The Dark Art of iOS Application Hacking

(and a few ideas to write better apps)

VIAFORENSICS

innovative digital forensics and security

Jonathan Zdziarski
Sr. Forensic Scientist
@JZdziarski

I am…

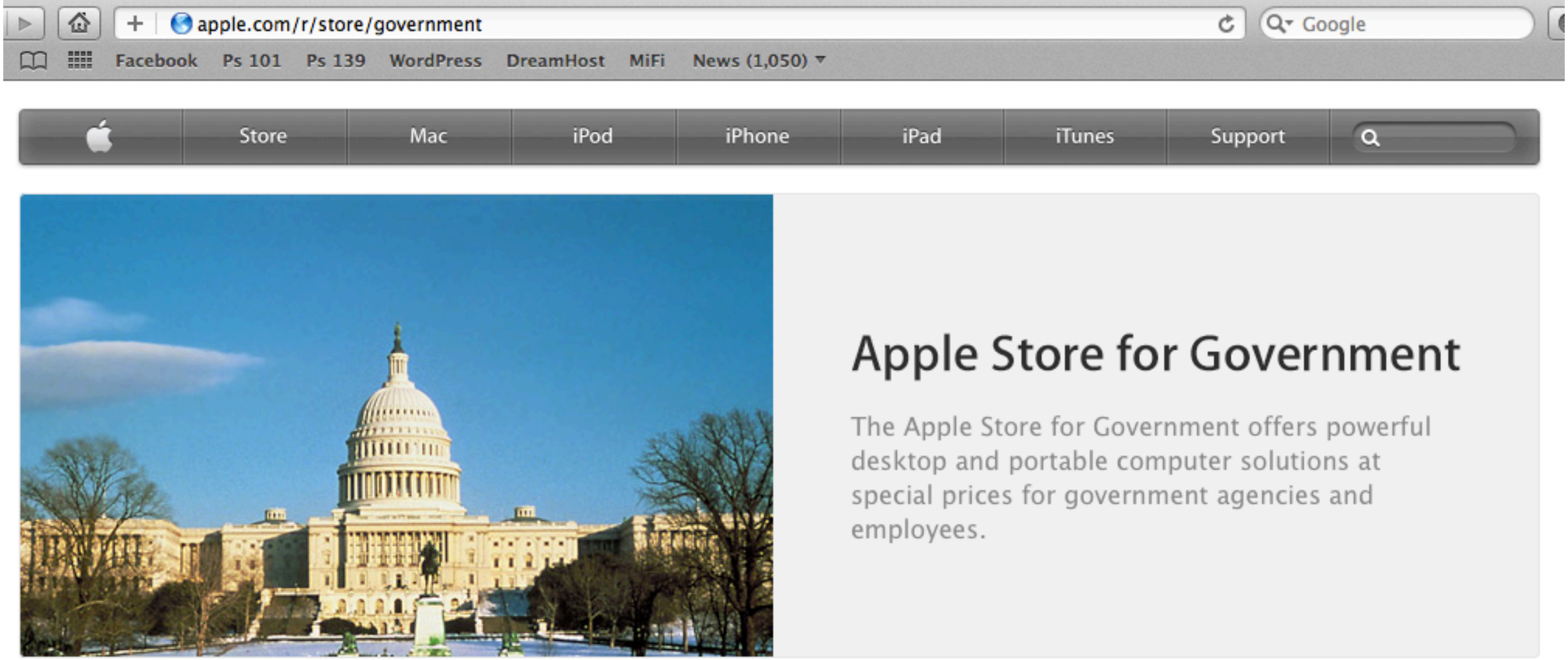
- Sr. Forensic Scientist for viaForensics
  - Among other things, I get paid to hack banks and government systems for a living
- Author for O'Reilly Media
  - iPhone Open Application Development, iPhone SDK Application Development, iPhone Forensics, Hacking and Securing iOS Applications
  - And a neat little book for NoStarch titled, "Ending Spam"
- Freelance as a consultant, training governments in iOS forensic imaging and investigations and assisting in high profile cases (I help put A-holes in jail)

VIAFORENSICS

Monoculture

- Didn’t this used to be a bad thing?
- Pro’s:
  - Reduced attack surface
  - Rapid prototyping
  - Fewer holes to blame on the developer
- Con’s:
  - Homogenous attack surface: Hack one, hack them all
  - Security now an afterthought
  - More holes to blame on Apple

Somehow, this happened...

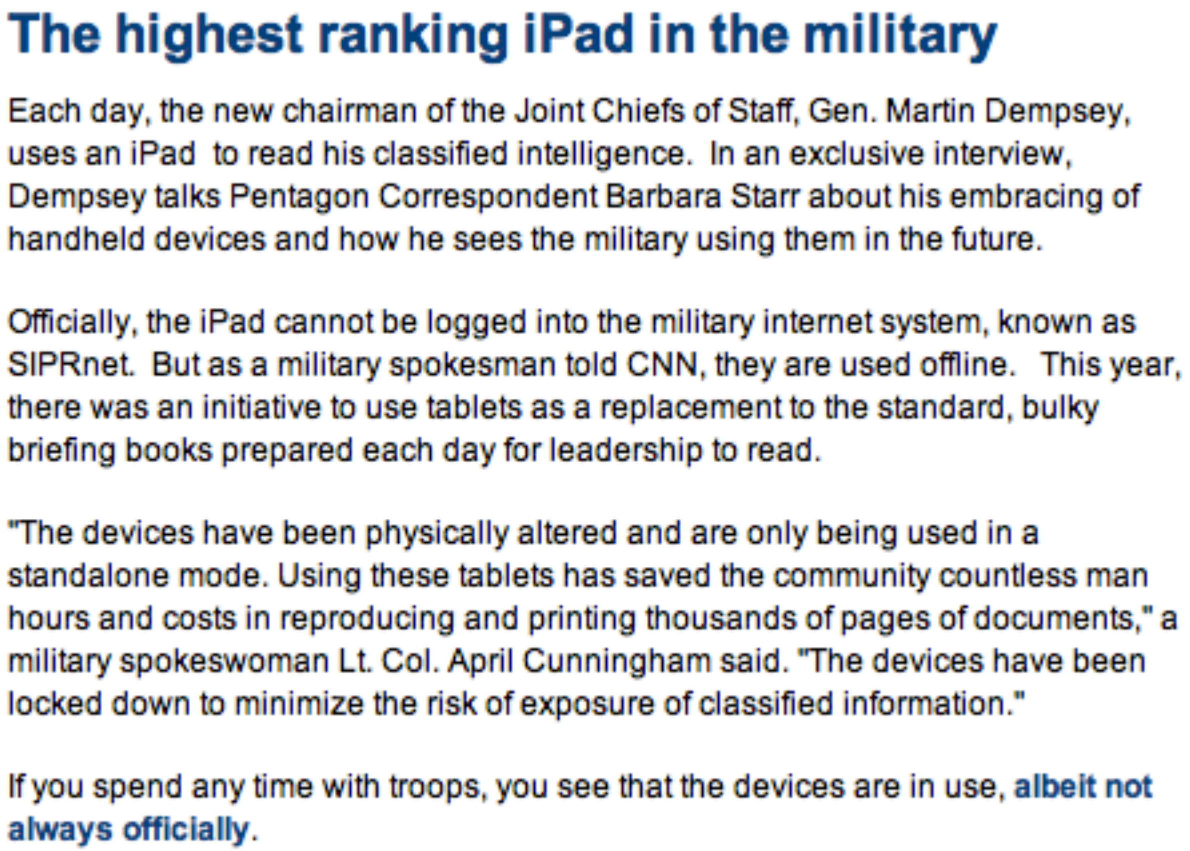


Apple Store for Government

The Apple Store for Government offers powerful desktop and portable computer solutions at special prices for government agencies and employees.

VIAFORENSICS

Then suddenly this happened…



The highest ranking iPad in the military

Each day, the new chairman of the Joint Chiefs of Staff, Gen. Martin Dempsey, uses an iPad to read his classified intelligence. In an exclusive interview, Dempsey talks Pentagon Correspondent Barbara Starr about his embracing of handheld devices and how he sees the military using them in the future.

Officially, the iPad cannot be logged into the military internet system, known as SIPRnet. But as a military spokesman told CNN, they are used offline.   This year, there was an initiative to use tablets as a replacement to the standard, bulky briefing books prepared each day for leadership to read.

"The devices have been physically altered and are only being used in a standalone mode. Using these tablets has saved the community countless man hours and costs in reproducing and printing thousands of pages of documents," a military spokeswoman Lt. Col. April Cunningham said. "The devices have been locked down to minimize the risk of exposure of classified information."

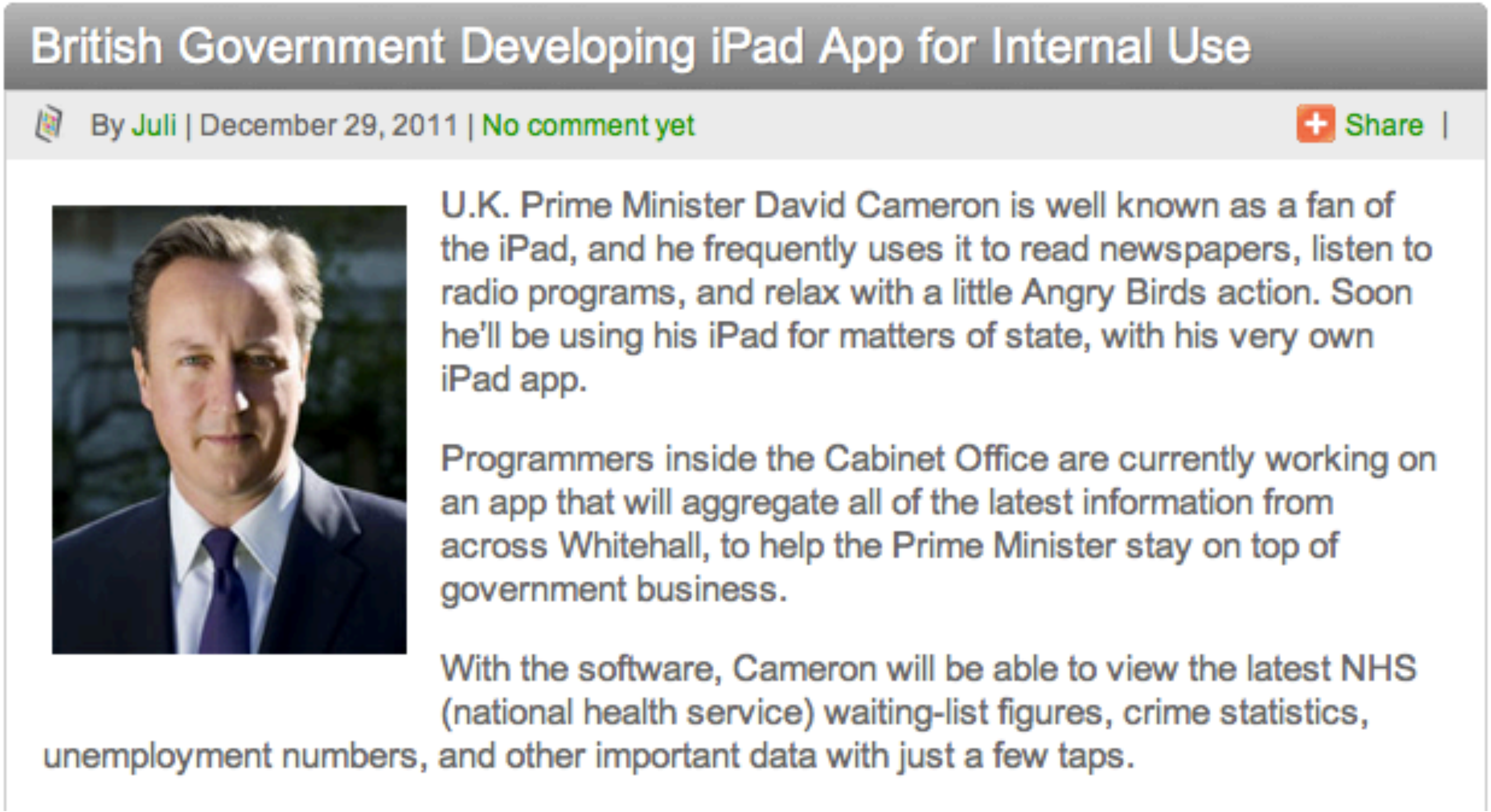
If you spend any time with troops, you see that the devices are in use, **albeit not always officially**.

VIAFORENSICS

Source: CNN Security Blogs

Then it became cool…

Then it went viral…



British Government Developing iPad App for Internal Use

By Juli | December 29, 2011 | No comment yet

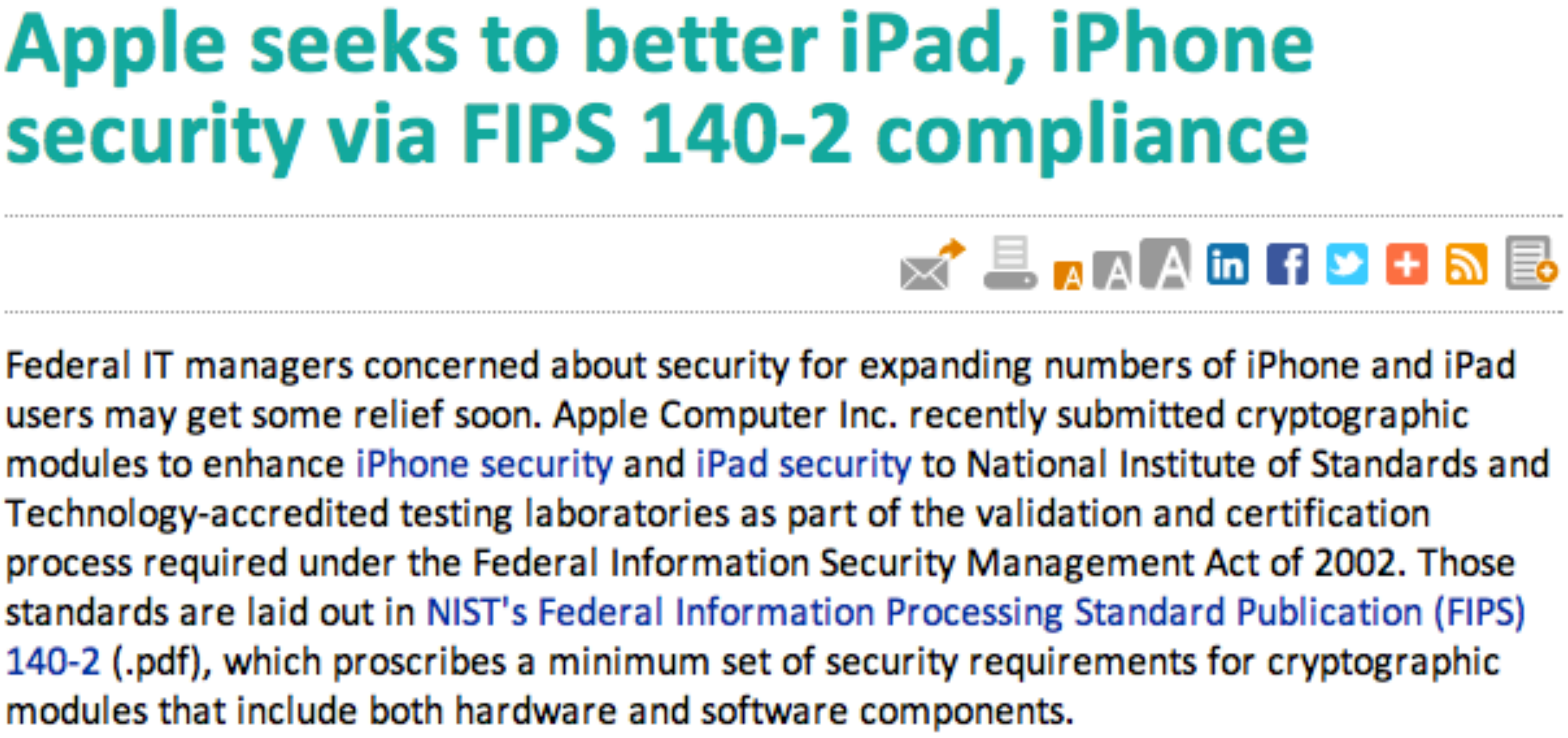
Share |

U.K. Prime Minister David Cameron is well known as a fan of the iPad, and he frequently uses it to read newspapers, listen to radio programs, and relax with a little Angry Birds action. Soon he'll be using his iPad for matters of state, with his very own iPad app.

Programmers inside the Cabinet Office are currently working on an app that will aggregate all of the latest information from across Whitehall, to help the Prime Minister stay on top of government business.

With the software, Cameron will be able to view the latest NHS (national health service) waiting-list figures, crime statistics, unemployment numbers, and other important data with just a few taps.

All probably after this happened…



Apple seeks to better iPad, iPhone security via FIPS 140-2 compliance

Federal IT managers concerned about security for expanding numbers of iPhone and iPad users may get some relief soon. Apple Computer Inc. recently submitted cryptographic modules to enhance iPhone security and iPad security to National Institute of Standards and Technology-accredited testing laboratories as part of the validation and certification process required under the Federal Information Security Management Act of 2002. Those standards are laid out in NIST's Federal Information Processing Standard Publication (FIPS) 140-2 (.pdf), which proscribes a minimum set of security requirements for cryptographic modules that include both hardware and software components.

VIAFORENSICS

FIPS != Security

- FIPS is about compliance
- Does Apple’s cryptographic module comply with FIPS encryption requirements?
  - Probably
- Does this mean very little in terms of actual security?
  - Probably

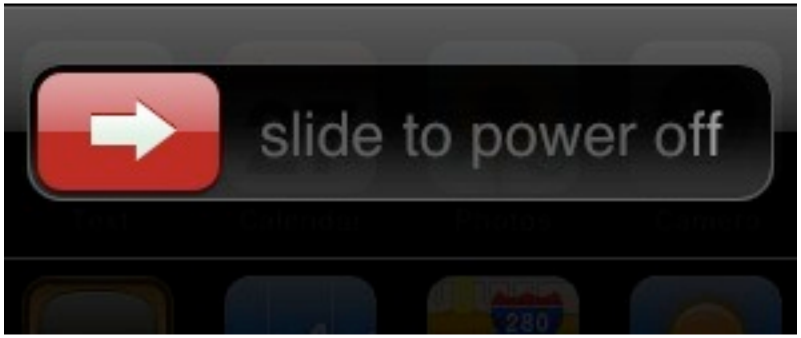
What FIPS is NOT

- FIPS is NOT a red team
- FIPS is NOT a guarantee of security
- FIPS is NOT a security solution
- FIPS is NOT a fuzzy kitten either…
  - Promotes compliance
  - A daunting process

  … but often mistaken for security

VIAFORENSICS

Compromising “Remote Wipe”



VIAFORENSICS

Compromising Encryption

- Brute force (Not for feint of heart)
  - See also: Utah data center
- Attack the (usually poor) implementation
  - Sogeti's free tools (decrypt keychain + raw disk)
    - http://code.google.com/p/iphone-dataprotection/
- … Or …

VIAFORENSICS

Attack the Application

Attacking the Application

- Intimate knowledge of the application’s design
- Human analysis of possible attack points (via debugger, hex editor, etc)
- Customized injection to specifically target the application

Intimate Knowledge of Application Design: The Old School Way

- Meet class-dump / class-dump-z
  - http://code.google.com/p/networkpx/wiki/class_dump_z
  - Effectively draws a map of application classes and instance variables
  - Works great after you decrypt Apple's DRM (discussed later)
  - Provides a list of all application classes and instance variables

Intimate Knowledge of Application Design: Decrypt AppStore Binaries

```
#define LC_ENCRYPTION_INFO 0x21
struct encryption_info_command {
  uint32_t cmd;
  uint32_t cmdsize;
  uint32_t cryptoff;   // file offset of first encrypted byte
  uint32_t cryptsize;  // file size of encrypted data
  uint32_t cryptid;    // method of encryption
};
```

Intimate Knowledge of Application Design: Decrypt AppStore Binaries

```
$ file PayPal
PayPal: Mach-O universal binary with 2 architectures
PayPal (for architecture armv6):      Mach-O executable arm
PayPal (for architecture armv7):      Mach-O executable arm

$ otool -l PayPal  | grep -i crypt
          cmd LC_ENCRYPTION_INFO
    cryptoff  8192
    cryptsize 2174976
    cryptid   1
          cmd LC_ENCRYPTION_INFO
    cryptoff  8192              <- File offset for executable code
    cryptsize 1720320                  (relative to armv7 arch)
    cryptid   1
```

VIAFORENSICS

Intimate Knowledge of Application Design: Decrypt AppStore Binaries

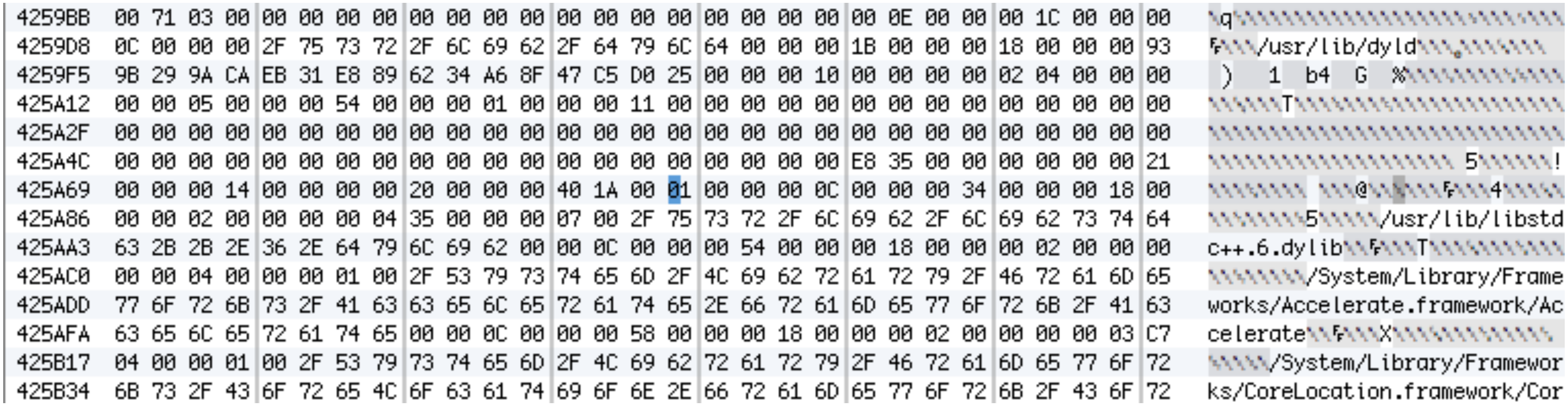
```
$ otool -f PayPal -arch armv7
Fat headers
fat_magic 0xcafebabe
nfat_arch 2
architecture 0
    cputype 12
    cpusubtype 6
    capabilities 0x0
    offset 4096
    size 4341632
    align 2^12 (4096)
architecture 1
    cputype 12
    cpusubtype 9
    capabilities 0x0
    offset 4345856      <- File Offset for armv7
    size 3884000
    align 2^12 (4096)
```

Decrypt AppStore Binaries: How xSellize and other piracy tools do it

```
# gdb –q ./PayPal
Reading symbols for shared libraries . Done

(gdb) rb doModInitFunctions

Breakpoint 1 at 0x2fe0cece
<function, no debug info>
__dyld__ZN16ImageLoaderMachO18doModInitFunctionsERKN11ImageLoader11LinkCont
extE;

(gdb) r

Starting program: /private/var/mobile/Applications/B68610BB-
C5C4-4F02-847D-615297BF6D1C/PayPal.app/PayPal

Breakpoint 1, 0x2fe0cece in
__dyld__ZN16ImageLoaderMachO18doModInitFunctionsERKN11ImageLoader11LinkCont
extE ()
(gdb) dump memory armv7.bin 0x3000 1732608
(gdb) q
```

Intimate Knowledge of Application Design: Decrypt AppStore Binaries

```
$ dd seek=4354048 bs=1 conv=notrunc if=armv7.bin of=PayPal
(1732608 + 8192 = 4354048)
```



```
4259BB  00 71 03 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0E 00 00 00 1C 00 00 00   q
4259D8  0C 00 00 00 2F 75 73 72 2F 6C 69 62 2F 64 79 6C 64 00 00 00 1B 00 00 00 18 00 00 00 93       /usr/lib/dyld
4259F5  9B 29 9A CA EB 31 E8 89 62 34 A6 8F 47 C5 D0 25 00 00 00 10 00 00 00 00 02 04 00 00 00    )   1  b4  G  %
425A12  00 00 05 00 00 00 54 00 00 00 01 00 00 00 11 00 00 00 00 00 00 00 00 00 00 00 00 00 00         T
425A2F  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
425A4C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 E8 35 00 00 00 00 00 00 21                       5       !
425A69  00 00 00 14 00 00 00 00 20 00 00 00 40 1A 00 01 00 00 00 0C 00 00 00 34 00 00 00 18 00               @          4
425A86  00 00 02 00 00 00 00 04 35 00 00 00 07 00 2F 75 73 72 2F 6C 69 62 2F 6C 69 62 73 74 64           5     /usr/lib/libstd
425AA3  63 2B 2B 2E 36 2E 64 79 6C 69 62 00 00 0C 00 00 00 54 00 00 00 18 00 00 00 02 00 00 00  c++.6.dylib      T
425AC0  00 00 04 00 00 00 01 00 2F 53 79 73 74 65 6D 2F 4C 69 62 72 61 72 79 2F 46 72 61 6D 65          /System/Library/Frame
425ADD  77 6F 72 6B 73 2F 41 63 63 65 6C 65 72 61 74 65 2E 66 72 61 6D 65 77 6F 72 6B 2F 41 63  works/Accelerate.framework/Ac
425AFA  63 65 6C 65 72 61 74 65 00 00 0C 00 00 00 58 00 00 00 18 00 00 00 02 00 00 00 00 03 C7  celerate      X
425B17  04 00 00 01 00 2F 53 79 73 74 65 6D 2F 4C 69 62 72 61 72 79 2F 46 72 61 6D 65 77 6F 72       /System/Library/Framewor
425B34  6B 73 2F 43 6F 72 65 4C 6F 63 61 74 69 6F 6E 2E 66 72 61 6D 65 77 6F 72 6B 2F 43 6F 72  ks/CoreLocation.framework/Cor
```

Intimate Knowledge of Application Design: Decrypt AppStore Binaries

```
$ otool -l PayPal | grep -i crypt
          cmd LC_ENCRYPTION_INFO
    cryptoff  8192
    cryptsize 2174976
    cryptid   1
          cmd LC_ENCRYPTION_INFO
    cryptoff  8192
    cryptsize 1720320
    cryptid   0

$ class-dump-z PayPal         <- Fun and profit
```

id objc_msgSend(id self, SEL op, ...)

```
#import <Foundation/Foundation.h>

@interface SaySomething : NSObject
- (void) say: (NSString *) phrase;
@end

@implementation SaySomething

- (void) say: (NSString *) phrase {
    printf("%s\n", [ phrase UTF8String ]);
}
@end

int main(void) {
    objc_msgSend( objc_msgSend(
        objc_msgSend(
            objc_getClass("SaySomething"),
                NSSelectorFromString(@"alloc")),
            NSSelectorFromString(@"init")),
        NSSelectorFromString(@"say:"),
    @"Hello, world!" );

    return 0;
}
```

id objc_msgSend(id self, SEL op, ...)

```
0x00002f10 <main+48>: bl 0x2f94 <dyld_stub_objc_msgSend>
0x00002f14 <main+52>: str r0, [sp, #16]
0x00002f18 <main+56>: ldr r1, [pc, #100] ;0x2f84 <main+164>
0x00002f1c <main+60>: ldr r1, [pc, r1]
0x00002f20 <main+64>: bl 0x2f94 <dyld_stub_objc_msgSend>
0x00002f24 <main+68>: str r0, [sp, #8]
0x00002f28 <main+72>: str r0, [r7, #-16]
0x00002f2c <main+76>: ldr r1, [pc, #84] ; 0x2f88 <main+168>
0x00002f30 <main+80>: ldr r1, [pc, r1]
0x00002f34 <main+84>: ldr r2, [pc, #80] ; 0x2f8c <main+172>
0x00002f38 <main+88>: add r2, pc, r2
0x00002f3c <main+92>: bl 0x2f94 <dyld_stub_objc_msgSend>
0x00002f40 <main+96>: ldr r0, [sp, #8]
...
```

VIAFORENSICS

What’s going on underneath…

```
(gdb) break main
Breakpoint 1 at 0x2eec
(gdb) run
Starting program: /private/var/root/HelloWorld
Reading symbols for shared libraries .............................. done
Breakpoint 1, 0x00002eec in main () (gdb)

(gdb) break objc_msgSend
Breakpoint 2 at 0x34008c96
(gdb) continue
Continuing.
Breakpoint 2, 0x34008c96 in objc_msgSend () (gdb)

(gdb) x/a $r0
0x30cc <OBJC_CLASS_$_SaySomething>: 0x30b8 <OBJC_METACLASS_$_SaySomething>
(gdb) x/s $r1
0x35e89f8c: "alloc"
(gdb)
```

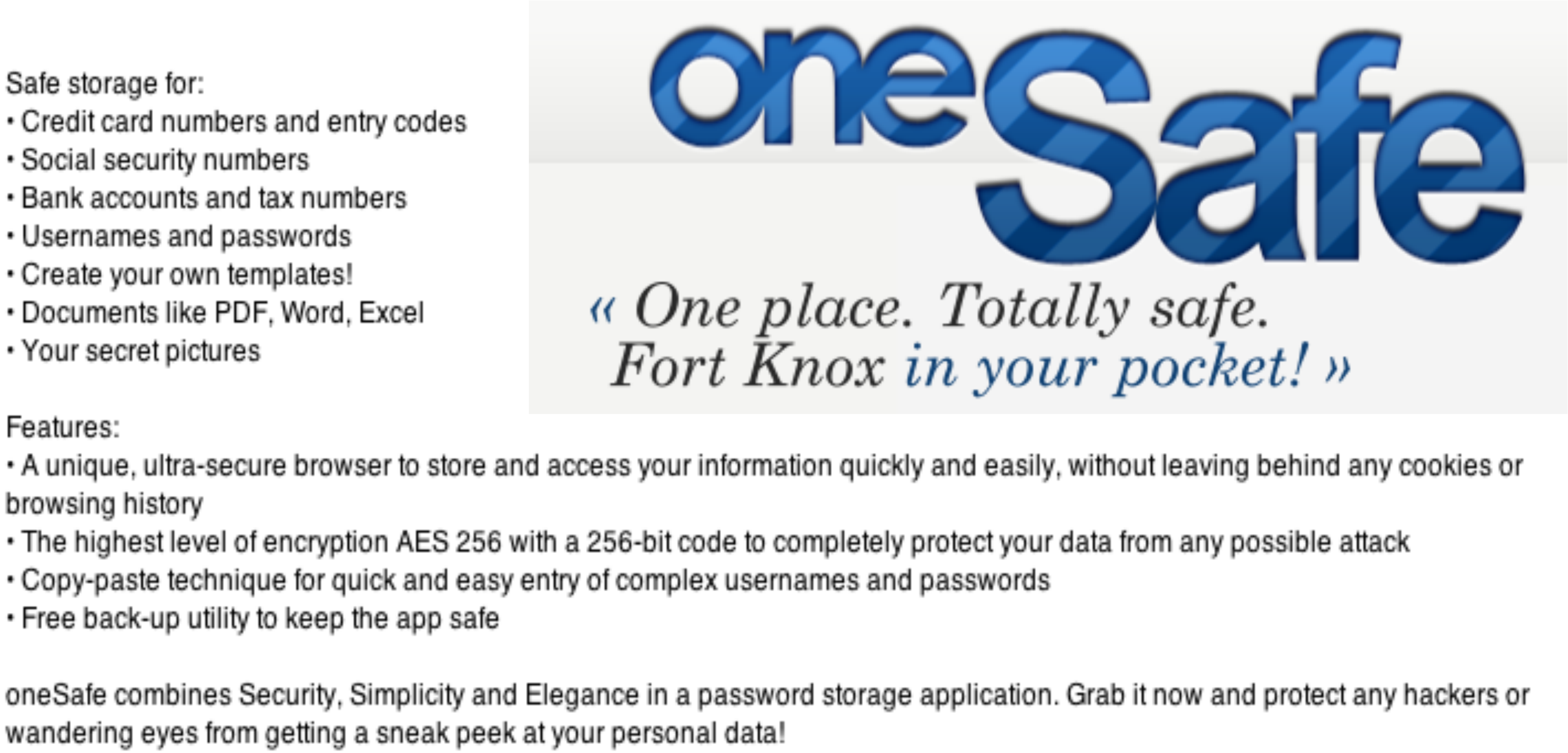
What’s going on underneath…

```
(gdb) break objc_msgSend
(gdb) commands
Type commands for when breakpoint 1 is hit, one per line. End with a line saying just "end".
>printf "-[%s %s]\n", (char *)class_getName(*(long *)$r0, $r1), $r1
>c
>end
(gdb) c
Continuing.

[Switching to process 1629 thread 0x1503]
Breakpoint 1, 0x34008c96 in objc_msgSend () -[UIDevice currentDevice]
Breakpoint 1, 0x34008c96 in objc_msgSend () -[UIDevice isWildcat]
Breakpoint 1, 0x34008c96 in objc_msgSend () -[UIKeyboardLayoutStar hitBuffer]
Breakpoint 1, 0x34008c96 in objc_msgSend () -[UIKeyboardImpl sharedInstance]
Breakpoint 1, 0x34008c96 in objc_msgSend () -[UIKeyboardImpl orientation]
```

Manipulating the Framework

```
(gdb) call (void *) objc_msgSend( \
(void *) objc_getClass(“UIApplication”), \
(void *) sel_registerName(“sharedApplication”) \ )

$3 = (void *) 0x29acf0 <- [ UIApplication sharedApplication ]
```

Hacking oneSafe: oneWay to Profit



Safe storage for:

- Credit card numbers and entry codes
- Social security numbers
- Bank accounts and tax numbers
- Usernames and passwords
- Create your own templates!
- Documents like PDF, Word, Excel
- Your secret pictures

Features:

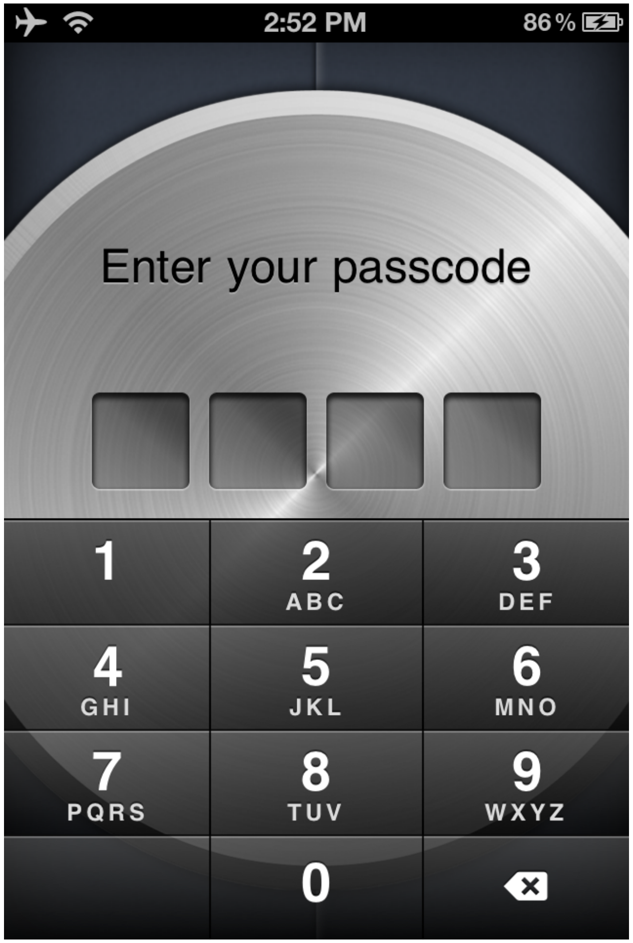
- A unique, ultra-secure browser to store and access your information quickly and easily, without leaving behind any cookies or browsing history
- The highest level of encryption AES 256 with a 256-bit code to completely protect your data from any possible attack
- Copy-paste technique for quick and easy entry of complex usernames and passwords
- Free back-up utility to keep the app safe

oneSafe combines Security, Simplicity and Elegance in a password storage application. Grab it now and protect any hackers or wandering eyes from getting a sneak peek at your personal data!

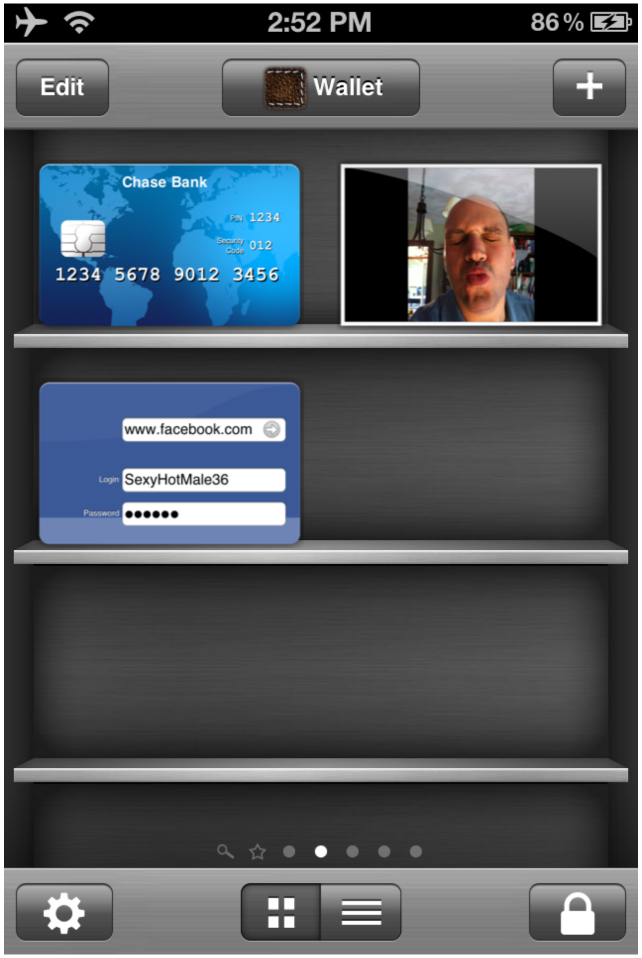
Not-so oneSafe

```
# gdb -p 353
GNU gdb 6.3.50-20050815 (Apple version gdb-1704) (Fri Jul  1 07:18:51 UTC
2011)
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you
are
welcome to change it and/or distribute copies of it under certain
conditions.
Attaching to process 353.
Reading symbols for shared libraries . done
Reading symbols for shared
libraries .................................................................
...........................................................................
.................................. done

0x323d6010 in mach_msg_trap ()
(gdb) call (void *) [ [ [ UIApplication sharedApplication ] \
delegate ] userIsLogged: 1 ]
$1 = (void *) 0x25eb20
(gdb) c
Continuing.
```

From



To



Chase Bank
PIN 1234
Security Code 012
1234 5678 9012 3456

www.facebook.com
Login SexyHotMale36
Password ••••••

VIAFORENSICS

Why bother attacking one application's code…

When an attacker can [write a virus to] attack every application’s code at once?

Recap: Attacking an Application

- Intimate knowledge of the application's design
- Human analysis of possible attack points (via debugger, hex editor, etc)
- Customized injection to specifically target the application

Intimate Knowledge of the Application Design: The Easier Way

- Monocultures are easy to attack
- Apple Foundation classes well documented
  - Example: **NSURLConnection** (HTTPS POST)
  - Most applications use foundation classes for a number of “secure” operations
- Can Attack every application on the device at once, by infecting the underlying foundation classes

Intimate Knowledge of the Application Design: The Easier Way

Other tempting targets to an attacker

- NSString
- NSData
- NSCoder
- NSKeyedArchiver
- …

Human Analysis

- Apple classes are standardized; work the same away across multiple applications
- Attack the most ubiquitous, standard classes = wide breadth with a single virus / attack
- Automated attacks **without concern for actual application code**
- Easy to infect all new versions of an application (developer will have a hard time “patching out” the hacks)

Customized Injection

- Apple classes are standardized; no targeting of a specific application are necessary
- Attack by simple code insertion:
  - DYLD_INSERT_LIBRARIES
  - cynject / cycript
  - MobileSubstrate
- Debuggers and hex editors optional

Automated Widespread Attack

- Start with a typical zero-day
- OR: Target an individual device with a custom RAM disk
- Inject your code via DYLD, cynject, substrate…
- Code will infect every application on the device
- All applications using base classes affected

Defeating Encryption

- Many applications store master keys in the keychain
- Sogeti’s [free] tools can brute force a 4-digit PIN in under 20 minutes
- Only hardcore nerds use complex passphrases
- CEOs, executives, etc are rarely hardcore nerds

Defeating GOOD Encryption

- – NSProtection classes…
  - Encryption keys unavailable until the user authenticates, soooo…
- – Infect with a worm, wait for user to authenticate
- – Prior to authentication, fopen() either fails, or file is filled with zeroes
- – Just poll the files until the first N bytes are not all zeroes

Defeating Encryption: Spyd

- – Sit and poll encrypted email "Protected Index"
- – Read 128 bytes, wait for nonzero
  - SQLite header, really only need to read first few bytes
- – When file is accessible, copy off, send to remote host via network socket.

… It's as easy as memcmp()

Broad-Based Spyware

- Infect the device (e.g. all applications)
- Attack networking foundation class, steal all SSL-encrypted data before it's even sent
- Send credentials to remote server
- All applications using NSURLConnection susceptible…

Thwarting Broader Attacks

- Don’t rely so heavily on the monoculture
  - Use a solid, but independent encryption implementation, and networking if possible
  - Don’t store encryption keys in the keychain
  - Encryption should depend on a passphrase: use a good key derivation function

Program Logic

- Don’t depend on program logic to enforce security
  - Financial applications that rely on originalPurchasePrice to enforce refundAmount
  - isLoggedIn (REALLY?)
  - Security by obscurity: your functions all nicely wrapped in secretDecryptMyData() can be called directly

Improve Security of the Runtime

- Use dladdr() to verify method/function source files and functions

```
pointer 0x7fff8e7aba62
dli_fname: /System/Library/Frameworks/
Foundation.framework/Versions/C/Foundation
dli_sname: -
[NSMutableURLRequest(NSMutableHTTPURLRequest)
setHTTPBody:]
dli_fbase: 0x7fff8e633000
dli_saddr: 0x7fff8e7aba62
```

Improve Security of the Runtime

- Use dladdr() to verify method/function source files and functions

```
# DYLD_INSERT_LIBRARIES=injection.dylib ./TestConnection
pointer 0x5adc
dli_fname: /private/var/root/injection.dylib
dli_sname: infectDelegateInit
dli_fbase: 0x5000
dli_saddr: 0x5adc
Danger, will robinson! Danger!
```

Jailbreak Detection

- Not really a good “guaranteed” approach, but lots of decent techniques…

- fork() sandbox integrity check
- File system tests
  - /Library/MobileSubstrate/MobileSubstrate.dylib
  - /var/log/syslog
  - /bin/sh
  - Etc…

Jailbreak Detection

- Size and mod times of system files (/etc/fstab, etc)
- Evidence of symlinking (user jailbreaks -> /var/stash)

Questions?

Twitter: @jzdziarski
jonathan@zdziarski.com

VIAFORENSICS