Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.35, NO.1, Feb. 2025 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2025.35.1.67

시스템 API 호출 순서 정보를 통한 안드로이드 악성 코드 패밀리 분류 기법*

최 재 민,[1†] 최 상 훈,[2] 박 기 웅[3‡]
[1,3]세종대학교 (대학원생, 교수), [2]세종대학교 SysCore Lab. (박사후 연구원)

Android Malware Family Classification Techniques Using System API Call Sequence Data*

Jae-Min Choi,[1†] Sang-Hoon Choi,[2] Ki-Woong Park[3‡]
[1,3]Sejong University (Graduate Student, Professor),
[2]Sejong University SysCore Lab. (Researcher)

요 약

오늘날 모바일 기기는 필수적인 도구로 자리 잡았으며, 보안 중요성도 증가하고 있다. 이에 따라, 안드로이드 시스템에서 API 호출 정보를 활용한 악성코드 탐지 및 분류 방법에 대한 연구가 활발히 진행 중이다. 본 논문에서는 시스템 API 호출 순서 정보를 활용하여 악성코드의 행동 패턴을 분석하고, 유사한 변수명과 기술을 사용하는 악성코드를 대상으로 Dynamic Time Warping (DTW) 알고리즘을 기반으로 악성코드 패밀리를 분류하는 방법을 제안한다. 본 연구는 더 정확한 순서 정보를 활용하여 각 패밀리의 특징을 추출하며, 이를 통해 악성코드 패밀리 분류의 정확도를 높인다. 향후, 많은 데이터셋을 확보하여 시스템 API 호출 순서 정보를 수집하여 머신러닝에 적용하거나 정적 분석과 병행하여 분류를 진행하는 연구를 수행하고자 한다.

ABSTRACT

Today, mobile devices have become ubiquitous and have a large user base. The importance of security for these mobile devices is growing. To this end, classification and detection methods using Android system API call data have been actively researched. Our proposed classification system API call sequence data to analyze the behavior patterns of malicious codes that exhibit similar variable names and techniques. We establish classification criteria based on the DTW algorithm for each family, enabling more accurate classification of malicious code families. In the future, we plan to acquire more datasets to collect system API call sequence data information and apply it to machine learning or conduct research on classification in parallel with static analysis.

Keywords: Android Malware, Malware Classification, Malware Analysis, Malware Detection

Received(10. 04. 2024), Modified(1st: 12. 19. 2024, 2nd: 02. 03. 2025), Accepted(02. 03. 2025)

* 본 논문은 2024년도 한국정보보호학회 하계학술대회에서 발표한 우수논문을 개선 및 확장한 것임.

* 본 연구는 과학기술정보통신부의 재원으로 정보통신기획평가원(IITP)의 실감콘텐츠핵심기술개발(Project No. RS-2023-00228996, 20%), 정보통신방송기술 국제공동연구(ProjectNo. RS-2022-00165794, 20%), 국방ICT융합연구(Project No. 2022-11220701, 20%), 정보통신기획평가원(IITP)의 정보보호핵심원천기술개발(Project No. RS-2024-00438551,20%), 한국연구재단(NRF) 중견후속연구사업(Project No. RS-2023-00208460, 20%)의 지원을 받아 수행된 연구임.

† 주저자, c.jaem7532@gmail.com

‡ 교신저자, woongbak@sejong.ac.kr(Corresponding author)

I. 서 론

IT 기술의 발전에 따라 모바일 기술도 빠르게 진화하고 있으며, 오늘날 모바일 기기는 전 세계적으로 필수적인 도구로 자리 잡았다. 이러한 광범위한 사용자층을 보유한 모바일 기기에서 보안의 중요성은 날로 증가하고 있다.

2004년 'Cabir' 악성코드를 시작으로, 모바일 악성코드는 점점 더 정교해지고 다양한 형태로 발전하고 있다[1]. 현재 모바일 악성코드에는 RAT(Remote Access Trojan), 랜섬웨어, 트로이목마, 애드웨어 등 여러 형태가 존재하며, 구글 플레이스토어에 정상적인 절차를 통해 등록된 안드로이드 APK가 업데이트 과정을 거치면서 악성코드로 변이하는 사례도 있다[2]. 이와 같은 고도화된 악성코드는, 최초에는 안전한 APK로 보이지만 시간이 지나면서 악성 APK로 변하여 사용자의 개인정보를 탈취하는 등 심각한 피해를 초래할 수 있다. 이처럼 모바일 기기를 대상으로 한 악성코드는 PC 환경의 악성코드와 마찬가지로 점점 더 정교해지고 있으며, 이를 효과적으로 탐지하고 분류하기 위한 연구가 활발히 진행되고 있다.

모바일 악성코드가 발전함에 따라 난독화와 암호화 그리고 안티디버깅 기법을 통해 정적 분석과 동적 분석을 어렵게 만들고 있다. 이러한 모바일 악성코드들을 탐지 및 분류하기 위해 시스템 API 호출 횟수를 기반으로 한 연구들이 진행되고 있다[5,6,7,8]. 그러나 해당 방법은 유사한 기술을 사용하는 악성코드들에 대해 탐지 및 분류가 어려울 수 있다는 한계가 있다. 또한, 행동 패턴이 복잡하거나 유사한 호출 횟수를 가진 악성코드의 경우, 정확한 탐지와 분류가 더욱 어렵다는 단점도 존재한다. 우리는 이를 해결하기 위해 본 논문에서 시스템 API 호출 순서 정보를 활용하여 악성코드 패밀리를 분류하는 프레임워크를 제안한다. 이 프레임워크는 보다 정확한 분류 기준을 제시하며, 악성코드 패밀리를 효과적으로 분류할 수 있는 가능성을 보여준다.

본 연구는 2024년 한국정보보호학회 하계학술대회에서 발표된 연구를 확장한 연구이다. 이전 연구의 내용은 가설 증명을 위해 8개의 샘플에 대하여 프레임워크 제작 및 추출한 로그에 대하여 수동 분석을 진행하였고, 이 논문은 실 환경에서 구동하기 위한 전체 자동화 프레임워크 및 수동 분석의 여러 가지 특징들을 자동화 프로세스에 맞추어 개발 및 확장한 내용이다.

본 논문의 구성은 다음과 같다. 2장에서는 안드로이드 시스템과 시스템 API 호출에 대하여 기술하고, 현재 시스템 API 호출을 기반으로 한 연구에 대해 기술한다. 3장에서는 본 논문에서 제안하는 프레임워크에 대한 설계 및 시스템 API 호출 추출, 전처리, 분류, 분류 기준 수립에 대하여 기술한다. 4장에서는 확보한 악성코드 샘플을 이용하여 프레임워크를 통한 분류 결과를 기술하고, 테스트 샘플을 이용하여 수립한 분류 기준에 대하여 검증을 진행한다. 5장에서는 결론과 향후 연구를 기술한다.

II. 배경지식 및 관련 연구

2.1 안드로이드 시스템 API 배경지식

2.1.1 안드로이드 시스템 구조

안드로이드 시스템은 Linux 커널을 기반으로 다양한 안드로이드 서비스와 Daemon이 동작하는 구조로 이루어져 있다. Linux 커널은 프로세스 관리, 메모리 관리, 네트워크 스택, 보안 관리 등의 핵심 기능을 제공하며, SELinux를 통해 시스템 보안을 강화한다. 해당 커널 위에서 Android Runtime (ART)가 동작하여 애플리케이션의 바이트코드를 프로세서 명령어로 변환하여 실행한다[3]. ART는 Ahead-Of-Time (AOT)과 Just-In-Time (JIT) 컴파일을 통해 성능을 최적화하며, 앱 실행 성능을 높인다. 이때, 앱의 바이트코드를 실행하며 Dalvik Executable(dex) 파일이 ART에 의해 처리되고, 애플리케이션이 시스템 자원에 접근할 때 시스템 API 호출이 발생한다.

이 API 호출은 Linux 커널을 통해 처리되어 메모리, 파일 시스템, 네트워크와 같은 시스템 자원에 접근할 수 있도록 한다. 또한, 안드로이드 시스템의 중요한 프로세스인 Zygote는 모든 애플리케이션의 부모 프로세스로 작동하여 앱 실행 시간을 단축시킨다. 이와 같은 구조는 안드로이드 시스템이 다양한 애플리케이션을 원활하게 실행하고, 자원을 효율적으로 관리할 수 있도록 돕는다.

2.1.2 Android Debug Bridge

Android Debug Bridge(ADB)는 안드로이드에서 제공하는 다목적 명령줄 도구로 안드로이드 기기에

앱 설치 및 디버깅과 같은 기기 작업을 가능하게 하는 Client-Server 프로그램이다[4]. 이를 이용해 안드로이드 기기에 Unix shell 명령을 전송할 수 있고, 안드로이드 기기로 파일을 업로드, 설치 그리고 기기 내의 파일을 Server 측으로 다운로드 할 수 있다.

2.2 시스템 API 호출 기반 연구

2.2.1 시스템 API 호출 기반 악성코드 탐지 연구

Taniya Bhatia는 악성코드 시스템 API 호출 횟수 분석을 통한 악성코드 탐지방안을 제안하였다[5]. 해당 연구는 APK가 실행된 뒤 monkey 도구를 이용하여 1분간 APK가 발생시킨 시스템 API 호출 횟수를 추출하여 악성 APK와 정상 APK를 분류할 수 있음을 보여주었다. 해당 연구에서는 난독화와 암호화가 적용된 악성 APK를 효과적으로 분류하였다.

M. Jaiswal 외 2인은 악성 게임 APK의 시스템 API 호출 횟수 분석을 통한 악성코드 탐지방안을 제안하였다[6]. 해당 연구는 정상 APK와 복제된 악성 APK가 발생하는 시스템 API 호출 횟수를 추출하여 정상 APK와 악성 APK를 분류할 수 있음을 보여주었다. 해당 연구에서는 정적 분석 없이 복제 및 재패키지 악성 APK를 효과적으로 분류하였다.

2.2.2 시스템 API 호출 기반 악성 코드 분류 연구

S. Mahdavifar 외 4인은 악성 코드 시스템 API 호출을 기반으로 5가지의 카테고리를 분류하는 분류방안을 제안하였다[7]. 해당 연구는 CICMalDroid2020 이라는 데이터셋을 개발하여 Adware, Banking, SMS, Riskware, Benign 5가지로 분류하였다. 해당 데이터셋은 정적 및 동적 특징을 포함한 데이터셋으로 17,341개의 샘플이 포함되어 있다. 시스템 API 호출과 기본 Binder, 행동 등 여러 동적 특징을 추출하여 이를 딥러닝 모델에 학습하여 분류하였다. 해당 연구에서 제안한 모델은 F1-Score 97.84%로 높은 성능을 보여주었고, 비교적 적은 양의 라벨링된 데이터를 통해 효과적으로 분류하였다.

A, Moutaz 외 4인은 악성 코드 시스템 API 호출과 XML 파일 내의 권한 요청 항목을 기반으로 안드로이드 악성 코드를 Ambiguous, Risky, Disruptive 3가지 그룹으로 나누어 분류하는 방안을 제안하였다[8]. 해당 연구에서는 시스템 API 호출과 XML 파일 내의 권한 요청 항목을 각 동적, 정적 분석을 통하여 추출한 뒤 두 가지 항목을 융합한 하이브리드 분석을 제안하였다. 해당 특징을 RF, J48, RT, K-NN, NB 알고리즘을 이용하여 머신러닝 모델에 학습하였고 최종적으로 94.3%의 F1-Score를 달성하였다.

2.3 Dynamic Time Warping 알고리즘

2.3.1 Dynamic Time Warping 알고리즘 정의

Dynamic Time Warping(DTW)는 속도 혹은 길이에 따라 움직임이 다른 두 시계열 간의 유사성을 측정하는 알고리즘이다. DTW 알고리즘은 거리가 최소화되는 방향으로 매칭시켜 누적 거리가 최소가 되는 Warping 경로를 찾아 유사성을 비교한다. DTW 알고리즘은 주로 그래픽, 비디오, 오디오 분야에서 사용되고 있다.

예를 들어, 길이가 상이한 n, m인 두 시계열 데이터를 이용하여 유사성을 비교할 때, 두 시계열 데이터의 거리를 기존 Euclidean 거리를 통해 유사성을 측정하는 것은 어려움이 있다. 하지만, DTW 알고리즘은 Euclidean 거리와 다르게 1:1 매칭이 아닌 1:n 매칭을 통하여 거리를 찾고, 1:n 매칭 시 발생하는 Warping(시간적 뒤틀림) 거리를 이용하여 Warping 거리들의 최소가 되는 경로를 찾아낸다. 해당 Warping 거리가 0에 가까울수록 두 시계열 데이터가 유사하다고 판별한다.

2.3.2 Dynamic Time Warping 알고리즘 선정 이유

DTW 외에도 시계열 데이터에 대한 유사도를 비교할 수 있는 여러 가지 알고리즘이 존재한다. 본 연구에서 추출한 데이터를 Cosine과 SBD 알고리즘을 활용하여 유사도를 비교하고 히트맵으로 추출하면 Fig. 1.과 같다.

우선 SBD 알고리즘을 적용하였을 때 대부분의 데이터에 대하여 0에 가깝게 표현되고 있다. 즉, 유사하지 않음을 의미한다. 하지만 해당 데이터에는 동일한 패밀리가 포함되어있고, 그림에서 19, 20번 2개의 샘플에 대하여서만 동일하다고 표현한다. 해당 알고리즘을 통하여 올바르게 분류가 이뤄지지 않음을 확인했다.

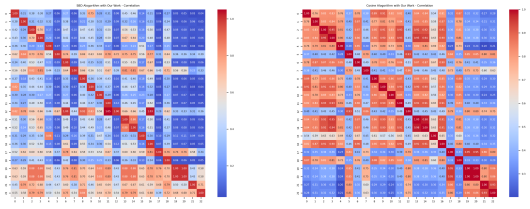

Fig. 1. SBD and Cosine Algorithm Heatmap

Cosine 알고리즘은 거리가 동일해야 한다는 문제점이 있다. 이를 위해 데이터 길이가 다른 시계열 데이터에 대하여 '0'패딩을 통하여 길이를 동일하게 맞춰주었다. 이후 추출된 데이터를 SBD와 비교하였을 때 비교적 나은 성능을 보여주고 있다. 하지만, 데이터의 19~20번 샘플에 대하여 두 개의 샘플은 동일하지만 21, 22번 샘플과는 약 0.8~0.86의 유사도를 보여준다. 하지만 해당 샘플은 모두 동일한 패밀리로 길이에 차이가 있는 샘플이다. 이는 '0'패딩을 통해 길이는 맞추었지만, 약간의 길이 차이로 인해 유사도 평가의 결과가 상이함을 확인하였다.

따라서, 본 연구에서는 Cosine, SBD 알고리즘이 아닌 DTW로 진행하였다.

2.4 기존 연구와의 차이점

기존의 연구에서는 Android Virtual Device를 이용하여 악성코드 분석을 진행하는 경우가 다수이다. 이로인해 Android Virtual Device를 탐지하여 우회하는 악성 코드에 대하여 대응이 어렵지만, 본 연구에서는 호스트에서 모든 행위를 수행하도록 설계되어 있어 QEMU/KVM 환경이 아닌 USB Debug를 이용한 스마트폰 환경에서도 분석이 가능하다.

기존 시스템 API 호출을 기반으로 한 연구들의 경우 동일한 패키지명, 변수명을 사용하거나 동일한 공격 기법을 사용하는 경우 정적 분석을 통하여 분석하였을 때 동일한 패밀리로 오분류를 할 가능성이 있다. 또한, 동일한 공격 기법을 사용하는 경우 유사한 API 호출 횟수를 갖고 있어 오분류를 할 가능성이 있다. 우리 연구에서는 이러한 악성 코드에 대하여 보다 정확한 분류를 위해 순서 정보를 활용한다.

순서 정보를 활용함으로써 각 패밀리별로 라이브러리를 불러오는 순서, 변수를 할당하는 순서, 공격자 서버에 연결하는 순서, 랜섬웨어라면 암호화를 하는 순서가 모두 다르게 나타난다. 이를 이용하여 재패키징을 한 악성 코드, 한 패밀리에서 제작된 악성 코드를 리팩토링을 한 경우에도 순서 정보를 통해 올바르게 분류가 가능함을 기대할 수 있다.

기존의 순서 정보 기반 탐지 및 분류 연구 대부분은 strace로 추출한 모든 데이터를 머신러닝 혹은 딥러닝을 이용하여 학습을 진행하고 있다. 따라서, 학습에 소요되는 시간과 여러 악성코드들이 생성하는 데이터들의 양이 상당히 크다. 이에 따라, 학습과 분류 기준을 세우는 데에 있어 상당한 비용이 소요된다.

본 연구에서 제작 및 개발한 프레임워크는 기존의 strace로 추출한 모든 데이터에 대하여 학습 및 분류 기준을 세우는 것이 아닌 악성 APK의 카테고리별 사용 시스템 콜과 각 시스템 콜 요청의 매개변수를 기반으로 필터링한다. 추출된 데이터 크기는 Table 1과 같다. 모든 데이터에 대하여 최소 약 29%에서 최대 540.54%까지 데이터 크기가 감소하고, 동일한 DTW 알고리즘에 적용하였을 때 거리 계산 결과까지

Table 1. Reduce Data Size with Processing

Before Processing	After Processing
2692 Kb	40 Kb
1088 Kb	30 Kb
1439 Kb	35 Kb
1334 Kb	34 Kb
15564 Kb	42 Kb
599 Kb	3 Kb
2676 Kb	28 Kb
196 Kb	5 Kb
2311 Kb	37 Kb
1843 Kb	36 Kb
1990 Kb	37 Kb
2628 Kb	33 Kb
537 Kb	4 Kb
1321 Kb	34 Kb
1512 Kb	34 Kb
14054 Kb	26 Kb
2568 Kb	32 Kb
1305 Kb	3 Kb
9079 Kb	40 Kb
1344 Kb	3 Kb
1454 Kb	3 Kb
87 Kb	3 Kb
86 Kb	2 Kb

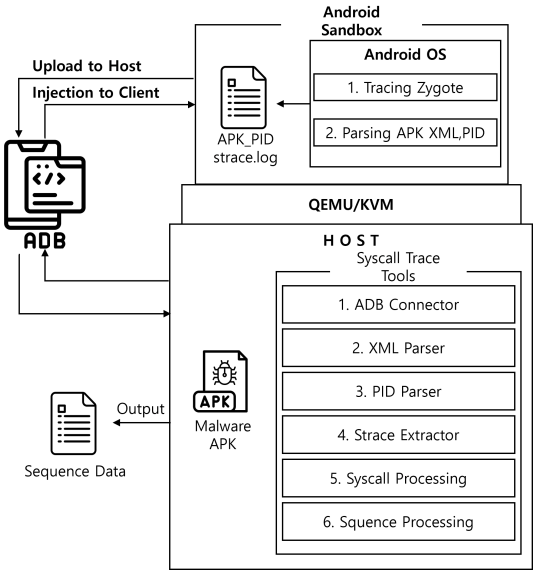

Fig. 2. Android System API Call Extraction Framework

소요되는 시간이 31,561.4 s에서 6.3 s으로 감소하였다. 이를 통하여 본 연구에서 제작 및 개발한 프레임워크를 통하여 학습 및 분류 기준을 세우는 데에 있어 비교적 저렴한 비용으로 학습 및 분류 기준을 세울 수 있다.

III. 시스템 API 호출 순서 정보를 통한 분류 프레임워크

본 장에서는 시스템 API 호출 순서 정보를 통한 악성코드 패밀리 분류를 위한 프레임워크와 프레임워크를 구성하는 각 모듈에 대하여 상세하게 기술한다. 프레임워크의 구조는 Fig. 2.과 같다.

해당 프레임워크내에서 데이터는 Fig. 3.와 같이 동작한다. 우선, APK에서 실행에 대한 전체 시스템 API 호출 정보를 추출하고 해당 데이터를 순서 정보로 가공한다. 이후 DTW 알고리즘을 이용하여 기준을 수립하고 해당 기준에 맞추어 분류한다.

3.1 시스템 API 호출 순서 정보 추출

안드로이드 시스템 API 호출 순서 정보를 추출하기 위해 QEMU/KVM 기반의 분석 환경을 구축하였다. 해당 환경에서 동작하는 안드로이드 OS 내에서 strace 도구를 활용하여 시스템 API 호출 정보를 추

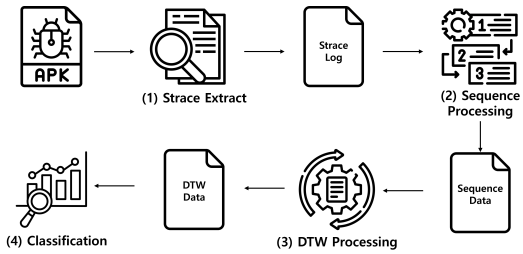

Fig. 3. System API Call Data Progress

출하였다. 이때, 안드로이드 OS 내에 APK 업로드, 설치, 실행을 위해서 안드로이드에서 제공하는 ADB를 이용하여 제어하였다. ADB는 리눅스 시스템에서 shell 명령어를 통하여 실행시킬 수 있고, 이를 기반으로 우리는 QEMU/KVM 실행, ADB 연동, ADB root 권한 상승, APK 업로드, APK 설치, strace 실행, strace 결과 데이터 추출 순서로 동작하는 Python 코드를 제작하였다.

시스템 API 호출 순서 정보를 통해 악성 코드 분류를 위해 AndroZoo[9]에서 제공하는 샘플을 다운로드 하였다. 해당 AndroZoo 샘플에서 우리는 카테고리 중 Trojan과 FakeInstall 두 개의 카테고리를 대상으로 샘플을 선정하였다. 선정한 두 개의 카테고리에서 Virus Total[10] 탐지 점수가 30 이상의 샘플을 최종 샘플로 결정하였다. 해당 샘플들은 안드로이드 버전, VM 환경에서의 문제로 인하여 실행이 불가한 샘플들이 다수있고, 이를 위해 안드로이드 버전 7과 9으로 실험환경을 추가 구축하여 총 135개의 샘플의 시스템 API를 호출하여 실험을 진행하였다.

해당 샘플에서 시스템 API 호출 정보를 올바르게 추출하기 위해 안드로이드 기초 프로세스인 'Zygote'를 대상으로 strace 도구를 실행한 뒤 해당 프로세스에서 생성된 분석 대상 프로세스의 패키지명을 pidof 명령어로 추출하여 저장하였다. 이후 저장된 패키지명을 기반으로 clone 시스템 호출을 통해 생성된 자식 프로세스 pid를 대상으로 추출을 진행한다. 해당 과정을 통하여 Fig. 4.과 같이 올바르게 대상 APK

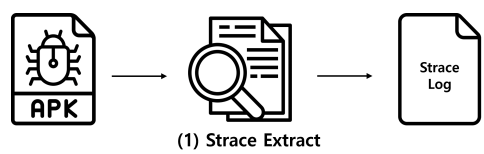

Fig. 4. Strace Extract and Process

strace log를 추출할 수 있다.

3.2 시스템 API 호출 순서 정보 전처리

strace를 통해 추출한 시스템 API 호출 정보에는 많은 노이즈가 발생한다. 안드로이드 화면 내에서 사용자가 마우스를 움직이면 실시간 좌표를 pread64 함수와 write 함수를 이용하여 통신하기 때문에 순식간에 많은 시스템 API 호출이 발생한다.

또한, 사용자가 버튼을 클릭하면 pread64 함수와 write 함수, ioctl 함수 그리고 recvfrom 함수를 이용하여 return 값 232가 발생하고, 뒤로가기 버튼을 이용하면 return 값 72인 시스템 API 호출이 발생한다. 해당 값들은 우리가 목표로하는 악성 행위의 시스템 API 호출 정보가 아니기에 이러한 노이즈를 제거하는 과정이 필요하다. 우리는 노이즈를 제거하기 위해 strace 도구 내에서 지원하는 정규표현식 기능을 이용하여 Table 2. 에 기술된 시스템 API 호출 함수를 필터링하여 추출하였다.

이후 추출된 시스템 API 호출 정보에는 우리가 원하는 File I/O, Process Contorl, Socket Connect와 관련된 시스템 API 호출 함수만이 포함되어있다. 시스템 API 호출은 발생할 때 10ms 내에서 많은 시스템 API 호출이 복잡하게 실행되는 구조로 이루어져 있다. 따라서, 하나의 시스템 API 호출이 실행될 때 해당 호출이 모두 끝나기 전에 1ms 내에서 다른 호출이 발생하는 경우도 있고 이러한 경우는 strace 도구에서 〈continued〉 와 〈unfinished〉 의 매개변수를 포함하여 추출된다. 해당 값은 우리가 필요로 하는 순서 정보에서는 노이즈이기 때문에 제거하는 과정이 필요하다.

우리는 이를 위해 각 카테고리별 분석을 통해 특정 시스템 함수를 대상으로 매개변수를 선정하였다.

Table 2. Filtering System API Call Function List

Category	System API Call Function
Process Control	clone, execve, fork, getuid, getuid32, geteuid, geteuid32
File I/O and Socket Connect	accept, bind, connect, getsockopt, mkdir, mkdirat, open, openat, pread64, read, readlinkat, recv, recvfrom, recvmsg, rename, renameat, rmdir, send, sendto, sendmsg, setsockopt, socket, stat, unlink, unlinkat, vfor k,write, writev

APK는 실행될 때 내부 저장소와 APK 자신이 가지고 있는 바이트코드와 그림, 내부 데이터를 자신 패키지명으로 된 폴더에서 관리하기 때문에 내부 저장소에 접근하는 시스템 API 호출이 많이 발생한다. Trojan 카테고리와 FakeInstall 계열은 내부 저장소의 데이터에 접근, 저장하는 행위를 하고 있다. 이처럼 동일한 시스템 함수지만 악성과 정상 행위를 위해 모두 사용되고 있고, 이를 구분 짓기 위해 특정 매개변수만을 선정하여 추출하였다. 매개변수가 순서 정보에 큰 영향을 끼치지 않는 함수의 경우 매개변수 여부와 상관없이 추출하였다. 또한, 안드로이드 버전에 따라 매개변수의 내용이 변화한다. 안드로이드 7 버전의 경우 Activity를 실행할 때 'LAUNCH_ACTIVITY' 매개변수를 사용하고, 9 버전의 경우 'performOnCreate' 매개변수를 사용한다. 이외의 시스템 API 호출 함수의 매개변수는 변하지 않는다. 최종적으로 버전 7, 9 두 개의 버전에 대하여 'writev' 함수의 매개변수를 추가로 추출하였다. 선정한 시스템 API 호출 함수별 매개변수는 Table 3.와 같다.

최종적으로 전처리 단계에서 노이즈를 제거하는 과정을 진행한다. 노이즈에 포함되는 데이터에는 악성 행위가 아닌 시스템 API 호출 정보이고, 해당 데이터는 사람이 분석하여 선정한 시스템 API 호출 함

Table 3. Filtering System API Call Parameter List

Function	System API Call Parameter
openat()	base.apk, /storage/emulated/0, /data/user/0, /sdcard
Version 7 writev()	onCreate, onDestroy, onResume, LAUNCH_ACTIVITY, handlePauseActivity, destroy, handleRelaunchActivity, performPause
Version 9 writev()	MAIN_ACTIVITY, performCreate, performPause, performRestartActivity, performDestroy, handleStartActivity, RESUME_ACITIVTY, apache
write()	〈?xml, getaddrinfo, Timeout, http, telnet
clone()	!zygote
mkdirat()	/storage/emulated/0, /data/user/0, /sdcard
renameat()	/storage/emulated/0, /data/user/0, /sdcard
connect()	AF_INET, AF_INET6

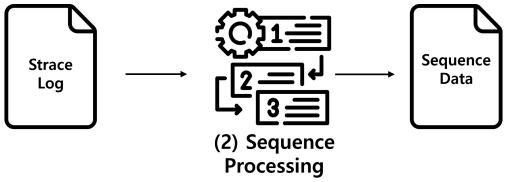

Fig. 5. Sequence Data Process

수와 매개변수를 기반으로 정한다. 해당 과정을 통하여 Fig. 5.와 같이 추출된 Strace Log를 Sequence Data로 변경한다.

Table 4. System API Call Function to Integer for DTW Algorithm

System API Call Fucntion	Integer	System API Call Function	Integer
clone()	1	recv()	19
execve()	2	recvfrom()	20
fork()	3	recvmsg()	21
getuid()	4	rename()	22
getuid32()	5	renameat()	23
accept()	6	rmdir()	24
geteuid()	7	send()	25
geteuid32()	8	sendto()	26
accept()	9	sendmsg()	27
connect()	10	setsockopt()	28
getsockopt()	11	socket()	29
mkdir()	12	stat()	30
mkdirat()	13	unlink()	31
open()	14	unlinkat()	32
openat()	15	vfork()	33
pread64()	16	write()	34
read()	17	writev()	35
readlinkat()	18		

3.3 시스템 API 호출 순서 정보 분류

분류를 진행함에 있어, 해당 악성코드가 올바르게 동작하지 않았을 가능성도 존재한다. 이러한 데이터가 DTW 알고리즘에 적용되면 노이즈가 발생한다. 따라서, Code Coverage를 측정하기위해 공개된 도구들을 사용해보았지만, 악성코드 특성상 빌드시 SDK 버전을 완벽하게 추출할 수 없고, 이로인하여 자동화된 Code Coverage 추출 도구를 활용함에 있어 어려움이 발생하였다. 따라서, 이를 해결하고자 전처리 후 결과 파일의 크기가 1024KB가 넘지않는 샘플들을 제거하였다.

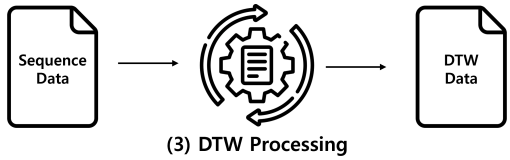

Fig. 6. DTW Data Process

카테고리에 따라 악성 행위가 다르기 때문에 순서 정보가 상이하다. 따라서, 카테고리 기준 순서 정보 분석을 통해 카테고리별 악성 행위를 분류해야 한다. Trojan과 Fakeinstall 두 개의 카테고리를 대상으로 진행하였고, 이후 패밀리별로 순서 정보를 분류한다. 이때, 같은 패밀리 내에서 다른 순서 정보 양상을 보여주는 경우가 존재하는데, 이는 동일 패밀리에서도 팀이 나뉘어져 있는 경우가 존재한다[11]. 이러한 경우 같은 패밀리로 판단하여 한 번 더 소분류를 진행한다. 추출된 데이터를 DTW 알고리즘을 사용하여 DTW 거리 값으로 나타낸다. 각각의 시스템 API 호출 함수는 DTW 알고리즘에 사용하기 위하여 Table 4.과 같이 정수 형태로 변환하여 진행한다.

3.4 시스템 API 호출 순서 정보 분류 기준 수립

최종적으로 Fig. 6.에서 추출된 DTW 거리를 기반으로 분류 기준을 수립한다. 각 패밀리 별 최솟값의 최솟값으로 진행한다. 미탐을 제하고자 위와 같이 기준을 수립하였다. 이때, 어떠한 패밀리와도 매칭이 안 된다면 해당 패밀리는 새로운 패밀리로 분류한다.

IV. 분류 결과 및 검증

본 장에서는 시스템 API 호출 순서 정보를 통한 분류 프레임워크를 통해 분류한 샘플에 대하여 분류 결과 및 검증 결과에 대하여 기술한다.

4.1 분류 결과

본 논문에서 제안하는 프레임워크에 대한 실험 환경은 Table 5.와 같다. 우리는 다음과 같은 가설을 토대로 분류를 진행하였다. “동일한 카테고리 내에서 동일한 패밀리는 같은 순서 정보 양상을 보일 것이

Table 5. An Environment for Framework

Environment	Environment Spec
Host OS	Ubuntu 64bit 22.04
Guest OS	Android-x86-9.0
CPU	Intel(R) Core i7-13700KF
Memory	DDR5 32GB

다. 이를 통하여 악성 코드 패밀리 분류가 가능할 것이다." 해당 가설을 기반으로 실험과 분류를 진행하였다. 해당 가설을 검증하기 위해 본 논문에서는 AndroZoo를 통해 획득한 123개의 샘플(2종의 카테고리, 21개의 패밀리)와 AnMtyh Rat 생성기를 통해 생성한 12개의 샘플을 대상으로 분석을 진행하였다. 샘플별로 추출한 악성 행위를 진행하는 시스템 API 호출 함수는 Table 6.와 같다. Table 6.에는 표기를 위해 각 시스템 API 호출 함수명을 간략하게 표기하였다. openat-op, mkdirat-mk, connect-co, writev-wv, write-w, sendto-st 등. 또한, 동일한 구문이 연속되어 발생하는 경우 #n으로 표기하였다. 모든 샘플을 기술하기에 어려움이 있어 각 패밀리 별 2종의 샘플을 기술하였다. Table 6.에서 각 카테고리 내에서 다른 패밀리가 시스템 API 호출 순서 정보 양상이 다르다는 것을 확인할 수 있다. ddlight와 goldrem 패밀리의 경우 구글의 gstatic에 접근하는 모습을 보이는데 두 개의 패밀리는 wv(APACHE_CONNECT)의 유무가 눈에 띄게 들어온다. 이처럼 동일한 공격 기법을 사용하는 경우 유사한 순서 정보가 나타나고 이를 통해 분류가 가능함을 확인하였다.

순서 정보를 DTW 알고리즘에 적용하는 과정은 Fig. 7.과 같다. 우선, 추출한 순서 정보를 txt 파일로 저장하여 입력으로 받는다. 이후 해당 순서 정보를 DTW 알고리즘에 적용하기 위하여 시스템 API 호출 함수만을 추출한다. 이후 해당 호출 함수를 정수로 변환하여 DTW 알고리즘에 사용한다. DTW 알고리즘에서는 두 개의 입력값을 받고 두 입력값의 거리를 비교한다. 따라서, 순서 정보를 리스트에 저장하고 1번째부터 1,2 … n 까지 거리를 구

Table 6. System API Call Sequence Data by Malicious Behavior

Category	Family	SHA256	System call Sequence
FakeInst	rufraud	03d1 … 8702	mk〉mk〉co〉co〉wv(Create)〉wv(handleStartActivity)〉wv(ResumeActivity)〉st〉op〉wv(Pause)〉wv(RestartActivity)〉wv(handleStartActivity)〉wv(ResumeActivity)
FakeInst	rufraud	063f … b73a	mk〉mk〉co〉co〉wv(Create)〉wv(handleStartActivity)〉wv(ResumeActivity)〉st〉op〉wv(Pause)〉wv(RestartActivity)〉wv(handleStartActivity)〉wv(ResumeActivity)
FakeInst	opfake	02c9 … 9c9f	mk#4〉w〉op#7〉w〉wv(APACHE_CONNECT)#7〉op#10〉wv(Create)〉wv(handleStartActivity)〉wv(ResumeActivity)〉mk
FakeInst	opfake	053a … e357	mk#4〉w〉op#7〉w〉wv(APACHE_CONNECT)#7〉op#10〉wv(Create)〉wv(handleStartActivity)〉wv(ResumeActivity)〉mk
Trojan	ddlight	0a7f … ec48	mk〉mk〉mk〉mk〉wv(Create)〉wv(handleStartActivity)〉wv(ResumeActivity)〉mk〉mk〉co〉co〉w〉co〉w〉w
Trojan	ddlight	246b … a615	mk〉mk〉mk〉mk〉wv(Create)〉wv(handleStartActivity)〉wv(ResumeActivity)〉mk〉mk〉co〉co〉w〉co〉w〉w
Trojan	glodream	141a … 0298	mk#4〉w〉wv(Create)〉wv(handleStartActivity)〉wv(ResumeActivity)〉w#10〉mk〉co〉co〉w〉co〉w
Trojan	glodream	16a0 … 7649	mk#4〉w〉wv(Create)〉wv(handleStartActivity)〉wv(ResumeActivity)〉w#10〉mk〉co〉co〉w〉co〉w
Trojan	lotoor	04c5 … 80d1	mk〉w〉w〉wv(Create)〉wv(handleStartActivity)〉wv(ResumeActivity)〉w〉co〉co〉st〉co〉co〉w〉co〉co〉st〉w〉co〉co〉st〉co#10〉st#5
Trojan	lotoor	201f … bb06	mk〉w〉w〉wv(Create)〉wv(handleStartActivity)〉wv(ResumeActivity)〉w〉co〉co〉st〉co〉co〉w〉co〉co〉st〉w〉co〉co〉st〉co#10〉st#5
RAT	AnMyth	951b … 75f7	mk〉wv(Create)〉wv(handleStartActivity)〉wv(ResumeActivity)〉wv(Pause)〉co〉co〉st〉co〉co〉st〉st
RAT	AnMyth	99d8 … 312d	mk〉wv(Create)〉wv(handleStartActivity)〉wv(ResumeActivity)〉wv(Pause)〉co〉co〉st〉co〉co〉st〉st

Table 7. System API Call Sequence Data to DTW Algorithm

Family	adrd	*au	*bb	*dd	*fd	*gm	*gd	*ic	*im	*ks	*lo	*nd	*of	*pj1	*pj2	*ra	*ru	*uu1	*uu2	*uu3	*wi	*ah
adrd	0	79.6	34.7	67.5	111.1	109.8	114.1	104.1	64.8	102.1	112.3	44.1	84.0	90.2	56.0	115.2	24.9	63.3	83.9	19.3	46.7	29.7
*au	79.6	0	68.6	42.4	94.4	54.4	61.1	58.5	41.6	58.2	57.2	56.3	50.9	46.9	46.1	51.4	56.4	43.4	48.9	84.1	48.9	62.9
*bb	34.7	68.6	0	57.7	111.4	91.5	101.0	100.8	69.7	94.0	96.9	74.4	80.8	82.5	42.8	103.4	33.5	60.9	77.1	69.3	38.7	77.6
*dd	67.5	42.4	57.7	0	79.2	55.2	62.8	51.4	42.0	66.4	54.6	57.6	44.1	48.0	42.2	52.2	51.5	42.7	36.5	71.3	45.6	66.8
*fd	111.1	94.4	111.4	79.2	0	73.0	77.6	71.3	114.9	49.3	69.5	148.4	81.0	71.5	97.2	105.1	108.7	111.2	76.0	112.8	103.5	144.5
*gm	109.8	54.4	91.5	55.2	73.0	0	47.9	57.8	64.3	61.0	45.7	86.6	59.2	45.8	72.9	59.8	80.8	66.5	50.8	119.5	73.5	100.3
*gd	114.1	61.1	101.0	62.8	77.6	47.9	0	59.3	63.5	66.4	53.4	85.9	58.3	52.6	75.8	62.6	84.9	71.5	63.2	121.2	78.3	103.9
*ic	104.1	58.5	100.8	51.4	71.3	57.8	59.3	0	60.8	68.0	51.0	84.7	55.1	47.8	65.4	57.9	79.0	56.5	50.4	110.0	69.7	88.4
*im	64.8	41.6	69.7	42.0	114.9	64.3	63.5	60.8	0	68.8	58.8	37.3	41.0	52.2	31.6	47.0	42.4	34.2	60.5	64.6	31.6	45.4
*ks	102.1	58.2	94.0	66.4	49.3	61.0	66.4	68.0	68.8	0	57.7	75.4	57.2	59.0	71.9	71.3	79.0	60.7	57.8	109.2	74.2	78.9
*lo	112.3	57.2	96.9	54.6	69.5	45.7	53.4	51.0	58.8	57.7	0	77.7	52.2	47.1	70.8	57.0	82.4	54.9	49.3	116.6	71.0	82.7
*nd	44.1	56.3	74.4	57.6	148.4	86.6	85.9	84.7	37.3	75.4	77.7	0	54.6	77.8	59.2	59.0	39.9	47.4	84.2	46.5	30.5	34.0
*of	84.0	50.9	80.8	44.1	81.0	59.2	58.3	55.1	41.0	57.2	52.2	54.6	0	52.3	48.6	55.5	57.8	40.9	46.5	87.2	46.8	65.1
*pj1	90.2	46.9	82.5	48.0	71.5	45.8	52.6	47.8	52.2	59.0	47.1	77.8	52.3	0	57.9	52.8	72.3	50.7	44.2	97.5	58.7	83.2
*pj2	56.0	46.1	42.8	42.2	97.2	72.9	75.8	65.4	31.6	71.9	70.8	59.2	48.6	57.9	0	58.2	49.4	35.2	49.7	55.4	27.1	60.8
*ra	115.2	51.4	103.4	52.2	105.1	59.8	62.6	57.9	47.0	71.3	57.0	59.0	55.5	52.8	58.2	0	63.6	47.5	44.7	126.4	51.2	62.2
*ru	24.9	56.4	33.5	51.5	108.7	80.8	84.9	79.0	42.4	79.0	82.4	39.9	57.8	72.3	49.4	63.6	0	40.6	74.0	31.4	38.8	24.8
*uu1	63.3	43.4	60.9	42.7	111.2	66.5	71.5	56.5	34.2	60.7	54.9	47.4	40.9	50.7	35.2	47.5	40.6	0	45.7	67.1	26.8	45.6
*uu2	83.9	48.9	77.1	36.5	76.0	50.8	63.2	50.4	60.5	57.8	49.3	84.2	46.5	44.2	49.7	44.7	74.0	45.7	0	89.8	58.4	95.1
*uu3	19.3	84.1	69.3	71.3	112.8	119.5	121.2	110.0	64.6	109.2	116.6	46.5	87.2	97.5	55.4	126.4	31.4	67.1	89.8	0	47.8	22.5
*wi	46.7	48.9	38.7	45.6	103.5	73.5	78.3	69.7	31.6	74.2	71.0	30.5	46.8	58.7	27.1	51.2	38.8	26.8	58.4	47.8	0	37.4
*ah	29.7	62.9	77.6	66.8	144.5	100.3	103.9	88.4	45.4	78.9	82.7	34.0	65.1	83.2	60.8	62.2	24.8	45.6	95.1	22.5	37.4	0

*au:andup,*bb:basebirdge,*dd:ddlight,*fd:fakedoc,*gm:gingermaster,*ic:iconosys,*im:imlog,*ks:ksapp,*lo,lotoor,*nd:nandrobox,*of:opfake,*pj1:pjapps-1,*pj2:pjapps-2,*ra:ramnit,*ru"rufraud,*uu1:uuserv-1,*uu2:uuserv-2,*uu3:uuserv-3,*wi:winge.*ah:anmyth

Algorithm 1 System Call Mapping and DTW Algorithm

Input: Two sequences of system calls $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_m)$
Output: DTW distance $D(n, m)$
Define System call mapping function $f_{syscall}(s)$
for $i = 1$ to n **do**
    Map System call x_i to integer: $x_i \leftarrow f_{syscall}(x_i)$
end for
for $j = 1$ to m **do**
    Map System call y_j to integer: $y_j \leftarrow f_{syscall}(y_j)$
end for
Initialize matrix D of size $(n+1) \times (m+1)$ with $D(0,0) = 0$
for $i = 1$ to n **do**
    $D(i, 0) = \infty$
end for
for $j = 1$ to m **do**
    $D(0, j) = \infty$
end for
for $i = 1$ to n **do**
    for $j = 1$ to m **do**
        $cost = |x_i - y_j|$
        $D(i, j) = cost + \min(D(i-1, j), D(i, j-1), D(i-1, j-1))$
    end for
end for
return $D(n, m)$

Fig. 7. System Call Mapping to DTW Algorithm

한다. 이후 해당 계산을 반복하여 n*n의 거리 행렬을 추출한다. 추출한 행렬을 최종적으로 csv로 추출하여 분류를 진행한다. 추출한 csv 결과는 Table 7.과 같다.

4.2 분류 성능 검증

우리가 분류한 분류 기준에 올바르게 악성 코드가 분류됨을 확인하기 위해 *AndroZoo*를 통해 획득한 샘플을 이용하여 검증을 진행하였다. 검증 결과는 Table 8.과 같다. Table 8.를 보면 각각 *bb,l *fd, adrd, *ru, *nd 에서 DTW 거리가 0혹은 거리값 최소가 나오는 모습을 볼 수 있다. 이는 각 샘플이 해당 패밀리에 속함을 의미한다. 이는 *Virus Total* 분석 결과와 *AndroZoo*에서 제공하는 샘플 라벨을 통해 패밀리가 동일함을 확인하였다. 해당 데이터의 순서 정보를 모두 적기에 어려움이 있어 생략한다. 각 데이터는 기준이 되는 순서 정보와 1~2개의 순서 정보가 차이가 발생하였다. 하지만 DTW 알고리즘을 적용한 결과 동일한 기준이 되는 데이터와 거리가 0혹은 최소값으로 계산되었고 이는 DTW 알고리즘의 시간축 보정을 통해 동일하게 계산된다.

Table 8. Validate Baseline With Sample Data

Family	sample1	sample2	sample3	sample4	sample5
adrd	34.73	101.99	**11.49**	24.88	114.87
*au	68.63	79.23	75.24	56.36	56.96
*bb	**2.24**	104.99	32.88	33.48	96.65
*dd	57.70	76.04	68.53	51.49	55.48
*fd	111.50	**50.20**	117.42	108.72	68.67
*gm	91.63	68.93	101.55	80.81	46.79
*gd	101.03	70.11	109.13	84.90	53.94
*ic	100.85	66.51	105.75	78.96	53.01
*im	68.82	95.56	63.36	42.36	58.62
*ks	94.04	54.14	97.04	79.00	59.97
*lo	96.87	58.53	98.88	82.40	77.34
*nd	74.85	130.38	46.71	39.94	**19.82**
*of	80.81	72.40	88.80	57.81	50.99
*pj1	82.54	66.48	91.36	72.27	47.24
*pj2	42.79	88.75	56.99	49.43	70.15
*ra	103.33	101.48	87.30	63.65	56.36
*ru	32.30	99.76	27.59	**0**	83.77
*uu1	60.89	100.02	66.47	40.63	54.28
*uu2	77.08	69.18	85.22	74.03	48.82
*uu3	69.32	103.70	45.41	31.38	119.43
*wi	38.70	94.05	49.75	38.78	70.73
*ah	77.86	131.87	36.48	24.78	81.80

최종적으로 약간의 순서 정보 변형이 있는 데이터에 대해서도 올바르게 순서 정보를 통한 패밀리 분류가 가능함을 검증하였다.

V. 결 론

본 논문에서는 시스템 API 호출 순서 정보를 통한 모바일 악성 코드 패밀리 분류를 제안한다. 해당 방법을 통해 총 135개의 샘플 3개의 카테고리에서 순서 정보를 추출하고, 해당 순서 정보를 각 카테고리별 주요 시스템 콜과 시스템 콜 내의 매개변수를 기반으로 전처리를 진행하고, 이후 DTW 알고리즘을 적용하여 22종의 패밀리 별 올바른 분류가 가능함을 보여주었다. 이를 통해 시스템 API 호출 순서 정보를 통해 모바일 악성 코드 패밀리 분류가 가능함을 제시한다.

최근 정적 분석과 동적 분석을 결합한 탐지 및 분류 방법론, 하이브리드 방법론에 대한 제안이 이뤄지

고 있다. 이러한 연구의 동적 분석에는 strace -c 옵션을 사용하거나, strace 전체를 추출하는 방식이 존재한다. 하이브리드 방법은 정적, 동적 한 개의 특징만을 사용하던 과거와 달리 두 개의 혹은 그 이상의 특징들을 학습 해야 한다. 해당 방법론을 이용하여 보다 경량화된 데이터와 정적 분석과 동적 분석을 결합하여 분류를 진행하면 더욱 더 좋은 분류 및 탐지 방법론이 될 것이라고 기대한다.

우리가 제안한 연구는 데이터셋의 한계, SDK 버전 패치로 인하여 더 많은 카테고리, 패밀리를 분류하지 못한다는 한계가 있다. 향후 대규모 데이터셋을 확보하여 머신러닝에 해당 방법론을 적용하여 머신러닝을 통한 패밀리 분류를 진행해보고자 한다.

References

[1] Kaspersky, https://securelist.com/it-threat-evolution-q1-2024-mobile-statistics/112750/, (accessed Feb. 05, 2025)

[2] McAfee, https://www.mcafee.com/blogs/other-blogs/mcafee-labs/new-hidden ads-malware-that-runs-automatically-and-hides-on-google-play-1m-users-affected/, (accessed Feb. 05, 2025)

[3] Android Source Document, https://source.android.com/docs/core/architecture, (accessed Feb, 05, 2025)

[4] Android Developers Document, https://developer.android.com/tools/adb, (accessed Feb. 05, 2025)

[5] Taniya Bhatia and Brishabh Kaushal, "Malware detection in android based on dynamic analysis," 2017 International Conference on Cyber Security And Protection Of Digital Services (Cyber Security), pp. 1-6, June. 2017.

[6] M. Jaiswal, Y. Malik and F. Jaafar, "Android gaming malware detection using system call analysis," 2018 6th International Symposium on Digital Forensic and Security (ISDFS), pp. 1-5, March. 2018.

[7] S. Mahdavifar, A. F. Abdul Kadir, R. Fatemi, D. Alhadidi and A. A. Ghorbani, "Dynamic android malware category classification using semi-supervised deep learning," 2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech), pp. 515-522, Aug. 2020.

[8] Moutaz Alazab, Mamoun Alazab, Andrii Shalaginov, Abdelwadood Mesieh, Albara Awajan, "Intelligent mobile malware detection using permission requests and API calls," Future Generation Computer Systems, vol. 107, no. C, pp. 509-521, June. 2020

[9] K. Allix, T. F. Bissyandé, J. Klein and Y. L. Traon, "AndroZoo: Collecting millions of android apps for the research community," 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR), pp. 468-471, May. 2016.

[10] Virustotal, https://www.virustotal.com, 2004, (accessed Feb. 05, 2025)

[11] Kaspersky, https://usa.kaspersky.com/about/press-releases/kaspersky-uncovers-new-malware-family-used-by-lazarus-subgroup-andariel, (accessed Feb. 05, 2025)

〈저자소개〉

최 재 민 (Jae-Min Choi) 학생회원
2022년 3월: 대전대학교 정보보안학과 학사
2024년 3월~현재: 세종대학교 정보보호학과 석사과정
〈관심분야〉 시큐어 코딩, 모의 해킹, 악성코드 분석

최 상 훈 (Sang-Hoon Choi) 정회원
2014년 3월: 대전대학교 정보보안학과 학사
2016년 3월: 대전대학교 전산정보보안학과 석사
2023년 2월: 세종대학교 정보보호학과 박사
2023년 3월~현재: 세종대학교 SysCore Lab. 박사후 연구원
〈관심분야〉 하이퍼바이저, 시스템 모니터링, 시스템 메모리, 악성코드 분석, 딥러닝

박 기 웅 (Ki-Woong Park) 종신회원
연세대학교 Computer Science 학사
KAIST Electrical Engineering 석사
KAIST Electrical Engineering 박사
2009년 10월: Microsoft Research, Graduate Research Fellow
2012년 8월: 국가보안기술연구소 연구원
2016년 8월: 대전대학교 정보보안학과 교수
2016년 9월~현재: 세종대학교 정보보호학과 교수
〈관심분야〉 클라우드 시스템 보안, 초고속 보안 시스템, 시스템 인스펙션, 디지털 포렌식