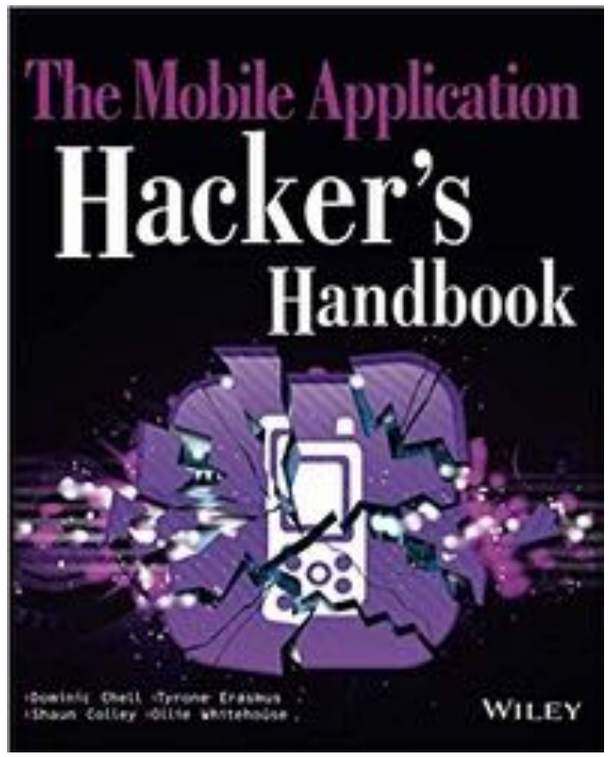
CNIT 128
Hacking Mobile Devices



3. Attacking iOS Apps
Part 1

Topics: Part 1

- Introduction to Transport Security
- Identifying Insecure Storage
- Patching iOS Applications with Hopper

Topics: Part 2

- Attacking the iOS Runtime
- Understanding Interprocess Communication
- Attacking Using Injection

Attack Scenarios

- From the network
  - Tainted data from server-side applications
- Physical access to the phone
- Interactive access to the phone
  - Control of another app on the phone

Introduction to Transport Security

Cleartext Channels

- Such as HTTP
  - Never safe
  - Even if not transmitting sensitive data like passwords
- Because an attacker could inject JavaScript

Finding Cleartext

- Examine traffic with Wireshark or Burp

Three SSL/TLS Implementations

- The URL loading system
- The Carbon Framework
- The Secure Transport API

The URL Loading System

- High-level classes and methods like
  - **NSURLConnection**
  - **NSURLSession**
- Simplest method
- Most widely adopted

Carbon Framework

- More granular API than the URL loading system
  - Gives developers greater control over network requests
  - Implemented using the **CFNetwork** class

Secure Transport API

- Low-level API
  - The foundation of CFNetwork and the URL loading system
- Greatest control over the transport
  - Complex to implement
  - Rarely used directly

Certificate Validation

- SSL and TLS use certificate-based authentication to
  - Ensure that you are communicating with the desired server
  - Prevent eavesdropping and tampering attacks
- Unless the validation is weakened

Trusted CA

- Certificates must be signed by a trusted Certificate Authority (CA)
- Accepting self-signed or unvalidated certificates undermines TLS and SSL
  - Allowing MiTM attacks

NSURLConnection Class

- Developer can allow self-signed certificates
  - By customizing the **didReceiveAuthenticationChallenge** method

Carbon Framework

- Can allow self-signed certficates by setting up an SSL settings dictionary
- That sets the **kCFStreamSSLValidatesCertificateChain** constant to false

Secure Transport API

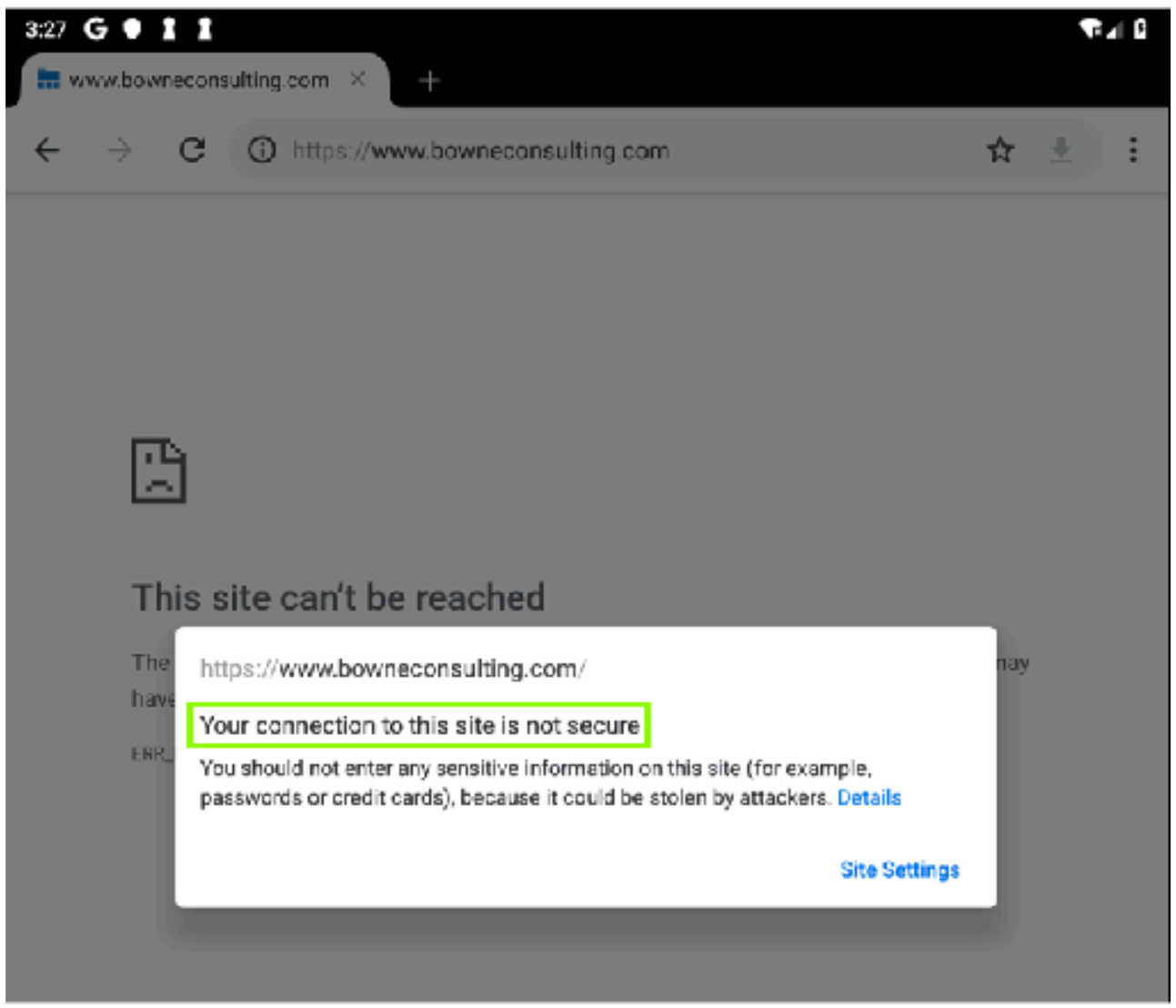
- Setting the **kSSLSessionOptionBreakOnServerAuth** option
  - Disables the API's certificate validation
- But the app might have its own trust evaluation routines, like certificate pinning

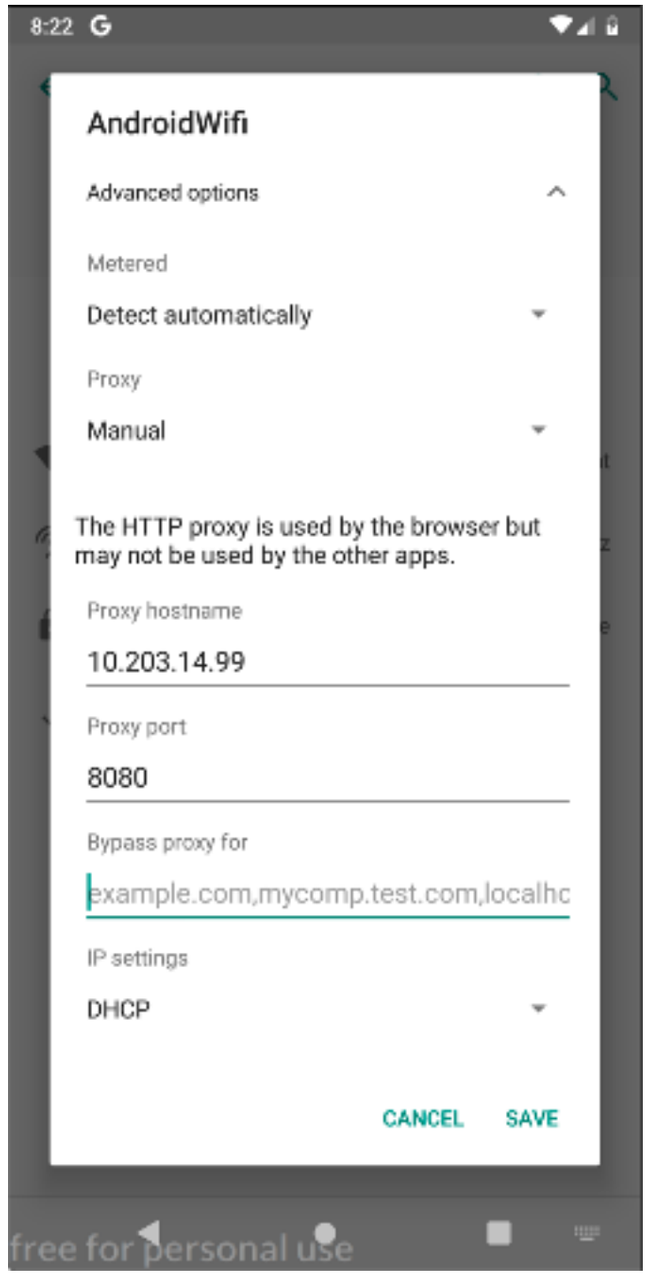
Certificate Problems

- Self-signed certificates
- Expired certificates
- Mismatched hostnames
- Expired root CA certificates
- Allowing any root certificate

Dynamic Testing

- Route traffic through the Burp proxy
- Browser detects the SSL error





GOTO Fail


imperialviolet.org/2014/02/22/applebug.html
ImperialViolet
Apple's SSL/TLS bug (22 Feb 2014)
Yesterday, Apple pushed a rather spooky security update for iOS that suggested that something was horribly wrong with SSL/TLS in iOS but gave no details. Since the answer is at the top of the Hacker News thread, I guess the cat's out of the bag already and we're into the misinformation-quashing stage now.

So here's the Apple bug:

```
static OSStatus
SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer signedParams,
                                 uint8_t *signature, UInt16 signatureLen)
{
        OSStatus        err;
        ...

        if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
                goto fail;
        if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
                goto fail;
                goto fail;
        if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
                goto fail;
        ...

fail:
        SSLFreeBuffer(&signedHashes);
        SSLFreeBuffer(&hashCtx);
        return err;
}
```

Note the two `goto fail` lines in a row. The first one is correctly bound to the if statement but the second, despite the indentation, isn't conditional at all. The code will always jump to the end from that second goto, `err` will contain a successful value because the SHA1 update operation was successful and so the signature verification will never fail.

SSL Session Security

- There are other possible SSL errors if an app is using the Carbon framework or the Secure Transport API
- But not if it uses the high-level URL loading API
  - Because there is no way to modify the SSL/TLS session properties

Protocol Versions

- CFNetwork and Secure Transport APIs
  - Both allow a developer to modify the protocol version
  - SSLv2 and SSLv3 are vulnerable

CFNetwork API (Carbon Framework) Protocol Settings

- These settings specify vulnerable versions
  - **kCFStreamSocketSecurityLevelSSLv2**
  - **kCFStreamSocketSecurityLevelSSLv3**
  - **kCFStreamSocketSecurityLevelTLSv1**
- These settings allow negotiation of insecure versions
  - **kCFStreamSocketSecurityLevelNone**
  - **kCFStreamSocketSecurityLevelNegotiatedSSL**

Secure Transport API Protocol Settings

- These settings allow vulnerable versions
  - **kSSLProtocolUnknown**
  - **kSSLProtocol3**
  - **kTLSProtocol1**
  - **kTLSProtocol11**
  - **kDTLSProtocol1**
- This is the preferred setting
  - **kTLSProtocol12**

Cipher Suite Negotiation

```
Transport Layer Security
▼ TLSv1.2 Record Layer: Handshake Protocol: Client Hello
    Content Type: Handshake (22)
    Version: TLS 1.0 (0x0301)
    Length: 512
  ▼ Handshake Protocol: Client Hello
      Handshake Type: Client Hello (1)
      Length: 508
      Version: TLS 1.2 (0x0303)
    ▶ Random: 44a5542a2dd5d4f8b355e9e0ac102f3530b70516de37f2ef…
      Session ID Length: 32
      Session ID: ef4db6d9dc128c25e72a7dc9ee100d5e3269aa3b6c2b82ba…
      Cipher Suites Length: 34
    ▼ Cipher Suites (17 suites)
        Cipher Suite: Reserved (GREASE) (0xbaba)
        Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)
        Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
        Cipher Suite: TLS_CHACHA20_POLY1305_SHA256 (0x1303)
        Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
        Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
        Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
        Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
        Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca9)
        Cipher Suite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca8)
        Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
        Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
        Cipher Suite: TLS_RSA_WITH_AES_128_GCM_SHA256 (0x009c)
        Cipher Suite: TLS_RSA_WITH_AES_256_GCM_SHA384 (0x009d)
        Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
        Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
        Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x000a)
```

```
50412 → 443 [SYN, ECN, CWR]
443 → 50412 [SYN, ACK, ECN]
50412 → 443 [ACK] Seq=1 Ack=
Client Hello
```

Specific Cipher Suite

- Developer may specify an insecure suite
- Secure Transport and CFNetwork APIs allow this

```
SSLCipherSuite *ciphers = (SSLCipherSuite *)malloc(1 * \

                                sizeof(SSLCipherSuite));

ciphers[0] = SSL_RSA_WITH_RC4_128_MD5;

SSLSetEnabledCiphers(sslContext, ciphers, 1);
```

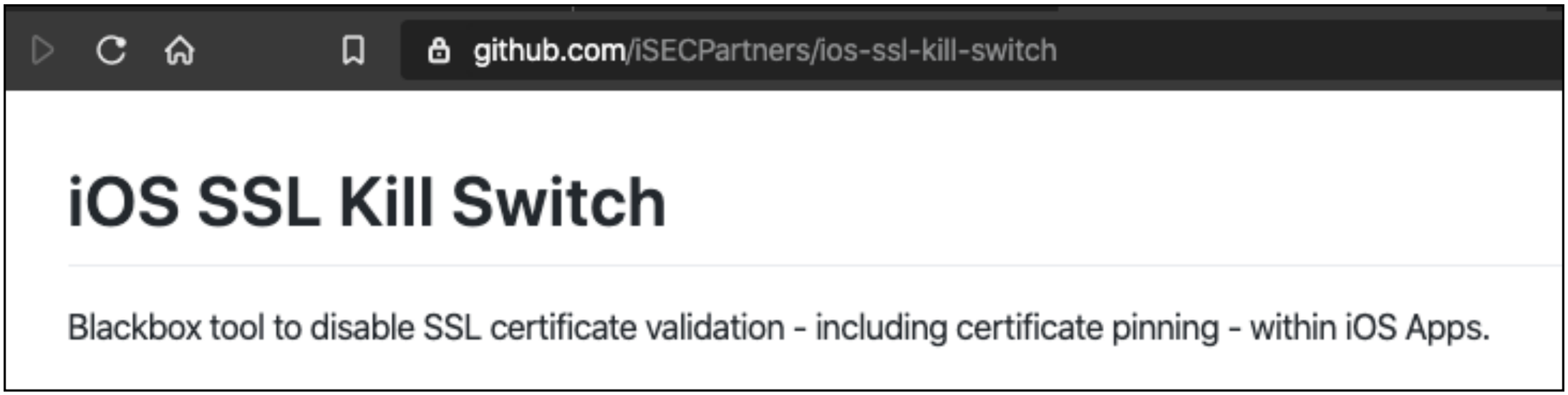
Intercepting Encrypted Communications

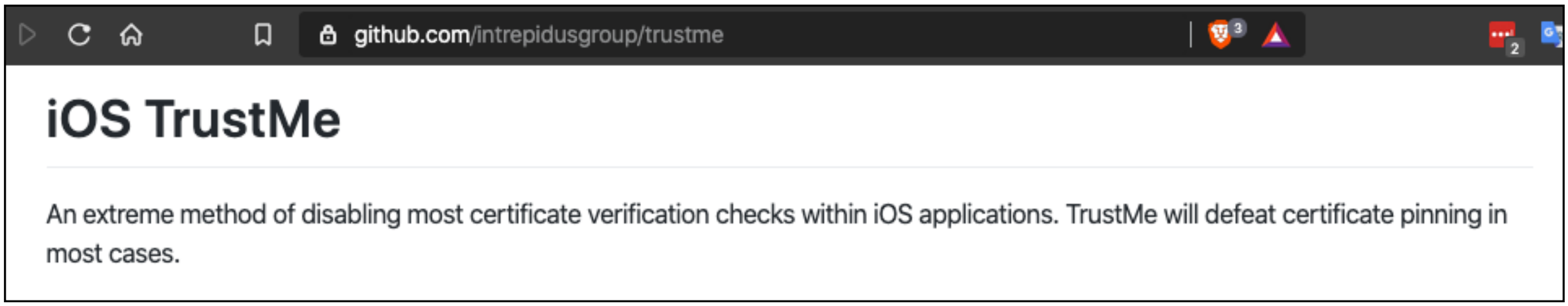
- Necessary to test apps
- Must install the Burp certificate on the phone

Bypassing Certificate Pinning

- An app has information about the correct certificate embedded in it
    - And refuses to connect with other certificates
- This must be bypassed to view the network traffic

Substrate Tweaks





Identifying Insecure Storage

Local Storage

- An attacker can steal local data when
  - The phone is stolen while unlocked
  - The Touch ID sensor is bypassed
  - Remote compromise through exploitation
  - Default credentials on jailbroken phones
  - There is no passcode
  - Pairing with a malicious computer
  - Exploiting the boot chain

Storage Errors

- Stored by app in plaintext
- Using custom encryption with insecure key
- **Stored with wrong data protection class**
- Inadvertently stored by iOS

Data Protection API

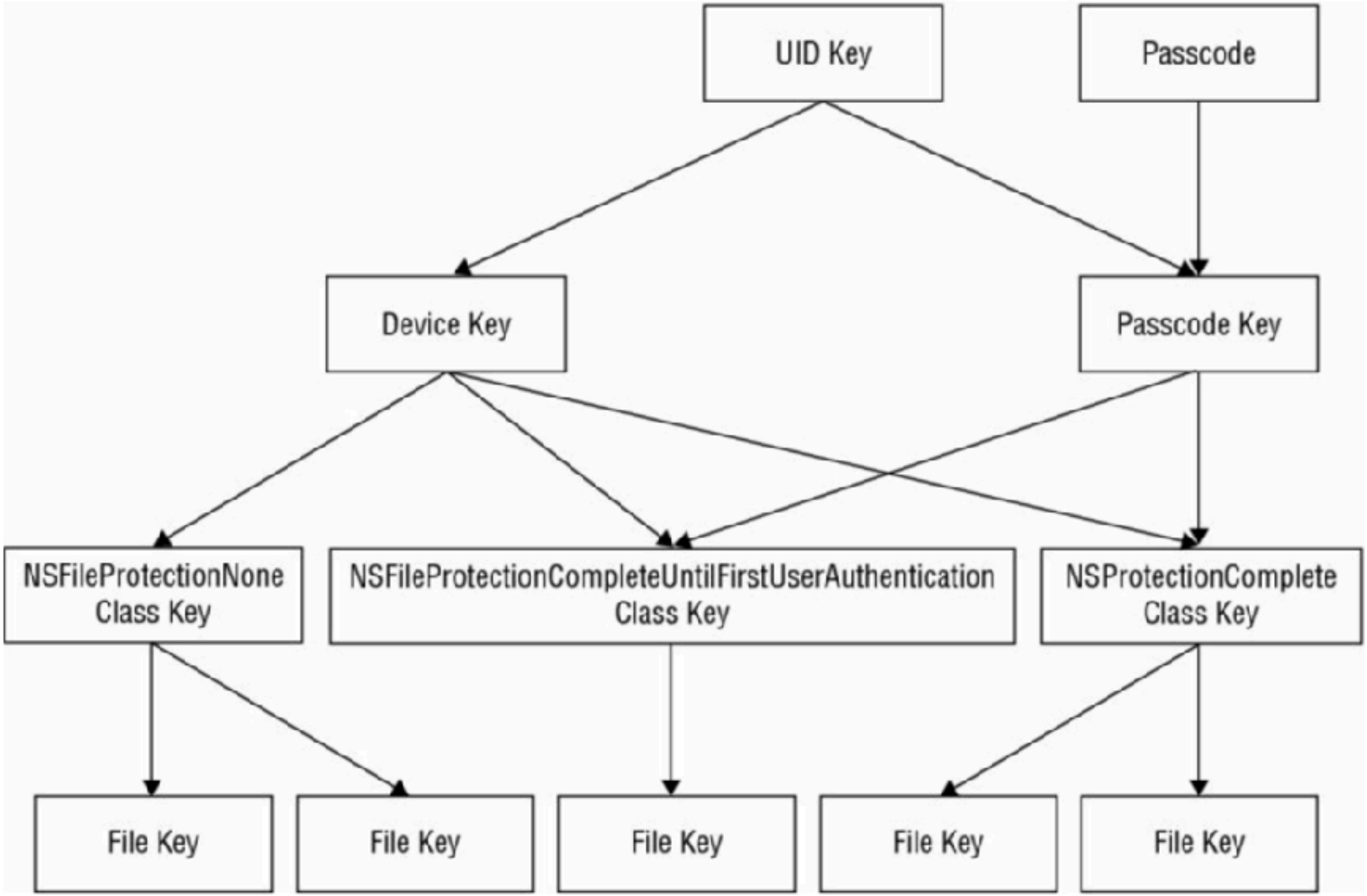


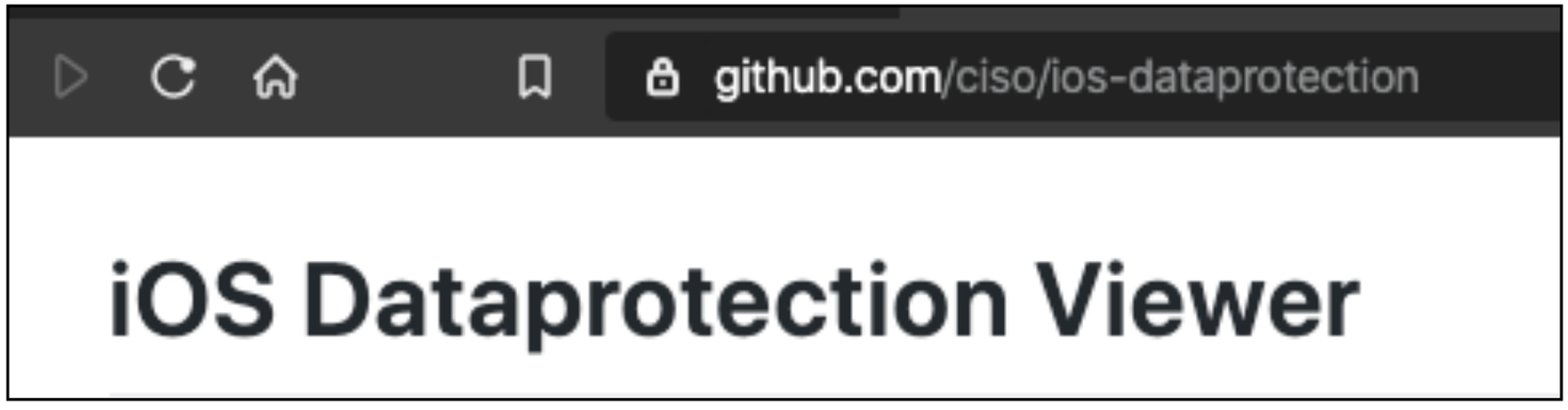
Figure 2.4 The data protection key hierarchy

Protection Classes

- **No Protection**
  - Not encrypted
  - Unsuitable for sensitive data
- **Complete Until First User Authentication**
  - Discouraged for sensitive data

Identifying Data Protection Classes

- If data can be backed up
  - Back up, then use




```
$ grep mdsec ~/Desktop/analysis.csv

com.mdsec.lab1-1a,1,NSFileProtectionComplete,Library/Preferences/
```

Keychain Items

- **keychain_dump** creates **plist** files
- **protection_class** key inside them

```
<key>protection_class</key>

<string>WhenUnlocked</string>
```

Dynamic Analysis

- Tests stored data that doesn't get backed up
- Use Cydia Substrate
  - The old Snoop-It tool was helpful, but it's gone

Patching iOS Applications with Hopper

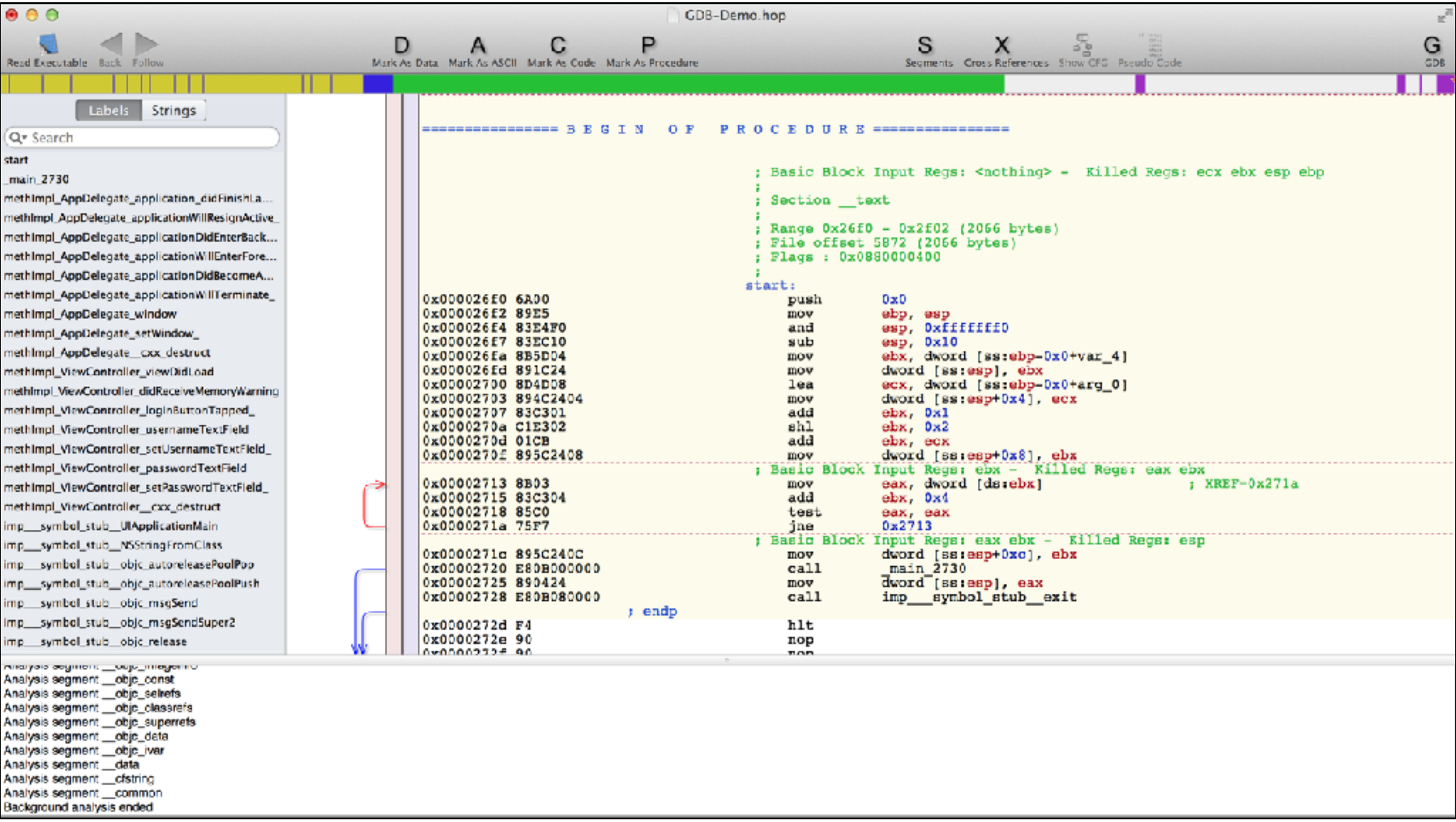
Get the Binary

- Project M 702

```
Sam-2:test sambowne$ dump-ipa -l
 PID  Name           Identifier
----  ------------   --------------------------
9833  App Store      com.apple.AppStore
9718  Mail           com.apple.mobilemail
9764  Photos         com.apple.mobileslideshow
9837  Ringdahl EMS   com.ringdahlfieldguide.com
9786  Safari         com.apple.mobilesafari
9767  Settings       com.apple.Preferences
   -  Books          com.apple.iBooks
   -  Calculator     com.apple.calculator
```

```
Sam-2:test sambowne$ dump-ipa Ringdahl\ EMS
Start the target app Ringdahl EMS
unexpected error while probing dyld of target process
Sam-2:test sambowne$ dump-ipa Ringdahl\ EMS
Start the target app Ringdahl EMS
Dumping Ringdahl EMS to /var/folders/7b/6pk3st191kdf66g1ypjxtc540000gn/T
start dump /var/containers/Bundle/Application/7017F297-BB69-4D7D-B624-14AA6FED7CF9/apps4ems.app/apps4ems
apps4ems.fid: 100%|[redacted]| 832k/832k [00:00<00:00, 4.93MB/s]
Adult Cardiac Arrest.pdf: 17.0MB [00:03, 5.25MB/s]
0.00B [00:00, ?B/s]Generating "Ringdahl EMS.ipa"
0.00B [00:00, ?B/s]
Sam-2:test sambowne$
```

Disassemble with Hopper



Jailbreak Detection

```
0000b090        movw    r0, #0x1518
0000b094        movt    r0, #0x0
0000b098        movw    r2, #0x152a
0000b09c        movt    r2, #0x0
0000b0a0        add     r0, pc                  ; @selector(alloc)
0000b0a2        add     r2, pc                  ; objc_cls_ref_UIAlertView
0000b0a4        ldr     r1, [r0]                ; @selector(alloc), argument #2 for method imp___symbolstub1__objc_msgSend
0000b0a6        ldr     r0, [r2]                ; objc_cls_ref_UIAlertView, argument #1 for method imp___symbolstub1__objc_
0000b0a8        blx     imp___symbolstub1__objc_msgSend
0000b0ac        movw    r1, #0x1500
0000b0b0        mov.w   r12, #0x0
0000b0b4        movt    r1, #0x0
0000b0b8        movw    r2, #0x158a
0000b0bc        add     r1, pc                  ; @selector(initWithTitle:message:delegate:cancelButtonTitle:otherButtonTit
0000b0be        movt    r2, #0x0
0000b0c2        movw    r3, #0x158a
0000b0c6        add     r2, pc                  ; @"Security Violation"
0000b0c8        movt    r3, #0x0
0000b0cc        movw    r9, #0x1598
0000b0d0        ldr     r1, [r1]                ; @selector(initWithTitle:message:delegate:cancelButtonTitle:otherButtonTit
0000b0d2        movt    r9, #0x0
0000b0d6        add     r3, pc                  ; @"This device is jailbroken, please remove the jailbreak and try again."
0000b0d8        add     r9, pc                  ; @"OK"
```

Pseudo Code

Pseudo Code

```
void -[ViewController viewDidLoad](void * self, void * _cmd) {
    sp = sp - 0x14;
    arg_C = self;
    arg_10 = *0xc5d8;
    [[&arg_C super] viewDidLoad];
    r0 = sub_b1fc();
    if (r0 != 0x0) {
            r0 = [UIAlertView alloc];
            asm{ stm.w      sp, {r4, r9, r12} };
            r4 = [r0 initWithTitle:@"Security Violation" message:@"This device is jailbroken, please remove the jailbreak and try
again." delegate:STK1 cancelButtonTitle:STK0 otherButtonTitles:STK-1];
            [r4 show];
            r0 = [r4 release];
    }
    return;
}
```


Kahoot!