Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.29, NO.6, Dec. 2019 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2019.29.6.1235

안드로이드 악성코드 분류를 위한 Flow Analysis 기반의 API 그룹화 및 빈도 분석 기법*

심 현 석,† 박 정 수, 단티엔북, 정 수 환‡
숭실대학교

API Grouping Based Flow Analysis and Frequency Analysis Technique for Android Malware Classification*

Hyunseok Shim,† Jungsoo Park, Thien-Phuc Doan, Souhwan Jung‡
Soongsil University

요 약

본 논문에서는 머신러닝 기반의 악성코드 분류에 있어 오버피팅 문제를 비롯하여 실제로 실행되지 않는 코드가 APK에 포함되는 문제 등을 해결하기 위해 모든 API들의 연관성을 통해 그룹화하며, 제어 흐름 분석을 통해 실제로 실행되는 코드에 대한 분석을 수행하는 툴을 개발하였다. 툴은 약 1,500라인으로 이루어진 자바 기반의 소프트웨어로, 전체 API에 대한 빈도 분석을 수행하거나 생성된 제어 흐름 그래프를 바탕으로 빈도 분석을 수행한다. 툴을 이용하여 모든 버전에서의 총 39032개의 메서드에 대해 4972개의 그룹으로 축소할 수 있으며, 클래스를 포함한 결과로는 총 12123개의 그룹으로 축소할 수 있다. 결과 분석을 위해서 본 논문에서는 총 7개의 패밀리에서 7,000개의 APK를 랜덤으로 수집하였으며, 수집된 APK를 이용하여 feature를 축소하는 기법을 검증하였다. 또한, 추출된 데이터에서 빈도가 20% 이상으로 나타난 API만을 선별하여 feature를 더욱 축소하여 최종적으로 263개의 feature로 축소하였다.

ABSTRACT

While several machine learning technique has been implemented for Android malware categorization, there is still difficulty in analyzing due to overfitting problem and including of un-executable code, etc. In this paper, we introduce our implemented tool to address these problems. Tool is consists of approximately 1,500 lines of Java code, and perform Flow analysis on set of APIs, or on control flow graph. Our tool groups all the API by its relationship and only perform analysis on actually executing code. Using our tool, we grouped 39032 APIs into 4972 groups, and 12123 groups with result of including class names. We collected 7,000 APKs from 7 families and evaluated our feature reduction technique, and we also reduced features again with selecting APIs that have frequency more than 20%. We finally reduced features to 263-numbers of feature for our collected APKs.

Keywords: Android malware, Malware classification, Feature selection, API grouping, Flow analysis

Received(09. 18. 2019), Modified(11. 28. 2019), Accepted(11. 28. 2019)

* 본 논문은 2019년도 한국정보보호학회 하계학술대회에 발표한 우수논문을 개선 및 확장한 것임

* 이 논문은 2019년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No.2019-0-00477, 가상화된 신뢰실행환경을 이용한 안드로이드 보안 프레임워크 기술 개발)

† 주저자, ant_tree@naver.com

‡ 교신저자, souhwanj@ssu.ac.kr(Corresponding author)

I. 서 론

MTC에서 예측한 통계에 따르면 스마트폰의 보급은 2020년까지 28.7억대가 보급될 것으로 예상되며, 이러한 모바일 기기 중 안드로이드 운영체제의 비율은 Gartner에서 분석한 결과 85.9%를 넘어섰다[1]. 또한 안드로이드 악성코드의 경우 그 코드의 복잡도가 크며, 리소스의 양이 많으므로 악성코드 카테고리를 분류하는 데에 어려움이 있다.

따라서 악성코드 분류 및 탐지를 위해 여러 기법이 제안되었는데, 그 분석 기법으로는 크게 정적 분석 기법과 동적 분석 기법으로 나눌 수 있다. 이 중 정적 분석의 경우 앱을 실행시키지 않는 환경에서 Op-Code 혹은 Manifest, API Call을 기반으로 확인하는 것이다. 실제로 정적 분석에서는 패턴을 기반으로 머신러닝, 혹은 딥러닝을 이용한 방법이 많이 활용되고 있다. 하지만 딥러닝, 혹은 머신러닝을 이용한 분류에서 가장 문제가 되는 것은 올바른 feature의 선정으로, feature가 적을 때와 많을 때 모두 문제가 발생한다[2]. feature가 너무 많은 경우에 트레이닝 셋에 대해 학습을 오래 시킨다면 오버피팅 문제가 발생하며[3], feature가 너무 적다면 얼마나 오래 학습을 시켰는지에 관계없이 언더피팅이 발생하는 경향이 있다[4]. 오버피팅이 일어나는 경우에는 트레이닝 데이터 외의 새로운 데이터가 입력되는 경우에 새로운 데이터를 올바르게 분류하지 못하며, 언더피팅이 일어나는 경우에는 테스트 데이터 뿐 아니라 트레이닝 과정에서도 모델을 올바르게 학습시킬 수 없다. 이러한 문제를 해결하기 위해서 이전 연구[5][6][7]에서 feature reduction 혹은 normalization을 통해 분류하는 방법을 제안하였으나, 이러한 방법들은 일부 feature를 제거하는 방법이거나, 안드로이드 분야에서 적합하지 않은 방법이다.

따라서 본 논문에서는 정적 분석 환경에서의 올바른 feature 선정을 통한 API 호출 관계 분석을 수행하는 것을 목표로, 그 과정에서 API 호출 관계의 그룹화와 제어 흐름 기법을 이용하기로 한다.

서론에 이어 본 논문은 2장에서 관련연구로 기존의 안드로이드 악성코드 분석기법에 대해 소개하며, 3장에서는 제어 흐름 분석 기법과 API 그룹화를 통한 빈도 분석 기법에 대해 설명한다. 4장에서는 개발된 툴을 이용한 추가 축소 기술과 분석에 대해 설명한다. 마지막으로 6장에서는 본 논문의 결론과 향후 연구 및 방향에 대해 기술한다.

II. 관련 연구

2.1 악성코드 분석 기법

안드로이드 악성코드 분석 기법은 Fig. 1과 같이 크게 정적 분석 기법과 동적 분석 기법, 그리고 하이브리드 분석으로 나누어지며, 각 기법에서도 세부적인 기법으로 나누어진다[8].

본 논문에서는 정적 분석 통해 분석을 수행하며, 하이브리드 분석은 정적 분석을 통해 얻은 결과를 동적 분석에 사용하는 것으로, 관련 연구에서는 정적 분석과 동적 분석만을 소개하도록 한다.

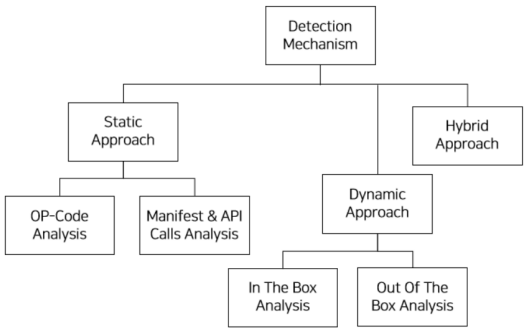

Fig. 1. Android malware detection mechanisms

2.1.1 정적 분석 기법

정적 분석 기법은 크게 Op-Code 기반의 분석과 Manifest & API Call 기반의 분석으로 나눌 수 있다[9]. Op-Code 기반의 분석은 악성코드와 정상 파일의 Dex 파일을 N-Gram Op-Code 단위로 디컴파일하여 그 패턴을 분석하는 것으로, 일반적으로 머신러닝을 통해 패턴 분석이 이루어진다. Manifest & API Call 기반의 분석은 Manifest를 통해 제공되는 권한, 인텐트, 컴포넌트와 같은 요소들과 API 호출 정보를 이용하여 패턴 인식을 진행하는 분석 기법이다.

2.1.2 동적 분석 기법

동적 분석 기법은 크게 In The Box 분석과 Out Of The Box 분석으로 나눌 수 있다. In The Box

분석의 경우 악성코드와 분석 시스템이 같은 privileged level에서 수행되며, Out Of The Box 분석의 경우 시스템의 안전을 확보한 뒤 수행된다. 동적 분석은 안드로이드 실행 환경, 즉 Android Run Time(ART)에서 분석하기 위한 것으로, 타겟 어플리케이션이 실행 중일 때 분석한다. 안드로이드 애플리케이션은 API에 대한 호출과 브로드캐스트 이벤트와 인텐트에 대한 응답(response)을 로깅하여 동적으로 분석할 수 있다.

2.2 안드로이드 구조

안드로이드 어플리케이션은 classes.dex, res, lib, assets, META-INF, resources.arsc, 그리고 AndroidManifest.xml로 구성된 Android package, 즉 APK의 형태를 가지고 있다[10]. 안드로이드의 정적 분석 과정에서는 APK를 디컴파일하여 내부의 리소스 및 API들을 추출하며, 주로 classes.dex와 AndroidManifest.xml의 정보를 활용한다. 각 구성성분들의 역할은 Table 1과 같다[11].

Table 1. Name and role of each component of Android

Name	Description
classes.dex	Files converted class files into byte codes for recognition within the Android Dalvik virtual machine.
res	A directory aggregates non-compile images and xml resources.
lib	Directory contains library files, which are compiled with NDK.
Meta-INF	A directory related to signature. It contains MANIFEST.MF, CERT.SF, and store signature encrypted with SHA1 and base64.
resources.arsc	File that record information about resource files. Store types and ids of resource files located in *res* directory.
assets	A directory aggregates application's information that can be managed by *AssetsManager*.
AndroidManifest.xml	An xml file for managing applications, specifying the application's permissions, component information such as content, services, and activity, and information about the SDK version.

III. 구 현

본 논문의 전체적인 구조는 Fig. 2와 같다. 해당 구조에서 전처리 과정을 위해 모든 버전에서의 Android API 정보를 필요로 한다. 각 버전에서의 API는 frequency analysis를 통해서 여러 개의 그룹으로 매칭되며, 각 그룹은 feature에 해당한다. 우리의 목표는 이러한 그룹화를 통해 feature를 축소시키는 것이기 때문에, 각 APK의 control flow analysis 결과를 각 그룹에 매칭시켜 분석 대상(APK)의 feature를 축소시킨다. 마지막으로 그룹화된 API 호출 정보를 통해 빈도 분석을 수행하며, 각각 카테고리별로 분석하여 출현 빈도가 20% 이상으로 나타난 그룹들만을 분석한다.

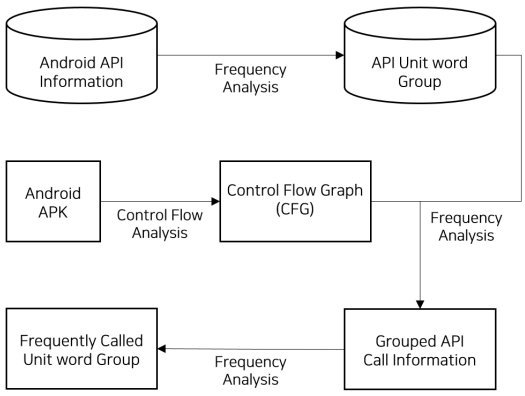

Fig. 2. Overall structure of implemented tool

3.1 제어 흐름 분석

본 논문에서는 정적 분석을 위해 각 APK에서의 시스템 API 호출과, 그 시스템 API의 호출 여부를 활용하고자 한다. 호출관계의 추출을 위해 자바(Java) 바이트 코드 기반의 제어 흐름 분석 툴인 *FlowDroid*[12]를 활용하였으며, 각 APK에 대해서 메서드 간 제어 흐름 그래프를 얻을 수 있었다. 제어 흐름 그래프에서 실제 실행 여부를 확인하기 위해 메서드의 callee만을 추출하였으며, caller의 경우 시스템 API가 사용되는 것은 Throwable 클래스의

```
"<org.bonehead.aHpJGLJtienHDCXlWiezjUtoX: void animate()>": [
    "<android.view.animation.AnimationSet: void <init>(boolean)>",
    "<android.view.animation.TranslateAnimation: void <init>(int, float, int, float, int, float, int, float)>",
        ...
    "<android.view.animation.Animation void setFillAfter(boolean)>:
],
"<org.bonehead.ZeDRoPSXbMtZEwwnrUsHIlrm: android.util.pair lkTRHsXsBiopFlAgKjurNmMB(java.lang.String)>": [
    "<java.util.regex.Pattern: java.util.regex.Pattern compile(java.lang.String)>",
    "<java.io.RandomAccessFile: void close()>",
        ...
    "<java.io.RandomAccessFile: int read(byte[])>"
]
```

Fig. 3. FlowDroid output as json format. Key as caller and value as callees.

〈init〉() 혹은 Exception 클래스의 〈init〉()과 같은 부분으로, APK 실행 경로에서 큰 영향을 미치지 않는 부분이므로 무시할 수 있다.

추출된 APK의 제어 흐름 그래프는 Json format을 따르며, caller와 callee에 대해 key, value의 형태로 나타내었다. Fig. 3는 제어 흐름 그래프를 출력한 결과로, 분석에 사용되는 데이터는 이 중 callee에 해당되는 value에 해당되는 jsonElement 들이다. 각 element 중 시스템 API에 해당되는 element들은 3.2의 API의 그룹화를 통해 매칭되는 unit word와 그룹핑한다.

3.2 API 그룹화

3.1에서 추출한 제어 흐름 그래프에서 호출된 시스템 API의 종류와 그 횟수를 알 수 있지만, API 버전 1부터 28까지의 모든 시스템 API의 개수는 메서드의 이름 중복을 제외하고 총 39032개로, 모든 API에 대해 학습할 경우 오버피팅 문제를 초래할 수 있다. 따라서 API간의 연관성을 통해 그룹화를 진행하여, 그 결과를 통해 분류를 진행한다.

각 메서드 간 연관성 파악을 위해 모든 메서드 이름을 유닛 단어 단위로 분리하는데, 자바에서는 Camel Case를 이용하여 단어를 구분하기 때문에 각 단어의 대문자 시작 지점에서 분리하였다. 각 메서드에 대해 단어를 분리할 때 주의해야 할 것은, 일부 단어는 대문자만으로 이루어진 경우가 있어, 연속된 대문자로 이루어진 단어에 대해서는 각 대문자로 분리하지 않아야 한다는 것이다. 예를 들어 opengl 패키지의 glFramebufferTextureEXT()과 같은 메서드의 경우에는 EXT를 하나의 단어로 고려해야 한다. 따라서 정규식을 통해 메서드의 이름을 분리하도록 하며, 그 코드는 Fig. 4와 같다

```
public String[] onSplitMethodNameByUnit(String methodName){
    String[] units;
        ...
    return methodName.split("(?<!(^|A-Z]))(?=[A-Z])|(?<!^)(?=[A-Z][a-z])");
}
```

Fig. 4. Code to split method names by units

위의 과정을 통해 우리는 유닛 단어 단위로 나눠진 메서드 이름을 얻을 수 있다.

다음으로는 유닛 단어를 통해 메서드의 연관성을 파악하여 그룹화 하는 과정으로, 이 과정에서 발생 빈도가 낮은 사건이 자주 발생하는 사건보다 정보량이 많다는 정보이론의 핵심 아이디어를 채용하였다. Shannon Entropy[13]는 확률 변수 $X: P \rightarrow E$가 분포 $f: E \rightarrow R$를 따를 때 다음 식과 같이 나타난다.

$$H(X) = -E(\ln f) = -\int_E f(x)\ln(f(x))dx \qquad (1)$$

식 (1)에서 로그함수의 정의에 따라 f(x)가 작아질수록 $H(X)$가 커지며, $H(X)$는 정보량으로 정의하므로, 결국 확률이 낮을수록 정보량이 크다. 따라서 메서드 이름의 유닛 단어 중 발생 빈도가 낮은 단어는 정보를 많이 가지고 있는 것이므로, 해당 단어로 메서드 이름을 그룹화 할 수 있다. 그 결과 각 메서드는 하나의 유닛 단어를 가지고 있는 그룹에 매핑(mapping)된다.

모든 시스템 API에 대해 그룹화를 수행하면, 중복되는 메서드 명을 제외하고 총 4972개의 그룹을 얻어낼 수 있다. Table 2는 메서드 이름을 통한 그룹화 결과를 나타낸 것이다.

Table 2. Unit word-methods relationship by API grouping

Unit Word	Method Name
Gain	getPostGain setPostGain getGain …
Below	assertOffScreenBelow
Cable	onVirtualCableUnplug
handshake	handshakeCompleted
Cursive	getCursiveFontFamily setCursiveFontFamily …

3.3 APK 분석

APK 분석 단계에서는 각 그룹에 매핑된 API의 메서드명과 APK에서 추출된 API를 매칭시켜, APK에서의 그룹 호출 횟수를 추출하는 것을 우선한다. 각 APK에서 추출된 제어 흐름 그래프는 마찬가지로 호출자-피호출자(caller-callee) 관계를 가지고 있으므로, 모든 피호출자에 대해 그룹 호출 빈도를 조사하여 벡터화 시킨다. 예를 들어 getUserData()가 호출되는 경우 해당 method의 유닛 단어인 'User'의 호출 빈도가 증가하며, 그룹-빈도 관계를 Table 3와 같이 출력할 수 있다. 그룹-빈도 관계는 그룹 유닛 단어의 알파벳 순으로 정렬되며, 호출되지 않은 그룹의 경우 0으로 채워진다.

IV. Feature 추가 축소 및 분석

우리가 본 논문에서 제안하는 기법은 드롭아웃과 같은 regularization을 미리 취하고 학습을 진행하는 것과 같은 이치로, 오버피팅에 영향을 받지 않는 것이 목표이다. 따라서 API를 그룹화 시킨 상태에서 빈도 분석을 수행하면 특정 빈도보다 높은 그룹들을 찾아낼 수 있는데, 이러한 그룹들만을 feature로 선별하여 학습을 진행하면 feature를 더욱 축소할 수 있다. 이 과정에서 필연적으로 오버피팅이 발생하게 되는데, 제안되는 기법은 그룹화 과정에서 API 관련 특징을 정규화 하기 때문에, 오버피팅이 발생하더라도 일반화 되는 특징을 갖는다. 따라서 feature의 추가 축소를 통해 중복되는 그룹을 제외하고 263개의 feature로 축소할 수 있다. 이는 모든 카테고리의 API 중 빈도가 20% 이상으로 나타난 그룹만을 선별한 것이다. 이러한 선별을 위해 빈도를 계산하는 Java 기반 코드를 추가로 작성하였으며, feature는 Ⅲ에서 관찰한 결과인 4972개에 비해 94.7% 감소한 수를 가지게 된다.

Table 3. Unit word-frequency relationship of single APK

Word	Called	Word	Called
Action	3	Create	1
Appeaer	0	Delivery	0
Backed	0	Positive	1
Base	0	Pssh	0
Capability	0	Recommend	0
Double	0	Reduced	0
Exclusion	0	SCRIPT	0
Fingerprint	0	Send	0

패밀리별 악성 어플리케이션의 수집은 Android Malware Dataset(AMD)[14]의 데이터셋을 활용하였으며, 각 패밀리 당 1,000개 이상의 APK를 보유하고 있는 패밀리만을 이용하였다. 전체 패밀리에 대한 빈도 분석 결과는 Table 4에 나타내었으며, 각 결과는 패밀리 당 1,000개의 APK를 분석한 결과이다. 결과에서 20% 이상의 빈도를 나타내는 그룹의 수를 확인해보면, Adware 카테고리의 패밀리들은 상대적으로 많은 그룹을 가지고 있는 것을 확인할 수 있다. 이는 Adware의 특성상 여러 그룹에 속하는 API를 사용하기 때문이며, 다른 category의 APK의 경우 특정 API의 사용이 지배적인 특성을 띄고 있다. 따라서 이러한 각 패밀리에 대해 특정 유닛 단어의 출현 빈도가 높게 나타나는 것을 확인하여 머신러닝의 feature로서 활용할 수 있으며, 이를 통해 패밀리별 분류가 가능하다.

반면 이러한 결과는 규칙 기반(rule-based) 머신러닝과 같은 양상을 띠게 된다. 따라서 모든 API를 사용하여 학습한 결과임에도 불구하고, 지나치게 적

Table 4. Unit word-frequency relationship of single APK

Family	Category	# of groups emerged with more than 20% frequency
Airpush	Adware	102
Dowgin	Adware	123
FakeInst	Trojan-SMS	28
Fusob	Ransom	18
Kuguo	Adware	169
Mecor	Trojan-Spy	33
Youmi	Adware	185

```
Input : APIs : Set of API names.
Output:  : rootWordMap : Map of root word and API name.
Variable: frequencyMap : Map of Unit word (of API name) and frequency.
Variable: units : array of unit words from single API name.
Variable: rootWord : a unit word of API that is frequently emerged.
procedure iterateOverAPIs(APIs)
    frequencyMap <- Ø;
    rootWordMap <- Ø;
    FOREACH API name of APIs DO
        units <- split API name with Capital or Capitals;
        calculateFrequency(units);
    FOREACH API name of APIs DO
        //re-split the API into unit words for rootWord matching
        units <- split API name with Capital or Capitals;
        rootWord <- getRootWord(units);
        put rootWordMap the rootWord and API name;
    return rootWordMap;
procedure calculateFrequency(units)
    FOREACH unit word of units DO
        IF frequencyMap has frequency of unit DO
            add frequency of unit and update frequencyMap;
        ELSE
            put unit into frequencyMap with frequency 1;
procedure getRootWord(units)
    SET rootWord null;
    FOREACH unit word of units DO
        SET rootWord into unit for initial state;
        get frequency of unit word from frequencyMap;
        IF frequency of unit word is bigger than frequency of currenty rootWord DO
            rootWord <- unit word;
    return rootWord;
```

Fig. 5. Overall process of frequency analysis as pseudo code.

은 수의 feature를 사용하였으므로 적은 수의 데이터 셋에는 유리하지만 새로운 유형의 데이터에는 대응하기 어렵다. 결국 263개의 feature는 기존에 존재하는 패밀리에 대해서는 높은 정확도의 분류가 가능하지만, 알려지지 않은 패밀리의 분류를 위해서는 Ⅲ에서 제시된 바와 같이 최소 4972개의 feature를 사용하여야 한다.

V. 결 론

본 논문에서는 제어 흐름 그래프를 통해 API 간의 호출 관계를 분석하고, 추출된 시스템 API를 바탕으로 APK 내에서 사용되는 API를 그룹화, 축소시켜 악성행위 카테고리 분류를 수행하였다. 결과적으로는 39032개의 feature를 기존에 비해 87.3% 만큼의 축소율을 가지는 4972개의 API 그룹으로 축소할 수 있으며, 이를 축소된 feature로 사용 가능하다. 이는 머신러닝의 오버피팅에 대응하기 위함이며, 오버피팅으로 인해 새로운 데이터 셋에 대응하지 못하는 문제를 해결 가능하다. 그 이유는 feature의 일반화(Nomalization)를 수행하기 때문이며, 또한 모든 API를 feature에 포함하여 학습하기 때문이다.

본 논문에서 제안된 방식은 Java bytecode를 기반으로 하는 *Soot*[15]와 *FlowDroid*를 활용하였으므로, native code에 대해 탐지가 불가능한 단점이 있다. 또한 본 논문에서 제안된 방식으로는 특정 카테고리에 빈번하게 사용되는 API를 수동으로 분류해야 한다는 단점이 있다. 따라서 향후 연구를 위해서 native code에 대해서도 제어 흐름 그래프를 생성하는 것과, 자동화된 툴에 의해 악성 어플리케이션의 카테고리를 분류하는 것이 필요하다.

References

[1] Gartner, "Gartner Says Worldwide Sales of Smartphones Recorded First Ever Decline During the Fourth Quarter of 2017," Feb, 2018.

[2] H. Vafaie and K. De Jong, "Genetic algorithms as a tool for feature selection in machine learning," Proceedings Fourth International Conference on Tools with Artificial Intelligence TAI '92, pp. 200-203, Nov. 1992.
[3] S. Lawrence and C. L. Giles, "Overfitting and neural networks: conjugate gradient and backpropagation," Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000, vol.1, pp. 114-119, Jul. 2000.
[4] Ian Goodfellow and Yoshua Bengio and Aaron Courville, Deep Learning, MIT Press, 2016.
[5] Arnulf B. A. Graf, Alexander J. Smola, Silvio Borer, "Classification in a normalized feature space using support vector machines," IEEE Transactions on Neural Networks, vol.14, No.3, pp. 597-605, May. 2003.
[6] J. Zico Kolter, Andrew Y. Ng, "Regularization and feature selection in least-squares temporal difference learning", ICML '09 Proceedings of the 26th Annual International Conf. on Machine Learning, pp. 521-528, Jun. 2009.
[7] Saurabh Mukher jee and Dr. Neelam Sharma, "Intrusion Detection using Naive Bayes Classifier with Feature Reduction," 2nd International Conf. on Computer, Communication, Control and Information Technology (C3IT-2012), pp. 119-128, Dec. 2012.
[8] Parnika Bhat and Kamlesh Dutta. "A Survey on Various Threats and Current State of Security in Android Platform," ACM Comput. Surv, vol.52, no.1, pp. 21-35, Feb. 2019.
[9] Li Li, Tegawendé F. Bissyandé, Mike Papadakis, Siegfried Rasthofer, et al, "Static analysis of android apps: A systematic literature review," Information and Software Technology, vol. 88, no. C, pp. 67-95, Aug. 2017.
[10] Naser Peiravian and Xingquan Zhu, "Machine Learning for Android Malware Detection Using Permission and API Calls," 2013 IEEE 25th International Conference on Tools with Artificial Intelligence, pp. 300-305, Nov. 2013.
[11] Suleiman Y. Yerima, Sakir Sezer, Gavin McWilliams, Igor Muttik. "A New Android Malware Detection Approach Using Bayesian Classification," 2013 IEEE 27th International Conference on Advanced Information Networking and Applications, pp. 121-128, Jun. 2013.
[12] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden et al. "Flowdroid: Precise context, flow, field, object-sensitive and life-cycle-aware taint analysis for android apps," Acm Sigplan Notices, vol. 49, no. 6, pp. 259-269, Jun. 2014.
[13] Shannon CE, The Mathematical Theory of Communication, The University of Illinois Press, Jan. 1999.
[14] Fengguo Wei, Yuping Li, Sankardas Roy, Xinming Ou et al. "Deep Ground Truth Analysis of Current Android Malware," International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA'17), pp.252-276, Jul. 2017.
[15] Raja Vallée-Rai, Phong Co, Etienne Gagnon, Laurie Hendren et al. "Soot: A java bytecode optimization framework," In CASCON First Decade High Impact Papers, pp.214 - 224, Nov. 2010.

〈저자소개〉

심 현 석 (Hyunseok Shim) 학생회원
2019년 2월: 숭실대학교 전자정보공학과 졸업
2019년 3월~현재: 숭실대학교 정보통신공학과 석사과정
〈관심분야〉 모바일 보안, 제어 흐름 분석, 개인정보 보호

박 정 수 (Jungsoo Park) 학생회원
2013년 2월: 숭실대학교 전자정보공학과 졸업
2015년 2월: 숭실대학교 융합소프트웨어학과 석사
2015년 3월~현재: 숭실대학교 융합소프트웨어학과 박사
〈관심분야〉 모바일 보안, 클라우드 보안

단티엔북 (Thien-Phuc Doan) 학생회원
2018년 10월: Vietnam National University, Cyber Security, B.S.
2019년 3월~현재: 숭실대학교 정보통신공학과 석사과정
〈관심분야〉 모바일 보안, 웹 보안, 악성코드 분석

정 수 환 (Souhwan Jung) 종신회원
1985년 2월: 서울대학교 전자공학과 졸업
1987년 2월: 서울대학교 전자공학과 석사
1996년 6월: University of Washington 박사
1988년~1991년: 한국통신 전임 연구원
1997년~현재: 숭실대학교 전자정보공학부 교수
〈관심분야〉 클라우드 보안, 모바일 보안, 네트워크 보안