# Reverse Engineering Swift Apps

Michael Gianarakis

Rootcon X 2016

# # whoami

@mgianarakis

Director of SpiderLabs APAC at Trustwave

SecTalks Organiser (@SecTalks_BNE)

Flat Duck Justice Warrior #ducksec

Trustwave® SpiderLabs®

# Motivation

- Seeing more and more Swift being used in apps that we test (fan boys like me tend to adopt new Apple technology quickly)

- Google is even considering using Swift as a first class language on Android… (http://thenextweb.com/dd/2016/04/07/google-facebook-uber-swift/)

- Wanted to dive into some of the key differences with Swift and look at the challenges with respect to Swift app pen testing

- Focus is on “black box” app pen testing - for a deeper dive into Swift language RE I recommend Ryan Stortz’s talk at Infiltrate (http://infiltratecon.com/archives/swift_Ryan_Stortz.pdf)

Trustwave® SpiderLabs®

# How Does Swift Affect Testing?

- Will dive into the detail in the presentation but the reality is not much in most areas, quite a bit in others?

- Most issues in iOS and OS X apps are due to poor design decisions or misconfiguration and incorrect implementation of Apple and third party frameworks and libraries.

- The main thing that has changed is how you reverse engineer the application

Trustwave® SpiderLabs®

# Quick Overview of Swift

# What is Swift?

- Compiled language created by Apple
- Released publicly in 2014 at WWDC and has seen multiple revisions since.
- Open source with official implementations for iOS, OS X and Linux.
- Intended to replace Objective-C eventually

Trustwave® SpiderLabs®

# Syntax (just the basics to follow along)

```
// Variables and Constants

let constant = "immutable value"
var variable = "mutable value"

// Type Annotation

let constantWithType: String = "Swift infers types but can be explicit"
```

Trustwave® SpiderLabs®

# Syntax (just the basics to follow along)

```
class Duck {

    var duckType: String // Property
    var name: String // Property
    var owner: String = "Owner" // Property w/ default value

    // Initilisation
    init(duckType: String, name: String) {
        self.duckType = duckType
        self.name = name
    }

    // Class Methods
    class func quack() {
        print("Quack")
    }
```

Trustwave® SpiderLabs®

# Syntax (just the basics to follow along)

```
    // Instance Methods
    func printDuckType () {
        print("Your duck type is \(self.duckType)")
    }

    func changeOwner(newOwner: String) {
        self.owner = newOwner
    }

    func isDuckAtRootcon(duckName name: String) -> Bool {
        if name == "Xntrik" {
            return false
        } else {
            return true
        }
    }
}
```

Trustwave® SpiderLabs®

# Syntax (just the basics to follow along)

```
import Foundation

var flatDuck = Duck(duckType: "Flat", name: "Richo")
var uprightDuck = Duck(duckType: "Upright", name: "Xntrik")

// Calling class method

Duck.quack()

// Calling instance method

flatDuck.printDuckType()
flatDuck.changeOwner("Snare")
print(flatDuck.owner)

uprightDuck.printDuckType()
uprightDuck.isDuckAtRootcon(duckName: "Xntrik")
```

Trustwave® SpiderLabs®

# Types

- All basic C and Objective-C types -> String, Bool, Int , Float etc.
- Collection Types -> Array, Set, Dictionary
- Optional Types -> works with all types, no more nil pointers like Objective-C
- Swift is a type safe language

Trustwave® SpiderLabs®

# Objective-C Compatibility

- Objective-C compatibility and interoperability
    - Uses the same runtime environment
    - Still supports C and C++ in the same app but can't be called from Swift like Objective-C
    - Can allow for some dynamic features and runtime manipulation

Trustwave® SpiderLabs®

# Other Language Features

- Barely scratched the surface
    - Structs, Protocols, Extensions, Closures, Enumerations, Optionals, Generics, Type Casting, Access Control, Error Handling, Assertions….
    - Automatic Reference Counting
    - Unicode…
        - var 💩 = 💩💩💩💩💩()

Trustwave® SpiderLabs®

# Other Language Features

The Swift Programming Language

*Swift 2.2 Edition*

Trustwave® SpiderLabs®

# Challenges Reversing Swift Apps

# Challenges

- Less dynamic than Objective-C
    - Less flexible than Objective-C in some areas
    - Can make it harder to do some of the standard tasks you would do on a standard app pen test
    - Less of an issue now because most Swift apps will include be mixed with Objective-C
- Limited tooling
    - We will explore this in more detail

Trustwave® SpiderLabs®

# Challenges

- Rapidly evolving syntax, APIs and features and Apple doesn't care too much about breaking changes.
    - v1.0 - September 2014
    - v1.1 - October 2014
    - v1.2 - April 2015
    - v2.0 - September 2015 (Open Sourced, Linux)
    - v2.2 - March 2016
    - v3.0 - Late 2016

Trustwave® SpiderLabs®

# Reversing Swift Apps

- Two primary reverse engineering activities when conducting a “black box” pen test
  - Dumping and analysing class information from the binary
  - Retrieving information at runtime using debuggers, function hooking, tracing etc.

Trustwave® SpiderLabs®

# Retrieving Class Information

# Class Dump?

- The most common and easiest way to retrieve class data from an Objective-C binary is the class-dump utility

- class-dump-z retrieves class information and formats to look like the equivalent of an Objective-C header file

- Usually one of the first things you do when looking at an app

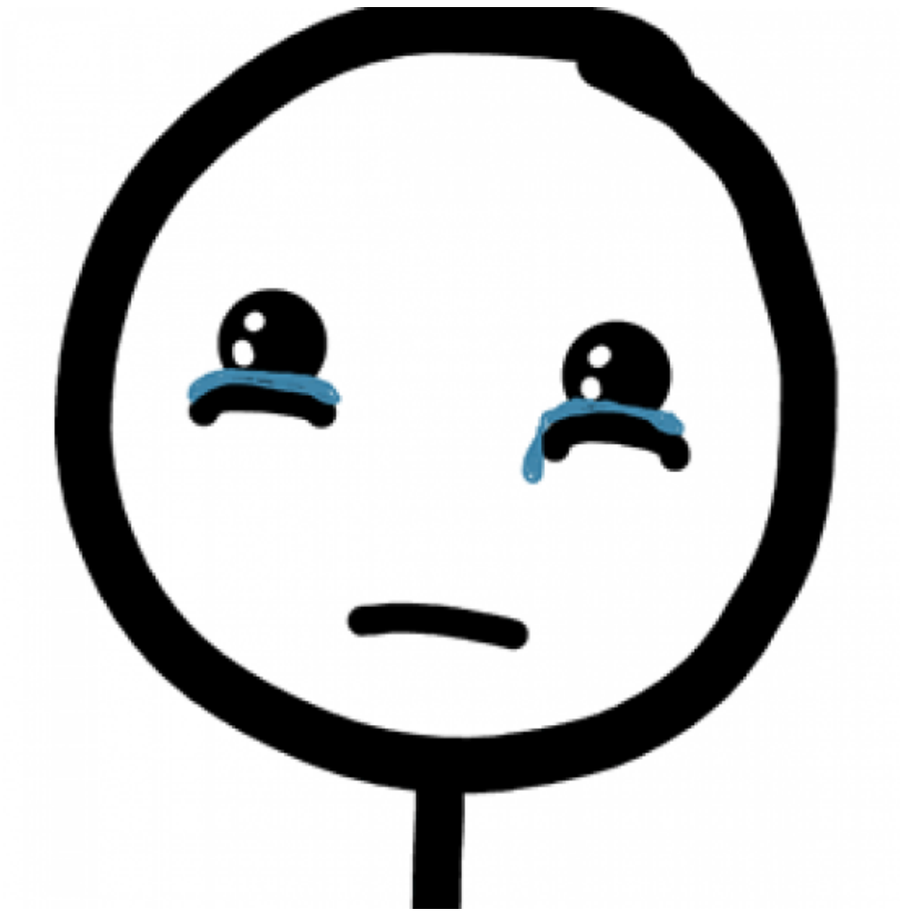
Trustwave® SpiderLabs®

# Class Dump?

```
@interface PTHOAuthHandler : NSObject
{
    NSMutableDictionary *_authDictionary;
}

+ (id)sharedController;
- (void).cxx_destruct;
- (void)handleOAuthURL:(id)arg1;
- (void)authenticate:(id)arg1 completion:(CDUnknownBlockType)arg2;
- (id)init;

@end
```

Trustwave® SpiderLabs®

# Class Dump?



Trustwave® SpiderLabs®

# What next?

- So class-dump-z doesn't work with Swift binaries :(
- Now what?
- Let's start diving into the binary

Trustwave® SpiderLabs®

# Symbol Table

- What do we get if we dump the symbol table?

```
[Rootcon] nm -gUj rootcon-demo | head -n 20
_NS_Swift_NSCoder_decodeObject
_NS_Swift_NSCoder_decodeObjectForKey
_NS_Swift_NSCoder_decodeObjectOfClassForKey
_NS_Swift_NSCoder_decodeObjectOfClassesForKey
_NS_Swift_NSKeyedUnarchiver_unarchiveObjectWithData
_NS_Swift_NSUndoManager_registerUndoWithTargetHandler
_OBJC_CLASS_$_SwiftObject
_OBJC_CLASS_$__SwiftNativeNSArrayBase
_OBJC_CLASS_$__SwiftNativeNSDictionaryBase
_OBJC_CLASS_$__SwiftNativeNSEnumeratorBase
_OBJC_CLASS_$__SwiftNativeNSError
_OBJC_CLASS_$__SwiftNativeNSSetBase
_OBJC_CLASS_$__SwiftNativeNSStringBase
_OBJC_CLASS_$__TtCs17_SwiftNativeNSSet
_OBJC_CLASS_$__TtCs18_EmptyArrayStorage
_OBJC_CLASS_$__TtCs19_NSContiguousString
_OBJC_CLASS_$__TtCs19_SwiftNativeNSArray
_OBJC_CLASS_$__TtCs20_SwiftNativeNSString
_OBJC_CLASS_$__TtCs21_SwiftDeferredNSArray
_OBJC_CLASS_$__TtCs24_ContiguousArrayStorage1
```

# Symbol Table

- What if we look for something we know is in the binary?

```
· nm -gUj rootcon-demo | grep printDuckType
```

Trustwave® SpiderLabs®

# Symbol Table

- What if we look for something we know is in the binary?

```
[Rootcon] nm -gUj rootcon-demo | grep printDuckType
__TFC12rootcon_demo4Duck13printDuckTypefT_T_
__TWoFC12rootcon_demo4Duck13printDuckTypefT_T_
```

Trustwave® SpiderLabs®

# Name Mangling

- Looks promising but it’s a far cry from the output of class-dump and is kind of hard to make out
- Swift stores metadata about a function in it’s symbols in the process “mangling” the name.

Trustwave® SpiderLabs®

# Name Mangling

```
__TFC12rootcon_demo4Duck13printDuckTypefT_T_
```

Indicates it's a Swift method

Trustwave® SpiderLabs®

# Name Mangling

```
__TFC12rootcon_demo4Duck13printDuckTypefT_T_
```

Indicates it's a Swift method

Indicates it's a function

Trustwave® SpiderLabs®

# Name Mangling

```
__TFC12rootcon_demo4Duck13printDuckTypefT_T_
```

Indicates it's a Swift method

Indicates it's a function

**Function of a class**

Trustwave® SpiderLabs®

# Name Mangling

```
__TFC12rootcon_demo4Duck13printDuckTypefT_T_
```

Indicates it's a Swift method

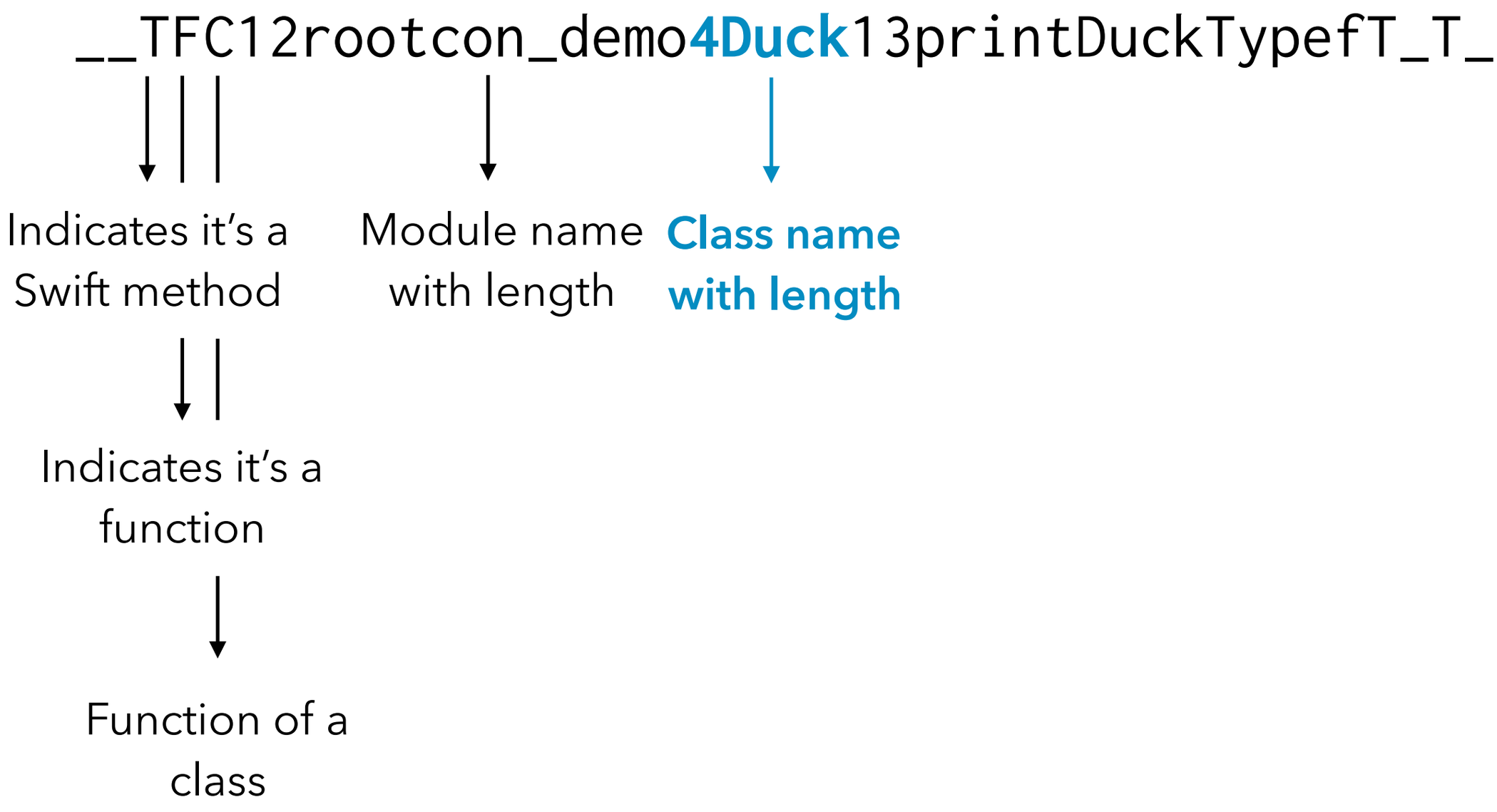
**Module name with length**

Indicates it's a function

Function of a class

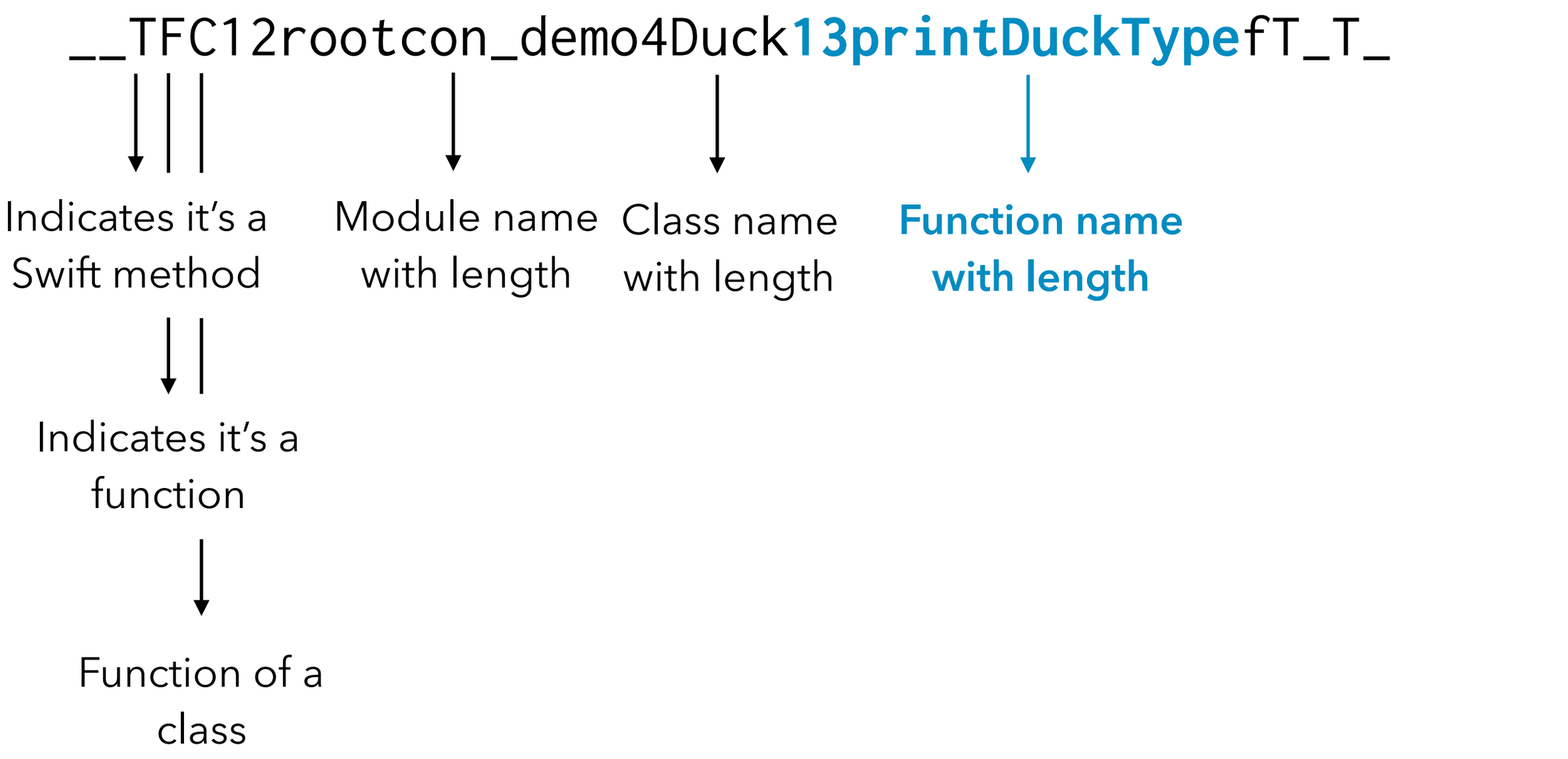
Trustwave® SpiderLabs®

# Name Mangling



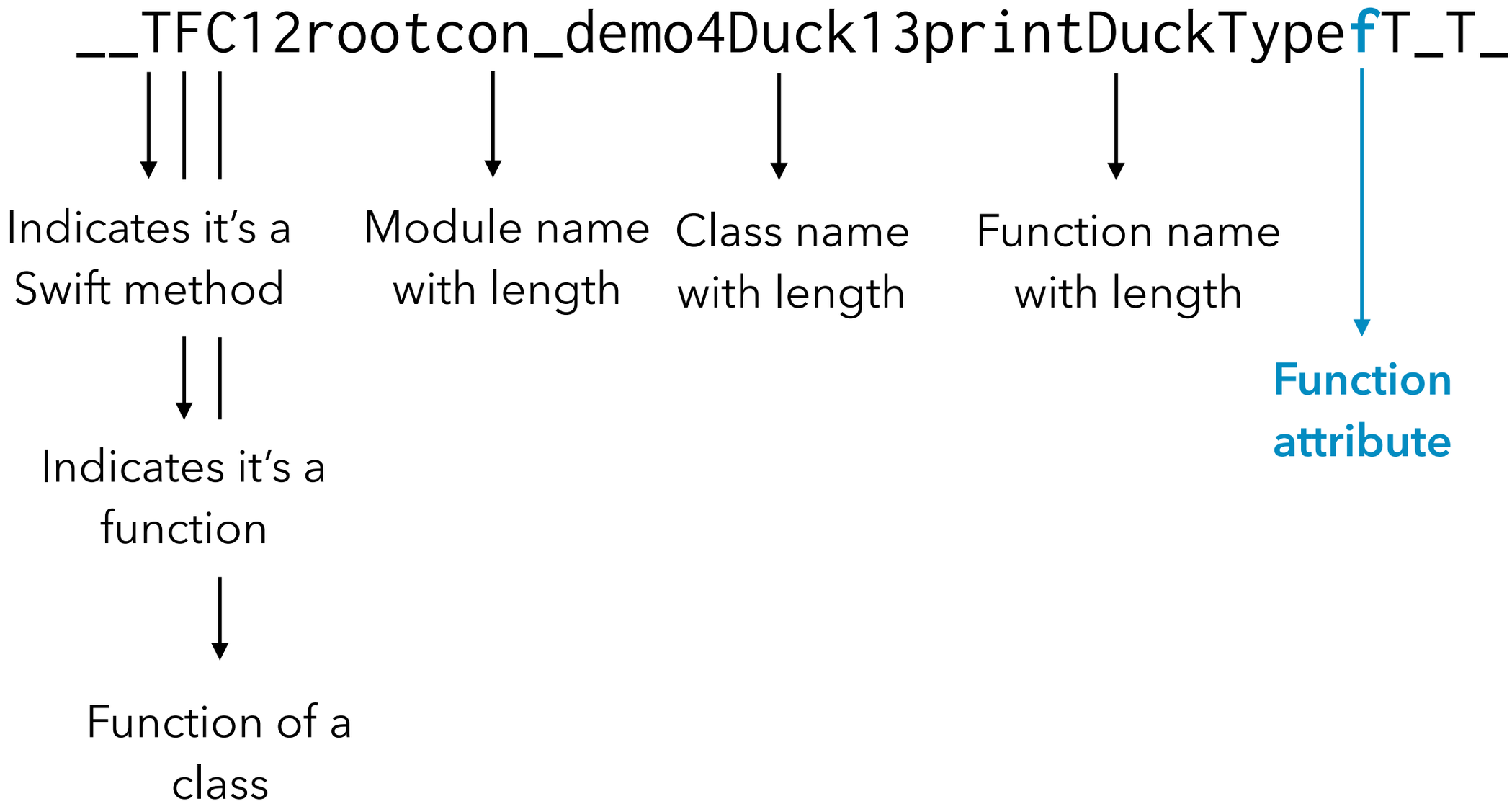

Trustwave® SpiderLabs®

# Name Mangling



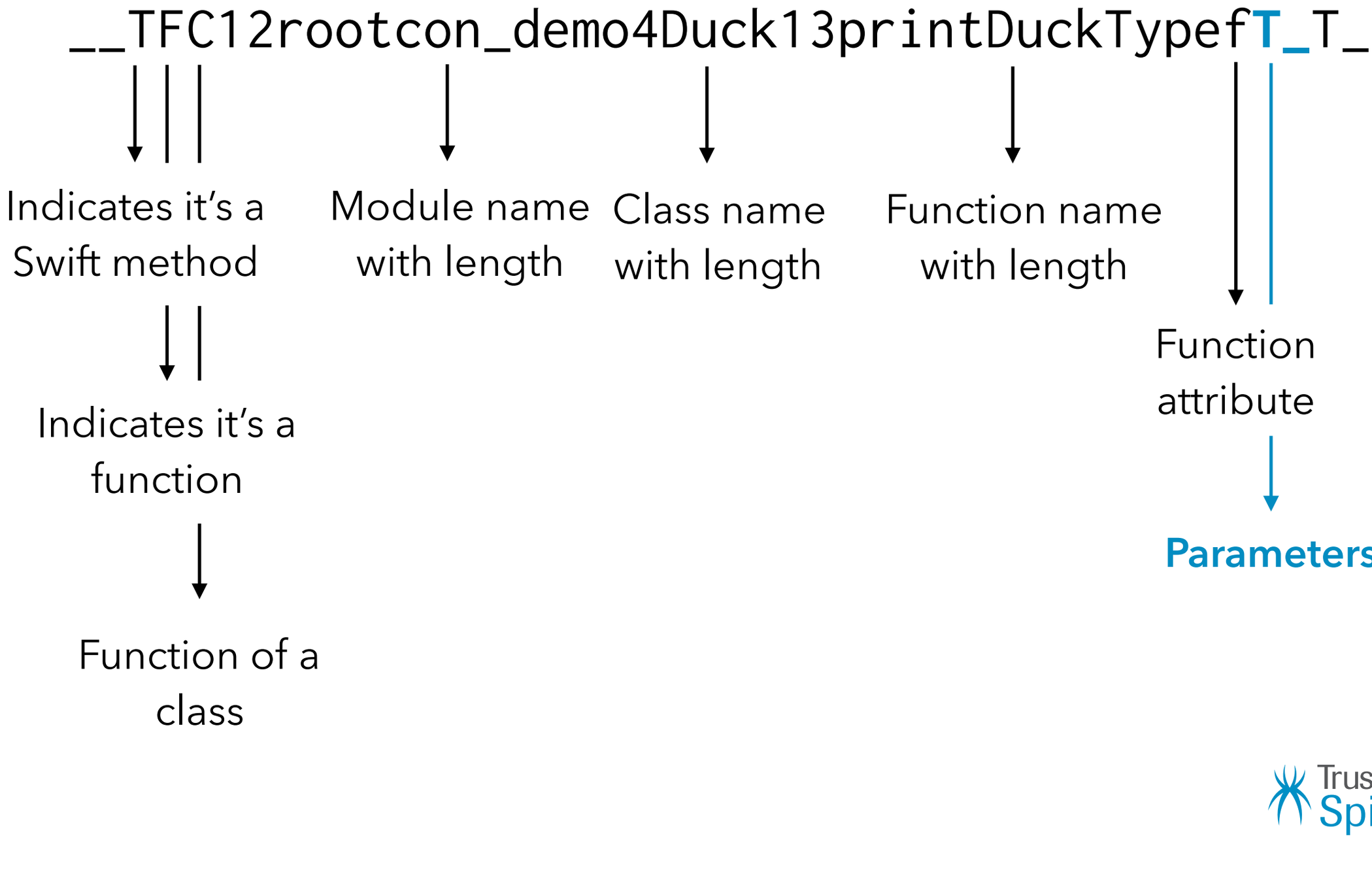

Trustwave® SpiderLabs®

# Name Mangling



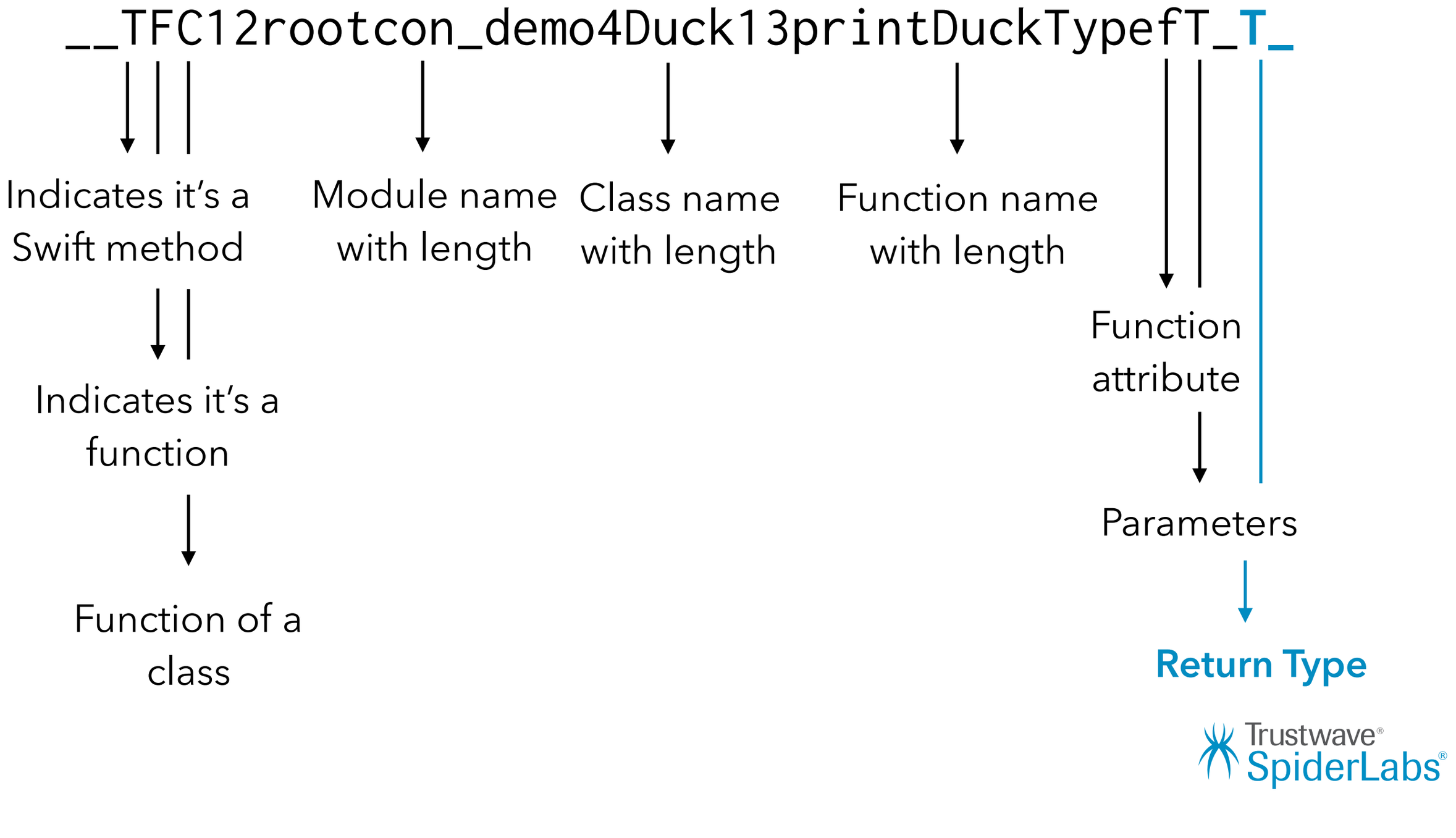

Trustwave® SpiderLabs®

# Name Mangling




Trustwave® SpiderLabs®

# Name Mangling




Trustwave® SpiderLabs®

# Function Attributes

| | |
|---|---|
| f | Normal function |
| s | Setter |
| g | Getter |
| d | Destructor |
| D | Deallocator |
| c | Constructor |
| C | Allocator |

Trustwave® SpiderLabs®

# Return Types

| | |
|---|---|
| a | Array |
| b | Boolean |
| c | Unicode Scalar |
| d | Double |
| f | Float |
| i | Integer |
| u | Unsigned Integer |
| Q | Implicitly Unwrapped Optional |
| S | String |

Trustwave® SpiderLabs®

# swift-demangle

- So now we know roughly the way the names are mangle you could use this to create a script that “de-mangles” the names

- Apple has already thought of that and includes a utility called swift-demangle to do just that

Trustwave® SpiderLabs®

# swift-demangle

```
[Rootcon] swift-demangle __TFC12rootcon_demo4Duck13printDuckTypefT_T_
_TFC12rootcon_demo4Duck13printDuckTypefT_T_ ---> rootcon_demo.Duck.printDuckType () -> ()
[Rootcon] swift-demangle -compact __TFC12rootcon_demo4Duck13printDuckTypefT_T_
rootcon_demo.Duck.printDuckType () -> ()
[Rootcon] swift-demangle -compact -simplified __TFC12rootcon_demo4Duck13printDuckTypefT_T_
Duck.printDuckType() -> ()
[Rootcon] swift-demangle -expand __TFC12rootcon_demo4Duck13printDuckTypefT_T_
Demangling for _TFC12rootcon_demo4Duck13printDuckTypefT_T_
kind=Global
  kind=Function
    kind=Class
      kind=Module, text="rootcon_demo"
      kind=Identifier, text="Duck"
    kind=Identifier, text="printDuckType"
    kind=Type
      kind=UncurriedFunctionType
        kind=ArgumentTuple
          kind=Type
            kind=NonVariadicTuple
        kind=ReturnType
          kind=Type
            kind=NonVariadicTuple
_TFC12rootcon_demo4Duck13printDuckTypefT_T_ ---> rootcon_demo.Duck.printDuckType () -> ()
```

Trustwave® SpiderLabs®

# swift-demangle

- With nm and swift-demangle and some shell scripting you should be able to easily grab the function signatures from an app
- Should be all you need to get basically the same information you would from class-dump to start assessing the app

Trustwave® SpiderLabs®

# class-dump-s

- Hacked together script that demangles names and formats the output to approximate the output of class-dump

- Written in Swift

Trustwave® SpiderLabs®

# Demo

# Stripped Binaries

- CAVEAT: If the developer stripped symbols from the binary then these techniques obviously won't work.

- Reverse engineering stripped binaries is a bit more complicated

Trustwave® SpiderLabs®

# Objective-C Compatibility

- Part of the reason it's much easier to get class information from Objective-C binaries is because it's necessary for the Objective-C runtime to have that info

- So what happens when you import Objective-C frameworks or use Objective-C in your app?

Trustwave® SpiderLabs®

# Revisiting Class Dump

- The latest branch of class-dump by Steven Nygard (the original class-dump utility) has limited support for Swift.

- Need to download and build from source (no binary release yet)

- https://github.com/nygard/class-dump

Trustwave® SpiderLabs®

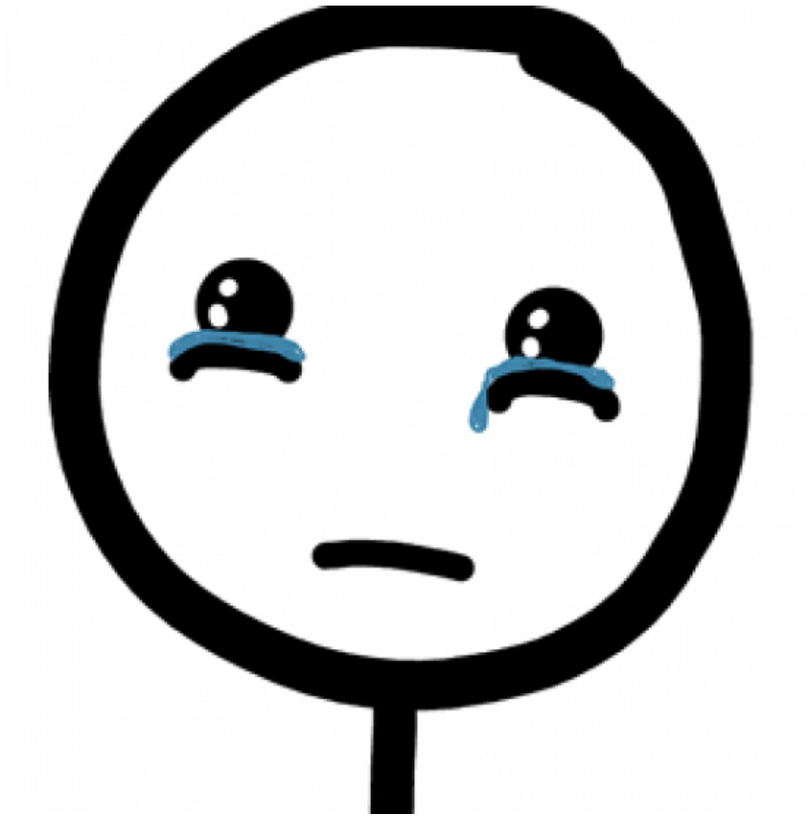
# Revisiting Class Dump

```
class HITB {

    var howGreatIsHITB = 7.5

    func isClassDumpGoingToWork(name: String) -> Bool {
        return false
    }

    func isClassDumpGoingtoWorkWithObjCRuntime(runtime name: String) -> Bool {
        if name == "ObjC" {
            return true
        } else {
            return false
        }
    }
}
```

Trustwave® SpiderLabs®

# Demo

# Revisiting Class Dump



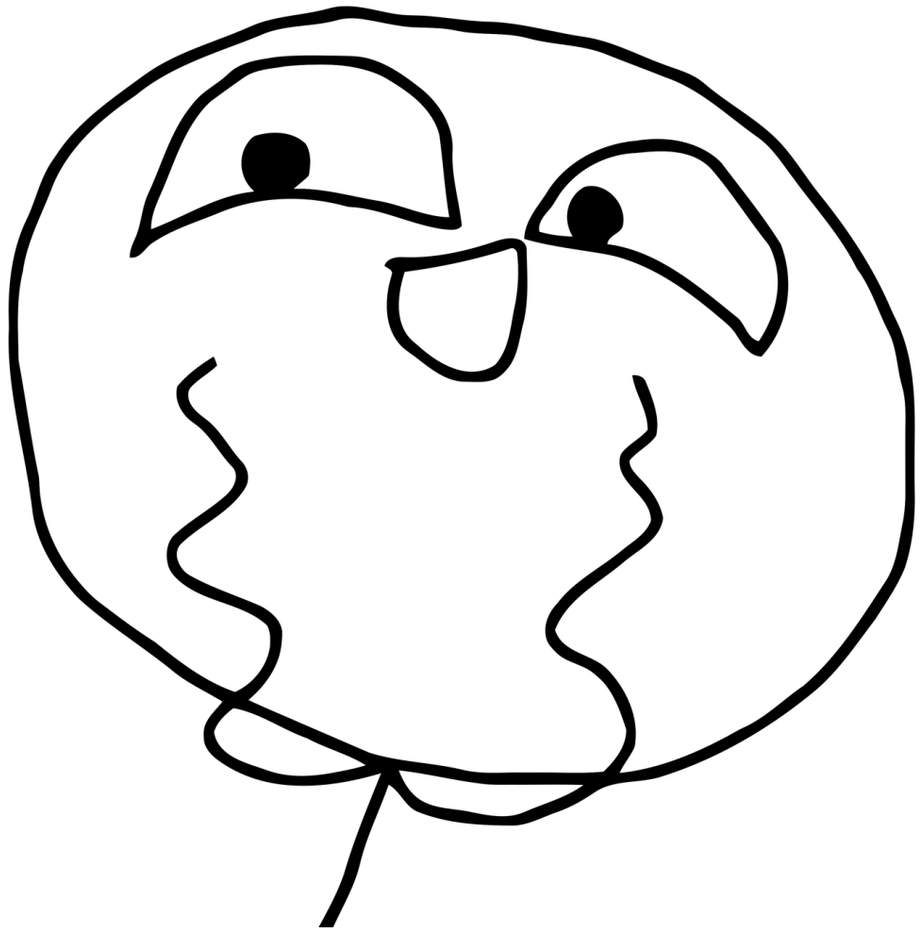
Trustwave® SpiderLabs®

# Revisiting Class Dump

```
class HITB : NSObject {

    var howGreatIsHITB = 7.5

    func isClassDumpGoingToWork(name: String) -> Bool {
        return false
    }

    func isClassDumpGoingtoWorkWithObjCRuntime(runtime name: String) -> Bool {
        if name == "ObjC" {
            return true
        } else {
            return false
        }
    }
}
```

Trustwave® SpiderLabs®

# Revisiting Class Dump



Trustwave® SpiderLabs®

# Other Options

# Other Options

- classdump-dyld (successor to weakclassdump.cy)
- Disassemblers (i.e. Hopper, IDA Pro)
  - Necessary for lower level insight into the app
  - To demangle Swift function names https://github.com/Januzellij/hopperscripts
- cycript? frida?

Trustwave® SpiderLabs®

# Function Hooking

# Hooking Swift Methods

- Still possible.
- Much easier with in mixed Swift/Objective-C binaries.
- Can still write tweaks with Mobile Substrate.

Trustwave® SpiderLabs®

# Hooking Swift Methods

```
class RootconHook {
    var howGreatIsRootcon: Int

    init() {
        howGreatIsRootcon = 5
    }
}
```

Trustwave® SpiderLabs®

# Hooking Swift Methods

- Hooking getter method (works!)

```
int (*howGreatIsRootcon)(id,self);

MSHook(int, howGreatIsRootcon, id self) {
    return 10;
}

%ctor {
    howGreatIsRootcon = (int (*)(id, self)) dlsym(RTLD_DEFAULT, "_TFC9swifttest7Rootcon17howGreatIsRootconSi");
    MSHookFunction(howGreatIsRootcon, MSHake(howGreatIsRootcon));
}
```

Trustwave® SpiderLabs®

# Hooking Swift Methods

- Hooking setter method (kinda works…)

```
int (*howGreatIsRootcon)(id newValue, id self);

MSHook(void, setHowGreatIsRootcon, int newValue) {
    _setHowGreatIsRootcon(10, self);
}

%ctor {
    setHowGreatIsRootcon = (void (*)(int newValue, self)) dlsym(RTLD_DEFAULT,
        "_TFC9swifttest7Rootcon17howGreatIsRootconSi");
    MSHookFunction(setHowGreatIsRootcon, MSHake(setHowGreatIsRootcon));
}
```

Trustwave® SpiderLabs®

# Hooking Swift Methods

- Certain functions in Swift are inlined and the class constructor is one of them (which is directly setting the instance variable)
- So in this case the setter will only be called again by the top level code.
- If you call from there it works.

Trustwave® SpiderLabs®

# Hooking Swift Methods

- Changing the instance variable directly (works but not a good idea probably)

```
int (*howGreatIsRootcon)(id newValue, id self);

MSHook(void, setHowGreatIsRootcon, int newValue) {
    MSHookIvar<int>(self, "howGreatIsRootcon") = 10;
}

%ctor {
    setHowGreatIsRootcon = (void (*)(int newValue, self)) dlsym(RTLD_DEFAULT,
        "_TFC9swifttest7Rootcon17howGreatIsRootconSi");
    MSHookFunction(setHowGreatIsRootcon, MSHake(setHowGreatIsRootcon));
}
```

Trustwave® SpiderLabs®

# Wrap Up

# Wrap Up

- So not all hope is lost when it comes to your standard pen test workflows with Swift apps

- A bit more of a pain in the arse if you don't get access to the source code

- Most issues in iOS and OS X apps are due to poor design decisions or misconfiguration and incorrect implementation of Apple and third party frameworks and libraries.

Trustwave® SpiderLabs®

# Next Steps

- Improve the class-dump-s script :)
  - Remove dependency on swift-demagle
  - Ivars, stripped binaries?
- Runtime inspection
  - cycript works but not as straightforward as with Objective-C
  - LLDB works well if you are familiar with it
  - Will hopefully write a blog post soon

Trustwave® SpiderLabs®

# Q&A?

Trustwave® SpiderLabs®