Data Exfiltration from Internet of Things Devices: iOS Devices as Case Studies

Christian J. D'Orazio, Kim-Kwang Raymond Choo, and Laurence T. Yang

Abstract—**Increasingly, big data (including sensitive and commercial-in-confidence data) is being accessible and stored on a range of Internet of Things (IoT) devices, such as our mobile devices. Therefore, any vulnerability in IoT devices, operating system or software can be exploited by cybercriminals seeking to exfiltrate our data. In this paper, we use iOS devices as case studies and highlight the potential for pairing mode in iOS devices (which allows the establishment of a trusted relationship between an iOS device and a personal computer) to be exploited for covert data exfiltration. In our three case studies, we demonstrate how an attacker could exfiltrate data from a paired iOS device by abusing a library and a command line tool distributed with iTunes. With the aim of avoiding similar attacks in the future, we present two recommendations.**

Index Terms— **Big data security; Internet of Things (IoT) security; iOS data exfiltration; iOS pairing.**

I. Introduction

BIG data exfiltration, the process of gaining unauthorised access to vulnerable systems and leakage of private information, is a threat that is unlikely to go away anytime soon. In a recent research, for example, 522 cyber-security professionals from 1,155 organisations globally were interviewed. These professionals reported that they had experienced at least one large-scale data breach in their working career [27], and insiders were reportedly responsible for 43% of the incidents (21% intentional and 22% accidental) and perpetrators outside the organisation were responsible for the remaining 57%. The security breaches mainly impacted on customer, employee, financial, and payment card information as well as intellectual property. In a security report in 2014 [5], Spanish anti-virus firm Panda Security also reported a significant number of data breaches, such as crypto-ransomware attacks, which affected thousands of the world's biggest organisations and home users.

C. J. D'Orazio is with the School of Information Technology & Mathematical Sciences, University of South Australia, Adelaide, SA 5001, Australia (e-mail: christian.dorazio@mymail.unisa.edu.au).

K.-K. R. Choo [corresponding author] is with (1) Department of Information Systems and Cyber Security, University of Texas at San Antonio, USA, (2) School of Information Technology & Mathematical Sciences, University of South Australia, Australia, and (3) School of Computer Science, China University of Geosciences, Wuhan, China (e-mail: raymond.choo@fulbrightmail.org).

L. T. Yang is with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China; and 5Department of Computer Science, St. Francis Xavier University, Antigonish, NS B2G 2W5, Canada (e-mail: ltyang@stfx.ca).

Traditionally, personal computers and computer networks have been the targets of data theft attacks using conventional tactics, such as rootkits, botnets, spyware, covert channels, and man-in-the-middle (MiTM) attacks [4][14]. However, a wide range of systems, including Internet of Things (IoT) devices, are increasingly targeted as these systems are rich big data sources (e.g. containing personal and sensitive information of individuals, organisations and governments). For example, organisations that support a Bring-Your-Own-Device (BYOD) policy or provide employees with IoT devices could potentially be exposed to a higher risk of data exfiltration if such devices were used to access and store sensitive corporate information. Once a compromised device is targeted and successfully compromised, covert data exfiltration could easily occur.

Security of big data, such as those stored on IoT devices, is an understudied area. For example, cybercriminals can exploit weaknesses in hardware, software level and/or network layer to covertly exfiltrate data stored on vulnerable systems (e.g. cloud and IoT devices). Mobile devices are one common example of IoT devices, and source of big data. A report by Palo Alto Networks [1], for example, indicated that jailbroken devices have been targets of malware (i.e. AdThief, Unfold, Mekie, AppBuyer, Xsser, and WireLurker) designed to steal sensitive user data. The same report reveals that the WireLurker malware can even initiate attacks against non-jailbroken devices by abusing the Apple's iOS Developer Enterprise Program in order to distribute apps that are not subject to review. As a result of a successful attack, data such as device information (i.e. Serial number, Phone number, Wi-Fi address, UDID, etc.), user data (i.e. Apple IDs, users' Email, SMS, IM's logs, etc.), and application usage data may be vulnerable. A study in 2014 [2] also highlighted that malicious apps freely available on the Internet (signed with enterprise certificates and distributed using enterprise provisioning profiles) can steal information from non-jailbroken devices. In a more recent work, researchers identified a new family of iOS malware named "KeyRaider", which has successfully stolen more than 225,000 valid Apple accounts with passwords, certificates, and private keys [3]. It appears that "KeyRaider" was distributed through third-party Cydia repositories, and users from more than 18 countries have been infected.

Apple is known to have a stringent vetting process for apps in the centralised App Store and mandatory code signing to control the distribution of malicious apps. This significantly reduces the number of attack vectors, but there is no foolproof solution. For example, in September 2015, it was reported that "[t]he Xcode development tools used by iOS app makers was copied, modified, and distributed online, by hackers to inject malicious code into apps available on the App Store" [33], which resulted in the infection of 39 popular iOS apps and "potentially impacting hundreds of millions of users" [34].

In this paper, we highlight a potential attack vector – i.e. an attacker is able to abuse the iOS pairing mode and target a third-party system previously trusted by the device user, such as a personal computer. Even in the event that the 'trust' to create new relationships has been revoked (e.g. the pair-lock profile remotion feature is subsequently set to "never"), the attacker will be able to exfiltrate data surreptitiously stored on the iOS device via the compromised (previously) trusted third-party system. We then present a data exfiltration model that allows us to exploit the trusted relationship between a personal computer and an iOS device to collect and transmit user data from the victim device to an attacker. Such a model can be used to systematically scan iOS devices for vulnerabilities against data exfiltration. To demonstrate utility of the data exfiltration model, we develop a prototype consisting of a client application and a server application, and evaluate the prototype in a controlled environment. The prototype could also be deployed in a real-world setting, for example, by distributing the client application to targeted IoT systems using conventional malware dissemination techniques. However, such an activity is criminalised in Australia (see Part 10.7 - Computer Offences of the *Criminal Code Act 1995* (Cth)).

The remainder of the paper is as follows. Section II provides the related background. Section III discusses related work. Section IV outlines our data exfiltration model. Section V describes the experiment setup. In the next two sections, we present and discuss our case study findings, and recommendations for device manufacturers and device users. The last section concludes this paper.

II. Background

A. Pairing Mode

The pairing mode allows the establishment of a trusted relationship which is used to determine whether access to an iOS device is granted from a computer. Introduced in iOS 7 and later, this feature provides users with the capability to control access to their personal data. The user needs to set a passcode, so that the iOS device has to be unlocked before accepting connections from new or 'untrusted' computers.

To achieve a higher level of security on devices running iOS 8 and later, pair-lock can be activated to prevent any party (including a law enforcement forensic investigator) from creating new trusted relationships with an iOS device even when the device is unlocked or does not have a passcode. Apple Configurator or iPhone Configuration Utility can also be used to generate and install a pairing profile that cannot be removed. If the pair-lock profile remotion feature is set to "never", the device will not be able to establish a connection to other computers unless the device is erased or a previous copy of the pairing record stored on the computer can be recovered.

However, both the pairing and pair-lock features could be exploited to covertly exfiltrate user data from (previously) trusted third-party systems, as we demonstrate in this paper.

B. File System Access

Apple iOS devices, such as iPhones and iPads, run a special service called Apple File Conduit (AFC), which is used by iTunes to access the file system over USB connections. Natively, this service is 'jailed' so that access is limited to the Media folder containing photos, videos, music, downloads, and apps data. In order to provide unrestricted access to the entire file system structure, a second service called AFC2 would be installed during the jailbreaking of an iOS device. However, due to security concerns, modern jailbreaks, such as TaiG, do not install AFC2 by default. Hence, a device that has been jailbroken to run pirated apps or apps not signed by Apple (e.g. apps distributed on the Cydia repository) might still keep a non-jailbroken file system with access restricted to the Media folder. However, users can choose to install AFC2 from Cydia in order to gain complete access to other user data not stored on the Media folder (e.g. Addressbook, Emails, SMS, Notes, Calendar, and Call history) using unofficial tools, such as DiskAid, iFunBox, and iPhoneBrowser.

In this paper, we demonstrate that data in the Media folder is fully accessible, but the amount of other sensitive user data exfiltrated is partly dependent on the file system jailbreak status.

C. iTunes Backup

Apple's iTunes is an application for iOS mobile device management. One of its features is data backup which allows users to store important data on a personal computer (or iCloud) for eventual restoration. When a backup is performed, sensitive data is saved into several files within a backup folder. On Windows platforms, the folder is located in "C:\Users\[USER]\AppData\Roaming\Apple Computer\MobileSync\Backup\[UDID]", and the folder name is the device UDID, a unique identifier specific to the iOS device.

By default, backups are not encrypted and backup data includes media and databases, such as Addressbook, SMS, Notes, Calendar, and Call history. In our research, we determined that Emails are not included in backups, and in the backup folder, each filename is encoded using a SHA1 hash function of the actual home domain (followed by a dash), path and filename on the iOS device (see Table I).

In Section VI.B.2, we demonstrate that a non-jailbroken device, for which a backup password has not been set, could be exploited to exfiltrate sensitive data stored outside the Media folder.

D. Passcode and Data Encryption

Designed by Apple [6], File Data Protection is an additional security layer based on a hierarchical encrypted file system in which each file is assigned to a class, depending on the type of

TABLE I
MAPPING OF DATABASES AND THEIR NAMES IN AN IOS BACKUP

Content	SHA1(Domain + '-' + Path + Filename on device)	Filename backup
Addressbook	SHA1('HomeDomain-Library/AddressBook/AddressBook.sqlitedb')	31bb7ba8914766d4ba40d6dfb6113c8b614be442
SMS	SHA1('HomeDomain-Library/SMS/sms.db')	3d0d7e5fb2ce288813306e4d4636395e047a3d28
Notes	SHA1('HomeDomain-Library/Notes/Notes.sqlite')	ca3bc056d4da0bbf88b5fb3be254f3b7147e639c
Calendar	SHA1('HomeDomain-Library/Calendar/Calendar.sqlitedb')	2041457d5fe04d39d0ab481178355df6781e6858
Call history	SHA1('WirelessDomain-Library/CallHistory/call_history.db')	2b2b0084a1bc3a5ac8c27afdf14afb42c61a19ca

data and security levels required. Every time a file is created in the volume, File Data Protection generates a new 256-bit key (the per-file key) which is used by the cryptographic engine to encrypt the file when it is written to the flash memory using AES CBC mode. Based on RFC3394 specifications (see http://www.ietf.org/rfc/rfc3394.txt), the per-file key is then wrapped with the key of a security class which the file belongs to. There are four protection classes that use different policies to determine when and how file contents are accessible (e.g. file encryption might depend on the user passcode). Table II shows the protection classes responsible for securing the data we attempted to exfiltrate in this paper. As outlined in the "Class key Availability" column of Table II, the device lock status determines the class key availability.

TABLE II
ASSOCIATION OF PROTECTION CLASSES AND THE USER DATA EXFILTRATED

Class name	Class key availability	User data exfiltrated in our case studies
NSFileProtectionComplete	When unlocked	Emails
NSFileProtectionCompleteUnlessOpen	While locked	NA
NSFileProtectionCompleteUntilFirstUserAuthentication	After first unlock	AddressBook, SMS, Notes, Calendar, Call history, Media files
NSFileProtectionNone	Always	NA

NA: Not Applicable to data exfiltrated in our case studies.

III. RELATED WORK

Existing academic literature (see [15][16][17][18][19][20][31]) and security technical reports (see [28][29][30]) generally focus on data exfiltration or leakage detection strategies rather than demonstrating how data exfiltration can occur, particularly in a practical setting. One reason data exfiltration may not have been thoroughly explored is, perhaps, due to the rate of hardware and software advancements and the challenge in proposing a general approach to uncover security flaws that could be exploited for data exfiltration.

Farley and Wang [13] presented a surveillance intrusion method using a microphone hijacker for Windows in order to stream live microphone data to a remote attacker. The attacker uses a botnet framework to exploit spyware-infected computers connected to the Internet. The authors also extended their research to the mitigation of the threat by implementing detection mechanisms that hook into WinAPI function calls, which are generally used in microphone recording programs. In another recent research [21], Do et al. proposed the first adversary model for Android covert data exfiltration that allows an Android device to encode and transmit data inaudibly via the device speaker. Using another Android device, an attacker receives and decodes the inaudible audio transmission.

A small number of existing studies also focus on using the audio sources as a data exfiltration medium. A team of researchers from MIT, Microsoft Research, and Adobe Research [22], for example, demonstrated that small vibrations of an object's surface resulted from sound hitting the object can be extracted using high-speed cameras. They were able to recover audio samples by visualising the fluctuation of the sound-related vibrations. In another independent, yet related, work, researchers from Fraunhofer FKIE [23] implemented a covert acoustical mesh network that utilises audio modulation/demodulation to transmit data between computer systems over the air. It was demonstrated that two computers that are not connected using typical network interfaces (e.g. IEEE 802.3 Ethernet or IEEE 802.11 WLAN) are able to transmit data between each other using their audio input and output devices (microphones and speakers). Similarly, O'Malley and Choo [25] presented a method that uses audio input and output capabilities of laptop computers to transmit inaudible sounds from the speakers of an air-gapped system to a nearby computer equipped with a microphone. The method could be used, for example, by an insider to install malicious software into corporate systems. A security researcher [24] describes how NSA's implantable devices (also known as 'retro reflectors') operate to surreptitiously listen in on ambient sounds and collect keystrokes and on-screen images. Once attached to a personal computer, the reflector is controlled using radio systems to emit a high-power radar signal to wirelessly transmit the exfiltrated data to an attacker.

Researchers have also investigated the iOS platform and demonstrated how data can be covertly exfiltrated from iOS devices. Xing et al. [9] identified a number of vulnerabilities in mobile applications (apps) that could be exploited for data exfiltration on iOS devices and OS X computers. More specifically, sandboxed malicious apps, approved by Apple for distribution in the App Store, could gain unauthorised cross-app resource access (XARA) by exploiting inter-app interaction services in order to steal highly-sensitive information, such as the passwords for the iCloud. The study also discloses that the Android platform is affected by similar vulnerabilities. Caviglione and Mazurczyk [10] demonstrated that Siri—a native iOS service that controls iPhones and iPads via voice commands—could be exploited using an information-hiding method that surreptitiously leaks data from iOS systems. The method abuses the Siri traffic to embed secrets, which could be used by a botmaster to extract sensitive data from an infected device. Agarwal and Hall [11] designed a system capable of detecting access to private data at runtime by apps on iOS devices. After analysing 225,685 apps, it was discovered that 69.4% of these apps accessed user private data. With the aim of making privacy recommendations to users, the system also collects and analyses user protection decisions.

In our previous work, we proposed an adversary model that allows users to reveal vulnerabilities in the apps which may be exploited to circumvent digital rights management (DRM) protections and obtain unauthorised access to exfiltrate copyrighted content. To demonstrate the utility of the adversary model, we revealed previously unpublished vulnerabilities in four popular VoD apps, one live TV app, and a widely used security DRM protection module [35].

Zdziarski [26] identifies iOS services and surveillance mechanisms that could be exploited by malicious parties to conduct espionage. The study also highlights the importance of pairing records stored on trusted computers, which is key to initiating any iOS service in order to transfer user data from a connected iOS device. Porras et al. [12] reverse engineered the iKee.B iPhone Botnet and discovered that the malware infects jailbroken iOS devices and propagates by scanning specific Internet IP address ranges for the SSH service. When the SSH service responds, the malware attempts to connect as root using the default password ‘alpine’. If a vulnerable device is found, the malware copies and unpacks a payload to the victim’s device. In addition to the use of scanning and propagation subroutines, it was reported that iKee.B redirected users to phishing sites in order to steal user sensitive data, such as bank account information.

A recent review of adversary models by Do et al. [38] found that such models are typically used in designing and validating cryptographic and secure systems, and increasingly popular in Android security literature. However, no adversary model for iOS security literature was presented.

In addition, our review also suggests that little has been done to understand how a trusted relationship between an iOS device and a personal computer can be abused to gain unauthorised access to sensitive and private user data stored on iOS devices. Such understanding would have assisted future design and implementation of hardened security solutions, as well as alerting users to the dangers of unsafe practices like jailbreaking. This is the literature gap we attempt to address. More specifically, in this paper, we develop a data exfiltration model comprising three viable attack methods to exploit the file system jailbreak status and backup password settings. This would provide Apple and iOS users a systematic way of scanning their devices for vulnerabilities against data exfiltration.

IV. Data Exfiltration Model (and Prototype)

Our model, based on a client-server TCP/IP architecture, allows an attacker to conduct covert data exfiltration from iOS devices connected to trusted computers over USB. In this model, the client application resides on target computers to interact with connected iOS devices. The server application runs on a remote computer controlled by the attacker. To facilitate data exfiltration, the server application creates a socket for each client requesting connection.

The client application accepts connections from multiple devices connected to the same computer, and the server application handles connection requests from multiple computers and allows for data exfiltration from several devices simultaneously. In order to support this architecture, both the client and server concurrently execute independent process threads (i.e. sequences of computer program instructions that can be executed simultaneously by a unique process running on a single processor). More specifically, a client runs as many threads as the number of devices connected to the target computer, and the server runs as many threads as the total number of devices connected among all the active clients. This is an improvement to previous data exfiltration attempts, such as those of Do, Martini and Choo [21] and O’Malley and Choo [25], which only allows a one-to-one connection.

Once a device is connected, the client application is notified of the event and start requesting for device details (see Section VI.A). The latter includes iOS Device Information (see Fig. 2) and the list of media files. If the device is paired with the computer running the client application, a data exfiltration attack will then be attempted. In the attack, the client application sends a message to the server application, which is actively listening for incoming connections.

To secure the communication between the client and server applications, both applications implement RSA public-key cryptography to exchange a session key to be used in a pre-established AES-128 symmetric cipher. More specifically, once the connection between the client and server applications is established, the server application requests that the client application generates a random AES key to be used to encrypt data-in-transit during the session. Subsequently, the client application encrypts the device details with this key, which is also stored for later use during data exfiltration. Then, the AES key is encrypted with the server’s public key and sent to the server. The server uses the private key to decrypt the AES key and then requests the device details. Upon receiving the information, the server decrypts the device details with the AES key and displays the exfiltrated data (e.g. media file list, file system jailbreak status and backup password settings of the target iOS device) to the attacker.

The session (AES) key, which is needed for the exchange of device details, is also used by the client and server applications to encrypt and decrypt exfiltrated user data obtained by communicating with the iOS Services during the attacks, respectively. Fig. 1 illustrates a detailed cross-functional flowchart of our data exfiltration model.

A. The Client application

For successful data exfiltration, it is necessary for the client

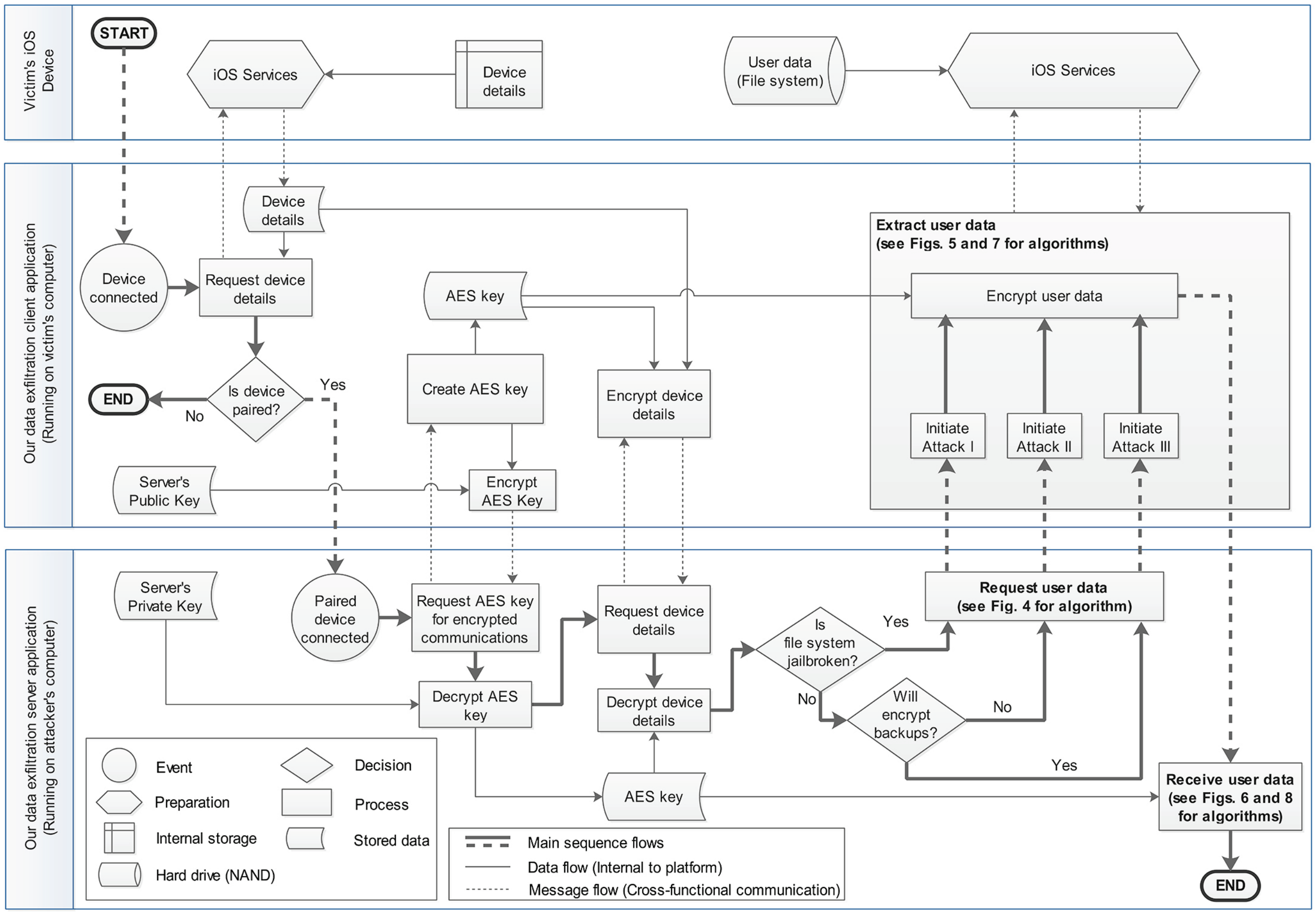


Fig. 1. Data exfiltration model

application to monitor events occurring on the target computer, such as the connection and disconnection of iOS devices to and from USB ports. In addition to receiving notifications of these events, there must exist a means of communicating with connected devices in order to exfiltrate user data, for example, by implementing native communication, and/or communicating via iTunes libraries.

In the former approach, one would need to develop complete software libraries that implement the protocols supported by iOS devices. In other words, to implement native communication, we would need to create a proprietary set of libraries or reuse existing projects, such as *libimobiledevice* (an open source cross-platform software library to communicate with iOS devices). The *libimobiledevice* project is aimed at supporting the Linux community, and therefore, redesigning *libimobiledevice* to run on Windows platform is outside the scope of this paper.

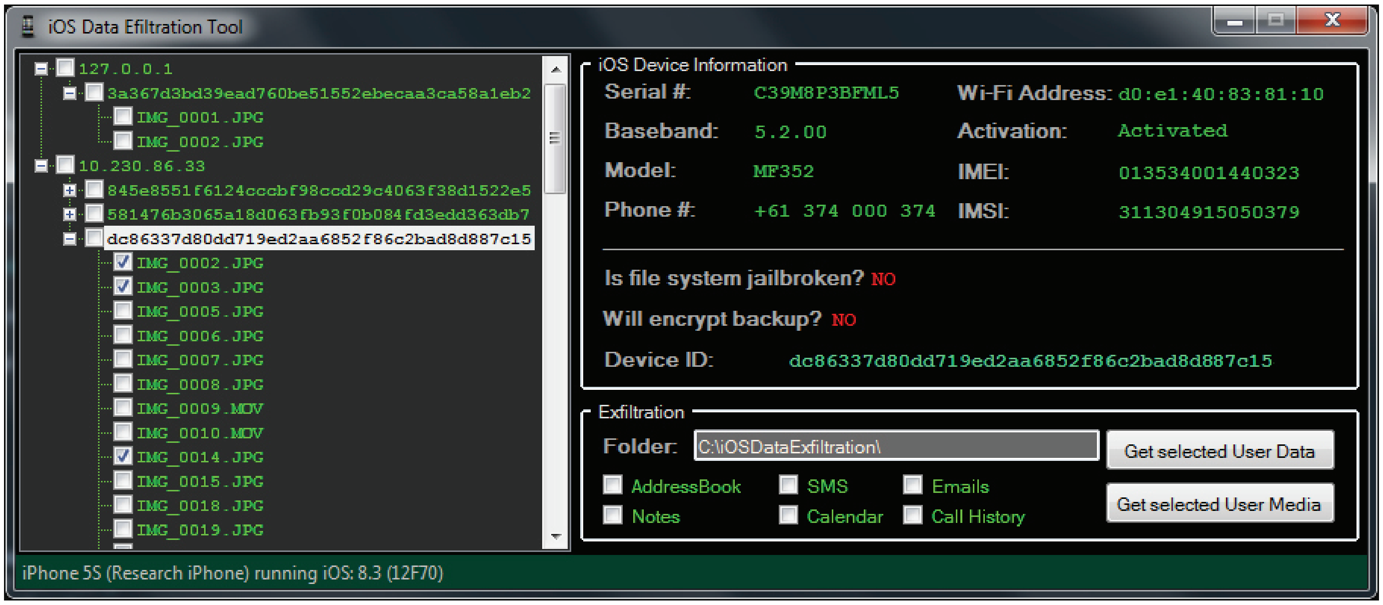
In this paper, our client application adopts the second approach (i.e. exfiltrating data via iTunes libraries), which involves the use of *iTunesMobileDevice.dll*, an existing library distributed with iTunes for Windows. The client application is also designed to run as a background process on the target computer.

B. The Server Application

Once the client application detects that a paired iOS device has connected, a message will be sent to the server. Incoming connections will be accepted, and device information will be requested by the server application. The server application is used for remote data exfiltration and has a graphical user interface (GUI) that comprises a Device Media Treeview, an iOS Device Information Panel, and an Exfiltration Panel (see Fig. 2).

Device Media Treeview: This is the view on the left of the GUI that uses a tree structure to display a list of media files stored on the target devices. The media files are grouped by hosts (the target computers identified by IP addresses; e.g. 10.230.86.33) and target devices (identified by the Device IDs; e.g. dc86337d80dd719ed2aa6852f8c2bad8d887c15). The media files can be individually selected for exfiltration.

iOS Device Information Panel: The panel displays basic device information that could be of potential interest for the attacker, such as Serial number, Baseband, Model, Phone number, Wi-Fi address, Activation status, IMEI, and IMSI. For example, the attacker who has successfully exfiltrated compromising materials (e.g. child abuse images / videos or images / videos that could embarrass the individual) could also use the phone number obtained to contact the victim for



```
iOS Data Efiltration Tool

127.0.0.1
  3a367d3bd39ead760be51552ebecaa3ca58a1eb2
    IMG_0001.JPG
    IMG_0002.JPG
10.230.86.33
  845e8551f6124cccbf98ccd29c4063f38d1522e5
  581476b3065a18d063fb93f0b084fd3edd363db7
  dc86337d80dd719ed2aa6852f86c2bad8d887c15
    [✓] IMG_0002.JPG
    [✓] IMG_0003.JPG
    IMG_0005.JPG
    IMG_0006.JPG
    IMG_0007.JPG
    IMG_0008.JPG
    IMG_0009.MOV
    IMG_0010.MOV
    [✓] IMG_0014.JPG
    IMG_0015.JPG
    IMG_0018.JPG
    IMG_0019.JPG

iOS Device Information
Serial #:    C39M8P3BFML5          Wi-Fi Address: d0:e1:40:83:81:10
Baseband:    5.2.00                Activation:    Activated
Model:       MF352                 IMEI:          013534001440323
Phone #:     +61 374 000 374       IMSI:          311304915050379

Is file system jailbroken? NO
Will encrypt backup? NO
Device ID:   dc86337d80dd719ed2aa6852f86c2bad8d887c15

Exfiltration
Folder: C:\iOSDataExfiltration\        [Get selected User Data]
AddressBook   SMS        Emails
Notes         Calendar   Call History  [Get selected User Media]

iPhone 5S (Research iPhone) running iOS: 8.3 (12F70)
```

Fig. 2. Server application GUI

extortion. Due to privacy and security concerns, the device information displayed in Fig. 2 has been modified. In addition, this panel includes the Device ID as well as vital information regarding the file system jailbreak status and backup password settings, respectively identified with the labels ‘Is file system jailbroken?’ and ‘Will encrypt backup?’. Such information can be used to determine which and how data can be successfully exfiltrated.

Exfiltration Panel: This panel allows the attacker to set the folder name where exfiltrated data will be stored by the server and to select, in addition to media files, other user data to be exfiltrated (e.g. AddressBook, Email, SMS, Calendar, Notes, and Call History).

V. Experiment Setup

In our experiment, we used *TaiG* (*http://www.taig.com/en/*), the only jailbreak program supporting the iOS versions on our devices at the time of this research, to jailbreak two of the six iPhones 5S (see D2 and D3 in Table III). The iPhone 4 (i.e. D5) has previously been jailbroken using *evasi0n* (*www.evasi0n.com*).

In addition, we used four computers connected to the University network to play either the role of an attacker or the role of a target (see Table IV).

We installed the latest iTunes version at the time of this research (i.e. iTunes 12.1.2.27) on the three target computers. The iOS devices were then connected to each of the target computers in order to initiate pairing. Devices running iOS 7 and later versions required us to manually confirm the trusted relationship (similarly to a real-world usage when a user first uses iTunes on a new / untrusted computer). Devices running iOS 6 paired automatically without requesting user confirmation. In addition, all target computers had anti-malware software with up-to-date signature and complied with the IT security rules of our University (e.g. these computers also had Microsoft Endpoint Protection and Windows Firewall installed and running).

TABLE III
Research iOS Device

Device#	Device model	iOS version
D1	iPhone 5S (Non-Jailbroken)	8.3
D2	iPhone 5S (Jailbroken)	8.3
D3	iPhone 5S (Jailbroken)	8.1.2
D4	iPhone 4S (Non-Jailbroken)	7.1
D5	iPhone 4 (Jailbroken)	6.1.2
D6	iPhone 3G (Non-Jailbroken)	6.1.2

TABLE IV
Research Computers

Computer#	Hardware/OS	Role
C1	PC 1 (Windows 7, 64-bit)	Target
C2	PC 2 (Windows 7, 64-bit)	Target
C3	Laptop (Windows 8, 64-bit)	Target
C4	PC 3 (Windows 7, 64-bit)	Attacker

Similar to the approach of O’Malley and Choo [25], we do not consider dissemination and installation of the client application to be an issue. Therefore, in our experiment, we manually set up and run the client process on the target computers. In addition, we used a configuration file so that the client application knows the connection parameters of the server (the IP address of the attacker’s computer and the port number to establish connection). In a real-world deployment, these parameters would be obfuscated by the attacker.

An example implementation of our data exfiltration model is illustrated in Fig. 3, and the prototype is designed for Microsoft Windows, the most popular desktop/laptop operating system [7][8].

VI. Findings

In Section VI.A, we describe how the client application communicates with devices via the iOS Services. In Section

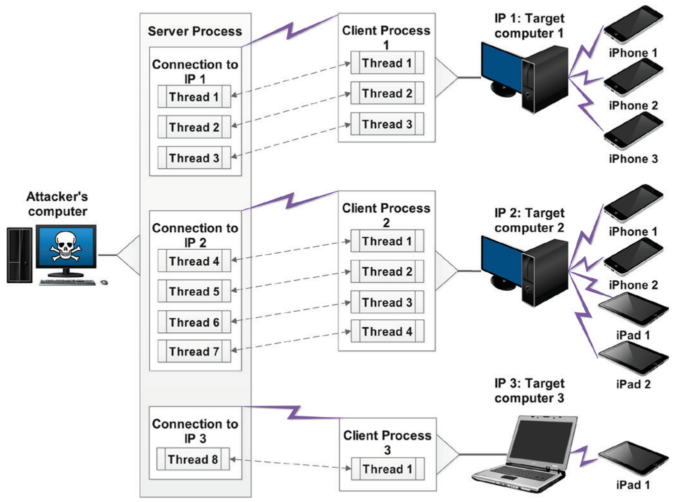


Fig. 3. An example implementation of our data exfiltration model

VI.B, we demonstrate how the data exfiltration can be carried out.

A. iOS Services

In order to communicate with iOS services, our client application imports a set of methods available in the *iTunesMobileDevice.dll* library. Initially, the client application calls *AMDeviceNotificationSubscribe* to subscribe to the notification of events indicating when an iOS device has connected to the target computer. Then, the client establishes a connection with the device, by sequentially calling the following methods: *AMDeviceConnect*, *AMDeviceIsPaired*, *AMDeviceStartSession*, *AMDeviceStartService*, and *AFCConnectionOpen*. If no error occurs, the connection is successfully established and the client is ready to query the device details as well as requesting and receiving data from the iOS device using the following methods: *AMDeviceCopyValue_Int*, *AFCFileRefOpen*, *AFCFileRefRead*, *AFCKeyValueRead*, and *AFCDirectoryRead.*

B. The Attacks

In this section, we demonstrate three types of data exfiltration attacks, namely: Attack I, II, and III, which are based on the file system jailbreak status and backup password settings of the target device. The connections between the devices listed in Table III are as follows:

- Connection 1: Devices D1, D4, and D6 connected to computer C1;
- Connection 2: Devices D2 and D5 connected to computer C2;
- Connection 3: Device D3 connected to computer C3.

The above configuration was designed to demonstrate that the server application is able to handle simultaneous connections, and different connection arrangements of the devices and computers would not impact on the data exfiltration outcome. In our experiments, we attempted to exfiltrate the following data:

- Media files (only images and videos for simplicity), and
- Important database files, namely: AddressBook, Email, SMS, Calendar, Notes, and Call history.

1) *Attack I: Jailbroken file system*

To investigate data exfiltration from jailbroken devices, we connected devices D2 and D5 to computer C2 and device D3 to computer C3. In requesting device details, the client application calls the *AMDeviceStartService* method in order to start the AFC2 service, namely *com.apple.afc2*. A successfully initiation of AFC2 indicates that the target device is jailbroken; therefore, access to its entire file system is granted via AFC2.

In addition, the client application creates a list of media files stored on the device by calling *AFCDirectoryRead* and sends the list to the server application, together with the file system jailbreak status. If the device is jailbroken, the attacker would know that any file (media or database file) can be directly read by calling *AFCKeyValueRead*, *AFCFileRefOpen*, and *AFCFileRefRead.* In such an event, the attacker would be able to select important databases that contain sensitive data (e.g., AddressBook, Emails, SMS, Notes, Calendar, and Call history) in addition to media files of interest. Then, the attacker sends the exfiltration request to the client. Next, the client reads and encrypts the requested files with the session (AES) key and sends them to the server. Using the same key, the server decrypts the received data and creates the files locally on the attacker's computer. Table V depicts the

TABLE V
ALGORITHMS USED IN ATTACK I TO REQUEST, EXTRACT, AND RECEIVE USER DATA

Process name (See Fig. 1)	Algorithm
Request user data	Algorithm 1 (see Fig. 4)
Extract user data	Algorithm 2 (see Fig. 5)
Receive user data	Algorithm 3 (see Fig. 6)

```
ALGORITHM 1 (Executed on the server)
/* REQUEST USER DATA */
media_exfiltration_list = Select_media_files_( device_details );
db_exfiltration_list = Select_database_files_( device_details );
Send_to_client_( media_exfiltration_list );
Send_to_client_( db_exfiltration_list );
```

Fig. 4. Algorithm 1 (Process to request user data in Attacks I, II, and III)

```
ALGORITHM 2 (Executed on the client)
/* EXTRACT USER DATA */
media_exfiltration_list = Get_server_media_exfiltration_request_();
db_exfiltration_list = Get_server_db_exfiltration_request_();
for each ( file_element in ( media_exfiltration_list || db_exfiltration_list ) ) {
    Read_content_from_device_file_system_via_AFC2_( file_element, buffer );
    Encrypt_( buffer, session_AES_key );
    Send_to_server_( buffer );
    // Wait for server response to exfiltrate next file
    Wait_for_server_ACK_();
}
```

Fig. 5. Algorithm 2 (Process to extract user data in Attack I)

```
ALGORITHM 3 (Executed on the server)

/* RECEIVE USER DATA */
for each ( file_element in ( media_exfiltration_list || db_exfiltration_list ) ) {
    Wait_for_exfiltrated_data_();
    Read_ exfiltrated_data _( buffer );
    Decrypt_( buffer, session_AES_key );
    Write_file_( file_element, buffer );
    Send_to_client_( ACK );
}
```

Fig. 6. Algorithm 3 (Process to receive user data in Attacks I and II)

algorithms used in Attack I.

In summary, we were able to exfiltrate media files and the important databases from the three devices used in this case study (i.e. D2, D3, and D5). Since data protected by the *NSFileProtectionComplete* class is secured when the device is locked, it was necessary to unlock the target devices to read the email database. Other databases, as well as media files, could be read even though the devices were locked.

With non-jailbroken devices, data has to be exfiltrated using either Attack II or Attack III, respectively described in Sections VI.B.2 and VI.B.3.

2) Attack II: Non-jailbroken file system and no backup password set

In this attack, we explain how data can be exfiltrated from non-jailbroken devices which do not have a backup password created. We connected devices D1, D4, and D6 to computer C1.

To determine the jailbreak status of a device, the client application first attempts to start the AFC2 service, as with Attack I. If AFC2 cannot be initiated but the client successfully initiates AFC, namely *com.apple.afc*, one could deduce that the file system is non-jailbroken. As AFC2 cannot be initiated, the client application is not able to access and read the important databases via the same means used in Attack I (i.e. by calling *AFCKeyValueRead*, *AFCFileRefOpen*, and *AFCFileRefRead*). Media files, however, can be trivially exfiltrated using the same process described in Section VI.B.1 (Attack I) via AFC service rather than AFC2.

To exfiltrate data from important databases, the client application exploits the lack of a password to encrypt device backups. We remark that this attack does not exfiltrate databases from existing backups on the target computer. Instead, a new backup is surreptitiously initiated and stored in a temporary folder on the target computer for subsequent data exfiltration. By communicating with the iOS Services, the client application reads the device details including the Device ID and the value of the "*WillEncrypt*" key from the "*com.apple.mobile.backup*" domain. To read "*WillEncrypt*", the client application calls *AMDeviceCopyValue_Int* indicating the aforementioned key and domain names. The call returns a pointer to a Boolean value that is *false* if the backup password is not set. Otherwise, a pointer to a Boolean value *true* will be returned.

Since the backup password is not set, the client application is able to use the *AppleMobileBackup.exe* command line tool distributed with iTunes and exfiltrate data from the important databases. More specifically, when the attacker requests data from any of the important databases, the client application launches *AppleMobileBackup.exe* with the following parameters:

- -b, which indicates to perform a new backup;
- --target *deviceid*, which indicates the Device ID of the target device to make a backup of;
- --root *foldername*, which indicates the temporary folder to store the backup data in the form described in Section II.C.

Since the client application launches *AppleMobileBackup.exe* in the background mode, there will no pop-up window. Therefore, the user will have no idea that a new process is being executed. After the initiation of the backup, the client application performs periodic checks with intervals of three seconds to verify whether all requested databases have been read and saved in the temporary folder. Once all the requested databases have been extracted and stored in the temporary folder, the backup is interrupted by the client application by killing the *AppleMobileBackup.exe* process. The client application also deletes the backup folder to remove any trace of data exfiltration. In our experiments, we did not detect any attempt—neither from iTunes, nor from the device—to complete or restart the interrupted backup when the device was subsequently connected to the victim's computer. This reduces the chance of detection. The databases are then read from the temporary folder and sent to the server application. Table VI depicts the algorithms used in Attack II.

TABLE VI
ALGORITHMS USED IN ATTACK II TO REQUEST, EXTRACT, AND RECEIVE USER DATA

Process name (See Fig. 1)	Algorithm
Request user data	Algorithm 1 (see Fig. 4)
Extract user data	Algorithm 4 (see Fig. 7)
Receive user data	Algorithm 3 (see Fig. 6)

TABLE VII
COMPARATIVE SUMMARY OF PERFORMING A COMPLETE BACKUP (TEST I) AND AN INTERRUPTED BACKUP (TEST II)

Device#	Test I (Complete Backup)		Test II (Interrupted Backup)	
	Data size[a]	Time	Data size[b]	Time
D1	1.31 GB	81 sec.	15.4 MB	10 sec.
D4	1.77 GB	95 sec.	9.69 MB	10 sec.
D6	592 MB	92 sec.	11.66 MB	11 sec.

a. Size of extracted data when the complete backup ended.

b. Size of extracted data when the backup was interrupted, just after reading all the databases requested for exfiltration.

We observed that the interruption of the backup allowed us to expedite the data exfiltration process significantly. To demonstrate the speed of this method, we performed two separate tests on the devices used for this attack (devices D1, D4, and D6). The first test consisted of measuring the time required for each device to perform a complete backup, and the second test consisted of measuring the time for all the requested databases to be extracted and the backup interrupted – see Table VII.

```
ALGORITHM 4 (Executed on the client)
/* EXTRACT USER DATA (MEDIA FILES) */
media_exfiltration_list = Get_server_media_exfiltration_request_();
db_exfiltration_list = Get_server_db_exfiltration_request_();
for each ( file_element in media_exfiltration_list ) {
    Read_content_from_device_file_system_via_AFC_( file_element, buffer );
    Encrypt_( buffer, session_AES_key );
    Send_to_server_( buffer );
    // Wait for server response to process next file
    Wait_for_server_ACK_();
}
/* EXTRACT USER DATA (DATABASE FILES) */
If ( db_exfiltration_list. Empty_() == FALSE ) {
    // Initiate backup
    Run_( "AppleMobileBackup.exe", DEVICE_ID, BACKUP_FOLDER );
    // Wait until all requested files are extracted
    do {
        Wait_( TIME_OUT_INTERVAL );
    } until_( All_files_in_( db_exfiltration_list ).AreExtracted_() == TRUE );
    // Interrupt backup */
    Kill_( "AppleMobileBackup.exe" );
    // Send data to server
    for each ( file_element in db_exfiltration_list )    {
        Read_content_from_backup_( file_element, buffer );
        Encrypt_( buffer, session_AES_key );
        Send_to_server_( buffer );
        // Wait for server response to process next file
        Wait_for_server_ACK_();
    }
}
```

Fig. 7. Algorithm 4 (Process to extract user data in Attacks II and III)

We found that the complete backups contained all media files and important databases as well as data that was not subject to our exfiltration attempts. On the other hand, the interrupted backups contained the requested databases and several files not subject to our exfiltration attempts, but not media files. Therefore, it appears that we could exfiltrate important databases from a device in approximately 10 seconds.

We also remark that users who create unencrypted backups are at risk of data exfiltration including Addressbook, SMS, Notes, Calendar, and Call history. The email database is not vulnerable to this attack as it is not included in backups.

3) Attack III: Non-jailbroken file system and backup password set

To explain how data can be exfiltrated from a non-jailbroken file system which has a backup password, we connected devices D1, D4, and D6 to computer C1. The setting is similar to Attack II, except that in this attack, we used iTunes to set a backup password prior to attempting the data exfiltration.

In this attack, we attempted to obtain files found in the media folder, which can directly be accessed through AFC regardless of the device lock status. In such a scenario, the media files can be read by calling *AFCKeyValueRead*, *AFCFileRefOpen*, and *AFCFileRefRead,* similar to Attack II described in Section VI.B.2.

To exfiltrate data from the important databases, the attacker would first decrypt the exfiltrated databases with the AES-128 key used to encrypt the session between client and server. Then, the attacker would perform an offline dictionary or brute-force attack against the databases in order to discover the backup password. The offline dictionary or brute-force attack can be undertaken at the server using tools such as a forensic kit like Elcomsoft Phone Breaker. Table VIII depicts the algorithms used in Attack III.

C. Summary of Findings

TABLE VIII
ALGORITHMS USED IN ATTACK III TO REQUEST, EXTRACT, AND RECEIVE USER DATA

Process name (See Fig. 1)	Algorithm
Request user data	Algorithm 1 (see Fig. 4)
Extract user data	Algorithm 4 (see Fig. 7)
Receive user data	Algorithm 5 (see Fig. 8)

```
ALGORITHM 5 (Executed on the server)
/* RECEIVE USER DATA */
for each ( file_element in ( media_exfiltration_list  || db_exfiltration_list )) {
    Wait_for_exfiltrated_data_();
    Read_ exfiltrated_data _( buffer );
    Decrypt_( buffer, session_AES_key );
    Write_file_( file_element, buffer );
    Send_to_client_( ACK );
}
/* BRUTE FORCE BACKUP PASSWORD to DECRYPT DATABASE FILES */
if ( db_exfiltration_list.IsEmpty_() == FALSE ) {
    backup_password := brute_force_backup_password_( db_exfiltration_list );
    for each ( file_element in  db_exfiltration_list  ) {
        Read_file_( file_element, buffer );
        if ( backup_password != NULL ) {
            Decrypt_( buffer, backup_password );
        }
        Write_file_( file_element, buffer );
    }
}
```

Fig. 8. Algorithm 5 (Process to receive user data in Attack III)

In summary, we demonstrated that data from both jailbroken and non-jailbroken devices can be covertly exfiltrated from trusted computers, and our attacks were not detected by the security measures mandated by the University's IT security policy (e.g. not detected by anti-malware software, Microsoft Endpoint Protection and Windows Firewall).

Our findings, summarised in Table IX, demonstrate that a jailbroken file system is potentially a critical vulnerability that can be exploited by an attacker to exfiltrate different types of data, including the email database if the device is unlocked. In the event that the file system is not jailbroken, the attacker can remotely initiate a backup to obtain the AddressBook, SMS, Notes, Calendar, and Call history databases. This attack

TABLE IX
SUMMARY OF FINDINGS

User data	Jailbroken file system (Access to files via AFC2)		Non-Jailbroken file system (Access to files via backup)		Non-Jailbroken file system (Access to files via AFC)	
	Attack I		Attack II	Attack III	Attack II/III	
	Passcode locked	Unlocked	No backup password set	Backup password set	Passcode locked	Unlocked
Media (photo/video files)	✓	✓	NA[a]	NA[a]	✓	✓
Addressbook	✓	✓	✓	Brute-force	–	–
SMS	✓	✓	✓	Brute-force	–	–
Notes	✓	✓	✓	Brute-force	–	–
Calendar	✓	✓	✓	Brute-force	–	–
Call history	✓	✓	✓	Brute-force	–	–
Email	–	✓	NA[b]	NA[b]	–	–

NA: Not Applicable

a. Direct access to Media is always granted to trusted computers via AFC. No backup or exploitation of a jailbreak (AFC2) is needed to exfiltrate media files.

b. Email database is not saved in backups.

–: Data is not readable.

can only be thwarted if the user had used a password for the backups. However, a weak backup password could still be cracked using an offline dictionary or brute-force attack. Even in the most securely configured setting where the device file system has not been jailbroken and the attacker is unable to guess the backup password, media files can be exfiltrated from an iOS device, regardless its lock status, once it is connected to a trusted computer.

VII. Discussion

Data exfiltration threat is not new, but the topic is of ongoing interest to security researchers and practitioners (as well as cybercriminals). In this paper, we successfully exfiltrated data from four generations of iOS devices (iPhone 3G, 4, 4S, and 5S) running different versions of iOS (i.e. 6.1.2, 7.1, 8.1.2, and 8.3), by exploiting the pairing and pairing-lock features which, ironically, were designed to minimise the impact on data privacy in the event an iOS device is misplaced, lost or stolen. Access to a paired computer is not unrealistic in a real-world attack. For example, if malicious features were embedded in popular software for iOS device data management, say DiskAid, iFunBox, or iPhoneBrowser, which was installed on a paired computer by the user, data exfiltration could easily be facilitated via these features (i.e. backdoors). This also highlights the risks of using third-party software.

Other attacks could also exploit the backup password. For example, if backup encryption is not turned on via iTunes, an attacker could set a backup password unknown to the user using the *AppleMobileBackup.exe* command-line tool. Although this would not prevent the user from performing further backups, data restoration could not be performed without knowing the secret password. Consequently, it will be necessary to factory reset the device to delete the backup password set by the attacker. If backup encryption is turned on by the user, the backup password could be captured using a keylogger installed on the mobile device or a trusted computer. The captured password could be subsequently used to decrypt files from the exfiltrated encrypted backup.

Prior to iOS 8, all user data exfiltrated in our case studies, with the exception of e-mails, was encrypted with the key associated with the *NSFileProtectionNone* class (see Table II). As this key is not derived from the user passcode and it is always available in memory, all user data encrypted with such a key can be decrypted.

With the introduction of iOS 8, however, AddressBook, SMS, Notes, Calendar, Call history, and Media files are encrypted with the *NSFileProtectionCompleteUntilFirstUserAuthentication* class key, which is derived from the user passcode. However, we remark that a device running iOS 8 that has been unlocked just once since the last reboot will operate the same way as a device running iOS 7 or below. In other words, the decryption keys will remain in the memory after the device has been unlocked for the first time, even if the device is subsequently locked. Therefore, our data exfiltration model could be used by an attacker to target devices running iOS 8.

In summary n Section VI.A, we describe how the client application communicates with devices via the iOS Services. In Section VI.B, we demonstrate how the data exfiltration can be carried out.

As a result of our findings, we propose the following recommendations to improve the security for iOS devices.

Recommendation 1 (for device manufacturers): In order to achieve a more fine-grained restriction that is not just limited to paired computers, Apple should provide mechanisms that allow users to selectively authorise a client software to access device resources on the user's behalf. For example, Apple could consider the implementation of an OAuth-like approach in order for developers to register their client software and obtain a unique token in return. Clients requesting permission for the first time to access resources on an iOS device would supply the token for authorisation via iCloud in order to verify that the token is still valid and has not been revoked by Apple. Hence, when a user authorised a client to interact with their iOS device, a new record would be created and stored in the device keychain. The validity of the token should then be checked on a regular basis based on the configuration settings (e.g. when an Internet connection is detected). In addition, we recommend that iTunes be redesigned to operate in a similar basis.

Recommendation 2 (for device users): Users should practise security hygiene, for example, by not installing applications of unknown origin. As we demonstrated in this paper, jailbroken devices could be exploited by malicious applications running on a personal computer used to connect to an iOS device. In addition, it is good practice for users to create a backup password not only to protect databases containing sensitive data (i.e. Addressbook, SMS, Notes, Calendar, and Call history) but also to prevent an attacker from setting an arbitrary password that would require to factory reset the device in order to be removed. The consequence of losing one's data in the increasingly digitalised world is dramatically demonstrated in the incident involving a senior staff writer with WIRED.

In the space of one hour, my entire digital life was destroyed. First my Google account was taken over, then deleted. Next my Twitter account was compromised, and used as a platform to broadcast racist and homophobic messages. And worst of all, my AppleID account was broken into, and my hackers used it to remotely erase all of the data on my iPhone, iPad, and MacBook [36]

VIII. Concluding Remarks

IoT security is a topic of ongoing research interest [39][40], and both big data and IoT security 'space' can be seen as a race, not only to keep up with technological advances, but also from software and hardware modifications made by end users that could be exploited by cybercriminals. For example, unsafe applications on mobile devices, a common IoT device and source of big data, can result in security and data privacy issues to the users [32], as well as the organisations they work for [37]. For example, we demonstrated in this paper that user's private data can be covertly exfiltrated from iOS devices by exploiting the pairing and pairing-lock features.

While the attacks described in this paper may only affect the current devices and iOS version, we contributed towards a better understanding of an attacker's capabilities in data exfiltration from iOS devices connected to trusted computers, by presenting a data exfiltration model. The model is then implemented in a proof of concept application to demonstrate the practicality of the approach.

Future research includes extending the work to other IoT devices and big data systems.

References

[1] C. Xiao (2014), *Wirelurker: A New Era in iOS and OS X Malware*, Palo Alto Networks, Santa Clara, CA.

[2] FireEye (2014), *A Comprehensive Mobile Threat Assessment of 7 Million iOS and Android Apps*, FireEye Inc., Milpitas, CA.

[3] C. Xiao (2015), *KeyRaider: iOS Malware Steals Over 225,000 Apple Accounts to Create Free App Utopia*, media release, 30 August, Palo Alto Networks, viewed 1 February 2016, <http://researchcenter.paloaltonetworks.com/2015/08/keyraider-ios-malware-steals-over-225000-apple-accounts-to-create-free-app-utopia/>.

[4] N. Provos, D. McNamee, P. Mavrommatis, K. Wang, and N. Modadugu (2007), "The ghost in the browser analysis of web-based malware", *Proceedings of the conference on First Workshop on Hot Topics in Understanding Botnets, Berkeley*, CA, USA. USENIX Association.

[5] PandaLabs (2015), *PandaLabs neutralized 75 million new malware samples in 2014, twice as many as in 2013*, Panda Security, viewed 1 February 2016, <http://www.pandasecurity.com/mediacenter/src/uploads/2015/02/Pandalabs2014-DEF2-en.pdf>.

[6] Apple (2015), *iOS Security*, viewed 1 February 2016, <https://www.apple.com/business/docs/iOS_Security_Guide.pdf>.

[7] Statista (2015), *Global market share held by operating systems Desktop PCs from January 2012 to June 2015*, viewed 1 February 2016, <http://www.statista.com/statistics/218089/global-market-share-of-windows-7/>.

[8] Netmarketshare (2015), *Desktop Operating System Market Share*, viewed 1 February 2016, <https://www.netmarketshare.com/operating-system-market-share.aspx>.

[9] L. Xing, X. Bai, T. Li, X. Wang, K. Chen, X. Liao, S-M. Hu, and X. Han (2015), *Unauthorized Cross-App Resource Access on MAC OS X and iOS*, Cornel University Library, viewed 1 February 2016, <http://arxiv.org/pdf/1505.06836v1.pdf>.

[10] L. Caviglione and W. Mazurczyk (2015), "Understanding Information Hiding in iOS", *IEEE Computer*, vol. 48, no. 1, pp. 62–65.

[11] Y. Agarwal and M. Hall (2013), "ProtectMyPrivacy: detecting and mitigating privacy leaks on iOS devices using crowdsourcing", *Proceedings of the international conference on Mobile systems, applications, and services*. New York, NY, USA, pp. 97–110.

[12] P. Porras, H. Saïdi, and V. Yegneswaran (2010), "An Analysis of the iKee.B iPhone Botnet", *Security and Privacy in Mobile Information and Communication Systems,* Springer Berlin Heidelberg, pp. 141–152.

[13] R. Farley and X. Wang (2010), "Roving bugnet: Distributed surveillance threat and mitigation", *Computers & Security*, vol. 29, no. 5, pp. 592–602.

[14] A. Giani, V. H. Berk, and G.V. Cybenko (2006), "Data exfiltration and covert channels", *in Sensors, and Command, Control, Communications, and Intelligence (C3I) Technologies for Homeland Security and Homeland Defense V.*

[15] W. D. Yu, S. Nargundkar, and N. Tiruthani (2009), "PhishCatch - A Phishing Detection Tool", *Proceedings of the IEEE International Conference on Computer Software and Applications*, vol. 2, pp. 451–456.

[16] G. Wang, J. W. Stokes, C. Herley, and D. Felstead (2013), "Detecting malicious landing pages in Malware Distribution Networks", *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 1–11.

[17] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang (2012), "RiskRanker: scalable and accurate zero-day android malware detection", *Proceedings of the international conference on Mobile systems, applications, and services*, NY, USA, pp. 281–294.

[18] A. Sami, B. Yadegari, H. Rahimi, N. Peiravian, S. Hashemi, and A. Hamze (2010), "Malware detection based on mining API calls", *Proceedings of the ACM Symposium on Applied Computing,* New York, NY, USA, pp. 1020–1025.

[19] M. Antonakakis, R. Perdisci, W. Lee, N. Vasiloglou, and D. Dagon (2011), "Detecting malware domains at the upper DNS hierarchy", *Proceedings of the USENIX conference on Security.*

[20] M.A. Rajab, L. Ballard, N. Jagpal, P. Mavrommatis, D. Nojiri, N. Provos, and L. Schmidt (2011), "Trends in circumventing web-malware detection", *Google Technical Report*.

[21] Q. Do, B. Martini, and K-K. R. Choo (2015), "Exfiltrating data from Android devices", *Computers & Security*, vol. 48, pp. 74–91.

[22] A. Davis, M. Rubinstein, N. Wadhwa, G. J. Mysore, F. Durand, and W. T. Freeman (2014), "The visual microphone: passive recovery of sound from video", *ACM Trans Graph*, vol. 33, no. 4, pp. 79:1–79:10.

[23] M. Hanspach and M. Goetz (2014), "On covert acoustical mesh networks in air", *Journal of Communications*, vol. 8, no. 11, pp. 758–767.

[24] P. Marks (2014), *Hackers reverse-engineer NSA's leaked bugging devices*, media release, 18 June, New Scientist, viewed 1 February 2016, <https://www.newscientist.com/article/mg22229744-000-hackers-reverse-engineer-nsas-leaked-bugging-devices/>.

[25] S. J. O'Malley and K-K. R. Choo (2014), "Bridging the Air Gap: Inaudible data exfiltration by Insiders", *Proceedings of the Americas Conference on Information Systems*, pp. 1–12.

[26] J. Zdziarski (2014), "Identifying back doors, attack points, and surveillance mechanisms in iOS devices", *Digital Investigation,* vol. 11, no. 1, pp. 3–19.

[27] McAfee (2015), *Grand Theft Data. Data Exfiltration Study: Actors, Tactics, and Detection*, McAfee Inc., viewed 1 February 2016, <http://www.mcafee.com/us/resources/reports/rp-data-exfiltration.pdf>.

[28] A. Rashid, R. Ramdhany, M. Edwards, S. M. Kibirige, A. Babar, D. Hutchison, and R. Chitchyan (2014), *Detecting and Preventing Data Exfiltration*, Lancaster University, Security Lancaster Centre.

[29] D. Chismon, M. Ruks, M. Michelini, and A. Waters (2014), *Detecting and Deterring Data Exfiltration*, viewed 1 February 2016, <https://www.mwrinfosecurity.com/system/assets/642/original/Detecting___Deterring_Data_Exfiltration_-_Executive_Briefing_Paper.pdf>.

[30] McAfee (2015), *Stop Data Exfiltration*, McAfee. Part of Intel Security, Santa Clara, CA.

[31] Y. Liu, C. Corbett, R. Archibald, B. Mukherjee, and D. Ghosal (2009), "SIDD: A Framework for Detecting Sensitive Data Exfiltration by an Insider Attack", *Proceedings of the IEEE Hawaii International Conference on System Sciences.*

[32] A. Jain and D. Shanbhag (2012), "Addressing Security and Privacy Risks in Mobile Applications", *IT Professional*, vol. 14, no. 5, pp. 28–33.

[33] J. Leyden (2015), "Apple cleans up iOS App Store after first big malware attack", *The Register*, 21 September, viewed 1 February 2016, <http://www.theregister.co.uk/2015/09/21/xcodeghost_apple_ios_store_malware_zapped/>.

[34] C. Xiao (2015), *Update: XcodeGhost Attacker Can Phish Passwords and Open URLs through Infected Apps*, media release, 18 September, Palo Alto Networks, viewed 1 February 2016, <http://researchcenter.paloaltonetworks.com/2015/09/update-xcodeghost-attacker-can-phish-passwords-and-open-urls-though-infected-apps/>.

[35] C. D'Orazio and K-K. R. Choo (2016), "An adversary model to evaluate DRM protection of video contents on iOS devices", *Computers & Security*, vol. 56C, pp. 94–110, [DOI: http://dx.doi.org/10.1016/j.cose.2015.06.009].

[36] M. Honan (2012), "How Apple and Amazon Security Flaws Led to My Epic Hacking", *The Wired*, media release, 6 August, viewed 1 February 2016, <http://www.wired.com/2012/08/apple-amazon-mat-honan-hacking/>.

[37] J. Imgraben, A. Engelbrecht, and K-K. R. Choo (2014), "Always connected, but are smart mobile users getting more security savvy? A survey of smart mobile device users", *Behaviour & Information Technology*, vol. 33, no. 12, pp. 1347–1360.

[38] Q. Do, B. Martini, K-K. R. Choo (2015), "A Forensically Sound Adversary Model for Mobile Devices", *PLoS ONE*, vol. 10, no. 9, [DOI: http://dx.doi.org/10.1371/journal.pone.0138449].

[39] H. Li, M. Dong, and K. Ota (2015), "Radio Access Network Virtualization for the Social Internet of Things", *IEEE Cloud Computing*, vol. 2, no. 6, pp. 42–50.

[40] J. Wu, M. Dong, K. Ota, L. Liang, and Z. Zhou (2015), "Securing distributed storage for Social Internet of Things using regenerating code and Blom key agreement", *Peer-to-Peer Networking and Applications*, vol. 8, no. 6, pp. 1133–1142.

Christian J. D'Orazio received the B.Sc. degree in Computer Engineering from University of La Matanza, Argentina, in 2000 and the M.Sc. degree in Cyber Security and Forensic Computing from the University of South Australia, Australia, in 2013. He is currently pursuing the Ph.D. degree in Cyber Security and Digital Forensics at the University of South Australia, and is the co-inventor of a provisional patent application on iOS app security.

Kim-Kwang Raymond Choo (SM'15) received the Ph.D. in Information Security from Queensland University of Technology, Australia. He is currently a cloud technology endowed associate professor at University of Texas at San Antonio, an associate professor at the University of South Australia, and a guest professor at China University of Geosciences. He was named one of 10 Emerging Leaders in the Innovation category of The Weekend Australian Magazine / Microsoft's Next 100 series in 2009, and is the recipient of various awards including ESORICS 2015 Best Research Paper Award, Highly Commended Award from Australia New Zealand Policing Advisory Agency, British Computer Society's Wilkes Award, Fulbright Scholarship, and 2008 Australia Day Achievement Medallion. He is also a Fellow of the Australian Computer Society.

Laurence T. Yang (SM'15) received the B.E. degree in Computer Science and Technology from Tsinghua University, Beijing, China, and the Ph.D. degree in Computer Science from the University of Victoria, Victoria, BC, Canada. He is a professor with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China, and with the Department of Computer Science, St. Francis Xavier University, Antigonish, NS, Canada. His research interests include parallel and distributed computing, embedded and ubiquitous/pervasive computing, and big data. His research has been supported by the National Sciences and Engineering Research Council, Canada, and the Canada Foundation for Innovation.