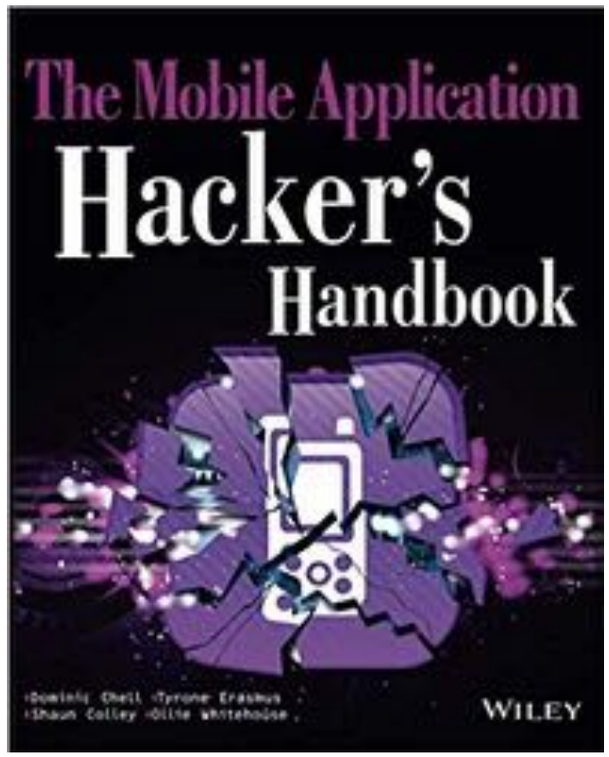
CNIT 128
Hacking Mobile Devices



2. Analyzing iOS Apps
Part 2

Topics: Part 1

- The Security Model
- iOS Apps
- Jailbreaking Explained

Topics: Part 2

- The Data Protection API
- The iOS Keychain
- TouchID
- Reverse Engineering iOS Binaries

The Data Protection API

www.theregister.co.uk/2010/07/27/citi_iphone_app_weakness/

Citigroup says its iPhone app puts customers at risk

Warning: contents include account details

By Dan Goodin 27 Jul 2010 at 00:00 9 SHARE ▼

In a letter, the US banking giant said the Citi Mobile app saved user information in a hidden file that could be used by attackers to gain unauthorized access to online accounts. Personal information stored in the file could include account numbers, bill payments and security access codes, according to *The Wall Street Journal*, which reported the vulnerability earlier.

File Encryption in iOS

- Per-file encryption key
  - Locked with a protection class
  - Protection classes govern when the class keys are kept in memory
- ***Secure enclave*** manages keys
  - In devices with A7 or later chip

Passcode Key

- Uses Password-Based Key Derivation Function 2 (PBKDF2)
  - Inputs are user's passcode and a device-specific key known as the UID key
- UID key is embedded in the device's hardware-based crypto accelerator
  - Cannot be accessed by software on the device

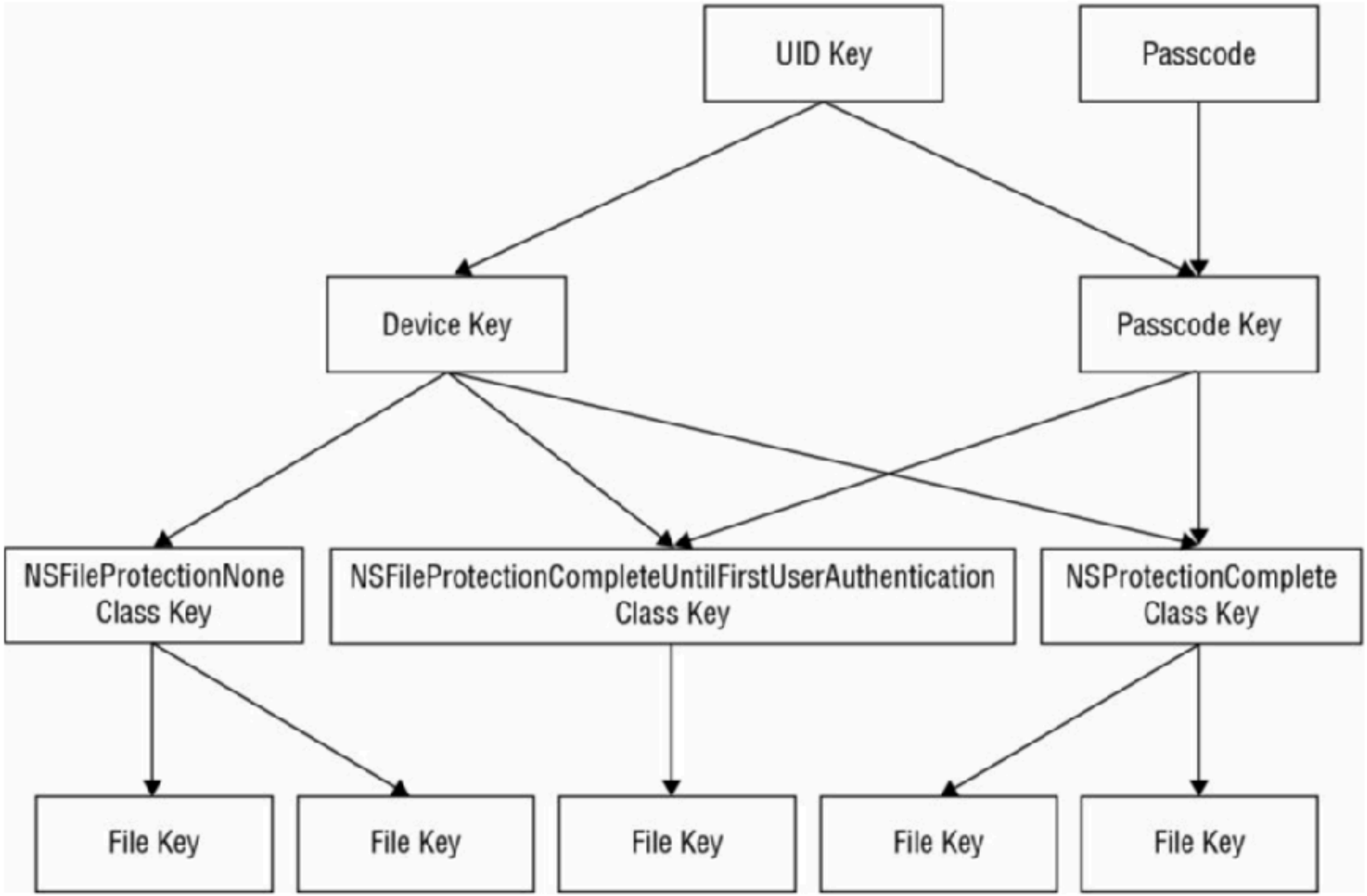


Figure 2.4 The data protection key hierarchy

Protection Classes

- **No Protection**
  - Not encrypted
- **Complete Protection**
  - Encrypted; inaccessible when device is locked
- **Complete Unless Open**
  - If file is in use, it's unencrypted when device is locked

Protection Classes

- **Complete Until First User Authentication**
  - Encrypted only from bootup to first unlock
  - Only protects it from attacks requiring a device reboot
  - This is the default setting

The iOS Keychain

Keychain Protection Classes

- **kSecAttrAccessibleAlways**
  - The keychain item is always accessible
- **kSecAttrAccessibleWhenUnlocked**
  - The keychain item is accessible only when the device is unlocked
- **kSecAttrAccessibleAfterFirstUnlock**
  - The keychain item is only accessible after the first unlock from boot. This offers some protection against attacks that require a device reboot

Keychain Protection Classes

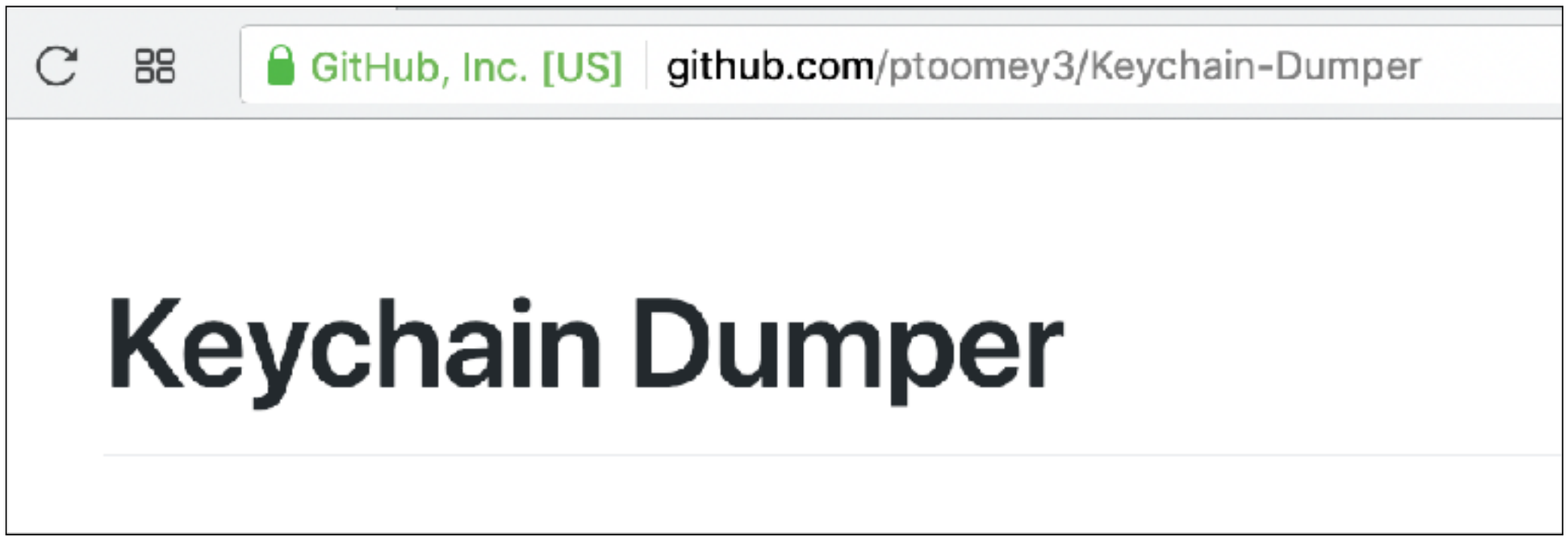
- **SecAttrAccessibleAlwaysThisDeviceOnly**
  - The keychain item is always accessible but cannot be migrated to other devices.
- **kSecAttrAccessibleWhenUnlockedThisDeviceOnly**
  - The keychain item is only accessible when the device is unlocked and may not be migrated to other devices
- **kSecAttrAccessibleAfterFirstUnlockThisDeviceOnly**
  - The keychain item is accessible after the first unlock from boot and may not be migrated to other devices
- **kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly**
  - Only allows you to store keychain items if a passcode is set on the device. These items are accessible only when a passcode is set; if the password is later unset, they cannot be decrypted

Access Control and Authentication Policies

- Introduced in iOS 8
- Prevents access to keychain items when no passcode is set on the device
- Requires entry of the passcode or Touch ID (fingerprint)

Accessing the iOS Keychain

- SQLite database in **/var/Keychains**
- **Keychain Dumper** works on jailbroken iOS devices



TouchID

Device Locking

- If Touch ID is not enabled:
  - Secure Enclave holds the data protection class keys
  - When device is locked the key material is discarded
    - For the "Complete Protection" class

With Touch ID Enabled

- Keys are not discarded, but held in memory
  - Wrapped with a key that is available only to the Touch ID system
- After Touch ID authentication, the wrapping key becomes available

www.ccc.de/en/updates/2013/ccc-breaks-apple-touchid

Chaos Computer Club breaks Apple TouchID

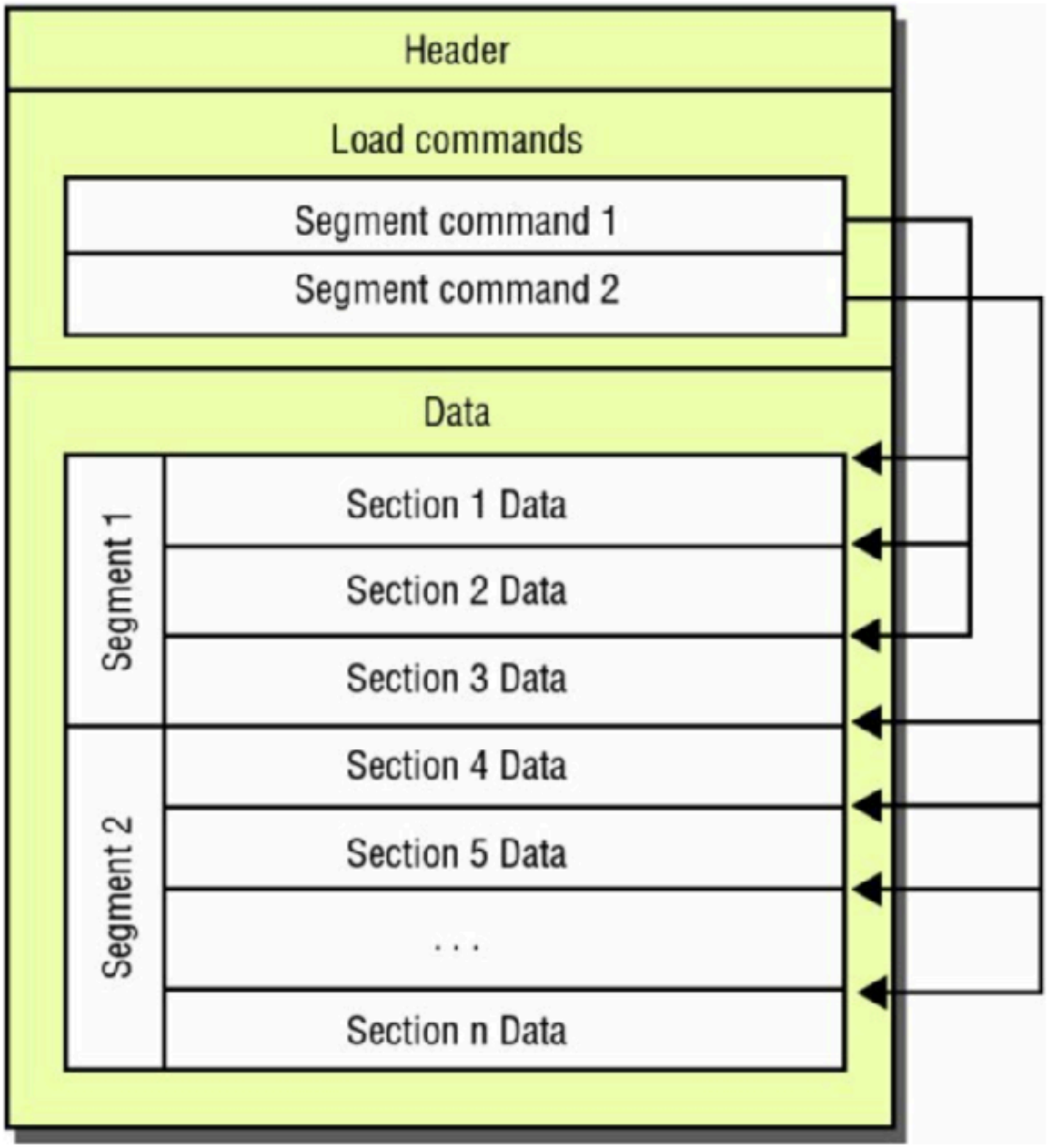
2013-09-21 22:04:00, frank

The biometrics hacking team of the Chaos Computer Club (CCC) has successfully bypassed the biometric security of Apple's TouchID using easy everyday means. A fingerprint of the phone user, photographed from a glass surface, was enough to create a fake finger that could unlock an iPhone 5s secured with TouchID. This demonstrates – again – that fingerprint biometrics is unsuitable as access control method and should be avoided.

Reverse Engineering iOS Binaries

Mach-O Binaries

- **Header**
  - Identifies file format & architecture
- **Load commands**
  - **File layout**
    - Location of symbol table
    - Encrypted segments
    - Shared libraries
- **Data**


Header
Load commands
Segment command 1
Segment command 2
Data
Segment 1
Section 1 Data
Section 2 Data
Section 3 Data
Segment 2
Section 4 Data
Section 5 Data
. . .
Section n Data

Fat Binaries

- Contain multiple Mach-O files
  - Archived in one binary
- To support different architectures

Table 2.4 Architecture Support in Modern iOS Devices

ARCHITECTURE	IPHONE	IPOD TOUCH	IPAD	IPAD MINI
Armv7	3GS, 4, 4S, 5, 5C, 5S	3rd, 4th, 5th generation	All versions	All versions
Armv7s	5, 5C, 5S	No support	4th generation, Air	2nd generation
Arm64	5S, 6, 6 Plus	No support	Air	2nd generation

otool

When I use otool to locate the architectures, I can see I have 2, an ARMv7(cpusubtype 9) and a ARM64 (cpysubtype 0)

```
iPad:~/map/MyApp.app root# otool -arch all -Vh MyApp
MyApp (architecture cputype (12) cpusubtype (9)):
Mach header
  magic cputype cpusubtype  caps    filetype ncmds sizeofcmds
MH_MAGIC     ARM          9  0x00      EXECUTE    41        4760

MyApp (architecture cputype (16777228) cpusubtype (0)):
Mach header
  magic cputype cpusubtype  caps    filetype ncmds sizeofcmds
MH_MAGIC_64 16777228          0  0x00      EXECUTE    41        536
```

ARM Architectures

ARCHITECTURE	CPU TYPE	CPU SUBTYPE
ARMv6	12	6
ARMv7	12	9
ARMv7S	12	11
ARM64	16777228	0

Removing Architectures

- **lipo** can extract one architecture from a fat binary

```
$ lipo -thin armv7 mahhswiftapp -output mahhswiftappv7
```

Security-Related Features

Position-Indendent Executable (PIE)

- A compiler option
- Required for **Address Space Layout Randomization (ASLR)**
- All built-in Apple apps have PIE, since iOS 5
- You can check other apps with **otool**

Stack Smashing Protection

- Places a ***canary*** value on the stack
  - To protect the saved base pointer and instruction pointer (**rbp** and **rip**)
  - The LLVm compiler applies it intelligently
    - Only to functions that need it
    - Typically, ones that use character arrays
- Enabled by default for apps compiled with recent versions of Xcode

Automatic Reference Counting (ARC)

- Introduced in iOS SDK v 5.0
- Reduces risk of **use-after-free** and **double-free** vulnerabilities
  - Heap memory corruption

Decrypting App Store Binaries

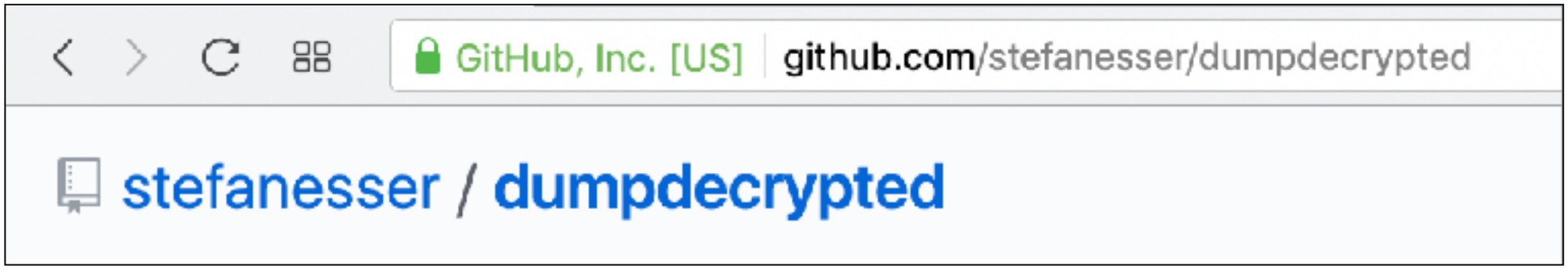
Encryption

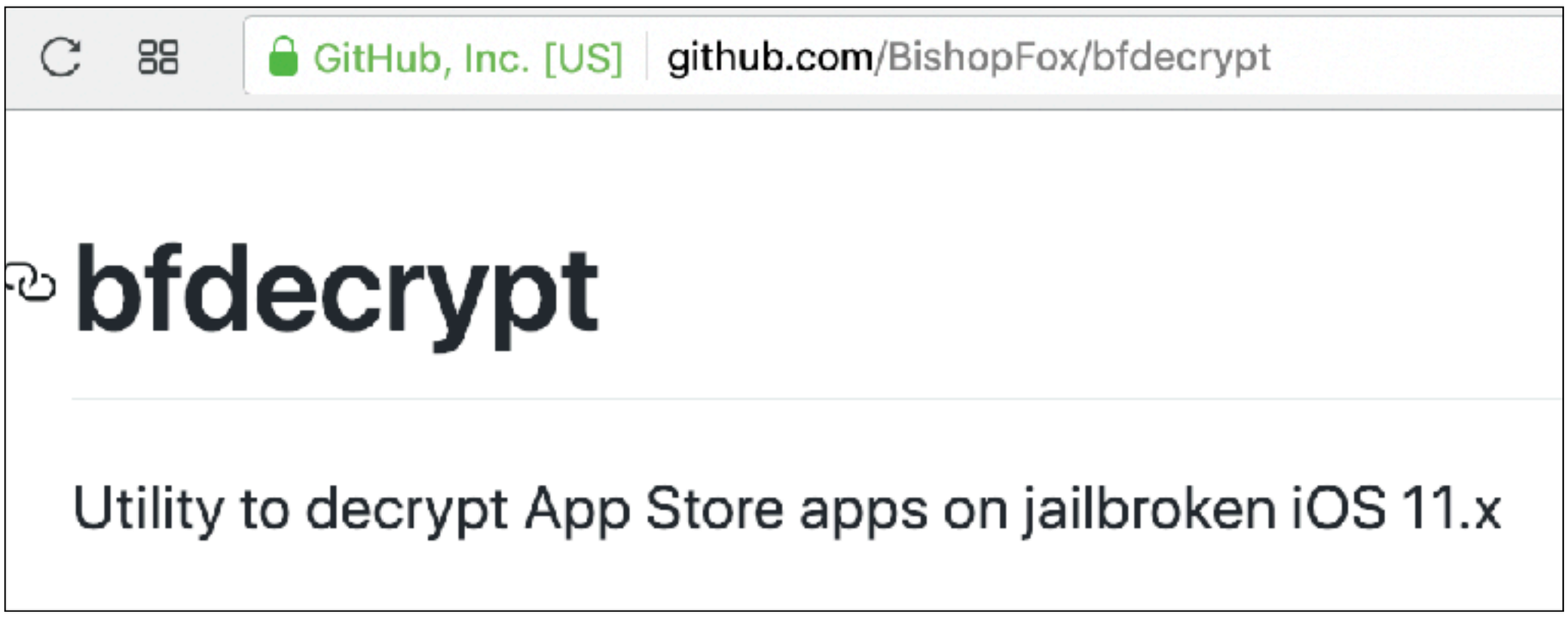
- Apps in the App Store
  - Are encrypted with FairPlay Digital Rights Management (DRM)
- Apps are decrypted at run time by the loader

Decrypting with a Debugger

1. Run the app & use **gdb attach**
2. Find the base address with **info sharedlibrary**
3. Retrieve memory segment with **dump memory**
4. Write decrypted segment over original encrypted segment with **dd**

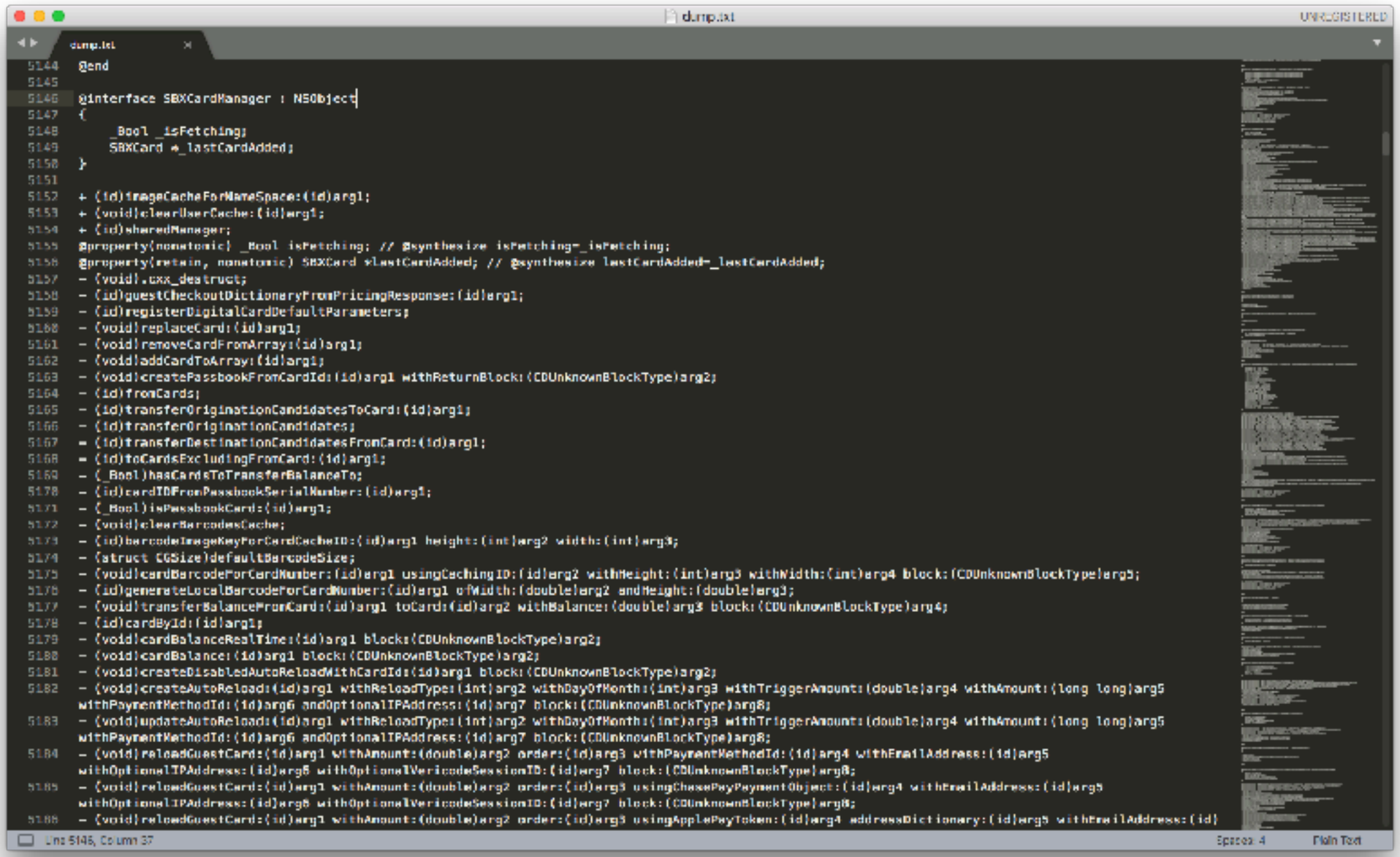
Automating the Decryption Process





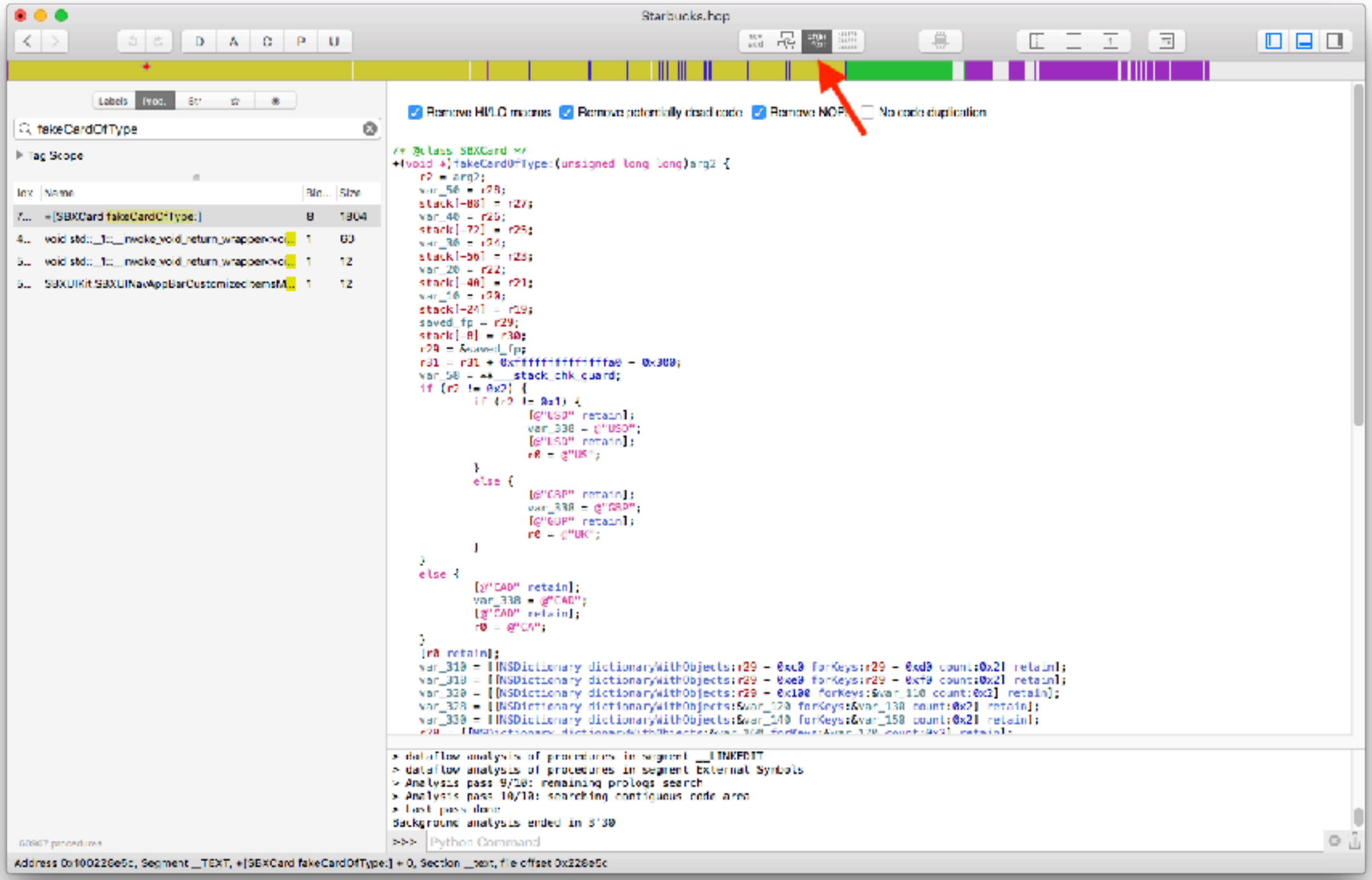
Inspecting Decrypted Binaries

- **class-dump-z** and **class-dump**
  - Shows details about internal classes, methods, and variables
- The next page shows the class-dump output for the Starbucks app
- To disassemble and decompile apps, use **Hopper** or **IDA Pro**



- https://ivrodriguez.com/reverse-engineer-ios-apps-ios-11-edition-part2/

Hopper


Starbucks.hop
Remove HLIL macros
Remove potentially dead code
Remove NOPs
No code duplication
fakeCardOfType
Tag Scope
Name
-[SBXCard fakeCardOfType:]
/* @class SBXCard */
+(void *)fakeCardOfType:(unsigned long long)arg2 {
Python Command
Address 0x100226e5c, Segment __TEXT, +[SBXCard fakeCardOfType:] + 0, Section __text, file offset 0x226e5c


Kahoot!