모바일 어플리케이션의 정적 분석을 통한 개인정보 유출 차단

Blocking Personal Information Leakage of Mobile Application using Static Analysis

요 약

스마트폰의 빠른 보급에 따라 거의 대부분의 사람들이 스마트폰을 통해 어플리케이션을 사용하는 시대가 도래했다. 그러나 특히 안드로이드 플랫폼 상에서의 어플리케이션은 권한 시스템의 한계점으로 인해 어플리케이션이 어떤 정보를 요구하는지 등을 알기 어려우며, 요구하는 정보들 중 일부만 거부하는 것도 불가능하다. 또한, 획득된 정보가 기기 내에서만 사용되는지, 외부로 전송되는지도 알 수 없는 것이 현실이다. 따라서 본 연구에서는 모바일 어플리케이션을 정적 분석한 결과를 토대로 개인정보가 유출될 수 있는 부분에서 사용자의 의도를 물어보고 사용자가 원하는 방향으로 실행될 수 있도록 하고, 별도로 개인정보 관리 기능을 제공하여 추후에도 이미 부여된 권한을 수정하고 체계적으로 관리할 수 있도록 하는 시스템을 구현해 보았다.

모바일 어플리케이션의 정적 분석을 통한 개인정보 유출 차단

Blocking Personal Information Leakage of Mobile Application using Static Analysis

I. 서론

1.1 연구의 필요성

2013년 인터넷이용실태조사 결과[1]에 따르면 스마트폰을 보유한 가구의 비율은 79.7%로 5가구 중 4가구는 스마트폰을 사용하고 있다. 여러 통계 자료로 볼 때 우리의 일상에서 이미 스마트폰은 땔 레야 땔 수 없는 중요한 전자 기기가 되어 있는 것이다. 또한, 2012년 하반기 스마트폰이용실태조사 결과[2]에 따르면 스마트폰 이용자의 1인 평균 설치된 어플리케이션의 개수는 46.1개로 상당히 많음을 알 수 있다. 한편, 삼성 갤럭시S 내의 '거울' 위젯 어플리케이션이 동작하는 기능에 비해 엄청난 권한을 요구하고 있던 것[3]이 밝혀진 바 있으며, 얼마 전에는 유명 손전등 어플리케이션에서 유심 칩 번호 등의 중요한 개인 정보를 외부로 유출한다는 것[4]이 드러난 바가 있다. 앞에서 보이는 것처럼 원래의 목적과는 다르게 사용자 몰래 개인정보를 유출할 수 있는 악성 어플리케이션이 증가하고 있다. 또한, 악성 어플리케이션이 아니라도 사용자 위치 기반 광고 서비스 등 부가 서비스를 위해 사용자의 개인정보 영역에 접근하는 경우가 많이 늘어나고 있다.

그림1 손전등 앱 개인정보 훔쳐가… 유심칩 번호까지 유출[4]

Fig.1 Flashlight app steals personal information… Leakage up to USIM chip number[4]

일반적으로 사람들이 널리 사용하는 스마트폰은 크게 iOS와 Android로 구분된다. iOS를 사용한 아이폰의 경우에는 어플리케이션이 요청하는 모든 권한에 대해 승인 여부를 물어보는 팝업 창을 띄우고 사용자의 답변을 받는 형태로 개인 정보를 관리하며 그림2처럼 설정 창에서 각 권한별로 승인 여부를 쉽게 변경할 수 있도록 한다.

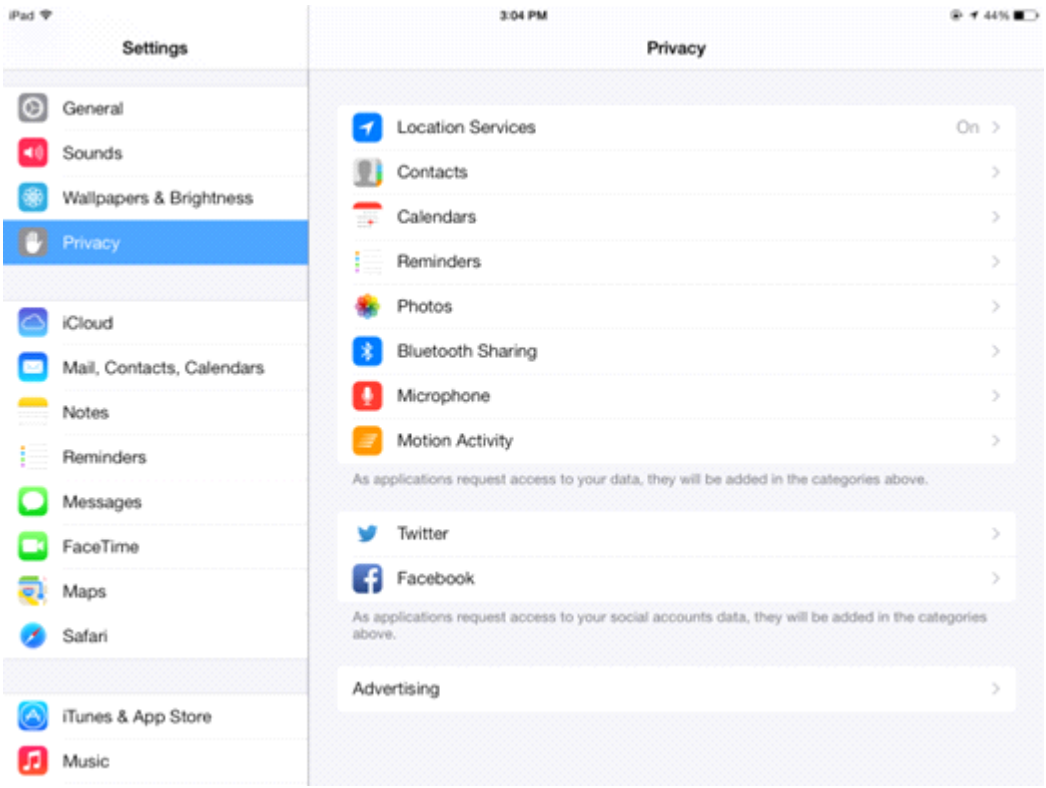

그림2 iOS 상에서는 사용자가 각 개인정보에 대해 사용을 허가하거나 거부할 수 있음

Fig.2 On iOS, user can grant or refuse for use of specific kind of personal information

이에 반해, 2014년 1월 기준으로 대한민국 스마트폰 사용자의 93.4%가 사용하는 플랫폼인 Android[5]는 그림3처럼 처음에 설치할 때에 단 한 번, 이 어플리케이션이 이러한 개인정보에 접근할 수 있음을 표시할 뿐이다. 그마저도 이 중 하나라도 거부한다면 그 어플리케이션은 설치조차 되지 않는다. 따라서 특정 기능을 이용하고 싶은 사용자는 울며 겨자 먹기로 어플리케이션이 요구하는 권한을 모두 수락할 수밖에 없는 것이다. 한편, 어플리케이션이 특정 권한을 획득한다면, 그 이후부터는 사용자에게 알려지는 것 하나 없이 개인정보에 쉽게 접근할 수도 있다. 따라서 Android 상에서도 어플리케이션이 개인정보를 사용할 수 있도록 하는 권한을 각 유형별로 사용자가 관리할 필요가 있다.

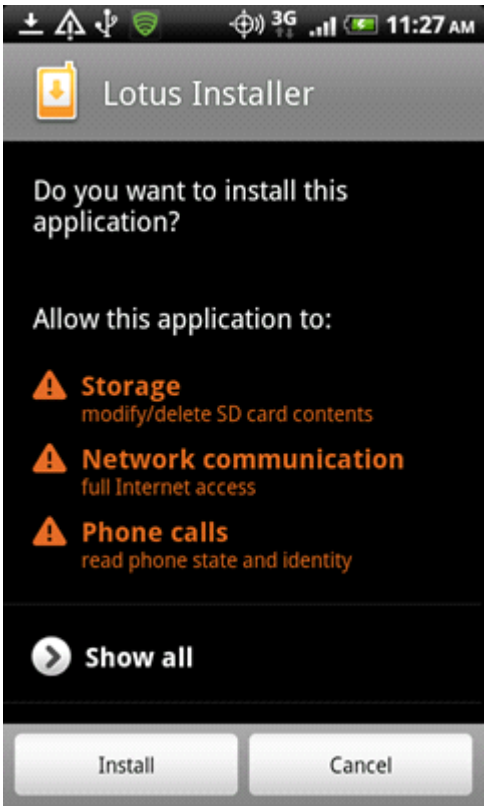

그림3 어플리케이션 설치 시 사용자는 모든 권한에 동의해야만 함

Fig.3 When user installs application, user must accept all of permisions

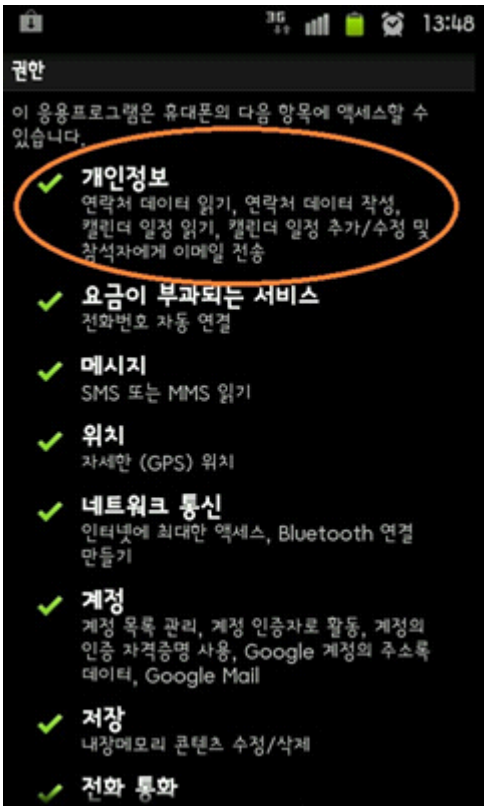

그림4 개인정보에 접근할 수 있는 권한이 있는 어플리케이션

Fig.4 Application which has permission to access personal information

1.2 연구 현황

1.2.1 ScanDal[6]

달빅 가상머신 언어로 기술되어 있는 안드로이드 어플리케이션을 분석 대상으로 하는 정적 분석기이다. 달빅 프로그램에 대해 분석한다는 것은 자바 역컴파일러등의 툴을 이용하지 않는다는 뜻이다. 악성 행동을 의도한 어플리케이션의 경우 달빅 바이트코드 수준에서 코드의 수정이 있을 수 있고, 그 경우 자바로의 역컴파일러가 제대로 작동하지 못 할 수 있다. 따라서 달빅 바이트코드의 실행 의미를 직접 분석한다. 분석이 어플리케이션으로부터 알아내고자 하는 성질은 어플리케이션이 개인정보를 누출시키는지의 여부로, 개인정보의 누출은 정보의 소스(개인정보를 운영체제로부터 꺼내오는 API 함수 호출 위치)로부터 싱크(임의의 데이터를 기기 밖으로 내보낼 수 있는 API 함수 호출 위치)로 데이터 흐름의 존재로 정의하여 요약 해석 기법을 통해

구현된 도구이다.

1.2.2 TaintDroid

TaintDroid는 Android Open Source Project(AOSP)를 기반으로 하여 제작된 시스템으로, 실행되는 어플리케이션이 어떤 정보에 접근하는지를 실시간으로 감시한다. 이를 통해 개인정보에 접근하거나 유출하는 것이 확인될 경우에는 사용자에게 통지를 하거나 작동 여부를 물어보는 역할을 한다. 그러나 동적 분석의 특성상 실제로 실행을 해 봐야 하며 작동 속도도 느려진다. 관련 논문[7]에 따르면 IPC 벤치마크를 사용하여 성능을 평가했을 때 원래의 Android 시스템에 비해 27% 더 느리다.

1.3 이론적 배경

1.3.1 정적 분석[6]

정적으로 분석한다는 것은 프로그램을 실행시켜 보지 않고 원하는 성질을 얻어내는 것이다. 이를 자동으로, 즉 일단 분석기가 완성되면 사람의 손을 거치지 않고 분석기가 스스로 대상 프로그램을 분석한다. 요약 해석에서 제안하는 조건들을 만족하는 분석기를 디자인할 경우, 그 분석기는 해당 실행 의미에 대해 모든 실행과정을 안전하게 포섭하게 된다. 이 방법은 임의의 언어와 임의의 성질에 대해 활용할 수 있고, 디자인과 구현에 따라 다양한 정밀도를 얻어낼 수 있다.

1.3.2 Android 어플리케이션 파일의 구조

Android 시스템에서 어플리케이션은 APK 파일로 지정되어 있다. 이 APK 파일은 일반적으로 사용되는 압축 파일 형식인 ZIP 파일과 같은 구조를 가지고 있다. 이 파일의 내부에는 그림5에서 나타난 것과 같이 이미지, 음원과 같은 리소스 파일들과, 컴파일된 바이너리들로 이루어진 classes.dex 파일, 어플리케이션의 컴포넌트 목록, 권한 등과 같은 정보를 포함하고 있는 AndroidManifest.xml 등의 파일로 이루어져 있다.

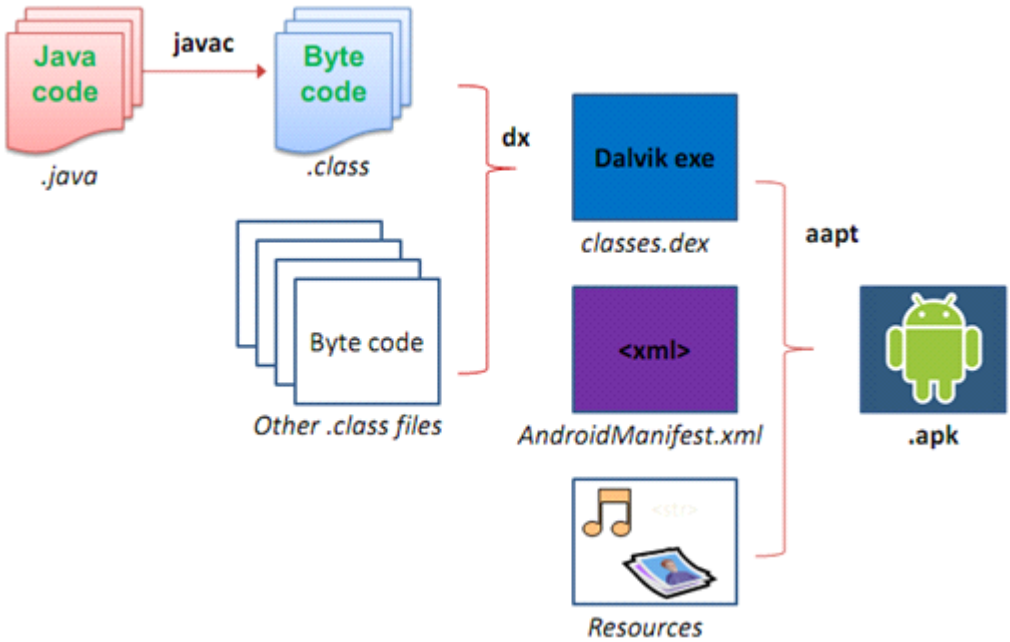

그림5 Android 어플리케이션 파일의 구조[8]

Fig.5 Structure of Android application file[8]

1.3.3 Dalvik VM 및 smali 언어

Android 시스템은 Dalvik이라는 가상 머신 위에서 실행되는데, 이 가상 머신의 명령 코드를 포함하는 것이 위에서 설명한 classes.dex 파일이다. 여러 개의 java 소스 코드로부터 컴파일된 class 파일들이 합쳐져 Dalvik OpCode들로 바뀌어 바이너리 생성되는 것이다.

한편, 이렇게 생성된 바이너리 파일은 사람이 읽을 수 없기에 조금 더 사람이 쉽게 볼 수 있도록 하는 변환 작업이 필요하다. baksmali라는 과정을 통하면 classes.dex 파일은 사람이 읽기 좋도록 const(상수 지정 명령), move-result(함수의 실행 결과를 레지스터로 복사), invoke-static(정적 멤버 함수 호출) 등으로 쉽게 풀어 써져 있는 smali 코드 형태로 바뀌게 된다. 이를 이용하면 어플리케이션의 특정 부분을 바꿀 수 있다. 다시 smali라는 툴을 이용하면 이 smali 코드 파일들을 classes.dex로 바꿀 수도 있다.

II. 연구과정 및 방법

2.1 구현 내용

2.1.1 개인정보 흐름의 차단

본 연구에서는 개인정보를 유출을 ScanDal에서 분석한 결과와 같이 프로그램상의 두 부분에서 차단하고자 한다. 첫 번째는 어플리케이션이 개인정보를 시스템에서 얻어오는 부분(이하 source, 소스)이고, 두 번째는 얻어온 개인정보를 인터넷, MMS, 파일 등을 통해서 외부로 유출하려 하는 것으로 추정되는 부분(이하 sink, 싱크)이다.

2.1.2 시스템의 구조

본 연구에서 구현한 시스템은 크게 2가지 프로그램으로 이루어져 있다.

첫 번째 프로그램은 주어진 APK 파일과 이 어플리케이션을 ScanDal로 분석한 결과를 이용해서 어플리케이션을 압축 해제한 후, smali 코드를 고쳐서 classes.dex 파일을 바꾼 뒤 다시 압축하여 수정된 APK 파일을 만들어주는 프로그램인 PrivateRepacker이다. 두 번째는 Android 시스템 내에서 어플리케이션이 개인정보 유출로 추정되는 부분을 차단할지 그대로 실행할지를 선택, 관리 할 수 있게 하는 어플리케이션인 PrivateManager이다.

PrivateManager는 그림6과 같이 크게 2가지 역할을 하는데, 첫 번째는 어플리케이션이 개인정보에 접근하거나, 개인정보를 유출하려는 시도를 할 때, 다이얼로그를 띄워서 코드의 해당 부분의 실행을 승인할지 거부할지 선택하게 하는 역할이다. 두 번째 역할은 이미 사용자가 선택한 적이 있는 개인정보의 흐름에 대해서, 이전에 입력한 승인/거부 여부를 변경할 수 있게 설정하는 부분이다.

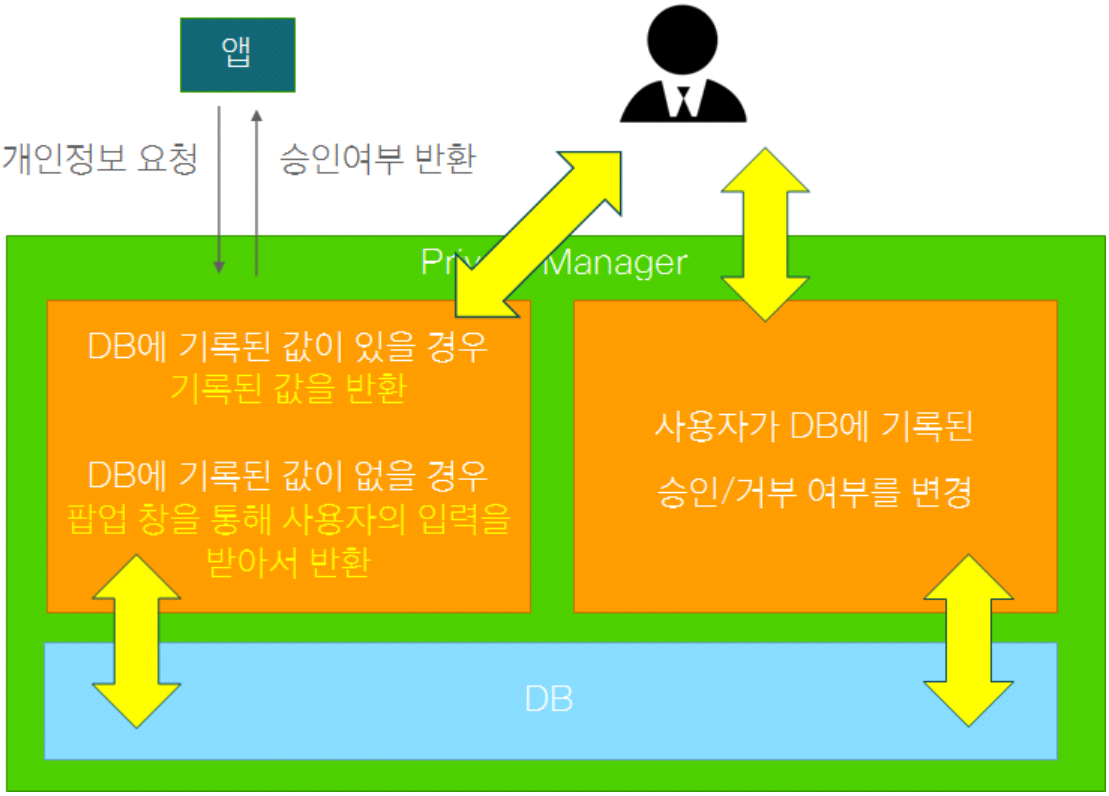

그림6 PrivateManager가 하는 일

Fig.6 What PrivateManager does

이 PrivateManager에서 차단 할 필요가 있는 프로그램의 실행부분 각각의 실행 여부를 관리할 수 있게 하고, 사용자가 설정한 적이 있는 실행지점의 경우 다시 묻지 않고 저장된 기록을 활용함으로써, 본 연구에서는 과다한 다이얼로그의 노출로 사용자가 어플리케이션 사용 중 불편을 겪는 일이 최소가 되게 할 수 있도록 했다. 정적 분석의 특성상 나타나는 개인정보의 유출지점이 아님에도 불구하고 개인정보의 유출로 판단해 차단하는 부분이나, 어플리케이션이 동작하는데 필수적으로 활용되는 개인정보의 활용의 경우에 차단해서 어플리케이션이 정상적으로 동작하지 않으면, 설정을 변경하여 어플리케이션이 정상적으로 동작할 수 있게 하여 정적 분석의 한계점을 해결하였다.

2.2 구현 방법

2.2.1 ScanDal과의 상호작용

ScanDal에서 정적 분석을 통해 얻어낸 소스와 싱크 정보를 각각 json 파일 형식으로 전달한다. json 파일은 프로그램의 실행 부분을 나타내는 JsonObject의 Array로 되어있으며, 각각의 Object에는 어떤 객체의 어떤 함수에서 실행되는 부분이고, 해당 함수의 몇 번째 명령어인지, 어떤 함수를 호출하는 부분인지, 어떤 종류의 개인정보를 받아오고, 어떤 경로를 통해서 나가는 지와 같은 정보가 아래와 같이 주어진다.

{“type”: “src”, *// 소스면 src, 싱크면 sink*

“category”:”geo”, *// 위치정보는 geo, 전화번호는 number, 기기번호는 imei, …*

“pos”: {

“caller”: ["Lcom/test/Test;”, “test:()V”],

“line”:”1”, *// 호출위치. 리스너의 인자일 경우 “arg”: "1”*

“callee”: [“…”, “…”]

}

}

2.2.2 inAppClass

어플리케이션에 코드를 추가할 때, 반복되는 코드를 줄이고 작업의 편의성을 높이기 위하여 inAppClass라는 클래스를 자바 코드로 작성한 뒤 smali 코드로 변환하여 원래 어플리케이션의 smali 코드와 함께 컴파일하였다. inAppClass가 하는 일은 크게 두 가지인데, 첫 번째는 어플리케이션이 실행 중에 현재 소스나 싱크가 차단되는지 승인되는지 확인해야 할 때 해당 어플리케이션의 고유 식별자와 소스/싱크의 번호를 인자로 이 함수를 호출하면 승인/차단 여부를 boolean으로 리턴해 주는 일이다. 이 함수가 호출이 되면 시스템은 우선 PrivateManager가 설치되어 있는지 확인한다. 설치되어 있다면 우선 PrivateManager와 통신하기 위한 로컬 소켓을 하나 생성한 뒤, 로컬 소켓에 접근할 수 있는 경로와 소스/싱크의 정보, 호출한 시간 등의 정보를

Intent에 담아서 PrivateManager를 호출한다. 마지막으로, PrivateManager가 결과를 로컬 소켓을 통해 전송할 때 까지 최대 9초 기다린다. 이는 Android 시스템의 ANR(Application Not Responding, 어플리케이션이 10초 동안 아무런 동작도 하지 않을 경우 강제로 어플리케이션을 종료시키는 과정)을 피하기 위한 수단이다.

두 번째 역할은, 프로그램의 Context를 관리하는 것이다. Android 시스템에서는 시스템 함수를 호출 할 때, 어플리케이션을 구분하기 위해 각각에 고유한 Context라는 클래스의 인스턴스를 부여한다. 이 Context는 다른 어플리케이션의 컴포넌트를 호출하거나 사용자와 상호작용하기 위한 다이얼로그 등이 포함되어 있는 UI 작업을 하는 데에 필수적으로 필요하기 때문에, inAppClass에서 PrivateManager를 호출하는 시점에서 어플리케이션의 Context의 인스턴스가 있어야 한다. 그러나 소스/싱크 같은 임의의 프로그램 실행 지점에서 콘텍스트를 얻어오는 방법은 경우의 수가 많고 구할 수 없을 수도 있다. 따라서 어플리케이션의 컴포넌트가 시작되는 지점에서, inAppClass의 setContext 함수를 호출하여 inAppClass에 정적 멤버 변수로 Context를 저장 해 놓고, 필요할 때 불러서 쓸 수 있게 하였다.

2.2.3 PrivateRepacker

PrivateRepacker는 어플리케이션을 본 연구에서 구현한 기능이 포함될 수 있도록 수정하는 프로그램이다. java 언어를 기반으로 작성되었고, 자체적으로 수정한 smali/baksmali[9] 및 json-simple[10], signapk[11], AXMLPrinter2[12] 등의 공개 프로젝트들을 사용하였다.

PrivateRepacker는 APK 파일을 읽어 와서 가장 먼저 해당 어플리케이션의 고유 식별자로 사용되는 해시 값을 MD5 알고리즘을 이용하여 생성한다. 악의적으로 변경된 어플리케이션의 경우 원래의 정상적인 어플리케이션과 버전 코드, 버전 이름까지 동일하게 따라할 수 있으므로 최종적으로 생성된 APK 파일 자체의 해시 값만 고유 식별자로 사용한다. 그 후 내장된 압축 해제 라이브러리를 통해 파일의 압축을 해제한

다. 그리고 자체적으로 수정한 baksmali 프로그램을 이용하여 classes.dex 파일을 smali 파일들로 변환한다. 그 이후, 어플리케이션에 smali 코드를 추가한다. 우선 inAppClass 관련 파일들을 추가 한 뒤, ScanDal이 분석한 json 파일을 json-simple 라이브러리를 활용해 읽어서 소스/싱크 목록을 얻어온다. 각각의 소스/싱크에 대해서 함수 호출 및 결과를 받아오는 부분을 실행 할지 뛰어 넘을지 선택하게 해야 한다. 따라서 그림7과 같이 해당 부분을 실행하기 전에 2.2.2에서 기술한 PrivateRequest 함수를 호출하여 결과 값을 받아온 뒤, 분기문을 통해서 원래의 코드를 실행하거나 그렇지 않게 하였다.

그림7 변경된 smali 코드의 모습

Fig.7 Figure of changed smali code

이 과정에서, 수정되는 함수마다 추가적인 레지스터(지역변수)가 필요하게 되었는데, 원래에 있던 레지스터는 임의로 사용할 경우 어플리케이션에 심각한 오류를 발생시킬 수 있으므로 레지스터를 별도로 추가하여 사용하였다. 한편, 어떤 함수를 호출할 때에 그 인자는 무조건 가장 뒤에 있는 레지스터부터 들어가므로 함수가 시작될 때에 인자들을 원래 있던 자리로 옮겨주는 작업을 추가해 주었다. 이 과정에서 원래의 baksmali/smali는 전체 레지스터의 개수만 볼 수 있거나 인자의 개수만 볼 수 있기에 이를 둘 다 볼 수 있도록 baksmali/smali를 수정하였다. 또한, Dalvik VM 특성상

레지스터가 16개를 넘어가는 경우 다른 형태의 명령어(wide/range 계열)를 사용해야 하기에 이를 구현해 주었다.

smali 코드를 수정한 이후에는, 이를 이용하여 새로운 classes.dex 파일을 생성한 뒤, 나머지 파일들과 함께 압축하였다. 리소스들은 STORED 메서드를 사용하여 압축함으로써 오동작을 방지하였다. 최종적으로는 압축된 파일에 서명을 해서 기기에 설치할 수 있게 한다. 이후 사용자의 요구에 따라 생성된 APK 파일을 바로 기기에 설치하도록 할 수도 있다.

2.2.4 PrivateManager

PrivateManager는 2.1.1에서 설명한 두 가지 기능을 구현하기 위해서 두 가지의 Activity를 구현한다.

첫 번째는 PrivateRequestActivity로, inAppClass의 PrivateRequest에서 호출하는 컴포넌트 이다. 이 Activity가 호출이 되면, 우선 Intent를 통해 전달된 정보와 일치하는 실행 지점에 대한 정보가 SharedPreference에 저장되어 있다면 이 정보를 그대로 전달된 로컬 소켓을 통해서 boolean 값을 넘겨준다. 저장되어 있지 않다면, AlertDialog를 띄워서 사용자에게 inAppSmali에서 Intent를 생성한 시간 이후로 최대 9초까지 사용자에게 해당 실행 지점을 실행할지 말지에 대한 입력을 받은 후 사용자의 선택을 로컬 소켓을 통해서 보내주고, SharedPreference에 저장한다.

두 번째로 구현한 Activity는 PrivateSettingActivity로, 이 Activity에서는 사용자가 선택한 기록이 있는 어플리케이션의 목록을 SharedPreference로부터 읽어 와서 ListView를 통해 보여준다. 사용자가 특정 어플리케이션을 선택하면, 새로운 ListView에 사용자가 선택한 기록이 있는 프로그램의 실행 지점을 보여주고, 그 부분의 실행 여부에 대한 선택을 사용자가 변경하면, SharedPreference에 저장된 기록을 변경하도록 하였다.

III. 결과 및 토의

3.1 연구 결과

구현된 PrivateRepacker를 사용하여 생성된 APK 파일을 Android 기기에 설치해 본 결과, 아래의 그림8, 그림9와 같이 의도한 대로 사용자의 개인정보에 접근할 때 및 유출이 가능할 때 사용자의 동의 여부를 물어보는 창이 나타났고, PrivateManager에서 다시 그 권한을 수정할 수 있는 것을 확인할 수 있었다.

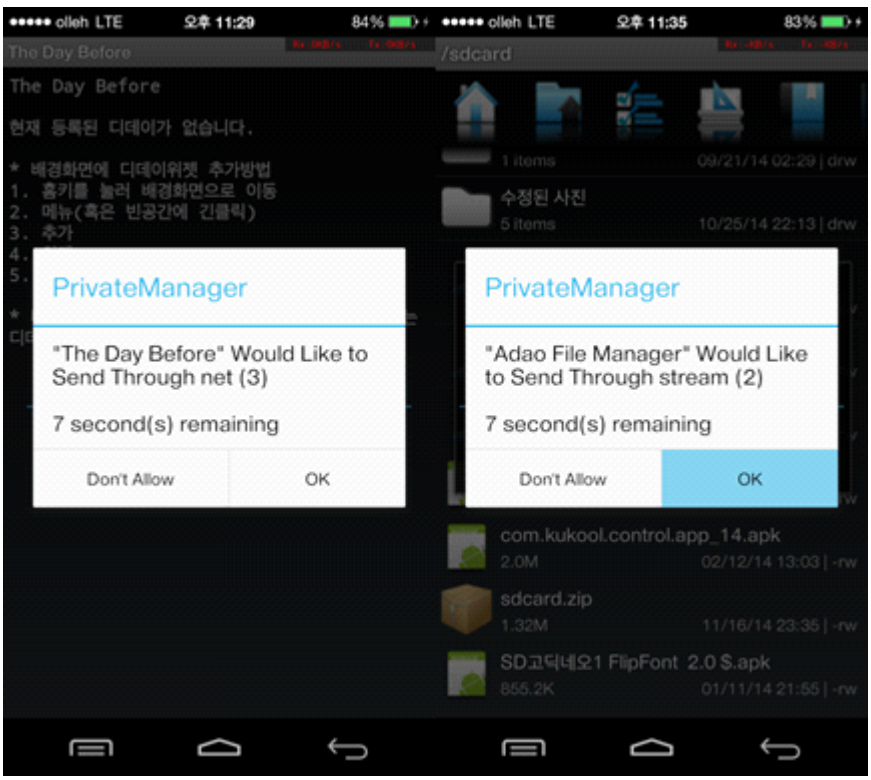

그림8 개인정보가 획득/유출되려는 부분에서 사용자의 의사를 묻는 창이 나타남

Fig.8 Asking dialog is appeared when personal information is going to be acquired/spilled

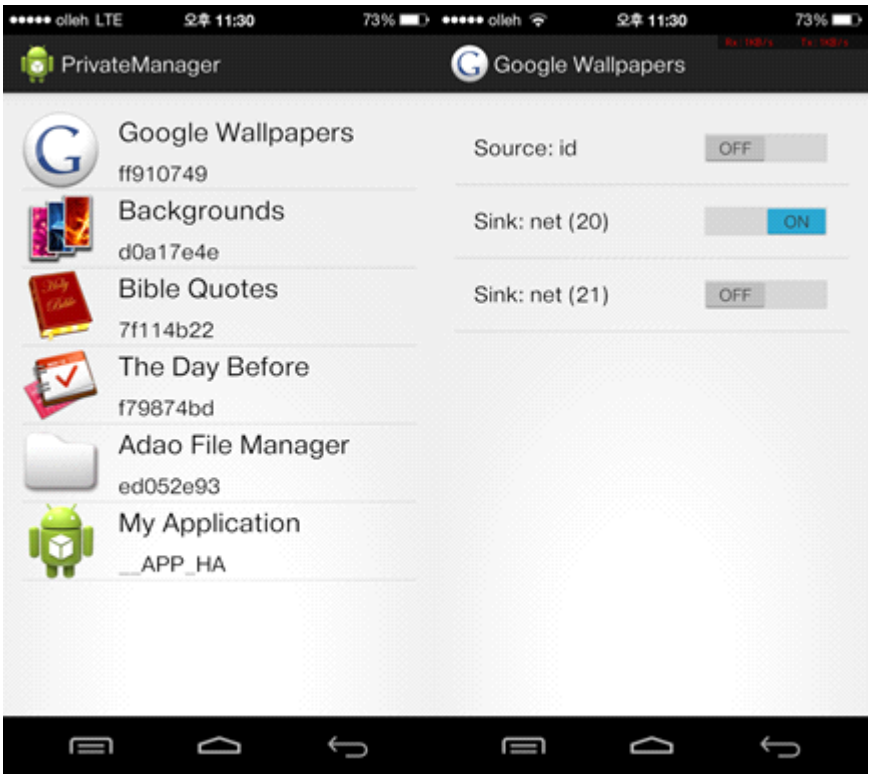

그림9 최초 1회의 다이얼로그 이후에도 사용자가 개인정보 접근/유출에 관한 허가 여부를 변경할 수 있다

Fig.9 User can change whether access/spill is permitted or declined after the first dialog

3.2 연구의 한계점 및 발전 가능성

본 연구의 한계점으로는 정적 분석 결과 개인정보가 유출되는 것으로 추정되는 싱크 부분에서 사용자가 프로그램 실행 지점의 실행여부를 선택할 때 참고할 수 있는 자료가 거의 없다는 점이 있다.

이러한 한계점을 해결하기 위해서 고안한 개선 방법으로는, 우선 소스-싱크 매칭이 있다. 지금은 독립적으로 존재하는 싱크의 목록을, ScanDal에서 어떤 소스에서 나온 개인정보가 어떤 싱크로 유출되는지에 대한 정보를 이용해서, 현재 선택해야 하는 싱크가 어떤 종류의 개인정보를 유출할 가능성이 있는지에 대해서 보여준다면, 지도

어플리케이션이 현재 검색 장소를 찾기 위해 위치정보를 전송하는 것과 같이 어플리케이션이 정상적으로 동작하는데 필수적으로 요구되는 개인정보의 전송을 제어하는 데에 도움이 될 수 있을 것이다.

두 번째로 고안한 개선 방법으로는, 여러 사용자가 이 시스템을 사용하면서 얻어진 개인정보 접근 또는 유출에 대한 동의/거부 여부를 서버를 이용해 수집하여 통계 자료를 제공하는 것이다. 많은 사용자가 이 시스템을 이용하게 된다면, 어떤 어플리케이션의 특정 실행 지점에서 여러 사람들이 선택한 통계 자료가 존재할 것이다. 많은 사람들이 거부한 곳에 대해서는 자동적으로 거부하도록 할 수도 있고, 다이얼로그를 띄울 때 통계 자료를 함께 제공하여 이를 참고하여 선택할 수 있게 하면, 더욱 정확한 판단을 하게 할 수 있을 것이다.

IV. 결론

스마트폰은 편리함을 무기로 하여 수많은 개인정보 유출 위험을 지니고 있는 도구이다. 따라서 편리하면서도 개인정보 유출 위험을 최소화하기 위해 사용자의 의도를 파악하고 이에 따라 어플리케이션이 개인정보에 선택적으로 접근할 수 있도록 하는 시스템에 대한 연구를 진행하였다. 이 시스템은 정적 분석을 통해 나온 결과를 이용함으로써 직접 실행시켜 보지 않아도 적용할 수 있으므로 시간적으로 이득이 상당하며, 어플리케이션 실행시 모든 동작을 계속 추적함으로써 성능이 하락되는 TaintDroid와 다르게 성능 하락도 전혀 없다. 정적 분석이 유출 가능성이 있는 부분을 모두 출력함으로써 과도하게 접근이 제한되는 부분과 어플리케이션의 정상적인 작동을 위해 필수적으로 요구되어지는 부분도 사용자가 직접 설정할 수 있도록 하여 오작동을 최대한 방지할 수 있도록 했다. 이 연구에서 나온 결과물이 잘 사용된다면 사용자가 편리하면서도 체계적으로 자신의 개인정보를 지킬 수 있게 되어 보안 수준을 한 단계 더 높일 수 있을 것이다. 이후에도, 소스-싱크 매칭 및 통계 자료 활용을 적용한다면 더 스마트하게 자신의 개인정보를 관리할 수 있게 될 것이다.

V. 참고문헌

[1] 한국인터넷진흥원 (2013), “2013년 인터넷이용실태조사 결과 발표”, Retrieved November 30 2014 from GSHS, Web site: http://isis.kisa.or.kr/board/?pageId=060200&bbsId=3&itemId=801

[2] 한국인터넷진흥원 (2013), “2012년 하반기 스마트폰이용실태조사 결과발표”, Retrieved November 30 2014 from GSHS, Web site: http://isis.kisa.or.kr/board/index.jsp?pageId=060200&bbsId=3&itemId=799

[3] 동아일보 (2011), “삼성 갤럭시S ‘거울’앱 속에 당신의 정보 몰래 보는 ‘눈’이 있다”, Retrieved November 30 from GSHS, Web site: http://news.donga.com/3/all/20111205/42360234/1

[4] MBC (2014), “[단독] ’손전등 앱’ 개인정보 훔쳐가…유심칩 번호까지 유출”, Retrieved November 30 from GSHS, Web site: http://imnews.imbc.com/replay/2014/nwdesk/article/3553208_13490.html

[5] 연합뉴스 (2014), "한국은 세계 1위 안드로이드 공화국…93.4%가 사용", Retrieved November 30 from GSHS, Web site: http://www.yonhapnews.co.kr/it/2014/01/19/2405000000AKR20140119057700017.HTML

[6] 윤용호 (2013), “안드로이드 앱에서 개인정보 누출을 검출하는 정적분석기 설계”, 공학석사 학위논문, 서울대학교.

[7] William Enck, Peter Gilbert, Byung-gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. (2010) “TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones”, Symposium on Operating Systems Design and Implementation (OSDI), pp.11-12.

[8] Ajinkya Saswade (2014), “Decompile and Secure android apk”, Retrieved November 30 from GSHS, Web site: http://decompileandsecureapk.wordpress.com/2014/05/10/decompile-and-secure-android-apk/

[9] jesusfreke@jesusfreke.com 외 2인 (2014), "smali: An assembler/disassembler for Android’s dex format", Retrieved November 30 from GSHS, Web site:

https://code.google.com/p/smali/

[10] fangyid...@gmail.com 외 5인 (2012), “json-simple: JSON.simple - A simple Java toolkit for JSON", Retrieved November 30 from GSHS, Web site: https://code.google.com/p/json-simple/

[11] atomicdryad (2010), "signapk: onboard apk signing script for android devices", Retrieved November 30 from GSHS, Web site: https://code.google.com/p/signapk/

[12] Dmitry.S...@gmail.com (2008), "android4me: J2ME port of Google’s Android", Retrieved November 3 from GSHS, Web site: http://code.google.com/p/android4me/downloads/detail?name=AXMLPrinter2.jar