안드로이드 스마트폰에서 앱 설치 정보를 이용한 리패키징 앱 탐지 기법★

전영남* · 안우현**

요 약

최근 안드로이드 스마트폰에서 리패키징을 이용한 악성코드가 급증하고 있다. 리패키징은 이미 배포되고 있는 앱의 내부를 수정한 후 다시 패키징하는 기법이지만, 악성코드 제작자가 기존 앱에 악성코드를 삽입하여 배포할 때 흔히 사용되고 있다. 하지만, 앱을 제공하는 안드로이드 마켓이 다양하고, 각 마켓에서 제공하는 앱이 매우 많기 때문에 모든 앱을 수집해서 분석하는 것은 불가능하다. 이를 해결하기 위해 본 논문은 RePAD 기법을 제안한다. 이 기법은 사용자의 스마트폰에 탑재된 클라이언트 앱과 원격 서버로 구성되는 시스템이다. 클라이언트는 적은 부하로 사용자가 설치한 앱의 출처와 정보를 추출하여 원격 서버로 전송하고, 서버는 전송된 정보를 바탕으로 앱의 리패키징 여부를 탐지한다. 따라서 리패키징 앱 판별을 위해 앱의 정보를 수집하는 시간과 비용을 줄일 수 있다. 실험을 위해 클라이언트 앱과 원격 서버를 갤럭시탭과 윈도우즈 기반의 PC에 각각 구현하였다. 여러 마켓에서 수집된 앱 중 7 쌍의 앱이 리패키징된 것으로 판정하였고, 갤럭시탭에서 평균 1.9%의 CPU 부하와 최대 3.5M의 메모리 사용량을 보였다.

Detecting Repackaged Applications using the Information of App Installation in Android Smartphones

Young Nam Joun* · Woo Hyun Ahn**

ABSTRACT

In recently years, repackaged malwares are becoming increased rapidly in Android smartphones. The repacakging is a technique to disassemble an app in a market, modify its source code, and then re-assemble the code, so that it is commonly used to make malwares by inserting malicious code in an app. However, it is impossible to collect all the apps in many android markets including too many apps. To solve the problem, we propose RePAD (RePackaged App Detector) scheme that is composed of a client and a remote server. In the smartphone-side, the client extracts the information of an app with low CPU overhead when a user installs the app. The remote server analyzes the information to decide whether the app is repackaged or not. Thus, the scheme reduces the time and cost to decide whether apps are repackaged. For the experiments, the client and server are implemented as an app on Galaxy TAB and PC respectively. We indicated that seven pairs of apps among ones collected in official and unofficial market are repackaged. Furthermore, RePAD only increases the average of CPU overhead of 1.9% and the maximum memory usage of 3.5 MB in Galaxy TAB.

Key words : Android, Application Security, Malware, Repackaging, Multi-Market

접수일(2012년 8월 22일), 수정일(1차: 2012년 8월 29일), 게재확정일(2012년 8월 30일)

★ 이 논문은 2012년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임 (No.2012-0003399).

★ 이 논문은 2012년도 광운대학교 교내학술연구비 지원에 의해 연구되었음.

* 광운대학교 컴퓨터과학과

** 광운대학교 컴퓨터소프트웨어학과

1. 서 론

최근 안드로이드 악성 앱이 급격하게 증가하고 있다. 2011년 한 해 동안 안드로이드 악성 앱은 3,300% 이상의 증가율을 보였다[1]. 이런 악성 앱의 급증은 리패키징(repackaging) 때문이며, 2010년 8월부터 2011년 10월까지 발견된 안드로이드 악성 앱 샘플 중 약 86%가 리패키징 기반의 악성 앱으로 확인되었다[2]. 따라서 리패키징은 안드로이드 스마트폰에서 가장 위협적인 기술이다.

리패키징이란 이미 배포되고 있는 실행파일을 디컴파일하여 내부 코드를 수정한 후, 다시 패키징하는 제작 기법이다. 대부분의 리패키징 악성 앱은 패키지 이름과 앱 이름을 기존 앱과 동일하게 만들어 배포하기 때문에 사용자는 리패키징 앱을 구별할 수 없다. 안티바이러스 업체에서 리패키징 앱을 선별하여 악성 코드 유무를 판별하고자 해도 모든 마켓에 분포된 모든 앱을 수집하여 분석하는 것은 불가능하다. 이는 공식 또는 비공식 마켓이 매우 다양하고, 각 마켓을 통해 배포되는 앱의 수가 수십만 개 이상이기 때문이다.

본 논문은 여러 사용자가 설치한 앱의 정보를 사용하여, 다양한 마켓에 분포된 리패키징 앱을 탐색하는 RePAD(RePackaged App Detector) 기법을 제안한다. 이 기법은 스마트폰에서 동작하는 클라이언트와 원격 서버로 구성된다. 클라이언트는 사용자의 터치 이벤트를 모니터링하여 어떤 마켓 앱이 실행되었는지 감지하고, 그 마켓을 통해 어떤 앱이 다운로드되어 설치되었는지에 대한 앱의 정보를 서버로 전송한다. 서버는 전송된 앱의 정보를 바탕으로 앱의 리패키징 여부를 결정한다. 결국 안티바이러스 업체는 RePAD 기법으로 결정된 리패키징 앱에 대해 악성행위 탐지 기술들[3,4]을 사용하여 악성 앱을 효율적으로 탐지할 수 있다.

RePAD의 장점은 다음과 같다. 첫째, 스마트폰의 CPU와 메모리를 적게 사용하면서 앱의 출처를 파악한다. 기존 안드로이드 API로는 앱의 출처를 파악할 수 없기 때문에 RePAD는 터치 이벤트를 모니터링하여 앱의 출처를 파악한다. 둘째, RePAD는 다양한 마켓에 존재하는 리패키징 앱을 수집할 때 요구되는 인력과 시간을 절약할 수 있다.

실험 검증을 위해 클라이언트와 서버를 구축하였다. 서버는 윈도우즈 7 운영체제 기반의 PC에서 데이터를 저장하기 위해 MySQL DB를 사용하였다. 클라이언트는 안드로이드 3.2 버전이 설치된 갤럭시탭에 구현하였다. 그 실험 결과, RePAD는 갤럭시탭에서 평균 3.5 MB의 메모리를 사용하였고, CPU 부하를 최대 1.9%를 증가시켰다.

본 논문의 구성은 다음과 같다. 2장에서는 기존 안드로이드의 보안정책과 관련연구에 대해 설명한다. 3장에서는 RePAD의 원리와 구현에 대해 설명한다. 4장에서는 실험과 결과에 대해 설명한다. 마지막 5장에서는 결론을 정리한다.

2. 배경 지식

안드로이드 앱은 컴포넌트(component) 단위로 구성된다[5]. 안드로이드의 컴포넌트에는 Activity, Service, Broadcast Receiver, Content Provider가 있다. Activity는 UI와 관련된 코드로 구성되며 사용자와 직접 통신하는 컴포넌트이다. Service는 UI가 없이 백그라운드로 동작하는 컴포넌트이다. Broadcast Receiver는 시스템에서 발생하는 Broadcast 메시지를 받는 컴포넌트이고, Content Provider는 앱의 데이터를 공유하기 위해 데이터 집합을 만드는 컴포넌트이다. 특정 컴포넌트를 활성화시키기 위해서는 Intent 메시지를 이용해야 한다. Intent는 컴포넌트에서 처리할 작업에 대해 명시한 메시지이고, Activity, Service, Broadcast Receiver를 활성화 할 수 있다.

안드로이드의 기본적인 보안정책은 두 가지로 구성된다[6]. 첫째, 최하위 계층인 리눅스 운영체제에서 제공되는 보안정책이다. 이 정책은 안드로이드의 모든 앱에 고유한 사용자 ID를 부여하고, 각 ID에 독립된 환경을 제공한다. 둘째, 안드로이드 미들웨어에서 제공되는 보안정책이다. 이 정책은 앱을 구성하는 컴포넌트 간의 통신이 시작되면 정의된 권한을 준수하는지 감시한다. 하지만 이들 보안 정책은 악성 앱의 많은 비중을 차지하는 리패키징 악성 앱을 방어하는데 한계가 있다. 이는 리패키징 악성 앱이 컴포넌트 간의 통신을 비정상적으로 수행하거나, 권한이 없는 행동을 하는 것이 아니라 보안 정책의 범위 내에서 개인정보

등을 유출하는 악성 행위를 하기 때문이다.

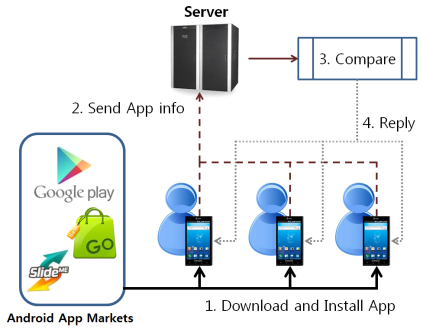
안드로이드 마켓은 구글 공식 마켓 이외의 많은 비공식 마켓들이 존재하며, 사용자들은 보통 하나 이상의 마켓을 사용한다. 따라서 많은 마켓에 악성 앱을 등록하면 그만큼 많은 사용자들에게 노출될 수 있다. 현재 리패키징 악성 앱은 모든 악성 앱 중 약 86%를 차지하며, 이 중 대부분은 앱 이름과 패키지 이름이 기존 앱과 동일하다. 리패키징은 안드로이드 스마트폰의 가장 위협적인 기술임에 틀림없다.

리패키징 악성 앱의 피해를 막기 위한 연구가 진행되었다. 리패키징 판별 시스템인 DroidMOSS[7]는 보통 리패키징을 통해 적은 양의 코드가 변경되는 것을 이용하여 판별한다. 이 기법은 Fuzzy-Hashing 알고리즘을 사용하여 원본 앱과 대상 앱의 실행파일의 해시 값을 추출하고, 이들 해시 값의 거리를 측정하여 리패키징을 판별한다. 하지만, 이 기법은 많은 마켓에 분포된 매우 많은 앱을 수집해야지만 그 탐지 능력을 발휘할 수 있다. 하지만 기하급수적으로 증가하는 앱을 수집한다는 것은 불가능하다. 또한 리패키징 악성 앱의 대부분이 원본 앱과 동일한 이름을 가짐에도 불구하고 모든 앱의 실행파일을 분석하는 것은 매우 큰 시간적, 하드웨어적인 비용을 증가시킬 수 있다.

3. RePAD 기법

3.1 기본 원리

다중 마켓 환경에서 모든 앱에 대해 리패키징 여부를 판단하는 것은 불가능하지만, 그 판단 여부는 악성 앱의 피해를 줄이기 위해 필요하다. 본 논문은 사용자가 설치한 앱의 정보를 이용하여 리패키징 앱을 탐지하는 기법인 RePAD를 제안한다. 이 기법은 스마트폰의 안티바이러스 앱에 탑재될 수 있는 클라이언트와 원격 서버로 구성된다. 사용자가 앱을 설치할 때 클라이언트는 해당 앱의 정보를 서버로 전송하고, 서버는 정보를 이용하여 리패키징을 판별한다. 이런 리패키징 판별은 보안업체로 하여금 다양한 마켓에 분포된 모든 앱이 아닌 리패키징 앱에 대해서만 보안 검사를 수행할 수 있는 힌트를 줌으로써 효율적으로 악성 앱을 탐지할 수 있도록 한다.



(그림 1) RePAD의 개요

(그림 1)은 RePAD의 기본적인 동작을 나타내며, 그 동작은 네 개의 단계로 구성된다. 첫째, 사용자는 마켓에서 앱을 다운받아 설치한다. 둘째, 클라이언트는 사용자가 설치한 앱의 정보를 서버로 전송한다. 앱의 정보는 앱의 이름, 패키지 이름, 버전, 해시, 마켓이름으로 구성된다. 셋째, 서버는 클라이언트에서 받은 정보를 저장하고, 이전에 저장된 앱의 정보와 함께 비교하여 리패키징을 판단한 후 클라이언트로 메시지를 보낸다. 이때 패키지 이름은 같지만, 마켓 출처와 실행파일의 시그너처 또는 해시 값이 다르면 비교된 앱들이 리패키징 되었다고 판단한다. 해시 값이 다르다는 것은 앱 내부가 수정되었다는 것을 의미하기 때문이다. 마지막으로 클라이언트는 서버의 메시지를 사용자에게 보여준다.

RePAD는 다음과 같은 장점들을 가진다. 첫째, 클라이언트와 서버가 상호 통신하는 자동화된 시스템이다. 이는 안티바이러스 업체에게 앱 정보를 수집하는데 드는 비용을 절감할 수 있도록 한다. 둘째, 사용자가 설치하는 앱에 대해서만 판별하기 때문에 효율적이다. 많은 마켓에서 모든 앱 정보의 수집은 현실적으로 불가능하다. 아울러 마켓에 있는 앱들 중 다운로드되지 않는 앱은 고려 대상이 되지 않음에도 불구하고 이들의 정보를 획득하는 것은 비효율적이다. 하지만 RePAD는 사용자가 설치하는 앱의 정보를 이용하여 판별하기 때문에 시간과 비용을 절약할 수 있다.

3.2 클라이언트 설계

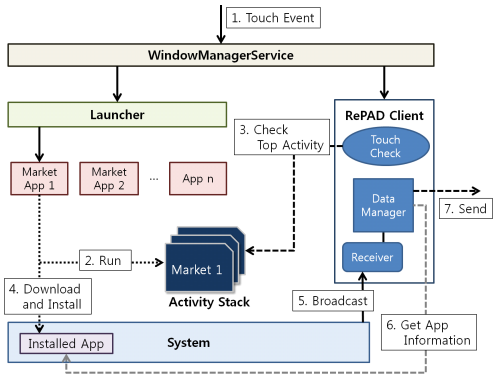
클라이언트를 설계할 때 다음과 같은 두 가지가 고

려되어야 한다. 첫째, 작은 시스템 자원을 사용하여 앱의 정보를 획득하여야 한다. 스마트폰은 비교적 작은 사양의 자원으로 동작하기 때문에 클라이언트가 CPU와 메모리를 많이 사용한다면 사용자에게 불편함을 줄 수 있다. 둘째, 앱이 어떤 마켓에서 다운로드 되었는지 그 출처를 파악해야 한다. 안드로이드 플랫폼에서 제공하는 API를 사용하면 공식 마켓에서 설치했는지 여부는 알 수 있으나, 그 외의 다른 마켓에 대해서는 구분할 수 없다.

CPU 부하를 최소화화기 위해 폴링(polling) 방식이 아닌 터치 이벤트 모니터링 기법을 사용한다. 안드로이드 스마트폰은 터치를 이용하여 앱을 실행하기 때문에 터치 이벤트는 앱의 실행과 밀접한 관계가 있다. 이 점에 근거하여, RePAD는 사용자의 터치 이벤트를 모니터링하여 마켓 앱의 실행을 감지한다. 그 다음으로 설치된 앱의 정보 추출을 위해서는 앱의 설치가 완료됨을 파악할 필요가 있다. 이를 위해 RePAD는 Broadcast Receiver를 이용한다. 이것을 이용하면 앱의 설치가 완료된 후에 시스템 내부에서 발생하는 Broadcast 메시지를 받을 수 있고, 이 메시지를 통해 설치된 앱의 정보를 추출할 수 있다. 터치 이벤트 모니터링 기법을 이용한 RePAD의 구현은 수시로 마켓의 실행 확인과 새로운 앱의 설치를 확인해야 하는 폴링 방식보다 자원을 적게 소모한다.

(그림 2)는 클라이언트의 구조를 나타내며, 크게 TouchCheck와 Receiver로 구성된다. TouchCheck는 안드로이드의 Service 컴포넌트로서 백그라운드에서 터치 이벤트를 모니터링하고 마켓 앱의 실행을 감지한다. 안드로이드 컴포넌트인 Receiver는 앱 설치 완료 후 시스템에서 발생하는 Broadcast 메시지를 받는다. DataManager는 앱에서 데이터를 추출하고, 서버로 전송하는 모듈이다.

클라이언트의 동작 과정은 다음과 같다. 사용자가 바탕화면 앱인 Launcher에서 앱을 다운로드 받고자 하는 마켓 앱을 터치하면, WindowManagerService가 터치 이벤트를 Launcher와 RePAD 클라이언트로 전달한다(①). Launcher는 터치된 마켓 앱을 실행하고(②), TouchCheck는 제일 최근에 실행된 Activity를 분석하여 마켓 앱이 실행되었는지 확인한다(③). 만약 마켓 앱이 실행되었다면, 이 마켓 앱의 이름을 일시 저



(그림 2) RePAD 클라이언트의 구조와 동작

장한다. 사용자가 실행된 해당 마켓 앱에서 임의의 앱을 다운로드하고 설치하게 되면(④), 시스템은 앱 설치 완료를 감지할 수 있는 모든 Receiver에게 설치 완료 메시지를 보낸다(⑤). RePAD 클라이언트에 구현된 Receiver는 설치 완료 메시지를 받아 해당 앱의 패키지 이름을 추출하고, DataManager 모듈을 실행시켜 설치된 앱의 정보를 추출한다(⑥). 추출이 완료되면 DataManager 모듈은 앞서 추출된 마켓 앱의 이름과 추출된 정보를 서버로 전송한다(⑦).

3.3. 서버 설계

RePAD 서버는 클라이언트에서 정보를 받아 저장하고, 저장한 정보로 앱의 리패키징 여부를 결정한다. 서버는 많은 앱의 정보를 저장하고 이들 정보에 대해 검색 작업을 빈번히 수행한다. 따라서 다량의 앱의 정보를 효율적으로 관리하기 위해 MySQL DB를 사용한다. RePAD의 클라이언트 앱은 외부 서버로 데이터를 전송하기 위해서 HTTP 프로토콜 관련 API(예로, HttpPost)를 사용한다. 서버는 클라이언트와의 HTTP 기반의 통신을 위해 Apache 웹서버를 탑재하였다. 그리고 MySQL DB를 제어하는 데이터 삽입, 삭제 및 검색 등의 명령어를 PHP 스크립트로 구성하고, 이 스크립트를 실행하여 클라이언트에서 전달된 앱의 정보를 DB에 저장한다.

클라이언트에서 앱의 정보를 받은 서버는 다음과 같은 동작을 수행한다. 첫째, 전송된 정보를 DB에 저장한다. 이때 해당 정보와 동일한 정보가 이미 DB에 존재하면 저장하지 않는다. 둘째, 저장된 앱 정보를 바

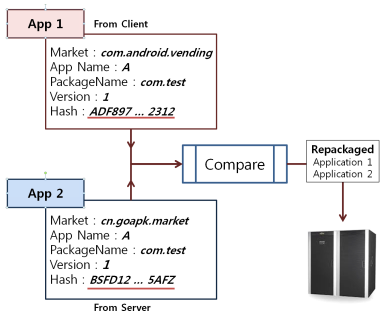
탕으로 리패키징을 판별한다. 해당 앱의 리패키징 여부는 3.4절에서 구체적으로 기술한다.

서버는 리패키징 판별 수행 외에도 리패키징 앱이 악성 앱인지를 탐지하기 위해 리패키징 앱과 그 원본 앱에 대해 다음과 같은 분석을 수행할 수 있다. 첫째, 한 쌍의 앱이 가지는 AndroidManifest 파일에서 권한을 분석하고, 보안 정보 유출 또는 과금을 유발할 때 사용되는 권한이 리패키징 앱에 추가되었는지 확인할 수 있다[4]. 둘째, 한 쌍의 앱에 대한 dex 실행파일을 분석하여 사용자의 정보 유출을 막을 수 있다. 예로 원본 앱에는 없지만 리패키징 앱에 외부와 통신이 가능한 API가 존재하면 사용자의 정보를 외부로 유출할 가능성이 있다. 이런 경우에 사용자에게 유출 가능성에 대한 주의를 주면 피해를 줄일 수 있다.

3.4 리패키징 앱 판별

클라이언트에서 추출한 앱의 이름, 패키지 이름, 버전만으로는 정확하게 리패키징을 판별할 수 없다. 예를 들면, 개발자가 같은 앱을 각기 다른 마켓에 올릴 수 있다. 이런 경우는 출처는 다르고, 패키지 이름과 버전, 앱 이름은 같다고 분석할 수 있다. 하지만 개발자가 코드를 변경했는지 여부는 확인할 수 없기 때문에 리패키징 여부를 정확히 확인할 수 없다. 이를 위해 RePAD는 dex 실행파일의 해시 값을 리패키징 판별에 사용한다. dex 실행파일의 헤더에는 시그너처 필드가 있고, 이 필드는 앱의 코드를 기반으로 SHA-1 알고리즘으로 생성된 해시 값을 저장한다. 리패키징 앱을 만들기 위해 원본 앱의 일부 코드를 수정하기 때문에 리패키징 앱은 원본 앱과 다른 해시 값을 가진다. 그러므로 이름과 버전이 동일하지만 서로 다른 해시 값을 가지는 앱들에 대해서 리패키징되었다고 판단할 수 있다.

RePAD는 리패키징 여부를 판별할 때 앱 이름 외에 동일한 버전을 가지는 앱들을 서로 비교한다. 이는 동일한 앱이어도 버전이 다르다면 해시 값이 달라질 수 있기 때문이다. (그림 3)은 앱의 리패키징 여부를 판단하는 예를 보여준다. App 1은 클라이언트에서 전송받은 앱의 정보이고, App 2는 DB 서버가 관리하는 앱의 정보이다. 클라이언트에서 App 1의 정보가 서버로 전송되면, 서버는 DB에서 앱 이름과 버전이 동일한 App



(그림 3) 리패키징 판별의 예제

2의 정보와 비교한다. 두 앱의 해시 값과 마켓이 다르기 때문에 리패키징 앱으로 판단한다.

4. 실 험

4.1 실험 환경

RePAD의 리패키징 판별 기능 및 성능 검증을 위해 다음과 같은 환경에서 실험하였다. 클라이언트는 안드로이드 3.2 버전이 적용된 갤럭시탭을 사용했으며, 서버는 Intel i5-2500, 3.30 GHz CPU, 4GB 메모리를 갖춘 시스템이다. 서버에 탑재되는 웹서버와 DB를 위해 Apache와 MySQL을 각각 사용하였다.

리패키징 판별의 검증을 위해 RePAD를 이용하여 공식 마켓(170개)과 비공식 마켓인 SlideME(45개), go APK(45개)에서 총 260개의 앱을 수집하였다. 또한 클라이언트의 성능 측정은 안드로이드의 쉘에서 기본적으로 제공하는 유틸리티인 top을 이용하여 측정하였다. top은 실시간으로 CPU 점유율을 비롯한 프로세스의 정보를 확인할 수 있는 유틸리티이다.

4.2 실험 결과

리패키징 판별에 대한 실험을 위해 수집 된 260개의 앱 중 앱 이름과 패키지 이름이 같은 13쌍의 앱을 확인하였으며, 이 중 7쌍(14개)이 리패키징으로 판별하였다. <표 1>은 리패키징으로 판별된 앱들의 정보를 나타낸다. 7쌍의 앱은 앱 이름과 패키지 이름, 버전이 같지만 출처와 해시가 일치하지 않아서 리패키징 되었음을 확인할 수 있다.

<표 1> 리패키징 앱 (M1-구글 공식 마켓, M2-SlideME 비공식 마켓, M3-goAPK 비공식 마켓)

앱 이름	패키지 이름	버전	수집된 마켓	카테고리
DrumHead	com.mobileamusements.DrumHead	3.0.0	M1, M2	음악
Kids Cleanup	com.russpuppy.kidscleanup	1.0	M1, M2	아동
iReader	com.chaozh.iReaderFree	1.7.2.0	M1, M3	문서
Magic Piano	com.smule.magicpiano	1.0	M1, M3	음악
Tiny Music Quiz	pt.bwdrake.amquiz	1.2	M1, M2	게임
Viber	com.viber.voip	2.1.6.632	M1, M2	통신
WhatsAPP	com.whatsapp	2.7.9450	M2, M3	통신

클라이언트의 성능 측정은 RePAD 클라이언트가 활성화 때의 CPU 부하와 메모리 사용량을 측정하였다. 갤럭시탭에서 클라이언트는 평균 1.9%의 CPU 부하와 최대 3.5MB의 메모리 사용량을 보였다.

5. 결 론

본 논문은 안드로이드의 다중 마켓 환경에서 효과적으로 리패키징 된 앱을 수집하기 위한 방법인 RePAD를 제안했다. RePAD는 적은 부하로 스마트폰에 설치되는 앱의 출처와 정보를 추출하여 저장하고, 저장된 정보를 바탕으로 리패키징을 판별한다. 실험을 통해 스마트폰에서 RePAD의 동작 시 평균 1.9% CPU 부하와 최대 3.5MB의 메모리 사용을 보여 부하가 적다는 것을 입증하였고, 정확한 리패키징 판별을 보여주었다. 따라서 RePAD는 현재 안드로이드 환경에 무리 없이 적용될 수 있을 것으로 예상한다.

참고문헌

[1] ZdNet, “Report : Android Malware Up 3,325% in 2011”, http://www.zdnet.com/blog/hardware/ report -android-malware-up-3325-in-2011/18449.

[2] Y. Zhou and X. Jiang, “Dissecting Android Malware: Characterization and Evolution”, In Proc. of the 33rd IEEE Symposium on Security and Privacy, May 2012.

[3] E. Chin, A. Felt, K. Greenwood, and D. Wagner, “Analyzing Inter-Application Commun- ication in Android”, 9th Annual International Conference on Mobile Systems, Applications, and Services, June 2011.

[4] 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 Proc. of the 19th Network and Distributed System Security Symposium, February 2012.

[5] M. Murphy, “Beginning Android 2”, Apress, 2010.

[6] W. Enck, M. Ongtang, and P. McDaniel, “Understanding Android Security”, IEEE Security and Privacy Magazine, Vol. 7, No. 1, pp. 10-17, January 2009.

[7] W. Zhou, Y. Zhou, X. Jiang, and P. Ning, “Detecting Repackaged Smartphone Application Third-Party Android Marketplaces”, In Proc. of the 2nd ACM Conference on Data and Application Security and Privacy, February 2012.

[저 자 소 개]

전 영 남 (Young Nam Joun)

2011년 광운대학교 컴퓨터소프트웨어 학과(학사)
2011년~ 현재 광운대학교 컴퓨터과 학과(석사 재학중)

email : youngnam@kw.ac.kr

안 우 현 (Woo Hyun Ahn)

1996년 경북대학교 전자공학과 (학사)
1998년 KAIST 전기 및 전자공학과(석사)
2003년 KAIST 전자전산학과(박사)
2003년~2005년 삼성전자 기술총괄 소프트웨어연구소 책임연구원
2006년 3월~2011년 8월 광운대학교 컴퓨터소프트웨어학과 조교수
2011년 9월~현재 광운대학교 컴퓨터소프트웨어학과 부교수

email : whahn@kw.ac.kr