논문 19-44-06-09 The Journal of Korean Institute of Communications and Information Sciences '19-06 Vol.44 No.06
https://doi.org/10.7840/kics.2019.44.6.1084

안드로이드 앱 보호 기능 탐지 시스템 구현 및 앱 보호 실태 분석

조 동 민◆, 최 형 기°

Implementing Android App Hardening Detector and Analyzing Status of App Protection

Dongmin Jo◆, Hyoung-Kee Choi°

요 약

안드로이드 앱 시장이 폭발적으로 성장하면서 안드로이드 앱에 대한 보안 위협이 증가하고 있다. 보안 위협을 최소화하기 위해 학계에서 앱 보안성을 분석하는 연구를 제안하고 있으나, 분석 앱의 개수와 범위가 한정적이며 평가 방법을 자동화하는 연구는 부재하다. 본 논문에서는 안드로이드 앱의 보안성을 평가하는 시스템을 제안하고 국내 앱 생태계가 취약하다는 점을 밝힌다. 제안하는 시스템은 앱에 적용하는 보호 기능 중 다섯 가지 기능을 탐지하여 앱의 보안성을 평가한다. 제안하는 시스템을 이용하여 상위 매출 1322개의 애플리케이션을 분석한 결과, 61.58%의 앱이 어떠한 보호 기능도 탑재하지 않아 보안 위협을 노출하고, 88.51%의 앱이 위/변조 후 배포 행위에 취약하며, 97.21%의 앱이 디버깅을 통한 메모리 조작에 취약하다는 사실을 발견하였다.

Key Words : Android app, app hardening, automatic detection, dynamic analysis, static analysis

ABSTRACT

With the Android app market exploding, security threats to Android apps are on the rise. To minimize security threats, academic researches have been proposed to analyze app security. However, the number and the scope of apps have been limited on the research. Furthermore, there is a lack of research about systems that detect security features applied to apps. This paper proposes a system to assess the security of Android apps and reveals that the app ecosystem is vulnerable. The proposed system detects five of the app hardenings applied to apps. Using our system, we analyzed the top 1,322 applications in sales. From the result, we found that 61.58% of the apps do not have any protection features, which exposes security threats, 88.51% are vulnerable to app modification, and 97.21% are vulnerable to memory manipulation through debugging.

Ⅰ. 서 론

안드로이드 애플리케이션(이하 앱) 시장이 급증함과 동시에 앱에 대한 공격 위협이 증가하고 있다. 공격 위협에는 앱을 위/변조하여 악성 코드를 배포하는 행위, 실행하는 앱의 메모리를 조작하여 무결성을 해치는 행위, 정적 및 동적 분석을 통해 취약점을 악용하는 행위 등이 존재한다[1]. 공격자는 앱 내부의 소스

※ 이 논문은 2019년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임. (No.2014-6-00909, (안전성 연구 2세부) 운영체제 안전성 연구)
◆ First Author : Sungkyunkwan University Department of Software, jo4965@skku.edu, 학생회원
° Corresponding Author : Sungkyunkwan University Department of Software, meosery@skku.edu, 정회원
논문번호 : 201903-031-B-RU, Received March 26, 2019; Revised April 15, 2019; Accepted April 23, 2019

코드를 Apktool[2]과 같은 오픈 소스 분석 도구로 쉽게 확인이 가능하여 분석 및 위/변조가 가능하다. 또한 루트 권한을 획득하여 시스템을 변조하는 공격자들은 메모리 조작을 통해 앱의 실행 무결성을 침해하는 것이 가능하다. 앱에 대한 낮은 분석 난이도는 안드로이드 앱에 대한 보안 위협을 높이고 있다.

증가하는 보안 위협에 대비하기 위해 앱을 보호하는 다양한 연구가 제안되고 있다[3-5]. 보호 기능의 공통적인 목적은 악의적 공격자에 의한 피해를 방지하는 것이며, 보호 기능의 종류에는 안티 에뮬레이터, 안티 디버깅, 코드 가상화, 루팅 환경 검사 등 개발자의 구현에 따라 무수히 많은 기능들이 존재한다[6,7].

안전한 앱 설계를 위해 앱에 적용된 보호 기능을 분석하는 연구가 제안되고 있으나[8-10] 국내의 앱에 대한 안전성 연구 결과는 부재하다. 기존의 연구는 분석하는 앱 분야와 개수가 한정적이며, 자동화 방법이 아닌 직접 분석을 이용한다. 또한 보호 기능을 위탁하여 적용하는 보안 솔루션들을 분석하는 연구가 존재하나[6,11] 개발사가 자체적으로 보호하는 앱들에 대한 조사 결과는 부족하다. 결론적으로 앱 전반에 대한 보안성 평가를 위해서는 앱 종류에 관계없이 보호 기능을 자동으로 탐지하는 도구가 필요하다.

본 논문에서는 보호 기능을 탐지하는 시스템을 개발하여 안드로이드 앱의 보호 기능 적용 실태를 조사한다. 시스템을 설계하기 위해 자동화 탐지가 가능한 앱 보호 기능을 선별하여 정리하였다. 다음으로, 정리한 자동화 탐지 방법론을 구현하여 정적 분석 및 동적 분석을 통해 보호 기능을 탐지하는 시스템을 개발하였다. 마지막으로 개발한 시스템을 통해 국내 Google Play Store에서 표기하는 최고 매출 앱들을 분석하였다. 전체 분야 상위 600개의 앱과 가장 사용률이 높은 게임, 금융, 쇼핑의 세 개 분야에서 각각 상위 300개의 앱을 분석하여 취약한 앱 보호 현황을 제시한다.

본 논문의 구성은 다음과 같다. 2절에서는 관련 연구를 소개하고 3절에서는 탐지하는 보호 기능의 목적과 구현 방식을 소개한다. 4절에서는 제안하는 시스템의 요구사항을 정의하고 구현한 시스템이 보호 기능을 탐지하는 방법을 설명한다. 다음으로 5절에서는 수집한 앱을 분석한 결과를 보이고 6절에서는 제안하는 시스템의 한계를 설명한다. 마지막으로 7절에서는 결론을 맺는다.

Ⅱ. 관련 연구

관련 연구로 앱에 적용된 보호 기능을 분석하는 연구와 탐지하는 연구가 제안되었다.

2.1 앱 보호 기능 분석에 관한 연구

이우진[8] 등은 10종의 뱅킹 앱이 모바일 백신 프로그램을 실행하는 지점을 탐색하여 보호 기능을 우회하는 연구를 제안하였다. 유재욱[9] 등은 간편 결제 앱 및 보호 솔루션에 대한 보호 기능을 분석하여 우회하는 연구를 제안하였다. 두 연구 모두 정적 분석 및 동적 분석을 직접 수행하여 보호 기능을 탐색하고 우회 여부를 분석하였다. 본 논문에서 제안하는 시스템은 앱의 종류를 보안 솔루션이나 금융 앱에 한정하지 않고 모든 종류의 앱에 대해 보호 기능 탐지가 가능하다.

2.2 앱 보호 기능 탐지 자동화에 관한 연구

Z. Qu[10] 등은 앱의 실행 코드를 외부에서 호출하는 기법인 동적 클래스 로딩을 적용하는 앱을 조사하였다. 동적 클래스 로딩을 적용하는 앱을 탐지하는 시스템을 개발하였으며, 조사한 앱 중 일부는 앱을 보호하기 위해 동적 클래스 로딩을 사용함을 밝혔다. 제안된 시스템은 앱이 분석 로그를 남기도록 앱 내부 코드를 수정하므로 코드 무결성 검사가 적용된 앱의 경우 조사가 불가능하다. 본 논문에서 제안하는 시스템은 앱을 수정하지 않아 앱에 적용된 코드 무결성 검사에 의해 영향을 받지 않는다.

Y. Duan[11] 등은 앱에 적용된 실행 코드 암호화 기법을 무력화하여 코드의 원본을 추출하는 도구를 제안하였다. 제안된 도구는 안드로이드 에뮬레이터를 이용하여 6종의 보안 솔루션을 무력화하였다. 본 논문에서 제안하는 시스템은 실제 기기에 구현된 안드로이드의 오픈 소스를 수정하여 보호 기능 중 에뮬레이터 검사에 저항성을 갖도록 설계하였다.

Ⅲ. 자동화 탐지 대상 보호 기능

앱이 탑재하는 많은 보호 기능 중 공통적인 구현 방식을 가지는 일부 기능을 선별하여 정리하였다. 본 연구에서는 앱의 보호 기능 중 다섯 가지를 구분하여 탐지한다.

3.1 루트 인터페이스 검사

안드로이드 권한 정책에서는 모든 앱들이 샌드박스로 보호되어 앱 외부 영역을 접근하는 과정에 제한이 존재한다. 루팅을 활성화한 기기는 루트 인터페이스를 설치하고, 일반 앱은 루트 인터페이스를 이용해 루트

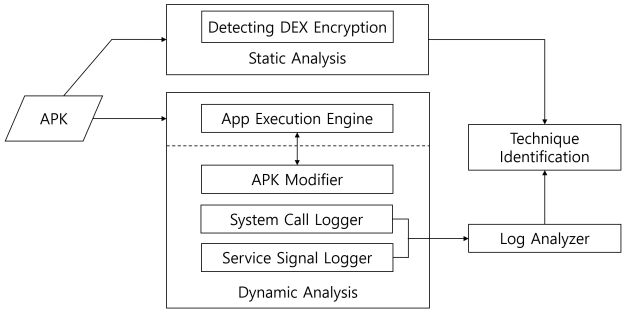

그림 1. 제안하는 시스템 구조. APK 파일을 입력하면 정적 분석을 통해 DEX 암호화를 탐지하고, 동적 분석을 통해 나머지 보호 기능을 탐지한다.
Fig. 1. Proposed System Architecture. APK files are analyzed with static analysis for detecting DEX encryption and with dynamic analysis for detecting other app hardenings.

권한을 획득한다. 루트 권한을 획득한 앱은 안드로이드의 권한 정책을 우회하여 다른 앱의 영역을 자유롭게 접근하는 능력을 갖는다. 기기 사용자는 루트 권한을 가진 앱을 이용해 다른 앱의 파일 및 메모리를 조작한다. 사용자가 루팅이 활성화된 기기에서 앱을 조작하는 것을 막기 위해 앱은 루트 인터페이스 검사를 수행한다. 루트 인터페이스는 "su" 파일을 의미하며 일반 앱이 접근할 수 있도록 PATH 환경 변수에 적재한다. 루팅 환경을 검사하는 앱은 PATH 환경 변수에 존재하는 "su" 파일의 존재 유무를 검사한다. 검사하는 방법에는 open, stat, access 등 시스템 콜을 통해 존재 유무, 권한 등을 확인하거나 일반 앱과 유사하게 execve 시스템 콜을 통해 "su" 파일을 실행하여 앱 자신의 권한 상승 여부를 확인하는 방법들을 사용한다.

3.2 안티 디버깅

디버깅 기능을 이용하는 사용자는 앱의 흐름이나 메모리 값을 조작하는 능력을 갖는다. 디버깅 기능에는 안드로이드 환경에서 사용가능한 자바 디버깅과 리눅스 환경에서 공통적으로 사용 가능한 네이티브 디버깅이 존재한다. 자바 디버깅은 안드로이드 런타임에서 제공하는 기능으로, 자바 언어 기반으로 디버깅이 가능하며 JDWP(Java Debug Wire Protocol)을 이용한다. 자바 디버깅을 막기 위한 방법으로는 앱 자신을 구동하는 안드로이드 런타임에서 JdwpState 구조체 안의 값들을 수정하는 방법 등으로 안티 디버깅을 구현한다. 네이티브 디버깅은 기계어 기반으로 디버깅하는 것을 의미하며 리눅스 시스템 콜 ptrace를 이용한다. 네이티브 디버깅 방지 기법에는 앱 실행시 ptrace 시스템 콜을 자기 자신에게 부착(attach)하여 타 프로세스의 ptrace 접근을 막는 방법, 주기적으로 /proc/self/stat을 확인하여 자신 프로세스의 디버깅 상태를 확인하는 방법 등이 존재한다.

3.3 안티 리패키징

리패키징은 앱의 전신인 APK(Android Application Package) 파일의 내용을 조작하여 재배포하는 것을 의미한다. 악의적 공격자에 의해 앱이 조작되어 배포될 경우 개발사와 일반 사용자가 모두 피해를 입을 가능성이 존재한다. 안티 리패키징을 구현한 앱은 실행하고 있는 앱의 상태를 확인하여 개발자가 배포한 원본 상태와의 일치 여부를 검사한다. APK 파일에는 개발자가 공개키 방식을 이용한 서명이 포함되어 있다. 공격자는 개발자의 APK 서명을 위조하지 못하므로 안티 리패키징은 APK 서명을 검사하는 기능을 포함한다.

3.4 동적 코드 수정

동적 코드 수정은 공격자의 정적 분석을 어렵게 하기 위해 앱 실행 도중 메모리 상에서 코드를 수정하는 여러 기법들을 통칭한다. 메모리 상에서 코드를 수정하는 기법에는 암호화된 코드를 복호화하는 경우, 사용하는 라이브러리들을 후킹(hooking)하여 흐름을 복잡화하는 경우 등이 속한다. 동적 코드 수정 기법을 적용한 앱은 실행시 로딩된 라이브러리의 코드 영역을 수정하여 정적 분석 결과와는 상이한 동작 방식을 보인다.

3.5 DEX(Dalvik Executable) 암호화

DEX 파일은 앱의 주요 로직이 담긴 파일로, 자바 기반의 코드가 중간 언어인 smali로 변환된 형태이다. DEX 파일은 기계어가 아닌 중간 언어 형태를 가져 원본 자바 언어로 복원 및 분석이 용이하다. DEX 파일 분석을 통한 역공학을 방지하고자 주요 로직이 담

긴 DEX 파일은 암호화하여 앱 내부의 다른 공간에 두고, 기존의 파일을 dummy DEX 파일로 대체한다. 앱 실행시 dummy DEX 파일이 앱 내부의 암호화된 DEX 파일을 복호화하여 실행하는 기능을 수행한다[7]. 이 과정에서 DEX 파일을 메모리에 적재하는 DexClassLoader, PathClassLoader, InMemoryDexClassLoader 등 DEX 동적 로딩 API를 사용하거나 메모리에 적재된 안드로이드 런타임을 수정한다. 소단원에 관한 내용을 간단히 살펴본다. 여기서는 소단원에 관한 내용을 간단히 살펴본다.

Ⅳ. 보호 기능 탐지 시스템

4.1 시스템 요구 사항

4.1.1 앱 보호 기능에 대한 저항성

탐지 시스템의 오탐을 최소화하기 위해서는 시스템이 앱의 보호 기능에 의해 무력화되지 않아야 한다. 에뮬레이터를 기반으로 한 탐지 시스템은 에뮬레이터 검사 기능을 수행하는 앱 분석이 불가능하다. 또한, 탐지 시스템이 보호 기능 검사를 위해 앱을 수정하여 실행할 경우 안티 리패키징을 적용한 앱 분석이 불가능하다. 제안하는 시스템은 실제 Google 사의 스마트폰에 구현된 오픈 소스 운영 체제를 기반으로 하고, 실기기인 Nexus 5X에 구현하여 에뮬레이터 검사에 저항성을 갖는다. 또한 안티 리패키징 외의 보호 기능을 검사할 경우 앱을 수정하지 않아 안티 리패키징에 저항성을 갖는다.

4.1.2 검사 대상 앱의 수용성

국내 전반 앱에 대한 보안성 평가를 위해서는 탐지 시스템이 모든 종류의 앱을 검사할 수 있어야 한다. 후킹(hooking)을 이용해 앱 내 DEX 코드에서 호출하는 자바 API 만을 감지할 경우 게임, 금융 앱 등 네이티브 코드에서 보호 기능을 호출하는 앱은 분석하기 어렵다. 제안하는 시스템은 안드로이드 시스템과 커널을 수정하므로 호출 주체에 관계없이 모든 기능을 감시한다.

4.2 시스템 구조

보호 기능 탐지 시스템은 Google의 오픈 소스 운영 체제인 AOSP(Android Open Source Project)[12]에서 Android 7 버전의 안드로이드 시스템과 커널을 일부 수정하여 구현하였다. 안티 에뮬레이터 기능을 우회하기 위해 구현한 시스템을 Nexus 5X 기기에 설치하고 AOSP 관련 환경 변수 및 프로퍼티들을 제거하였다.

전반적인 시스템의 구조는 그림 1과 같이 앱을 실행하지 않는 정적 분석 모듈과 앱을 실행하는 동적 분석 모듈로 구성된다. 정적 분석 모듈은 입력된 APK 파일의 내부를 분석하여 DEX 암호화 기능의 존재 유무를 검사한다. 동적 분석 모듈은 입력된 APK 파일을 설치하고, 설치된 앱을 실행하여 앱이 호출하는 시스템 콜을 로그로 저장한다. 앱은 입력한 시간 동안 실행되며 탐지 기능을 구분하기 위해 총 두 번 실행된다. 처음 실행에서는 루팅 환경 검사, 안티 디버깅, 동적 코드 수정과 관련된 시스템 콜과 함수를 기록한다. 두 번째 실행에서는 안티 리패키징을 검사한다. 안티 리패키징을 검사하기 위해 APK 파일을 변조한 후 실행하여 액티비티 매니저 서비스의 종료 신호를 기록한다. 정적 분석 모듈과 동적 분석 모듈이 동작을 완료하면 로그 분석기가 저장된 로그를 분석하여 결과를 산출한다.

4.3 시스템 동작 과정

4.3.1 DEX 암호화 탐지

입력된 APK 파일에서 메인 로직이 담긴 classes.dex 파일들과 앱이 사용하는 컴포넌트들이 선언된 AndroidManifest.xml 파일을 비교 분석하여 DEX 암호화를 탐지한다. AndroidManifest.xml 파일은 앱이 사용하는 액티비티, 서비스와 같은 컴포넌트들을 선언한다. 선언한 컴포넌트에는 각 컴포넌트를 실제로 구현하는 자바 클래스 이름을 명시한다. classes.dex 파일에는 앱이 사용하는 모든 클래스를 포함하고, AndroidManifest.xml 파일은 컴포넌트와 관련된 일부 클래스 이름만을 포함한다. 즉, AndroidManifest.xml 파일에 선언한 모든 클래스 이름이 classes.dex 내부에 존재해야 한다. 정적 분석 모듈은 AndroidManifest.xml 파일에 선언한 컴포넌트 클래스 이름을 추출하여 classes.dex 클래스 목록에서의 존재 여부를 검사한다. 결론적으로 AndroidManifest.xml 파일에 선언하였지만 classes.dex 파일에 존재하지 않는 클래스는 암호화되어 존재하는 것으로 판단한다.

거짓 양성(false positive)을 줄이기 위해 OAuth 관련 클래스는 검사 대상에서 제외한다. OAuth 서비스를 구현한 앱은 OAuth 관련 클래스가 classes.dex 파일에 구현되어 있지 않고, OAuth API 제공자로부터 공급받은 별개의 jar 혹은 dex 파일에 구현되어있다. 즉, OAuth 서비스를 구현한 앱은

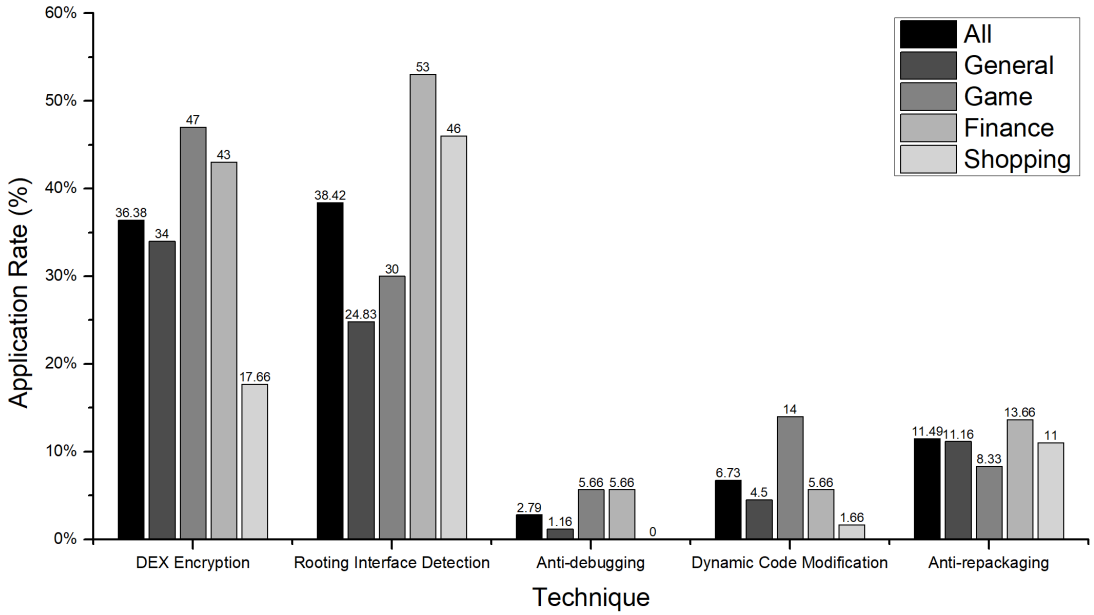

그림 2. 앱 보호 기능의 적용률
Fig. 2. The application rate of Android app hardenings

AndroidManifest.xml 파일에 OAuth 관련 클래스가 선언되어있지만 classes.dex가 아닌 파일로부터 OAuth 클래스를 호출하므로 DEX 암호화로 탐지될 수 있다. OAuth 관련 클래스는 클래스 이름에 API 제공 서비스 이름과 "oauth" 관련 문자열이 포함되어 있으므로, 정적 분석 모듈은 AndroidManifest.xml에서 추출한 클래스가 이러한 문자열을 포함한다면 해당 클래스를 검사 대상에서 제외한다.

4.3.2 루트 인터페이스 검사 탐지

루트 인터페이스 검사 탐지를 위해 앱이 "su" 파일을 탐색하는 함수 및 시스템 콜의 호출 여부를 검사한다. 앱이 파일 존재 여부를 판단하는 시스템 콜 open, stat, access, execve의 호출 여부를 감지하며, 추가적으로 시스템 콜이 아닌 readdir 함수를 감지한다. readdir 함수는 디렉토리 내부 파일 정보를 반환하는 함수로, 디렉토리에 대한 파일 지시자를 인자로 한다. readdir 함수를 사용하기 위해서는 디렉토리를 대상으로 한 시스템 콜 open이 먼저 호출되어야한다. readdir으로 구현한 루팅 인터페이스 검사 기능은 "su" 파일이 아닌 디렉토리를 대상으로 open 시스템 콜을 호출하므로 시스템 콜 검사만으로는 감지가 어렵다. 결론적으로 동적 분석 모듈은 open, stat, access, execve, readdir의 다섯 개 시스템 콜 및 함수가 "su" 파일을 대상으로 한다면 루트 인터페이스를 검사하는 것으로 판단한다.

4.3.3 안티 디버깅 탐지

안티 디버깅 탐지를 위해 ptrace 시스템 콜의 호출 여부를 검사한다. ptrace는 디버깅을 목적으로 하는 시스템 콜이며, 한 프로세스가 ptrace를 호출하면 다른 프로세스에 부착(attach)하여 메모리에 접근 가능하다. 메모리 접근을 완료하고 나면 분리(detach)하여 디버깅을 종료한다. 안드로이드 시스템에서는 부착하려고 하는 대상이 이미 다른 프로세스에게 부착되어 있다면 부착이 불가능하다. 이러한 제약을 통해 앱은 자기 자신을 ptrace로 부착하여 안티 디버깅을 구현한다. 공격자가 디버거 프로세스로 디버깅을 시도하여도 대상 앱이 이미 부착 상태이므로 ptrace 시스템콜이 부착할 수 없어 메모리에 접근할 수 없다.

JDWP를 이용한 자바 디버깅은 앱이 사용하는 네이티브 코드에 대한 디버깅이 불가능한 반면 ptrace를 이용한 디버깅은 네이티브 코드와 자바 기반의 디버깅이 모두 가능하다. 완전한 안티 디버깅을 구현하기 위해서는 ptrace 기반 안티 디버깅을 적용해야 한다. 동적 분석 모듈은 네이티브 기반 안티 디버깅에서 호출하는 ptrace 시스템 콜 호출 여부를 기준으로 안티 디버깅 적용 여부를 판단한다.

4.3.4 동적 코드 수정 탐지

동적 코드 수정 탐지의 기본 원리는 앱 실행시 mprotect 시스템 콜의 호출 여부 검사이다. mprotect는 메모리 상 접근 권한을 변경하는 시스템 콜이다. 안드로이드 시스템에서는 라이브러리를 메모리에 적재시 코드 영역에 쓰기 권한을 부여하지 않는다. 동적 코드 수정 기법을 적용한 앱이 다른 라이브러리의 코드 영역을 수정하거나 암호화된 코드 영역을 복호화하기 위해서는 mprotect를 이용해 쓰기 권한을 부여해야 한다. mprotect의 호출을 감지하면 앱이 수행하는 동적 코드 수정을 탐지하는 것이 가능하다.

동적 분석 모듈은 앱 메인 로직에서 호출하는 mprotect 호출만을 탐지한다. mprotect는 다양한 시스템 라이브러리에서 공유 라이브러리 적재 등과 같은 이유로 호출한다. 앱에서 호출하는 동적 코드 수정을 정확히 구분하기 위해서는 앱 프로세스 내에서 mprotect를 호출하는 라이브러리를 구분하여야 한다. 동적 분석 모듈은 mprotect 호출을 감지한 순간 앱의 메모리 맵(/proc/pid/maps)을 탐색하여 호출 라이브러리를 확인한다. 호출 라이브러리가 libc.so와 같은 시스템 라이브러리가 아니고 앱에 포함된 라이브러리일 경우에만 동적 코드 수정으로 판단한다.

4.3.5 안티 리패키징 탐지

안티 리패키징 탐지는 앱을 수정 후 실행하여 종료를 감지하는 것으로 구현한다. 안티 리패키징은 APK 서명 검사를 포함하므로, 입력한 앱에서 APK 서명을 제거한 후 실행한다. 이후 액티비티 매니저 서비스에서 앱 종료시 로그를 기록하여 안티 리패키징을 감지한다. 거짓 양성(false positive)을 줄이기 위해 동적 분석 모듈은 먼저 수정되지 않은 원본 앱을 실행하여 종료를 감지한다. 다음으로 개발자의 APK 서명을 제거한 변조 앱을 실행하여 종료를 감지한다. 원본 앱에서는 종료가 감지되지 않고, 위조 앱에서 종료가 감지되면 앱이 안티 리패키징 기능을 적용하는 것으로 판단한다.

Ⅴ. 애플리케이션 분석 결과

5.1 수집한 데이터

2019년 3월 13일자로, Google Play Store에서 무료로 다운로드 가능한 앱을 수집하였다. Google Play Store에서 표기하는 일반 분야 앱 중 최고 매출 앱 상위 600개를 수집하였다. 수집한 600개의 앱에서 가장 많은 사용율을 보이는 세 개의 분야를 선별하고, 각 분야에서 상위 300개 앱을 수집하였다. 선별한 분야는 각각 게임, 금융, 쇼핑 분야이며 중복된 앱을 제외하고 총 1322개의 앱을 수집하였다.

5.2 탐지 결과

그림 2는 수집한 앱에 대한 보호 기능 적용률을 조사한 결과이다. 순서대로 1322개 전체 앱과 600개의 일반 분야 앱에 대한 결과, 뒤이어 각각 300개의 게임, 금융, 쇼핑 분야의 조사 결과를 나타내었다.

전체 앱에 대한 결과를 평균으로 보면, 금융 분야의 경우 다섯 가지 보호 기능 중 네 가지 보호 기능에 대해 평균 이상의 적용률을 보이고, 게임 분야의 경우 세 가지에 대해 평균 이상의 적용률을 보이나, 쇼핑 분야의 경우 상대적으로 구현이 쉬운 루트 인터페이스 감지를 제외하고는 모두 평균 이하의 적용률을 보인다. 특히 쇼핑 분야에서는 안티 디버깅을 전혀 적용하지 않는다. 모든 분야의 앱을 일부 포함하는 일반 분야의 적용률은 전체 앱의 적용률에 비해 낮다.

결과를 통해 전체 보호 기능의 적용률이 38.42% 이하로, 61.58%의 앱들이 공격에 취약하다는 점을 확인하였다. DEX 암호화와 동적 코드 수정의 경우 각각 36.38%, 6.73%의 적용률로, 각각 나머지 63.62%, 93.27%의 앱이 공격자의 정적 분석에 취약하다. 루팅 인터페이스 검사는 38.42%의 가장 높은 적용률을 보이나 마찬가지로 61.58%의 앱이 간단한 루팅 환경을 막지 못한다. 안티 디버깅과 안티 리패키징의 적용 비율은 가장 위험한 상태로, 각각 2.79%와 11.49%이다. 즉, 97.21%의 앱이 디버깅을 통한 메모리 조작에 취약하고, 88.51%의 앱이 위/변조 후 배포 행위에 취약하다.

5.3 토론

과반수의 앱이 보호 기능을 적용하지 않아 보안 위협에 노출되고 있다. 특히 일반 분야 앱이 높은 사용률에도 불구하고 전체 평균 보다 낮은 적용률을 보이고 있어 실질적인 위험도는 더 높을 것으로 예상한다.

Ⅵ. 한 계

제안하는 시스템은 보호 기능을 탐지하는 과정에서 일부 한계가 존재한다.

6.1 미탐 가능성

앱이 보호 기능을 실행하는 시점에 따라 미탐 가능성이 존재한다. 제안하는 시스템은 앱을 입력한 시간

동안 실행하고 종료한다. 입력한 시간 안에 앱이 보호 기능을 실행해야만 제안하는 시스템이 탐지한다. 일반적으로 앱을 보호하기 위해서는 앱 실행 후 보호 기능이 다른 모든 기능보다 먼저 실행되나 결제 기능과 같은 특정한 기능을 이용할 경우에만 보호 기능이 실행된다면 제안하는 시스템은 이러한 앱들을 탐지할 수 없다.

안티 리패키징 기법을 검사하는 과정에서 미탐 가능성이 존재한다. 제안하는 시스템은 앱을 변조하여 실행 후 종료 여부를 확인한다. 안티 리패키징 기법 중 앱의 변조 여부를 확인하면 앱을 종료하지 않는 대신, 화면에 표시되는 요소를 변조하여 사용자의 앱 사용을 방해하는 기법이 제안되었다[5]. 제안하는 시스템은 변조 여부 파악 후에 종료하지 않는 앱은 탐지할 수 없다.

6.2 오탐 가능성

DEX 암호화의 경우 앱 개발 과정에서 오탐 가능성이 존재한다. 제안하는 시스템은 AndroidManifest.xml 파일에 선언한 클래스 이름이 classes.dex와 같은 앱 내부 파일에 속하지 않는다면 DEX 암호화로 판단한다. 앱 개발자들이 앱을 업데이트하는 과정에서 더 이상 필요 없는 클래스를 앱 내부에서 완전히 제거하였지만 AndroidManifest.xml 파일에서는 실수로 제거하지 않을 가능성이 있다. 앱 내부에 존재하지 않는 클래스가 AndroidManifest.xml 파일에 선언되어 있으므로, 제안하는 시스템은 이를 DEX 암호화로 오탐할 가능성이 있다.

Ⅶ. 결 론

본 논문에서는 안드로이드 앱 보호 실태를 파악하기 위해 구현 방법을 특정할 수 있는 다섯 가지의 보호 기능을 분류하였다. 분류한 보호 기능들을 탐지할 수 있는 방법론을 제시하고, 방법론을 기반으로 보호 기능 탐지 시스템을 구현하였다. 설계한 시스템으로 현 상위 매출 1322개의 앱을 분석하여 61% 이상의 앱이 공격에 노출되어 있음을 지적하였다. 추후에는 본 논문에서 탐지하지 않는 보호 기능들의 구현 방법을 특정하여 구현한 시스템의 기능과 앱 보호 실태 조사 범위를 확장하고자 한다.

References

[1] Y. Zhou and X. Jiang, "Dissecting android malware: characterization and evolution," *IEEE Symp. Secur. and Privacy (S&P)*, pp. 95-109, San Francisco, USA, May 2012.

[2] iBotPeaches, *Apktool*, Retrieved Mar. 13, 2019, from https://github.com/iBotPeaches/Apktool.

[3] R. Yu, "Android packers: facing the challenges, building solutions," *Virus Bulletin Int. Conf. (VB)*, pp. 266-275, Seattle, USA, Sep. 2014.

[4] L. Dresel, M. Protsenko, and T. Muller, "ARTIST: The android runtime instrumentation toolkit," *Int. Conf. ARES*, pp. 107-116, Salzburg, Austria, Sep. 2016.

[5] L. Luo, Y. Fu, D. Wu, S. Zhu, and P. Liu, "Repackage-proofing android apps," *IEEE/IFIP DSN*, pp. 550-561, Toulouse, France, Jun. 2016.

[6] Y. Zhang, X. Luo, and H. Yin, "Dexhunter: Toward extracting hidden code from packed android applications," *ESORICS*, pp. 293-311, Vienna, Austria, Sep. 2015.

[7] V. Haupert, D. Maier, N. Schneider, J. Kirsch, and T. Muller, "Honey, I shrunk your app security: The state of android app hardening," *DIMVA*, pp. 69-91, Paris, France, Jun. 2018.

[8] W. Lee and K. Lee, "A study on the vulnerability of using intermediate language in android : Bypassing security check point in android-based banking applications," *J. KIISC*, vol. 27, no. 3, pp. 549-562, Jun. 2017.

[9] J. You, M. Han, K. Kim, J. Jang, H. Jin, H. Ji, J. Shin, and K. Kim, "A study on method for bypassing verification function by manipulating return value of android payment application's security solution," *J. KIISC*, vol. 28, no. 4, pp. 827-838, Aug. 2018.

[10] Z. Qu, S. Alam, Y. Chen, X. Zhou, W. Hong, and R. Riley, "DyDroid: Measuring dynamic code loading and its security implications in android applications," *IEEE/IFIP DSN*, pp. 415-426, Denver, USA, Jun. 2017.

[11] Y. Duan, M. Zhang, A. V. Bhaskar, and H. Yin, "Things you may not know about android (un) packers: a systematic study based on

whole-system emulation," *NDSS*, pp. 18-21, San Diego, USA, Feb. 2018.

[12] Google, *Android Open Source Project*, Retrieved Mar. 13, 2019, from https://source.android.com/.

조 동 민 (Dongmin Jo)

2018년 2월 : 성균관대학교 컴퓨터공학과 학사

2018년 3월~현재 : 성균관대학교 전자전기컴퓨터공학과 석사과정

<관심분야> 안드로이드 보안, 시스템 보안, 취약점 분석

최 형 기 (Hyoung-Kee Choi)

1992년 2월 : 성균관대학교 전자공학과 학사

1996년 2월 : Polytechnic University in Brooklyn, NY 석사

2001년 2월 : Georgia Institute of Technology in Atlanta, GA 박사

2001년 1월~2004년 12월 : Lancope 근무

2004년 3월~현재 : 성균관대학교 소프트웨어대학 교수

<관심분야> 네트워크 보안, 리버스 엔지니어링

[ORCID: