http://dx.doi.org/10.7840/kics.2014.39C.8.738

선형 SVM을 사용한 안드로이드 기반의 악성코드 탐지 및 성능 향상을 위한 Feature 선정

김 기 현*, 최 미 정°

Linear SVM-Based Android Malware Detection and Feature Selection for Performance Improvement

Ki-Hyun Kim*, Mi-Jung Choi°

요 약

최근 모바일 사용자들이 증가하면서 모바일 어플리케이션 또한 계속적으로 증가하고 있다. 모바일 어플리케이션이 증가하면서 사용자들은 모바일 장치에 은행정보, 위치정보, 아이디, 패스워드 등의 민감한 정보들을 저장하고 있다. 따라서 최근에는 PC를 타겟으로 하는 악의적인 어플리케이션보다 모바일 장치를 타겟으로 하는 악의적인 어플리케이션들이 증가하고 있는 추세이다. 특히 안드로이드 플랫폼의 경우 오픈 플랫폼으로써 사용자들에게 악성코드를 포함한 어플리케이션을 배포하기 유리한 환경을 가지고 있다. 본 논문에서는 안드로이드 환경에서 악성코드를 포함한 어플리케이션을 탐지하기 위해 선형 SVM(Support Vector Machine) 기계학습 분류기를 적용한 악성코드 탐지 시스템의 성능을 분석한다. 또한 모바일 악성코드의 탐지 성능 향상을 위한 feature를 제시하고, 의미있는 feature를 선정한다.

Key Words : Android, Malware Detection, Feature Selection, SVM(Support Vector Machine)

ABSTRACT

Recently, mobile users continuously increase, and mobile applications also increase As mobile applications increase, the mobile users used to store sensitive and private information such as Bank information, location information, ID, password on their mobile devices. Therefore, recent malicious application targeted to mobile device instead of PC environment is increasing. In particular, since the Android is an open platform and includes security vulnerabilities, attackers prefer this environment. This paper analyzes the performance of malware detection system applying linear SVM machine learning classifier to detect Android malware application. This paper also performs feature selection in order to improve detection performance.

Ⅰ. 서 론

최근 다양한 스마트폰이 출시되면서, 스마트폰을 포함한 모바일 사용자의 수가 급증하는 것을 볼 수 있다. 2013년 4월 기준 국내 스마트폰 가지입자 수는 약 3,500만 명을 넘어섰으며 이는 스마트폰 도입 초기인 2009년 11월 기준 47만 명에 비해 무려 74배가 증가한 것으로 스마트폰 시장이 빠르게 성장하고 있음을

※ 이 논문은 2013년도 정부(미래창조과학부)의 재원으로 한국연구재단의 기초연구사업 지원을 받아 수행된 것임 (2013R1A1A3011698)
◆ First Author : Dept. of Computer Science, Kangwon National University, kkh1258@kangwon.ac.kr, 학생회원
° Corresponding Author : Dept. of Computer Science, Kangwon National University, mjchoi@kangwon.ac.kr, 종신회원
논문번호 : KICS2014-05-216, Received May 31 2014; Revised July 10, 2014; Accepted July 10, 2014

알 수 있다.[1] 스마트 시장의 발전과 함께 모바일 어플리케이션이 증가하면서 모바일 단말에 개인정보, 은행정보, 위치정보, 아이디, 비밀번호 등의 민감한 개인정보들이 저장되기 시작하였다. 이와 같이 모바일 장치에 민감한 정보들이 저장됨에 따라 기존에는 PC환경에서 악의적인 어플리케이션을 배포하는 공격이 이루어진 반면, 최근에는 모바일 장치를 타겟으로 하는 공격들이 증가하고 있다. 그 중에서도 악성코드를 포함한 악의적인 어플리케이션을 통한 공격이 주로 이루어지고 있다.

악의적인 어플리케이션에 의한 피해를 최소화하기 위해 모바일 플랫폼에서 악성코드를 분석하여 탐지하는 연구가 활발히 이루어지고 있다. 기존 연구 중에서 악성코드 탐지를 위한 연구는 크게 시그니처 기반 분석 방법[2,3]과 행동기반 분석 방법[4-7]으로 분류할 수 있다. 시그니처 기반 분석 방법은 특정 문자열이나 패턴을 시그니처로 정의하여 악성코드를 탐지하는 방법으로 기존 PC 환경에서 많이 적용되던 방법이다. 하지만 시그니처 기반 분석 방법은 시그니처를 생성하고 관리하는 대용량의 저장소가 필요하므로 모바일 환경에 적용하기 부적합하다. 또한 새로운 악성코드에 대한 시그니처가 없을 경우 악성코드를 탐지할 수 없다는 한계점을 가지고 있다. 행동 기반 분석 방법은 프로세스의 정상적인 상황과 비정상적인 상황의 feature 값을 수집하여 공격 패턴을 분석한 뒤, 침입 탐지 여부를 분석하는 방법이다. 특히 새로운 악성코드에 대한 시그니처의 생성 없이 악성코드를 탐지할 수 있는 장점 때문에 최근 가장 주목 받는 연구 중 하나이다.

모바일 장치에 사용되는 모바일 운영체제에는 iOS, 안드로이드, 윈도우, 블랙베리, 심비안 등이 있다. 모바일 운영체제 중에서 세계적으로 유명한 운영체제는 안드로이드와 iOS이다. 특히 안드로이드는 오픈 플랫폼으로 다른 운영체제에 비해 많은 보안 취약점을 가지고 있다[5]. 안드로이드 플랫폼은 어플리케이션을 설치할 때, 안드로이드 마켓뿐만 아니라 인터넷 블로그 또는 블랙마켓으로부터 다운로드 받아 바로 설치가 가능하기 때문에 악성코드 피해가 더 증가하고 있다[8].

실제로 안철수 연구소 ASEC 리포트[9]에 따르면 2013년 1년간 총 132만 6139건이 악성 어플리케이션으로 진단됐다. 이중 상반기에 67만 3599건, 하반기에 65만 2540건이 추가로 진단됐다. 그림 1은 2013년 월별 모바일 악성코드 접수량을 보여준다. F-Secure 보고서[10]에 따르면 안드로이드 운영체제에서 실행되도록 설계된 악성코드는 2014년 1분기(1월~3월) 동안 나타난 새로운 악성 어플리케이션의 99%를 차지한다. 나머지 1%는 iOS와 심비안에서 발생하였다. 이처럼 악성코드가 계속 증가하고 있으며 특히, 안드로이드 플랫폼을 대상으로 공격자들이 악성코드를 포함한 어플리케이션을 계속 배포하고 있다. 따라서 모바일 환경에서 악성코드를 탐지하는 방법에 대한 연구와 탐지 시스템 개발이 필요하다. 본 논문에서는 기계학습 분류기 중 문서 분류, 트래픽 분류에 많이 사용되는 선형 SVM(Support Vector Machine) 분류기를 적용한 악성코드 탐지 시스템을 제안한다. 제안한 시스템의 악성코드 분석 성능을 보이고, 성능 향상을 위한 feature 선정에 대해서 설명하고 feature 선정 결과 향상된 성능 결과를 보인다.

그림 1. 2013년 월 별 모바일 악성코드 접수량
Fig. 1. Monthly receipt rate of mobile malware in 2013

Ⅱ. 관련 연구

본 장에서는 기존 모바일 환경에서 악성코드를 탐지하는 기법 중 행동 기반 분석 방법을 적용한 논문에 대해 소개하고자 한다. 또한 기존의 분석방법과 본 논문에서 제안하고자 하는 방법의 차이를 설명한다.

[4]는 행동 기반 분석 방법을 사용한 대표적인 논문이다. 스마트폰 기기에서 88개의 feature를 추출하여 Logistic Regression, Decision Tree, Bayesian Network, Naive Bayes와 같은 기계학습 분류기를 통해 정상 어플리케이션과 비정상 어플리케이션을 분류한다. 88개의 feature는 Touch Screen, Keyboard, Scheduler, CPU Load, Message-ing, Power, mMemory, Application, Calls, Operating System, Network, Hardware, Binder, LEDS로 크게 14가지로 분류할 수 있다. 하지만 이와 같이 스마트폰 기기에서 88개의 feature를 추출할 경우 feature를 수집하는 시간과 저장하는 공간에 대한 오버헤드와 탐지를 위한 수행 시간이 길어지는 단점이 있다. 또한 위에서 제시

한 일부 feature의 경우 현재 안드로이드 버전에서 추출하기 위해서는 루팅을 수행해야만 한다. 따라서 본 논문에서는 루팅을 수행하지 않고 수집할 수 있는 feature를 사용한다. 기존 연구[11]에서 feature 값을 분석 서버로 전송하여 분석을 수행하였는데, 본 논문에서도 스마트폰 기기에서 추출한 feature를 스마트폰에 저장하지 않고 분석 서버로 전송하여 분석을 수행함으로써 저장용량 측면의 오버헤드 문제를 해결한다.

[5]에서 수행한 연구 또한 행동 기반 분석 방법을 사용한 악성코드 탐지 연구이다. [5]에서는 32개의 feature를 추출하고 선형 SVM 분류기를 적용하여 정상 어플리케이션과 비정상 어플리케이션을 분류한다. 32개의 feature는 메모리, CPU, 네트워크, 전화, 메시지, 배터리, 프로세스로 크게 7가지로 분류 할 수 있다. 또한 feature 중에서 중요한 요소인 메모리 분야를 Dalvik과 Native로 나누어 실험하였다. 하지만 [5]의 연구에서는 Dalvik과 Native를 나누었을 때와 나누지 않았을 때의 분석 성능에 대한 언급이 없었다. 따라서 본 논문에서는 Dalvik, Native, Other로 나누는 것이 비정상 어플리케이션 탐지에 영향을 주는지 분석하기 위해 메모리 영역을 나누었을 때와 나누지 않았을 때로 나누어 실험을 수행하였다. 또한 [5]의 연구에서는 성능 향상을 위해 information gain 알고리즘을 사용한 feature selection을 수행하여 상위 10개의 feature를 표로 나타내었다. 하지만 feature selection 실험 수행 시 10개, 15개, 20개와 같이 임의로 feature의 수를 결정하여 feature selection을 수행했다는 한계점이 존재한다. 본 논문에서는 40개의 feature 중 최적의 feature 및 feature의 개수를 찾기 위해 feature selection을 수행하고 최적의 feature에 대한 결과를 나타내었다. 자세한 내용은 4장에서 서술한다.

Ⅲ. 선형 SVM 기반의 악성코드 탐지 방법

본 장에서는 스마트폰 기기에서 40개의 feature를 추출하여 선형 SVM 분류기를 사용한 악성코드 탐지 과정을 설명한다. 또한 수집한 데이터 셋과 사용한 어플리케이션에 대한 설명과 비정상 어플리케이션을 탐지하기 위해 수집한 feature를 나타낸다.

3.1 악성코드 탐지 시스템의 구조

본 논문에서는 안드로이드 악성코드를 탐지하기 위해 기계학습 분류기 중 하나인 선형 SVM을 적용한 탐지 시스템을 설계하였다. 악성코드 탐지 과정은 데이터 수집, 학습, 탐지와 같은 세 가지의 과정으로 나누어진다. 그림 2는 본 논문에서 제안한 악성코드 탐지 시스템의 구조를 나타내고 있다.

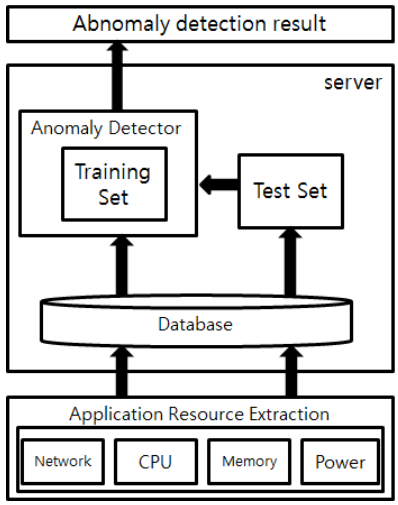

그림 2. 악성코드 탐지 시스템 구조
Fig. 2. Malware detection system architecture

첫 번째, 데이터 수집 과정은 스마트폰 기기에서 정상 어플리케이션과 비정상 어플리케이션을 24시간 동안 30초의 간격으로 표 1에서 나타낸 어플리케이션들을 하나씩 실행한다. 데이터의 추출은 4개의 안드로이드 스마트폰 기기에서 각각 feature들을 하나의 벡터 형태로 통합하며, 통합된 feature를 서버로 전송하여 서버의 데이터베이스에 저장한다. 여기에서 사용한 안드로이드 스마트폰 기기들은 2.3.3 버전을 사용한다.

두 번째, 학습 과정은 데이터 수집 과정에서 추출한 데이터를 이용하여 트레이닝 셋을 구성한다. 4개의 스마트폰 기기 중에서 세 개의 스마트폰 기기의 데이터를 모아서 트레이닝 셋을 구성한다. 이와 같이 구성한 학습 데이터를 기계학습 분류기의 트레이닝 셋으로 적용되며, 학습을 시킨 트레이닝 셋은 anomaly detector로써 작용한다.

세 번째, 탐지 과정은 데이터 수집 과정에서 모은 데이터 중 트레이닝 셋에 사용되지 않은 나머지 하나의 기기에서 추출한 데이터를 모두 모아 테스트 셋을 구성한다. 구성한 테스트 셋을 anomaly detector에 적용시켜 악성코드 탐지 결과를 얻을 수 있다.

3.2 사용한 데이터 셋

본 논문에서 악성코드 탐지를 위해 정상 어플리케이션 9개와 악성코드를 포함한 어플리케이션 10개를 사용하였다. 표 1은 어플리케이션에 대한 정보이다. 이 어플리케이션들은 정상이든 비정상이든 현재 가장

많이 사용되고 있는 어플리케이션들이다. 또한 표 2는 4가지 악성코드 유형에 따라 비정상 어플리케이션을 분류한 정보이다.

각 스마트폰 기기에서 수집한 데이터의 수는 38,000개이다. 4개의 스마트폰 기기 중에서 3개의 스마트폰 기기의 데이터를 모아 트레이닝 셋을 구성하고 나머지 하나의 스마트폰 기기에서 테스트 셋을 구성한다. 테스트 셋을 구성하는 과정은 각각의 스마트폰 기기가 번갈아가며 수행된다. 즉, 네 개의 기기 중 에서 한 개의 스마트폰에서 수집된 데이터는 테스트 셋으로 사용하고 나머지 기기에서 수집된 데이터는 트레이닝 셋으로 사용하게 된다. 이렇게 각각의 스마트 폰이 테스트 셋으로 하여 탐지율에 대해서 평균을 구한 것이다. 이렇게 만들어진 4개의 트레이닝 셋 각각의 데이터의 수는 114,000개이다. 또한 4개의 테스트 셋 각각의 데이터의 수는 38,000개이다. 4개의 트레이닝 셋과 테스트 셋을 만든 이유는 4-Folds Cross Validation을 사용하였기 때문이다. 정상 데이터와 비정상 데이터의 비율은 1:1로 구성하였다.

표 1. 사용한 어플리케이션의 종류
Table 1. Normal/Abnormal Application Lists

Data type	Application name
Normal application	Browser
	Band
	Calculator
	Calender
	Facebook
	Kakao Story
	Kakao Talk
	Naver Search
	Twitter
Abnormal application	Basebridge
	DroidKungFu
	FakeInst
	Geimini
	GoldDream
	Opfake
	PjApps
	Rooter.BT
	SMSHider
	Snake

표 2. 비정상 어플리케이션의 악성코드 유형별 분류
Table 2. Malware Categorization of Abnormal Applications

Malware type	Application name
Trojan	DroidKungFu
	Opfake
	FakeInst
	GoldDream
Spyware	Geimini
	Snake
	PjApps
Exploit	Rooter.BT
	Basebridge
Dropper	SMSHider

3.3 모니터링에 사용한 feature

어플리케이션의 모니터링 과정에서 feature를 추출하는데 사용한 feature는 네트워크, CPU, 프로세스, 이벤트 감지 및 디스패치, 메모리로 크게 5개의 유형으로 분리하며 총 40개의 feature를 추출한다.

표 3은 모니터링에 사용한 40개의 feature를 나타낸다. 메모리 영역의 경우 Dalvik, Native, Other로 나누어 세분화하여 feature를 모니터링 하였다.

스마트폰 기기에서 추출된 40개의 feature는 하나의 벡터 형태로 벡터화 하여 데이터베이스에 저장된다. 벡터 값으로 저장된 데이터는 그림 3과 같은 형식을 갖는다. 그림 3의 형식은 선형 SVM 분류기에 적

표 3. 모니터링 대상 feature
Table 3. Monitored Features

Resource type	Monitoring Feature	
Network	RxBytes, TxBytes	
CPU	Vmpeak, Vmsize, Vmlck, Vmhwm, Vmrss, Vmdata, Vmstk, Vmexe, VmLib, VmPIE, Threads, Voluntary, nonVoluntary	
Process	Process Name	
Event detection and dispatch	Views, Views_root, App_context, Activities, Assets_manage, Assets, Local_binder, Proxy_binder, Death_Recipients, OpenSSL	
Memory	Native	Native_size, Native_allocated, Native_free, Native_pss, Native_shared, Native_priv,
	Dalvik	Dalvik_size, Dalvik_allocated, Dalvik_free, Dalvik_pss, Dalvik_shared, Dalvik_priv,
	Other	Other_pss, Other_shared, Other_priv

```
Normal  1:36572 2:25969 3:237576 4:171528
```

그림 3. Feature의 벡터화
Fig. 3. Feature Vectorization

용하기 위한 데이터 모델이다.

Ⅳ. 성능 향상을 위한 최적의 Feature 선정

본 장에서는 악성코드 탐지 성능을 향상시킬 수 있는 최적의 feature를 선정하기 위해 feature selection을 수행하는 과정에 대해 설명한다. 또한 기존 연구에서 메모리 영역을 Dalvik과 Native로 따로 분류한 실험 결과가 실제로 타당한지 증명하기 위해 메모리 영역을 분할한 결과와 분할하지 않고 결합했을 때의 탐지율 결과를 비교 분석한다. 여기서 말하는 탐지율이란 정답률로서 표 1에서 종류별로 나눈 각 어플리케이션들을 자기 자신으로 어느 정도 비율로 정확히 찾아내는지의 비율을 의미한다.

4.1 Feature selection

본 논문에서는 안드로이드 환경의 악성코드 탐지를 위해 40개의 feature를 사용한다. 40개의 feature는 기존의 연구와 비교해 볼 때 많은 편은 아니다. 하지만 현재 사용하고 있는 40개의 feature 중에는 성능에 많은 영향을 주는 feature가 있을 수 있고, 성능에 영향을 거의 주지 않는 feature가 있을 수 있다. 또한 40개의 feature를 추출할 때 오버헤드를 발생 할 수 있기 때문에 불필요한 feature는 제거해야 할 필요가 있다. 따라서 불필요한 feature를 제거하고 최적의 feature를 선정하기 위해 feature selection을 수행한다.

Feature selection은 일단 모든 feature를 사용하였을 때의 탐지 결과를 측정한다. 그 후 각각의 feature를 하나씩 제거하여 악성코드 탐지 결과를 측정한다. 그림 3과 같이 각 feature에 붙여진 숫자에 해당하는 feature 값을 하나씩 제거한 후 각각의 탐지 결과 중 가장 높은 성능을 갖는 트레이닝 셋과 테스트 셋을 선택한다. 즉, 1개의 feature를 제거했을 때 가장 탐지율이 높은 feature를 제거하는 것이다. 1개의 feature를 제거한 데이터 셋과 테스트 셋에서 또 다시 하나씩의 feature 값을 제거한다. 탐지 결과 중 가장 높은 성능을 갖는 트레이닝 셋과 테스트 셋을 선택한다. 즉, 두 개의 feature를 제거했을 때 성능이 가장 높은 테스트 셋과 데이터 셋을 선택한다. 위와 같은 과정을 반복하여 탐지 결과가 이전 결과 결과보다 낮아질 때 까지 feature를 하나씩 제거하며 최적의 탐지율이 나올 때까지 반복하여 수행한다.

Feature selection 이 후 최적의 feature를 선정한 데이터 셋은 SVM 분류기를 사용하여 악성코드의 탐지 성능을 더 향상 시킬 수 있다. 악성코드 탐지 시스템의 실험 결과는 5장에서 서술한다.

4.2 메모리 영역 분할과 결합

우리는 기존 연구[5]에서 악성코드를 탐지하는데 중요한 요소인 메모리 영역을 나누어 사용하였다. 메모리 영역은 Native, Dalvik, Other로 세분화할 수 있다. 메모리 영역을 나누었을 때와 나누지 않고 전체 하나의 영역으로 결합하였을 때의 비정상 탐지율을 살펴보고자 한다. 메모리 영역을 나누어 사용할 경우 총 feature의 개수는 40개이고, 메모리 영역을 나누지 않을 경우 총 feature의 개수는 31개이다. 메모리를 나누었을 때와 합쳤을 때의 데이터를 트레이닝 셋과 테스트 셋으로 구성한 뒤 악성코드 탐지 시스템에 적용시켜 메모리 영역을 세분화하여 feature를 구성하는 것이 악성코드 탐지율에 어떤 영향을 끼치는지 분석하였다. 실험 결과는 5장에서 서술한다.

Ⅴ. 실험 결과

본 장에서는 4장에서 제안한 내용을 토대로 실험한 결과에 대해 서술한다. 첫 번째 실험 결과는 feature selection을 수행하였을 때 성능 향상을 나타낸다. 두 번째 실험 결과는 메모리 영역을 세분화 하였을 때와 메모리 영역을 결합하였을 때의 성능 차이를 나타낸다.

5.1 성능 지표

본 장에서는 실험 결과의 성능을 증명하기 위해서 사용된 성능 지표에 대해 설명한다. 본 장에서는 실험 결과의 성능을 증명하기 위한 성능 지표에 대해서 설명한다. 본 논문에서 사용한 성능지표는 TPR(True Positive Rate), FPR(False Positive Rate), Precision, Accuracy이다. 각 성능지표를 구하기 위해서는 판단 결과에 대한 통계 정보가 필요하다. 표 4는 성능지표를 계산하기 위한 Confusion Matrix 이다.

TP(True Positive)는 정상 어플리케이션을 정상 어플리케이션으로 판단한 수치를 의미한다. TN(True Negative)은 악성코드가 포함된 어플리케이션을 악성코드가 포함된 어플리케이션으로 판단한 수치를 의미한다. 또한 FN(False Negative)은 실제 정상 어플리케이션임에도 불구하고 악성코드가 포함된 어플리케이션으로 판단한 수치를 의미한다. FP(False Positive)는

표 4. 성능지표 계산을 위한 Confusion Matrix
Table 4. Confusion Matrix for Performance Index Calculation

		Predicted data	
		Positive	Negative
Actual data	Positive	TP(True Positive)	FN(False Negative)
	Negative	FP(False Positive)	TN(True Negative)

실제 악성코드가 포함된 어플리케이션임에도 불구하고 정상 어플리케이션으로 판단한 수치를 의미한다. 위의 통계 정보를 바탕으로 본 논문에서는 TPR(True Positive Rate), FPR(False Positive Rate), Precision, Accuracy를 구한다. 각 지표에 대한 수식은 다음과 같다.

$$TPR: \frac{TP}{TP+FN} \times 100 \quad (1)$$

$$FPR: \frac{FP}{FP+TN} \times 100 \quad (2)$$

$$\mathrm{Precision}: \frac{TP}{TP+FP} \times 100 \quad (3)$$

$$Accuracy: \frac{TP+TN}{TP+TN+FP+FN} \times 100 \quad (4)$$

TPR은 정상 어플리케이션을 Positive라고 가정하였을 때, 실제 정상 어플리케이션 중에서 정상 어플리케이션으로 판단한 비율(1)을 의미한다. FPR은 악성코드가 포함된 어플리케이션 중에서 정상 어플리케이션으로 잘못 판단한 비율(2)을 의미한다. Precision은 판단치의 오차를 나타내는 지표로, 정상 어플리케이션으로 판단한 것 중에서 실제 정상 어플리케이션의 비율(3)을 의미하며, Accuracy는 시스템의 정확도를 나타내는 지표로, 정상 어플리케이션과 비정상 어플리케이션을 각각 정확하게 판단한 비율(4)을 의미한다.

5.2 Feature Selection 수행 결과

Feature selection의 결과를 나타내기 위해 4개의 기기에서 추출한 트레이닝 셋과 데이터 셋 각각의 feature selection을 수행한 후, 4-Folds Cross Validation을 수행하였다. 표 3은 feature selection의 수행 결과를 나타낸다.

4.1장에서 설명 하였듯이, 40개의 feature중에서 각 feature를 하나씩 제거하고, 그 중에서 TPR 측면에서 가장 높은 성능 향상을 보인 feature를 제거할 feature로 정한다. 그 다음으로 39개의 feature중에서 다시 각 feature를 하나씩 제거하여 성능을 측정한 후에, 가장 높은 TPR 값을 갖는 feature를 제거할 feature로 선정한다. 위와 같은 방법을 반복하여 성능이 낮아지는 단계까지 실험을 수행한다.

표 5에 따르면 feature selection을 수행한 결과 제거한 feature의 수는 총 40개 중에서 5개의 feature를 제거하였을 때, 가장 높은 결과가 나왔다. 5개 이상 제거를 하였을 때, 성능은 점점 떨어지기 시작한다. 제거한 feature의 종류는 Proxy_binder, OpenSSL, native_free, dalvik_allocated, VmRss 이다. TPR은 85.41%에서 96.85%로 약 9%의 성능 향상을 보였고, FPR의 경우 1.37%에서 0.19%로 약 1%가 넘는 향상을 보였다. 또한, precision의 경우 89.15%에서 98.32%로 약 9%의 높은 성능 향상을 보였고, accuracy의 경우 97.99%에서 99.68%로 약 2%의 성능 향상을 보였다.

40개의 feature중 첫 번째로 선정된 Proxy_binder를 제거한 결과 TPR은 약 10%의 성능 향상을 보였고, FPR은 약 1%의 성능 향상을 보였다. 39개의 feature 중 두 번째로 OpenSSL을 제거한 결과 TPR은 약 1%의 성능 향상을 보였고, FPR은 약 0.1%의 성능 향상을 보였다. Proxy_binder는 Proxy를 사용할 때 사용되는 이벤트 중 하나로써, Proxy_binder feature의 값을 어플리케이션에서 추출한 결과, 대부분의 어플리케이션에서 유사한 값들이 추출된다. 유사한 값이 추출되면서 어플리케이션들을 판별하기에 불필요한 feature 중 하나가 될 수 있다. 또한 네트워크, CPU, 메모리, 프로세스에 영향을 주지 않기 때문에 어플리케이션들을 분류하는데 도움이 되지 않는다. OpenSSL은 네트워크를 사용할 때, 암호화를 시켜주

표 5. Feature selection 수행 결과
Table 5. Performance Results of Feature Selection

Feature 제거 수	제거한 feature	TPR	FPR	Precision	Accuracy
0	X	85.41%	1.37%	89.15%	97.99%
1	Proxy _binder	95.32%	0.32%	97.59%	99.51%
2	OpenSSL	96.72%	0.20%	98.26%	99.66%
3	native _free	96.83%	0.19%	98.3%	99.68%
4	dalvik _allocated	96.84%	0.19%	98.31%	99.68%
5	VmRss	96.85%	0.19%	98.32%	99.68%
6	VmData	96.13%	0.16%	98.64%	99.66%
7	Other_pss	96.21%	0.19%	98.06%	99.64%

는 알고리즘으로 사용된다. 하지만 모든 어플리케이션들이 OpenSSL 암호화를 사용하지 않기 때문에 OpenSSL feature를 추출한 모든 값이 0값을 갖는다. 그러므로 어플리케이션 판별에 의미 없는 feature라고 할 수 있다. 향후 추가적으로 악성코드 유형별, 각 종류별 탐지를 위한 최적의 feature를 선정하는 연구가 필요하다.

5.3 메모리 영역 분할과 결합

메모리 영역을 Dalvik과 Native로 분할한 것이 타당한지 알아보기 위해 Dalvik, Native, Other로 분할한 feature로 구성된 트레이닝 셋과 테스트 셋을 악성코드 탐지 시스템에 적용하고, 결합한 feature로 구성한 트레이닝 셋과 테스트 셋을 악성코드 탐지 시스템에 적용하였다. 메모리 영역을 분할한 실험 결과는 표 6과 같다. 또한 메모리 영역을 결합하였을 때의 결과는 표 7과 같다.

분석 결과 메모리 영역을 분할했을 경우, 평균 TPR은 85.52%이고, 평균 FPR은 1.38%이다. 메모리 영역을 결합한 feature의 경우, 평균 TPR은 83.94%로 메모리 영역을 분할했을 때 보다 약 2% 낮은 결과를 보였다. 또한 평균 FPR은 1.70%로 메모리 영역을 분할했을 때 보다 평균 0.4% 높은 결과를 보였다. 각 클래스를 비교해 보았을 때, 가장 높은 차이를 보이는 클래스는 Normal이다. 메모리 영역을 결합하였을 때 Normal 클래스를 보면 95.39%이지만, 메모리 영역을 분할하였을 때 Normal 클래스는 88.58%로 약 7%의 낮은 결과를 보였다. Normal 클래스의 FPR의 차이를 보았을 때, 높은 성능 차이를 보이지는 않지만 메모리를 결합 했을 때 0.5% 더 높은 결과를 보인다. 또한

표 6. 메모리 영역 분할한 feature의 실험 결과
Table 6. Performance Results of Divided Memory Feature

	TPR	FPR	Precision	Accuricy
Normal	95.38%	7.90%	92.41%	93.65%
Snake	75.07%	1.44%	81.88%	97.26%
PjApps	74.10%	0.06%	96.16%	98.51%
Geimini	97.98%	0.78%	89.08%	99.14%
GoldDream	86.9%	0.71%	88.12%	98.59%
FakeInst	83.43%	0.28%	95.41%	98.80%
SMSHider	73.18%	0.00%	99.84%	98.53%
Opfake	99.42%	0.08%	98.54%	99.83%
Rooter.BT	78.78%	0.07%	98.58%	98.73%
Basebridge	73.65%	2.19%	58.04%	96.47%
DroidKungFu	98.67%	1.57%	82.61%	98.43%
Average	85.41%	1.37%	89.15%	97.99%

표 7. 메모리 영역 결합한 feature의 실험 결과
Table 7. Performance Results of Combined Memory Feature

	TPR	FPR	Precision	Accuracy
Normal	88.58%	8.81%	91.81%	89.95%
Snake	73.7%	1.39%	76.18%	97.29%
PjApps	73.52%	0%	99.84%	98.59%
Geimini	98.86%	0.25%	95.67%	99.7%
GoldDream	80.51%	3.43%	61.33%	95.74%
FakeInst	85.41%	0.68%	88.88%	98.58%
SMSHider	76.02%	0%	99.98%	98.73%
Opfake	98.6%	0.55%	92.01%	99.40%
Rooter.BT	90.47%	1.25%	87.37%	98.31%
Basebridge	75.25%	2.13%	58.38%	96.68%
DroidKungFu	82.42%	0.24%	91.36%	98.84%
Average	83.94%	1.70%	85.71%	97.44%

precision은 약 4% 정도의 높은 성능 차이를 보이고, accuracy의 경우 성능 차이를 보이지 않았다. 메모리 영역을 결합하였을 때 보다 메모리 영역을 분할하였을 때 정상을 정상이 아닌 비정상으로 더 많이 분류한다는 것을 알 수 있었다.

실험 결과를 정리하면, 메모리 영역을 결합하여 feature를 추출하였을 때 보다 메모리 영역을 분할하여 feature를 추출하여 악성코드를 탐지하였을 때 더 높은 탐지율을 보였다.

Ⅵ. 결론 및 향후 연구

본 논문에서는 안드로이드 기반의 악성코드 탐지 성능 향상을 위해 중요한 feature 중 하나인 메모리 의 영역을 세 가지 분야로 분할하였을 때와 이를 통합하였을 때의 실험을 수행하여 그 결과를 보였다. 또한 최적의 feature를 선정하기 위해 feature selection을 수행하고 그 결과를 보였다.

앞의 두 실험을 통해, 메모리 분야를 나누었을 때는 통합했을 때보다 TPR이 약 2%의 성능 향상을 보였고, FPR의 경우 약 0.4%의 성능 향상을 보였다. 또한 precision과 accuracy도 약간의 성능 향상을 보인 것을 알 수 있다.

Feature selection의 경우 총 40개의 feature에서 5개의 feature를 제거하였을 때, TPR은 약 9%의 높은 성능 향상을 보였으며, FPR의 경우 약 1%의 성능 향상을 보였다. 또한 precision은 약 9%의 높은 성능 향상과 accuracy는 2%의 성능 향상을 보였다. Feature selection은 Proxy_binder, OpenSSL, native_free, dalvik_allocated, Vmrss feature를 제거 하였을 때의

성능이 최고 성능이라는 것을 알 수 있다. 위 두 실험을 통해 feature들 중에서 메모리

분야의 feature를 합쳤을 때 보다 나누었을 때 더 높은 성능이 나온다는 타당성을 제시하였고, 메모리 분야를 나누고, feature selection을 수행하였을 경우 더 높은 성능 향상을 보인다는 결과를 보였다.

향후 연구로 악성코드 유형별로 비정상을 탐지하기 위한 최적의 feature selection 연구를 수행할 예정이다. 다음으로 클러스터링 기법을 이용한 안드로이드 기반의 악성코드 탐지에 대한 연구를 수행할 것이다. 또한 현재는 2.3.3 안드로이드 버전을 사용하였지만 많은 사용자들이 4.0 이상의 안드로이드 버전을 사용하기 때문에, 4.0 이상의 안드로이드버전에서 악성코드 탐지를 수행할 예정이다.

References

[1] J. Park, and M. Kim, “Usage patterns and market development of smartphone,“ in *Proc. IEIE Conf.*, pp. 572-575, 2013.

[2] M. Choi, C. Jin, and M. Kim, “Classification of client-side application-level HTTP traffic,” *J. KICS*, vol. 36, no. 11, pp. 1277-1284, 2011.

[3] J. Park, S. Yoon, J. Park, H. Lee, S. Lee, and M. Kim, “Performance improvement of the payload signature based traffic classification system,” *J. KICS*, vol. 35, no. 9, pp. 1287-1294, 2010.

[4] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, “Andromaly: A behavioral malware detection framework for android devices,” *J. Intell. Inf. Syst.*, vol. 38, no. 1, pp. 161-190, 2012.

[5] H. S. Ham, and M. J. Choi, “Analysis of android malware detection performance using machine learning classifiers,” *Int. Conf. ICT Convergence*, pp. 490-495, Jeju Island, Korea, Oct. 2013.

[6] R. M. Elbasiony, E. A. Sallam, T. E. Eltobely, and M. M. Fahmy, “A hybrid network intrusion detection framework based on random forests and weighted k-means,” *J. Ain Shams Eng.*, vol. 4, no. 4, pp. 753-762, 2013.

[7] J. Kim and E. Im, “Android malware detection using dynamic analysis in virtual system,” in *Proc. KICS Conf.*, pp. 816-817, 2012.

[8] Y. Fledel, A. Shabtai, D. Potashnik, and Y. Elovici, “Google android: an updated security review,” *Mobile Comput., Appl. Serv.*, Springer Berlin Heidelberg, vol. 76, pp. 401-414, 2012.

[9] AhnLab, Security Center, “ASEC Report 2013,” vol. 48, 2013.

[10] F-Secure, “Mobile Threat Report Q1 2014,” 4, 2014.

[11] B. Amos, H. Turner, and J. White, “Applying machine learning classifiers to dynamic android malware detection at scale,” *IEEE Int. Wirel. Commun. Mobile Computing Conf. (IWCMC)*, pp. 1666-1671, Sardinia, Jul. 2013.

김 기 현 (Ki-Hyun Kim)

2014년 2월 : 강원대학교 컴퓨터과학과 학사

2014년 3월~현재 : 강원대학교 컴퓨터과학과 석사

<관심분야> 네트워크 관리, 정보보안

최 미 정 (Mi-Jung Choi)

1998년 : 이화여자대학교 컴퓨터공학과 학사

2000년 : 포항공과대학교 컴퓨터공학과 석사

2004년 : 포항공과대학교 컴퓨터공학과 박사

2004~2005년 : 프랑스 INRIA 연구소 박사후 연구원

2005~2006년 : 캐나다 워터루대학 컴퓨터과학부 박사후 연구원

2006~2008년 : 포항공대 컴퓨터공학과 연구 조교수

2008년~현재 : 강원대학교 컴퓨터과학과 부교수

<관심분야> 네트워크 관리, 정보보안, SDN/NFV 관리 및 통합