Lockup: A software tool to harden iOS by disabling default *lockdown* services

Luis Gómez-Miralles, Joan Arnedo-Moreno
Internet Interdisciplinary Institute (IN3)
Universitat Oberta de Catalunya
Av. Carl Friedrich Gauss, 5. Parc Mediterrani de la Tecnologia
08860 Castelldefels (Barcelona), Spain
pope,jarnedo@uoc.edu

***Abstract*—Smartphones and mobile devices nowadays accompany each of us in our pockets, holding vast amounts of personal data. The iOS platform has gained popularity in the last years, in particular in enterprise deployments, due to its supposed higher level of security. Recent research has pinpointed a number of mechanisms that are being abused today in order to compromise the security of iOS devices. In this paper, we present *Lockup*, a proof of concept tool that applies various mitigation measures in order to protect iOS devices against those attacks.**

Keywords: Security, Hardening, Privacy, Apple, iOS, iPhone, iPad

I. Introduction

Smartphones have rapidly become ubiquitous in our life. In barely one decade, these small devices have managed to enter our pockets and, nowadays, accompany us at all times, storing all kinds of personal information - often without the user's knowledge: emails and SMS messages, calendars, address books, to-do lists, history of visited places, photographs, voice memos, etc. Moreover, vendors have already started to produce wearable devices which hold an even closer relation with their users, gathering and quantifying diverse data about their life habits - a tendency that will only grow in the future years with devices such as Apple Watch and Google Glass [1].

The rise of mobile technologies has introduced great changes in the information security landscape. Blackberry, the platform that dominated every corporate environment for years thanks to its security features, failed to keep up with its competitors and by Q2 2014 its market share was below 1% [2]. In contrast, 84.7% of the devices sold in that period were Android devices, and 11.7% were iOS devices. However, when it comes to business environments, 67% of new devices activated in a corporate context during the same period were iOS ones [3].

As it tends to happen with every software product, the iOS operating system and the core applications shipped with it have suffered from a number of vulnerabilities in the past, with different degrees of criticality. The most serious ones, for instance, made it possible for remote websites to gain full control over a device browsing them with MobileSafari, the integrated web browser [4]. Fortunately, many of the vulnerabilities uncovered by researchers have been dully patched by Apple in subsequent iOS versions. However, recently, Zdziarski [5] exposed a number of attack vectors against iOS devices through the abuse of certain background services available in all iOS devices. Under certain conditions, these services can leak all kinds of personal data stored in the device, bypassing the optional backup encryption password, and showing no indication at all to the user.

In this paper, we present an analysis of several mitigation techniques that can be used to reduce the attack surface exposed by these services. As a result of this analysis, we introduce *Lockup*, an accompanying software tool that we have created to implement those measures, some of which are novel and, to our knowledge, have not been implemented before. This tool also serves as a proof of concept that such measure can be deployed in an iOS device.

This paper is structured as follows. Section II provides an overview of the iOS security architecture, and presents the problem of potentially dangerous services that can be abused to extract an enormous amount of user data from the device. Section III discusses a number of possible mitigation strategies that can be applied to enhance the device security. Section IV presents *Lockup*, the software tool that we have developed in order to implement those mitigations. Concluding the paper, Section V summarizes the paper contributions and outlines future work.

II. Security and trust in the iOS environment

In this section, we present an overview of the main components of the iOS security architecture and its trust model, and the existing privacy risks. This will allow us to show that, some of the risks originating from certain weaknesses in the iOS trust model have an impact much higher than expected because of a number of iOS background services, named *lockdown*, which have no known legitimate purpose.

A. Remote access via device trust relationship

When it comes to sharing information with an external device (be it a desktop computer, an alarm clock that can play music, a car audio system, etc.) the iOS security model works as follows. Whenever the iOS device is connected via cable to a previously unknown computer (or another external device), it presents a dialog on screen prompting the user whether the computer should be trusted, as seen in Figure

1. Upon receiving the user's consent, both devices create and interchange a series of certificates which from that moment will be used to authenticate each other and initiate a secure, encrypted connection. A pairing record consisting of these certificates is stored in well-known filesystem paths in both the computer and the iOS device. However, There is no way for the user of an iOS device to review the list of external devices he has chosen to trust, or to revoke that trust - other than to reinstall the device completely.

As exposed by [5], [14], a computer that has successfully paired with an iOS device can initiate a connection to it and invoke a number of services exposed via the *lockdown* daemon - even wirelessly and without the user receiving any visual indication. The same can be done from any other computer or device, as long as the pairing record is extracted from the trusted computer.

Unfortunately, a number of the *lockdown* services are designed in such a manner that they may leak significant amounts of personal information, even bypassing the user's backup encryption password. Given that any trusted device (alarm clock, car stereo, etc.) gets a pairing record which gives access to all the services, this can be exploited by either placing malicious devices in common areas (airports, coffee shops...) [14] or compromising trusted devices to steal the pairing record stored in it. Then, those pairing records can be used to establish connections to the iOS device, even over the air through either Wi-Fi or cellular connection, in order to perform surreptitious actions such as deploying malicious software or extracting information from the device.

B. Sensitive iOS device services

A computer that connects to an iOS device (through a USB cable or Wi-Fi) can invoke a series of services which, from the iOS side, are offered through the *lockdown* daemon [5], [14]. These services have diverse roles, such as allowing iTunes syncing or remote management for MDM purposes, while others have no known purpose and seem to be the perfect backdoors to be exploited by intelligence agencies, forensic products, and malicious actors all alike.

The complete list of services can be explored by checking the file */System/Library/Lockdown/Services.plist*. A fresh installation of iOS 7.1.2 on an iPhone 5 exposes a total of 32 services via *lockdown*, of which Zdziarski [5] identified the following ones as being valuable from a forensic standpoint:

- *com.apple.file_relay.* Can be used to extract huge amounts of information, bypassing the backup encryption setting.
- *com.apple.pcapd.* A network sniffer.
- *com.apple.mobile.MCInstall.* Installs managed configurations, such as the ones used in MDM.
- *com.apple.mobile.diagnostics_relay.* Diagnostics: hardware state, battery level...
- *com.apple.syslog_relay.* Various system logs.
- *com.apple.iosdiagnostics.relay.* Network usage statistics per application.
- *com.apple.mobile.installation_proxy.* Used by iTunes to install applications.
- *com.apple.mobile.house_arrest.* Used by iTunes to transfer documents in and out of applications.
- *com.apple.mobilebackup2.* Used by iTunes to backup the device.
- *com.apple.mobilesync.* Used by iTunes to sync certain data such as Safari bookmarks, notes...
- *com.apple.afc.* Exposes the complete *Media* folder - audio, photographies and videos.
- *com.apple.mobile.heartbeat.* Used to maintain the connection to other services being accessed.

As can be seen, the list includes some potentially dangerous services capable of capturing network traffic, installing applications to the device, or dumping the data stored in the device while bypassing the backup encryption password - all of this without showing any indication to the user.

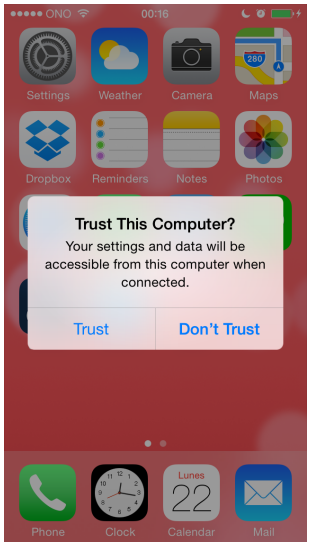

Fig. 1. An iPhone prompting the user whether it should trust the connected computer.

III. Mitigation strategies

There are different mitigation measures that can be applied to cope with the weaknesses introduced by the most sensitive iOS services. We try to summarize the most relevant ones.

A. Delete existing pairing records

One way to mitigate the problem would be to control the number of trust certificates in the iOS device. This is the approach adopted by the *unTrust* tool [15]: it runs in a computer connected to an iOS device connected via USB and removes all pairing records existing in the device except the one for the computer being used to execute the tool.

One drawback of this approach is that the iOS device still keeps trusting one computer - hence there is still the risk that the pairing record is stolen from the computer and used to connect to the device services. In addition, if the user decides or needs to temporarily trust an external device, away from that computer (such as an audio system), there is no way to revoke

that trust or purge the list of trusted devices until the user can get access to the trusted computer and execute *unTrust* again.

B. Limit sensitive services to USB (disable over wireless)

Another approach would be to limit the sensitive services to run only over USB, minimizing the risk for over-the-air attacks. The *lockdown* daemon, responsible for all the sensitive services described in this paper, implements an option (*USBOnlyService*) to limit certain services to USB connections only, disabling the connection to those services over wireless networks. However, this option is only used by one service in iOS 7 (*com.apple.webinspector*, for debugging mobile websites) and one more in iOS 8 (*com.apple.pcapd*, the network sniffer).

C. Disable some services

Finally, it would be ideal to disable the most sensitive services - something that has not been done so far.

The various security measures applied by iOS to any application would prevent us from making these modifications. In order to overcome this, our tool needs to bypass those security measures using the process known as *jailbreak*, as will be explained later.

D. Lock pairing with new devices

Another option worth mentioning is to block pairing with new devices, as implemented by Zdziarski in *pairlock*. This was useful up to iOS 6, given that in those versions external devices would be trusted blindly, without the iOS device presenting any prompt to the user. Since iOS 7 addressed this concern by asking for user permission before trusting new devices, *pairlock* has not been updated to work in iOS 7. Its approach leaves some doors open, as it does not allow the user to revoke existing trust relationships, nor does it address the risk of a pairing record being stolen from a computer or other trusted device.

IV. *Lockup*: iOS HARDENING AND ANTI-FORENSICS

In this paper we present *Lockup*, a software tool that can be installed in devices running iOS versions 7 and 8. As a proof of concept, it *Lockup* hardens the security of the device by addressing the issue of sensitive services using three different approaches:

1) Reducing the attack surface by disabling the most sensitive services. In addition, the user is offered several *profiles*, allowing him to tailor which services are published, and enabling only those needed for the intended use of the device. The rest are eliminated. For instance, a user whose device is not enrolled in MDM systems does not need to allow remote installation of software and configuration profiles.
2) Limiting exploitation opportunities by restricting the rest of services to USB only, eliminating over-the-air threats. This is automatically done in most of the *profiles* mentioned above.
3) Limiting trust relationships by automatically purging all pairing records after a configurable period of time. This constitutes an additional line of defense against attackers capable of stealing a trusted certificate from sources such as the user's computer.

Lockup allows the user to choose between a series of *profiles*, each one increasingly restrictive, depending on what the user needs to do with the device at every moment. In order to define the various *profiles*, we tried a number of configurations, enabling and disabling each service selectively, and attempting various common actions to make an iOS device interact with other external devices. In particular, we tried the following actions:

- Use iTunes in a Mac computer to install applications in the iOS device.
- Use iTunes to transfer files in and out of the applications installed in the iOS device.
- Use iTunes to perform a backup of the data stored in the device.
- Use a bluetooth hands-free device to access the address book of the iOS device and place calls through it.
- Use iPhoto in a Mac computer to import the device's camera roll.
- Use a stereo system to play the audio coming out from the iOS device.

This list illustrates the problems of granting excessive privileges to external devices that access the iOS device's *lockdown* services. If a user does not regularly backup to iTunes, why should those services be exposed when the device is connected to, say, an alarm clock? With *Lockup*, the user can adjust the behavior of the device as needed.

A. Tool capabilities

The main capabilities of *Lockup* can be summarized as follows.

- Controlling the device's trust relationships, by purging the stored pairing records; and
- Disabling certain *lockdown* services and preventing others from being invoked over Wi-Fi connections, in order to disable over-the-air attacks.

One common concern about the potential abuse of iOS services is that iOS lacks a way to see which other devices or computers have been paired with in the past, or to revoke those trust relationships. If the user just hits the wrong button by mistake, the connected device will be trusted forever. This can pose a significant risk, especially considering the possibility of an attacker stealing the pairing record from inside the trusted device and using it to establish remote connections to the iOS device.

In our solution we opted for including a background task that will wipe all trust relationships from the iOS device after a configurable period of time; this applies the mitigation strategy discussed in subsection . Once that happens, connecting to that device will require the user to confirm the trust relationship from the iOS screen. We recommend setting this to low values such as 5-10 minutes, which should suffice for any task involving connection to another device - with the exception

of very long synchronization sessions with iTunes, as usually happens when an iOS device is synced with a computer for the first time. During our tests we observed that even values as low as 30 seconds allow normal functioning of standard features such as iTunes syncing; once an iTunes syncing session has started, it will complete successfully even if the pairing record is deleted in the middle of the process.

In addition, as we have introduced earlier in this paper, there are sensitive services potentially very dangerous and with no known purpose (such as *com.apple.mobile.file_relay*, that can be used to extract all kinds of personal information bypassing the backup encryption protection, or *com.apple.pcapd*, which can be used to turn the device into a sniffer that will capture the network traffic it can receive), and it seems obvious to us that these offending services should be removed from every device.

There are also other services that, despite having a legitimate purpose, can also be exploited to leak significant amounts of personal information or inject malicious software into the device. Examples include *com.apple.mobile.installation_proxy* (used by iTunes to install applications in the device), *com.apple.house_arrest* (used by iTunes to copy application files from or to the device) and *com.apple.mobilebackup2* (used by iTunes to backup the data stored in the device).

We propose to define different service levels and keep the device in the most restrictive level that is suitable depending on what the user needs at every moment – a measure that has not been implemented before, to the authors' knowledge. For instance, it is not necessary to keep all the iTunes-related services enabled unless the user wants to connect the device to iTunes, and even then, it is not necessary to expose those services over-the-air if the user uses a cable to sync. Similarly, a lot of users will prefer to disable the MDM-related services, which can be exploited to install software into their devices. This approach applies the mitigation strategies explained in subsections III-B and III-C.

B. Service profiles

Next, we describe the different profiles that we have implemented in *Lockup*, with each profile being increasingly restrictive and consequently more secure. To the knowledge of the authors, a similar solution has not been implemented before, neither in iOS nor in other platforms. In order to decide which services should be disabled in each profile, we have followed two different criteria.

On one hand, the services that we disable first are those that pose a higher privacy risk to the user. These are, for instance: the services that make it possible to bypass the backup encryption password, to capture network traffic, to deploy configuration profiles and applications to the device, etc.

At the same time, the first services that we disable are the ones likely to be needed by a reduced number of users. We first disable the totally unneeded services, afterwards we disable MDM, and then we disable other features that users may need at particular moments (such app installation via iTunes) while we still allow iTunes to obtain backups of the device data.

The following list details which services are disabled in each level. Levels are incremental, meaning that any given level X also applies all the steps performed in previos levels 1, 2, ... X-1.

- Level 1, for MDM - disables:
  - *com.apple.file_relay.*
  - *com.apple.pcapd.*
- Level 2, for synchronizing applications - disables:
  - *com.apple.mobile.MCInstall.*
  - *com.apple.mobile.diagnostics_relay.*
  - *com.apple.syslog_relay.*
  - *com.apple.iosdiagnostics.relay.*
  - Sets all remaining services to *USBOnly*.
- Level 3, for backup - disables:
  - *com.apple.mobile.installation_proxy.*
  - *com.apple.mobile.house_arrest.*
- Level 4, for synchronizing media files - disables:
  - *com.apple.mobilebackup2.*
  - *com.apple.mobilebackup.*
- Level 5, for media sharing - disables:
  - *com.apple.mobilesync.*
- Level 6, no sensitive services. In addition to all the above, disables:
  - *com.apple.afc.*
- Level 7, no *lockdown* services at all.
  - Completely removes every *lockdown* service, including *com.apple.mobile.heartbeat.*

Even in the strictest mode, the device is still capable of interacting with external devices. In particular, in this mode one can successfully connect the iOS device to: a hands-free device (via bluetooth) to import the address book and place phone calls; a Mac computer (through USB cable) to import pictures through iPhoto; and an audio system (again, through USB cable) to play the songs stored in the device.

C. The iOS jailbreak

The term *jailbreak* (also known in other platforms as *rooting*) refers to the act of circumventing vendor's restrictions in order to run code on the device with full privileges. The use of jailbroken devices is very popular among developers and researchers, as it gives them much more control over the device's internals [7]. Although it is hard to find global data about the number of jailbroken devices, a recent report focused in China found that over 30% of iOS devices being used in that country were jailbroken in January 2013 – a number that fell to 13% by December of the same year [16]. Jailbreak has become increasingly popular among users and there are thousands of applications, both free and paid, that can be installed in jailbroken devices - applications that would never make their way into the official distribution channels, given that they infringe the App Store's rules in one way or another.

Examples include software emulators and all kinds of system-wide tweaks that change the device's global aspect [17], alter global elements such as the Control Center or the Notification Center [18], or inject code into other existing applications to change their behavior [19].

In our case, we have leveraged the jailbreak technique to disable specific *lockdown* services and test different connectivity scenarios, and to develop and test the *Lockup* tool. Other users and researchers can install it and benefit from its features, provided that they are using an iOS version for which a jailbreak is available. And, of course, vendors could implement this kind of tool in future OS versions.

Given that jailbreaking a device deactivates some important security features (code signing and sandboxing), it opens the door to a number of threats [20], [21]. Consequently, users must be careful about the origin of the software packages they install in jailbroken devices. We recommend installing only the core software packages needed by the jailbreak process itself, and always changing the passwords of the *root* user as well as the regular *mobile* user.

Nevertheless we believe our contribution is useful, not only as a proof of concept implementation, but also as a real tool that can be used in a number of real-world scenarios, for instance in old iOS devices which no longer receive software updates from Apple - and which are usually left with an iOS version for which a jailbreak exists.

D. Implementation details

Lockup is designed to run in jailbroken versions of iOS 7 and 8. It can easily be ported to new major iOS versions as soon as a jailbreak is available for them, which typically happens a few weeks after the official iOS release. In the worst case so far, iOS 7.0 took 95 days until a public jailbreak was available for iOS 7; in contrast, iOS 8 was jailbroken 35 days after its official release. It is also remarkable that many users of jailbreak applications usually stick to an older iOS version until a jailbreak for the new one is available.

The different service profiles are defined by creating multiple copies of the */System/Library/Lockdown/Services. plist* file. In each profile, we disable an increasing number of services. In addition, in most of the profiles, the flag *USBOnlyService* is applied to sensitive services, so that these cannot be abused over the air, either via a Wi-Fi connection, or through the user's cellular connection.

In order to set a profile, the user executes the command *lockup-profile*. This can be done either using a terminal application such as *MobileTerminal* or accessing the device via SSH, if it has been installed. When the command is invoked, the user is presented with a menu as shown in Figure 2. After the user picks a profile, the corresponding service list file is copied over */System/Library/Lockdown/Ser- vices.plist*. For the changes to take immediate effect, a *SIGTERM* signal is sent to the *lockdown* daemon with the *kill* command, which makes it restart and read its new configuration file. Additional options allow the user to enumerate the services exposed by the present profile and dump the whole contents of the *Services.plist* file, which may be specially useful to detect and investigate additional services that may have been installed inadvertently.

```
Current profile: 0. Default iOS service set. Wi-Fi sync ON.

Available profiles:
0. Default iOS service set. Wi-Fi sync ON.
1. MDM environments. Wi-Fi sync ON.
2. Sync apps. Wi-Fi sync OFF.
3. Backup device data. Wi-Fi sync OFF.
4. Sync media files. Wi-Fi sync OFF.
5. Share media files. Wi-Fi sync OFF.
6. No sensitive services. Wi-Fi sync OFF.
7. No services at all (paranoid mode).
--
9. ABORT - Exit this program.
D. Dump installed profile.
E. Enumerate services exposed by the current profile.

Press key and ENTER...
```

Fig. 2. Menu presented by *lockup-profile*.

For the periodic purging of pairing records, *Lockup* uses various files. First, a shell script in charge of deleting the pairing records is installed. Secondly, a *launch daemon*, which will run the previous script periodically, is loaded through */System/Library/LaunchDaemons/es.pope.lockup-purge.plist*. An additional script, *lockup-interval*, can be used to change the interval at which pairing records are deleted (one hour by default). Figure 3 summarizes the main components and the interactions between them.

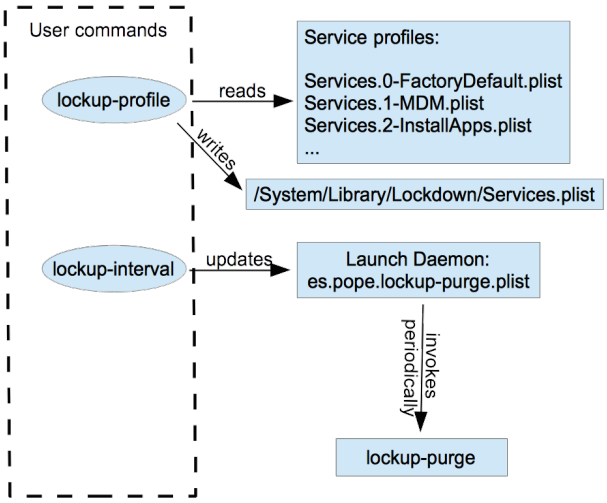

Fig. 3. *Lockup* components and main interactions.

V. Conclusions and Future Work

In this paper we have reviewed the security and privacy risks presented by certain background services that exist in the iOS operating system. We have presented a number of mitigation measures that can be used to reduce those risks. The main contribution of this paper is *Lockup*, a software tool that hardens the security of iOS devices by defining a number of *profiles* which reduce the number of exposed services. In

addition, we have discussed the anti-forensic implications of our solution, and the anti-anti-forensics countermeasures that could be used to bypass it. Given the huge amount of personal information that can be extracted by abusing these sensitive services, we believe it is worth exploring this kind of solutions. The expected rise of wearable devices will only increase the need for solutions that enhance the devices' security and privacy levels.

Lockup will be released as free software so that other researchers or developers can adapt it as they find convenient. We also have the intention of continuing working in *Lockup*, maintaining it and adding new features, such as: monitoring and logging connection attempts to *lockdown* services, alerting the user in real time; adding a graphical interface to the software; monitoring the set of available services and alerting the user if new services are added... It would also be possible to integrate it with other solutions such as *activator* [24]. However, from a security standpoint, it would be preferable to keep the software as simple as possible, both in terms of size and in terms of dependencies.

The purpose of the proof-of-concept tool presented in this paper is to fight the security risks presented by a number of iOS unwanted services. It must be kept in mind, however, that our solution will only work in jailbroken devices, and the process of jailbreaking itself implies circumventing and disabling a number of native iOS security mechanisms.

Future research work includes the possibility of creating custom jailbreak tools that after deploying our software return the device to its original state to the best possible extent - this would keep most of the benefits and security features of stock Apple devices, while avoiding exposure through unwanted services.

Acknowledgements

This work was partly funded by the Spanish Government through projects TIN2011-27076-C03-02 CO-PRIVACY and SMARTGLACIS (TIN2014-57364-C2-2-R).

References

[1] T. Starner, Wearable computing: Through the looking glass, in: Proceedings of the 2013 International Symposium on Wearable Computers, ISWC '13, ACM, New York, NY, USA, 2013, pp. 125–126.

[2] International Data Corporation, Worldwide quarterly mobile phone tracker Q2 2014, International Data Group, 2014.

[3] Good Technology, Good Technology mobility index report Q2 2014, http://media.www1.good.com/documents/rpt-mobility-index-q2-2014.pdf (2014).

[4] N. Allegra, J. Freeman, JailbreakMe 3.0, http://jailbreakme.com (2011).

[5] J. Zdziarski., Identifying back doors, attack points, and surveillance mechanisms in ios devices, Digital Investigation 11 (2014) 3–19.

[6] Apple Computer, App Store sales top $10 Billion in 2013, http://www.apple.com/pr/library/2014/01/07App-Store-Sales-Top-10-Billion-in-2013.html (2014).

[7] C. Miller, D. Blazakis, D. D. Zovi, S. Esser, V. Iozzo, R. Weinmann, iOS hacker's handbook, Wiley, 2012.

[8] A. Masna, Camera+ pulled from the App Store over hidden feature, http://www.macworld.com/article/1153337/cameraplus_pulled.html (2010).

[9] I. Fried, Emulator runs DOS, Windows on an iPad, http://www.cnet.com/news/emulator-runs-dos-windows-on-an-ipad/ (2010).

[10] M. Rose, Arcade emulator iMAME punted out of App Store, http://www.tuaw.com/2011/12/23/arcade-emulator-imame-punted-out-of-app-store/ (2011).

[11] M. Beasley, Rogue App Store app lets you hide built-in apps and disable iAds, http://9to5mac.com/2013/03/11/rogue-app-store-app-lets-you-hide-built-in-apps-and-disable-iads/ (2013).

[12] Apple Computer, Resources for IT and enterprise developers, https://developer.apple.com/enterprise/ (2014).

[13] Apple Computer, iPhone in business, https://www.apple.com/iphone/business/ios (2014).

[14] B. Lau, Y. Jang, C. Song, T. Wang, P. ho Chung, P. Royal, Mactans: injecting malware into iOS devices via malicious chargers (2013).

[15] Stroz Friedberg, unTRUST, https://github.com/strozfriedberg/unTRUST (2014).

[16] Umeng, Umeng Insight report: China mobile internet 2013 (2014).

[17] J. Freeman, WinterBoard, http://cydia.saurik.com/package/winterboard/ (2014).

[18] D. Lisiansky, CCControls, http://cydia.saurik.com/package/com.danyl.cccontrols/ (2014).

[19] J. Freeman, Cydia Substrate, http://www.cydiasubstrate.com (2014).

[20] A. Apvrille, Inside the iOS/AdThief malware, https://www.virusbtn.com/pdf/magazine/2014/vb201408-AdThief.pdf (2014).

[21] P. Porras, H. Sadi, V. Yegneswaran, An analysis of the ikee.b iphone botnet, in: A. Schmidt, G. Russello, A. Lioy, N. Prasad, S. Lian (Eds.), Security and Privacy in Mobile Information and Communication Systems, Vol. 47 of Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Springer Berlin Heidelberg, 2010, pp. 141–152.

[22] K. Oestreicher, A forensically robust method for acquisition of icloud data, Digital Investigation 11, Supplement 2 (0) (2014) S106 – S113, fourteenth Annual {DFRWS} Conference.

[23] K. Ruan, J. Carthy, T. Kechadi, I. Baggili, Cloud forensics definitions and critical criteria for cloud forensic capability: An overview of survey results, Digital Investigation 10 (1) (2013) 34 – 43.

[24] R. Petrich, Activator, https://rpetri.ch/cydia/activator/ (2014).