General Release

Apple iPad In the Work Place

Written By Russ Spooner

Portcullis Computer Security LTD
The Grange Barn
Pike's End
Pinner
Middlesex
HA5 2EX

Tel: 020 8868 0098
Fax: 020 8868 0017
rus@portcullis-security.com

Document Reference Whitepapers/WPIOS2011/wp_WPIOS2011_0.3

Version 0.3

Date 16 February 2011

© Copyright Portcullis Computer Security Limited 2011

1 Document History

Revision	Author	Role	Date	Comments
0.1	RUS	Author	09/02/2011	Initial First Draft
0.2	RUS	Author	15/02/2011	Minor Revisions
0.3	RUS	Author	16/02/2011	Updated to reflect new version of redsn0w

Table 1: Document Revision History

Contents

List of Figures

List of Tables

3 Introduction

Security considerations for iOS 4.2.1 and earlier, iPad.

iOS devices such as the iPad are becoming increasingly prevalent in work environments largely due to their ease of use and flexibility, but also due to the so-called "halo effect".

What most users, both corporate and individual, often do not acknowledge are the security weaknesses in the Apple iOS operating system, and additionally iTunes which can easily result in the exposure of highly sensitive information and the compromise of the device itself.

In this whitepaper I will outline and enumerate many of the issues surrounding the introduction of the iPad into the workplace with particular regard to the exposure and theft of sensitive information, countermeasures employed by Apple and how in most cases they are trivially bypassed.

The information provided in this whitepaper is not entirely my own work, and references publicly available tools and information, if I have missed any attribution, please do not hesitate to contact me.

The intended audience for this is technical/managerial, that is to say, in parts it will be moderately technical, but the key focus will be the provision of information to those planning or evaluating roll outs of iOS based devices in order that they are able to accurately understand the risks associated with this.

The reason I am writing this paper, is due to the fact that Portcullis have been approached with increasing frequency with regard to performing security assessments of the iPad, to give our perception of the devices security or to provide guidance with regard to deploying them securely.

In a sense this is to be considered a summation of my findings, it is not by any means intended to dissuade, impede or scaremonger, but rather to enable informed understanding of the risks that these devices may introduce. Wherever possible I will suggest mitigating strategies, in some cases they are not possible.

Also I will wherever possible be steering away from naming specific 3rd party applications, or vendors as it is not my intent to either endorse or condemn them. Also trademarks or registered trademarks are the property of their respective owner(s).

4 The State Of Play

For the sake of this document we are going to assume that we are dealing with the iPad 3G running iOS 4.2.1.

Although there are other versions of hardware that run iOS currently in circulation, and they will be mentioned where it is meritorious or useful as a comparison, we are looking into iPad deployments.

5 Evolution

The iPhone was first released to the public in 2007, running a derivative of Mac OSX/Darwin compiled for the ARM processor, which became known as iOS.

Apple released Darwin, an open source operating system, in 2000; it is POSIX compliant and is compatible with the Single UNIX Specification version 3 (SUSv3).

Despite the inherent flexibility of the base operating system, initial releases of iOS provided no intended facility for the running of 3rd party applications, instead relying on web applications to deliver functionality beyond that delivered by the built in applications.

However, enterprising individuals on the internet were quickly able to "Jailbreak" the operating system, effectively gaining interactive access to the underlying operating system. Common UNIX utilities were easily ported to iOS and soon many unofficial 3rd party application started to appear.

Perhaps as a reaction to this Apple announced an official SDK on October 2008.

In March 2008 Apple released its first beta of the iPhone SDK which would permit developers to officially develop native applications for the operating system. The applications would be distributed via the Apple "App Store" and, of course, iTunes.

Despite this Jailbreaking remained popular, in some part due to the restrictions placed upon developers. For instance, applications which used Apple "Private" APIs were (and are) rejected by Apple and therefore the only viable release vector for them was through the "unofficial" app stores (cydia, icy and rock).

Unoffical 3rd party applications which extend the functionality and customisability of the iOS interface/launcher (Springboard) such as Winterboard have also helped to ensure that Jailbreaking is a popular and sought after procedure.

Naturally a sizeable number of Jailbreakers do so in order to "pirate" Official 3rd party applications, and also to remove carrier locks from the iPhone baseband.

It is estimated that anywhere between 10 and 20% of iOS devices have been Jailbroken. Which equates to a vast number of devices. For instance analysts anticipate that Apple will have sold over 100 million iPhones by 2011, and the Wall Street Journal estimates 20 Million iPads will sell in the same year.

Apple responded with each subsequent release of iOS with countermeasures intended to close vulnerabilities in the operating system and its components effectively stopping Jailbreaking. However this has led to a "cat and mouse" approach to vulnerability research and development. i.e. vulnerabilities which may result in "root" level compromise of the device are closely guarded and shared principally amongst members of the Jailbreaking development community.

Naturally suppression of Jailbreaking has not been the sole motive of updates to iOS. New features and functionality have been gradually introduced to address many of the perceived shortcomings of the operating system, such as Multitasking, copy and paste, improved battery usage etc.

Successive iterations of iOS devices have also sought to improve performance and indeed security of the device. For instance with the iPhone 3GS and the iPad hardware encryption was introduced.

The iPad was actually developed before the iPhone, but it was realised that the technology would work well as a mobile phone platform, and emphasis was shifted in that direction. The iPad was finally announced in January 2010, and released in April.

Versions of iOS running on the iPad, and iPhone were converged in November 2010 with the release of version 4.2.

6 Deployments

6.1 Why are we deploying?

There are many reasons why companies, institutions and even schools are seeking to deploy the iPad. Many of these are not business related:

- Ease of access to information
- A consistent, easily maintained platform
- Portability
- Robust hardware
- To reduce paper use
- Improved communications
- Relatively low cost
- Security features (Often said with a straight face)
- They are "shiny"

6.2 Where are we deploying?

At Portcullis we deal with companies from all sectors, usually those with a very low risk appetite. We have so far been approached by clients looking at small deployments (less than 100) in areas such as:

- Financial
- Media
- Communications

But anecdotally and via the press we are seeing large demand in:

- Education
- Local Government
- Healthcare

So that covers most sectors.

What is interesting is that we are seeing the primary demand and deployment targets coming from and to the "board level".
Students and medical staff seem to be the principal targets in the public sector.

6.3 Personal vs Private Property?

It is important to understand who "owns" the device. Is this a personal device that is being connected to corporate resources?
Is it a corporate device that is being connected to personal resources?

The devices are media centric, meaning that they are designed for games, music, photos, films as well as web-browsing and information viewing/sharing.

Corporations and institutions need to be aware that there is a strong possibility that data on the device can become blended, i.e. that corporate data is stored alongside personal data.

Such blended data can then either be synchronised into a corporate or a home environment.

Obvious risks are that if a device that has been used to access corporate data is backed up to a personal computer, then that corporate data is then effectively propagated to that computer.

Conversely there may be support and policy considerations, (not to mention potential copyright issues) should a device be synchronised with a corporate computer. For instance it may be against company policy for iTunes to be installed.

Other less obvious risks are features within applications such as MobileMail. The "unified inbox" is a good example of this. If multiple email accounts are configured on the device, they can become effectively merged into one "inbox", it can then become very easy to compose or forward mails via the "wrong" account. This may bypass content filtering requirements, or email archiving policies.

6.4 Why does this matter?

In essence, who uses iPads shouldn't matter, however when you consider the reasons for deploying them, and who they are being deployed to and then contrast that to the probability of information exposure we have a rather unappealing scenario.

It would appear that highly sensitive information storage is being carried out by a low security system.

7 Core Security Features

Here we are going to take a look at the core security features of iOS devices, both at the hardware and software levels:

7.1 Device policies/profiles

Passcodes can be set by users, or by applying an MS Exchange ActiveSync policy.

The default is a 4 digit PIN, which, if entered 10 times incorrectly will cause the device to wipe.

The only way to change from this default is by applying an Exchange policy which will then enable the following to be set:

- Enforce password on device
- Minimum password length
- Maximum failed password attempts
- Numbers and letters both required
- Inactivity time in minutes

With MS Exchange Server 2007 the following additional policies are supported.

- Simple password allowed or prohibited
- Password expiration
- Password history
- Policy refresh interval
- Minimum number of complex characters in a password

These policies are either deployed "over the air" or as part of a configuration profile that users can install.

The policies can be signed, and password protected requiring an "administrator" to remove them. A policy "lock" can be enforced, which will require the device to be wiped upon removal of the policy.

Configuration profiles are XML files which can contain information such as server settings, security policies that will be applied to the device.

Its interesting to note that when configuration profiles are encrypted, they automatically enforce encryption of backups in iTunes.

Device restrictions can be applied which can prevent users performing certain actions, such as installing applications, accessing YouTube, etc.

7.2 Filesystem Encryption Fallacy

Although iOS devices (iPhone 3GS+, and iPad) have a hardware-level encrypted filesystem, there is a misconception that the information is actually protected.

The filesystem is effectively decrypted at boot-time (the bootloader needs to access the filesystem to start iOS), thereby effectively rendering the encryption redundant in terms of protecting information on a running iOS device.

Where the encryption comes into play is when a "remote wipe" command is pushed to the device, via either MS Exchange or MobileMe. At this point iOS deletes the encryption keys and forces a reboot thereby rendering the information on the device inaccessible, and indeed unbootable.

8 Where is information stored?

In order to understand the risk of information exposure or theft we need to understand where information is stored and how.
Although iOS is a UNIX based operating system and uses HFS as the filesystem, iOS relies on two main types of files to store and retrieve information, and to store configuration information:

Plists (preference lists) are XML based plain text files, (or in some case binary) that contain various settings and other information pertaining to applications and how the operating system is configured.

SQLite databases typically contain application specific data.

Tools are freely available to interrogate both files either using a GUI or via a command line interface.

Clearly these files are the key point of interest for individuals seeking to extract information from iOS devices. Indeed, those familiar with UNIX commandline tools such as grep will be able to extract very interesting information from either of these file types.

For all intents and purposes each application will use SQLite databases to store data, this will include Emails, Images and so forth.

8.1 Problems with SQLite

SQLite, according to wikipedia is:

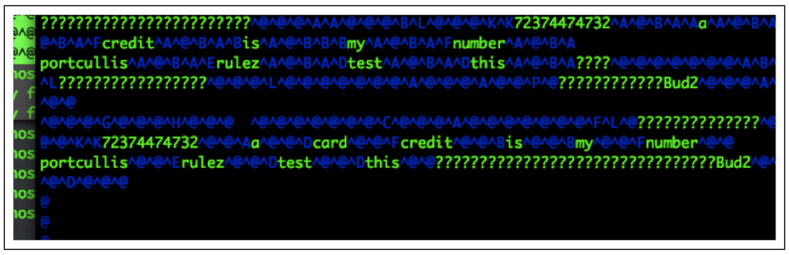
"an ACID-compliant embedded relational database management system contained in a relatively small (approx 225 kB) C programming library. The source code for SQLite is in the public domain."

Which makes it ideal for a low footprint, swift and easy to use platform for data manipulation on a small device.

These databases can be accessed either by copying them off the device after Jailbreak or by accessing the iTunes backup. Once retrieved there are many SQLite database viewers, which come in very useful in examining live data on the device.

What we have found in our investigations is that data stored in these databases is persistent and quite tenacious.

For instance, when you delete a notes entry it is just flagged as deleted, it isn't actually removed. This information cannot be accessed using standard SQLite browsers, however simple tools like "vi" or "strings" can be used to view the "deleted" data:

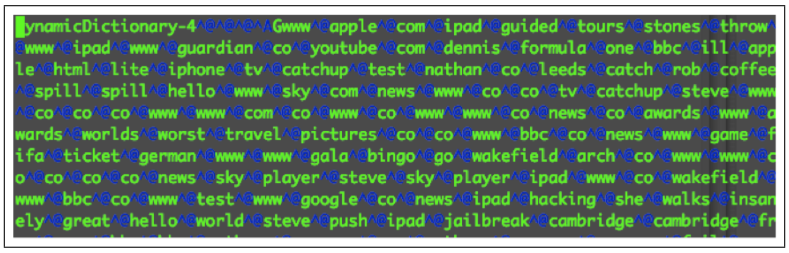


```
???????????????????????^@^@^@^A^A^@^@^@^B^L^@^@^@^K^K72374474732^A^@^B^A^Aa^A^@^B^A
@^B^A^Fcredit^A^@^B^A^Bis^A^@^B^B^Bmy^A^@^B^A^Fnumber^A^@^B^A
portcullis^A^@^B^A^Erulez^A^@^B^A^Dtest^A^@^B^A^Dthis^A^@^B^A????^@^@^@^@^@^@^A^B^
^L?????????????????^@^@^@^L^@^@^@^@^@^@^@^A^@^@^@^A^@^@^P^@????????????Bud2^@^@^@^A^
^@^@
^@^@^@^G^@^@^@^H^@^@^@  ^@^@^@^@^@^@^@^C^@^@^@^A^@^@^@^@^@^@^@^F^L^@??????????????^@
@^@^K^K72374474732^@^@^Aa^@^@^Dcard^@^@^Fcredit^@^@^Bis^@^@^Bmy^@^@^Fnumber^@^@
portcullis^@^@^Erulez^@^@^Dtest^@^@^Dthis^@^@???????????????????????????????Bud2^@^
^@^D^@^@^@
@
@
```

Figure 1: Deleted Data

Also:

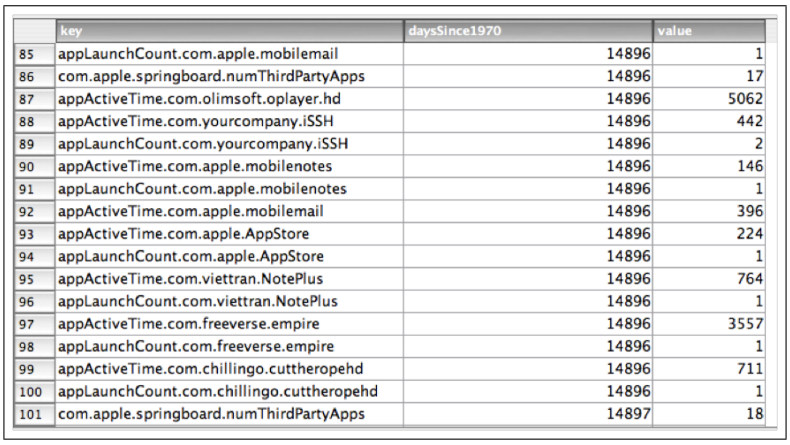
- the Dynamic Dictionary feature stores whole phrases in a database (including, under certain circumstances could include credit card numbers, passwords, etc)



```
ynamicDictionary-4^@^@^@^AGwww^@apple^@com^@ipad^@guided^@tours^@stones^@throw^
@www^@ipad^@www^@guardian^@co^@youtube^@com^@dennis^@formula^@one^@bbc^@ill^@app
le^@html^@lite^@iphone^@tv^@catchup^@test^@nathan^@co^@leeds^@catch^@rob^@coffee
^@spill^@spill^@hello^@www^@sky^@com^@news^@www^@co^@co^@tv^@catchup^@steve^@www
^@co^@co^@co^@www^@www^@com^@co^@www^@co^@www^@www^@co^@news^@co^@awards^@www^@a
wards^@worlds^@worst^@travel^@pictures^@co^@co^@www^@bbc^@co^@news^@www^@game^@f
ifa^@ticket^@german^@www^@www^@gala^@bingo^@go^@wakefield^@arch^@co^@www^@www^@c
o^@co^@co^@co^@news^@sky^@player^@steve^@sky^@player^@ipad^@www^@co^@wakefield^@
www^@bbc^@co^@www^@test^@www^@google^@co^@news^@ipad^@hacking^@she^@walks^@insan
ely^@great^@hello^@world^@steve^@push^@ipad^@jailbreak^@cambridge^@cambridge^@fr
```

Figure 2: Dynamic Dictionary

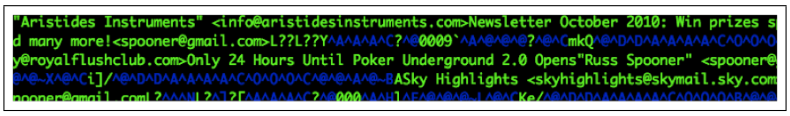
- iOS logs when and how often applications have been launched:



	key	daysSince1970	value
85	appLaunchCount.com.apple.mobilemail	14896	1
86	com.apple.springboard.numThirdPartyApps	14896	17
87	appActiveTime.com.olimsoft.oplayer.hd	14896	5062
88	appActiveTime.com.yourcompany.iSSH	14896	442
89	appLaunchCount.com.yourcompany.iSSH	14896	2
90	appActiveTime.com.apple.mobilenotes	14896	146
91	appLaunchCount.com.apple.mobilenotes	14896	1
92	appActiveTime.com.apple.mobilemail	14896	396
93	appActiveTime.com.apple.AppStore	14896	224
94	appLaunchCount.com.apple.AppStore	14896	1
95	appActiveTime.com.viettran.NotePlus	14896	764
96	appLaunchCount.com.viettran.NotePlus	14896	1
97	appActiveTime.com.freeverse.empire	14896	3557
98	appLaunchCount.com.freeverse.empire	14896	1
99	appActiveTime.com.chillingo.cuttheropehd	14896	711
100	appLaunchCount.com.chillingo.cuttheropehd	14896	1
101	com.apple.springboard.numThirdPartyApps	14897	18

Figure 3: Application Launch Log

- The "Envelope Index" stores Email headers, even for deleted accounts, and account information persists in various files:



```
"Aristides Instruments" <info@aristidesinstruments.com>Newsletter October 2010: Win prizes s
d many more!<spooner@gmail.com>L??L??Y^A^A^A^C?^@0009`^A^@^@^@?^@^CmkQ^@^D^D^A^A^A^A^C^O^O^O
y@royalflushclub.com>Only 24 Hours Until Poker Underground 2.0 Opens"Russ Spooner" <spooner@
@^@~X^@^Ci]/^@^D^D^A^A^A^A^A^C^O^O^O^C^@^@^A^@~BASky Highlights <skyhighlights@skymail.sky.com
pooner@gmail.comL?^^^NL?^]?[^A^A^A^A^C?^@000^A^H]^F^@^@^@~L^@^CKe/^@^D^D^A^A^A^A^A^A^C^O^O^O^B^@^@
```

Figure 4: Envelope Index

8.2 iTunes

Each time you connect your device to your PC/Mac iTunes can be configured to automatically back up your device. This is a very useful feature, and it is a very thorough backup, to the extent that if you were to lose your device and get a new one, you can restore this backup and barely even notice you had a new one (more on this later).

These backups are stored in the following locations:

- Windows XP:

`C:\Documents and Settings\$USERNAME\Application Data\Apple Computer\MobileSync\Backup`

- Windows Vista and 7:

`C:\Users\$USER\AppData\Roaming\Apple Computer\MobileSync\Backup`

- OSX:

`~/Library/Application Support/MobileSync/Backup/`

In the relevant folder you will find what appears to be a folder or folders whose name consists of a Unique Identifier. Within this folder are all the backed up files pertaining to your device.The files are, simply put, preference lists and SQLite databases. They do not have meaningful names, but that wont deter us as you will see later.

Key SQLite databases and plists that are synchronised are listed in appendix A.

8.3 Local Filesystem

iOS devices have two main partitions, the "root" or system partition where operating system files are stored:

```
/dev/disk0s1 / rw 0 1
```

and the "media" partition where "user" files are stored:

```
/dev/disk0s2 /private/var hfs rw,noexec 0 2
```

Under normal (i.e. non-Jailbroken) circumstances, it is only possible for users to access the media partition; even then this access is heavily restricted by application sandboxing.According to Apple applications can only access files and directories in their "area" on the filesystem for instance an example directory structure could be:

```
|Application GUID
|
|_Application.app
|_Documents/
|_Library/Preferences/
|_tmp/
```

Thus any given application should only have access to its "own" files.However, once Jailbroken the full filesystem is available to applications, which as we will see greatly impacts the security of the device.

9 Accessing the data

A nefarious individual's objective is to access this information covertly, with minimal physical access, leaving little or no evidence of tampering, ideally persistently and of course getting lots of sensitive stuff either for blackmail or commercial advantage.

9.1 Simple attacks

By default iTunes stored a backup of the device unencrypted. Whether or not the backup is encrypted is enforced by a flag set in a plist on the device itself. A user specified encryption key is also stored on the device in the keychain. The keychain is an encrypted database of passwords stored by the device.

Accessing an unencrypted iTunes backup is trivial as we have seen above, but what else can we do with these backups?

We can copy them form the host computer to our computer for analysis, we can even restore the backup to our phone, effectively cloning it. Unfortunately the keychain does not survive this (due to the way it is encrypted) so we wont be able to retrieve passwords this way.

We can also edit backups. They aren't signed. We can then restore them back.

If we go back a little bit and look at how iOS and iTunes handles backups crudely put this is how it occurs:

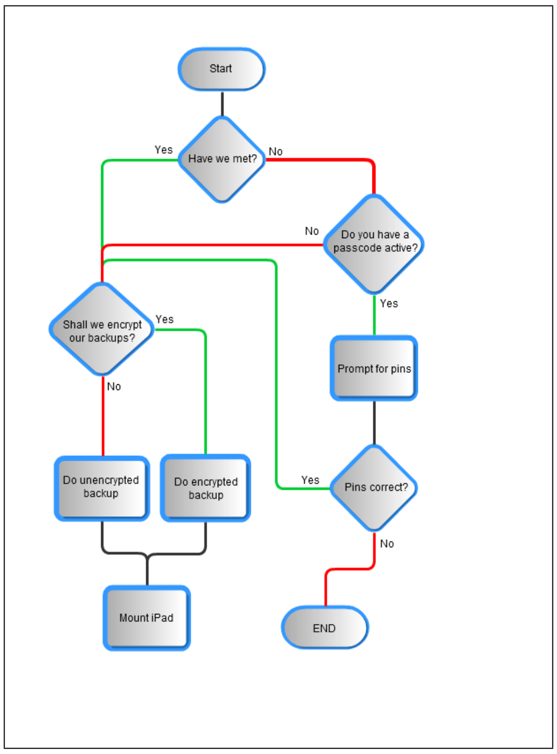


Figure 5: Synchronisation Handshake

In essence we need to either have "paired" the iOS device to iTunes or we need to know the passcode in order to gain access to an unencrypted backup.

There are other speed bumps as well, for instance an Exchange policy could mandate encrypted backups, but again we can overcome this.

We can remove the passcode by editing the relevant plist, (this is where grep comes in handy as all the plists have what seem to be "random" names) we are looking for something like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.
0.dtd">
<plist version="1.0">
<dict>
        <key>PasswordInformation</key>
        <dict>
                <key>pinTimeStamp</key>
                <date>2010-07-20T11:46:22Z</date>
        </dict>
</dict>
</plist>
```

Remove everything from the inner "<dict>" save the file and restore the device. Bingo. No passcode.Another means of removing the passcode is to delete the keychain from the backup, but as this would also erase other passwords stored on the device, it is counter productive in that it would hinder our ability to retrieve further information from the device once it is unlocked.

This backup tampering can be used to defeat a large number of security features that may be enforced by Exchange policies. Things such as: Policy refresh intervals, auto lock, lock interval. You can also use this technique to increase your high scores in certain games. The only limits are your own ingenuity. Remember to keep a backup of yourĚ backup.

What if the passcode is present and active on the device and it hasn't been "paired" with your computer?

This is not as simple as it may seem at first glance. iOS keeps track of which computers it has paired with, so we have a "chicken and egg" scenario: In order to remove the passcode by backup tampering we have to first bypass the passcode.

We could attempt to brute-force guess the passcode, however, by default, after 10 failed attempts the device will wipe itself, which would render our mission to extract data, a failure.

Thinking briefly about the default 4 digit PIN based passcode. There are 10,000 possible numbers, (0000-9999). Giving you a 1 in 1,000 chance of guessing the right combination within 10 attempts. We could conceivably reduce this, by shoulder surfing, using "common" PIN numbers, using social engineering tactics or interestingly examine the screen to see if there are fingerprints around the keypad area on the device which could expose digits present in the PIN.

The paring mechanism seems to be quite robust, so the rather obvious advice here is that if you want to keep your data safe, ensure that you are very cautious about what computers you connect your device to.

Backup encryption does present another challenge. There are only really two options open to us; Either we have to bruteforce guess the backup password or we are going to have to resort to exploiting the device…

9.2 Jailbreaking

As discussed earlier, iOS restricts access to the entire filesystem to the base operating system itself. Additionally it provides no native means to access the underlying operating system.

Jailbreaking essentially foils that restriction, allowing for unrestricted access to the device. Effectively put it means we can run any code on the device we like, ignoring restrictions such as application signing, adhering to prescribed application sandboxing, and read/write access to the system partition.

In order to Jailbreak, vulnerabilities must be identified in the software or firmware running on the device. These vulnerabilities must have certain characteristics in order to be useful in Jailbreaking. The most important of these is that it must enable us to be able to run arbitrary code as the "root" user.

There are a large number of "Jailbreaking" tools available for a variety for versions of iOS. As apple patches vulnerabilities in iOS or the Bootroms of their devices, Jailbreakers have to find new vulnerabilities to incorporate into their tools.
Jailbreaking is also divided into two broad categories:

- Untethered - Meaning that once Jailbroken the device, if rebooted, will start normally with no intervention
- Tethered - with this type of Jailbreak, user intervention is required in order for the device to restart. The device will need to be connected to a computer and effectively be re-Jailbroken in order to boot.

Whether a device can be Jailbroken untethered (which is the optimal route) is dependant on the bootrom version, and the firmware version.

Currently iOS versions 3.2.2 and earlier can be Jailbroken untethered, more recent versions will require further steps to be taken in order to remove the tether (such as running Greenpois0n, an alternate Jailbreaking tool, **after** Jailbreaking the device with redsn0w). In coming months these additional steps are likely to become redundant, thus for brevity the following Jailbreak steps will work cleanly on iOS versions earlier than 4.2.1.

I.e. if you are going to attempt what I will describe do some research first: there is a high degree of risk for the uninitiated and unless you are lucky or well informed you might end up with a highly desirable, expensive placemat.

Update:

As of 16th February redsn0w was updated to version 4.2, which is an untethered jailbreak. Therefore all versions of iOS up to and including 4.2.1 can be jailbroken safely, and untethered. Thus the following steps can be replicated on all version of iOS for the iPad and iPhone.

9.3 Owning the Device

A common, and flexible, tool available to use with iOS 4.2.1 (and earlier) is redsn0w.

Accurately speaking we do not need to fully Jailbreak in order to access data on the device, we just need to be able to boot the device with a custom ramdisk. iOS devices have the facility to do this either by dropping into "recovery mode" (intended for operating system recovery or upgrade) or DFU mode (intended for firmware upgrade).

Redsn0w depends upon an exploit known as Limera1n, which takes advantage of both of these modes, employing both a bootrom exploit as well as a userland exploit to fully Jailbreak the device.

However as we do not have access to the code for redsn0w, we can't change its behaviour to stop it fully Jailbreaking the device.

If we were able to customise the actions it would be a simple matter to remove the device passcode by editing the following file:

```
/private/var/Managed Preferences/mobile/com.apple.springboard.plist
```

(as we would have done in the backup tampering method).However if we wanted to remove the backup encryption as well, we would have to do a little more.

By deleting (or renaming) the keychain:

```
/var/Keychains/keychain-2.db
```

we not only remove the passcode, but also the key used to encrypt the backups, thus the backups will be unencrypted. Sadly we do sacrifice other passwords, too such as email passwords, etc.Redsn0w, though ostensibly a Jailbreaking tool, is actually a little more: it can be used to install custom bundles. Custom bundles are essentially compressed archives containing content (such as executable binaries, or even preference lists), which are copied to the device.

Thus we can use this feature, to fully Jailbreak the device, copy some scripts and tools to the device in order to compromise it. And we can compromise it in such a way that the device shows almost no evidence to the user that it has been.

Redsn0w depends on having a copy of the restore image for the device being Jailbroken (these are freely available from apple) and will have to be downloaded in advance.

So, we have the following scenario:

- An iPad with a passcode set, backup encryption enabled.
- We have a laptop (running OSX or Windows)

- An up-to-date version of redsn0w
- A custom bundle (with OpenSSH, APT, and a few other tools and scripts)
- A collection of restore images
- 5 minutes left alone with the device.
- A network connection

We don't need to worry about the device being paired to our laptop as it sidesteps the device pairing requirement. As a part of the "Jailbreak" the device is put into DFU mode. In this mode the device is in a state where it is to all intents and purposes unable to check whether it is paired to the computer it is connected to.

The iPad

We can take a guess to see if it is running 4.2.1 by flicking the hardware switch on the side. If it mutes the volume on the device there it is quite likely to be 4.2.x. Prior versions used this hardware switch to engage the orientation lock. In later versions users were given the option to choose between mute and lock; the default being mute.

Guessing the version of the firmware incorrectly is not fatal, it will simply mean that the Jailbreak will fail and you will have to go through it again.

The Laptop

We have redsn0w, we have launched it and selected the relevant firmware for the device.

Even before we connect the iPad we can allow redsn0w to process the firmware and we are presented with the following choice:

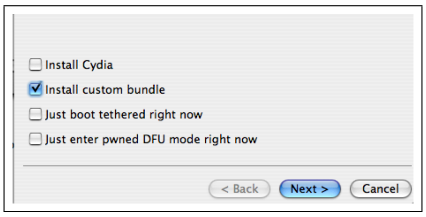

☐ Install Cydia
☑ Install custom bundle
☐ Just boot tethered right now
☐ Just enter pwned DFU mode right now

Figure 6: Redsn0w Options

We don't want cydia to be installed, or the victim will see the icon on their springboard instead we are going to use one of our custom bundles.

We can then follow the steps through redsn0w, the device will reboot. Once this has completed it will be Jailbroken and the passcode will have been removed.

What this custom bundle does:

Installs a large number of basic unix tools, and some key packages: OpenSSH (and a launch script so it starts at boot) and APT (so we can install additional packages from the shell).

It also runs a shell script at start up that renames the keychain. This removes the passphrase.

In order to get its IP address (so we can SSH into it) we can just look in the network preferences for its IP address.

Once we have that we can then simply SSH into the device as "root" (the default password is "Alpine").

In order to remove any obvious traces of us having compromised the device we can rename the keychain back, this won't immediately restore the passcode, we will need to reboot for that to happen. First there are some other things we can do.

Using APT we can install other tools and utilities:

Using "recAudio" will cause the ipad to start recording audio, this is a highly effective way to listen in on meetings. It stores the audio in aiff file, and this can then be copied off the device, or a script could be generated to record at predetermined intervals and then upload the resulting audio file to a web or ftp server.

Other tools such as "pirni" can be scheduled to run, Pirni is an arp-spoofing tool that acts as a man-in-the-middle, sniffing all data on the wireless network. Again, the resultant data can be uploaded to an external server for collection by the attacker.

"Nmap" can be used to map the wireless network, and metasploit can then be used to attack and compromise hosts identified, thereby using the iPad to "pivot" into the corporate environment.

"Netcat" can be configured to initiate a reverse shell to a host on the internet for remote control.

However we also have our "increased stealth" custom bundle, one that:

- Leaves the keychain intact
- Installs OpenSSH
- Installs the above tools
- Gathers information from the device (the dynamic dictionary, Emails, calendar entries etc) and uploads it to my webserver.
- Schedules recordings and uploads them to my webserver
- Attempts a reverse shell to my server each time it detects a network connection.
- Tweets the geographical location of the device daily

10 3rd Party Application security

Even legitimate applications can introduce risks into a corporate environment. As I mentioned in the introduction I am going to avoid naming specific applications or vendors (they will or have been contacted directly) with regard to security issues.

Broadly speaking I have identified two prevalent categories of risk:

10.1 Applications storing sensitive data insecurely

Many applications have the ability to access sensitive data. This data could be as basic as social networking sites, downloading and viewing documents or as complex as remote desktop functionality for accessing corporate resources.

In any case we have identified a large number of applications that store credentials locally in plain text. These credentials can be for corporate servers, internet file stores, websites or even for local access to the application.

Other applications we have seen cache information locally on the device, so that it can be viewed or manipulated offline. These local caches are rarely encrypted.

In both cases insecure storage of cached data or credentials is a bad thing, as it will be synchronised back to iTunes in clear text (if encrypted backups are not enabled).

Failing that, the data can easily be retrieved after a Jailbreak.

Therefore when assessing applications for use within a corporate environment it is important to ensure that the developers have elected to use the iOS encrypted keychain (as recommended by apple) and that they are encrypting any locally cached data.

Another interesting "feature" of iOS itself can introduce weaknesses in application indirectly: the Dynamic Dictionary. Even if an application is not storing information or credentials in an insecure format, it is mre than likely that the Dynamic Dictionary will. It effectively acts as a keylogger on the device. We have seen instances where the dictionary has stored passwords, contact information and all manner of information that would give any individual cause to pale.

10.2 Applications that open services on a network

There are several methods that applications can use facilitate the transfer of data from other sources such as the local network, the internet or a desktop computer.

It is fairly common for applications that view or manipulate documents to run a webserver to facilitate file transfers. Users then can use a web browser on another device or computer to connect to the iPad to upload content.

This may not seem a particularly high level risk, however in some cases we have seen these applications broadcast these services via "bonjour" and almost without exception use predictable TCP ports for their services making them easy to identify on a network. Such servers usually (if not always) by default require no authentication.

iPads deployed in corporate environments will almost certainly be used to view and share sensitive information. It may be that users are inadvertently sharing this information when they connect to the free wireless at their local coffee shop.

11 Good Practise (i.e. How do we fix it)

11.1 Physical security

Clearly physical control of the device is paramount. Detecting if a device has been stealthily Jailbroken without actually Jailbreaking it is tricky, it can be done, but it is better to not let it happen.

If you do lose physical control of the device what then?

If it was left alone for a period of time, or if it was lost and then returned you should assume that it has been compromised. Restore the device; this will effectively remove the Jailbreak, (iTunes doesn't backup any of the Jailbreak or its data).

If it has gone missing attempt to remote wipe the device, however be aware that simply removing the SIM from the device can defeat this. Also remember do the remote wipe before you cancel the SIM, for obvious reasons.

11.2 Policy Controls

Corporate policies should forbid Jailbreaking. There is no guarantee that even if the device has been Jailbroken "legitimately" that applications installed via 3rd party application stores such as Cydia do not contain hostile code.

Determine what data should be permitted on the device. Corporate and personal data should not mix. Highly sensitive information should never be stored locally on the device unless appropriately encrypted.

Control what applications should be run on the device, 3rd party applications can introduce threats.

User awareness and education is paramount. Make certain that users are educated as to the threats to their own as well as company data.

11.3 Technical restrictions

Use applications that enforce data segregation. There are several applications that use their own email, calendar and contact programs, and which enforce local encryption effectively creating a secondary "sandbox" in which corporate data can be handled. Some of these applications use Jailbreak detection and refuse to run if a policy is set to that effect.

Employ Exchange security policies to their best effect, lock down as much as possible. Remember if the device is synchronised regularly it doesn't matter if it is wiped after 3 failed passcode attempts, it can be restored.

Protect the computer that the device is being synchronised to! If you lose the backup of the device, you lose control of the data that has been stored on it.

Examine the capabilities of 3rd party apps. Do they open network ports for document sharing? Have them security tested for vulnerabilities that could expose sensitive information.

Consider using devices as thin clients. There are many remote desktop clients out there that are serviceable. (but again, ensure that they aren't caching credentials in plaintext on the device. Get them tested!)

Restore frequently. A monthly restore of the device should provide some assurance that it is not compromised.

Ensure that devices are maintained at their latest firmware version

© Copyright Portcullis Computer Security Limited 2011

Appendix A List of key files backed up by iTunes

Library_AddressBook_AddressBook.sqlitedb
Library_AddressBook_AddressBookImages.sqlitedb
Library_Calendar_Calendar.sqlitedb
Library_CallHistory_call_history.db
Library_Cookies_Cookies.plist
Library_Keyboard_dynamic-text.dat
Library_LockBackground.jpg
Library_Mail_Accounts.plist
Library_Mail_AutoFetchEnabled
Library_Maps_Bookmarks.plist
Library_Maps_History.plist
Library_Notes_notes.db
Library_Preferences_.GlobalPreferences.plist
Library_Preferences_SBShutdownCookie
Library_Preferences_SystemConfiguration_com.apple.AutoWake.plist
Library_Preferences_SystemConfiguration_com.apple.network.identification.plist
Library_Preferences_SystemConfiguration_com.apple.wifi.plist
Library_Preferences_SystemConfiguration_preferences.plist
Library_Preferences_com.apple.AppSupport.plist
Library_Preferences_com.apple.BTServer.plist
Library_Preferences_com.apple.Maps.plist
Library_Preferences_com.apple.MobileSMS.plist
Library_Preferences_com.apple.PeoplePicker.plist
Library_Preferences_com.apple.Preferences.plist
Library_Preferences_com.apple.WebFoundation.plist
Library_Preferences_com.apple.calculator.plist
Library_Preferences_com.apple.celestial.plist
Library_Preferences_com.apple.commcenter.plist
Library_Preferences_com.apple.mobilecal.alarmengine.plist
Library_Preferences_com.apple.mobilecal.plist
Library_Preferences_com.apple.mobileipod.plist
Library_Preferences_com.apple.mobilemail.plist
Library_Preferences_com.apple.mobilenotes.plist
Library_Preferences_com.apple.mobilephone.plist
Library_Preferences_com.apple.mobilephone.speeddial.plist
Library_Preferences_com.apple.mobilesafari.plist
Library_Preferences_com.apple.mobileslideshow.plist
Library_Preferences_com.apple.mobiletimer.plist
Library_Preferences_com.apple.mobilevpn.plist
Library_Preferences_com.apple.preferences.network.plist
Library_Preferences_com.apple.preferences.sounds.plist
Library_Preferences_com.apple.springboard.plist
Library_Preferences_com.apple.stocks.plist
Library_Preferences_com.apple.weather.plist
Library_Preferences_com.apple.youtube.plist
Library_Preferences_csidata
Library_SMS_sms.db

Library_Safari_Bookmarks.plist
Library_Safari_History.plist
Library_Voicemail_.token

© Copyright Portcullis Computer Security Limited 2011

Appendix B Citations/Further reading

http://www.apple.com/uk/ipad/business/integration/
http://blog.iphone-dev.org/
http://www.theiphonespot.net/?p=7561
http://www.zdziarski.com/blog/?cat=11
http://xsellize.com/index.php
http://www.greenpois0n.com