Vol.2021-DPS-186 No.25
Vol.2021-CSEC-92 No.25
2021/3/15

IoTマルウェアの分類における画像化を用いた手法とシステムコール列を用いた手法の比較

イボット アリジャン[1,a)]　大山 恵弘[1]

概要：近年，IoT の流行により IoT デバイスを狙ったマルウェアが増加している．これに伴い，増加したマルウェアを正しく把握するために，IoT マルウェアの正しい分類が求められている．一方で多くの IoT マルウェアはシンボルが削除された静的リンクを使用しており，リンクされた関数の情報を用いた分類が困難であることが知られている．この問題を解決するために，リンクされた関数の情報を用いない手法が複数提案されている．それらの中では，最近提案されたマルウェアを画像化する手法が，システムコール列の利用などをする従来手法と比較して高い精度が得られると報告されている．しかし画像化手法を提案する多くの研究では，全検体を使用した時の精度のみを比較対象としており，従来手法との詳細な比較が行われていない．本研究では IoT マルウェアの分類を対象に，画像化を用いた手法と，従来手法であるシステムコール列を用いた手法の詳細な比較を行う．

キーワード：IoT マルウェア, 分類, 画像化, システムコール, 比較

Comparison between Image Based and System Call Based Methods for Classification of IoT Malware

Ibot Arijan[1,a)]　Yoshihiro Oyama[1]

Abstract: Recently, with the popularization of IoT, the number of malware targeting IoT devices is increasing. Consequently, correct classification of IoT malware is needed to understand the increasing malware correctly. However, many IoT malware programs are statically linked and symbol stripped, and hence it is well-known that classifying them by information of linked functions is difficult. To solve this problem, several methods have been proposed that do not use the information of linked functions. In those methods, a recently proposed method that imaging malware is reported to be more accurate than previous methods that use system call sequences. However, most of those papers only compared the accuracy using all collected malware samples and did not compare them in detail. In this paper, we compare in detail the imaging-based method and the previous method that uses system call sequences by the classification of IoT malware.

Keywords: IoT malware, classification, imaging, system call, comparison

1. はじめに

近年，IoT デバイスの普及に伴いセキュリティが貧弱な IoT デバイスが増加し，それを狙ったマルウェアが増加している．Nokia 社のセキュリティ調査レポート [1] では，2020 年にモバイルネットワーク上で観測された感染デバイスの 32.72%を IoT デバイスが占めており，2019 年の 16.17%から約 2 倍になっていると報告されている．このように増加し続ける IoT マルウェアの動向を把握するために，マルウェアを正しく分類する手法が必要とされている．

今まで Windows マルウェアを対象としたファミリー分類の研究は多く行われてきた [4, 5, 8, 10, 11]．しかし IoT マルウェアは，今まで研究されてきた Windows マルウェアと異なる点がある．それは多くの IoT マルウェアが静的

[1] 筑波大学
University of Tsukuba
[a)] ibot@syssec.cs.tsukuba.ac.jp

Vol.2021-DPS-186 No.25
Vol.2021-CSEC-92 No.25
2021/3/15

リンクでありながらシンボル情報が削除されており，解析が難しいという点である．静的リンクとは実行ファイルにライブラリ関数などを埋め込む手法である．Windows では OS が提供するライブラリ関数の多くが `ntdll.dll` や `kernel32.dll` などの DLL (Dynamic Link Library) ファイルで扱われており，完全に静的リンクを実現することは難しい．一方で IoT デバイスによく使われる Linux では静的リンク用のライブラリが用意されていることが多く，容易に静的リンクを実現できる．

静的リンクされたファイルの実行領域には，マルウェア本体とリンクされたライブラリ関数のプログラムが混在している．一部の検体ではシンボル情報として，リンクされたライブラリ関数名などをファイル内に記録しているが，シンボル情報はファイルサイズの削減やアンチリバースエンジニアリングを目的として削除されることも多い．実際に我々がマルウェア共有サイト VirusShare [2] から取得した IoT マルウェアのうち，95%以上が静的リンクされた検体であり，そのうち半分以上がシンボル情報を削除されていた．

静的リンクかつシンボル情報が削除されたファイルを解析しても，何の関数がリンクされたか判断することは難しい．そのため今まで Windows マルウェアで行われていた，リンクされたライブラリ関数を用いたマルウェア分類手法 [9] などが，多くの IoT マルウェアでは使えないという問題がある．リンクされたライブラリ関数を特定する手法も提案されている [16, 17] が，特定の状況やアーキテクチャにしか対応することが出来ない．

この問題を解決する手法として，リンクされた関数の情報を必要としない手法がいくつか知られている．バイナリに存在する命令列の N-gram を用いた手法 [8] や，静的解析によって得られた各システムコール命令の数を用いた手法 [6] などが挙げられる．

近年，リンクされた関数の情報を必要としない新たな手法としてマルウェアの画像化を用いた手法 [10] が提案されている．これはマルウェアのファミリー分類問題を画像分類問題として扱うことで，画像認識分野の技術をマルウェア分類に応用しようとする試みである．従来の手法と比較して，画像化を用いた手法は高い精度でマルウェアの分類に成功したと報告されている．

また，画像化したマルウェアの分類を行う機械学習アルゴリズムに深層学習を用いた手法 [11, 12] も提案されており，従来の機械学習アルゴリズムを用いた場合と比較して，より高い精度が得られたと報告されている．

画像化手法を提案する多くの研究論文では，従来手法と比較する際に，従来手法の論文に書かれている精度か，最新のデータセットを従来手法に適用して得られた精度を使用している．機械学習においては，学習用データセットとなるマルウェア検体の質や量が精度に与える影響は大きく，学習に用いたマルウェア検体が異なる場合の精度比較は，正しい結果が得られない可能性がある．また，大量の検体を含む最新のデータセットを従来手法に適用し，その精度と比較することが行われる一方で，従来手法の実験で使用されるような少数のデータセットに対し，画像化手法がどのような精度を出すかは明らかにされていない．

本研究では IoT マルウェアの分類を対象に，画像化を用いた手法と，従来手法である各システムコールの命令数を用いた手法について詳細に比較を行う．また，画像化を用いた手法でも機械学習アルゴリズムに深層学習を使った場合と，深層学習を使わない場合で別々に実験を行った．マルウェアの数に応じた複数の比較条件を用意し，各条件下で上記 3 つの手法を比較し，各手法の特徴を考察する．

2. 関連研究

システムコールやカーネル API コールを利用してマルウェアの検知・分類をする研究が行われている．Warrender らの研究 [4] では動的解析によって得られたシステムコール列を用いたマルウェアの検出手法を提案している．また Kolosnjaji らの研究 [5] ではシステムコール列と深層学習を用いたマルウェア分類手法を提案している．彼らの手法では Windows マルウェアをサンドボックス上で実行し，マルウェアのトレースによって得られたシステムコールを利用している．本研究ではこのような動的解析を用いた手法は対象としない．画像化を用いた手法は静的解析に分類されるため，同じく静的解析を用いた手法を比較する．

Bai らの研究 [6] では静的解析によって得られるシステムコールを使用したマルウェア検知手法を提案している．彼らの研究では動的リンクされた IoT マルウェアを対象に，リンクされた関数からマルウェアが使用するシステムコールを計算している．しかし近年の IoT マルウェアは静的リンクされた検体が大半であり，彼らの手法は直接利用できないことが多い．本研究では静的リンクされたマルウェアを対象に，静的解析でバイナリ内のシステムコールを直接検出してマルウェアの分類に使用する．

システムコール命令以外の静的解析によって得られる特徴を使った分類手法として，バイト列と N-gram を用いた手法がある．Schultz らの研究 [7] ではマルウェア内のバイト列を N-gram で表現し，マルウェア検出に利用している．同様に Kolter らの研究 [8] でも同様にバイト列の N-gram を用いてマルウェアの検出・分類を行っている．バイト列の N-gram は動的リンクや静的リンクを問わずに特徴量として扱えるが，全てのバイト列を特徴量として使うため計算コストが高い．本研究では計算コストがより低い，各システムコール命令の数を用いた手法を比較対象とする．

Sami らの研究 [9] では，Windows マルウェアの IAT (Import Address Table) を用いたファミリー分類手法を提案している．IAT は Windows プログラムが実行時に必要と

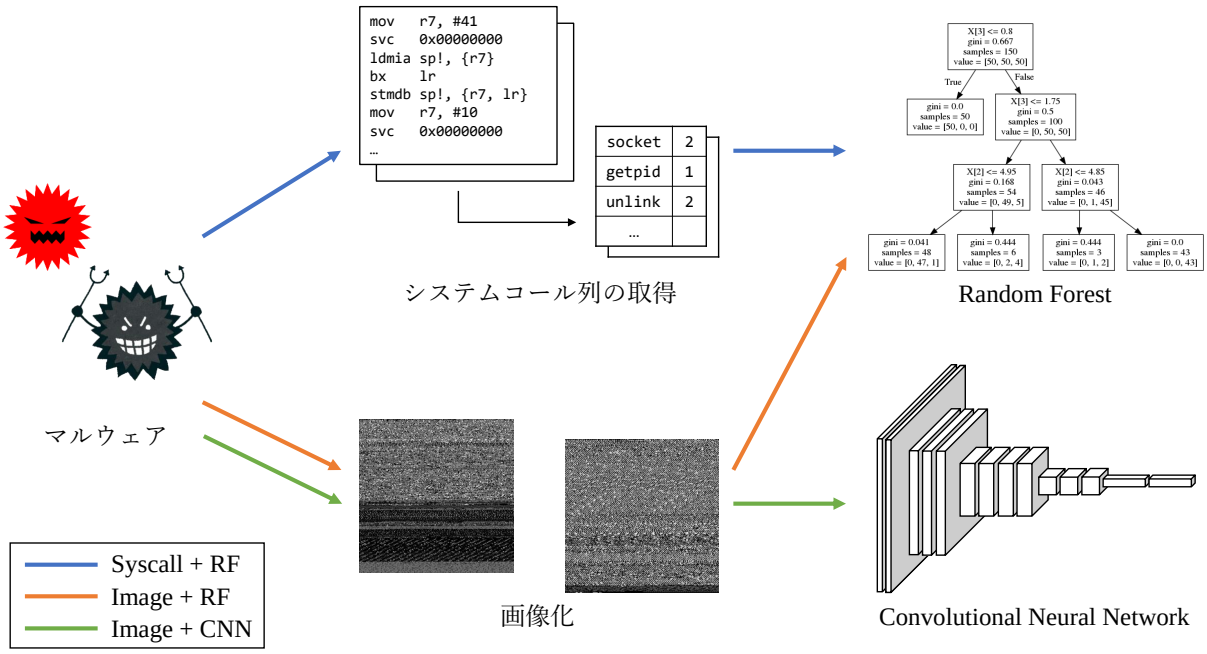


図 1: 本研究で比較する各マルウェア分類手法の概要

する外部 API を記録する領域である．マルウェアの IAT を確認することでマルウェアの目的を推定し，ファミリーの分類を行う．IoT マルウェアにおいても動的リンクされた検体であれば彼らの手法を使えるが，本研究が対象とする静的リンクされたマルウェアでは IAT が存在しないため，彼らの手法を使うことは難しい．

Nataraj らの研究 [10] では Windows マルウェアのバイナリをグレースケール画像に変換し，既存の画像分類手法を用いてマルウェアファミリーの分類を行っている．この手法は難読化やパッキングされたマルウェアにも効果があると報告されている．

画像化したマルウェアの分類に深層学習を使う手法も提案されている．Kalash らの研究 [11] では，画像化した Windows マルウェアの分類に CNN (Convolutional Neural Network) を使用し，従来の機械学習アルゴリズムを使用した画像化手法と比較して高い精度が得られたと報告している．また，矢倉らの研究 [12] では CNN に注意機構を加えることで，画像化した Windows マルウェアの分類をより高精度にしている．

画像化を用いた手法は Windows マルウェアだけでなく IoT マルウェアにも有効である．Su らの研究 [13] では画像化した IoT マルウェアを対象として，ファミリーの分類を行っている．計算機資源が貧弱な IoT デバイスを狙ったマルウェアは軽量なプログラムであることが多く，Windows マルウェアの分類に用いた CNN よりコンパクトなモデルでも高い精度でマルウェアの分類ができたと報告している．Huang らの研究 [14] では画像化した Android マルウェアを対象としてマルウェアの検出を行っている．彼らの手法ではバイナリをグレースケール画像ではなくカラー画像に変換している．

このようにバイナリの画像化を用いたマルウェアの検知・分類は多く行われている．本研究ではこれら画像化を用いたマルウェア分類手法と，静的解析によって得られる各システムコール命令の数を用いたマルウェア分類手法の詳細な比較を行う．

3. 比較する手法

本研究では 3 つの手法を比較する．各マルウェア分類手法の概要を図 **1** に示す．また，本研究では Arm アーキテクチャの IoT マルウェアを対象とする．

3.1 Syscall + RF

この手法では，静的解析で得られた各システムコール命令の数と Random Forest を用いてマルウェア分類を行う．

静的解析では，以下の手法でシステムコールを検出する．Arm のシステムコールは `svc` 命令によって実行される．システムコールを実行する `svc` 命令のオペランドには，システムコール番号か `0` が与えられる．どちらが与えられるかは Arm のバージョンに依存する．また，`svc` 命令のオペランドに `0` が与えられた場合 `r7` レジスタの値がシステムコール番号として扱われる．

システムコールを検出した後，次のようなアルゴリズムでシステムコール番号を取得する．マルウェアを逆アセンブルし `svc` 命令と `svc` 命令のオペランドを取得する．オペランドが `0` でない場合，その値をシステムコール番号として扱う．オペランドが `0` であった場合 `svc` 命令の前に実行

Vol.2021-DPS-186 No.25
Vol.2021-CSEC-92 No.25
2021/3/15

される命令を 5 つ取得し，svc 命令実行時の r7 レジスタの値を推定する．そしてこれをシステムコール番号として扱う．稀に r7 レジスタの値が推定できない場合もあったが，その頻度は全推定のうち 1%未満だった．そのため推定に失敗した場合は，その svc 命令を無視して進める．

最新の Arm 版 Linux ではシステムコールが 397 個存在するため，各システムコール命令の数を 397 次元の特徴ベクトルで表現することが出来る．マルウェア検体から得られた特徴ベクトルを，決定木ベースの機械学習アルゴリズムである Random Forest に与えて学習を行う．

3.2 Image + RF

この手法では画像化したマルウェアと Random Forest を用いて分類を行う．**Syscall + RF** と同じ機械学習アルゴリズムを使用したのは，マルウェアの特徴量を各システムコール命令の数からバイナリの画像に変えた場合の違いを明確にするためである．

マルウェアの画像化アルゴリズムを **Algorithm 1** に示す．本研究ではマルウェアのバイナリを正方形のグレースケール画像に変換する．まずマルウェアのファイルサイズに応じた正方形の画像を作成し，1 バイトを 1 ピクセルとしたグレースケール画像に変換する．マルウェア毎にファイルサイズが違うため，異なるサイズの画像が生成される．

Random Forest では入力サイズを統一する必要がある．マルウェアの画像は全て正方形で作成されるため，拡大縮小によってアスペクト比を維持したままリサイズすることができる．リサイズでは大きい画像に合わせると特徴量を維持することができるが，学習に必要な計算コストが大きくなる．一方で小さい画像に合わせると特徴量が減ってしまい精度が下がってしまう可能性がある．本研究では全ての画像を 224×224 にリサイズする．224×224 というサイズは画像認識でよく用いられる CNN である VGG16 [15] というモデルの入力として使われるサイズである．

3.3 Image + CNN

この手法では画像化したマルウェアと CNN を用いて分類を行う．**Image + RF** の機械学習アルゴリズムを Random Forest から CNN に変えただけであるため，画像化アルゴリズムは割愛する．また，CNN でも入力サイズを統一する必要があるため，Random Forest を用いた場合と同様に，全ての画像を 224×224 にリサイズする．CNN のモデルには Kalash らの研究 [11] と同じく VGG16 と呼ばれる 16 層の CNN モデルを使用する．

4. 実験

マルウェアの検体数を 500 ずつ増やした時の各手法の精度を取得する．機械学習のランダム性を考慮し，同じ検体数で 5 回実験を行い，その精度の平均値を扱う．

Algorithm 1 マルウェアの画像化アルゴリズム

Require: $binary$
Ensure: $image$
  $s \leftarrow \textsc{Sqrt}(\textsc{Sizeof}(binary))$
  $image[s][s] \leftarrow 0$
  for $y = 0, \cdots, s-1$ **do**
    for $x = 0, \cdots, s-1$ **do**
      $image[y][x] \leftarrow binary[y \times s + x]$
    end for
  end for
  $image \leftarrow \textsc{Resize}(image, (224, 224))$
  return $image$

表 1: 実験に用いたマルウェアのファミリー

ファミリー名	検体数	割合 (%)
Mirai	3120	34.67
Gafgyt	2134	23.71
Lightaidra	1658	18.42
Occamy	1335	14.83
Yakuza	304	3.38
DemonBot	235	2.61
Adload	193	2.14

4.1 実験に用いたマルウェア

マルウェア共有サイト VirusShare [2] から，2016 年以降に確認されたマルウェアを収集した．実験に使用した全マルウェアの検体数は 8999 であった．

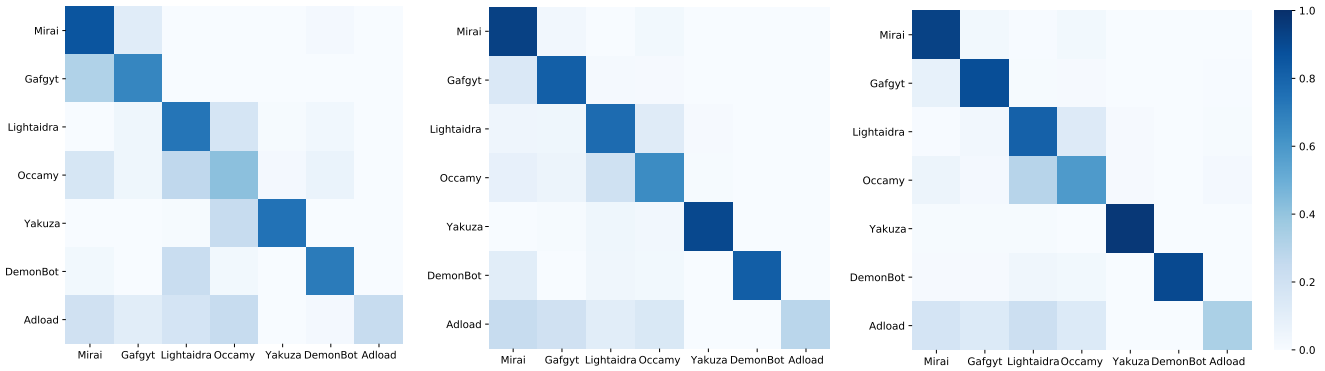
マルウェアファミリーのラベリングには VirusTotal [3] の Microsoft の判定結果を使用し，50 検体以上存在するファミリーを取得した．ファミリーの内訳を**表 1** に示す．全検体の 8 割弱を占める Mirai，Gafgyt，Lightaidra は，いずれもボットネットを構築する IoT マルウェアであり，類似した機能を持っている．そのためリンクされたライブラリ関数も似通ったものが多く，IoT マルウェアの分類は難しい問題だと推測できる．

実験では静的リンクされた検体のみを使用した．その理由として，静的リンクされていない検体ではシステムコール命令の取得が難しいという点が挙げられる．システムコール命令の多くはライブラリ関数内で実行される．静的リンクされていない検体では，多くの使用するシステムコールを外部ライブラリに依存しており，実行ファイルから各システムコールの命令数を正確に取得できない．また，我々が収集した全検体のうち静的リンクされた検体は 95%以上を占めており，静的リンクされていない検体は非常に少ないことがわかる．これらの理由から，本研究では静的リンクされていない検体を対象外とした．

また，実験では Arm アーキテクチャの検体のみを使用した．その理由として，異なるアーキテクチャでは生成される画像の特徴が大きく異なるためである．もし複数のアー

表 2: 各手法のまとめと精度

	特徴量	機械学習アルゴリズム	特徴ベクトルのサイズ	精度
Syscall + RF	各システムコールを実行する命令の数	Random Forest	397×1	0.715
Image + RF	バイナリを画像化したマルウェア	Random Forest	$224 \times 224 \times 3$	0.811
Image + CNN	バイナリを画像化したマルウェア	CNN (VGG16)	$224 \times 224 \times 3$	0.844



(a) **Syscall + RF** (b) **Image + RF** (c) **Image + CNN**

図 2: 各手法による分類結果のヒートマップ

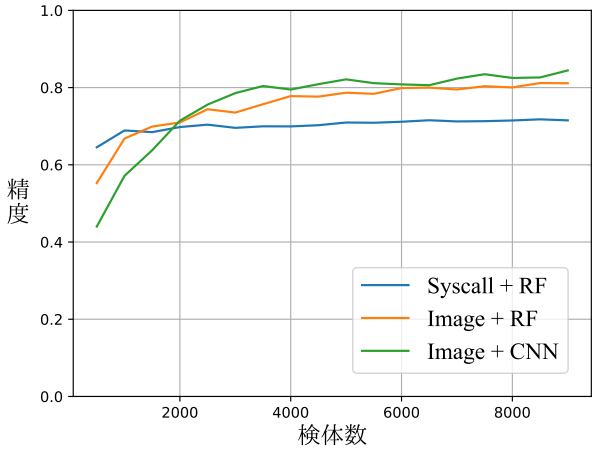


図 3: 検体数による精度の推移

キテクチャの検体を用いた場合，機械学習アルゴリズムはアーキテクチャ毎の特徴パターンを学習することになる．しかし各アーキテクチャで検体数が異なる場合，アーキテクチャ間で精度に差が生じる可能性がある．例えば，検体数が多いアーキテクチャでは学習が進み，高い精度で分類できたが，検体数が少ないアーキテクチャでは学習がうまく進まず，精度が低くなる事などが考えられる．実験では検体数を増やしていった時の精度を調べるため，アーキテクチャを統一しないと正しい結果が得られない．

4.2 実装

機械学習アルゴリズムに与える検体は，訓練用に 7 割，テスト用に 3 割として実験を行った．また，CNN を使用する手法では訓練データの 1 割を検証データとして使用した．バイナリの逆アセンブルには `objdump-2.17` を使用し，命令列からレジスタの値を推定するには `radare2-4.4.0` を使用した．CNN の作成には `TensorFlow-2.4.1` と `Keras-2.4.3` を使用した．CNN のモデルは VGG16 [15] を使用し，エポック数は 25，最適化関数は SGD，バッチサイズは 6 に設定した．

5. 評価

各手法のまとめと，全検体を用いて実験を行った時の精度を**表 2** に示す．検体数を 500 ずつ増やしたときの各手法の精度とその推移を**図 3** に示す．全検体を用いた際の各手法の混同行列をヒートマップ化した結果を**図 2** に示す．縦軸が真のファミリーであり，横軸が推測されたファミリーである．真のファミリーを元に，推測されたファミリーの比率を色で表現した．各部分の青色の濃さは，その部分に対応する推測が行われた検体数を表しており，濃ければ濃いほど検体数が多いことを示している．

約 9000 個の全検体を用いた場合 **Image + CNN** の精度が最も高く，先行研究と同じ結果が得られた．しかし検体数が少ない場合，先行研究とは異なる結果が得られた．検体数が 500 または 1000 のとき，最も精度が高いのは **Syscall + RF** であった．画像化を用いた手法 2 つが，**Syscall + RF** の精度を超えたのは検体数が 2000 以上のときであった．

Syscall + RF は検体数が 1000 以上のとき，検体数を増やしても大きな精度の向上は見られなかった．検体数が 1000 のとき精度は 68.9%であり，検体数が 8999 になっても精度は 71.5%で，その差は約 2%ほどであった．そのかわり，検体数が 500 であっても精度が大きく減ることはなく，64.5%を維持していることが確認できた．このことか

Vol.2021-DPS-186 No.25
Vol.2021-CSEC-92 No.25
2021/3/15

ら **Syscall + RF** は，検体数が少ない場合でも精度が大きく減少しないかわりに，検体数を増やしても精度の向上が少ないということがわかった．

Image+RF は，検体数が 1500 以上の時に **Syscall+RF** より精度が高くなった．しかし検体数が 500 のときの精度は 55.3%であり **Syscall+RF** と比較して 9.2%低かった．一方で全検体を用いたときの精度は 81.1%であり **Syscall + RF** と比較して 9.6%高かった．**Image+CNN** は検体数が 2000 以上の時，全手法の中で最も精度が高かった．しかし検体数が 1500 以下のときは最も精度が低く，検体数が 500 のときの精度は 43.9%であった．このことから画像化を用いた手法 2 つは，検体数が少ないときは精度が低いものの，検体数を増やしていったときの精度の上昇量が大きく，最終的には **Syscall + RF** より約 10%ほど精度が高くなることがわかった．

分類結果をヒートマップ化した図 2 を見ても，画像化を用いた手法の精度が高いことがわかる．Mirai と Gafgyt は全検体の半分以上を占めるファミリーでありながら，非常に似通った機能を持っているため分類が難しい．そして **Syscall + RF** では Mirai と Gafgyt の分類がうまくできておらず，互いに誤分類が多く発生していることがわかる．これは **Syscall + RF** が各システムコール命令の数を用いた手法であり，類似した機能を持つマルウェアの分類を苦手とするからと考えられる．一方で **Image + CNN** は高い精度で Mirai と Gafgyt の分類に成功していることがわかる．Mirai と Gafgyt が同じくボットネットを構築するマルウェアであっても，Mirai の方が攻撃に使えるプロトコルが多く，辞書攻撃用のデータベースを持っていたりするため，その違いを画像認識アルゴリズムが検知したと考えられる．

6. 考察

実験に用いた検体数がマルウェア分類を行う機械学習アルゴリズムに与えた影響について考察する．

6.1 学習するパラメータの数

機械学習アルゴリズムでは学習するパラメータが多いほど，より複雑な分類が可能で精度が高くなると考えられている．一方で入力となるデータ数が少ないと学習がうまく進まず，逆にパラメータを減らしたほうが精度が高くなることもある．深層学習は中間層と呼ばれる領域でパラメータを大量に確保し，正しい結果を出せるようにパラメータの調整を進めていくアルゴリズムである．実験で使用した VGG16 というモデルは 16 の層から構成される CNN であり，中間層には約 1 億 4000 万個のパラメータが存在する．

一方で Random Forest は決定木ベースのアルゴリズムである．入力データのある要素に対して閾値を設け，データの 2 分化を続けることで分類を行うアルゴリズムである．

表 3: 分類において重要度の高いシステムコール

(a) 検体数 500 のとき

名前	重要度 (%)
accept	5.58
chdir	5.43
recvfrom	5.30
unlink	5.20
prctl	5.18
access	3.89
times	3.73
pipe	3.73
getcwd	3.60
listen	3.48

(b) 検体数 8999 のとき

名前	重要度 (%)
recvfrom	6.43
times	6.41
accept	6.35
chdir	6.01
prctl	4.82
unlink	4.49
bind	3.89
getppid	3.77
listen	3.35
readlink	3.10

ここでデータを 2 分化するための閾値を 1 つのパラメータとして捉えると，検体数が 8999 のとき **Syscall + RF** のパラメータ数は約 600，**Image + RF** のパラメータ数は約 2000 であった．また，検体数が 500 のとき **Syscall + RF** のパラメータ数は約 150，**Image + RF** のパラメータ数は約 110 であった．

6.2 検体数と精度の関係性

十分な検体数を確保できなかった場合，深層学習アルゴリズムはパラメータの調整に必要なデータが足りず，精度が低くなると予想される．実際に本研究で行った実験では，検体数が 1500 以下のとき **Image + CNN** が最も低い精度となった．**Image + RF** においても入力データの特徴量が多く，どの特徴量を閾値に設定するか学習が進まなかったと推測される．これは検体数が 500 のとき **Image + RF** より **Syscall + RF** の方が，分類を行う閾値が多く設けられているからである．この理由として **Syscall + RF** は，ヒューリスティックにバイナリから特徴量を抽出しているからだと考えられる．本来データ数が増加すれば機械学習アルゴリズムは自ら特徴量を選択するが，データ数が十分でない時はそれが成立しない．よってヒューリスティックに特徴量を選択して，それを入力とした方が高い精度が得られるということである．

表 3 は **Syscall + RF** が決定木を作成する際に，どのシステムコールが閾値として設定されやすいかを重要度として示したものである．検体数が 500 であっても 8999 であっても重要度の高いシステムコールはほぼ共通しており，検体数が少なくとも重要度の高いシステムコールを正しく検出できることを示している．

一方で十分なデータ数を確保できた場合，パラメータの数が最も多い深層学習アルゴリズムが一番高い精度を出せるはずである．実際に本研究で行った実験では，検体数が 2000 以上のとき **Image + CNN** が最も高い精度でマルウェアを分類できた．また，同じ Random Forest を用い

表 4: VirusShare から収集したマルウェアのアーキテクチャ

アーキテクチャ	検体数	割合 (%)
Arm	16538	29.13
x86	12288	21.65
MIPS	9337	16.45
PowerPC	4229	7.45
SuperH	4142	7.30
Motorola 68k	4026	7.10
SPARC	3099	5.46
x86-64	2790	4.91
(others)	320	0.56

た手法であっても **Image + RF** のほうが入力データの特徴量が豊富で，分類を行う閾値が多く設けられたため高い精度が得られたと考えられる．

6.3 IoT マルウェアの対象アーキテクチャと検体数

画像化によるマルウェアの分類の精度は検体数に大きく左右されるため，十分に多くの検体を入手する必要があるということがわかった．ここで注意するべき点として，画像化手法の機械学習アルゴリズムはアーキテクチャ毎に独立して学習を行うという点である．アーキテクチャが異なる場合，生成される命令列が全く異なり，命令列を含むバイナリから生成される画像も全く異なるものとなるからである．特に Windows マルウェアと違い IoT マルウェアは対象アーキテクチャが多様であるため，注意が必要である．

我々が VirusShare から収集したマルウェアの比率を**表 4** に示す．本研究では最も検体数が多かった Arm アーキテクチャのマルウェアを使用したが，他のアーキテクチャを用いる際は検体数に気をつける必要がある．例えば VirusShare から取得できる x86-64 の IoT マルウェアの検体数は，Arm の約 17%である．今回実験に用いた Arm の検体数は 8999 であるため，x86-64 で同様の実験を行う場合は約 1500 検体しか入手できないと考えられる．この場合，画像化を用いた手法の学習がうまく行かず，各システムコール命令の数を用いた手法より精度が低くなる可能性がある．これは検体数が少ない macOS を狙ったマルウェアなどでも同様である．

7. おわりに

本研究では IoT マルウェアの分類を対象に，画像化を用いた手法と各システムコール命令の数を用いた手法で比較を行った．また，その際に検体数を少しずつ増やしたときの精度の推移を調査した．その結果，マルウェアの検体数が十分に存在するとき，画像化を用いた手法はシステムコールを用いた手法と比較して約 10%精度が高いことがわかった．一方でマルウェアの検体数が十分に存在しないとき，画像化を用いた手法はシステムコールを用いた手法と比較して 10〜20%精度が低いこともわかった．これは機械学習アルゴリズムが学習を必要とするパラメータの数と，入力データにおける特徴量の数が影響していると考えられる．実験では IoT マルウェアで最も多いアーキテクチャの Arm を対象として実験を行ったが，IoT デバイスに少ないアーキテクチャや macOS を狙ったマルウェアなど，検体が少ない環境で画像化を用いた手法を使用するときは，学習データが足りているか注意する必要がある．

謝辞 本研究の一部は JSPS 科研費 20K11741 の助成を受けている．

参考文献

[1] Nokia, “Threat Intelligence Report 2020”, https://onestore.nokia.com/asset/210088
[2] VirusShare.com, https://virusshare.com
[3] VirusTotal, https://virustotal.com
[4] Warrender, C., Forrest, S., Pearlmutter, B.: Detecting Intrusions Using System Calls: Alternative Data Models, *IEEE Symposium on Security and Privacy* (1999), pp. 133-145
[5] Kolosnjaji, B., Zarras, A., Webster, G., Eckert, C.: Deep Learning for Classification of Malware System Call Sequences, *Australasian Joint Conference on Artificial Intelligence* (2016), pp. 137-149
[6] Bai, J., Yang, Y., Mu, S., Ma, Y.: Malware Detection Through Mining Symbol Table of Linux Executables, *Information Technology Journal 12(2)* (2013), pp. 380-384
[7] Schultz, M. G., Eskin, E., Zadok, F., Stolfo, S. J.: Data Mining Methods for Detection of New Malicious Executables, *IEEE Symposium on Security and Privacy* (2001), pp. 38-49
[8] Kolter, J. Z., Maloof, M. A.: Learning to Detect and Classify Malicious Executables in the Wild, *Journal of Machine Learning Research 7(12)* (2005)
[9] Sami, A., Yadegari, B., Rahimi, H., Peiravian, N., Hashemi, S., Hamze, A.: Malware Detection Based on Mining API Calls, *ACM Symposium on Applied Computing* (2010), pp. 1020-1025
[10] Nataraj, L., Karthikeyan, S., Jacob, G., Manjunath, B. S.: Malware Images: Visualization and Automatic Classification, *International Symposium on Visualization for Cyber Security* (2011), pp. 1-7
[11] Kalash, M., Rochan, M., Mohammed, N., Bruce, N. D., Wang, Y., Iqbal, F.: Malware Classification with Deep Convolutional Neural Networks, *IFIP International Conference on New Technologies, Mobility and Security* (2018), pp. 1-5
[12] Yakura, H., Shinozaki, S., Nishimura, R., Oyama, Y., Sakuma, J.: Neural malware analysis with attention mechanism, *Computers & Security* (2019), Vol.87
[13] Su, J., Vasconcellos, D. V., Prasad, S., Sgandurra, D., Feng, Y., Sakurai, K.: Lightweight Classification of IoT Malware Based on Image Recognition, *IEEE Computer Software and Applications Conference* (2018), pp. 664-669
[14] Huang, T. H., Kao, H. Y.: R2-D2: ColoR-inspired Convolutional NeuRal Network (CNN)-based AndroiD Malware Detections, *IEEE International Conference on Big Data* (2018), pp. 2633-2642
[15] Simonyan, K., Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition, arXiv (2014), https://arxiv.org/abs/1409.1556

Vol.2021-DPS-186 No.25
Vol.2021-CSEC-92 No.25
2021/3/15

[16] Jacobson, E. R., Rosenblum, N., Miller, B. P.: Labeling Library Functions in Stripped Binaries, *ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools* (2011), pp. 1-8
[17] 赤羽 秀, 岡本 剛.: シンボル情報が消去された IoT マルウェアに静的結合されたライブラリ関数の特定, コンピュータセキュリティシンポジウム 2020 論文集, pp. 543-550