http://dx.doi.org/10.7840/kics.2014.39B.5.309

안드로이드 앱 변조 방지를 위한 APK 덮어쓰기 기법

최 병 하◆, 심 형 준*, 이 찬 희*, 조 상 욱*, 조 성 제°

An APK Overwrite Scheme for Preventing Modification of Android Applications

Byungha Choi◆, HyungJoon Shim*, ChanHee Lee*, Sangwook Cho*, Seong-je Cho°

요 약

안드로이드 앱(Android app, APK)을 역공학하여 디컴파일된 소스 코드를 획득하는 것이 용이하다. 공격자는 디컴파일된 소스코드를 불법적으로 사용하여 경제적 이득을 얻거나 악성코드를 삽입하여 앱을 변조하기도 한다. 이러한 문제를 해결하기 위해, 본 논문에서는 역공학 방지 방법을 사용하여 안드로이드 앱에 대한 불법 변조를 방지하는 APK 덮어쓰기 기법을 제안한다. 연구 대상은 임의 프로그래머에 의해 작성된 앱들이다. '대상 앱'(원본 앱)에 대해, 서버 시스템은 (1) 대상 앱의 복사본 생성, (2) 그 대상 앱을 암호화, (3) 복사본의 DEX (Dalvik Executable) 부분을 스텁(stub) DEX로 교체하여 스텁 앱 생성, (4) 암호화된 대상 앱 및 스텁 앱을 배포한다. 스마트폰 사용자는 암호화된 대상 앱 및 스텁 앱을 다운받는다. 스텁 앱이 스마트폰에서 실행될 때마다, 스텁 앱은 런처(launcher) 앱과 협력하여 암호화된 대상 앱을 복호화한 후 자신을 덮어쓰게 하여 원본 대상 앱이 실행되게 한다. 실행이 끝나면 복호화된 앱은 삭제된다. 제안 기법의 가능성을 검증하기 위해 여러 대중적인 앱들로 실험하여 보았다. 실험 결과, 제안 기법이 안드로이드 앱에 대해 역공학 및 변조 공격을 방지하는데 효과적임을 알 수 있다.

Key Words : Android App Package(APK), APK Overwrite, Dalvik Executable(DEX), Stub, Modification

ABSTRACT

It is easy to reverse engineer an Android app package file(APK) and get its decompiled source code. Therefore, attackers obtains economic benefits by illegally using the decompiled source code, or modifies an app by inserting malware. To address these problems in Android, we propose an APK overwrite scheme that protects apps against illegal modification of themselves by using a new anti-reverse engineering technique. In this paper, the targets are the apps which have been written by any programmer. For a target app (original app), server system (1) makes a copy of a target app, (2) encrypts the target app, (3) creates a stub app by replacing the DEX (Dalvik Executable) of the copied version with our stub DEX, and then (4) distributes the stub app as well as the encrypted target app to users of smartphones. The users downloads both the encrypted target app and the corresponding stub app. Whenever the stub app is executed on smartphones, the stub app and our launcher app decrypt the encrypted target app, overwrite the stub app with the decrypted target one, and executes the decrypted one. Every time the target app ends its execution, the decrypted app is deleted. To verify the feasibility of the proposed scheme, experimentation with several popular apps are carried out. The results of the experiment demonstrate that our scheme is effective for preventing reverse engineering and tampering of Android apps.

※ 본 연구는 미래창조과학부가 지원한 2014년 정보통신·방송(ICT) 연구개발사업과 문화체육관광부 및 한국저작권위원회의 2014년도 저작권 기술개발사업의 연구결과로 수행되었습니다.

◆ First Author : Research Institute of Information and Communication Convergence Technology, Dankook University, notanything@dankook.ac.kr, 정회원

° Corresponding Author : Dept. of Computer Science, Dankook University, sjcho@dankook.ac.kr, 정회원

* 단국대학교 컴퓨터학과, ggudwns@dankook.ac.kr, lchan12@dankook.ac.kr, anaislover@dankook.ac.kr

논문번호 : KICS2014-04-118, Received April 7, 2014; Revised April 16, 2014; Accepted April 16, 2014

Ⅰ. 서 론

최근 스마트폰을 비롯한 다양한 모바일 기기의 보급이 급속도로 대중화 되면서 안드로이드 기반의 앱(Application, App)의 수가 급증하여, 2012년에는 아이폰(iPhone) 앱의 수를 따라잡아 700,000 개의 앱이 구글 플레이에 존재한다[1-3]. 이에 따라 안드로이드 앱의 보안 침해 사고도 빠르게 증가하고 있다. 이는 2012년 79%의 악성코드가 안드로이드를 기반으로 활성화한 것으로도 알 수 있다[4]. 특히 안드로이드 앱은 아이폰 앱과 달리 비교적 쉽게 소스코드를 추출하여 분석할 수 있으므로, 앱의 불법 복제 및 위변조로 악성코드를 삽입하는 해킹 공격의 주요 대상이 되고 있다.

이러한 안드로이드 앱의 문제들을 해결하고 불법 복제 및 위변조를 차단하기 위해 다양한 소스 난독화 기법 및 구글의 LVL(Licensing Verification Library)이 제시되었으며, 앱 분석시간을 늦추거나 역공학을 어렵게 하는 부분적인 효과를 거두었다. 그러나 기존 방어 기법은 역분석 및 위변조를 효과적으로 차단하기에는 한계가 있다.

이러한 한계를 해결하기 위하여 본 연구에서는 APK 덮어쓰기 기법을 제안한다. 본 기법은 안드로이드 앱에 대한 역공학 공격을 방어하기 위한 것으로, 원본 앱을 본 연구에서 설계한 서버에 등록시 암호화하여 저장하는 한편, 동시에 특정 스텁(Stub) 앱으로 변환하여 개발자에게 돌려주어 앱스토어에 스텁 앱을 등록하게 한다. 스텁 앱을 앱스토어에서 다운 받은 사용자는 스텁 앱 실행시 본 연구에서 개발한 서버에서 암호화되어 저장된 원본 앱을 모바일 장치로 다운 받아 복호화한 후 스텁 앱을 원본 앱으로 덮어써서 실행한다. 또한 종료 시에는 복호화된 원본 앱을 삭제하고 스텁 앱만 드러나게 하는 기법이다.

본 기법은 소스 난독화시키는 기법에 비해 역공학 및 소스 분석이 힘들며 위변조 또한 어렵게 한다. 따라서 개발자 입장에서 어렵게 개발한 소스가 부정 사용되어지는 위협을 예방하고, 위변조로 인한 앱의 신뢰성 하락을 방지할 수 있다.

본 논문의 구성은 다음과 같다. 2장은 관련연구로 난독화 기법들에 분석하고 이들의 문제점을 제시한다. 3장에서는 APK 덮어쓰기 기법을 제안하며 4장에서 제안 기법을 검증 분석 및 타 기법과 비교하며 5장의 결론으로 본 논문을 마무리 짓는다.

Ⅱ. 관련연구

본장에서는 기존 난독화 기법과 관련 문제점을 분석한다.

2.1 난독화 적용 대상

난독화는 악의적인 소프트웨어 역공학으로부터 프로그램의 코드를 보호하기 위해 특정 방식으로 변환하여 바이너리나 소스코드가 역공학에 의해 분석되는 것을 어렵게 하기 위한 기술이다[5].

난독화 대상은 다음과 같은 프로그램의 컴파일된 형태에 따라 분석할 수 있다. 바이너리 코드(Binary Code)는 C나 Java 및 c# 등이 컴파일된 형태이다. C의 바이너리코드는 네이티브 코드(native code)로 불리고 Java 및 C#의 컴파일된 코드는 바이트 코드(Byte code) 및 매니지드 코드(mananged code)라고 불리는데 이들의 공통점은 중간언어(intermediate language)로 컴파일된 바이너리 코드라는 점이다. 즉 이들 바이트 코드 및 매니지드 코드는 가상머신 위에서 실행될 때 실시간으로 완전한 네이티브 코드로 생성이 되기 때문에 플랫폼 독립성이 보장된다. 반면에 네이티브 코드보다 바이트 코드에는 역공학에 필요한 정보가 보다 많이 포함되어 있어 쉽게 역공학이 가능하다.

아이폰의 경우 Object-C를 사용하며 네이티브 코드로 컴파일되어 비교적 역공학이 어려워 소스코드 유출의 문제가 크지 않다. 그리고 C#의 경우는 모바일 장치에서 사용되는 비중이 크지 않기 때문에 현재까지는 많은 연구가 진행되지 않았으며 이의 필요성도 많이 요구되지 않는다.

그러나 안드로이드 기반 앱의 경우 대부분 Java로 구성되어 바이트 코드로 컴파일된 형태이므로 이들의 역공학이 쉽고 위변조 및 불법적인 부정사용이 문제점으로 많이 대두되고 있다. 그러므로 최근 연구의 연구대상과 요구사항은 대부분 Java 기반의 안드로이드 앱이며 이의 난독화가 가장 큰 요구사항이라고 볼 수 있다.

2.2 난독화 기법

난독화 기법은 식별자 변환(renaming), 제어 흐름(control flow), 문자열 암호화(string encryption), API 은닉(API hiding)과 클래스 암호화(class encryption)으로 분류된다[6].

식별자 변환은 클래스, 변수, 함수 등의 이름을 의미없는 이름으로 대체하는 기법이고 흐름 제어는 파

악하기 힘든 불투명 명령 또는 쓰레기 명령을 삽입하여 역공학시 제어흐름 분석을 어렵게 하거나 잘못된 결과를 발생시키는 기법이다. 문자열 암호화는 특정 문자열을 암호화하고 실행시 복호화 메쏘드(method)로 복원하여 사용하는 것이다. API 은닉은 민감한 라이브러리 및 메소드 호출을 감추는 기법이며, 클래스 암호화는 특정 클래스 파일 자체를 암호화하여 실행시 복호화하는 기법이다.

식별자 변환 및 제어 흐름 기법은 분석하는 시간만 지체시킬 뿐이고 문자열 암호화 및 특정 클래스 암호화는 실행시 흐름을 따라가며 암호화된 내용을 유추할 수 있다. 그러므로 난독화를 최대한 어렵게 하려면 클래스 전체를 암호화 시키는 방법이 가장 효과적이다.

2.3 난독화 도구 및 다른 기법

난독화 도구로써 다음과 같은 것이 있다. Proguard, Allatori Dex Guard, Stringer Java Obfuscator, 등이 있으며 표 1과 같은 기능을 제공한다[7,8].

난독화 도구 이외에 구글에서는 라이선스 라이브러리인 LVL(LIcense Validation Library)을 배포하고 있다[8]. 그러나 이미 이를 우회하여 비공인 앱스토어인 블랙마켓에서 다운 받을 수 있는 방법이 나온 상태이며 이는 설치파일을 추출하여 재배포할 수 있다는 것을 의미한다는 점에서 근본적인 해결책이 되기 어렵다.

또한 국내 서비스로는 한국저작권위원회에서 소스코드 난독화와 소스코드 저작자 확인 서비스를 제공해주고 있다. 이는 소스코드 유출 및 도용시 삽입된 저작자 정보를 검증하여 저작권 침해 사실을 용이하게 입증하는 서비스다. 그러므로 이 서비스 역시 소스 유출에 대한 근본적인 해결책으로 사용하기에는 한계가 있다[9,10].

표 1. 난독화 도구의 기능
Table 1. Functionalities of Obfuscation Tools

	Pro guard	Dash OPro	Allatori	Dex Guard	Stringer
Renaming	Y	Y	Y	Y	N
Control Flow	N	Y	Y	Y	N
API Hiding	N	N	N	Y	N
Class Encryption	N	N	Y	Y	Y(String Encryption)

2.4. 기존 기법의 문제점

앞서 분석대로 대부분의 역공학 문제는 안드로이드 기반의 자바로 이루어진 앱에서 발생한다. 난독화는 근본적으로 소스 코드를 복원 못하도록 하는 기능이 없다. 다만 분석을 어렵게 하고 역공학을 지연시키는 기능을 가졌을 뿐이다. 또한 LVL은 블랙마켓에 앱이 유출되어 이미 무의미한 기법이 되어 버렸다.

그러므로 이러한 문제점을 해결해야한다.

첫째, 안드로이드의 자바기반 앱을 난독화시켜야 한다.

둘째, 난독화된 앱은 역공학으로 분석이 기존 앱보다 어려워야 한다.

셋째, LVL처럼 앱이 유출되어 블랙마켓에 공개되더라도 역공학이 쉽게 되지 않아야 한다.

Ⅲ. APK 덮어쓰기 기법

본 장에서는 앞장에서 분석한 기존 기법의 문제점을 해결하고 AES로 암호화하여 부정사용 및 소스코드 유출을 차단하는 개선된 기법을 제안한다.

3.1 핵심 구성 요소

본 연구의 모바일 장치에서 핵심 구성요소는 런처 앱(Launcher App)과 스텁 앱(Stub App)으로 나누어진다.

안드로이드 부팅 시부터 안드로이드 앱 파일인 APK(Android Application Package File) 접근하고 관리하는 앱을 런처라고 하며 이는 안드로이드 기본 런처와 사용자 정의 런처로 나눌 수 있다[11,12]. 본 기법의 런처 앱은 시스템 권한을 가진 앱으로 본 연구의 스텁 앱을 제어하고 암호화 된 원본 앱을 복호화하는 등의 기능을 가진 앱이다. 본 기법인 APK 덮어쓰기를 사용하기 위해서는 모바일 장치에 가장 먼저 설치되어 있어야 하며, 시스템 권한을 가지기 위해서는 모바일 장치 제조사에서 시스템 권한을 제공하여 설치하거나 본 연구의 개발자가 모바일 장치 버전마다 시스템 권한을 직접 개발할 수 있다.

또한 스텁 앱은 원본 앱의 주요 실행파일을 제거하고, 단순히 원본 앱을 실행하기 위한 앱으로, 해당 앱의 폴더, 이미지, 기타 자원(Resource)을 미리 설치하는 파일이다. 암호화된 원본 앱을 런처앱이 복호화한 후 덮어쓸 때 미리 설치된 자원파일과 설정 등이 모두 설치되어 있어야 실행 가능하므로 스텁 앱은 이를 위해 미리 설치하는 파일이다. 그러므로 악의적인 개발

자가 스텁 앱을 분석하더라도 핵심 실행 파일을 얻을 수 없도록 하는 역할을 가지고 있다.

3.2 전체 구조

그림 1은 제안 기법의 전체 처리과정을 나타내는 것으로 다음과 같다.

개발자가 먼저 앱을 개발하여 본 연구에서 개발한 서버에 먼저 등록한다. 이는 개발자가 개발한 원본 앱을 본 서버에서 암호화하고 스텁 앱을 생성하기 위해서이다. 스텁 앱을 개발자가 본 서버에서 수령받아 이를 구글 플레이라는 앱스토어에 등록한다. 사용자는 실제 앱스토어에서 원본 앱을 다운 받는 것이 아니라 스텁 앱을 다운 받도록 되어 있다. 스텁 앱이 사용자의 모바일 장치에서 실행되며, 이때 앱의 구성요소를 호출하거나 메시지를 보낼 때 사용하는 인텐트로 스텁 앱이 런처앱에게 알린다. 런처앱은 스텁 앱을 설치하기 전에 모바일 장치에 설치되어 있어야 한다. 런처앱은 본 서버에서 암호화된 원본 앱을 다운 받는다. 그리고 이 앱을 복호화하여 스텁 APK파일을 덮어쓰고 실행시킨다. 그리고 마지막 앱이 종료되는 순간 런처앱은 다시 복호화된 앱을 삭제하고 스텁 앱만 남긴다.

이들의 과정을 자세히 분석하면 제안 서버에서의 처리과정과 모바일 장치에서의 처리과정으로 나눌 수 있다.

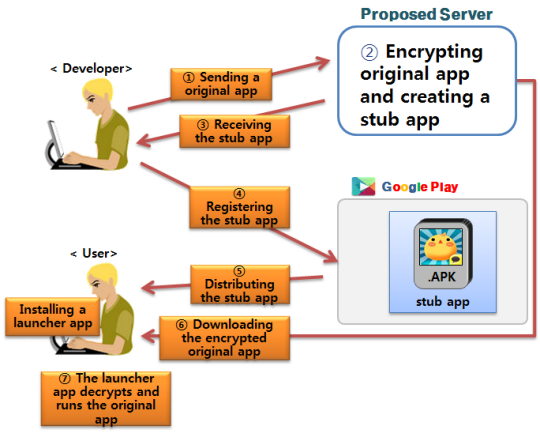

그림 1. 제안 기법의 전체구조
Fig. 1. Overview of the Proposed Scheme

3.3 서버 측의 처리과정

서버 측에서는 다음과 같이 두 가지 흐름으로 처리된다. 그림 2와 같이 이 과정들은 개발자가 먼저 원본 앱을 본 서버에 등록하는 것부터 시작된다.

첫 번째 흐름(과정)은 원본 앱을 스텁 앱으로 생성

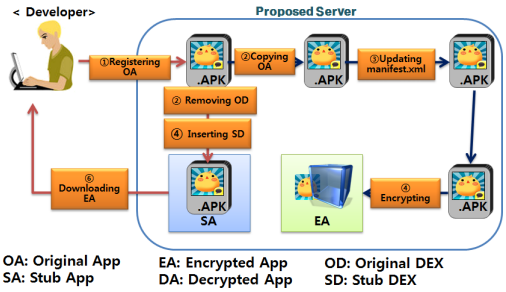

그림 2. 서버 측의 처리
Fig. 2. Server-side processing

하는 것으로 서버에서 Zip파일 압축형식인 APK의 원본 앱에서 주요 실행 파일인 classes.dex 파일을 제거하고 런처 앱에게 인텐트로 원본 앱을 실행하라는 내용을 담은 classes.dex로 교체한다. 이로써 스텁 앱이 완성된다.

또 하나의 과정은 암호화 과정으로 원본 앱을 AES(Advanced Encryption Standard)-128 bit로 암호화한다. 그리고 원본 앱이 런처앱으로부터 인텐트를 받을 수 있도록 AndroidMenifest.xml을 수정한다. 그리고 이를 다시 암호화시켜 암호화된 앱으로 완성한다.

3.4 모바일 장치에서의 처리과정

모바일 장치에서의 과정은 그림 3과 같으며, 이를 실행하기 위해 모바일 장치에서는 시스템 권한을 가진 런처앱이 먼저 설치되어야 한다. 일반적인 안드로이드 사용자 앱은 먼저 키(Keystore)를 생성하고 컴파일한 후 서명(Signing)을 해야 한다. 시스템 권한을 가지려면 Platform Key라고 명명된 키가 필요하다. 그러므로 Platform Key를 얻기 위해서 모바일 장치 개발사가 시스템 권한을 가진 앱을 설치하거나, 개발자가 운영체제 버전 또는 하드웨어 버전별로 Platform Key를 획득하여 설치하면 시스템 권한을 가진 앱을 설치할 수 있다.

사용자가 스텁 앱을 실행시키면 그림 3과 같은 과정을 거치며 이는 다음과 같다.

1) 런처 앱이 설치되어 있는 상태에서 스텁 앱은 런처 앱에게 자신이 실행되어야 한다는 메시지를 인텐트를 통해 보낸다. 런처앱이 메시지를 받았으면 스텁 APK 파일을 백업한다. 이는 앱 종료시 복호화된 앱을 삭제하고 원래 상태의 스텁 앱으로 복원하기 위해서이다.
2) 본 연구의 서버에 이미 암호화되어 있는 해당 원본 앱을 다운 받는다.

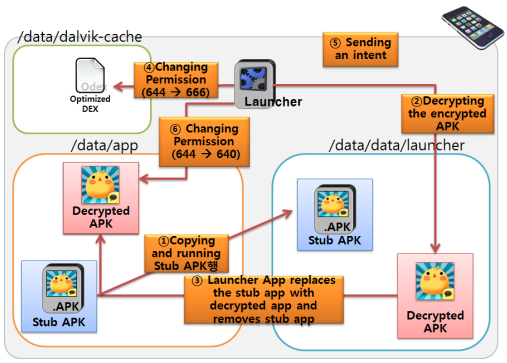

그림 3. 모바일 장치에서의 처리
Fig. 3. Mobile Device-side processing

3) 암호화된 앱을 런처가 다시 복호화하고 /data/app 폴더에 존재하는 스텁 APK을 복호화된 원본 APK로 덮어쓴다.
4) /data/dalvik-cache에 해당 앱의 oDex 파일의 권한을 변경한다. oDex 파일은 안드로이드 앱에서 달빅(Dalvik) 가상머신이 달빅 캐쉬 폴더에 최적화시켜 실행 시에 더욱 빠르게 앱을 구동시킬 수 있는 파일로써 실행전에 APK 클래스의 classes.dex파일과 동일한 파일로 최적화되어 있는지 검증한다. 이때 동일한 파일로 최적화되지 않으면 안드로이드 앱은 새로 덮어쓰려고 하는데, 일반적으로 사용자 권한으로는 덮어쓸 수 없어, 권한이 없으면 실행되지 못한다. 그러므로 oDex 파일을 덮어쓸 수 있는 권한을 주어야 스텁 앱의 oDex 파일을 복호화된 원본 앱으로 덮어쓰고 실행이 가능한 상태가 된다.
5) 런처앱이 복호화된 앱을 실행시키기 위해 인텐트를 전송한다.
6) 복호화된 앱의 소스 유출을 막기 위해 타 앱이나 사용자가 읽지 못하도록 하는 권한으로 변경한다.

그리고 종료 후 복호화된 원본 앱을 삭제하고 백업된 스텁 APK파일로 교체하여 원본 앱의 소스 유출을 막는다.

Ⅳ. 검 증

본 장에서는 3가지의 실험을 통해 본 기법의 우수성을 검증한다.

4.1 유효성 검증

본 절에서는 본 기법이 일반적으로 많이 사용되는 앱에 적용가능한가에 대해 검증한다. 데이터셋은 구글 마켓에서 원본 APK 파일 자체를 PC에서 다운 받을 수 없으므로 모바일 장치에 설치되어 있는 앱에서 추출하여 APK 파일을 생성하였다. 많이 사용되고 다양한 기능을 가진 앱일수록 복잡도가 높고 다양한 기법을 사용하므로 본 기법을 적용하기가 어려워질 수 있다. 그러므로 본 기법이 적용가능한지에 대한 검증이 필요하다.

실험 환경은 표 2와 같이 하였고 표 3는 본 기법을

표 2. 환경설정
Table 2. Configuration

Server	Tools for Tests	Mobile Device
Ubuntu 12.04 LTS 64-bit	Android SDK, APKBuilder, APKTool 2.0.0.b9, Java JDK 1.6.45, Java JDK 1.7.51	Samsung Galaxy S 2 (Android 4.4.2 Custom ROM for Developer)

표 3. 데이터셋
Table 3. Dataset

Name of Applications	Name of File	Number of Download
Naver BAND	com.nhn.android.band	more than 10,000,000
Coupang	com.coupang.mobile	more than 10,000,000
마약팅	hn.mayak.freeting	more than 100,000
d SHOT wave	com.d.shot.wave	more than 100,000
할로윈 얼굴 만들기	com.day.dayhalloween	more than 10,000
야구의 달인	com.mobcast.mbpBrowserKR	more than 100,000
Yonhap News	kr.psynet.yhnews	more than 50,000
전국맛집 TOP1000	com.menupan.mptop	more than 500,000
TravelDiary	co.kr.wook.traveldiary	more than 5,000
풀어서 잠금해제	com.day.dayunlockquiz	more than 100,000
Joara(조아라)	com.joara.mobile	more than 500,000
Afreeca TV	kr.co.nowcom.mobile.afreeca	more than 10,000,000
candycrushsaga (캔디크러시사가 for Kakao)	com.king.candycrushsagakakao	more than 50,000,000
Kakaotalk	com.kakao.talk	more than 100,000,000

적용하기 위한 데이터셋이다.

표 4. 평가 결과
Table 4. Results of Evaluation

Name	Results		Description
	Creating and Installing	Running	
Naver BAND	success	partial sucess	failure of sign-in
Coupang	success	partial sucess	when stub App is recovered, Abnormal termination of Laucher app
마약텡	success	success	
d SHOT wave	success	success	
할로윈 얼굴 만들기	success	success	
야구의 달인	success	success	
Yonhap News	success	success	
전국맛집 TOP1000	success	success	
TravelDiary	success	success	
풀어서 잠금해제	success	sucess	error message occasionally occurs but it works
Joara(조아라)	success	success	
Afreeca TV	success	success	
candycrushsaga(캔디크러시사가 for Kakao)	success	failure	the App doesn't show the initial window
Kakaotalk	success	failure	

암호화를 시키는 서버는 Ubuntu에서 사용하였으며 그림 2의 디버깅과 분석을 위해 처리과정을 자동화시키지 않고 수작업으로 처리하였다.

3장의 제안한 것처럼 표 3의 데이터셋의 APK파일을 압축 풀어 스텁 Dex 파일을 교체하여 다시 패키징하여 스텁 앱으로 변환하였다. 다만 원본 소스 코드가 없는 상태에서 시도하므로 일부분 실패하였다. 그러므로 이 실험의 또다른 의미는 유명 앱이라도 본 기법을 적용할 수 있다는 것은 소스코드가 없더라도 역공학을 통해 악성코드를 삽입하여 변조가 가능하다는 것을 의미한다. 이들이 실제 실행 가능한지에 대한 결과는 표 4와 같다.

본 기법의 적용을 위한 표 4의 14개 앱의 소스코드가 없는 상태에서 역공학으로 소스를 분석한 후 본 기법을 적용했을 때 9개의 앱은 정상적으로 작동하고 나머지 5개 앱은 비정상적으로 작동하였다. 카카오톡은 역공학을 통해 위변조를 방지하기 위해 안드로이드 무결성 검증 기술을 채용하고 있을 것으로 판단되며, 나머지 실패한 앱들도 완벽한 소스코드가 역공학되지 않거나 무결성 검증 기술로 인해 실패한 것으로 분석된다. 그러나 이 실험 결과로 볼 때 소스코드가 없는 유명 앱들도 대부분 역공학하여 악성코드를 삽입할 수 있는 여지가 있으며 악성코드 삽입 대신에 본 기법의 적용도 64%의 성공률을 보인다.

4.2 타 기법과의 비교

타 기법과의 비교를 하기 위해 오픈 소스 난독화도구로 무료이며 가장 많이 사용되는 ProGaurd와 본 기법을 비교하였다. 실험 방법은 오픈소스로 완전한 소스코드가 있는 앱을 이용하여 ProGuard를 적용한 앱과 본 기법을 적용한 앱을 위변조 가능한지 실험하였다. 결과는 표 5와 같으며 ProGuard를 이용하여 난독화시킨 앱은 쉽게 원하는 로그를 삽입하여 변조하기 쉬웠으며 AES 적용한 본 기법은 AES를 크랙할 수 있는 툴을 발견하기 힘들 뿐만 아니라 NSA에 의해 보안 취약점이 없는 것으로 알려졌다. 그러나 최근에 몇몇 공격 방법이 제안되었으나, 현재까지 시간복잡도가 상당히 높아 단시간내에는 복호화하기 힘든 것으로 알려져 있으며[13,14] 본 실험에서도 복호화하는데 실패하였다. 또한 스텁 앱은 그 핵심 내용이 없는 상태이므로 소스 유출이 되더라도 무의미하다.

표 5. 프로가드와 제안 기법의 비교
Table 5. Comparison with Proguard and Proposed scheme

Name of File	Progard	Proposed Scheme	
		Stub App	Encrypted Original App
android.SimpleMusicPlayer	modified	modified	decryption failure
kiv.janecekz	modified	modified	decryption failure
org.jfedor.frozenbubble	modified	modified	decryption failure
org.vudroid	modified	modified	decryption failure
source: source.google.com			

4.3 제안 기법과 기존 기법의 문제점 해결

2장에서 기존 기법에서 해결할 이슈로 다음과 같은 3가지를 제시하였다. 안드로이드의 자바기반 앱을 난

독화시켜야 하며, 난독화된 앱은 역공학으로 분석이 기존 앱보다 어려워야 한다. 그리고 마지막으로 앱이 블랙마켓에 공개되더라도 역공학이 되지 않아야 된다.

본 장의 실험은 안드로이드 자바기반 앱을 난독화 시켰고, 표 5에 의하면 기존 기법보다 제안 기법의 역공학이 어려웠다. 또한 표 5의 앱들은 정식적인 경로인 구글 스토어가 아닌 본 연구자의 자료에서 생성한 앱을 대상으로 역공학을 시도하였으므로 3번째 문제점도 고려된 것이라고 볼 수 있다.

V. 결 론

본 기법은 원본 앱의 핵심 실행 파일인 DEX 파일을 특정 스텁 Dex로 교체한 후, 스텁 앱이 실행 시에 암호화시킨 원본 앱을 다운 받아 복호화하여 스텁 APK파일을 덮어써서 원본 앱으로 복구시킨다. 실행 후 종료 시에는 복구된 앱을 스텁 앱으로 다시 덮어써서 소스 유출 및 위변조를 방지하는 기법을 제안하였다.

안드로이드 자바 기반의 앱에 대해 역공학을 어렵게 하여 암호화된 앱이 유출되더라도 소스를 역분석하지 못하게 하는 기능을 제시하여, 기존 기법의 문제점을 해결하였다. 검증을 위해, 무료로 많이 사용되는 난독화 도구인 프로가드와 비교하여 제안 기법이 변조에 강인한 내구성을 가졌음을 보였다.

본 기법의 단점으로는 런처앱과 스텁 앱 그리고 암호화된 원본 앱, 3가지를 처리하여 실행하므로 원본 앱보다 실행시간이나 사용 자원이 많은 것으로 분석되었다.

이러한 단점을 해결하기 위해 메모리에 복호화되어 있는 내용을 보호하는 기법과 부분적으로 핵심 코드만 선택하여 부분 암호화하는 방식으로 클래스 파일인 DEX파일만 암호화하여 실행 속도와 사용자원을 줄이는 기법을 향후 연구로 제시한다.

References

[1] A. Datta, K. Dutta, S. Kajanan, and N. Pervin, “Mobilewalla: A mobile application search engine,” *Mobile Computing, Applications, and Services, Springer,* Berlin Heidelberg, pp. 172-187, 2012.

[2] Yun-Jae Jang, Kyoung-Wook Park, and Sung-Keun Lee, “A Home Automation system based on Smart phone,” *J. KICS,* vol. 6, no. 4, pp. 589-594, Aug., 2011.

[3] J. S. Lim, H. M. Lee, and S. C. Kim, “Black-box app for vehicle using android phone,” in *Proc. KICS*, pp. 688-691, Feb. 2012.

[4] F-Secure Team, Mobile Threat Report Q4 2012, F-Secure Technical Report, Dec. 2012.

[5] J. Kim and E. Lee, “A strategy of effectively applying a control flow obfuscation to programs,” *J. Korea Soc. Comput. Inf.*, vol 16, no. 6, pp. 41-50 Jun. 2011.

[6] Y. Piao, J. Jung, and J. H. Yi, “Structural and Functional Analyses of ProGuard Obfuscation Tool,” *J. KICS*, vol. 38, no. 8, pp. 654-662, Aug. 2013.

[7] D. Kim and Y. Park, “Effective detection and analysis for malicious-related API calls in obfuscated android malware,” in *Proc. KIISE*, pp. 790-792, Nov. 2013.

[8] S.-R. Kim, “Copy protection system for android app using public key infrastructure,” *J. Security Eng.*, vol. 9, no. 1, pp. 121-134, Feb. 2012.

[9] DigiCaP Team, “DigiCAP Codejam Service,” Retrieved Mar., 30, 2014 from http://www.codejam.or.kr/codejam

[10] J. Jang, S. Han, Y. Cho, U J. Choe, and J. Hong, “Survey of security threats and countermeasures on android environment,” *J. Security Eng.*, vol. 11 no. 1, pp. 1-12, 2014.

[11] C.-H. Lee, Y.-U. Park, J.-H. Lim, H. Kim, C.-H. Lee, S.-J. Cho, and J. Yang, “Access control mechanism preventing application piracy on the android platform,” *J. KIISE: Comput. Practices and Lett.,* vol. 18, no. 10, Oct. 2012.

[12] K. I. Shin, J. S. Park, J. Y. Lee, and J. H. Park “Design and implementation of improved authentication system for android smartphone users,” *IEEE WAINA*, pp. 704-707, Fukuoka, Mar. 2012.

[13] A. Biryukov, O. Dunkelman, N. Keller, D. Khovratovich, and A. Shamir, “Key recovery attacks of practical complexity on AES-256 variants with up to 10 rounds,” *Advances in Cryptology –EUROCRYPT 2010, Springer*, vol.

6110, pp. 299-319, 2010.

[14] B. Alex and D. Khovratovich. "Related-key cryptanalysis of the full AES-192 and AES-256," *Advances in Cryptology –ASIACRYPT 2009, Springer*, vol. 5912, pp. 1-18, 2009.

최 병 하 (Byungha Choi)

2009년 8월 : 단국대학교 정보통신학과 석사
2014년 2월 : 단국대학교 컴퓨터학 박사
2014년 3월~현재 : 단국대학교 정보통신 융합 연구원
<관심분야> 네트워크 보안

심 형 준 (Hyungjoon Shim)

2013년 2월 : 단국대학교 소프트웨어학과 학사
2013년 3월~현재 : 단국대학교 컴퓨터학 석사 과정
<관심분야> 운영체제 및 보안

이 찬 희 (ChanHee Lee)

2013년 2월 : 단국대학교 소프트웨어학과 학사
2013년 3월~현재 : 단국대학교 컴퓨터학 석사 과정
<관심분야> 운영체제 및 보안

조 상 욱 (Sangwook Cho)

2014년 2월 : 단국대학교 소프트웨어학과 학사
2014년 2월~현재 : 단국대학교 컴퓨터학 석사 과정
<관심분야> 운영 체제 및 보안

조 성 제 (Seong-je Cho)

1989년 : 서울대학교 컴퓨터공학과 졸업
1991년 : 서울대학교 컴퓨터공학과 공학석사
1996년 : 서울대학교 컴퓨터공학과 공학박사
2001년 : 미국 University of California, Irvine 객원 연구원
2009년 : 미국 University of Cincinnati 객원연구원
1997년 3월~현재 : 단국대학교 컴퓨터학과 교수
<관심분야> 시스템 보안, 시스템 소프트웨어, 스마트폰 보안, 소프트웨어 보안 및 보증