

APPLE WITHOUT A SHELL – IOS UNDER TARGETED ATTACK

Tao Wei, Min Zheng, Hui Xue & Dawn Song
FireEye, Inc., USA

Email {tao.wei, min.zheng, hui.xue, dawn.song}@fireeye.com

ABSTRACT

Apple has a strict review process for apps published in its *App Store*. The review guidelines not only disallow use of the powerful private APIs but also forbid dangerous or deceptive behaviours. The review process, though imperfect, provides good protection for *iOS* users and makes it difficult for malware to exist in the *App Store*. However, apps may also be distributed using enterprise provisioning profiles without having to go through such a review process. Apps distributed in this way have become a new attack vector. Attackers can launch targeted attacks by delivering malicious apps leveraging private APIs to the victim's device. In this paper, we explain the risk of an attacker distributing apps using enterprise provisioning profiles to conduct targeted attacks, including remote installation through spear phishing, autostart after reboot, background monitoring and bypassing certificate revocation. We show that serious, targeted attacks on *iOS* are both feasible and realistic. We also discuss the implications this has on the *iOS* security architecture and the challenges of addressing them.

1. INTRODUCTION

By the end of 2013, the number of *iOS* users had reached 800 million [1] and there were over one million apps in the *iOS App Store* [2]. Despite the platform's popularity, little *iOS* malware has been discovered [3]. It has also been reported [4] that *iOS* is more secure than *Android* due to its controlled distribution channel and rigorous app review process. However, there are still potential risks for *iOS* systems.

There are limited attack surfaces for traditional targeted attacks against *iOS* devices. While distributing malware through the *App Store* is difficult, spear phishing and drive-by downloads are not easy either. Attacks against *Safari* and PDF readers call for advanced skills, and *Apple* can fix these vulnerabilities by pushing out updates quickly.

In contrast, this paper describes a new type of security risk for *iOS* devices, where attackers may potentially utilize a bigger attack surface, which becomes harder to fix. This new type of risk leverages *Apple's* enterprise program that can distribute apps to an unlimited number of devices without going through *Apple's* app review process. By bypassing the review process, a malicious app can employ powerful private APIs hidden in *iOS's* frameworks to steal sensitive information and attack various vulnerabilities on the system.

In contrast to traditional attacks that have limited attack surface, a malicious app installed through enterprise provisioning profiles can conduct malicious behaviours by abusing private APIs,

deceiving users with fake UIs, or exploiting all kinds of known or even zero-day vulnerabilities. It's hard for *Apple* to cope with apps outside of the *App Store*, which don't comply with its review guidelines, and hard for it to stop them from attacking existing vulnerabilities.

The malicious apps may also use tricks to ensure that they are launched automatically after the system reboots and that they keep running in the background continuously. Given that these apps can use private APIs, they may, for example, monitor the user continuously by silently logging the user's inputs in the background, even without bypassing the sandbox.

In this paper, we study the security risk posed by this new attack method and examine every step involved in a potential campaign. In section 2, we describe *Apple's* review process for apps in the *App Store* and what kind of protections it enforces. Section 3 explains the power of private APIs, and Section 4 explains how enterprise distribution works. Section 5 studies the new attack vector made possible by using private APIs in apps distributed using enterprise provisioning. Section 6 discusses related issues, including the implications of *iOS* security architecture and the challenges in addressing them. Section 7 gives a conclusion.

2. APPLE REVIEW PROCESS

Apple's review process enforces a set of review guidelines [5], which includes over 100 rules. The rule categories cover various aspects, such as user interface, location, push notifications, trademarks, violence, religion, gambling, charities, privacy and advertising. Here are some examples of the rules extracted from [5]:

- Apps that crash will be rejected.
- Apps that include undocumented or hidden features inconsistent with the description of the app will be rejected.
- Apps that use non-public APIs will be rejected.
- Apps that read or write data outside their designated container areas will be rejected.
- Apps that download code in any way or form will be rejected.
- Apps that install or launch other executable code will be rejected.
- Apps that duplicate apps already in the *App Store* may be rejected, particularly if there are many of them, such as fart, burp, flashlight, and Kama Sutra apps.
- Apps that are intended to provide trick or fake functionality that are not clearly marked as such will be rejected.
- Multitasking apps may only use background services for their intended purposes: VoIP, audio playback, location, task completion, local notifications, etc.
- Apps that browse the web must use the *iOS* WebKit framework and WebKit JavaScript.
- If you attempt to cheat the system (for example, by trying to trick the review process, steal data from users, copy another developer's work, or manipulate the ratings) your apps will

be removed from the store and you will be expelled from the developer program.

- Apps that create alternate desktop/home screen environments or simulate multi-app widget experiences will be rejected.
- Apps cannot transmit data about a user without obtaining the user’s prior permission and providing the user with access to information about how and where the data will be used.
- Location data can only be used when directly relevant to the features and services provided by the app to the user or to support approved advertising uses.

Apple uses the review process to prevent apps from conducting undesirable behaviours. However, if attackers can bypass the review process, they can break all these rules and carry out malicious behaviours that have severe security consequences on a victim’s device. For example, attackers can use *iOS* private APIs for powerful attacks.

3. PRIVATE APIS

iOS apps interact with the underlying system using Application Programming Interfaces (APIs). However, not all APIs are equally open to app developers. Apple forbids some of the APIs, known as ‘private APIs’, from being used in the apps on *App Store*. Apple stipulates that these private APIs should only be used by the framework classes internally or by the *iOS* system apps [5], and these private APIs remain undocumented.

Private APIs are considerably more powerful than their public API counterparts. For example, on *iOS 6.0*, one app can call some public *Twitter* APIs to post a Tweet on the user’s *Twitter* page (Figure 1) and the user must consent by clicking the ‘post’ button. On the contrary, by using private APIs, the app can post the Tweet without notifying the user [8] at all.

Though Apple forbids the use of private APIs, and provides no documentation about them, an attacker can still obtain a list of private APIs. To do so, one can begin by using otool [12] or classdump [13] to obtain a complete list of APIs, both public and private, from the *iOS* framework binaries shipped within the SDK. One can then obtain private APIs by subtracting the



Figure 1: Public *Twitter* APIs are called to post a Tweet on the user’s *Twitter* page – the user must click the ‘post’ button.

documented public APIs [14]. Table 1 lists several examples of private APIs.

Review process vs. private APIs

Apple forbids apps in the *App Store* from using private APIs, and bans app developers/vendors who do so. In February 2012, Apple banned all apps from *Qihoo* [6], a prominent Chinese anti-virus, web browser and search engine vendor. This major incident happened because *Qihoo* used *iOS* private APIs and encrypted the function calls in its *iOS* apps – Apple has a policy that forbids any non-Apple apps in its *App Store* from using private APIs.

4. DISTRIBUTING IOS APPS THROUGH ENTERPRISE PROVISIONING

Besides the *iOS App Store*, *iOS* apps can also be distributed under enterprise provisioning profiles to an unlimited number of users. The *iOS Developer Enterprise Program* [15] enables a company to sign in-house apps with its enterprise distribution certificate and distribute the apps to employees using enterprise provisioning profiles.

In practice, many app developers use this venue to distribute apps to the public [10]. As mentioned before, apps distributed in this manner don’t have to go through Apple’s review process [5] and don’t have to conform to the rules in Apple’s guidelines on

Method	Framework	Usage	iOS 6.x availability	iOS 7.x availability
CTSIMSupportCopyMobileSubscriberIdentity()	Core telephony	Get Device IMSI	Yes	No
[[UIDevice currentDevice] UniqueIdentifier]	UIKit	Get Device UDID	Yes	No
SBSCopyApplicationDisplayIdentifiers()	SpringBoardServices	Get the array of current running app bundle IDs	Yes	No
[[CTMessageCenter sharedMessageCenter] incomingMessageWithId: result]	Core telephony	Get the text of the incoming SMS message	Yes	Yes
MobileInstallationLookup()	Mobile installation	Get the bundle ID list of installed iOS apps	Yes	Yes

Table 1: Private API examples.

library usage, privacy, user interface, etc. Thus, not only can these apps freely use private APIs, they can also do other tricks such as mimicking apps originally bundled with the device, such as *App Store* or *iTunes Store*, or creating alternative home screen environments.

Distributing apps using enterprise provisioning profiles combined with unregulated usage of private APIs creates a new attack vector that enables attackers to distribute malware leveraging private APIs. Benign apps distributed under enterprise provisioning profiles also become valuable targets for attackers since many of them use private APIs.

4.1 Revocation of enterprise certificates

Apple may revoke an enterprise distribution certificate, if it suspects abuse. Revoking a distribution certificate invalidates all of the apps that have been signed with it. Apple allows the enterprise apps to be used by employees of the developer company only, rather than by everyone in the public. For this reason, Apple revoked the enterprise distribution certificate of Qihoo, which released its ‘enterprise’ apps to the public [7].

Apple uses the Online Certificate Status Protocol (OCSP) to validate enterprise certificates. According to the iOS Deployment Technical Reference [15], the first time a user opens an app distributed using the enterprise provisioning profile, iOS contacts Apple’s OCSP server to validate its distribution certificate. A revoked distribution certificate will prevent the app from launching. The OCSP response will be cached on the device for three to seven days [15]. However, ‘inability to contact or get a response from the OCSP server isn’t interpreted as a revocation’ [15]. That means iOS won’t prevent the app from launching if it can’t reach the OCSP server.

4.2 Real-world apps distributed through enterprise provisioning

We collected 1,408 apps from the Internet which were distributed through enterprise provisioning. We parsed each app’s Info.plist file to determine its development region. As shown in Table 2, most apps were from the United States, China, England and France.

Country	Number of apps
United States	660
China	361
England	223
France	62
Others	102
All	1408

Table 2: App numbers by development region.

Since these apps don’t go through Apple’s review process, they can abuse the powerful private APIs. We found that, within these 1,408 apps, 844 (60%) used private APIs.

5. TARGETED ATTACKS THROUGH ENTERPRISE PROVISIONING

Figure 2 shows the steps in a targeted attack using enterprise certificates. Conceptually, the attacker first sends out a spear phishing email or SMS to the victim, who may be lured to click on a link and install the app. Once the victim launches the app, it can leverage private APIs and some exploits to keep monitoring the user, steal sensitive information in the background, and avoid being invalidated by Apple.

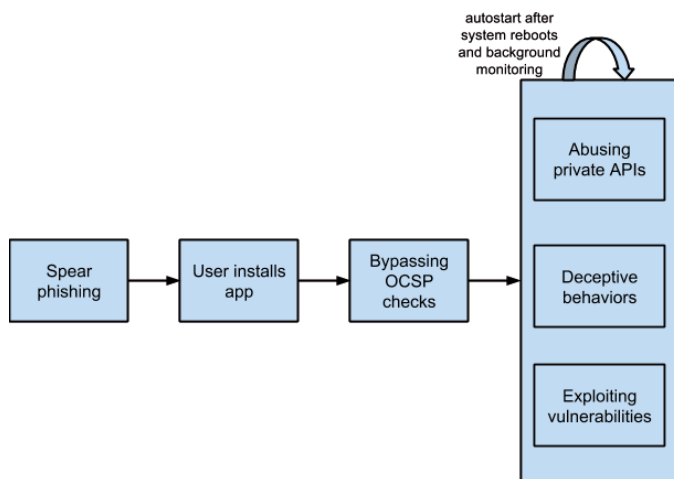


Figure 2: Targeted attacks against iOS through enterprise provisioning.

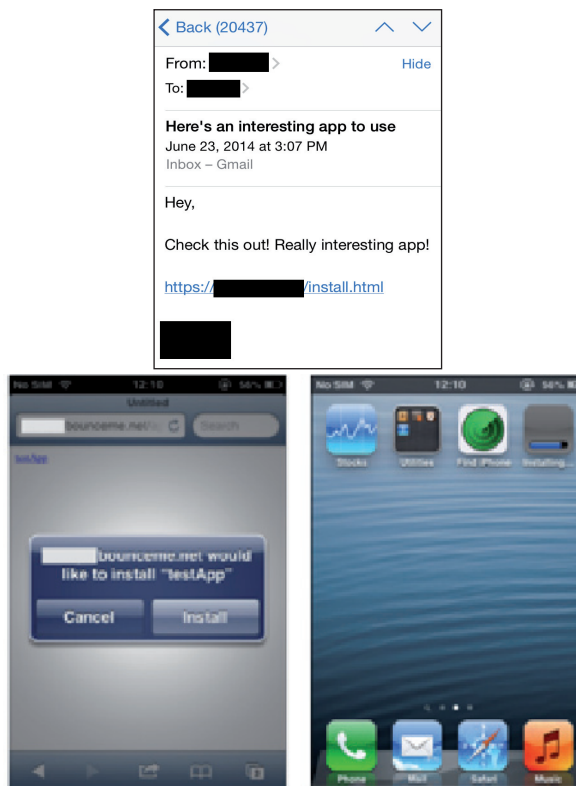


Figure 3: Installing an enterprise app.

5.1 Spear phishing through enterprise provisioning

An attacker may lure a victim to install an app through a spear-phishing email or SMS that contains a web link. Once the user clicks on the link, there will be a pop-up letting the user install the app, as shown in Figure 3. If the user chooses to install and launch the app, it can, with the help of private APIs, keep monitoring the user and steal sensitive information in the background.

5.2. Persistent monitoring

After installation, the malicious app can monitor the victim's activity continuously, including on occasions when it is switched to the background or the system reboots.

While malware can use standard 'background app refresh' to monitor the system continuously, *iOS7* provides a setting for 'background app refresh' that will disable unnecessary background refreshing, and may prevent malware from working. However, this can be bypassed. For example, an app can play music in the background without turning on its 'background app refresh' switch. Thus a malicious app can disguise itself as a music app to conduct background monitoring.

On *iOS*, ordinary apps can't start automatically after rebooting. However, VoIP apps are allowed to start automatically after the system reboot. *Apple* forbids non-VoIP apps in the *App Store* from using this feature. However, without being regulated by *Apple*'s review process, the attacker can disguise a malicious app as a VoIP app, which enables the app to start automatically after the device reboots, and thus monitor the victim continuously. Specifically, the malicious app can include the 'voip' value in the `UIBackgroundModes` key so that the system allows it to run in the background and launches it in the background again after system reboot.

5.3 Disabling OCSP

Apple will validate the status of enterprise certificates roughly every three to seven days, at which point it has the chance of finding some abnormal behaviour and disabling the corresponding apps. To prevent this, attackers can disable OCSP.

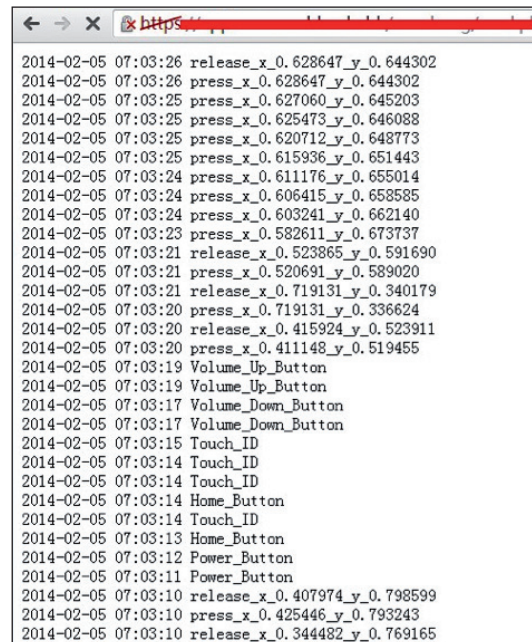
Attackers may leverage existing exploits to modify the device's OCSP cache to maintain a valid state for its certificate.

Based on the findings from Wang *et al.* [16], under certain conditions, `syslogd` will do 'chmod 777' and 'chown mobile' to '/var/mobile/Library/Logs/CrashReporter'. Thus, the malicious app can evade the sandbox and replace /var/mobile/Library/Logs/CrashReporter with a symbolic link to some other part of the system. This will be changed to writable by `syslogd`, which doesn't carry out proper checks on symbolic links. The malicious app can then modify the OCSP cache to keep its OCSP response valid all the time.

5.4 Attacks by abusing private APIs

Private APIs are powerful. However, since private APIs are not intended to be available to app developers, their design may not have sufficient security considerations.

In February 2014, we found a vulnerability in *iOS* private APIs [11] which meant that a malicious app making use of certain private APIs can monitor a user's input. In this attack, a malicious app can use a private API `IOHIDEventSystemClientRegisterEventCallback()` method within `IOKit.framework` to register a callback to receive system-wide user input. This vulnerability can enable malware to record all of the user's touch/press events in the background, including touches on the screen, home button press, volume button press, and TouchID presses, as shown in Figure 4. Attackers can use such information to reconstruct every character the victim inputs. Upon our notification, *Apple* issued CVE-2014-1276 for this issue and pushed out a fix shortly afterwards.



```

2014-02-05 07:03:26 release_x_0.628647_y_0.644302
2014-02-05 07:03:26 press_x_0.628647_y_0.644302
2014-02-05 07:03:25 press_x_0.627060_y_0.645203
2014-02-05 07:03:25 press_x_0.625473_y_0.646088
2014-02-05 07:03:25 press_x_0.620712_y_0.648773
2014-02-05 07:03:25 press_x_0.615936_y_0.651443
2014-02-05 07:03:24 press_x_0.611176_y_0.655014
2014-02-05 07:03:24 press_x_0.606415_y_0.658585
2014-02-05 07:03:24 press_x_0.603241_y_0.662140
2014-02-05 07:03:23 press_x_0.582611_y_0.673737
2014-02-05 07:03:21 release_x_0.523865_y_0.591690
2014-02-05 07:03:21 press_x_0.520691_y_0.589020
2014-02-05 07:03:21 release_x_0.719131_y_0.340179
2014-02-05 07:03:20 press_x_0.719131_y_0.336624
2014-02-05 07:03:20 release_x_0.415924_y_0.523911
2014-02-05 07:03:20 press_x_0.411148_y_0.519455
2014-02-05 07:03:19 Volume_Up_Button
2014-02-05 07:03:19 Volume_Up_Button
2014-02-05 07:03:17 Volume_Down_Button
2014-02-05 07:03:17 Volume_Down_Button
2014-02-05 07:03:15 Touch_ID
2014-02-05 07:03:14 Touch_ID
2014-02-05 07:03:14 Touch_ID
2014-02-05 07:03:14 Home_Button
2014-02-05 07:03:14 Touch_ID
2014-02-05 07:03:13 Home_Button
2014-02-05 07:03:12 Power_Button
2014-02-05 07:03:11 Power_Button
2014-02-05 07:03:10 release_x_0.407974_y_0.798599
2014-02-05 07:03:10 press_x_0.425446_y_0.793243
2014-02-05 07:03:10 release_x_0.344482_y_0.769165

```

Figure 4: Background monitoring.

We have since found (and notified *Apple* about) another flaw on *iOS 7* devices that enables telephone and SMS activity to be monitored from the background. Malware can register a callback by using the 'CTTelephonyCenterAddObserver' function in the `CoreTelephony.framework` and then it can record all of the telephone and SMS events in the background, including incoming phone number, the SMS sender's number and SMS content, and then it can send all user events to any remote server. In this way, malware can eavesdrop on sensitive communication and bypass two-factor authentication based on SMS.

5.5 Attacks by deceptive behaviours

The *App Store* review guidelines [5] list many app behaviours as forbidden. However, these behaviours become possible for an app distributed with enterprise provisioning. We list two cases where the attacker can break the guidelines for malicious activities. Attackers can do more based on their social engineering techniques:

- A malicious app may create alternative home screen environments or mimic the apps bundled on *iPhone*, such as *App Store* and *iTunes Store*. By doing so, the attacker can trick the user into using a fake *iTunes Store* app, and prompt the user to enter their password. For a user who lacks security knowledge, this phishing prompt may be enough for the attacker to steal the user's *Apple ID* password successfully.
- A malicious app may also disguise itself as another popular app and lure the user to use it. The attacker can embed malicious code inside such fake apps to carry out further attacks.

5.6 Attacks by using root exploits

It's known that *Apple* can't fix all known vulnerabilities, or may fix them incorrectly [16]. Malware installed through enterprise provisioning has more freedom to exploit known or zero-day vulnerabilities. Attackers can even use dynamic code downloading to prevent the exploit from being captured by security vendors.

6. DISCUSSION

6.1 UDID and ad-hoc provisioning

Besides enterprise certificates, apps can also be distributed using ad-hoc provisioning. Compared with enterprise certificates, ad-hoc distribution has the limitation that each development account can only distribute to 100 devices per membership year. Each of the devices receiving apps will have its unique device ID (UDID) registered in the ad-hoc provisioning profile to use the app.

However, stealing the UDID from a target device on which the attacker wants to install a malicious app is not a hard job. Previously stealing the UDID may be treated just as private information leakage. However, stealing the UDID is a crucial link towards targeted attacks: attackers can use the UDID to deliver ad-hoc distributed apps to the victim's phone.

6.2 Abusing private APIs through bypassing review

Since private APIs are loaded as framework code into the app's address space, together with the app developer's own code, there are no obstacles to calling private APIs from a technical perspective. *Apple* does prohibit doing so. However, works like Jekyll [8] have shown the possibility of bypassing *Apple*'s review process.

Fooling *Apple* still has the risk of being banned once caught [6]. However, since distributing apps using enterprise certificates avoids the *App Store*, there's no regulation on usage of private APIs. Currently, *Apple* doesn't have an ideal way to supervise and manage these enterprise certificates.

6.3 Challenges of iOS security architecture against targeted attacks

While *Apple* does a good job of protecting ordinary *App Store*

users from being infected by malware, there is still a big gap for enterprise security. Targeted attacks through enterprise provisioning pose a severe threat for enterprise users. Once attackers compromise victims' devices, they can access useful information such as intellectual property, steal numerous accounts of cloud services, and take photos or record audio/video through *iOS* devices.

Currently, security on *iOS* runs into a dilemma: *Apple* doesn't allow security vendors to implement system-level protections, whereas malware can freely call powerful private APIs and exploit vulnerabilities through enterprise provisioning. Furthermore, since most *iPhones* can access the Internet directly through their carriers (e.g. *AT&T* and *Verizon*) when they are not connected to a company-managed wireless network, classic network security devices in company networks can't protect these devices all the time.

In the long run, *Apple* needs more investment in improving enterprise-level security against advanced targeted attacks. *Apple* should consider bringing dedicated security vendors into its platform to help with enterprise-level security solutions.

7. CONCLUSION

Though *Apple* enforces a rigorous review process forbidding apps on *App Store* from conducting many dangerous/deceptive behaviours, enterprise provisioning becomes a valid venue for apps to circumvent *Apple*'s regulations. Apps distributed using enterprise provisioning profiles can abuse powerful private APIs, deceive users and exploit vulnerabilities, thus becoming a severe threat to enterprise users. Using these apps, an attacker can use a bigger attack surface to launch persistent and targeted attacks against the victim's device. *Apple* may improve its architecture to co-operate with security vendors in order to provide a better enterprise-level security solution.

REFERENCES

- [1] Tim Cook to shareholders: iPhone 5s/c outpace predecessors, Apple bought 23 companies in 16 months. <http://appleinsider.com/articles/14/02/28/tim-cook-at-shareholder-meeting-iphone-5s-5c-outpace-predecessors-apple-bought-23-companies-in-16-months>.
- [2] How Many Apps Are in the iPhone App Store. <http://ipod.about.com/od/iphonesoftwareterms/qt/apps-in-app-store.htm>.
- [3] Felt, A. P.; Finifter, M.; Chin, E. Hanna, S.; Wagner, D. A survey of mobile malware in the wild. In Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, pp.3–14. ACM, 2011.
- [4] When Malware Goes Mobile. <http://www.sophos.com/en-us/security-news-trends/security-trends/malware-goes-mobile/why-ios-is-safer-than-android.aspx>.
- [5] App store review guidelines. <https://developer.apple.com/appstore/resources/approval/guidelines.html>.

- [6] Apple Bans Qihoo Apps From iTunes App Store. February 2012. <http://www.techinasia.com/apple-bans-qihoo-apps/>.
- [7] Qihoo Double Blow as iOS Apps Banned by Apple, China Warns of Anti-Competitive Practices. January 2013. <http://www.techinasia.com/qihoo-apps-banned-apple-app-store/>.
- [8] Wang, T.; Lu, K.; Lu, L.; Chung, S.; Lee, W. Jekyll on iOS: when benign apps become evil. Presented as part of the 22nd USENIX Security Symposium, pp.559–572, 2013.
- [9] CVE-2014-1276. <http://support.apple.com/kb/HT6162>.
- [10] How Apple’s Enterprise Distribution Program was abused to enable the installation of a GameBoy emulator. 2014. <http://www.imore.com/how-gameboy-emulator-finding-its-way-non-jailbroken-devices>.
- [11] Background Monitoring on Non-Jailbroken iOS 7 Devices – and a Mitigation. <http://www.fireeye.com/blog/technical/2014/02/background-monitoring-on-non-jailbroken-ios-7-devices-and-a-mitigation.html>.
- [12] otool. <https://www.opensource.apple.com/source/cctools/cctools-499/otool/>.
- [13] classdump. <http://stevenygard.com/projects/class-dump/>.
- [14] iOS Developer Library Reference. <https://developer.apple.com/library/ios/navigation/#section=Resource%20Types&topic=Reference>.
- [15] iOS Deployment Technical Reference. http://images.apple.com/iphone/business/docs/iOS_Deployment_Technical_Reference_EN_May14.pdf.
- [16] Exploiting unpatched iOS vulnerabilities for fun and profit. <https://www.blackhat.com/us-14/briefings.html#exploiting-unpatched-ios-vulnerabilities-for-fun-and-profit>.