Breaking "Secure" Mobile Applications

BSidesMCR

June 2014

@domchell
@MDSecLabs

Agenda

- Background
- The problem
- Case studies
- Binary protections
- Final case study
- Conclusions

Introduction

Background

- Spent ~18 months from 2011 on one project, helping to make their app “secure”
- Application protections weren’t mainstream or documented at the time
- Continuous arms race; stuck in a break, advice, develop cycle
- Learned a lot and contributed to what I believe to be a quite an impressive project

The Problem
Mobile Data

- Lots of applications, doing lots of ***interesting*** things
  - Banking
  - Social networks
  - Gambling
  - Privacy driven applications
  - Enterprise applications with internal integrations

- Lots of interesting data
  - PII
  - Financial data
  - Sensitive corporate data

The Attack Surface

- Mobile application insecurities are well documented
  - Insecure storage
  - Transport insecurities
  - Injection vulnerabilities
  - Tampering attacks
- What are the attack scenarios?
  - Jailbreaking/Rooting by user/attacker – Pangu?
  - Attacks from malware, e.g. Unflod Baby Panda
  - App imitation and repacking with malicious code
  - Targeted exploitation
  - Casual drive by download

The Problem
Application Insecurities

- Direct remediation of application insecurities is generally well documented

- Many applications have started to follow recommended actions to address the traditional vulnerabilities:
  – Client side authentication
  – Encryption of persistent data
  – Certificate pinning

- Do these protect against all attack scenarios?
  – How trivial is it to bypass from on-device?
  – What about resource/app modification – adware/spyware insertion?

- The arms race begins…

Case Studies

- “Secure” is banded around a lot, not just in mobile
  - “Secure end-to-end messaging”
  - “Secure storage”
  - “Secure device management”

- Decided to challenge some of these claims

Case Study #1
MobileIron

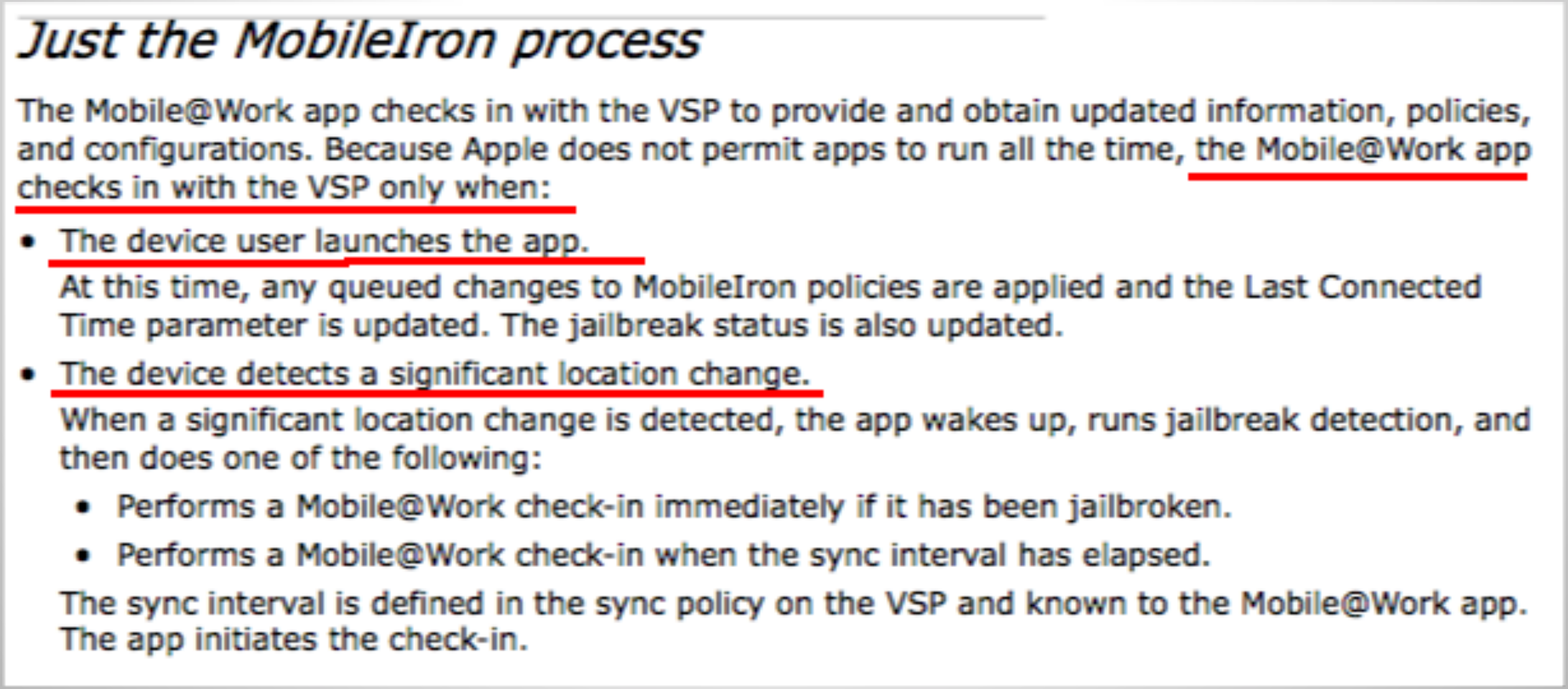
- Popular MDM driven by policies set on the VSP (Virtual Smartphone Platform)

- Reports back to the VSP/Sentry devices of policy violations such as jailbreaking and can react accordingly, examples include removing e-mail and VPN access

Case Study #1

MobileIron

- Well known limitation of the software, forcibly closing the app and disabling location services prevents it detecting policy violations and reporting back to the VSP



Just the MobileIron process

The Mobile@Work app checks in with the VSP to provide and obtain updated information, policies, and configurations. Because Apple does not permit apps to run all the time, the Mobile@Work app checks in with the VSP only when:

- The device user launches the app.

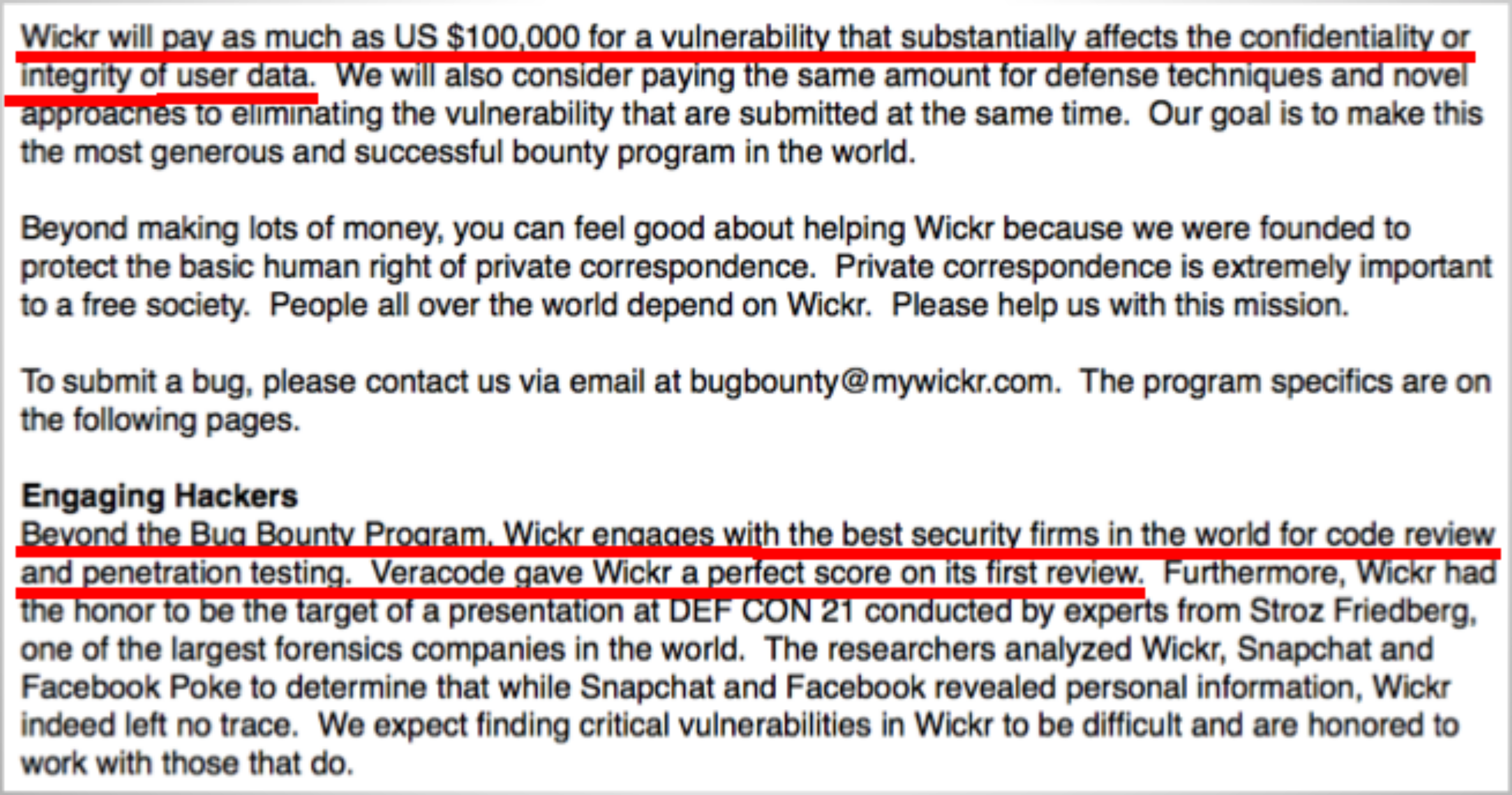
  At this time, any queued changes to MobileIron policies are applied and the Last Connected Time parameter is updated. The jailbreak status is also updated.
- The device detects a significant location change.

  When a significant location change is detected, the app wakes up, runs jailbreak detection, and then does one of the following:
  - Performs a Mobile@Work check-in immediately if it has been jailbroken.
  - Performs a Mobile@Work check-in when the sync interval has elapsed.

  The sync interval is defined in the sync policy on the VSP and known to the Mobile@Work app. The app initiates the check-in.

Case Study #2
Wickr

- In January 2014 Wickr announced a bug bounty



Wickr will pay as much as US $100,000 for a vulnerability that substantially affects the confidentiality or integrity of user data. We will also consider paying the same amount for defense techniques and novel approaches to eliminating the vulnerability that are submitted at the same time. Our goal is to make this the most generous and successful bounty program in the world.

Beyond making lots of money, you can feel good about helping Wickr because we were founded to protect the basic human right of private correspondence. Private correspondence is extremely important to a free society. People all over the world depend on Wickr. Please help us with this mission.

To submit a bug, please contact us via email at bugbounty@mywickr.com. The program specifics are on the following pages.

Engaging Hackers
Beyond the Bug Bounty Program, Wickr engages with the best security firms in the world for code review and penetration testing. Veracode gave Wickr a perfect score on its first review. Furthermore, Wickr had the honor to be the target of a presentation at DEF CON 21 conducted by experts from Stroz Friedberg, one of the largest forensics companies in the world. The researchers analyzed Wickr, Snapchat and Facebook Poke to determine that while Snapchat and Facebook revealed personal information, Wickr indeed left no trace. We expect finding critical vulnerabilities in Wickr to be difficult and are honored to work with those that do.

Case Study #2
Wickr

- The application is a privacy driven instant messaging service
- Tag line "leave no trace" – supposedly forensically sound
- Self-destructing messages, pictures, fully encrypted
- The app employed no binary protections so tampering was fairly trivial
- Within 24 hours there were some interesting findings

Case Study #2

Wickr

WICKR DEMO

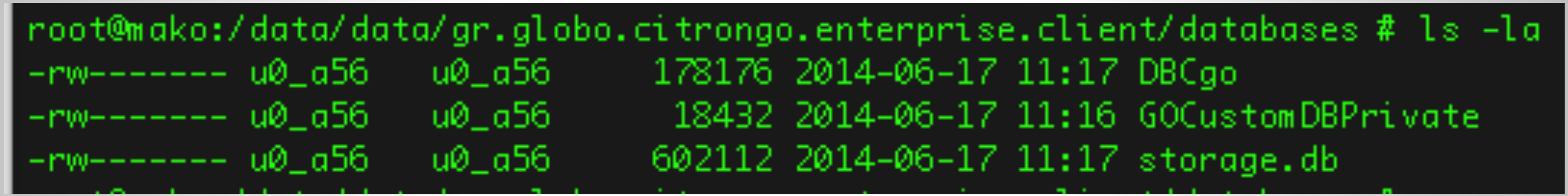
Case Study #3
GO!Enterprise

MDSec
Setting The Standard.

- BYOD container that allows a separate workspace for mail, contacts, secure browsing, file storage etc.
- "The GO!Enterprise mobility platform was designed from the ground up with security in mind. Thus GO!Enterprise solutions inherit a wealth of security features that minimize the risk of unauthorized access, data leakage and security breaches."
- All managed from a per enterprise cloud instance

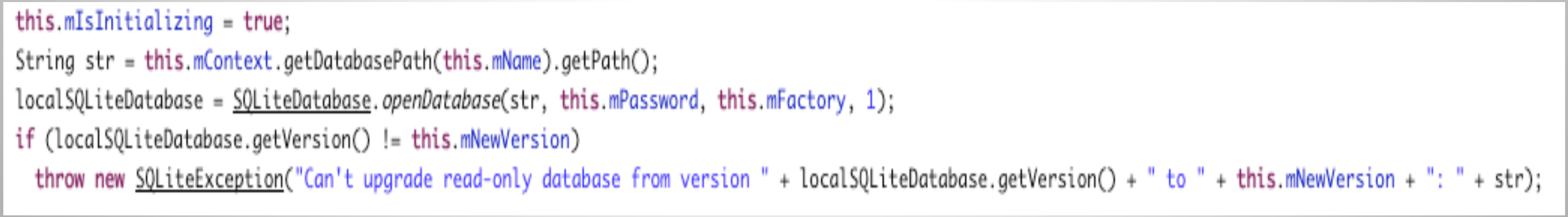
Case Study #3
GO!Enterprise

- Installed the app and synchronised data; appeared to be using these databases for storage



```
root@mako:/data/data/gr.globo.citrongo.enterprise.client/databases # ls -la
-rw------- u0_a56   u0_a56    178176 2014-06-17 11:17 DBCgo
-rw------- u0_a56   u0_a56     18432 2014-06-17 11:16 GOCustomDBPrivate
-rw------- u0_a56   u0_a56    602112 2014-06-17 11:17 storage.db
```

- Decompiling and analysing the APK, revealed it was using SQLCipher

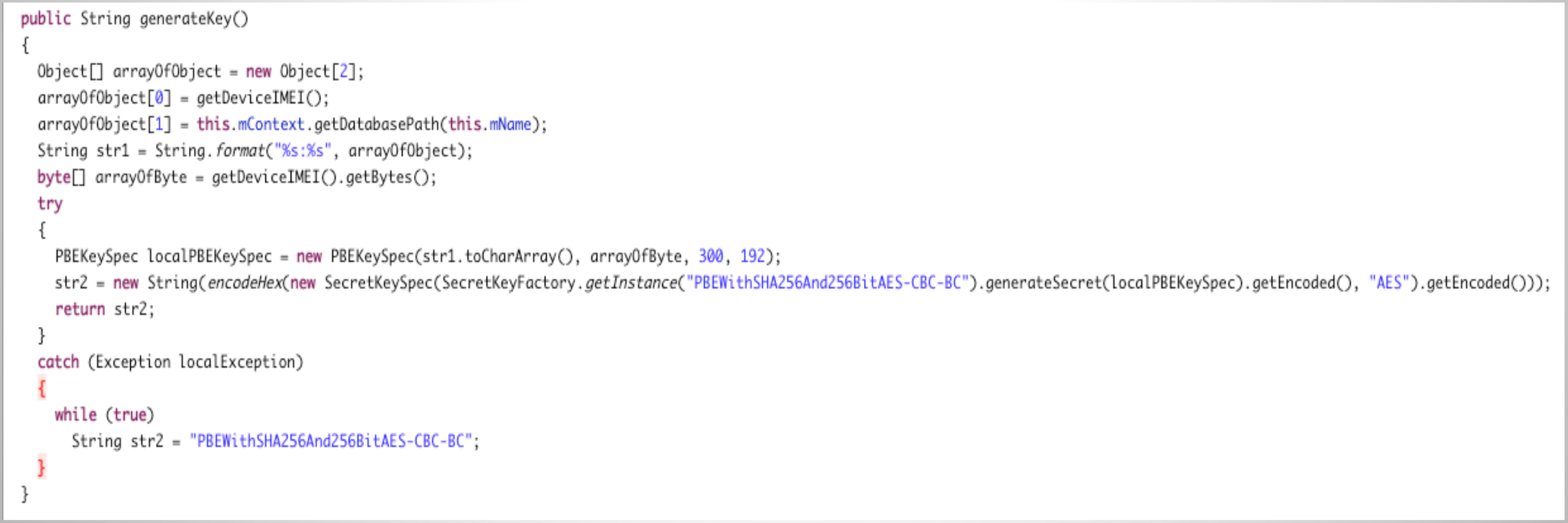


```
this.mIsInitializing = true;
String str = this.mContext.getDatabasePath(this.mName).getPath();
localSQLiteDatabase = SQLiteDatabase.openDatabase(str, this.mPassword, this.mFactory, 1);
if (localSQLiteDatabase.getVersion() != this.mNewVersion)
  throw new SQLiteException("Can't upgrade read-only database from version " + localSQLiteDatabase.getVersion() + " to " + this.mNewVersion + ": " + str);
```

Case Study #3

GO!Enterprise

- Next step was to find where mPassword comes from:

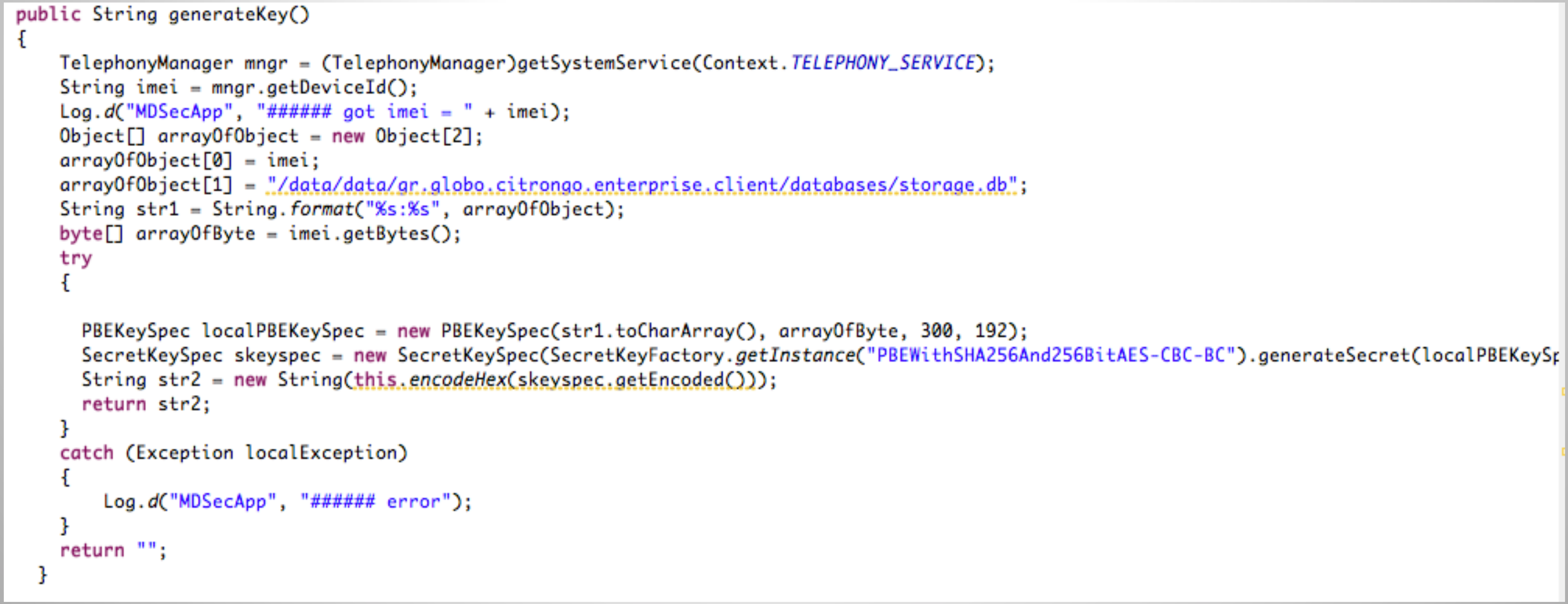


```
public String generateKey()
{
  Object[] arrayOfObject = new Object[2];
  arrayOfObject[0] = getDeviceIMEI();
  arrayOfObject[1] = this.mContext.getDatabasePath(this.mName);
  String str1 = String.format("%s:%s", arrayOfObject);
  byte[] arrayOfByte = getDeviceIMEI().getBytes();
  try
  {
    PBEKeySpec localPBEKeySpec = new PBEKeySpec(str1.toCharArray(), arrayOfByte, 300, 192);
    str2 = new String(encodeHex(new SecretKeySpec(SecretKeyFactory.getInstance("PBEWithSHA256And256BitAES-CBC-BC").generateSecret(localPBEKeySpec).getEncoded(), "AES").getEncoded()));
    return str2;
  }
  catch (Exception localException)
  {
    while (true)
      String str2 = "PBEWithSHA256And256BitAES-CBC-BC";
  }
}
```

Case Study #3

GO!Enterprise

- The key appears to be derived from the IMEI and the path to the database, using the IMEI as a salt
- This can be verified by reproducing the code in another app

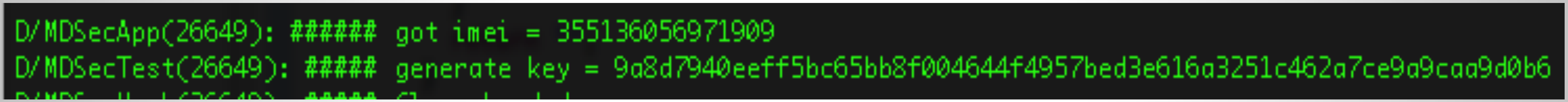


```
public String generateKey()
{
    TelephonyManager mngr = (TelephonyManager)getSystemService(Context.TELEPHONY_SERVICE);
    String imei = mngr.getDeviceId();
    Log.d("MDSecApp", "###### got imei = " + imei);
    Object[] arrayOfObject = new Object[2];
    arrayOfObject[0] = imei;
    arrayOfObject[1] = "/data/data/gr.globo.citrongo.enterprise.client/databases/storage.db";
    String str1 = String.format("%s:%s", arrayOfObject);
    byte[] arrayOfByte = imei.getBytes();
    try
    {

      PBEKeySpec localPBEKeySpec = new PBEKeySpec(str1.toCharArray(), arrayOfByte, 300, 192);
      SecretKeySpec skeyspec = new SecretKeySpec(SecretKeyFactory.getInstance("PBEWithSHA256And256BitAES-CBC-BC").generateSecret(localPBEKeySp
      String str2 = new String(this.encodeHex(skeyspec.getEncoded()));
      return str2;
    }
    catch (Exception localException)
    {
      Log.d("MDSecApp", "###### error");
    }
    return "";
  }
```

Case Study #3

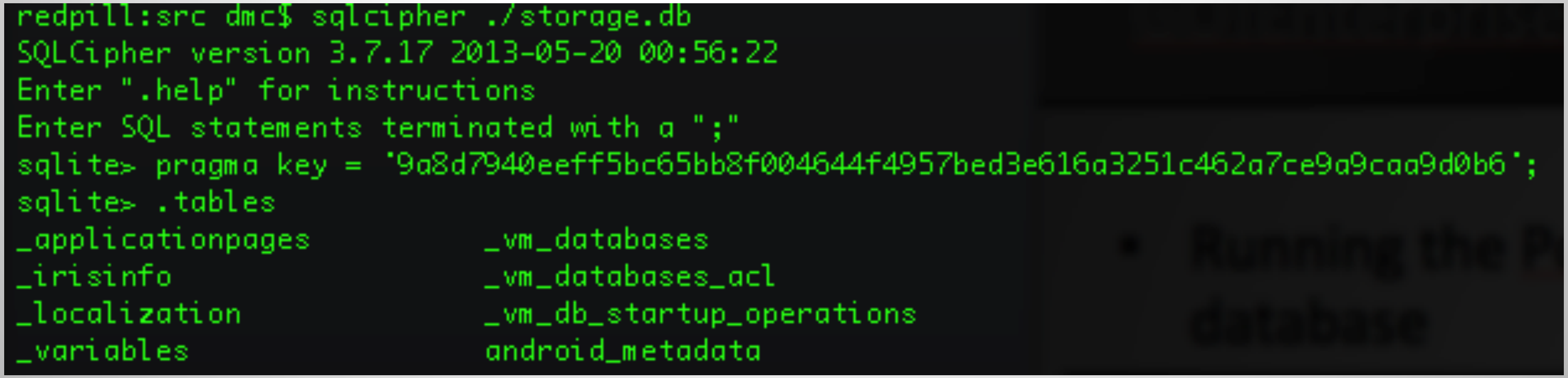
GO!Enterprise

- Running the PoC generates the following key for the "storage.db" database



```
D/MDSecApp(26649): ###### got imei = 355136056971909
D/MDSecTest(26649): ##### generate key = 9a8d7940eeff5bc65bb8f004644f4957bed3e616a3251c462a7ce9a9caa9d0b6
```

- Using the key it's possible to view the database

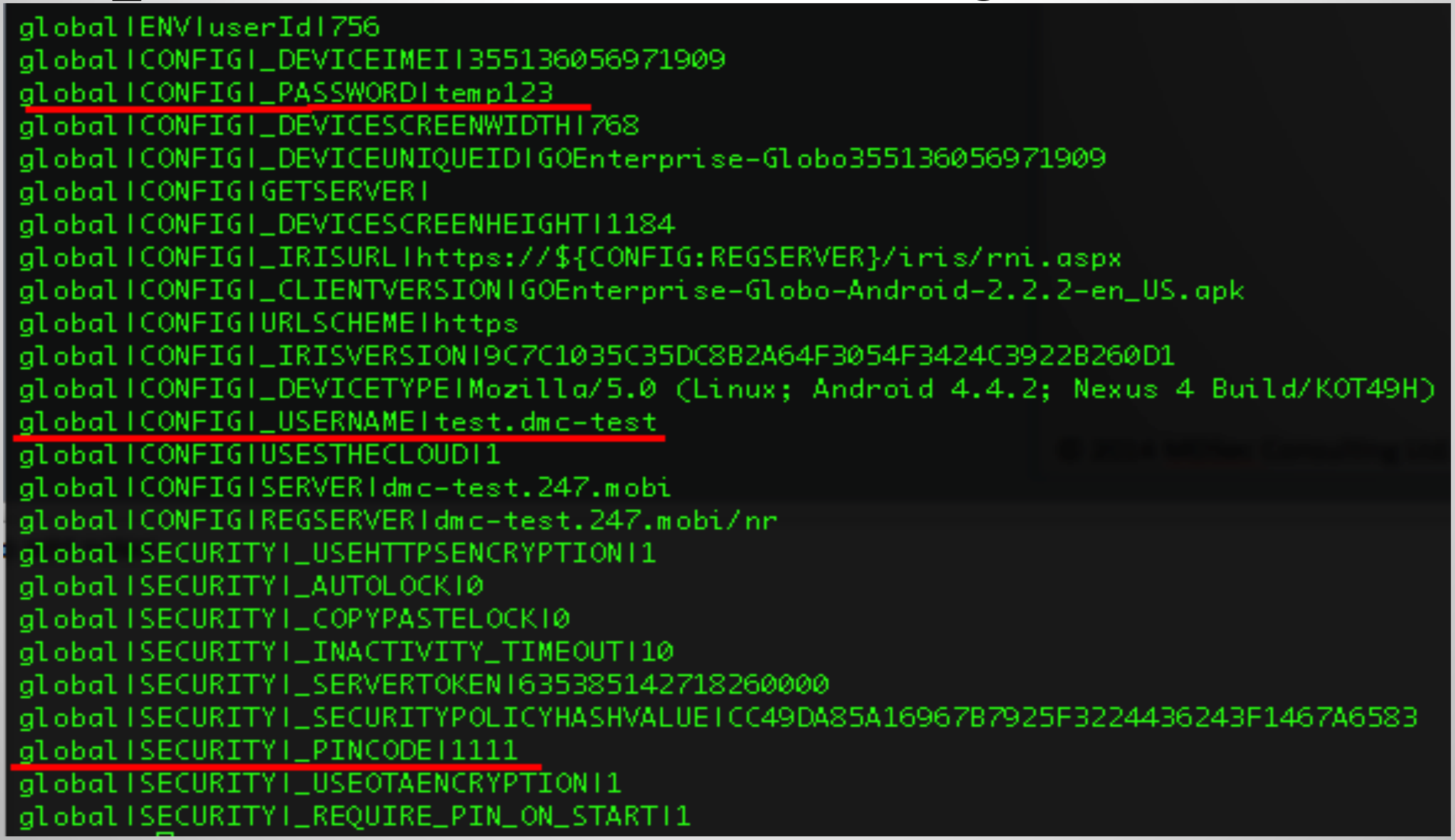


```
redpill:src dmc$ sqlcipher ./storage.db
SQLCipher version 3.7.17 2013-05-20 00:56:22
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> pragma key = '9a8d7940eeff5bc65bb8f004644f4957bed3e616a3251c462a7ce9a9caa9d0b6';
sqlite> .tables
_applicationpages          _vm_databases
_irisinfo                  _vm_databases_acl
_localization              _vm_db_startup_operations
_variables                 android_metadata
```

Case Study #3

GO!Enterprise

- The _variables table has some interesting data…



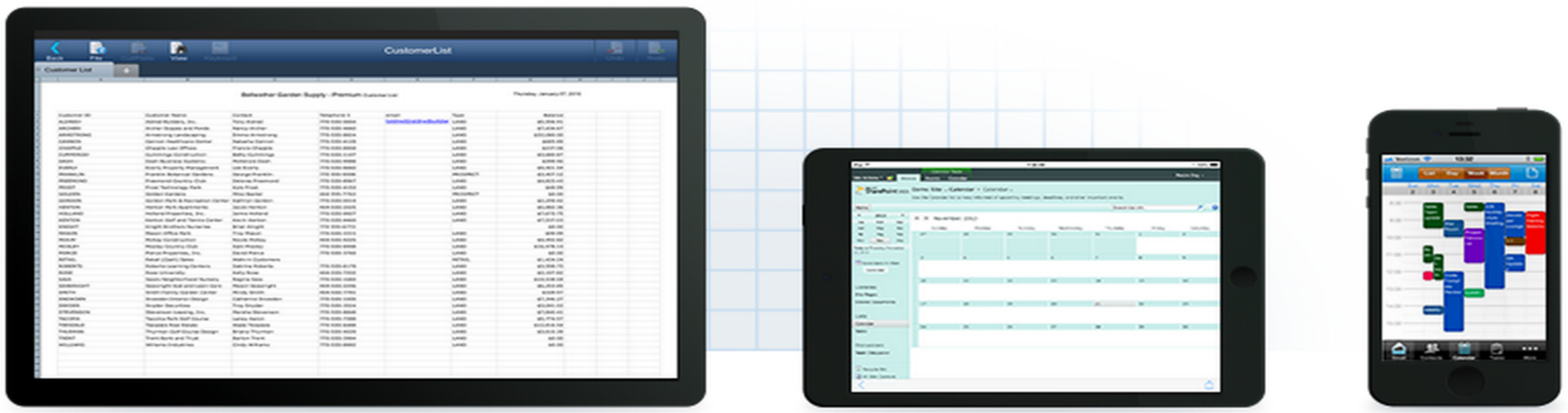
```
global|ENV|userId|756
global|CONFIG|_DEVICEIMEI|355136056971909
global|CONFIG|_PASSWORD|temp123
global|CONFIG|_DEVICESCREENWIDTH|768
global|CONFIG|_DEVICEUNIQUEID|GOEnterprise-Globo355136056971909
global|CONFIG|GETSERVER|
global|CONFIG|_DEVICESCREENHEIGHT|1184
global|CONFIG|_IRISURL|https://${CONFIG:REGSERVER}/iris/rni.aspx
global|CONFIG|_CLIENTVERSION|GOEnterprise-Globo-Android-2.2.2-en_US.apk
global|CONFIG|URLSCHEME|https
global|CONFIG|_IRISVERSION|9C7C1035C35DC8B2A64F3054F3424C3922B260D1
global|CONFIG|_DEVICETYPE|Mozilla/5.0 (Linux; Android 4.4.2; Nexus 4 Build/KOT49H)
global|CONFIG|_USERNAME|test.dmc-test
global|CONFIG|USESTHECLOUD|1
global|CONFIG|SERVER|dmc-test.247.mobi
global|CONFIG|REGSERVER|dmc-test.247.mobi/nr
global|SECURITY|_USEHTTPSENCRYPTION|1
global|SECURITY|_AUTOLOCK|0
global|SECURITY|_COPYPASTELOCK|0
global|SECURITY|_INACTIVITY_TIMEOUT|10
global|SECURITY|_SERVERTOKEN|635385142718260000
global|SECURITY|_SECURITYPOLICYHASHVALUE|CC49DA85A16967B7925F3224436243F1467A6583
global|SECURITY|_PINCODE|1111
global|SECURITY|_USEOTAENCRYPTION|1
global|SECURITY|_REQUIRE_PIN_ON_START|1
```

Case Study #4

Kaseya BYOD

Bring Your Own Device (BYOD) Management

With the Kaseya BYOD Suite, organizations can give employees the freedom to work on their personal devices, while ensuring the security of enterprise data and applications on those devices. The solution provides employees with access to corporate applications, email, and documents from their personal devices , while providing IT with peace of mind through unprecedented BYOD security and ease of deployment and administration.



Kaseya BYOD Suite running on different device types

Case Study #4

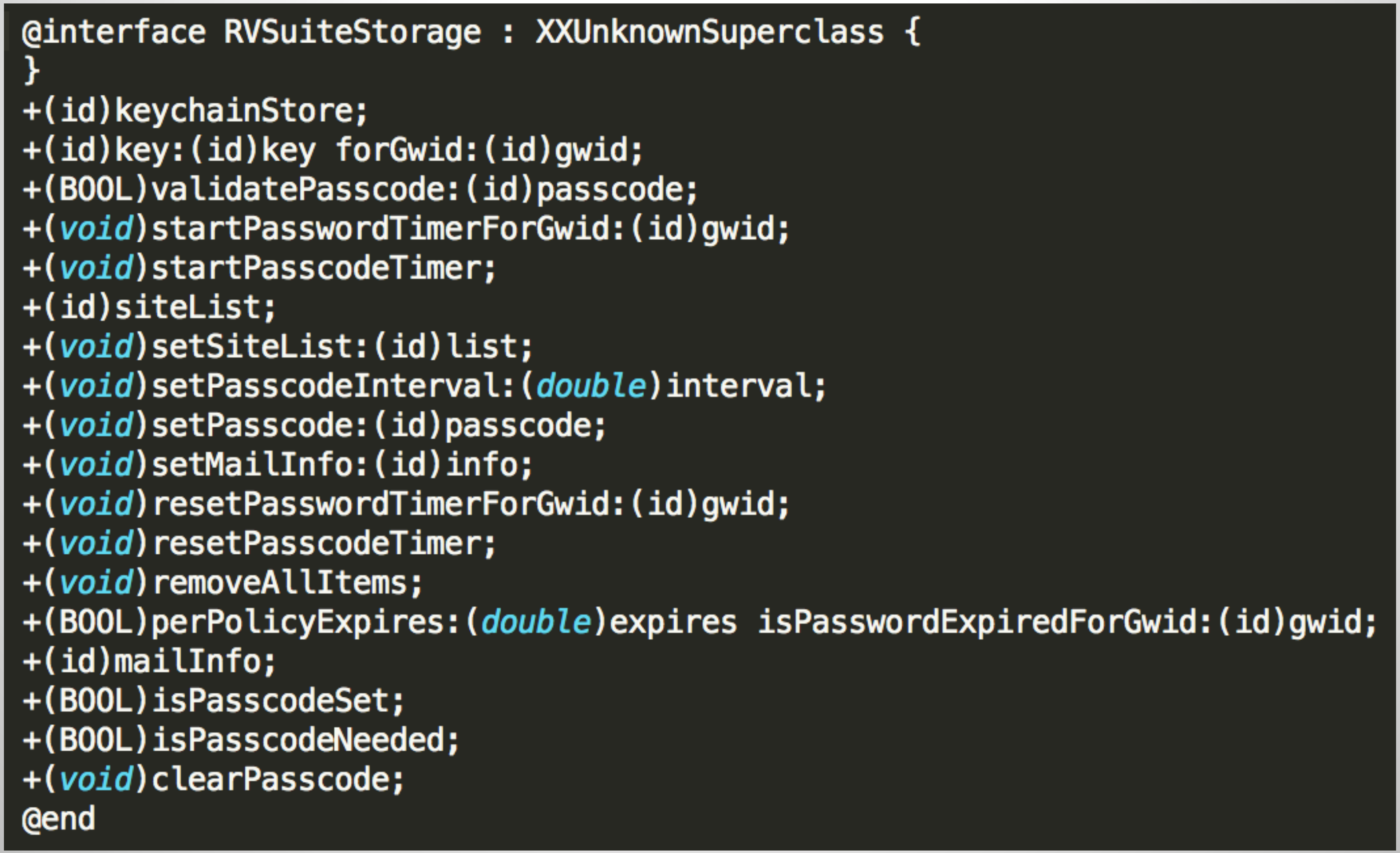
Kaseya BYOD

- BYOD application that provides access to documents, e-mail, and a browser
- Apps connect to a gateway that proxies to internal resources such as intranet applications and file shares
- Access to the app is protected via a PIN

Case Study #4

Kaseya BYOD

- Quickly identified an interesting class:



```
@interface RVSuiteStorage : XXUnknownSuperclass {
}
+(id)keychainStore;
+(id)key:(id)key forGwid:(id)gwid;
+(BOOL)validatePasscode:(id)passcode;
+(void)startPasswordTimerForGwid:(id)gwid;
+(void)startPasscodeTimer;
+(id)siteList;
+(void)setSiteList:(id)list;
+(void)setPasscodeInterval:(double)interval;
+(void)setPasscode:(id)passcode;
+(void)setMailInfo:(id)info;
+(void)resetPasswordTimerForGwid:(id)gwid;
+(void)resetPasscodeTimer;
+(void)removeAllItems;
+(BOOL)perPolicyExpires:(double)expires isPasswordExpiredForGwid:(id)gwid;
+(id)mailInfo;
+(BOOL)isPasscodeSet;
+(BOOL)isPasscodeNeeded;
+(void)clearPasscode;
@end
```

Case Study #4

Kaseya BYOD

Kaseya BYOD DEMO

Overview

- Introduced to the OWASP Mobile Top Ten at OWASP AppSec California in January 2014

- Attempts to achieve the following goals:
  - Prevent software operating in an untrusted environment
  - Thwart or increase the complexity of reverse engineering
  - Thwart or increase the complexity of modification or tampering attacks
  - Detect/Prevent attacks from on-device malware

- How common are these protections?
  - 2013 study by HP : "86 percent of applications tested ***lacked*** binary hardening"

Binary Protections
Overview

- So what are the risks?
  - Theft of Intellectual Property from reverse engineering
  - Circumvention of security controls; authentication, encryption, licensing, DRM, jailbreak/root detection
  - Loss of revenue from piracy
  - Brand/Reputational damage from app imitation and/or code modification

Binary Protections

Overview

- Some of the binary protections you may have encountered:
  - Jailbreak/Root detection
  - Resource and code integrity checksums
  - Anti-debugging
  - Runtime tamper protection
  - Obfuscation

- Not a silver bullet!

Jailbreak/Root Detection

- Attempts to detect if the application is running on a jailbroken or rooted device

- If a compromise is detected the app usually does one or more of:
  - Warn the user
  - Wipe any sensitive data
  - Report back to a management server
  - Exit / Crash

Jailbreak/Root Detection

- Jailbreak/Root detection implementations usually perform the following activities:
  - Examine the filesystem
  - Check open ports
  - Test sandbox restrictions
  - Permissions on memory pages
  - Evidence of modifications (e.g. build keys)

- Often trivial to bypass unless other protections are in place

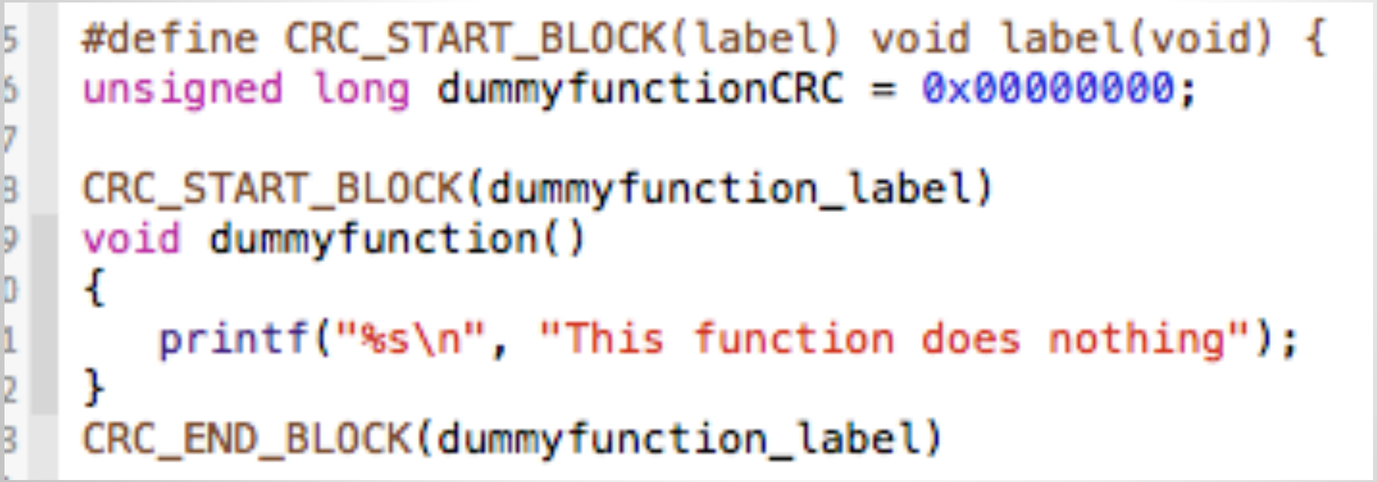
Binary Protections

Integrity Checksums

- Attempt to ensure that application resources or internal code structures haven't been modified or new code inserted
- If tampering is detected more often than not a crash is triggered
- Typically implemented by embedding a "web" of self validating checksum functions in to an application
- Checksum calculations performed on specific functions or across a class, as well as portions of the code segment

Integrity Checksums

- For native code can be implemented using C
  - Insert a label before and after the functions you want to checksum to get the function size

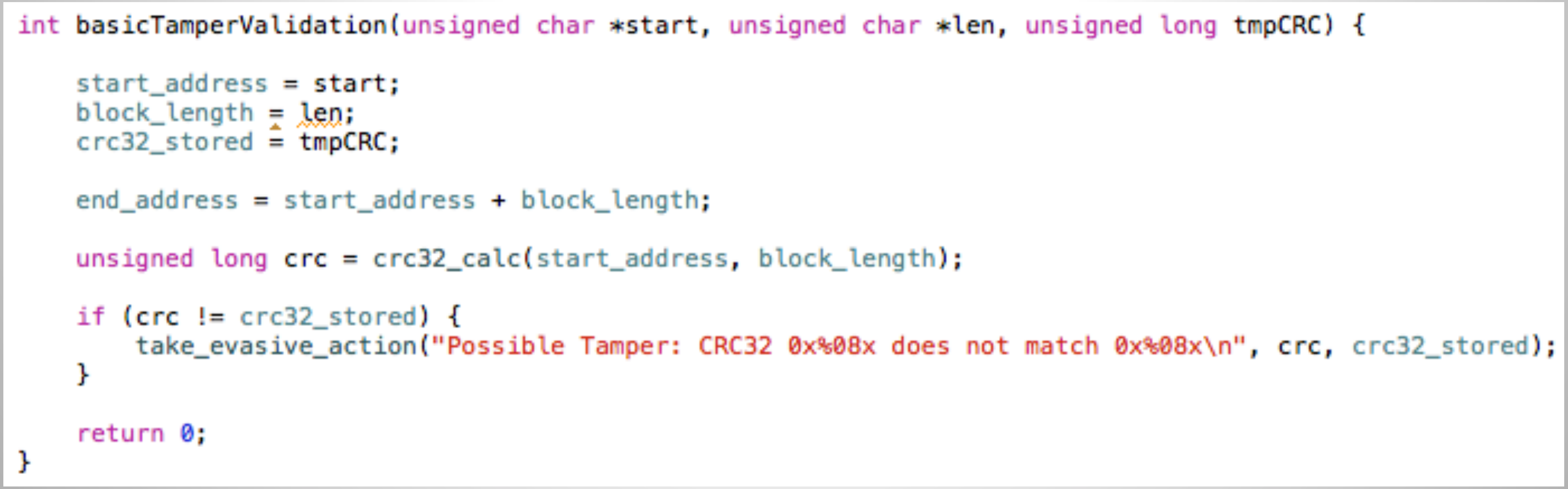


```
#define CRC_START_BLOCK(label) void label(void) {
unsigned long dummyfunctionCRC = 0x00000000;

CRC_START_BLOCK(dummyfunction_label)
void dummyfunction()
{
   printf("%s\n", "This function does nothing");
}
CRC_END_BLOCK(dummyfunction_label)
```

Binary Protections

Integrity Checksums

- A checksum can then be calculated based on the start address + the length and compared with a stored checksum
- Similar checks should be embedded across the code



```
int basicTamperValidation(unsigned char *start, unsigned char *len, unsigned long tmpCRC) {

    start_address = start;
    block_length = len;
    crc32_stored = tmpCRC;

    end_address = start_address + block_length;

    unsigned long crc = crc32_calc(start_address, block_length);

    if (crc != crc32_stored) {
        take_evasive_action("Possible Tamper: CRC32 0x%08x does not match 0x%08x\n", crc, crc32_stored);
    }

    return 0;
}
```

Integrity Checksums

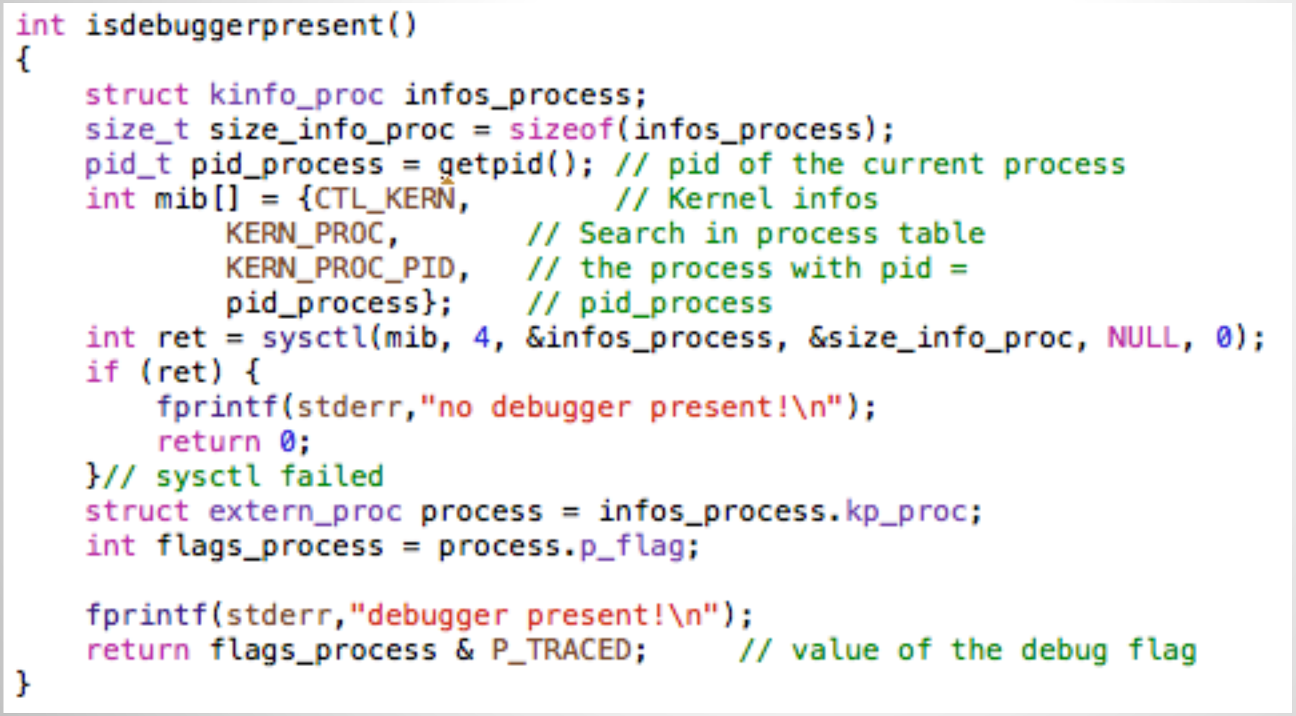
- There are several shortcomings in this method of implementation:
  - The application first needs to be run to calculate the stored CRC which is then embedded in to the code
  - The location of the checksums is difficult to randomize across builds
- A better but complex approach can be achieved using the LLVM compiler
  - During compilation the JIT engine can compile the functions that you want to protect
  - This can be used to calculate the relevant checksums then validation code can be embedded using the LLVM IR

Binary Protections
Anti-Debugging

- With a debugger an attacker is able to trivially manipulate application behavior
- For example, in iOS applications it is possible to simulate method calls to objects by invoking calls to `objc_msgSend`
- Anti-debugging protections attempt to detect and prevent a debugger being attached
- Unlikely to thwart an advanced adversary

Anti-Debugging

- On iOS the process status can be queried using sysctl



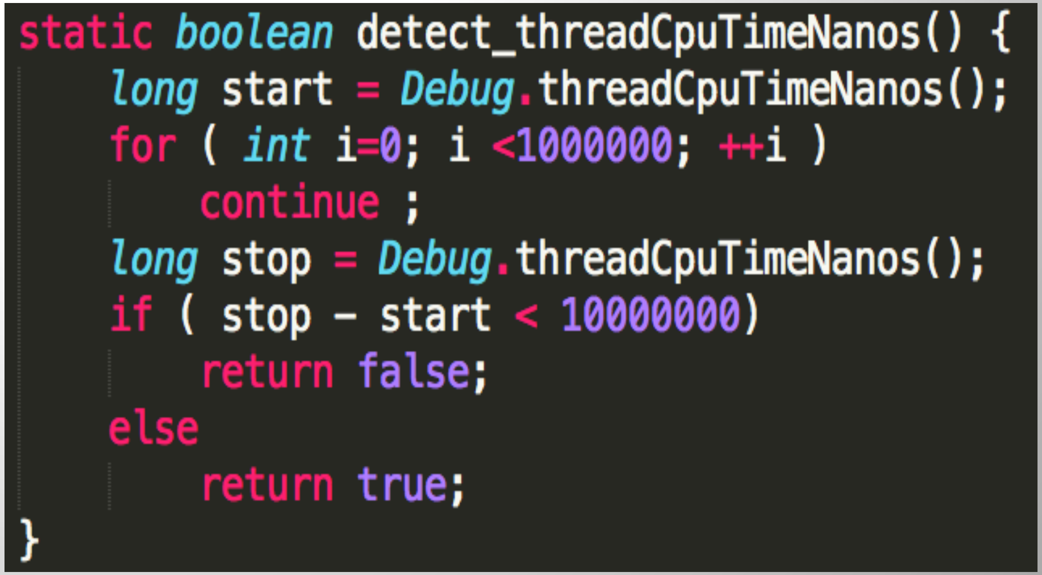
```
int isdebuggerpresent()
{
    struct kinfo_proc infos_process;
    size_t size_info_proc = sizeof(infos_process);
    pid_t pid_process = getpid(); // pid of the current process
    int mib[] = {CTL_KERN,        // Kernel infos
            KERN_PROC,        // Search in process table
            KERN_PROC_PID,    // the process with pid =
            pid_process};     // pid_process
    int ret = sysctl(mib, 4, &infos_process, &size_info_proc, NULL, 0);
    if (ret) {
        fprintf(stderr,"no debugger present!\n");
        return 0;
    }// sysctl failed
    struct extern_proc process = infos_process.kp_proc;
    int flags_process = process.p_flag;

    fprintf(stderr,"debugger present!\n");
    return flags_process & P_TRACED;      // value of the debug flag
}
```

- The `PT_DENY_ATTACH` flag can also be set

Binary Protections

Anti-Debugging

- Several common implementations for Android applications
- DVM has the `Debug.isDebuggerConnected` class
- Can also be read directly from the DVM via JNI rather than using the API
- Timing thread execution



```
static boolean detect_threadCpuTimeNanos() {
    long start = Debug.threadCpuTimeNanos();
    for ( int i=0; i <1000000; ++i )
        continue ;
    long stop = Debug.threadCpuTimeNanos();
    if ( stop - start < 10000000)
        return false;
    else
        return true;
}
```

Runtime Tamper Protection

- Frameworks like Cydia Substrate make hooking of the Objective-C or Dalvik runtimes trivial
- Allows an adversary or malware to invoke or modify internal methods
  - Bypass security controls
  - Leak/Steal sensitive data
- Fairly unique situation that a developer cannot trust their own runtime

Runtime Tamper Protection

- Attempts to determine whether functions have been hooked at runtime

- Several tricks for iOS that can help identify runtime tampering, but yet to see anything for Android DVM (this doesn't mean it doesn't exist! ☺)

Runtime Tamper Protection

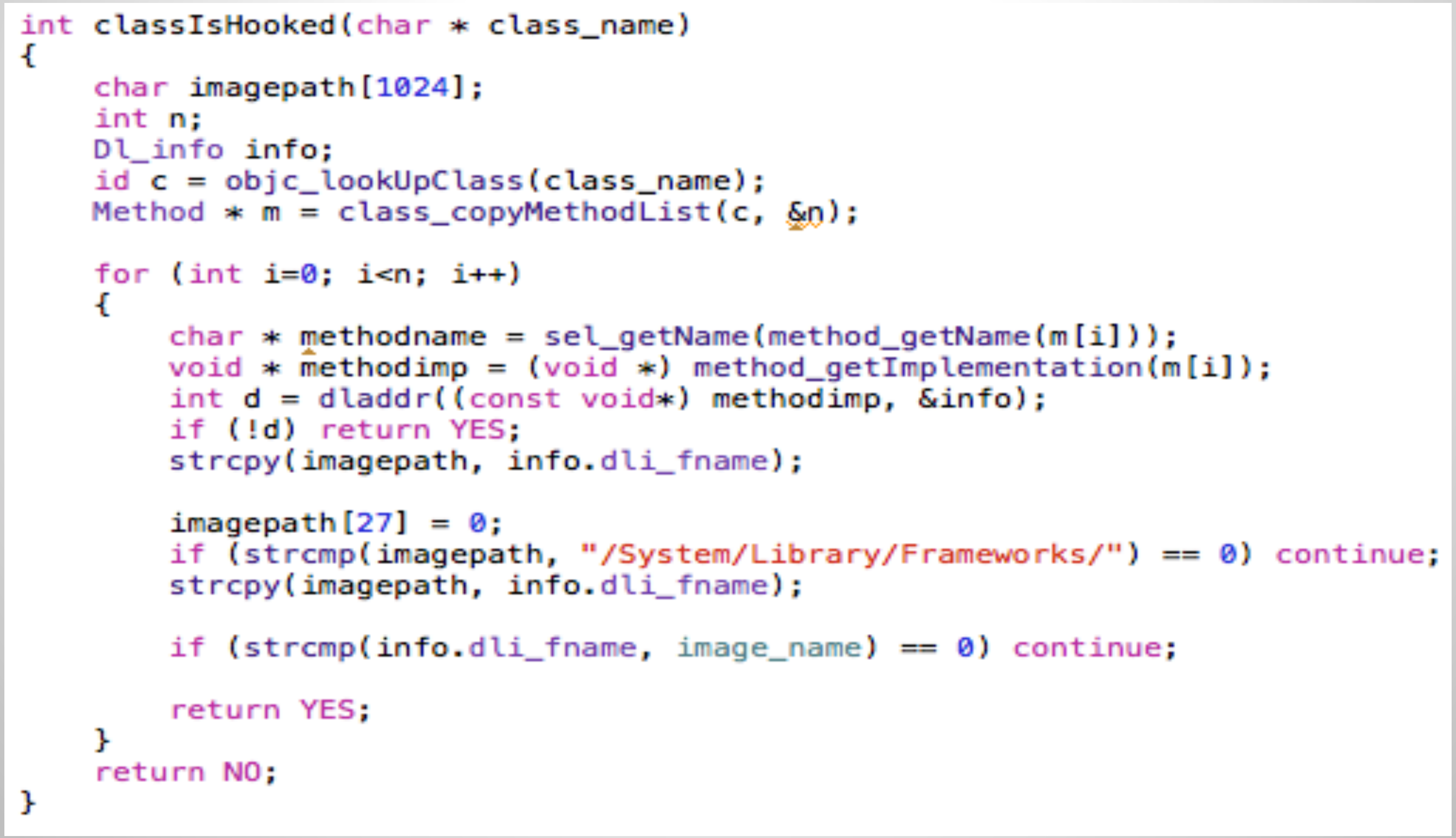
- Check #1 : Validating the source image location

- The locations for dylibs with the SDK methods is a finite set of directories:
  - /usr/lib
  - /System/Library/Frameworks
  - /System/Library/PrivateFrameworks
  - /System/Library/Accessibility
  - /System/Library/TextInput

- `Dladdr` takes a function pointer and returns details on the source image

Binary Protections
Runtime Tamper Protection

- Retrieve the image name and compare it to known values



```
int classIsHooked(char * class_name)
{
    char imagepath[1024];
    int n;
    Dl_info info;
    id c = objc_lookUpClass(class_name);
    Method * m = class_copyMethodList(c, &n);

    for (int i=0; i<n; i++)
    {
        char * methodname = sel_getName(method_getName(m[i]));
        void * methodimp = (void *) method_getImplementation(m[i]);
        int d = dladdr((const void*) methodimp, &info);
        if (!d) return YES;
        strcpy(imagepath, info.dli_fname);

        imagepath[27] = 0;
        if (strcmp(imagepath, "/System/Library/Frameworks/") == 0) continue;
        strcpy(imagepath, info.dli_fname);

        if (strcmp(info.dli_fname, image_name) == 0) continue;

        return YES;
    }
    return NO;
}
```

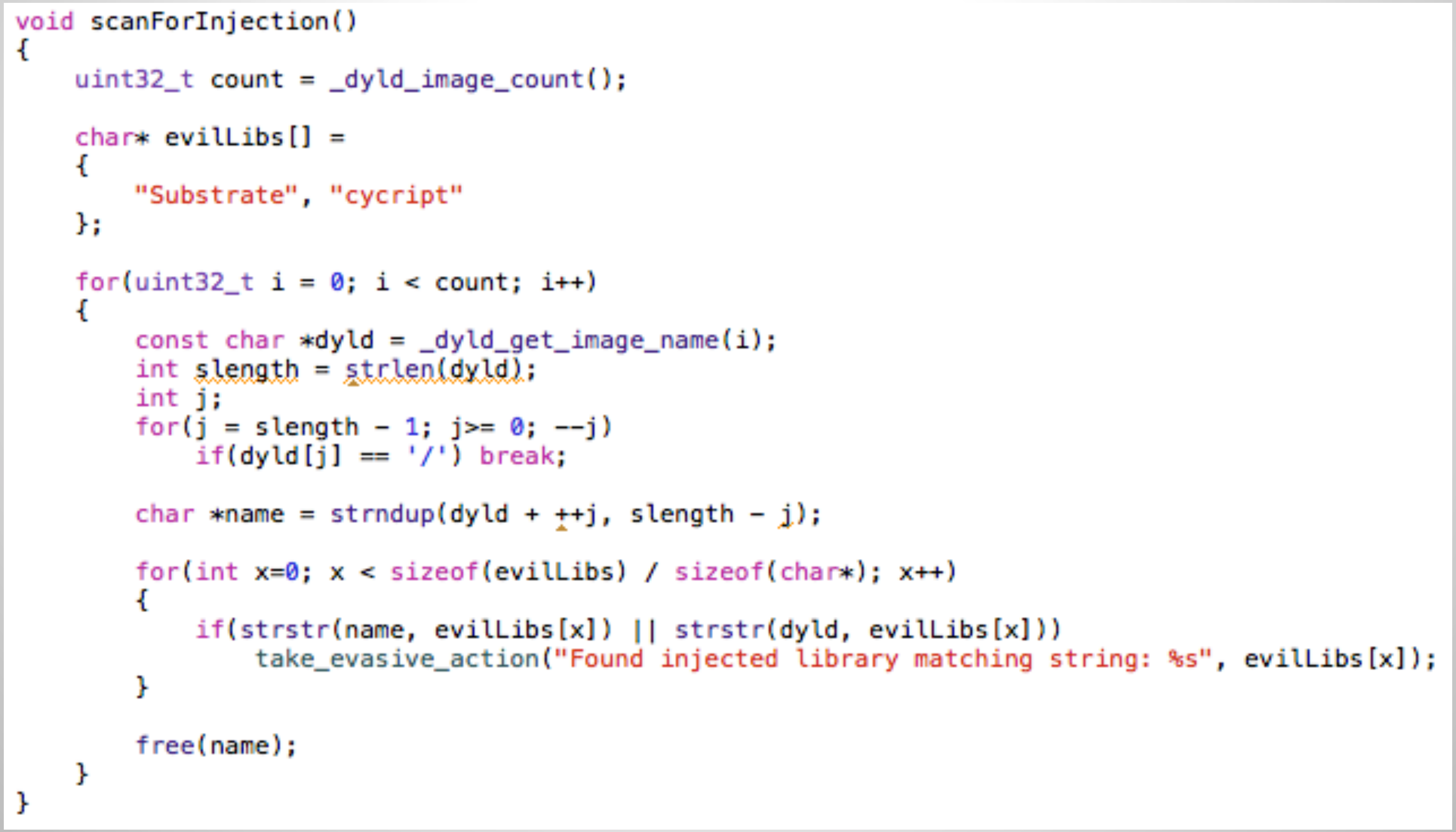
Runtime Tamper Protection

- Check #2: Scan for malicious libraries
- Cydia Substrate and Cycript will inject a dylib in to the process when it launches
- It's possible to iterate the list of loaded libraries and search for common jailbreak associated libraries such as "Substrate" and "cycript"

Binary Protections

Runtime Tamper Protection

- Get a list of linked libraries and scan for jailbreak strings

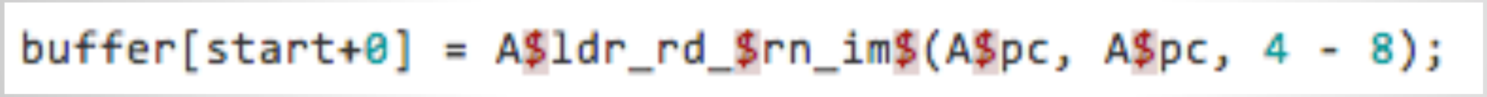


```
void scanForInjection()
{
    uint32_t count = _dyld_image_count();

    char* evilLibs[] =
    {
        "Substrate", "cycript"
    };

    for(uint32_t i = 0; i < count; i++)
    {
        const char *dyld = _dyld_get_image_name(i);
        int slength = strlen(dyld);
        int j;
        for(j = slength - 1; j>= 0; --j)
            if(dyld[j] == '/') break;

        char *name = strndup(dyld + ++j, slength - j);

        for(int x=0; x < sizeof(evilLibs) / sizeof(char*); x++)
        {
            if(strstr(name, evilLibs[x]) || strstr(dyld, evilLibs[x]))
                take_evasive_action("Found injected library matching string: %s", evilLibs[x]);
        }

        free(name);
    }
}
```

Runtime Tamper Protection

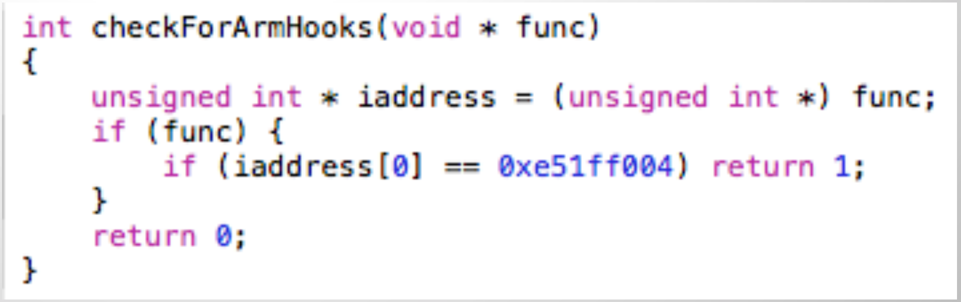
- Check #3: Check for Cydia Substrate patches

- Examining the code (see `SubstrateHookFunctionARM`) we can see what it does:



```
buffer[start+0] = A$ldr_rd_$rn_im$(A$pc, A$pc, 4 - 8);
```

- Trampoline is inserted, jumps to an absolute address
  – ldr pc, [pc, -0x4]



```
int checkForArmHooks(void * func)
{
    unsigned int * iaddress = (unsigned int *) func;
    if (func) {
        if (iaddress[0] == 0xe51ff004) return 1;
    }
    return 0;
}
```

Obfuscation

- Attempts to complicate reverse engineering by making it difficult or complex to understand
- Obfuscation typically achieves this by doing some or all of the following (and more!):
  - Obscure names of classes, fields and methods
  - Insert bogus code
  - Modify the control flow
  - Substitution of instructions
- Android comes with ProGuard for release builds, llvm-obfuscator is an opensource native code equivalent

Case Study #5

App Protection Product

- Reviewed a binary protection solution for a vendor
- Unfortunately work performed under NDA ;(
- The solution worked by embedding similar protections to those described, including runtime tampering, checksum protection etc to LLVM IR
- The protections worked an onion and each one needed to be pealed off one at a time, starting with the integrity checksumming

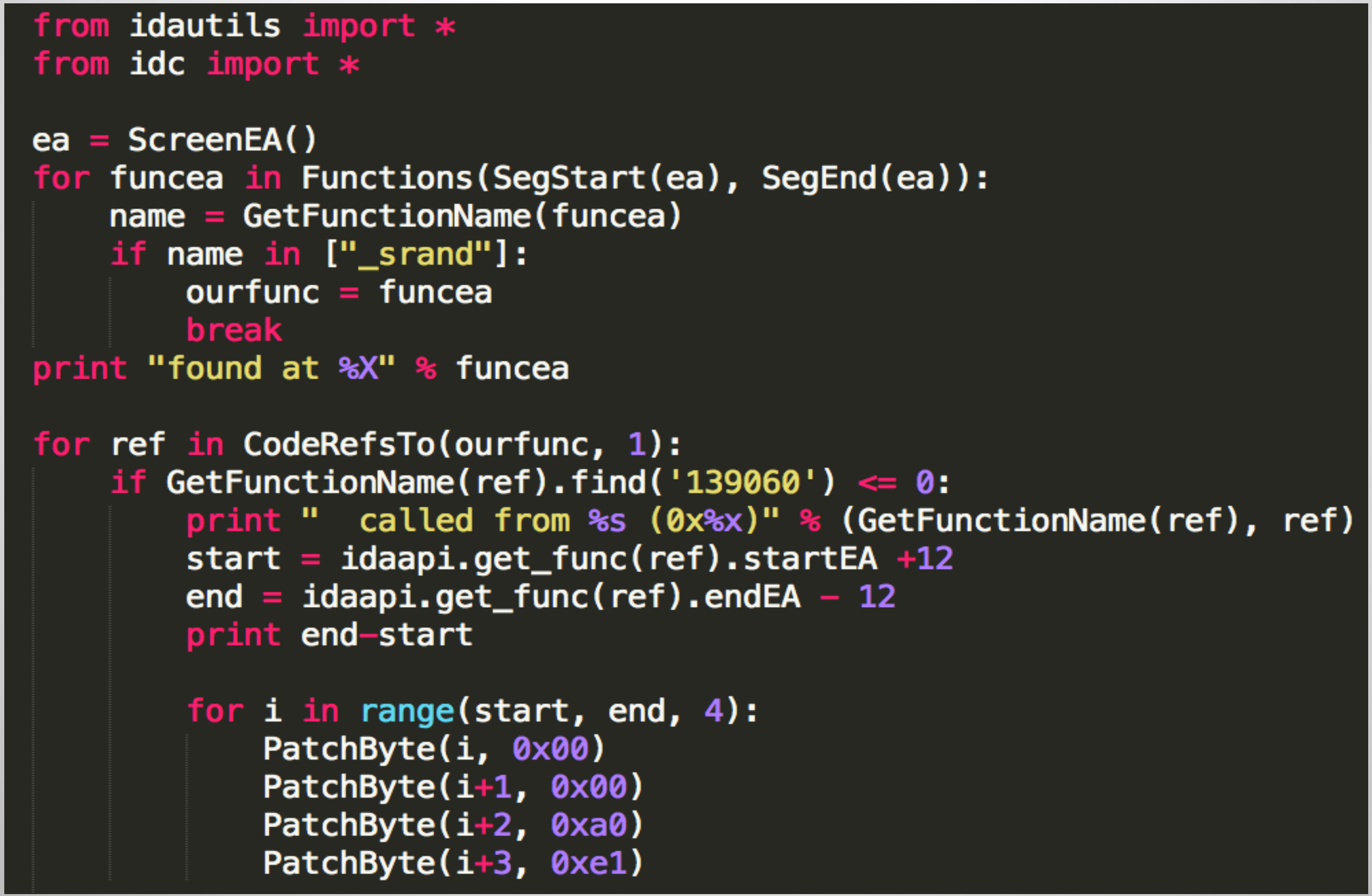
Case Study #5
App Protection Product

- Patching the binary and triggering a crash lead us to find some examples of the validation routine from the call stack
- Reversing some of these functions we found a common denominator, they all called `srand()`
- In theory, it should be possible to identify all of the checksumming functions by cross references to `srand()`

Case Study #5

App Protection Product

- IDAPython to the rescue!



```python
from idautils import *
from idc import *

ea = ScreenEA()
for funcea in Functions(SegStart(ea), SegEnd(ea)):
    name = GetFunctionName(funcea)
    if name in ["_srand"]:
        ourfunc = funcea
        break
print "found at %X" % funcea

for ref in CodeRefsTo(ourfunc, 1):
    if GetFunctionName(ref).find('139060') <= 0:
        print "  called from %s (0x%x)" % (GetFunctionName(ref), ref)
        start = idaapi.get_func(ref).startEA +12
        end = idaapi.get_func(ref).endEA - 12
        print end-start

        for i in range(start, end, 4):
            PatchByte(i, 0x00)
            PatchByte(i+1, 0x00)
            PatchByte(i+2, 0xa0)
            PatchByte(i+3, 0xe1)
```

Conclusions
Overview

MDSec
Setting The Standard.

- Secure doesn't always mean secure
- Binary protections aren't a silver bullet!
- Protections need to be layered

Q & A
That's all folks!

QUESTIONS?

- **Online**:
  - http://www.mdsec.co.uk
  - http://blog.mdsec.co.uk
  - https://github.com/mdsecresearch
- **E-Mail**:
  - dominic [at] mdsec [dot] co [dot] uk
- **Twitter:**

  @domchell

  @MDSecLabs