공학석사학위논문

안드로이드 보안 강화를 위한 새로운 앱 마켓 시스템

A Novel App Market System for Enhancing Security on Stock Android

2013년 8월

서울대학교 대학원

전기·컴퓨터공학부

전 철

안드로이드 보안 강화를 위한 새로운 앱 마켓 시스템

A Novel App Market System for Enhancing Security on Stock Android

지도교수 조유근

이 논문을 공학석사 학위논문으로 제출함
2013년 6월

서울대학교 대학원
전기·컴퓨터공학부
전 철

전철의 공학석사 학위논문을 인준함
2013년 6월

위 원 장 하 순 회 (인)

부위원장 조 유 근 (인)

위 원 이 창 건 (인)

국문초록

안드로이드 보안 취약점을 이용한 악성 애플리케이션의 수가 빠르게 증가함에 따라 안드로이드 보안을 강화하기 위한 여러 연구들이 진행되었다. 그러나 제안된 대부분의 기법들은 안드로이드 운영체제와 시스템의 광범한 수정을 필요로 하기 때문에 각 제조사를 통해 출시된 안드로이드 장치를 사용 중인 사용자들에게 적용될 수 있는 실용적인 방법은 아니다.

본 논문에서는 안드로이드 시스템 수정 없이 보안을 강화할 수 있는 새로운 앱 마켓 시스템을 제안한다. 제안하는 시스템은 각 애플리케이션 실행파일에 IRM (Inline Reference Monitor)를 삽입함으로써 악성 애플리케이션이 마켓의 검수과정을 우회하여 사용자 폰에 설치되더라도 사용자가 직접 애플리케이션의 동작을 감시하고 통제할 수 있도록 한다. 뿐만 아니라 제안하는 시스템은 경우에 따라 사용자에게 보안패치 (Security Patch)가 적용된 버전의 애플리케이션을 제공한다. 이러한 보안패치는 공격자가 애플리케이션 또는 운영체제 취약점을 이용한 공격을 할 수 없도록 차단한다.

제안하는 마켓 시스템은 안드로이드 실행파일을 재작성하고 재패키징하기 위한 툴인 ADR(Android DEX Rewriter)을 제공한다. ADR은 오픈 소스 툴인 Apktool에 비해 빠른 성능을 가지기 때문에 많은 수의 애플리케이션을 효율적으로 재패키징 할 수 있을 뿐만 아니라 적은 비용으로 기존 앱 마켓 시스템에 적용될 수 있다.

안드로이드 공식 앱 마켓에서 다운으로 받은 100개의 애플리케이션을 대상으로 재패키징을 수행하였으며 성공적으로 IRM이 애플리케이션에 삽입됨을 확인하였다.

주요어 : 모바일, 안드로이드 보안, 앱 마켓

학 번 : 2011-23381

목 차

표 목 차

그 림 목 차

제 1 장 서 론

안드로이드는 지난 몇 년 간 스마트폰 분야에서 가장 빠르게 성장해 왔으며 현재는 전 세계 스마트폰 시장의 70%를 자지 할 정도로 사용자들에게 가장 인기 있는 스마트폰 플랫폼이 되었다[27]. 그러나 이렇게 안드로이드가 급속히 대중화됨에 따라 안드로이드의 보안 취약점은 부각되기 시작했고 이를 악용한 악성 애플리케이션의 수도 급증하였다. 통계자료에 따르면 안드로이 악성 애플리케이션의 수는 2010년과 2011년 사이에 400% 증가한 것을 알 수 있다[28].

안드로이드 보안 문제는 주로 coarse-grain 퍼미션, all-or-nothing 결정, 보안정책 실패, 개발자의 보안 의식 결여, 그리고 오픈소스(opensource)와 관련되어 있다. 이러한 안드로이드 보안 문제점들을 보완하기 위해 여러 연구가 진행되었으며 그 결과로 여러 기법들이 제안되었다.

coarse-grain 퍼미션 문제와 all-or-nothing 결정 문제를 다루기 위해 여러 연구들이 진행되었다[2,11,12,21,23]. 실제 제안된 기법들은 안드로이드 퍼미션 관련 문제를 해결하는데 효과적이었지만 광범위한 안드로이드 운영체제와 애플리케이션 프레임워크의 수정을 요구하는 단점이 있다.

폰 안에 저장된 민감한 개인정보가 외부로 유출되는 문제를 다루기 위해 정적 코드 분석(static code analysis) 기법과 동적 오염 분석(dynamic taint analysis) 기법이 사용되었다[16,19,20]. 이러한 툴 들은 온/오프라인(offline) 상태에서만 데이터의 흐름을 분석하여 탐지만 할 수 있을 뿐 차단을 할 수 없다는 단점이 있다.

안드로이드는 기본적으로 애플리케이션 컴포넌트에 대한 외부 접근을 허용하고 있다. 이러한 정책은 여러 문제점을 야기하였고 그 중에서도 주로 권한 상승(Privilege Escalation) 공격의 원인이 되었다.

권한 상승 공격은 주로 높은 권한을 가진 애플리케이션의 취약한 인터페이스를 악용하여 낮은 권한을 가진 애플리케이션 자신이 수행할 수 없는 일을 높은 권한을 가진 애플리케이션이 대신 수행하도록 만든다.

특히, 안드로이드에서는 애플리케이션 컴포넌트에 대한 외부 접근을 기본적으로 허용하기 때문에 이러한 공격의 비중은 상당히 높은 편이다. 이러한 권한 상승 문제는 여러 연구에서 다루어졌다[13,22,24].

최근 연구 중에서 앞서 설명한 연구들의 부족한 실용성의 한계를 극복하기 위해 애플리케이션 수준에서 보안을 강화할 수 있는 기법들이 제안되다[3,4,7]. 이러한 기법은 애플리케이션 실행파일에 IRM(Inline Reference Monitor)를 삽입하기 하기 때문에 안드로이드 시스템과 애플리케이션 프레임워크의 수정을 요구하지 않아 순정 안드로이드 운영체제를 사용하는 이용자들에게 실용적인 장점이 있다.

그러나 이러한 기법들은 역시 몇 가지 문제점을 가지고 있다. 먼저, 정확성(Accuracy) 문제가 있다. 이 연구들에서 사용하는 오픈소스 툴(Tool)인 Apktool[26]의 버그 때문에 일부 애플리케이션에 IRM을 정상적으로 삽입할 수 없는 문제가 발생하였다[4]. 또한 Apktool은 내부적으로 자바 언어로 작성된 달빅 머신 코드의 어셈블러/디스어셈블러인 smali/baksmali를 사용하기 때문에 속도가 느리다. 따라서 확장성(Scalability)에도 문제가 있다. 그리고 개발자 서명 훼손 문제가 있다. 기존 애플리케이션에 IRM을 삽입하고 리패키징(Repackaging)을 할 때 반드시 서명을 해야 하는데 원 개발자의 키로 서명할 수 없어 임의의 키를 생성하여 서명한다는 문제점이 있다. 따라서 원 개발자의 신원을 확인할 수 없을 뿐만 아니라 원 개발자가 설정한 애플리케이션 간의 신뢰 관계를 보존 할 수 없게 되어 애플리케이션의 오작동 (side-effect)를 야기할 수 있다. 마지막으로 분할 실행 코드 이용한 우회공격을 막을 수 없다.

애플리케이션 수준 보안 중에서 AppGuard[3]는 몇 가지 추가적인 문제를 가지고 있다. AppGuard는 앱 형태로 제작되어 설치된 애플리케이션을 대상으로 사용자가 직접 사용해 볼 수 있다는 장점이 있지만 앱 마켓에서 보호(Protected)로 설정된 애플리케이션에는 IRM을 삽입할 수 없으며 Apktool의 느린 성능이 더 많은 배터리를 소비를 야기할 수 있다는 단점이 있다.

본 논문에서는 기존 애플리케이션 수준 보안의 한계점들을 극복하고

범용적으로 모든 안드로이드 디바이스의 보안을 향상 시킬 수 있는 새로운 앱 마켓 시스템을 제안한다. 제안한 앱 마켓 시스템은 기존에 제안된 애플리케이션 수준 보안의 단점인 정확성 (accuracy), 확장성 (scalability)를 모두 만족 시킨다. 이를 위해 Apktool을 사용하는 대신에 안드로이드 실행파일(.dex)을 바이너리 수준에서 재 작성할 수 있는 ADR (android dex rewriter)를 개발하였다. 그리고 개발자 서명 문제와 분할 실행 코드를 이용한 우회 공격을 다루기 위해 중앙 집중적인 방식으로 개발자 서명 키를 보관하도록 설계했으며 분할 실행 코드를 위한 인증 시스템을 따로 두어 인증되지 못한 분할 실행 코드가 애플리케이션을 통해 수행될 수 없도록 하였다.

추가적으로 제안된 마켓 시스템에서는 운영체제와 애플리케이션에서 발생한 취약점도 다룬다.

안드로이드에서 발생한 보안 취약점에 대한 패치(patch)가 각 제조사에 의해 적용되기까지 오랜 시간이 걸리며 사후지원 기간이 종료된 안드로이드 디바이스의 경우에는 보안 취약점이 그대로 방치될 가능성이 높다. 따라서 제안하는 마켓 시스템에서는 이러한 문제에 빠르게 대응하기 위해 먼저, 이를 해결할 수 있거나 완화할 수 있는 새로운 정책을 수립하고 수립된 정책을 반영한 IRM 패치를 생성한다. 이렇게 생성된 IRM 패치는 앱 마켓에 등록되고 사용자 디바이스에 설치된 모든 애플리케이션 혹은 특정 애플리케이션에 일괄적으로 적용된다. 따라서 기존 보다 더 빠르게 보안 취약점에 대응할 수 있게 되어 그 취약점 따른 문제를 최소화 할 수 있게 된다.

제안된 마켓 시스템을 통해 설치된 모든 애플리케이션은 IRM을 통해 그 행위가 모두 관찰되며 특히, 의심스러운 행위로 간주는 되는 것들은 모두 사용자에게 보고된다. 보고된 정보를 통해 사용자는 그러한 의심스러운 행위를 허용하거나 차단 할 수 있으며 여의치 않을 경우에는 가짜 값을 반환하게 할 수도 있다[2,11,21]. 예를 들어 애플리케이션 폰 번호를 요청 할 때 000-0000-0000과 같은 가짜 번호를 넘겨주는 것이다.

결과적으로 우리가 제안한 새로운 앱 마켓 시스템은 기존 애플리케이

션 수준 보안의 한계를 극복하고 사용자에게는 실시간으로 애플리케이션의 의심스러운 행위를 감독하고 통제 할 수 있는 능력을 부여하여 한층 더 강화된 안드로이드 보안을 경험할 수 있게 한다.

본 논문은 다음과 같이 구성된다. 2장에서는 안드로이드 소개와 안드로이드 보안 매커니즘인 샌드박스, 개발자 서명, 그리고 퍼미션을 다룬다. 3장에서는 관련 연구들을 살펴본다. 4장에서는 본 논문에서 제안하는 앱 마켓 시스템에 대해 살펴본다. 5장에서는 실험결과를 보이고 제안 기법에 대해 평가를 한다. 마지막으로 6장에서는 결론을 맺는다.

제 2 장 안드로이드 개요

2.1 안드로이드 소개

안드로이드는 Android inc. 社에 의해서 개발되었고 2005년도에 Android Inc. 社가 구글 (Google)에 인수된 후 부터는 빠르게 성장하여 현재는 가장 많은 사용자를 보유한 스마트폰 플랫폼이 되었다.

안드로이드 프로젝트에는 구글 뿐만 아니라 Open Handset Alliance (OHA)에 속한 Intel, TI, T-Mobile, and NTT DoCoMo 등의 기업들이 참여하고 있으며 설계, 개발, 그리고 배포를 위해서 서로 협력하고 있다. 초창기부터 안드로이드 프로젝트는 오픈소스 정책을 통해 소스코드를 모두 공개하고 있으며 제조사에서 이를 원하는 대로 수정할 수 있도록 허용하고 있다. 이 같은 구글의 정책은 과거 애플 (Apple) 社의 iOS, 리서치인모션 (Research In Motion) 社의 블랙베리 (BlackBerry) , 그리고 노키아 (Nokia)의 심비안 (Symbian) 같은 독자적인 스마트폰 운영체제가 없었던 제조사들에게 큰 환영을 받았고 빠른 속도로 안드로이드 운영체제가 보급될 수 있었던 계기가 되었다. 2012년 Gartner의 통계자료에 따르면 안드로이드는 현재 스마트폰 시장의 약 60%를 차지하고 있다.

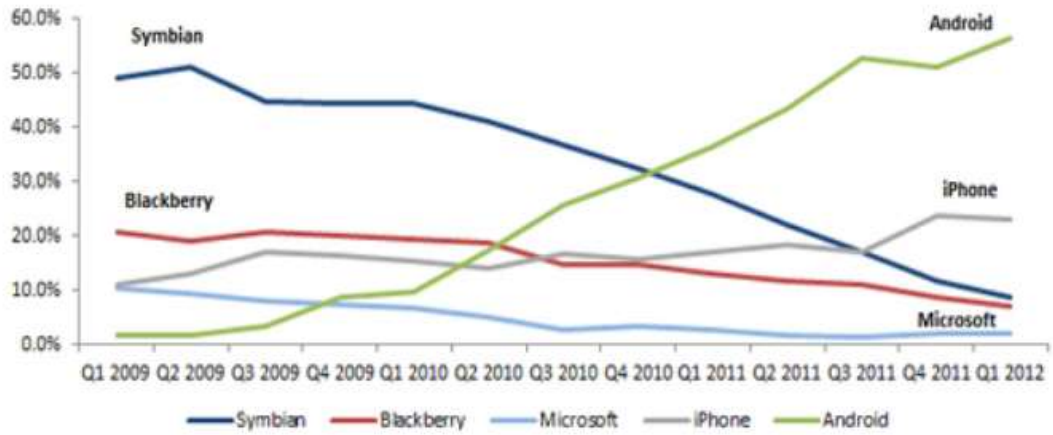

그림 2.1: 세계 스마트폰 시장 점유율 - 가트너 (Gartner)

안드로이드 운영체제는 리눅스 커널을 기반으로 하고 있다. 그러나 안드로이드가 사용하고 있는 리눅스 커널은 기존 리눅스 커널과 몇 가지 차이점을 갖는다. 기존 리눅스 커널은 기존 PC 나 서버 환경을 고려하여 구현되었기 때문에 모바일 장치 (Mobile Device)의 특성을 반영하지 못 했을 뿐만 아니라 제한된 자원을 사용하는 모바일 장치에 사용하기에는 무리가 있다. 따라서 안드로이드에서는 모바일 장치의 특성에 맞도록 메모리 관리자 (Memory Manager)와 전원 관리자 (Power Manager) 같은 일부 커널의 기능과 그 외에 라이브러리나 장치 드라이버를 수정하였다. 그러나 일부 라이브러리의 라이센스 (license) 문제로 몇 가지 기술과 라이브러리를 직접 개발하였다. 예를 들어, GNU C 라이브러리 대신에 Bionic 이라는 이름의 새로운 C 라이브러리를 직접 구현 하였으며 또한 기존 썬 (Sun)社의 자바 가상머신 대신에 달빅 (Dalvik) 가상머신을 개발하였다.

2.2 안드로이드 시스템 구조

안드로이드 시스템은 계층 방식 (layered approach)으로 설계 되었다. 애플리케이션 (application), 애플리케이션 프레임워크 (application framework), 라이브러리, 안드로이드 런타임 (runtime), 그리고 리눅스 커널 이렇게 총 5개의 계층으로 이루어져 있으며 각 계층 별로 다른 역할을 하고 있다.

2.2.1 리눅스 커널

리눅스 커널은 안드로이드 시스템의 근간을 이루며 시스템을 동작 시키기 위한 가장 중요한 역할을 한다. 주로 하는 일은 메모리 관리, 프로세스 관리, 그리고 네트워크 통신 뿐만 아니라 모바일 장치를 다루기 위한 각종 장치 드라이버(device driver)를 관리한다.

안드로이드에 적용된 리눅스 커널은 기존 리눅스 커널에 비하여 대폭 수정된 버전이며 메모리 관리, 프로세스 관리, 전원 관리 그리고 IPC (Inter-Process Communication) 기능 등이 변경되었다. 특히, IPC 기능

은 애플리케이션간의 통신을 강화하기 위해 과거에 Palm OS에 적용되었던 Binder IPC 드라이버로 대체 되었다.

2.2.2 라이브러리

라이브러리 계층에서는 C/C++로 작성된 네이티브 (native) 라이브러리들이 존재한다. 라이브러리에는 Bionic C 라이브러리, SQLite 데이터베이스, Webkit 브라우저 엔진, 그리고 그래픽을 다루기 OpenGL ES 라이브러리 등이 존재한다. 이 외에도 그리고 사용자 인터페이스 (UI) 창을 다루기 위한 SurfaceFlinger 등이 존재한다.

2.2.3 안드로이드 런타임

안드로이드 런타임 계층에는 달빅 가상머신과 핵심 라이브러리들이 위치한다. 안드로이드의 대부분의 기능은 핵심 라이브러리를 통해 제공되지만 개발자가 직접 자바 네이티브 인터페이스 (JNI)를 통해 시스템 네이티브 라이브러리에 접근할 수 도 있다.

2.2.4 애플리케이션 프레임워크

애플리케이션 프레임워크에는 애플리케이션이 동작하는데 필요한 실질적인 모든 기능을 포함하고 있다. 애플리케이션 프레임워크에는 여러 관리자 (manager)들이 존재하며 각 관리자별로 독립적인 기능을 관리하고 애플리케이션에 서비스를 제공한다. 몇 개의 대표적인 관리자에 대해서 살펴보면 ActivityManager는 애플리케이션의 생존주기 (lifecycle)를 관리하고 LocationManager는 위치 정보를 제공하는 역할을 한다. 그리고 TelephonyManager는 통화와 문자메시지 기능을 제공한다.

2.2.5 애플리케이션

애플리케이션 계층에는 개발자들이 애플리케이션을 구현할 때 실제로 다루게 되는 컴포넌트들이 위치한다. 이러한 컴포넌트들은 대부분 애플리케이션 프레임워크의 기능을 사용하여 구현되어있다.

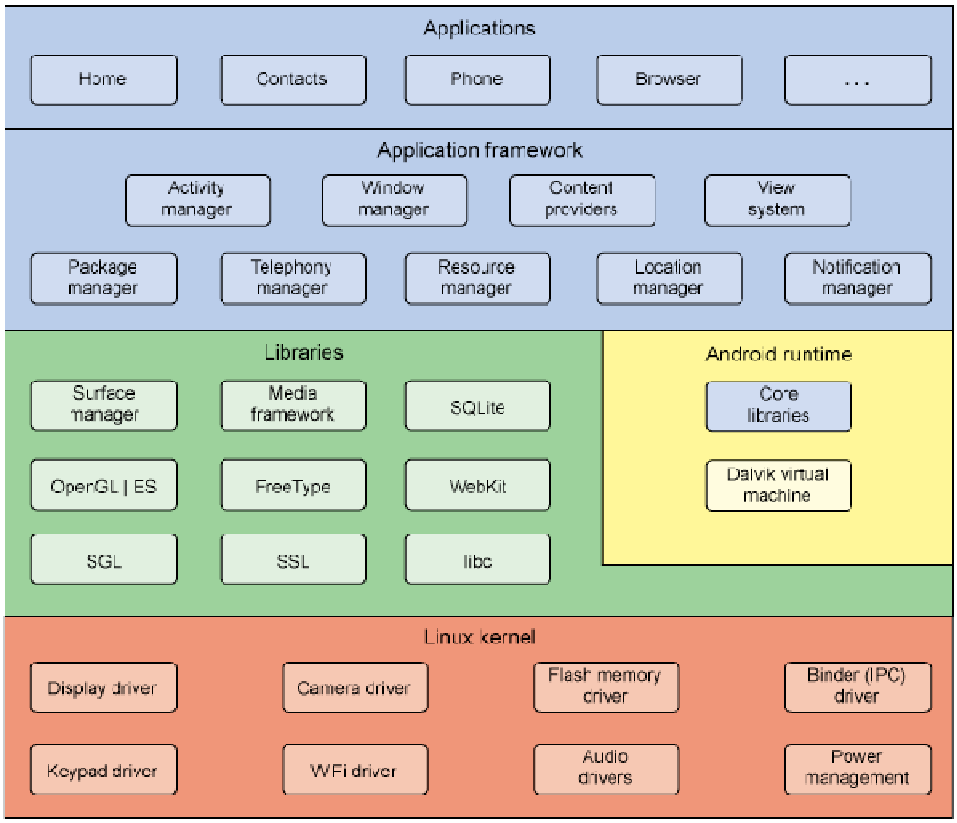

그림 2.2: 안드로이드 시스템 구조

2.3 안드로이드 애플리케이션 컴포넌트

안드로이드는 애플리케이션을 개발을 하기 위한 애플리케이션 컴포넌트를 제공하며 제공된 각 애플리케이션 컴포넌트는 애플리케이션의 시작점(entry point)이 될 수 있다.

안드로이드에서는 Activity, Service, Broadcast Receiver, 그리고 Content Provider 이렇게 총 4가지 형태의 애플리케이션 컴포넌트를 제공하고 있으며 각 컴포넌트는 서로 다른 목적과 생명주기를 가지고 있다.

2.3.1 액티비티 (Activity)

액티비티는 사용자 인터페이스를 가진 하나의 화면을 나타낸다. 예를 들어 이메일 애플리케이션은 메일의 목록을 보여주는 액티비티와 메일을

작성할 수 있게 하거나 메일의 내용을 보여주는 다른 액티비티를 가질 수 있다. 사용자 입장에서는 이런 액티비티들이 연속적으로 동작하도록 구성된 것 처럼 보이지만 실제로 하나의 액티비티는 다른 액티비티에 대하여 독립적이다. 따라서, 다른 애플리케이션들은 인터페이스가 공개된 이메일 애플리케이션의 액티비티를 실행 시킬 수 있다. 예를 들어 카메라 애플리케이션은 사용자가 사진을 이메일을 통해 공유할 수 있도록 하기 위해 이메일 애플리케이션의 액티비티를 시작 시킬 수 있다.

2.3.2 서비스 (Service)

서비스는 사용자 인터페이스를 제공하지 않으며 장기간 실행되는 연산을 백그라운드 (background)에서 수행을 시키거나 원격 (remote) 프로세스 작업을 수행하기 위한 컴포넌트이다. 예를 들어 사용자가 다른 애플리케이션을 수행하고 있는 동안 서비스는 백그라운드에서 음악을 플레이 할 수 있다.

서비스도 액티비티와 마찬가지로 다른 애플리케이션이 서비스를 시작시킬 수 있으며 서비스에 연결하여 상호작용을 할 수 있다.

2.3.3 콘텐트 프로바이더 (Content Provider)

개발자는 데이터를 파일 시스템이나 다른 영구 저장소에 저장 할 수 있다. 안드로이드에서는 개발자가 쉽게 로컬 저장소에 데이터를 저장하고 관리할 수 있도록 콘텐트 프로바이더라는 콤포넌트를 제공한다. 콘텐트 프로바이더를 통해 다른 애플리케이션들은 질의를 할 수 있으며 심지어 데이터를 수정할 수도 있다. 예를 들어 안드로이드 시스템은 사용자의 연락처를 관리하는 콘텐트 프로바이더를 제공한다. 따라서, 적절한 퍼미션을 가진 애플리케이션은 특정 사용자의 정보를 질의하거나 수정할 수 있다.

콘텐트 프로바이더는 애플리케이션의 사적인(private) 정보를 읽고 저장하는데 유용하다. 예를 들어 노트패드 (notepad) 애플리케이션은 노트를 저장하기 위해 콘텐트 프로바이더를 사용한다.

2.3.4 브로드캐스트 리시버 (Broadcast Receiver)

브로드캐스트 리시버는 시스템이나 애플리케이션에서 발생 시킨 브로드캐스트 메세지에 응답하기 위한 콤포넌트이다. 대부분의 브로드캐스트 메세지들은 시스템으로 부터 생성된다. 예를 들어 브로드캐스트는 화면이 꺼졌거나 배터리 용량이 적을 경우에 발생한다. 시스템처럼 애플리케이션 역시 브로드캐스트 메세지를 발생 시킬 수 있다. 예를 들어 어떤 데이터가 다운로드 되고 있음을 다른 애플리케이션들에게 알리고 그 데이터를 다운로드 할 수 있게 한다.

브로드캐스트 리시버 그 자체는 사용자 인터페이스를 제공하지 않지만 브로드캐스트 이벤트가 발생했을 때 사용자에게 알려줄 상태 바 알림을 생성할 수 있다.

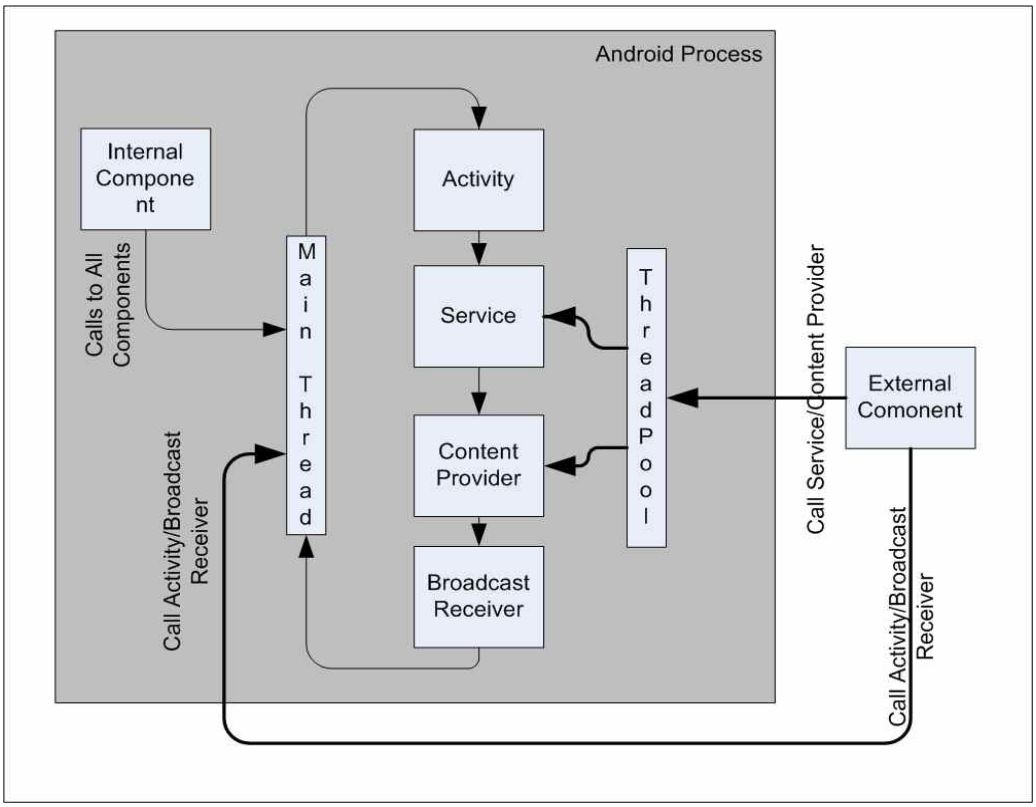

그림 2.3: 안드로이드 애플리케이션 내부 구조

2.4 안드로이드 보안 매커니즘

안드로이드 보안 모델은 전통적인 데스크탑 보안 모델과 다른 형태를 갖는다. 전통적인 데스크탑 보안 모델은 사용자를 기반으로 하기 때문에 특정 사용자에 의해서 수행된 애플리케이션은 해당 사용자가 가진 모든 권한을 가지고 동작을 하게 된다. 따라서 전통적인 데스크탑 보안 모델 안에서는 루트권한을 가진 악성 애플리케이션이 발생할 수 있으며 그 결과 시스템의 무결성이 훼손되는 문제가 발생할 수 있다. 이와는 대조적으로 안드로이드 보안 모델은 애플리케이션 기반이기 때문에 전통적인 데스크탑 모델과는 다른 몇 가지 보안 매커니즘 (mechanism)을 가지고 있다. 안드로이드 보안 매커니즘은 샌드박스 (sandbox), 애플리케이션 서명, 그리고 퍼미션 이렇게 3가지로 나눌 수 있다.

2.4.1 샌드박스 (Sandbox)

안드로이드의 샌드박스는 리눅스의 임의 접근 제어(discretionary access control)를 이용하여 설치된 각 애플리케이션에 유일한 사용자 아이디(uid)와 그룹 아이디(gid)를 할당함으로써 구현된다. 따라서 안드로이드에 설치된 애플리케이션은 기본적으로 서로의 자원에 접근할 수 없으며 각 애플리케이션은 독립된 프로세스에서 고립되어 실행된다.

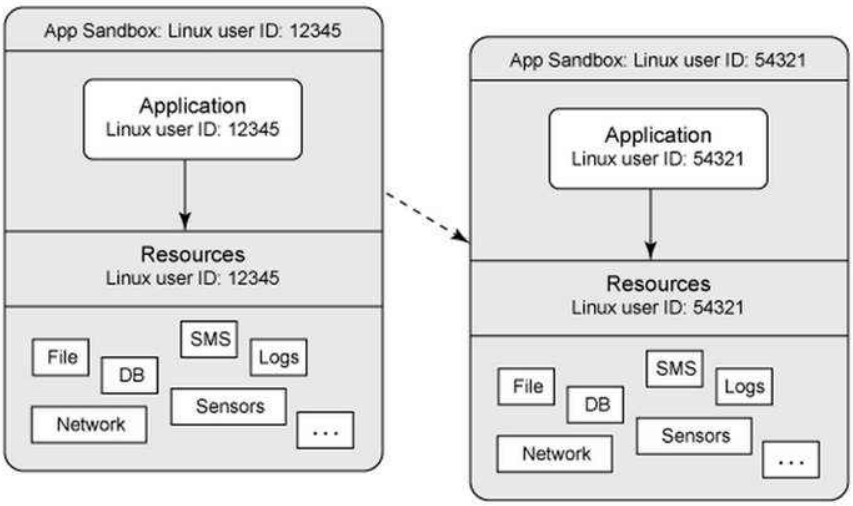

그림 2.4: 안드로이드 샌드박스 구조

2.4.2 애플리케이션 서명 (Application Signing)

애플리케이션 서명은 저작권자를 애플리케이션에 명시하기 위해 사용된다. 안드로이드 시스템은 모든 애플리케이션이 개발자 인증서를 이용하여 서명되어지도록 요구하며 서명되지 않은 애플리케이션의 설치는 허용하지 않는다.

또한 애플리케이션 서명은 애플리케이션 간의 신뢰 관계를 정의하기 위해 사용된다. 애플리케이션들이 동일한 개발자의 키로 서명되고 매니페스트 파일에 동일한 사용자 아이디를 공유한다는 것을 명시한다면 아래의 그림 2.5처럼 애플리케이션들은 서로의 자원에 아무런 제약 없이 접근할 수 있게 된다.

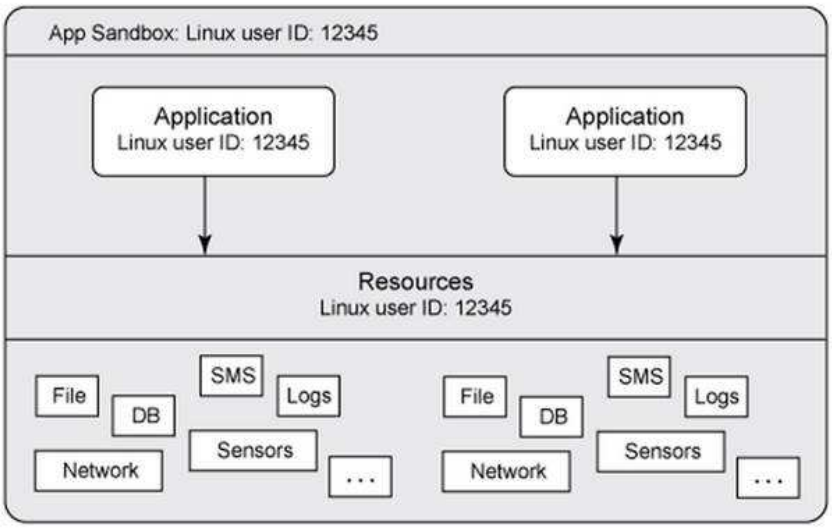

그림 2.5: 안드로이드 애플리케이션 신뢰관계

2.4.3 퍼미션 시스템 (Permission System)

퍼미션은 안드로이드의 샌드박스 매커니즘에 의해 제한된 능력을 갖는 애플리케이션이 시스템이나 다른 애플리케이션이 제공하는 기능과 자원을 사용할 수 있도록 허가한다. 이러한 접근을 다루기 위해 안드로이드에서는 또 다른 접근제어 방법인 강제적 접근 제어 (Mandatory Access Control) 방식을 사용한다.

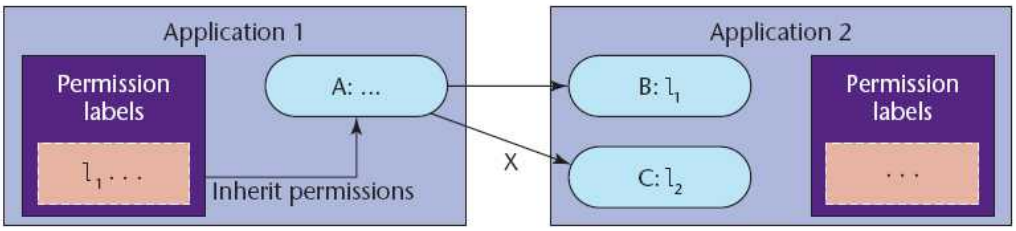

그림 2.6: 퍼미션을 이용한 애플리케이션 컴포넌트간의 접근제어

그림 2.6은 애플리케이션 1에서 애플리케이션 2로 접근하기 위한 과정을 나타낸다. 애플리케이션 2는 콤포넌트 B와 C의 접근을 보호하기 위해 각각 퍼미션 l1과 l2를 선언하고 있다. 따라서 외부 애플리케이션은 컴포넌트 B 또는 C에 접근하기 위해선 퍼미션 l1 또는 l2를 보유하고 있어야 한다. 그러나 애플리케이션 1이 허가받은 퍼미션 목록에는 l1 퍼미션만 있고 l2 퍼미션은 없으므로 애플리케이션2의 컴포넌트 B에는 접근할 수 있지만 컴포넌트 C에는 접근할 수 없다. 또한 시스템의 기능에 대해서도 동일한 방식으로 접근 제어를 한다.

현존하는 퍼미션 시스템은 퍼미션을 허가 하시는 시점에 따라서 설치시간 (Install-Time) 퍼미션 시스템과 사용시간 (Time-of-Use) 퍼미션 시스템으로 분류될 수 있다.

(1) 설치시간 퍼미션 시스템 vs. 사용시간 퍼미션 시스템

퍼미션 시스템은 일반적으로 설치시간(install-time) 시스템과 사용시간(time-of-use) 시스템으로 나누어진다. 설치시간 퍼미션 시스템은 애플리케이션이 설치될 때 애플리케이션에 의해 요청되어진 모든 퍼미션을 사용자에 보여준다. 이 퍼미션 시스템 안에서 사용자는 오직 모든 퍼미션을 허가하여 설치를 허용하거나 모두 거절하여 설치를 거부할 수만 있다. 이러한 특성은 all-or-nothing 결정이라고 불린다. 따라서 실행 중에 할당된 퍼미션을 동적으로 회수(revoke)하는 것이 불가능하며 오직 삭제를 통해 회수 할 수 있다. 반면에 사용시간 시스템 안에서 애플리케이션은 실행 중에 필요한 퍼미션을 요청한다. 따라서 사용자는 동적으로 퍼미션을 허용하거나 거부 할 수 있다. 현존하는 스마트폰 플랫폼들이 적

용한 퍼미션 시스템을 살펴보면 iOS는 사용시간 퍼미션을 사용하며 안드로이드는 설치시간 퍼미션 시스템을 사용한다. 따라서 안드로이드는 설치시간 퍼미션 시스템이 가지는 특징을 그대로 갖는다.

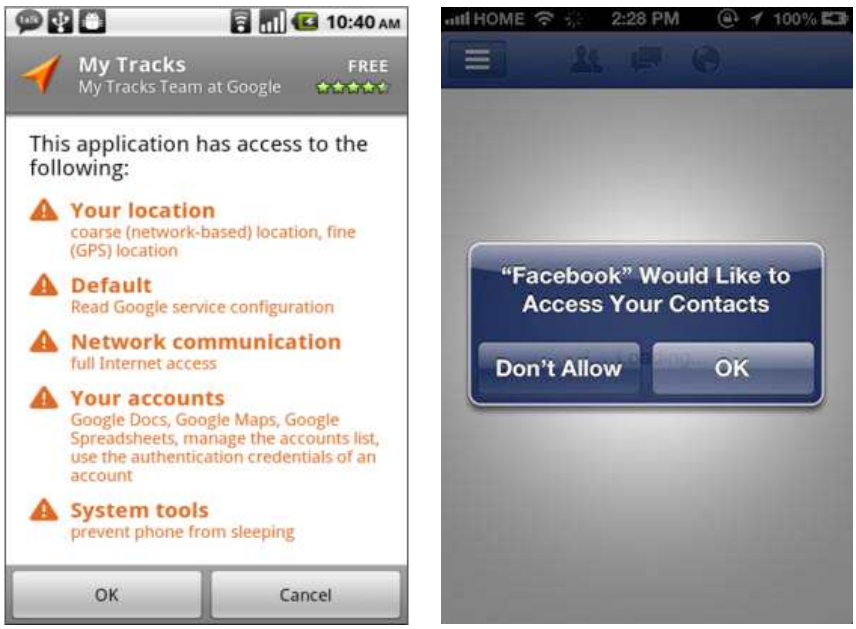

그림 2.7: 설치시간 퍼미션 시스템(안드로이드) vs. 사용시간 퍼미션 시스템(iOS)

(2) 안드로이드 퍼미션 시스템의 장단점

설치시간(install-time) 퍼미션 시스템의 장점은 사용자 동의(user consent), 종심방어(defense-in-depth), 검수 분류 (review triaging)이 있다[3]. 첫째로, 애플리케이션에서 요구하는 모든 퍼미션은 사용자 동의가 있어야 사용이 가능하다. 만약 악의적인 행위가 의심이 될 경우 사용자는 요구된 퍼미션의 사용을 거부할 수 있다. 둘째로, 설치된 애플리케이션의 취약점의 영향은 해당 애플리케이션에 허가된 퍼미션으로 한정된다. 마지막으로 애플리케이션을 검수하는 시간을 줄이는데 도움을 준다. 애플리케이션 검수자(reviewer)는 위험한(dangerous) 퍼미션을 사용하는 애플리케이션만을 집중적으로 검토하면 되기 때문이다.

설치시간(install-time) 퍼미션 시스템의 대표적인 단점은 coarse-grain 퍼미션과 all-or-nothing 결정이다. 하나의 퍼미션은 특정

시스템 API들의 집합을 나타낸다. 다루는 집합의 크기에 따라서 그 집합의 크기가 작을 경우 fine-grain 퍼미션이라 하고 반대의 경우 coarse-grain 퍼미션이라고 한다. fine-grain 퍼미션은 다루는 API 집합의 크기가 작아 그것을 사용하는 개발자의 부담이 크지만 필요 이상의 기능을 애플리케이션에 허가할 가능성이 적다는 장점을 갖는다. 반면에 coarse-grain 퍼미션의 경우에는 개발자의 부담은 적지만 필요 이상의 기능을 애플리케이션에 부여할 가능성이 높다. 특히 사용자의 개인정보 접근을 다루는 퍼미션이 coarse-grain 하다면 해당 퍼미션을 사용하는 애플리케이션은 필요 이상의 권한을 가지게 되며 사용자는 해당 퍼미션을 사용하는 애플리케이션의 정확한 의도를 파악하기 어렵게 된다. 예를 들어 안드로이드 애플리케이션은 READ_PHONE_STATE 퍼미션만 할당 받으면 기본적인 장치 정보 뿐만 아니라 민감한 사용자 정보인 폰 번호, 유심(USIM)정보 등과 같은 중요한 폰 정보에 접근할 수 있다. 따라서 애플리케이션이 이러한 coarse-grain 퍼미션을 요청할 경우 사용자는 그 사용 목적을 정확히 판단하기 어렵다. 또한 설치시간 퍼미션 시스템의 all-or-nothing 결정 특성으로 인하여 사용자는 요청된 퍼미션의 일부 만을 선택적으로 허용할 수 없고 오직 요구된 퍼미션을 모두 허가하여 설치가 되도록 하거나 설치를 거부할 수 밖에 없다. 안드로이드 퍼미션 시스템에 대한 전문적인 이해와 지식이 없는 대부분의 사용자들은 원하는 기능을 사용하기 위해 설치를 허가하는 경우가 많다.

2.4.4 접근 제어 (Access Control)

앞서 설명한 대로 안드로이드 시스템은 임의 접근 제어 (DAC)와 강제 접근 제어 (MAC) 이렇게 두 가지의 접근 제어를 사용하고 있으며 서로 다른 계층과 자원을 대상으로 적용되고 있다. 아래의 그림에서 보여지듯이 애플리케이션 콤포넌트 간의 통신과 네트워크 소켓 통신을 제어하기 위해 강제 접근를 사용하며 파일 시스템과 기존 리눅스 IPC를 제어하기 위해서는 임의 접근 제어를 사용하고 있다.

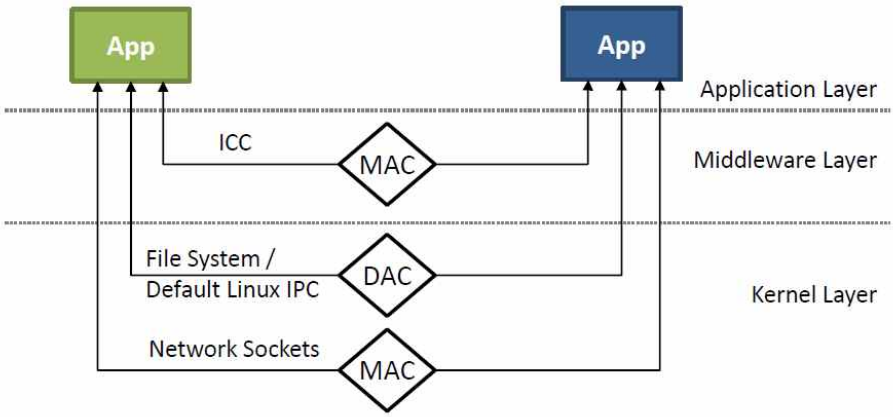

그림 2.8: 안드로이드 접근제어

제 3 장 관련연구

3.1 운영체제 수준 보안

3.1.1. Kirin

Enck는 퍼미션 집합을 이용하여 악성 애플리케이션을 탐지하는 Kirin이라는 시스템을 제안하였다[12]. Kirin은 그림 1처럼 안드로이드 애플리케이션 인스톨러를 확장하였고 설치 시에 애플리케이션 매니페스트(manifest) 파일을 분석하여 애플리케이션이 요구하는 퍼미션의 조합을 검사한다. 만약 퍼미션의 조합이 보안 룰에 위배되는 경우 해당 애플리케이션의 설치는 거부된다.

Kirin에서 위험한 퍼미션의 조합으로 분류되는 조건은 GPS, 폰 정보 등과 같은 민감한 정보에 접근할 수 있는 소스 (source) 퍼미션과 외부로 정보를 유출할 수 있는 능력을 가진 싱크 (sink) 퍼미션이 조합을 이루는 경우다.

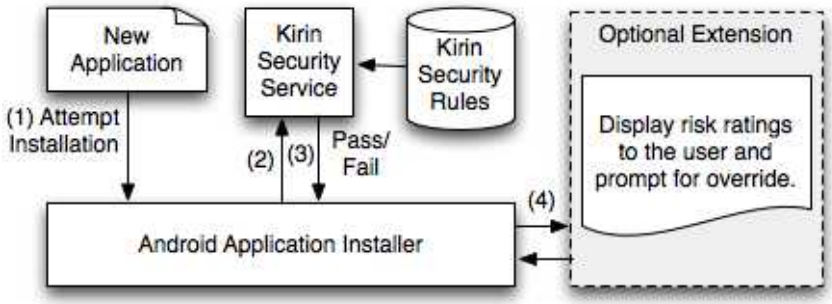

그림 3.1: Kirin 구조

3.1.2 Saint

Saint[23]는 기존 안드로이드 퍼미션 시스템을 확장하여 개발자가 애플리케이션 매니페스트 파일에 각 컴포넌트에 대한 접근 정책을 기존 보다 더 상세하게 기술할 수 있도록 하였다. Saint가 적용된 안드로이드 시스템 안에서 개발자는 각 애플리케이션 콤포넌트를 호출할 수 있는 애

플리케이션을 구체적으로 명시할 수 있다. 이를 통해 Saint는 애플리케이션 인터페이스를 보호할 수 있으며 그 결과 권한 상승 공격 (privilege escalation attack)에 의해 애플리케이션 인터페이스가 악용되는 문제를 해결한다.

그림 2에서 보여지듯이 Saint는 안드로이드 애플리케이션 인스톨러를 확장하여 설치 시에 애플리케이션 매니페스트 파일에 기술된 콤포넌트 접근 정책을 추출하여 Saint App Policy Provider에 저장한다. 이렇게 저장된 앱 정책 (App Policy)은 애플리케이션 콤포넌트간의 통신이 발생할 때 마다 Saint Mediator에 의해 읽혀지며 Saint Mediator는 호출하는 애플리케이션이 호출되어지는 애플리케이션의 앱 정책을 만족할 경우에만 통신을 허가한다.

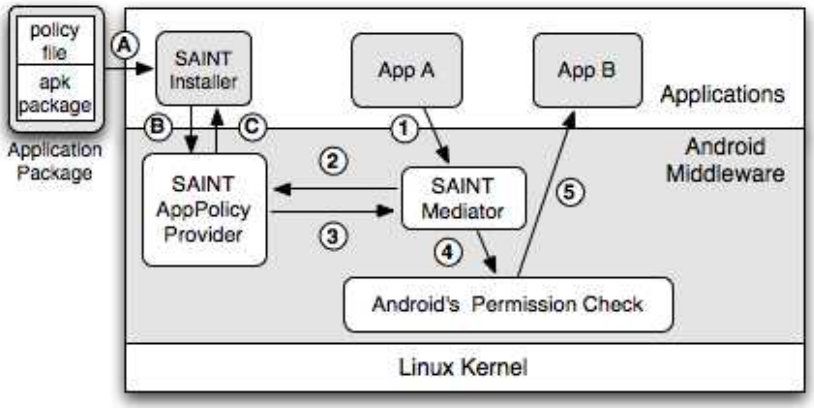

그림 3.2: Saint 구조

3.1.3 Apex

Apex[2]는 안드로이드 퍼미션 시스템의 단점인 coarse-grain 퍼미션 문제와 all-or-nothing 결정 문제를 다루기 위해 Nauman에 의해 제안되었다. coarse-grain 퍼미션은 하나의 퍼미션이 많은 API 다루기 때문에 발생하며 이러한 퍼미션을 가진 애플리케이션은 필요 이상의 기능을 갖게 된다. 그리고 all-or-nothing 결정 특성은 설치 시에 사용자가 애플리케이션이 요구하는 모든 퍼미션을 허가해야만 애플리케이션을 설치가 가능하게 하며 애플리케이션 실행 중에 동적으로 권한을 회수할 수 없게

한다. 따라서 사용자는 선택적으로 퍼미션을 허가할 수 없기 때문에 대부분의 경우 원하는 기능을 사용하기 위해 의심이 가는 애플리케이션의 설치를 허가하게 된다.

이러한 안드로이드 퍼미션 시스템의 단점을 해결하기 위해 실행 중에 애플리케이션의 권한을 동적으로 회수할 수 있는 시스템을 제안하였다. Apex는 애플리케이션이 민감한 정보나 자원에 접근하려는 행위를 사용자에게 보고하고 사용자가 그러한 동작을 허용할 것인 차단할 것인지 결정할 수 있게 한다.

3.2 정적 & 동적 프로그램 분석

3.2.1 ScanDroid (Security Certifier for Android)

안드로이드 애플리케이션은 자신의 데이터와 기능을 다른 애플리케이션과 공유할 수 있다. 안드로이드에는 데이터를 저장하고 공유를 편리하게 하기 위해 콘텐트 프로바이더라는 콤포넌트를 제공한다. 그러나 공유를 하는 과정에서 외부의 접근이 신중하게 다루어지지 못한다면 중요한 정보가 외부로 유출될 가능성이 높다.

Scandroid[19]는 이러한 문제를 다루기 위해 애플리케이션을 분석하여 애플리케이션 간의 데이터 흐름을 정적으로 분석한다. 특히, 콘텐트 프로바이더를 중심으로 퍼져나가는 데이터의 흐름을 추적한다. 그러나 애플리케이션의 소스코드를 필요로 하기 때문에 분석할 수 있는 애플리케이션은 한정되어 있다.

3.2.2 Stowaway

안드로이드 API는 퍼미션에 의해 보호된다. 이렇게 퍼미션에 의해 보호된 API를 애플리케이션에서 사용하기 위해 개발자는 애플리케이션 매니페스트 파일에 적절한 퍼미션을 기술 해야한다. 그러나 개발자가 필요 이상의 퍼미션을 기술하는 경우 필요 이상의 특권 (privilege)이 애플리케이션에 부여된다. 이렇게 필요 이상의 특권 가진 애플리케이션은 1)과도한 퍼미션을 요구를 하기 때문에 사용자에 의해 설치가 거부 될 수 있

으며 반대로 2) 사용자가 과도한 퍼미션을 허가하는 것을 익숙하게 만들 수 있다. 또한 3) 애플리케이션 취약점에 따른 잠재적인 손상 (damage)을 증가 시킬 수 있다.

Stowaway[17]는 애플리케이션이 사용하는 API를 분석하여 애플리케이션이 필요 이상의 퍼미션을 요구하는지 분석한다. 또한 Stowaway는 안드로이드 2.2 버전의 퍼미션 맵 (map)을 제공하고 있다.

3.2.3 ComDroid

ComDroid는 인텐트를 이용한 애플리케이션 통신의 취약점을 탐지하는 툴 (tool)이다[16]. Comdroid는 하나의 애플리케이션을 분석하여 해당 애플리케이션의 취약한 인터페이스와 보안 문제 발생할 수 있는 인텐트/브로드캐스트 전송을 파악한다. 예를 들어 ComDroid는 애플리케이션 개발자가 민감한 데이터를 공개된 브로드캐스트를 이용하여 전달하는 것을 경고한다. 왜냐하면 악성 애플리케이션에서 브로드캐스트 리시버를 등록하여 해당 브로드캐스트 메시지를 도청할 수 있기 때문이다. 이렇게 ComDroid를 이용하여 취약한 애플리케이션 통신을 분석할 수 있지만 하나의 애플리케이션만을 분석하기 때문에 권한 상승 (privilege escalation) 공격을 탐지 할 수 없다.

3.2.4 Woodpecker

Grace는 Woodpecker라는 툴을 개발하여 안드로이드 시스템 애플리케이션을 대상으로 취약한 인터페이스를 분석하였다[13]. 삼성, HTC, 모토롤라에서 제작한 안드로이드 폰에서 추출한 시스템 애플리케이션을 대상으로 분석해 보았을 때 취약한 인터페이스가 존재함을 확인하였고 이를 통해 안드로이드 퍼미션 시스템이 깨질 수 있음을 실험을 통해 밝혔다. 이러한 취약점이 발생한 이유는 각 제조사 별로 순정 안드로이드 커스터마이즈 (customize) 하는 과정에 취약한 인터페이스를 가진 시스템 애플리케이션을 탑재하여 발생한 문제였다.

3.2.5 TaintDroid

TaintDroid[20]는 실시간으로 폰 내부의 민감한 정보를 외부로 유출시키는 애플리케이션을 감시한다. 실시간으로 데이터의 흐름을 추적하기 위해 동적 오염 분석 (Dynamic Taint Analysis)을 수행한다. 그러나 모든 데이터의 흐름을 다루기에는 부하 (overhead)가 커서 스마트폰의 에너지 소모와 속도에 영향을 줄 수 있기 때문에 폰 상태와 위치 정보 같은 일부의 민감한 데이터만을 추적한다. 또한 TaintDroid는 민감한 데이터의 흐름을 추적하고 외부 유출을 탐지할 수는 있지만 차단을 할 수 없는 한계가 있다.

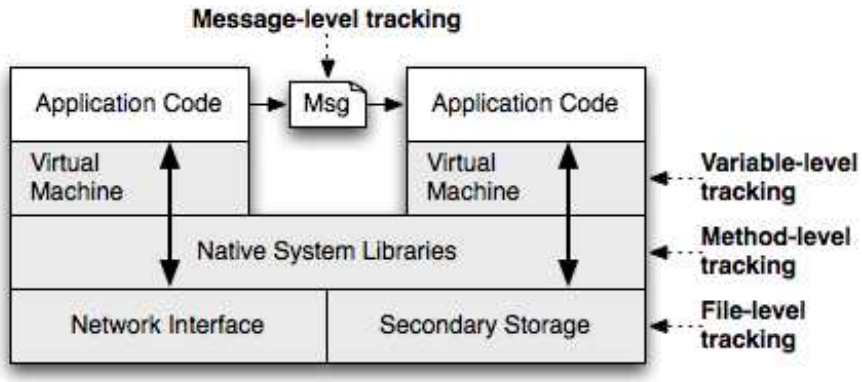

그림 3.3: TaintDroid 구조

3.3 애플리케이션 수준 보안

3.3.1 Dr. Droid and Mr. Hide, Aurasium

Jeon과 Xu의 연구는 기존의 연구들과는 다르게 안드로이드 운영체제의 수정과 루팅 없이 애플리케이션에 보안 정책을 강제할 수 있는 실용적인 기법을 제안하고 있다[4,7]. 이 연구들은 기존 연구인 Apex가 제안했던 fine-grain 정책과 사용 시간 퍼미션 시스템의 환영 (illusion)을 구현하고 있다. 이러한 기능을 제공하기 위해 이 연구들에서는 IRM (Inline Reference Monitor)을 각 애플리케이션 실행파일에 삽입한다. 삽입된 IRM (Inline Reference Monitor)은 설정된 보안 정책에 따라 애플리케이션의 동작을 감시하고 통제한다. 그러나 이러한 기법은 애플리케

이션을 재패키지 할 때 임의의 키를 생성하여 서명을 하기 때문에 애플리케이션 간의 신뢰관계 (Trust Relationship)를 훼손시킬 수 있다. 또한 이들의 서비스를 사용자들이 손쉽게 사용하기 위한 환경을 구축되어있지 않다.

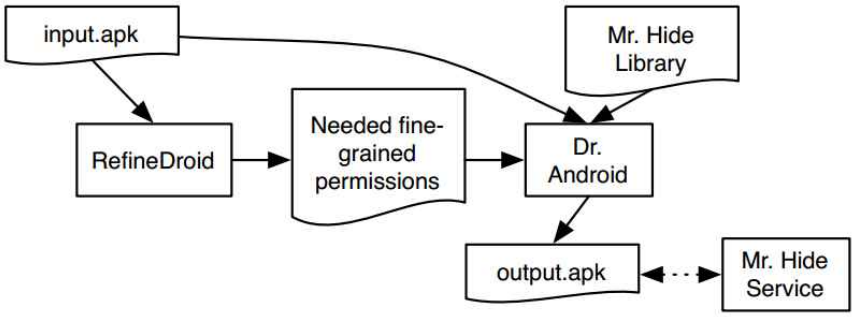

그림 3.4: Dr. Android and Mr. Hide 구조

3.3.2 AppGuard

AppGuard는 3.3.1절에 설명한 기법들을 사용자들이 사용해볼 수 있도록 만들어진 안드로이드 애플리케이션이다[3]. 사용자들은 AppGuard를 통해 사용자들은 설치된 애플리케이션에 보안 정책을 적용해 볼 수 있다. 그러나 안드로이드 애플리케이션의 업데이트가 발생할 때마다 새롭게 사용자가 보안 정책을 적용해주어야 하며 안드로이드 앱 마켓에서 보호 (protect)로 설정된 애플리케이션에 대해서는 보안 정책을 적용할 수 없다는 단점이 있다.

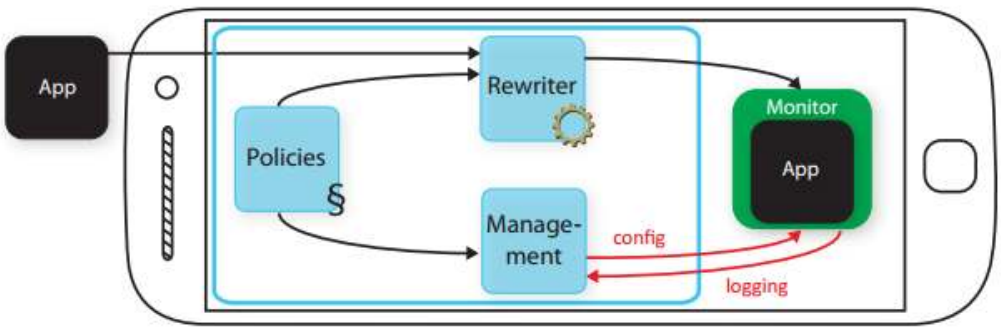

그림 3.5: AppGuard 구조

제 4 장 안드로이드 보안 강화를 위한 새로운 앱 마켓 시스템

4.1 기존 앱 마켓 시스템의 한계

현재의 앱 마켓 시스템은 단순히 사용자에게 애플리케이션을 검색하고 다운로드하는 기능만을 제공하고 있으며 사용자의 디바이스에 설치된 애플리케이션의 실행과정에는 전혀 개입하지 않는다. 따라서 일단 악의적인 목적을 가진 애플리케이션이 앱 마켓의 검수과정(review process)을 우회한다면 사용자의 디바이스에 성공적으로 설치될 수 있으며 그 결과로 시스템의 무결성은 무너지고 사용자의 개인정보는 외부로 누출 될 가능성이 있다.

하루가 다르게 수많은 애플리케이션이 앱 마켓에 등록되는 현실에서 모든 애플리케이션을 일일이 검수하는 것은 시간과 비용이 많이 드는 일이며 거의 불가능에 가깝다. 현재 안드로이드 마켓에서는 구글 바운서(google bouncer)라는 안티바이러스 (anti-virus) 시스템 통해 자동화된 애플리케이션 검수를 진행한다. 이러한 자동화된 검수는 2011년도 1~2분기 사이에 약 40%정도의 악성 애플리케이션을 탐지하는데 기여하였다[30]. 그러나 여전히 과반수의 악성 애플리케이션은 탐지할 수 없었으며 더 큰 문제는 구글 바운서를 우회할 수 있는 방법이 Black Hat과 몇몇의 보안 전문가들에 의해서 공개되어졌다는 점이다[29]. 따라서 악의적인 의도를 가진 개발자들은 이러한 정보에 기초하여 손쉽게 앱 마켓의 검수과정을 우회할 수 있는 악성 애플리케이션을 제작할 수 있다.

4.2 안드로이드 보안 시스템의 한계

안드로이드 보안 문제는 주로 coarse-grain 퍼미션, all-or-nothing 결정, 보안정책 실패, 개발자 보안의식 결여, 그리고 오픈소스(opensource) 정책과 관련되어 있다.

4.2.1 coarse-grain 퍼미션

하나의 퍼미션은 특정 시스템 API들의 집합을 나타낸다. 다루는 집합의 크기에 따라서 그 집합의 크기가 작을 경우 fine-grain 퍼미션이라 하고 반대로 클 경우 coarse-grain 퍼미션이라고 한다. fine-grain 퍼미션은 다루는 API 집합의 크기가 작아 그것을 사용하는 개발자의 부담이 크지만 필요 이상의 기능을 애플리케이션에 허가할 가능성이 적다는 장점을 갖는다. 반면에 coarse-grain 퍼미션의 경우에는 개발자의 부담은 적지만 필요 이상의 기능을 애플리케이션에 부여할 가능성이 높다. 특히 민감한 (sensitive) 정보의 접근을 다루는 퍼미션이 coarse-grain 하다면 해당 퍼미션을 사용하는 애플리케이션은 필요 이상의 권한을 가지게 되며 사용자는 해당 퍼미션을 사용하는 애플리케이션의 정확한 의도를 파악하기 어렵게 된다. 예를 들어 안드로이드 애플리케이션은 READ_PHONE_STATE 퍼미션만 할당 받으면 기본적인 장치 정보 뿐만 아니라 민감한 사용자 정보인 폰 번호, IMEI (International Mobile Equipment Identity) 정보 등과 같은 중요한 폰 정보에 접근할 수 있다. 따라서 coarse-grain 퍼미션은 사용자의 판단을 흐리게 만들기 때문에 안드로이드 퍼미션 시스템에 대한 이해가 부족한 대부분의 사용자들은 위험성을 파악하지 못하고 단지 원하는 기능을 사용하기 위해 설치를 허가하게 된다.

4.2.2 All-or-nothing 결정

설치시간 퍼미션 시스템의 all-or-nothing 결정 특성으로 인하여 사용자는 선택적으로 퍼미션을 허가할 수 없을 뿐만 아니라 오직 애플리케이션이 요구하는 퍼미션을 모두 허가하여 설치가 되도록 하거나 설치를 거부할 수밖에 없다. 또한 실행 중에 할당된 퍼미션을 동적으로 회수하는 것이 불가능하며 오직 애플리케이션의 삭제를 통해서만 권한을 회수 할 수 있다. 따라서 악성 애플리케이션은 설치과정만 통과하면 아무런 제약 없이 동작할 수 있게 된다.

4.2.3 콤포넌트 개방 정책

안드로이드의 모든 애플리케이션은 Activity, Service, Broadcast Receiver, 그리고 Content Provider 이렇게 4개의 컴포넌트로 구성되며 각 컴포넌트에 대한 외부 접근은 기본적으로 개방되어 있다(안드로이드 4.2버전 전까지). 따라서 이러한 정책 수립의 실패는 애플리케이션 컴포넌트의 접근을 무분별하게 허용하도록 만들었기 때문에 이러한 헛점을 이용한 공격의 위험성은 여러 연구들을 통해 다루어졌다[6,13,16,19]. 따라서 이러한 안드로이드 정책 수립의 실패는 심각한 보안 취약점을 야기할 수 있다. (이 문제를 피해가기 위해 개발자는 매니페스트 파일안에서 해당 컴포넌트를 exported 속성을 false로 하거나 새로운 응용수준 퍼미션을 정의하여 외부 접근을 통제해야한다.)

4.2.4 개발자 보안의식 결여

정책 수립 실패와는 대조적으로 개발자들의 보안 개념에 대한 이해 부족은 보안이 취약한 애플리케이션이 양산되는데 기여했다. 이러한 문제는 비단 간단한 애플리케이션 뿐만 아니라 민감한 사용자 정보 다루는 애플리케이션에서도 발생하고 있다. 대표적인 예가 지난 2011년 4월 15일에 보고된 Skype 애플리케이션 취약점이다[25]. 보고된 취약점의 원인은 개발자가 사용자의 민감한 정보를 저장하는 파일의 퍼미션을 MODE_WORLD_READABLE/WRITABLE 모드로 설정해서 발생한 문제였다. 파일을 권한을 MODE_WORLD_READABLE/WRITABLE로 설정하게 되면 시스템에 설치된 모든 애플리케이션의 접근을 허용하게 된다. 따라서 이를 통해 사용자명, 전화번호, 생년월일, 주소, 계정잔고, 채팅로그 등에 접근할 수 있는 문제가 발생하였다.

4.2.5 오픈소스 정책

안드로이드의 오프소스(opensource) 정책은 여러 제조업체들이 안드로이드 운영체제를 원하는 대로 고쳐 쓸 수 있는 기회를 주었고 이 같은 결정은 안드로이드 플랫폼이 스마트폰 시장에서 점유율을 높이는데 큰

기여를 하였다. 그러나 공격자들 역시 공개된 안드로이드 소스코드를 통해 운영체제의 취약점을 찾아내기가 더 쉬워졌다.

4.3 새로운 앱 마켓 시스템

본 논문에서 제안하는 새로운 앱 마켓 시스템은 앞서 설명한 기존 마켓 시스템이 지닌 한계를 보완할 뿐만 아니라 안드로이드 시스템에 본질적으로 내재된 보안 문제를 해결한다. 또한 제안하는 앱 마켓 시스템은 기존에 제안된 연구들과는 다르게 안드로이드 커널 또는 애플리케이션 프레임워크의 수정을 요구하지 않기 때문에 순정 안드로이드 OS를 사용하는 모든 디바이스에 바로 적용될 수 있어 매우 실용적이다.

제안하는 앱 마켓 시스템은 4.1절과 4.2절에서 설명한 안드로이드 보안 시스템의 한계를 극복하기 위해 아래의 6가지 기능을 제공하는 것을 목표로 한다.

1. 애플리케이션 실행 중 동적으로 권한 회수
2. 보안 정책
   - Fine-grain 퍼미션 정책
   - 동적 실행 파일(.dex) 정책
   - Java Reflection API 정책
3. 보안 취약점 패치
   - 운영체제 보안 취약점 보완
   - Third-party 애플리케이션 취약점 보완
4. 개발자 키 관리
5. 동적 실행파일 인증
6. 공개된 콘텐트 프로바이더 인터페이스를 가진 애플리케이션 등록 차단

본 논문에서는 위의 기능을 제공하기 위해 안드로이드 애플리케이션 실행파일(.dex)에 IRM(Inline Reference Monitor)를 삽입한다. IRM의 삽입은 안드로이드 시스템에 Install-Time 퍼미션 시스템과 Time-of-Use

퍼미션 시스템이 공존하는 듯한 일루전 (illusion)을 사용자에게 제공한다. 이를 통해 모든 민감한 자원에 접근하는 애플리케이션의 행위는 사용자에게 보고되며 악의적인 행위로 의심될 시에 사용자는 애플리케이션의 권한을 동적으로 회수할 수 있게 된다. 또한 IRM은 미리 정의된 보안 정책 (policy)에 따라 애플리케이션의 동작을 제어하거나 이미 존재하는 coarse-grain 퍼미션에 fine-grain 퍼미션 정책을 적용할 수 있게 한다. 예를 들어 안드로이드에서 대표적인 coarse-grain 퍼미션인 INTERNET 퍼미션을 획득한 애플리케이션은 아무런 제한 없이 모든 인터넷 도메인에 접속 할 수 있다. 그러나 본 논문에서 제안된 시스템 안에서는 애플리케이션이 INTERNET 퍼미션을 획득하더라도 특정 도메인만을 접속하도록 제한할 수 있다.

Fine-grain 정책 뿐만 아니라 동적 실행파일을 다루기 위한 보안 정책이 설정되어있다. 제안된 시스템에서 동적 실행파일을 수행하기 위해서는 반드시 동적 실행파일은 마켓 시스템으로 부터 인증을 받아야 한다. 인증 받은 동적 실행파일에는 IRM이 탑재 되기 때문에 수행이 되더라도 모든 동작이 모니터링 될 수 있다. 그러나 인증 받지 않은 동적 실행파일의 동작은 감시 할 수 없기 때문에 동적 실행파일이 적재 될 경우 심각한 보안 헛점이 될 수 있다. 따라서 반드시 인증된 동적 실행파일만이 적재되어 수행될 수 있다.

자바 reflection API를 이용한 함수 호출을 막기 위한 정책도 존재한다. 자바 reflection API를 이용하여 함수를 호출할 경우 정상적인 함수의 호출과 다르기 때문에 기존 static analysis 툴을 우회하여 sensitive API를 호출할 가능성이 높다. 따라서 이를 다루기 위한 보안 정책도 존재한다.

IRM 삽입과 더불어 운영체제와 애플리케이션 취약점을 보완하기 위해 코드 패칭(Code Patching) 기술이 사용된다. 코드 패칭 기술은 바이너리 수준에서 동작하며 애플리케이션에 존재하는 취약점을 보완하거나 운영체제의 취약점에 이용하려는 시도를 방지하는 보안패치를 모든 애플리케이션에 적용하기 위해 사용된다. 보통 안드로이드 보안 패치의 경우

제조사를 통해 안드로이드 디바이스에 적용되기까지 수 개월 이상 걸리며 또한 사후지원이 종료된 안드로이드 디바이스에 경우 보안패치가 적용되지 않기 때문에 본 논문에서 제안하는 코드 패칭을 이용한 보안패치의 적용은 상당히 실용적이며 큰 의미가 있다.

4.3.1 설계

제안하는 앱 마켓 시스템의 핵심 설계 목표는 정확성(accuracy), 확장성(scalability) 그리고 효율성(efficiency)이다. 안드로이드 시스템 안에서 사용시간 퍼미션 시스템의 일루전 (illusion)을 사용자에게 제공하기 위해 모든 애플리케이션에 IRM(Inline Reference Monitor)를 삽입한다. 기존 연구들에서는 smali/baksmali를 사용하여 이러한 작업을 수행하였다 [3,4,7]. 그러나 하루에도 수 천개의 새로운 애플리케이션이 등록되는 현실에서 자바 언어로 작성된 안드로이드 어셈블러(assembler)/디스어셈블러(disassembler)인 smali/baksmali를 이용한 IRM 삽입은 오버헤드가 크며 또한 smali/baksmali의 버그는 일부 애플리케이션의 IRM의 삽입을 어렵게 만든다. 결과적으로 smali/baksmali를 이용한 IRM의 삽입은 설계목표인 정확성, 확장성, 그리고 효율성을 만족 시킬 수 없다. 따라서 이를 만족시키기 위해 안드로이드 실행파일(.dex) 재작성 프로그램인 Android DEX Rewriter를 개발 하였다. Android DEX Rewriter는 앞서 설명한 3가지 설계목표를 만족시키지 위해 C언어로 작성되었을 뿐만 아니라 바이너리 수준에서 IRM을 삽입하기 때문에 smali/baksmali의 역컴파일/컴파일과 같은 변환 작업이 필요 없어 정확하고 더 빠른 속도로 실행파일(.dex)을 재작성할 수 있다.

4.3.2 시스템 구조

제안된 앱 마켓 시스템은 크게 마켓 서버와 클라이언트 애플리케이션으로 구성되어 있다. 마켓 서버는 기존 마켓서버와 동일하게 애플리케이션 설치파일의 저장소를 유지할 뿐만 아니라 안드로이드 보안 강화를 위해 각 애플리케이션에 IRM(Inline Reference Monitor)를 삽입하고 코드 패치(Code Patch)를 적용하기 위한 ADR(Android DEX Rewriter)를 탑재하고 있다. 또한 애플리케이션을 재패키지(repackage)할 때 동일한 개발자 키(key)로 서명하기 위한 키 저장소를 따로 유지하고 있다. 마켓 서버 시스템 아키텍쳐는 아래와 같다.

4.3.2.1 마켓 서버 시스템의 구조

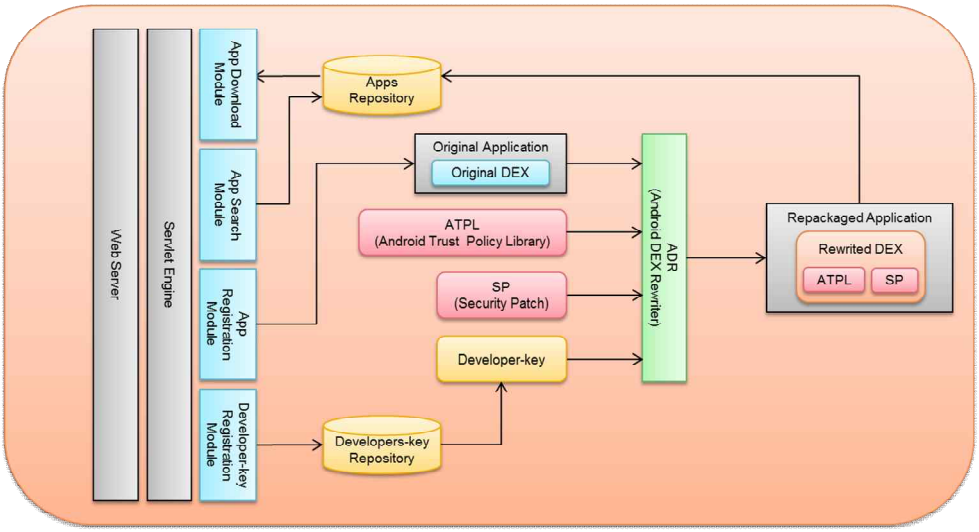

그림 4.1: 마켓 서버 구조

제안하는 마켓 시스템 서버는 다운로드 모듈, 검색 모듈, 앱 등록 모듈, 그리고 개발자 키 등록 모듈로 구성되어있다. 다운로드 모듈, 검색 모듈은 기존 앱 마켓 시스템의 기능과 동일한 역할을 하는 반면에 앱 등록 모듈의 기능은 기존과 다르며 개발자 키 등록 모듈은 기존에 없던 기능이다.

(1) 애플리케이션 등록 모듈

애플리케이션 등록 모듈에서는 먼저 콘텐트 프로바이더 인터페이스의 공개/비공개 여부를 검사하여 공개된 콘텐트 프로바이더를 가진 애플리케이션의 등록을 거부하고 있으며 반드시 새로운 퍼미션을 정의하여 해당 콘텐트 프로바이더에 대한 접근을 보호하거나 비공개 (exported=false)로 정의된 콘텐트 프로바이더의 등록 만을 허용한다. 그 이유는 기본적으로 외부의 접근이 공개된 개방된 콤포넌트는 악성 애플리케이션의 표적이 될 가능성이 높기 때문이다. 특히, 각 애플리케이션의 데이터베이스로 사용되는 콘텐트 프로바이더의 경우 민감한 사용자 정보나 인증 토큰, 그리고 애플리케이션 설정 등을 저장하는 용도로 사용되기 때문에 인증토큰 (authentication token)과 같은 저장된 민감한 정보가 외부로 유출될 수 있으며 또한 저장된 애플리케이션 설정을 변경하여 애플리케이션의 오작동 (side-effect)을 야기할 수도 있다[ContentScope].

그 다음 과정으로 새롭게 등록되는 애플리케이션 중에서 위험 (dangerous) 퍼미션을 하나 이상 필요로 하는 애플리케이션을 대상으로 리패키지 작업을 수행한다. 그 이유는 위험 퍼미션을 보유하지 않은 애플리케이션은 시스템 무결성에 영향을 주는 퍼미션을 요구하지 않기 때문에 정상 애플리케이션으로 분류할 수 있기 때문이며 또한 이러한 애플리케이션들을 리패키지 작업에서 제외하여 리패키지 작업에 따른 부하를 줄일 수 있기 때문이다.

위험 퍼미션을 하나 이상 요구하는 애플리케이션은 시스템 무결성에 영향을 주거나 폰 번호, IMEI, 그리고 GPS와 같은 중요한 정보에 접근할 수 있다. 따라서 이러한 애플리케이션들의 동작을 감시하고 통제하기 위해서 Android Trust Policy Library (ATPL)를 삽입한다. ATPL에는 IRM (Inline Reference Monitor)과 관련 라이브러리들이 포함되어있다. IRM은 민감한 (sensitive) 정보에 접근할 수 있는 API의 호출을 감시하며 삽입된 정책에 따라 이를 통제한다.

또한 보안 취약점을 이용하여 시스템의 무결성에 영향을 줄 수 있는 공격을 수행할 가능성이 있기 때문에 보안 패치 (security patch)도 함께

애플리케이션에 적용된다. 안드로이드 시스템 보안 패치는 제조업체를 통해 실제 장비에 적용되기까지 많은 시간이 걸리며 사후 서비스 지원 기간이 지난 경우에는 보안 패치가 적용되지 못할 가능성이 높기 때문에 그 취약점의 심각한 문제가 될 수 있다. 또한 특정 애플리케이션의 보안 취약점에 대한 패치가 개발자의 늦장 대응 혹은 연락 두절로 문제가 될 수 있다. 따라서 보안 패치는 안드로이드 시스템 자체의 보안 취약점 뿐만 아니라 특정 애플리케이션의 보안 취약점을 함께 다룬다.

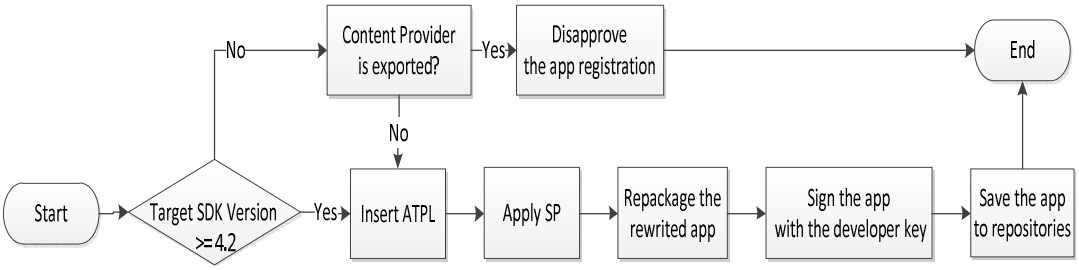

그림 4.2: 애플리케이션 등록 모듈 동작 과정

(2) Android DEX Rewriter (ADR)

안드로이드 애플리케이션에 IRM를 삽입하고 보안 패치를 적용하기 위해선 안드로이드 실행 파일(.dex)을 수정하기 위한 툴이 필요하다. 기존 연구에서는 이러한 작업을 수행하기 위해 Apktool[26] 이라는 오픈소스 DEX 어셈블러/디스어셈블러를 사용하여 제작하였다. 그러나 Apktool의 처리 속도가 느리기 때문에 하루에도 수천 개의 애플리케이션이 등록되는 환경에서 사용하기에는 문제가 있으며 기존 연구에 따르면 일부 애플리케이션을 재패키징 하는데 문제가 발생함이 확인되었다[4]. 따라서 이러한 Apktool의 확장성 (scalability) 문제와 정확성 (accuracy) 문제를 다루기 위해 Android DEX Rewriter (ADR)라는 새로운 툴을 개발하였다. ADR은 자바 언어로 작성된 Apktool과는 다르게 C언어로 작성되었으며 바이너리 수준에서 DEX 파일을 수정하기 때문에 속도가 빠르며 정확하게 DEX 파일을 수정한다.

(3) 동적 실행 파일 인증

동적 실행 파일은 애플리케이션을 수행하기 위해 필요한 기능의 일부이지만 초기 설치 파일의 용량이나 메모리 사용량을 줄이기 위해 포함되지 않는다. 이러한 동적 실행 파일은 보통 네트워크를 통해 다운로드 되며 애플리케이션 실행 중에 동적으로 메모리에 적재되어 수행된다. 이렇게 동적 실행 파일은 제안된 마켓 시스템을 우회하여 설치되고 수행될 수 있기 때문에 공격자들이 이 점을 악용하여 악의적인 코드를 따로 모아서 동적 실행파일로 구성한다면 제안된 마켓 시스템을 통한 보안을 무너뜨릴 수 있다. 따라서 제안된 마켓 시스템에서는 반드시 인증 된 동적 실행 파일만을 수행하게 한다.

동적 실행 파일에 대한 인증은 애플리케이션 등록 모듈을 통해 수행되며 애플리케이션 등록 때와 마찬가지로 IRM이 탑재되며 보안 패치가 적용된다. 그리고 추가적으로 인증된 동적 실행 파일과 인증되지 않은 동적 실행 파일을 구분하기 위해 인증된 동적 실행 파일에는 특별한 식별자를 삽입한다.

(4) 개발자 키 등록 모듈

기존 애플리케이션에 수정을 가하게 될 경우 개발자의 서명이 훼손되어 수정 후 새롭게 패키징을 할 때 마다 개발자의 키로 새롭게 서명을 해주어야 한다. 기존 연구들에서는 원 개발자의 키를 가지고 있지 못하기 때문에 임의의 키를 생성하여 서명을 하였다[3,4,7]. 그러나 이러한 임의의 키를 이용한 애플리케이션 서명은 원 개발자가 설정한 애플리케이션 간의 신뢰관계를 훼손할 가능성이 높다. 애플리케이션 간의 신뢰관계가 깨질 경우 애플리케이션의 오작동 (side-effect)을 야기할 수 있다. 이러한 문제를 해결하기 위해 제안된 시스템에서는 개발자가 애플리케이션 등록 시 의무적으로 개발자의 키를 마켓 서버 시스템에 등록하게 한다. 이렇게 함으로써 애플리케이션이 재패키지 되더라도 원 개발자의 키로 서명할 수 있게 된다.

4.3.2.2 마켓 클라이언트 시스템의 구조

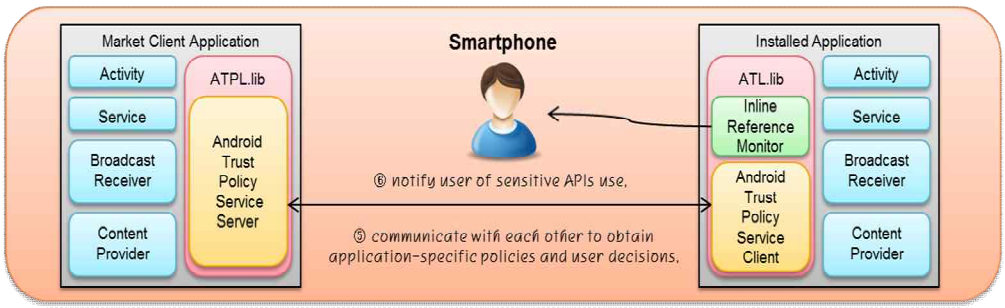

그림 4.3: 마켓 앱 클라이언트 구조

기존 마켓 클라이언트 애플리케이션은 단순히 애플리케이션의 검색, 다운로드, 및 설치 기능 만을 제공한다. 반면에 제안된 시스템의 마켓 클라이언트 애플리케이션은 설치된 각 애플리케이션의 보안 정책과 사용자 결정을 추가적으로 관리한다.

사용자의 결정은 각 애플리케이션 안에 존재하는 Android Trust Policy Service (ATPS)를 통해 마켓 클라이언트 애플리케이션에 전달되며 전달된 정보는 마켓 클라이언트 애플리케이션 내부 저장소에 보관된다. 설치된 모든 애플리케이션은 Android Trust Policy Service (ATPS)를 통해 저장된 결정 내용을 재 사용할 수 있으며 또한 사용자가 직접 저장된 결정 내용을 마켓 클라이언트 애플리케이션을 통해 변경할 수도 있다.

제안된 시스템에서 사용자 폰에 설치된 모든 애플리케이션은 ATPL (Android Trust Policy Library)를 탑재하고 있으며 특히 ATPL에 포함된 IRM은 미리 설정된 정책에 따라서 애플리케이션 동작을 감시한다. 애플리케이션이 GPS, 네트워크, 그리고 IMEI 등과 같은 민감한 자원이나 정보에 접근할 때마다 IRM은 사용자에게 팝업을 띄워 해당 접근을 허용할 것인지 아니면 애플리케이션을 종료 시킬 것인지 결정하게 한다. 그리고 경우에 따라서는 사용자가 임의의 값을 반환할 수 있게 한다. 예를 들어 애플리케이션이 TelephonyManager.getLine1Number() 메소드를 호출하여 폰 번호에 접근하려 할 때 IRM은 팝업을 띄워 이러한 접근을 사용자에게 알린다. 사용자가 허용버튼을 클릭 할 경우

TelephonyManager.getLine1Number() 메소드의 호출은 정상적으로 이루어지며 정상적인 폰 번호가 해당 애플리케이션에 전달된다. 반대로 쉐도우 버튼을 클릭하면 000-0000-0000과 같은 임의의 폰 번호가 애플리케이션에 전달되어 이러한 자료의 수집과 사용을 무의미하게 만들 수 있다.

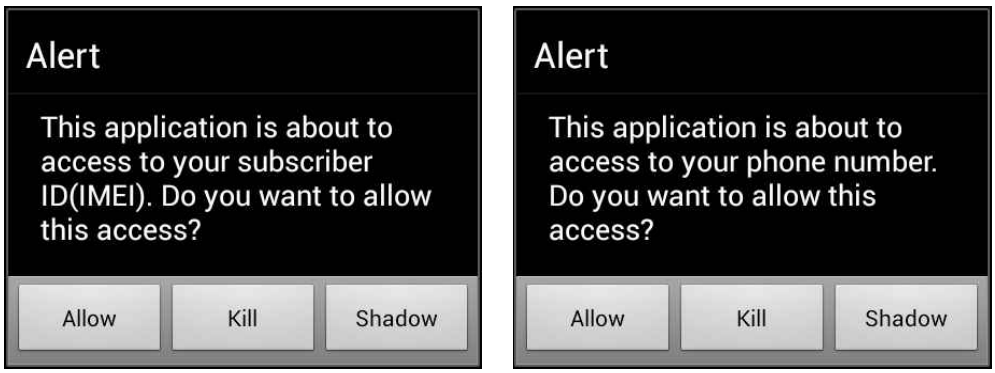

그림 4.4: 알림 팝업

4.3.3 보안 정책

(1) Fine-grain 보안 정책

안드로이드 퍼미션 중 INTERNET, READ_PHONE_STATE 등과 같이 민감한 정보에 접근할 수 있는 퍼미션이 coarse-grain하기 때문에 설치 시간 시스템의 장점인 검수 분류 (review triaging)을 제대로 활용할 수 가 없게 되어 악성 애플리케이션의 설치가 쉽게 이루어 지고 있다. 제안된 시스템에서는 이러한 coarse-grain 퍼미션이 다루는 API의 사용을 감시하고 통제하기 위해 기본적으로 IRM (Inline Reference Monitor)에 fine-grain 정책을 적용하고 있다. 이렇게 함으로써 사용자는 실행 중에 원치 않는 API의 사용을 차단할 수 있게 된다.

(2) 동적 실행코드 차단 보안 정책

동적 실행코드는 사용하는 애플리케이션은 기존 애플리케이션 기반 보호 기법[3,4,7]을 우회할 수 있다. 따라서 이러한 동적 실행코드를 이용한 공격을 예방하기 위해 제안된 마켓 시스템에서는 동적 실행코드를 인

증하기 위한 시스템을 따로 운영하고 있다. 인증된 동적 실행코드에는 일반 애플리케이션과 동일하게 IRM과 보안 패치가 적용되며 추가적으로 인증된 동적 실행코드와 그렇지 않은 동적 실행코드를 구분하기 위한 특별한 식별자를 삽입된다. 인증된 동적 실행코드는 정상적으로 실행되며 삽입된 IRM을 통해 그 동작이 감시되고 통제된다. 인증되지 않은 동적 실행코드는 실행이 차단된다.

(3) Java Reflection API 보안 정책

리플렉션은 실행 중에 객체의 구조와 행위를 검사하거나 수정할 수 있는 능력을 말한다. 보통 리플렉션은 고수준의 가상머신 프로그래밍 언어에서 사용되어진다. 안드로이드에서 사용 중인 자바 언어 또한 대표적인 고수준의 가상머신 프로그래밍 언어이다. 따라서 자바에서는 리플렉션 API를 제공하고 있으며 이를 통해 애플리케이션은 실행 중에 객체의 구조와 행위를 검사하고 수정할 수 있게 된다. 이러한 특성을 이용하여 악성 애플리케이션 개발자는 IRM의 감시를 피해 민감한 정보에 접근하는 API를 호출하기 위해 리플렉션 API를 사용할 수 있다. 그림 X는 일반적인 API 호출과 리플렉션을 이용한 API호출을 나타내고 있다. 따라서 객체를 생성하거나 메소드를 호출하는데 사용되는 리플렉션 API를 감시하여 민감한 API를 우회적으로 호출하려는 행위를 차단한다.

리플렉션을 사용하지 않은 함수 호출
new Foo().hello();
리플렉션을 사용한 함수 호출
Class<?> clazz = Class.forName("Foo"); clazz.getMethod("hello").invoke(clazz.newInstance());

그림 4.5: 일반적인 메소드 호출과 리플렉션을 이용한 메소드 호출 비교

(4) 보안 취약점 패치 정책

운영체제 혹은 애플리케이션의 보안 취약점이 알려지고 그에 대응하는 패치가 적용되기까지 많은 시간이 걸리는 경향이 있다. 특히, 운영체제의 경우에는 더욱 그러하다. 안드로이드 팀에서 만든 패치가 제조사를 통해 사용자의 스마트폰에 적용되기까지 많은 시간이 걸리는 것이 일반적이며 사후지원 기간이 지난 스마트폰의 경우에는 보안 취약점이 그대로 방치될 가능성이 높다. 반면에 애플리케이션의 경우 개발자의 실수나 보안 의식의 부족으로 인하여 애플리케이션에 취약점이 발생하는 경우가 많다. 그리고 취약점이 존재 하더라도 잘 알려지지 않을 가능성이 높으며 알려지더라도 개발자와 연락을 취하기 어려운 경우가 존재한다.

이러한 보안 취약점에 따른 문제의 파급을 최소화하기 위한 보안 패치 정책을 포함하는 IRM을 각 애플리케이션에 적용한다. 경우에 따라 추가적으로 코드 패치를 적용한다. 예를 들어 파일의 권한을 MODE_WORLD_READABLE/WRITABLE로 설정하는 경우 외부의 애플리케이션에서 아무런 제약없이 해당 파일에 접근할 수 있다. 만약 이러한 파일에 민감한 사용자 정보 또는 인증토큰 (authentication token) 등을 보관하고 있다면 심각한 보안 문제를 야기할 수 있다. 실례로 2011년 4월 15일에 보고된 Skype 애플리케이션 취약점이 있다[]. 보고된 취약점의 원인은 개발자가 사용자의 민감한 정보를 저장하는 파일의 퍼미션을 MODE_WORLD_READABLE/WRITABLE 모드로 설정해서 발생한 문제였다. 이를 통해 사용자명, 전화번호, 생년월일, 주소, 계정잔고, 채팅로그 등에 접근할 수 있는 문제가 발생하였다.

제 5 장 실험 및 평가

본 절에서는 제안된 기법에 대한 실험 결과를 보인다. 실험을 통해 본 논문에서 제안한 ADR (Android DEX Rewriter)과 Apktool의 성능을 비교하여 ADR이 설계목표인 확장성과 효율성을 만족하는지 검증하였다.

나머지 실험은 원본 애플리케이션과 재패키지 된 애플리케이션 사이의 오버헤드를 측정하는데 중점을 두었으며 측정된 오버헤드는 크게 성능 오버헤드, 사이즈 오버헤드, 그리고 에너지 오버헤드로 나눌 수 있다.

5.1 실험환경

본 논문의 실험은 PC와 구글 레퍼런스 폰인 Nexus One 상에서 진행되었다. 실험에 사용된 각 장치의 사양은 아래와 같다.

프로세서	AMD E-450 APU with Radeon(tm) HD Graphics 1.65 GHz
메모리	Samsung DDR3 4 GB PC3-12800
디스크	Samsung 840 Series 250GB (SSD)
운영체제	Crunchbang Linux (Debian)
커널버전	2.6.32

표 5.1: 테스트 PC 사양

프로세서	QUALCOMM QSD8250 1 GHz
내부 메모리	512MB RAM, 512MB ROM
외부 저장소	microSDHC 16GB
Display	OLED display (AMOLED)
WLAN	Wi-Fi 802.11 a/b/g
운영체제	Android 4.2.2 (jellybean)
커널버전	2.6.38.8-evervolve-perdo-jellybean

표 5.2: Nexus One 사양

5.2 ADR과 Apktool(smali/baksmali) 성능 비교

7.8MB~46MB사이의 크기를 갖는 25개의 APK파일을 대상으로 본 논문에서 소개한 ADR (Android DEX Rewriter)과 오픈소스 툴인 Apktool 각각을 이용하여 APK 파일을 재패키지 하는데 걸리는 시간을 측정하였다.

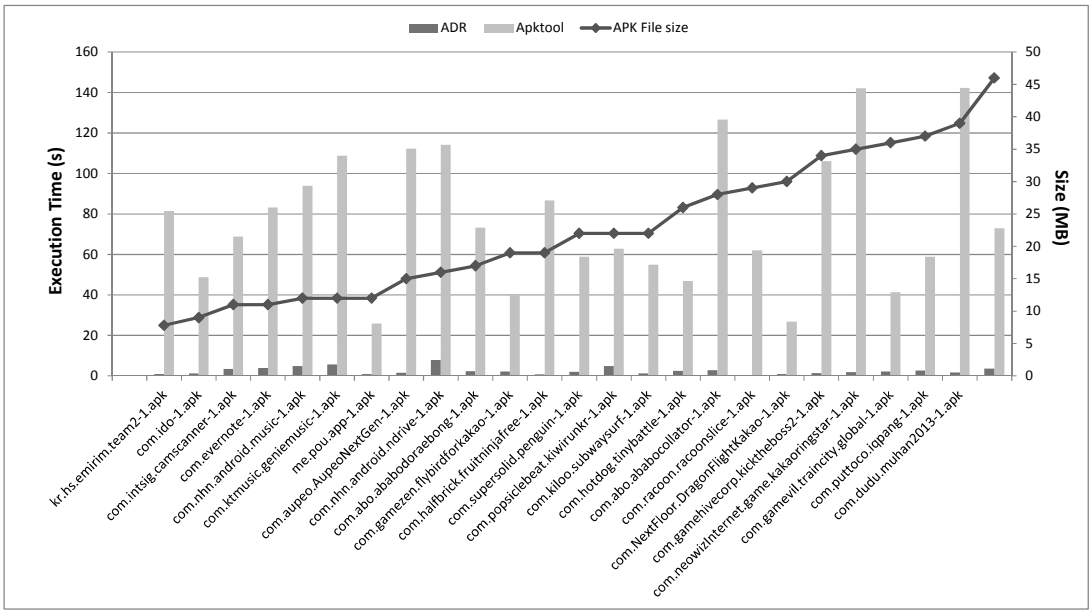

그림 5.1: ADR (Android DEX Rewriter)와 Apktool 성능 비교

실험결과에서 나타나듯이 ADR은 Apktool와 비교하여 처리속도가 최대 190배에서 최소 13배 빨랐으며 평균적으로 46배 빨랐다. 또한 ADR은 APK 파일의 크기에 따른 처리시간의 변동이 1초~6초 사이를 유지하고 있는 반면에 Apktool은 처리시간이 26초~143초 사이로 성능의 변동 폭이 큰 것을 알 수 있다.

위 실험 결과를 통해서 Apktool은 대량의 애플리케이션을 처리해야하는 마켓서버에서 사용하기에는 규모 확장성과 효율성면에서 부족한 것을 알 수 있다. 반면에 ADR은 규모 확장성과 효율성을 모두 만족 시키고 있다. 따라서 기존 마켓 시스템에서 본 논문에서 제안한 시스템을 적용하더라도 ADR이 규모 확장성과 효율성을 만족시키므로 추가적인 장비 증설에 대한 비용을 줄일 수 있다.

5.3 APK파일 사이즈 오버헤드

100개의 APK파일을 대상으로 ADR을 적용하여 적용 전과 후의 APK 파일의 크기를 비교하였다. 아래의 그림은 그 중 25개의 결과를 나타낸다. 실험을 통해 ADR 적용에 따른 크기 증가율은 최대 30%에서 최소 0.5%이며 평균적으로는 3.6%인 것을 알 수 있다. 따라서 본 논문에서 제안한 기법을 적용하더라도 앱 마켓 시스템에서 저장공간 추가 확보를 위해 들어가는 비용을 최소화 할 수 있다.

ADR이 적용된 APK파일은 원본 APK파일과 비교하여 19KB와 518KB사이에서 증가하였고 증가된 크기는 APK파일 크기에 비례하지 않았다. 그 이유는 각 애플리케이션마다 민감한 API의 사용 빈도에 따라 ATPL의 API로 대체되는 수가 다르기 때문이며 추가적으로 ADR에서 DEX 파일을 재작성하는 알고리즘과 ZIP 압축 알고리즘의 영향을 받기 때문이다.

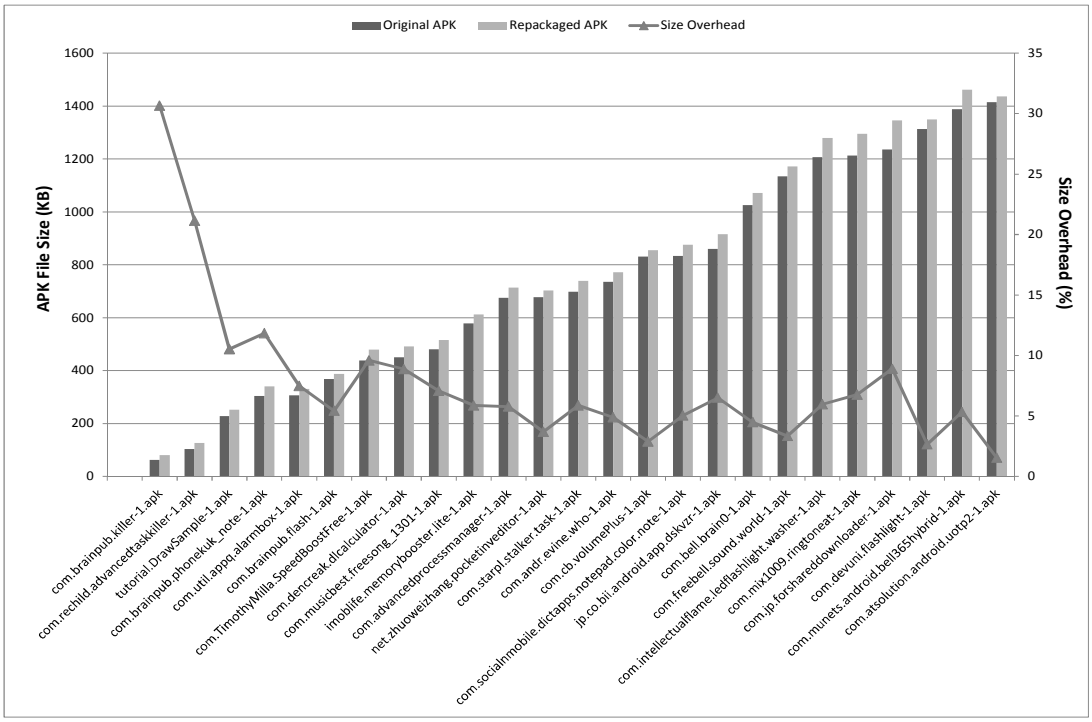

그림 5.2: 실행파일 사이즈 오버헤드

5.4 재패키징의 안정성

본 논문에서 제안한 기법은 애플리케이션 실행파일에 임의의 코드를 삽입하거나 기존 코드를 대체한다. 그러나 이러한 작업은 원본 실행파일의 출력을 변경 시킬 수 있는 가능성이 있다. 따라서 재패키지 된 애플리케이션이 원본 애플리케이션과 동일한 출력을 가짐을 보장하는 것은 중요한 일이다.

본 절에서는 재패키지 된 애플리케이션의 안정성에 대해서 검증하도록 하겠다.

5.4.1 Binary Instrumentation

Binary Instrumentation이란 소스코드 없이 임의의 코드를 프로그램에 삽입하는 기술을 말한다. 일반적으로 Binary Instrumentation은 프로그램 디버깅이나 퍼포먼스를 측정하기 위한 목적으로 사용되며 본 논문에서는 애플리케이션에 IRM (Inline Reference Monitor)을 삽입하기 위해 해당 기술을 사용하였다.

성공적으로 Binary Instrumentation을 하기 위해서는 원본 프로그램의 동작을 훼손하면 안 된다. 다른 말로 표현하면 수정된 프로그램은 주어진 입력에 대해 원본 프로그램과 동일한 결과를 출력해야한다는 것을 뜻한다[]. 따라서 본 논문에서 제안한 Binary Instrumenter인 ADR (Android DEX Rewriter)은 이를 보장해야한다.

Bernat의 연구에 따르면 수정된 프로그램이 원본 프로그램의 실행결과 (visible behavior)를 보존하기 위해서는 제어흐름제약 (control flow constraint)과 출력제약 (output constraint)를 만족해야한다[8].

• **제어흐름제약**: 동일한 입력에 대해 원본 프로그램 P와 수정된 프로그램 P'이 반드시 동일한 CFG (control flow graph)를 거쳐 동일한 출력에 도달해야한다는 것을 뜻한다. 결과적으로 P와 P'가 동일한 CFG를 가짐을 보이면 된다. 이를 위해 $CFGp'$의 경로 p'에 나타난 모든 instrumentation code를 삭제하는 $Filt$라는 함수를 도입한다. P와 P'가

아래의 조건을 만족하면 두 프로그램은 동일한 *CFG*를 갖는다.

$$\text{if } \forall x \in Inputs_p \text{ and } \forall x' \in Inputs_{p'} \\ s.t.\ x' \sqsupseteq x, Filt(ExecPath(P', x')) = ExecPath(P, x)$$

• **출력제약**: 원본 프로그램 P와 수정된 프로그램 P'가 제어흐름제약을 만족할 때 반드시 동일한 값을 출력해야함을 뜻한다.

5.4.2 안정성 검증

원본 애플리케이션의 *CFGp* = (V, E, Vs, Ve)에 대해 CFG의 시작 기본 블록을 Vs, 종료 기본 블록을 Ve라 할 때 재패키지 된 애플리케이션의 *CFGp'*가 *Filt*(*GFGp'*) = *CFGp*를 만족하면 재패키지 된 애플리케이션은 원본 애플리케이션과 동일한 *CFG*를 갖는다고 말할 수 있다.

실제로 소셜 커머스 애플리케이션인 위메프 (com.wemakeprice) 애플리케이션의 원본과 재패키지 본의 *CFG*를 분석해 보았다. 아래의 그림은 경로는 com.wemakeprice.common.Utils 클래스의 getIMEIDeviceID() 메소드의 호출을 시작점으로 하는 하나의 경로를 보인다. 이 경로에서는 폰 정보를 유출하는 TelephonyManager->getDeviceId() 메소드가 호출되기 때문에 ADR (Android DEX Rewriter)는 이 메소드의 호출을 ATPL (Android Trust Policy Library)의 AegisServiceClient->getDeviceID() 메소드를 호출하도록 대체한다.

그림 5.3을 통해 원본과 재패키지 된 애플리케이션의 실행흐름이 동일하다는 것을 알 수 있다. 이렇게 두 애플리케이션이 동일한 실행 흐름을 가질 수 있는 이유는 단순하게 원본에서 호출하는 일부 민감한 (sensitive) API 호출을 ATPL (Android Trust Policy Library)의 API의 호출로 대체하기 때문이다. 그림 5.4처럼 대체한 ATPL의 API는 원래의 API를 단순히 감싸고 있는 형태를 가지며 원래의 API에 전달되려하던 파라미터의 내용을 전혀 변경하지 않는다.

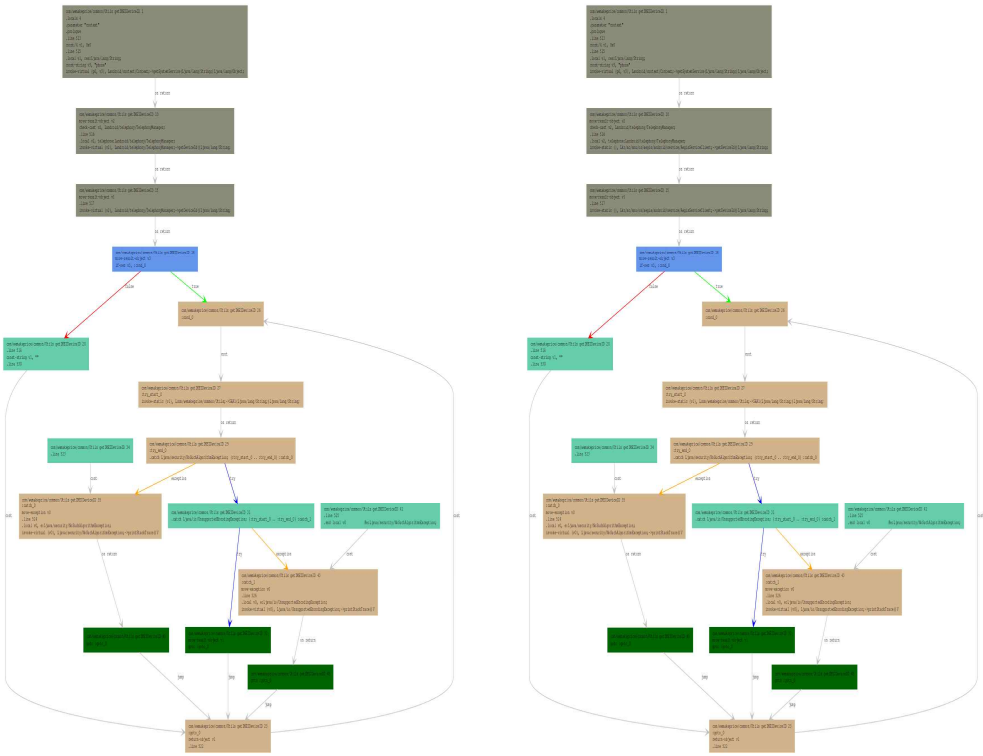

그림 5.3: 원본(좌측)과 리패키지 본(우측)의 CFG 중 하나의 경로

좀 더 상세하게 설명하면 그림 5.5처럼 실제 민감한 API의 호출은 ATPL을 경유하여 실행된다. 이러한 재배치 (relocation) 방식으로 실행파일을 수정하는 것은 애플리케이션 오작동의 가능성을 최소화 시킬 수 있는 방법이다.

그림 5.5는 원본 애플리케이션의 민감한 API의 호출이 리패키지 된 애플리케이션에서 ATPL의 API로 대체된 모습을 보여준다.

원본 애플리케이션에서 호출되는 TelephonyManager->getDeviceId()가 재패키지 된 애플리케이션에서는 AegisServiceClient->getDeviceID()로 변경된 것을 알 수 있으며 또한 ATPL로 우회된 실행 흐름이 원래의 실행흐름으로 복귀하는 것을 알 수 있다.

위 분석결과를 통해 본 논문에서 제시한 Binary Instrumentation 툴인 ADR은 제어흐름제약과 출력제약을 제안하는 만족하기 때문에 원본 애플리케이션의 실행 흐름을 그대로 보존하면서 애플리케이션을 재패키지 한다는 사실을 알 수 있다.

```
public static Location getLastKnownLocation(String provider) {
    Location location = null;
    showModalDialog("Alert", "This application is about to access to your location information. Do you want to allow this access?");

    switch (flag) {
    case AegisServiceClient.MANDATORY_ACCESS_ALLOW:
    {
        LocationManager locationManager = (LocationManager)mainActivity.getSystemService(Context.LOCATION_SERVICE);
        location = locationManager.getLastKnownLocation(provider);
    }
    break;
    case AegisServiceClient.MANDATORY_ACCESS_KILL:
    {
        android.os.Process.killProcess(android.os.Process.myPid());
        mainActivity.finish();
    }
    break;
    case AegisServiceClient.MANDATORY_ACCESS_SHADOW:
    {
        location = new Location(provider);
        location.setLongitude(0);
        location.setLatitude(0);
    }
    break;
    }
    return location;
}
```

그림 5.4: ATPL (Android Trust Policy Library)의 getLastKnownLocation 메소드

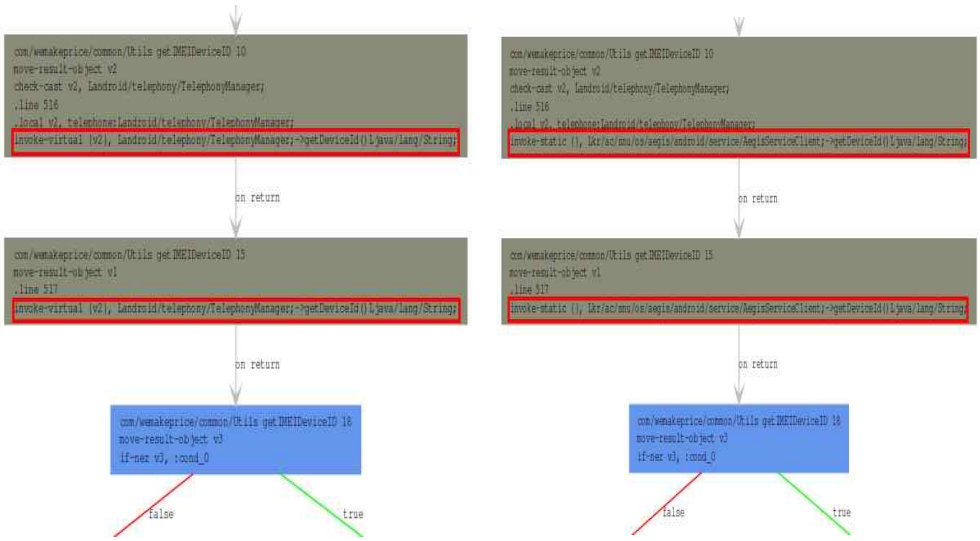

그림 5.5: 원본(좌측)과 리패키지 본(우측)의 대체된 API 비교

5.5 사례연구

본 절에서는 원본 애플리케이션과 재패키징 된 애플리케이션의 동작 과정을 비교하여 본 논문에서 제안된 시스템 안에서 사용자가 얼마나 강력한 보안을 경험할 수 있는지에 대해서 살펴보도록 하겠다.

위메이크프라이스 (wemakeprice.com) 애플리케이션은 소셜 커머스(social commerce) 애플리케이션 중에 하나이며 이 애플리케이션을 통해 사용자는 저렴한 가격으로 물건이나 서비스를 구매할 수 있다. 현재 이 애플리케이션은 공식적으로 구글 Play에 등록된 상태이며 자유롭게 다운로드하여 사용해볼 수 있다.

먼저 구글 Play로부터 위메이크프라이스 애플리케이션을 다운로드하여 폰 안에 저장된 설치파일을 추출한 후 원본 애플리케이션을 ADR (Android DEX Rewriter)를 이용하여 재패키징 하였다.

원본 애플리케이션과 재패키징 된 애플리케이션을 폰 안에 설치한 후 실험을 진행한 결과 몇 가지 악성행위를 탐지할 수 있었다.

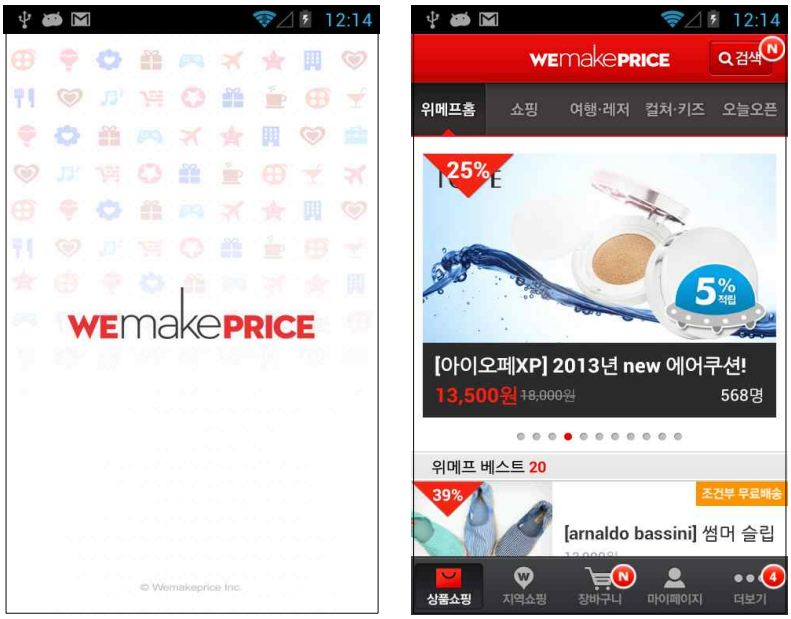

그림 5.6: 원본 애플리케이션의 실행화면

재패키징 된 애플리케이션을 수행한 결과 위메이크프라이스 애플리케이션은 실행 초기에 폰 안의 심 카드(SIM Card) 정보와 장치 ID에 접근하려는 시도를 하며 실행 중에는 지속적으로 위치정보에 접근하려함을 확인하였다.

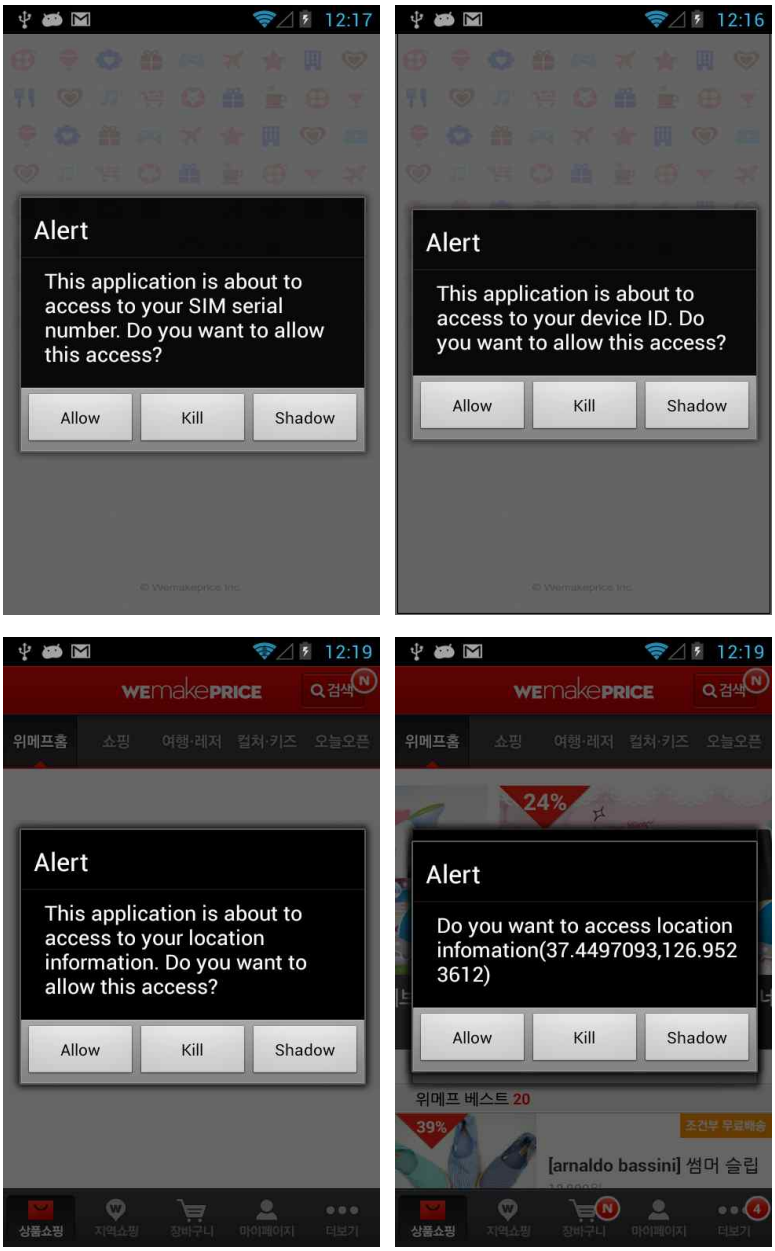

그림 5.7: 재패키징 된 애플리케이션의 실행화면

이렇게 사용자는 알람 팝업을 통해 애플리케이션의 악의적인 동작을 파악할 수 있을 뿐만 아니라 추가적으로 허용 (allow), 종료 (kill), 그리고 쉐도우 (shadow) 버튼을 통해 그러한 동작에 대한 결정을 할 수 있게 된다. 사용자가 허용 버튼을 선택할 경우 재패키징 된 애플리케이션은 원본 애플리케이션과 동일하게 동작하게 되며 쉐도우 버튼을 선텍하게 되면 임의의 값을 반환하게 된다. 마지막으로 종료 버튼을 선택하게 되면 애플리케이션은 종료가 된다.

제 6 장 결론

본 논문에서는 안드로이드 보안을 강화하기 위한 새로운 앱 마켓 시스템을 제안하였다. 제안된 새로운 앱 마켓 시스템은 각 애플리케이션 실행파일에 IRM (Inline Reference Monitor)를 삽입하여 악성 애플리케이션이 마켓의 검수과정을 우회하여 사용자 폰에 설치되더라도 사용자가 직접 애플리케이션의 동작을 감독할 수 있도록 하였다. 이는 마치 사용시간 (Time-of-Use) 퍼미션 시스템이 기존 안드로이드 퍼미션 시스템인 설치시간 퍼미션 시스템과 함께 공존하는 듯한 환영 (illusion)을 사용자에게 제공한다. 뿐만 아니라 본 논문에서 제안한 시스템은 설정된 보안 정책을 IRM을 통해 각 애플리케이션에 적용한다. 이를 통해 애플리케이션이 중요한 사용자 정보나 시스템 자원을 접근하려는 의심스러운 행위를 통제할 뿐만 아니라 애플리케이션 또는 운영체제 취약점을 이용한 공격을 차단할 수 있다.

제안된 마켓 시스템은 많은 수의 애플리케이션을 재패키지 해야 하는 부담이 있다. 기존 연구들에서 사용하는 Apktool을 이용할 경우 제안된 마켓 시스템의 설계 목표인 정확성, 규모 확장성, 그리고 효율성을 만족시킬 수 없다. 따라서 제안하는 마켓 시스템에 특화된 새로운 안드로이드 실행파일 재작성 프로그램인 ADR (Android DEX Rewriter)을 구현하였고 실험을 통해 ADR이 Apktool보다 평균적으로 46배 빠르게 애플리케이션 실행파일을 재작성 할 수 있음을 확인하였다. 뿐만 아니라 ADR을 통해 재패키징 된 애플리케이션이 원본 애플리케이션의 동작을 보존함을 증명하였다.

총 100개의 애플리케이션을 가지고 실험을 진행하였고 그 결과 재패키지 된 애플리케이션의 악의적인 행위가 모두 사용자에게 보고됨을 확인하였다.

현재 안드로이드의 보안 취약점이 부각되고 악성 애플리케이션 제작자들의 주요 목표대상이 되고 있는 시점에서 본 논문에서 제안하는 앱 마켓 시스템은 모든 안드로이드 장치에 적용될 수 있을 뿐만 아니라 손쉽게 기

존 앱 마켓 시스템과 통합 될 수 있어 실용적이다. 따라서 본 논문에서 제안한 새로운 앱 마켓 시스템은 안드로이드 보안을 강화하는데 큰 기여를 할 것으로 예상된다.

참 고 문 헌

[1] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, “A survey of mobile malware in the wild,” in Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, New York, NY, USA, 2011, pp. 3 - 14.

[2] M. Nauman, S. Khan, and X. Zhang, “Apex: extending android permission model and enforcement with user-defined runtime constraints,” in Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, 2010, pp. 328 - 332.

[3] M. Backes, S. Gerling, C. Hammer, M. Maffei, and P. Styp-Rekowsky, “AppGuard - real-time policy enforcement for third-party applications,” Jul. 2012.

[4] R. Xu, H. Saıdi, and R. Anderson, “Aurasium: Practical Policy Enforcement for Android Applications,” in Proceedings of the 21st USENIX conference on Security, 2012.

[5] W. Enck, “Defending users against smartphone apps: Techniques and future directions,” International Conference on Information System. Security, pp. 49 - 70, 2011.

[6] Y. Zhou. X. Jiang, “Detecting Passive Content Leaks and Pollution in Android Applications,” in Proceedings of the 20th Network and Distributed System Security Symposium (NDSS), 2013.

[7] J. Jeon, K. K. Micinski, J. A. Vaughan, A. Fogel, N. Reddy, J. S. Foster, and T. Millstein, “Dr. Android and Mr. Hide: Fine-grained Permissions in Android Applications,” ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices, 2012.

[8] A. R. Bernat, K. Roundy, and B. P. Miller, “Efficient, sensitivity resistant binary instrumentation,” in Proceedings of the 2011 International Symposium on Software Testing and Analysis, pp. 89 - 99, New York, NY, USA, 2011.

[9] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, “Hey, you, get off of my market: Detecting malicious apps in official and alternative Android markets,” in Proceedings of the 19th Annual Network and Distributed System Security Symposium (NDSS), 2012.

[10] B. Davis, B. Sanders, A. Khodaverdian, and H. Chen, “I-arm-droid: A rewriting framework for in-app reference monitors for android applications,” IEEE Mobile Security Technologies (MoST), San Francisco Ca, 2012.

[11] A. R. Beresford, A. Rice, N. Skehin, and R. Sohan, “MockDroid: trading privacy for application functionality on smartphones,” in Proceedings of the 12th Workshop on Mobile Computing Systems and Applications, pp. 49 - 54, 2011.

[12] W. Enck, M. Ongtang, and P. McDaniel, “On lightweight mobile phone application certification,” in Proceedings of the 16th ACM conference on Computer and communications security, pp. 235 - 245, 2009.

[13] M. Grace, Y. Zhou, Z. Wang, and X. Jiang, “Systematic detection of capability leaks in stock Android smartphones,” in Proceedings of the 19th Annual Symposium on Network and Distributed System Security (NDSS), 2012.

[14] A. P. Felt, K. Greenwood, and D. Wagner, “The effectiveness of application permissions,” in Proceedings of the USENIX Conference on Web Application Development, 2011.

[15] W. Enck, M. Ongtang, and P. McDaniel, “Understanding android security,” Security and Privacy. IEEE, vol. 7, no. 1, pp. 50 - 57, 2009.

[16] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner, “Analyzing inter-application communication in Android,” in Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services, pp. 239 - 252, 2011.

[17] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, “Android permissions demystified,” in Proceedings of the 18th ACM conference on Computer and Communications Security, pp. 627 - 638, 2011.

[18] M. Dietz, S. Shekhar, Y. Pisetsky, A. Shu, and D. S. Wallach, “Quire: Lightweight provenance for smart phone operating systems,” in 20th USENIX Security Symposium, 2011.

[19] A. P. Fuchs, A. Chaudhuri, and J. S. Foster, “SCanDroid: Automated security certification of Android applications,” Technical report, University of Maryland, 2009.

[20] W. Enck, P. Gilbert, B. G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, “TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones,” in Proceedings of the 9th USENIX conference on Operating systems design and implementation, pp. 1 - 6, 2010.

[21] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall, “These aren’t the droids you’re looking for: retrofitting android to protect data from imperious applications,” in Proceedings of the 18th ACM conference on Computer and Communications Security, pp. 639 - 652, 2011.

.

[22] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, and A. R. Sadeghi, “XManDroid: A new Android evolution to mitigate privilege escalation attacks,” Technical Report TR-2011-04, Technische Universität Darmstadt, 2011.

[23] M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel, “Semantically rich application-centric security in Android,” Security and Communication Networks, vol. 5, no. 6, pp. 658 - 673, 2011.

[24] A. P. Felt, H. J. Wang, A. Moshchuk, S. Hanna, and E. Chin, “Permission re-delegation: Attacks and defenses,” in Proceedings of the 20th USENIX Security Symposium, pp. 22 - 37, 2011.

[25] A. Desnos, “Android: Static Analysis Using Similarity Distance,” in 2012 45th Hawaii International Conference on System Science (HICSS), pp. 5394 - 5403, 2012.

[26] "Apktool - A tool for reverse engineering Android apk files.", https://code.google.com/p/android-apktool/.

[27] "IDC: Android Market Share Reached 75% Worldwide In Q3 2012.", http://techcrunch.com/2012/11/02/idc-android-market-share-reached-75-worldwide-in-q3-2012/, 2012.

[28] "Malicious Mobile Threats Report 2010/2011.", http://www.juniper.net/us/en/local/pdf/whitepapers/2000415-en.pdf, 2011.

[29] N. J. Percoco, "Adventures in BouncerLand", Black Hat USA 2012, http://media.blackhat.com/bh-us-12/Briefings/Percoco/BH_US_12_Percoco_Adventures_in_Bouncerland_WP.pdf, 2012.

[30] “Google reveals Android malware ‘Bouncer,’ scans all apps” http://www.computerworld.com/s/article/9223949/Google_reveals_Android_malware_Bouncer_scans_all_apps, Computerworld, 2012.

Abstract

A Novel App Market System for Enhancing Security on Stock Android

Cheol Jeon

Department of Computer Science and Engineering

The Graduate School

Seoul National University

As the number of malicious applications exploiting Android vulnerabilities has rapidly increased, a number of researches have been conducted. However, most researches on enhancing Android security require extensive modifications to the operation system or application framework, so they are not feasible for end users who use stock Android.

This thesis proposes a novel app market system that can enhance Android security without any modification of Android system. The proposed app market system grants users the ability to monitor and control installed application behaviors at runtime by inserting IRM (Inline Reference Monitor) into the application, even if the malicious applications are successfully installed on user devices circumventing

the review processes. Moreover, in some cases, the proposed system provides users with an alternative application version to which security-related patches are applied such that we can prevent attackers from exploiting the operation system or application vulnerabilities.

The proposed app market system provide a tool, ADR (Android DEX Rewriter), that rewrites Android executable and repackages it much faster than Apktool, an open source tool for reverse engineering Android binary. So, ADR can efficiently repackage large number of applications relatively and be applied to existing app market systems at a low cost.

We repackaged 100 applications downloaded from official Android app market (Google Play) and successfully installed IRM into the applications.

keywords : Mobile, Android, Security, App Market
Student Number **: 2011-23381**

감사의 글

지난 2년간의 석사과정동안 학문의 길로 인도해 주시고 자상한 조언과 배려를 아끼지 않으셨던 조유근 교수님께 감사드립니다. 그리고 이 논문이 완성되기까지 자기 일처럼 관심을 가져 주신 연구실 선배님들과 후배님들께 감사드립니다. 그동안 연구실 생활을 하면서 그분들의 근면과 성실함에 탄복하고 존경해 왔음을 뒤늦게나마 고백합니다. 이렇게 훌륭하신 분들과 함께 일할 수 있어서 영광이었습니다. 그분들의 앞길에 축복이 깃들기를 기원합니다.

지나온 시간을 되돌아보면 철없던 어린 시절부터 부족한 저를 돌보아 주시고 사랑으로 감싸주신 훌륭하신 분들 덕분에 어려운 환경 속에서도 꿈을 꿀 수 있었고 앞으로 나아갈 수 있었던 것 같습니다. 저를 자식처럼 아껴주시고 많은 조언을 해주셨던 중·고등학교 은사님이신 최춘환 선생님과 이정희 선생님께 감사드립니다. 그리고 어려운 가정환경 속에서도 학업을 이어갈 수 있도록 중·고등학교 동안 학자금을 지원해 주셨던 라이온스 클럽 관계자 분들께 감사의 말씀을 드립니다. 그리고 군 복무 시절에 큰 위로와 도움을 주셨던 최대성 주사님께 감사드립니다. 그리고 중학교 때부터 지금까지 변함없는 우정을 보여준 지훈이와 범석이에게 감사의 말을 전합니다. 그리고 지난 10년이 넘는 시간동안 저를 가족처럼 대해준 대학교 동아리 선·후배님들께도 감사드립니다.

젊어서는 어린자식들을 홀로 키우느라 고생하시고 나이가 들어서는 자식들의 미래를 걱정하느라 하루도 마음 편할 날이 없는 가여운 나의 어머니를 생각하면 가슴이 먹먹해지고 눈시울이 붉어 옵니다. 어머니의 은혜에 고개 숙여 감사드립니다. 그리고 가장으로서 고생하고 있을 형님께도 감사의 말씀을 전합니다.