



# GOTO: HACK



# Practical iOS Applications Hacking

Mathieu RENARD - @GOTOHACK  
mathieu.renard[-at-]gotohack.org  
mathieu.renard[-at-]sogeti.com



\* RIG, S1, S2 MOUNTED ON PANEL  
K AT D1 = CATHODE

U1—TL082 DUAL OP AMP [RS 275-1715]  
U2—1A, 50 PIV BRIDGE RECTIFIER [RS 276-1161]  
S1—ROTARY 5 POSITION, 2 POLE [RS 275-1385]  
S2—TOGGLE SWITCH, CPDT. [RS 275-625]  
C2, C4—5-50 pF OR 12-100 pF [RS 272-1340 OR MOUSER 21AA067]  
D1—1N4733 5.1 V, 1 W [RS 276-565]  
ALL RESISTORS—1/4 W, 1% (PART OF RS 271-309)

Hack To Learn...

# Agenda

- # Regular devices
- # Let's Jailbreak our device
- # Reversing iOS Applications
- # Hooking iOS Applications
- # The Truth about Jailbreak detection
- # Security Worst Practices
- # Defensives Measures



## Working with a regular device [Not jailbroken]



# Attack vectors : Regular device

**USB: AFC**

Applications

**Network**

Bluetooth

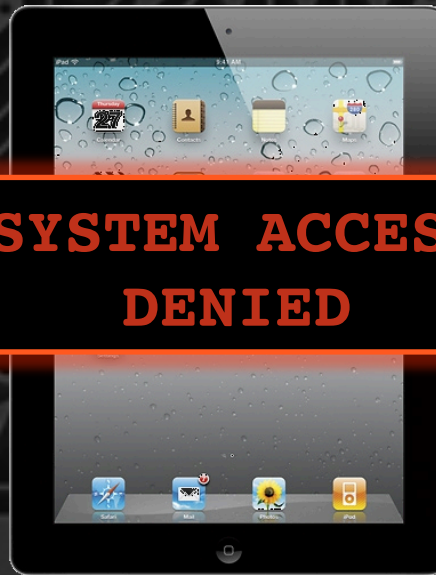
**SYSTEM ACCESS  
DENIED**

WiFi

Simcard

**Backups**

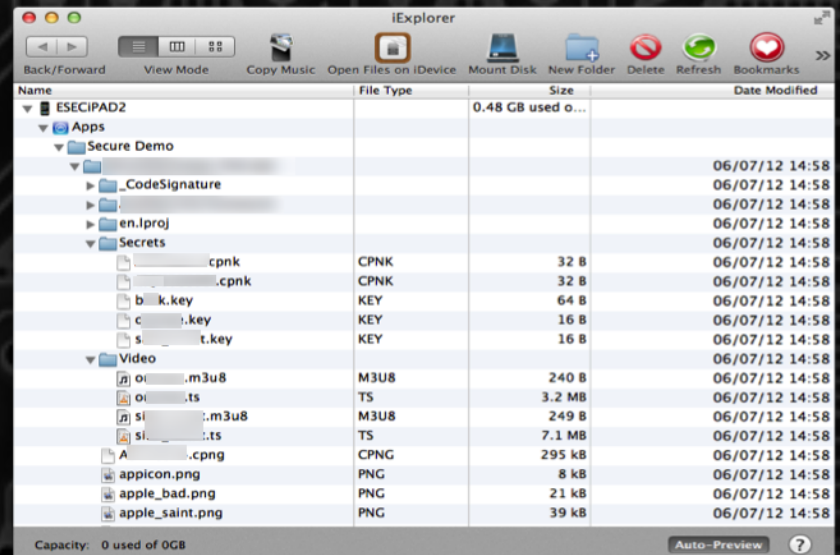
Baseband



# Abusing AFC protocol

- # AFC (Apple File Connection)
  - Service running on all iDevices
  - Handled by `/usr/libexec/afcd`
  - Used by iTunes to exchange files
  - AFC clients can access certain files only
    - Files located in the Media folder
    - User installed applications folders
  - Implemented in `libiMobileDevice`

- # What you can do
  - Access to default pref file
  - Access app resources
  - **Only if the iDevice unlocked**

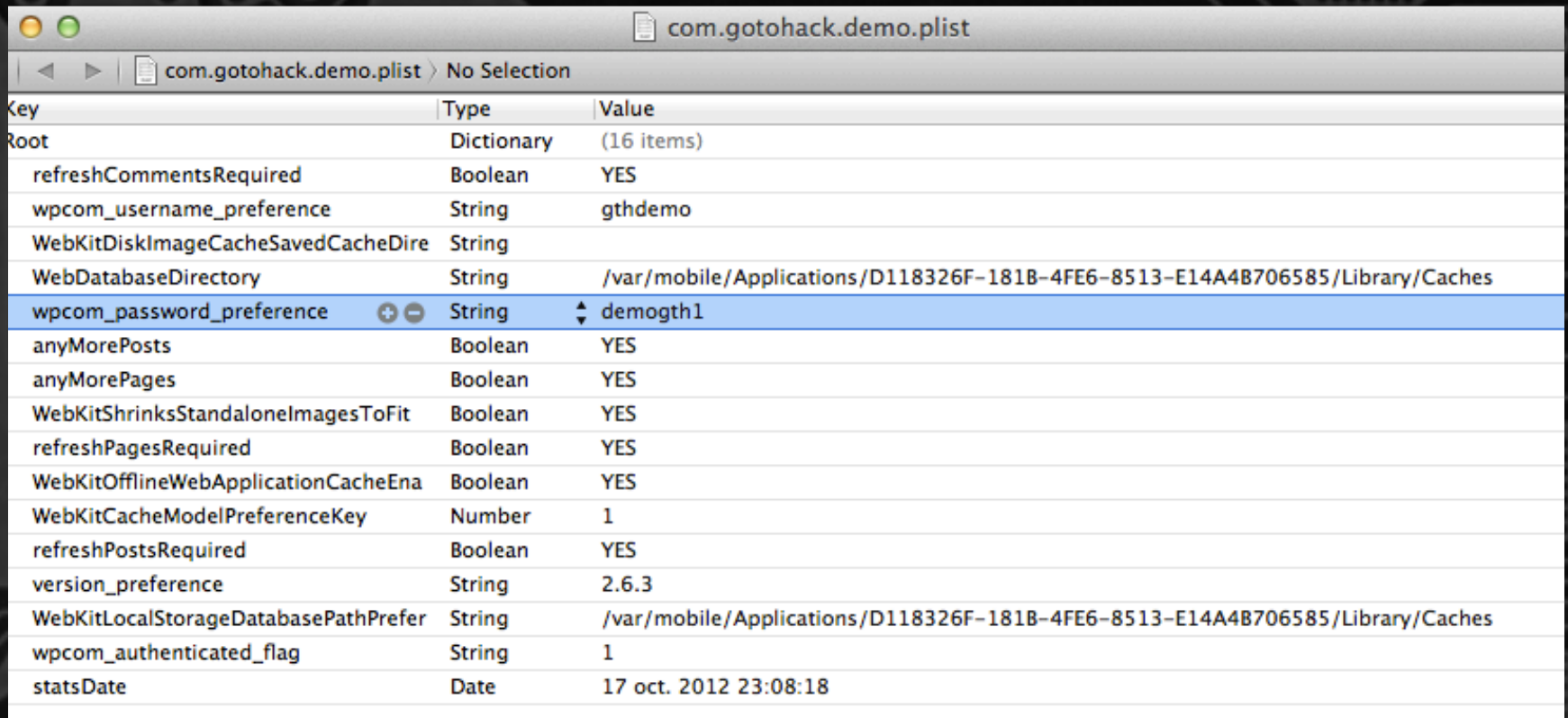


# iPown Dock & Evil Maid...

# DEMO



# Unsecure credential storage



The image shows a screenshot of a macOS-style window titled "com.gotohack.demo.plist". The window displays a table of key-value pairs from a plist file. The key "wpcom\_password\_preference" is highlighted in blue, showing its value as "demogth1".

Key	Type	Value
Root	Dictionary	(16 items)
refreshCommentsRequired	Boolean	YES
wpcom_username_preference	String	gthdemo
WebKitDiskImageCacheSavedCacheDire	String	
WebDatabaseDirectory	String	/var/mobile/Applications/D118326F-181B-4FE6-8513-E14A4B706585/Library/Caches
wpcom_password_preference	String	demogth1
anyMorePosts	Boolean	YES
anyMorePages	Boolean	YES
WebKitShrinksStandaloneImagesToFit	Boolean	YES
refreshPagesRequired	Boolean	YES
WebKitOfflineWebApplicationCacheEna	Boolean	YES
WebKitCacheModelPreferenceKey	Number	1
refreshPostsRequired	Boolean	YES
version_preference	String	2.6.3
WebKitLocalStorageDatabasePathPrefer	String	/var/mobile/Applications/D118326F-181B-4FE6-8513-E14A4B706585/Library/Caches
wpcom_authenticated_flag	String	1
statsDate	Date	17 oct. 2012 23:08:18



# Having fun with backups

## # Backup storage

- %APPDATA%/Apple Computer/MobileSync/Backup/<udid>
- Can be password protected
- Encrypted (AES-256 CBC)
- Filenames : SHA1 hashes

## # Using iPhoneDataProtection Framework

- Developed by Jean SIGWALD – Sogeti ESEC Lab
- Bruteforce backup password [require some scripting skills] [ Extremely slow ]
  - I do recommend Elcomsoft Phone Password Breaker (35 000 pwd/s on GPU)
- Extract backup content
- Extract keychain stored data





# Having Fun With backups

```
python_scripts — bash — 113x32
java bash Python ...
(~/iphonetools/iphone-dataprotection/python_scripts) $ python backup_tool.py /Users/DrK/Library/Application\ Supp
ort/MobileSync/Backup/94b707/ /Users/DrK/Desktop/
Device Name :
Display Name :
Last Backup Date :
IMEI : 012 ;78
Serial Number : DN6C J1
Product Type : iPad2,2
Product Version : 5.0.1
iTunes Version : 10.5.1
Extract backup to /Users/DrK/Desktop/ ? (y/n)
y
Backup is encrypted
Enter backup password :
test
Writing SystemConfiguration/VPN-net.juniper.sslvpn.plist
Writing Library/Preferences/com.apple.voiceservices.plist
Writing Library/SpringBoard/IconState.plist
Writing Library/Notes/notes.sqlite
Writing Documents/branding.bundle/Root.plist
Writing Library/Preferences/com.imesart.USBDisk-iPad.plist
Writing Documents/MIClient.sqlite
Writing Library/SpringBoard/LockBackground.cpbitmap
Writing Library/Voicemail/voicemail.db
Writing SystemConfiguration/com.apple.PowerManagement.plist
Writing Library/Preferences/com.apple.accountsettings.plist
Writing Library/Calendar/Calendar.sqlitedb
Writing Media/PhotoData/MISC/PreviewWellImage.tiff
Writing Library/Preferences/com.apple.imessage.plist
Writing Library/ConfigurationProfiles/VPN_1007+1365626610.stub
Writing Library/ConfigurationProfiles/PublicInfo/MCMeta.plist
Writing Library/ConfigurationProfiles/EXCHANGE_1004+195116611.stub
```



# iOS Keychain

# Almost the only place to store critical data:

- Crypto keys
- Credentials
- ...

# Apple defined 6 values to define when a keychain item should be readable

- **kSecAttrAccessibleAfterFirstUnlock**
- **kSecAttrAccessibleAfterFirstUnlockThisDeviceOnly**
- **kSecAttrAccessibleAlways**
- **kSecAttrAccessibleAlwaysThisDeviceOnly**
- **kSecAttrAccessibleWhenUnlocked**
- **kSecAttrAccessibleWhenUnlockedThisDeviceOnly**



# iOS Keychain

## # Protection class for built-in application items

Item	Accessibility
Wi-Fi passwords	Always
IMAP/POP/SMTP accounts	AfterFirstUnlock
Exchange accounts	Always
VPN	Always
LDAP/CalDAV/CardDAV accounts	Always
iTunes backup password	WhenUnlockedThisDeviceOnly
Device certificate & private key	AlwaysThisDeviceOnly

Can be extracted  
without jailbreak

Extraction requires the 0x835 hardware key => Jailbreak is mandatory



# Extracting Keychain data

```
python_scripts — bash — 113x32
java bash Python ...
~/iphonetools/iphone-dataprotection/python_scripts) $ python keychain_tool.py -d /Users/DrK/Desktop/backup/
/keychain-backup.plist /Users/DrK/Desktop/backup//Manifest.plist
If you have key835 for device 94b07 enter it (in hex)

-----
                          Passwords
-----
Service :      AirPort
Account :      Livebox-
Password :     FC6
Agrp :  apple

-----
Service :      Apple-token-sync
Account :
Password :
Agrp :  appleaccount

-----
Service :      com.apple.certui
Account :      https: mobsync. .com - 595d8
Password :     <binary plist data>
Agrp :  apple

-----
Server :      :0
Account :      DataAccess-7
Password :     Azerty1$
```

**WIFI KEY**

**APPLE TOKEN**

**APPLE CERT (PUSH MSG)**

**MAIL ACCOUNT**



# Analyzing network connexion

## # Remote virtual interface

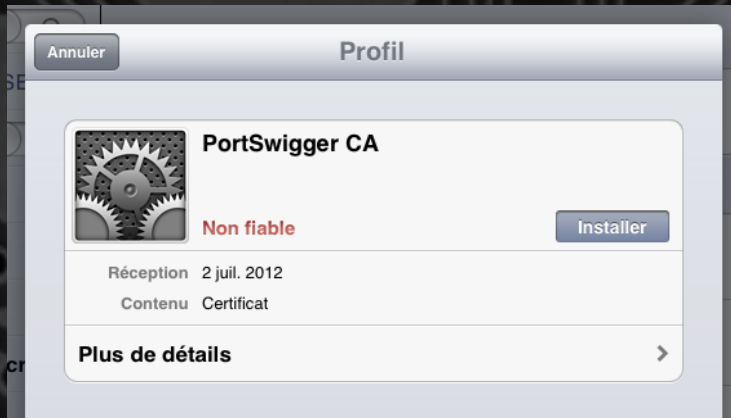
- When enabled all network traffic is mirrored to this interface
  - No need to jailbreak the device
  - Does not allow SSL interception
- Mac OS
  - Connect the device over usb
  - Get the device ID
  - Launch `rvictl -s <UID>`
  - Launch wireshark on the newly created network device

```
$ tcpdump -n -t -i rvi0 -q tcp
tcpdump: WARNING: rvi0: That device doesn't support promiscuous mode
(BIOCPRMISC: Operation not supported on socket)
tcpdump: WARNING: rvi0: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on rvi0, link-type RAW (Raw IP), capture size 65535 bytes
IP 192.168.1.66.55101 > 192.168.1.64.51712: tcp 117
IP 192.168.1.64.51712 > 192.168.1.66.55101: tcp 0
```



# HTTPS traffic interception

- # Like other web applications
  - Launch your proxy (Burp, Charles, Paros,...)
  - Setup the proxy on the device
  - If the application check for certificate validity
  - Extract your proxy CA and install it on the device
    - Link-it on a web page
    - Download the CA and install it





# GOTO: HlaICK



## Let's Jailbreak\* the device



\*BIG UP For the Jailbreak dream team!

Hack To Learn....

# Attack vectors : Jailbroken device

USB: AFC

Applications

Network

Bluetooth



WiFi

Simcard

Backups

Baseband





# Jailbroken device

## # Jailbreaking allows

- root access to the operating system
- downloading & installing new apps
  - Additional applications (ssh, gdb, ...)
  - Retrieve application and data stored on the device
  - Retrieve all data stored in the Keychain
    - We can extract the 0x835 hardware key
- Decrypting and reversing the application



# Getting the 0x835 Key

## # Getting 0x835 key on jailbroken device

- Kernel\_patcher
  - By default accessing to the hardware keys form user land is forbidden;
- Device\_info
  - Extracting hardware keys

```
python_scripts — ssh — 106x39
Python ... ssh
~/iphonetools/iphone-dataprotection/python_scripts) $ scp -P 2222 ../../iphone/* root@127.0.0.1:~/
root@127.0.0.1's password:
device_infos                               100% 60KB 60.3KB/s 00:00
kernel_patcher                             100% 13KB 13.2KB/s 00:00
~/iphonetools/iphone-dataprotection/python_scripts) $ ssh -p 2222 root@127.0.0.1
root@127.0.0.1's password:
iPad:~ root# chmod +x device_infos
iPad:~ root# chmod +x kernel_patcher
iPad:~ root# ./kernel_patcher
```





## Reversing iOS Applications



# iOS Binaries : ARM



# ARM7



# ARM7



# ARM7s

# RISC

# Load-store architecture

# 32-bit (ARM) & 16-bit (Thumb) instruction sets

# Registers

- R0-R3 > Used to pass params
- R7 > Frame pointer
- R13 > SP, Stack Pointer
- R14 > LR, Link register
- R15 > PC, Program counter

# CPSR Current Program Status Register

- N > Negative
- Z > Zero
- C > Carry
- V > Overflow



# iOS Binaries : Fat & Thin

- # Some executable are fat binaries
  - They contain multiple mach objects within a single file
    - Each one for a different architecture or platform

```
demo — bash — 80x18
bash bash bash
(~/Desktop/demo) $ otool -f demoFat
Fat headers
fat_magic 0xcfebab
nfat_arch 2
architecture 0
  cputype 12
  cpusubtype 6
  capabilities 0x0
  offset 4096
  size 2069824
  align 2^12 (4096)
architecture 1
  cputype 12
  cpusubtype 9
  capabilities 0x0
  offset 2076672
  size 1894896
  align 2^12 (4096)
```

No need to reverse both objects  
Lipo can convert a universal binary to a single architecture file, or vice versa.



# iOS Binaries : Mach-O

## # Contains three parts

- Header
- Load commands
- Data

## # Header

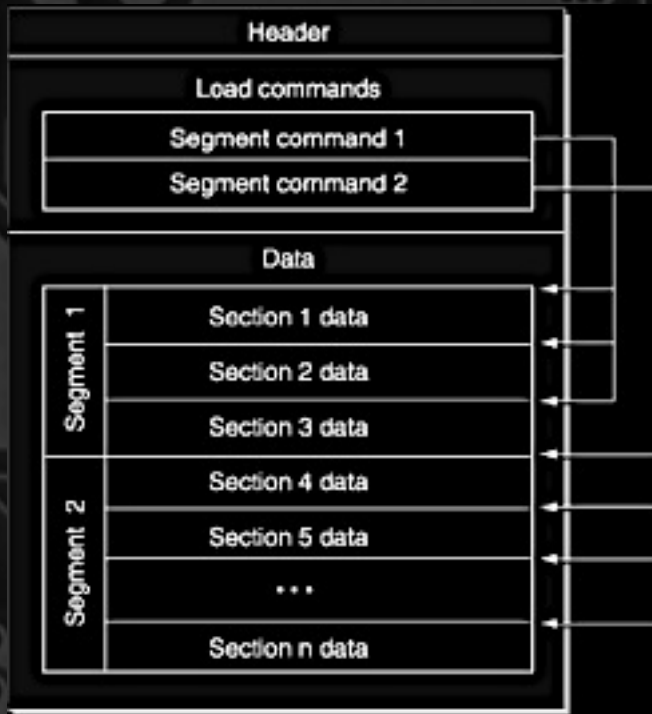
- Magic
- Cputype
- Cpusubtype
- Filetype
- Ncmds
- Sizeofcmds
- Flags

## ¶ Data

- Segments sections
- \_\_\_PAGEZERO
- \_\_\_TEXT
- \_\_\_DATA
- Rw-
- **\_\_\_OBJC**
- ...

## ¶ Load commands

- Indicates memory layout
- Locates symbols table
- Main thread context
- Shared libraries



# iOS Binaries : Cryptid

# Load commands & cryptid

```
demo — bash — 80x24
bash bash bash
(~/Desktop/demo) $ otool -l demoCryptid | grep -i -5 Cryptid
demoCryptid (architecture armv6):
Load command 0
  cmd LC_SEGMENT
  cmdsize 56
  segname __PAGEZERO
  vmaddr 0x00000000
--
--
Load command 11
  cmd LC_ENCRYPTION_INFO
  cmdsize 20
  cryptoff 8192
  cryptsize 1671168
  cryptid 1
Load command 12
  cmd LC_LOAD_DYLIB
  cmdsize 88
  name /System/Library/Frameworks/AVFoundation.framework/AVFoundation (of
fset 24)
  time stamp 2 Thu Jan 1 01:00:02 1970
--
--
Load command 38
```



# Apple Fairplay Encryption

- # The mandatory way to install applications is through Apple's App Store.
  - Application bundle is downloaded and stored in a zip archive
  - Zip file contains
    - The application itself (the binary),
    - Data files, such as images, audio tracks, or databases, and
    - Meta-data related to the purchase.
- # All Apple Store published applications are encrypted
  - When an application is synchronized onto the mobile device,
    - iTunes extracts the application folder and stores it on the device.





# Defeating Fairplay Encryption

## # Manually using GDB

- Launch GDB
- Set a breakpoint
- Run the application
- Extract the unencrypted executable code
- Patch the architecture specific binary

```
$CryptSize=1671168
```

```
$CryptOff=8192
```

```
echo -e "set sharedlibrary load-rules \".*\\" \".*\\" none\r\n\  
set inferior-auto-start-dyld off\r\n\  
set sharedlibrary preload-libraries off\r\n\  
set sharedlibrary load-dyld-symbols off\r\n\  
dump memory dump.bin $(( $CryptOff + 4096 )) $(( $CryptSize + $CryptOff + 4096 ))\r\n\  
kill\r\n\  
quit\r\n" > batch.gdb
```

```
gdb -q -e demoCryptId -x batch.gdb -batch
```



# Defeating Fairplay Encryption

## # Lamers way : Using Crackulous (Angel)

- With only one click
  - Decrypt apps & Unset CryptID
  - Provide fully functional cracked ipa
  - Generate credit file.
  - Automatic uploading
  - Automatic submission
- Bug
  - **Does not handle Thin binaries**



# Defeating Fairplay Encryption

## # The smart way : Dumpdecrypted (i0n1c)

```
iPod:~ root# DYLD_INSERT_LIBRARIES=dumpdecrypted.dylib /var/mobile/Applications/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/Scan.app/Scan
mach-o decryption dumper

DISCLAIMER: This tool is only meant for security research purposes, not for application
crackers.

[+] Found encrypted data at address 00002000 of length 1826816 bytes - type 1.
[+] Opening /private/var/mobile/Applications/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/Scan.app/Scan for reading.
[+] Reading header
[+] Detecting header type
[+] Executable is a FAT image - searching for right architecture
[+] Correct arch is at offset 2408224 in the file
[+] Opening Scan.decrypted for writing.
[-] Failed opening. Most probably a sandbox issue. Trying something different.
[+] Opening /private/var/mobile/Applications/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/tmp/Scan.decrypted for writing.
[+] Copying the not encrypted start of the file
[+] Dumping the decrypted data into the file
[+] Copying the not encrypted remainder of the file
[+] Closing original file
[+] Closing dump file
```



# Analyzing `__OBJC` Segment

## # `__OBJC`

- `__objc_classlist` : list of all classes for which there is an implementation in the binary.
- `__objc_classref` : references to all classes that are used by the application.

# By parsing these section it is possible to retrieve classes and methods prototypes



# Introducing Classdump

```
@property(nonatomic) __weak UITextField *textPass1; // @synthesize textPass1=_textPass1;
- (void).cxx_destruct;
- (void)authentifierAndChangeView:(id)arg1;
- (void)didReceiveMemoryWarning;
- (void)viewDidLoad;
- (BOOL)isPasswordValid:(id)arg1;
- (id)mainviewController;
- (id)dao;

@end























@interface SqliteDao : NSObject
{
    struct sqlite3 *database;
    NSString *databasePath;
}

@property struct sqlite3 *database; // @synthesize database;
@property(retain) NSString *databasePath; // @synthesize databasePath;
- (void).cxx_destruct;
- (void)closeDatabase;
- (int)openCryptoDatabase:(id)arg1;
- (void)addUser:(id)arg1:(id)arg2:(id)arg3:(id)arg4:(id)arg5:(id)arg6;
- (id)getPassword:(id)arg1;
- (int *)getNombreUsers;

@end
```



# Introducing IDA Pro

Function name	Segment
 <code>-[WebSocket didOpen]</code>	<code>__text</code>
 <code>-[WebSocket sendMessage:]</code>	<code>__text</code>
 <code>-[WebSocket didReceiveMessage:]</code>	<code>__text</code>
 <code>-[WebSocket didClose]</code>	<code>__text</code>
 <code>-[WebSocket socket:didReadData:withTag:]</code>	<code>__text</code>
 <code>-[WebSocket socketDidDisconnect:withError:]</code>	<code>__text</code>
 <code>-[WebSocket websocketQueue]</code>	<code>__text</code>
 <code>sub_28A4C</code>	<code>__text</code>
 <code>sub_28BFC</code>	<code>__text</code>
 <code>-[HTTPConnection(SecureKey) httpRespons...</code>	<code>__text</code>
 <code>+[SECCrypt encryptData:key:iv:source:error:]</code>	<code>__text</code>
 <code>+[SECCrypt decryptData:key:iv:source:error:]</code>	<code>__text</code>
 <code>-[SECHttpServer setSecret:]</code>	<code>__text</code>
 <code>-[SECHttpServer dealloc]</code>	<code>__text</code>
 <code>-[SECHttpServer getKey:]</code>	<code>__text</code>
 <code>+[SECKeychain initialize]</code>	<code>__text</code>
 <code>+[SECKeychain mapServiceName:]</code>	<code>__text</code>
 <code>+[SECKeychain getKeychainQuery:]</code>	<code>__text</code>
 <code>+[SECKeychain save:data:]</code>	<code>__text</code>
 <code>+[SECKeychain load:]</code>	<code>__text</code>
 <code>+[SECKeychain delete:]</code>	<code>__text</code>
 <code>+[SECKeys getKeyForFile:ofKind:withSecret:]</code>	<code>__text</code>

Line 941 of 1345



# Objective-C

## # Calling convention

- C++
  - `ObjectPointer->Method(param1, param2)`
- Objective-C
  - `[ObjectPointer Method:param1 param2Name:param2`
    - `objc_msgSend(ObjectPointer, @selector(Method))`
- ARM calling convention
  - Arg1: `ObjectPointer` → r0
  - Arg2: `@selector(Method)` → r1
- Backtracing calls to `objc_msgSend`
  - By hand
  - Using Zynamics IDAPython script or IDA Pro > 6.1



# Where to start ?

## # Where to start ?

- Locate the main class
  - UIApplicationDelegate
    - ApplicationDidFinishLaunching
    - ApplicationDidFinishLaunchingWithOptions
  - Locate views initialisation
    - UI\*ViewController
      - » ViewDidLoad

## # Where to look ?

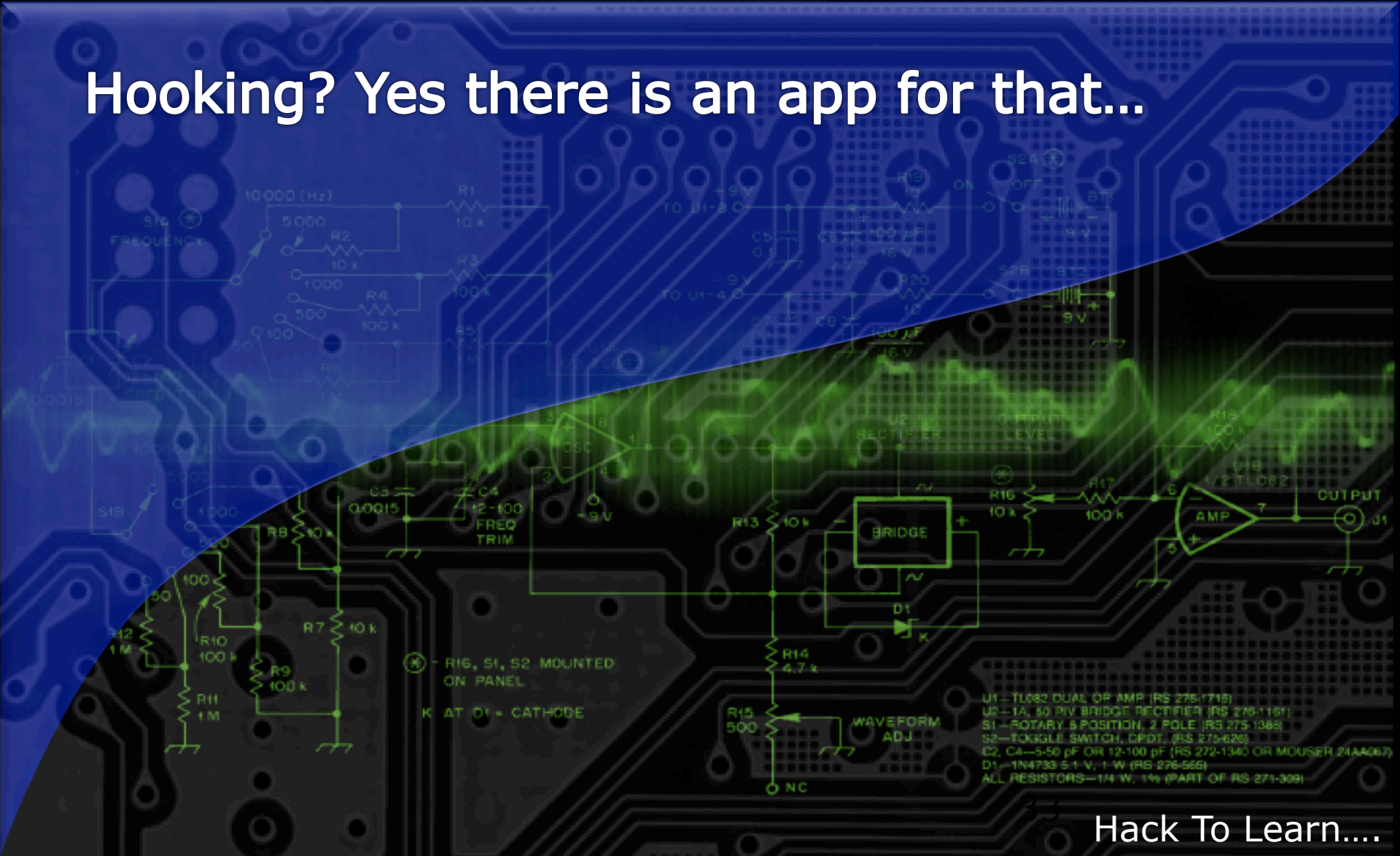
- URL > NSURL\*
- Socket > CFSocket\*
- Keychain > ksecAttr\*, SecKeychain\*
- Files Handling > NSFileManager\*
- Crypto > CCCrypt\*







## Hooking? Yes there is an app for that...



\* RIG, S1, S2 MOUNTED ON PANEL  
K AT D1 = CATHODE

U1—TL082 DUAL OP AMP (RS 275-1715)  
U2—1A, 50 PIV BRIDGE RECTIFIER (RS 276-116\*)  
S1—ROTARY 5 POSITION, 2 POLE (RS 275-1388)  
S2—TOGGLE SWITCH, CPDT. (RS 275-625)  
C2, C4—5-50 pF OR 12-100 pF (RS 272-1340 OR MOUSER 214A067)  
D1—1N4733 5.1 V, 1 W (RS 276-569)  
ALL RESISTORS—1/4 W, 1% (PART OF RS 271-309)

# Hooking made easy: MobileSubstrate

## # MobileSubstrate

- Allows developers to provide run-time patches
  - MobileLoader will first load itself into the run application using `DYLD_INSERT_LIBRARIES`
  - Looks for all dynamic libraries in the directory `/Library/MobileSubstrate/DynamicLibraries/` and load them.
- MobileHooker is used to replace system functions
  - `MSHookMessageEx()`
    - Replace the implementation of the Objective-C message [class selector] by replacement, and return the original implementation..
  - `MSHookFunction()`
    - like `MSHookMessageEx()` but is for C/C++ functions.



# DEMO: Stealing Crypto keys

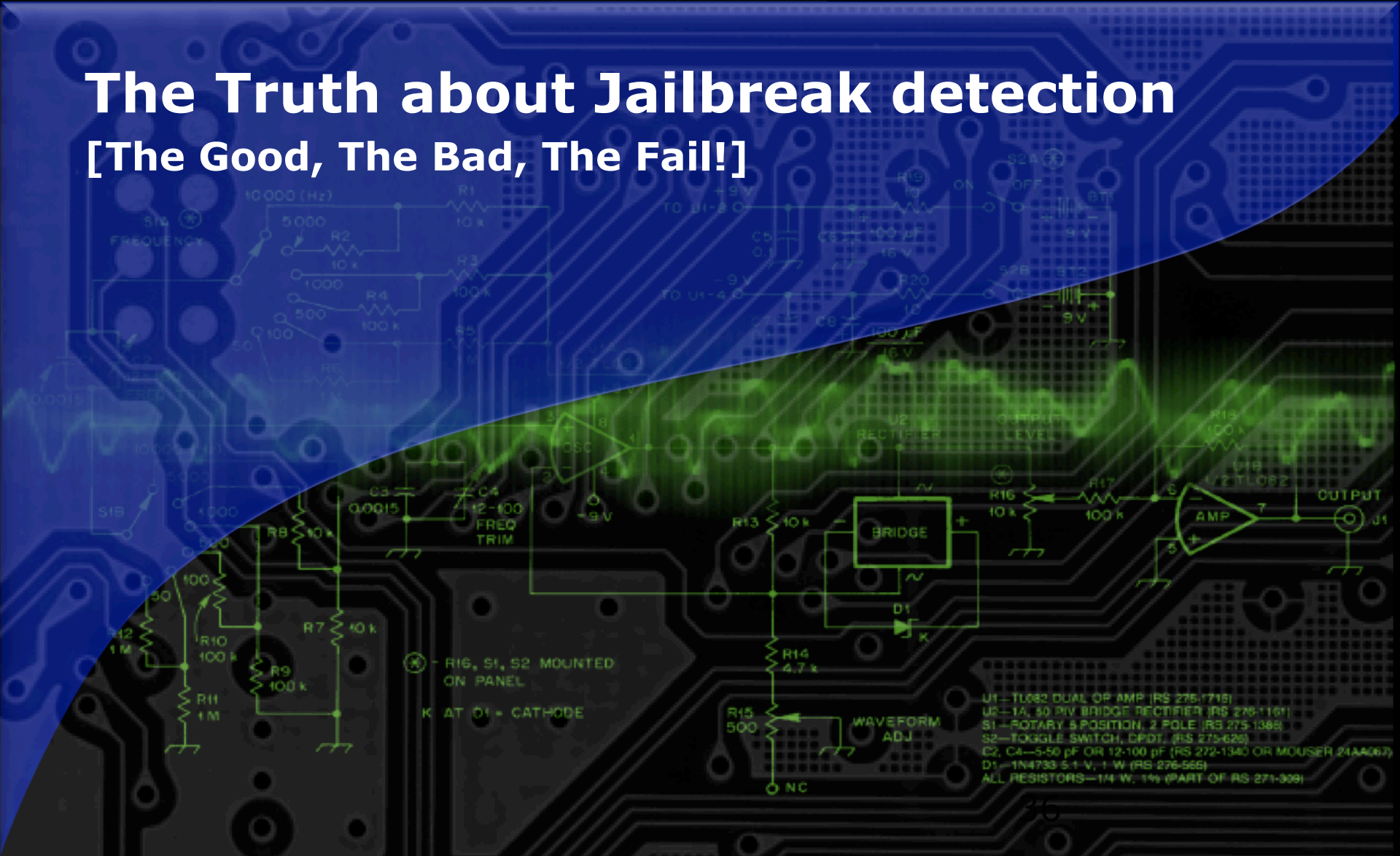
## # Hooking the CCEncrypt(3cc) API

```
CCEncrypt(CCEncryption op, CCEncryptionAlgorithm alg, CCEncryptionOptions options, const void *key, size_t keyLength, const void *iv, const void *dataIn, size_t dataInLength, void *dataOut, size_t dataOutAvailable, size_t *dataOutMoved);
```





## The Truth about Jailbreak detection [The Good, The Bad, The Fail!]



# Jailbreak detection classic checking for shell [The good]

## # Checking for shell

```
+ (BOOL)doShell {  
    if (system(0)) {  
        return YES;  
    }  
    return NO;  
}
```

## # Bypassing the check

```
static int (*old_system)(char *) = NULL;  
int st_system(char * cmd){  
    if (!cmd){  
        return nil;  
    }  
    return old_system(cmd);  
}  
  
__attribute__((constructor)) static void initialize() {  
    NSLog(@"StealthJBInitialize!");  
    MSHookFunction(system, st_system, &old_system);  
}
```



# Jailbreak detection Classics

## Jailbreak files detection [The bad]

### # Checking for jailbreak files (Cydia, SSH, MobileSubstrate, Apt, ...)

```
+ (BOOL)doCydia {
    if ([[NSFileManager defaultManager]
        fileExistsAtPath: @"/Applications/Cydia.app"]){
        return YES;
    }
    return NO;
}
```

### # Bypassing the check (hooking NFSFileManager)

```
void* (*old_fileExistsAtPath)(void* self, SEL _cmd, NSString* path) = NULL;
void* st_fileExistsAtPath(void* self, SEL _cmd, NSString* path){
    if ([path isEqualToString:@"/Applications/Cydia.app"]){
        NSLog(@"=>hiding %@", path);
        return 0;
    }
    return old_fileExistsAtPath(self,_cmd,path);
}

__attribute__((constructor)) static void initialize() {
    NSLog(@"StealthJBInitialize!");
    MSHookMessageEx([NSFileManager class], @selector(fileExistsAtPath:),
                    (IMP)st_fileExistsAtPath, (IMP *)&old_fileExistsAtPath);
}
```



# DEMO: Bypassing jailbreak detection



# Jailbreak detection classics [The fail!]

- # Sandbox check using fork
- # Documented in some books and blog posts
  - If the process can fork, the device is jailbroken.

```
+(BOOL) doFork () {
    int res = fork();
    if (!res) {
        exit(0);
    }

    if (res >= 0) {
        #if TARGET_IPHONE_SIMULATOR
            NSLog("fork_check -> Running on the simulator!");
            return 0;
        #else
            return 1;
        #endif
    }
    return 0;
}
```





# Jailbreak detection classics [The fail!]

# From the iphonewiki:

## Sandbox Patch

- fixes the sandbox problems caused by moving files
- access outside `/private/var/mobile` is allowed
- access to `/private/var/mobile/Library/Preferences/com.apple` is going through original evaluation
- access to other subdirs of `private/var/mobile/Library/Preferences` is granted
- everything else goes through original checks

# Jailbreak detection classics [The fail!]

- # Sandbox check using fork
- # Not working!
  - The sandbox patch doesn't affect this part of the sandbox!

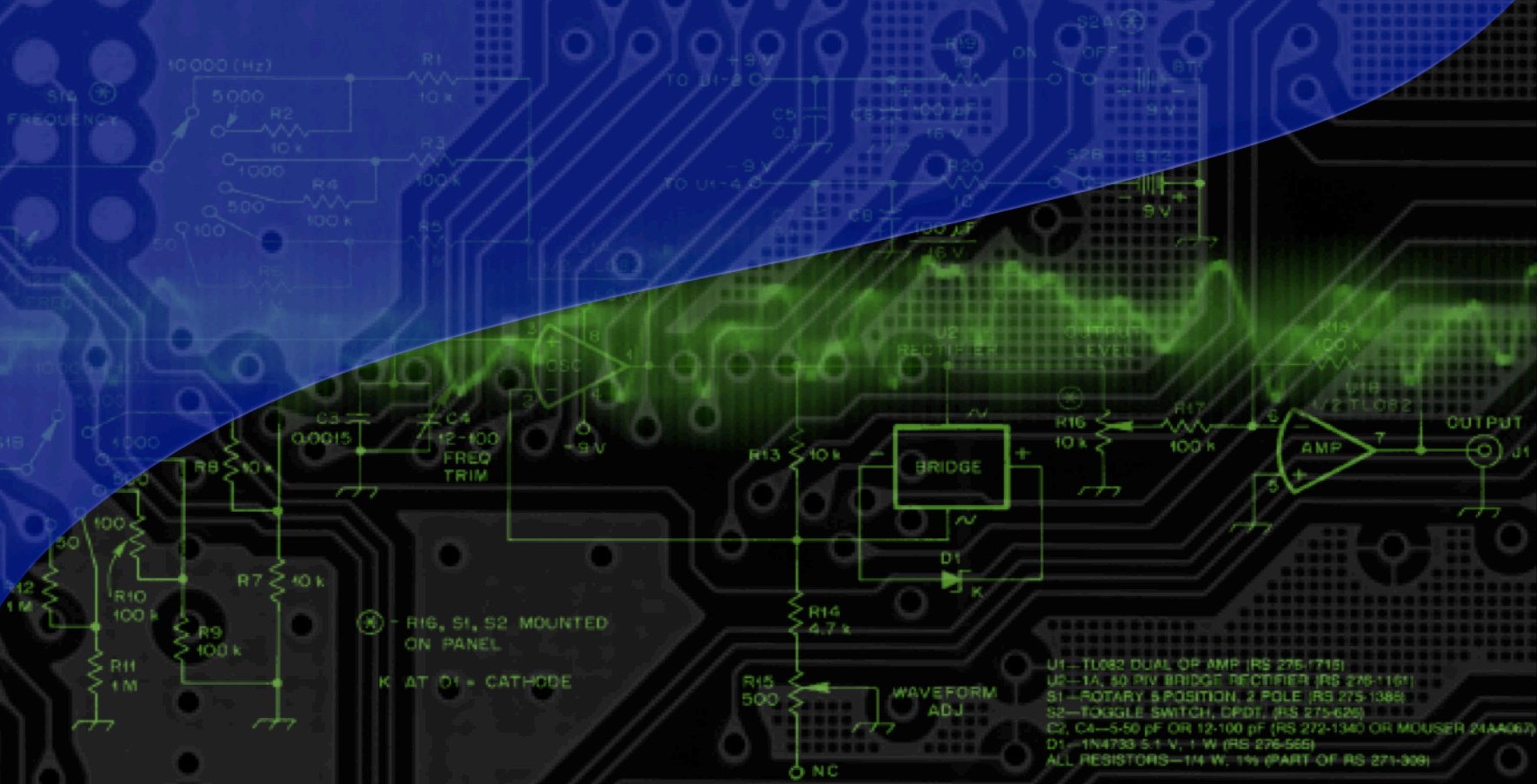
```
+(BOOL) doFork () {
    int res = fork();
    if (!res) {
        exit(0);
    }

    if (res >= 0) {
        #if TARGET_IPHONE_SIMULATOR
        NSLog("fork failed. Running in the simulator.");
        return 0;
        #else
        return 0;
        #endif
    }
    return 0;
}
```

**FAIL**



## Security Worst Practices





# Hardcoded crypto key...

```
v31 = objc_msgSend(&OBJC_CLASS__NSString, "stringWithFormat:", CFSTR("HiddenTreasures"));
v32 = objc_msgSend(&OBJC_CLASS__NSData, "dataFromBase64String:", v31);
v33 = objc_msgSend(&OBJC_CLASS__NSBundle, "mainBundle");
v34 = objc_msgSend(v33, "pathForResource ofType:", v13, CFSTR("cpng"));
v35 = objc_msgSend(&OBJC_CLASS__NSData, "dataWithContentsOfFile:", v34);
v36 = objc_msgSend(&OBJC_CLASS__FBEncryptorAES, "decryptData:", v35, v32);
v37 = objc_msgSend(&OBJC_CLASS__UIImage, "imageWithData:", v36);
v38 = objc_msgSend(&OBJC_CLASS__UIImageView, "alloc");
v39 = objc_msgSend(v38, "initWithImage:", v37);
__asm { VM0V.F32 S14, #9.0 }
v207 = 1126957056;
v206 = 1134297088;
__asm
{
    UADD.F32 D7, D8, v207;
    USTR S14, v206;
}
v205 = 1114898432;
v40 = v39;
objc_msgSend(v39, "setFrameRect:", v40);
objc_msgSend(v5, "addSubview:", v40);
objc_msgSend(v40, "release");
```



```
1 from Crypto.Cipher import AES
2 from binascii import b4encode, b4decode, hexlify
3 import sys
4
5 SECRET_KEY = b64decode('HiddenTreasures') + Padding
6
7 print len(SECRET_KEY), len(SECRET_KEY)
8 secret=SECRET_KEY + ("\x00" * (16-len(SECRET_KEY)))
9 IV = "\x00" * 16 # Default IV
10 cipher_for_decrypt = AES.new(secret, AES.MODE_CBC, IV)
11
12 def DecryptWithAES(cipher_for_decrypt, encrypted_data):
13     decoded_data = encrypted_data
14     decrypted_data = cipher_for_decrypt.decrypt(encrypted_data)
15     return decrypted_data
16
17
18 open(sys.argv[1], "rb").read()
19 print "Input File >", sys.argv[1], "Size >", len(i)
20 print "AES Key >", binascii.hexlify(secret)
21 decrypted_data = DecryptWithAES(cipher_for_decrypt, i + ("\x00" * (16-len(i)%16)))
22 outfile = sys.argv[1] + '.dec.png'
23 o = open(outfile, 'wb+')
24 o.write(decrypted_data)
25 print "decrypted File >", outfile, "Size >", len(i)
```



# Secure browser... Really ?

Search: cookie [X] Refresh

```
domain:\.twitter.com,guest_id=v1%3a133951399233296825\ path:\^
isSecure:FALSE>"
)
for URL: https://mobile.twitter.com/

mar. [REDACTED] Cookies: (
  "<NSHT... version:0 name:"
  value:"86.28... 51399233296825" expiresDate: [REDACTED]
  15:13:12 +0000 id: [REDACTED] +0000 (3.61207e+08)
  sessionOnly:FALSE
  domain:\.twitter.com,guest_id=v1%3a133951399233296825\ path:\^
  isSecure:FALSE>"
)
for URL: https://mobile.twitter.com/signup
```



# DEMO: Authentication Bypass





## Defensives Measures





# Defensives Measures

## # Antidebug technics

- Old School GDB Killer : `PTRACE_DENY_ATTACH`
- Checking the `P_TRACED` flag

## # Anti Hooking technics

- Validating address space : Using `dladdr()` & `DI_info` structure
- Inlining

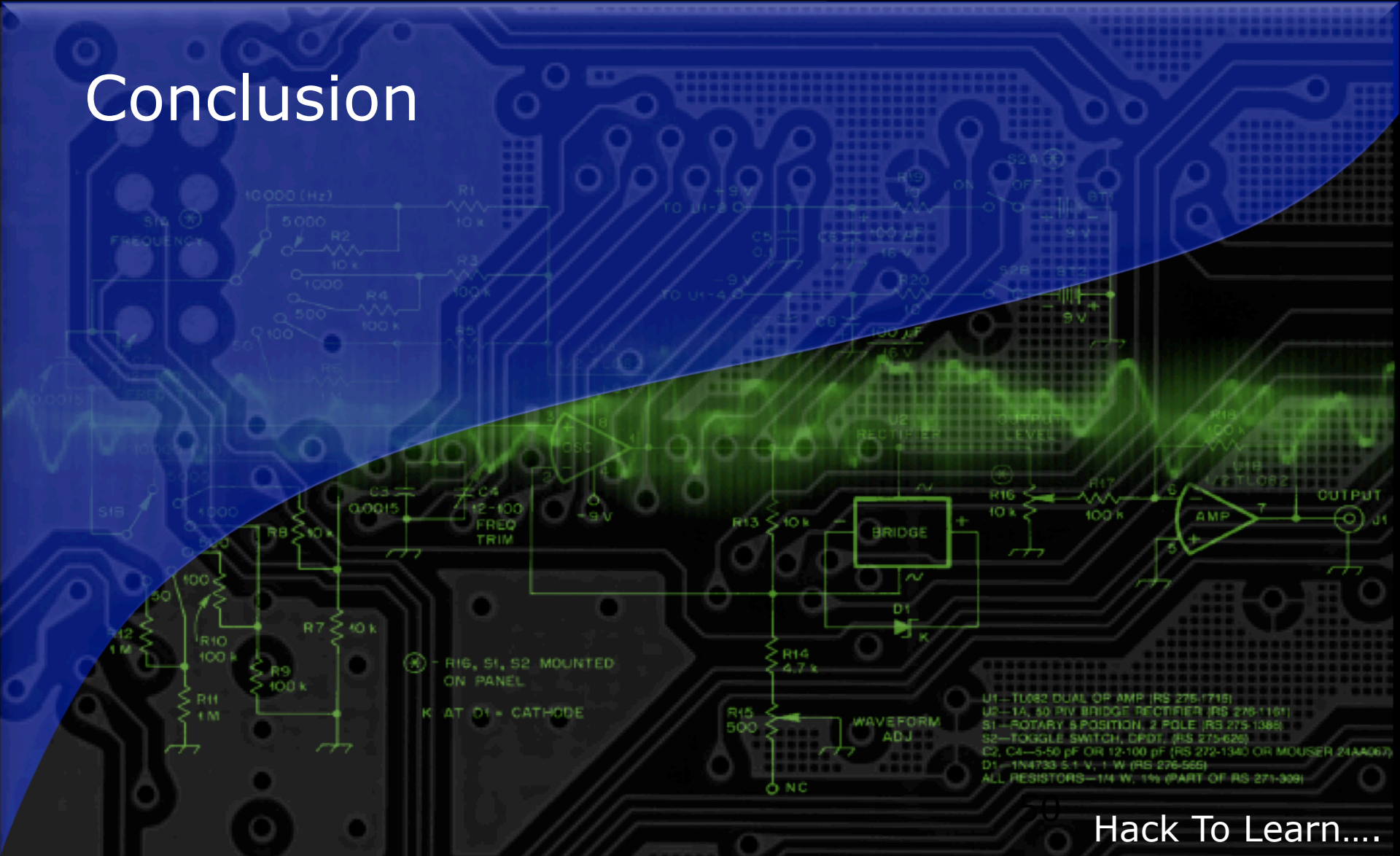
## # Obfuscation

- No public tools for Objective C code obfuscation.
- Objective C is a dynamic language,
  - Based on message passing paradigm,
  - Most of bindings are resolved run time
  - It is always possible for attacker to track, intercept and reroute calls, even with obfuscated names.
- Manually implementing obfuscation can slow down attackers analysis
  - Renaming classes and methods
  - Dynamic string generation





## Conclusion



# Conclusion

- # Regarding security most of iOS applications are not mature!
- # Developers should follow the following recommendation in order to mitigate the risks.
  - Do not rely only on iOS security
  - Do not store credential using standardUserDefaults method.
  - Encrypt your data even when stored in the keychain
  - Do not store crypto keys on the device
  - Check your code, classes, functions, methods integrity
  - Detect the jailbreak
  - Properly implement cryptography in applications
    - simple implementation are the most secure
  - Remove all debug information from the final release
  - **Minimize use of Objective-C for critical functions & security features.**





## Thank you for Listening Questions ?

mathieu.ranard[-at-]sogeti.com - <http://esec-pentest.sogeti.com>  
mathieu.renard[-at-]gtohack.org - <http://www.gtohack.org>

