
MACTANS: INJECTING MALWARE INTO IOS DEVICES VIA MALICIOUS CHARGERS

*BILLY LAU, YEONGJIN JANG, CHENGYU SONG,
TIELEI WANG, PAK HO CHUNG, AND PAUL ROYAL
GEORGIA INSTITUTE OF TECHNOLOGY, ATLANTA, GA, USA
BILLY@CC.GATECH.EDU, YEONGJIN.JANG@GATECH.EDU,
CSONG84@GATECH.EDU, TIELEI@GATECH.EDU,
PCHUNG34@MAIL.GATECH.EDU, PAUL@GTISC.GATECH.EDU*

1. ABSTRACT

Apple iOS devices are considered by many to be more secure than other mobile offerings. In evaluating this belief, we investigated the extent to which security threats were considered when performing everyday activities such as charging a device. The results were alarming: despite the plethora of defense mechanisms in iOS, we successfully injected arbitrary software into current-generation Apple devices running the latest iOS software. All users are affected, as our approach requires neither a jailbroken device nor user interaction.

In this paper, we show how an iOS device can be compromised within one minute of being plugged into a malicious charger. We first examine Apple's existing security mechanisms to protect against arbitrary software installation, and then describe how USB capabilities can be leveraged to bypass these defense mechanisms. To demonstrate persistence of the resulting infection, we detail how an attacker can hide their software in the same way Apple hides its own built-in applications.

To demonstrate practical application of these vulnerabilities, we built a proof-of-concept malicious charger, called *Mactans*, using a BeagleBoard. This hardware was selected to demonstrate the ease with which innocent-looking but malicious USB chargers can be constructed. While *Mactans* was built with a limited amount of time and a small budget, we also briefly consider what more motivated, well-funded adversaries could accomplish. Finally, we recommend ways in which users can protect themselves and suggest security features Apple could implement to make the attacks we describe substantially more difficult to pull off.

2. INTRODUCTION

In recent years, consumers have shifted their attention from traditional computing devices such as PCs and laptops to mobile devices such as smart phones and tablets. Although these devices come with new security features, they are not bulletproof. Given the predominance of Apple's iOS devices, we set out to explore how well users are protected against various types of attacks when using products such as the iPhone or iPad.

Currently, iOS is considered by many to be more secure than other mobile offerings, based on security mechanisms such as mandatory code signing, app sandboxing, and a single, centralized app store. In evaluating this belief, we examined how Apple's existing security mechanisms protect against arbitrary software installation and execution. Specifically, we investigated the extent to which security threats were considered when performing everyday activities such as charging a device. The results were alarming: despite the plethora of defense mechanisms in iOS, we successfully injected arbitrary software (in a surreptitious manner) into current-generation Apple devices running the latest operating system. So far, this attack works on devices equipped with iOS versions up to and including iOS 6.

To demonstrate practical application of these vulnerabilities, we built a proof-of-concept malicious charger, called *Mactans*, using a BeagleBoard. This hardware was selected to demonstrate the ease with which innocent-looking but malicious USB chargers can be constructed. The name was chosen to portray the characteristics of a species of spider commonly known as the Black Widow, whose bite delivers potent neurotoxin that can be deadly to humans. While *Mactans* was built with a limited amount of time and a small budget, in **Section 5.1** we also briefly consider what more motivated, well-funded adversaries could accomplish.

To provide a defense against such attacks, in **Section 5.3** we recommend mitigations that Apple could implement to make the attacks we describe substantially more difficult to pull off. Following our disclosure to Apple, we received an email from Apple Product Security that invited us to test iOS 7 Beta 2. Upon examination, we discovered that Apple had implemented one of our recommendations to require user consent for an iOS device to be paired with an unknown host for the first time.

3. OBSERVATIONS

We begin by describing observations made during the course of our security research that enabled us to circumvent existing security features of iOS, install and execute arbitrary apps. The attack consists of injecting an arbitrary app into an iOS device through a USB cable connected to a custom-made malicious charger. Additional details are provided in **Section 4**, while possible mitigations are suggested in **Section 5**.

3.1 PHYSICAL WEAKNESS DESCRIPTIONS

The weaknesses we describe affect iOS devices up to and including those running iOS 6. Following the client-server communication model, we refer to the iOS device as client, and the entity that attempts to communicate with the client as the host. Using this terminology, below is a list of weaknesses we discovered.

3.1.1 ANY HOST CAPABLE OF ESTABLISHING A SESSION WITH THE CLIENT IS IMPLICITLY TRUSTED BY THE CLIENT

As a result, without the user's permission, any host that understands the proprietary RPC communications protocol like that used by iTunes to communicate with an iOS device can similarly and directly query or modify the state of the client. We note that this communication protocol lacks proper authentication and assumes trust too broadly. The consequence of this weakness is further described in **Section 4**.

3.1.2 THE CLIENT DOES NOT SEEK THE USER'S CONSENT FOR ACTIONS THAT WOULD ALTER ITS STATE AND PROVIDES NO INDICATION TO THE USER WHEN ITS STATE (I.E., UDID) IS QUERIED

As a result, the scenarios described in **Section 4.2** through **Section 4.4** can occur automatically without the user's consent or knowledge. This weakness is a security problem with significant consequences.

3.1.3 APPLICATIONS INSTALLED ON THE CLIENT CAN BE HIDDEN IN THE SAME WAY APPLE HIDES ITS OWN IOS SYSTEM APPLICATIONS (E.G., FIELDTEST.APP)

As a result, the execution (through a weakness described in **Section 3.1.4**) of a hidden application installed by the host will not be visible (e.g., via SpringBoard or the iOS main screen) to the user. This characteristic contradicts the popular assumption that all installed apps are visible and therefore enumerable from the SpringBoard. In addition, it permits malware-like apps to be installed without leaving any traces visible to the user.

3.1.4 THE HOST CAN EXECUTE APPLICATIONS ON THE CLIENT (I.E., DEBUGSERVER) WITHOUT THE USER'S CONSENT

As a result, and in combination with the weaknesses described in the previous sections, the host could mount an Apple-signed disk image (`DeveloperDiskImage.dmg`) and launch `com.apple.debugserver` to execute an installed application regardless of whether it is hidden.

3.1.5 THE USE OF THE APPLE PROVISIONING PORTAL CAN BE EASILY AUTOMATED TO OBTAIN A PROVISIONING PROFILE

As a result, provisioning profiles can be obtained automatically by submitting UDIDs of target devices. Thus, potential attacks do not need to depend on availability of an Enterprise Provisioning Profile, which while imposing no cap on the number of devices, is more difficult to obtain.

3.2 UNIFIED DATA, CONTROL, AND POWER INTERFACE

Due to space and user-convenience considerations, iOS designers have built a unified hardware interface that serves two primary functions:

1. Charging the battery of the iOS device, and
2. Facilitating data communications and device control.

Such choices can be seen in the form of Apple's 30-Pin dock connector (for older devices) and Lightning USB interfaces (for newer devices). This minimalism continues in software – we noticed that there are no visual indicators on the screen when an iOS device is being plugged into a host which can alter the state of a device. These observations motivated us to explore attacks that exploit this absence of information.

4. IOS MALWARE INJECTION ATTACK

Using the observations described in the previous section, we chained together weaknesses and the potential threat vector mentioned in **Section 3.2** to construct an end-to-end malware injection attack for iOS devices. As a proof-of-concept, we successfully injected a malicious app into a target iOS device that was plugged into a fake USB charger; this attack requires neither a jailbroken device nor user interaction. While some users may already be aware that connecting a mobile device to a compromised computer could lead to a compromise of the device, there is usually little concern given when the connection for a mobile device appears to be simply a device charger. As a result, the charger is often trusted implicitly.

With the above context as a guide, we investigated the extent to which commodity USB-based functionalities can be miniaturized and arrived at the idea of integrating a computer into the space profile of a charger, which we later called Mactans. With Mactans, the assumption that chargers are trustable does not hold. Using small financial and time budgets, we were able to build a proof-of-concept charger out of inexpensive, commodity hardware, the BeagleBoard, which is a functional mini-computer on an 8cm x 7.5cm board.

4.1 PROOF-OF-CONCEPT REQUIREMENTS

Below are the requirements for the proof-of-concept attack we describe in this section.

- Apple 30-Pin or Lightning USB cable
- Active iOS individual developer's license
- iOS device (target)
- Internet connection (via Wi-Fi or cellular data connection)
- Mactans charger, consisting of:
 - o A USB port that can provide power
 - o Small scale microprocessor/microcontroller
 - o Linux operating system
 - o iOS RPC communications library (e.g., `libimobiledevice`)

4.2 OBTAINING UDID

A Unique Device Identifier (UDID) is a 40 digit hexadecimal number that serves as a fingerprint of an iOS device. It was originally used by app developers to uniquely identify different devices for various purposes. However, today the UDID is considered a sensitive piece of information and its use in regular apps has been deprecated since iOS 5.0¹. Therefore, unauthorized access to the UDID can be considered a privacy leak.

In our proof-of-concept attack, obtaining the UDID is an essential preliminary step in preparing the target device for app injection. To obtain the UDID, we simply query the device's USB

1

https://developer.apple.com/library/ios/#documentation/uikit/reference/UIDevice_Class/DeprecationAppendix/AppendixADeprecatedAPI.html

identifier using standard tools such as `lsusb`. The UDID can be obtained even if the device is passcode-locked.

4.3 OBTAINING & INSTALLING A PROVISIONING PROFILE

Once obtained, the UDID can be used to create a provisioning profile for the target device, which will allow the injection of attacker-decided applications. A provisioning profile is a cryptographically signed file that contains information about the developer who created the profile as well as UDIDs of devices that can execute apps signed by this developer. To maintain control of the walled garden model, all provisioning profiles are signed by Apple. Without an appropriate provisioning profile, the installation of arbitrary apps will be rejected by iOS on the target device.

Creation of a provisioning profile introduces the requirement of a working internet connection for Mactans. Specifically, the UDID must be submitted to `developer.apple.com` for profile generation. There are at least two ways to fulfill this requirement:

1. Mactans can be equipped with 3G/4G cellular capabilities via a SIM card module. Moreover, there are SIM vendors that provide anonymous cellular activation; Mactans can thus be on-air anonymously. Therefore, Mactans can directly connect to the Apple Provisioning Portal, submit the UDID of a target device, and then obtain a provisioning profile for that device.
2. Mactans can be equipped with Wi-Fi capabilities via various wireless modules. Under this option, connecting to the Internet will be a matter of scanning for unprotected access points, cracking weak access points², brute forcing wireless passwords, or tunneling over DNS.

With Internet connectivity, Mactans can generate a provisioning profile that is unique to the victim device containing the UDID obtained previously. With a provisioning profile in hand, Mactans can trivially install it onto the target device through communication with `com.apple.misagent` (via `lockdownd`).

4.4 INJECT MALICIOUS APP

After the provisioning profile has been installed successfully, Mactans will proceed to inject an arbitrary app into the iOS device. This step is performed via communication with `com.apple.mobile.installation_proxy`. In our proof-of-concept, we demonstrate the significance of a Mactans attack by showing how it can be used to inject a Trojan horse Facebook app; please see the presentation slides that accompany this whitepaper for additional details.

² Aircrack & Reaver

4.5 PAYLOAD

Even though Mactans can inject any app into the target device, another hurdle exists for the payload: app sandboxing. In our proof-of-concept, Mactans does not perform jailbreaking nor does it escape from the sandbox; the injected app has the same privileges as other regular apps (i.e., those of the ‘mobile’ account). However, a Mactans-injected app completely bypasses Apple’s App Store review process. In combination with publicly available information about various private iOS APIs³, attackers can create apps that would otherwise be rejected during the app review process.

As examples of private API abuses, we introduce two proof-of-concept capabilities of a potential payload app. First, as a live background process, such an app can take a screenshot of the current foreground screen by making a private API call. Second, an injected app could generate screen touch events and simulate the hardware button presses by exploiting functionality available in private libraries present in `DeveloperDisk.dmg` after mounting it through communication with `com.apple.mobile_image_mounter`.

³ <https://github.com/nst/iOS-Runtime-Headers>

5. DISCUSSION

5.1 ATTACK SCENARIOS

In this section, we discuss possible scenarios in which an attack could be successful.

5.1.1 GOVERNMENT TARGETED ATTACK

In this scenario, an attacker wishing to target a particular individual could provide a suitably packaged Mactans charger to the target. While this vector requires careful construction of a malicious charger that looks indistinguishable from an original Apple accessory, such an approach by a nation state is well within the realm of plausibility.

Alternatively, a priori knowledge of the target could be leveraged by a resourceful attacker to selectively modify the environment. Examples include installation of a custom, Mactans-like charger in a specific airplane seat or hotel room (e.g., built into a console or desk).

5.1.2 GENERAL ATTACK

In a more general scenario, a Mactans charger can be installed in a public waiting area. One example of such a station is pictured in **Figure 1**. High-traffic areas, such as airports, could result in many hundreds of victims each day.

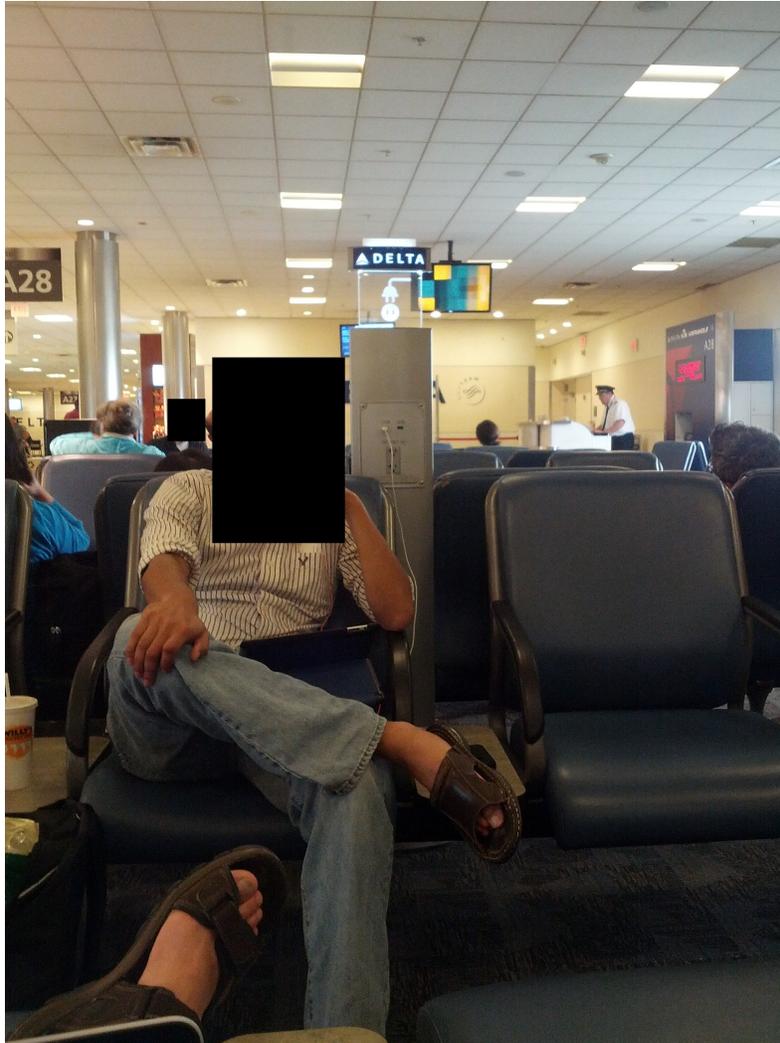


Figure 1: A user replenishing the battery of his iPad at a public charging station while he continues to use it.

5.2 SECURITY CONSEQUENCES

Although we deliberately chose to implement weaker payloads as in our proof-of-concept, it is not inconceivable that adversaries could easily engineer a payload with substantially higher impact. In fact, Mactans may offer a new dimension to the phenomenon of espionage (if not already present) and targeted attacks.

However, the difficulty of attacks can be substantially increased if the weaknesses discussed in **Section 3.1** are addressed by the suggested mitigations proposed in **Section 5.3**. The authors are thankful that after communication with the vendor, an update was released (iOS 7 Beta 2) that is not susceptible to the attack described in **Section 4**.

5.3 MITIGATIONS

Possible mitigations to overcome the weaknesses we describe in **Section 3.1** are listed here. In particular, we believe that any mobile OS should by default inform the user before the state of their device is queried or modified by a USB-connected host. More specifically, we think that it is important to require the user's consent in the following cases:

- 1) Prior to the process of USB device pairing, which enables additional capabilities, including those stated next in 2).
- 2) Prior to installing a provisioning profile or side-loading an application associated with a provisioning profile (as described in **Section 3.1.2**).

A primary possible mitigation is to require the user's consent (e.g., via introduction of a debugging mode setting on the client) in order for the host to launch applications on the client (or perhaps to launch specific applications such as `debugserver`).

In the case of iOS, Apple could also prevent third party developers from setting the `SBAppTags` with the value `hidden` in an app's `Info.plist` so that side-loaded apps cannot be invisible. Furthermore, the process of obtaining provisioning profiles can be made less-automatable by requiring iOS developers to solve a CAPTCHA prior to issuing a profile for a device specified by a UDID.

5.4 LIMITATIONS

If an iOS device is passcode-protected, Mactans requires the phone to be unlocked at least once after being connected. While this requirement may seem to render Mactans impractical, we posit that users will regularly create this situation while charging their device.

Given that our proof-of-concept relies on an individual developer license, a Mactans charger equipped with one individual license can accommodate only 100 devices. However, more resourceful adversaries are likely to have access to an enterprise developer license, which waives this limit. Enterprise license possession also lowers the bar for provisioning profile injection, as a UDID need not be submitted to Apple's Provisioning Portal to generate a provisioning profile.

Diligent, security-minded users may detect attempts to compromise their iOS device if they check installed developer licenses in the Settings section of their device. However, we believe that regular users may not know of the existence or purpose of this information and therefore will not check or understand this setting. Furthermore, even if the provisioning profile is removed, the injected app will continue to run until the device is rebooted. Upon a subsequent connection to a Mactans charger, the attack can be repeated.

6. CONCLUSION

In this paper, we have shown that for iOS devices up to and including those running iOS 6, arbitrary apps can be injected into a user's mobile device when connected to a malicious host. We demonstrated the potential danger of this capability through a proof-of-concept implementation of a malicious charger that injects a Trojan horse app with a payload. The relevance of our work is represented by Apple's release of an update to iOS 7 that implements a mitigation we recommended in our disclosure; **Figure 2** shows a screenshot of iOS 7 when an unknown host tries to communicate with the phone through a USB connection.

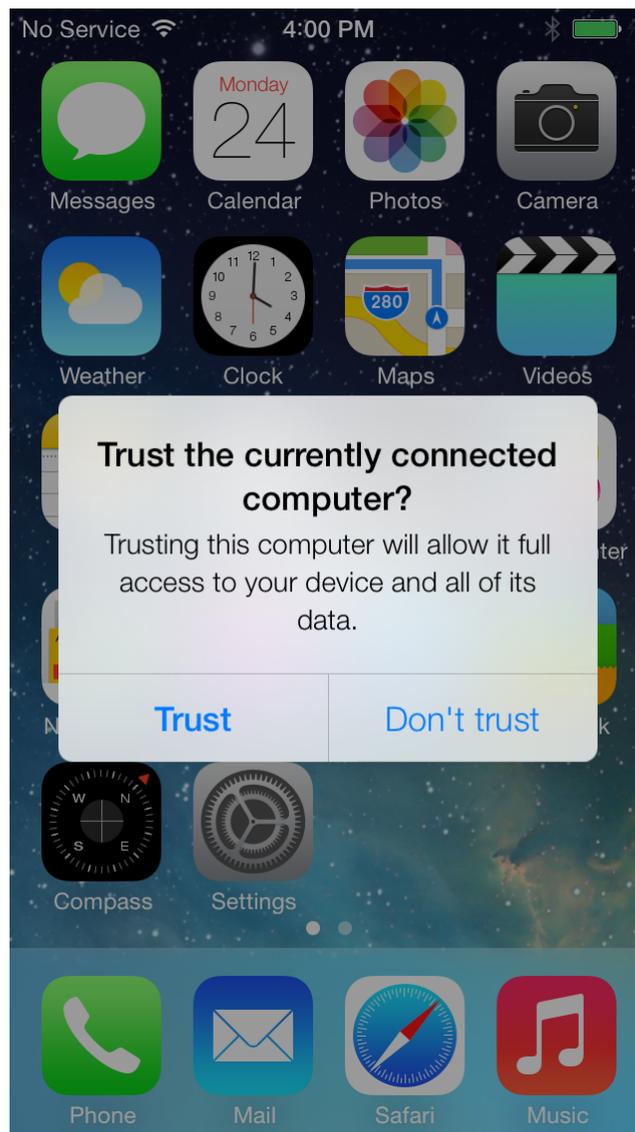


Figure 2: Screenshot of iOS 7 Beta 2 when the device is plugged into an unknown host that tries to pair with the phone.