Case Study: iOS Security

6.1600 Course Staff

Fall 2024

There are several things we might worry about when it comes to the security of a smartphone, spanning software, platform, and hardware security:

- Malicious apps that steal contacts, eavesdrop on calls, or steal you credit card data
- Someone stealing a phone and extracting secret data
- A non-Apple operating system that is loaded on a phone (i.e. Jailbreaking)
- Malicious chips installed in the factory, en route to the store, or during repairs
- People selling fake phones as real iPhones

The iOS platform is a very well-designed integration of the topics that have been covered in the platform security section and which addresses many of these considerations effectively.

Disclaimer: This set of notes is a work in progress. It may have errors and be missing citations. It is certainly incomplete. Please let the staff know of any errors that you find.

1 App Security

In any platform, there is a balance between the "openness" of the platform—who is able to run software on it—and the access that software has to sensitive data. Of course, the most secure platform is one that doesn't do anything at all: it is very closed and provides no access to sensitive data. More practical systems have a different balance:

- A typical laptop is very open—anyone can write software for it—and every application is able to access the entire filesystem.
- The web is still very open, as any website can include JavaScript code that will execute on your machine, but access to sensitive data is tightly locked down via language sandboxing.

iOS improves its security story by locking this down: applications are given some (checked) access to sensitive data, but are very closed—applications must go through a review process and only approved developers can write software for iOS.

In addition to security policies, the review process checks that applications follow policies in place for business reasons, such as the restriction that digital purchases must go through apples In-App Purchase mechanism.

iOS apps run in a sandbox that does not provide access to a shared filesystem and that allows communication between apps only though limited APIs provided by the operating system. If an application wants to interact with data that is controlled by the operating system, such as VPN configurations or health data, the application developer

must ask for access to special APIs and have that access approved as part of the review process.

1.1 *Things Can Still Go Wrong*

Even with these checks and the sandboxing, malware can slip through the gaps. As we discussed earlier, XCode Ghost was a compromised version of the XCode compiler distributed via mirrors behind China's firewall that inserted malware into apps developed by honest developers. This malware was not found in App Store reviews, allowing apps that people expected to be honest to run arbitrary malicious code. Even with the sandbox, an app has quite a bit of access to do potentially sensitive actions:

- Learn Country
- Learn Language
- Learn UUID, before recent privacy changes
- Read and modify clipboard contents, including copied passwords or credit card numbers
- Open a URL that points to a phishing webpage

Despite this, isolation buys a lot of security. A malicious app that makes it through review cannot access your text messages, browser history, etc.

2 *iOS Secure Boot*

For security and business reasons, Apple would like to ensure that an Apple-signed operating system is running on the phone. This prevents a malicious actor from distributing a backdoored operating system and convincing people to install it on their phone and against malware that tries to persistently modify the operating system. It also prevents users from installing a customized operating system on their phone, bypassing the Apple restrictions on apps and other policies.

To achieve this, iOS uses a secure boot system as described in the last chapter. Each phone ships with a Boot ROM that cannot be changed that is burned with some public key for a secret key that Apple knows. This boot ROM is responsible for verifying that this secret key signed the code for a low-level bootloader and running that bootloader. This bootloader will then verify and check the operating system kernel. This allows the bootloader and the kernel to be updated as necessary, but places a root of trust in the boot ROM that cannot be updated. If the bootloader is updated by anyone besides Apple, however, the boot ROM's signature check will fail and the boot ROM will refuse to run the bootloader.

Many device owners that wanted to customize their devices

sought to modify the operating system to add new features. This process generally is referred to as "jailbreaking".

2.1 *Checkra1n Jailbreak*

One set of exploits that constituted a jailbreak was named `checkra1n`. This took advantage of complex code in the unchangeable boot ROM—the devices supported running code directly via USB, bypassing the low-level bootloader and the OS kernel, which meant that the boot ROM contained code to act as a USB peripheral. USB code is quite complex, and as with most complex code, it had bugs. `checkra1n` took advantage of these USB bugs to trick the phone into executing arbitrary code.

Because the Boot ROM can never be updated, it was impossible for Apple to fix these bugs: this jailbreak will work forever on the devices that had the bug.

However, these bugs did not allow the jailbreak to bypass the signature check entirely, so this exploit needed to be run on every boot—if you tried to reboot your phone without running this, it would no longer be jailbroken.

3 *iOS protection for data at rest*

If a phone is stolen, it would be nice if the theif could not learn any sensitive information from the device. As with any time that we want to hide data, encryption is the answer here—a simple solution is to encrypt all data on the phone with 128-bit AES. However, this does not tell the whole story: in order to decrypt the data, the key for this encryption must be stored somewhere. We can't store the key in normal flash memory, since then anyone could use it to decrypt the data. We also can't use the only secret that the user knows, their 6-digit PIN, since it is much too short to be an AES key.

Even an approach like using a PRF based on the key to extend it into key for AES will not be secure—6 digits is so short that it can easily be brute-forced. Instead, recent iPhones use a special chip called a "secure enclave" that holds the key and provides access to it only if the user enters the correct PIN. Doing so allows the secure enclave to enforce strict guess limits that prevent brute-forcing the PIN.

The secure enclave is essentially another processor that runs its own operating system. It uses a similar secure boot system to prevent tampering with the secure enclave's operating system, but it also uses *measured boot* to derive the secret encryption key from the contents of the OS being run—if an attacker modifies the secure enclave's

operating system, the encryption key will change and the attacker will not be able to decrypt the phone data.

Importantly, the secure enclave has access to two things that the main application processor does not. First, on the first boot: the secure enclave generates a long-term secret unique ID and burns it into internal fuses. The enclave also has access to a secure NVRAM module that has a limited amount of secure storage with support for real deletion. This secure storage contains the root encryption key itself, a hashed version of the user's PIN salted with the UID, and a guess counter that keeps track of how many incorrect guesses have been made.

In typical storage, deleting data does not really delete the data—instead, it marks it as deleted and indicates that the operating system should overwrite it in the future. However, if someone inspects the storage directly, they are likely to be able to recover the data that was deleted. Apple's "effacable storage" that is used for the secure enclave supports deleting data such that it cannot be recovered.

The communication between the secure enclave and the secure storage is also encrypted with a key burned into the enclave and into the secure storage. This prevents an attacker with a probe on the wire from learning the data.

When put together, these elements enable the following process:

1. User enters a PIN
2. iOS passes the PIN to the secure enclave
3. The enclave enforces some delay after each guess
4. The enclave passes $H(\text{PIN}, \text{UID})$ to the secure storage.
5. Secure storage uses included logic to check whether the hashed PIN matches the stored hash.
   - If correct, return the root AES key and zero the guess counter
   - If incorrect, increment the guess counter. If the guess counter is too high, erase the key from the effacable storage.
6. If the PIN was correct, the enclave passes the key returned from the storage to the AES engine.
7. The application processor sends data to the AES engine to be encrypted or decrypted.

Note that the application processor never sees the AES key—it is seen only by the secure enclave and the (hardware) AES engine.

3.1 Biometric Unlock

For convenience, however, iPhones do not require entering a PIN on every unlock. A PIN unlock is always required on the first unlock after boot, but afterwards they allow unlocking with a Biometric such as Face/Touch ID. The phone includes dedicated biometrics hardware that is responsible for reporting the "hash" of the measured face to the secure enclave. The secure enclave then checks this against the stored one, and unlocks the device if there is a match.

A possible attack, then, might look something like the following: steal a phone that has been unlocked at least once and relocked (almost always the case). Then, before it runs out of battery and shuts down, replace the Face ID chip with a malicious one that always reports the correct hash. Without protection against replacing the biometric hardware, an attack like this would allow an attacker to access all the data on your phone. To address this, iPhone include

yet another shared key between the secure enclave and the biometric hardware: if the Face ID module is replaced, the keys will not match and the phone will refuse to unlock.