情報保護學會論文誌
第 21 卷 第 1 號, 2011. 2

안드로이드 앱 악성행위 탐지를 위한 분석 기법 연구

심 원 태,[1†] 김 종 명,[1‡] 류 재 철,[2] 노 봉 남[3]
[1]한국인터넷진흥원, [2]충남대학교 전기정보통신공학부, [3]전남대학교 시스템보안연구센터

Android Application Analysis Method for Malicious Activity Detection

WonTae Sim,[1†] Jong-Myoung Kim,[1‡] Jae-Cheol Ryou,[2] Bongnam Noh[3]
[1]Korea Internet & Security Agency, [2]Division of Electrical and Computer Engineering, Chungnam National University, [3]System Security Research Center, Chonnam National University

요 약

스마트폰 시장이 급속하게 성장함에 따라 보안위협도 동시에 증가하고 있다. 가장 큰 스마트폰 보안위협 중 하나는 스마트폰 마켓에 안전성이 검증되지 않은 어플리케이션이 유통되고 있다는 점이다. 안드로이드 마켓의 경우 어플리케이션 검증을 수행하지 않아 악성 어플리케이션이 유통되고 있는 상황이다. 이와 같이 마켓을 통해 유포되는 악성 어플리케이션에 대응하기 위해서는 안드로이드 어플리케이션의 악성 행위 여부를 탐지할 수 있는 기술이 필요하다. 본 논문에서는 안드로이드 어플리케이션의 악성행위를 탐지할 수 있는 분석 방법을 제안하고 구현내용을 소개하고자 한다.

ABSTRACT

Due to the rapid growth of smartphone market, the security threats are also increased. One of the smartphone security threats is that unverified applications are distributed on the smartphone market. In the case of Andoroid market, Google have no Application Approval Process that can detect malicious android application so many malicious android applications are distributed in the Android market. To reduce this security threat, it is essential the skill to detect the malicious activities of application. In this paper, we propose the android application analysis method for malicious activity detection and we introduce the implementation of our method which can automatically analyze the android application.

Keywords: Smartphone, Android, Application, Malicious, Detection

I. 서 론

최근 국내 스마트폰 시장이 급속하게 성장함에 따라 스마트폰 보안위협도 함께 증가하고 있다. 이러한 상황을 반영하듯 국내에서도 최초로 모바일악성코드(WinCE\TerDial)가 발견되어 스마트폰 보안에 대한 관심이 높아진 상황이다[1].

스마트폰 보안위협 중 가장 큰 위협은 안전성이 검증되지 않은 스마트폰 어플리케이션이 어플리케이션 마켓에 유통되고 있다는 점이다. 아이폰의 경우 앱스토어에 어플리케이션을 유통하기 전에 어플리케이션을 검증하여 침해사고 위협을 최소화하고 있지만 안드로이드 구글 마켓의 경우 검증절차가 없어 논란이 되고 있다. 특히 최근 정보유출로 문제가 되었던 Jackeey Wallpaper 및 Tapsnake의 경우는 안드로이드 마켓에 버젓이 유통되어 많은 이용자가 감염이 된 사례이다[2][3].

이러한 안드로이드 악성 어플리케이션의 피해를 최소화하기 위해서는 안드로이드 마켓에 안전성 검증 절

접수일(2011년 1월 7일), 게재확정일(2011년 2월 14일)
† 주저자, wtsim@kisa.or.kr
‡ 교신저자, jmkim@kisa.or.kr

〔표 1〕 안드로이드 앱 구조

파일	설명
AndroidManifest.xml	어플리케이션에 대한 설명 및 실행권한 등의 정보를 지니는 XML 파일
Classes.dex	Dalvik 가상머신에서 동작하는 바이너리 실행 파일
/res	컴파일되지 않은 리소스 파일(아이콘, 이미지, 음악 등)들이 포함된 폴더
/META-INF	프로그램 자체 정보, 배포시 인증서로 서명한 내용, SHA-1을 이용한 APK 파일내 폴더, 파일에 대한 해쉬값
resources.arsc	컴파일된 리소스파일

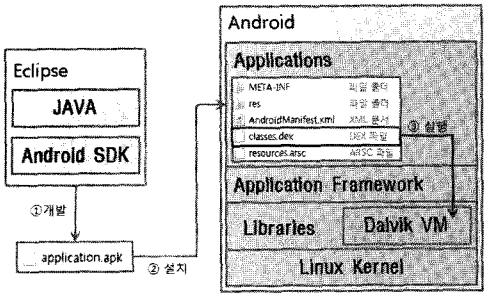
차를 도입하거나 이용자가 다운받은 어플리케이션의 악성여부를 확인해주는 서비스가 필요하다. 본 논문에서는 이와 같은 서비스가 가능하도록 안드로이드 어플리케이션의 악성행위 여부를 확인하여 악성 어플리케이션을 탐지하는 안드로이드 앱 악성행위 탐지를 위한 분석 기법에 대해 제안하고자 한다. 이를 위해 II장에서는 관련연구로 안드로이드 어플리케이션 분석 방법에 대해 살펴보고 III장에서 본 논문에서 제안하는 안드로이드 앱 악성행위 탐지를 위한 분석 기법에 대해 설명한 후 IV장에서 제안하는 분석 기법의 특징을 소개한다. V장에서는 직접 구현한 내용을 소개하고 마지막으로 VI장에서는 향후연구에 대해 논의한다.

II. 관련연구

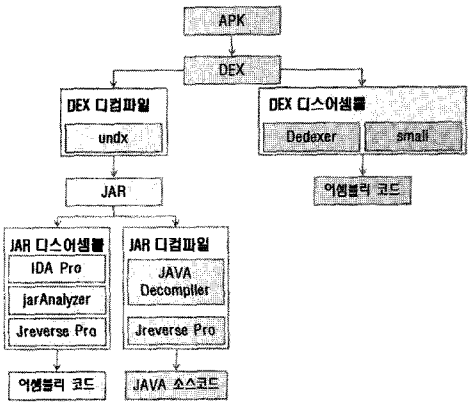
안드로이드 앱 분석 방법 소개에 앞서 간략하게 안드로이드 앱의 구조 및 동작에 대해 설명한 후, 안드로이드 정적 분석 및 동적 분석에 대해 소개한다.

2.1 안드로이드 앱 구조 및 동작

안드로이드 앱은 확장자가 '.apk'인 압축파일(ZIP)이다. APK는 Android Package의 약자로 안드로이드 플랫폼에 어플리케이션 설치를 위한 패키지, 즉 압축파일을 의미하며 〔표 1〕과 같은 파일로 구성된다. 어플리케이션 실행 권한 등의 정보를 포함하고 있는 AndroidManifest.xml 파일과 어플리케이션 실행파일인 .dex 파일, 그 외에 리소스 파일인 resource.arsc, res(folder), META-INF(folder)로



〔그림 1〕 안드로이드 어플리케이션 생성/설치/실행 과정



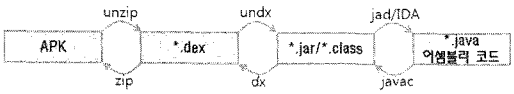
〔그림 2〕 안드로이드 앱 정적분석 로드맵

구성된다.

APK 파일은 〔그림 1〕과 같이 JAVA 기반의 안드로이드 개발 환경을 통해 생성되며, 이를 안드로이드 환경에 설치하면 압축이 풀리게 된다. 설치된 파일 중 Classes.dex 파일은 JAVA 기반의 Class파일을 DEX(Dalvik Executable)라는 형태로 변환한 것으로 JAVA와 마찬가지로 가상머신인 Dalvik에서 동작한다. Dalvik 가상머신은 리눅스 커널 위에서 적은 메모리 요구사항에 최적화되어 DEX 포맷으로 변환된 Class를 실행한다.

2.2 안드로이드 앱 정적분석

안드로이드 앱 정적분석이란 DEX 파일을 디스어셈블 혹은 디컴파일 하여 얻어진 코드상에서 어플리케이션의 행위를 분석하는 방법을 의미한다. 〔그림 2〕는 안드로이드 앱을 정적분석하는 방법을 분류한 그림이다. 정적분석방법은 크게 DEX 파일을 디컴파일하는 방법과 DEX 파일을 디스어셈블하는 방법으로 나뉜다.



〔그림 3〕 DEX 디컴파일을 이용한 분석

〔표 2〕 JAVA 디컴파일 및 디스어셈블 도구

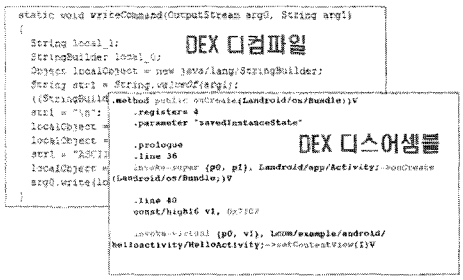
도구	설명
JAVA Decompiler〔5〕	자바 언어의 압축파일 형태인 JAR 파일에서 실제 구현된 자바 코드와 유사한 형태의 코드를 생성하는 디컴파일 도구
IDA Pro〔6〕	가장 널리 이용되는 역공학 전문 도구로 JAR 파일 또한 디스어셈블 가능
Jreverse Pro〔7〕	JAVA로 작성된 JAVA 디컴파일 및 디스어셈블 도구
jarAnalyzer〔8〕	JAR 파일에 포함되어 있는 라이브러리들을 분석해 주는 도구로 바이너리 분석이 아닌 단순 라이브러리만의 출력이 필요한 경우 매우 유용

2.2.1 DEX 디컴파일을 이용한 분석

DEX 디컴파일을 이용한 분석 방법은 〔그림 3〕과 같다. 먼저 APK 파일의 압축을 해제하여 DEX 파일을 추출한 후, DEX 파일을 JAVA 클래스파일(.jar)로 변환해주는 디컴파일 도구인 undx〔4〕를 이용하여 JAR 파일을 생성한다. 마지막으로 생성된 JAR 파일을 다시 디스어셈블 및 디컴파일 과정을 통해 어셈블리 코드 또는 JAVA 소스코드를 확보하고 이를 분석한다. 이때 JAR 파일을 디스어셈블 및 디컴파일하기 위해 사용되는 도구는 〔표 2〕와 같다. 이중 JAVA Decompiler를 이용할 경우 JAVA 소스코드와 거의 유사한 코드를 확보할 수 있어 분석이 용이하다는 장점이 존재한다. 그러나 실험결과 undx를 이용해 JAR 파일을 확보하는 과정에서 일부 코드가 누락되는 단점이 발견되어 이에 대한 보완이 필요하다.

2.2.2 DEX 디스어셈블을 이용한 분석

DEX 디스어셈블을 이용한 분석 방법은 APK 파일로부터 추출된 DEX 파일에 대하여 직접 디스어셈블을 수행한다. 이 방법은 DEX 디컴파일을 이용해 분석하는 방법과는 달리 DEX 파일을 직접 디스어셈블하여 분석하기 때문에 분석과정이 최소화되고 디컴파일 과정에서 발생하는 코드 누락이 최소화된다는 장

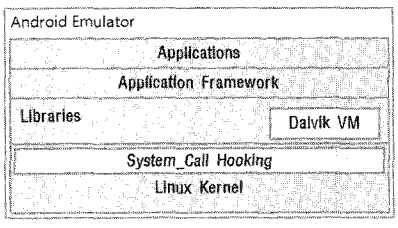


〔그림 4〕 DEX 디컴파일 및 디스어셈블 결과 비교

점이 존재한다. 그러나 상대적으로 디컴파일을 이용할 경우 소스코드 수준까지 복원된다는 점을 감안한다면 〔그림 4〕와 같이 상대적으로 코드 가독성이 떨어진다는 단점이 있다. 대표적인 DEX 디스어셈블 도구로는 smali〔9〕와 dedexer〔10〕가 존재한다. 두 도구 모두 DEX 파일을 어셈블리 코드로 변환시켜준다.

2.3 안드로이드 앱 동적 분석

안드로이드 앱 동적분석은 안드로이드 어플리케이션을 실행하여 호출되는 API를 모니터링하고 이를 분석하는 방법을 의미한다. 현재까지 공개된 도구는 없으나 최근 Android Application Sandbox (AASandbox)라는 분석방법이 공개되었다〔11〕. 엄격하게 이야기하면 AASandbox는 안드로이드 앱 정적분석 방법과 동적분석 방법을 모두 활용한다. AASandbox는 정적분석 방법으로 smali를 이용하여 디스어셈블을 수행한 후 악성 API 패턴을 탐지하고 있다. 동적분석 방법에는 〔그림 5〕와 같이 안드로이드 에뮬레이터를 이용하고 있으며, 안드로이드 라이브러리 계층에서 리눅스 커널로 요청하는 시스템 콜을 후킹하여 호출되는 API를 출력한다. AASandbox 논문에 따르면 리눅스 커널상에서 리눅스의 시스템 콜을 후킹하는 이유는 안드로이드 어플리케이션을 통해 안드로이드 시스템에 리눅스 악성코드가 감염되는 경우〔12〕를 탐지하기 위해서이다.



〔그림 5〕 AASandbox 시스템 콜 후킹

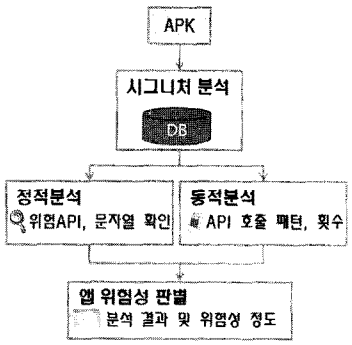
그러나 AASandbox의 경우 정적분석 결과로 얻어진 어셈블리 코드로부터 호출되는 API 패턴을 분석하고 있으나 어셈블리 코드는 클래스단위로 분석결과가 도출되기 때문에 프로그램 실행 순서를 반영할 수 없어 API 패턴 분석에 한계점을 지닌다. 또한 동적분석시에도 리눅스 커널에서 요청되는 시스템 콜을 후킹하기 때문에 실제 안드로이드 어플리케이션이 호출하는 함수(클래스, 메소드)는 파악할 수 없다. 특히 앞으로 악성 어플리케이션의 일부 코드가 패킹(실행압축, 즉 프로그램이 실행될 때 압축된 코드를 풀어 실행하는 형태)된 형태로 유포될 경우 정적분석을 통해서도 이용되는 API를 확인할 수 없다.

III. 안드로이드 앱 악성행위 탐지를 위한 분석 기법

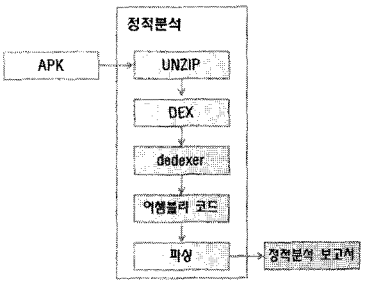
본 논문에서 제안하는 안드로이드 앱 악성행위 탐지를 위한 분석 기법은 [그림 6]과 같다. 크게 1) 시그니처 분석을 통해 이미 탐지된 악성어플리케이션인지를 판단하고, 2) 정적분석을 통해 위험 API 목록 및 이용되는 문자열을 파악한 후, 3) 동적분석하여 호출되는 API 순서를 확인하고 총체적인 어플리케이션 위험성 정도를 제공함으로써 악성 어플리케이션 여부를 판단한다.

3.1 시그니처 분석

시그니처 분석 단계에서는 분석하려는 안드로이드 어플리케이션이 이미 악성으로 판별되었었는지를 1차적으로 확인하는 과정이다. 악성코드으로 분류된 안드로이드 어플리케이션의 시그니처를 DB에 등록하고 비교함으로써 악성 여부를 판별한다. 시그니처 값으로는 MD5, SHA-1 등의 간단한 해쉬값을 사용한다.



[그림 7] 안드로이드 앱 악성행위 탐지



[그림 6] 안드로이드 앱 정적분석 순서

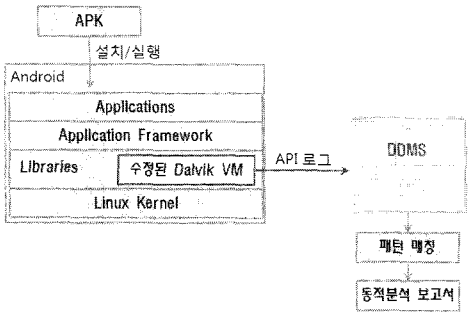
3.2 정적분석

앞서 살펴봤듯이 안드로이드 정적분석 도구는 다양하다. 그 중 본 논문에서 제안하는 분석 기법은 dedexer 도구를 활용한다. DEX 디컴파일을 이용한 도구들은 undx 도구를 활용하는 과정에서 일부 코드가 누락되는 단점이 존재했고 DEX 디스어셈블 도구인 smali는 다수의 실험결과 dedexer에 비해 안정성이 떨어졌기 때문이다.

[그림 7]은 정적분석 순서도이다. APK 파일을 분석하기 위해 압축을 해제하고 DEX 파일을 추출한 후, 디스어셈블 도구인 dedexer를 이용하여 어셈블리 코드를 얻는다. 마지막으로 어셈블리 코드를 파싱하여 정보유출 및 시스템 명령 실행 등 악성행위와 관련

[표 3] 위험 API

API	설명
getDeviceID	IMEI 값을 가져옴
getSubscriverId	IMSI 값을 가져옴
getIccSerialNumber	USIM 시리얼번호 확인
getSimSerialNumber	USIM 시리얼번호 확인
OpenFileOutput	파일을 열고 쓰는 함수
OpenFileInput	파일을 열고 읽는 함수
isWifiEnabled	Wi-Fi 에 접속 여부 확인
getLatitude	GPS 정보를 가져오는 함수
getLongitude	GPS 정보를 가져오는 함수
getContentResolver	저장된 주소록을 열람
getVoiceMailNumber	Voice Mail 번호 열람
openConnection	인터넷 연결 요청
setRequestMethod	인터넷 연결방식 설정
sendTextMessage	SMS 전송
getMessageBody	SMS를 열람
setLatestEventInfo	Notification 쓰기 함수
getInstalledApplications	설치된 어플리케이션 확인
Build$VERSION.RELEASE	제품 버전정보 추출
Build.Model	제품 모델명 추출
getLine1Number	스마트폰 번호를 확인
exec	시스템 명령을 실행



〔그림 8〕 안드로이드 앱 동적분석 순서

된 위험 API와 문자열을 추출한다. 위험 API는 〔표 3〕과 같다. 위험 API 및 문자열을 추출하는 이유는 분석가가 효율적으로 악성 여부를 판단할 수 있도록 돕는데 있다. 아무리 작은 어플리케이션이라 하더라도 어셈블리 코드로 변환하면 분량이 상당히 많기 때문에 분석하는데 시간이 많이 소모된다. 악성행위에 이용되는 API 및 문자열을 중심으로 분석할 수 있도록 어셈블리 코드를 정제함으로써 분석 시간을 단축시킬 수 있다.

3.2 동적분석

동적분석 방법을 표현한 개념도는 〔그림 8〕과 같다. Dalvik 가상머신을 수정하여 호출되는 API의 순서 및 횟수를 파악한 후 악성행위 패턴과 비교·분석하여 어플리케이션의 위험성 정도를 판단하는 것이다. 안드로이드는 소스코드가 공개되어 있기 때문에 수정이 용이할 뿐만 아니라 안드로이드 개발을 위한 로그 함수 및 디버그 모니터 도구(DDMS, Dalvik Debug Monitor System)를 제공하고 있어 호출되는 API를 쉽게 출력할 수 있다.

패턴 매칭은 어플리케이션이 악성행위를 하기 위해 호출해야하는 위험 API들의 순서 및 횟수를 바탕으로 한다. 예를 들어 SMS, 위치정보 등을 수집하는 API가 호출된 후 인터넷 연결 API가 주기적으로 호출된다면 정보유출 악성코드일 가능성이 높다고 판단하는 것이다.

IV. 제안하는 분석 기법의 특징

본 논문에서 제안하는 악성행위 탐지를 위한 분석 기법은 악성코드를 정적분석방법 뿐만 아니라 동적분석방법을 모두 이용하여 안드로이드 앱의 악성행위를 효율적으로 탐지할 수 있도록 한다. 〔표 4〕는 제안하는 분석 기법과 AASandbox를 비교한 표이다.

〔표 4〕 제안하는 분석 기법과 AASandbox의 비교

	제안하는 분석 기법	AASandbox
시그니처 분석	MD5, SHA-1 등의 해쉬함수 이용	없음
정적분석	dedexer를 이용	smali를 이용
동적분석	달빅 머신상에서 안드로이드 API 후킹 및 분석	리눅스 커널에서 리눅스 시스템 콜 후킹 및 분석
	안드로이드에서 리눅스로 감염되는 악성코드 탐지 가능	안드로이드에서 리눅스로 감염되는 악성코드 동적분석 가능
패턴매칭	동적분석을 통해 호출되는 API 순서에 따른 패턴매칭	정적분석을 통해 확보된 어셈블리 코드를 이용한 패턴매칭
	패턴매칭 정확도 보장	패턴매칭에 한계 존재
분석환경	단말기 및 에뮬레이터	에뮬레이터

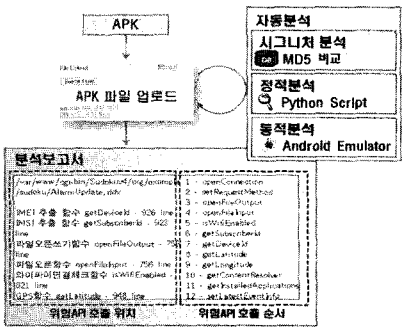
AASandbox의 경우 정적분석을 통해 얻어진 어셈블리 코드를 이용하여 패턴 매칭을 하고 있다. 그러나 어셈블리 코드는 클래스 단위로 표현되기 때문에 실제 어플리케이션의 실행 순서와 달라 패턴 매칭에 한계가 존재 한다. 본 논문에서 제안하는 분석 기법은 Dalvik 가상머신을 수정하여 API를 로깅하기 때문에 호출되는 API 순서를 정확히 파악할 수 있어 패턴 매칭의 정확도를 보장할 수 있다는 장점이 있다.

안드로이드에서 리눅스로 전파되는 리눅스 악성코드 분석을 위해 AASandbox는 리눅스 시스템 콜을 후킹하는 동적분석 방법을 이용하고 있다. 그러나 이 방법은 오히려 Dalvik 가상머신이 호출하는 리눅스 시스템콜과 혼동되어 실제 리눅스로 전파되는 악성코드를 확인하기에 한계가 있다. 본 논문에서는 오히려 리눅스 시스템 명령을 실행하는 안드로이드 API를 위험 API로 정의하고 위험 API 호출을 분석하여 최종 분석보고서에 호출되는 위험 API 및 관련 문자열 등을 제공하여 분석가가 악성행위 여부를 판단할 수 있도록 하여 분석 효율을 극대화하고 있다.

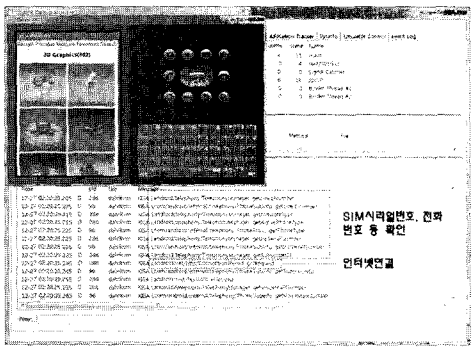
V. 구현

본 논문에서 제안하는 안드로이드 앱 악성행위 탐지를 위한 분석 기법을 자동화하여 〔그림 9〕와 같은 자동분석도구를 개발하였다. 웹을 통해 APK 파일을 제출하면 자동으로 시그니처 분석 및 정적·동적분석을 수행하고 최종적으로 분석 보고서를 출력한다.

시그니처 분석은 MD5를 이용하여 입력되는 APK



〔그림 9〕 안드로이드 앱 자동분석 구현



〔그림 10〕 'Jackeey Wallpaper' 동적분석 화면

를 기존 악성코드와 간단하게 비교할 수 있도록 구현하였으며 정적분석을 위해 파이선 스크립트를 작성하여 DEX 파일 추출부터 어셈블리 코드 파싱까지의 단계를 자동으로 수행하도록 하였다. 스크립트 언어를 이용한 이유는 위험 API 목록 업데이트를 수시로 할 수 있도록 하기 위해서이다. 동적분석의 경우 안드로이드 에뮬레이터 상에서 Dalvik 가상머신을 수정하여 API 호출 로그를 DDMS로 출력시킨 뒤 API 호출 패턴을 비교하도록 구현하였다.

〔그림 10〕은 최근 구글 안드로이드 마켓에서 개인정보유출 문제를 일으켰던 'Jackeey Wallpaper' 어플리케이션의 동적분석 화면이다. 정보수집 및 데이터 전송을 위한 API가 순서대로 호출됨을 확인 할 수 있었으며 정적분석결과를 통해 전송되는 서버의 주소도 추출할 수 있었다.

VI. 결 론

본 논문에서 제안하는 안드로이드 앱 악성행위 탐지를 위한 분석 기법은 정적·동적 분석을 통해 위험 API 및 문자열을 추출하고 API 호출 패턴을 탐지함으로써 효율적으로 어플리케이션의 악성 여부를 판단하는 수단으로 활용될 수 있다.

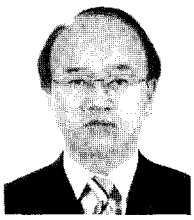
앞으로 다양한 어플리케이션 테스트를 통해 위험 API 및 API 패턴을 확보함으로써 분석 방법의 신뢰성을 높이고 안드로이드 에뮬레이터가 아닌 실제 디바이스에 수정된 Dalvik 가상머신을 적용함으로써 에뮬레이터가 지니는 한계를 극복하고자 한다.

안드로이드의 가장 큰 보안위협은 마켓에 어플리케이션 검증 체계가 존재하지 않다는 점이다. 현재 검증체계는 존재하지 않지만 앞으로 보안위협이 더욱 커지고 실제 피해사례가 많이 발생한다면 언젠가는 어플리케이션 검증 체계를 도입할 것으로 생각된다. 본 논문에서 제안한 안드로이드 앱 악성행위 탐지를 위한 분석 기법은 안드로이드 마켓에서 어플리케이션 검증을 위한 기반 기술로 활용될 수 있을 것이다.

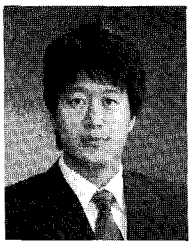
참고문헌

〔1〕 보안뉴스, http://www.boannews.com/media/ view.asp?idx=20547&kind=1
〔2〕 Lookout, http://blog.mylookout.com/2010/07/mobile-application-analysis-blackhat/
〔3〕 Symantec, http://www.symantec.com/connect/blogs/androidostapsnake-watching-your-every-move
〔4〕 undx, http://undx.sourceforge.net
〔5〕 Java Decompilerm, http://java.decompiler.free.fr
〔6〕 IDA Pro, http://hex-rays.com/idapro
〔7〕 Jreverse Pro, http://jreversepro.blogspot.com
〔8〕 JarAnalyzer, http://www.kirkk.com/main/Main/JarAnalyzer
〔9〕 smali, http://code.google.com/p/smali
〔10〕 dedexer, http://dedexer.sourceforge.net
〔11〕 T. Blasing, L. Batyuk and A.D. Schmidt, "An Android Application Sandbox System for Suspicious Software Detection," Malicious and Unwanted Software (MALWARE), Oct 2010.
〔12〕 A. Shabtai, Y. Fledel, and Y. Elovici. Securing androidpowered mobile devices using selinux. IEEE Security and Privacy, 99(PrePrints), 2009.

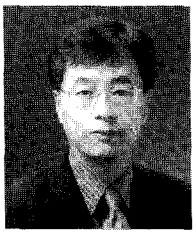
〈著者紹介〉



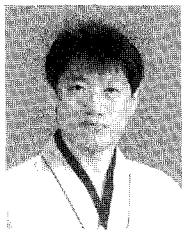
심 원 태 (WonTae Sin) 정회원
1986년 2월: 서울대학교 계산통계학과 졸업
1988년 2월: KAIST 전산학과 석사
1987년~2000년: (주)데이콤 부장
2000년~2003년: (주)인젠 연구소장
2003년~현재: 한국인터넷진흥원 단장
〈관심분야〉 시스템 및 네트워크 보안, 클라우드 보안



김 종 명 (Jong-Myoung Kim) 정회원
2007년 2월: 성균관대학교 정보통신공학 졸업
2009년 2월: 성균관대학교 전자전기컴퓨터공 석사
2009년 1월~현재: 한국인터넷진흥원 주임
〈관심분야〉 정보보호, 모바일 보안, 악성코드 분석



류 재 철 (Jae-Cheol Ryou) 정회원
1988년 5월: Iowa State University 전산학과 석사
1990년 12월: Northwestern University 전산학과 박사
1991년~현재: 충남대학교 전기정보통신공학부 교수
2003년~현재: 인터넷침해대응기술연구센터장
〈관심분야〉 정보보호, 네트워크보안, 암호학, 보안프로토콜



노 봉 남 (Bongnam Noh) 정회원
1978년 2월: 전남대학교 수학교육학과 학사
1982년 2월: KAIST 대학원 전산학과 석사
1994년 2월: 전북대학교 대학원 전산과 박사
1983년~현재: 전남대학교 전자컴퓨터공학부 교수
〈관심분야〉 컴퓨터와 네트워크 보안, 정보보호시스템