Journal of the Korea Convergence Society
Vol. 8. No. 4, pp. 25-36, 2017 https://doi.org/10.15207/JKCS.2017.8.4.025

모바일 앱 실행시 커널 계층 이벤트 시퀀스 유사도 측정을 통한 악성 앱 판별 기법

이형우
한신대학교 컴퓨터공학부

Malicious App Discrimination Mechanism by Measuring Sequence Similarity of Kernel Layer Events on Executing Mobile App

Hyung-Woo Lee
School of Computer Engineering, Hanshin University

요 약 최근 스마트폰 사용자가 증가함에 따라 특히 안드로이드 기반 모바일 단말을 대상으로 다양한 어플리케이션들이 개발 및 이용되고 있다. 하지만 악의적인 목적으로 개발된 악성 어플리케이션 또한 3rd Party 오픈 마켓을 통해 배포되고 있으며 모바일 단말 내 사용자의 개인정보 또는 금융정보 등을 외부로 유출하는 등의 피해가 계속적으로 증가하고 있다. 따라서 이를 방지하기 위해서는 안드로이드 기반 모바일 단말 사용자를 대상으로 악성 앱과 정상 앱을 구별할 수 있는 방법이 필요하다. 이에 본 논문에서는 앱 실행시 발생하는 시스템 콜 이벤트를 추출해서 악성 앱을 탐지하는 기존 관련 연구에 대해 분석하였다. 이를 토대로 다수의 모바일 단말에서 앱이 실행되는 과정에서 발생하는 커널 계층 이벤트들에 대한 발생 순서간 유사도 분석을 통해 악성 앱을 판별하는 기법을 제안하였으며 상용 단말을 대상으로 실험 결과를 제시하였다.

• **주제어 :** 안드로이드, 악성 앱, 시스템 콜, 이벤트 유사도, 탐지 메커니즘

Abstract As smartphone users have increased in recent years, various applications have been developed and used especially for Android-based mobile devices. However, malicious applications developed by attackers for malicious purposes are also distributed through 3rd party open markets, and damage such as leakage of personal information or financial information of users in mobile terminals is continuously increasing. Therefore, to prevent this, a method is needed to distinguish malicious apps from normal apps for Android-based mobile terminal users. In this paper, we analyze the existing researches that detect malicious apps by extracting the system call events that occur when the app is executed. Based on this, we propose a technique to identify malicious apps by analyzing the sequence similarity of kernel layer events occurring in the process of running an app on commercial Android mobile devices.

• **Key Words :** Android, malicious apps, system calls, event similarity, detection mechanisms.

이 논문은 한신대학교 학술연구비 지원에 의하여 연구되었음.
*Corresponding Author : 이형우(hwlee@hs.ac.kr)
Received February 21, 2017 Revised March 16, 2017
Accepted April 20, 2017 Published April 28, 2017

1. 서론

최근 안드로이드 기반 모바일 단말 사용자가 지속적으로 증가하면서 시간과 장소에 상관없이 다양한 형태의 모바일 어플리케이션(Mobile Application)을 매우 편리하게 사용할 수 있게 되었다. 하지만 악성 모바일 어플리케이션(Malicious Mobile Application) 또한 상대적으로 급증하고 있어 사용자도 모르게 모바일 단말 내 개인정보 및 금융정보 등이 외부로 불법 유출되는 문제점이 발생하고 있다[1]. 이를 절차적으로 분석하면 사용자는 해당 어플리케이션을 오픈 마켓(Open market)이나 블랙 마켓(Black market)으로부터 다운로드 및 설치한 후 이를 안드로이드 플랫폼 기반 모바일 단말에서 실행하면 과도하고 불법적인 접근권한(permission) 변경을 통해 단말 내 주요 개인정보 또는 금융정보 등이 공격자가 지정한 외부 서버로 유출되는 문제점이 발생하고 있다[2]. 또한 단말내 불특정 형태의 네트워크 장애를 유발하거나 사용자도 모르게 SMS 메시지를 발송하며, 모바일 백신에 검출되지 않도록 앱 내부에 더미(Dummy) 내부 정보를 포함하여 앱이 설치되기도 하며 배터리를 급속하게 소모시키는 등 악의적인 행동을 수행하는 안드로이드 모바일 앱이 지속적으로 증가하고 있다[3]. 따라서 이와 같은 은닉 형태의 악성 앱 확산으로 인한 피해를 사전에 방지하기 위해서는 모바일 단말을 대상으로한 악성 행위를 검출하고 앱 실행시 발생하는 이벤트를 모니터링 하는 기법[4,5] 등이 제시되었다. 하지만 기존 방식인 경우 Dalvik 가상머신 환경에서 구동하는 방식이므로 실제 환경에 제한적으로 적용할 수 있기 때문에 상용 모바일 단말을 대상으로 악성 앱과 정상 앱을 판별할 수 있는 메커니즘이 개발되어야 한다. 이에 본 논문에서는 기존 이벤트 추출 및 악성 앱 탐지와 관련된 기법[6,7,8]에 대한 분석을 수행하여 기존 기법의 문제점을 도출하였으며 개선방안으로 안드로이드 모바일 앱 실행시 발생하는 커널 계층 시스템 콜 이벤트에 대한 유사도(Similarity) 기반 악성 앱 판별 기법을 제시하였다.

구체적으로 상용 모바일 단말을 대상으로 악성 앱과 정상 앱을 구별하기 위해서 본 논문에서는 각 사용자의 모바일 단말냐 설치된 모바일 앱(Mobile App.) 실행시 발생하는 시스템 콜 이벤트(System call event)를 추출한 후 이들 간의 유사도 분석(similarity analysis) 과정을 수행하였다. 정상 앱 및 악성 앱이 각각 실행되었을 때 발생하는 시스템 콜 이벤트를 수집하여 각각의 이벤트 집합에 대한 특성을 도출하고 정상 이벤트와 악성 이벤트 집합내 연관성 분석(Correlation analysis) 과정을 수행하였다. 다수의 모바일 단말에서 발생하는 시스템 콜 이벤트를 추출하고 이를 데이터베이스에 저장하도록 하였다. 데이터베이스 내에 저장된 정상 및 비정상 시스템 콜 이벤트 집합에 대한 유사도 분석 결과를 토대로 임의의 앱 실행시 발생하는 시스템 콜 이벤트를 실시간으로 수집하여 정상/악성 앱 여부를 판별하는 과정을 수행하였다.

본 논문의 구성은 다음과 같다. 2장에서 기존에 연구되어왔던 시스템 콜 이벤트 추출 기반의 악성 앱 판별 기법에 대해 분석한다. 3장에서는 커널 계층 시스템 콜 이벤트 정보를 이용하여 악성 앱을 판별하는 기법에 대해 제시한다. 4장 및 5장에서는 실험 및 유사도 분석 결과를 제시하며, 6장에서 결론을 제시한다.

2. 기존 악성 판별 시스템 분석

3rd Party 오픈 마켓과 비공식적인 블랙 마켓 등을 통해 악성 코드가 삽입된 모바일 어플리케이션이 배포되고 있다. 특히 “Fake Player”, “Geinimi” 등과 같은 악성 앱들이 매변 100% 이상 기하급수적으로 증가하고 있다[9]. 따라서 이에 효율적으로 대응하며 동시에 악성 앱에 대한 판별을 위해 Crowdroid 시스템[6]이 제시되었다.

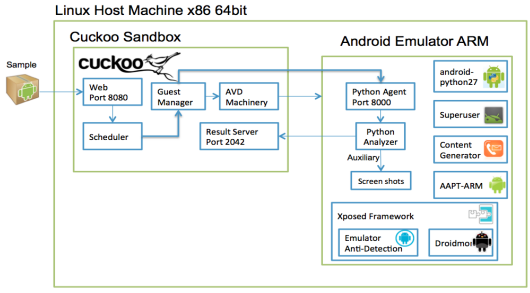
2.1 Crowdroid 시스템 구조 및 판별 메커니즘

Crowdroid 시스템은 안드로이드 어플리케이션이 실행될 경우 발생한 모든 시스템 콜 이벤트들을 추적하고 이를 수집하여 출력 파일로 생성하는 기능을 제공하는 Strace 툴을 사용하여 모바일 단말내 발생하는 이벤트를 수집한다. 따라서 안드로이드 커널내 이벤트 정보를 수집하는 Strace를 이용하여 악성 소프트웨어를 탐지하고 그 결과를 해당 서버로 전송하는 과정을 수행한다. 작동방식을 구체적으로 살펴보면 안드로이드 단말내 시스템 콜 이벤트 데이터 획득, 데이터 처리 및 악성 분석과 탐지의 세 가지 모듈로 구성된 Crowdsourcing 앱을 단말내에 설치한 후 발생하는 시스템 콜 이벤트 정보를 행동기반의 악성 소프트웨어 탐지 서버로 전송하면, 안드로이드 커뮤니티 사용자들에 의해서 수집된 이벤트 정보를 토대로 악성 여부를 판별하게 된다. 악성 앱 판별 서버에서는 Perl 언어 기반 데이터 분석 스크립트(Data analysis script)를 이용하여 Strace로부터 획득된 정보에

대한 클러스터링 과정을 수행한다. K-means 알고리즘 기반 클러스터링 기법을 적용하여 정상 행위와 비정상 행위 이벤트와 앱에 대한 판별 과정을 수행한다.

2.2 CuckooDroid 시스템 구조 및 판별 메커니즘

CuckooDroid[10]는 Android 악성 코드 분석을 위해 아래 [Fig. 1]과 같이 의심스러운 파일에 대한 자동 분석을 위해 개발된 Cuckoo Sandbox의 오픈 소스 소프트웨어를 확장한 것이다. CuckooDroid는 Cuckoo에게 Android 앱 실행 및 분석 기능을 제공한다. 구체적으로 CuckooDroid는 정적 및 동적 APK 검사는 물론 특정 VM 회피 방지 기술, 암호화 키 추출, SSL 검사, API 호출 추적, 기본 동작 서명 및 검출 기능 등을 포함하고 있다. 이 프레임 워크는 커스터마이징이 가능하고 확장이 가능하며 대규모의 기존 Cuckoo 커뮤니티의 리소스를 활용한다.



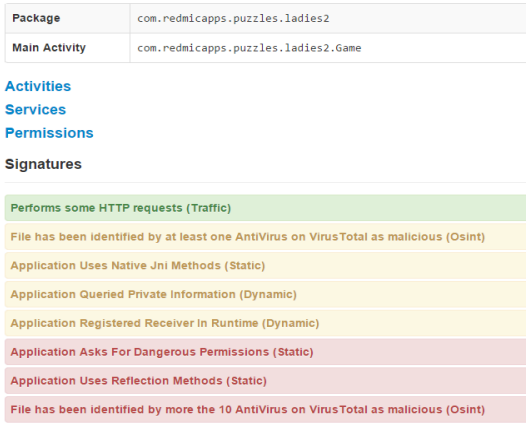
[Fig. 1] CuckooDroid malicious app analysis system internal structure

CuckooDroid 시스템은 가능한 한 대부분의 분석 프로세스를 자동화하는 방법을 사용하였다. Cuckoo 샌드 박스 및 오픈 소스 프로젝트를 기반으로 자동화 된 크로스 플랫폼 에뮬레이션 및 분석 프레임워크를 구축하였다.

2.3 기존 기법의 문제점 분석

Crowdroid 시스템인 경우 수집된 시스템 콜 이벤트에 대한 명확한 판별 및 특성 분류 과정 없이 시스템 콜 이벤트에 대한 유사도 측정 결과만을 이용한다는 문제점이 있다. 또한 CuckooDroid는 악성 앱에 대한 정적/동적 분석 과정을 통해서 악성 앱 내부 API 정보간 연관성(API Correlation) 및 의미 분석(Semantic Analysis) 과정을 수행하지만, 위 [Fig. 1]과 같이 안드로이드 애뮬레이터를 기반으로 구동하는 것이며 실제 상용 단말을 대상으로 실시간으로 발생하는 커널 계층 시스템 콜 이벤트 정보에 대한 수집/분석 과정을 수행하지 못한다는 문제점이 발생하고 있다.

모바일 앱에 대해 자동화된 분석 기능을 수행하여 아래 [Fig. 2]와 같이 해당 앱에 대한 보고서를 작성하는 기능을 제공하는 장점이 있기는 하지만. 분석 대상이 되는 샘플을 Cuckoo Sandbox에 연결한 후에 안드로이드 애뮬레이터 내부에서 발생하는 이벤트를 분석하는 과정을 통해 악성 여부를 판별하는 방식이다. 따라서 최근 발견된 동적 형태의 은닉코드 기반 악성 앱 실행시 발생하는 실시간 이벤트를 효율적으로 검출하지 못한다는 문제점이 있다.



[Fig. 2] Example of CuckooDroid-based malicious app analysis results

3. 제안하는 기법

본 논문에서는 안드로이드 기반 상용 모바일 단말을 대상으로 안드로이드 커널 내에 Strace를 설치해서 어플리케이션이 실행되었을 때 나타나는 시스템 콜 이벤트를 수집 및 분석하였다. 안드로이드 플랫폼을 기반으로하는 다수의 상용 모바일 단말을 대상으로 정상 앱과 악성 앱 실행시 발생하는 시스템 콜 이벤트를 수집하였고, 이를 토대로 악성 어플리케이션을 탐지할 수 있도록 하였다. 또한 상용 모바일 단말에서 실시간으로 실행되는 일반 앱들에 대한 시스템 콜 이벤트들의 발생 및 호출 상호 연관성 및 시퀀스를 분석해서 악성 앱과의 유사도를 분석하였다.

3.1 커널 계층 시스템 콜 이벤트 수집

리눅스 시스템은 어플리케이션, 쉘, 라이브러리 루틴들로 구성되어 있으며 이들은 시스템 콜 방식을 통해 커널로 메시지를 전달한다. 이중 커널은 운영 체제의 하위 수준의 작업을 처리하는 운영 체제의 핵심이며, 커널에 접근할 수 있도록 다양한 시스템 콜 이벤트를 제공한다[4]. 따라서, Strace를 이용해서 악성/정상 앱 실행시 커널 내에서 발생하는 시스템 콜 이벤트를 추출하고, 정상 앱과 악성 앱 내 어떤 시스템 콜 이벤트에 대한 빈도수와 연관성을 측정할 수 있다[3,11,15].

안드로이드 커널 내 Strace의 구조는 크게 user process와 kernel로 나뉜다. user process에 의해 어플리케이션이 실행되며 kernel내에 Strace 코드가 적재되어 어플리케이션에서 kernel로 송수신되는 시스템 콜 이벤트를 획득하게 된다. Strace 모듈에서는 메모리 할당 함수인 malloc을 이용하여 어플리케이션에서 실행되는 시스템 콜 이벤트를 커널 내에서 추출하며 다음과 같은 과정을 수행한다.

① adb shell을 이용하여 Strace를 실행한다.
② Strace -p <process id> 명령어를 실행한 뒤, Stderr Descriptor를 통해서 프로세스 id를 화면에 출력한다.
③ Redirection 연산자를 이용해서 리눅스 커널 내 실행되고 있는 시스템 콜 이벤트를 로그 파일로 추출한다.

3.2 시스템 콜 이벤트 기반 판별 메커니즘

아래 단계와 같이 안드로이드 플랫폼 기반 상용 모바일 단말에 Strace 모듈을 설치한 후에 또한 다수의 정상 앱과 악성 앱에서 실시간으로 발생하는 시스템 콜 이벤트를 수집하였다.

① 안드로이드 기반 상용 모바일 단말 내에 Strace기반 시스템 콜 이벤트 수집 모듈을 설치한다.
② 상용 모바일 단말을 대상으로 이미 알려진 정상 앱 3개($AppNml_{n=1..3}$,)와 악성 앱 3개($AppMal_{m=1..3}$)를 각각 실행한다.
③ 앱이 실행하면서 발생한 시스템 콜 이벤트들을 adb shell을 통해 확인한 후 이를 서버로 전송한다.

그리고 다음 단계로는 수집된 각각의 시스템 콜 이벤트에 대한 비교 분석 과정을 수행하였다. 이를 통해 정상 앱 시스템 콜 이벤트 집합과 악성 앱 시스템 콜 이벤트 집합 각각에 대한 특징을 도출할 수 있었으며, 또한 정상 및 악성 앱 이벤트 발생 유사도를 계량화하여 측정하였다.

다음 메커니즘은 유사도를 측정하는 과정을 나타낸다. 앞서 제시한 것처럼 Strace를 이용해서 커널 계층 이벤트를 수집/분석한 후 악성 앱의 시스템 콜 이벤트의 시퀀스를 찾아서 정상 앱의 시스템 콜 이벤트와 비교해보고 최종적으로 악성 앱 시스템 콜 이벤트와의 유사도를 분석하였다.

① 정상 앱 $AppNml_n$ 집합에서 발생한 시스템 콜 이벤트 SC_i^{AppNml} 집합과 악성 앱 $AppMal_m$ 집합으로부터 발생한 시스템 콜 이벤트 SC_i^{AppMal} 집합을 대상으로 각 집합내 특성에 대해 분석한다.
② 정상 앱 이벤트 집합에서는 모든 앱에서 1회 이상 빈번하게 발생하지 않는 시스템 콜 이벤트를 추출하였고, 악성 앱 이벤트 집합에서는 모든 앱에서 공통적으로 1회 이상 빈번하게 발생하는 시스템 콜 이벤트를 분석해서 두 집합 사이의 교집합에 해당하는 시스템 콜 이벤트를 찾는다.
③ 교집합에 포함된 공통된 시스템 콜 이벤트를 대상으로 악성 앱에서 시스템 콜 이벤트의 시퀀스를 추출하고 이들 시스템 콜 이벤트에 대한 유사도를 분석하였다.

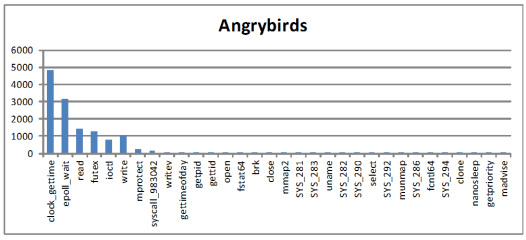
4. 커널 계층 이벤트 수집/분석 결과

본 논문에서는 총 6개의 어플리케이션을 가지고 분석하였다. 정상 앱은 기존에 존재하는 정상 앱들 중에서 많이 사용되는 세 가지 앱(AngryBirds, Astro 및 HNC Viewer)을 선택하여 시스템 콜 함수의 발생 특성을 분석하였고, 악성 앱은 상용 단말에 대한 불법적인 루팅(Rooting) 과정을 수행하여 단말내 개인정보와 금융 정보 등을 외부로 유출하는 기능을 포함한 세 개의 악성 앱(WonBreak, Andoku, WonRD)을 사용하여 시스템 콜 이벤트 특성을 분석하였다. 본 연구에서 사용한 세 개의 악성 앱인 경우 단말내 설치되어 사용될 경우 사용자도 모

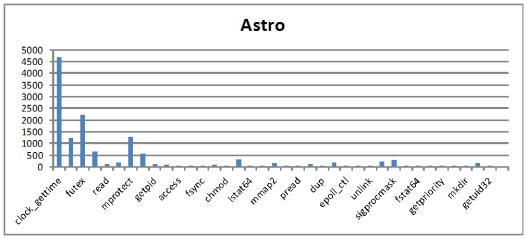
르게 루팅 과정이 수행되어 단말내 개인 정보를 외부 서버로 전송하는 기능을 포함하고 있다.

4.1 정상 앱 시스템 콜 이벤트 특성 분석

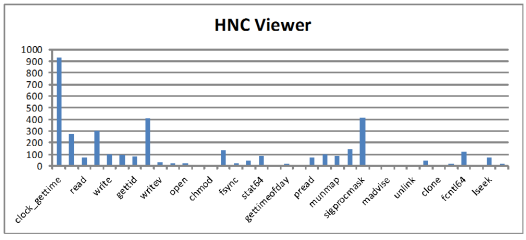
우선 정상 어플리케이션 실행시 발생하는 시스템 콜 이벤트 특성을 분석하면 [Fig. 3]과 같다. 아래 그림과 같이 정상 앱들의 시스템 콜 이벤트 빈도를 분석하면 공통적으로 clock_gettime(), epoll_wait(), read(), futex(), ioctl() 및 write() 시스템 콜 이벤트가 많이 발생하는 것을 확인할 수 있다.



(a) Normal App - Angrybirds



(b) Normal App - Astro File Manager



(c) Normal App - Hancom Viewer

[Fig. 3] Normal App Kernel Layer System Call Event Result

정상 앱에서 발생하는 시스템 콜 이벤트의 주요 특징을 분석하면 다음과 같다.

① clock_gettime()은 안드로이드 커널 내의 시간 측정을 확인할 수 있는 함수로 일반적으로 가장 많이 호출되는 함수라는 것을 알 수 있었다.

② epoll_wait()는 실제 어플리케이션 내에서 이벤트가 발생하는 것을 기다리고 있다가, 이벤트가 발생하면 이벤트와 관련된 정보를 커널에 넘겨주는 역할을 하는 함수이다.

③ read()는 응용 프로그램을 디바이스와 관련된 파일에서 실행했을 때 호출되는 함수로 커널내로 관련 정보를 전송하는 기능을 제공한다.

④ futex()는 공유 메모리 영역에서 동작하므로 프로세스 사이에 공유가 가능하기 때문에 프로세스 간 POSIX 동기화를 제공할 뿐만 아니라 프로세스 사이에서도 futex를 공유할 수 있다.

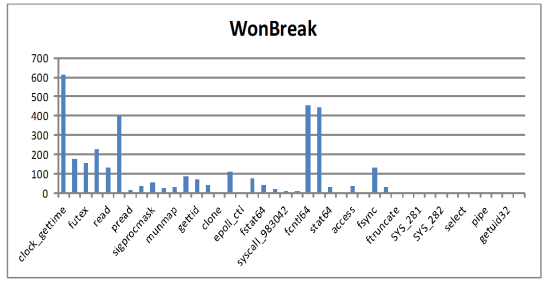
⑤ ioctl()은 특수 파일의 장치 인자를 조절한다. 특히, 문자 특수 파일의 많은 특징적인 동작은 ioctl의 요구에 의해 제어된다.

⑥ write()는 응용 프로그램을 디바이스와 관련된 파일에 write를 수행했을 때 호출되는 함수로 커널로부터 관련 정보를 전송하는 기능을 제공한다[10].

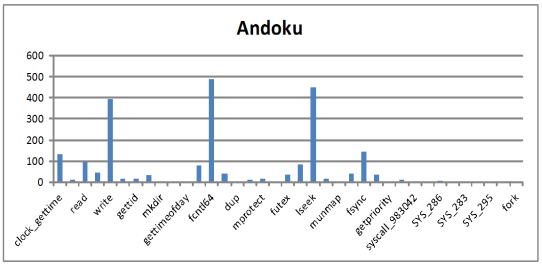
따라서, 정상적인 앱에 대한 시스템 콜 이벤트를 분석한 결과 앱 내부 리소스에 대한 자체적인 접근 및 처리 등의 과정만을 수행할 뿐 대부분 정상적인 형태의 시스템 콜 이벤트가 발생하는 것을 확인할 수 있었으며, 시스템 콜 이벤트 실행 순서 역시 앱 실행 초기에 대부분의 시스템 콜 이벤트가 정상적으로 실행되는 것을 확인할 수 있었다.

4.2 악성 앱 시스템 콜 이벤트 특성 분석

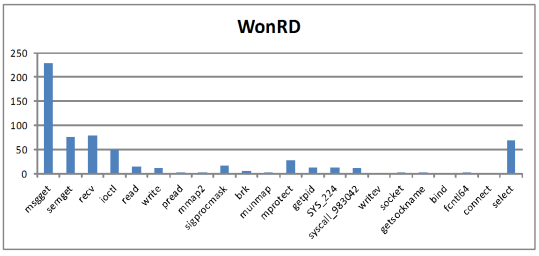
악성 어플리케이션 실행시 발생하는 시스템 콜 이벤트 특성을 분석하면 [Fig. 4]와 같으며 이는 앞서 제시한 정상 앱 실행시 시스템 콜 이벤트 특성([Fig. 3])과는 다른 패턴의 이벤트가 발생한다는 것을 확인할 수 있었다.



(a) Malicious App - Wonbreak



(b) Malicious App - Andoku



(c) Malicious App - WonRD

[Fig. 4] Malicious App Kernel Layer System Call Event Result

[Fig. 4-(a)] 악성 앱 Wonbreak를 실행하였을 경우 발생하는 시스템 콜 이벤트 정보와 [Fig. 4-(b)] 악성 앱 Andoku를 사용하였을 때 발생하는 시스템 콜 이벤트 정보는 사용자들의 개인정보를 탈취하는 기능을 포함하고 있기 때문에 fcntl64(), lseek(), write() 및 fsync() 등의 시스템 콜 이벤트가 많이 호출되는 것을 확인할 수 있었으며 각각의 특징은 다음과 같다.

① fcntl64()는 파일 전체나 파일의 일부를 다른 프로세스에서 사용하는 것을 제한할 수 있도록 해 준다. 파일을 사용하기 위해 open()나 fopen()의 mode를 이용하여 다른 프로세스가 읽기나 쓰기를 제한할 수 있다. 만일 파일 내용을 변경하기 위해서는 파일을 닫았다가 다시 open()하는 과정을 수행하게 된다.

② lseek()는 응용 프로그램의 읽기 및 쓰기를 위해서 파일의 위치를 재지정하는 기능을 하며, 파일의 offset을 파일의 끝으로 설정할 수 있다. 만일 데이터가 나중에 쓰여진다면, 그 사이에 연속적으로 읽은 데이터는 데이터가 실제로 쓰여질 때까지 0 바이트들을 반환한다.

③ write()는 응용 프로그램에서 디바이스 파일에 write를 수행했을 때 호출되는 함수로, 쓰기를 처리한다.

④ fsync()는 파일 내 아직 쓰기가 되지 않은 메모리의 내용을 모두 쓰기가 되도록 한다. sync()가 모든 메모리에 대해서라면 fsync()는 지정된 파일 디스크립터에 대해 진행하여 파일의 내부 상태를 장치와 동기화시키는 역할을 한다.

따라서, 악성 앱을 실행하였을 경우 불법적인 루팅 과정을 수행하면서 내부 리소스에 대한 빈번한 접근을 수행하기 때문에 fcntl64(), lseek() 및 fsync()등과 같은 리소스 접근 이벤트 등이 많이 발생하는 것을 확인할 수 있었다. 이는 정상적인 앱 실행과정과는 다른 형태의 시스템 콜 이벤트가 발생하는 것이며 내부 리소스에 대한 빈번한 접근을 수행하는 것을 확인할 수 있었다. 또한 [Fig. 4-(c)]과 같이 악성 앱 WonRD를 실행하였을 경우 금융정보 유출과 관련된 부분이 실행되면서 내부에서 msgget(), semget(), recv() 및 select() 등의 시스템 콜 이벤트가 많이 발생하는 것을 확인할 수 있었다. 따라서 본 실험을 통해 부가적으로 각각의 시스템 콜 이벤트이 다음과 같은 기능은 수행한다는 것을 확인 할 수 있었다.

① msgget()은 커널 내애 새로운 메시지 큐를 만들기 위하여 또는 존재하는 큐에 접근하기 위해 사용된다.

② semget()은 새로운 세마퍼 집합을 만들거나 존재하는 집합에 접근하기 위하여 시스템 콜 이벤트를 사용한다.

③ recv()는 소켓으로부터 메시지를 받는 시스템 호출 함수이다.

④ select()는 동시에 여러 개의 파일 기술자를 읽기, 쓰기 행동이 있는지 알아볼 수 있다. 또한, 상태가 변경되는 파일 기술자들의 수를 기다리는 기능을 가진다.

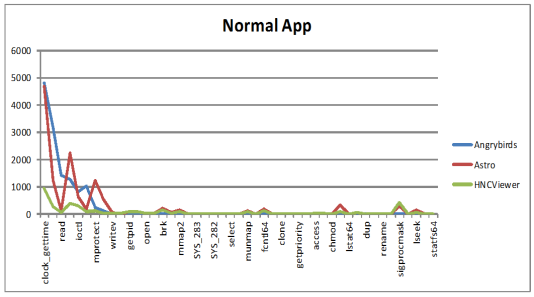
4.3 시스템 콜 이벤트 발생 빈도 비교 분석

[Fig. 3]과 [Fig. 4]에서 제시된 정상 앱과 악성 앱에 발생하는 시스템 콜 이벤트를 대상으로 각각의 시스템 콜 이벤트에 대한 발생 빈도수를 비교 분석하면 다음 [Fig. 5]와 같다.

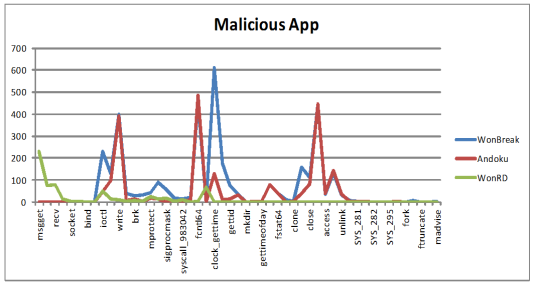
[Fig. 5-(a)]와 같이 정상 앱은 일반적으로 clock_gettime(), epoll_wait(), read(), futex(), ioctl(), write() 및 mprotect() 시스템 콜 이벤트가 주로 발생하는 것을 확인할 수 있었으며, [Fig. 5-(b)]와 같이 악성 앱인

경우 정상 앱과는 달리 msgget(), semget(), recv() 및 fcntl64() 등의 시스템 콜 이벤트가 많이 발생하는 것을 확인할 수 있었다.

이는 악성 앱인 경우 사용자 모르게 단말 내부 리소스에 대한 접근 및 변경 등의 과정을 수행하기 때문에 fcntl64(), lseek() 및 fsync() 등의 시스템 콜 이벤트가 많이 호출되는 것을 확인할 수 있었다. 또한 외부 네트워크와의 연동 및 네트워크 송수신 과정을 수행하므로 msgget(), recv() 등과 같이 메시지 큐 정보 획득 및 소켓 연결 과정과 관련되는 이벤트가 상대적으로 많이 발생하는 것을 확인할 수 있었다.



(a) System Call Events Activated from Normal Apps



(b) System Call Events Activated from Malicious Apps

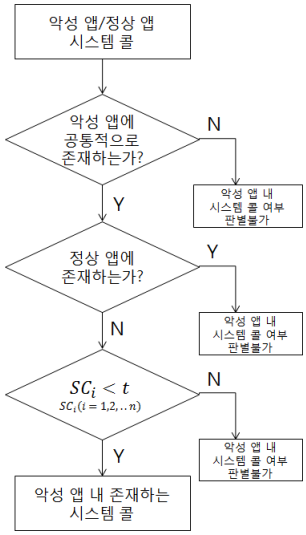
[Fig. 5] Comparison of kernel-level system call event results when running normal and malicious apps

5. 커널 계층 이벤트 시퀀스 유사도 분석

5.1 악성 앱 유사도 분석 알고리즘

기존 [6,8,10] 기법과 달리 본 연구에서는 Strace를 기반으로 어플리케이션을 실행해서 발생한 시스템 콜 이벤트를 가지고 발생 빈도와 특징을 토대로 악성 앱 실행과 관련된 시스템 콜 이벤트 시퀀스를 찾고자 한다. 이를 위해 우선 정상 앱과 악성 앱의 시퀀스의 공통점을 발견한 뒤, 악성 앱에서만 발생하는 시스템 콜 이벤트를 추출하였다.

우선 악성 앱과 정상 앱에서 각각 발생하는 시스템 콜 이벤트를 대상으로 악성 앱을 판별할 수 있는 시스템 콜 이벤트 특성을 다음과 같이 추출하였다. [Fig. 6]과 같은 알고리즘을 통해 n 개의 악성 앱에서 발생하는 시스템 콜 이벤트 중에서 공통적으로 1회 이상 호출되었으면서 또한 m 개의 정상 앱의 시스템 콜 이벤트에서는 호출되지 않은 시스템 콜 이벤트를 추출하였다. 이를 통해 악성 앱 검출 및 판별 과정에 사용되는 시스템 콜 이벤트 특성을 도출할 수 있다.



[Fig. 6] Kernel layer system call event sequence analysis algorithm

[Fig. 6]에서 제시한 알고리즘은 [1단계] 정상 앱과 악성 앱으로 부터 각각 발생하는 시스템 콜 이벤트를 수집한 후, [2단계] 악성 앱 실행으로부터 수집된 시스템 콜 이벤트 집합을 대상으로 집합 내에 공통적으로 존재하는 시스템 콜 이벤트가 있는지 분석을 한다. 만약에 공통적으로 존재하는 시스템 콜 이벤트가 있다면 다음 단계로 넘어가게 되고, 존재하는 시스템 콜 이벤트가 없다면 그 시스템 콜 이벤트들로 인한 악성 앱의 유사도 분석은 판별이 불가하다. [3단계] 이제 악성 앱에 공통적으로 존재하는 시스템 콜 이벤트가 정상 앱의 시스템 콜 이벤트 집합이 공집합이 아닐 경우($|SC_i| \neq 0$), 이는 정상 앱과 악성 앱에서 공통적으로 실행되는 시스템 콜 이벤트가 존재한다는 것을 의미한다. 공집합일 경우 악성 앱의 유사도 판별은 불가하다. [4단계] 정상 앱에서 발생하는 시

스템 콜 이벤트의 발생 빈도 임계치 값과 악성 앱에서 발생하는 시스템 콜 이벤트 발생 임계치를 구한 다음 악성 시스템 콜 이벤트로 의심이 되는 집합 SC_i가 해당 임계치 범위내에 존재하면 악성 앱 관련 시스템 콜 이벤트라고 판별하게 된다.

5.2 유사도 기반 악성 앱 시스템 콜 이벤트 분석

앞에서 제시한 시스템 콜 이벤트 유사도 분석 알고리즘을 적용하여 악성 앱에서 발생하는 시스템 콜 이벤트를 추출하는 과정을 수행하였으며 이를 통해 악성 앱과 관련된 시스템 콜 이벤트 발생 시퀀스 특징을 도출할 수 있었다. 유사도 분석 알고리즘을 적용하여 우선 악성 앱에 공통적으로 발생하는 시스템 콜 함수와 발생 빈도를 분석하였다. <Table 1>을 통해 악성 앱에 공통적으로 포함되는 시스템 콜 함수 11개를 도출할 수 있었다. 따라서 악성 앱 실행 과정시 공통적으로 발생하는 11개의 시스템 콜인 경우 1차적으로 임의의 앱 실행시에도 발생할 가능성이 높은 시스템 콜 함수라는 것을 알 수 있었다.

$$SC_i^{Mal}=\{\text{ioctl(), read(), write(), mmap2(), brk(), munmap(), getpid(), mprotect(), writev(), syscall_983042(), fcntl64(), select()}\}$$

〈Table 1〉 Kernel events when malicious apps are running

WonBreak		Andoku		WonRD	
Command	Count	Command	Count	Command	Count
clock_gettime	613	clock_gettime	131	msgget	229
epoll_wait	178	epoll_wait	10	semget	77
futex	158	read	96	recv	79
ioctl	229	ioctl	47		
read	131	write	393		
write	400	getpid	15	ioctl	49
pread	18	gettid	13	read	16
mmap2	38	stat64	33	write	11
sigprocmask	57	mkdir	1	pread	3
brk	30	chmod	2	mmap2	3
munmap	34	gettimeofday	2	sigprocmask	17
getpid	89	open	79	brk	6
gettid	74	fcntl64	487	munmap	3
mprotect	43	fstat64	41	mprotect	27
clone	3	dup	1	getpid	14
close	112	mmap2	9	SYS_224	14
epoll_ctl	6	mprotect	17	syscall_983042	12
open	79	clone	3	writev	1
fstat64	41	futex	38	socket	2
writev	22	close	82	getsockname	3
syscall_983042	13	lseek	448	bind	1
dup	11	brk	14	fcntl64	2
fcntl64	452	munmap	2	connect	1
lseek	445	access	41	select	69
stat64	35	fsync	144		
gettimeofday	2	unlink	36		
access	36	getpriority	4		
chmod	1	writev	12		
fsync	132	syscall_983042	4		
unlink	33	SYS_281	2		
ftruncate	1	SYS_286	5		
getpriority	8	SYS_282	1		
SYS_281	2	SYS_283	1		
SYS_286	5	select	1		
SYS_282	1	SYS_295	1		
SYS_283	1	pipe	4		
select	1	fork	1		
SYS_295	1				
pipe	4				
fork	1				
getuid32	1				
madvise	2				

하지만 이와 같은 이벤트는 정상적인 앱에서도 실행될 가능성이 높기 때문에 추가적인 분류 및 유사도 분석 과정을 수행할 필요가 있다.

정상 앱에 존재하는 시스템 콜 함수들 중에서는 빈번하게 발생하지 않으면서도 악성 앱에서 공통적으로 발생하였던 시스템 콜 함수를 도출하였다. <Table 2>를 토대로 아래 세 가지 그룹으로 분류할 수 있었다.

① clock_gettime(), epoll_wait(), read(), futex(), ioctl(), write(), mprotect() 및 syscall_983042() 등의 시스템 콜 함수는 정상 및 악성 앱 실행시 모두 많은 빈도수로 실행되었기 때문에 악성 앱의 유사도를 분석하기 위한 시스템 함수로는 부적절하다는 것을 알 수 있었다.

② chmod(), mkdir(), getuid32(), clone(), select(), getpriority(), getuid32(), madvise() 및 epoll_ctl() 등의 시스템 콜 함수는 정상 앱 실행시 빈도수가 적으면서도 악성 앱 내에서도 존재하는 시스템 함수들로 악성 앱 실행 여부를 판별하는데 적용이 가능한 함수들이라는 것을 알 수 있었다.

③ uname(), SYS_290(), lstat64(), statfs64() 등의 시스템 함수는 정상 앱에만 존재하고 악성 앱 내에는 존재하지 않는 시스템 함수들로써 악성 앱 실행과는 연관성이 없는 시스템 콜 함수라는 것을 알 수 있었다.

〈Table 2〉 Kernel events when normal apps are running

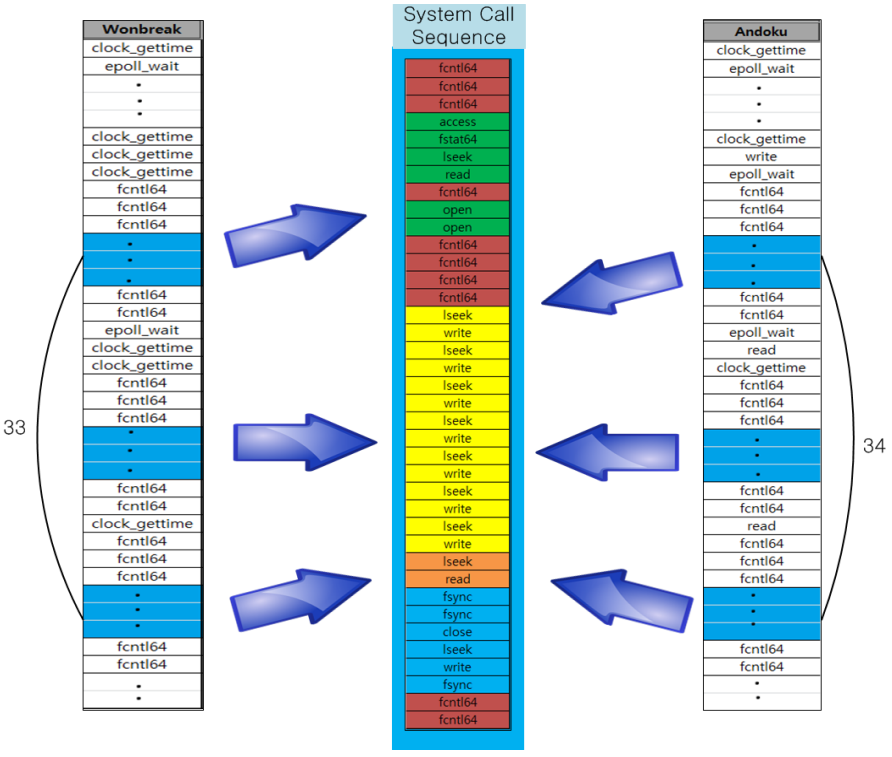
Angrybirds		Astro		HNCViewer	
Command	Count	Command	Count	Command	Count
clock_gettime	4807	clock_gettime	4686	clock_gettime	928
epoll_wait	3163	epoll_wait	1240	epoll_wait	275
read	1432	futex	2227	read	71
futex	1283	ioctl	632	ioctl	303
ioctl	824	read	110	write	104
write	1042	write	187	getpid	97
mprotect	257	mprotect	1255	gettid	80
syscall_983042	125	syscall_983042	561	futex	410
writev	6	getpid	106	writev	28
gettimeofday	9	gettid	88	access	23
getpid	9	access	47	open	26
gettid	3	open	43	mkdir	2
open	2	fsync	46	chmod	6
fstat64	1	close	72	mprotect	135
brk	13	chmod	8	fsync	26
close	5	stat64	344	close	47
mmap2	7	lstat64	7	stat64	90
SYS_281	6	gettimeofday	11	lstat64	2
SYS_283	4	mmap2	141	gettimeofday	20
uname	2	clone	9	getuid32	1
SYS_282	2	pread	55	pread	70
SYS_290	1	munmap	122	mmap2	104
select	2	dup	16	munmap	84
SYS_292	1	fcntl64	199	brk	145
munmap	3	epoll_ctl	6	sigprocmask	415
SYS_286	6	rename	8	getpriority	2
fcntl64	4	unlink	13	madvise	2
SYS_294	1	brk	217	rename	1
clone	2	sigprocmask	308	unlink	7
nanosleep	1	writev	53	syscall_983042	43
getpriority	1	fstat64	29	clone	8
madvise	1	nanosleep	4	dup	14
		getpriority	4	fcntl64	119
		madvise	4	epoll_ctl	4
		mkdir	8	lseek	74
		lseek	163	fstat64	19
		getuid32	8		
		statfs64	1		

5.3 악성 앱 시스템 콜 이벤트 시퀀스 분석

[Fig. 4-(a)]와 [Fig. 4-(b)]에서 제시한 악성 앱 Wonbreak 및 Andoku 인 경우 실행시 단말내 개인정보를 탈취하는 기능을 포함하고 있는 악성 앱이다. 유사한 기능을 포함하고 있는 악성 앱이기 때문에 앞에서 제시한 시스템 콜 이벤트 정보를 토대로 두 앱에서 발생하는 이벤트의 발생 순서의 유사도를 분석하였다. 먼저, 악성 앱에 공통적으로 존재하는 시스템 함수 11개 중에서 select()를 제외한 10개의 시스템 함수들은 정상 앱에서도 모두 존재하는 것을 확인할 수 있었다. 반면, 시스템 함수 select()는 오직 정상 앱의 Angrybirds에서만 존재함으로써 악성 앱 내 존재할 수 있을 시스템 함수라고 추측할 수 있다. 또한, 11개의 시스템 함수의 빈도를 악성 앱과 정상 앱 내에서 빈도를 조사하여, 악성 앱 내에서는 빈도수의 평균보다 높은 것과 정상 앱 내에서의 빈도수의 평균보다 낮은 것을 찾아보면 fcntl64()가 이에 해당한다는 것을 알 수 있었다. 따라서 악성 앱 내에서 존재하는 시스템 함수 fcntl64()의 위치를 찾고 fcntl64() 시스템 콜 함수가 나타난 부분의 시스템 콜 이벤트 시퀀스를 분석한 결과 다음과 같은 유사도를 확인할 수 있었다. [Fig. 7]에서 보는 것과 같이 시스템 함수 fcntl64()를 기준으로 보았을 때 Wonbreak와 Andoku 모두 위와 같은 거의 동일한 시스템 콜 이벤트 시퀀스와 그 수가 존재한다는 것을 확인할 수 있었다. 따라서 이와 같은 검출 패턴을 통해서 임의의 앱에 대한 시스템 콜 이벤트 분석시 악성 여부를 판별할 수 있다.

5.4 성능 비교 분석

본 논문에서 제안한 기법과 앞서 2장에서 서술했던 기존 Crowdroid 기법[6] 그리고 CuckooDroid 기법[10]과의 유사점 및 차이점을 살펴보면 다음 <Table 3>과 같다. 우선 기존 기법과 유사하게 Strace 모듈을 적용하여 리눅스 커널 기반 안드로이드 플랫폼에서 발생하는 시스템 콜 호출을 수집 및 분석하였다. 하지만 기존 Crowdroid에서는 악성 및 정상 앱 실행 결과 발생하는 커널 계층 시스템 콜 호출의 빈도만을 중심으로 정상과 악성 앱을 분류하였다. 그러나 기존 기법에서는 구체적으로 어떤 시스템 콜 호출이 어느 정도 발생하였는지에 대한 분석



[Fig. 7] Generate a common kernel layer event sequence when running malicious apps

없이 전체적인 시스템 콜 발생 빈도의 형태만을 중심으로 악성 여부를 판별하였다. 이에 본 논문에서는 [8]과 달리 직접 상용 모바일 단말을 대상으로 안드로이드 플랫폼 기반으로 Strace를 실행하여 발생하는 시스템 콜 함수에 대한 빈도 및 유사도 분석 과정을 수행하였고 실험 결과를 제시하였다.

〈Table 3〉 Experimental results and performance comparison

Items	Crowdroid[6]	CuckooDroid[10]	Proposed
Target Platform	Android	Android	Android
Aggregation Method	Strace	Cuckoo Sandbox	Strace
Event Extraction	○	△	○
Target Device	1개	n개	n개
Realtime Aggregation	×	○	○
Feature Analysis & Classification Function	△	○	○
Decision Algorithm	K-mean Analysis	Static/Dynamic Analysis	Event Sequence & Similarity Analysys
Malicious App Detection	○	○	○
Sequence Extraction	×	△	○

기존 Crowdroid 및 CuckooDroid 기법[6,10]과 본 논문에서 제시한 기법에 대한 실험 결림를 비교 분석하면, 상용 모바일 단말에 대한 실험 결과 악성 앱인 경우 사용자 모르게 단말 내부 리소스에 대한 불법적인 접근 및 변경 등의 과정을 수행하기 때문에 [Fig. 7]에 제시된 것과 같이 fcntl64(), lseek() 및 fsync() 등의 시스템 콜 이벤트가 정상적인 앱에 비해 상대적으로 많이 호출되는 것을 확인할 수 있었다. 또한 악성 앱에 의해 사용자도 모르게 외부 네트워크와의 연동 및 네트워크 송수신 과정을 수행하기 때문에 msgget(), recv() 등과 같이 메시지 큐 정보 획득 및 소켓 연결 과정과 관련된 시스템 콜 함수가 정상 앱에 비해 상대적으로 많이 발생하는 것을 확인할 수 있었다.

K-mean 분석 기법을 이용하는 Crowdroid 및 Sandbox 기반 정적/동적 분석 과정을 수행하는 CuckooDroid 시스템과 달리 본 연구에서 구현한 시스템인 경우 이벤트 시퀀스에 대한 유사도 분석 과정을 수행하였고, 시스템 콜 함수의 빈도수 분석을 함께 수행하여 악성 앱 실행시 발생하는 시스템 콜 함수에 대한 통합 유사도를 측정할 수 있었다. 이를 통해 fcntl64() 함수를 중심으로 약 30여개 정도의 시스템 콜 함수가 Wonbreak 및 Andoku 악성 앱에서 순차적으로 실행되는 것을 확인할 수 있었다.

6. 결 론

안드로이드 플랫폼 기반 모바일 어플리케이션은 자바 언어를 통해 오픈 소스 형태로 개발되기 때문에 오픈 마켓을 통해 마치 정상 앱인 것처럼 위장하여 악성코드를 삽입한 형태로 배포가 가능하다. 따라서 안드로이드 플랫폼을 기반으로 하는 상용 모바일 단말인 경우 악성 앱을 통한 공격 위협에 노출되어 있다. 이에 본 논문에서는 안드로이드 기반 상용 모바일 단말 환경에서 사용자가 손쉽게 설치 및 이용할 수 있는 악성 앱을 효율적으로 탐지하기 위한 기법을 제시하였다. 우선 앱 실행시 발생하는 시스템 콜 이벤트를 수집 및 분석하는 기존 Crowdroid 기법에서의 접근 방법과 CuckooDroid 시스템 기반 연구 결과에 대해 분석하였다. 이를 토대로 안드로이드 기반 단말 내 커널 계층 이벤트를 추출하는 모듈을 구현하였으며 이를 토대로 악성 앱을 판별하는 메커니즘을 제시하였다. 안드로이드 커널에서 시스템 콜 이벤트를 수집할 수 있는 Strace 모듈을 이용하여 정상 및 악성 앱에서 발생하는 실시간 시스템 콜 이벤트 특성을 비교 분석하였다. 또한 발생하는 이벤트에 대한 빈도수 및 시퀀스 유사도 분석 알고리즘을 이용하여 악성 앱을 판별할 수 있는 메커니즘을 제시하였다.

본 연구에서 제시한 기법을 이용할 경우 정상 앱과 악성 앱 실행시 발생하는 시스템 콜 이벤트의 특징을 분석할 수 있었으며 이를 통해 임의의 모바일 앱에 대해 악성 여부를 판단하는 방법에 적용될 수 있다. 또한, Strace에서 추출한 시스템 콜 이벤트를 바탕으로 시퀀스 분석을 통해 악성 앱에서는 공통적으로 존재하면서 빈도수가 높고 상대적으로 정상 앱에서는 빈도수가 낮은 시스템 함수를 분류할 수 있었다. 이를 토대로 앱 실행시 일정한 패턴의 시스템 콜 함수 시퀀스가 존재하는 것을 직접 확인할 수 있었다.

향후 연구 과제로는 본 연구에서 제시한 기법을 보다 많은 형태의 모바일 앱에 적용하여 임의의 앱에 대한 악성 여부 판별의 정확성을 높이는 것이며 이를 통해 보다

안전한 상용 모바일 단말 사용자 환경을 제공하는 것이다.

ACKNOWLEDGMENTS

이 논문은 한신대학교 학술연구비 지원에 의하여 연구되었음

REFERENCES

[1] W. R. Jeon, J. Y. Kim, Y. S. Lee, D. H. Won, “Analysis of Threats and Countermeasures on Mobile Smartphone”, Journal of the Korean Society of Computer and Information, Vol. 16, No. 2, pp.153-163, 2011.

[2] W. Enck, M. Ongtang, P. McDaniel, “Understanding android security,” IEEE Security & Privacy Magazine, Vol. 7, No. 1, pp. 50-57, 2009.

[3] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, “Google Android: A State-of-the-art Review of Security Mechanisms,” Technical Report, Cornell University, 2009.

[4] Sushma Verma, Sunil Kumar Muttoo, S.K. Pal, “MDroid: Android based Malware Detection Using MCM Classifier,” International Journal of Engineering Applied Sciences and Technology, Vol.1, No.8, pp. 206-215, 2016.

[5] J. W. Park, S. T. Moon, G. W. Son, I. K. Kim, K. S. Han, E. G. Im, I. G. Kim, “An Automatic Malware Classification System using String List and APIs”, Journal of Security Engineering, Vol.8, No.5, pp.611-626, 2011.

[6] I. Burguera, U. Zurutuza, S. Nadjm-Tehrani, “Crowdroid: Behavoir-Based Malware Detection System for Android,” Proceeding of the 1st ACM workshop on security and privacy in smartphones and mobile devices(SPSM'11), ACM, Vol. 1, pp. 15-26, 2011.

[7] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, Y. Weiss, “Andromaly: a behavioral malware detection framework for android devices,” Journal of Intelligent Information Systems, Vol. 38, No. 1, pp. 161-190, 2012.

[8] Y. Zhong, H. Yamaki, H. Takakura, “A Malware Classification method Based on Similarity of Function Structure,” 12th International Symposium of Applications and the Internet(SAINT), pp.256-261, 2012.

[9] More than 50 Android apps found infected with rootkit malware, http://www.guardian.co.uk/technology/blog/2011/mar/02/android-market-apps-malware.

[10] CuckooDroid - http://cuckoo-droid.readthedocs.org/

[11] Y. J. Ham, H. W. Lee, “Design and Implementation of Malicious Application Detection System Using Event Aggregation on Android based Mobile Devices,” Journal of The Korea Society of Internet and Information, Vol.14, No.6, pp.125-139, 2013.

[12] Y. J. Ham, H. W. Lee, “Malicious Trojan Horse Application Discrimination Mechanism using Realtime Event Similarity on Android Mobile Devices,” Journal of Internet Computing and Services, Vol.15, No.3, pp.31-43, 2014.

[13] H. W. Lee, "Android based Mobile Device Rooting Attack Detection and Malicious Application Event Monitoring," Review of Korean Society for Internet Information, Vol. 13, No. 1, pp.30-38, 2012.

[14] http://www.malgenomeproject.org, 2013. 4

[15] Y. J. Ham, D. Y. Moon, H. W. Lee, J. D. Lim, J. N. Kim, "Android Mobile Application System Call Event Pattern Analysis for Determination of Malicious Attack", International Journal of Security and Its Applications(IJSIA), Vol.8, No.1, pp.231-246, 2014.

[16] S. W. Cho, W. J. Jang, H. W. Lee, “Development of User Oriented Vulnerability Analysis Application on Smart Phone”, Journal of the Korea Convergence Society, Vol. 3, No. 2, pp. 7-12, 2012.

[17] B. S. Yu, S. H. Yun, "The Design and Implementation of Messenger Authentication Protocol to Prevent Smartphone Phishing", Journal of the Korea Convergence Society, Vol. 2, No. 4, pp.

9-14, 2011.

[18] M. H. Lee, "A Study on N-Screen Convergence Application with Mobile WebApp Environment", Journal of the Korea Convergence Society, Vol. 6, No. 2, pp. 43-48, 2015.

저자소개

이 형 우(Hyung-Woo Lee) **[종신회원]**

- 1994년 2월 : 고려대학교 컴퓨터학과 (학사)
- 1996년 2월 : 고려대학교 컴퓨터학과 (석사)
- 1999년 2월 : 고려대학교 컴퓨터학과 (박사)
- 1999년 3월 ~ 2003년 2월 : 백석대학교 정보통신학부 조교수
- 2003년 3월 ~ 현재 : 한신대학교 컴퓨터공학부 부교수, 정교수

<관심분야> : 모바일 보안, 디지털포렌식, 정보보호, 네트워크 보안