# 안드로이드 리패키징(Repackaging) 탐지 기술 설계 및 구현

박종섭*, 박상호*, 박찬암*, 이종호*, 신동휘**
*루멘소프트 보안기술연구팀, **성균관대학교 정보보호연구실
email : jspark@lumensoft.co.kr*, shpark0101@lumensoft.co.kr*,
cpark@lumensoft.co.kr*, jhlee@lumensoft.co.kr*, dhshin@security.re.kr**

# Design and Implementation of Android Repackaging Detection Technique

Jong-seop Park*, Sang-ho Park*, Chanam Park*, Jong-ho Lee*,
Donghwi Shin**
*Lumensoft Advanced Security Research Team,
**Information Security Group, Sungkyunkwn University

요 약

스마트폰 사용이 급증하고 있는 현재, 안드로이드 OS 기반 스마트폰 점유율이 가장 큰 상승세를 보이고 있다. 하지만 안드로이드 OS는 자가-서명 인증서(self-signed certificate)로 애플리케이션을 검증하여, 많은 보안상의 취약점을 내재하고 있다. 자가-서명 인증서의 검증 취약점을 이용하여, 악의적인 공격자는 기존 정상 애플리케이션에 악성코드를 삽입, 리패키징(Repackaging) 하여 마켓에 유포할 수 있다. 이러한 문제를 해결하기 위해서, 본 논문에서는 안드로이드 애플리케이션의 서명 파일을 이용한 애플리케이션 리패키징 여부를 탐지하는 기술을 설계 및 구현한다.

## 1. 서론

최근 컴퓨터 기능이 내장된 휴대폰인 스마트폰은 전 세계적으로 그 사용자가 급증하여, 필수 휴대품이 되어 가고 있다. 많은 스마트폰 OS가 있지만, 단연 안드로이드 OS의 점유율이 가장 큰 상승세를 보이고 있다. 안드로이드 OS는 무료로 사용할 수 있는 개방 OS라는 강점으로 점유율은 계속 상승할 것으로 예상되고 있다.

안드로이드는 오픈 마켓으로 운영하고 있으며, 개발자 등록비만 결제하게 되면 별다른 보안 검증 없이 자유롭게 애플리케이션을 등록 할 수 있다. 안드로이드 애플리케이션의 설치는 마켓에서 다운받아 유료 혹은 무료로 설치할 수 있다. 또한 안드로이드는 마켓을 통하지 않고, 개인 PC를 통해 애플리케이션 설치도 가능하다. 설치 시 안드로이드는 자가-서명 인증서로 애플리케이션을 검증한다. 악의적인 공격자는 자가-서명 인증서의 취약점을 이용하여 정상적인 애플리케이션을 다운로드 받아, 악성코드를 삽입하고 리패키징 하여 안드로이드 마켓에 유포할 수 있다.

따라서, 본 논문에서는 안드로이드 OS에서 악성코드 유포로 가장 많이 사용되는 리패키징 문제를 해결하기 위해서, 애플리케이션에 포함되어 있는 서명 파일을 이용한 리패키징 탐지 기술을 설계 및 구현한다. 본 논문의 구성은 2장에서 안드로이드의 코드 서명 기법 분석 및 취약점을 살펴보고, 3장에서 서명 파일을 이용한 애플리케이션의 리패키징 탐지 기술을 자세히 설명하며, 4장에서 구현된 리패키징 탐지 기술을 적용한 결과를 보인다. 마지막으로 5장에서 결론을 맺도록 한다.

## 2. 기반 기술 연구

본 장에서는 안드로이드에서 사용하는 코드 서명 기법을 분석한다. 또한 안드로이드 코드 서명의 취약점에 대해서도 기술한다.

### 2.1 안드로이드 코드 서명 기법

안드로이드는 사용자에게 배포되는 모든 애플리케이션을 인증서로 코드 서명 값을 생성한다. 코드 서명에 사용되는 인증서는 CA(Certificate Authority)가 아닌 개발자 스스로 JDK의 표준 툴인 keytool를 이용하여 생성한다.

그리고 개발자는 생성된 인증서를 자바 서명 툴인 jarsigner를 이용하여 애플리케이션의 자가-인증 서명 값을 생성한다.

안드로이드 마켓에서는 개발자가 애플리케이션을 업로드 하면, 마켓의 정책을 준수하는지 점검한 후 곧바로 일반 사용자가 다운로드 받을 수 있도록 마켓에 등록된다. 즉 자가-인증 서명은 개발자의 신원 증명을 위한 서명용도로 사용되지 않으며, 애플리케이션의 업데이트와 데이터 교환 등을 위한 목적으로 사용된다.

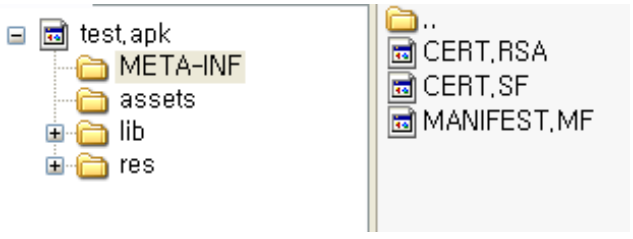
## 2.2 안드로이드 코드 서명의 취약점

안드로이드 자가-서명 인증서를 사용하여 코드 서명을 하기 때문에 개발자 검증이 불가능하여, 사용자는 설치하는 애플리케이션이 신뢰할 수 있는 것인지 판단하기 어렵다. 그러므로 자가-서명 인증서의 검증 취약점을 이용하여, 악의적인 공격자는 기존 정상 애플리케이션에 악성코드를 삽입, 리패키징 하여 마켓에 유포 할 수 있다.

또한, 안드로이드는 개인 PC를 통해 애플리케이션 설치가 가능하기 때문에, 악성코드가 삽입되어 리패키징 된 애플리케이션은 마켓을 통하지 않고 외부 사이트, 블랙 마켓 또는 P2P 등을 통한 유포도 가능하다.

## 2.3 서명된 안드로이드 애플리케이션 파일 구조

안드로이드는 먼저 서명되지 않은 애플리케이션을 생성한 후, 자가-서명 인증서를 사용하여 서명한다. 서명된 애플리케이션에는 서명 파일을 저장하는 META-INF 폴더가 추가된다. (그림 1)은 CERT.RSA와 CERT.SF 및 MANIFEST.MF로 구성된 META-INF 폴더이다.




(그림 1) 서명된 애플리케이션의 META-INF 폴더 구성

### 2.3.1 CERT.RSA

서명 알고리즘에 따라 확장자가 RSA, DSA로 나뉘며, 확장자를 제외한 파일이름 또한 바뀔 수 있다. 본 파일은 서명에 사용한 인증서와 CERT.SF에 대한 서명값이 들어 있다.

### 2.3.2 CERT.SF

MANIFEST.MF 전체 파일에 대한 SHA1 값을 Base64로 Encoding하고, MANIFEST.MF의 각 파일에 대한 이름과 SHA1 값을 Base64로 Encoding한 문자열을 다시 SHA1과 Base64 Encoding한 값을 저장하고 있다.

### 2.3.3 MANIFEST.MF

서명하지 않은 애플리케이션을 구성하는 모든 파일에 대한 각각의 SHA1 값을 Base64로 Encoding한 값을 저장한다.

## 3. 서명 파일을 이용한 리패키징 탐지 기술

제안하는 리패키징 탐지 기술은 인증서 정보가 포함되어 있는 확장자가 RSA ,DSA인 파일을 사용한다. 2장에서 설명한 것과 같이 이 파일들은 애플리케이션의 자가-서명 값을 생성할 때, 사용한 파일들이다. 앞서 언급했듯이 정상적인 개발자가 자신이 생성한 인증서를 가지고 자가-서명 후, 안드로이드 마켓에 업로드하면 정상적인 자가-서명 값을 포함하고 있을 것이다.

그러나 임의의 사용자가 다운로드 받아 변조하는 경우, 임의의 사용자는 개발자가 사용한 인증서를 사용할 수 없다. 그러므로 애플리케이션을 변조하는 공격자는 앱을 변조하고 자신이 별도로 생성한 또는 Debug 인증서를 사용하여 애플리케이션의 자가-서명 값을 생성한다. 다시 말해 공격자는 정상적인 애플케이션과 동일한 자가-서명 값을 생성할 수 없다.

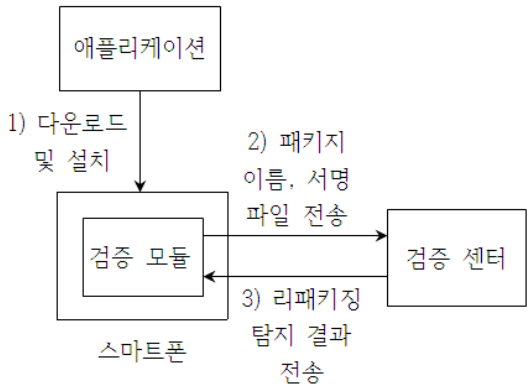
그리고 공격자는 임의의 애플리케이션을 배포하기 위해 래패키징 기술 사용할 때 사용자를 속이기 위해 배포하는 애플리케이션의 패키지 이름을 동일하게 한다. 그러나 동일한 패키지 이름의 애플리케이션이라 할지라도 애플리케이션이 변조되었기 때문에 결국 서명 값은 서로 다를 수밖에 없다.

그러므로 공격자가 또는 임의의 사용자가 애플리케이션을 변조하고 리패키징하여 배포한다면, 기본적으로 애플리케이션이 포함하고 있는 인증서는 정상 애플리케이션의 인증서와 서로 다를 수밖에 없으며 또한 자가-서명 값 역시 서로 다를 수밖에 없다.

이제 남은 문제는 인증서를 비교하는 방법이다. 논문에서 리패키징 검증 역할을 수행하는 검증 센터에서 패키지 이름과 서명 파일 일치 검사 시, 서명 파일에 포함되어 있는 인증서 지문(Certificate fingerprint)을 사용한다. 인증

서 지문은 MD5 또는 SHA1 해쉬 값이기 때문에 서로 다른 인증서는 동일한 해쉬 값을 가질 수 없다. 그 이유는 해쉬 알고리즘은 데이터 무결성 및 메시지 인증 등에서 사용 할 수 있는 알고리즘으로써 고정된 길의 출력 값인 해쉬 코드로 압축시키며, 강한 충돌저항성을 지니기 때문이다.

그리고 keytool로 생성한 같은 인증서로 각각의 애플리케이션을 자가-인증 서명하면 애플리케이션의 코드가 서로 다르기 때문에 생성되는 서명 파일의 값은 모두 다르다. 하지만 서명 단계에서 동일한 인증서를 사용했기 때문에 서명파일에 포함된 인증서 지문은 모두 동일하다. 따라서 애플리케이션 버전에 상관없이 한 개의 인증서 지문만으로 리패키징 검사가 가능하다.




(그림 2) 애플리케이션 리패키징 탐지 절차

본 논문에서는 이와 같은 점을 착안하여 애플리케이션의 리패키징을 탐지할 수 있는 방안을 고안하였으며 이를 구현하여 검증하였다.

사용자가 애플리케이션을 설치할 때, 검증 모듈은 애플리케이션의 패키지 이름, RSA, DSA 확장자 파일 그리고 서명 파일 개수를 검증 센터로 전송한다. 검증 센터는 보안성이 검증된 애플리케이션 패키지 이름과 RSA, DSA 확장자 파일을 DB에 저장하여 관리하고 있다. 검증 센터는 검증 모듈로부터 수신한 패키지 이름과 RSA, DSA 확장자 파일을 DB에서 검색하여 패키지 이름과 서명 파일이 일치 하는지 검사한다. 패키지 이름과 서명 파일이 서로 일치하면 정상적인 애플이케이션이고 일치하지 않으면 리패키징 된 애플리케이션이다. 해당 애플리케이션의 리패키징 여부를 탐지하고, 결과를 검증모듈로 전송한다. (그림 2)는 앞서 설명한 애플리케이션 설치 시 리패키징 탐지 절차이다.

## 4. 구현 및 기대 효과

본 논문에서 제안한 리패키징 탐지를 위해 애플리케이션 설치 시, 리패키징을 검사하는 검증 애플리케이션과 검증 센터를 구현 하였다. (그림 3)은 보안성이 검증된 애플리케이션 설치 시 검증 애플리케이션 로그이다.

```
E/CA      ( 4180): Check package : test.com
E/CA      ( 4180): test.com is not repackaging
```

(그림 3) 보안성이 검증된 애플리케이션 설치 시

(그림 4)는 리패키징된 애플리케이션 설치 시 검증 애플리케이션 로그이다.

```
E/CA      ( 4180): Check package : test.com
E/CA      ( 4180): test.com is repackaging
```

(그림 4) 리패키징된 애플리케이션 설치 시

위 결과와 같이 제안 기술은 애플리케이션 설치 시, 검증 애플리케이션이 자동으로 리패키징 검사를 실시한다. 리패키징된 애플리케이션 탐지 시 해당 인증서 지문을 블랙리스트로 등록하여, 안드로이드 OS의 강화된 보안성을 높일 수 있다. 또한 리패키징된 애플리케이션을 설치 시 검사하여, 악성코드 실행로 인해 발생할 수 있는 사용자의 피해를 줄일 수 있을것으로 기대한다.

## 5. 결론

본 논문에서는 안드로이드 OS 자가-인증 서명의 취약점을 이용한 애플리케이션 리패키징을 탐지하기 위해, 애플리케이션 서명 파일을 이용한 리패키징 탐지 기술을 제안하였다. 제안 기술은 서명 파일의 인증서 지문을 이용한 애플리케이션 리패키징 탐지 방법으로, 4장의 제안 기술의 구현을 통해 리패키징 탐지 기술을 검증하였다. 또한, 제안 기술은 리패키징 애플리케이션이 실행되기 전에 탐지하기 때문에 출력을 신뢰할 수 있다.

하지만, 본 논문에서는 검증센터 DB에 저장하는 패키지 이름과 서명 파일이 보안성이 검증된 애플리케이션이라는 가정 하에 실행되는 검증방법이다. 따라서, 향후 애플리케이션 보안성 검증에 관한 연구도 진행되어야 할 것으로 사료된다.

## 참고문헌

[1] 문남미, 조태남, 오정민, 변재희, 송주홍, 이용섭, 백길선, 서종성, 김윤성 “국내외 스마트폰 애플리케이션 블랙 마켓 현황 조사 및 국내 스마트폰 애플리케이션 마켓에 적합한 코드서명 기술 연구”, 한국인터넷진흥원 연구 보고서, November 2011.

[2] 이재영, 조도은, 이지영 “스마트폰에서 안전한 어플리케이션을 위한 무결성 검증기법”, 한국정보기술학회 논문지, pp. 223-228, 2011.

[3] 안드로이드 개발 가이드 홈페이지, http://developer.android.com/guide/basics/what-is-android.html

[4] Android Signing, http://developer.android.com/guide/publishing/app-signing.html

[5] keytool, http://docs.oracle.com/javase/1.3/docs/tooldocs/win32/keytool.html

[6] jarsigner, http://docs.oracle.com/javase/1.3/docs/tooldocs/win32/jarsigner.html