Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.31, NO.3, Jun. 2021 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2021.31.3.433

사용자 맞춤형 서버리스 안드로이드 악성코드 분석을 위한 전이학습 기반 적응형 탐지 기법*

심 현 석,[1†] 정 수 환[2‡]
[1,2]숭실대학교 (대학원생, 교수)

Customized Serverless Android Malware Analysis Using Transfer Learning-Based Adaptive Detection Techniques*

Hyunseok Shim,[1†] Souhwan Jung[2‡]
[1,2]Soongsil University (Graduate student, Professor)

요 약

안드로이드 어플리케이션은 생산성과 게임 등의 다양한 카테고리에 걸쳐 출시되며, 사용자는 개인의 사용 패턴에 따라 다양한 어플리케이션 및 악성코드에 노출된다. 반면 대부분의 분석 엔진은 기존에 존재하는 데이터셋을 활용하며, 주기적인 업데이트가 이루어진다고 해도 사용자의 선호도를 반영하지 않는다. 따라서 알려진 악성코드에 대한 탐지율은 높은 반면, 애드웨어와 같은 유형의 악성코드는 탐지가 어렵다. 또한 기존의 엔진은 서버를 거쳐야 하므로, 추가적인 비용이 발생하며, 사용자는 가용성과 실시간성을 보장받지 못하는 문제가 발생한다. 이러한 문제를 해결하기 위해 논문에서는 서버와 단 한번만의 통신이 요구되는 on-device 악성코드 분석과 전이학습을 통한 모델 재훈련을 수행하는 분석 시스템을 제안한다. 또한 해당 시스템은 디바이스 내부에서 디컴파일을 포함한 전체 프로세스가 이루어지므로, 서버 시스템에서의 부하를 분산할 수 있다. 이러한 분석 시스템을 구현하여 테스트한 결과, 전이 학습 이전 기준 최대 90.3%의 정확도를 얻었으며, Adware 카테고리에 대하여 전이학습을 수행한 뒤 최대 95.1% 의 정확도로, 기존 대비 4.8% 높은 정확도를 얻을 수 있었다.

ABSTRACT

Android applications are released across various categories, including productivity apps and games, and users are exposed to various applications and even malware depending on their usage patterns. On the other hand, most analysis engines train using existing datasets and do not reflect user patterns even if periodic updates are made. Thus, the detection rate for known malware is high, while types of malware such as adware are difficult to detect. In addition, existing engines incur increased service provider costs due to the cost of server farm, and the user layer suffers from problems where availability and real-timeness are not guaranteed. To address these problems, we propose an analysis system that performs on-device malware detection through transfer learning, which requires only one-time communication with the server. In addition, The system has a complete process on the device, including decompiler, which can distribute the load of the server system. As an evaluation result, it shows 90.3% accuracy without transfer learning, while the model transferred with adware catergories shows 95.1% of accuracy, which is 4.8% higher compare to original model.

Keywords: Android malware, Transfer learning, On-device analyzer, Adaptive system

Received(02. 22. 2021), Modified(05. 03. 2021), Accepted(05. 19. 2021)

* 본 연구는 과학기술정보통신부 및 정보통신기획평가원의 대학ICT연구센터지원사업의 연구결과로 수행되었음 (IITP-2021-2020-0-01602)

† 주저자, ant_tree@naver.com
‡ 교신저자, souhwanj@ssu.ac.kr(Corresponding author)

I. Introduction

안드로이드는 2008년 첫 버전이 출시된 이후 지난 몇 년간 안드로이드 악성코드 분석에 대한 연구가 꾸준히 이루어졌다. 그럼에도 불구하고 안드로이드 악성코드는 꾸준히 증가하였으며, 이러한 악성코드는 다양한 카테고리에 존재한다 [1]. 반면 사용자는 그 유형에 따라 자주 접근하는 어플리케이션 카테고리, 혹은 악성코드의 기준이 다르다. 예를 들어 특정 사용자의 디바이스 사용 유형이 게임으로 치우쳐져 있다면 게임 형태의 악성코드에 노출될 확률이 높다. 또한 adware 형태의 악성코드의 경우 사용자의 기준에 따라 악성 판별 여부가 결정된다. Adware 유형의 악성코드는 해당 어플리케이션에 삽입된 광고의 양과 그에 대한 사용자의 불쾌 여부가 다르다. 일부 사용자는 특정 어플리케이션에 삽입된 광고의 양이 적절하다고 느낄 수 있는 반면, 일부 사용자는 해당 어플리케이션으로 인한 사용자 경험을 악성코드 수준으로 판단할 수 있다. 따라서 사용자의 어플리케이션 사용 패턴 및 주관적인 사용자 경험에 대응되는 개인 환경에 최적화된 악성코드 탐지가 필요하다.

일반적으로 안드로이드 악성코드의 분석은 Android Run Time(ART)을 기반으로 하는 동적 분석과, Android Package(APK) 파일 정보를 기반으로 하는 정적 분석으로 나누어진다. 두 분석 방법 모두 연구에서 머신러닝 기법을 활용한 패턴 분석을 통해 악성코드를 탐지하는 방향으로 진행되어 왔다. 반면 기존의 연구들은 전달 받은 APK를 서버에서 분석하는 것을 목표로 하였으며, 이로 인해 서버단의 부하와 실시간성, 가용성의 부재가 발생하였다.

머신러닝 분야에서 흔히 일컫는 No *Free Lunch Theorem*[2]에 의하면 빌드된 모델은 학습용 데이터셋에 기반한 것으로, 아무리 최적화된 모델이라 하더라도 다른 문제에 마주하게 되면 의도대로 동작하지 않을 가능성이 높다. 특정 데이터로 학습된 모델은 해당 데이터에 종속된 도메인에서의 문제를 해결하기 위한 것으로, 그 외의 알려지지 않은 데이터를 분류하기에는 적합하지 않다.

본 논문에서는 위의 문제를 해결하기 위해 디바이스 내에서 Tensorflow Lite[3]를 활용한 on-device 분석을 진행하고자 한다. 사용자의 환경에 최적화된 모델을 생성하기 위해 베이스 모델에서의 전이학습을 수행하며, 전이된 모델은 사용자 디바이스로 전달된다. 이러한 과정을 위해 분석 시스템은 클라이언트 단에서 단 한번의 서버 요청만이 필요하며, 이 후 모든 분석은 디바이스 내에서 수행된다.

논문은 다음 장부터 배경지식을 설명하며, 3장에서 기존 연구와의 비교가 이루어진다. 이후 4장에서 논문에서 제안한 방법에 대한 구조를 설명하며, 5장에서 툴을 구현하기 위한 방법을 설명한다. 다음 6장에서는 툴에 대한 평가가 이루어지며, 논문은 7장의 결론을 통해 끝맺는다.

II. Background

2.1 Android Application

안드로이드 어플리케이션은 APK 파일 형태의 실행파일로 구성되며, 자바 기반의 classes.dex 파일과 AndroidManifest.xml 파일, 리소스를 담당하는 res/ 디렉터리와 리소스의 id 매핑을 담당하는 resources.arsc 파일 등으로 나눠진다[4]. 이러한 APK 파일 내부의 구성 요소는 Fig. 1과 같다.

이 중 classes.dex 파일은 안드로이드 어플리케이션 기능과 관련된 소스코드의 대부분을 가지고 있으며, 이는 디컴파일 시 smali 확장자의 파일로 추출 가능하다. 안드로이드 운영체제는 Dalvik VM(DVM) 혹은 Android Run Time(ART) 방식을 위해 APK 파일을 컴파일 하며, 각 방식에 따라 컴파일 형식이 다르다. ART 방식에서는 OAT 형태의 파일을 실행시키기 위해 컴파일 시 dex 파일을 odex 파일로 변환하는 과정이 수행된다[5].

반면 AndroidManifest.xml 파일은 안드로이드 소스코드를 포함하지는 않지만, 안드로이드 APK를

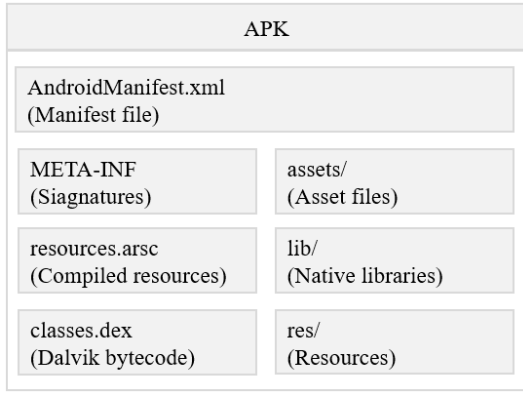

Fig. 1. Android APK internal structure.

구성하는 속성에 대한 정보를 가진다. 이는 APK 파일의 패키지명을 비롯하여 사용 권한 정보, 선언된 컴포넌트의 이름 및 어플리케이션 버전 등이 포함된다. AndroidManifest.xml 파일은 컴파일되는 classes.dex 파일과는 다르게 APK 파일을 압축해제하면 즉시 열람 가능한 특징을 가진다.

2.2 Transfer Learning

전이학습은 학습 데이터가 부족한 분야의 모델 구축을 위해 데이터가 풍부한 분야에서 훈련된 모델을 재사용하는 머신러닝 학습 기법으로, 좋은 성능의 모델의 일부 계층을 재활용하는 기법이다 [6]. 대부분의 모델 학습에서는 학습 및 예측을 위해 대량의 데이터를 필요로 하지만 실제로는 이러한 데이터셋을 구축하는 과정에 많은 리소스가 소모된다. 따라서 이러한 전이 학습을 통해 사전 학습된 모델의 fine tuning을 수행하며, 재사용된 모델을 이용하는 것이 효율적이다. 이러한 방식은 각각 기존 모델의 성능을 상속받는다는 점과 부족한 데이터로 인한 모델 훈련 문제를 해결할 수 있다는 장점이 있다.

모델의 fine tuning 과정에는 미리 학습된 모델의 가중치를 새로운 모델에 적용하는 방식과 기존 모델의 일부 layer만을 재사용하는 방식이 있다[7]. 반면 새로 훈련할 데이터가 적을 때는 전체 모델에 대한 fine-tune은 수행하지 않는다. 이는 오버피팅의 위험성이 있기 때문이며, 따라서 최종 선형 분류기만 학습을 수행한다. 반면 새로 훈련할 데이터가 많은 경우에는 오버피팅 가능성이 낮으므로 전체 레이어에 대한 fine-tune을 수행할 수 있다.

반면 이 경우에도 기존의 데이터 셋과 전이학습을 수행하는 데이터가 유사하지 않을 때를 주의해야 하는데, 기존 데이터와 유사하지 않으면 전체 모델을 재학습하는 것이 좋다. 데이터가 적은 경우 기존에 학습에 사용된 데이터와 상충되며, 데이터가 많은 경우는 무방하지만 전체 모델에 대해 fine-tuning이 이루어지는 것이 더 좋은 성능을 보인다[8].

III. Related Works

안드로이드 어플리케이션의 분석은 이전부터 꾸준히 연구된 분야로, 최근 디바이스 내의 분석을 수행하는 on-device 분석 관련 연구가 수행되었다. 이러한 연구는 서버를 거치지 않는 형태로, 일반적으로 디바이스 내에서 Tensorflow Lite를 통한 분석을 수행한다. 최근 연구 중 **Ruitao Feng el al.**은 실시간성과 빠른 반응성을 위해 디바이스 내에서 딥러닝을 통한 악성코드 탐지를 수행하였다[9]. 반면 Tensorflow Lite는 디바이스 내에서의 모델 빌드 기능을 제공하지 않기 때문에, 생성된 모델을 이용한 추론만이 가능하다. 따라서 기존 모델을 이용한 디바이스 내에서의 분석만이 가능하며, 이는 사용자 디바이스와의 최적화와는 관련이 없다.

Wei Yuan et al.은 경량화된 on-device 탐지 모델을 구현하여 디바이스 내 모델 빌드를 수행하였다 [10]. **Wei Yuan et al.**은 on-device 환경에서의 모델 빌드를 위해 Tensorflow Lite를 사용하지 않았으며, broad learning을 통한 모델을 생성하였다. 반면 이 경우에는 각 디바이스에서 모델을 처음부터 끝까지 생성해야 하며, 데이터의 편향과 데이터의 질적 수준을 보장할 수 없기 때문에 실제 환경에서 신뢰성의 문제가 발생한다.

IV. System Design

4.1 Architecture

본 논문에서 제안하는 구조는 Fig. 2와 같다. 사전 정의된 APK 데이터셋을 통해 베이스 모델을 훈련한다. 이후 서버단에 베이스 모델이 위치하게 되며, 이후 주기적으로 APK 정보를 통해 베이스 모델을 갱신한다. 사용자는 최초로 분석 시스템을 설치하는 시점에 디바이스에 설치된 어플리케이션 리스트에

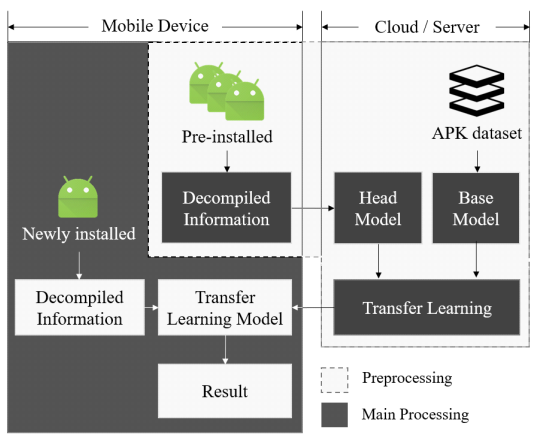

Fig. 2. System degisn with data processing steps / analysis pipeline.

서 비선호 여부를 결정한다. 결정된 리스트는 서버로 전송되어 헤드 모델을 생성하며, 이는 베이스 모델과의 전이 학습에 활용된다. 이 때 실제 환경에서는 어플리케이션이 사용자에 의해 선택되지 않은 경우 정상, 선택된 경우 악성으로 간주되어 전이 학습 모델에 반영된다. 논문에서는 이러한 실험을 위해 베이스 모델을 학습할 때 정상 앱과 악성 앱을 모두 사용하며, 각 타입에 따라 나눠진 카테고리의 어플리케이션을 사용한다. 또한 전이 학습을 진행하는 것은 모바일 내에서 불가능하므로, 전이 학습은 서버에서 진행된다. 해당 전이 학습 모델을 생성하는 것으로 Fig. 2에서의 *Preprocessing* 부분이 완료된다.

이후 분석 시스템은 디바이스로 전이 학습 모델을 전달하며, 디바이스 내에서 분석을 시작한다. 해당 과정은 Fig. 2에서의 *Main Processing* 부분에 해당하며, 이후에는 신규 설치 어플리케이션에 대한 분석을 수행한다. 이러한 분석 과정은 on-device 환경으로, 서버를 거치지 않는 디바이스 내 분석이다. 분석이 완료되면 사용자에게 분석 결과를 전달하며, 사용자의 필요에 따라 서버와 통신하며 새로운 전이 학습 모델을 생성할 수 있다.

4.2 Determining Preference

실제로 사용자가 악성앱과 정상앱을 구분하는 과정에서는 사용자의 선호도만이 반영된다. 논문에서의 아이디어는 미리 정의된 악성앱과 정상앱을 분석하는 것이 아닌, 사용자가 선호하는 앱과 선호하지 않는 유형을 구분하는 것이다. 따라서 사용자는 자신이 원하지 않는 유형(지나치게 많은 광고가 포함되어 있거나, 앱의 전반적인 사용성이 떨어지는 경우)의 어플리케이션을 사전에 학습시켜 이후 비슷한 유형이 어플리케이션이 설치되는 경우 알림을 받을 수 있다. 논문에서는 이러한 아이디어 중 전이학습을 통한 구현 가능성 여부만을 판단하며, 이를 위해 사전 정의된 선호 여부(악성/정상)을 사용하였다.

사용자의 APK를 분석하며 이를 통한 모델을 분석하는 과정에서는 개인정보에 대한 이슈가 발생할 수 있다. 한편 논문에서 제시된 방법은 APK 전체를 전송하지 않으며, 부분적으로 추출된 최소 정보만을 전송하기 때문에 개인정보 이슈에서 안전하다. 추출된 정보는 해당 정보를 통한 사용자 혹은 개인정보의 특정이 불가능하다.

V. Implementation

5.1 On-Device Decompiler

분석 시스템은 디바이스 내에서의 구동을 목표로 하므로, APK의 디컴파일은 디바이스 내에서 수행되어야 한다. 따라서 APK 디컴파일러인 **ApkTool**[11]을 안드로이드 내에서 실행 가능하도록 수정하여 사용한다. **ApkTool**은 기본적으로 리눅스와 맥, 윈도우 환경에서만 실행 가능하도록 설정되어 있지만, Fig. 3와 같이 *AaptManager* 내에서 해당 운영체제 관련 분기 처리를 제거하여 안드로이드 환경에서 구동 가능하다.

이러한 안드로이드 디바이스 내부에서의 디컴파일은 디컴파일을 위한 리소스 부족 혹은 일부 의존성의 문제가 발생한다. 반면 이런 디컴파일 오류가 발생하는 부분은 res/ 디렉터리 내의 컴파일되지 않은 리소스 혹은 레이아웃 관련 리소스에서만 발생한다. 따라서 어플리케이션 기능의 특징을 담고 있는 classes.dex 파일과 매니페스트 파일은 정상적인 디컴파일이 가능하다.

```
1  try {
2      if (OSDetection.isMacOSX()) {
3          aaptBinary = Jar.getResourceAsFile(
4              "/prebuilt/macosx/"
5              + aaptVersion
6              , AaptManager.class);
7      } else if (OSDetection.isUnix()) {
8          aaptBinary = Jar.getResourceAsFile(
9              "/prebuilt/linux/"
10             + aaptVersion
11             , AaptManager.class);
12     } else if (OSDetection.isWindows()) {
13         aaptBinary = Jar.getResourceAsFile(
14             "/prebuilt/windows/"
15             + aaptVersion + ".exe"
16             , AaptManager.class);
17     } else {
18         throw new BrutException(
19             "Could not identify platform: "
20             + OSDetection.returnOS());//Disable this line
21     }
22 } catch (BrutException ex) {
23     throw new BrutException(ex);
24 }
```

Fig. 3. AaptManager of ApkTool. Disabling throw statement in else block will enable ApkTool in Android environment.

5.2 Tensorflow Lite

Tensorflow Lite는 사전 훈련된 베이스 모델을

기반으로 전이학습을 지원한다. 이러한 Tensorflow Lite 모델은 .tflite 확장자의 모델을 입력으로 받는다. 이를 위해 기존의 Keras를 통해 학습된 h5 모델을 tflite 형태의 모델로 변환해야 한다. 이러한 파일간의 변환 프로세스는 Fig.4와 같다.

Keras 호환 모델인 h5 모델은 Tensorflow 내의 내장 라이브러리를 통해 pb 파일로의 변환을 수행할 수 있다. 반면 pb파일의 tflite 모델로의 변환은 내장 라이브러리를 통해 수행할 수 없다. 따라서 모델 간의 변환을 위하여 오픈소스 모델 변환 툴인 **Toco**[12]를 이용한다. **Toco**는 모델의 구조를 나타내는 바이너리인 pb 파일을 tflite 형태의 모델로 변환할 수 있기 때문에 h5 모델은 우선 pb 파일로 변환되어야 한다. 변환된 pb 파일은 다시 한번 tflite로 변환을 수행하며, 이를 위해서는 입력 데이터의 특징 셋 정보를 명시해야 한다. 이러한 과정은 Fig.5와 같으며, output_format 옵션에 TFLITE를 입력하여 Tensorflow Lite에서 이용 가능한 모델로의 변환을 수행할 수 있다.

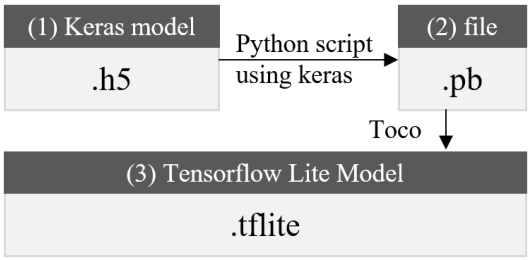

Fig. 4. File converting process.

```python
from tensorflow import keras
model = keras.models.load_model(trained_model.h5, compile=False)

export_path = '/model/pb/'
model.save(export_path, save_format='tf')
```

(a) Converting a keras model to a pb file

```
toco --input_file=converted_model.pb \\
    --output_file=converted_model.tflite \\
    --input_format=TENSORFLOW_GRAPHDEF \\
    --output_format=TFLITE \\
    --input_shape=1,12,12,1 \\
    --input_array=input \\
    --output_array=output \\
    --inference_type=FLOAT
```

(b) Converting a pb file to a tflite model

Fig. 5. Converting of keras file into a tflite model.

5.3 Input Data

베이스 모델은 CNN을 기반으로 하며, 따라서 2d-matrix를 입력 데이터로 사용한다. 현재 안드로이드 문서에서 정의한 권한의 경우 165개로, 이 중 사용 빈도가 낮은 21개의 권한을 제외한 144개의 데이터와 180개의 API 시그니처를 입력 데이터로 사용한다. 각 권한의 사용 빈도는 이전 연구에서 [13] 행해진 바와 같이 AOSP 상에서 정의되거나 사용된 권한의 빈도와 같으며, 상위 빈도 기준으로 사용되지 않은 권한을 제외하였다. 또한, 동일 수준으로 사용된 권한들은 해당 권한이 가지는 위험 레벨에 따라 분류되었다. 또한 API 시그니처는 고위험군의 권한을 사용하는 API를 상위 빈도 순으로 선별하였다.

각 권한의 경우 APK의 매니페스트 안에서 사용이 정의된 경우 1로 표시되며, 정의되지 않는 경우 0으로 표시된다. 따라서 학습과 테스트에 사용되는 모든 APK는 이러한 18*18 형태의 행렬로 변환된 데이터가 사용되며, 정확도를 향상시키기 위해 0 과 1을 flip하여 데이터화 하는 과정을 포함한다.

VI. Evaluation

구현된 시스템에 대한 분석은 다음과 같다. 시스템은 평가를 위해 안드로이드 10 운영체제를 사용하는 단말을 사용하였으며, Ubuntu 16.0.4 운영체제 기반의 전이학습 서버를 구현하였다. 평가는 시뮬레이션 기반으로, 사용자가 지정하는 어플리케이션에 대한 전이학습을 수행한 후 단말에서 모든 어플리케이션에 대해 평가를 수행하는 방향으로 이루어졌다.

평가를 위한 데이터셋은 Fig.6와 같이 구성된다. 데이터셋은 1,200개의 악성 APK와 300개의 양성 APK로 나누어지며, 따라서 총 1,500개의 APK로 이루어진다. 양성 APK는 **Google Play**[14]를 통해 수집된 APK를 **VirusTotal**[15]을 통해 검증하였다. 악성 APK는 **AMD Dataset**[16]과 **Adware Campaign**[17]을 통해 수집되었다. 수집된 APK는 총 750개의 adware와 450개의 그 외 APK 데이터셋으로 구성된다.

베이스 모델을 생성하기 위하여는 Inception-v3와 Inception-ResNet-v2의 두 가지를 통해 빌드를 수행하였다. 이 과정에서 각 알고리즘별로 optimizer는 RMSprop, SGD와 Adam 세가지를

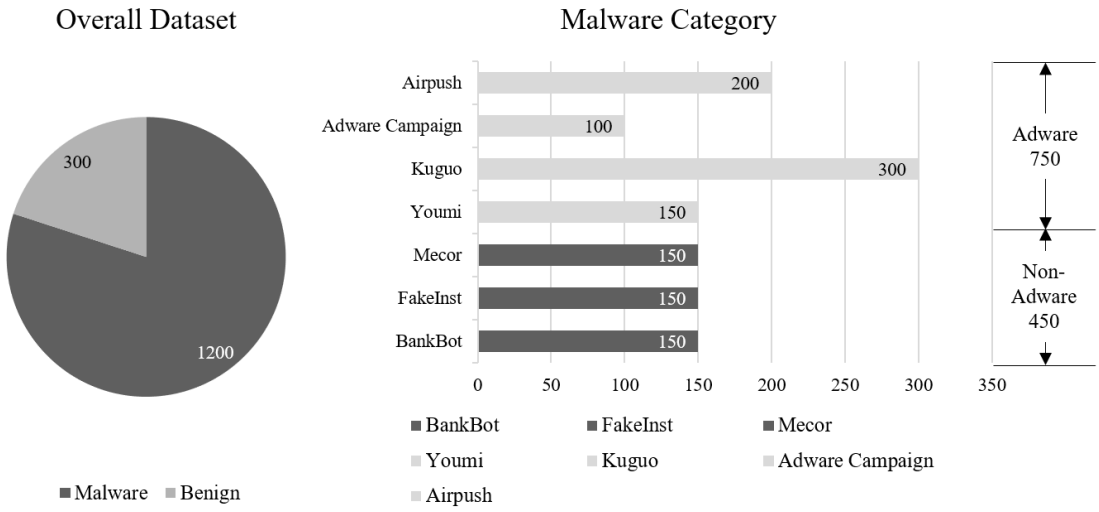

Fig. 6. Overall dataset portion with malware category. Malware dataset is consists of adware categories and non-adware categories.

사용하였다. 모델은 전이학습을 거치기 전 베이스 모델의 훈련 결과, 모든 알고리즘에서 세 optimizer 중 SGD가 가장 높은 성능을 보였다. 따라서 평가 결과에서는 SGD를 통한 결과만을 나타내었다.

각 평가는 하나의 전이학습을 거치지 않는 타입과, 3가지의 전이학습 타입으로 나누어 평가하였다. 전이학습을 거치지 않는 경우에는 양성 데이터셋을 포함한 총 3가지의 카테고리를 통한 학습을 수행하며, 나머지 5가지의 카테고리에서의 성능을 평가한다. 그 외에는 Table.1 에서의 *Base dataset*에 대응하는 데이터셋을 통해 베이스 모델을 생성하며, *Head dataset*에 대응하는 데이터셋을 통해 헤드 모델을 생성하여 전이 학습을 진행한다. 전이 학습은 adware을 탐지하기 위한 두가지 데이터셋 구성을 포함한다. 해당 두 데이터셋은 탐지하고자 하는 데이터셋인 *Test against*에 해당하는 카테고리의 50%를 전이 학습에 사용하는지 여부로 구분된다. 또한 모든 전이학습은 마지막 레이어만을 재사용하였으며, 모델 전체에 대한 fine-tuning은 수행하지 않았다.

평가 결과는 Fig.7과 같다. 전이학습을 수행하지 않았을 때 정확도는 Inception-v3와 Inception-ResNet-v2에서 각각 88.3%와 90.3%를 나타내었으며, 전이학습을 수행하였을 때는 각각 91.2%와 93.8%로 전이학습을 수행하지 않았을 때와 비교하여 평균 3.2% 높은 성능을 보였다.

Adware를 탐지하기 위한 데이터셋 구성은 양성 APK를 포함한 non-adware 카테고리와 adware 카테고리로 나눠진다. 베이스 모델의 생성은

Table 1. Overall dataset composition used for evaluation.

Type	Base dataset	Head dataset	Test against
Without Transfer Learning	{Benign + BankBot + Mecor}	-	{FakeInst + Youmi + Kuguo + Airpush + AdwareCampaign}
With Transfer Learning	{Benign + BankBot + Mecor}	{Youmi + Airpush}	{FakeInst + Kuguo + AdwareCampaign}
Adware Aspect	{Benign + BankBot + Mecor + FakeInst}	{Youmi + Kuguo}	{Airpush + AdwareCampaign}
Adware Aspect with 50% Transferred	{Benign + BankBot + Mecor + FakeInst}	50 % of {Youmi + Kuguo + Airpush + AdwareCampaign}	50 % of {Youmi + Kuguo + Airpush + AdwareCampaign}

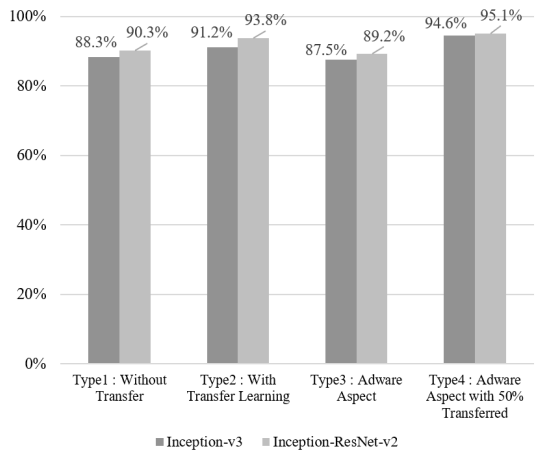

Fig. 7. Accuracy of each type of transfer learning. Each are trained with two different model, Inception-v3 and Inception-ResNet-v2.

adware와 관계없는 카테고리만이 사용되며, 평가는 adware에 대하여 이루어졌으며, 이 때 전이학습을 거치기 전의 성능보다 정확도가 낮아짐을 확인할 수 있었다. 이는 adware 카테고리에 해당하는 APK들에 대한 특징을 모델에 전혀 반영하지 않은 상태로 평가를 진행하기 때문으로, 전이학습 수행 이전 대비 약 0.9% 정도 낮은 정확도를 보인다. 따라서 Table.1의 마지막 타입인 50%의 adware 카테고리의 APK를 전이학습에 사용한 결과로는 가장 높은 성능을 보인다. 이는 adware 카테고리의 APK들에 대한 전반적인 특징을 학습하였기 때문이며, 전이학습 수행 이전 대비 5.6% 높은 정확도를 보인다. 또한 모든 타입에서 Inception-ResNet-v2가 Inception-v3 대비 높은 정확도를 나타내었다.

이러한 과정을 위한 안드로이드 내 리소스는 다음과 같이 분석 가능하다. 전체 과정 중 리소스를 소모하는 주요 부분은 디컴파일 부와 Tensorflow Lite 부분이다. 한편 Tensorflow Lite는 안드로이드 프레임워크에서 기본으로 지원하는 기능이므로 일반적인 수준의 리소스 소모가 발생함을 예상할 수 있다. 반면 디컴파일 과정에서는 시스템에서 의도하지 않은 큰 리소스 소모가 발생하기 때문에 정확한 리소스의 측정이 필요하다.

따라서 디컴파일 과정에서 소모되는 리소스를 각 타입별 카테고리의 APK들에 대해 수행한 결과, Table.2와 같은 결과를 확인하였다. 이때 APK는 각 카테고리에서 테스트에 사용된 APK만이 측정되었다. 또한 디컴파일 과정의 리소스는 커스텀된 ApkTool을 기반으로 측정되었으며, 과정중에서 소모된 총 메모리 및 소요 시간의 평균으로 구성된다.

Table 2. Resource consumption of on-device decompiler

Type	Avg. conusumed memory (kB)	Avg. conusumed time (ms)
1	108,430	16,312
2	99,344	9,088
3	121,145	17,634
4	105,775	15,158

해당 디컴파일 과정은 디바이스 내에서 수행되었으며, 평균적으로 109MB의 메모리 소모와 함께 14.5초가 소요되었다. 이 과정에서 디바이스는 백그라운드에서 비동기 동작으로 디컴파일을 수행하며, 메모리 소모량은 일반적인 어플리케이션의 소모량과 거의 동일한 수준이므로, 사용성에 문제가 없다. 모든 어플리케이션에 대해서는 순차적인 디컴파일이 수행되므로 메모리 소모는 평균적인 수준으로 유지되지만, 시간적인 측면에서는 전체 사용자 앱을 분석하는 과정에서 수십분의 시간이 소요된다. 따라서 실제 적용을 위해서는 포그라운드 서비스 상의 프로세스로 유지된 채 분석을 수행해야 한다.

VII. Conclusion

본 논문에서는 사용자 맞춤형 안드로이드 악성코드 분석을 위하여 Tensorflow Lite를 통해 디바이스 내에서 구동 가능한 분석 시스템을 제안하였다. 또한 전이학습을 위한 시스템을 구현하여 서버로의 통신을 1회만 수행하며 이후 디바이스 내에서 디컴파일과 분석을 모두 수행한다. 평가를 위한 시뮬레이션에서 전이학습을 수행하기 이전 대비 4.8% 높은 95.1%의 정확도로 adware 카테고리의 APK들을 탐지하였다.

이러한 분석 시스템은 실제로 사용자의 디바이스에서의 테스트는 이루어지지 않았다. 이는 사용자 디바이스에 배포되는 APK 카테고리의 객관성이 떨어지기 때문이며, 전반적인 사용자 패턴에 대한 데이터를 수집하는 과정에서 사용자의 사용 패턴과 같은 개인정보 유출이 필연적이기 때문이다. 따라서 이러한 분석 시스템의 실제 배포 데이터를 검증하기 위해서는 별도의 사용 패턴 데이터를 레퍼런스로 하는 실험이 필요하다. 따라서 향후 연구 방향은 실서비스화

이후 사용자 데이터를 기반으로 하는 연구 혹은 유사 연구 공개 이후 해당 연구 데이터를 기반으로 하는 방향이 될 것이다.

References

[1] Forbes, "Many Popular Android Apps Leak Sensitive Data, Leaving Millions Of Consumers At Risk", https://www.forbes.com/sites/ajdellinger/2019/ 06/07/ many- popular-android-apps-leak-sensitive -data-leaving-millions-of-consumers-at-risk/ #69643a7b521e, last accessed Jan 2021.

[2] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," IEEE Transactions on Evolutionary Computation, vol.1, no.1, pp. 67-82, Apr, 1997.

[3] Tensorflow lite, "Deploy machine learning models on mobile and IoT devices", https://www.tensorflow.org/lite, last accessed Feb 2021.

[4] N. Peiravian and X. 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, 2013.

[5] M. Backes, S. Bugiel, O. Schranz, P. Von Styp-Rekowsky and S. Weisgerber, "ARTist: The Android Runtime Instrumentation and Security Toolkit," 2017 IEEE European Symposium on Security and Privacy (EuroS&P), pp.481-495, 2017.

[6] J. Wang, Y. Chen, S. Hao, W. Feng and Z. Shen, "Balanced Distribution Adaptation for Transfer Learning," 2017 IEEE International Conference on Data Mining (ICDM), pp. 1129-1134, 2017.

[7] Tasfia Shermin et al., "Enhanced Transfer Learning with ImageNet Trained Classification Layer," Pacific-Rim Symposium on Image and Video Technology, pp.142-155, 2019.

[8] Jason Yosinski, Jeff Clune, Yoshua Bengio and Hod Lipson, "How transferable are features in deep neural networks?," Advances in Neural Information Processing Systems, pp.3320-3328, 2014.

[9] Ruitao Feng et al., "MobiDroid: A Performance-Sensitive Malware Detection System on Mobile Platform," 2019 24th International Conference on Engineering of Complex Computer Systems (ICECCS), pp.61-70, 2019.

[10] W. Yuan, Y. Jiang, H. Li and M. Cai, "A Lightweight On-Device Detection Method for Android Malware," IEEE Transactions on Systems, Man, and Cybernetics: Systems, pp.1-12, 2019.

[11] Ryszard Wiśniewski, Connor Tumbleson, "APKtool", https://ibotpeaches.github.io/Apktool/install/, 2020, last accessed Jan 2021.

[12] Steve Norum, "Toco", https://pypi.org/project/toco/, last accessed Feb 2021.

[13] Hyunseok Shim and Souhwan Jung, "Semantic-aware Comment Analysis Approach for API Permission Mapping on Android," NLPIR 2020: Proceedings of the 4th International Conference on Natural Language Processing and Information Retrieval, pp.61-69, 2020.

[14] Google Play, "Android distribution", https://developer.android.com/distribute/google-play, last accessed Jan 2021.

[15] Google, "VirusTotal", https://www.virustotal.com/, last accessed Jan 2021.

[16] Wei Fengguo, Li Yuping, Roy Sankardas, Ou Xinming and Zhou Wu, "Deep Ground Truth Analysis of Current Android Malware," International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, pp.252-276, 2017.

[17] Trend Micro, "Adware Campaign Ident

ified From 182 Game and Camera Apps on Google Play and Third-Party Stores Like 9Apps", https://blog.trendmicro.com/trendlabs-security-intelligence/adware-campaign-identified-from-182-game-and-camera-apps-on-google-play-and-third-party-stores-like-9apps/, last accessed Jan 2021.

〈저자소개〉

심 현 석 (Hyunseok Shim) 학생회원
2019년 2월: 숭실대학교 전자정보공학과 졸업
2019년 3월~2021년 2월: 숭실대학교 정보통신융합학과 석사과정
〈관심분야〉 AI 보안, 모바일 보안, 흐름 분석, 개인정보 보호

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〉 AI 보안, 모바일 보안, 클라우드 보안, 네트워크 보안