API들의 동시 출현 그래프와 커뮤니티 탐지용 Louvain 방법을 활용한 안드로이드 악성 앱 탐지

노경민[1] 이승민[2] 김유담[2] 안석현[3] 조성제[2]
단국대학교 사이버보안학과[1], 단국대학교 소프트웨어학과[2], 단국대학교 인공지능융합학과[3]
{imsie1, seungmin_lee, kyd1012kr, seokhyun, sjcho}@dankook.ac.kr

Android Malware Detection using Co-occurrence Graphs of APIs and Louvain Method for Community Detection

Kyoungmin Roh[1] Seungmin Lee[2] Yudam Kim[2] Seokhyun Ahn[3] Seong-je Cho[2]
Department of Cybersecurity, Dankook University[1]
Department of Software Science, Dankook University[2]
Department of AI-based Convergence, Dankook University[3]

요 약

머신러닝 기반의 안드로이드 악성 앱 탐지 연구가 많이 수행되어 왔지만, 시간에 따른 악성 행위 양상의 변화, 즉 개념 드리프트(concept drift)로 인해 악성 앱 탐지 기법의 성능 저하 문제가 발생한다. 본 논문은 개념 드리프트에 강건하며 재학습 없이도 비교적 안정적으로 악성 앱을 탐지하는 기법을 제안한다. 우리는 앱 내 API 호출 간 동시 출현 그래프(co-occurrence graph)를 구성하고, Louvain 알고리즘을 통해 의미 기반 커뮤니티를 정의한다. 각 API는 Node2Vec 임베딩을 통해 벡터로 표현되며, 앱 단위 표현은 평균 풀링과 커뮤니티 통계 정보를 결합해 생성된다. 이후, 커뮤니티별 RF(Random Forest) 분류기를 학습한다. 제안한 방법은 2014–2017년 데이터를 학습에 활용하고, 2018–2023년 데이터를 테스트한 결과, 평균 F1-score 0.8559 및 Accuracy 0.8384를 달성하였다. 실험 결과, 제안 기법은 단일 전역 분류기로 구성된 RF 모델에 비해 개념 드리프트에 강건함을 보여준다.

1. 서 론

안드로이드 플랫폼은 높은 시장 점유율과 개방성으로 인해 대표적 모바일 앱 생태계를 형성하고 있다. Fedscoop에 따르면, 2015년 기준 전체 모바일 악성 앱의 92%가 안드로이드에 집중되어 있다[1]. 이에 따라 안드로이드 악성 앱 탐지는 모바일 보안의 핵심 과제로 주목받고 있다. 특히 머신러닝 기반의 정적 악성 앱 탐지 기법은, 빠른 분석 속도와 실행 환경 비의존성 등의 장점으로 널리 연구되고 있다.

한편, 탐지 기법을 우회하기 위해 악성 앱이 지속적으로 진화하고, 또한 시간이 지남에 따라 데이터 분포가 바뀌는 개념 드리프트(concept drift, 개념 변화)로 인해 기존 악성 앱 탐지 모델의 성능이 급격히 저하되는 문제가 발생한다. 주기적 재학습이나 드리프트 인지를 통해 이를 대응하는 연구가 수행되고 있으나, 현실적으로 높은 유지 비용과 운영 복잡성을 수반한다는 한계가 있다[2, 3].

본 논문에서는 개별 API 호출 여부보다, API 간 동시 출현 그래프 구조(co-occurrence structure)가 악성 행위를 더 체계적으로 설명할 수 있다는 점에 착안하여, 동시 출현 그래프를 생성하고 의미적으로 응집된 API 그룹을 Louvain 커뮤니티 탐지 알고리즘으로 악성 앱을 탐지한다. 각 API는 Node2Vec을 통해 임베딩되며, 앱 단위 표현은 호출된 임베딩 벡터의 평균 및 커뮤니티 통계 정보를 기반으로 생성된다. 이후, 우세 커뮤니티 단위로 분류기를 학습하고, 테스트 시에는 Youden's J를 통한 최적 임계값 설정을 통해 개념 드리프트에 적응한다.

제안한 기법은 2014–2017년 샘플을 학습 데이터로, 2018–2023년 샘플을 테스트 데이터로 하여 실험하여, 재학습 없이 평균 F1-score는 0.8574, Accuracy는 0.8408의 성능을 달성했다. 또한, 단일 전역 분류기에 비해 더 나은 성능을 나타내, 개념 드리프트에 더 강건함을 보여준다. 본 논문의 주요 기여는 다음과 같다.

- API 호출 간 동시 출현 관계를 기반으로 한 의미적 구조 학습을 위해, Louvain 커뮤니티 탐지 알고리즘과 Node2Vec 임베딩을 결합한 정적 악성 앱 탐지 프레임워크를 제안하였다.
- 커뮤니티 통계 정보를 결합하여 앱 단위 구조 표현을 구성하고, 커뮤니티별 RF (Random Forest) 분류기를 독립적으로 학습하였다.
- Youden's J를 이용하여 민감도와 특이도를 반영하여 최적 임계값을 선정하였다.
- 학습 데이터보다 나중에 수집한 테스트 데이터에 대해 재학습 없이도 비교적 높은 성능을 달성하였다.

2. 관련 연구

안드로이드 악성 앱 탐지 연구를, (1) 정적 분석 기반

탐지, (2) 그래프 구조 활용 기법, (3) 개념 드리프트 대응 전략으로 구분할 수 있다. 본 절에서는 이 세 가지 주제에 따라 관련 연구를 정리하고, 본 연구와의 차별점을 기술한다.

2.1 정적 분석 기반 악성 앱 탐지

정적 분석 기법은 앱 실행 없이 코드, 메타데이터, 리소스 등에서 특징을 추출하여 악성 앱을 탐지한다. 초기 연구들은 AndroidManifest.xml 및 바이트코드 기반의 권한(permissions), API 호출, 문자열 등을 조합해 전통적인 머신러닝 모델을 학습하였다 [4 - 6]. 이후 딥러닝 기반 모델들이 도입되면서 수작업 특징 설계의 한계를 보완하려는 시도도 이어졌다 [7 - 11].

그러나, 많은 기존 연구는 특정 API 존재 여부 등 개별 정보에만 의존했으며, API들 간의 의미 관계나 구조적 상호작용을 고려하지 않았다. 이에, 새로운 악성 행위 패턴을 반영하지 못해, 전반적으로 개념 드리프트에 취약하다.

2.2 그래프 구조를 활용한 탐지 기법

악성 앱의 행위 패턴을 모델링하기 위해, API 호출 흐름이나 권한 구조 등을 그래프 형태로 표현하는 연구들도 수행되었다. 예를 들어, API 호출 시퀀스를 마르코프 체인 또는 코드 호출 그래프로 변환하여 탐지하는 방식이 대표적이다[12]. 최근에는 GCN (Graph Convolutional Network) 등 그래프 신경망을 활용한 탐지 모델도 제안되었다[13 - 16].

이러한 방식은 행위 구조를 효과적으로 표현할 수 있는 장점이 있으나, 그래프 생성 과정이 복잡하고 계산 비용이 높으며, 그래프 구조 변화에 취약하다는 단점이 있다. 또한 많은 연구가 고정된 시점의 그래프 구조에 초점을 두고 있어 개념 변화에 대한 적응성이 부족한 편이다.

2.3 개념 변화 탐지 및 적응 연구

개념 드리프트는 실제 세계에서 악성 앱 탐지 모델의 성능을 떨어뜨리는 핵심 요인이다. 이를 해결하기 위해 드리프트 탐지 기법, 재학습 기반 적응 전략 등이 제안되었다[2, 3, 17]. 대표적으로는 분포 변화 감지를 통해 모델을 업데이트하거나, 스트리밍 방식의 적응형 학습이 시도되었다[18 - 20].

하지만 이러한 접근은 지속적인 데이터 수집, 주기적 재학습, 높은 운용 비용을 요구하며 실제 환경 적용에 있어 현실적인 제약이 있다.

3. 제안 방법

본 논문에서는 개념 드리프트에 강건한 정적 분석 기반 안드로이드 악성 앱 탐지 기법을 제안한다. 핵심 개념은 API 호출 간의 동시 출현 구조를 그래프로 구성하고, Youden's J 값이 최대가 되는 지점의 최적 임계값을 설정하여 최종 탐지를 진행한다. 전체적인 악성 앱 탐지 구조가 그림 1에 나타나 있다.

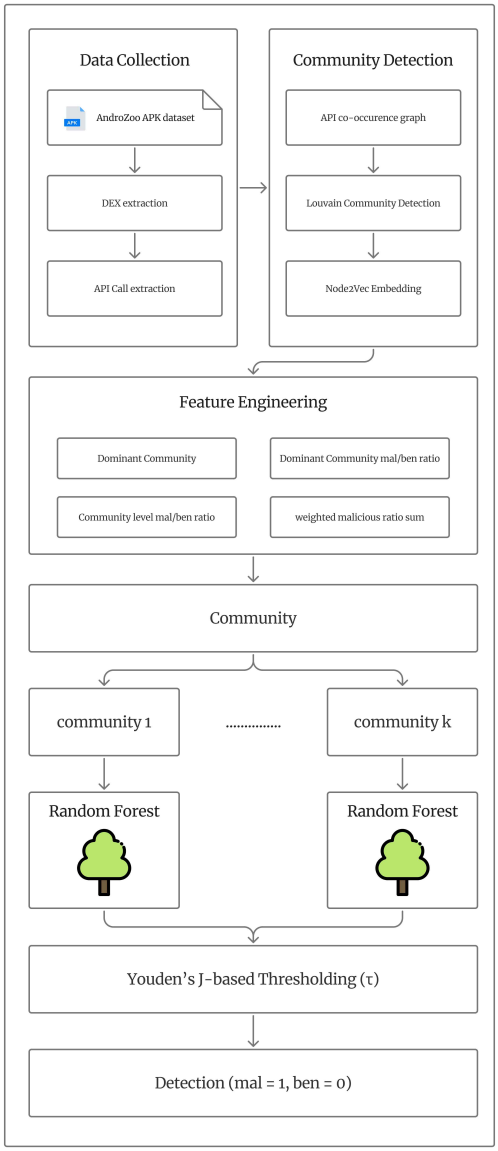

그림 1. 제안 방법의 프레임워크 구조도

3.1 API 동시 출현 그래프 구성

안드로이드 애플리케이션은 외부 자원이나 시스템 기능과 상호작용하기 위해 다양한 API를 호출한다. 이러한 호출 정보는 각 앱마다 API 호출 여부를 나타내는 이진 벡터로 표현할 수 있으며, 본 연구에서는 이를 바탕으로 API 간의 동시 출현 관계를 모델링한다.

특히 본 논문에서는 앱별로 개별 그래프를 구성하는 방식이 아닌, 전체 학습 데이터에 포함된 수많은 앱들의 호출 정보를 종합하여 하나의 전역 동시 출현 그래프를 구축한다. 이 그래프는 API 간의 공동 호출 빈도를 기반으로 하며, 각 노드는 개별 API를, 엣지는 동일한 앱 내에서 두 API가 함께 호출된 횟수를 의미하는 가중치로 구성된다. 예를 들어, 앱 A와 앱 B 모두에서 API1과 API2가 함께 호출되었다면, 두 API 사이의 엣지 가중치는

2로 설정된다.

이와 같이 구성된 전역 그래프는 전체 앱 생태계에서 관찰되는 호출 패턴의 일반화된 구조를 반영한다. 단일 앱의 호출 정보로는 드러나지 않는 기능적 연관성과 의미적 유사성을 전역적 시야에서 포착할 수 있으며, 이후 그래프 기반 커뮤니티 탐지와 의미 표현 학습의 기반이 된다.

3.2 Louvain 알고리즘 기반 커뮤니티 탐지

동시 출현 그래프 G가 구성되면, 해당 그래프의 구조적 응집성을 분석하여 의미적으로 밀접한 API 호출 그룹을 식별하는 것이 중요하다. 이를 위해 본 연구에서는 Louvain 알고리즘을 적용하여 그래프 내 커뮤니티 구조를 탐지하였다. Louvain 알고리즘은 그래프의 모듈성(modularity)을 최대화하는 방향으로 노드들을 클러스터링하며, 이때 각 커뮤니티는 동일한 기능적 맥락 내에서 자주 함께 호출되는 API들의 집합으로 해석된다.

각 API 노드는 하나의 커뮤니티 C에 속하게 되며, 각 앱은 자신이 호출한 API들 중 가장 많이 속한 커뮤니티를 우세 커뮤니티(dominant community)인 $c^*(x)$로 할당받는다. 이 과정은 다음과 같은 수식 (1)로 정의된다.

$$c^*(x) = \arg\max_{c \in C} \sum_{a \in A_x} 1\{a \in c\} \quad \cdots(1)$$

여기서 A_x는 앱 x에서 호출된 API 집합이다. 이와 같은 우세 커뮤니티 할당은 각 앱의 주요 기능적 중심축을 규정하며, 이후 특징 생성 및 탐지기의 분리 학습 단계에서 기준 역할을 수행한다. Louvain 알고리즘은 계산 효율성 측면에서도 우수하여 대규모 API 그래프에도 효과적으로 적용 가능하다.

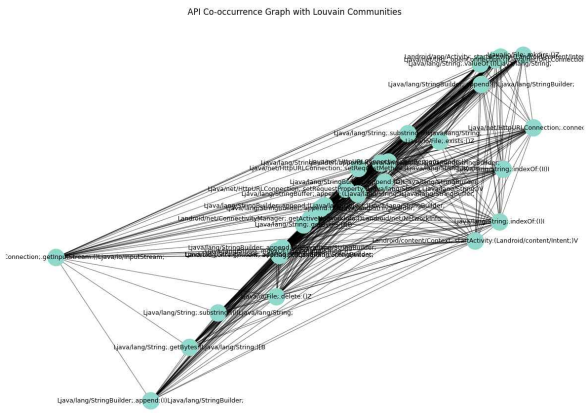

그림 2. Louvain 커뮤니티 구조 기반 API 동시 출현 그래프 시각화 (상위 30개 API 호출)

이를 보다 직관적으로 이해하기 위해, 그림 2는 학습 데이터에서 생성된 동시 출현 그래프의 일부분을 시각화한 결과를 보여준다. 전체 그래프에서 연결 중심성이 높은 API 30개를 선택하여 서브그래프를 구성하였으며, 각 노드는 API 호출을 나타내고, 엣지는 동일한 앱 내에서 두 API가 함께 호출된 빈도에 비례한 가중치를 갖는다. Louvain 알고리즘은 이 그래프 전체 G에 대해 적용되었다.

3.3 커뮤니티 기반 특징 추출

커뮤니티 구조가 정의된 이후, 각 앱에 대해 커뮤니티 기반 통계 특징과 구조 임베딩 특징을 포함하는 표현 벡터를 구성한다. 첫째, 각 앱의 우세 커뮤니티에 속한 앱들 중 악성 또는 정상 앱의 비율을 사전 계산하고, 이를 해당 앱에 각각 특징 정보 이름을 dominant_comm_mal (c^*_{mal}) 및 dominant_comm_ben (c^*_{ben}) 값으로 하여 부여한다. 이는 다음과 같은 수식으로 각각 정의된다.

$$c^*_{\text{mal}}(x) = P(y=1 \mid c^*(x)) \quad \cdots(2)$$
$$c^*_{\text{ben}}(x) = 1 - P(y=1 \mid c^*(x)) \quad \cdots(3)$$

이는 앱이 속한 커뮤니티의 악성도 수준을 구조적으로 요약한 값이며, 추후 탐지기의 중요한 입력 정보로 활용된다.

둘째, 각 앱이 호출한 API들에 대해 학습(훈련) 데이터에서 사전 계산된 악성 비율의 평균을 계산하여 코드에서 malicious_api_ratio로 정의한다. 이는 수식 (4)로 정의된다. 수식에서 MR은 Malicious Ratio의 약자로 학습 데이터에서 사전 계산된 악성 앱 비율이며, 그것의 mean은 악성 앱 비율의 평균을 의미한다.

$$\text{MR}_{\text{mean}}(x) = \frac{1}{|A_x|} \sum_{a \in A_x} P(y=1 \mid a) \quad \cdots(4)$$

셋째, 각 앱의 API 호출 목록에 대해 Node2Vec 알고리즘을 적용하여 생성된 API 임베딩 벡터들의 평균값을 계산함으로써 앱 전체의 고차원 의미 기반 표현을 구성한다. 이는 다음과 같은 수식 (5)로 정의된다.

$$\text{embedding}(x) = \frac{1}{|A_x|} \sum_{a \in A_x} \text{v}_a \quad \cdots(5)$$

여기서 v_a는 API a의 8차원 Node2Vec 임베딩 벡터이다. 이러한 구조적 임베딩은 단순 통계치만으로는 포착하기 어려운 의미적 관계를 보완하며, 탐지기의 표현력을 높인다.

3.4 분류기 학습 및 예측

각 앱은 우세 커뮤니티 ID에 따라 하나의 커뮤니티에 소속되며, 각 커뮤니티에 대해 별도의 RF 분류기를 학습한다. 이를 통해 커뮤니티별 행위 특성에 최적화된 탐지가 가능해지고, 단일 전역 모델이 가지는 표현 한계를 보완할 수 있다. 학습은 2014년부터 2017년까지 수집된 데이터를 기반으로 진행되며, 테스트는 2018년부터 2023년까지의 앱을 대상으로 한다. 이와 같은 연도별 데이터

분할은 실제 환경에서 발생하는 개념 드리프트 현상을 모사하기 위한 설정이다.

각 커뮤니티 분류기는 학습 데이터에서 출력한 예측 확률을 기반으로 ROC 곡선을 계산하고, 다음과 같이 정의되는 Youden's J 통계량이 최대가 되는 지점에서 최적 임계값을 선정한다. J는 수식 (6)로 정의된다.

$$J(\tau) = \mathrm{TPR}(\tau) - \mathrm{FPR}(\tau) \qquad \cdots(6)$$
$$\text{where } \tau^* = \arg\max_{\tau} J(\tau)$$

여기서 τ는 임계값, TPR은 True Positive Rate, FPR은 False Positive Rate이다. 이 방식은 민감도와 특이도 간의 균형을 기반으로 하며, 클래스 불균형 상황에서도 안정적인 기준점을 제공한다.

4. 실험 및 결과

4.1 실험 환경

실험에는 2014년부터 2023년까지 Androzoo에서 수집한 총 76,000개의 안드로이드 애플리케이션 데이터셋을 사용하였다.

데이터셋을 연도 기준으로 학습 데이터(2014-2017)와 테스트 데이터(2018-2023)로 구분하였는데, 이는 개념 드리프트를 모사하기 위함이다. 각 연도별로는 악성과 정상 샘플 수를 동일하게 구성하였다. 즉, 학습 및 테스트 모두 클래스 레이블 기준으로 정상/악성 비율이 1:1로 균형을 이루도록 구성되었다. 표 1은 연도별 데이터 구성을 요약한 것이다.

표 1. 연도별 정상/악성 앱 수 분포

연도	정상	악성
2014	5,000	5,000
2015	5,000	5,000
2016	5,000	5,000
2017	5,000	5,000
2018	3,000	3,000
2019	3,000	3,000
2020	3,000	3,000
2021	3,000	3,000
2022	3,000	3,000
2023	3,000	3,000
전체	38,000	38,000

학습 데이터는 총 40,000개(연도별 10,000개), 테스트 데이터는 총 36,000개(연도별 6,000)개로 구성된다. 학습 및 예측 과정 전에는 모든 string 타입 특징(feature)을 제거하였으며, API 호출 특징을 중심으로 정규화 처리를 수행하였다. 이때 정규화에는 학습 데이터에 대해 적합된 StandardScaler를 사용하였고, 동일한 스케일링 파라미터를 테스트 데이터에도 적용하였다. 이러한 정규화는 커뮤니티 탐지, 임베딩 학습, 분류기 학습의 안정성 개선에 도움을 줄 수 있다.

4.2 데이터 전처리

API 호출 정보가 매우 희귀하거나 동시 출현 그래프 내 주요 커뮤니티에 포함되지 않은 API만 사용하는 일부 앱의 경우, 우세 커뮤니티를 할당할 수 없어 dominant_comm = -1로 설정되었다. 이러한 앱은 제안한 커뮤니티 기반 분류기 구조와 호환되지 않기 때문에 성능 평가에서 제외되었다. 전체 테스트 샘플 중 약 2.2%가 이에 해당하며, 이로 인해 실제 레이블 기준 전체 악성 앱 수보다 다소 줄어든 수의 악성 앱이 최종 테스트셋에 포함되었다. 해당 앱들은 일반적인 커뮤니티 기반 API 구조와 상이한 outlier로 해석될 수 있다.

4.3 성능 평가

4.3.1 테스트 데이터에 대한 성능 분석

제안한 모델은 2018년부터 2023년까지 총 3,6000개의 테스트 데이터를 대상으로 평균 정확도 0.8384, F1-score 0.8559의 성능을 달성하였다. 각 샘플은 우세 커뮤니티 기준의 RF 분류기에 입력되었고, 커뮤니티별 임계값을 적용하여 최종 이진 분류를 수행하였다. 연도별 분석 결과, 모든 연도에서 F1-score가 0.82 이상으로 유지되었으며, 특히 학습 기간과 유사한 API 패턴이 유지된 2018년에는 0.9378로 가장 높은 성능을 기록하였다. 이러한 결과는 제안한 모델이 개념 드리프트에 비교적 강건하며, 재학습 없이도 탐지 성능을 어느 정도 유지할 수 있음을 보인다. 다음 표 2, 표 3는 테스트 데이터에 대한 성능을 나타낸 것이다.

표 2. 연도별 정확도 및 F1-score

연도	정확도	F1-score
2018	0.9373	0.9378
2019	0.8243	0.8473
2020	0.8101	0.8352
2021	0.8096	0.8277
2022	0.8313	0.8553
2023	0.8169	0.8408
전체 평균	0.8384	0.8559

표 3. 전체 테스트 데이터에 대한 혼동 행렬

	예측 정상	예측 악성
실제 정상	12,813	5,034
실제 악성	749	17,181

4.3.2 비교 실험: 베이스라인 모델과 비교

본 절에서는 제안한 탐지 모델의 성능을, 단일 전역 분류기로 구성된 RF-Baseline과 비교 분석하였다. RF-Baseline은 전체 학습 데이터를 통합하여 하나의 RF 분류기를 학습하고, 모든 테스트 앱에 대해 같은 기준으로 예

측을 수행하는 구조로, 기존의 일반적인 안드로이드 악성 앱 탐지 방식과 유사하다.

연도별 성능 비교 결과, 제안한 커뮤니티 기반 탐지 모델은 RF-Baseline 모델보다 전반적으로 크게 우수한 성능을 보였다. 전체 테스트 데이터를 기준으로 계산된 정확도는 약 25%p, F1-score는 약 15%p 더 높은 수치를 기록하였다. 특히 2019년 이후 연도에서는 RF-Baseline의 성능이 급격히 하락했지만, 제안 모델은 모든 연도에서 안정적인 성능을 유지하며, 개념 드리프트 상황에서도 높은 탐지 신뢰성을 유지하는 결과를 보였다.

그림 3과 4는 두 모델에 대해, 연도별 테스트 데이터에 대한 성능 차이를 나타낸 것이다.

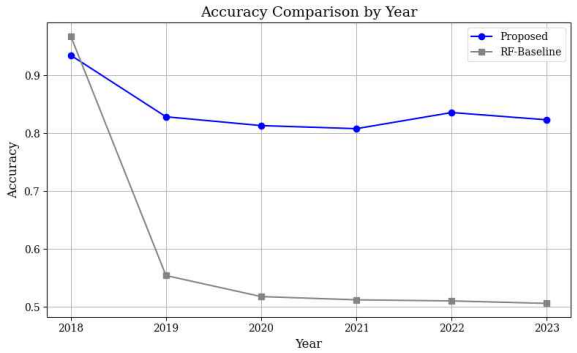

그림 3. 연도별 정확도 비교

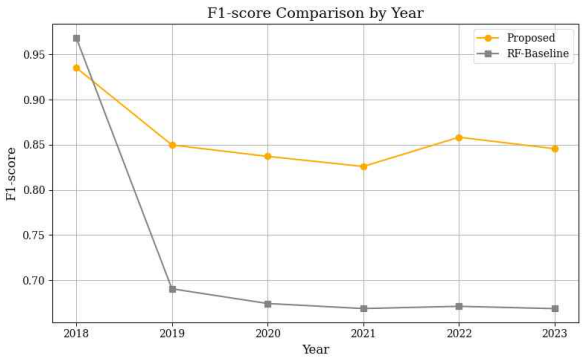

그림 4. 연도별 F1-score 비교

전반적으로 RF-Baseline은 학습 시점과 가까운 2018년에는 높은 성능을 보였으나, 이후 연도가 지날수록 F1-score와 Accuracy가 급격히 저하되는 현상이 관찰되었다. 반면, 제안 기법은 모든 연도에서 F1-score 0.82 이상을 유지하며, 개념 드리프트가 발생하는 상황에서도 안정적인 탐지 성능을 보였다.

이러한 결과는 제안한 커뮤니티 기반 구조화 전략이 시간에 따른 악성 앱 행위 양상의 변화를 효과적으로 반영하고, 재학습 없이도 장기적인 일반화 성능을 유지할 수 있음을 의미한다.

5. 결론 및 한계점

본 논문에서는 시간에 따라 변화하는 안드로이드 악성 앱의 행위 양상, 즉 개념 드리프트 문제에 강건하게 대응할 수 있는 정적 분석 기반 탐지 모델을 제안하였다. 우리는 API 호출 간의 동시 출현 관계를 기반으로 그래프를 구성하고, Louvain 알고리즘을 통해 의미 기반 커뮤니티를 정의하였다. 이후 각 커뮤니티에 대해 독립적인 분류기를 학습하고, 통계 기반 구조적 특징 및 Node2Vec 임베딩을 함께 사용함으로써, 개념 드리프트에도 어느 정도 강건한 모델을 개발하였다.

실험 결과, 본 모델은 재학습 없이도 2018년부터 2023년까지의 테스트 데이터에 대해 평균 F1-score 0.8574의 성능을 유지하였다.

그러나 본 연구는 다음과 같은 한계도 함께 가진다. 첫째, 커뮤니티 구조에 속하지 못하는 앱들(dominant_comm = -1)에 대해 고려하지 않기 때문에 전체 탐지 coverage가 제한될 수 있다. 둘째, 커뮤니티 간 데이터 불균형 문제로 인해 일부 소형 커뮤니티에서 학습이 제한되거나 성능이 제한될 수 있다. 셋째, 본 연구는 정적 분석 정보만을 사용하였으며, 실행 시점에서 나타나는 동적 행위나 context-aware sequence 정보는 반영하지 못한다. 마지막으로, 2018년부터 2019년 사이에 발생하는 concept drift를 해결하지 못하였다.

향후 이러한 한계점을 보완하기 위해, 커뮤니티 외부 앱을 위한 전역 fallback 모델 설계, 커뮤니티 간 적응형 분류기 구조 도입, 정적-동적 분석 결합 및 sequence-aware embedding의 통합이 고려될 수 있다. 이를 통해 더욱 높은 탐지 성능과 실용성을 확보할 수 있을 것으로 기대한다.

Acknowledgments

본 연구는 과학기술정보통신부 및 정보통신기획평가원의 학석사연계ICT핵심인재양성사업의 연구결과로 수행되었음. (IITP-2023-00259867) 또한, 본 연구는 산업통상자원부(MOTIE)와 한국에너지기술평가원(KETEP)의 지원을 받아 수행한 연구 과제임. (No. RS-2021-KP002461)

참고 문헌

[1] Fedscoop, “As government turns to Android Smartphones, so does malware,” Fedscoop, Jan. 2015.

[2] Q. Hu, W. Wang, H. Song, and S. Guo, “ASDroid: Resisting Evolving Android Malware With API Clusters Derived From Source Code,” IEEE Transactions on Dependable and Secure Computing, vol. 22, no. 2, 2025.

[3] M. Yudin, A. Reddy, S. Venkatesan, and R. Izmailov, “Backdoor Attacks During Retraining of Machine Learning Models: A Mitigation Approach,” Proceedings of the 11th International Conference on Information Systems Security and Privacy (ICISSP), 2024.

[4] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, “Drebin: Effective and explainable detection of android malware in your pocket,” Proceedings of the Network and Distributed

System Security Symposium (NDSS), pp. 23–26, 2014.
[5] M. Zhao, J. Liu, X. Zhang, and S. Wang, "MFDroid: Feature selection based stacking ensemble learning for Android malware detection," Computers & Security, vol. 110, p. 102432, 2021.
[6] S. Sharma and B. Sahay, "An intelligent hybrid approach for Android malware detection using permissions and API calls," Computers & Security, vol. 92, p. 101748, 2020.
[7] S. Kim, S. Kim, S. Lee, and H. Kim, "MAPAS: A practical deep learning-based android malware detection system," International Journal of Information Security, vol. 21, pp. 217–233, 2022.
[8] S. S. Islam, M. A. Hossain, and M. F. Bari, "DL-Droid: Deep learning based android malware detection using real devices," Computers & Security, vol. 90, p. 101722, 2020.
[9] Y. Wang, X. Zhang, and C. Wang, "A Deep Learning Based Android Malware Detection System with Static Analysis," Proceedings of the 46th IEEE Computers, Software, and Applications Conference (COMPSAC), pp. 1161–1166, 2022.
[10] S. U. Rehman, Z. U. Khan, and K. Muhammad, "DeepMDFC: A deep learning based android malware detection and family classification method," Security and Privacy, vol. 6, no. 6, e347, 2023.
[11] Y. Zhang, Y. Liu, and H. Liu, "A Systematical Study for Deep Learning Based Android Malware Detection," Proceedings of the 2nd International Conference on Artificial Intelligence and Advanced Manufacturing (AIAM), pp. 1–5, 2020.
[12] M. Mariconti, L. Onwuzurike, P. Andriotis, E. De Cristofaro, G. Ross, and G. Stringhini, "MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models," Proceedings of the Network and Distributed System Security Symposium (NDSS), 2017.
[13] Y. Wang, J. Zhang, and Y. Wang, "Android Malware Detection Method Based on Graph Convolutional Networks," Proceedings of the IEEE 6th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), pp. 1–6, 2024.
[14] S. S. Islam, M. A. Hossain, and M. F. Bari, "Graph Convolutional Networks for Android Malware Detection with System Call Graphs," Proceedings of the IEEE International Conference on Communications Workshops (ICC Workshops), pp. 1–6, 2020.
[15] R. K. Sahu, S. K. Sahu, and S. K. Rath, "Android Malware Detection using Large-scale Network Representation Learning," Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM), pp. 1599–1602, 2018.
[16] S. R. Gurram, R. S. S. K. Prasad, and M. S. S. R. K. Prasad, "Heterogeneous Graph Convolutional Networks for Android Malware Detection using Callback-Aware Caller-Callee Graphs," TechRxiv, preprint, 2021.
[17] S. Park, H. Lee, D. Kim, H. J. Moon, S. Cho, and Y. Hwang, "Enhancing the Sustainability of Machine Learning-Based Malware Detection Techniques for Android Applications," IEEE Transactions on Information Forensics and Security, vol. 20, no. 1, 2025.
[18] J. Zhang, Y. Wang, and X. Liu, "Adaptive Android Malware Detection via Distribution Change Detection and Model Update," IEEE Transactions on Information Forensics and Security, vol. 18, pp. 1234–1245, 2023.
[19] L. Chen, M. Zhao, and S. Wang, "Streaming-based Adaptive Learning for Android Malware Detection," IEEE Access, vol. 12, pp. 56789–56799, 2024.
[20] H. Kim and J. Lee, "Concept Drift Adaptation in Android Malware Detection Using Streaming Data," Proceedings of the IEEE International Conference on Data Mining (ICDM), pp. 789–796, 2023.