Computer Security Symposium 2020
26 - 29 October 2020

Androidマルウェアの家系図作成

野村 和也[1,a)]　千葉 大紀[2]　秋山 満昭[2]　内田 真人[1]

概要：Android を標的としたマルウェアの脅威は増加し続けている．Android マルウェアは時間経過とともに種別（ファミリ）が増加するため，その対策のためにはある時点に単にマルウェアを検知するだけでなく，マルウェアの時系列変化を考慮した解析を行う必要がある．そこで本研究では，新たに検知された Android マルウェアが，既存の Android マルウェアやそのファミリとどのような関係にあるか，またそれらが時間経過とともにどう変化してきたかを表現可能な「Android マルウェアの家系図」を自動作成する手法を提案する．実際の Android マルウェア 18,958 個を利用した評価の結果，提案の家系図はマルウェアのファミリ間の時間変化を精度良く表現できることがわかった．

キーワード：Android, マルウェア, 家系図

Creating Family Tree of Android Malware

Kazuya Nomura[1,a)]　Daiki Chiba[2]　Mitsuaki Akiyama[2]　Masato Uchida[1]

Abstract: Android malware has been a growing threat. For an effective countermeasure against Android malware, we need to not only detect the malware at a certain point in time but also analyze the time-series changes of malware, taking into account that the Android malware family will increase over time. Therefore, we propose a new method to automatically create a "family tree" of Android malware that can represent how the newly detected Android malware is related to existing Android malware and its families, and how they have changed over time. Our evaluation using 18,958 actual Android malware shows that our proposed family tree is able to accurately represent time-series changes between malware families.

Keywords: Android, Malware, Family tree

1. はじめに

Android は全世界で 25 億台以上のデバイスで稼働している OS であり，そのシェアは 2020 年時点で全世界の 85% に到達する [1]．世界に普及した Android は，当然攻撃者の標的になっており，2018 年時点で，日々 35 万件以上の新種の Android マルウェアが発見されている [1]．

Android マルウェアの脅威が深刻化する中，Android マルウェアを対象とした検知技術が多数提案されてきた．PC を対象とするマルウェアと同様に，Android マルウェアも時間経過とともに攻撃手法やその種別（ファミリ）が増加する．そのため，ある時点の検知性能だけでなく，その時系列変化を考慮することの重要性が指摘されている [2]．また，単純に Android マルウェアかどうかを特定するだけでなく，そのマルウェアと攻撃キャンペーンや攻撃のトレンドとの関連付けまで行うことが重要になってきた [3]．

そこで本研究では，複数の Android マルウェア間の先祖にあたるものから子孫にあたるものまでの関係や，その時系列変化を表現することができる，いわば「家系図」を自動的に作成する手法を提案する．この家系図を得ることで，新たに検知された Android マルウェアが，過去のマルウェアやファミリとどのような関係にあるのかを認識することができる．さらに，ある時点以後に出現する既存のファミリとの関係が得られない「突然変異」のような新種のファ

[1] 早稲田大学
Waseda University
[2] NTT セキュアプラットフォーム研究所
NTT Secure Platform Laboratories
[a)] kazuya-1997@asagi.waseda.jp

ミリを即座に特定することができる．提案手法で得られる上記の情報は，日々変化し続ける Android マルウェアによる攻撃の検知や，その攻撃のキャンペーンやトレンドの把握に有用である．

本研究の貢献は次の通りである．

- 新たに検知された Android マルウェアが，既存の Android マルウェアやそのファミリとどのような関係にあるか，またそれらが時間経過とともにどう変化してきたかを表現可能な「Android マルウェアの家系図」を自動作成する手法を提案した．
- 実際の Android マルウェア 18,958 個を対象に提案手法で家系図を作成した結果，その家系図が Android マルウェアのファミリ間の関係を精度良く表現しており，また新種ファミリの出現を正しく検知できることを確認した．

2. Android マルウェア

2.1 Android マルウェアの概要と解析手法

代表的な Android マルウェアには，アドウェアや迷惑なアプリケーション（PUA），トロイの木馬，ランサムウェアが存在する．攻撃者は，容易に検知されるのを回避するための仕組みを用意している．例えば，良性アプリケーションをデコンパイルし悪性なコードを埋め込み再コンパイルすることで検出を回避するリパッケージや，コードの難読化のような仕組みがある [4]．

Android マルウェアの解析手法は動的解析と静的解析に大別できる．動的解析は，実際にマルウェアを動かして動作をモニタリングし，解析する手法である．例えば，CPU 使用率やプロセスなど OS の動作をモニタリングする手法や，通信先のようなネットワークをモニタリングする手法がある．この手法は難読化に強く，マルウェアの動作が正確にわかる一方，十分に動作させないと満足な解析結果が得られないことが多い．特に Android マルウェアの場合，マルウェアの動作に画面上での操作を必要とする場合が多く解析が難しい．

静的解析は，マルウェアを動作させずにわかる情報で解析する手法である．例えば，Android アプリケーションが要求するパーミッションに着目する手法や，Java もしくは Android SDK で提供される API コールに着目する手法がある．難読化されたコードの解析や，実際の通信などの観測が難しい一方，実際に動かす手間がなく高速で解析，検出が可能である．

2.2 Android マルウェアの検知手法

Android マルウェアを検知する手法としては，既知のマルウェアのシグネチャを基に検知する手法だけでなく，機械学習を利用した検知手法が多く提案されてきた．具体的には，前節の解析手法で得られる特徴量を利用して二値分類を行う研究 [5], [6], [7] や，マルウェア内のファミリ分類を行う研究 [4], [8] がある．

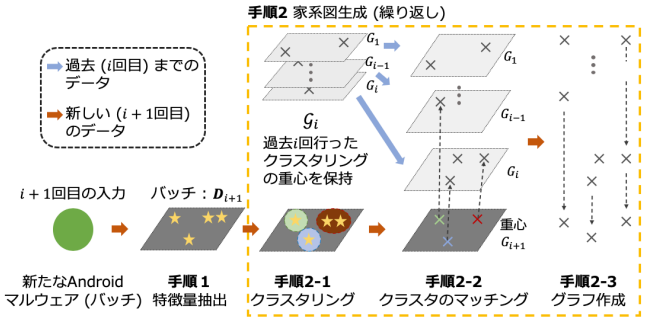

図 1　提案手法の概略図

Android マルウェアのように時間経過とともに増加し続けるものを対象にして機械学習を適用する場合，ある時点の検知性能だけでなく，時系列変化を考慮したロバストな検知性能が重要である．機械学習において，このように予測対象のデータの性質が大きく変化することはコンセプトドリフトと呼ばれる [2]．例えば，「マルウェアのファミリ名を予測する」タスクの場合，学習データに存在しない新種のファミリが出現した時点でコンセプトドリフトが発生し，正確な予測ができない．Feargus ら [9] は，Android マルウェアの検出を行う代表的な先行研究がコンセプトドリフトを考慮しておらず，時系列に基づいた現実的な実験条件で検出アルゴリズムを学習すると，検出精度が低下することを示している．

3. 提案手法

本研究では Android マルウェアのトレンドや新種の出現の観測，また過去の解析済みのマルウェアとのマッチングを実現するために，時系列変化に基づいた Android マルウェアの家系図を作成する手法を提案する．家系図は Android マルウェア間の関係を時系列で示し，悪性の動作や目的の近いファミリを結びつけるものと定義する．提案手法は，図 1 に示すように，Android マルウェアを入力し，2 つの手順（特徴量抽出，家系図作成）を経て，最終的に家系図を作成する．

3.1 手順 1：特徴量抽出

本研究では長期間に渡って収集される，多数の Android マルウェアについて家系図を作成する．そこで，高速に実行可能な静的解析に基づいて家系図作成に必要な特徴量を抽出する．具体的には，本研究では静的解析ツールである MobSF [10] を用い，マルウェアから特徴量を静的解析により抽出する．本研究ではマルウェアの「家系」を特定するため，マルウェアの動作や構造の特徴を捉えられるような特徴量を選定する．具体的には，Android マルウェアが要求するパーミッションとソースコードのパッケージ名，

クラス名の 2 つに着目する．

3.1.1 パーミッション

パーミッションは，Android アプリケーションの動作のために，ユーザに要求する様々な権限を示す．パーミッションによる特徴量は Android マルウェアの動作を明確に示すため，過去の研究でも多く用いられている．Arora ら [11] は，ロバストな Android マルウェア検出には，パーミッション情報の活用が重要であることを示している．

Android マルウェアが要求するパーミッションは，マニフェストファイル (`AndroidManifest.xml`) から取得できる．例えば，Android マルウェアがインターネットへのアクセスを要求する場合，`android.permission.INTERNET` というパーミッションが XML ファイルに記述される．本研究ではこのパーミッションの記述について，以下の操作を行い特徴量として用いる．

(1) ピリオドで区切り，出現したパーミッション名をトークン化する
(2) ある Android マルウェアが要求するパーミッションについて， トークンが出現したら対応する One-Hot ベクトルを立てる
(3) ただし，要求するパーミッションの内容に関係なく，極めて多く出現するトークン `android` と `permission` については除外する

3.1.2 ソースコードのパッケージ名・クラス名

Android アプリケーションは，ソースコードとそのディレクトリ構造 (Java におけるパッケージの構造) も含めて逆コンパイルが可能である．本研究では，このパッケージ名とクラス名 (ファイル名) を特徴量に用いる．例えば，逆コンパイルした結果，`com.example.myfirstapp.MainActivity(.java)` のようなクラスとパッケージの名前空間が得られる．この逆コンパイルした結果得られるパッケージの名前空間に着目すると，慣習的につけられる国別の TLD，ライブラリの開発者，クラス名など様々な情報が得られる．共通するライブラリや開発者名などのドメインを抽出し，より多くの共通項を持つマルウェアを家系図上でつなげるため，この特徴量を用いる．例えばアドウェアでは，共通のライブラリが使用されていることが考えられる．また，マルウェアの開発者が流用したパッケージやクラスなど，マルウェアの作成のされ方についての情報も抽出できる可能性がある．本研究ではこのパッケージ構造の名前空間に着目し，以下の操作を行い特徴量として用いる．

まず，Android SDK が提供する API に着目し，マルウェアの動作に関係するような処理を行なっているクラスについて，そのパッケージ構造とクラス名を取り出す．Aefer らの研究 [5] で示されているように，Android マルウェアの悪性活動には，共通して使われる API が多く存在する．本研究ではそのような API コール 49 種類を選定し，使用する．例えば，ソースコード中で `java.lang.Runtime` クラスの `getRuntime().exec()` や `getRuntime()` が呼び出されていれば，このソースコードは Android OS の上でプロセスをフォークし，コマンドを実行している可能性がある．本研究ではこれらの抽出に，MobSF [10] を用いた[*1]．この処理は，逆コンパイルして得られたソースコードのファイル一つ一つについて行う．

次に，得られた名前空間をピリオドで区切り，パーミッションと同様にトークン化する．ある Android アプリケーションから抽出されたパッケージ名・クラス名のトークンについて，トークンが出現したら対応する One-Hot ベクトルを立てる．ただし，以下のトークンについては除外する．

(1) トークン `android` と `com` については，一般的に出現するトークンであり，Android アプリケーション固有の情報を抽出するという意味で適さないため除外する
(2) 英文字 1 文字のものは特に難読化されている可能性が高く (`b.b.a.a.a.java` など)，こちらも得られるトークンが Android アプリケーションの内容に紐づいている可能性が低いので除外する

このようにして得られる各 Android アプリケーションの特徴量をデータ $\mathbf{D}$ とする．$\mathbf{D}$ は行列形式で定義する．各行が一つの Android アプリケーションに対応し，各列が一つの特徴量に対応する．以降，$\mathbf{D}$ の各行はマルウェアの出現順に並んでいるものとする．なお，トークンをベクトル化する手法は，新しいトークンが出現した場合モデルをすべて再計算する必要がある．本研究では取得するトークンを固定した場合に作成される家系図の評価実験も行う．

3.2 手順 2：家系図作成

本節では，Android アプリケーションから特徴量を抽出した後，家系図を作成する方法について説明する．家系図は図 1 に示すようにクラスタリング，クラスタのマッチング，グラフ作成の 3 つの手順を繰り返して作成される．この家系図の作成の基本的なアイデアは，日々新たに得られるまだ十分に解析されていない Android マルウェアを，過去に得られたマルウェアと関連付けていくというものである．2 節で説明した通り，Android マルウェアはこれまで知られているファミリではない新種のファミリが時間経過とともに出現する．そのため，すでに決まったクラスへの分類を考える教師あり学習のアプローチではなく，教師なし学習のアプローチをとる．具体的には，過去の世代のクラスタリング結果と，新しい世代のクラスタリングの結果をマッチングさせる手順を繰り返すことにより家系図を作成する手法を提案する．

[*1] すべての抽出のルール： https://github.com/MobSF/Mobile-Security-Framework-MobSF/blob/master/StaticAnalyzer/views/android/rules/android_apis.yaml

3.2.1 手順 2-1：クラスタリング

まず，3.1 節で説明した方法で生成した特徴量に基づいて，時系列順に得られる複数のマルウェアのクラスタリングを行う．このクラスタリングには，最適なクラスタ数を自動決定できる X-means 法 [12] を用いる．以降，データ $\mathbf{D}$ の各行，すなわち各マルウェアの特徴量を行ベクトルで表したものを「サンプル」と呼ぶ．各サンプルは One-Hot ベクトルで表されている．また，データ $\mathbf{D}$ を重複のないように時系列順（つまり行番号順）に行単位で m 分割した部分行列 $\mathbf{D}_1, \mathbf{D}_2, \cdots, \mathbf{D}_m$ を「バッチ」と呼ぶ．バッチは複数のマルウェアの特徴量を行列で表したものとなる．

バッチ $\mathbf{D}_i$, $(i \in \{1, 2, \cdots, m\})$ の行ベクトル全体からなるサンプルの集合に対して X-means 法を用いてクラスタリングし，c_i 個のクラスタが得られたとする．各 c_i 個のクラスタの重心 $\mathbf{g}_{i,l}$, $(l \in \{1, , 2, \cdots, c_i\})$ を，そのクラスタに属するサンプルの特徴量を表す行ベクトルの平均で定義する．また，クラスタの重心の集合を $G_i = \{\mathbf{g}_{i,1}, \mathbf{g}_{i,2}, \cdots, \mathbf{g}_{i,c_i}\}$，その和集合を $\mathcal{G}_i = \bigcup_{j=1}^{i} G_j$ と表す．集合 $\mathcal{G}_i$ はバッチ $\mathbf{D}_1, \mathbf{D}_2, \cdots, \mathbf{D}_i$ に対してクラスタリングを行ったときに生成されるクラスタの重心をまとめたものとなる．

3.2.2 手順 2-2：クラスタのマッチング

新たにバッチ $\mathbf{D}_{i+1}$ が得られたとする．$\mathbf{D}_{i+1}$ を新たにクラスタリングし，c_{i+1} 個のクラスタと重心の集合 G_{i+1} を得る．ここで，過去のクラスタリング結果から，現在のクラスタリングの結果に近いものを探す．k 近傍法を用い，新たに得られた重心の集合 G_{i+1} の要素である c_{i+1} 個の重心 $\mathbf{g}_{i+1,1}, \mathbf{g}_{i+1,2}, \cdots, \mathbf{g}_{i+1,c_{i+1}}$ のそれぞれについて，その近傍を $\mathcal{G}_i$ の要素から k 個ずつ探索する．k 近傍法の距離関数にはユークリッド距離を用いる．以上の操作を「マッチング」と呼ぶ．マッチングの結果，$\mathbf{D}_{i+1}$ のあるクラスタの重心 $\mathbf{g}_{i+1,l}$, $(l \in \{1, 2, \cdots, c_{i+1}\})$ について，$\mathcal{G}_i$ から探索した k 個の近傍を要素にもつ集合 $Neighbor_{i+1,l}$ と，それらの k 個の近傍と $\mathbf{g}_{i+1,l}$ との距離を要素に持つ集合 $Distance_{i+1,l}$ が得られる．なお，探索する近傍の個数 k は任意であるが，最初のバッチ $\mathbf{D}_1$ に対するクラスタに対して 2 回目のクラスタをマッチングする場合，最初のクラスタの個数以上の近傍を得ることができない．そのため k は $\mathbf{D}_1$ に対するクラスタの個数 c_1 とする．

3.2.3 手順 2-3：グラフ作成

家系図 $P = (N, E)$ を有向グラフにより定義する．N はノードを示し，クラスタの重心の集合 $\mathcal{G}_i$ の要素と一対一で対応する．E はエッジを示し，距離 d を保持する．エッジの方向は，時間の方向を表す．

$\mathbf{D}_{i+1}$ から求められた c_{i+1} 個のクラスタの重心 $\mathbf{g}_{i+1,l}$ に対応するノード $n_{i+1,l}$ を作成する $(l \in \{1, 2, \cdots, c_{i+1}\})$．このノードに，$Neighbor_{i+1,l}$ の要素である過去の重心と対応するノードをエッジで結ぶ．ただし，距離の閾値 θ をパラメータとし，θ を超える場合はエッジの接続を切る．

表 1 評価実験で用いた Android マルウェア

ファミリ	Android マルウェアの個数
Airpush	7,843
Dowgin	3,356
FakeInst	2,168
Mecor	1,817
Youmi	1,300
Fusob	1,275
Kuguo	1,199
合計	18,958

この操作により，より近いクラスタのノードが接続され，家系図のエッジとして時系列を保ったクラスタ間の関係を示すことができる．

4. 評価

4.1 データセット

3 節で示した提案手法の有効性を，実際の Android マルウェアを用いて検証した．本研究では，実際の Android マルウェアのデータセットとして，ArgusLab によるデータセット [13] を用いた．このデータセットには Android マルウェアの APK ファイル 24,650 個が含まれ，マルウェアのファミリ名を示す正解ラベルが含まれている．ここで，マルウェアのファミリ名は VirusTotal [14] の検出結果を基に与えられ，50%以上のアンチウイルスが検出した結果の内容に基づいてラベル付けが行われている．

本評価実験において，データセット中で用いた Android マルウェアのファミリと個数を表 1 に示す．このデータセットは本来 71 個のファミリをもつが，ファミリに含まれるサンプルの数が非常に少なく，家系図が示す家系を長期間にわたって評価するのには適さないファミリが多く存在する．そこで本実験では，データセットの中で 1,000 個以上のサンプルを持つファミリのマルウェアをすべて抽出し，7 つのファミリのマルウェア，合計 18,958 個について，提案手法による家系図の作成を行った．

なお，このデータセットに含まれるのは Android マルウェアの APK ファイルとその正解ラベルのみで，本研究で着目する時系列変化を追うことができない．そこで，VirusTotal が提供する API を利用し，各 APK ファイルが初めて観測された日付を取得した．その結果，表 1 に示した 18,958 個の Android マルウェアは 2011 年 9 月 26 日〜2016 年 3 月 1 日までの期間に各々はじめて観測されたことが判明した．各 7 種類のファミリのマルウェアについて，縦軸にサンプル数，横軸に年月を示したグラフを図 2 に示す．各色のグラフはファミリごとに累計の出現回数を示し，グラフ中の横軸は年月，縦軸は出現回数を示す．

この図から，期間によって，データセット中に出現するマルウェアのファミリや割合が異なることが観測できる．また，本データセット中のマルウェアファミリは，データセット中全期間に渡って出現するファミリだけでなく，ある日時以降にのみ出現する「新種」が存在することもわかる．

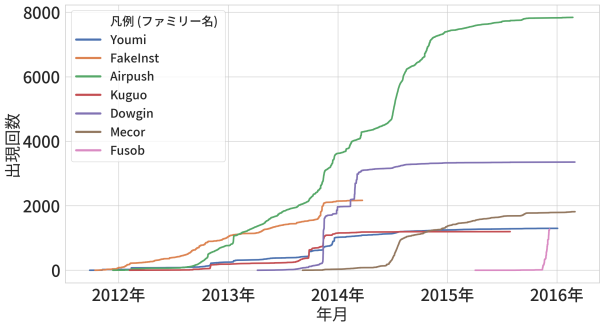

図 2　マルウェアファミリの時間変化による出現の様子

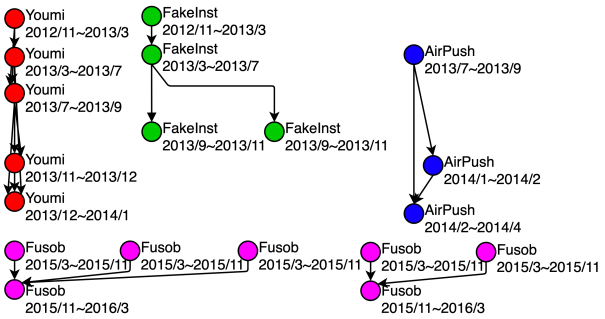

図 3　作成される家系図の例 (一部抜粋)

4.2　家系図の作成

3 節の提案手法で示した手順に従い，データセットから特徴量抽出を行い，家系図を作成した．ただし，計算効率を向上させるため，特徴量は 10 個以上のマルウェアに共通して出現するパーミッション，及び 80 個以上のマルウェアに出現するパッケージ・クラス名のトークンのみを用いた．以降の実験ではマルウェアのバッチを，4.1 節で取得した時系列順に 1,000 個ずつ分割して，評価を行った．また，各ノードとノードに紐つく重心が示すクラスタを同値と見なす．

提案手法において距離の閾値 θ を 1.5 に設定し，作成した家系図の一部を図 3 に示す．グラフは下方向に進むに従って，時間が進むことを示す．ノードの色は，そのノードに対応するクラスタ内で最多のファミリを示す．家系図では図のように関連するマルウェアのクラスタが同じ家系で結ばれ，エッジを逆向きに辿ることで過去のマルウェアとの関連付けも容易に行える．

4.3　クラスタリングの評価

クラスタリング結果や，家系図を作成した結果の妥当性について評価を行う．まず，家系図を作成する前の一つ一つのクラスタについて，クラスタリングによるファミリ分類の精度評価を行う．精度評価には，データセット中に含まれる正解ラベルのファミリ名を用いる．また，以下 2 つの指標を用いてクラスタリングの精度を評価する．

最多ファミリの割合の平均：各クラスタ内における最多ファミリが占める割合の平均を示す．このスコアは直感的であるが，クラスタのサイズや混同したファミリの数を考慮することができないという欠点がある．

表 2　クラスタリングのスコア

評価指標	スコア ± 標準偏差
最多ファミリの割合の平均	0.973± 0.006
homogeneity_score	0.784± 0.047

homogeneity_score：上記の欠点を補うため，homogeneity_score [15] を用いてクラスタリングの精度評価を行う．n 種類のクラスラベル $C = \{c_1, c_2, \cdots, c_n\}$ を持つ N 件のデータを，m 個のクラスタ $K = \{k_1, k_2, \cdots, k_m\}$ に分けたとき， homogeneity_score h は以下のように定義される．

$$h = \begin{cases} 1, & H(C|K) = 0 \\ 1 - \frac{H(C|K)}{H(C)}, & \text{otherwise} \end{cases}$$

ここで，H はエントロピーを示し，クラス c，クラスタ k に含まれるデータの個数を $a_{c,k}$ とすると，

$$H(C|K) = -\sum_{k=1}^{|K|}\sum_{c=1}^{|C|}\frac{a_{c,k}}{N}\log\frac{a_{c,k}}{\sum_{c=1}^{|C|}a_{c,k}}$$

$$H(C) = -\sum_{c=1}^{|C|}\frac{\sum_{k=1}^{|K|}a_{c,k}}{N}\log\frac{\sum_{k=1}^{|K|}a_{c,k}}{N}$$

と定義される．homegeneity_score では，すべてのクラスタが単一のファミリのみを含む場合，スコアが高くなり 1 に近づく．

表 2 に，クラスタリングを 10 回行った時の，上記の 2 つの指標のスコアの平均と標準偏差を示す．表より，各クラスタに含まれるファミリ名の 97% が同一のファミリ名であり，homogeneity_score も 0.784 と高い値になっていることがわかる．以上より，家系図の作成に必要なクラスタリングが，教師なし学習により安定した精度で行えたことがわかる．

4.4　家系図の精度評価

マルウェアのファミリや動作を家系図から推定するためには，家系図の示す家系が同一のファミリで占められていることが望ましい．本節では，このような「家系」が妥当に示されているかを評価する．

家系図において，グラフの向きにかかわらず，「ノード同士が接続しているか」を考慮する木構造を考える．以降，この部分グラフを本研究では「家系」とする．ノードがつながっている家系を一つの大きなクラスタとみなし， その大きなクラスタについて正解ラベルのファミリ名を用いて精度の評価を行い，家系図の妥当性を示す．同時に，家系図の家系を結ぶ距離の閾値を変化させたときの精度の変化を評価する．評価指標には homogeneity_score を用いる．

一方，グラフの枝刈りをする距離の閾値と家系図の精度は，トレードオフの関係にある．各クラスタを結ぶエッジ

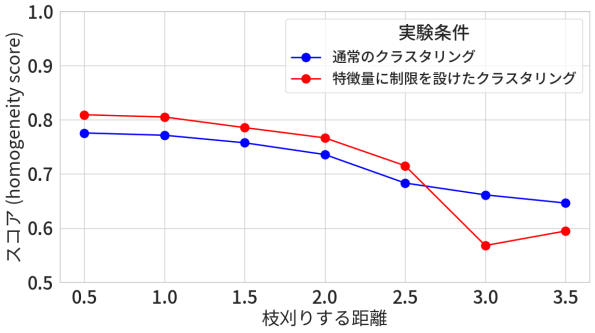

図 4　家系図の精度評価

を枝刈りする距離の閾値を大きくすると，グラフ中の家系が示す一つのクラスタはより大きくなる．しかし，関係の弱いクラスタ同士がエッジで結ばれる可能性も大きくなり，精度の悪化が懸念される． 本節ではこのトレードオフも評価する．

図 4 に，枝刈りをする距離の閾値による精度の変化を，グラフの青色の線で示す． (赤色の線についての評価は後の 4.5.1 節で行う．) グラフの横軸は家系図の枝刈りをする閾値，縦軸は homogeneity_score を示す．グラフから，家系図上の家系をクラスタとしてみなしても，4.3 節で示した単純なクラスタリングの精度と比較して大きく変わらないことがわかる．つまり，家系図上で接続されているクラスタは，ファミリの分類という観点で実際に妥当な接続となっており，新たなクラスタが属するマルウェアのファミリを推定することもできることがわかる．

グラフのエッジを結ぶ距離のトレードオフについて考察する．枝刈りする距離が短ければ，より近いクラスタ同士が家系図上でつながり，過去のサンプルとの対応づけが簡潔になる．一方で，枝刈りする距離が長ければより多くのクラスタ同士がつながり，過去のサンプルを参照できるノードが増える．また，近縁種などの示唆を多く得ることができる．なお，グラフのノード同士の関係の近さについては，枝刈りを行わずに，それに対応するクラスタ間の距離によって連続的に判断することもできる．本研究の以下の実験においては，距離の閾値 1.5 で評価を進める．

4.5　コンセプトドリフト発生時の評価

データセット内で観測される新種のマルウェアファミリが出現（これを以後コンセプトドリフトと呼ぶ）した際の，提案手法のロバスト性を評価する．続いて，提案手法を用いたコンセプトドリフト検知により，教師あり学習によりマルウェアのファミリを分類する機械学習モデルの精度低下が防げることを示す．

4.5.1　コンセプトドリフトに対するロバスト性の評価

提案手法では，特徴量抽出において，出現しうるパーミッション・ライブラリ名やクラス名を列挙し，出現したものをトークンとして One-Hot ベクトルを生成する．しかし，SDK，Android API のバージョンアップや，ライブラリ自体のアップデートにより，出現しうるトークンが変化し，モデルの再計算が必要になることが考えられる．

この変化を考慮するために，データセット内のマルウェアが出現した期間中，ちょうど半数のマルウェアが出現した 2013 年 12 月 19 日以前のマルウェアについて，その時までに出現したトークンを用いて One-Hot ベクトルを生成した．特徴量の抽出以外は 4.2 節と同様の実験を行う．ただし，特徴量は 5 個以上のマルウェアに共通して出現するパーミッション，及び 40 個以上のマルウェアに出現するパッケージ・クラス名のトークンのみを用いた．

4.2 節と同様，図 4 に，閾値による家系図の精度の変化を示すグラフを赤線で示す．本図より，4.2 節で示した青線のグラフと比較しても，精度の大きな変化は観測できず，特徴量の抽出に制限を加えても，精度への影響が少ないことがわかった．この理由として，マルウェアが既存のコードやパッケージを再利用して作成されていることが考えられる．また，マルウェアの動作にあたって必要不可欠なパーミッションが存在し，それらが大きく変わらないことも考えられる．後者については，Arora らの研究 [11] でも，パーミッションに注目することの有効性が述べられている．実際には，Android API のメジャーバージョンアップが 1 年に 1 回程度あるため，そのタイミングを考慮し，特徴量やモデルを再計算することが妥当である．

4.5.2　コンセプトドリフト検知の評価

時系列に基づいて家系図を作成することのメリットとして，コンセプトドリフトをいち早く観測することができ，機械学習モデルの精度低下にも対応できることが挙げられる．この評価として，あるマルウェアが与えられたときどのファミリに属するのかを予測する，多クラス分類の精度評価を行った．具体的な実験手法としては，データを日時順に並べ替え，訓練用 1,000 個，テスト用 1,000 個ずつをずらしながら RandomForest モデルの学習・評価を 18 回行うことで評価を行った．また，モデルの学習には提案手法と同じ特徴量を用いた．

提案手法によるコンセプトドリフトの検知手法として，家系図内のノードについて，各バッチで過去のどのノードとも結びつかないノード (以下，新ノードとする) が出現したとき，そのノードに属するマルウェアの個数が一定個数以上の場合再学習するモデルを評価する．比較として，以下の 5 つのモデルを評価する．

- 毎回再学習するモデル (全学習)
- 新ノードに属するマルウェアの個数が，25 個を超えた時のみ再学習するモデル (25 個スキップ)
- 新ノードに属するマルウェアの個数が，50 個を超えた時のみ再学習するモデル (50 個スキップ)
- 2 回に 1 回再学習するモデル (半分学習)

表 3　教師あり学習モデルの学習回数と精度の評価

実験	Accuracy (全 18 回の平均)	学習のスキップ回数
全学習	0.960	0 回
25 個スキップ	0.961	5 回
50 個スキップ	0.961	7 回
半分学習	0.903	9 回
初回のみ学習	0.491	17 回

表 4　リパッケージマルウェアに対するクラスタリングのスコア

評価指標	スコア ± 標準偏差
最多ファミリの割合の平均	0.704± 0.026
homogeneity_score	0.870± 0.024

- 最初の 1,000 個のみ学習するモデル (初回のみ学習)

各実験における全 18 回の Accuracy の平均を表 3 に示す．また，表には最終的にモデルの学習をスキップした回数も示す．表より，以下のことがわかる．まず，毎回再学習するモデルに対し，学習回数を無闇に減らすと，モデルの精度が低下することがわかる．特にモデルを長期間学習しないと，新種の分類ができず，精度が著しく低下する．一方，提案手法に基づいてコンセプトドリフトを検知する手法 (25 個スキップ，50 個スキップ) は学習回数が少なく，毎回学習するモデルに対して精度の遜色がない．つまり，提案手法によりコンセプトドリフトを検出して正しいタイミングで再学習できることがわかる．

4.6　リパッケージマルウェア

リパッケージにより生成されたマルウェアについて，クラスタリングの精度評価を行った．データセット中には，過去の解析結果から正規アプリケーションのリパッケージにより生成されることがわかっているファミリが 24 個あり，そのファミリに属するマルウェア 1,963 個が含まれている．それらを抽出し，クラスタリングによる分類精度の評価を行った．4.2 節で示した手順に従って算出したクラスタリングのスコアを表 4 に示す．

リパッケージにより生成されたマルウェアは，本物の良性な Android アプリケーションの機能やコードを含むことから Android マルウェアと区別しにくく，一般的に検出や分類が難しいとされている．表 4 より，提案手法による教師なし学習においては，精度は落ちるもののリパッケージマルウェアの分類が可能であることがわかった．

4.7　ケーススタディ

提案手法により得られた家系図を実際に辿ることによりわかる情報を，ケーススタディとして紹介する．

4.7.1　新種マルウェアファミリ出現の観測

家系図の作成により，新種マルウェアファミリの出現が確認できることを示す．図 2 に示すマルウェアのファミリごとの検出数の推移に注目すると，特に「Fusob」ファミリが 2015 年以降に短期間で多数現れていることがわかる．図 3 に示す家系図において，下側に存在するピンク色のノードが Fusob ファミリである．このピンク色のノードは，あるバッチから突然出現しており，家系を作っていた．このように，提案の家系図を利用することで，あるバッチの時点で過去のマルウェアファミリとは関連が低い新種のファミリが出現したことを知ることができる．例えば，4.5.2 節に示した教師あり学習モデルによるコンセプトドリフト検知の評価において，自動的に作成された家系図によりこの「Fusob」の出現が検知され，その後モデルが再学習されることでモデルの精度が大きく向上したことを確認できた．

表 5　家系図上で生成された家系の例

主なファミリ	家系の名前	家系に属するマルウェアの個数
Youmi	Youmi.a	190
Youmi	Youmi.b	5
Youmi	Youmi.c	5
Youmi	Youmi.d	15
FakeInst	FakeInst.a	15
FakeInst	FakeInst.b	60
FakeInst	FakeInst.c	47

4.7.2　家系図上のマルウェアの評価

家系図が示すマルウェアの「家系」について，どのような示唆が得られるかを調査した．提案手法で作成した家系図中でも特に繋がっているノードの多かった「Youmi」，「FakeInst」について，家系図の中身を手動で調査した．手動調査の際には，VirusTotal API を利用して動的解析結果を取得し，Android マルウェアの実際の挙動を確認した．

それぞれのファミリを主に含むクラスタの家系の例を表 5 に示す．表には生成された家系に主に含まれるファミリ，説明に用いる家系の名前，及びその家系に含まれるマルウェアの個数を示す．

Youmi：マルウェアファミリ Youmi は，本データセット中全期間にわたって出現している PUA のファミリである．提案手法にて作成された家系図において，Youmi を含む家系の例として 4 つの家系（Youmi.a，b，c，d）を取りあげる．各家系に属するマルウェアの外部への通信挙動を確認すると，各家系に含まれる各々のマルウェアが通信するドメイン名の組合せが大きく異なることがわかった．これは同じファミリ名が付与されるマルウェアであっても，その通信挙動は異なり，その特性を提案手法で作成した家系図が正しく捉えていることを意味している．

FakeInst：マルウェアファミリ FakeInst は，他の Android アプリケーションのインストーラに偽装したマルウェアの一種である．FakeInst を含む家系の例として 3 つの家系（FakeInst.a，b，c）を取りあげる．Youmi の事例と同様に，各々の家系によって通信先のドメイン名の組合せが異なることが確認できた．

以上のように，同じ正解ラベルを持つファミリ内でも異なる通信先を持つマルウェアを，家系図上の木である「家系」を辿ることにより，特定できることがわかる．また，以上のケーススタディから，家系図の作成により近縁種などの関係を把握できることがわかる．

5. 関連研究

マルウェアの家系図を作成する研究として，Oyen らの研究 [16] が挙げられる．この研究ではベイジアンネットワークを使用し，DAG (有向非巡回グラフ) により家系図を作成している．ただし，グラフの向き (親子関係) は，人から与えられた「ヒント」に基づき決定される．一方，本研究では時系列順にクラスタリングを実施し，グラフの向きを自動的に決定する．また，Oyen らの研究では個々のマルウェアファミリについて亜種等を見出し，局所的な家系図を作成している．一方，本研究では複数のファミリを含むデータセットについて，長期的で大域的な分析を行う．その結果，新しいファミリの出現を観測できることも示した．

また，階層型クラスタリングによりマルウェアの類似度をデンドログラムで可視化する研究として，Erd'elyi ら [17] や，Ardimento ら [18] の研究が挙げられる．これらの研究ではデンドログラムによる系統図が得られるが，階層化は類似度によって行われており，時系列情報を反映しているわけではない．

コンセプトドリフトの検知をする研究としては，Pendlebly ら [9] や Jordaney ら [19] の研究が挙げられる．これらは機械学習により Android マルウェアを検知するタスクにおいて，コンセプトドリフトを検出する研究である．しかし，本研究においてコンセプトドリフト検出は，家系図作成による副次的な効果である．時系列変化に基づいてマルウェアの家系図を作成し，他のマルウェアサンプルとの関連からファミリや動作を推定する本研究の目的とは異なる．

クラスタリングにより Android マルウェアの分類を行う最新の研究として，Zhang らの研究 [1] が挙げられる．この研究は VirusTotal の検出結果から真のラベルを推定する研究であり，VirusTotal の検出結果以外にもマルウェアのソースコードの類似度，及びパーミッションなどマニフェストの情報を用い，深層学習でベクトルを生成しクラスタリングを行っている．しかし，時系列によるマルウェアの変化を検出したり，過去に遡って検出結果を辿ったりすることはできない．

6. 終わりに

本研究では Android マルウェアの時系列変化に基づいた家系図の作成手法を提案し，実際の Android マルウェア 18,958 個を用いた実験により，家系図作成の妥当性と効果を示した．本研究で提案するマルウェアの時系列変化に対応した家系図を作成することにより，マルウェアのトレンドの把握やファミリ推定など，多くの示唆が得られる．実際に運用する場合，日々収集されるマルウェアを効率よく解析，分類し，様々な情報と紐つけて脅威情報として活用されることが期待される．今後の展望として，時系列変化やリパッケージ，難読化によりロバストな特徴量の模索や，より大規模かつ直近のデータを含むデータセットでの実験の準備を行っている．

謝辞 本研究の一部は，日本学術振興会における科研費（20K11800）の助成を受けている．また，本研究を進めるにあたり，NTT セキュアプラットフォーム研究所の福士直翼氏に多くの助言を頂いた．ここに謝意を表す．

参考文献

[1] Zhang, Y. et al.: Familial Clustering for Weakly-Labeled Android Malware Using Hybrid Representation Learning, *IEEE Trans. Information Forensics and Security* (2020).
[2] Lu, J. et al.: Learning under Concept Drift: A Review, *IEEE Trans. Knowledge and Data Engineering* (2019).
[3] Tounsi, W. and Rais, H.: A survey on technical threat intelligence in the age of sophisticated cyber attacks, *Computers & Security* (2018).
[4] Tam, K. et al.: The evolution of android malware and android analysis techniques, *ACM CSUR* (2017).
[5] Aafer, Y. et al.: DroidAPIMiner: Mining API-Level Features for Robust Malware Detection in Android, *Security and Privacy in Communication Networks* (2013).
[6] Arp, D. et al.: DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket, *Proc. NDSS* (2014).
[7] Mariconti, E. et al.: MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models, *Proc. NDSS* (2017).
[8] Mirzaei, O. et al.: AndrEnsemble: Leveraging API Ensembles to Characterize Android Malware Families, *Proc. ACM AsiaCCS* (2019).
[9] Pendlebury, F. et al.: TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time, *Proc. USENIX Security* (2019).
[10] MobSF. https://github.com/MobSF/Mobile-Security-Framework-MobSF.
[11] Arora, A. et al.: PermPair : Android Malware Detection Using Permission Pairs , *IEEE Trans. Information Forensics and Security* (2019).
[12] Pelleg, D. and Moore, A. W.: X-means: Extending K-means with Efficient Estimation of the Number of Clusters, *Proc. ICML* (2000).
[13] Wei, F. et al.: Deep Ground Truth Analysis of Current Android Malware, *Proc. DIMVA* (2017).
[14] VirusTotal. https://www.virustotal.com/.
[15] Rosenberg, A. and Hirschberg, J.: V-Measure: A Conditional Entropy-Based External Cluster Evaluation Measure, *Proc. EMNLP-CoNLL* (2007).
[16] Oyen, D. et al.: Bayesian Networks with Prior Knowledge for Malware Phylogenetics, *Proc. AAAI Workshop: Artificial Intelligence for Cyber Security* (2016).
[17] Erd'elyi, G. and Carrera, E.: Digital genome mapping: ad-vanced binary malware analysis, *Proc. Virus Bulletin Conference* (2004).
[18] Ardimento, P. et al.: Malware Phylogeny Analysis using Data-Aware Declarative Process Mining, *Proc. IEEE EAIS* (2020).
[19] Jordaney, R. et al.: Transcend: Detecting Concept Drift in Malware Classification Models, *Proc. USENIX Security* (2017).