J. Internet Comput. Serv.
ISSN 1598-0170 (Print) / ISSN 2287-1136 (Online)
http://www.jics.or.kr

중복 허용 범위를 고려한 서바이벌 네트워크 기반 안드로이드 저자 식별☆

Survival network based Android Authorship Attribution considering overlapping tolerance

황 철 훈[1] 신 건 윤[1] 김 동 욱[1] 한 명 묵[1]*
Cheol-hun Hwang Gun-Yoon Shin Dong-Wook Kim Myung-Mook Han

요 약

안드로이드 저자 식별 연구는 좁은 범위에서는 출처를 밝히기 위한 방법으로 해석할 수 있으나, 넓은 범위에서 본다면 알려진 저작물을 통해 유사한 저작물을 식별하는 통찰력을 얻기 위한 방법으로 해석할 수 있다. 안드로이드 저자 식별 연구에서 발견되는 문제점은 안드로이드 시스템 상 중요한 코드이지만 의미가 없는 코드들로 인하여 저자의 중요한 특징을 찾기 어렵다는 것이다. 이로 인해 합법적인 코드 또는 행동들이 악성코드로 잘못 정의되기도 한다. 이를 해결하기 위하여 서바이벌 네트워크 개념을 도입하여 여러 안드로이드 앱에서 발견되는 특징들을 제거하고 저자별로 정의되는 고유한 특징들을 생존시킴으로써 문제를 해결하고자 하였다. 제안하는 프레임워크와 선행된 연구를 비교하는 실험을 진행하였으며, 440개의 저자가 식별된 앱을 대상으로 실험한 결과에서 최대 92.10%의 분류 정확도를 도출하였고 선행된 연구와 최대 3.47%의 차이를 보였다. 이는 적은 양의 학습데이터를 이용하였으나 저자별 중복된 특징 없이 고유한 특징들을 이용하였기에 선행 연구와 차이가 나타났을 것으로 해석하였다. 또한 특징 정의 방법에 따른 선행 연구와의 비교 실험에서도 적은 수의 특징으로 동일한 정확도를 보일 수 있으며, 이는 서바이벌 네트워크 개념을 통한 지속적으로 중복된 의미 없는 특징을 관리할 수 있음을 알 수 있었다.

☞ 주제어 : 안드로이드 저자 식별, 저자 식별, 중복 특징 제거, 서바이벌 네트워크

ABSTRACT

The Android author identification study can be interpreted as a method for revealing the source in a narrow range, but if viewed in a wide range, it can be interpreted as a study to gain insight to identify similar works through known works. The problem found in the Android author identification study is that it is an important code on the Android system, but it is difficult to find the important feature of the author due to the meaningless codes. Due to this, legitimate codes or behaviors were also incorrectly defined as malicious codes. To solve this, we introduced the concept of survival network to solve the problem by removing the features found in various Android apps and surviving unique features defined by authors. We conducted an experiment comparing the proposed framework with a previous study. From the results of experiments on 440 authors' identified apps, we obtained a classification accuracy of up to 92.10%, and showed a difference of up to 3.47% from the previous study. It used a small amount of learning data, but because it used unique features without duplicate features for each author, it was considered that there was a difference from previous studies. In addition, even in comparative experiments with previous studies according to the feature definition method, the same accuracy can be shown with a small number of features, and this can be seen that continuously overlapping meaningless features can be managed through the concept of a survival network.

☞ keyword : Android Authorship Attribution, Authorship Attribution, Remove duplicate features, Survival network

1. 서 론

블록체인, 5G 네트워크 등의 여러 분야 IT 기술은 산업 전 분야에 적용되어 사용자에게 향상된 편의성과 접근성을 제공한다. 안드로이드 모바일 플랫폼 역시 사용자들과의 생활에 밀접한 관계를 맺고 있는 대표적인 예시로 기존의 통신, 검색 등의 수단에서 뱅킹, 인증과 같은 중요 정보를 이용하는 서비스로 확대되고 있다. IT 기술

1 Department of Software, Gachon University, Seongnam-si, 13120, Korea.
* Corresponding author (mmhan@gachon.ac.kr)
[Received 26 June 2020, Reviewed 29 July 2020(R2 12 October 2020), Accepted 9 November 2020]
☆ 이 논문은 2018년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (NRF-2018R1D1A1B07050864).

의 발전은 항상 긍정적인 영향만을 갖지 않는다. 악성코드 모듈화, 표준화로 인한 생산력 강화와 암호화, 변이 등의 우회 기술 고도화로 인해 악성공격의 위협에 손쉽게 노출되고 있다[1].

연구자들은 이러한 악성 앱, 악성 메일 등의 악성공격에 대응하기 위한 방법으로 머신러닝, 인공지능을 활용하고자 하며, 악성코드 탐지 또는 분류를 위해 머신러닝, 인공지능을 도입함으로써 빠르게 증가하고 변이되는 악성코드에 대응하고자 하였다[2, 3].

저자 식별 연구는 알려진 저작물에 대한 저자 정보를 특징으로 정의하여 알려지지 않은 저작물에 대해 저자를 분류하는 연구이다. 좁은 범위에서 저자 식별 연구는 '누가 이것을 제작하였는가?'에 대한 질문에 답을 하기 위한 연구 분야로써 제작물에 대한 출처를 남기는 것으로 해석할 수 있다. 넓은 범위에서 본다면 알려진 악성코드와 유사한 악성코드를 식별하는 통찰력을 얻기 위한 연구로 확장하여 해석할 수 있다[4, 5, 6].

안드로이드 저자 식별 연구는 배포되고 있는 앱을 대상으로 저자 또는 저자 그룹에 대한 특징을 정의하여 저자를 알 수 없는 앱에 대하여 저자를 식별하는 연구이다. 주로 dex 파일을 이용하며, string, class, method 영역에서 함수 호출 순서, 함수명, 함수 정의 클래스 등의 정보를 추출하여 특징으로 이용한다. 안드로이드 특성 상 중요 정보를 얻기 위해서는 사전에 정의된 API를 이용해야 하며, 함수에 따라 매개 변수가 필요하다. 이러한 특징은 변이 기술로도 변경할 수 없는 민감한 부분이다. 선행 연구에서는 이러한 특징을 이용하여 저자 또는 저자 그룹을 대상으로 민감한 애플리케이션 프로그래밍 인터페이스 그래프(Sensitive API Graph)특징을 정의하였으며, 저자 또는 저자 그룹별 반복되는 특징에 대한 가중치를 산출해 특징을 선택하였다[7].

본 연구에서는 안드로이드에서 저자 또는 저자 그룹별 고유한 특징을 정의하여 저자 식별을 할 수 있는 방법을 제안한다. 고유한 특징이란 저자별 중복되는 특징을 최소로 정의하는 것을 말한다. 안드로이드의 특징 정의 단계에서 나타나는 문제점은 보일링 코드로 인한 의미 없는 함수 호출 또는 저자 스타일에 관련 없는 형식적인 함수 흐름이 특징으로 선택될 수 있다는 것이다. 이러한 문제를 보완하기 위해 서바이벌 네트워크 개념을 저자별 특징 정의 단계에 도입하여 중복 허용 범위를 지정하는 것으로 저자의 고유한 특징을 유동적으로 선택할 수 있는 방법을 제안한다.

본 구성은 다음과 같다. 제 2장에서 관련 연구를 간략히 소개하고, 3장에서 제안하는 안드로이드 저자 식별 프레임워크를 설명한다. 4장을 통해 안드로이드 저자 식별에 대한 선행 연구와 비교 실험 및 결과에 대해 작성하였으며, 5장에서 결론 및 향후 연구에 대해 작성하였다.

2. 관련 연구

2.1 악성코드 저자 식별

1970년대 초, 최초의 컴퓨터 바이러스가 등장하여 ARPANET(Advanced Research Projects Agency Network)와 부딪힌 이후 보안 커뮤니티의 관심은 악성코드 작성자의 신원을 노출하는 것으로 해결하고자 하였다. 초기에는 악성코드 저자의 경험 부족으로 인해 상대적으로 단순한 악성코드와 싸웠기 때문에 간단히 악성코드 저자를 식별할 수 있었다. 그러나 현대의 악성코드 저자들은 수많은 악성코드 생성기를 통해 광범위하고 다양한 형태의 악성코드를 생성할 수 있으며, 고급 난독화 기술을 통해 저자의 신원을 쉽게 숨길 수 있게 되었다[8, 9]. 그러나 이와 같은 어려운 상황 속에서도 저자 식별을 통해 악성코드 저자를 파악하는 통찰력을 얻게 됨으로써 알려지지 않은 악성코드에 대한 식별의 가능성을 얻게 되는 이점은 무시하기 아깝다[10]. 연구자는 안드로이드 저자 식별을 위한 점진적인 접근 방식으로 프로파일을 이용하는 방법을 제안하였다[11]. 프로파일 방법은 학습을 통해 Dex 파일에서 전체 클래스 수, 인터페이스가 포함된 전체 클래스 수, 어노테이션이 포함된 전체 클래스 수, 전체 가상 함수의 수, 전체 추상 함수의 수 등과 같은 13개의 특징에 대해 저자 또는 저자 그룹별 정의하는 것을 의미한다. 이를 벤치마크 프로필(Benchmark profile)이라고 정의하였으며, 유사성 측정(Similarity Assesment)을 통해 입력된 데이터의 특징과 비교하여 유사도를 측정하는 방법과 분류 모델을 학습을 통해 저자를 분류하는 방법을 같이 사용하는 하이브리드 형태의 프레임워크를 제안하였다. 제안된 프레임워크는 시장에 배포한 무작위 앱 131,000개를 대상으로 실험을 진행하였으며, 97.5%의 정확도로 저자를 식별하였다.

2.2 안드로이드 악성 앱 분류

안드로이드 악성 앱 수가 급격히 증가하면서 악성코드 예방 시스템에 큰 문제가 발생하였다. 악성코드 샘플 수가 많을수록 악성코드 분석 시스템이 따라가질 못하기

```
1  .class public final Lcom/geinimi/c/f;
2  .method public constructor <init> (Lcom/geinimi/Adservice;)V
3      ...
4      invoke-virtual {p0},Landroid/telephony/TelephonyManager;->getDeviceId()Ljava/lang/String;
5      invoke-virtual {p0},Landroid/telephony/TelephonyManager;->getLine1Number()Ljava/lang/String;
6      invoke-virtual {p0},Landroid/telephony/TelephonyManager;->getVoiceMailNumber()Ljava/lang/String;
7      move-result-object v0
8      sput-object v0,Lcom/geinimi/c/f;->t:Ljava/lang/String;
9      new-instance v0,Landroid/os/Build;
10     invoke-direct v0,Landroid/os/Build;-><init>()V
11
12 .class public final Lcom/xlabtech/MonsterTruckRally/rally/e/k;
13 .method public constructor <init> (Lcom/xlabtech/MonsterTruckRally/rally/e;)V
14     ...
15     invoke-virtual {p0},Landroid/telephony/TelephonyManager;->getDeviceId()Ljava/lang/String;
16     invoke-virtual {p0},Landroid/telephony/TelephonyManager;->getLine1Number()Ljava/lang/String;
17     invoke-virtual {p0},Landroid/telephony/TelephonyManager;->getVoiceMailNumber()Ljava/lang/String;
18     move-result-object v0
19     sput-object v0, Lcom/xlabtech/MonsterTruckRally/rally/e/k;->v:Ljava/lang/String;
```

(그림 1) geinimi 악성 앱의 동일 기능에 대한 변이 예시
(Figure 1) Example of variations for the same function in geinimi malicious app

때문이다[12, 13]. 연구자는 동일한 제품군이 동일하게 갖는 기능들을 특징으로 정의함으로써 악성코드 탐지 및 검사를 가속화할 수 있을 것으로 가정하였다[7]. 기존 악성코드 예방 시스템의 문제는 합법적인 특징을 악성 특징으로 판단한다는 것이다. 또한 변형 공격에 대하여 탐지하기 힘들다는 문제가 있다. 전자의 경우 안드로이드 특성 상 Layout 및 Interface 관련 클래스 호출과 같이 시스템 상 중요한 함수이며 빈번하게 나타나지만 실질적인 의미를 갖지 않은 것들이 많기 때문이다. 이러한 특성으로 인하여 일부 합법적인 특징들도 악성 앱 분석 과정에서 빈번하게 나타남으로써 악성 특징이라고 판단하는 오류를 가질 수 있다. 후자의 경우 그림 1과 같은 상황을 통해 설명할 수 있다. 그림 1은 geinimi 악성 앱 제품군의 변이 결과를 보여준다. 상단과 하단으로 구분되며, 동일한 기능을 수행한다. 해당 함수는 디바이스 ID, 전화번호, 이메일 정보를 얻기 위한 역할을 한다. 빨강색으로 표시된 부분은 변이된 부분으로 함수를 정의한 클래스 명, 함수 매개 인자가 다르게 정의되었으며, 상단은 추가적으로 2개의 명령문이 함수 내부에 포함되어 있다. 하지만 중요 정보를 얻기 위한 API 함수는 변하지 않는다. 연구자는 관찰된 결과를 통해 변이 과정에서 바뀌지 않은 중요 API 함수를 찾아 특징으로 정의함으로써 언급한 두 가지 문제 모두를 해결할 수 있을 것으로 가정하였다. 연구자는 이러한 API 함수를 Sensitive API Call 특징으로 정의하였으며, 동일 제품군에서 빈번하게 나타나는 Sensitive API Call 특징들을 Fregreaph로 정의하였다. Fregraph에 포함된 Senstivie API Call 특징들은 각 가중치를 갖으며, 가중치 산출 식은 다음과 같다.

$$w(s,f) = per(s,f) * \log \frac{allNum}{totalNum(s)} \quad (1)$$

f는 악성코드 제품군을 의미하며, s는 해당 제품군의 Sensitive API Call을 의미한다. $allNum$은 수집된 악성코드 샘플의 크기를 나타내며, $totalNum(s)$는 s의 전체 크기를 나타낸다. $per(s,f)$는 해당 제품군에서 정의한 s의 크기를 전체 샘플과 비교한 비율이며, 해당 식은 다음과 같다.

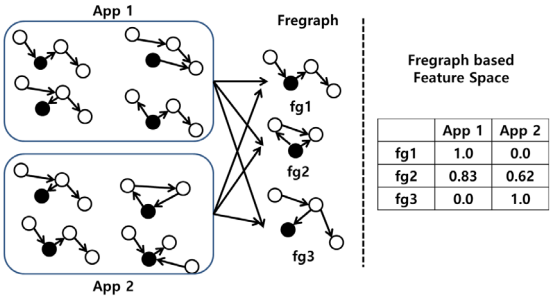
$$per(s,f) = \frac{Num(s,f)}{falNum(f)} \quad (2)$$

$Num(s,f)$는 f에 정의된 s의 전체 크기이며, $falNum(f)$은 f의 전체 크기를 의미한다. 정의된 식들을 통해 f에 대한 s의 가중치 $w(s,f)$를 산출할 수 있다. 가중치는 해당 Sensitive API Call 특징이 Fregraph에서 얼마나 중요한지를 나타낸 값이다. 이는 위에 설명한 다형성으로부터 변이되지 않는 중요 행동 특징을 찾기 위한 기준으로 이용된다.

연구자는 Fregraph를 통해 분류 모델을 학습하고 악성코드를 분류하는 시스템인 FalDroid를 제안하였다[7]. 제안된 시스템의 분류 모델을 비교하기 위해 서포트 벡터 머신(Support Vector Machine, SVM), C4.5, K-NN, 랜덤포레스트(Random Forest)를 이용하였다. 실험 결과, SVM 모델이 전체적으로 높은 정확도를 보였으며, 0.9 이상의 가중치를 기준으로 특징을 정의할 때 전체적으로 낮은 정확도를 보였다. 0.5에서 0.8사이의 가중치 값으로 특징을 정의하였을 때 무난한 분류 결과를 얻을 수 있다. 이를 통해 특징 선별 시 가중치가 높다고 해서 반드시 좋은 결과를 갖지 않음을 알 수 있었다.

2.3 Sensitive API graph

안드로이드 앱은 정의된 API를 통하여 기능 및 화면을 구현한다. 이러한 API들 중에는 Layout, Interface 등과 같이 시스템 상으로 중요하며 자주 사용되지만 앱의 고유한 의미를 갖는 특징으로써 이용하기는 힘든 API가 존재한다. 이를 해결하기 위하여 의미 없는 반복된 특징을 제거하고 안드로이드 앱의 고유한 특징을 내포하고 있는 API를 특징으로 정의하여 탐지 및 분류하는 연구가 진행되고 있다[13, 14]. 안드로이드 앱의 고유한 특징으로써 이용될 수 있으며, 탐지 및 분류에 있어 중요한 API들을 '민감한 데이터(Sensitive data)[13]', '민감한 어플리케이션 프로그래밍 인터페이스(Sensitive API)[14]'라고 정의하였다. Ming Fan 외 8명의 연구원들이 진행한 연구[14]에서는 앞선 Sensitive API를 정의하는 목적을 포함하여 바이



(그림 2) Fregraph based Feature Space 정의 과정
(Figure 2) Fregraph based Feature Space definition process

러스의 다형성에 대한 문제를 해결하기 위해 Sensitive API 특징들을 3 sub-graph 형태로 정의하는 'Frequent Subgraph'를 정의하였다. 정의된 Fregraph는 알려진 악성 앱을 탐지 및 분류할 수 있는 민감한 특징이며, 알려지지 않은 악성 앱으로부터 Fregraph를 정의하여 알려진 Fregrph와 유사성을 가중치로 산출한다. 이를 'Fregraph based Feature Space'라 정의하였다. 해당 과정은 그림 2와 같다. 검은색 원은 Sensitive API이며 흰색 원은 Sensitive API를 중심으로 순차적으로 진행된 API이다. 이를 방향성을 가진 그래프로 각 특징을 표현하며 이를 Fregraph라 정의한다. 각 'App 1'과 'App 2'의 Fregraph를 정의하며, 알려진 악성 앱으로부터 사전 정의된 Fregraph인 'fg1', 'fg2', 'fg3' 특징과 비교한다. 각 안드로이드 앱과 사전 정의된 악성 앱의 Fregraph 비교를 통해 각 앱이 어떤 특징과 유사성을 보이는지 표로 정의하며, 정의된 'Fregraph based Feature Space'를 안드로이드 앱에 대한 최종 특징으로 이용한다. 연구에서는 최종 정의된 특징을 기반으로 SVM (linear kernel), Decision Tree (C4.5), K-NN (k=1), Random Forest를 통한 분류 연구를 진행하였다. 실험 결과에서는 SVM을 통한 분류에서 높은 정확도를 보였으며, 30개의 악성 패밀리를 대상으로 분류 정확도를 평가한 결과 94.5%의 정확도로 분류할 수 있음을 보였다.

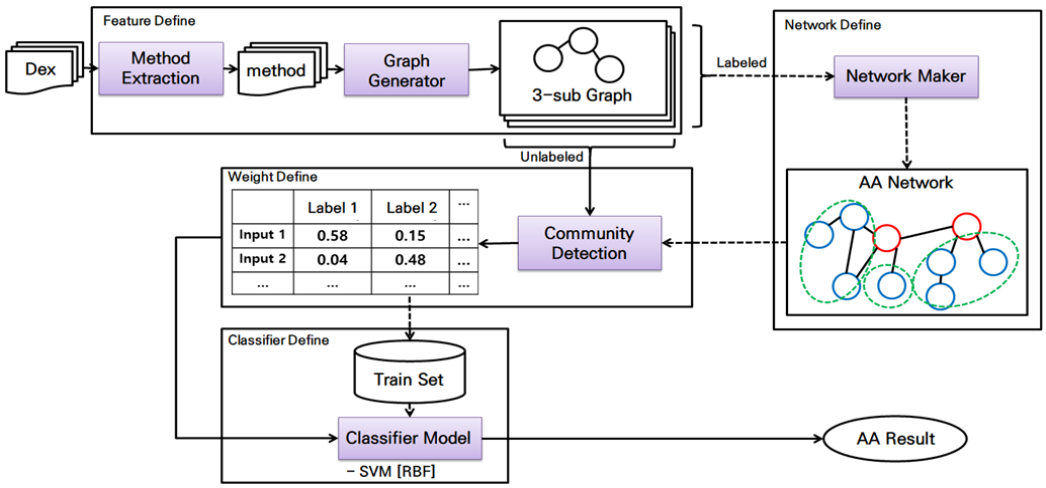
연구에서는 'Community Detection', 'Subgraph Matching', 'Subgraph Clustering' 과정을 통해 Fregraph를 정의해야 된다. 이 과정에서 Support Threshold를 이용하여 Fregraph를 유연하게 정의할 수 있음을 보였다. 본 연구에서는 위의 선행 연구를 바탕으로 하였으며, Sensitive API 특징을 정의하는 방법을 단순화하고 지속적인 학습을 통해 각 악성앱의 고유한 특징들이 관리될 수 있는 방법에 대한 연구를 진행하였다.

3. 안드로이드 저자 식별 방법

본 연구에서 제안하는 방법은 Ming Fan 외 8명의 연구원들이 진행한 연구[14]들의 선행 연구를 중점으로 안드로이드 앱과 악성 앱의 중요 특징인 Senstive API를 정의하고 지속적인 학습을 통하여 동적으로 관리할 수 있는 방법이다. 기본적으로 정의된 특징은 그래프 형태이며, 이를 하나로 통합하여 커뮤니티(Community)를 정의할 수 있다. 본 연구자는 이 과정에서 '만약 악성 앱들의 그래프 특징을 하나의 커뮤니티로 통합된다면, 서로 다른 악성 앱 간의 중복된 특징과 같은 악성 앱 간의 중복된 특징의 의미는 다를 것.'으로 가정하였다. 즉, 통합된 악성 앱들의 특징에서 서로 다른 앱 간의 중복은 의미 없는 API 특징으로 정의할 수 있으며, 반대로 같은 앱 간의 중복된 특징은 Sensitive API로 정의될 수 있음을 가정한 것이다. 이는 선행 연구에서 Support Threshold를 통해 동일 악성 앱의 Subgraph 특징 간의 어느 정도 유사도를 통해 Fregraph 특징으로 정의할 것인지 정의하는 과정에서 제안하는 가정이 타당성을 가질 수 있음을 나타낼 수 있다. 제안하는 방법이 실현되기 위해서는 각 그래프 특징을 하나의 커뮤니티로 정의하는 과정에서 중복된 특징 간의 제거 또는 생존을 정의해야 한다. 이를 해결하기 위해 서바이벌 네트워크 개념을 이용하였다. 서바이벌 네트워크는 네트워크 내에 규칙을 통해 노드 또는 간선에 대하여 생존 또는 제거를 하여 네트워크를 관리하는 것이다. 이 개념을 통해 저자별 특징 간의 중복성을 기준으로 제거 또는 생존을 결정하는 것이다. 기준은 수식, 조건, 기준, 수치 등과 같은 형태로 정의할 수 있다. 이를 이용하여 보안 분야에서는 네트워크 침입 탐지 연구가 진행되었다[3].

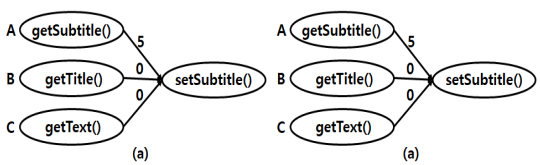
제안하는 시스템은 그림 3과 같다. 총 4개의 단계로 구분되며, 첫 단계는 Dex 파일로부터 method 정보를 추출하고 3-sub Graph 특징을 정의하는 특징 정의(Feature Define) 단계이다. 해당 단계에서 그림 2와 동일하게 API 함수를 하나의 노드(node)로 정의하며, API의 순서에 따른 순차성을 가진 3-sub Graph 형태로 정의한다. 단, 이 과정에서 선행 연구와는 다르게 중요 API 정보는 동적으로 찾기 위하여, Layout, Interface와 같은 화면 구성 API를 제외한 모든 API를 대상으로 진행한다.

네트워크 정의(Network Define) 단계에서는 이전 단계에서 정의된 3-sub Graph 특징 중 저자 또는 저자 그룹에 대한 라벨 정보가 존재하는 경우에 진행된다. 네트워크 제작기(Network Maker)는 라벨 정보가 포함된 3-sub Graph를 통합하여 하나의 Network를 정의한다. 통합 과정



(그림 3) 제안하는 안드로이드 저자 식별 프레임워크
(Figure 3) Proposed Android Author Attribution framework

에서 서바이벌 네트워크 개념을 도입하여 저자별 특징 중복이 나타날 경우에 유사도 가중치를 산출하여 해당 노드에 대한 생존 또는 제거를 결정한다. 유사도 가중치는 Cosine Similarity를 이용하여 산출되며, 그림 4를 통해 예시를 설명한다. 그림 4와 같이 5개의 학습 데이터를 통해 a와 b라는 두 개의 상황이 있다고 가정한다. A, B, C는 각 저자 또는 저자 그룹을 나타내며, 간선의 가중치는 해당 함수가 다음 함수로 이동한 횟수이다. a의 상황에 대하여 먼저 유사도를 산출한다면, A, B, C의 순서대로 (5, 0, 0)으로 정의할 수 있다. 해당 값을 정규화 하여 (1, 0, 0)으로 변환한다. 정규화 된 값은 A(1, 0, 0), B(0, 1, 0), C(0, 0, 1)와 각각 코사인 유사도(Consine Similarity)를 통해 유사도를 산출한다. 결과적으로 (getSubtitle() | setSubtitle()) 특징이 A에 대하여 1.0의 유사도를 갖으며, 나머지 B, C 클래스와는 0.0의 유사도 값을 갖는다. b의 상황에서도 a의 상황과 동일한 과정을 통해 유사도를 산출하며, 각 0.9, 0.3, 0.3의 유사도를 갖는다. 만약 중복 허용 기준이 0.5 이상일 경우, a, b 두 상황 모두 중복된 특징에 대하여 A 특징(getSubtitle() | setSubtitle())으로 정의할 수 있다. 하지만 중복 허용 기준이 0.95 이상일 경우, a의 상황에서만 중복된 특징에 대하여 빨강 특징으로 정의할 수 있다. 이런 과정을 통해 산출한 노드별 가중치를 기준으로 저자 식별 네트워크(Authorship Attribution Network, AA Network)를 정의할 수 있다. 가중치에 대한 조건을 통해 서바이벌 네트워크 규칙을 정의하는 것으로 생존한 노드와 제거된 노드로 구분되며, 저자별 생존 노드를 기준으로 저자 고유의 특징이 정의된 커뮤니티를 정의할 수 있다.



(그림 4) 유사도 가중치 산출 예시
(Figure 4) Example of calculating similarity weight

세 번째 단계에서는 첫 단계에서 정의한 3-sub Graph 특징 중 저자 및 저자 그룹 라벨이 정의되지 않은 데이터와 2단계에서 정의한 저자 식별 네트워크를 통해 가중치 테이블을 작성한다. 커뮤니티 검출(Community Detection)은 두 정보를 통해 입력된 데이터에 대한 가중치를 산출한다. 가중치 산출 식은 다음과 같다.

$$f(C_m) = \frac{\sum_{i=1}^{n} cmp(x_i, C_m)}{n} [m = 1,2,...,N] \quad (3)$$

입력 데이터 G에서 정의된 3-sub Call Graph의 크기를 n으로 정의하고, 각 저자별 커뮤니티를 C라고 정의한다.

m은 저자별 커뮤니티에 크기를 나타낸다. x는 3-sub Call Graph의 값[$G=x_1, x_2, ..., x_n$]이며, y는 클래스별 커뮤니티[$C=y_1, y_2, ..., y_m$]를 의미한다. $cmp(x_i, C_m)$은 다음과 같은 조건을 통해 값을 반환한다.

$$cmp(\alpha,\beta)\begin{cases}1 & \text{if } \alpha \ maches \ \beta \\ 0 & otherwise\end{cases} \quad (4)$$

α의 값이 β에 포함되는 경우 1의 값을 반환하며, 포함되지 않는 경우 0의 값을 반환한다. 가중치 산출 과정은 입력데이터에 정의된 특징들의 전체 크기에 대하여 저자별 커뮤니티에 포함된 비율을 산출하는 것이다.

마지막 단계인 분류 정의(Classifier Define) 단계는 3단계에서 정의한 입력데이터로부터 산출한 가중치를 이용한다. 만약 라벨이 부여된 경우, Train Set에 저장하여 분류 모델의 학습 데이터로 이용한다. 반대로 라벨이 부여되지 않은 경우, 분류모델의 입력데이터로 이용되어 최종 저자 분류 결과(Authorship Attribution Result, AA Result)를 반환한다.

4. 실험 및 결과

4.1 실험 데이터

안드로이드 저자 식별 실험을 위한 데이터 셋은 2018년 서스캐처원 대학의 보안 연구실에서 안드로이드 앱의 string 정보를 이용한 안드로이드 저자 식별 연구에서 이용되었다[15]. 데이터 셋은 13명의 저자를 대상으로 하였으나, 본 연구에서는 데이터 크기 문제로 인하여 12명의 저자를 대상으로 총 440개의 APK 파일을 실험에 이용하였다. 데이터 구성은 표 1과 같다.

(표 1) 안드로이드 저자 식별 데이터 셋
(Table 1) Transfer learning verification results

Author	Size	Author	Size
A	27	**G**	21
B	50	**H**	22
C	58	**I**	22
D	24	**J**	29
E	39	**K**	46
F	24	**L**	78

4.2 안드로이드 저자 식별 비교 실험

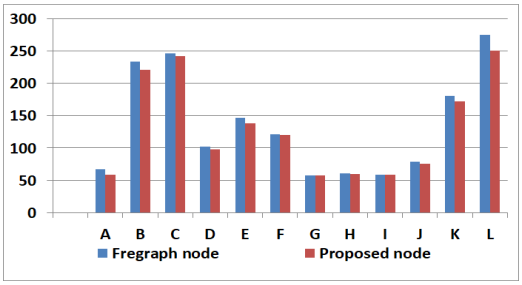
본 논문에서 제안하는 프레임워크는 서바이벌 네트워크 개념을 도입하여 저자별 특징의 중복 정도를 기준으로 특징 제거 또는 생존을 통해 저자 식별에 영향을 주는 고유의 특징을 정의할 수 있음을 전제로 한다. 총 실험은 두 단계로 비교하여 진행하였다.

첫 단계는 Fregraph를 이용한 선행 연구[14]와 제안한 방법의 비교를 통해 특징 정의 방법에 따른 정확도 변화를 확인하기 위함이며, 두 번째 단계는 제안하는 프레임워크의 효과를 확인하기 위하여 가중치를 이용하여 Sensitive API Call 특징을 통해 악성코드를 분류하는 선행연구[7]와 프로파일 방법을 이용한 저자 식별 선행 연구[11]를 통한 정확도 비교 실험을 진행하였다. 정확도는 F1-measure를 이용하여 산출하며, 다중 분류를 대상하였다. 이를 위해, 기존의 SVM (Linear Kernel)이 아닌 Cost, gamma 매개변수를 통해 다중 분류에 대처 가능한 SVM (RBF kernel)을 분류모델로 진행하였다. 각 실험은 10번의 교차 검증(Cross Validation) 방법을 통해 검증을 진행하였으며, 각 저자의 25%를 무작위로 선정하여 Test Set으로 이용하였고 나머지는 Training Set으로 이용하였다. Support Threshold와 Cosine Similarity는 실험을 통해 가장 최적의 값을 선정하였으며, 본 실험에서는 0.5와 0.90으로 각각 정의하였다. 첫 단계 실험 결과는 표 2와 그림 5와 같다. 제안하는 방법과 선행 연구 간의 정확도 비교에서는 두 실험 모두 동일한 정확도가 산출되었다. 하지만 두 실험에서 정의된 특징 노드의 수는 제안하는 방법이 기존 방법보다 낮은 수치를 보인다. 이는 네트워크로 정의된 커뮤니티 관리의 차이로 보이며, 학습을 통해 중복되는 특징을 지속적으로 제거 또는 생존을 정의하고 다른 악성 앱 간의 중복성을 제거하기 때문으로 보인다. 해당 실험을 통해 기존보다 더 중요한 특징들을 정의하여 적은 수의 특징으로 악성 앱을 분류할 수 있음을 알 수 있다.

두 번째 단계의 실험은 제안하는 프레임워크가 대체적으로 높은 정확도를 보였다. Sensitive API Call 특징을 이용한 연구는 데이터 크기가 50 이상 일 때 제안하는 프레임워크와 같은 정확도를 보이고 있지만 다른 저자에 대한 정확도는 세 가지 실험 중 가장 낮았다. 프로파일 방법을 이용한 저자 식별 연구는 F, I, K, L에 대한 저자 분류 정확도는 같으나 나머지 저자에 대한 정확도는 비교적 낮게 나타났다. 프로파일 특징 정의 방법과 실험 결과를 종합하여 해석한다면 프로파일 특징 정의 방식은 적은 양의 학습데이터 문제보다 저자가 얼마나 뚜렷한 특징을 보이는 가에 따라 정확도 결과가 달라질 것으로 보았다. 제안하는 프레임워크가 다른 실험보다 정확도가 높게 나타난 이유로 저자별 중복된 특징을 제거하여 저자별 간

(표 2) Sensitive API 정의 방법에 따른 비교 결과표
(Table 2) Comparison result table according to Sensitive API definition method

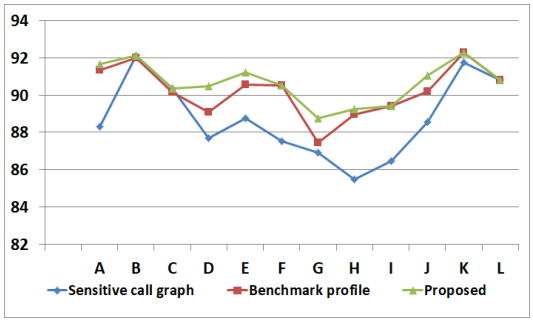
Author	Fregraph node	Fregraph Accuracy	Proposed node	Proposed Accuracy
A	67	91.64	59	91.64
B	234	92.1	221	92.1
C	246	90.33	242	90.33
D	102	90.47	98	90.47
E	147	91.2	138	91.2
F	121	90.5	120	90.5
G	58	88.74	58	88.74
H	61	89.26	60	89.26
I	59	89.4	59	89.4
J	79	91.04	76	91.04
K	181	92.27	172	92.27
L	275	90.82	251	90.82



(그림 5) Sensitive API 정의 방법에 따른 특징 노드 수
Figure 5) The number of feature nodes according to the Sensitive API definition method

(표 3) 안드로이드 저자 식별 연구 간의 비교 결과표
(Table 3) Comparison result table between Android author identification studies

Author	Size	Sensitive API Call	Benchmark Profile	Propose
A	27	88.30	91.33	**91.64**
B	50	**92.10**	91.98	**92.10**
C	58	**90.33**	90.14	**90.33**
D	24	87.68	89.10	**90.47**
E	39	88.76	90.56	**91.20**
F	24	87.54	**90.50**	**90.50**
G	21	86.93	87.46	**88.74**
H	22	85.50	88.97	**89.26**
I	22	86.47	**89.40**	**89.40**
J	29	88.56	90.20	**91.04**
K	46	91.75	**92.27**	**92.27**
L	78	**90.82**	**90.82**	**90.82**



(그림 6) 안드로이드 저자 식별 연구 간의 비교 실험
(Figure 6) Comparative experimental between Android author identification studies

섭이 완화됐기 때문으로 해석하였다. 각 저자에 따른 판단 기준이 되는 특징이 중복되지 않음으로써 분류 모델의 판단 기준이 명확해지기 때문에 적은 양의 학습 데이터에도 다른 선행 연구보다 높은 정확도로 저자를 식별할 수 있었다.

5. 결 론

악성코드의 기술 발전과 생산성 강화로 인하여 악성 공격에 대한 위협이 크게 증가하고 있다. 이를 해결하기 위한 방법으로 저자 식별 연구가 진행되었다. 안드로이드 저자 식별 연구의 가장 큰 문제는 저자에 대한 의미는 없으나 시스템 실행에 중요한 함수가 빈번하게 나타나기 때문에 특징 정의 시 합법적인 특징에 대하여 악성 특징으로 잘못 정의되는 일이 발생한다는 것이다. 이를 해결하기 위해 서바이벌 네트워크 개념을 도입하였다. 이는 여러 악성 앱의 특징을 하나의 커뮤니티로 정의하며, 중복 허용 범위를 통해 저자별 중복되는 특징의 정도를 기준으로 하여 저자의 고유한 특징을 유동적으로 정의할 수 있게 되었다. 실험을 통해 선행 연구보다 적은 수의 특징을 정의하여 동일한 결과를 얻을 수 있었으며, 다른 안드로이드 저자 식별 연구와 비교하여 적은 양의 학습데이터를 이용하더라도 저자별 중복된 특징이 제거되어 서로 간의 간섭이 사라져 다른 선행 연구보다 다소 높은 정확도 결과를 확인할 수 있었다.

추가적으로 다수의 저자를 대상으로 했을 때, 정확도 변화의 차이를 확인해야 되며, 변이 문제에 대한 대응을 가능성에 대하여 추가적인 연구를 진행할 것이다.

참고문헌(Reference)

[1] Y. Ye, T. Li, D. Adjeroh, S.S. Iyengar, “A survey on malware detection using data mining techniques,” ACM Computing Surveys (CSUR), Vol.50, No.2, pp. 1–41, 2017.
https://doi.org/10.1145/3073559

[2] E. Stamatatos, “A Survey of Modern Authorship Attribution Methods,” American Society for Information Science and Technology, Vol.60, No.3, pp 538-556, 2009.
https://doi.org/10.1002/asi.21001

[3] V. Q. Marinho, G. Hirst, D. R. Amancio, “Authorship Attribution via network motifs identification,” 2016 5th Brazilian Conference on Intelligent Systems, pp. 355-360, 2016.
https://doi.org/10.1109/bracis.2016.071

[4] G. Y. Shin, D. W. Kim, S. S. Hong, M. M. Han, “The Identification Framework for source code author using Authorship Analysis and CNN,” Journal of Internet Computing and Services, Vol.19, No.5, pp 33-41, 2018.
https://doi.org/10.7472/jksii.2018.19.5.33

[5] V. Kalgutkar, R. Kaur, H. Gonzalez, N. Stakhanova, A. Matyukhina, “Code authorship attribution: Methods and challenges,” ACM Computing Surveys (CSUR), Vol.52, No.1, 2019.
https://doi.org/10.1145/3292577

[6] S. Alrabaee, P. Shirani, M. Debbabi, L. Wang, ″On the Feasibility of Malware Authorship Attribution,” International Symposium on Foundations and Practice of Security, pp 256-272, 2016.
https://doi.org/10.1007/978-3-319-51966-1_17

[7] M. Fan, J. Liu, X. Luo, K. Chen, Z. Tian, Q. Zheng, T. Liu, “Android malware familial classification and representative sample selection via frequent subgraph analysis,” IEEE Transaction on Information Forensics and Security, Vol.13, No.8, pp. 1890-1905, 2018.
https://doi.org/10.1109/tifs.2018.2806891

[8] N. E. Rosenblum, B. P. Miller, Z, Zhu, “Extracting compiler provenance from program binaries,” In Proceedings of the 9th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering, pp. 21-28, 2010.
https://doi.org/10.1145/1806672.1806678

[9] N. E. Rosenblum, X, Zhu, B. P. Miller, “Who wrote this code? identifying the authors of program binaries,” In European Symposium on Research in Computer Security, pp. 172-189, 2011.
https://doi.org/10.1007/978-3-642-23822-2_10

[10] R. Chouchane, N. Stakhanova, A. Walenstein, A. Lakhotia, “Detecting machine-morphed malware variants via engine attribution,” Journal of Computer Virology and Hacking Techniques, Vol.9, No.3, pp. 137-157, 2013.
https://doi.org/10.1007/s11416-013-0183-6

[11] H. Gonzalez, N. Stakhanova, A. A. Ghorbani, “Authorship Attribution of Android Apps,” Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy, pp. 277-286, 2018.
https://doi.org/10.1145/3176258.3176322

[12] Y. Zhou, X. Jiang, “Dissecting Android malware: Characterization and evolution,” In 2012 IEEE symposium on security and privacy, pp. 95–109. 2012.
https://doi.org/10.1109/sp.2012.16

[13] Y. Feng, S. Anand, I. Dilling, A. Aiken, “Apposcopy: Semantics-based detection of android malware through static analysis,” In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 576-587, 2018.
https://doi.org/10.1145/2635868.2635869

[14] F. Ming, L. Jun, L. Xiapu, C. Kai, C. Tianyi, T. Zhenzhou, Z. Xiaodong, Z. Qinghua, and L. Ting, “Frequent Subgraph based Familial Classification of Android Malware,” In Proceedings of 2016 IEEE 27th International Symposium on Software Reliability Engineering, pp. 24-35, 2016.
https://ieeexplore.ieee.org/document/7774504

[15] V. Kalgutkar, N. Stakhanova, P. Cook, A. Matyukhina, “Android authorship attribution through string analysis,” In Proceedings of the 13th International Conference on Availability, Reliability and Security, pp. 1-10, 2018.
https://doi.org/10.1145/3230833.3230849

◐ 저 자 소 개 ◑

황 철 훈(Cheol-hun Hwang)
2019년 가천대학교 컴퓨터공학과(공학사)
2019년~현재 가천대학교 일반대학원 소프트웨어학과(공학석사)
관심분야 : 정보보호, 머신러닝, 인공지능, 모바일
E-mail : qewqsa@naver.com

신 건 윤(Gun-Yoon Shin)
2017년 가천대학교 인터랙티브 미디어 융합학과 학사
2018년 가천대학교 일반대학원 컴퓨터공학과(공학석사)
2018년~현재 가천대학교 컴퓨터공학과 박사과정
관심분야 : 기계 학습, 악성코드 분석, 공격자 식별, 저자 분석, 인공지능
E-mail : tlsrjsdbs@gmail.com

김 동 욱 (Dong-Wook Kim)
2015년 가천대학교 컴퓨터공학과(공학사)
2017년 가천대학교 일반대학원 컴퓨터공학과(공학석사)
2017년~현재 가천대학교 컴퓨터공학과 박사과정
관심분야 : Data Mining, 인공지능, Data fusion, Anomaly Detection
E-mail : kog7306@naver.com

한 명 묵(Myung-Mook Han)
1980년 연세대학교 공과대학(공학사)
1987년 뉴욕공과대학교 대학원 컴퓨터공학과(공학석사)
1997년 오사카시립대학교 대학원 정보공학부(이학박사)
1998년~현재 가천대학교 소프트웨어학과 교수
관심분야 : 정보보호, 인공지능
E-mail : mmhan@gachon.ac.kr