Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.26, NO.4, Aug. 2016 ISSN 2288-2715(Online)
http://dx.doi.org/10.13089/JKIISC.2016.26.4.895

DEX 파일을 이용한 효율적인 안드로이드 변종 악성코드 탐지 기술*

박 동 혁,† 명 의 정, 윤 주 범‡
세종대학교

Efficient Detection of Android Mutant Malwares Using the DEX file*

Dong-Hyeok Park,† Eui-Jung Myeong, Joobeom Yun‡
Sejong University

요 약

스마트폰 보급률이 증가하여 이용자 수가 증가함에 따라 이를 노린 보안 위협도 증가하고 있다. 특히, 안드로이드 스마트폰의 경우 국내에서 약 85%에 달하는 점유율을 보이고 있으며 안드로이드 플랫폼 특성상 리패키징이 용이하여 이를 노린 리패키징 악성코드가 꾸준히 증가하고 있다. 안드로이드 플랫폼에서는 이러한 악성코드를 방지하기 위해 다양한 탐지 기법을 제안하였지만, 정적 분석의 경우 리패키징 악성코드 탐지가 쉽지 않으며 동적 분석의 경우 안드로이드 스마트폰 자체에서 동작하기 어려운 측면이 있다. 본 논문에서는 리패키징 악성코드의 코드 재사용 특징을 이용하여 안드로이드 애플리케이션에서 DEX 파일을 추출해 역공학 과정을 거치지 않고 DEX 파일에 기록된 클래스 이름과 메소드 이름 같은 특징으로 악성코드를 정적 분석하는 방법과 이를 이용하여 리패키징 악성코드를 보다 효율적으로 탐지하는 방법을 제안한다.

ABSTRACT

Smart phone distribution rate has been rising and it's security threat also has been rising. Especially Android smart phone reaches nearly 85% of domestic share. Since repackaging on android smart phone is relatively easy, the number of re-packaged malwares has shown steady increase. While many detection techniques have been proposed in order to prevent malwares, it is not easy to detect re-packaged malwares by static analysis and it is also difficult to operate dynamic analysis in android smart phone. Static analysis proposed in this paper features code reuse of repackaged malwares. We extracted DEX files from android applications and performed static analysis using class names and method names. This process doesn't not include reverse engineering, so it is possible to detect malwares efficiently.

Keywords: Android, Malware, Repackage, Static Analysis, Lightweight, DEX File

I. 서 론

최근 스마트폰의 시장이 활성화 되면서 스마트폰의 보급률이 증가하고 있다. 국내 스마트폰 보급률은 2014년 3월 79.5%에서 2015년 3월 83.0%로 3.5% 증가하였다[1]. 또한 스마트폰에 탑재되는 플랫폼의 경우 Google 사의 안드로이드 플랫폼(android platform)이 84.11%의 점유율로 가장 높은 비율을 보였다[2]. 하지만 안드로이드 스마트폰 이용률이 증가함에 따라 안드로이드 스마트폰을

Received(03. 22. 2016), Modified(07. 06. 2016), Accepted(07. 08. 2016)
* 이 논문은 2015년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No.2015R1C1A1A02036511)
† 주저자, skypia0906@gmail.com
‡ 교신저자, jbyun@sejong.ac.kr(Corresponding author)

노린 보안 위협 또한 증가하고 있다. 2014년 발견된 모바일 악성코드 가운데 99%가 안드로이드 스마트폰을 노린 공격[3]이며 이에 따라 다양한 스마트폰 이용자 중 안드로이드 스마트폰 이용자의 피해 또한 가장 높다.

안드로이드 애플리케이션은 주로 자바(java)를 이용해 작성되며 컴파일(compile), 패킹(packing), 코드사인(code-sign) 과정을 통해 APK(Android application Package) 파일을 생성하고 이를 이용해 스마트폰 단말기에 애플리케이션을 설치한다. 보통 정상적인 경우 구글 플레이 스토어(Google Play Store) 또는 각 통신사 앱 마켓에서 설치할 수 있다. 하지만 안드로이드 애플리케이션은 위와 같은 공식 애플리케이션 저장소 이외에 PC 또는 웹에서 APK 파일을 다운로드하여 직접 설치할 수 있으며, 비공식 저장소를 경유한 다운로드 과정을 통해 악성코드가 유입될 수 있다.

안드로이드 악성코드의 제작 방법에는 다양한 방법이 존재하며 특히 리패키징(repackaging) 기법이 주로 사용된다. 리패키징 기법은 정상 애플리케이션에 일부 악성 기능을 추가하여 정상 애플리케이션처럼 위장하거나 기존 안티 바이러스 시스템을 우회하기 위해 기존 악성코드를 일부 수정하고 이를 다시 다양한 경로를 통해 배포한다. 리패키징 기법을 통한 악성코드 제작 방법은 APK 파일의 생성 과정을 역순으로 진행하여 언패킹(unpacking)과 디컴파일(decompile) 과정을 통해 원본의 소스코드를 복원한 뒤 악성 행위를 수행하는 코드를 삽입하고 이후 APK 생성 과정을 재수행하여 악성코드를 생성한다[4]. 특히 리패키징 기법을 거친 악성코드는 해시(hash) 값과 같은 파일 시그니처(signature)를 통해 악성코드를 탐지하는 기존 안티 바이러스 시스템 같은 보안 솔루션의 탐지를 회피하기 때문에 그 위험성은 더욱 높다.

Trend Micro 사에서 진행한 최근 연구 결과에 따르면 Google Play에 등록된 비즈니스, 미디어, 비디오, 게임 등 다양한 범주의 상위 50개 애플리케이션 가운데 80% 가까이 리패키징 기법을 통한 변종 애플리케이션이 있다고 밝혔으며[5], Andorid Malware Genome Project에서 수집한 1260개의 악성코드 샘플 가운데 리패키징된 악성코드의 비율은 86%로 여러 악성코드 유형 비율 중 가장 높았다[6]. Ahnlab 또한 리패키징 기법을 이용한 안드로이드 악성코드가 2011년 처음 발견된 이후 계속 증가하며 다양한 경로로 배포되고 있다고 밝혔다[4].

리패키징 유형의 악성코드는 악성코드의 제작 시간 및 제작비용 절감, 안티 바이러스 시스템 우회, 유사 악성 행위 반복 등의 목적으로 제작되며[7] 정상 애플리케이션처럼 위장하여 일반 사용자가 의심 없이 사용하도록 유도한다. 이러한 이유로 현재 안드로이드 악성코드 유형 중 리패키징 기법을 이용한 악성코드가 높은 비율로 발생하고 있다. 현재 난독화나 NDK(Native Development Kit)를 이용한 애플리케이션 개발 방법으로 정상 애플리케이션의 리패키징 기법을 이용한 악성코드 제작을 어렵게 하고 있지만 완벽한 해결책은 아니다.

따라서 본 논문에서는 안드로이드 플랫폼에서 수행되는 악성코드와 리패키징 기법이 적용된 변종 악성코드의 탐지를 위한 기술을 제안한다. 본 논문에서는 제안하는 탐지 방법은 APK 파일에 포함된 DEX 파일을 추출하여 DEX 파일에 기록된 클래스(class), 메소드(method) 이름 등의 정보를 통해 DEX 파일의 역공학 과정을 거치지 않고 안드로이드 악성코드를 탐지할 수 있는 경량화된 변종 안드로이드 악성코드 탐지 기술을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 안드로이드 악성코드 탐지 및 분석에 관한 기존 연구를 살펴보며, 3장에서는 DEX 파일의 구조와 논문에서 제안하는 변종 악성코드 탐지 기법에 대해 설명한다. 4장에서는 실제 배포된 악성코드를 제안한 탐지 기법을 이용하여 탐지 성능을 평가하며, 마지막으로 5장에서는 결론을 맺는다.

II. 관련 연구

본 장에서는 안드로이드 악성코드 탐지에 관한 기존 연구들을 정적 분석, 동적 분석, 유사도 기반 분석으로 분류하여 살펴본다.

2.1 정적 분석

안드로이드 악성코드에 대한 정적 분석은 안드로이드 플랫폼 수정이나 다른 기법의 적용 없이 악성코드의 특징만으로 탐지할 수 있어 분석에 필요한 시간과 비용이 적다는 장점이 있다[8]. 하지만 악성코드에 존재하는 고유 시그니처 또는 특정 문자열을 기반으로 탐지하기 때문에 이를 우회하고자 악성코드 제작자가 코드의 일부분을 수정하여 탐지를 회피할 수

있다는 단점이 존재한다[8]. 특히 안드로이드 애플리케이션의 경우 리패키징이 용이하여 변종 악성코드를 탐지하기 어렵다.

기존 안드로이드 악성코드 정적 분석에 관한 연구는 위와 같은 정적 분석의 단점을 보완하고자 APK 파일에 포함된 DEX 파일을 추출하고 디컴파일 또는 디스어셈블(disassemble) 과정을 통해 소스코드를 복원한다. 이후, 사용된 API(Application Program Interface)를 분석하여 악성코드를 탐지하는 방법이다[9]. 하지만 이는 악성 행위로 의심되는 API를 미리 선정해야 하며 원래의 소스코드로 완벽히 복원되지 않고, 복원 과정 또한 단말 상에서 수행되기 어려운 단점이 존재한다. 이를 보완하고자 최근에는 기계 학습을 이용하여 악성 API 학습 후 악성코드 탐지에 이용하기도 하지만 악성 API 학습에 있어 많은 양의 정보를 필요로 하며 또한 분석에 있어 상당한 지연 시간이 소요된다.

2.2 동적 분석

기존 안드로이드 악성코드 동적 분석에 관한 연구로는 주로 안드로이드 플랫폼 수정을 통한 분석과 후킹(hooking) 기법을 이용해 사용되는 API를 관찰하는 분석 방법이 있다.

2.2.1 안드로이드 플랫폼 수정을 이용한 탐지

안드로이드 악성코드 동적 분석을 위해 기존 안드로이드 플랫폼을 수정하여 악성코드 동적 분석을 수행한다. 주로 안드로이드 플랫폼에서 애플리케이션이 실제로 동작하는 DVM(Dalvik Virtual Machine)의 코드를 수정하여 사용되는 API를 추적하거나 로그를 남겨 악성 애플리케이션의 행위를 탐지한다[9]. 하지만 해당 기법은 안드로이드의 동작 과정을 상세히 파악하여야 하며, 안드로이드 플랫폼을 수정해야하는 단점이 존재한다.

2.2.2 안드로이드 API 후킹을 이용한 탐지

안드로이드 악성코드 동적 분석을 위해 안드로이드 플랫폼 구조 중 리눅스 커널 계층에서 사용되는 API를 후킹하여 악성코드 동적 분석을 수행한다. 후킹 기법을 적용하기 위해 LKM(Loadable Kernel Module)[10] 또는 공유 라이브러리 파일을 삽입하여 안드로이드 플랫폼에서 발생되는 API를 모니터링하며, 이에 대한 결과로 악성코드를 탐지한다[11]. 하지만 LKM 기법을 이용해 후킹을 적용하여 악성코드를 탐지하는 방법은 최근 안드로이드 버전 정책에 따라 개발자가 제작한 모듈을 삽입할 수 없어 임의로 안드로이드 커널 버전을 낮춰야 하는 단점이 존재한다. 또한 공유 라이브러리 파일을 삽입하는 방법은 안드로이드의 플랫폼을 수정하지 않아도 되는 장점이 있지만 스마트폰을 루팅(rooting)하여 시스템의 최고 권한을 획득해야 한다. 또한 안드로이드의 하드웨어 특성상 하나의 프로세스가 메모리 자원을 많이 차지할 수 없기 때문에 후킹을 적용할 수 있는 API가 비교적 적고 후킹 기법 자체에 메모리 점유율이 높아 시스템에 부하되는 오버헤드가 상당하다.

2.3 유사도 기반 분석

기존 유사도를 기반으로 한 안드로이드 악성코드 탐지에 관한 연구로는 애플리케이션 소스코드 상에 포함된 클래스 정보를 추출하여 유사도를 측정하는 방법과 스마트폰에서 애플리케이션을 실행 시 발생되는 이벤트를 수집하여 악성 유무를 판별하는 방법이 있다.

2.3.1 클래스 정보 유사도 기반 탐지

유사 클래스 정보 기반 악성 앱 탐지 기법[12]에서는 악성코드 탐지를 위해 달빅(Dalvik) 바이트코드 명령어 추출, 시그니처 정의, 악성 판단, 시그니처 추가 정의 과정을 거친다. 해당 기법에서는 악성 애플리케이션 간 서로 유사한 클래스를 갖는 점을 이용하여 탐지하는 방법이 본 논문과 유사하다. 하지만 해당 방법은 악성 클래스 정보를 추출하기 위해 애플리케이션을 디스어셈블 하여 달빅 바이트 코드 명령어에 대한 빈도수를 추출한 후, 악성 여부를 판단한다. 본 논문에서는 디스어셈블과 같은 역공학 과정을 거치지 않고 DEX 파일 내의 특징으로 시그니처 비교뿐만 아니라 기존 악성코드와 유사한 변종 악성코드를 탐지하여 해당 방법과 차이점을 두었다.

2.3.2 이벤트 유사도 기반 탐지

안드로이드 단말기 상에서 실시간 이벤트 유사도를 통한 악성 앱 탐지 방법[13]에서는 정상 애플리

케이션과 악성 애플리케이션에서 발생되는 이벤트를 strace 도구를 이용하여 수집 후 이벤트 집합을 구성한다. 그리고 새로운 애플리케이션을 설치 및 실행하여 발생되는 이벤트를 strace 도구를 이용해 수집 및 미리 구성된 이벤트 집합과 이벤트 유사성을 통해 새로 설치된 애플리케이션의 악성 유무를 판별한다. 해당 방법은 설치되는 애플리케이션에 대해 실시간으로 분석 및 악성 탐지가 가능하지만 strace 도구를 통해 이벤트를 수집하는 과정에서 동적 분석과 마찬가지로 오버헤드가 크며, 정상 애플리케이션과 악성 애플리케이션에서 공통적으로 발생되는 이벤트로 구성된 악성 애플리케이션 경우 탐지율이 저하될 가능성이 있다.

III. 시스템 구성

3.1 DEX 파일 구조

안드로이드 애플리케이션은 통상 자바를 사용하여 개발하며 자바에서의 컴파일 시 생성되는 클래스 파일과 다른 포맷인 DEX 파일을 생성한다. DEX 파일은 안드로이드 애플리케이션이 실행되기 위한 실제 실행 코드가 기록된 파일로 보통 APK 파일 내부에 Classes.dex 파일로 존재한다. DEX 파일은 Header, String IDs, Type IDs, Proto IDs, Field IDs, Method IDs, Class Defs, Data, Link Data 섹션(Section)으로 구성되어 있다 [14][15].

Fig.1.은 DEX 파일의 형태이다. Header 섹션에서는 String IDs, Type IDs 등과 같은 Header 섹션 이후에 나올 섹션에 대한 크기와 위치(Offset) 정보를 표시한다. DEX 파일에서는 사용된 문자열 전체를 String IDs 섹션을 통해 별도로 관리하며 해당 레이아웃은 전체 문자열의 개수와 전체 문자열의 시작 위치를 가리킨다. Type IDs는 소스코드 상의 기록된 함수의 반환 값, 인자 값에 사용된 변수 반환 형식이 나타난 것으로 DEX 파일은 String IDs와 마찬가지로 이들을 별도로 관리하고 있다. Proto IDs는 메소드 원형에 대한 정보를 가지며 Filed IDs는 소스코드 상의 패키지 이름, 클래스 이름 등의 정보를 표현하며 위와 마찬가지로 DEX 파일에서 별도로 관리하고 있다. Method IDs는 소스코드에서 사용된 전체 Method 정보의 개수와 시작위치를 표현하며, 마지막으로 Class

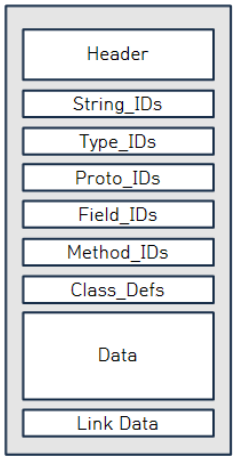


Fig. 1. DEX File Format

Defs에는 애플리케이션에 있는 모든 클래스에 대한 정보가 저장되어 있다. 해당 레이아웃을 통해 클래스의 타입, 상위 클래스 존재 여부, 인터페이스 존재 여부 등의 정보를 파악 할 수 있다[14].

위와 같은 정보를 가진 DEX 파일은 다른 애플리케이션 구성 요소 파일과 함께 APK 파일 형태로 구성된다. 이후 안드로이드 스마트폰에서 애플리케이션이 실행 시 DEX 파일은 실행 파일 형태로 안드로이드 DVM 상에서 동작을 수행한다.

3.2 제안 방법

본 논문에서는 DEX 파일 포맷에 포함된 정보를 바탕으로 애플리케이션의 역공학 과정 없이 악성코드와 위조 및 변조 악성코드를 탐지하는 방법을 제안한다. 2.1절에서 살펴보았듯이 악성코드 제작자는 일반적으로 정상적인 애플리케이션을 리패키징 과정을 통해 악성 행위를 수행하는 코드를 삽입하여 배포하거나 기존 악성코드를 재사용하여 간단한 변형을 통해 백신과 같은 안티 바이러스 시스템의 탐지를 우회한다.

이때, 악성코드 제작 시간과 비용을 줄이기 위해 코드를 재사용하며 사용한 클래스 이름이나 메소드 이름은 변경하지 않은 채 내부 구조만 변경하여 배포하는 경우가 대다수이다. 따라서 3.1절에서 설명한 DEX 파일 포맷에서 각 섹션에서 기록된 값을 조합하여 악성코드 제작에 사용된 클래스 이름과 메소드 이름과 같은 정보를 추출한다. 제안한 방법은 기존 시그니처 기반 백신과 비교하여 정보 수집 방법은 비

슷하지만 추출된 클래스 이름 및 메소드 이름은 미리 구축된 DB와 매핑(Mapping)을 통해 기존의 악성코드 뿐만 아니라 기존 악성코드를 이용한 리패키징 악성코드 또한 탐지 할 수 있다.

Fig.2.는 제안 시스템의 구성도이다. 다운로드 된 APK 파일에서 DEX 파일을 추출한 뒤 먼저 오프셋 0xC에 위치한 SHA-1 시그니처를 Pattern DB에 저장된 기존 악성코드 시그니처와 비교하여 해당 APK 파일의 악성 유무를 판단한다. 이후 SHA-1 패턴 비교에서 탐지하지 못한 변종 악성코드 탐지를 위해 DEX 파일에서 Method IDs 정보에 포함된 class_idx, proto_idx, name_idx 아이템을 이용해 사용된 클래스 이름과 메소드 이름을 추출하여 텍스트 파일에 저장한다. 저장된 텍스트 파일은 Pattern DB와 비교하여 악성 여부를 탐지한다. Fig.3.은 악성 애플리케이션에 제안 시스템을 적용하여 사용된 메소드를 텍스트 파일로 추출한 일부이며, 추출된 텍스트 파일 결과 중 690번째 항목에서 com/b/sm/KB_Card_Psw$2는 클래스 이름을 의미하고 onTextChanged()는 메소드 이름을 의미한다.

Fig.3.과 같이 DEX 파일에서 추출한 사용된 모든 클래스와 메소드 이름 중 악성행위로 의심이 되는 클래스 및 메소드 이름을 선정하여 DB 내에 패턴을 구축한다. 악성행위로 의심되는 클래스 및 메소드 이름은 미리 수집한 악성 APK 중 분석을 통해 자주 사용되는 클래스 및 메소드 이름을 선정하였다. 선정 과정을 거친 후, SHA-1 시그니처 비교와 마찬가지로 구축된 DB 내 정의된 패턴을 KMP 알고리즘

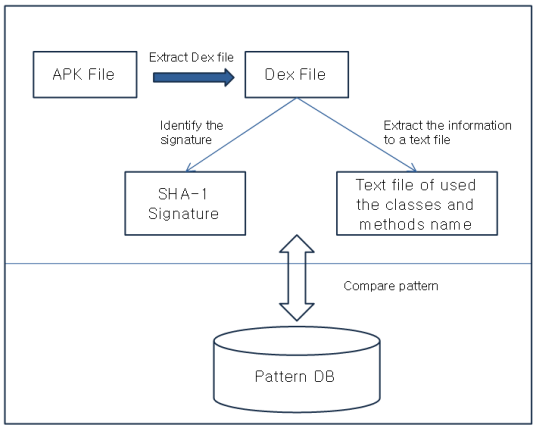


Fig. 2. Proposal System Architecture

```
[0687] Lcom/b/sm/KB_Card_Psw$2;.<init>()
[0688] Lcom/b/sm/KB_Card_Psw$2;.afterTextChanged()
[0689] Lcom/b/sm/KB_Card_Psw$2;.beforeTextChanged()
[0690] Lcom/b/sm/KB_Card_Psw$2;.onTextChanged()
[0691] Lcom/b/sm/KB_Card_Psw$3;.<init>()
[0692] Lcom/b/sm/KB_Card_Psw$3;.afterTextChanged()
[0693] Lcom/b/sm/KB_Card_Psw$3;.beforeTextChanged()
[0694] Lcom/b/sm/KB_Card_Psw$3;.onTextChanged()
[0695] Lcom/b/sm/KB_Card_Psw$4;.<init>()
[0696] Lcom/b/sm/KB_Card_Psw$4;.onCheckedChanged()
[0697] Lcom/b/sm/KB_Card_Psw$5;.<init>()
[0698] Lcom/b/sm/KB_Card_Psw$5;.onCheckedChanged()
```

Fig. 3. Extracted Classes and Methods

(Knuth Morris Pratt algorithm)[16]을 이용하여 매칭 후 해당 APK 파일의 악성 여부를 판단한다. 따라서 제안 방법은 기존 역공학 과정을 거쳐 추출된 API를 분석하는 방법과 비교하여 경량화된 성능을 갖는다. 또한 SHA-1 시그니처 및 특정 문자열 매핑 방법과 비교하여 코드의 일부분을 수정하는 리패키징 악성코드를 탐지할 수 있는 장점이 있다. 단, DEX 파일에 자체에 난독화 또는 암호화가 적용된 사례는 본 논문의 범위를 벗어난다.

IV. 제안 기법 탐지 성능

본 장에서는 제안한 기법을 실제 안드로이드 악성 애플리케이션을 대상으로 실험하여 제안 기법의 성능을 측정한다.

성능 측정에는 실제 배포된 346개의 안드로이드 악성 애플리케이션 중 본 논문에서 다루지 않는 DEX 파일 자체에 난독화 및 암호화 기법이 적용된 악성 애플리케이션을 제외한 200개의 악성 애플리케이션으로 탐지 성능을 측정하였다. 안드로이드 악성코드 탐지는 각 APK 파일마다 포함된 DEX 파일 안에 유일하게 존재하는 SHA-1 시그니처와 DEX 파일에서 클래스 및 메소드 이름을 추출한 텍스트 파일에서 의심되는 클래스 및 메소드 이름을 선정한 후 이를 패턴화 시켜 DB를 구성한 후, 수집한 악성 애플리케이션과 패턴 비교를 통해 탐지한다.

Fig.4.와 Fig.5.는 실제 배포된 스미싱 애플리케이션인 vertu_jp.apk를 제안 시스템에 테스트한 결과이며 Fig.6.과 Fig.7.은 위 애플리케이션의 변종인 vertu_kr.apk를 테스트한 결과이다. Fig. 5.를 보면 vertu_jp.apk는 SHA-1 패턴 비교를 통해 탐지가 되었지만 Fig.7.의 vertu_kr.apk는 SHA-1 패턴 비교를 통해 탐지가 되지 않았다. 두 애플리케이션은 수행하는 악성 기능은 동일하지만

```
[0020] Lcom/vertu/jp/MainActivity;.<init>()
[0021] Lcom/vertu/jp/MainActivity;.getSystemService()
[0022] Lcom/vertu/jp/MainActivity;.getWindow()
[0023] Lcom/vertu/jp/MainActivity;.onCreate()
[0024] Lcom/vertu/jp/MainActivity;.setContentView()
[0025] Lcom/vertu/jp/R$attr;.<init>()
[0026] Lcom/vertu/jp/R$drawable;.<init>()
[0027] Lcom/vertu/jp/R$id;.<init>()
[0028] Lcom/vertu/jp/R$layout;.<init>()
[0029] Lcom/vertu/jp/R$string;.<init>()
[0030] Lcom/vertu/jp/R;.<init>()
[0031] Lcom/vertu/jp/catchsms2;.<init>()
[0032] Lcom/vertu/jp/catchsms2;.abortBroadcast()
[0033] Lcom/vertu/jp/catchsms2;.onReceive()
```

Fig. 4. Extracted Classes Methods of vertu_jp.apk

```
F:₩>apk_static_malware_analysis.py vertu.jp.apk
DB Pettern Analysis : Trojan-Spy/Android.SmsCatch.A
Static Huristic Pettern Analysis : Malware
```

Fig. 5. Detection Result of vertu_jp.apk

```
[0020] Lcom/vertu/kr/MainActivity;.<init>()
[0021] Lcom/vertu/kr/MainActivity;.getSystemService()
[0022] Lcom/vertu/kr/MainActivity;.getWindow()
[0023] Lcom/vertu/kr/MainActivity;.onCreate()
[0024] Lcom/vertu/kr/MainActivity;.setContentView()
[0025] Lcom/vertu/kr/R$attr;.<init>()
[0026] Lcom/vertu/kr/R$drawable;.<init>()
[0027] Lcom/vertu/kr/R$id;.<init>()
[0028] Lcom/vertu/kr/R$layout;.<init>()
[0029] Lcom/vertu/kr/R$string;.<init>()
[0030] Lcom/vertu/kr/R;.<init>()
[0031] Lcom/vertu/kr/catchsms2;.<init>()
[0032] Lcom/vertu/kr/catchsms2;.abortBroadcast()
[0033] Lcom/vertu/kr/catchsms2;.onReceive()
```

Fig. 6. Extracted Classes Methods of vertu_kr.apk

```
F:₩>apk_static_malware_analysis.py vertu.kr.apk
DB Pettern Analysis : None
Static Huristic Pettern Analysis : Malware
```

Fig. 7. Detection Result of vertu_kr.apk

SHA-1 시그니처가 다르기 때문에 기존 정적 분석으로는 탐지하기 어렵다. 본 논문에서 제시하는 방법을 통해 두 애플리케이션에서 공통적으로 사용되는 32번째 항목 'catchsms2' 클래스 이름과 'abortBroadcast()' 메소드를 DB 패턴에 포함시켜 vertu_jp.apk의 변종인 vertu_kr.apk를 탐지할 수 있으며 차후에 위의 두 애플리케이션을 리패키징한 다른 악성 애플리케이션에 대해서도 탐지가 가능하다.

Table 1.은 제안 시스템의 탐지 비율을 나타낸 것이다. 단순 SHA-1 시그니처 패턴 탐지율는 74%이며 제안한 방법의 탐지율은 88%이다. SHA-1 패턴 비교 탐지의 경우 안티 바이러스 시스템과 동일한 탐지 방식으로 리패키징 악성 애플리케이션을 탐지하

Table 1. Detection Ratio

	nProtect AV	Proposed Method
Number of malware	200	200
Number of detection	148	176
Detection rate	74%	88%

는데 있어 한계가 있다. 하지만 클래스 및 메소드 이름을 이용한 제안 방법의 경우 악성 애플리케이션 각각의 고유 시그니처에 초점을 맞춘 것이 아닌 정상 애플리케이션에서 악성 행위로 위조되거나 악성 애플리케이션에서 악성 행위를 수행할 클래스 및 메소드 이름에 초점을 맞추었다. 또한 실험에 있어 하나의 악성 애플리케이션 악성 유무를 판단하는데 평균 35.74ms가 소요 되었다. 또한 기존 정적 분석을 통한 악성코드 탐지[17] 방법은 84.3%의 탐지율을 가져 기존의 방법보다 3.7% 높은 탐지율을 가진다.

본 논문에서 제안한 방법은 고유 시그니처를 이용한 탐지보다 위조 및 변조된 리패키징 악성 애플리케이션에 대해서 효과적으로 대처할 수 있으며 역공학 과정을 거치지 않기 때문에 탐지에 있어 보다 경량화된 성능으로 악성 애플리케이션을 탐지할 수 있다.

V. 결 론

안드로이드 악성코드에는 다양한 종류가 존재하며 특히 리패키징 과정을 통한 위조 및 변조 악성코드가 대다수를 차지한다. 리패키징 기법이 적용된 악성코드는 정상 애플리케이션으로 위장하거나 기존 악성코드의 구조를 일부 변경하여 시그니처 기반인 백신과 같은 안티바이러스의 탐지를 우회하기 때문에 그 위험도가 높다. 이와 같은 리패키징 악성코드는 주로 이전에 사용된 코드를 재사용한다는 특징을 가지며 특히 클래스 이름이나 메소드 이름과 같은 구조를 변경하지 않고 내부 구조만 달리하여 제작된다.

따라서 본 논문에서는 리패키징 악성코드의 코드 재사용 특징에 착안하여 기존 시그니처 매핑을 통한 탐지와 동시에 DEX 파일에 존재하는 클래스와 메소드 이름을 추출하여 미리 구축된 DB와 매핑을 통한 탐지 방법을 제안하였다.

기존에 DEX 파일에서 역공학 과정을 통해 API

를 추출, 비교하여 악성 유무를 판별하는 연구가 있었지만 역공학 과정에서 악성 애플리케이션의 소스코드가 완벽히 복원되지 않는 가능성이 존재하며 이를 보완하고자 기계학습 기법을 적용했지만 기계학습에 있어 많은 양의 학습 데이터가 필요하고 오버헤드가 큰 단점이 존재했다.

제안한 방법은 역공학 과정 없이 DEX 파일에 존재하는 특징을 추출하기 때문에 보다 경량화된 성능을 가지고 있다. 하지만 리패키징 된 악성코드가 내부 구조뿐만 아니라 클래스나 메소드 이름까지 변경 시 탐지를 우회 할 가능성이 존재하며 사전에 방대한 양의 DB 구축이 필수적이다. 또한 APK 파일 내에 포함된 DEX 파일 자체에 난독화 또는 암호화 기법의 적용하여 탐지를 우회할 가능성도 존재한다. 향후 위와 같은 문제점을 보완하고자 좀 더 많은 악성 애플리케이션을 분석하여 사용된 클래스와 메소드 정보를 DB로 구축하고 패턴으로 유출하는 연구를 진행할 예정이다. 또한 제안한 방법과 동시에 보다 경량화된 동적 분석에 대한 연구를 함께 진행할 예정이다.

References

〔1〕 Yonhap News, http://www.yonhapnews.co.kr/bulletin/2015/07/07/0200000000AKR20150707175600017.HTML

〔2〕 KISA Report, http://www.ebn.co.kr/news/view/784827

〔3〕 "Cisco 2015 Annual Security Report," http://www.cisco.com/web/offer/gist_ty2_asset/Cisco_2014_ASR.pdf

〔4〕 AhnLab Tech Report, http://www.ahnlab.com/kr/site/securityinfo/secunews/secuNewsView.do?seq=19269

〔5〕 http://blog.trendmicro.com/trendlabs-security-intelligence/a-look-into-repackaged-apps-and-its-role-in-the-mobile-threat-landscape/

〔6〕 Yajin Zhou and Xuian Jiang, "Dissecting Android Malware: Characterization and Evolution," In security and Privacy(SP), 2012 IEEE Symposium on, pp. 95-109. IEEE, May, 2012.

〔7〕 Tae-guen Kim and Eul-gyu Im, "Analysis Method Reuse Code to Detect Variants of Malware," Journal of The Korea Institute of information Security & Cryptology, 24(1), pp. 32-38, Feb. 2014

〔8〕 Moon Hwa Shin, Bo-heung Chung, Yong Sung Jeon, Jung-nyu Kim. "A Survey of Mobile Malware Detection Techniques," 2013 Electronics and Telecommunications Trends, 28(3), pp. 39-46. ETRI, Jun, 2013.

〔9〕 Seung-wook Min, Hyung-jin Cho, Jin-seop Shin and Jae-Cheol Ryou, "Android Malware Analysis and Detection Using Machine Learning," Journal of KIISE : Computing Practices and Letters, 19(2), pp. 95-99, Feb. 2013

〔10〕 Woo-tak Jung, Seung-wook Min and Jae-Cheol Ryou, "System-Level Malware Detection Methods for Android," Proceedings of Symposium of the Korean Institute of communications and Information Sciences, pp. 745-746, Jun. 2013

〔11〕 Yun-sik Jeong, Seong-wook Kang, Seong-je Cho and In-sik Song, "A Kernel-based Monitoring Approach for Analyzing Malicious behavior on Android," Korea Computer Congress, pp. 127-129, Jun. 2013

〔12〕 Jung-tae Kim and Eul-gyu Im, "Malicious Family Detection Based on Andorid Using Similar Class Information," Jounal of Security Engineering, 10(4), pp. 441-454, Aug. 2013

〔13〕 You-joung Ham and Hyung-woo Lee, "Malicious Trojan Horse Application Discrimination Mechanism using Realtime Event Similarity on Android Mobile Device," Journal of Internet Computing and Services. 15(3), pp. 31-43, Jun. 2014

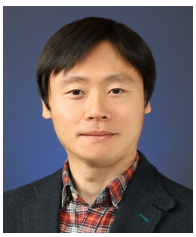
〔14〕 The Android Open Source Project,Dex-Dalvik Executable Format, http://source.android.com/tech/dalvik/dex-format.html

〔15〕 Keith Makan and Scott Alexander-Bown, Android Security Cookbook, Packt Publishing, Jul, 2013
〔16〕 Donald Knuth, James H. Morris, Jr, Vaughan Pratt, "Fast pattern matching in strings", SIAM Journal on Computing, Vol. 6, no. 2, pp. 323-350, 1977.
〔17〕 Hotak Hong, Jinlee Lee, Won Shin and Chunhyon Chang, "Extracting Candidates of Malicious Android Applications using Static Analysis based on Sink", Korea Computer Congress, pp. 1833-1835, Jun. 2014

〈저자소개〉

박 동 혁 (Dong-Hyeok Park) 학생회원
2016년 8월: 세종대학교 컴퓨터공학과 학사 졸업 예정
〈관심분야〉 모바일 보안, 시스템 보안, 임베디드 보안

명 의 정 (Eui-Jung Myeong) 학생회원
2015년 8월: 학점은행제 컴퓨터공학과 학사 졸업
2016년 9월~현재: 세종대학교 정보보호대학원 석사과정
〈관심분야〉 소프트웨어 취약점, Automatic Exploit Generation



윤 주 범 (Joobeom Yun) 종신회원
1999년 2월: 고려대학교 컴퓨터학과 학사
2001년 2월: 서울대학교 컴퓨터공학과 석사
2012년 2월: KAIST 전산학과 박사
2001년 3월~2015년 2월: ETRI 부설연구소 선임연구원
2015년 3월~현재: 세종대학교 정보보호학과 조교수
〈관심분야〉 네트워크 보안, 시스템 보안, 클라우드 컴퓨팅 보안