

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

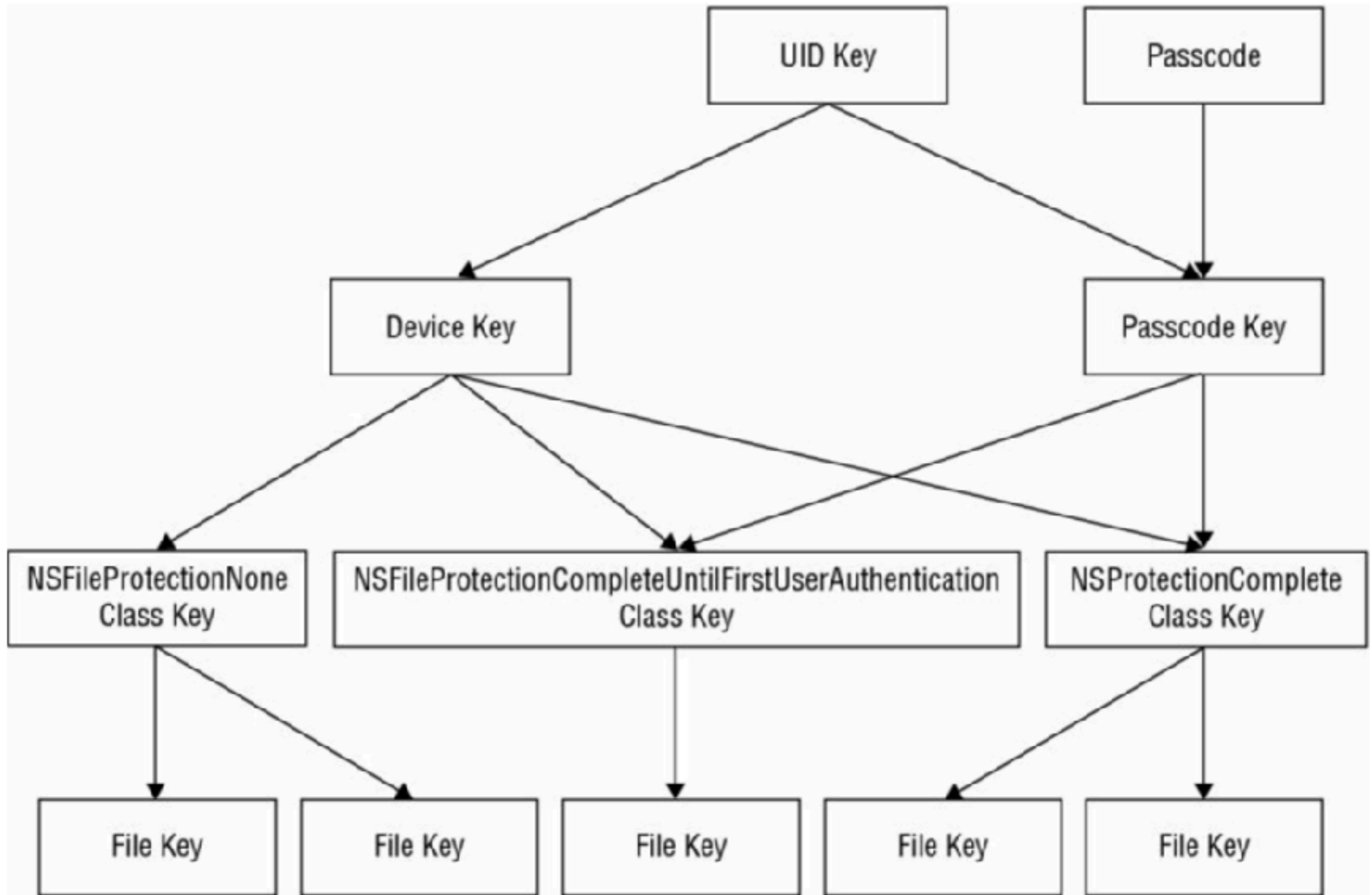


Figure 2.4 The data protection key hierarchy

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

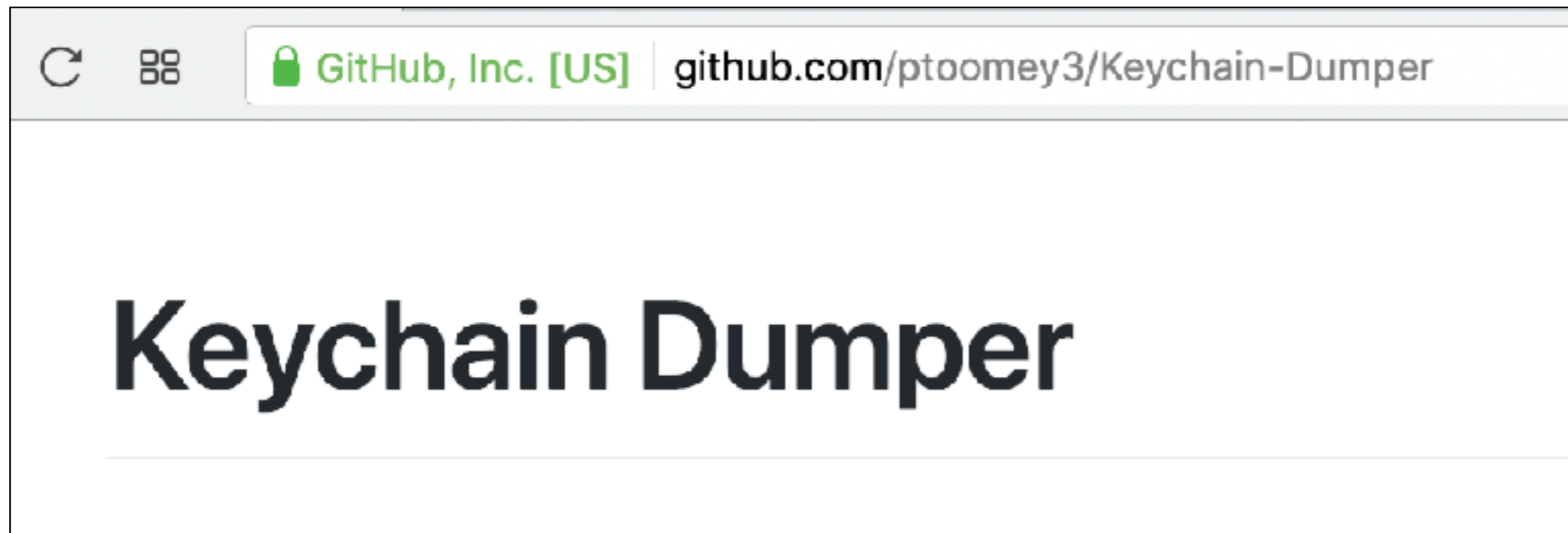
- **SecAttrAccessibleAlwaysThisDeviceOnly**
 - The keychain item is always accessible but cannot be migrated to other devices.
- **kSecAttrAccessibleWhenUnlockedThisDeviceOnly**
 - The keychain item is only accessible when the device is unlocked and may not be migrated to other devices
- **kSecAttrAccessibleAfterFirstUnlockThisDeviceOnly**
 - 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

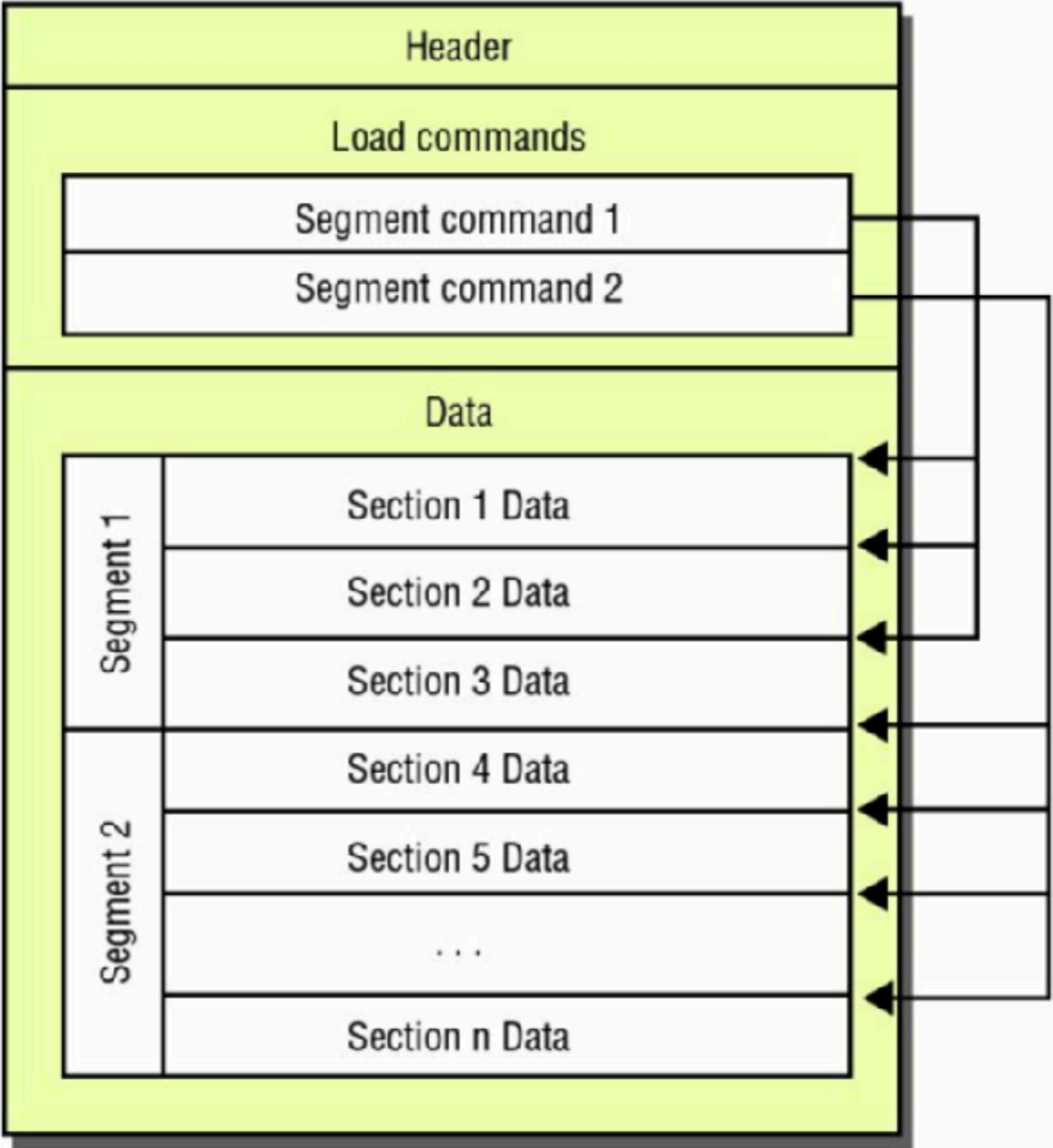
2013-09-21 22:04:00, frank

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**



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	3 rd , 4 th , 5 th generation	All versions	All versions
Armv7s	5, 5C, 5S	No support	4 th generation, Air	2 nd generation
Arm64	5S, 6, 6 Plus	No support	Air	2 nd generation

otool

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

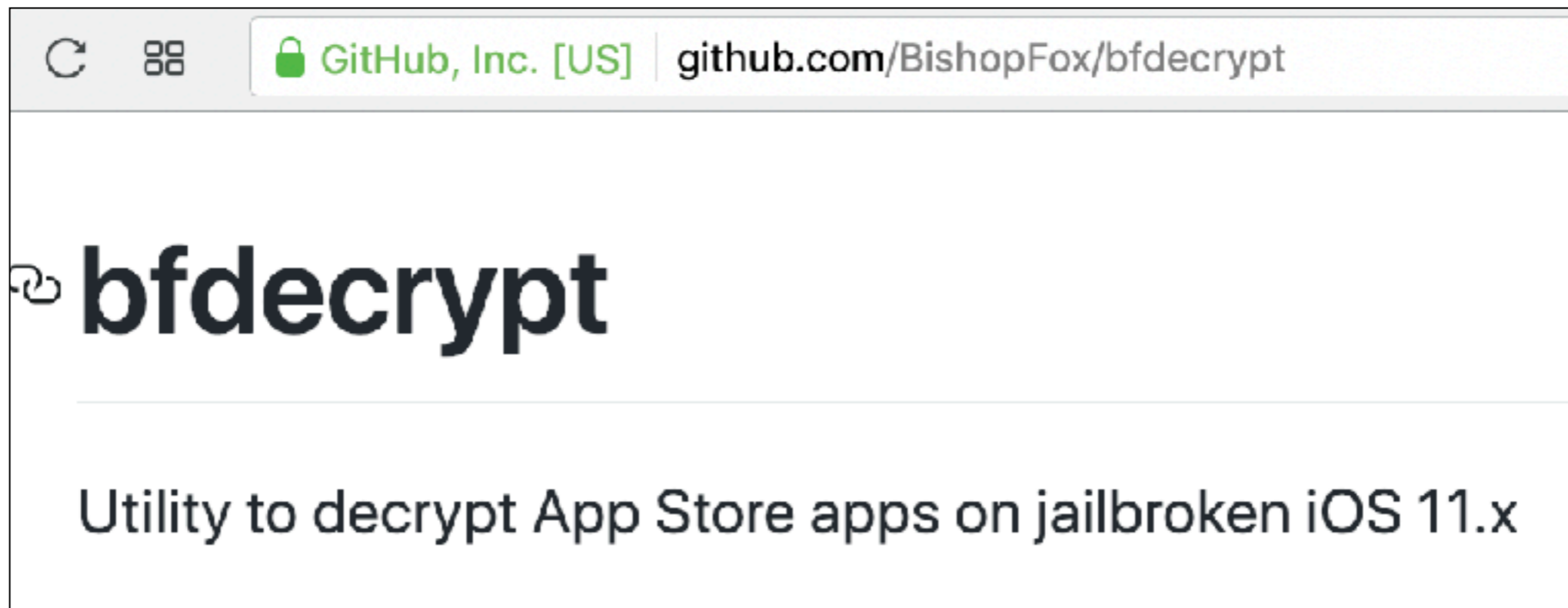
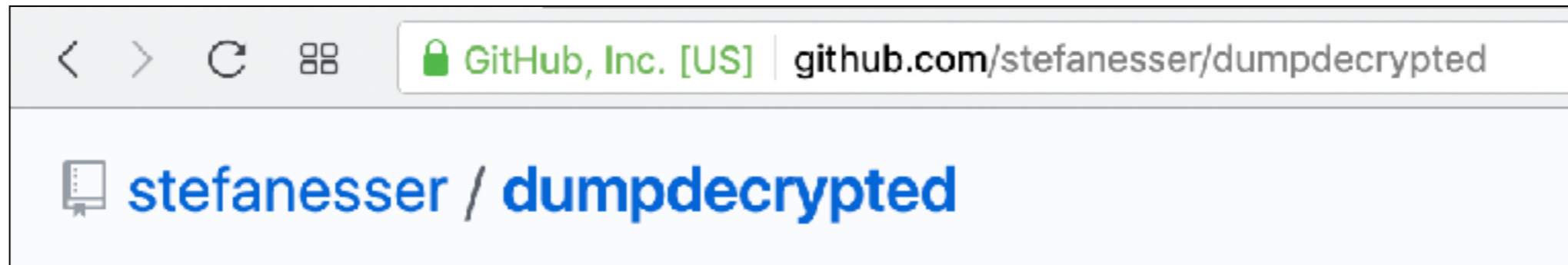
Encryption

- 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**

```
dump.txt
UNREGISTERED
dump.txt
5144 @end
5145
5146 @interface SBXCardManager : NSObject
5147 {
5148     _Bool _isFetching;
5149     SBXCard *_lastCardAdded;
5150 }
5151
5152 + (id)imageCacheForNameSpace:(id)arg1;
5153 + (void)clearUserCache:(id)arg1;
5154 + (id)sharedManager;
5155 @property(nonatomic) _Bool isFetching; // @synthesize isFetching=_isFetching;
5156 @property(retain, nonatomic) SBXCard *_lastCardAdded; // @synthesize lastCardAdded=_lastCardAdded;
5157 - (void).cxx_destruct;
5158 - (id)questCheckoutDictionaryFromPricingResponse:(id)arg1;
5159 - (id)registerDigitalCardDefaultParameters;
5160 - (void)replaceCard:(id)arg1;
5161 - (void)removeCardFromArray:(id)arg1;
5162 - (void)addCardToArray:(id)arg1;
5163 - (void)createPassbookFromCardId:(id)arg1 withReturnBlock:(CDUnknownBlockType)arg2;
5164 - (id)fromCards;
5165 - (id)transferOriginationCandidatesToCard:(id)arg1;
5166 - (id)transferOriginationCandidates;
5167 - (id)transferDestinationCandidatesFromCard:(id)arg1;
5168 - (id)toCardsExcludingFromCard:(id)arg1;
5169 - (_Bool)hasCardsToTransferBalanceTo;
5170 - (id)cardIDFromPassbookSerialNumber:(id)arg1;
5171 - (_Bool)isPassbookCard:(id)arg1;
5172 - (void)clearBarcodesCache;
5173 - (id)barcodeImageKeyForCardCacheID:(id)arg1 height:(int)arg2 width:(int)arg3;
5174 - (struct CGSize)defaultBarcodeSize;
5175 - (void)cardBarcodeForCardNumber:(id)arg1 usingCachingID:(id)arg2 withHeight:(int)arg3 withWidth:(int)arg4 block:(CDUnknownBlockType)arg5;
5176 - (id)generateLocalBarcodeForCardNumber:(id)arg1 ofWidth:(double)arg2 andHeight:(double)arg3;
5177 - (void)transferBalanceFromCard:(id)arg1 toCard:(id)arg2 withBalance:(double)arg3 block:(CDUnknownBlockType)arg4;
5178 - (id)cardById:(id)arg1;
5179 - (void)cardBalanceRealTime:(id)arg1 blocks:(CDUnknownBlockType)arg2;
5180 - (void)cardBalance:(id)arg1 block:(CDUnknownBlockType)arg2;
5181 - (void)createDisabledAutoReloadWithCardIds:(id)arg1 block:(CDUnknownBlockType)arg2;
5182 - (void)createAutoReload:(id)arg1 withReloadTypes:(int)arg2 withDayOfMonth:(int)arg3 withTriggerAmount:(double)arg4 withAmount:(long long)arg5
withPaymentMethodId:(id)arg6 andOptionalIPAddress:(id)arg7 block:(CDUnknownBlockType)arg8;
5183 - (void)updateAutoReloads:(id)arg1 withReloadTypes:(int)arg2 withDayOfMonth:(int)arg3 withTriggerAmount:(double)arg4 withAmount:(long long)arg5
withPaymentMethodId:(id)arg6 andOptionalIPAddress:(id)arg7 block:(CDUnknownBlockType)arg8;
5184 - (void)reloadGuestCard:(id)arg1 withAmount:(double)arg2 order:(id)arg3 withPaymentMethodId:(id)arg4 withEmailAddress:(id)arg5
withOptionalIPAddress:(id)arg6 withOptionalVericodeSessionID:(id)arg7 block:(CDUnknownBlockType)arg8;
5185 - (void)reloadGuestCard:(id)arg1 withAmount:(double)arg2 order:(id)arg3 usingChasePayPaymentObject:(id)arg4 withEmailAddress:(id)arg5
withOptionalIPAddress:(id)arg6 withOptionalVericodeSessionID:(id)arg7 block:(CDUnknownBlockType)arg8;
5186 - (void)reloadGuestCard:(id)arg1 withAmount:(double)arg2 order:(id)arg3 usingApplePayToken:(id)arg4 addressDictionary:(id)arg5 withEmailAddress:(id)
```

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

Hopper

The screenshot displays the Hopper Disassembler interface for a file named Starbucks.hop. The main window shows the decompiled code for the `takeCardOfType` method. The code is written in C++ and includes various stack operations, variable assignments, and conditional logic based on the card type (e.g., USD, GBP, CAD, etc.).

On the left side, there is a search bar containing the text `takeCardOfType` and a "Tag Scope" table. The table lists the following entries:

Idx	Name	Bit...	Size
7...	+([SBXCard takeCardOfType:])	8	1804
4...	void std::_T::__nwke_void_return_wrappe...	1	63
5...	void std::_T::__nwke_void_return_wrappe...	1	12
5...	SBXUIKit.SBXUINavigationController.tenstM...	1	12

The code editor shows the following decompiled code:

```
/* @class SBXCard */
+([void +)takeCardOfType:(unsigned long long)arg2 {
    r2 = arg2;
    var_56 = r28;
    stack[-88] = r27;
    var_40 = r25;
    stack[-72] = r25;
    var_36 = r24;
    stack[-56] = r23;
    var_20 = r22;
    stack[-40] = r21;
    var_16 = r20;
    stack[-24] = r19;
    saved_fp = r29;
    stack[8] = r30;
    r29 = &saved_fp;
    r31 = r31 + 0xfffffffffffffa0 - 0x308;
    var_58 = &__stack_chk_guard;
    if (r2 != 0x0) {
        if (r2 != 0x1) {
            [@"USD" retain];
            var_338 = @"USD";
            [@"USD" retain];
            r8 = @"USD";
        }
        else {
            [@"GBP" retain];
            var_338 = @"GBP";
            [@"GBP" retain];
            r8 = @"UK";
        }
    }
    else {
        [@"CAD" retain];
        var_338 = @"CAD";
        [@"CAD" retain];
        r8 = @"CA";
    }
    [r8 retain];
    var_318 = [NSMutableDictionary dictionaryWithObjects:r29 - 0xc0 forKeys:r29 - 0xd0 count:0x2] retain];
    var_318 = [NSMutableDictionary dictionaryWithObjects:r29 - 0xe0 forKeys:r29 - 0xf0 count:0x2] retain];
    var_320 = [NSMutableDictionary dictionaryWithObjects:r29 - 0x100 forKeys:&var_110 count:0x2] retain];
    var_328 = [NSMutableDictionary dictionaryWithObjects:&var_120 forKeys:&var_130 count:0x2] retain];
    var_330 = [NSMutableDictionary dictionaryWithObjects:&var_140 forKeys:&var_150 count:0x2] retain];
    r28 = [NSMutableDictionary dictionaryWithObjects:r29 - 100 forKeys:&var_120 count:0x2] retain];
}
```

At the bottom, the console window displays the following output:

```
> dataflow analysis of procedures in segment __LINKEDIT
> dataflow analysis of procedures in segment External Symbols
> Analysis pass 9/10: remaining prologs search
> Analysis pass 10/10: searching contiguous code area
> Last pass done
Background analysis ended in 3'30
Python Command
```

The status bar at the bottom indicates the current address: `Address 0x100226e5c, Segment __TEXT, +[SBXCard takeCardOfType:] + 0, Section __text, file offset 0x226e5c`.

Kahoot!