/SecureCodingGuide/Introduction.html