Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.27, NO.6, Dec. 2017 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2017.27.6.1457

안드로이드 환경에서 보안 토큰을 이용한 앱 난독화 기법

신 진 섭,[†‡] 안 재 환
한국전자통신연구원 부설연구소

An Application Obfuscation Method Using Security Token for Encryption in Android

JinSeop Shin,[†‡] Jaehwan Ahn
The Attached Institute of ETRI

요 약

스마트기기 시장의 성장과 함께 모바일 환경에서 악성행위가 그 영역을 점차 확대하고 있다. 이에 따라 악성앱 분석에 대한 연구가 진행되어 앱 분석을 위한 자동 분석 도구가 나오면서, 오히려 이런 자동 분석도구들로 인해 기존의 앱 보안을 위한 도구들이 공격자에게 무력해지는 부작용이 일어난다. 본 논문은 일반적인 안드로이드 앱에 적용할 수 있는 범용적인 보호 기법이 아닌 보안 토큰을 가진 스마트 기기 사용자가 이용하는 안드로이드 앱에 적용할 수 있는 앱 보호 기법에 대해 제안한다. 보안 토큰이 삽입되지 않은 경우 앱이 정상적으로 메모리로 적재되지 못하며, 해당 기법으로 보호된 부분은 노출되지 않도록 하는 것을 특징으로 한다.

ABSTRACT

With the growing of smart devices market, malicious behavior has gradually expanded its scope. Accordingly, many studies have been conducted to analyze malicious apps and automated analysis tools have been released. However these tools cause the side effects that the application protection tools such as ProGuard, DexGuard become vulnerable to analyzers or attackers. This paper suggests the protection mechanism to apply to the Android apps using security token, rather than general-purpose protection solutions that can be applied in malicious apps. The main features of this technique are that Android app is not properly loaded in the memory when the security token is abnormal or is not inserted and protected parts using the technique are not exposed.

Keywords: Android, Encryption, Security Token

I. 서 론

스마트기기 시장의 성장으로 인하여 스마트기기는 우리들의 생활 깊숙이 관여하고 있다. 이러한 상황을 반영하듯 일반 PC에 집중되던 여러 가지 사이버 범죄는 보안이 비교적 느슨한 모바일 기기로 눈을 돌려 피해 사례가 증가하고 있다[1]. 로컬 및 원격 취약점을 이용한 기기 장악 뿐만 아니라 이메일 및 문자 메시지를 이용한 악성 앱 설치, 공격코드 실행, 모바일용 랜섬웨어 등 상당히 다양한 방식으로 공격이 이루어진다[2].

주로 악성코드 및 악성 애플리케이션(이하, 앱)은 백신에서 주로 사용하는 필터링이나 기타 보호 솔루션에 걸리지 않기 위해서 패킹(packing)을 적용하거나, 난독화 도구를 적용하여 백신 및 분석가가 악성행위를 쉽게 알아채지 못하도록 지능적으로 진화하고

Received(09. 15. 2017), Modified(10. 23. 2017), Accepted(10. 26. 2017)
† 주저자, whiterick@nsr.re.kr
‡ 교신저자, whiterick@nsr.re.kr(Corresponding author)

있다[2].

이와 마찬가지로 상용 보안 솔루션을 자동으로 언팩(unpack)하거나 코드가 난독화되어 쉽게 해석되지 않는 코드를 원본의 코드 수준으로 역난독화하는 연구, 악성앱을 분석 및 대응하기 위한 분석 자동화 도구의 연구가 진행되고 있다[3][4].

하지만 이로 인해 보안성 향상을 위해 적용하는 패킹도구나 난독화 도구를 적용한 앱의 안전도 하락이 우려된다. 하루가 다르게 증가하는 악성앱의 분석을 원활하게 하려는 방식이 도리어 앱 보호를 위한 방식을 무용지물로 만들기 때문이다.

본 논문은 앱을 보호하기 위해 안드로이드의 수정 없이 보안토큰을 이용한 앱 보호 기법에 대해 제안한다.

2장에서는 안드로이드 실행환경과 앱에 관련한 배경지식 그리고 안드로이드 앱을 보호하기 위해 취할 수 있는 방식에 대해 알아본다. 3장에서는 보안 토큰을 이용한 DEX 암호화 기법에 대해 제안한다. 4장에서는 실험 결과를 보이고, 끝으로 5장에서 결론을 맺는다.

II. 관련 연구

2.1 Dalvik VM과 Android Runtime

Dalvik은 2013년 발표한 안드로이드 4.4(코드명 Kitkat)까지 안드로이드의 주된 실행 객체로 사용되었다[5]. 구글은 Kitkat에서 처음으로 ART(Android RunTime)와 Dalvik을 사용자로 하여금 선택하여 사용하도록 했으며, 안드로이드 5.0(코드명 Lollipop) 버전부터는 ART가 주된 안드로이드 실행 환경으로 자리 잡고 있다[6].

자바언어를 사용하도록 설계된 안드로이드는 자바의 JVM(Java Virtual Machine)과 같은 역할을 수행하는 장치가 필요하다. 자바는 어느 환경에서나 실행되는 특성을 가지고 있고, 구동 플랫폼은 컴파일된 자바 파일을 해석해주는 장치가 필요한데 이 장치가 JVM이다. 자바 컴파일러는 자바 코드를 중간언어로 바꿔주고, JVM이 이를 해석해 해당 플랫폼에서 실행할 수 있는 코드로 변환하여 실행한다. 자바는 이러한 과정을 거치기 때문에 실행 플랫폼에 맞게 결과 파일을 생성하여 실행하는 네이티브(native) 언어의 경우와 비교해 매번 실행 시 JVM에서 컴파일이 필요하므로 시간을 비교적 더 소모할 수 밖에 없다.

이런 점은 제한된 자원을 가진 모바일 기기에서는 아주 치명적인데, 이를 해결하기 위해 JVM의 역할을 수행하는 Dalvik VM은 zygote라는 미리 초기화된 객체를 생성하여 그 오버헤드를 줄인다. Dalvik VM 위에서 실행되는 파일은 DEX(Dalvik EXecutable)의 포맷을 가지며, 여러 class 파일을 합친 DEX 파일은 Fig. 1.과 같은 형태를 가진다.

한편 안드로이드 실행 환경이 ART로 바뀌면서 안드로이드에서 실행되는 파일은 ELF 파일 포맷을 가지는 OAT 파일로 바뀌었다. ART는 앱의 컴파일 시점부터 실행 시점까지를 포함하는 전반적인 새로운 실행환경이라고 볼 수 있으며, Dalvik VM은 ART의 한 구성요소로 포함되어 있다. 그 파일의 구조는 Fig. 2.와 같다. OAT 파일은 DEX 파일을 포함하며, 각 DEX 파일이 컴파일되어 OAT 파일의 text 영역에 자리 잡는다.

ART로 안드로이드 실행 환경이 변화하였지만

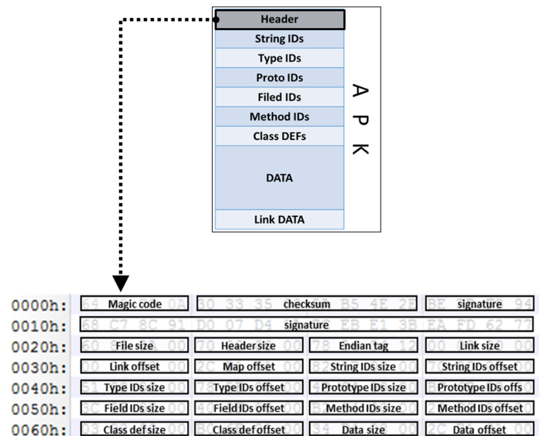


Fig. 1. DEX file structure and header

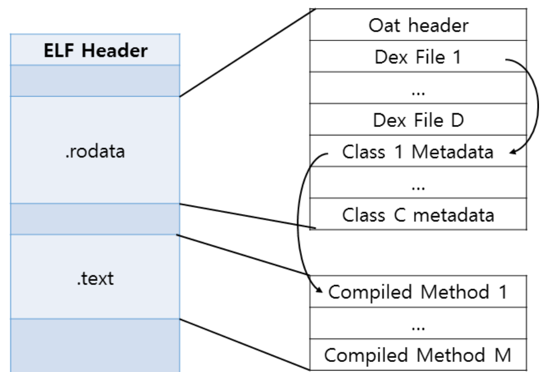


Fig. 2. OAT file structure

APK 파일의 구조는 변함이 없다. 다만 앱 설치시 내부 실행파일인 dex2oat가 동작해 DEX 파일을 OAT 파일로 변환하며, 변환하는 과정에서 DEX 파일이 머신코드로 컴파일된다. ART 실행 환경에서는 기본 실행 단위가 OAT 이므로 ART 런타임에서 DEX를 메모리로 적재하기 위해서는 dex2oat를 거쳐야 한다.

2.2 안드로이드 앱 구조

안드로이드 프레임워크에서 기본 실행 단위인 APK(Android PacKage) 파일은 크게 DEX 파일과 기타 파일로 나눌 수 있다. APK 파일은 앱 설치 파일로 언급되기도 한다.

자바 파일을 javac를 이용해 컴파일하면 class 파일이 생성되고, 복수 개의 class 파일들을 dx로 컴파일하면 하나의 DEX 파일이 생성된다. 이 DEX 파일이 앞서 살펴본 Dalvik VM이 해석하는 파일이다.

APK 파일은 Fig. 3.에서와 같이 위에서 만들어진 DEX 파일과 리소스파일 및 외부참조라이브러리 등이 ZIP 으로 압축된 형태로 만들어진다.

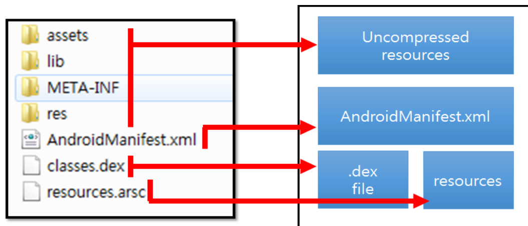


Fig. 3. Structure of APK

2.3 안드로이드 앱 보호 방식

안드로이드 환경에서 앱을 보호하기 위해서는 그 적용 시점과 방법에 따라 3가지 방식으로 나눌 수 있다.

2.3.1 컴파일 시점에 적용하는 앱 보호 방식

컴파일 시점에 앱 보호를 위한 상용 도구로 DexGuard, Allatori, ProGuard 등이 있으며, 흔히 난독화 도구로 알려져 있다. 컴파일 시점에 적용하기 때문에 앱을 보호하기 위해서는 앱 개발 환경에서 위 도구들을 적용해야 한다. 난독화 적용으로 인해 앱 실행시 잘못된 주소참조 등의 여러 문제가 발생할 수 있다. 해당 방식은 컴파일 단계에서 코드해석이 어렵도록 여러 가지 장치가 적용되므로 디컴파일을 하여 코드를 획득하여도 공격자가 읽기 쉽지 않도록 코드가 구성되어 있고, 이런 이유로 코드 분석시 공격자로 하여금 분석 시간의 증가를 야기할 수 있다[7][8].

2.3.2 APK 파일 자체에 적용하는 앱 보호 방식

APK 파일 자체에 적용하는 앱 보호 방식은 주로 SaaS(Security As a Service)형태를 띈다. 서비스 공급자는 클라우드 서비스를 통하여 안드로이드 앱 보호 방식을 제공하며, 사용자는 웹을 통해 APK 파일을 올리는 것으로 보호 기법이 적용된 APK 파일을 얻을 수 있다. 이 방식의 경우 주로 컴파일된 파일을 대상으로 하며, 파일 수정, 패킹, 암호화, 인코딩 등의 방법을 취하여 앱을 보호한다[9]. 이를 통해 정적 역공학 분석을 시도하는 공격자에게 원천적인 공격 차단 효과를 기대할 수 있다. 하지만 동적 가상 머신 환경을 이용해 앱 실행 환경을 꾸며 분석하는 경우 보호된 원본이 유출될 가능성이 있어 동적 분석 방지를 위한 추가적인 보안 모듈이 필요하다.

2.3.3 암호화를 이용한 앱 보호

이 방법은 앱 내의 컴파일된 파일이나 인증서와 같은 중요한 데이터를 보호하기 위한 방법으로 앱 실행 파일을 암호화 하는 방법이다. 앞서 언급한 두 가지 보호 방식에서 보안을 강화하기 위해 사용하기도 한다. 이 방식의 경우 암호 알고리즘에 안전성을 맡기지만 역공학을 통해 키 유도 알고리즘이나 초기 암호키 등이 노출될 소지가 있어 이에 대한 관리가 중요하다.

기존의 암호화를 이용한 역공학 방지 방법들을 분석하여 그 문제점들을 살펴볼 수 있는데, [10][11]과 같은 방식의 경우 스마트기기의 펌웨어의 수정과 키 관리의 문제점을 가지고 있다. 또한 키 유도를 위해 SIM 카드의 시리얼 번호와 서버가 생산한 salt값이 있어야한다. SIM 카드 시리얼 번호로 불리는 ICCID(Integrated Circuit Card IDentifier)를 키 생성에 사용해 각 SIM 카드마다 각기 다른 암호키로 앱을 암호화 할 수 있어 앱이 유출 되더라도 SIM과 함께 유출되지 않을 경우 역공학을 방지할 수 있다. 하지만 암호키로 사용하는 ICCID는 개인정보로 여겨질 수 있고, 악의적인 공격자가 스마트기기를 탈취하거나 악성 앱을 설치하도록 유도하는 등의 방

법을 통해 해당 정보를 쉽게 얻어 낼 수 있다[12].

제안한 방식을 위해서는 스마트기기 제조사에서 보안서비스를 위한 새로운 스마트기기 펌웨어 제작이 필요하고, 앱 마켓 유통사는 별도의 보안 기능을 서버에 추가해야 한다. 앱을 다운로드 할 때 마다 salt를 이용해 APK 파일 내의 DEX 파일을 암호화하고 다시 패키징하는 작업을 수행하기 때문에 앱 마켓 서버가 버거울 수 있고 또한 스마트 기기 측면에서는 각 앱마다 다른 salt값을 저장할 공간이 필요한데, 제조사가 지원하지 않는다면 현재 스마트기기에 적용하기에는 쉽지 않은 문제를 가지고 있다. 게다가 암호화를 통해 파일을 보호하는 시스템에서 암호키의 관리는 매우 중요하다. 암호키의 관리에는 통신사와 제조사 그리고 앱 마켓 유통사가 모두 관련되어 있기 때문에 그 비용이 적지 않게 든다. 새로운 펌웨어 업데이트 사이클이 길고 모든 단말이 업데이트를 할 수 없는 안드로이드 OS 특성상 적용이 쉽지 않다.

2.4 커스텀 DexClassLoader

Dalvik VM은 커스텀 클래스 로딩 기능을 지원한다[13]. 이 기능을 통해 안드로이드 기본 실행 블록인 DEX 파일을 내부 저장소 혹은 네트워크 상에서 읽어 들일 수 있으나, 이러한 기법은 일반적으로 흔하게 사용하지 않는다. 특히 네트워크 상에서 dexclassloader를 사용할 경우 정책위반 가능성이 높기 때문에 주의를 기울여야 한다[14].

dexclassloader는 주로 다음과 같은 이유로 사용한다. 단일 DEX 파일 내 참조될 수 있는 메서드의 총 개수는 65,536로 제한된다. 따라서 65,536 이상의 메서드 개수를 가진 앱은 이를 모두 나타내지 못하는데, 이럴 경우 다수의 DEX 파일을 생성한 뒤 dexclassloader를 이용해 앱 동작 중 불러오는 방법을 이용한다. 혹은 앱 동작 중 앱 기능의 확장이 필요할 경우 dexclassloader를 이용한다. 이 경우 앱 마켓을 통한 업데이트 없이 기능의 확장 및 수정이 이루어질 수 있다. 단적인 예로 소셜 네트워크 앱의 하나인 'facebook'은 APK 파일의 압축을 풀면 여러 개의 DEX 파일로 이루어져 있는 것을 확인할 수 있는데, 이는 담고 있는 기능이 많아 65,536개의 메서드로 기능을 모두 담을 수 없기 때문이다.

하지만 Fig. 2에서 볼 수 있듯이 안드로이드 런타임 환경 변화로 ART에서 실행되는 파일이 ELF 파일 포맷인 OAT로 바뀌면서 다수의 DEX 파일을 한 파일에 넣을 수 있기 때문에 dexclassloader의 필요성이 비교적 줄어들었다[6].

III. 보안SD를 이용해 유도한 키로 DEX 파일 암호화 보호 기법 설계

본 논문에서는 2.3.3 절에서 언급한 기존 기법의 문제점을 해결하기 위해 보안토큰으로 유도한 키를 이용하여 DEX 파일을 암호화하여 보호하는 방식을 제안한다. 이 방식은 안드로이드 Dalvik 또는 ART 환경에 구애 받지 않으나, 복호화의 경우에는 환경에 따라 복호화를 위한 후킹 지점이 달라질 수 있다.

3.1 보안토큰

Fig. 4.는 보안 서비스를 위한 보안 토큰으로 사용하는 SD 카드를 도식화한 것이다. 보안 토큰 기능을 하는 스마트카드가 내재된 SD 카드(이하 보안SD)는 대다수의 안드로이드에 존재하는 슬롯에 장착할 수 있는 장점이 있다. 보안 SD는 별도의 키주입장치로 PSK(Pre-Shared Key)를 주입받으며, 비정상적인 전압/온도/개봉/보호막제거 등을 방지하는 하드웨어 회로가 보안 SD 안의 스마트카드 칩 내부에 내장된다. 또한 보안 SD는 자체 전원 없이 스마트폰 배터리로부터 전력을 공급받으므로 소비전력을 최소화시켜야 한다. 키유도를 위해 사용하는 PSK는 플래시 영역에 위 보호기법을 통해 안전하게 쓰여져 있기 때문에 외부에서는 접근할 수 없다.

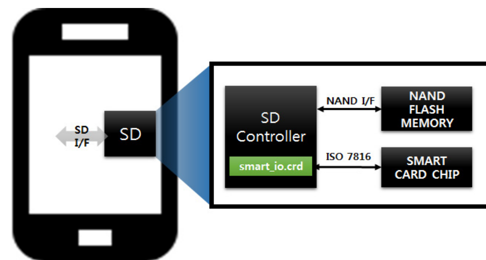


Fig. 4. Schematized Secure SD-Card

3.2 전체 구조

본 논문에서 제안하는 기법은 앱 마켓에 배포하기 전에 적용하는 방식을 취하고, 보안 SD를 지급받은 제한된 특정 집단 구성원이 앱을 지급받아 설치하여

실행하는 가정을 둔다. 본 기법을 적용하는 과정은 보안 SD를 이용한 APK 파일 내 DEX 파일 암호화 단계, 앱의 패키징 단계, 구동 단계로 구성된다.

3.2.1 DEX 파일 암호화

악의적인 공격자가 DEX 파일을 획득한 뒤 dex2jar 과 같은 디컴파일 도구를 이용하는 등 역공학을 시도하고자 할 때, 이를 원천적으로 봉쇄하기 위해 DEX 파일에 대해 암호화를 수행한다.

Fig. 5.는 암호키 유도방법에 대해 나타낸다. 보안 SD에 이미 주입된 PSK를 활용하여 2nd KEY를 유도하고 이 키를 이용해 DEX 파일을 암호화한다. Fig. 6.은 보안 SD를 통해 키를 유도하고 암호화하는 과정을 도식화한 것이다. 암호화를 하기 위해서는 보안 SD가 필요하므로, 암호화를 수행하는 기기에서 삽입된 SD 카드가 보안 SD인지 그 상태를 확인하는 과정이 수행된다. 키를 유도하기 위한 seed값은 DEX 파일 헤더에 위치한 20 바이트 길이의 signature 부분을 사용한다. 이 signature는 DEX 파일에서 magic, checksum, signature를 제외한 데이터로 SHA1 해시를 수행한 20 바이트에 해당하는 값이다. 이렇게 추출한 값을 특정 프로토콜에 담아 보안 SD로 전송한다. 보안 SD는 PSK로 seed를 암호화 한 후 그 결과값을 스마트기기로 전송한다. 이 값을 DEX 파일 암호를 위한 암호키로 사용한다.

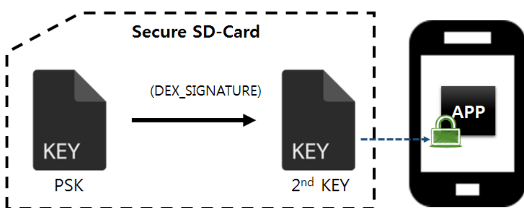


Fig. 5. Cryptography key derivation

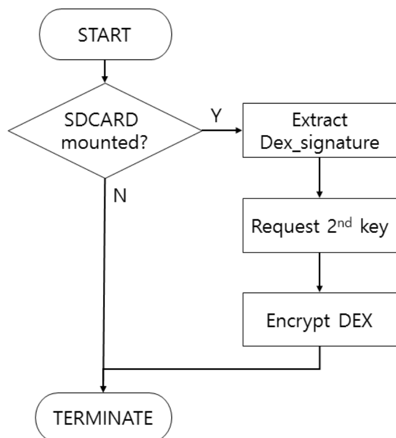


Fig. 6. Flowchart of key derivation and encryption process

제안하는 DEX 파일 암호화 기법은 DEX 파일 암호화를 위한 PSK의 노출을 하지 않을 뿐만 아니라 그 유도 방식 또한 스마트 기기에 노출하지 않는다. 유도를 통해 생성된 2nd KEY는 보안 SD가 없다면 알아낼 수 없다. 또 앱 패치 시 DEX 파일 헤더의 signature 값이 매번 달라지므로 암호키는 매번 다르게 적용할 수 있다. 이로써 앱이 유출되어 분석이 되더라도 보안 SD 없이는 DEX 파일을 복호화 할 수 없게 된다.

3.2.2 안드로이드 앱 패키징

2.1절에서 살펴본 바와 같이 안드로이드 앱을 실행하기 위해서는 실행 코드가 담겨진 DEX 파일이 필요하다. 하지만 암호화를 통해 보호된 DEX 파일은 Dalvik VM에서 해석할 수 없는 데이터로 구성되어 있기 때문에 복호화하여 실행해야 하는데, 이 복호화 시점을 제어할 수 있는 방법이 필요하다. 따라서 앱의 정상적인 실행과 암호화된 DEX 파일 처리를 위해 임의의 DEX 파일을 만들어 그 역할을 수행하도록 한다. 임의의 DEX 파일은 암호화된 DEX 파일을 복호화 하도록 절차를 만들고 복호가 완료된 DEX 파일을 앱 실행 매커니즘에 맞게 메모리에 적재하여 코드를 실행하도록 유도하는 역할을 수행한다.

네이티브 라이브러리를 사용하여 보호된 DEX 파일을 복호화하는데, 이 라이브러리는 보호된 DEX 파일로부터 seed를 추출하는 기능, 보안 SD와 통신하여 암호키 값 획득하는 기능, 복호화 시점 지연을 위한 dexclassloader 함수 후킹 기능, 후킹 지점에 복호화 루틴 삽입 기능을 가진다. 자세한 사항은 다음 절에서 설명한다.

네이티브 라이브러리로 복호화를 수행하는 이유는 네이티브 라이브러리로 작성한 코드가 자바 코드로 작성한 것보다 비교적 보안성이 강하며, dexclassloader의 후킹과 후킹 지점에 코드 삽입이 더욱 수월하기 때문이다.

보호된 DEX 파일 복호화 루틴이 Fig. 7.에서와

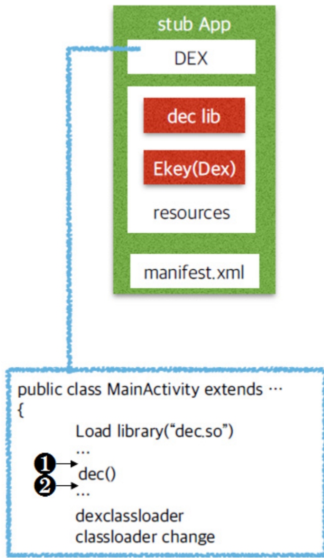


Fig. 7. Typical protected app

같이 특정 함수 호출 위치(dec() 호출)에서 이루어지면 공격자는 행위 분석을 통해 DEX 파일 복호화 동작을 수행하는 위치를 발견하기 수월하고, ❷ 부분에 DEX 파일을 복사하는 코드를 삽입하여 복호화된 DEX 파일을 얻을 수 있다.

그러나 네이티브 코드로 되어 있는 DEX 파일의 메모리 적재 기능을 수행하는 함수를 후킹하여 복호화 루틴을 삽입한다면 분석가는 앱의 코드상에서 DEX 파일 복호화 위치를 알아채기 어렵다. 또한 복호화 함수를 직접적으로 노출하지 않고 여러 방법으로 동적으로 그 복호화 루틴을 생성할 수 있으므로 DEX 파일을 보호하는데 있어 효율적이다. 이후 DEX 파일을 메모리로 적재하는 dexclassloader 호출시 API의 주된 수행에 앞서 후킹을 통해 삽입한 코드가 실행되어 복호화가 수행된다.

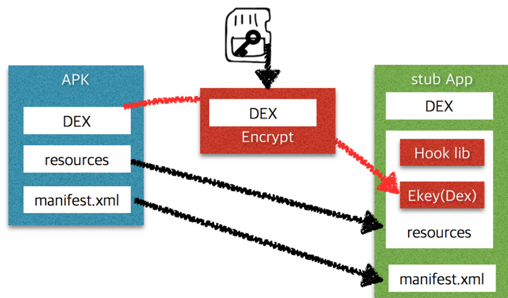


Fig. 8. Stub App generation

따라서 Fig. 8.과 같이 임의의 DEX 파일을 담고있는 스텁(stub) 앱은 수정된 AndroidManifest, 임의의 DEX, 보호된 DEX, 후킹을 위한 루틴이 포함된 네이티브 라이브러리, 기타 앱 구성 파일로 구성된다.

3.2.3 복호화

구글 플레이스토어를 통해 혹은 폐쇄 마켓을 통해 Fig. 9.과 같이 구성된 앱이 배포되어 안드로이드 기반 스마트 기기에 설치되어 동작한다. Fig. 10.에 나타낸 절차대로 다음과 같은 과정을 거친다.

안드로이드 앱이 실행되면 스텁 앱의 DEX 파일이 메모리에 적재되고 처음 실행할 부분에 대한 정보를 담고 있는 AndroidManifest.xml 파일을 참조하여 그 코드를 실행한다. 코드가 실행되면서 네이티브 라이브러리(hook.so)를 메모리에 적재하고 라이브러리 내 함수를 호출한다. 이 함수는 dexclassloader API 흐름에서 심볼값을 통해 메

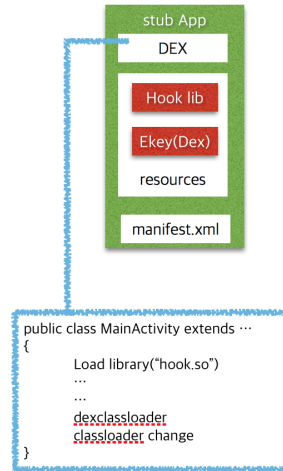


Fig. 9. form of stub app

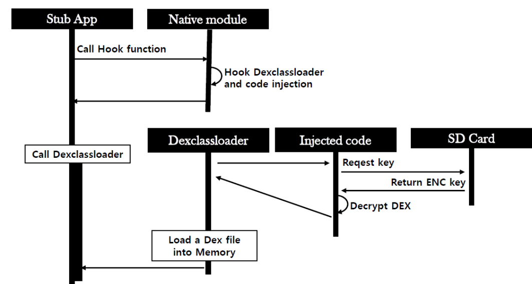


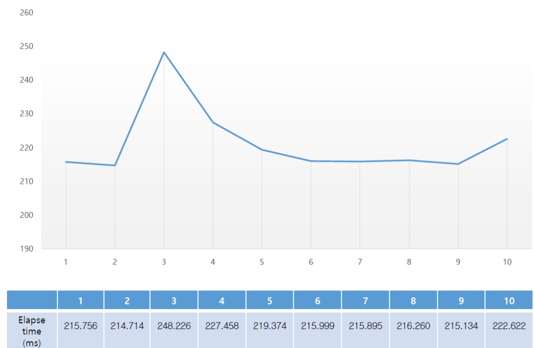
Fig. 10. Procedure for Encrypted DEX

모리에 적재된 주소 값을 얻어와 적당한 부분을 후킹한 후 dexclassloader 특정 시점에서 코드 실행 전 특정 메모리로 실행흐름을 전환시킨다. 후킹이 이루어진 후 dexclassloader를 실행한다면 후킹한 부분에서 추가로 코드가 실행되며 복호화를 수행한다.

dexclassloader 흐름을 후킹해 삽입한 특정 메모리의 코드는 Fig. 6.에서 암호화로 나타낸 과정을 다시 거친다. 먼저 보안 SD 삽입 여부 확인과 정상적인 보안 SD 여부를 확인한다. 암호키의 seed값으로 사용한 보호된 DEX 파일의 signature를 추출하고 특정 프로토콜에 담아 보안 SD로 전송한다. 보안 SD는 PSK를 이용해 signature를 암호화하여 그 결과값을 전송한다. 이렇게 유도된 암호키를 이용하여 보호된 DEX 파일을 복호화한다. 복호화된 DEX 파일을 메모리로 적재하고 가장 먼저 실행할 코드를 실행하도록 한다.

IV. 실험결과

본 논문에서는 제안하는 기법을 적용하기 위한 대상 안드로이드 스마트 기기로 LG 전자의 스마트폰인 G5를 선정하였다. 구글 레퍼런스 기기에는 SD 카드 슬롯이 없기 때문이다[15]. 대상 스마트 기기의 안드로이드 버전은 7.0이며, 커널 버전은 3.18.31이다. 제안하는 방식의 오버헤드를 알아보기 위해 보안 SD로 키가 유도되는 시간을 알아본다. DEX 파일을 복호화하거나 후킹하여 코드를 삽입하는 오버헤드는 여러 앱 보호 솔루션에서 주로 적용하는 방식이므로 그 오버헤드는 제외한다. Fig. 11.은 DEX 파일 복호화 수행 전 복호화를 위해 키를 유도하는 루틴 전체에 대한 소요시간을 측정한 결과이다. 소요시간은 평균 221.14 ms로 사용자가 불편을 느낄만한 수치에 미치지 않는 것을 볼 수 있다.



	1	2	3	4	5	6	7	8	9	10
Elapse time (ms)	215.756	214.714	248.226	227.458	219.374	215.999	215.895	216.260	215.134	222.622

Fig. 11. Decryption key request time

V. 결 론

본 논문은 특정 사용자 집단이 사용하는 안드로이드 스마트 기기에서 구동하는 앱의 보호를 위한 방법으로 보안 토큰을 이용하여 보호하도록 제안하였다. 본 논문에서 제안한 기법은 일반 사용자를 위한 방식이라기보다는 국가기관 또는 기업 고위직을 위한 방식으로 전용 스마트폰과 같이 보안 목적을 위해 금전적 비용 및 사용 시 불편함을 수용할 수 있는 집단군에 한정한다. 보안 토큰은 PSK가 있어 앱의 signature를 암호화 하여 각 앱마다 다른 암호키를 유도하였다. 또한 보호된 DEX 파일의 메모리 적재를 위해 호출하는 dexclassloader API를 후킹하여 복호화 과정을 삽입함으로써 DEX 파일 리패키징을 통한 복호 DEX 파일의 획득을 어렵게 하였다. 또한 보안 토큰을 키 유도에 적용함으로써 파일을 직접 보안 토큰에 암·복호를 요청하는 방식보다 소모 시간 측면에서 효율적으로 사용하였다. 이를 통해 앱 유출시 보안 토큰 없이 분석이 불가능하도록 원천적인 차단을 기대 할 수 있다.

하지만 본 제안기법은 모든 보안 토큰에 동일한 키가 삽입되므로 공격자가 보안 토큰을 획득한 경우 Known plaintext attack을 통해 키를 추출한다면 모든 앱을 복호화 할 수 있는 문제점을 가지고 있다. 또한 dexclassloader 이후부터 파일시스템 내 복호 DEX 파일은 반드시 존재하므로 race condition을 이용해 이를 가로챌 수 있는 문제점을 가지고 있다.

이를 위해 분실한 보안 토큰에 키 소거 명령을 보내 키 추출을 사전에 막는 방식 등이 필요할 것으로 보인다.

ART가 안드로이드 실행환경으로 자리 잡은 이후 매년 OS 버전이 발표될 때마다 안드로이드 실행환경은 최적화 과정을 거치며 이에 대해 꾸준히 패치노트에서 언급된다. 기존 Dalvik에서부터 서비스하던 안드로이드 앱 보호 솔루션은 대부분 Dalvik이 주된 실행환경일 때 사용되던 방식으로 현재까지 그대로 사용되는 것으로 보인다. 구글은 안드로이드 성능 향상을 위해 새로운 런타임 환경 최적화에 노력을 기울이지만 앱 보호가 적용된 앱은 그 노력의 결과를 온전히 누리지 못한다. 따라서 새로운 실행환경에서 동작하는 안드로이드 앱 보호 방식 연구가 필요해 보인

다. 또한 최근 들어 보안이 필요한 뱅킹 앱 등은 중요한 역할을 수행하는 코드를 네이티브 코드로 내리고, 자바코드에는 단순히 네이티브 코드를 호출하는 껍데기 역할을 수행하도록 변화하는 추세이다. 이 추세에 비춰 C/C++로 작성된 네이티브 라이브러리를 보호할 수 있는 연구 또한 필요할 것으로 판단된다.

References

〔1〕 Maria Garnaeva, Fedor Sinitsyn, Yury Namestnikov, Denis Makrushin and Alexander Liskin, "Kaspersky Security Bulletin:OVERALL STATISTIVS FOR 2016," Kaspersky Lab, Dec. 2016

〔2〕 Kimberly Tam, Ali Feizollah, Nor Badrul Anuar, Rosli Salleh and Lorenzo Cavallaro, "The Evolution of Android Malware and Android Analysis Techniques," ACM Computing Surveys (CSUR), vol. 49, no. 4, pp. 1-41, Feb. 2017

〔3〕 Se Young Lee, Jin Hyung Park, Moon Chan Park, Jae Hyuk Suk, Dong Hoon Lee, "A Study on Deobfuscation Method of Android and Implementation of Automatic Analysis Tool," *Jonornal of The Korea Institute of information Security & Cryptology*, 25(5), pp. 1201-1215, Oct. 2015

〔4〕 Yeongung Park, "We Can Still Crack You! General Unpacking Method for Android Packer (no root)," DEFCON, Mar. 2015

〔5〕 "Android KitKat unveiled in Google surprise move," BBC, 3 Sep, 2013

〔6〕 Anwar Ghuloum, Brian Carlstrom and Ian Rogers, "ART: ANdroid's Runtime Evolved," Google I/O 2014, Jun. 2014

〔7〕 J.Kim and E. Lee, "A strategy of effectively applying a control flow obfuscation to programes," J. Korea Soc. Comput. Inf., vol 16, no. 6, pp. 41-50 Jun. 2011

〔8〕 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

〔9〕 AppSuit, http://premium.appsu.it

〔10〕 Kim. Hee Moon, "Protection Framework for Android Application by Encrypting DEX files," The Graduate School of Hanyang University, Feb. 2011

〔11〕 JungHyun Kim, Kang Seung Lee, "Robust Anti Reverse Engineering Technique for Protecting Android Applications using the AES Algorithm," Journal of KIISE, Vol.42, No. 9, pp. 1100-1108, Sep. 2015

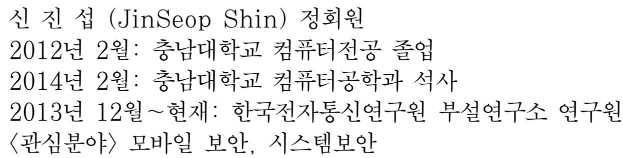
〔12〕 Privacy Statement, http://inside.olleh.com/html/infoView.asp

〔13〕 https://developer.android.com/reference/dalvik/system/DexClassLoader.html

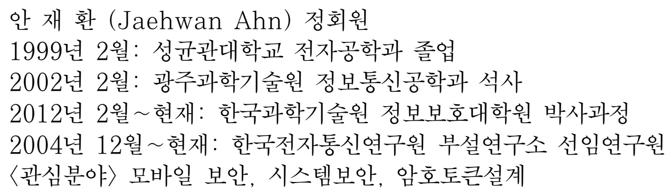
〔14〕 https://play.google.com/intl/ko_ALL/about/index.html

〔15〕 https://gadgetstouse.com/gadget-tech/sd-card-explained/32377

〈저자소개〉



신 진 섭 (JinSeop Shin) 정회원
2012년 2월: 충남대학교 컴퓨터전공 졸업
2014년 2월: 충남대학교 컴퓨터공학과 석사
2013년 12월~현재: 한국전자통신연구원 부설연구소 연구원
〈관심분야〉 모바일 보안, 시스템보안



안 재 환 (Jaehwan Ahn) 정회원
1999년 2월: 성균관대학교 전자공학과 졸업
2002년 2월: 광주과학기술원 정보통신공학과 석사
2012년 2월~현재: 한국과학기술원 정보보호대학원 박사과정
2004년 12월~현재: 한국전자통신연구원 부설연구소 선임연구원
〈관심분야〉 모바일 보안, 시스템보안, 암호토큰설계