Apple Inc.

Application Distribution and Deployment on MacOS X Embedded

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

October 14, 2007

Apple Inc.

The transition from a closed system to one with a more open developer model demands answers to questions of control and security. System integrity, core functionality and stability are maintained through careful control of interfaces, application deployment, and runtime execution. Methods for this control must be balanced against creating a reasonable environment for development and deployment of applications for the embedded

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

Assumptions

- Existence of a robust sandbox to contain applications and developers.

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 of the applications and development environment. This topic requires its own analysis and investigation.

Goals

Development

Foster development of applications on the device, by creative individuals, 3rd party ISVs and corporate entities.

Malware reduction

Active prevention and discouragement of malware is a critical part of our security plan. Application signing provides a process that enables legal protection and the identification method for application blacklisting. (Technical defenses primarily reside in the aforementioned sandboxing - to be covered in another document.) Examples include software that attempt to copy or transmit data off the device or tries to penetrate its security mechanisms. Restrict feature access

We intend to design a schema which allows us to partition system functionality into access groups to reserve certain portions of the stack for Apple use to satisfy carrier approval and control the user experience. A notable example is the software that allows access to the EDGE network.

Revenue from Apple distributed applications

Protect revenue for Apple and third parties who subscribe to Apple's distribution methods.

Independent application distribution

To enable enterprise application distribution we must enable application signing independent of Apple's primary signing and distribution service.

Distribution Methods

Distribution will be provided in two forms: Apple-signed and distributed and developer-signed and distributed. Developer-signed and distributed applications are currently slated for first deployment to enterprise operations where software will only be used internally (ie: not publicly). There is an open policy question around the models for freeware distribution .

iTunes (Apple signed)

Distribution through iTunes allows Apple to apply DRM to ensure non-transferability of purchased applications. Apple could also provide distribution without DRM through iTunes.

Apple Inc.

Mitch Adler October 13, 2007 8:06 PM
Apple Signed Only? Still investigating if we can delegate signing authority to developers who don't have other incentive to keep their signed applications restricted (e.g. EA)

Web (Apple signed)

Distribution through the web should be supported, but Apple will provide no transfer protection.

Internal (Developer signed)

Companies that wish to control distribution of their software can deploy packages internally. Again, Apple will provide no transfer protection, but will provide mechanisms for use only on devices controlled by the company's trusted IT department Application Development

Code signing can greatly slow the development process if injudiciously applied. Our system must be designed to ensure that the overhead of security does not hinder innovation. We are creating a system around three key phases - development, testing, and release. During development we must enable engineers to quickly iterate and test their code on Apple devices. During test phase, third parties will need to install software on a wider range of devices that are closer to production. Release refers to the final signed & distributed binary application. Appendix A contains a flow chart of 'a day in the life of a developer'.

Development

During development, engineers must have access to debug tools and their application. In order to enable development, we plan to tie a developer's identity and SDK to their development devices. This will allow them to iterate without going through an Apple-owned signing server, but restrict execution of their application to those development devices. It is critical that under-development software not be usable on shipping user devices. We also plan to make it impractical to convert an unauthorized (personal) system to a development system.

After agreeing to the terms of Apple's license/contract, the third party will receive a developer package. This package will contain the SDK, a debug image, and a development key. The debug image and the development key will be tied to that party. In order for a device to accept an application signed with that development key, the device must be provisioned for that unique party. This is the same mechanism we will use to support enterprise deployment.

We are crafting a system that allows Apple to control the number of devices a developer can use in this fashion.

Testing

We must design a system that allows wider deployment of under-development applications to closer-to-shipping devices for testing. We can employ the same methods outlined above, with a more stringent policy around development-enabled devices.

Enterprise customers will use their release mechanism for testing, as it is inherently restricted by design.

Release

As noted above, release will come from an Apple-signed and distributed channel, or a developer-signed and distributed channel (enterprise). The former requires an Apple department dedicated to review of software before signing. We plan to create tools to validate basic API-compliance. We should also create UI, power, and stability criteria for such signing. Whether this is developer-tested and submitted or Apple-tested is under consideration and will require significant resources to implement.

Apple-signed but developer-distributed software is still under policy consideration.

Apple will reserve the right to revoke an application's ability to run on devices for any reason, at any time.

Apple Inc.

Revocation

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.

The most important moment in the application lifetime to check for revocation is installation time, as at that moment the users device is yet to be effected by the revoked application.

Details of the mechanisms and policies around revocation need to be explored.

To Do/Open Issues

- System Security Implications
- Legal Restrictions/Remedies on Applications
- SDK Sub-CA Acquisition and Parameterization (Must be exclusive with existing Sub-CAs) - Long lead time
- Open GL API - Security, Compatibility w/2.0, Completeness (Thread and only what we needed)
- Write up on how Signing works (CodeSigning +? Sign Everything)
- Write up on Keys and Countersigning process for release, including issues around expiration

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