Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.27, NO.5, Oct. 2017 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2017.27.5.1087

Naive Bayes 기반 안드로이드 악성코드 분석 기술 연구*

황 준 호,† 이 태 진‡
호서대학교

Android Malware Analysis Technology Research Based on Naive Bayes*

Jun-ho Hwang,† Tae-jin Lee‡
Hoseo University

요 약

스마트 폰의 보급률이 증가함에 따라 스마트 폰을 대상으로 하는 악성코드들이 증가하고 있다. 360 Security의 스마트 폰 악성코드 통계에 따르면 2015년 4분기에 비해 2016년 1분기에 악성코드가 437% 증가하는 수치를 보였다. 특히 이러한 스마트 폰 악성코드 유포의 주요 수단인 악성 어플리케이션들은 사용자 정보 유출, 데이터 파괴, 금전 갈취 등을 목적으로 하는데 운영 체제나 프로그래밍 언어가 제공하는 기능을 제어할 수 있게 해주는 인터페이스인 API에 의하여 동작하는 경우가 대부분이다. 본 논문에서는 정적 분석으로 도출한 어플리케이션 내 API의 패턴을 지도 학습 기법으로 머신에 학습하여 정상 어플리케이션과 악성 어플리케이션 내의 API 패턴의 유사도에 따라 악성 어플리케이션을 탐지하는 메커니즘을 제시하고 샘플 데이터에 대하여 해당 메커니즘을 사용하여 도출한 label 별 탐지율과 탐지율 개선을 위한 기법을 보인다. 특히, 제안된 메커니즘의 경우 신종 악성 어플리케이션의 API 패턴이 기존에 학습된 패턴과 일정 수준 유사한 경우 탐지가 가능하며 향후 어플리케이션의 다양한 feature를 연구하여 본 메커니즘에 적용한다면 anti-malware 체계의 신종 악성 어플리케이션 탐지에 사용될 수 있을 것이라 예상된다.

ABSTRACT

As the penetration rate of smartphones increases, the number of malicious codes targeting smartphones is increasing. I 360 Security 's smartphone malware statistics show that malicious code increased 437 percent in the first quarter of 2016 compared to the fourth quarter of 2015. In particular, malicious applications, which are the main means of distributing malicious code on smartphones, are aimed at leakage of user information, data destruction, and money withdrawal. Often, it is operated by an API, which is an interface that allows you to control the functions provided by the operating system or programming language. In this paper, we propose a mechanism to detect malicious application based on the similarity of API pattern in normal application and malicious application by learning pattern of API in application derived from static analysis. In addition, we show a technique for improving the detection rate and detection rate for each label derived by

Received(06. 19. 2017), Modified(08. 14. 2017), Accepted(08. 28. 2017)
* 이 논문은 2017년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No. 2017-0-00683, 엔드포인트 포렌식 및 STIX 분석 머신러닝 기반 실시간 신종 악성코드 탐지/제어 시스템)
† 주저자, hwangso93@gmail.com
‡ 교신저자, kinjecs0@gmail.com (Corresponding author)

using the corresponding mechanism for the sample data. In particular, in the case of the proposed mechanism, it is possible to detect when the API pattern of the new malicious application is similar to the previously learned patterns at a certain level. Future researches of various features of the application and applying them to this mechanism are expected to be able to detect new malicious applications of anti-malware system.

Keywords: Malware, Classification, Naive bayes

I. 서 론

스마트 폰의 보급률이 증가함에 따라 스마트 폰을 대상으로 하는 악성코드들이 증가하고 있는데 국내에서 주로 사용하는 안드로이드 운영체제 스마트 폰의 경우에도 마찬가지로 악성코드를 내재한 악성 어플리케이션으로 인해 피해가 큰 폭으로 꾸준히 증가하고 있다. 그러나 기존의 anti-malware 시스템에서와 같이 전문가의 분석을 통하여 악성 어플리케이션에 대한 시그니쳐를 생성하고 이것을 기반으로 동일한 시그니쳐의 어플리케이션을 악성으로 탐지하는 방법의 경우에는 알려진 악성코드에 대해서는 확실한 방법이지만 2016년 기준 발생한 스마트 폰 신규 악성코드는 약 4억개로서 이러한 악성코드에 대해서 전문가가 일일이 분석하는 것은 한계가 있기 때문에 unknown 악성코드, 변종 악성코드에 대해서는 다른 방법이 요구된다. 특히, anti-malware 시스템은 악성코드가 개인과 조직의 자산에 해를 끼치기 전에 탐지 및 대응하는 것이 궁극적인 목표이지만 이러한 기존의 대응 방법은 신규 악성코드에 대해 분석 후 anti-malware 시스템에 대응책을 적용함으로서 신규 악성코드에 대한 피해에 즉각적으로 탐지 및 대응할 수 없으므로 자동화된 신규, 변종 악성코드에 대한 탐지 메커니즘이 필요한 것은 분명하다.

안드로이드 악성코드의 경우 다양한 형태로 존재할 수 있지만 주로 안드로이드 스마트 폰 사용자가 실행하고 사용하는 어플리케이션 내에 악성코드를 삽입 하여 동작하게 된다. 악성코드가 삽입된 악성 어플리케이션의 주요 목적은 사용자 정보 유출, 데이터 파괴, 금전 갈취 등에 있으며 이러한 목적을 수행하기 위해서는 안드로이드 어플리케이션 개발 언어인 java의 특성상 관련 기능을 수행하기 위한 method가 사용된다. 특히, 운영 체제나 프로그래밍 언어가 제공하는 기능을 제어할 수 있게 해주는 인터페이스인 API(Application Programming Interface)는 그 원형이 정해져 있고 디바이스를 정보를 반환하거나 암호화 등의 기능을 제공하여 악용될 여지가 있는 sensitive API를 사용하여 악의적인 목적에 사용하고자 하는 악성 어플리케이션 제작자의 특징을 고려하면 통계학적으로 정상 어플리케이션과 악성 어플리케이션을 구분하는 근거가 될 수 있다. 따라서 본 논문에서는 이러한 API의 통계학적 차이를 이용하여 정상 어플리케이션과 악성 어플리케이션을 구분짓는데 더하여 일정 수준 유사한 API 패턴을 보이는 신규, 변종 악성코드에 대한 탐지도 즉각적으로 대응 가능하도록 결과를 산출하는 메커니즘을 제시하고자 한다. 2장에서는 이러한 메커니즘에 필요한 프로세스인 머신 러닝과 안드로이드 구조에 대하여 기술하고 3장에서는 앞선 프로세스들을 바탕으로 본 논문에서 제안하는 API 패턴 분석을 통한 악성 어플리케이션 탐지 모델에 대하여 제시한다. 4장에서는 이러한 메커니즘에 대한 성능 분석 결과를 제시하며 5장에서는 성능 분석 결과를 토대로 본 메커니즘이 가지는 의미와 결론을 제시한다.

II. 관련 연구

안드로이드 악성코드 탐지를 위한 다양한 연구가 진행되고 있는데 일반적인 네트워크 기반 악성 코드 탐지와 행위 분석 및 특징에 대해 기술하고 탐지 시스템을 제안하는[1], 기존의 정적, 동적 분석 기법에 대해 분석하고 정적, 동적 분석 기법을 혼용한 hybrid 기법으로 악성코드 탐지를 제안 및 그 성능 평가에 대한[2], 머신 러닝 기법을 이용하여 안드로이드 악성코드를 탐지하는 workflow를 제안하는[3], 안드로이드 어플리케이션이 기능을 수행하기 위해서 요구하는 퍼미션을 분석하고 그 기반으로 어플리케이션의 위험도를 측정 및 악성 여부를 결정짓는[4,5], 어플리케이션 정적 분석을 통한 제어, 데이터, 구조 등의 정보로 위험 기능과 취약점을 밝히는 연구[6], 악성과 정상 어플리케이션을 구분하는 것이 아닌 API의 특성으로 악성코드의 위험도를 정량적으로 평가하여 어플리케이션의 위험도를 측정하는 기법[7] 등이 있다. 최근에는 머신 러닝 기법을 사용한 악성코드 탐지 시스템에 대한 연구가 활발하게 진행되고 있는데 기존의 머신 러닝 기반의 악성코드

탐지 시스템의 경우에 SVM(Support Vector Machines)[8, 9, 11]이나 Decision Tree[10]와 같은 머신 러닝 알고리즘을 주로 사용하는데 동일한 분류 범주의 알고리즘인 naive bayes는 이와 같은 진보된 방법의 알고리즘에 비해 간단하고 연산량이 적으며 학습에 필요한 data set이 비교적 적다는 장점이 있다. 반면에 다른 알고리즘에 대해 일반적으로 예측 값에 대한 성능이 부족하다고 평가받는데 naive bayes는 적절한 feature 전처리와 성능 향상을 위한 프로세스가 병행될 때 위의 방법들과 견주어 충분한 경쟁력을 갖출 수 있으며 이런 side effect를 감안할 때 naive bayes 알고리즘은 기존 시스템의 기법의 복잡성은 해소하면서 간단하고 연산량이 적은 방식으로 탐지 시스템을 구축할 수 있다. Fig.1. 은 5개의 알고리즘(SVM, naive bayes, decision tree, kNN, neural network)에 대한 일반적인 성능 비교[12]이다.

앞서 기술한 것과 같이 Fig. 1에 따르면 naive bayes는 다른 알고리즘에 비하여 학습에 필요한 샘플 개수, 분류 속도, 학습 속도 면에서 우수하다. 게다가 baysian 정리와 조건부 확률을 근간으로 하고 있기 때문에 단순히 word의 출현 빈도를 이용하여 통계학적으로 label을 판단하는 도메인의 경우 간단하면서 효과적으로 동작할 수 있다.

본 논문에서 악성 어플리케이션 탐지 프로세스로 사용하는 머신 러닝은 인공 지능의 한 분야로 어떤 문제를 해결하기 위해 머신 러닝 알고리즘을 사용하여 데이터를 분석, 학습하여 학습한 내용을 기반으로 판단이나 예측 값을 산출해 내는 것을 목표로 하는 기법이다.

다음으로 2.1절 에서는 지도 학습의 data set으로 사용될 feature들을 추출하기 위한 안드로이드 DEX파일 및 정적 정보 추출 방법에 대해 기술한다. 2.2절 에서는 naive bayes에 대한 개요 및 추출한 feature를 이용해서 naive bayes에 적용하여 결과를 출력하는 방법을 기술한다.

2.1 Feature select

APK(Application PacKage) 내의 실제 어플리케이션이 실행되는데 필요한 코드들이 들어있는 .dex 파일은 안드로이드 정적 분석 정보를 추출하는데 핵심적인 raw data이다. 본 논문에서는 악성 어플리케이션이 목적으로 하는 악의적인 기능들을 수행하는데 사용되는 API에 대한 통계학적 접근법으로 정상 어플리케이션과의 악성 어플리케이션을 판단하므로 .dex 파일에서 method 정보가 저장되어있는 영역에 접근하여 raw data를 추출한다. 안드로이드 .dex 파일의 구조는 Fig.2.와 같다.

여러 개의 영역으로 나뉘어져 어플리케이션 정보를 관리하는 다음과 같은 구조에서 .dex 파일의 문자열 전체를 관리하고 저장하고 있는 String_IDs와 method 정보를 가지고 있는 Method_IDs 영역의 데이터를 이용하여 method를 추출할 수 있다. 하지만 여기서 추출한 정보는 온전한 API가 아닌 다른 함수가 섞여있기 때문에 안드로이드 레퍼런스의 API 리스트로 parsing하여 API만 추출해 내는 과정 또한 필요하다.

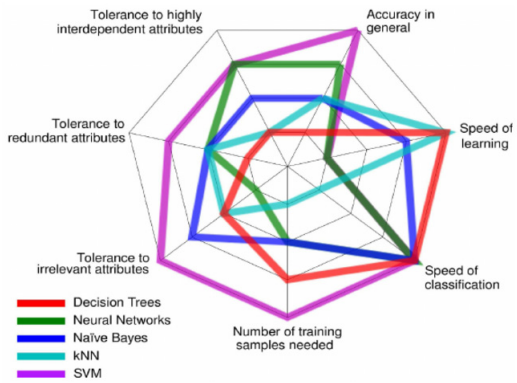

Fig. 1. General Performance Comparison of SVM, Naive Bayes, Decision Tree, kNN, Neural Network

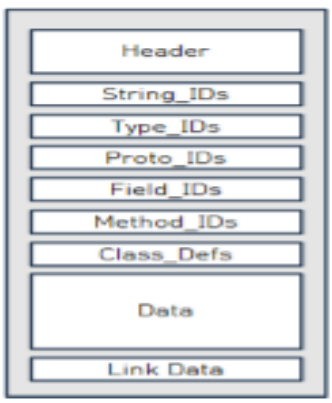

Fig. 2. Android Dex File Format

2.2 Naive Bayes

본 연구에서 머신 러닝 알고리즘으로 사용하는 naive bayes는 전통적으로 스팸 메일 분류 등에 사용되는 확률 분류기의 일종으로 적합한 프로세스를 사용하여 분류기를 설계할 시 다차원을 사용하는 SVM과 같은 분류 범주의 진보된 알고리즘 보다 직관적이며 연산량이 적어 효율적이다. naive bayes는 지도 학습 환경에서 주로 사용하는 알고리즘으로 bayes 정리와 조건부 확률을 이용하여 다음과 같이 나타낼 수 있다.

- c : 클래스 혹은 label
- x : 분류될 인스턴스들의 특성 벡터
- V : 특성 벡터 개수

$$\hat{P}(c|x) = \frac{count(c,x)}{\sum_{x \in V} count(c,x)} \quad (1)$$

Table 1.과 같이 4개의 training set이 존재하고 label은 Normal, Malicious 두 개가 존재한다고 가정하자. API List열은 training set으로 사용되는 어플리케이션에서 각각 사용하는 API들의 리스트를 의미하며 이것은 X(input : 분석하고자 하는 어플리케이션의 API List)에 대해서 bayesian 확률로 label별 유사도 측정을 위하여 사용된다.

Naive bayes는 X가 Normal에 속할 확률과 Malicious에 속할 확률을 계산하고 그 결과를 비교하여 더 큰 값의 확률로 X의 label을 결정하게 된다. X ={ GetdeviceID, SetString, Draw }라고 가정하면 해당 어플리케이션이 Normal일 확률은 수식 (1)에 의하여

Table 1. Training Document

Application	API List	Label
1	SetString, GetString, Draw	Normal
2	GetString, Show	Normal
3	GetDeviceID, SetString, Show	Malicious
4	GetDeviceID, Show, Draw	Malicious

- $APIs$: API List

$$P(Normal|APIs) = \frac{P(APIs|Normal) * P(Normal)}{P(APIs)} \quad (2)$$

Malicious일 확률은

$$P(Malicious|APIs) = \frac{P(APIs|Malicious) * P(Malicious)}{P(APIs)} \quad (3)$$

이다.

X에 대한 수식 (2)와 수식 (3)을 값을 비교하여 더 큰 쪽으로 label을 결정하게 되는데 수식 (1)의 naive bayes 조건부 확률의 문제는 판별하고자 하는 어플리케이션이 document에 없는 word를 사용할 경우이다. 수식 (2)에서 Normal label에는 GetDeviceID가 사용되지 않으므로

$$P(GetDeviceID|Normal) \quad (4)$$

는 0이 되어 0의 곱셈 특성으로 인하여 수식 (2)는 다른 API에 대해 어떠한 값이 나오더라도 0이 되어 버린다. 이 결과는 가지고 있는 training set에 존재 하지 않는 word가 X에 나타날 때의 문제로 수식 (1)을 수식 (5)로 보완해서 사용하여 해결할 수 있다.

$$\hat{P}(c|x) = \frac{count(c,x)+1}{(\sum_{x \in V} count(c,x)) + |V|} \quad (5)$$

Laplace smoothing은 naive bayes의 존재하지 않는 word가 X로 사용될 시의 문제점을 해결하기 위하여 조건부 확률의 분자에 1, 분모에 V를 더하는 기법으로naive bayes의 label별 유사도 확률이 0으로 귀결되는 것을 방지해 준다. 더해주는 1과 V값은 naive bayes의 alpha, 하이퍼 parameter(hyper-parameter)라고 불리는데 머신 러닝 문제 해결 로직에서 적절한 하이퍼 parameter의 선택은 예측 값 평가에 큰 영향을 미치게 할 수도 있는 요소로서 본 논문에서는 4장 실험 결과에서 그에 따른 결과를 보인다.

III. 제안 모델

Fig.3.는 본 논문에서 제안하는 모델의 동작절차를 나타내는 것으로 본 논문에서 제안하는 안드로이드 악성 어플리케이션 탐지를 위하여, label별 샘플의 .dex 파일에 대해 정적 정보를 추출하고 raw data를 pre-treatment 과정을 거쳐 API List를 만들고 data set을 구성한다. 다음으로 머신 러닝 프로세스를 수행하는데 over-fitting을 방지하기 위하여 data set을 training set과 validation set으로 나누는 data split 과정을 거치게 된다. split된 data set 중 training set은 예측 모델을 구성하고 validation set은 예측 모델과 머신 러닝 알고리즘을 통하여 label별 확률을 구하고 및 예측 결과를 산출하는데 사용된다.

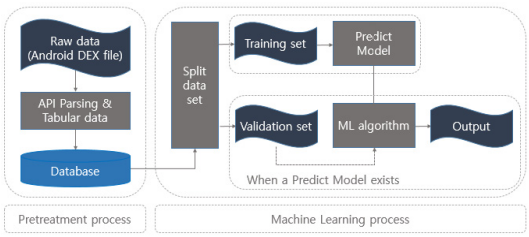

Fig. 3. Proposed Model

3.1 Pre-treatment

머신 러닝을 활용 하려면 관련 raw 데이터를 확보하여야 하고 그것에 대해서 data munging, pre-treatment 등의 데이터 가공 과정이 필요하다. 가공한 데이터는 머신러닝 알고리즘에 적용하기 위해 다시 data conversion을 통해서 tabular data 형태로 구성하여야 하는데 일련의 과정들은 pre-treatment 과정으로 볼 수 있고 Fig.4.와 같이 나타낼 수 있다.

샘플에서 raw data를 확보하기 위해서 안드로이드 .dex 파일의 구조를 파악하고 그것을 기반으로 데이터를 추출한다. .dex 파일은 Fig.5.의 string_IDs 영역에서는 문자열들에 대한 위치 값이 4바이트씩 저장되어 있는데 API는 문자열 형태로 정의되어 있으므로 해당 영역의 정보를 이용하여 접근 가능하다.

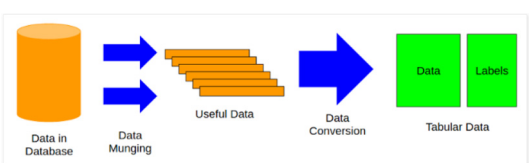

Fig. 4. Pre-treatment Process

string_IDs의 데이터는 다음 Fig.5.와 같다. 반면에 .dex 파일에서는 API 문자열만 별도로 관리하는 영역은 없지만 API는 method의 형태를 띄고 있으므로 method_IDs 영역에서 추출 가능하다. method_IDs 영역은 모든 함수, 즉 각각의 method에 대한 정보를 8바이트씩 할당하여 가지고 있는데 해당 위치의 string_IDs 영역을 참조하게 되면 method를 획득할 수 있다. method_IDs 영역의 크기만큼 method들을 추출하게 되면 해당 .dex파일 내의 모든 method를 추출 가능하다. Method 문자열은 API와는 달리 기본적으로 정의되어 있지 않고 어플리케이션 마다 그 형태가 특징이 크게 다르기 때문에 feature로는 부적합하다. 따라서 전체 method 문자열에 섞여있는 어플리케이션의 기능적 특징을 나타내는 API만 추출하게 되는데 본 논문에서는 안드로이드 레퍼런스의 API 리스트를 기준으로 하여 parsing한다. 하지만 모든 API를 feature vector로 사용하는 것은 연산량을 급격하게 가중시키는데 비해 성능 향상에 큰 의미가 없고 overflow등의 문제를 야기시킬 수 있으므로 본 논문에서는 우선 training set에 대해서 label 별로 자주 사용하는 API 300개를 안드로이드 레퍼런스의 API 리스트를 이용하여 parsing하고 추출한 후 그것을 기준으로 다시 training set의 API parsing에 사용해서 이 문제를 해결하였다. 또, 어플리케이션 마다 중복되어 사용하는 API 역시 마찬가지로 모두 vector로 사용할 시 샘플로 사용되는 어플리케이션 크기에 따라 예측 모델에 부정적인 영향을 미칠 수 있으므로 어플리케이션 별 각 API 사용 여부에 따라서만 vector를 구성한다. Table 2. 은 label별 API에 대한 빈도의 예시이다.

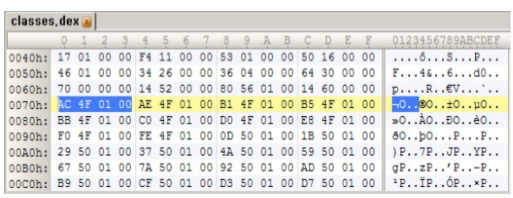

Fig. 5. Example of String_IDs Extraction

Table 2. API Frequency

API	Description	Frequency (Normal /Malious)
getCellLocation	get the Cell location of the device	16 / 117
getScanResults	get wifi AP infomation	22 / 103
setParameter	setting parameter	94 / 157
sendEvent	event sending	109 / 65
onSuccess	success thread	137 / 63
addListener	adding Listener	177 / 52

Data munging을 통해서 구성한 feature vector는 다음으로 머신 러닝 알고리즘에 적용하기 위하여 tabular data의 형태로 표현해야 한다. tabular data는 컴퓨터가 이해할 수 있는 배열 형태의 데이터를 의미하는 것으로 보통 각 행은 각각의 데이터 샘플의 vector에 해당하고 각 열, 혹은 여러 개의 열은 label을 의미하게 된다. 입력 데이터를 X, 데이터 샘플에 대한 label을 y로 표현하는데 본 논문에서는 행의 개수를 label의 개수인 2로 나누어 상위 영역과 하위 영역으로 label을 구분하고 각 열을 각각의 API 사용 여부로 표현하여 naive bayes의 입력 값에 적합한 tabular data로 구성하였다. Fig.6.은 naive bayes의 계산에 사용 되는 feature vector들에 대한 tabular data, X 그리고 y에 대한 예시로 각각의 데이터 샘플의

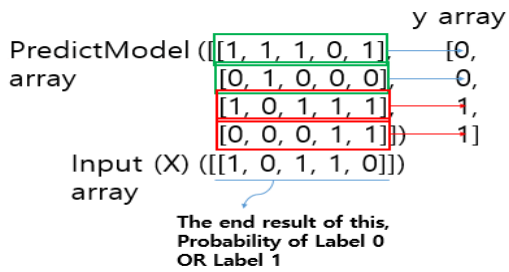

Fig. 6. Example of Conversion to Tabular Data Format

feature vector와 X의 경우 전체 feature vector 기준으로 사용되면 1, 사용되지 않으면 0으로 나타낸다. y의 경우 정상 어플리케이션 label을 0으로 악성 어플리케이션 label을 1로 나타낸다.

3.2 Split data set

지도 학습으로 구성한 메커니즘의 성능을 평가할 때 data set 모두를 training set으로 구성하여 훈련하고 그것을 다시 모델 평가에 사용하게 되면 모델의 Over-fiiting 정도를 측정하기 어려워질 수 있다. Fig. 7.과 같이 over-fitting은 모델의 견고함과 정확도를 저하시키는 요인이 될 수 있는데 over-fitting이란 기존의 training set에 대한 메커니즘의 성능이 기존에 없던 새로운 data set에

Table 3. Machine learning algorithm hyper-parameter

Model	Parameter to optimize	Good range of values
Linear Regression	▪ fit_intercept ▪ normalize	▪ True / False ▪ True / False
Ridge	▪ alpha ▪ fit_Intercept ▪ normalize	▪ 0.01, 0.1, 1.0, 10, 100 ▪ True / False ▪ True / False
k-neighbors	▪ N_neighbors ▪ p	▪ 2, 4, 8, 16 … ▪ 2, 3
SVM	▪ C ▪ Gamma ▪ class_weight	▪ 0.001, 0.01 … 1000 ▪ 'Auto', RS* ▪ 'Balanced', None
Logistic Regression	▪ Penalty ▪ C	▪ L1 or L2 ▪ 0.001, 0.01 … 100
Naive Bayes	▪ alpha	▪ 0.0001, 0.001 … 1.0
Lasso	▪ alpha ▪ Normalize	▪ 0.1, 1.0, 10 ▪ True / False
Random Forest	▪ N_estimators ▪ Max_depth ▪ Min_samples_split ▪ Min_samples_leaf ▪ Max features	▪ 120, 300, 500, 800, 1200 ▪ 5, 8, 15, 25, 30, None ▪ 1, 2, 5, 10, 15, 100 ▪ 1, 2, 5, 10 ▪ Log2, sqrt, None

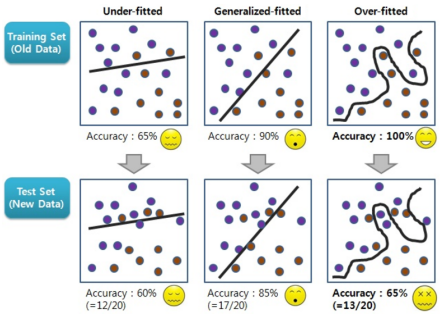

Fig. 7. Model Normalization

대한 성능이 training set으로 검증한 성능보다 현저하게 낮게 나타나는 현상으로 이러한 data set에 대한 일반화는 메커니즘 성능에 큰 영향을 미친다.

3.3 Hyperparameter select

머신 러닝 표준 모델에서는 기존 모델에서 존재하는 parameter와 구별하기 위하여 하이퍼 parameter라는 값을 사용한다. 하이퍼 parameter는 머신 러닝의 모델을 구성할 때 필요한 parameter와 bayes 정리와 같은 확률 공식 등과는 다르게 모델 복잡도나 속도와 같은 상위 수준을 표현하는 것으로 보통 하이퍼 parameter는 모델마다 다른 값으로 설정하여 어떤 모델이 가장 성능이 좋은지에 대해 결정하는 과정으로 선택되게 된다. 따라서 머신 러닝 메커니즘의 성능 평가는 머신 러닝 알고리즘의 하이퍼 parameter의 선택과도 직접적으로 연관되어 있다. 각 모델 별 하이퍼 parameter는 Table 3.과 같다.

naive bayes는 alpha라고 불리는 하이퍼 parameter를 사용하는데 이 값이 너무 작아지면 확률 값이 컴퓨터가 인식할 수 없을 정도로 작아서 오류가 나게되고 너무 크면 overflow로 귀결되기 때문에 일반적으로 적절한 값은 0.0001에서 1.0의 사이로 여겨진다. 4장 실험 결과에서는 본 메커니즘으로 도출한 샘플 별 탐지율/오탐율 결과를 기술하고 탐지율을 높이기 위하여 data split 프로세스로 over-fitting을 피하고 적합한 training set을 찾는 과정과 해당 모델에서 하이퍼 parameter 값의 변경으로 그 모델에 미치는 영향과 성능 향상에 대해서 기술한다.

IV. 실험 결과

본 논문에서 제안한 모델은 2.30GHz 듀얼 코어, 8Gb Ram과 Windows 10의 환경에서 구현하였다. 4.1절 에서는 샘플 개수 별 본 메커니즘의 성능 차이를 보이고 탐지율과 over-fitting의 정도를 고려하여 적합한 샘플 개수를 선택한다. 4.2절 에서는 선택한 샘플 개수로 모델을 구성하고 하이퍼 parameter 변화에 의한 모델 성능 변화를 보인다. 실험에 사용한 악성 샘플은 API 존재 유뮤를 고려하여 정상 샘플들과 유사한 크기를 보이는 샘플을 사용하였고 정상 샘플은 Virus total로 검사하여 탐지되지 않은 APKpure의 샘플을 사용하였다. 특히 정상 샘플의 경우 각 어플리케이션의 용도별 API 사용 빈도를 고려하여 용도별로 동일한 비율을 구성하였다.

4.1 샘플 개수별 모델 성능 검증

악성코드와 정상파일의 data-set을 구성한 후, 각각에 대해서 training에 사용할 데이터와 validation에 사용할 데이터로 구분하였다. Table 4.와 Table 5.은 over-fitting 정도를 측정하기 위해서, training에 사용할 data-set의 크기를 변화시켜 가면서 이에 따른 정상파일과 악성코드에 대한 accuracy를 측정하였다.

먼저 Table 4.에서와 같이 training set으로 샘플 200개를 사용하였을 때 Normal label의 training data에 대해서 정확도 97%, training set으로 샘플 240개를 사용하였을 때 Malicious label의 training data에 대해서 정확도 85%로 테스트 결과에서 가장 높은 정확도를 보였다. 또, Normal label의 경우에는 training set의 크기가 커질수록 정확도가 점점 낮아지는 특징을 보였고 Malicious label의 경우에는 240개 이후로 정확도

Table 4. Accuracy Analysis of Training Data

training set	Normal data accuracy	Malicious data accuracy
200	0.9700	0.8300
240	0.9583	0.8500
280	0.9642	0.7857
320	0.9625	0.7750
360	0.9555	0.7666
400	0.9450	0.7550

가 점점 낮아졌다. 테스트 결과에 의하면 해당 training set 범주 내에 대한 적합한 훈련 수치는 200개~240개라고 판단할 수 있다.

반면에 Table 5.는 training set에 사용하지 않은 샘플 200개로 구성된 validation set으로 training set 개수 별로 테스트 한 결과를 나타낸 것으로 training set으로 샘플 200개를 사용하였을 때 Normal label의 training data에 대해서 정확도 94%, training set으로 샘플 240개를 사용하였을 때 Malicious label의 training data에 대해서 정확도 74%로 테스트 결과에서 가장 높은 정확도를 보였다. 또, Normal label의 경우에 240개 보다 training set의 크기가 커지더라도 정확도에 변화가 없는 점을 알 수 있다. Malicious label의 경우에는 240개 이후로 정확도가 점점 낮아졌지만 400개에서 정확도가 미약하게 상승했다. training set에 대한 테스트 결과와 마찬가지로 해당 validation set 범주내의 적합한 훈련 수치는 200개~240개라고 판단할 수 있다. over-fitting의 정도는 training, validation set의 정확도 차이를 비교하면 측정 가능하다. 일반적으로 training의 정확도가 validation의 정확도보다 높은데 Table 6.에 그 결과가 나타난다.

Table 6.은 Normal label에 대한 training set의 정확도와 validation set의 정확도를 나타낸다. 모두 90%가 넘는 정확도와 training set과 validation set의 차이가 평균 5% 내외라는 점을 감안하면 모델이 적합하게 잘 훈련되었다고 평가할 수 있다.

Table 7.은 Malicious label에 대한 training set의 정확도와 validation set의 정확도를 나타낸다. training set에 대한 정확도는 75%~85%에서 분포하고 있고 validation set에 대한 정확도는 69%~75%에서 분포하고 있다. 특히 training set 240개로 모델을 구성하였을 경우 validation set과의 차이가 11%로 수긍할 만한 차이를 보였다.

Fig. 8.와 같이 테스트 결과 training set과 validation set 검증에서 가장 높은 성능을 보인 240개의 샘플로 모델을 구성한다. 다음으로는 이렇게 구성된 모델의 하이퍼 parameter 선택을 통한 성능 향상의 정도를 보인다.

Table 5. Accuracy Analysis of Validation Data

training set	Normal data accuracy	Malicious data accuracy
200	0.9375	0.7200
240	0.9062	0.7400
280	0.9062	0.7100
320	0.9062	0.6900
360	0.9062	0.6900
400	0.9062	0.7000

Table 6. Over-fitting Analysis of Normal Files

training set	Training data accuracy	Validation data accuracy
200	0.9700	0.9375
240	0.9583	0.9062
280	0.9642	0.9062
320	0.9625	0.9062
360	0.9555	0.9062
400	0.9450	0.9062

Table 7. Over-fitting Analysis of Malware

training set	Training data accuracy	Validation data accuracy
200	0.8300	0.7200
240	0.8500	0.7400
280	0.7857	0.7100
320	0.7750	0.6900
360	0.7666	0.6900
400	0.7550	0.7000

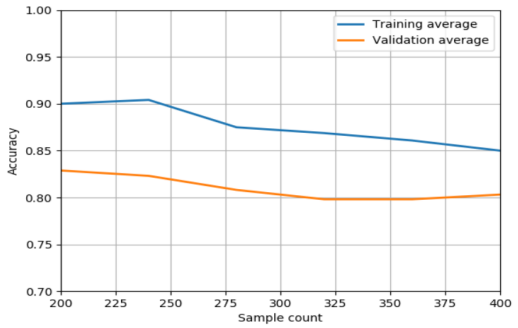

Fig. 8. Split Data Set Result

4.2 하이퍼 parameter별 모델 성능 검증

4.1절에서 구성한 모델에서 하이퍼 parameter별 training set에 대한 성능 변화는 Table 8.와 같다. Laplace smoothing을 적용한 naive bayes의 기본 하이퍼 parameter인 1.0에 비해 다

른 하이퍼 parameter들은 모두 정확도가 감소했다. 하이퍼 parameter를 변경한다고 항상 성능이 향상되거나 바뀌는 것은 아니기 때문에 training set에 대한 하이퍼 parameter별 모델 성능 검증에는 1.0이 가장 적합하다고 판단할 수 있다. 반면에 Validation set에 대한 성능 변화는 Table 9.과 같다.

Validation set에 대한 alpha 값에 대한 성능 변화는 나타나지 않는데 연산량을 고려하면 최적의 하이퍼 parameter는 1.0으로 판단할 수 있다. Fig.9.은 alpha 값에 따른 training set과 validation set의 정확도를 나타낸다.

Table 8. Hyper-parameter Changes for Training Set

Alpha	Normal accuracy	Malicious accuracy
1.0	0.9700	0.8500
0.1	0.9667	0.8250
0.01	0.9667	0.7916
0.001	0.9667	0.7916
0.0001	0.9667	0.7916

Table 9. Hyper-parameter Changes for Validation Set

Alpha	Normal accuracy	Malicious accuracy
1.0	0.9700	0.7400
0.1	0.9667	0.7400
0.01	0.9667	0.7400
0.001	0.9667	0.7400
0.0001	0.9667	0.7400

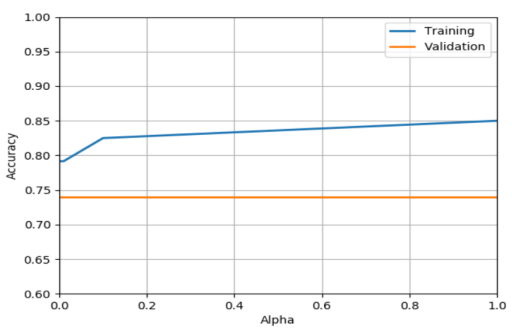

Fig. 9. Hyper-parameter Test Result

V. 결 론

본 논문에서는 매년 꾸준히 증가하고 있는 모바일 악성코드, 특히 국내에서 다수가 사용하는 안드로이드 운영체제를 목표로 하는 악성코드들에 대응하기 위해서 naive bayes를 기반으로 한 악성코드 탐지 메커니즘을 제안하였다. 악성코드가 유포되고 실행되어 피해를 유발하는 수단 중 대다수는 해당 코드를 실행하는 어플리케이션의 다운로드 및 실행으로 발생하게 되는데 이러한 악성코드가 내재한 악성 어플리케이션들의 주요 목적인 사용자 정보 유출, 데이터 파괴, 금전 갈취 등은 운영 체제나 프로그래밍 언어가 제공하는 기능을 제어할 수 있게 해주는 인터페이스인 API에 의하여 동작한다는 점을 착안하여 안드로이드 .dex 파일 내의 정적 API 정보를 지도 학습을 기반으로 한 naive bayes의 feature로 선정하였다. 제안 모델에서는 어플리케이션의 label별 API 빈도가 통계학적으로 차이를 나타냄을 보였고 이를 naive bayes 알고리즘에 적용할 수 있게 전처리 하는 과정을 보였다. 또, naive bayes의 성능 개선을 위하여 data split을 통해 적합한 training set을 찾는 과정을 보이고 학습 모델에 존재하지 않는 API가 입력으로 들어올 시 결과값이 0으로 귀결되는 문제를 해결하기 위해 laplace smoothing을 적용하고 그 하이퍼 parameter인 alpha 값에 대해 기술하고 실험 결과에서는 그 차이와 기존에 training set에 존재하지 않던 신규 데이터에 대해 탐지 가능함을 나타냄으로서 신종 악성코드에 대한 대응이 일정 수준 가능함을 보였다. 따라서 naive bayes 알고리즘이 악성코드 탐지 도메인 적합성과 탐지율 개선을 위한 방법을 보였지만 논문에서 제시한 training set이 다양하지 못했고 향후 더 많은 training set에 대해 메커니즘을 분석하고 결과를 나타내면 더 정확한 탐지율을 도출할 수 있다고 판단한다. 또, 본 논문에서 제안하는 메커니즘에 API 외의 파일 정적, 동적 정보와 같은 다른 feature들과의 교차 검증을 통하여 성능 향상이 가능하며 기존의 anti-malware 체계에 기존의 탐지 기법들과 병행하여 본 메커니즘을 적용 및 운용하면 신종 악성코드 탐지에 많은 기여할 수 있을 것이라 기대한다.

References

[1] Myeong-jae Seong, Hae-ryong Park, Bo-min Choi and Eul-gyu Im "Android Malware Detection Using TCP-flow se-

quence on Network Behavior-based," Journal of Security Engineering, pp.551-566, 2014.
〔2〕 Ki-hyeon Kim, "A hybrid Method for Android Malware Detection," MS. Thesis, Kangwon National University, Feb 2016.
〔3〕 Seung-wook Min, Hyung-jin Cho, Jin-seop Shin and Jae-cheol Ryou, "Android Malware Analysis and Detection Method Using Machine Learning," Journal of KIISE : Computing Pracices and Letters, Feb 2013.
〔4〕 Cheol Jeon, Joon-hyouk Jang, Bong-jae Kim, Jin-man Jung and Yoo-kun Cho, "A Robust Permission-Based Malicious Application Filtering Scheme for Effective Android Application Reviews," Journal of Security Engineering, April 2013.
〔5〕 Zami Aung and Win Zaw, "Permission-Based Android Malware Detection," INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH vol. 2, no. 3, March 2013.
〔6〕 W. Enck, P. McDaniel, D. Octeau and S. Chaudhuri, "A Study of Android Application Security," in USENIX Security Symposium, USA, vol. 2, no. 2, pp. 2-17, 2011.
〔7〕 Hyun-ki Kim, "Quantitative Risk Assessment Scheme of Mobile Malwares Based on Machine Learned API Characteristics," MS. Thesis, Soongsil University, Feb 2017.
〔8〕 Hyo-sik Ham, Hwan-hee Kim, Myung-sub Kim and Mi-jung Choi, "Android Malware Detection Technique Based on SVM," Journal of Applied Mathematics, Sep 2014.
〔9〕 Ki-hyun Kim and Mi-jung Choi, "Android-based Malware Detection Using Linear One-Class SVM," Korea Institue Of Communication Sciences, pp. 397-398, 2014.
〔10〕 Seok-min Yun, You-jeong Ham, Geun-shik Han and Hyung-woo LEE, "Malicious Application Determination Using the System Call Event," Korea Information Processing Society, pp. 169-176, 2015.
〔11〕 Ji-won Lee, Dong-hoon Lee and In-suk Kim, "Method of Detecting SmiShing using SVM," Journal of Security Engineering, pp. 655-668, 2013.
〔12〕 C. Widanapathirana, Y. Ahmet Sekercioglu, Milosh V. Ivanovich, Paul G. Fitzpatrick and Jonathan C. Li, "Automated Inference System for End-To-End Diagnosis of Network Performance Issues in Client-Terminal Devices," International Journal of Computer Networks & Communications (IJCNC), vol. 4, no. 3, May 2012.
〔13〕 Seung-wook Min, Hyung-jun Jo, Jin-sub Shin and Jae-chul Ryu, "Android Malware Detection Method Using Macchine Learning," Journal of KIISE : Korea Computer Congress, pp. 280-282, June 2012.
〔14〕 B. Amos, H. Turner and J. White, "Applying machine learning classifiers to dynamic android malware detection at scale," In Proceedings of the 9th International Wireless Communications and Mobile Computing Conference (IWCMC), pp. 1666 - 1671, July 2013.
〔15〕 I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, " Crowdroid: behavior-based malware detection system for android," In Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices (SPSM), pp. 15 - 26, 2011.
〔16〕 J. Abah, O. Waziri,M. Abdullahi, U. Arthur and O. Adewale, "A machine learning approach to anomaly-based detection on Android platforms," International Journal of Network Security and Its

Applications, pp. 15 - 35, 2015.
〔17〕 V. M. Afonso, M. F. de Amorim, A. R. A. Gregio, G. B. Junquera and P. L. de Geus, "Identifying Android malware using dynamically obtained features, " Journal of Computer Virology and Hacking Techniques, pp. 9 - 17, 2015.
〔18〕 Y. Zhou and X. Jiang, "Dissecting Android Malware: Characterization and Evolution," Proceedings of the 33rd IEEE Symposium on Security and Privacy, pp. 95 - 109, 2012.
〔19〕 Won-hyuk Choi, "Analyzing Class.dex Executable Code for Android Apps," ITL Inc, July 2014.
〔20〕 S. Kaveh, D. Ali and M. Ramlan, "Trends in Android Malware Detection, " Journal of Digital Forensics : Security and Law, vol. 8, no. 3, 2013.
〔21〕 Y. Nirmala, S. Aditi and D. Amit, "A Survey on Android Malware Detection," International Journal of New Technology and Research, vol. 2, no. 12, pp 47-53, December 2016.
〔22〕 S. Justin and K. Latifur, "A Machine Learning Approach to Android Malware Detection," European Intelligence and Security Informatics Conference, 2012.
〔23〕 K. Pallavi and J. Amit, "Malware Detection Techniques in Android," International Journal of Computer Applications, vol. 122, no. 17, July 2015.
〔24〕 Sung-tae Yu and Soo-hyun Oh, "Malware Analysis Mechanism using the Word Cloud based on API Statistics," Journal of the Korea Academia-Industrial Cooperation Society, vol. 16, no. 10, pp. 7211-7218, 2015.
〔25〕 K. Tetsuya, M. Yuki, I. Takashi, M.Makoto and I. Katsuro, "A Prototype of Comparison Tool for Android Applications Based on Difference of API Calling Sequences," Software Science 111 (107), IEICE, pp. 35-40, 2011.

〈저자소개〉

황 준 호 (Jun-ho Hwang) 학생회원
2012년 3월~현재: 호서대학교 정보보호학과
〈관심분야〉 머신러닝, 정보보호, 통계학

이 태 진 (Tae-jin Lee) 종신회원
2003년 1월~2017년 2월: 한국인터넷진흥원 팀장
2017년 3월~현재: 호서대학교 정보보호학과
〈관심분야〉 시스템 보안, 악성코드 분석, 침해사고 대응