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.2017.27.6.1361

지문 인증과 동적 로딩을 이용한 안드로이드 애플리케이션 코드 보호 기법

류 환 일,† 석 재 혁, 박 진 형, 이 동 훈‡
고려대학교 정보보호대학원

Android Application Code Protection Scheme Using Fingerprint Authentication and Dynamic Loading

Hwahn-il Lyoo,† Jae-Hyuk Suk, Jin-Hyung Park, Dong-Hoon Lee‡
Graduate School of Information Security, Korea University

요 약

비공개 애플리케이션 또는 핑거프린팅 기법이 적용된 애플리케이션을 외부의 공격자가 복사해 가져가는 경우 비공개 정보가 유출되거나 정당한 사용자가 애플리케이션의 불법 재배포자로 오인될 수 있어 심각한 보안 문제가 초래될 수 있다. 이러한 문제를 해결하기 위하여 본 논문에서는 지문 인증과 동적 로딩을 이용한 안드로이드 애플리케이션 코드 보호 기법을 제안한다. 본 논문에서 제안하는 기법은 하나의 애플리케이션을 CLR(Class LoadeR)과 SED(SEperated Dex)로 구성한다. CLR은 SED를 동적으로 로드하는 기능을 가진 APK 파일이며, SED는 애플리케이션 실행에 필요한 클래스들이 포함된 파일이다. SED는 암호화된 상태로 스마트폰 내부에 보관되며, 사용자는 지문 인증을 성공한 경우에만 SED를 복호화할 수 있다. 본 논문에서 제안하는 기법은 사용자의 스마트폰을 물리적으로 획득한 공격자로부터 애플리케이션 코드를 안전하게 보호할 수 있다.

ABSTRACT

If an external attacker takes from a victim's smartphone a copy of a secret application or an application to which fingerprinting technique is applied, secret information can be leaked or the legitimate user can be misunderstood as an illegal redistributor, which results in a serious security problem. To solve this problem, this paper proposes an Android application code protection scheme using fingerprint authentication and dynamic loading. The proposed scheme divides one application into CLR(Class LoadeR) and SED(SEperated Dex). CLR is an APK file with the ability to dynamically load the SED, and the SED is a file containing the classes required to run the application. The SED is stored inside the smartphone after being encrypted, and the SED can be decrypted only if the user is successfully authenticated using his or her fingerprint. The proposed scheme can protect the application code from the attacker who physically acquired user's smartphone.

Keywords: Android Application Code Protection, Fingerprint Authentication, Dynamic Loading

I. 서 론

오늘날 안드로이드 애플리케이션 시장의 성장과 함께 애플리케이션 불법 복제 사례가 증가하고 있다[1]. 일반적으로 불법 복제는 본인 소유의 애플리케이션을 외부로 유출하는 형태로 이루어지지만, 본 논

Received(09. 12. 2017), Modified(1st: 10. 26. 2017, 2nd: 11. 16. 2017), Accepted(11. 18. 2017)

† 주저자, eip343@gmail.com
‡ 교신저자, donghlee@korea.ac.kr(Corresponding author)

문에서는 타인의 스마트폰 내부에 존재하는 애플리케이션을 불법으로 복제하여 가져오는 행위에 초점을 맞춘다. 이는 비공개 애플리케이션은 공개된 마켓에서 구할 수 없기 때문에 타인의 스마트폰 내부에서 찾아야 하기 때문이다. 또한, 핑거프린팅 기법이 적용된 애플리케이션의 경우에도 본인의 애플리케이션을 무단으로 재배포하면 본인이 추적을 당할 수 있기 때문에 추적을 피하기 위해서는 타인의 애플리케이션을 배포해야 한다. 본 논문에서는 지문 인증과 동적 로딩을 이용하여 이러한 유형의 공격자로부터 애플리케이션의 무단 복제를 방어하는 기법을 제안한다.

본 논문에서는 '지문'과 '핑거프린트'가 각각 다른 의미로 사용된다. '지문'은 사용자의 손가락에 있는 고유한 문양을 의미하며, '핑거프린트'는 소프트웨어 불법 재배포자를 추적하기 위한 목적으로 소프트웨어에 은닉한 구매자 식별 값을 의미한다.

본 논문의 구성은 다음과 같다. 2장에서는 안드로이드 애플리케이션 코드 보호와 관련된 배경 지식 및 연구에 대하여 소개한다. 3장에서는 안드로이드 애플리케이션에 대한 위협 모델을 제시하며, 4장에서는 애플리케이션 코드 보호를 위하여 제안하는 기법에 대하여 서술한다. 5장에서는 제안 기법에 대한 안전성을 설명하고, 6장에서는 성능 평가 결과를 제시하며, 7장에서 결론을 맺는다.

II. 배경 지식 및 관련 연구

2.1 안드로이드 키스토어

안드로이드 키스토어는 암호 키의 무단 사용을 막아주는 역할을 수행한다. 안드로이드 애플리케이션은 키스토어 서비스에 암호 키 생성을 요청할 수 있다. 이렇게 생성된 암호 키는 KEK(Key-Encryption Key)로 암호화되어 보관되며, 암호 키를 생성한 애플리케이션 이외에는 해당 암호 키를 통한 암호화 및 복호화 작업을 키스토어에 요청할 수 없다.

일반 영역으로부터 분리된 안전한 실행 환경을 TEE(Trusted Execution Environment)라고 부른다[2]. KEK는 TEE에 보관되며, TEE 외부로 추출될 수 없다. 또한, 암호화 및 복호화 작업은 KEK로 암호화된 암호 키를 TEE로 가져와 복호화한 후에 TEE에서 수행된다. 루트 권한을 가진 사용자라 할지라도 TEE에 접근할 수 없으므로 사용자는 KEK 값 및 암호 키를 알 수 없다. Fig. 1.은 안드로이드 키스토어를 이용한 암호화 및 복호화 수행 과정을 나타낸다.

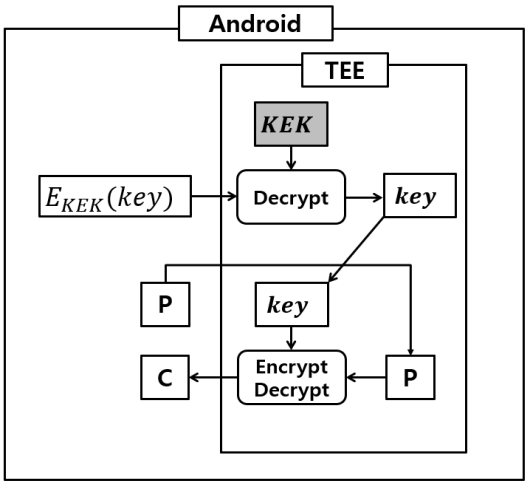

Fig. 1. Encryption and decryption process with Android keystore and TEE

루트 권한을 가진 공격자는 암호 키를 알 수 없지만, 암호 키를 사용할 수는 있다. 키스토어는 암호 키를 생성한 애플리케이션이 암호화 및 복호화 작업을 키스토어에 요청하는 것을 허용하기 때문에 루트 권한을 가진 공격자는 해당 애플리케이션을 실행하는 방법으로 암호 키를 사용할 수 있다.

루트 권한이 있는 공격자가 암호 키를 사용하지 못하게 하려면 추가적인 접근 제어가 필요하다. 안드로이드는 이러한 접근 제어 수단으로 PIN, 비밀번호, 패턴, 지문 인증을 제공한다[3]. 그러나 PIN, 비밀번호, 패턴의 경우 충분한 엔트로피(entropy)를 갖지 못하면 루트 권한이 있는 공격자가 전수 조사 공격을 통하여 인증을 통과할 수 있다는 위험성이 존재한다[4]. 특히 PIN과 패턴의 경우 비밀번호보다 경우의 수가 적어 전수 조사 공격에 훨씬 취약하다. 반면, 사용자의 지문 정보는 TEE에서만 접근 가능하며, 지문 인증은 TEE에서 수행되므로 루트 권한을 가진 공격자라고 할지라도 지문 인증을 우회할 수 없다. 또한, 사용자 입장에서는 충분한 엔트로피를 지닌 비밀번호를 입력하는 것보다 지문 인증을 수행하는 것이 훨씬 더 편리하다. 따라서 보안성과 편리성을 동시에 고려하는 경우 지문 인증을 이용하여 암호 키에 대한 접근 제어를 수행하는 것이 적합하다.

2.2 안드로이드 지문 인증

안드로이드는 키스토어와 TEE를 활용하여 안전한 지문 인증 기능을 제공한다[5]. 애플리케이션은 KeyGenParameterSpec.Builder를 통하여 암호키의 속성을 정의(암호 알고리즘, 운영모드, 사용자 인증 필요 여부 등)할 수 있다. 암호 키의 속성을 정의한 후에는 KeyGenerator.generatekey를 통하여 키스토어가 보호하는 암호 키를 생성할 수 있다.

암호 키를 생성한 후에는 getSystemService를 통하여 핑거프린트 매니저 서비스를 가져올 수 있으며, FingerprintManager.authenticate를 통하여 지문 인증을 요청할 수 있다. 지문 인증은 TEE에서 수행된다. 애플리케이션은 지문 인증에 성공한 경우에만 앞서 생성한 암호 키를 사용할 수 있다.

본 논문에서 제안하는 기법은 지문 인증을 이용하여 암호 키를 보호한다. 지문 인증은 암호 키에 대한 접근 제어 목적으로만 사용되며, 암호 키의 생성 그 자체에는 지문과 관련된 정보를 사용하지 않는다. 본 논문에서 해당 암호 키는 따로 분리한 클래스(SED)를 암호화하는 데 사용한다.

2.3 안드로이드 애플리케이션 복사 방지 기법

본 논문에서 제안하는 기법은 공격자가 애플리케이션을 복사하여 가져가지 못하도록 하는 기법이다. 이전에 제안된 애플리케이션 복사 방지 기법들을 살펴보면 다음과 같다.

[6]은 클래스를 안전한 공간에 보관하고, 지정된 프로세스만이 안전한 공간에 보관된 클래스를 동적으로 로드할 수 있도록 하는 기법을 제안한다. 이 기법은 지정된 프로세스 이외에는 클래스에 접근할 수 없도록 하여 공격자가 클래스를 복사하지 못하도록 한다. 그러나 이 기법은 커널의 변경이 필요하여 사용자 편의성이 부족하다는 단점이 있다.

[7]은 정당한 사용자가 인증을 통과한 경우에만 클래스를 서버로부터 동적으로 로드할 수 있는 기법을 제안한다. 그러나 애플리케이션을 실행할 때마다 매번 클래스를 서버로부터 다운받아야 하므로 네트워크 통신 비용이 증가한다는 단점이 있다.

[8]은 DEX(Dalvik EXecutable) 파일을 암호화한 상태로 배포하고, 실행 시에만 복호화하는 방식을 통하여 애플리케이션의 복사를 방지하는 기법을 제안한다. DEX 파일 암호화에는 SIM(Subscriber Identity Module) 카드 시리얼 번호와 솔트(salt) 값을 이용한다. 그러나 SIM 카드 시리얼 번호는 공격자가 쉽게 얻을 수 있으며, 솔트를 안전하게 보관하는 방법에 대한 고려가 없다는 문제점(격리공간에 저장되어 안전하게 보관된다고 가정함)이 존재한다.

본 논문에서 제안하는 기법은 커널의 변경이 필요하지 않고, 클래스를 스마트폰 내부에 암호화하여 저장하므로 추가적인 네트워크 통신 비용이 발생하지 않으며, 루트 권한을 가진 공격자도 지문 인증에 성공하지 못하면 암호 키에 접근할 수 없다는 점에서 기존에 제안된 기법에 비하여 의미를 가진다.

2.4 소프트웨어 핑거프린팅 기법

소프트웨어 핑거프린팅 기법은 소프트웨어 내부에 구매자 식별 값을 은닉하는 기법이다[9]. 이는 애플리케이션이 공격자에 의하여 무단으로 재배포된 경우, 개발자가 애플리케이션 속에 은닉된 구매자 식별 값을 확인하여 최초 무단 재배포자를 추적할 수 있도록 한다.

소프트웨어 워터마킹 기법은 소프트웨어 내부에 저작권 정보를 은닉하는 기법이다[10]. 소프트웨어 워터마킹 기법은 제거 공격(워터마크를 제거하는 공격), 변형 공격(소프트웨어에 변형을 가하여 워터마크가 훼손되도록 하는 공격), 추가 공격(올바르지 않은 워터마크를 추가하는 공격)에 안전해야 한다.

한편, 핑거프린팅 기법은 워터마킹 기법과 달리 소프트웨어마다 서로 다른 식별 값을 은닉하여야 한다. 따라서 핑거프린팅 기법은 핑거프린트가 삽입된 소프트웨어들의 차이점을 찾아 핑거프린트의 위치를 파악할 수 있는 공모 공격에도 안전해야 한다.

안드로이드를 대상으로 한 워터마킹 기법으로는 [11], [12] 등이 제안되었다. 이러한 워터마킹 기법을 활용하여 핑거프린팅 기법을 설계할 수 있으며, 이를 수행하는 일반적인 방법은 다음과 같다[13].

(1) 소프트웨어마다 동일한 값을 삽입하는 대신 서로 다른 값을 삽입함으로써 워터마킹 기법을 핑거프린팅 기법으로 대체할 수 있다.
(2) 위 소프트웨어에 난독화(obfuscation) 또는 랜덤화(randomization)를 적용하는 경우, 공모 공격에 안전한 핑거프린팅 기법을 설계할 수 있다.

III. 위협 모델

공격자의 목표는 타인의 안드로이드 스마트폰 내부에 존재하는 애플리케이션 코드를 복사해 가져오는 것이다. 공격자는 비공개 애플리케이션 또는 핑거프린팅 기법이 적용된 애플리케이션을 공격 대상으로 삼는다. 단, 공격 대상 애플리케이션은 정당한 사용자에 의해 설치되었으며, 최초 1회 정당한 사용자에 의해 실행이 완료되었다고 가정한다. 또한, 공격자는 타인의 스마트폰을 절도하거나 분실된 스마트폰을 획득하거나 혹은 타인으로부터 기기를 직접 구매하는 방법을 통하여 공격 대상 애플리케이션이 설치된 스마트폰을 물리적으로 획득하였다고 가정한다.

만일 피해자가 지문을 이용한 잠금 화면을 자의로 해제한 상태에서 스마트폰을 분실하였거나 절도를 당하였다면, 공격자가 획득한 스마트폰에는 잠금 화면이 존재하지 않을 것이다. 또한, 피해자가 기기를 판매하기 전에 직접 공장 초기화를 수행한 경우에도 잠금 화면이 존재하지 않을 것이다. 공장 초기화를 수행한 경우에도 애플리케이션 코드가 반드시 영구적으로 삭제되는 것은 아니기 때문에 공격자는 여전히 공격을 수행할 수 있는 가능성이 존재한다[14].

공격자가 획득한 스마트폰에 지문을 이용한 잠금 화면이 활성화되어 있는 경우에는 공격자는 잠금 화면을 우회하여 스마트폰 내부 데이터에 접근할 수 있다고 가정한다. 잠금 화면 우회 방법의 예로 플래시 메모리에 물리적으로 접근하여 데이터를 추출하는 방법이 있다[15]. 단, 이 방법은 스마트폰 기기 전체가 암호화되어 있지 않은 경우에 사용 가능하다.

본 논문에서는 아래와 같은 능력을 지닌 공격자가 등장한다.

- '루트 공격자(root attacker)'는 타인의 스마트폰에서 루트 권한을 행사할 수 있으며, 메모리에 존재하는 값을 언제든지 가로챌 수 있다.
- '복원 불가 루트 공격자' 또는 'no R 공격자(no Restoration root attacker)'는 타인의 스마트폰에서 루트 권한을 행사할 수 있지만 스마트폰에서 삭제된 파일은 복원할 수 없다.

'루트 공격자'가 'no R 공격자'보다 더 강력한 능력을 가지고 있다. 본 논문에서 제안하는 기법은 본 장에서 제시한 가정 하에서는 '루트 공격자'로부터 안전하지만, 다른 논문의 제안 기법과 본 논문의 안전성을 비교하기 위하여 'no R 공격자'를 추가하였다.

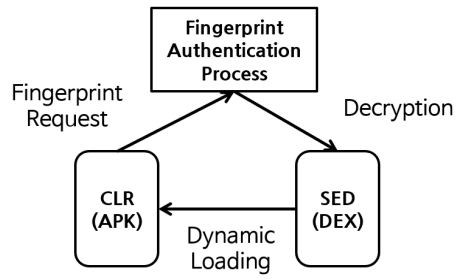

Fig. 2. Overview of the proposed scheme

스마트폰에 일시적으로 저장되었다가 삭제된 파일들은 파일 카빙(carving) 기법 등을 통하여 추후 복원할 수 있는 가능성이 존재한다. '루트 공격자'는 이를 고려한 공격자 모델이며, 'no R 공격자'는 이를 고려하지 않은 공격자 모델이다. 본 논문에서는 보호 대상 코드를 암호화하지 않은 상태로 스마트폰 내부에 일시적으로 저장하는 경우 '루트 공격자'에 안전하지 않다고 판단한다.

IV. 제안 기법

4.1 기법 개요

본 논문에서 제안하는 기법의 개요는 Fig. 2.와 같다. 제안 기법에서 하나의 애플리케이션은 CLR(Class LoadeR)과 SED(SEperated Dex)로 구성된다. CLR은 '지문 인증으로 보호되는 암호키'로 SED를 복호화하여 메모리에 동적으로 로드하는 기능을 가진 APK(Android application PacKage) 파일이다. SED는 애플리케이션 실행에 필요한 클래스들이 포함된 DEX 파일로써, '지문 인증으로 보호되는 암호 키'로 암호화되어 스마트폰 내부에 저장된다. CLR은 동적으로 로드된 SED를 실행하기 위하여 자바 리플렉션(reflection) 기법을 이용한다.

제안 기법에서는 지문 인증을 통해 암호 키를 보호한다. 지문 인증은 TEE에서 수행되고, 암호 키는 TEE에서 실행되는 서비스인 키마스터가 관리하므로 루트 권한을 가진 공격자도 지문 인증에 성공하지 못하면 암호 키를 사용할 수 없다.

본 장 2절에서는 지문 인증 API 및 암호 키에 대하여 서술하고, 3절에서는 CLR이 SED를 동적으로 로드하기 위하여 사용하는 API에 대하여 설명한다. 4절에서는 핑거프린팅 기법이 적용된 애플리케이션

을 배포하는 절차에 대하여 설명하고, 5절에서는 비공개 애플리케이션을 배포하는 절차에 대하여 서술한다. 마지막으로 6절에서는 애플리케이션을 실행하는 절차를 설명한다.

4.2 지문 인증 API 및 암호 키

제안 기법은 지문 인증을 위하여 2장 2절에서 설명한 KeyGenParameterSpec.Builder, KeyGenerator.generatekey, getSystemService, FingerprintManager.authenticate를 사용한다.

지문 인증은 SED를 암호화(CLR 최초 실행 시)하거나 복호화하기 직전에 수행하며, 지문 인증에 실패한 경우에는 SED를 암호화하거나 복호화할 수 없다. 지문 인증에 성공한 경우에는 암호 키를 사용하여 SED를 암호화하거나 복호화한다. 본 논문의 제안 기법에서 암호 알고리즘은 AES를 사용하고, 운영 모드는 CBC를 사용하며, 패딩은 PKCS#7을 사용한다.

4.3 동적 로딩 API

동적 로딩 기능을 수행하는 API로는 DexFile, DexClassLoader, PathClassLoader, InMemoryDexClassLoader가 있다.

4.3.1 DexFile

DexFile은 API 레벨 26 이후 사용이 중단될 예정(deprecated)이다. 이 함수를 사용하면 애플리케이션의 성능이 저하되고 바이트코드가 올바르지 않게 실행될 수 있기 때문에 안드로이드 표준에서는 애플리케이션이 이 함수를 직접 호출하는 것을 권장하지 않는다[16]. 따라서 이 함수는 제안 기법에서 사용하지 않는다.

4.3.2 DexClassLoader

DexClassLoader 함수는 API 레벨 3부터 사용할 수 있으며, 4개의 인자를 입력받는다. 4개의 인자는 각각 dexPath, optimizedDirectory, librarySearchPath, parent이다. dexPath는 클래스를 담고 있는 JAR, APK 또는 DEX 파일의 경로이고, optimizedDirectory는 최적화된 DEX 파일이 저장될 경로로 null 값이 들어갈 수 없다[17].

DexClassLoader 함수는 동적으로 DEX를 로드함과 동시에 이를 최적화하여 스마트폰 내부에 저장하기도 한다. 즉, 복호화된 SED에 대한 최적화된 DEX가 스마트폰 내부에 저장된다는 의미이다. 최적화된 DEX 파일은 생성 직후 삭제할 수 있으므로 'no R 공격자'로부터는 안전하지만, 삭제된 파일을 복원할 수 있는 능력을 가진 '루트 공격자'는 복호화된 SED 파일을 가져갈 수 있다.

그러나 API 레벨 26부터는 optimizedDirectory 인자에 들어가는 값에 상관없이 기본 시스템 경로에 최적화된 DEX 파일을 생성하며([18], [19]), CLR은 시스템 경로에 접근할 수 없다. 즉, 최적화된 DEX는 CLR이 삭제할 수 없기 때문에 DexClassLoader 함수를 이용하여 SED를 로드하는 경우 스마트폰 내부에 최적화된 DEX 파일이 남는다([19], [20]). 그러므로 DexClassLoader 함수 사용 시 API 레벨 26부터는 '루트 공격자'와 'no R 공격자' 모두 최적화된 SED 파일을 복사해 가져갈 수 있다는 문제점이 존재한다.

4.3.3 PathClassLoader

PathClassLoader 함수는 API 레벨 1부터 사용할 수 있으며, DexClassLoader와 달리 optimizedDirectory 인자가 존재하지 않지만[21] 최적화된 DEX 파일이 항상 기본 시스템 경로에 생성되도록 설계되어 있다[19]. 따라서 이 함수도 시스템 경로에 DEX 파일을 생성한다는 점에서 DexClassLoader와 유사한 문제점을 지니고 있기 때문에 제안 기법에서 PathClassLoader 함수는 사용하지 않는다.

4.3.4 InMemoryDexClassLoader

InMemoryDexClassLoader 함수는 API 레벨 26(2017년 8월 21일에 출시된 안드로이드 버전 8.0에서 사용되는 레벨[22])부터 사용할 수 있으며, 2개의 인자를 입력받는다. 2개의 인자는 각각 dexBuffer, parent이다. 이 중 dexBuffer는 DEX 파일의 내용이 담긴 버퍼이다[23]. InMemoryDexClassLoader 함수는 DexClassLoader 함수 또는 PathClassLoader 함수와 달리 최적화된 DEX 파일을 생성하지 않는다.

또한, DexClassLoader 함수 또는 PathClassLoader 함수는 복호화된 DEX 파일의 경로를 인자로 주어야 하지만 InMemoryDexClassLoader의 경우에는 암호화된 Dex 파일을 버퍼에 올린 후 메모리상에서 복호화하여 로드할 수 있으므로 스마트폰 내부에 복호화된 SED를 생성할 필요도 없다. 따라서 InMemoryDexClassLoader 함수를 사용하여 SED를 로드하는 경우 '루트 공격자'와 'no R 공격자' 모두 복호화된 SED를 획득할 수 없다.

4.3.5 동적 로딩 API 비교

위에서 살펴본 API들을 정리하면 Table 1.과 같다. Table 1.에서 O 표기는 해당 API를 사용하여 SED를 동적으로 로드한 경우에 '루트 공격자' 또는 'no R 공격자'가 복호화된 SED 코드를 탈취할 수 없어 안전함을 의미하며, X 표기는 공격자가 복호화된 코드를 탈취할 수 있어 안전하지 않음을 의미한다. 안드로이드 표준에서는 DexFile 함수의 사용을 권장하지 않는다. DexClassLoader 함수를 사용하는 경우 API 레벨 25 이하에서는 'no R 공격자'로부터는 안전하지만, API 레벨 26부터는 no R 공격자'로부터도 안전하지 않다(4.3.2 참고). PathClassLoader 함수를 사용하는 경우 '루트 공격자'와 'no R 공격자' 모두에게 취약하다. 이와 달리 InMemoryDexclassLoader 함수를 사용하는 경우 '루트 공격자'와 'no R 공격자' 모두에게 안전하다. 따라서 제안 기법에서는 동적 로딩을 위하여 InMemoryDexClassLoader 함수를 사용한다.

Table 1. Security of Class Loader API

API	API level	root	no R
DexFile	Deprecated	-	-
DexClassLoader	3	X	O〈26
PathClass-Loader	1	X	X
InMemory-DexClassLoader	26	O	O

Table 2. The notation used in the protocol

Notation	Description
$User$	User
$SEDServer$	SED Server
r_{User}, $r_{SEDServer}$	Random number generated by User and SED Server
ID_{CLR}	Application ID of CLR
$S_{User}()$, $S_{SEDServer}()$	A signing process with User and SED Server's private key from CA
$E_{Key}()$	Symmetric encryption with Key, and Key is random value shared between User and SED Server
SED_{Finger}	SED into which fingerprint is embedded
$E_{pk}()$	Encryption with User's public key

4.4 핑거프린팅 기법이 적용된 애플리케이션 배포 절차

Fig. 3.은 핑거프린팅 기법이 적용된 애플리케이션의 배포 절차를 나타낸다. 핑거프린팅 기법이 적용된 애플리케이션의 배포 절차는 아래와 같다.

(1) 개발자는 SED를 SED 서버에 올린다(Fig. 3.의 1).
(2) SED 서버는 다수의 SED 파일 복사본을 생성한다.
(3) SED 서버는 SED 파일마다 각각 다른 핑거프린트 값을 삽입한다.
(4) 개발자는 마켓에 CLR을 올린다(Fig. 3.의 2).
(5) 사용자는 마켓에서 CLR을 내려받는다(Fig. 3.의 3).
(6) 사용자가 CLR을 실행하면 SED 서버는 핑거프린트가 삽입된 SED를 사용자에게 전송한다(Fig. 3.의 4).
(7) SED 서버는 SED에 삽입된 핑거프린트 값과 해당 SED를 다운로드한 사용자 정보를 보관한다.

SED 서버가 핑거프린트가 삽입된 SED를 사용자에게 전송하는 구체적인 프로토콜은 Fig. 4.와 같으며, 프로토콜에 사용된 기호의 의미는 Table 2.와

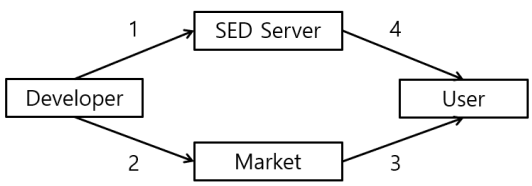

Fig. 3. Distribution process of fingerprinted application

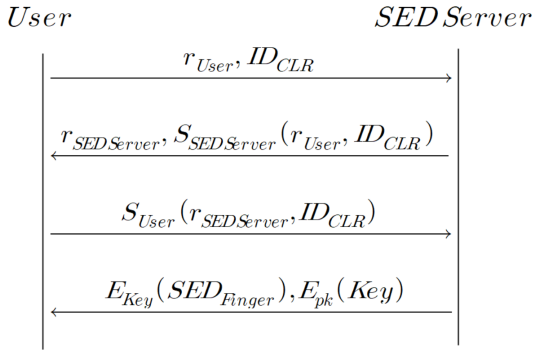

Fig. 4. Distribution protocol of fingerprinted application

같다.

(1) $User$는 $SEDServer$에 r_{User}와 ID_{CLR}을 전송한다.
(2) $SEDServer$는 $User$에 $r_{SEDServer}$와 $S_{SEDServer}(r_{User}, ID_{CLR})$을 전송한다.
(3) $User$는 $SEDServer$에 $S_{User}(r_{SEDServer}, ID_{CLR})$을 전송한다.
(4) $SEDServer$는 $User$에 $E_{Key}(SED_{Finger})$를 전송한다. 이때 Key를 사용자의 공개키로 암호화하여 함께 전송한다.

(1)에서 (3)까지는 사용자와 SED 서버가 서로 서명과 랜덤 값을 이용하여 양방향 인증을 수행하는 과정이다. (4)는 SED 서버가 사용자에게 핑거프린트 값이 삽입된 SED 파일을 전송하는 과정이다. 이때 SED 파일은 공격자가 가로챌 수 없도록 사용자에게 암호화하여 전송한다.

4.5 비공개 애플리케이션 배포 절차

비공개 애플리케이션을 배포하는 방법으로는 CLR과 SED를 각각 배포하는 방법과 CLR의 애셋(Asset)에 SED를 포함하여 배포하는 방법이 있다. 전자를 나타낸 것이 Fig. 5.이고 후자를 나타낸 것이 Fig. 6.이다. Fig. 5.와 Fig. 6.에 표기된 숫자는 CLR 파일 또는 SED 파일의 전송 순서를 나타낸다.

CLR과 SED를 각각 배포하는 절차는 아래와 같다.

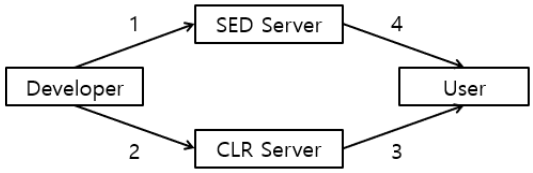

Fig. 5. Distribution process of secret application 1

Fig. 6. Distribution process of secret application 2

(1) 개발자가 SED를 SED 서버에 올린다(Fig. 5.의 1).
(2) 개발자는 CLR을 CLR 서버에 올린다(Fig. 5.의 2).
(3) 사용자는 CLR 서버에서 CLR을 내려받는다(Fig. 5.의 3).
(4) 사용자가 CLR을 실행하면 SED 서버는 랜덤하게 생성한 일회용 대칭키를 사용하여 SED를 암호화한 후, 대칭키를 사용자의 공개키로 암호화하여 함께 전송한다.

SED 서버가 SED를 사용자에게 전송하는 프로토콜은 핑거프린팅 기법이 적용된 애플리케이션을 전송하는 프로토콜과 유사하므로 설명을 생략한다.

CLR의 애셋에 SED를 포함하여 배포하는 절차는 아래와 같다.

(1) 우선 개발자는 CLR을 CLR 서버에 올린다(Fig. 6.의 1).
(2) 그 후 CLR 서버는 랜덤하게 생성한 일회용 대칭키를 사용하여 CLR을 암호화한 후, 대칭키를 사용자의 공개키로 암호화하여 함께 전송한다.

암호화된 CLR은 사용자가 복호화한 후에 설치할 수 있다. 또한, 사용자가 CLR을 최초로 실행하는 경우 CLR의 애셋에 포함된 SED는 내부 저장소로 복사된다.

CLR과 SED를 각각 배포하는 경우에는 CLR의 파일 크기가 작지만 배포 절차가 복잡해지며, CLR의 애셋에 SED를 포함하여 배포하는 경우에는 배포 절차는 간단해지지만 CLR의 파일 크기가 커진다.

4.6 실행 절차

Fig. 7.과 Fig. 8.은 CLR이 InMemoryDexClassLoader 함수를 사용하여 SED를 로드하는 동안의 실행 절차를 나타낸다. Fig. 7.은 애플리케이션의 최초 실행 절차를 나타내며, Fig. 8.은 애플리케이션의 최초 실행 이후의 실행 절차를 나타낸다.

F-Encrypted SED는 '지문 인증으로 보호되는 암호 키'에 의하여 암호화된 SED를 의미한다. Encrypted SED는 F-Encrypted SED 파일을 스마트폰 내부에 저장한 후에 삭제된다.

CLR이 정상적으로 실행되기 위해서는 스마트폰에 사용자의 지문이 등록되어 있어야 한다. 애플리케이션 실행 시 스마트폰에 지문이 등록되어 있지 않은 경우, CLR은 사용자에게 지문 등록 후에 애플리케이션을 재실행하도록 안내한다.

애플리케이션의 최초 실행 절차는 아래와 같다.

(1) 메모리는 암호화된 SED를 저장소에서 읽어 들인다(Fig. 7.의 1).
(2) 애플리케이션 배포 시 서버가 전송한 암호 키를 이용하여 암호화된 SED를 복호화한다(Fig. 7.의 2).
(3) 지문 인증을 수행한다.
(4) 사용자가 지문 인증에 성공한 경우에만, '지문 인증으로 보호되는 암호 키'를 사용하여 SED를 암호화한다(Fig. 7.의 3).
(5) 지문 인증을 통해 암호화된 SED는 스마트폰 내부에 저장한다(Fig. 7.의 4).
(6) 암호화된 SED를 삭제한다.
(7) 마지막으로 CLR은 메모리상의 복호화된 SED를 스마트폰 저장소를 거치지 않고 바로 로드한다(Fig. 7.의 5).

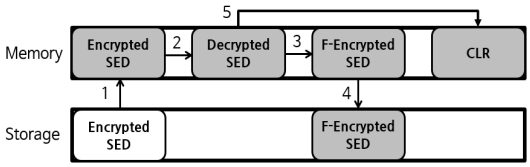

Fig. 7. First execution process

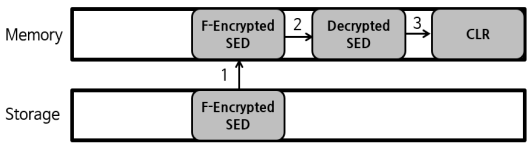

Fig. 8. Execution process after first execution

애플리케이션의 최초 실행 이후의 실행 절차는 아래와 같다.

(1) 메모리는 '지문 인증으로 보호되는 암호 키'로 암호화된 SED를 저장소에서 읽어 들인다(Fig. 8.의 1).
(2) 지문 인증을 수행한다.
(3) 사용자가 지문 인증에 성공한 경우에만 암호화된 SED를 복호화한다(Fig. 8.의 2).
(4) 마지막으로 CLR은 메모리상의 복호화된 SED를 스마트폰 저장소를 거치지 않고 바로 로드한다(Fig. 8.의 3).

InMemoryDexClassLoader 함수를 사용하여 SED를 로드하는 경우, Fig. 7.과 Fig. 8.에서 저장소에 복호화된 SED가 존재하는 순간이 없음을 확인할 수 있다. 따라서 제안 기법은 '루트 공격자'와 'no R 공격자' 모두로부터 안전하다.

V. 안전성 분석

본 논문에서 제안하는 애플리케이션 보호 기법과 기존에 제안된 유사 기법들에 대하여 분석한 결과는 Table 3.과 같다. Table 3.에서 O 표기는 해당 논문에서 제안한 기법을 사용한 경우에 '루트 공격자' 또는 'no R 공격자'가 보호 대상 코드를 탈취할 수 없어 안전함을 의미하며, X 표기는 공격자가 보호 대상 코드를 탈취할 수 있어 안전하지 않음을 의미한다.

본 논문에서 제안하는 기법은 지문 인증에 성공한 경우에만 메모리상에서 SED를 복호화한 후 동적으로 로드한다. 지문 인증은 정당한 사용자만이 성공할 수 있으며, 복호화된 SED는 스마트폰 내부에 일시

Table 3. Security comparison of the application protection schemes

Scheme	root	no R	kernel modification
This paper	O	O	X
[6]	X	X	O
[7]	X	O	X
[8]	X	X	O

Table 4. The comparison of the execution time with different SEDs (ms)

	First execution process		Execution process after first execution	Dynamic loading of SED
	SED decryption	Fingerprint authentication and SED encryption	Fingerprint authentication and SED decryption	
Talalarmo	2.4	104.7	110.9	7.9
Barcode Scanner	6.6	312.5	313.9	15.6
2048	7.5	514.4	522.9	32.5

적으로도 저장되지 않는다(복호화된 SED는 메모리 상에서만 존재한다). 따라서 본 논문에서 제안하는 기법은 '루트 공격자', 'no R 공격자'로부터 안전하며, 제안 기법을 사용하기 위하여 커널 코드를 변경할 필요가 없다.

[6]의 경우에는 클래스를 안전한 공간에 보관한다. 지정된 프로세스만이 안전한 공간에 보관된 클래스를 동적으로 로드할 수 있으며, 지정된 프로세스가 아닌 경우에는 루트 권한으로도 해당 공간에 접근할 수 없다. 그러나 공격자는 애플리케이션을 실행한 후에 메모리에 올라온 코드를 복사하여 가져갈 수 있으므로 '루트 공격자', 'no R 공격자'로부터 안전하지 않다. 또한, 커널 코드의 변경이 필요하여 사용자 편의성이 부족하다는 단점이 있다.

[7]의 경우에는 정당한 사용자가 인증을 통과한 경우에만 클래스를 서버로부터 동적으로 로드할 수 있다. 공격자가 인증을 우회하지 못하는 경우에는 클래스에 접근할 수 없으므로 사용자 인증 정보를 가로챌 수 없는 'no R 공격자'로부터 안전하다. 그러나 애플리케이션이 클래스 파일을 로드하기 직전에는 복호화된 클래스 파일을 스마트폰 내부에 일시적으로 저장하므로 삭제된 파일을 획득할 수 있는 '루트 공격자'로부터는 안전하지 않다.

마지막으로 [8]의 경우 공격자가 쉽게 알아낼 수 있는 SIM 카드 시리얼 번호와 솔트 값을 이용하여 애플리케이션 코드를 암호화한다. SIM 카드 시리얼 번호와 솔트 값은 루트 권한만 있으면 쉽게 획득할 수 있으므로 '루트 공격자', 'no R 공격자'로부터 안전하지 않다.

따라서 본 논문에서 제안하는 기법은 커널 변경 없이도 '루트 공격자', 'no R 공격자'로부터 안전하다는 점에서 기존에 제안된 기법에 비하여 의미를 가진다.

VI. 성능 평가

본 논문에서는 제안하는 기법의 성능을 평가하기 위하여 F-Droid(안드로이드 애플리케이션 오픈소스 저장소)[24]에서 제공하는 애플리케이션의 DEX 파일을 CLR이 로드하는 경우에 소요되는 시간을 구글 넥서스 5X 스마트폰에서 측정하였다. 스마트폰의 안드로이드 버전은 8.0.0이고, 프로세서는 Snapdragon 808 Processor, 1.8 GHz hexa-core 64-bit이다.

애플리케이션은 DEX 파일 크기를 기준으로 Talalarmo[25], Barcode Scanner[26], 2048[27]을 선택하였다. Talalarmo는 알람 시계 기능이 있는 애플리케이션이며, DEX 파일의 크기는 194KB이다. Barcode Scanner는 바코드 스캔 기능이 있는 애플리케이션이며, DEX 파일은 835KB의 크기를 갖는다. 마지막으로 2048은 퍼즐 게임이며, DEX 파일의 크기는 1,508KB이다.

Table 4.는 CLR이 SED를 동적으로 로드하기까지 소요된 시간을 30회씩 측정한 후 평균 시간을 나타낸 표이다. 측정 결과 SED의 크기가 클수록 CLR의 실행에 소요되는 시간이 증가함을 확인하였다. 3개의 SED 중 크기가 가장 큰 2048의 경우 CLR이 SED를 동적으로 로드하는 데 소요된 시간은 약 0.03초였으며, 지문 인증을 통하여 SED를 암호화하거나 복호화하는 과정은 약 0.5초 가량 소요되었다.

한편, CLR은 동적으로 로드된 SED의 함수를 사용하기 위하여 자바 리플렉션 기법을 이용한다. 자바 리플렉션 기법을 통하여 함수를 사용하는 과정은 다음과 같다.

(1) 동적으로 로드된 SED의 객체를 생성한다.
(2) getDeclaredMethod 함수를 호출한다. 이때, 해당 함수의 인자로는 사용하고자 하는 함수를

넣어준다.

(3) invoke 함수를 호출한다. 이때, invoke 함수의 인자로는 (1)에서 생성한 객체를 넣어준다.

리플렉션으로 인해 발생하는 오버헤드를 측정하기 위하여, 두 개의 정수를 더하는 doSum 함수를 자체적으로 제작한 후, 이를 호출하는 데 소요되는 시간을 50회 측정하였다. 측정 결과 (1)부터 (3)의 과정으로 소요되는 오버헤드는 평균 46ms로써, 매우 작은 오버헤드만 발생함을 확인할 수 있었다.

VII. 결 론

본 논문에서 제안하는 기법을 사용하면 '루트 공격자'로부터 비공개 애플리케이션과 핑거프린팅 기법이 적용된 애플리케이션을 안전하게 보호할 수 있다. 따라서 비공개 애플리케이션 또는 핑거프린팅 기법이 적용된 애플리케이션을 보유한 사용자는 스마트폰 분실, 도난 등으로 인한 보안 사고를 예방할 수 있다.

그러나 원격으로 스마트폰을 해킹하여 루트 권한을 획득한 공격자로부터 해킹으로 인한 보안 사고를 예방하는 데에는 한계가 존재한다. 이러한 공격자로부터의 공격을 방어하기 위해서는 루팅 탐지 기법, 안티 메모리 덤핑 기법 등이 필요하다. 위와 같은 기법을 함께 적용함으로써 본 논문의 기법을 보완하는 것이 향후 연구 계획이다.

References

[1] L Luo, Y. Fu, D. Wu, S. Zhu, and P. Liu, "Repackage-proofing android apps," Dependable Systems and Networks, pp. 550-561, Oct. 2016.

[2] S.A. Bailey, D. Felton, V. Galindo, F. Hauswirth, J. Hirvimies, M. Fokle, F. Morenius, and C. Colas, "The trusted execution environment: delivering enhanced security at a lower cost to the mobile market," GlobalPlatform White Paper, Feb. 2011.

[3] The Android Open Source Project, Authentication Overview, https://source.android.com/security/authentication/

[4] T. Cooijmans, J. Ruiter, and E. Poll, "Analysis of secure key storage solutions on android," Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices, pp. 11-20, Nov. 2014.

[5] The Android Open Source Project, Fingerprint HAL, https://source.android.com/security/authentication/fingerprint-hal

[6] Youn-sik Jeong, Yeong-ung Park, Jae-chan Moon, Seong-je Cho, Dong-jin Kim, and Min-kyu Park, "An anti-piracy mechanism based on class separation and dynamic loading for android applications," Proceedings of the 2012 ACM Research in Applied Computation Symposium, pp. 328-332, Oct. 2012

[7] K.Y. Tsai, Y.H. Chiu, and T.C. Wu, "Android app copy protection mechanism based on dynamic loading," The 18th IEEE International Symposium on Consumer Electronics, pp. 1-3, June 2014.

[8] Jung-hyun Kim and Kang-seung Lee, "Robust anti reverse engineering technique for protecting android applications using the aes algorithm," Journal of Korean Institute of Information Scientists and Engineers, 42(9), pp. 1100-1108, Sep. 2015

[9] C.S. Collberg, C. Thomborson, and G.M. Townsend, "Dynamic graph-based software fingerprinting," vol. 29, no. 6, article. 35, Oct. 2007.

[10] C. Collberg and C Thomborson, "Software watermarking: models and dynamic embeddings," Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pp. 311-324, Jan. 1999.

[11] W. Zhou, X. Zhang, and X. Jiang, "AppInk: watermarking android apps

for repackaging deterrence," Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security, pp. 1-12, May 2013
〔12〕 C. Ren, K. Chen, and P Liu, "Droidmarking: resilient software watermarking for impeding android application repackaging," Proceedings of the 29th ACM/IEEE international conference on Automated software engineering, pp. 635-646, Sep. 2014
〔13〕 K. Fukushima and Kouichi Sakerai, "A Software Fingerprinting Scheme for Java Using Classfiles Obfuscation," WISA 2003, LNCS 2908, pp. 303-316, 2004.
〔14〕 R. Schwamm and N.C. Rowe, "Effects of the factory reset on mobile devices," Journal of Digital Forensics, Security and Law, vol. 9, no. 2, pp.205-220, 2014
〔15〕 Soo-woong Eo, Woo-yeon Jo, Seok-jun Lee, and Tae-shik Shon, "Ensuring the Admissibility of Mobile Forensic Evidence in Digital Investigation," Journal of the Korea Institute of Information Security and Cryptology, 26(1), pp. 135-152, Feb. 2016
〔16〕 Android Developer, DexFile, https://developer.android.com/reference/dalvik/system/DexFile.html
〔17〕 Android Developer, DexClassloader, https://developer.android.com/reference/dalvik/system/DexClassLoader.html
〔18〕 Google Git, DexClassLoader.java, https://android.googlesource.com/platform/libcore/+/master/dalvik/src/main/java/dalvik/system/DexClassLoader.java
〔19〕 Google Git, DexPathList.java, https://android.googlesource.com/platform/libcore/+/master/dalvik/src/main/java/dalvik/system/DexPathList.java
〔20〕 P. Schulz, "Code protection in android," Institute of Computer Science, University of Bonn, 110, June 2012
〔21〕 Android Developer, PathClassloader, https://developer.android.com/reference/dalvik/system/PathClassLoader.html
〔22〕 Wikipedia, Android Oreo, https://en.wikipedia.org/wiki/Android_Oreo
〔23〕 Android Developer, InMemoryDexClassloader, https://developer.android.com/reference/dalvik/system/InMemoryDexClassLoader.html
〔24〕 F-Droid, What is F-Droid, https://f-droid.org/
〔25〕 F-Dorid, Talalarmo, https://f-droid.org/packages/trikita.talalarmo/
〔26〕 F-Dorid, Barcode Scanner, https://f-droid.org/packages/com.google.zxing.client.android/
〔27〕 F-Dorid, 2048, https://f-droid.org/packages/com.uberspot.a2048/

〈저자소개〉

류 환 일 (Hwahn-il Lyoo) 학생회원
2016년 2월: 경찰대학 법학과 졸업
2016년 3월~현재: 고려대학교 정보보호대학원 석사과정
〈관심분야〉 정보보호, 시스템 보안, 모바일 보안, 디지털 포렌식

석 재 혁 (Jae-Hyuk Suk) 학생회원
2012년 2월: 서울시립대학교 전자전기컴퓨터공학부 학사
2014년 2월: 고려대학교 정보보호대학원 석사
2014년 3월~현재: 고려대학교 정보보호대학원 박사과정
〈관심분야〉 소프트웨어 보안, 소프트웨어 역공학, 코드 난독화

박 진 형 (Jin-Hyung Park) 학생회원
2010년 2월: 건국대학교 컴퓨터공학과 졸업
2012년 2월: 고려대학교 정보보호학과 석사
2012년 3월~현재: 고려대학교 정보보호대학원 박사과정
〈관심분야〉 암호프로토콜, 모바일 보안, 암호 알고리즘, 스토리지 보안

이 동 훈 (Dong-Hoon Lee) 종신회원
1983년 8월: 고려대학교 경제학사 졸업
1987년 12월: Oklahoma University 컴퓨터공학과 석사
1992년 5월: Oklahoma University 컴퓨터공학과 박사
1993년 3월~1997년 2월: 고려대학교 전산학과 조교수
1997년 3월~2001년 2월: 고려대학교 전산학과 부교수
2001년 3월~현재: 고려대학교 정보보호대학원 교수
〈관심분야〉 암호 프로토콜, 암호이론, 함수 암호, 소프트웨어 보안, 모바일 보안, 자동차 보안, USN이론