안드로이드 코드서명의 취약점을 이용한 악성 앱에 관한 연구1)

박경용*, 조태남*
*우석대학교 정보보안학과
e-mail:1aze@naver.com, tncho@ws.ac.kr

A Study on Malicious App using Vulnerability of Android Code-Signing

GyeongYong Park*, Taenam Cho*
*Dept of Information Security, Woosuk University

요 약

스마트 폰의 보급량이 증가함에 따라 모바일 악성코드의 위협도 높아졌다. 여러 스마트 폰 플랫폼 중 안드로이드 플랫폼은 높은 점유율과 개방형 플랫폼이라는 특성상 다른 플랫폼에 비해 악의적인 공격에 취약하다. 안드로이드 앱이 스마트 폰에 설치, 실행되기 위해서는 개발자의 서명이 요구된다. 안드로이드 서명체계는 다중 서명을 허용하는데, 다중서명 체계상 악용될 수 있는 취약점이 존재한다. 본 연구에서는 안드로이드 코드서명의 취약점을 이용하여 악성코드를 실행시키고 다른 앱을 감염시키는 악성 앱을 개발하여 취약점의 악용 가능성에 대해 연구하였다.

1. 서론

최근 조사에 따르면 세계 스마트 폰 보급량이 14억대에 육박하는 것으로 조사되었다. 여러 플랫폼 중 안드로이드 플랫폼은 점유율이 57%에 달할 정도로 많이 사용되고 있다[1]. 안드로이드 플랫폼은 개방형 플랫폼이라는 특성상 다른 플랫폼에 비해 악의적인 공격에 취약하다. 플랫폼의 취약점이 발견된 뒤 이를 막을 수 있는 패치가 발표되기도 전에 수많은 악성 앱이 배포된다. 이러한 악성 앱들은 사용자 정보를 수집하기 위해 설계되었으며, 스마트 폰에서 사용자의 데이터 및 개인정보를 수집한다[2].

안드로이드 플랫폼에서 앱의 설치와 실행을 위해서는 앱에 대한 개발자의 코드서명이 필요하다. 그러나 이를 통해 개발, 유통 단계에서 앱의 변조여부를 확인하기 어렵다. 이러한 취약점을 이용하여 악성코드를 배포하는 방법으로 리패키징(repackaging) 기법이 있다. 공격자가 유명한 앱에 악성코드를 삽입한 후 공격자 자신의 키로 서명을 하여 배포해도 사용자들은 변조된 앱과 원래의 앱을 구분하기 어렵기 때문에 의심 없이 사용하게 된다. 또한 다중 서명을 허용함으로 인해 발생하는 취약점이 존재한다. 다중 서명이 된 앱에 정상적인 서명이 하나라도 포함되어 있다면, 만료된 서명, 잘못된 서명, 가짜 서명 등이 포함되어 있어도 검증이 되어 앱의 설치와 실행에 문제가 없다.

본 논문에서는 안드로이드 다중 코드서명 체계의 취약점을 이용한 악성 앱을 개발하였다. 기존 리패키징 기법을 이용한 방법과는 달리 서명 디렉터리에 서명을 위장한 악성코드를 추가시키기만 하면 감염된 앱을 만들 수 있다. 배포된 감염된 앱은 추후 컨트롤 앱을 통하여 악성코드가 실행될 수 있다. 즉, 다중서명 체계의 취약점은 새로운 방식의 악성코드 개발 및 배포에 악용될 수 있음을 보였다.

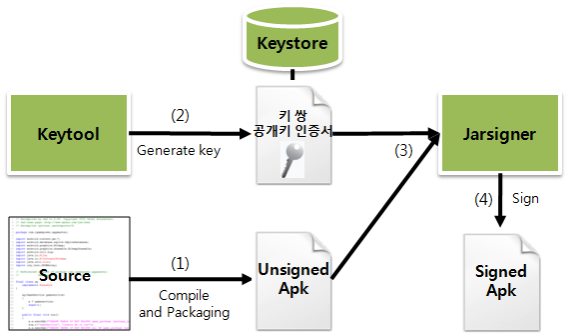
2. 안드로이드 코드서명 체계

2.1 코드서명 과정

안드로이드 시스템은 앱에 대한 개발자의 서명을 요구하며, 서명이 없이는 설치가 불가능하다. 안드로이드에서 서명은 앱의 저자 식별, 모듈화, 업그레이드, 코드 및 데이터 공유를 목적으로 하며 앱의 통제목적으로 사용되지는 않는다[3]. 서명에 사용되는 인증서는 개발자가 JDK의 Keytool을 이용하여 직접 생성한 인증서가 사용된다. 앱에 대한 서명과정은 다음과 같다(그림 1 참조).

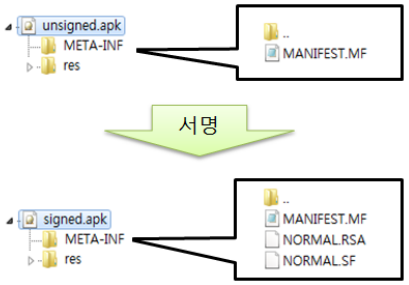
(1) 프로그램 소스 작성 후 compile과 packaging을 통해 apk 생성
(2) JDK에 포함된 Keytool을 이용하여 코드서명용 키 쌍과 공개키 인증서 생성하고 Keystore에 저장
(3) Jarsigner를 이용하여 Keystore에 저장된 개인키로 apk에 서명
(4) 서명 파일이 추가된 apk 생성 완료

1) 이 논문은 2012년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. 2012R1A1A3015030).

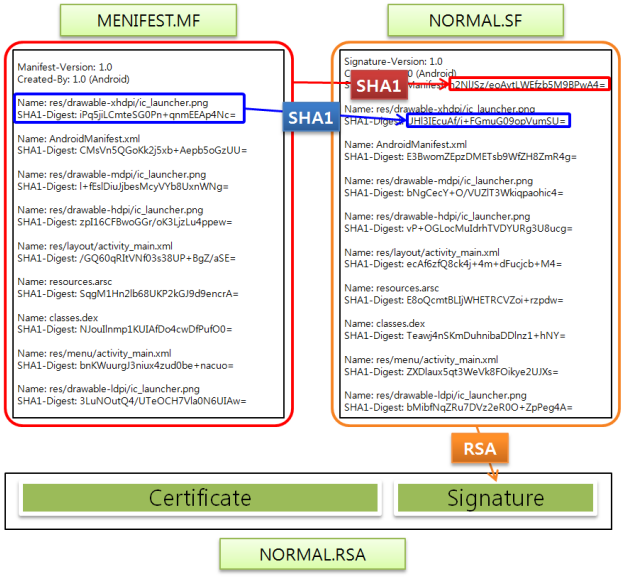


(그림 1) 앱 서명과정

서명이 완료되면 apk파일의 META-INF 디렉터리에 서명 파일들이 추가된다. 서명파일의 파일명은 키 생성 당시에 키에 부여한 alias로 지정되며 Jarsigner의 옵션을 통해 다른 이름으로도 지정이 가능하다. (그림 2)는 서명되기 전 apk의 META-INF 디렉터리와 alias가 NORMAL인 키로 서명한 후의 META-INF 디렉터리를 보여준다. 서명을 하면, 서명 파일인 NORMAL.RSA와 NORMAL.SF가 생성된 것을 볼 수 있다.



(그림 2) 서명 전, 후 apk 파일 구조

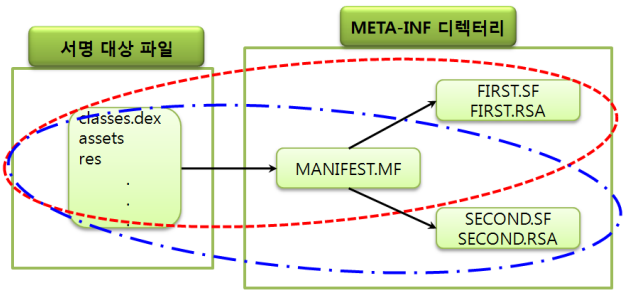


(그림 3) 서명파일 구조

MANIFEST.MF에는 apk에 포함된 각 파일명과 해당 파일의 SHA1 해쉬 값들이 저장되어있다. NORMAL.SF는 MANIFEST.MF에 포함된 각 해쉬 정보를 항목별로 다시 해쉬한 값들을 저장한다. NORMAL.RSA는 서명검증에 필요한 공개키 인증서와 NORMAL.SF에 대해 서명한 값이 포함된다(그림 3).

2.2 이중 서명

Jarsigner는 다중서명을 허용한다. 생성되는 서명파일명만 다르다면 서명을 여러 번 할 수 있다. (그림 4)는 alias가 FIRST인 키로 1차 서명한 앱을 다시 alias가 SECOND인 키로 2차 서명한 경우를 보여준다. 그림에서 보는 바와 같이 각 서명파일은 서로 독립적으로 생성된다. 2차 서명 파일은 1차 서명파일을 포함하지 않는다. 즉, 서명당시 META-INF 디렉터리에 이미 존재하는 서명 파일들을 서명대상에 포함시키지 않는다. 따라서 1차 서명을 2차 서명에서 포함하지 않음으로 인해 1차 서명에 대한 무결성을 제공하지 못한다.



(그림 4) 첫 번째 서명과 두 번째 서명의 관계

(그림 4)처럼 서로 연관성이 없는 서명파일간의 관계 때문에 취약점이 존재한다. 정상적인 서명이 된 앱에 만료된 키로 서명을 해도 설치와 실행에 문제가 없다. 이와 마찬가지로 다른 앱의 서명을 포함시키거나, 이미지 파일의 확장자를 변조하여 만든 가짜서명을 포함시켜도 정상적으로 설치되고 실행된다. 이 취약점은 악성코드가 서명파일로 위장되어 삽입되어도 서명 검증에 성공할 수 있음을 말해준다.

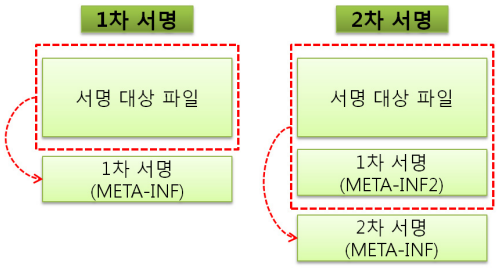
3. 관련 연구

본 연구진은 위의 취약점에 대한 보안책으로 이중서명 기법을 제안한 바 있다[4].

3.1 서명 알고리즘

먼저 기존의 서명이 존재하는지 점검한다. 1차 서명의 경우, 기존의 서명방식과 동일한 과정을 거친다. 만약 이미 서명이 존재할 경우, 1차 서명 파일들을 새로운 META-INF2 디렉터리로 이동함으로써 2차 서명 대상 파일로 만들어 준다. 1차 서명파일이 포함된 META-INF2를 포함한 전체 파일에 대한 2차 서명은 META-INF에 생성된

다(그림 5). 이 서명 방식은 3차 이상의 서명에도 일관되게 적용할 수 있다.



(그림 5) 서명 흐름

3.2 검증 알고리즘

서명 파일을 위장한 악성코드의 삽입을 방지하기 위하여 서명 파일 구조에 엄격한 제한사항을 추가하였다. 각 서명 폴더에는 하나의 서명만이 존재해야 하고, 각 서명은 유효한 서명이여야 하며, 확장자 변조파일 등 서명 형식에 맞지 않는 파일이 있어서는 안 된다. 먼저, 기존 방식처럼 META-INF 디렉터리의 서명파일들을 가지고 검증을 진행한다. 검증에 성공하면 META-INF2 디렉터리의 존재 여부를 통해 2차 서명이 되어있는가를 확인한다. 2차 서명이 된 앱의 경우, META-INF2 디렉터리에 존재하는 1차 서명파일로 검증을 한다. 이 때 META-INF 디렉터리의 파일들은 1차 서명 이후에 생성된 파일들이기 때문에 검증대상에서 제외된다.

4. 리패키징 악성 앱

안드로이드 악성 앱은 대부분 트로이목마형이다. 트로이목마는 정상적인 앱을 위장하여 사용자가 의심 없이 설치하도록 방심하게 만들고, 내부적으로 악성코드를 실행하여 피해를 유발시킨다. 공격자는 일반적으로 유명한 앱을 리패키징 하여 악성코드를 삽입한다. 리패키징이란 앱을 unpackaging하고 역컴파일(decompile)을 통해 얻어낸 소스를 수정하거나 다른 코드를 삽입하고, 다시 컴파일(compile)하여 패키징(packaging)하고 서명을 하는 과정이다. 다시 서명을 하는 이유는 소스가 수정되었기 때문에 기존 서명으로는 검증에 성공할 수 없기 때문이다.

사회 공학적 기법인 리패키징을 이용한 악성 앱 중 악명 높았던 앱 중의 하나인 Foncy는 “MADDEN NFL 12”라는 게임 앱으로 위장한 악성 앱이다[5]. 설치 후 실행하게 되면 사용자 모르게 강제로 스마트 폰을 루팅한다. 그리고 IRC 채널에 접속하여 원격 제어가 가능한 봇(Bot) 기능을 수행하고, SMS를 통해 과금을 발생시키는 악성 앱을 몰래 설치하고 백그라운드로 실행시킨다[6].

5. 이중 서명의 취약점을 이용한 악성 앱의 구현

본 논문에서는 전술한 이중 서명의 취약점을 이용한 악성 앱을 구현함으로써 새로운 공격 유형을 시사하였다. 공격자의 악성 앱 배포 및 실행 단계는 다음과 같다.

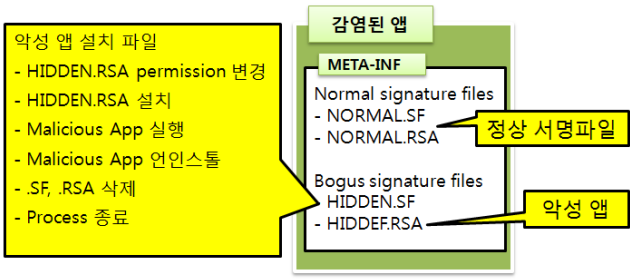
5.1 악성 앱 작성 단계

서명 파일은 .SF 파일과 .RSA 파일 쌍으로 이루어지므로, 공격자는 악성 앱을 서명 파일로 위장하기 위하여 2개의 가짜 서명 파일을 준비한다(그림 6 참조).

- HIDDEN.RSA: 공격자가 원하는 기능인 사용자 정보 유출이나 과금 등을 유발하는 악성 앱이다. 본 실험에서는 백그라운드로 동작하며, Toast 메시지로 자신이 실행 중임을 알리는 기능을 수행하는 앱으로 작성하였다.
- HIDDEN.SF: HIDDEN.RSA를 설치 및 실행시키고 이를 위해 필요한 작업을 수행하는 실행 파일이다. HIDDEN.RSA의 퍼미션을 변경하여 설치하고 실행시키며, 실행 종료 후 앱을 언인스톨(uninstall)하고 파일을 삭제한다.

5.2 희생 앱 감염 단계

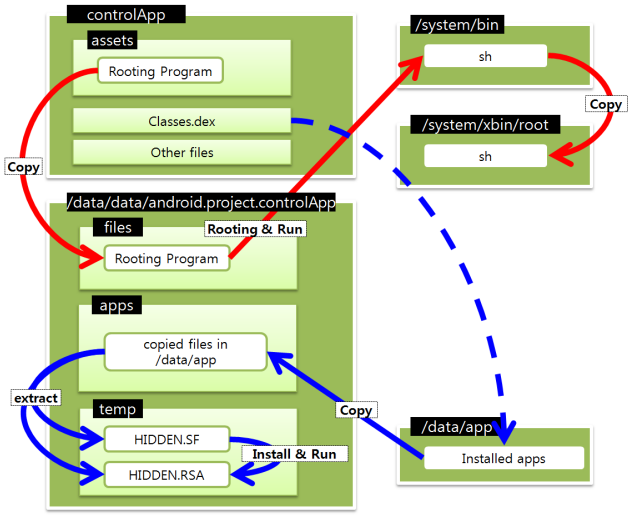
생성한 악성 앱으로 감염시킬 희생 앱을 다운로드하여 위에서 생성한 가짜 서명 파일을 META-INF에 삽입한다(그림 6 참조). 2.2절에서 기술한 바와 같이 희생 앱에 대한 서명 검증 시, 삽입한 파일과 상관없이 기존 서명의 검증은 성공할 것이며 가짜 서명 검증에 실패하더라도 아무런 문제없이 스마트 폰에 설치 및 실행이 될 것이다.



(그림 6) 감염된 앱의 구조

5.3 컨트롤 앱 작성 단계

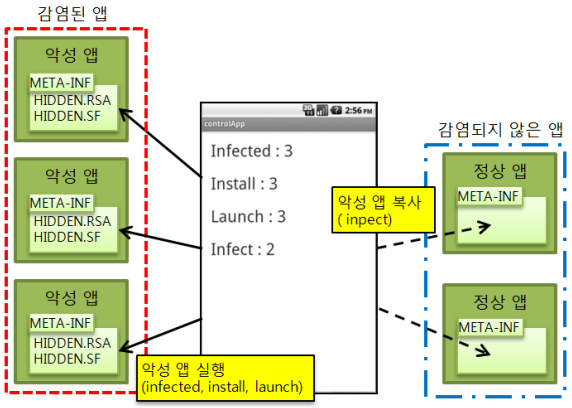
공격자는 희생 앱에 숨어 있는 악성 앱을 실행시키기 위해 트로이목마형 컨트롤 앱을 작성한다. 이 앱을 실행시키면, 사용자 동의없이 스마트 폰을 루팅하여 루트 권한을 획득한 뒤, 스마트 폰에 설치되어 있는 감염된 앱 즉, META-INF에 HIDDEN.SF와 HIDDEN.RSA가 존재하는 앱을 찾아 HIDDEN.SF를 실행시킨다(그림 7 참조). 또한 감염되지 않은 앱에 HIDDEN.SF와 HIDDEN.RSA를 복사하여 감염시키도록 할 수도 있다. 본 실험에서는 진저브레드(Gingerbread)의 vold volume manager 취약점을 이용한 [7] 진저브레이크(Gingerbreak)를 이용하여 루팅하였으며, 감염된 앱을 모두 찾아 실행시키고 감염된 앱의 실행 횟수를 화면에 출력하도록 작성하였다.



(그림 7) 컨트롤 앱의 기능

5.4 컨트롤 앱 및 감염 앱의 실행 단계

본 연구에서 개발한 악성 앱은 그 기능이 2개의 앱에 분산되어 있는 이원화된 구조를 가지고 있다. 감염된 앱이 일종의 보균자 혹은 숙주 역할을 하고 컨트롤 앱이 트리거(trigger) 역할을 한다. 감염된 앱과 컨트롤 앱이 스마트 폰에 설치되어 있고, 컨트롤 앱이 실행되면 5.3절에서 기술한 바와 같이 감염된 앱들에 숨어 있는 악성 앱들이 실행된다. 악성 앱은 사용자가 인식하지 못하도록 백그라운드로 실행되며 실행 후 언인스톨된다. (그림 8)은 실험 결과로서, 스마트 폰에 3개의 앱을 감염시킨 상태에서 컨트롤 앱에서 이 앱들을 찾아 실행시키고, 감염되지 않은 11개의 앱에 악성 앱을 삽입하여 감염시킨 것을 보여주는 결과 화면이다.



(그림 8) 컨트롤 앱 실행

6. 결론 및 향후 연구과제

안드로이드 플랫폼에서는 서명이 검증되어야 설치 및 실행이 되며, 한 앱에 대한 다중 서명을 허용한다. 그러나 각 서명은 서로 연관관계가 없이 독립적으로 생성 및 검증되기 때문에 취약점이 발생한다. 서명 디렉터리에 정상적인 서명이 하나만 존재해도 만료된 서명, 잘못된 서명, 가짜 서명 등을 무시하고 설치되고 실행된다. 다른 서명 파일들을 서명 대상으로 포함하지 않기 때문에 발생하는 취약점이다.

본 연구에서는 이러한 취약점을 이용한 악성 앱을 구현하였다. 악성 앱을 작성하여 정상적인 앱의 가짜 서명 파일로 추가시킴으로써 복잡한 리패키징 단계없이 감염시킨다. 컨트롤 앱은 스마트 폰에 설치된 감염 앱을 찾아내어 그 안에 포함된 악성 앱(가짜 서명파일)을 설치하고 실행시킨다. 뿐만 아니라 이 앱은 감염되지 않은 다른 앱에 가짜 서명파일을 삽입함으로써 스마트 폰 내에서 악성 앱을 간단히 전파시킬 수 있다. 본 악성 앱은 기존 리패키징 악성 앱과 달리 원래 앱의 복잡한 분석과 코드삽입 및 재서명의 단계 없이 앱을 감염시킬 수 있다. 또한 별도의 컨트롤 앱을 통해 감염된 모든 앱을 실행시킬 수 있는 새로운 유형의 악성 앱이다. 따라서 이중 코드서명의 취약점은 반드시 보완되어야 할 문제점이다.

추후, 이중 코드서명의 취약점을 이용한 악성 앱을 차단할 수 있는 서명 검증 방안을 구현하여 안드로이드 플랫폼에 적용할 계획이다.

참고문헌

[1] ABI Research, "45 Million Windows Phone and 20 Million BlackBerry 10 Smartphones in Active Use at Year-end; Enough to Keep Developers Interested", January 2013.

[2] Trend Micro, "Android Under Siege: Popularity Comes at a Price", October 2012.

[3] http://developer.android.com/tools/publishing/app-signing.html.

[4] 박경용, 위성근, 조태남, 서승현, "안드로이드 이중 코드서명 체계 연구", 제38회 한국정보처리학회 추계학술발표대회 논문집 제19권 제2호, November 2012.

[5] Total Defence, "Total Defense 2011 Threat Report: An Open Door to Malware", 2011.

[6] http://blog.ahnlab.com/ahnlab/1437.

[7] http://www.ahnlab.com/kr/site/securitycenter/asec/asecView.do?groupCode=VNI001&webNewsInfoUnionVo.seq=18528.