説明可能な AI に基づく IoT マルウェアの挙動解析に関する研究

代表研究者	森川　智博	兵庫県立大学　情報科学研究科　准教授
共同研究者	林　　宗男	国立台湾大学　教授
共同研究者	范　　俊逸	国立中山大学　教授

1 はじめに

スマートフォンの普及に伴い、ソーシャル、金融、教育、技術など多様な分野において、多くのアプリケーションが利用可能となり、日常生活に必要なサービスを提供している。統計によれば、現在のモバイルネットワークトラフィックの大部分はモバイル端末から発生しており、そのうち 70%以上が Android OS を搭載したデバイスによるものである [1]。これにより、Android は現在最も広く使用されているモバイルプラットフォームとなっている。一方で、Android はマルウェアの主要な標的にもなっており、2022 年には 20 万件以上の Android マルウェアが研究者によって報告されている [2]。このようなマルウェアの急増は、モバイルユーザのプライバシーおよびセキュリティに重大な脅威をもたらしている。

この課題に対処すべく、これまでに多くの Android マルウェア検知手法が提案されてきた。しかし、従来のシグネチャベースの検知方式では、日々増加する多種多様なマルウェアに対応するには限界がある。さらに、マルウェアは暗号化や難読化などの手法を用いて検知を回避する可能性がある。これを受けて、近年ではヒューリスティックベースの手法への関心が高まり、特に機械学習を応用した検知手法が注目されている。畳み込みニューラルネットワーク（CNN）、リカレントニューラルネットワーク（RNN）、多層パーセプトロン、オートエンコーダといった様々な手法が提案されており、アプリケーションのバイナリコードを特徴ベクトルへ変換し、それに基づいて正常／異常を分類することが試みられている。

Senanayake らの研究 [3] によれば、バイナリコードの解析に加え、Android システムに関する知識――たとえばシステム構成情報、インテントフィルタ、機密性の高い API の使用状況など――を活用することで、マルウェア検知の精度を向上させることが可能である。また、コードセマンティクスに基づくアプローチでは、アプリのバイナリから Smali バイトコードや Java ソースコードを復元し、制御フローやデータフローといった意味的特徴を抽出することで、より高い性能が実現されている。

さらに近年では、グラフ構造の解析に優れるグラフニューラルネットワーク（GNN）に基づく手法も提案されている [4, 5, 6, 7]。一部の研究では、関数呼び出しやレジスタ情報をグラフ構造として表現し、GNN によりマルウェアの特徴を学習することで、検知精度の向上が確認されている。GNN のトポロジー構造処理能力を活用することで、アプリケーション内部の構造的特徴を効果的に捉え、検知性能を高めることが可能となっている。

しかしながら、機械学習に基づく既存の Android マルウェア検知手法は、しばしば説明可能性に課題を抱えている。高性能なモデルの多くは複雑な構造を有しており、その判定過程を人間が理解することは困難である。こうした透明性の欠如は、モデルの信頼性や実運用に対する懸念を生じさせる要因となる。加えて、機械学習モデルがどのような根拠でアプリをマルウェアと判定しているのかについての明確な説明が困難であることも問題視されている。

以上の背景を踏まえ、本研究では、高性能かつ説明可能な Android マルウェア検知手法の確立を目的とし、グラフアテンションネットワーク（Graph Attention Network: GAT）に基づく新たな検知フレームワークを提案する。本手法は、モバイル端末上での早期検知を可能とし、モバイルインターネット全体への脅威拡散を未然に防止することを目指すものである。

2 提案手法

2-1 アーキテクチャ

図 1 に本提案手法の全体アーキテクチャを示す。矢印はデータの流れを、長方形は各ステージにおける詳細な処理手順を、カード型の図形はそれぞれの処理における入力および出力データを表している。右側に配置された 2 つのカードは本手法の出力を示しており、フレームワークは APK 形式の Android アプリを入力

公益財団法人電気通信普及財団
研究調査助成報告書　第40号　2025年度

として受け取り、2 つの出力──アプリがマルウェアである確率および疑わしい API 呼び出し列──を生成する。本フレームワークにおける検知プロセスは、大きく 3 つのステージに分かれる：グラフ抽出（ステージ 1)、API 特徴抽出（ステージ 2)、および悪意ある動作の検知（ステージ 3）である。 ステージ 1 では、到達定義解析（Reaching Definition Analysis）を用いて、疑わしい API 呼び出しにおけるデータ依存関係を抽出する。この解析によって得られる出力は、有向グラフとして表現された疑わしい API 呼び出しデータ依存グラフ（Suspicious API Call Data Dependency Graph: SACDDG）である。ステージ 2 では、自然言語処理（NLP）技術である単語埋め込み（Word Embedding）を導入し、Android API を意味的特徴空間に変換する。具体的には、Skip-Gram モデルを採用して API を分散表現ベクトルとして符号化する。ステージ 3 では、提案するグラフアテンションネットワーク（GAT）ベースのモデルを用いて、SACDDG における各データフローを分析し、悪意ある動作の有無を判定する。各データフローに対して API 特徴をもとにモデルが分析を行い、それが悪意ある動作を含む確率を出力する。最終的に、これらの確率を Softmax 関数によって集約し、対象アプリがマルウェアである全体的な確率を出力する。なお、ステージ 1 およびステージ 2 は前処理に相当し、その詳細については 2.2 節にて述べる。ステージ 3 は、提案する GAT ベースモデルによる処理であり、詳細は 2.3 節にて説明する。

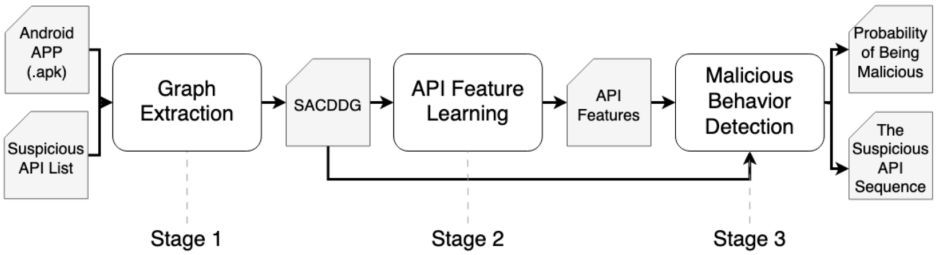

図 1

2-2 データの前処理

データ前処理は、モデルの性能──特にその有効性──に大きな影響を及ぼす要素の一つである。高性能な機械学習モデルを構築するためには、大量かつ高品質な学習データが必要となるが、その収集は依然として困難な課題である。そのため、本研究ではモデルの性能を最大限に引き出すことを目的として、データを慎重に処理している。また、機械学習モデルに適した形式へアプリケーションを変換する処理も不可欠である。提案する GAT ベースのモデルでは、2 種類のデータを入力として受け取る。一つは、疑わしい API の振る舞い的特徴を表現する API 特徴リストであり、もう一つはアプリケーション内のデータフローを記録した有向グラフ（SACDDG: Suspicious API Call Data Dependency Graph）である。

データ前処理は主に 3 つのステップから構成される。まず第 1 に、モデルが悪意ある動作に関連する重要な API に着目できるようにするため、本研究ではマルウェアとの関連性が高い疑わしい API のリストを選定する。Android API は、クラス、メソッド、およびディスクリプタによって定義されており、Android システムとアプリケーションの間をつなぐインタフェースとして機能する。Android API は、アプリにおける基本的な動作（例：Wi-Fi の有効化、SMS メッセージの読み取り、他アプリの起動など）を表しており、アプリケーションはこれらの API を順序立てて呼び出すことで、ビジネスロジックを構築する。このため、API の呼び出し列を観察することでアプリの処理内容を把握でき、それによりマルウェア的な振る舞いの有無を精密に判定することが可能となる。したがって、Android API の追跡はマルウェア検知の分野における代表的なアプローチの 1 つである。しかしながら、機械学習モデルに Android API を適用するにあたり、次の 2 点の課題が存在する。第 1 の課題は、Android API の種類が膨大である点である。すべての API を追跡対象とすることは、処理の複雑性やモデルのスケーラビリティの観点から現実的ではない。第 2 の課題は、同一のクラスおよびメソッド名を持つ API は、一般に同一または類似の動作を示すことが多いため、すべての API を個別に追跡することが必ずしもモデルの性能向上に寄与しない点である。これらの課題に対処するため、本研究では、データセット内で出現頻度の高い上位 50%の API（計 11,135 件）を疑わしい API 群として選定する。以降のデータ前処理工程では、これらの API に焦点を当て、それ以外の API は除外対象とする。

API の動作をより正確に記述するために、本研究では API 呼び出しデータ依存グラフ（API Call Data

Dependency Graph: ACDDG）を導入する。これは APK ファイルを表現するグラフ構造であり、各ノードは Android API を、エッジは同一データフロー上で操作される API 間の依存関係を表す。グラフ内で任意の 2 つのノードが接続されている場合、エッジは呼び出し順に従い、後に実行される API を指す向きで構成される。

第 2 に、アプリケーションを代表するグラフを構築するため、到達定義解析（Reaching Definition Analysis）を用いて SACDDG（Suspicious API Call Data Dependency Graph）を抽出する。ACDDG の生成は、2 段階のステップで行う。まず、APK 内の各メソッドに対して到達定義解析（Reaching Definition Analysis）を実施し、アプリ内でのデータフローを抽出する。データフローとは、複数の Android API をノードとしたシーケンスであり、アプリケーション内でのデータの寿命を記述するものである。具体的には、データは先頭の API によって生成・返却され、その後の API によって処理・消費される。次に、すべてのデータフローを統合し、ACDDG を構築する。この統合処理は 2 つのステップで実施される。まず、空のグラフを初期化する。次に、抽出された各データフローを順にグラフへ追加する。各フローに含まれるノードについて、既にグラフ内に存在するノードがある場合は、それらを統合し、元のノードの後続ノードおよび前駆ノードの関係を継承させる。グラフ内に存在しないノードであれば、新たに追加する。この処理をすべてのデータフローに対して繰り返すことで、最終的に ACDDG が完成する。ACDDG の構築後、グラフ剪定（Graph Pruning）を行い、疑わしくない API ノード（前述の頻出 50%外）をグラフから除去する。これにより、検知対象となる重要な API の依存関係に焦点を絞った簡潔なグラフ構造を得ることができる。

最後に、API 特徴量の獲得には Skip-Gram モデルを用いる。Android API の振る舞い的特徴を抽出する一般的な手法としては、単語埋め込みなどの自然言語処理（NLP）技術が用いられており、その一例として GSDroid は Skip-Gram モデルと API の共起関係を用いて API 特徴を学習している。 GSDroid における API 共起の定義は、「同一メソッド内に出現する任意の 2 つの API の組（タプル）」である。しかしながら、この定義には問題がある。すなわち、1 つのメソッド内で複数のデータフローを並列に処理している場合、異なるデータフローを処理している無関係な API 同士が共起として扱われてしまう可能性がある。このようなケースでは、Skip-Gram モデルによって学習される API 特徴が歪められ、誤った意味的近接性を反映してしまう。そこで本研究では、GSDroid の定義を基にしつつも、より厳密な定義を採用する。すなわち、同一データフロー上で動作する 2 つの API のみを共起関係として認めることで、共起関係が実際の動作上の意味的関連性を担保するようにしている。この厳密な定義により、共起する API は確実に同一の振る舞いに寄与する関係に限定される。API 共起の抽出処理は二つのステップから構成される：まず、SACDDG 内のすべてのデータフローを抽出し、それぞれの API の流れを得る。次に、各データフロー内で出現する API のペアを共起タプルとして構築する。すなわち、同一データフロー上にある API 同士のみを共起とみなす。このようにして構築された API 共起タプルをもとに、Skip-Gram モデルを用いて各 API の意味的特徴量（埋め込みベクトル）を学習する。

2-3 GAT ベースモデルによる処理

本モデルは、SACDDG（Suspicious API Call Data Dependency Graph）を入力として受け取り、悪意ある振る舞いを行う確率およびその振る舞いに関連する疑わしい API 呼び出し列の 2 つの出力を生成する。モデルは大きく 2 つの部分から構成されており、それぞれの目的に応じて異なる処理を担っている。

第 1 の部分は、個々のデータフローが悪意ある動作に関与している確率を計算することを目的としており、次元数を段階的に減少させた複数のグラフアテンションネットワーク（Graph Attention Network: GAT）層から構成されている。第 2 の部分は、各データフローの確率を統合し、アプリ全体がマルウェアである確率を出力することを目的としており、グラフプーリング（Graph Pooling）手法と追加の GAT 層を利用して集約処理を行う。最終的に、本モデルは赤色矩形内で算出されたアプリ全体の悪性確率と、青色矩形内の GAT 層におけるアテンション重みの分布を追跡することで抽出された疑わしい API 呼び出し列を出力する。

第 1 の部分における各 GAT 層は、次元数を段階的に減少させながら処理を行い、さらに活性化関数として LeakyReLU を用い、レイヤ正規化（Layer Normalization）を適用している。GAT 層の採用には、以下の 3 つの主な理由がある：

第一に、GAT 層のアテンション機構および特徴伝播機構により、各データフローの構造的な特徴を効果的に学習できる点である。たとえば、ある未知のアプリが API A と API B を順に呼び出すデータフローを持つ場合、SACDDG には API A および API B を表すノードが存在し、両者が同一データフロー上にあるた

め、これらのノード間に有向エッジが形成される。GAT 層を通じて、API A の特徴ベクトルが隣接ノードである API B に伝播され、API B の特徴と結合されることで、API 呼び出しの逐次的な実行が模擬される。このような処理により、モデルはデータフロー全体としての意味的な特徴を獲得する。

第二に、GAT 層のアテンション機構により、疑わしいデータフローに対して重点的に重みを与えることが可能となる。各ノードにおいて、悪意ある振る舞いとの関連性が高いデータフローに対して高いアテンション重みが割り当てられ、これにより伝播中に発生しがちな特徴量の消失（feature vanishment）を効果的に抑制できる。

第三に、各層において特徴次元数を減少させることで、モデルがより重要度の高い特徴に集中し、特に悪意ある振る舞いに強く関係する特徴量の抽出を促進するよう設計されている。

全ての特徴伝播処理が完了すると、最終的に各ノードの特徴ベクトルの次元数は 2 に縮小され、全ノードはそれぞれ疑わしいデータフローに関する特徴を保持する状態となる。その後、各ノードの特徴ベクトルに対して Softmax 関数を適用することで、当該データフローが悪意ある振る舞いを含む確率へと変換する。

第 2 の部分では、1 層の GAT を用いてノード間の悪性確率を集約し、アプリ全体のマルウェア確率を算出する。この処理は 3 つのステップから構成される。最初に、グラフプーリング（Graph Pooling）を実行する。これにより、グラフ外に仮想ノード（virtual node）を生成し、グラフ内のすべてのノードからこの仮想ノードへエッジを接続する。次に、この構造に対して GAT 層を適用し、各ノードに含まれる悪性確率を仮想ノードへと集約する。この GAT 層は、アテンション機構により各ノードの重要度を動的に評価し、意味的に重要な情報を仮想ノードへ効果的に伝達する。最後に、仮想ノードに集約された特徴に対して Softmax 関数を適用し、0 から 1 の範囲に正規化されたスカラー値へと変換する。この値を、当該 APK がマルウェアである確率として扱う。

3 性能評価

本研究で提案するフレームワークにおいて、疑わしいデータフローを検出するモデルは中核的な要素である。したがって、本研究ではこの部分に対して複数のモデル候補を評価した。

まず、候補 1（Candidate 1）は初期設計であり、特徴次元数を一定に保った 3 層の GAT（Graph Attention Network）により疑わしいデータフローの検出を試みた。しかし、予備実験の結果、満足のいく性能は得られなかった。そこで、本課題を克服するために以下の改善候補を検討した。

候補 2（Candidate 2）は候補 1 の課題に対処するため、GAT 層ごとに特徴次元数を段階的に減少させる構造を導入した。この設計により、モデルが悪意ある振る舞いに関連する特徴により集中することを期待した。本候補には、線形に減少させるパターン（Candidate 2a）と、急激に減少させるパターン（Candidate 2b）の 2 種類を実装した。

候補 3（Candidate 3）は、ジャンピングナレッジ（Jumping Knowledge）を用いることで、深層ニューラルネットワークにおいて頻発する特徴消失（Feature Vanishing）の問題に対応しようとするものである。具体的には、各 GAT 層の出力を統合し、それらを用いてマルウェア判定のための特徴を構築する手法である。

候補 4（Candidate 4）は、特徴次元の段階的減少とジャンピングナレッジの両手法を組み合わせた構成であり、両者の利点を同時に取り入れることを試みたものである。

Candidates	Accuracy	Precision	Recall	F1-Score
ver.1	51.69%	50.87%	98.44%	67.08%
ver.2a	**96.61%**	**96.13%**	**97.14%**	**96.63%**
ver.2b	81.25%	74.39%	95.31%	83.56%
ver.3	50.65%	50.33%	100.00%	66.96%
ver.4	90.76%	89.82%	91.93%	90.86%

表 1

表 1 に示した実験結果より、特徴次元を減少させる構造を持つ候補（候補 2 群）が他の候補よりも高い F1 スコアを示すことが確認された。また、Candidate 2a の性能が Candidate 2b よりも優れていたことから、後者の急激な次元削減はモデル性能に悪影響を及ぼした可能性がある。一方、ジャンピングナレッジを導入したアーキテクチャでは性能の顕著な低下が見られた。以上の結果に基づき、最も高い性能を示した Candidate 2a を本研究の最終モデルとして採用した。

Candidate	Accuracy	Precision	Recall	F1-Score
ver 2a-2	96.0%	96.6%	95.3%	95.9%
ver 2a-3	**96.6%**	**96.1%**	**97.1%**	**96.6%**
ver 2a-4	83.2%	80.0%	88.5%	84.1%
ver 2a-5	54.0%	52.1%	98.7%	68.2%
ver 2a-6	50.0%	50.0%	100.0%	66.7%

表 2

表 2 に示した実験結果より、F1 スコアの観点で Candidate 2a-3 が最良の性能を示したことが確認された。層の数が増加するにつれて F1 スコアが低下する傾向が見られ、これは特徴消失（Feature Vanishing）問題の発生を示唆している。また、Candidate 2a-2 の結果からは、層数が不足している場合の性能低下の影響も明らかとなった。これらの実験結果に基づき、最適な構造として Candidate 2a-3 を本研究で採用することとした。

4 解釈性

本フレームワークは、グラフアテンションネットワーク（GAT）のアテンション機構を活用することで説明性を実現している。これにより、研究者は関与する Android API の名称や実行順序などの情報を取得可能となる。この情報を用いて、フレームワークの予測結果の妥当性を検証し、さらに詳細なアプリケーション解析を行うことができる。説明性の根幹は GAT 層のアテンション機構に基づいている。具体的には、GAT 層のアテンション重みを解析することで、グラフ上のどのノードが最も注目されているかを把握できる。アテンション重みが高いノードは結果に対する寄与度が大きいため、重みの最も高いノードを重要ノード（critical nodes）と見なす。各 GAT 層から重要ノードを特定することで、グラフ上に重要経路（critical path）を抽出できる。さらに、ノードは Android API を表現しているため、この経路は API 呼び出し列として解釈可能であり、アプリの動作振る舞いを明らかにする手掛かりとなる。

Node Idx	The Corresponding APIs.	Description
65	Landroid/content/pm/PackageManager; getInstalledApplications	Get the installed App list of the device.
75	Lsun/net/www/protocol/http/HttpURL Connection; connect	Establish connections with external servers.
13	Landroid/media/MediaRecorder; setVideoSource	Configure for screen recording.
11	Lorg/apache/http/client/HttpClient; execute	Execute an HTTP request.

表 3

本提案手法の解析によって得られたデータフローは表 3 に示した。表 3 の結果から、対象アプリケーションには 2 つの疑わしい挙動が存在する可能性が示唆される。第一に、当該アプリケーションは端末にインストールされているアプリ一覧を取得し、その情報を外部サーバへ送信している。第二に、スクリーン録画の準備を行い、外部サーバからの指示を待って録画を開始するような処理が含まれている。実際にこれらの動作が実行されたか否かにかかわらず、得られた証拠は研究者にとって有効であり、さらなる解析の起点となり得る。研究者は、上記で示された API 情報を用いることで、当該アプリケーション内の該当するコードスニペットを特定し、検出結果の妥当性を評価することが可能である。

総括すると、本フレームワークの設計により、研究者はアプリケーション内における疑わしい挙動を特定可能となる。疑わしい挙動はノードとして構成され、各ノードは Android API を表現しているため、研究者は対応する API 情報を取得できる。さらに、この API 情報を活用することで、当該挙動に関する初期的な評価を行うことができる。この一連の過程により、本フレームワークは結果の説明可能性を実現している。本節では、実際のアプリケーションを通じてその説明手法を示した。

5 終わりに

モバイルデバイスの普及に伴い、Android システムはマルウェアの主要な標的となってきた。従来のマルウェア検出手法では、進化する攻撃手法に対して十分な対応ができず、その有効性に限界がある。また、近年注目されている機械学習に基づく検出手法は高い性能を示しているものの、説明性の欠如により予測結果の信頼性に疑念が生じやすく、実運用への導入を妨げる要因となっている。この課題を解決するために、本研究では説明性を備えた Android マルウェア検出フレームワークを提案する。本フレームワークは、アプリケーション内のデータフローを追跡することで悪意ある挙動を検出する。具体的には、まずアプリケーションに対して静的解析を行い、API の使用状況（呼び出し順序や名称など）を抽出する。次に、Skip-Gram モデルを用いて、これらの使用パターンから API の特徴量を学習する。最後に、提案する GAT ベースのモデルを用いて、アプリがマルウェアである確率および関係するデータフローを計算する。また、本フレームワークは、検出結果の説明として疑わしい API 呼び出し列を出力する機能を備えており、研究者による振る舞いの分析や結果の妥当性評価を支援する。実験の結果、本フレームワークは高い性能を示した：Accuracy：96.4%、Precision：97.3%、Recall：95.5%、F1 スコア：96.4%。さらに、具体例を通じて、本フレームワークの説明機能がアプリの疑わしい挙動を理解する上で有効であることを示した。

本フレームワークの今後の発展に向けては、以下の 3 つの方向性が考えられる。

第 1 に、性能最適化の観点から、ニューラルアーキテクチャサーチ（Neural Architecture Search）等の手法を用いて、ハイパーパラメータの構成を適切に調整することで、さらなる精度向上が期待できる。

第 2 に、本手法を C/C++で記述されたネイティブライブラリに適用することが挙げられる。これには、Linux カーネル API の選定、ライブラリに対する SACDDG（Suspicious API Call Data Dependency Graph）の構築、そして元の SACDDG との統合が含まれる。

第 3 に、アプリケーション内部のコンポーネント間通信を SACDDG 内の特殊なエッジとしてモデル化する手法である。これは Wu ら[4]により有効性が実証されており、さらなる検出性能の向上が期待できる。

さらに、本フレームワークの核心的なアイデアは、システムコールに基づいたデータフローの追跡により悪意あるアプリケーションを検出することである。この考え方を他のプラットフォームへ展開する際の最大の課題は、対象プラットフォームにおけるシステムコールの識別とデータフローの抽出である。特に、PC 環境では Android と比較して最適化の自由度が高く、例えばシステムコールをバイナリ内に埋め込む手法などが一般的である。このような最適化は、システムコールの識別をより困難にする可能性がある。したがって、他プラットフォームへの適用可能性と有効性を検証することも、本研究の今後の重要な課題の一つである。

【参考文献】

[1] “Mobile Operating System Market Share Worldwide.” [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/worldwide

[2] “200,000 new mobile banking Trojan installers discovered, double the 2021,” https://www.kaspersky.com/about/press-releases/2023 200000-new-mobile-banking-trojan-installers-discovered-double-the-2021, 2023

[3] J. Senanayake, H. Kalutarage, M. O. Al-Kadri, A. Petrovski, and L. Piras, “Android source code vulnerability detection: A systematic literature review,” ACM Comput. Surv., vol. 55, no. 9, jan 2023. [Online]. Available: https://doi.org/10.1145/3556974

[4] Y. Wu, J. Shi, P. Wang, D. Zeng, and C. Sun, “DeepCatra: Learning flow- and graph-based behaviours for Android malware detection,” IET Information Security, vol. 17, no. 1, pp. 118–

130, 2023.
[5] H. Gao, S. Cheng, and W. Zhang, “GDroid: Android malware detection and classification with graph convolutional network,” Computers & Security, vol. 106, p. 102264, 2021.
[6] R. S. Arslan and M. Tasyurek, “AMD-CNN: Android malware detection via feature graph and convolutional neural networks,” Concurrency and Computation: Practice and Experience, vol. 34, no. 23, p. e7180, 2022.
[7] M. Amin, B. Shah, A. Sharif, T. Ali, K.-I. Kim, and S. Anwar, “Android malware detection through generative adversarial networks,” Transactions on Emerging Telecommunications Technologies, vol. 33, no. 2, p. e3675, 2022.

〈発　表　資　料〉

題　　名	掲載誌・学会名等	発表年月
An Accurate and Robust Deep Learning-Based Stacked Generalization Method for Android Malware Detection	The 17th International Symposium on Pervasive Systems, Algorithms and Networks, I-SPAN 2025	2025 年 1 月
An Explainable Android Malware Detection Framework Based on Graph Neural Network	The 8th International Conference on Mobile Internet Security, MobiSec 2024	2024 年 12 月
深層学習ベースのスタッキングを用いた Android マルウェア検出	情報処理学会　コンピュータセキュリティシンポジウム（CSS: Computer Security Symposium）2024	2024 年 10 月