KIPS Tr. Comp. and Comm. Sys.
Vol.6, No.3 pp.159~168 pISSN: 2287-5891

https://doi.org/10.3745/KTCCS.2017.6.3.159

Study on DNN Based Android Malware Detection Method for Mobile Environment

Jinhyun Yu† · In Hyuk Seo† · Seungjoo Kim††

ABSTRACT

Smartphone malware has increased because Smartphone users has increased and smartphones are widely used in everyday life. Since 2012, Android has been the most mobile operating system. Owing to the open nature of Android, countless malware are in Android markets that seriously threaten Android security. Most of Android malware detection program does not detect malware to which bypass techniques apply and also does not detect unknown malware. In this paper, we propose lightweight method for detection of Android malware using static analysis and deep learning techniques. For experiments we crawl 7,000 apps from the Google Play Store and collect 6,120 malwares. The result show that proposed method can achieve 98.05% detection accuracy. Also, proposed method can detect about unknown malware families with good performance. On smartphones, the method requires 10 seconds for an analysis on average.

Keywords : Smartphone, Android, Malware Detection, Deep Learning

모바일 환경에 적합한 DNN 기반의 악성 앱 탐지 방법에 관한 연구

유 진 현† · 서 인 혁† · 김 승 주††

요 약

스마트폰 사용자가 증가하고 스마트폰이 다양한 서비스와 함께 일상생활에서 널리 사용됨에 따라 스마트폰 사용자를 노리는 악성코드 또한 증가하고 있다. 안드로이드는 2012년 이후로 가장 많이 사용되고 있는 스마트폰 운영체제이지만, 안드로이드 마켓의 개방성으로 인해 수많은 악성 앱이 마켓에 존재하며 사용자에게 위협이 되고 있다. 현재 대부분의 안드로이드 악성 앱 탐지 프로그램이 사용하는 규칙 기반의 탐지 방법은 쉽게 우회가 가능할 뿐만 아니라, 새로운 악성 앱에 대해서는 대응이 어렵다는 문제가 존재한다. 본 논문에서는 앱의 정적 분석과 딥러닝을 결합하여 스마트폰에서 직접 악성 앱을 탐지할 수 있는 방법을 제안한다. 수집한 6,120개의 악성 앱과 7,000개의 정상 앱 데이터 셋을 가지고 제안하는 방법을 평가한 결과 98.05%의 정확도로 악성 앱과 정상 앱을 분류하였고, 학습하지 않은 악성 앱 패밀리의 탐지에서도 좋은 성능을 보였으며, 스마트폰 환경에서 평균 10초 내외로 분석을 수행하였다.

키워드 : 스마트폰, 안드로이드, 악성 앱 탐지, 딥러닝

1. 서 론

스마트폰 시장이 꾸준히 성장하고 금융서비스나 회사업무와 같은 일상생활에서 스마트폰이 널리 사용됨에 따라 스마트폰의 보안 위협도 계속해서 증가하고 있다. 특히, 안드로이드 운영체제를 대상으로 하는 악성코드가 꾸준히 증가하고 있는데, Kaspersky Lab에서 발표한 보고서에 따르면 전체 모바일 대상 악성코드 중 안드로이드를 대상으로 하는 악성코드는 98.05%를 차지하고 있다[1]. 이처럼 안드로이드를 대상으로 하는 악성코드가 많은 이유는 스마트폰 운영체제에서 안드로이드가 높은 점유율을 차지하고 있고, 안드로이드 앱 마켓의 개방성으로 인해 악성 앱의 배포가 쉽기 때문이다. Gartner의 2015년 보고서에 따르면 안드로이드는 가장 많이 사용되는 스마트폰 운영체제이며 그 비율을 약 82%에 이른다[2]. 안드로이드는 구글 공식 스토어와 서드 파티 마켓을 통해 사용자에게 손쉽게 원하는 앱을 설치할 수 환경을 제공한다. 하지만 이러한 안드로이드 마켓의 개방성으로 인해 수

※ 본 연구는 미래창조과학부 및 정보통신기술진흥센터의 대학ICT연구센터 육성 지원사업의 연구결과로 수행되었음(IITP-2016-R0992-16-1006).
† 준 회 원 : 고려대학교 정보보호대학원 정보보호학과 석사과정
†† 종신회원 : 고려대학교 사이버국방학과/정보보호대학원 정교수
Manuscript Received : November 30, 2016
Accepted : December 13, 2016
* Corresponding Author : Seungjoo Kim(skim71@korea.ac.kr)

많은 악성 앱이 마켓을 통해 유포되어 사용자에게 위협이 되고 있다.

이러한 악성 앱의 위협으로부터 사용자를 보호하기 위하여 현재 많은 악성 앱 탐지 프로그램이 사용되고 있지만 한계가 존재한다. 대부분의 탐지 프로그램이 사용하는 규칙 기반의 탐지 방법은 쉽게 우회가 가능할 뿐만 아니라, 새로운 악성 앱이 등장하면 분석가가 해당 악성 앱을 분석하여 탐지 규칙을 갱신하기 전까지 탐지가 불가능하다는 단점이 존재한다. McAfee의 보고서에 따르면 매 분기마다 100만개의 새로운 악성 앱이 발견되고 있다[3]. 새롭게 발견되는 100만개의 악성 앱을 모두 분석하여 규칙을 갱신하는 것은 거의 불가능에 가깝다. 따라서 방대한 양과 함께 빠르게 변화하는 악성 앱에 대응하기 위해서는 새로운 악성 앱 탐지 방법에 대한 연구가 필요하다.

본 논문에서는 스마트폰 환경에서 앱을 분석하고 딥러닝을 이용하여 악성 앱을 탐지하는 경량의 악성 앱 탐지 시스템을 제안한다. 본 논문의 구성은 다음과 같다. 2장에서는 악성 앱을 탐지하는 방법과 관련된 연구를 정리한다. 3장에서는 제안하는 악성 앱 탐지 방법을 설명하고, 4장에서는 수집한 실제 앱 데이터 셋을 통해 실험한 결과를 분석하였다. 마지막으로 5장에서는 결론 및 향후 연구방향에 대해 서술한다.

2. 관련 연구

안드로이드에서 악성 앱이 등장한 이후 악성 앱을 탐지하기 위해 여러 연구가 진행되었다. 악성 앱을 탐지하는 방법은 크게 정적 분석을 이용한 방법과 동적 분석을 이용한 방법으로 나눌 수 있다.

2.1 정적 분석을 통한 탐지

정적 분석 방법은 앱을 실행시키지 않고 소스 코드로부터 악성 행위를 확인하는 방법이다. 정적 분석의 한 가지 방법으로 앱에서 요구되는 퍼미션(Permission)을 분석하는 방법이 있다. 안드로이드에서 앱은 설치되기 전에 사용자에게 자원에 접근할 수 있도록 퍼미션을 요청하고 사용자에게 알린다. 악성 앱은 정상 앱과 비교하여 특정 퍼미션을 더 자주 요청하는 경향이 있다. Felt 등의 연구에서는 실제 모바일 악성 앱을 분석하여 사용하는 퍼미션을 추출하였고, 퍼미션으로 악성 앱을 탐지하는 방법의 효과를 설명하였다[4]. Sarma 등의 연구에서는 정상 앱과 악성 앱의 퍼미션을 비교 분석하여 악성 앱을 탐지하는데 중요한 퍼미션을 제시하였다[5].

정적 분석 방법의 다른 모델은 API(Application Program Interfaces) 기반의 탐지 모델이다. API 기반의 탐지 모델에서는 앱의 소스 코드로부터 API 사용 정보를 추출하여 악성 앱을 분류하는데 사용한다[6]. 이와 유사하게 Stowaway는 앱의 허가되지 않은 행동을 탐지하기 위해 API 호출을 확인한다[7].

본 논문에서는 정적 분석의 접근 방법을 이용하며, 퍼미션 정보, API 호출 정보를 악성 앱을 식별하기 위해 사용한다. 하지만 이전의 연구와는 두 가지 주요 측면에서 차이점을 가진다. 첫째, 본 논문에서는 탐지 규칙을 수동으로 작성하는 대신 머신 러닝을 적용하여 악성 앱을 탐지하였다. 둘째, 본 논문의 분석은 효율성을 최적화하여 스마트폰에서 직접 앱을 분석하고 악성 앱을 탐지할 수 있도록 하였다.

2.2 동적 분석을 통한 탐지

동적 분석은 안드로이드 디버깅 도구를 이용하거나 Sandbox, 가상 머신 등의 분석 환경에서 앱을 실행시켜 앱의 행동을 분석하는 방법이다. AppsPlayground는 가상 머신에 안드로이드 환경을 구축하고 앱을 실행시켜 민감한 API를 관찰하여 정보 유출과 루팅(Rooting)을 탐지하였다[8]. TaintDroid와 DroidScope는 Sandbox환경에서 오염 분석을 기반으로 플랫폼의 여러 계층에서 앱의 행동을 동적으로 관찰할 수 있는 분석 시스템이다[9, 10]. 하지만, 이러한 동적 분석을 이용한 탐지는 앱의 동작에 대한 자세한 정보를 제공하지만 실행 시간 동안 수행되지 않은 악성 행위는 탐지할 수 없다는 단점과 기술적으로 너무 복잡하여 현재까지는 스마트폰에서 직접 적용할 수 없다는 단점이 있다.

2.3 머신러닝을 이용한 탐지

안드로이드 악성 앱의 탐지 규칙을 수동으로 작성하고 갱신하는 어려움을 해결하기 위해 머신러닝 기반의 탐지방법이 연구되었다. 이러한 머신러닝 기반의 방법을 사용하면 데이터 셋을 학습하여 악성 앱을 분류할 수 있는 규칙을 생성할 수 있을 뿐만 아니라 새로운 악성 앱이 등장할 때마다 수동으로 규칙을 수정할 필요가 없이 학습을 통하여 규칙을 갱신할 수 있다. Peiravian 등의 연구에서는 앱을 정적 분석하여 퍼미션과 API 사용 정보를 획득하고 머신러닝을 이용하여 악성 앱을 탐지하였다[11]. 본 논문의 방법과 가장 유사한 DREBIN은 스마트폰에서 직접 앱을 분석하여 사용하는 하드웨어, 퍼미션, API 정보 등을 획득하고 머신러닝을 이용하여 악성 앱을 탐지하였다[12]. 하지만, DREBIN은 사전에 학습하지 않은 악성 앱 패밀리에 대해서는 탐지 정확도가 크게 떨어지는 단점을 가지고 있다. 본 논문에서는 탐지 정확도와 효율성을 향상시키고, 학습하지 않은 패밀리의 악성 앱에 대해서도 효과적으로 탐지하는 방법을 제안한다.

3. 딥러닝 기반의 악성 앱 탐지 방법

머신러닝에서의 인공신경망(artificial neural network)은 생물학의 신경망에서 영감을 얻은 통계학적 학습 알고리즘이다. 인공신경망은 시냅스의 결합으로 네트워크를 형성한 인공 뉴런이 학습을 통해 가중치를 변화시켜, 문제 해결 능력을 가지는 모델 전반을 가리킨다. 딥러닝은 인공신경망 모델이 발전된 형태로서, 계층구조로 이루어진 인공신경망의 Hidden layer가 여러 단계로 이루어진 구조이다. 딥러닝

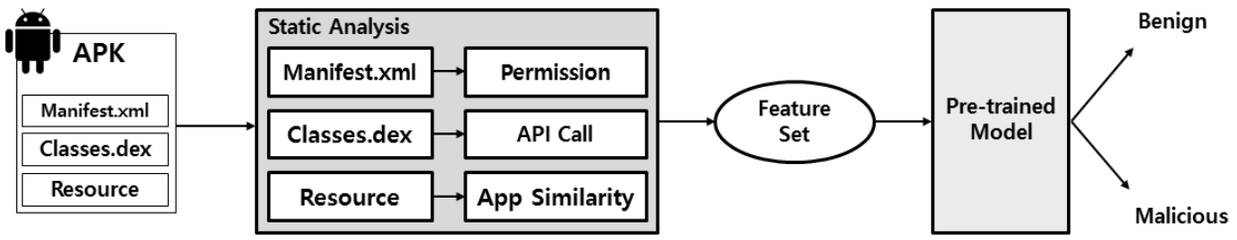

Fig. 1. Process of Proposed Method

의 장점은 더 많은 Hidden layer를 사용함으로써 데이터에 대한 표현 능력을 크게 증가시킬 수 있다는 것이다. 최근의 딥러닝 모델은 Hidden layer가 많아져서 노드를 연결하는 가중치의 수가 최대 수십억 개가 되기도 한다[13]. 과거에는 신경망을 학습시키는데 매우 오랜 시간이 걸렸고, 트레이닝 셋에 너무 가깝게 맞추어 학습되는 과적합 문제 등의 한계점으로 인해 사람들의 관심에서 멀어지게 되었다. 하지만 2000년대 들어 기존 신경망의 과적합 문제를 해결하는 방법이 연구되고, 학습시간을 단축하기 위해 GPU를 이용하거나 연산속도를 향상시키는 다양한 방법이 개발됨으로써 딥러닝이 다시 각광받게 되었다. 딥러닝은 다양한 분야에서 활용되고 있으며, 특히 자동 음성인식과 컴퓨터비전 분야에서 최고수준의 성능을 보여주고 있다.

본 논문에서는 딥러닝을 이용한 경량의 악성 앱 탐지 시스템을 제안한다. 제안하는 방법은 안드로이드 앱 설치파일인 APK(Android Application Package)파일을 입력으로 받아 악성 앱인지 탐지한다. 입력 받은 앱의 행위를 파악할 수 있도록 정적 분석을 통해 특성을 추출하고, 추출한 특성의 집합은 사전에 학습된 딥러닝 모델에 전달되며, 딥러닝 모델은 해당 앱이 악성 앱인지 정상 앱인지 결정한다. Fig. 1은 본 논문에서 제안하는 악성 앱 탐지 방법의 프로세스를 나타낸다.

3.1 정적 분석

제안하는 방법의 첫 번째 단계로, 주어진 앱에 대해서 정적 분석을 수행한다. 스마트폰의 제한된 자원을 가지고 분석을 수행하며, 분석 시간이 너무 오래 걸리는 것은 사용자로 하여금 불편함을 초래할 수 있기 때문에 효율적으로 추출할 수 있는 특성을 선택하는 것이 필수적이다. 본 논문에서는 APK파일의 구성요소인 AndroidManifest.xml, classes.dex, 리소스 파일에 중점을 두어 분석을 수행하였다. APK파일의 내부구성은 Fig. 2와 같으며, 해당 파일들은 APK파일의 압축을 풀어서 쉽게 획득할 수 있다.

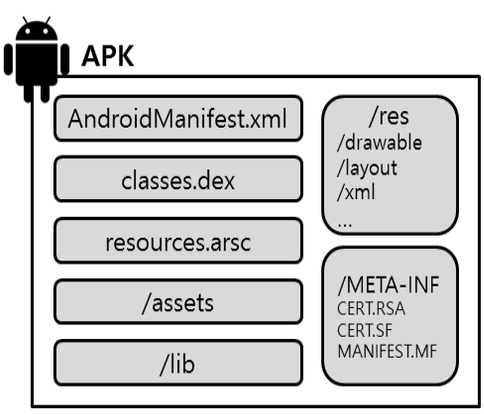

Fig. 2. Structure of APK

1) 퍼미션 정보 분석

AndroidManifest.xml파일은 앱이 사용하는 하드웨어 정보, 앱의 퍼미션 정보 등 앱의 실행을 지원하는 메타 정보를 포함하고 있다. 퍼미션 시스템은 안드로이드 보안 매커니즘에서 가장 중요한 요소 중 하나이다. 안드로이드 앱은 설치되기 전에 사용자에게 자원에 접근할 수 있도록 퍼미션을 요청하고 사용자에게 알린다. 사용자가 이를 승인함으로써 앱은 해당 자원에 접근할 수 있는 권한을 획득하게 되는데, 악성 앱은 정상 앱과 비교하여 특정 퍼미션을 더 자주 요청하는 경향이 있다[14]. 예를 들어, 많은 비율의 악성 앱이 SMS메시지를 보낼 수 있는 권한인 SEND_SMS 퍼미션을 요청하여 SMS메시지를 통해 사용자의 정보를 탈취하거나 과금을 유도하는 방식의 악성 행위를 수행한다. 제안하는 방법에서는 안드로이드 환경에서 xml 파서를 구현하여 AndroidManifest.xml파일에 나열된 모든 퍼미션을 문자열로 추출하였다.

2) API 호출 정보 분석

안드로이드 앱은 Java로 개발되고 달빅 가상머신에 최적화된 바이트코드로 컴파일 된다. 이 바이트코드는 APK파일에 classes.dex파일로 저장되어 달빅 가상머신에서 사용된다. classes.dex파일에는 앱에서 사용하는 API와 데이터에 대한 정보가 포함되어있다. 제안하는 방법에서는 classes.dex파일로부터 API 호출 정보를 특성으로 추출하여 사용한다. 안드로이드에서 제공하는 dex 라이브러리와 리플렉션 기법을 이용하여 API 정보를 추출하는 디스어셈블러를 구현하였다. API 정보는 다음의 3가지 방법을 통해서 추출한다.

a) 악성 앱에서 많이 사용되는 API 정보 추출

정상 앱에 비해서 악성 앱에서는 악성 행위를 수행하기 위해 특정 API를 더 자주 호출한다. 정상 앱과 비교하여 악성 앱에서 더 자주 사용하는 API 호출을 확인하기 위하여 정상 및 악성 앱의 샘플을 가지고 분석을 수행하였다. Fig. 3은 정상 앱과 악성 앱 각각 2,000개에서 사용하는 API를 비

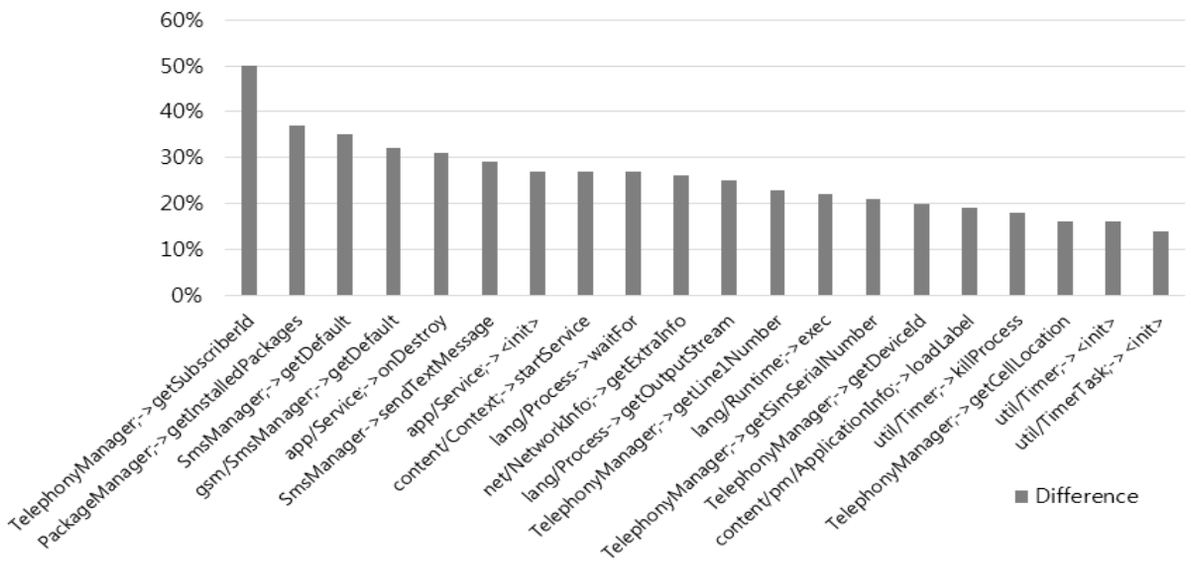

Fig. 3. Top 20 APIs with Highest Difference Between Malware and Benign Apps

교하여 가장 큰 차이를 보이는 상위 20개의 API를 나타낸다. 그래프의 가로축은 API의 이름을 나타내고 세로축은 정상 및 악성 앱에서 해당 API 사용유무의 차이를 나타낸다. 예를 들어 TelephonyManager클래스의 getSubscriberId() 호출은 스마트폰 단말기 가입자의 식별자인 IMSI(International Mobile Subscriber Identity)를 반환하는 API다. 분석 결과 getSubscriberId() 호출은 정상 앱과 비교하여 50%나 더 많은 악성 앱에서 호출되어 사용하는 것을 확인하였다. 또한, 단말기의 네트워크 정보를 획득할 수 있는 getExtraInfo() 호출의 경우에도 악성 앱에서 26%나 더 많이 호출되어 사용하는 것을 확인하였다. 본 논문에서는 분석의 결과에 따라서 가장 큰 차이를 보이는 상위 20개 API의 호출 정보를 특성으로 추출하여 악성 앱을 탐지하는데 사용하였다.

b) 민감한 정보에 접근하는 API 정보 추출

안드로이드에서는 민감한 데이터에 접근할 때 사용되는 특정한 API가 존재한다. 예를 들어 sendTextMessage() 호출은 SMS메시지를 보내는데 사용되고 ContentResolver클래스의 delete() 호출은 사용자의 데이터를 삭제할 때 사용된다. 이러한 민감한 데이터에 접근하는 API는 악의적인 행동으로 이어질 수 있으므로 본 논문에서는 민감한 데이터에 접근할 때 사용되는 API 정보를 특성으로 수집하였다.

c) 난독화에 사용하는 API 정보 추출

난독화는 변환 프로그램의 일종으로 프로그램 바이너리나 소스코드가 역공학에 의해 분석되는 것을 어렵게 하기 위한 기술이다. 안드로이드 악성 앱 제작자는 자신들이 제작한 악성 앱이 쉽게 분석이 되는 것을 막기 위해 난독화 기능을 적용한다. 예를 들어 몇몇 악성 앱에서는 문자열이나 클래스의 정보를 감추기 위해 Cipher.getInstance() 호출 등을 사용하여 암호화한다. 제안하는 방법에서는 앱의 난독화 적용 여부를 판단하기 위해 난독화 기법에서 자주 사용되는 API의 사용 정보를 추출한다.

3) 유사도 분석

Zhou 등의 연구에 따르면 악성 앱의 86%는 리패키징 기법을 사용하였다[14]. 리패키징 기법은 공격자가 기존의 앱을 디컴파일하여 코드를 수정한 후에 다시 패키징하여 배포하는 기법을 말한다. 공격자는 리패키징 기법을 통해 앱의 기능을 우회하는 등의 악의적인 행동을 수행하거나 인기 있는 앱에 악성 행위를 삽입하여 악성 앱을 널리 퍼뜨리는데 사용한다. 대부분의 리패키징 된 앱은 기존 앱에서 일부분만 수정하기 때문에 거의 동일한 구성을 유지하게 된다. 이러한 리패키징 된 앱을 탐지하기 위해서 dex파일을 디컴파일하고 소스 코드를 분석하여 탐지하는 연구가 진행되었다[15]. 하지만 소스 코드를 분석하는 방법은 프로그래밍 및 컴파일 과정에서 쉽게 우회가 가능하다. 본 논문에서는 리패키징 된 악성 앱을 탐지하기 위하여 APK파일의 리소스파일을 이용하여 시각적인 유사도를 분석한다.

앱의 유사도를 분석하는 개념도는 Fig. 4와 같다. 앱의 겉모양을 담당하는 layout파일과 앱에서 사용되는 이미지파일 등은 APK파일의 리소스 폴더에 저장되어있다. 유사도 분석 과정에서는 APK파일로부터 추출된 리소스파일의 해시 값을 기존에 수집된 앱과 비교하여 유사도를 측정한다. 본 논문에서는 해시 값을 구하기 위해 LSH(Locality Sensitive Hash) 함수를 사용하였다. LSH는 고차원의 데이터를 저차원의 데이터로 바꾸는 것으로, LSH의 핵심은 벡터 간 유사성이 보존되도록 해시를 하는 것이다. 따라서 유사한 데이터는 높은 확률로 같은 버켓(bucket)안에 들어가게 해준다. LSH의 매력적인 특징은 이론적으로 최악의 경우에도 성능을 보장한다는 것이다[16]. 이러한 특징을 바탕으로 LSH는 유사도를 비교하거나 데이터를 군집화하는데 주로 사용된다. Fig. 5

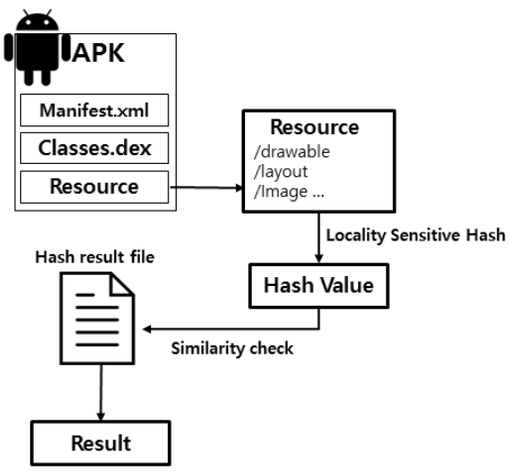

Fig. 4. Process of Visual Similarity Analysis

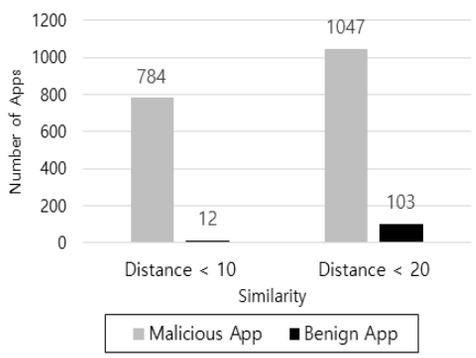

Fig. 5. Result of Visual Similarity Experiment

는 수집된 정상 앱과 악성 앱 각각 2,000개를 LSH해시를 사용하여 유사도를 비교한 결과를 나타낸다. 거리 값은 유사도를 비교할 때 얼마나 유사한지를 나타내는 값이다. 왼쪽의 그래프는 거리 값이 10이하로 자신과 매우 유사한 앱이 존재하는 경우를 나타내며, 오른쪽 그래프는 거리 값이 20이하로 유사한 경우를 나타낸다. 결과를 살펴보면 거리 값이 10이하로 자신과 매우 유사한 앱이 존재하는 악성 앱은 784개나 되는 반면에 정상 앱에서는 오직 12개만이 자신과 유사한 앱이 존재하는 것을 확인할 수 있다. 본 논문에서는 오픈 소스 LSH인 TLSH(Trend Micro Locality Sensitive Hash)를 안드로이드에 구현하여 사용하였다[17]. 유사도 비교에 필요한 해시 값 파일은 딥러닝 모델과 함께 학습 데이터로부터 함께 생성된다. 유사도 분석의 결과는 특성으로 사용하며, 사용자에게 탐지 결과와 함께 제공된다.

3.2 특성 집합 생성

정적 분석 과정이 완료되면, 추출한 특성으로 딥러닝 모델의 입력을 위한 특성 집합 S를 생성한다. 특성 집합 S는 정수형의 배열이며 특성 별로 다음의 규칙을 따라서 생성한다.

퍼미션 정보의 특성 집합 S_1은 다음의 수식 $f(x,p)$로 생성한다. 안드로이드에서 사용하는 모든 퍼미션 p에 대해서 해당 앱 x에서 사용하는 퍼미션을 대조하여 사용하는 퍼미션은 1, 사용하지 않은 퍼미션은 0으로 입력한다.

$$f(x,p)=\begin{cases}1\,, & \text{if app } x \text{ contains permission } p\\ 0\,, & \text{otherwise}\end{cases}$$

API 호출 정보의 특성 집합 S_2는 추출하는 API 호출 정보 집합 c에서 각각의 API가 앱 x에서 호출되는 횟수를 나타낸다.

$$f(x,c)= \text{ The number of API } c \text{ calls in app } x$$

리소스 유사도 특성 S_3의 경우에는 해당 앱 x와 유사한 앱 s의 개수를 나타낸다.

$$f(x,s)= \text{ The number of similar app } s \text{ to app } x$$

딥러닝 모델의 입력을 위한 최종 특성 집합 S는 다음과 같이 생성된다.

$$S= S_1 \cup S_2 \cup S_3$$

3.3 딥러닝 모델

본 논문에서는 안드로이드 악성 앱의 탐지 규칙을 수동으로 작성하고 갱신하는 어려움을 해결하기 위해 딥러닝 기반의 악성 앱 탐지를 수행한다. 이러한 딥러닝 기반의 방법을 사용하면 데이터 셋으로부터 학습을 통하여 악성 앱을 분류할 수 있는 규칙을 생성할 수 있을 뿐만 아니라 새로운 악성 앱이 등장할 때마다 수동으로 규칙을 수정할 필요가 없이 학습을 통하여 규칙을 갱신할 수 있다.

정적 분석 과정과 특성 집합 생성을 통해 만들어진 특성 집합 S는 딥러닝 모델의 입력 데이터로 사용된다. 딥러닝 모델은 특성 집합을 입력으로 받아 해당 앱이 악성 앱인지 정상 앱인지 결정한다. 제안하는 방법에서는 스마트폰에서 직접 딥러닝 모델을 학습시키지 않고 서버에서 학습시킨 딥러닝 모델을 스마트폰으로 전송하여 악성 앱을 탐지하는데 사용한다. 딥러닝 모델은 구글에서 개발한 머신러닝을 위한 오픈소스 라이브러리인 Tensorflow를 이용하여 생성하였다[18]. 생성한 딥러닝 모델의 구조는 4개의 히든 레이어로 구성된 Deep Feedforward Neural Network이다. 기본적인 Deep Feedforward Neural Network의 구조도는 Fig. 6A와 같다. 첫 번째 layer는 Input layer로 추출한 특성 집합을 입력으로 받는다. 중간에는 4개의 Hidden layer로 구성하여 데이터를 고차원으로 추상화하고 데이터의 표현 능력을 증가시키고자 하였다. 마지막 layer는 Output layer로 정상과 악성 두 개의 노드로 구성된다.

본 논문의 딥러닝 모델에서는 트레이닝 셋에 너무 가깝게 맞추어 학습되는 과적합 문제를 피하기 위해 Dropout 알고

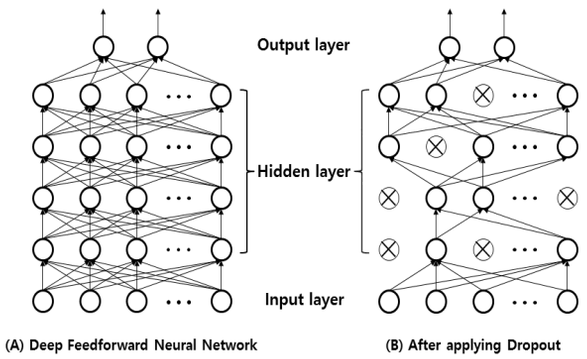

Fig. 6. Sturucture of Deep Feedfoward Neural Network

리즘을 사용하였다[19]. Dropout 알고리즘을 사용하는 방법은 과적합 문제를 해결하기 위해 각 layer를 사전훈련(pre-trainning)하여 가중치를 초기화하는 방법보다 훨씬 간단하고 좋은 성능을 보인다. Dropout 알고리즘은 모든 layer에 대해 학습을 수행하는 것이 아니라 신경망에 있는 Input layer나 Hidden layer의 일부 뉴런을 생략하고 줄어든 신경망을 통해 학습을 수행한다. Fig. 6B는 일부 뉴런을 생략하고 신경망을 학습하도록 Dropout 알고리즘을 적용한 모습을 그림으로 나타낸다. Dropout 알고리즘을 통해서 크게 두 가지의 효과를 볼 수 있다. 첫 번째로 투표(voting) 효과이다. Dropout을 통해 여러 개의 다른 망에 대해 학습을 하게 되면, 그 망에 대해서 과적합이 발생하고 이런 과정이 무작위로 반복 되면, 과적합 된 망이 늘어나 투표에 의한 평균 효과로 인해 결과적으로 과적합을 회피할 수 있다. 두 번째는 상호 적응(co-adaptation)을 피하는 효과이다. 신경망을 학습하다 보면 노드들의 연결 가중치 값들이 서로 비슷해지는 상호 적응 문제가 발생한다. Dropout을 이용하면 가중치나 바이어스가 특정 뉴런의 영향을 받지 않기 때문에 상호 적응을 피할 수 있다.

학습 과정을 통해 생성된 딥러닝 모델은 약 1Mbyte의 그래프 형태로 저장되며 스마트폰에 전송하여 사용한다.

3.4 탐지 정보 제공

딥러닝 모델은 특성 집합을 입력으로 하여 해당 앱이 악성 앱인지 정상 앱인지 판단한다. 악성 앱 탐지 애플리케이션은 딥러닝 모델의 결과를 전달받아 사용자에게 알린다. 제안하는 방법에서는 탐지의 결과뿐만 아니라 해당 앱의 정보를 제공한다. 이러한 정보는 해당 앱의 기능을 이해할 수 있도록 도와주며 앱 분석가에게는 악성 앱의 행위를 검사하고 더 깊이 이해할 수 있도록 도와준다. Fig. 7은 탐지 결과 화면을 나타낸다. 결과의 샘플은 DroidKungFu 패밀리에 속하는 악성 앱으로 결과를 보면, 설치 과정에서 요청하는 11개의 퍼미션 정보와 해당 앱에서 사용되는 API 호출 정보를 확인할 수 있다. 또한, 유사도 분석을 바탕으로 해당 앱과 유사한 앱의 정보를 제공하며 최종적으로는 해당 앱을 설치하거나 차단하는 기능을 제공한다.

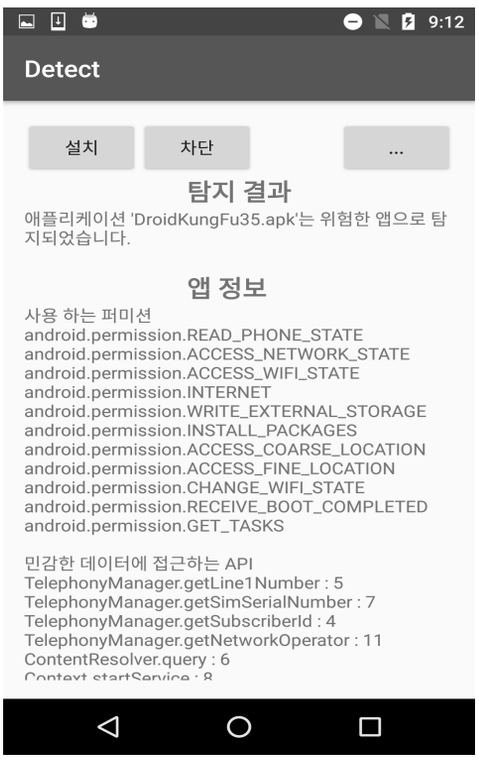

Fig. 7. Result for a Malware Sample

4. 평 가

4장에서는 본 논문에서 제안하는 악성 앱 탐지 방법의 성능을 평가하기 위해 실험한 결과를 설명한다. 성능 평가를 위한 실험은 다음과 같이 진행되었다.

a) 탐지 성능 평가 : 7,000개의 정상 앱과 6,320개의 악성 앱의 데이터 셋을 사용하여 제안하는 방법의 탐지 성능을 평가하였다. 또한, 관련 연구와 성능을 비교하였다.

b) 실행 시간 성능 평가 : 제안하는 방법의 실행 시간 성능을 평가한다. 일반적인 데스크탑 컴퓨터와 다른 종류의 스마트폰에서 성능을 평가하였고, 관련 연구와 성능을 비교하였다.

4.1 데이터 셋

평가를 위해 구글 앱 스토어에서 7,000개의 정상 앱을 수집하고 Contagio Community와 DREBIN Project로부터 6,320개의 악성 앱을 수집하였다[12, 20]. 데이터 셋은 정상 앱과 악성 앱의 비율을 맞추기 위해 정상 앱 6,320개와 악성 앱 6,320개를 사용하였다.

4.2 탐지 성능 평가

탐지 성능 평가에서 사용한 용어의 정의는 다음과 같다. TP(True Positive)는 악성 앱을 악성 앱이라고 정확히 탐지

한 개수이다. FN(False Negative)는 악성 앱을 정상 앱이라고 탐지한 개수이다. TN(True Negative)는 정상 앱을 정상이라고 정확히 탐지한 개수이고, FP(False Positive)는 정상 앱을 악성 앱이라고 탐지한 개수이다. 제안하는 방법의 평가의 척도로써 정확도(Accuracy), 재현율(Recall), 정밀도(Precision)를 사용하였다. 정확도, 재현율, 정밀도를 구하는 방법은 다음과 같다.

$$\text{정확도(Accuracy)} = \frac{TP+TN}{TP+TN+FP+FN}$$

$$\text{재현율(Recall)} = \frac{TP}{TP+FN}$$

$$\text{정밀도(Precision)} = \frac{TP}{TP+FP}$$

탐지 성능 평가의 신뢰도를 검증하기 위해 k-배 교차검증(k-fold Cross Validation) 기법을 적용하였다. k-배 교차검증은 데이터 셋을 동일한 크기로 임의 분할하고, 이 중 한 개를 검증 데이터 셋으로 사용하고, 나머지 (k-1)개를 학습 데이터 셋으로 사용하는 과정을 순차적으로 k번 반복하여, 주어진 데이터 셋 전체에 대한 검증을 실시하게 된다. 본 논문에서는 데이터 셋을 3개로 분할하여 3-배 교차검증 과정을 통해 탐지 성능을 평가하였다. 성능 평가의 결과는 Table 1과 같다.

교차검증을 통한 실험 결과 98%의 재현율로 악성 앱을 악성 앱이라고 탐지하였고, 모든 정상 앱과 악성 앱에 대해서는 98.05%의 정확도로 분류하였다.

1) 관련 연구와의 비교 평가

제안하는 방법의 성능을 정적 분석의 접근 방식으로 악성 앱을 탐지하는 연구인 Kirin, RCP, Peng의 연구 그리고 DRBIN의 탐지 성능과 비교하였다. 제안하는 방법의 성능은 DREBIN의 데이터 셋을 통해서 측정하였으며, 관련 연구들의 성능은 DEBIN의 실험 결과에서 참조하였다[12]. 실험 결과는 Fig. 8의 ROC 곡선으로 표현한다. ROC 곡선은 민감도(sensitive)와 특이도(specificity)가 어떤 관계를 갖고 변하는지를 이차원 평면상에 표현한 것으로 악성 앱을 얼마나 잘 찾아내는가 하는 기준을 민감도(Y축), 정상 앱을 얼마나 잘 분류하는가 하는 기준을 특이도(X축)로 나타낸다. 탐지

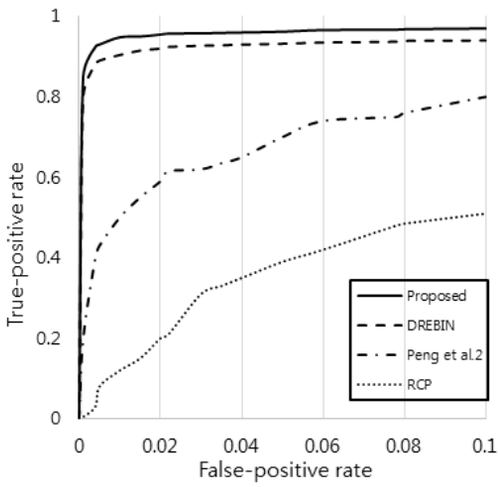

Fig. 8. Detection Performance as ROC Curve

방법의 정확도는 ROC 곡선 아래의 면적(AUC, area under curve)에 의해 측정될 수 있는데, 면적이 넓을수록 좋은 방법이라고 볼 수 있다. 제안하는 방법은 다른 접근 방법보다 좋은 정확도를 보이며, 1%의 FP 비율에서 약 98%의 정확도로 악성 앱을 탐지하였다.

2) 학습하지 않은 패밀리의 악성 앱 탐지 평가

머신러닝 기반 악성 앱 탐지 방법의 가장 큰 장점 중 하나는 학습 데이터로부터 스스로 규칙을 생성하는데 있다. 본 실험에서는 제안하는 방법이 학습하지 않은 악성 앱 패밀리에 대해서도 좋은 탐지 성능을 가지지 확인하고자 하였다. Table 2는 실험에 사용한 DREBIN 데이터 셋에서 가장 개수가 많은 20개의 악성 앱 패밀리를 나타낸다. 실험은 대상 악성 앱 패밀리를 제외한 19개의 악성 앱 패밀리로 딥러닝 모델을 학습한 후, 대상 악성 앱 패밀리의 탐지율을 측정하였다. 실험 결과는 Fig. 9와 같다. 왼쪽의 그래프는 DREBIN의 실험 결과를 나타내고 오른쪽 그래프는 제안하는 방법의 실험 결과를 나타낸다.

DREBIN은 앱을 정적 분석하여 앱에서 요구하는 하드웨어, 퍼미션, API 정보 등을 획득하고 머신러닝 기법 중 하나인 SVM(Support Vector Machine)을 이용하여 악성 앱을 탐지하였다. SVM 기반의 접근은 학습 데이터의 특성을 n차원의 평면에 위치하고, 이러한 학습 데이터 점들을 나눌 수 있는 경계를 찾아 분류하는 지도 학습 방법이다. 이러한 접

Table 1. Classification Accuracies with 3-fold Cross Validation

Data set	Malicious apps		Benign apps		Recall(%)	Precision(%)	Accuracy(%)
	TP	FN	TN	FP			
1	2,058	42	2,061	39	98.00	98.14	98.07
2	2.055	45	2,058	42	97.86	97.99	97.93
3	2.060	40	2.063	37	98.09	98.24	98.17
Overall	6,173	127	6,182	118	97.98	98.12	98.05

Table 2. Top Malware Families in DREBIN Dataset

Id	Family	#	Id	Family	#
A	FakeInstaller	925	K	Adrd	91
B	DroidKungFu	667	L	DroidDream	81
C	Plankton	625	M	LinuxLotoor	70
D	Opfake	613	N	GoldDream	69
E	GingerMaster	339	O	MobileTx	69
F	BaseBridge	330	P	FakeRun	61
G	Iconosys	152	Q	SendPay	59
H	Kmin	147	R	Gappusin	58
I	FakeDoc	132	S	Imlog	43
J	Geinimi	92	T	SMSreg	41

Fig. 9. Detection of Unknown Families

근은 학습되는 데이터 점의 위치에 따라서 최적의 경계를 찾는 방법이기 때문에 악성 앱의 악성 행위 자체를 학습한다고 보기는 어렵다. 따라서 학습하지 않은 패밀리에 대해서는 데이터 점의 위치가 크게 바뀔 수 있게 때문에 좋은 성능을 보이지 않았다. 반면에, 본 논문에서 제안하는 방법의 경우에는 딥러닝 모델을 구성하여 어떤 특성이 악성 행위에 연관되어 있는지 가중치를 가지기 때문에 평균 97%의 정확도를 가지고 학습하지 않은 패밀리를 탐지할 수 있었다. 되어 있는지 학습시키고자 하였기 때문에 학습하지 않은 패밀리의 악성 앱 탐지에 대해서도 좋은 성능을 보였다.

4.3 실행 시간 성능 평가

모바일 장치의 컴퓨팅 성능은 급속히 증가하고 있지만, 일반 데스크톱 컴퓨터에 비해 여전히 제한적이다. 따라서 모바일 장치에서 직접 실행하는 탐지는 효율적으로 작업을 수행해야 한다. 제안하는 방법의 실행 시간 성능을 평가하기 위해 구글 공식 스토어에서 애플리케이션 100개를 무작위로 선택하여 3가지 스마트폰(Nexus 5, Nexus 4, Galaxy S3)과 일반 데스크톱 컴퓨터에서 실행 시간을 측정하였다.

실행 시간 결과는 다음 Fig. 10과 같다. 3가지 스마트폰에

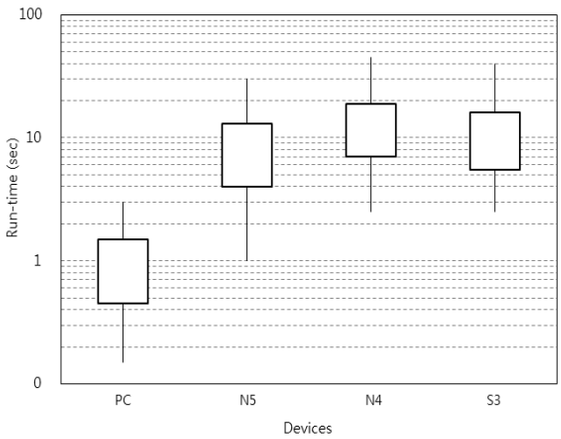

Fig. 10. Run-time Performance of Proposed Method

서 제안하는 방법은 평균적으로 10초 내외에 앱의 분석을 수행하고 탐지 결과를 나타낸다. 크기가 큰 앱에 대해서도 1분 이내에 분석을 완료하였다. Intel(R) Core(TM) i7-3770 @ 3.40GHz CPU와 8GB RAM을 사용한 데스크톱 컴퓨터에서는 평균 1초 이내에 분석을 완료하였다.

Fig. 11은 Galaxy S3 스마트폰에서 dex파일 크기에 따른 실행 시간 성능을 DRBIN과 비교하여 나타낸다. 안드로이드 앱의 구현 정보와 데이터는 APK 파일 내 dex파일에 저장된다. dex 파일의 크기가 커질수록 분석할 데이터의 양이 많아지므로 실행 시간은 dex파일의 크기에 따라서 증가하는 것을 확인할 수 있다. 본 논문에서는 효율적인 특성 선택과 정적 분석 과정의 경량 구현을 통해 DREBIN과 비교하여 보다 효율적인 실행 시간 성능을 달성하였다.

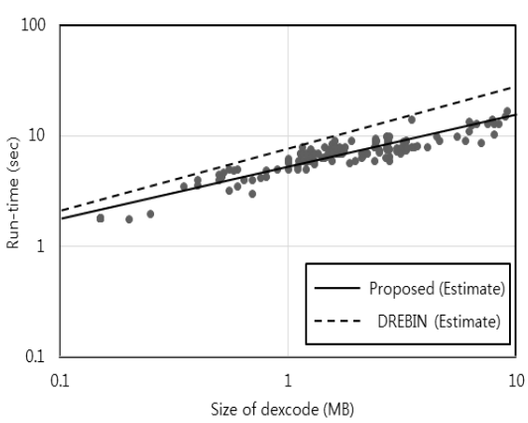

Fig. 11. Detailed Run-time Analysis

5. 결론 및 향후 연구 방향

안드로이드에서 악성 앱의 위협이 꾸준히 증가하는 가운데 백신 프로그램과 같은 기존의 탐지 방법으로는 악성 앱의 방대한 양과 다양성에 충분한 대처가 불가능하다. 빠르게 변하는 악성 앱에 대응하고자 본 논문에서는 앱의 정적

분석과 딥러닝을 결합하여 악성 앱을 탐지하는 방법을 제안하였다. 실제 악성 앱과 정상 앱을 수집하여 제안하는 방법을 평가한 결과 98.05%의 정확도로 악성 앱과 정상 앱을 분류하였고, 학습하지 않은 악성 앱 패밀리에 대해서도 약 97%의 정확도로 탐지하였다. 또한, 데스크톱 컴퓨터에서는 1초 이내에 분석을 수행하고 스마트폰에서도 평균 10초 내외에 분석하는 효율적인 실행 시간 성능을 보였다.

하지만 제안하는 방법이나 DEBIN과 같은 정적분석 기반의 악성 앱 탐지방법은 정적 분석을 기반으로 악성 앱을 탐지하기 때문에 바이트코드 암호화 기법이나 자바 리플렉션 기법과 같은 난독화가 적용된 악성 앱에 대해서는 직접적으로 분석이 불가능하다는 한계점을 가진다. 이러한 앱을 분석하기 위해서는 동적 분석 과정이 필요하다. 본 논문에서는 동적 분석의 부재를 완화하기 위하여 난독화와 정적 분석 회피 기법에 사용되는 API 호출 정보를 수집하였지만, 이는 난독화 기법의 적용 유·무를 확인할 뿐 직접적인 분석은 아니다. 따라서 난독화 기법이 적용된 악성 앱에 대해서도 효과적으로 분석이 가능하도록 향후 추가적인 연구가 필요하다. 또한, 제안하는 방법은 러닝머신 기반의 탐지방법으로 탐지 성능을 위해서는 학습에 사용할 많은 샘플이 확보하는 것이 중요하다. 정상 앱을 수집하는 것은 비교적 간단하지만 악성 앱을 수집하기 위해서는 많은 노력이 필요하다. 악성 앱을 지속적으로 충분히 수집하여 데이터 셋을 갱신할 수 있다면 좋은 성능의 악성 앱 탐지 시스템을 구축할 수 있을 것이다.

References

[1] Kasperskey Lab, 2014, “Mobile Cyber Threats” [Internet], http://media.kaspersky.com/pdf/Kaspersky-Lab-KSN-Report-mobile-cyberthreats-web.pdf.

[2] Gartner, “Worldwide Smartphone Sales to End User by Operating System in 2Q15,” 2015.

[3] McAfee, “Mobile Threat Report, What’s on the Horizon for 2016” [Internet], http://www.mcafee.com/us/resources/reports/rp-mobile-threat-report-2016.pdf.

[4] Felt, Adrienne Porter et al., “A survey of mobile malware in the wild,” *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices,* ACM, 2011.

[5] Sarma, Bhaskar Pratim et al., “Android permissions: a perspective combining risks and benefits,” *Proceedings of the 17th ACM symposium on Access Control Models and Technologies,* ACM, 2012.

[6] Aafer, Yousra, Wenliang Du, and Heng Yin, “DroidAPIMiner: Mining API-level features for robust malware detection in android,” *International Conference on Security and Privacy in Communication Systems*, Springer International Publishing, 2013.

[7] Felt, Adrienne Porter et al., “Android permissions demystified,” *Proceedings of the 18th ACM conference on Computer and communications security,* ACM, 2011.

[8] Rastogi, Vaibhav, Yan Chen, and William Enck, “AppsPlayground: automatic security analysis of smartphone applications,” *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*, ACM, 2013.

[9] Enck, William et al., “TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones,” ACM Transactions on Computer Systems (TOCS), Vol.32, No.2, p.5, 2014.

[10] Yan, Lok Kwong, and Heng Yin, “Droidscope: seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis,” *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*. 2012.

[11] Peiravian, Naser, and Xingquan Zhu, “Machine learning for android malware detection using permission and api calls,” *2013 IEEE 25th International Conference on Tools with Artificial Intelligence,* IEEE, 2013.

[12] Arp, Daniel et al., “DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket,” *NDSS,* 2014.

[13] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton, “Deep learning,” *Nature,* Vol.521, No.7553, pp.436-444, 2015.

[14] Zhou, Yajin, and Xuxian Jiang, “Dissecting android malware: Characterization and evolution,” *2012 IEEE Symposium on Security and Privacy*, IEEE, 2012.

[15] Crussell, Jonathan, Clint Gibler, and Hao Chen, “Attack of the clones: Detecting cloned applications on android markets,” *European Symposium on Research in Computer Security*, Springer Berlin Heidelberg, 2012.

[16] Indyk, Piotr, and Rajeev Motwani, “Approximate nearest neighbors: towards removing the curse of dimensionality,” *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing,* ACM, 1998.

[17] TLSH [Internet], https://github.com/trendmicro/tlsh.

[18] TensorFlow [Internet], https://www.tensorflow.org/.

[19] Srivastava, Nitish et al., “Dropout: a simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, Vol.15, No.1, pp.1929-1958, 2014.

[20] Contagio Community [Internet], http://contagiominidump.blogspot.com.

유 진 현

e-mail : gnyun@korea.ac.kr

2014년 전북대학교 컴퓨터공학부(학사)

2015년~현 재 고려대학교 정보보호대학원 정보보호학과 석사과정

관심분야 : Mobile Security, Software Security, Machine Learning

서 인 혁

e-mail : inhack@korea.ac.kr
2015년 한양대학교 ERICA캠퍼스 컴퓨터공학과(학사)
2015년～현 재 고려대학교 정보보호대학원 정보보호학과 석사과정
관심분야 : Software Testing, Software Security, Machine Learning

김 승 주

e-mail : skim71@korea.ac.kr
1994년 성균관대학교 정보공학과(학사)
1996년 성균관대학교 정보보호학과(석사)
1999년 성균관대학교 정보보호학과(박사)
1998년～2004년 KISA 팀장 (舊한국정보보호진흥원)
2004년～2011년 성균관대학교 정보통신공학부 조교수, 부교수
2011년～현 재 고려대학교 사이버국방학과/정보보호대학원 정교수
관심분야 : Security Engineering, Security Threat-Risk Modeling, Security Testing, Security Evaluation, Usable Security