Apple Inc.

Third Party Applications on Mac OS X Embedded

Prepared by: Mitch Adler, John Wright, Dallas De Atley, Et. Al.

November 2, 2020

Apple Inc.

We want to allow third parties to develop applications for the Mac OS X Embedded platform.

The transition from a closed system to an open model demands answers to questions of control and security. In order to maintain the integrity of the system we will have to restrict the manner in which third party applications can run, but also provide a compelling environment for development.

This document defines the goals, approach, and basic methods for application identification and privileged access on Apple devices running the Aspen stack.

Assumptions

- All code running on the device will be signed using an asymmetric key pair.

We intend to sign all applications that run on the device. This includes Apple binaries that ship with the operating system. By signing every binary, we can uniquely identify running code. We will know who created it and will have the ability to revoke its access to the system.

Code signing does not mean that we will solve all of our security problems. However, it does make it significantly harder to deploy on a wide scale malicious code that will affect our customers or partners. It will help reduce the problem of securing the platform.

- We will execute running code within a sandbox.

This document does not address creation of the sandbox, but assumes we will have one to constrain the behaviors of approved applications. This implies that we will employ current measures from Leopard (e.g. Seatbelt) to restrict access to the platform. This topic requires its own analysis and investigation.

Goals

Development

We want to encourage and foster development of applications by individuals, third party software vendors and corporations.

Malware reduction

We want to prevent and discourage the deployment of malware on the platform. Code signing provides a basis for identifying and revoking bad applications, in addition to the potential for legal action against its creator. In addition, we intend to iterate on our sandbox model to protect users from malicious code before it has the chance to cause problems. Examples include software that attempts to copy or transmit data off the device or tries to circumvent the security model to gain access to prohibited services.

By building a mechanism that identifies all of the running code on the system in a trusted fashion, we can then partition functionality and access as the need arises. We may choose to restrict access to specific system services or interfaces. For example, not all applications may be granted access to the EDGE network on an iPhone or other features on other products.

This document doesn't intend to define any categories for what third party applications can and cannot do. Instead, we want to build a foundation that allows us to make these decisions in the future.

Revenue from Apple distributed applications

We want to protect revenue for Apple and third parties who subscribe to Apple's distribution method.

Apple Inc.

Independent application distribution

We want to allow enterprise customers to deploy their own software on devices they control.

Distribution Method

We will distribute third party applications through the iTunes Music Store. However, our model will allow for third parties to distribute their own applications and for enterprise customers to deploy to their own devices.

In order for an application to run on any iPhone it must be signed by an Apple key. We will allow developers and enterprise customers to sign with their own keys, but those applications will be restricted to only running on devices provisioned to allow their keys.

Signing does not imply a specific distribution method, and it's left as a policy decision as to whether Apple signed applications are posted to the online store, or we allow developers to distribute on their own. This policy is easy to enforce for applications which are protected by encryption (probably Fairplay), but for applications not participating in Apple DRM we will not be able to use technology to prevent third parties from distributing software they get signed by Apple, only contractual limitations would stand in the way.

What remains to be defined is how signed applications will be installed on the device. The basic approach is that iTunes will copy the application to the device via AFC and then start an installation service. This service is responsible for verifying the application is legitimate and then placing it in the right location.

If we want to allow for third parties to distribute on their own, then we need to decide if we allow iTunes to install such applications, or publish the mechanism iTunes uses and let a third party developer equivalent support.

Development

The SDK itself will not be restricted. It will contain the headers and stub libraries and will represent the de facto public support for the embedded platform. If it's not in the SDK it is not supported.

The crucial problem to solve is allowing developers to run their code on their devices without having to get it signed by Apple every time they compile. Our goal is to restrict them to their own devices, so that they cannot simply build an app and post it on the web for everyone to use. In addition, we don't want to make the debug toolset easily available, as it represents a security threat.

~~Here is the basic p~~**Process for a third party to develop for the phone.**

The developer gets an identity from VeriSign or thawte. The intention of this identity is to have a way to tie requests back to a specific individual or entity.

The developer requests a development package from Apple.

Apple provides the developer with three things:

- The embedded SDK
- A development key used for signing his application
- A debug image associated with his identity

By default, his device will not accept applications signed with his development key. The device must be informed that his key is legitimate and can be trusted. At the same time, we want to ensure only his devices will accept applications signed with his key.

The developer requests a provisioning blob from Apple. His request contains the IDs of the devices he wishes to run applications signed with his development key. His request is signed with the private portion of his original identity.

The provisioning blob is embedded with three elements:

- The developer's public identity
- The public portion of his development key
- The set of IDs that can trust his development key.

This blob is then signed by Apple, and the device can trust it's contents. Once the developer puts this blob on his device, it will accept applications signed with his development key. In addition, it will accept the debug image that was also tied to his public identity.

At this point the developer is able to build and deploy his application to his own devices and debug. Once he has finalized his product, he submits it to Apple to be signed.

When Apple signs his GM application, it will now be accepted by any device and no longer needs a special provisioning blob.

This model provides two things. First, it allows Apple to prevent third parties from simply building and distributing their own applications on a wide scale. Secondly, it allows third parties to developer their applications independent of Apple until they're ready to deploy it to the world.

~~Lastly, this~~This model can be also be used by enterprise customers to deploy their own applications to their own set of devices. By applying a provisioning blob specific to the enterprise to their devices, they are able to restrict what devices can run their internal applications.

This model ~~does imply~~requires that Apple provides a service for creating provisioning blobs. However, by first requiring developers to present Apple with an identity, it allows us to constrain who gets blobs for a given identity.

The level of restriction is reduced to a policy question. How strong must the identity be before Apple will provide a development package and provisioning blobs?

Revocation

The main goal of signing all of the code that runs on a device is that it allows Apple to revoke specific applications later. It does not mean our system is secure, it simply raises the barrier. More importantly, it does not mean that once a bad application is discovered, Apple can remove them from customer's devices. Instead, it ~~simply~~ provides Apple with a way of preventing those applications from getting onto uninfected devices in the future.

Revocation implies that Apple maintains a service listing revoked applications. It is essentially a list of signatures or entire identities that can no longer be trusted.

What needs to be decided is how this is implemented. Our recommendation is that both iTunes and the installation mechanism on the device have the ability to retrieve the revocation list or query a service to determine if an application is valid. In addition, a nice to have feature would include the OS periodically asking Apple if the currently installed applications are legitimate, and inform the user if his device is infected.

The primary purpose of revocation is a deterrent for misbehavior. We hope not to have to employ it, but we must provide a mechanism to allow revocation to be able to inhibit the distribution of malware.

Apple Inc.

To Do/Open Issues

- Distribution policy for free applications (DRM, can 3rd parties be distributors of world signed Applications?)
- Legal Restrictions/Remedies on Applications
- SDK Sub-CA Acquisition and Parameterization (Must be exclusive with existing Sub-CAs) - Long lead time
- Write up on how Signing works (Countersigning, expiration)

Related Documents needed

- System Security Implications/Enforcement including Validation of a submitted Application
- Process and resources for validation team
- Open GL API - Security, Compatibility w/2.0, Completeness (Thread and only what we needed)
- ~~Write up on how Signing works (CodeSigning +? Sign Everything)~~

Apple Inc.

Appendix C - Developer Scenarios

Guy in his basement

1. Register with Apple and get SDK and Token for his 1 Unit
   1.1. Install Token into SDK
2. Connect device and invoke SDK
   2.1. SDK Signs build Application and downloads Disk Image and Token to Device
   2.2. Debugging/Development and Validation by the same individual
   2.3. Use device without Development image to validate
3. Decide you have final version to deploy
   3.1. Submit to Apple for Signing
   3.2. Get signed image and deploy as you wish

EA

1. Register with Apple and get SDK and Token for development and validation units
   1.1. Install Token into SDK
2. Connect device and invoke SDK
   2.1. SDK Signs build Application and downloads Disk Image and Token to Device
   2.2. Debugging/Development and Validation by the same individual
   2.3. Hand off signed development version
3. Decide you have final version to deploy
   3.1. Submit to Apple for Signing
   3.2. Get signed image and deploy (using a custom scheme, freeware or deal with Apple for DRM).

MIT

1. Register with Apple and get SDK and Token for development and validation units
   1.1. Install Token into SDK
2. Connect device and invoke SDK
   2.1. SDK Signs build Application and downloads Disk Image and Token to Device
   2.2. Debugging/Development and Validation by the same individual
   2.3. Most applications don't get widely distributed
3. Decide you have an application to deploy
   3.1. Submit to Apple for Signing
   3.2. Get signed image and deployment (using a custom scheme, freeware or deal with Apple for DRM).

Genentech

1. Set up a CA for signing applications
2. Register the public key of the CA with Apple and get it certified as an Application signer
3. Create a deployment package for enabling the CA as a signer for phones.
4. Create token for debugging with CA
5. Connect device and invoke SDK
   5.1. SDK Signs build Application and downloads Disk Image and Token to Device
   5.2. Debugging/Development and Validation by the same individual
   5.3. Hand off signed development version

Apple Inc.

6. Decide you have final version to deploy internally
   6.1. Sign with Signing server and deploy