앱스토어별 코드사이닝 기법을 이용한 인증 방식과 보안성 분석

김지홍*, 김영훈*, 임헌정**, 이준호**, 정태명*
*성균관대학교 정보통신공학부
**성균관대학교 전자전기통신공학과
e-mail: enigma-kim@hanmail.net, ehdeoddl2@naver.com,
{hjlim99, jhlee83}@imtl.skku.ac.kr, tmchung@ece.skku.ac.kr

Analysis of each appstores' certification way by code signing and security

Ji-Hong Kim*, Young-Hun Kim*, Hun-Jung Lim**, Jun-Ho Lee**, and Tai-Myoung Chung*
*School of Information Communication Engineering, Sungkyunkwan Univ
**Dept of Electrical and Computer Engineering, Sungkyunkwan Univ

요 약

스마트폰 사용자의 폭발적 증가와 함께 모바일 애플리케이션 시장에 대한 관심도 크게 증가하고 있다. 더불어 애플리케이션의 보안 위협 역시 증가하고 있는데 각 모바일 애플리케이션 마켓에서는 이에 대한 대처방안 중 하나로 코드사이닝 기법을 활용하고 있다. 코드사이닝 기법을 이용하여 애플리케이션 및 개발자에 대해 인증을 받는 방식에는 각 모바일 마켓별로 다른 점이 존재한다. 그 종류로는 개발자가 생성한 공개키와 해당 애플리케이션을 애플에 전송하고 애플이 이에 대한 검증을 하여 마지막 서명을 부여하는 방식의 애플 앱스토어, 개발자가 키와 증명서를 이용해 스스로 애플리케이션에 대해 증명서를 발급하는 방식의 구글 안드로이드 마켓, RIM에 등록되어 있는 코드사이닝 키를 이용하여 애플리케이션을 제출하면 자동적으로 서명을 받을 수 있는 블랙베리 앱월드, 사전에 개발자가 자기 증명을 통해 아이디를 발급받고 애플리케이션을 테스트 하우스에 등록하여 검증 및 서명을 받는 노키아 오비스토어 등이 있다. 이와 같이 개발자 및 애플리케이션을 인증 받는 방식이 각 마켓별로 차이가 있는데 이에 대한 자세한 분석과 그에 따른 보안 위협에 대한 안정성에 대해 분석해 보겠다.

1. 서론

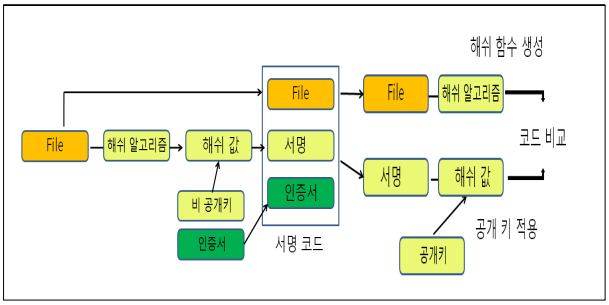
최근 스마트폰 시장이 크게 확장되면서 그에 따른 스마트폰 애플리케이션 마켓 역시 크게 확장되어 가고 있다. 현재 크게 대두되고 있는 마켓으로는 애플 앱스토어, 안드로이드 마켓이 대표적이고, 앱월드, T스토어, 오비스토어, 등이 존재한다. 미국 닐슨 리서치의 2010년 6월 보고서에 따르면 이 중에서 약 25만개의 앱을 보유한 애플이 경쟁에서 현저하게 앞서나가고 있는 상황이다.

초기의 앱스토어에서는 보안 위협에 대한 대응 능력을 충분히 갖추지 못했다. 따라서 애플리케이션을 다운 받는 과정에서 여러 보안 문제들이 발생하는 경우가 있었다. 이에 각 앱스토어들은 이러한 문제를 해결하는 방법 중 하나로 코드사이닝 기법을 활용하게 되었다. 본 논문에서는 각 앱스토어 별 코드사이닝 기법의 차이점을 살펴보고 그들의 장·단점 및 보안 위협에 대해 분석해 보고자 한다.

본 논문의 구성으로는 2장 관련연구에서 코드사이닝 기법을 통한 파일의 인증과 검증 방법에 대해 소개하고, 3장에서는 4개의 앱스토어 별 인증 방식을 분석한다. 4장에서는 4개의 앱스토어 별 인증 방식에 대한 비교와 요약이 다루어지며, 마지막 5장에서 결론으로 이루어진다.

2. 코드사이닝

코드사이닝은 특정 파일에 손실 또는 변형이 일어나지 않았음을 확인하는 기법중 하나이다. 아래의 (그림 1)은 코드사이닝 기법을 통한 인증 및 검증 작업을 일련의 흐름으로 나타낸 것이다.



(그림 1) 서명 및 검증 과정

파일의 인증은 크게 두 가지 경로로 이루어진다. 첫 번째 는 개발자 자신이 전용 툴을 이용해 인증을 하는 것이고 두 번째는 인증기관(CA)을 통해 인증을 받는 것이다. 이러한 인증 경로에 관계없이 코드사이닝 기법을 이용한 인증과 검증은 거의 동일한 방식으로 이루어진다.

인증 작업은 작업을 수행하는 측은 암호화를 위한 비

공개 키 값을 가지게 된다. 키 값을 기반으로 Secure Hash Algorithm(SHA), Message Digest Algorithm 5(MD5) 등의 알고리즘을 해쉬 함수로 이용하여 해쉬 값을 작성한다. 작성된 해쉬 값을 기반으로 서명이 이루어진다. 파일 내부에 서명 블록이란 공간을 만들어 서명과 인증 측의 인증서를 첨부하여 인증 작업이 완료 된다.

서명된 코드의 검증은 서명 코드 자체에 대한 무결성을 보장하기 위해 서명 블록 내에 첨부된 인증서에 대해 검증 작업을 실시한다. 인증기관을 통해 발급된 인증서라면, 상위 인증기관이 하위인증 기관에 발급한 인증서를 통해 검증 작업이 이루어진다. 인증서에 대한 검증 작업이 안료 되면 첨부된 서명 코드에 해쉬 알고리즘을 적용시켜 해쉬 값을 생성한다. 이를 인증서에 포함된 공개키를 적용하여 나온 값과 비교함으로서 검증 작업을 수행하며, 두 개의 값이 일치한다면 파일에 대한 무결성이 보장된다[4].

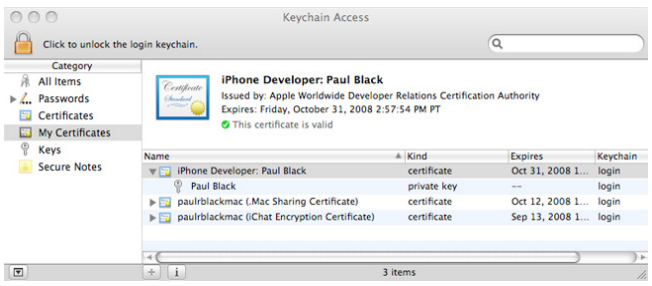
3. 앱스토어 별 코드사이닝 방식

3.1 애플 앱스토어

애플 앱스토어에서는 모든 아이폰 애플리케이션들이 애플에 등록된 아이폰 개발자들에게 발행한 서명 증명서와 함께 디지털 서명이 되어 있어야 한다. 이 서명은 그 애플리케이션의 개발자의 신원이 확실함을 보증하게 된다. 따라서 이는 그 애플리케이션이 수정되거나 변경되지 않았음을 확인시켜 줄 수 있다.

디지털 서명은 암호 키인 공개키와 비공개키를 요구한다. 공개키는 서명 증명서에 저장되어 있는데 이는 서명을 확인하는 데에 사용되고, 이와 분리되어 저장되어 있는 비공개키는 서명 절차에서 사용된다. 이 서명 증명서와, 그와 연관되어 있는 비공개키가 합쳐져서 한 개발자의 디지털 신원이 된다.

아이폰 애플리케이션 개발을 위한 서명 신분을 얻기 위해서는 Keychain Access 유틸리티에 있는 Certificate Assistant를 이용하여 Certificate Signing Request(CSR)을 생성해야 한다. 이 CSR은 프로그램 포털을 이용할 때 신분을 승인받기 위해 제출해야 한다. CSR을 생성하기 위하여 Certificate Assistant 유틸리티를 사용하면 이것이 자동으로 공개키와 비공개키를 생성한다. 여기서 생성된 증명서 안의 공개키가 애플에게 전송되고 주어진 keychain에 비공개키가 저장된다. 아래의 그림(2)는 Keychain Access 유틸리티를 이용하여 증명서를 생성하는 장면을 나타낸 그림이다.



(그림 2) Keychain Access를 이용한 증명서 생성

Keychain Access 유틸리티를 이용하여 아이폰 개발자 Paul Black이 비공개키를 생성하여 keychain에 저장한 그림이다. 이 과정까지가 완료된 개발자가 개발한 애플리케이션을 사용자가 설치하게 되면, 사용자의 아이폰은 또 다른 확인 작업을 하게 된다. 아이폰 운영체제가 그 애플리케이션의 서명을 확인하여 해당 애플리케이션이 애플에 의해 서명 받은 것인지, 서명을 받은 후에 다른 프로그램으로 대체되지는 않았는지 확인한다. 이 과정에서 그 서명이 유효하지 않거나 애플에 의해 서명 받은 것이 아니라면 아이폰 운영체제는 그 애플리케이션이 작동하지 못하도록 막는다.

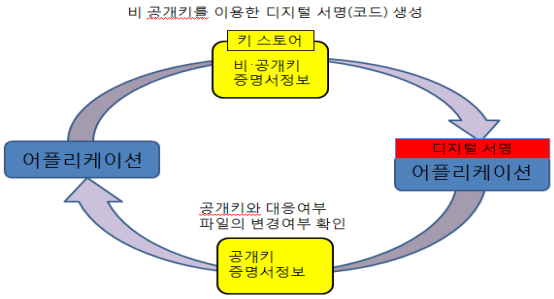
따라서 개발자가 개발한 애플리케이션을 애플에 승인받고 배포하기 위해서는 서명 증명서를 애플리케이션과 함께(단 비공개키는 제외) 보내야 한다. 그러면 애플에서는 그 서명이 등록된 개발자에게서 보내진 것인지, 오류가 포함된 것인지를 확인한다. 이 확인 과정이 모두 끝나면 애플에서 해당 애플리케이션에 포함된 서명 증명서에 마지막 서명을 끝내게 된다.

위와 같은 애플의 모든 정책은 아이폰이나 아이팟 터치 등의 장비를 가진 사용자들에게 그들이 다운로드 받을 수 있는 애플리케이션들이 확실히 등록된 개발자들에 의해 개발된 것이며, 개발 후에 또 다른 오류 등으로 변질된 것이 아님을 알게 해줘 그들의 장비에 안전성을 가져다 줄 수 있다.

3.2 구글 안드로이드 마켓

안드로이드 운영체제에서 애플리케이션을 다운 받고 설치하기 위해서는 해당 애플리케이션이 인증 작업을 통해 서명된 애플리케이션 이어야 한다. 그렇지 않다면 운영체제는 애플리케이션의 설치를 거부하게 된다.

애플리케이션 개발자는 프로그램 개발 후, 마켓에 등록하여 일반 사용자에게 공개하기 위해 Publishing 이라는 단계를 거친다. 이 단계에서 애플리케이션에 대한 인증 작업이 이루어지며 개발자는 인증을 위해 자기 서명 증명서(self-signed certificates)를 발급하게 된다. 아래의 (그림 3)은 자기서명 증명서의 발급 과정과 인증 방식을 간략히 나타낸 것이다.



(그림 3) 안드로이드 인증 및 검증 방식

자기 서명 증명서의 발급에는 일반적으로 Keytool과 Jarsigner라는 두 개의 툴이 사용된다. 대략적인 흐름은

일반적인 코드사이닝을 통한 인증 방식과 같이 이루어진다. 개발자는 서명 작업을 위해 공개키와 비공개키를 가지며, Keytool을 통해 비공개키와 그에 관련된 증명서들이 저장된 데이터베이스를 작성한다. 마지막으로 Jarsigner를 이용해 키스토어에 저장된 키와 증명서 정보를 이용해 디지털 서명을 생성한다.

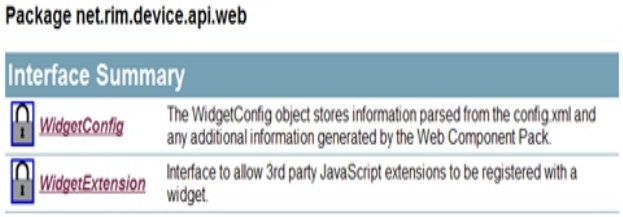
생성된 디지털 서명의 검증작업은 다음과 같다. 서명 생성에 사용된 비공개키가 공개키와 대응되는지를 먼저 검증한 뒤, 애플리케이션 내부의 파일을 비교분석하여 데이터의 변경이 이루어 지지 않았음을 검증한다.

이러한 코드사이닝 방식은 다음과 같은 특징을 가진다. 안드로이드 마켓에서 제공되는 애플리케이션에 대해 개발자가 작성한 비공개키가 노출되지 않는 한, 애플리케이션 다운로드 과정에서 발생하는 악의적 또는 우연적 데이터 손실·변경 등에서 안전하지만 개발자 스스로가 프로그램에 악의적 코드를 삽입하는 경우에는 보안 위협에 대한 대처 방안이 되지 못한다.

3.3 블랙베리 앱월드

RIM(Research In Motion)은 보안과 전송 컨트롤 상의 이유로 블랙베리 APIs(Application Program Interfaces)사용을 추적한다. 따라서 블랙베리 애플리케이션을 컨트롤하기 위해서는 그 애플리케이션의 .cod파일이 블랙베리 스마트폰에 로드되기 전에 RIM에서 제공하는 서명키를 이용하여 서명이 되어야 한다.

서명을 요구하는 APIs는 블랙베리 애플리케이션들이 컴파일 되는 과정에서 이용된다. 이 APIs는 자물쇠 아이콘으로 나타내지거나 "signed"라고 표시되어 있다. API가 자물쇠 아이콘으로 표시된 그림은 다음과 같다.



(그림 4) 블랙베리에 나타나 있는 APIs

앱월드에서 애플리케이션을 서명받기 위해서는 코드사이닝 키와 명령어들을 장비에 사용될 수 있도록 설정하여 저장해야 한다. 이 코드사이닝 키는 RIM에 등록되어 있는데 이는 개발자들에게 고유하게 할당된다. 그 코드사이닝 키를 받기 위해서 RIM은 개발자의 신원을 확인하는 간단한 등록과정을 거친다. 개발자가 등록 양식을 작성하여 블랙베리에 보내면 RIM에게 키가 전달되고 RIM이 개발자에게 다시 그 키를 보낸다. 이러한 키가 모든 애플리케이션을 서명받기 위해 사용된다.

애플리케이션을 서명받기 위해서는 Signature Tool을 이용해야 한다. 서명이 필요한 목록들을 보여주는 Signature Tool은 단순히 RIM에게 애플리케이션을 제출하고 RIM은 자동적으로 필요한 서명을 애플리케이션에 첨부하여 되돌려준다. 이 과정이 끝나면 애플리케이션이 로드될 수 있다.

하지만 이런 API를 이용하는 방법은 보안적인 결함이 있다. 예를 들어, 블랙베리 API는 개발자가 최종 사용자에게 보안 위험이 될 수 있는 사용자의 개인적 정보(PIM)에 접근하는 것을 허용함으로 최종 사용자에게 잠재적인 보안 위협이 있는 정보를 제공한다. 이러한 위협 요소들을 조절하기 위해서 기존 API가 아닌 Controlled API를 이용하는 장비가 늘어나고 있다. Controlled API는 적합하게 서명되지 않은 애플리케이션이 블랙베리 장비에서 작동되지 않도록 제어 할 수 있다.

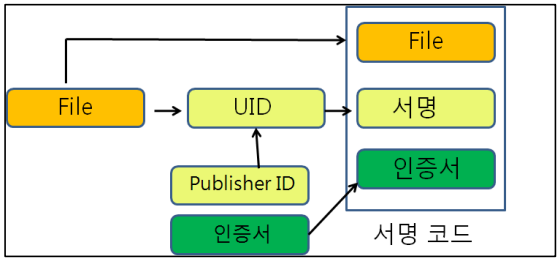
3.4 노키아 오비 스토어

노키아의 심비안 OS는 2001년 처음 공개된 이후로 현재까지 지속적으로 업데이트 되어왔다. 아래의 <표 1>은 다양한 버전의 심비안 OS에서 사용되는 애플리케이션과 그에 따른 인증방식을 정리한 것이다.

<표 1> 오비 스토어의 다양한 규격

애플리케이션	인증방식	인증형태
Qt App for symbian Sybbian C++ Symbian Themes Flash lite for Symbian	Symbian Signed	SIS SISX
S60 WRT Series 40 Themes Series 40 Flash lite Qt App for Maemo5	Ni signing	WGZ NTH NFL DEB
Jave ME	Jave Verified Java code signing certificates	JAD JAR

심비안 마켓에서는 애플리케이션의 종류에 따라 인증을 받아야 하거나 별도의 인증이 필요 없는 경우도 있으며 일부는 자바의 코드사이닝을 통한 인증을 받기도 한다. 여기서는 보안성을 대폭 강화하여 나온 최신 버전의 심비안 OS v9에서 사용하는 인증 방식에 대하여 분석한다. 심비안 OS v9에서 개발자가 애플리케이션에 대해 이루어지는 인증방식은 (그림 5)에서 간략하게 알 수 있다.



(그림 5) 오비스토어 인증방식

개발자는 애플리케이션 인증을 위해 먼저 Publisher ID를 취득해야 한다. 코드사이닝 방식을 이용한 인증의 암호화를 위해서는 키 값이 필요로 하는데, 취득한 Publisher ID를 통해서만 키 값의 생성이 가능하다. 이렇게 생성된 키 값을 이용해 애플리케이션 서명이 이루어진

다. 서명이 된 애플리케이션은 오비스토어의 테스트하우스에 제출되며, 모든 테스트를 정상적으로 통과 하게 되면 정식으로 인증이 되고 오비스토어에 등록된다.

이 때 애플리케이션 서명을 통한 인증에 사용되는 것은 UIDs(Unique Identifier)와 인증서이다. UIDs란 코드사이닝에서 최종적으로 생성된 코드 값을 뜻한다. UID는 0x00000000 에서 0xFFFFFFFF 까지의 32비트 범위를 갖는 문자열로서 심비안 OS파일의 처음 12바이트 공간 내에 3개가 저장되며, 애플리케이션을 식별하거나 데이터의 변조를 방지하는데 사용된다. 인증서는 일반적인 코드사이닝 기법을 통한 인증방식과 동일하게 사용된다.

오비 스토어에서 개발자가 Publisher ID를 얻기 위해서는 특정 조직의 소속여부, 신용카드의 소지 등을 알릴 필요가 있다. 애플리케이션에 대해서도 오비 스토어 자체적으로 테스트를 진행한다. 이러한 인증 방식은 개발자와 애플리케이션에 대해 일정 수준 이상의 보안 능력을 가진다.

4. 결론 및 비교분석

아래의 <표 2>는 각 앱스토어별로 개발자 및 애플리케이션의 검증 여부와 서명 형태를 간략히 정리한 것이다.

<표 2> 비교 분석

	애플 앱스토어	오비 스토어	앱월드	안드로이드 마켓
개발자검증 여부	O	O	X	X
App 검증 여부	O	O	X	X
App 서명 형태	마켓 서명	마켓 서명	마켓 서명	개발자 서명

애플 앱스토어와 오비스토어의 경우 개발자와 애플리케이션 모두 검증 작업을 거치며, 마켓에서 직접 인증을 내린다.

안드로이드 마켓의 경우 개발자와 애플리케이션 모두에 대해 별다른 검증 작업이 이루어지지 않으며 서명 역시 개발자 스스로가 하게 된다.

앱월드는 안드로이드 마켓과 유사하나 애플리케이션에 대한 서명은 마켓 자체에서 낸다.

이러한 차이는 각종 보안 위협에 대한 대응가능성에 대한 차이로 이어진다. 아래의 표는 몇 가지 보안 위협에 대한 대응 가능 여부를 각 마켓별로 구분한 것이다.

<표 3> 보안 수준

	애플 앱스토어	오비 스토어	앱월드	안드로이드 마켓
악성코드	O	O	X	X
정보유출	△	△	X	X
금전적손실	△	△	X	X
공격지활용	X	X	X	X
보안 수준	높음	높음	낮음	낮음

애플 앱스토어 및 오비스토어는 개발자 및 애플리케이션에 대한 검증 작업을 수행하여 애플리케이션 차원에서 발생하는 각종 보안 위협을 막을 수 있다. 반대로 앱월드, 안드로이드 마켓은 애플리케이션 차원에서 그러한 위협에 대응할 수 없다.

결과적으로 코드사이닝 기법을 이용한 애플리케이션 인증 부분에서는 애플 앱스토어와 오비 스토어가 더 높은 보안성을 갖추고 있다고 할 수 있다.

5. 참고 문헌

[1] 김정훈, “구글의 안드로이드와 안드로이드 마켓”, 한국콘텐츠학회, 2009.
[2] 강동호, 한진희, “스마트폰 보안 위협 및 대응 기술”, 전자통신동향분석 제2권 제4호, 2010. 6.
[3] 이기혁, “Mobile Security 현황과 통신사업장 대응방안”, 정보보호심포지움, 2009.
[4] 유혜정, “사용자 소프트웨어의 안전한 배포를 위한 Code Signing기술 연구”, 한국정보보호진흥원, 2003.
[5] 노병규, “오픈마켓에서 WAC지원을 위한 정책제안”, Internet and Infomation Security 제1권 제1호, 2010. 5.
[7] Jesse Burns, “Developing secure mobile application for android”, iSec Partners, 2008.
[8] 노키아 포럼, http://www.forum.nokia.com
[9] Symbian Developer, http://developer.symbian.org/
[10] Symbian, http://www.symbian.org/
[11] Java SE6 API 문서, http://xrath.com/java-api-docs-ko/
[12] Andriod Developers. http://developer.android.com/
[13] iAPP 한국개발자 그룹, http://cafe.naver.com/iappdev.cafe/
[14] Joyholic, http://joyholic.kr/314/
[15] Blackberry developers, http://na.blackberry.com/eng/developers/javaappdev/
[16] Developerlife, http://developerlife.com/
[17] Programming Blackberry, http://berrytutorials.blogspot.com/2009/10/code-signing-setting-up-eclipse-to-use.html