卒業論文

IoT マルウェアの画像分類手法への難読化による攻撃と対処の研究

公立はこだて未来大学
システム情報科学部　情報アーキテクチャ学科
情報システムコース　1018179

佐藤 隼斗

指導教員　稲村 浩　/　副指導教員　石田 繁巳

提出日　2022 年 1 月 25 日

BA Thesis

An Attack and Countermeasure using Obfuscator to IoT Malware Image Classification

by

Hayato SATO

Information Systems Course, Department of Media Architecture
School of Systems Information Science, Future University Hakodate
Advisor:　Hirohshi INAMURA　/　Co-advisor:　Shigemi ISHIDA

Submitted on January 25th, 2022

Abstract− The threat of IoT malware is rapidly increasing due to the generation of variants using the malware source codes publicly available. Consequently, image-based malware classification, which utilizes an image reconstructed from malware binary to classify malware, has been attracting a lot of interest. The image-based classification enables us to quickly and accurately analyze the rapid increase of IoT malware. Since the image-based classification is affected by the binary change of malware, we consider the binary changes without the change of semantics of the program as an attack on the image-based classification. In this research, we show the effectiveness of the attack by obfuscations and the possibility of countermeasures. As an attack attempt, the obfuscated malware families mirai, lightaidra and bashlite were all misclassified by an image classifier that was learned using the collected malware. As a countermeasure against the attack method using obfuscator, we conducted training including obfuscated samples and confirmed that it can be classify the samples with an accuracy of more than 85%. Furthermore, it was possible to subdivide the malware family for each obfuscation function and perform OR operations to classify the obfuscated samples with an accuracy close to 100 %.

Keywords: IoT malware, Image-based Malware Classification, LLVM, obfuscation

概 要： 公開されたソースコードを使用した亜種生成により IoT マルウェアが急増している．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，高速・正確に分類可能なマルウェアの画像化による分類手法が注目されている．画像化による分類手法はマルウェアのバイナリ変更の影響を受けるため，プログラムの意味が変わらないバイナリ変更は画像化による分類手法への攻撃手法と考えることができる．本研究では，難読化処理による攻撃の有効性と対処の可能性を示す．収集したマルウェアを用いて評価基準となる画像分類器を作成し，攻撃手法として oLLVM を用いた難読化を加えて生成したサンプルを分類させたところ，攻撃対象のマルウェアファミリである Mirai，Lightaidra，Bashlite は全て誤分類された．難読化処理による攻撃手法への対策として，難読化を施したサンプルを含めた訓練を行ったところ，85% 以上の精度でサンプルを分類可能であることが確認できた．更に，難読化機能毎にマルウェアファミリを細分化し論理和を取ることで，難読化を施したサンプルを 100% に近い精度で分類可能であることを示した．

キーワード： IoT マルウェア, マルウェアの画像分類, LLVM, 難読化

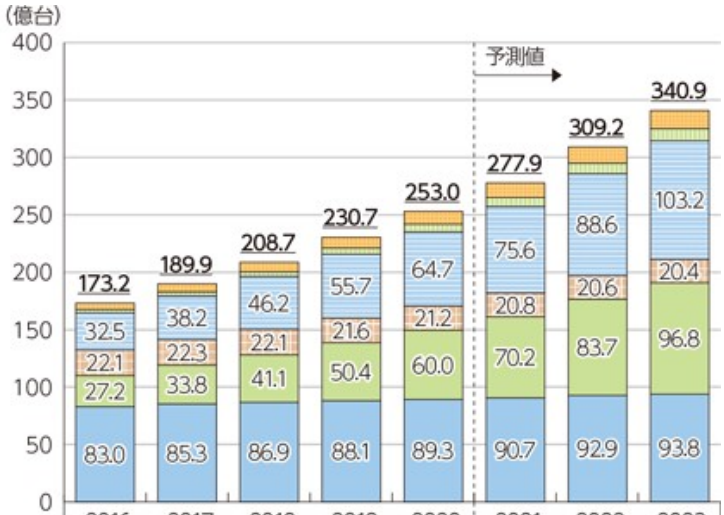
目次

第 1 章

序論

1.1 背景

近年，パソコンやスマートフォンなどの従来のインターネット端末に加えて，家電や自動車，ビルや工場などのあらゆる端末がネットワークに繋がるようになり，IoT デバイスが急速に普及し始めている．総務省によると世界の IoT デバイス数は **図 1.1** に示すように，2020 年時点で約 253 億台存在し，2022 年には約 341 億台まで増加すると予測されている [1].



	2016	2017	2018	2019	2020	2021	2022	2023
合計	173.2	189.9	208.7	230.7	253.0	277.9	309.2	340.9
自動車・宇宙航空	5.7	7.1	8.5	9.8	11.3	12.7	14.2	15.7
医療	2.6	3.3	4.0	5.1	6.5	7.8	9.2	10.9
産業用途	32.5	38.2	46.2	55.7	64.7	75.6	88.6	103.2
コンピューター	22.1	22.3	22.1	21.6	21.2	20.8	20.6	20.4
コンシューマー	27.2	33.8	41.1	50.4	60.0	70.2	83.7	96.8
通信	83.0	85.3	86.9	88.1	89.3	90.7	92.9	93.8

図 1.1: 世界の IoT デバイス数の推移及び予測（令和 3 年版 情報通信白書 [1] より引用）

IoT デバイスを狙ったマルウェアが急増している．多くは公開された主要な IoT マルウェアのソースコードを改変した亜種生成によるものであり，サイバー犯罪者やハッカーがコードを簡単にダウンロード・変更・再コンパイルして新たな亜種を生成できることが問題視されている [2].

センサーやウェブカメラなどの IoT デバイスは，機器の性能が限定されている，管理が行き届きにくい，ライフサイクルが長いなど，サイバー攻撃に狙われやすい特徴を持っている [3]．セキュリティ対策に不備のある IoT デバイスはマルウェアに感染しサイバー攻撃に悪用される恐れがあるため，セキュリティパッチの迅速な更新や機器の管理画面・管理ポートに対する適切なアクセス制限，推測されにくいパスワードの使用などの対策が必要である [3, 4].

IoT デバイスの普及に伴い増加し続ける 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必要がある．人手による解析は時間がかかるため，マルウェアの亜種の増加は解析者の負担を増加させる．この問題に対して，マルウェアファミリの分類が有用である．マルウェアファミリとは，マルウェアのオリジナルと亜種をグループ化したものである．同じファミリに属するマルウェアは機能が類似しているため，新たな解析を行う必要性が低い．解析を行う場合でも，オリジナルや他の亜種の解析結果を参考にできるため，少ない解析で済み解析者の負担軽減ができる．

IoT マルウェアのファミリ分類では，マルウェアの画像化による分類手法が役立つ [5]．多くの IoT マルウェアは静的リンクがなされており，シンボル情報も削除されている [6]．そのため，Windows で行われるようなリンクされたライブラリ関数によるマルウェア分類手法 [7] などを用いることができない．リンクされた関数の情報を使用しない分類手法は複数提案されているが，画像化による分類手法は従来手法と比較して高い精度（正解率）が得られるとの報告 [6] があり，IoT マルウェアの分類に適していると考えられる．

画像化による分類手法は，マルウェアのバイナリ変更の影響を受ける．そのため，プログラムの意味が変わらないバイナリ変更は，画像化による分類手法への攻撃手法と考えることができる．本研究では，このようなバイナリ変更の手段として，ソースコードの難読化処理を検討する．同様のバイナリ変更としてパッカーによる圧縮（パッキング）があるが，ソースコードの難読化処理はパッキングに比べてコード変更や難読化レベル，最適化レベルの変更ができるため自由度が高く，対策が難しいと考えられる．

自動化されたマルウェアファミリの分類手法に対して攻撃を受け有効性が失われた場合，新たに検出された全てのマルウェアの亜種に対してコストのかかる解析を行う必要があり，解析者の負担が増大しマルウェアの急増に対応できなくなる．マルウェアの画像化による分類手法の恒常的な精度向上のためには，上記のような今後想定される攻撃に対する対策を検討しておくことが重要である．

1.2 本研究の目的

本研究の目的は，マルウェアの画像化を用いた分類手法の恒常的な精度向上に資する，新たな攻撃手法とその対処の評価である．本研究で論ずる精度とは分類器の正解率を指す．想定される攻撃に対して事前に対策を講じることで，対策済みの攻撃の誤分類を未然に防ぐことを目指す．

マルウェアの画像化による分類手法とは，**図 1.2** のようにマルウェアを矩形画像に変換し機械学習により分類する手法である．画像認識技術の発展に伴い，訓練された深層学習モデルを流用することで簡単に高い精度を得られるようになった．深層学習による分類では，人間が過去の経験から設計したものよりも適した特徴量を自動的に獲得できる [10]．

画像化による分類手法は逆アセンブルやコード実行をしないため，マルウェア解析の専門知識が必要なく，リンクされた関数の情報も使用せず利点が多い．しかし，バイナリ画像を学習する都合上，プログラムの意味が変わらないバイナリ変更により画像に変化を加えることで分類精度が低下する可能性がある．

本研究では，ソースコードへの難読化処理による画像分類への攻撃手法を提案し，攻撃効果の検証と事前対策の実装・評価を行う．研究の第一段階として，Obfuscator-LLVM [8] を使用して同一のコードから複数の難読化を施したサンプルを生成し，画像分類手法に対して攻撃を行う．目標として，難読化サンプルのファミリ分類を混乱させることを一つの指標とする．難読化処理による攻撃の有効性を確認した後に，対策として難読化されたサンプルを含めた訓練の実装・評価を行う．詳細は第 3 章に示す．

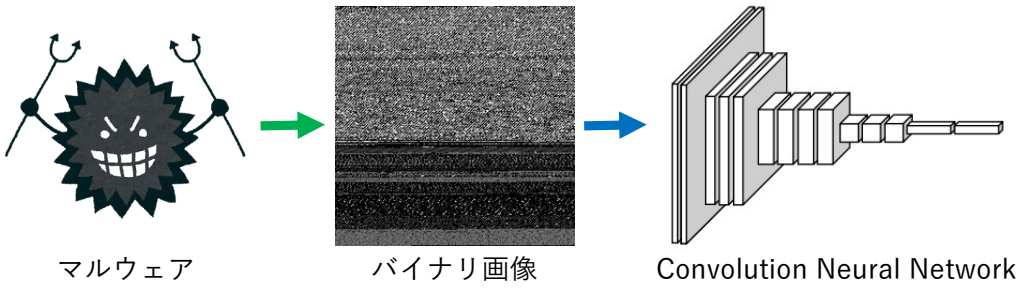


図 1.2: マルウェアの画像化による分類手法の概要

1.3 情報システムコースにおける本研究の位置づけ

マルウェアの画像化による分類手法は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が，バイナリ変更により分類結果が変動すると考えられる．本研究では，ソースコードに施す難読化を攻撃方法として想定し，その効果と対処を研究する．これは，情報システムコースにおけるディプロマ・ポリシーに基づき，分野横断的な探求力・構想力を持ち，実世界に実装する構成的手法に則り，新たな情報システムを創り出すことによって社会をデザインできる，身につけた専門能力を地域や社会の問題解決に適応させるとともに，新しい方法論や学問領域を切り拓くことを目指し，来るべき情報社会の構想に貢献できるという理念と一致する．このことから，マルウェアの画像化を用いた分類手法の恒常的な精度向上に資する，新たな攻撃手法とその対処の評価を行う．

1.4 論文の構成

本稿は全 7 章から構成されている．第 1 章は本研究の背景と研究目的について述べた．第 2 章では関連研究と得られる考察について述べる．第 3 章では難読化処理がバイナリ画像に与える影響とその効果を利用した攻撃及び対策のアプローチについて述べる．第 4 章では予備実験として評価基準となる IoT マルウェアの画像分類器を作成する．第 5 章では難読化を施したサンプルを使用して攻撃効果の検証を行う．第 6 章では難読化による攻撃への対策として，難読化を施したサンプルを用いた訓練の評価を行う．最後の第 7 章ではまとめと今後の課題について述べる．

第 2 章

関連研究・技術

マルウェアの画像化を用いた検知・分類は活発に行われている．しかし，筆者の調べた範囲では，ソースコードに施す難読化処理がマルウェアの画像分類手法へ与える影響は検証されていなかった．以降で IoT マルウェアの画像分類に関する研究, マルウェアの特有領域の検出に関する研究, バイナリ変更が画像分類に与える影響に関する研究についてそれぞれまとめ, 本研究との関連を述べる．更に，提案手法で使用する Obfuscator-LLVM という難読化ツールの機能詳細を記載する．

2.1 IoT マルウェアの画像分類に関する研究

イボットらの研究 [6] では，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詳細な比較を行っている．彼らの研究によると，検体数が十分であれば，CNN を用いた画像分類手法は各システムコール命令数を用いた分類手法より約 10% 精度が高く，およそ 85% の精度で分類可能である．検体数による精度推移を **図 2.1** に示す．本研究では，イボットらの研究に準じた IoT マルウェアの画像分類器を予備実験で作成し，これを用いた実験により難読化処理による攻撃と対策を行った．本研究での実験に用いたマルウェア検体の総数は 12498 であり，イボットらの研究と比較して十分な検体数から安定した精度の分類器が作成できた．

Su らの研究 [5] では，IoT マルウェアを対象として CNN を用いた画像分類手法の評価を行っている．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いため, Windows マルウェアの分類に用いる CNN よりもコンパクトなモデルでも高精度で分類可能であったと報告されている．マルウェアの難読化などの複雑なケースの検討が IoT マルウェアの検出率を向上させると考察されており，本研究に類する方向性を肯定している．

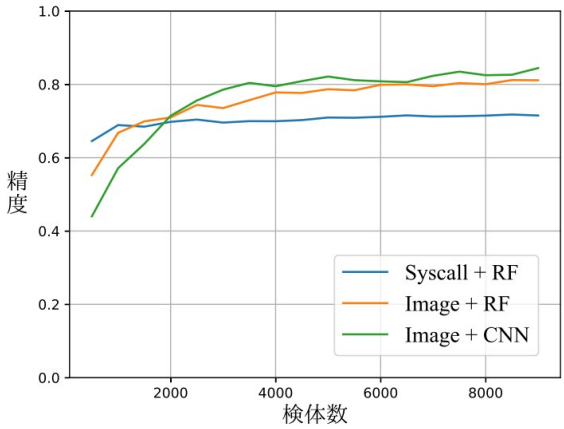


図 2.1: 検体数による精度推移 [6]

2.2 マルウェアの特有領域の検出に関する研究

Kalash らの研究 [9] では，Windows マルウェアに対して CNN による画像分類を行い，従来の機械学習アルゴリズムを使用した画像分類手法よりも高い精度が得られたと報告されている．CNN による分類では人間が設計したものよりも適した特徴量を自動的に獲得でき，マルウェアの画像分類の場合にはマルウェアの特有の領域が取得される．マルウェアの特有の領域とはファミリの共通の関数などを示している．

矢倉らの研究 [10] では，CNN に注意機構を加えることでマルウェアファミリの特有の領域を注意度マップに可視化している．注意度マップを元にマルウェアを解析することで，人手による解析の作業量が軽減されると考えられる．画像化したマルウェア及び対応した注意度マップを **図 2.2** に示す．マルウェアの特有の領域は同じファミリに属するマルウェア間で対応関係があり，特有の領域の位置が変化していても対応可能であったと報告されている．

本研究で行う Obfuscator-LLVM を使用した命令置換と偽の制御フローの追加により，マルウェアの特有領域が曖昧になり分類器が混乱すると考えられる．そのため，実験結果より各々の難読化機能がマルウェアの画像分類器に与える影響を検討する．

2.3 バイナリ変更が画像分類に与える影響に関する研究

小寺らの研究 [11] では，IoT マルウェアに対してコンパイラの最適化レベルの変更を行い，最適化レベル毎の類似度の比較を行っている．最適化レベルを変更した場合でも，画像間のテクスチャの見た目の類似性は維持されており，画像特徴によりマルウェアを検知できると推測されている．

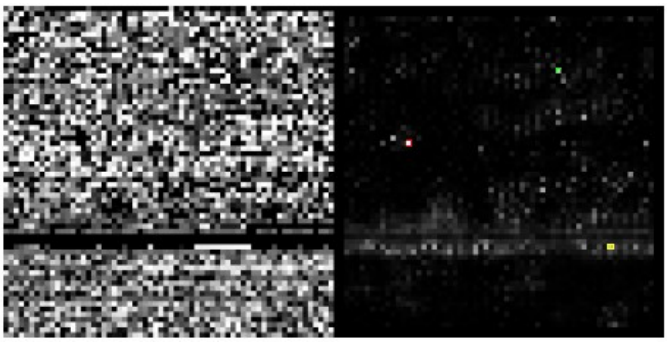


図 2.2: 画像化したマルウェアと対応した注意度マップ [10]

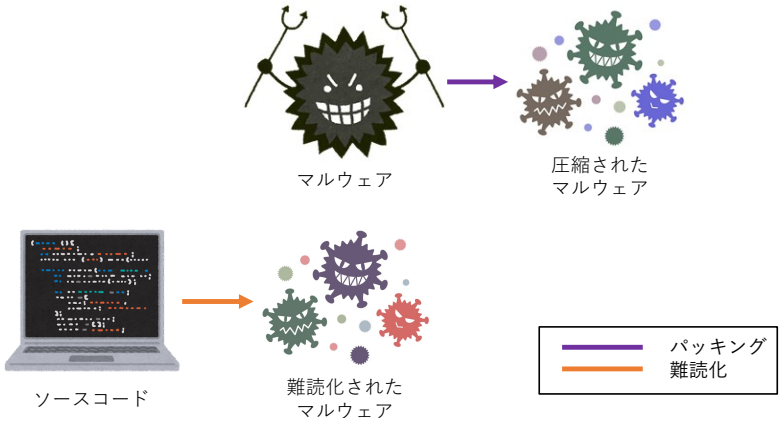


図 2.3: パッキングと難読化処理のマルウェア生成方法の違い

Nataraj らの研究 [12] では，Windows マルウェアに対して既存の画像分類手法を適用してマルウェアファミリの分類を行っており，パッカーにより圧縮（パッキング）されたマルウェアも分類可能であると報告されている．パッキングされたマルウェアは共通のバイト列を多く含んでいたため [10]，同様の理由から画像分類手法でも分類可能であった．ソースコードの難読化処理は，**図 2.3** のようにバイナリレベルでの変更という制限を受けずにコードレベルでの改変を行うことができるため対処が難しい．しかし，既存研究と同様に新しいマルウェアファミリとして学習させることで分類できると考えられる．

本研究では，ソースコードの難読化処理による攻撃効果の検証と対策の実装・評価を行い，難読化処理がマルウェアの画像分類手法に与える影響を明らかにし，対策済みの分類器の分類結果から事前対策の要否を検討する．

2.4 Obfuscator-LLVM の難読化技術

Obfuscator-LLVM（oLLVM）[8] とは，LLVM [13] をベースとした難読化ツールである．LLVM はコンパイルの際に LLVM IR と呼ばれる中間コードを作成し，最適化を施した後に実行可能なバイナリファイルを生成する．oLLVM は LLVM IR に対して難読化を行うため，同一のソースコードから毎回異なるバイナリを生成することができる．LLVM をバックエンドに用いる言語処理系のある C/C++，Swift，Go などのプログラミング言語にて，ターゲットプロセッサは x86，arm，x86_64 など様々なアーキテクチャに対応しており，コンパイル段階で難読化できるため利便性が高い．

oLLVM には，制御フローの平坦化，命令置換，偽の制御フローの追加の 3 つの難読化機能がある．制御フローの追加では，全ての基本ブロックを分割して switch 文と変数により制御するプログラムに変更する．命令置換では，標準の 2 項演算子を機能的に同等でありながらより複雑な命令に置き換える．偽の制御フローの追加では，プログラムに無駄な処理を追加し制御フローを複雑化する．oLLVM を用いた難読化処理の前後では，マルウェアのバイナリ画像が大きく異なる．詳細は 3.1.2 項で説明する．

ソースコードの内部処理やデータに価値がある場合，リバースエンジニアリングにより解読され，コードが流出する危険性がある．そのため，アプリケーションを商用展開する際には，解読をなるべく回避するために難読化が必要とされている．Android アプリケーションでは，正規のソフトウェアにおいて難読化が利用されている一方で，マルウェアについても適用が見られるという報告がある [14]．モバイルアプリケーションにおいてマルウェアの難読化が広まっているように，IoT マルウェアにおいてもコードレベルでの難読化という高度な防御手段の利用は十分に想定される．そのため，本研究では Obfuscater-LLVM を用いたソースコードへの難読化処理により，マルウェアの画像分類手法に対する攻撃の有効性と対策を研究する．

第 3 章

マルウェアの画像分類手法への難読化による攻撃手法の提案

画像化による分類手法はマルウェアのバイナリ変更の影響を受けるため，プログラムの意味が変わらないバイナリ変更は画像化による分類手法への攻撃手法と考えることができる．このようなバイナリ変更をもたらす方法としてソースコードへの難読化処理に着目する．本研究ではソースコードへの難読化を用いたマルウェアの画像分類への攻撃手法を提案する．オープンソースの IoT マルウェアの亜種生成の過程に難読化処理が加えられることを想定し，その攻撃の有効性と対処の可能性を検証する．以降では Obfuscator-LLVM（oLLVM）を用いた難読化による攻撃の概要とマルウェアバイナリの画像化手法を説明した後に，oLLVM を用いた難読化によるバイナリ画像の変化を利用した攻撃手法と難読化を施したサンプルを含めた訓練による対処を記載する．

3.1 Obfuscator-LLVM を用いた難読化処理による攻撃

Obfuscator-LLVM を用いた難読化による攻撃の概要を**図 3.1** に示す．マルウェアの画像分類手法では収集したマルウェアを画像化し，CNN（Convolutional Neural Network）による深層学習により分類を行う．この画像分類手法への攻撃方法として，oLLVM を用いてソースコードへの難読化を施したサンプルを生成する．難読化を施したサンプルはバイナリ表現が変化しているため，画像化するとオリジナルのマルウェアと視覚的な違いが生まれる．攻撃の実装では，評価基準となる IoT マルウェアの画像分類器を作成し，収集したマルウェアのテストデータを難読化を施したサンプルのバイナリ画像と入れ替えることで検証を行う．更に，難読化を施したサンプルを用いた訓練を行うことで対処を試みる．

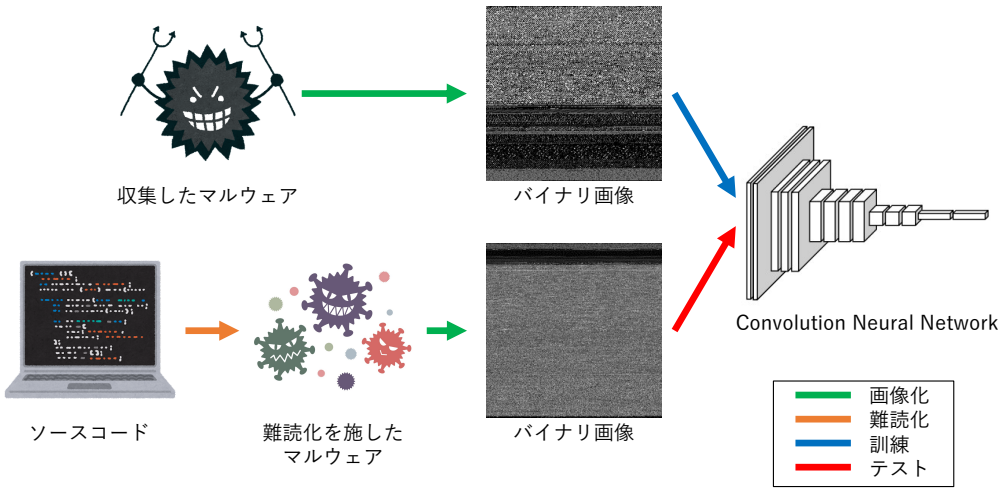


図 3.1: Obfuscator-LLVM を用いた難読化による攻撃

Algorithm 1 マルウェアの画像化アルゴリズム

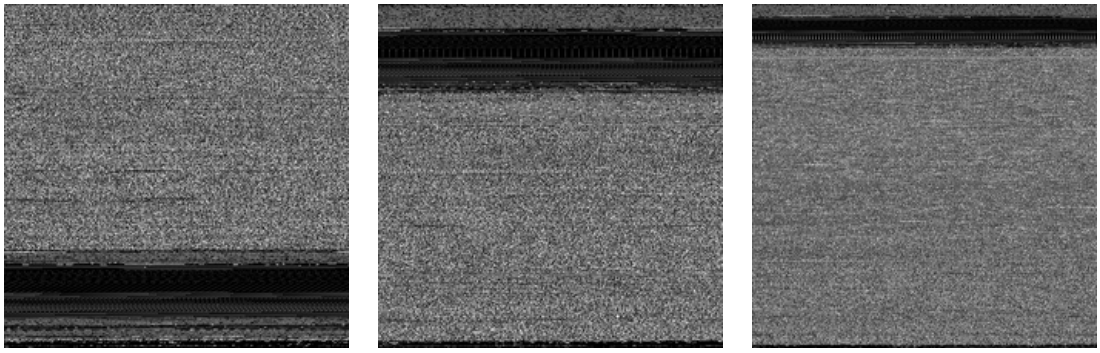
Require: *binary*

Ensure: *image*

$s \leftarrow \sqrt{\text{sizeof}(binary)}$
$image[s][s] \leftarrow 0$
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
$image \leftarrow \text{resize}(image, (224, 224))$
return *image*

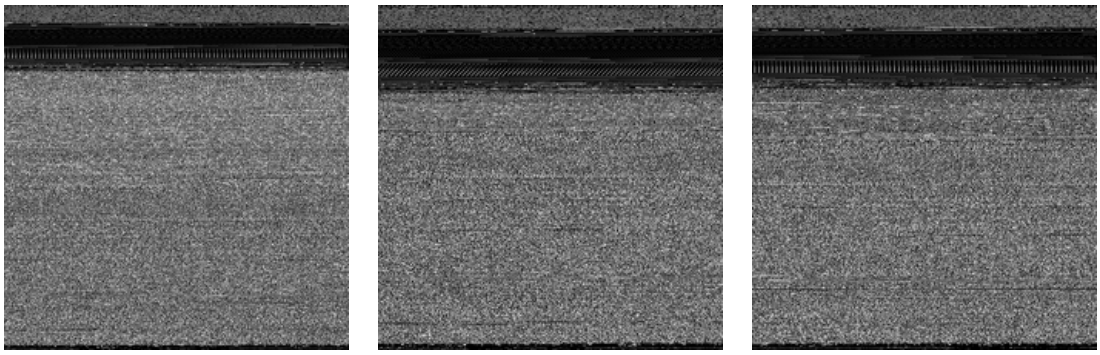
3.1.1 マルウェアバイナリの画像化

画像化アルゴリズム [6] を **Algorithm 1** に示す．本研究では，マルウェア実行形式バイナリのファイルサイズに応じて，1 バイトを 1 ピクセルとした正方形のグレースケール画像に変換する．マルウェア毎に異なるサイズの画像が生成されるが，CNN は入力サイズを統一する必要があるため，224 × 224 にリサイズする．224 × 224 というサイズは，本実験で使用する CNN による分類器の実装である VGG19 の入力に使われるサイズである．リサイズにより特徴量が増減するため，精度と対比した入力サイズの検討が必要である．



(a) gcc によるビルド　**(b)** clang によるビルド　**(c)** oLLVM によるビルド

図 3.2: 各ビルド手法のバイナリ画像



(a) 制御フローの平坦化　**(b)** 命令置換　**(c)** 偽の制御フローの追加

図 3.3: 難読化機能別のバイナリ画像

3.1.2 Obfuscator-LLVM を用いた難読化処理によるバイナリ画像の変化

本手法では，Obfuscator-LLVM [8] という難読化ツールを用いて同一のソースコードから毎回異なるバイナリを生成し，マルウェアの画像化による分類手法に対して攻撃を行う．oLLVM には制御フローの平坦化，命令置換，偽の制御フローの追加の 3 つの機能があることから，本手法では各機能を組み合わせた難読化処理を行う．

図 3.2 に，ひとつの IoT マルウェアを gcc，clang [15]，oLLVM でそれぞれビルドし 3.1.1 節で示した手法でバイナリ画像化した例を示す．clang 及び oLLVM はバックエンドコンパイラとして LLVM を用いており，gcc を用いた場合とセクション配置が異なる．そのため，gcc によるビルドと clang によるビルド及び oLLVM によるビルドを比較すると，一目で分かるほどの大域的な変化が生まれている．本研究で収集したマルウェアより，市中で蔓延しているマルウェアは gcc を用いてビルドされた検体が多いことを確認している．収集したマルウェアのバイナリ画像例を付録に掲載する．付録 A は gcc を用いてビルドされたと見られ

る検体であり，付録 B は静的解析を妨害する何らかのツールで生成されたと見られる検体である．付録 B に掲載した検体群はパッキングされたマルウェアであると考えられるが，学習しても分類できないものが存在するため，他の妨害ツールで生成されたマルウェアも含まれていると推測される．

難読化機能別のバイナリ画像を**図 3.3** に示す．難読化を施していないバイナリ画像は**図 3.2** の clang によるビルドと同義のため比較して見て頂きたい．制御フローの平坦化は序盤に平坦化処理が追加されるためコード部分が大きくなる．コード部分とはバイナリ画像の下側にある大きなブロックのことである．命令置換，偽の制御フローの追加ではバイナリ全体に対して冗長なコードが追加されるため，マルウェアのファイルサイズが大きくなり画像全体のテクスチャが細かくなる．画像単体では効果が分かりにくいが，全ての難読化機能を 1 度ずつ適用した**図 3.2** の oLLVM によるビルド画像を確認することで，画像全体の粒度が細かくなっていることが分かる．

LLVM を通したセクションの再配置によるバイナリ画像の大域的な変化と難読化処理による画像細部の変化により，マルウェアの画像分類器を混乱させることが可能であると考えられる．難読化による変動の大きさを制御しながら，都度新しいバイナリを生成することで，マルウェアの画像化による分類手法に攻撃を行う．

3.2 難読化処理を施したサンプルを含めた訓練による対処

難読化処理を施したサンプルを含めた訓練による対処の概要を**図 3.4** に示す．ソースコードに施す難読化によるマルウェアの画像分類手法への攻撃効果が認められた場合，難読化を施したサンプルを用いた訓練を行うことで対処可能であると考えられる．画像分類器の訓練を行う際は，学習に用いたマルウェアに加えて難読化を施したサンプルを使用し，難読化を施したサンプルは新しいマルウェアファミリとして学習させる．難読化なしの clang のみを用いた攻撃にも対応できるように，難読化機能を 0 回適用したサンプルも含めた訓練を行う．難読化を施したサンプルを含めて訓練した分類器が誤分類した場合，誤分類しているサンプルを確認することで攻撃効果が高い難読化機能を特定する．

3.3 画像分類手法への攻撃及び対策の検討の構成

IoT マルウェアのソースコードへ難読化を施し生成したバイナリを，収集したマルウェアで学習させた画像分類器を用いて判別させ，難読化処理による攻撃効果を検証する．次に，その攻撃への対策として，難読化を施したサンプルを用いて訓練したマルウェアの画像分類器を作成し，難読化の有無を混合したサンプルを判別させ精度を検証する．本研究におけるマルウェアの画像分類手法への攻撃及び対処の流れを**表 3.1** に示す．

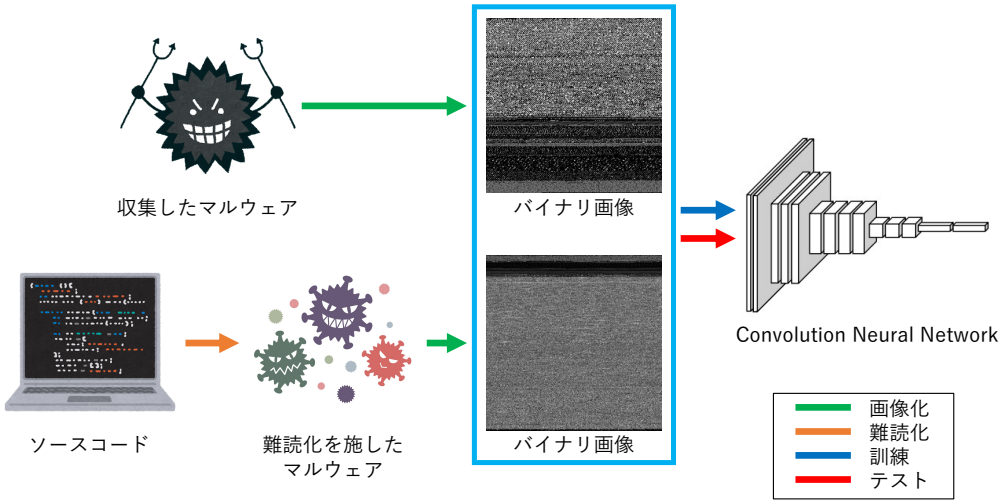


図 3.4: 難読化処理を施したサンプルを含めた訓練による対処

表 3.1: 画像分類手法への攻撃と対処の流れ

章立て	目的	学習データ	テストデータ
第 4 章	ベースライン分類器の作成	収集したマルウェア	収集したマルウェア
第 5 章	難読化による攻撃効果の確認	収集したマルウェア	難読化サンプル
第 6 章	サンプルの単純な学習の評価	収集したマルウェア＋ 難読化サンプル	収集したマルウェア＋ 難読化サンプル
第 6 章	収集したマルウェアへの影響の確認	難読化サンプル	難読化サンプル
第 6 章	難読化機能別の 細分化を用いた訓練の評価	難読化サンプル （細分化）	難読化サンプル （細分化）

第 4 章では，収集したマルウェアを用いて評価基準となる IoT マルウェアのベースライン画像分類器を作成する．難読化処理による攻撃効果の検証として，攻撃前後での分類結果の比較が必要とされるためである．第 5 章では，ベースライン分類器に対してテストデータとして難読化サンプルを分類させることで攻撃を行う．分類結果の違いからソースコードへの難読化による攻撃効果を検証する．第 6 章では，攻撃への対策として難読化を施したサンプルを用いて訓練したマルウェアの画像分類器を 3 つ作成する．1 つ目の分類器は，収集したマルウェアと難読化サンプルを単純に学習させ，難読化されたサンプルの正しいマルウェアファミリへの分類の可能性を評価する．2 つ目の分類器は，難読化サンプルのみを学習させ，収集したマルウェアに影響を及ぼさないことを確認する．難読化サンプルは Mirai, Lightaidra，Bashlite の 3 種であり，評価対象のマルウェアファミリが絞られている．3 つ目の分類器では，難読化機能毎に細分化を行い，分類の際に論理和を取ることでサンプル分類における精度向上を目指す．訓練時間の都合上，収集したマルウェアを含めた学習は行わない．

第 4 章

マルウェア検体の収集と ベースライン画像分類器の生成

本章では，データセットの収集及びラベリングとベースライン画像分類器の作成を行う．既存研究 [6] より，マルウェアの分類手法において安定した精度を出すには，少なくとも 4000 以上の検体数が必要である．VirusShare より ELF 形式のマルウェアをまとめて入手し，ARM アーキテクチャ対象かつ静的リンクの検体のみを抽出した後に，VirusTotal 経由で Microsoft の提供する判定結果を利用してラベリングを行った．実験に使用するマルウェアファミリとして 100 検体以上収集できたものを対象ファミリとした．難読化処理による攻撃前後での分類結果の比較が必要であるため，収集したマルウェアを用いて評価基準となる IoT マルウェアの画像分類器を作成した．画像分類器はイボットらの研究 [6] を参考にしながら，ファインチューニングやハイパーパラメータの調整を行い精度を高めている．生成したベースライン分類器のマルウェアファミリ分類精度は約 78% であった．IoT マルウェアの約 8 割を占める Mirai，Lightaidra，Bashlite の 3 つのマルウェアはソースコードが入手可能なため本研究の評価対象とした．

4.1 実験環境

マルウェア操作用の PC 及び機械学習用の PC をそれぞれ用意した．マルウェア操作用の PC の詳細を**表 4.1** に示す．この機器では IoT マルウェアの入手と画像化を行った．機械学習用 PC の詳細を**表 4.2** に示す．機械学習用の PC を使用して IoT マルウェアの画像分類器を作成した．本研究では旧型のメインストリーム向け PC を使用しているため，最新型の機械学習向け PC に比べると学習が遅い．

表 4.1: マルウェア操作に用いる PC のスペック

マルウェア操作用の PC	
OS	Windows 10 Pro 64bit ver.20H2
CPU	Intel Core i7-4790 4 コア/8 スレッド
RAM	8.0GB

表 4.2: 機械学習に用いる PC のスペック

機械学習用の PC	
OS	Windows 10 Pro 64bit ver.21H1
CPU	AMD Ryzen3 3300X 4 コア/8 スレッド
RAM	32.0GB
GPU	NVIDIA GeForce GTX1080 8GB

4.2 実験に用いるデータセット

実験に用いるデータとして，マルウェア共有サイト VirusShare [16] から，2014 年から 2020 年にかけて確認された ELF 形式のマルウェアをまとめて入手した．同様のマルウェアは「VirusShare_ELF_20190212」,「VirusShare_ELF_20200405」から入手可能である．

収集したマルウェアの各アーキテクチャの割合を**表 4.3** に示す．アーキテクチャが異なる場合は，そのバイナリから生成される画像の特徴も大きく異なるため，本実験では検体数が十分である Arm アーキテクチャを対象とした検体のみ使用する．Arm アーキテクチャ対象のマルウェアのリンク方式を調査したところ，多くのマルウェアが静的リンクであった．各リンク方式の割合を**表 4.4** に示す．静的リンクされていない検体は占有する割合が低いため，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．検体の抽出には，readelf コマンドを用いたバッチファイルを作成し使用した．

多くの IoT デバイスが ARM アーキテクチャに基づく CPU を搭載しており，IoT マルウェアの多くが静的リンクであることから，データセットの中から ARM アーキテクチャかつ静的リンクの検体のみを抽出しラベリングを行った．マルウェアファミリのラベリングには VirusTotal [17] 経由で Microsoft の提供する判定結果を使用し，100 検体以上存在するマルウェアファミリを取得した．本研究で収集し実験に用いたマルウェアの検体数は 12498 であった．

各マルウェアファミリの検体数を**表 4.5** に示す．IoT マルウェアの約 8 割は Mirai, Lightaidra，Bashlite で占められていることが分かる．これらのマルウェアは github にてソースコードが入手可能なため [18]，第 5 章以降での難読化処理を施したサンプル生成の際

はこれら 3 種類の IoT マルウェアのソースコードを利用する.

表 4.3: 収集したマルウェアの各アーキテクチャの割合

アーキテクチャ	検体数	割合 (%)
Arm	16468	30.55
x86	9702	18.00
MIPS	9318	17.29
PowerPC	4228	7.84
SuperH	4141	7.68
Motorola 68k	4026	7.47
SPARC	4026	5.63
x86-64	2741	5.09
（others）	242	0.45

表 4.4: Arm アーキテクチャ対象検体の各リンク方式の割合

リンク方式	検体数	割合 (%)
静的リンク	14014	85.10
動的リンク	2454	14.90

表 4.5: ラベリングの内訳

ファミリ名	検体数	割合 (%)
Mirai	4846	38.77
Lightaidra	2612	20.90
Bashlite	2301	18.41
Occamy	1466	11.73
Mploit	597	4.78
Skeeyah	318	2.54
Yakuza	134	1.07
Tsunami	118	0.94
Berbew	106	0.85

4.3 実験手順

収集したマルウェアのバイナリ画像を入力とし CNN を用いて評価基準となる IoT マルウェアの画像分類器を作成する．CNN の作成には TensorFlow [19] 及び Keras [20] を利用する．

機械学習アルゴリズムに与える検体はマルウェアファミリごとに分割し，訓練用を 7 割，テスト用を 3 割，訓練データの 1 割を検証データとして使用した [6]．収集したマルウェアは現実のマルウェアの出現頻度を反映していると仮定してリサンプリングは行わなかった．

CNN のモデルは VGG16 [21]・VGG19 [21] を転移学習して利用した．各モデルの詳細を**図 4.1** に示す．畳み込み層の 1 層，2 層と 3 層，4 層，5 層は同一の層が連続しているため，**図 4.1** では省略している．ピュアな全結合層は取り払い，新たな全結合層を作成した．

作成した全結合層の説明を行う．1 つ目の Dense 層は Relu 関数を使用し 2048 次元の出力を行っている．カーネルの初期化には he_normal，バイアスの初期化には Zeros を用いた．Dropout 層の値は 0.5 であり，過学習の抑制のために追加した．2 つ目の Dense 層は Relu 関数を使用し 1024 次元の出力を行っている．カーネルの初期化には he_normal，バイアスの初期化には Zeros を用いた．最後の Dense 層は softmax 関数を使用し分類結果の出力を行っている．カーネルの初期化には glorot_uniform，バイアスの初期化には Zeros を用いた．

ファインチューニングの前後での精度比較を行った．ファインチューニング前は新たな全結合層のみを学習し，全結合層以外の層の重みは学習済みのパラメータを維持して使用した．ファインチューニング後は最後の畳み込み層と新たな全結合層を学習し，それ以外の層は再学習を行わなかった．

ハイパーパラメータの調整前後での精度比較を行った．調整前はエポック数を 25，最適化関数を SGD，バッチサイズを 6，学習率を 0.01 で訓練した．調整後はエポック数を 30，最適化関数を SGD，バッチサイズを 6，学習率はエポック数に合わせて変化させ，学習の初めは 0.01，エポック数が 20 以上のときは 0.001 に収縮させ訓練した．

CNN モデルへのファインチューニング及びハイパーパラメータの調整を行い，最も精度が高い IoT マルウェアの画像分類器を難読化による攻撃検証の評価基準とする．

4.4 評価

各モデルの訓練中の精度推移を**図 4.2** に示す．縦軸が精度，横軸がエポック数であり，青色の線が訓練用データ，橙色の線が検証用データを表している．

テストデータを用いたモデル別の精度評価の結果は**表 4.6** に示す．訓練モデルと訓練方法の違いによる精度の差異をまとめている．

各モデルのマルウェアファミリ分類結果の混同行列を**図 4.3** に示す．縦軸が真のマルウェ

表 4.6: モデル別の精度評価

訓練モデル / 訓練方法	VGG16	VGG19
転移学習	0.5811	0.5398
ファインチューニング	0.7314	0.7565
ハイパーパラメータの調整	0.7722	0.7748

アファミリであり，横軸が推測されたマルウェアファミリである．真のファミリに対する推測されたファミリの比率を数値と色で表現している．青色が濃ければ濃いほど，対応する推測が行われた検体が多い．

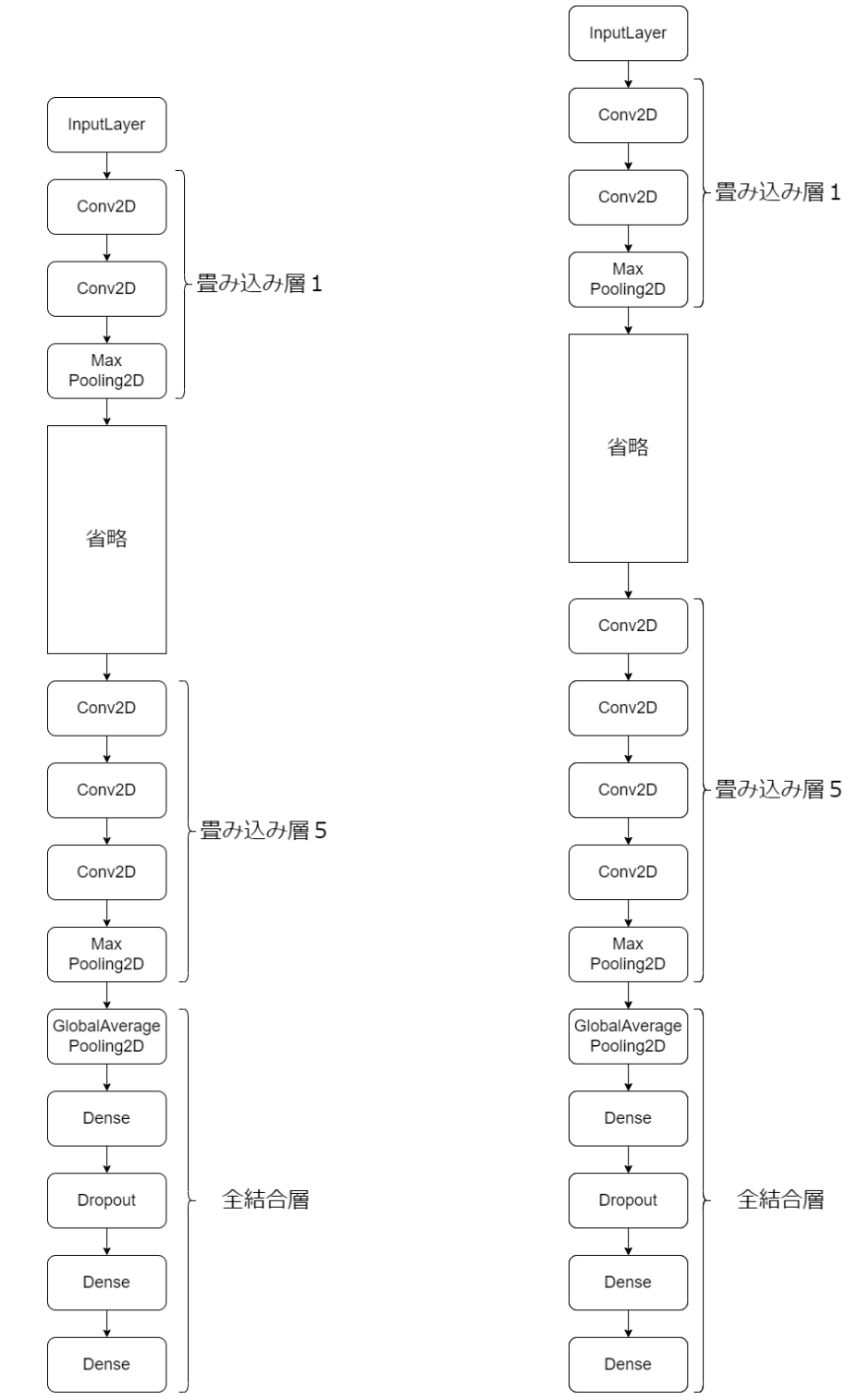
4.5 考察

表 4.6 より，全結合層のみを学習させた転移学習よりもファインチューニングを行った訓練の方が高い精度を示している．全結合層のみの学習では約 55% 前後の精度であり，上手く学習できていないことがわかる．ファインチューニングを行った訓練と更にハイパーパラメータの調整も行った訓練では，ハイパーパラメータの調整も行った訓練の方が約 3% 高い．学習率とエポック数の値をモデルに適したものに改良できたと考えられる．

図 4.2 より，全結合層のみを学習させた転移学習では検証用データの結果が上下していて安定していないが，ファインチューニングを行った後の結果ではブレがなくなり安定していることがわかる．エポック数が十分である訓練をすれば，訓練データでの成績は検証データでの成績を上回る [22] ため，その点からも学習が成功していることがわかる．

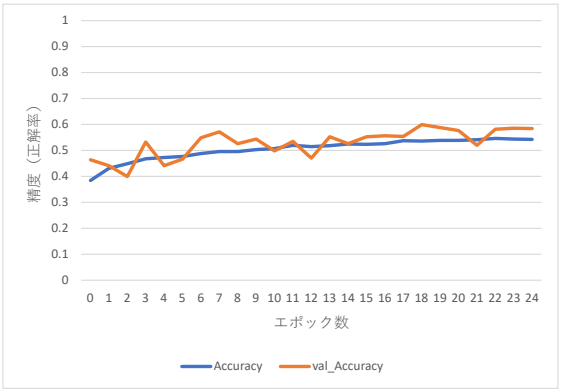
図 4.3 の結果を見ていく．全結合層のみを学習させた転移学習では，検体数の多い Mirai や Lightaidra しか学習できていない．VGG16 の場合は Yakuza も分類できているが，全体的な分類ができていないため信頼性がない．ファインチューニング後では Mirai，Lighaidra に加えて Bashlite，Yakuza，Tsunami，Berbew が分類できるようになっている．ハイパーパラメータの調整後では，分類傾向は変更されずに各ファミリごとの精度が上がっている．Tsunami と Berbew は検体数が少ないため精度が低くなっているが，大規模なデータセットを利用することで解決できると考えられる．Occamy，Mploit，Skeeyah は分類結果が混同している．Mploit と Skeeyah は先行研究 [6] では分類されていなかったため，最近になり流行り始めたマルウェアだと思われる．ラベリングの際に返ってきた亜種の種類が多いこと，マルウェアの画像分類手法において分類できないことから，何らかの静的解析の阻害を目的としたツールを使用して作成されたと考えられる．本研究の目的はソースコードへの難読化処理による攻撃効果の検証とその対策の評価であるためこの問題について深く触れないが，マルウェアの画像分類手法を実用化する場合は解決する必要がある．

VGG19 にファインチューニングを施し，ハイパーパラメータを調整して訓練した分類器の精度が最も高く約 78% であった．本研究ではこの分類器を評価基準となる IoT マルウェアの画像分類器とする．先行研究 [6] の分類結果より約 5 ポイント低い結果となったが，マルウェアの分類傾向は似ているため，使用したデータセットの違いによるものと考えられる．

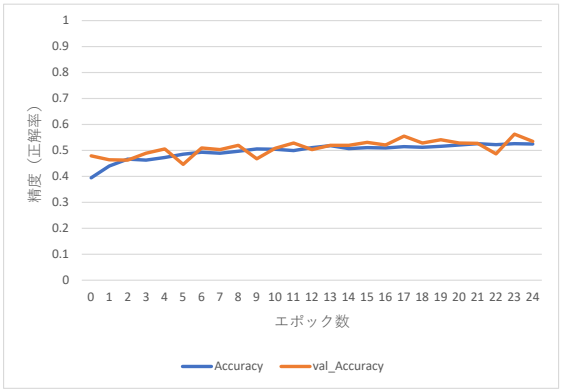


(a) vgg16 の転移学習　　**(b)** vgg19 の転移学習

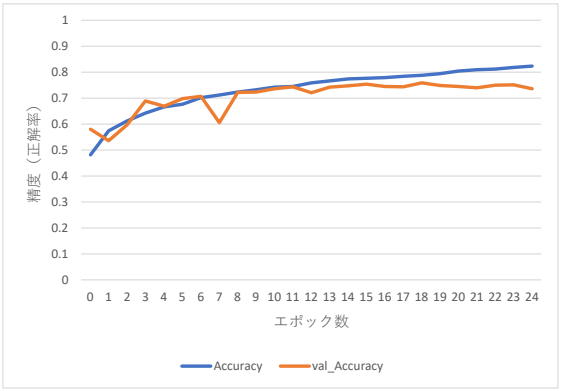
図 4.1: 使用した CNN モデルの階層構造



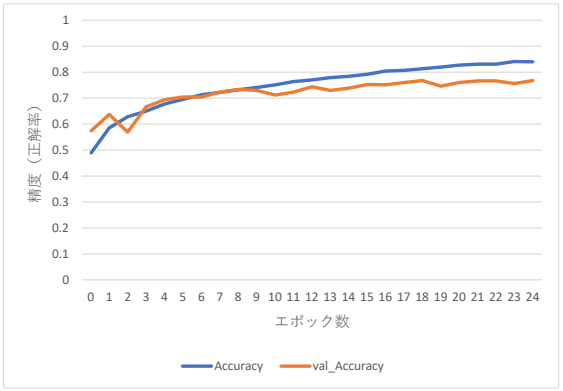
(a) vgg16 の転移学習



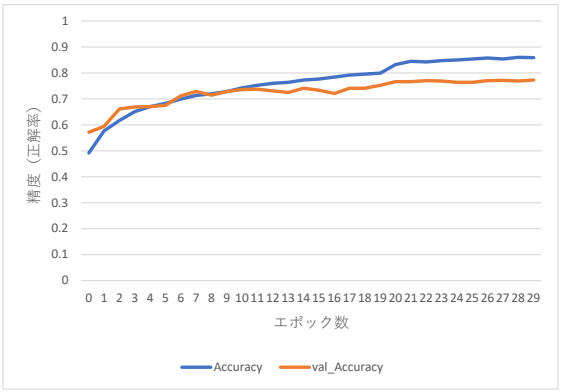
(b) vgg19 の転移学習



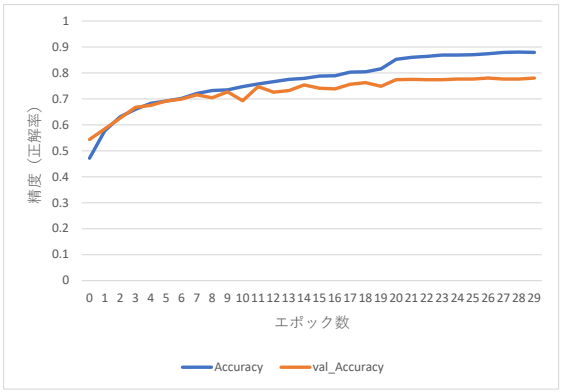
(c) vgg16 のファインチューニング



(d) vgg19 のファインチューニング

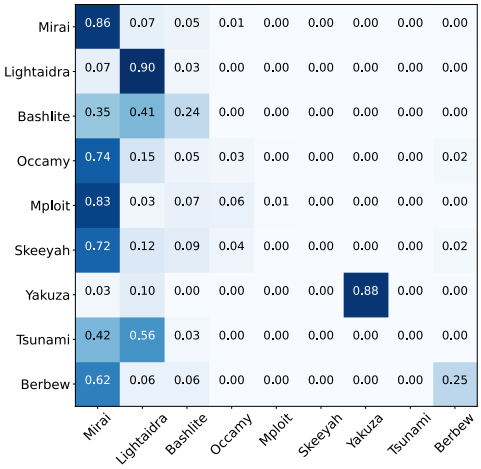


(e) vgg16 のハイパーパラメータの調整

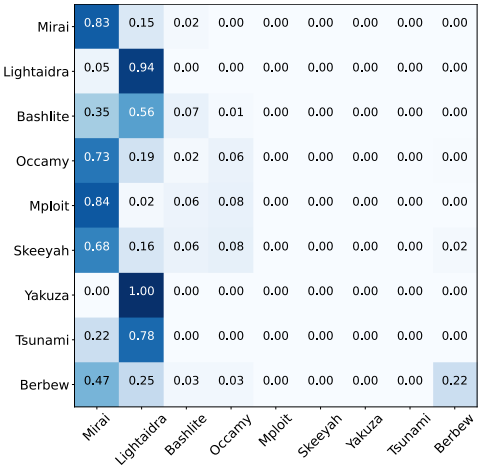


(f) vgg19 のハイパーパラメータの調整

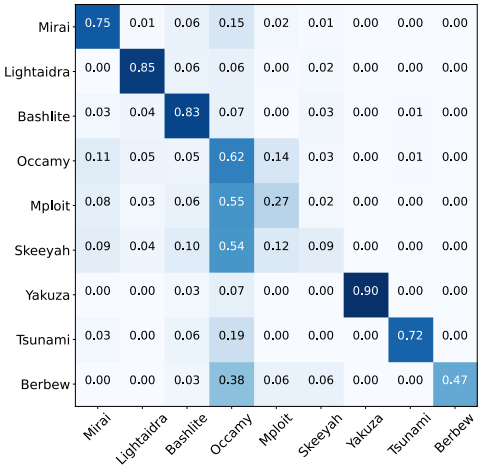
図 4.2: 各モデルの精度推移



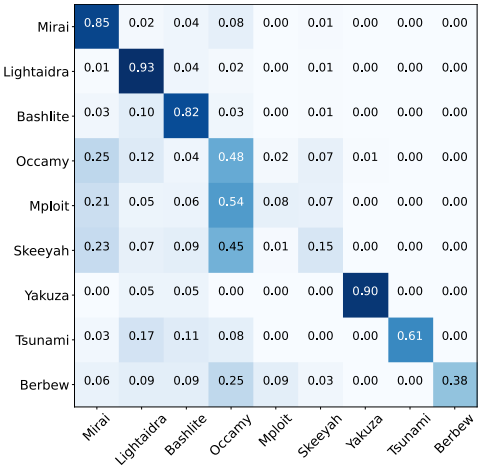
(a) vgg16 の転移学習



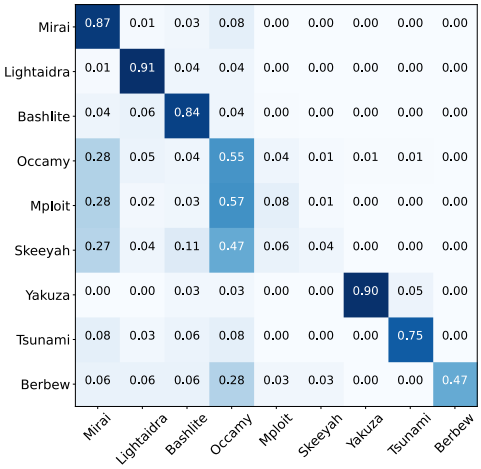
(b) vgg19 の転移学習



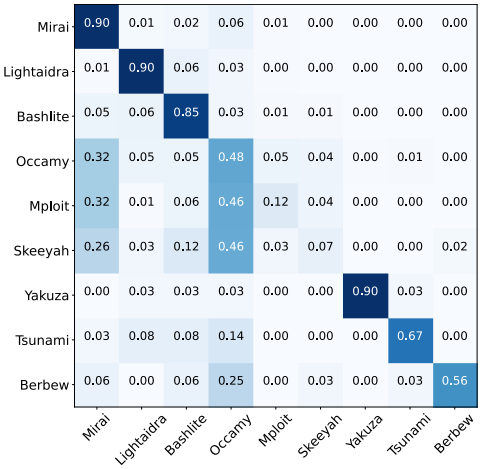
(c) vgg16 のファインチューニング



(d) vgg19 のファインチューニング



(e) vgg16 のハイパーパラメータの調整



(f) vgg19 のハイパーパラメータの調整

図 4.3: 各モデルのマルウェアファミリ分類結果の混同行列

第 5 章

Obfuscator-LLVM を用いた難読化処理による攻撃効果の検証

攻撃対象とするマルウェアファミリである Mirai，Lightaidra，Bashlite の精度からベースラインとの差分を求め，難読化処理による攻撃効果を検証する．oLLVM を用いて難読化を施したマルウェアバイナリを生成し，第 4 章で作成したベースライン分類器に対して，テストデータとして難読化サンプルを分類させることで攻撃を行った．本研究では，clang による攻撃と oLLVM による攻撃は分けて実装した．clang は LLVM 上で動作することを意図して作られたフロントエンドコンパイラである．oLLVM はバックエンドコンパイラとして LLVM を用いており，clang による攻撃を含有している．clang による LLVM を通したセクション再配置の攻撃効果と oLLVM による難読化処理の攻撃効果を個別に確認する．

5.1 実験環境

第 4 章と同様に，機械学習用の PC 及びマルウェア操作用の PC を使用した．マルウェア操作用の PC では，Linux 仮想環境を構築しサンプル生成及び画像化を行った．Linux 仮想環境の詳細を**表 5.1** に示す．機械学習用の PC では，評価基準となる IoT マルウェアの画像分類器に対して難読化処理による攻撃を行った．

表 5.1: Linux 仮想環境のリソース割り当て

Linux 仮想環境	
OS	Ubuntu 20.04 64bit
CPU	Intel Core i7-4790 4 スレッド
RAM	4.0GB

5.2 実験に用いるデータセット

第 4 章においてラベリング済みのデータセット及び clang でビルドしたサンプル，難読化を施したサンプルの 3 種類を使用した．サンプルのビルドには Mirai, Lightaidra, Bashlite のソースコードを用いた．clang（LLVM）でビルドしたサンプルを使用するのは，LLVM を通したセクション再配置と oLLVM による難読化処理の効果を個別に確認するためである．

oLLVM の難読化機能は，制御フローの平坦化，命令置換，偽の制御フローの追加を各自 1 回ずつ適用し，難読化を施したサンプルの総数はマルウェアファミリ毎にテストデータと同数用意した．この難読化機能の組み合わせを用いるのは，収集したマルウェアのファイルサイズから逸脱せず，全ての難読化機能が有効であり，ビルド時間が比較的短いためである．

5.3 実験手順

第 4 章の実験で作成した評価基準となるマルウェアの画像分類器に対して，難読化を施したサンプルのバイナリ画像を攻撃対象のマルウェアファミリ毎にテストデータと入れ替えることで攻撃を行った．攻撃対象のファミリは，収集したマルウェアのラベリングの際に全体の約 8 割を占めていた Mirai，Lightaidra，Bashhlite である．clang による LLVM を通したセクション再配置の攻撃効果と oLLVM による難読化処理の攻撃効果を個別に確認するため，clang による攻撃と oLLVM による攻撃は分けて実装した．

5.4 評価

難読化による攻撃前後の分類結果の混同行列を**図 5.1** に示す．縦軸が真のマルウェアファミリであり，横軸が推測されたマルウェアファミリである．真のファミリに対する推測されたファミリの比率を数値と色で表現している．青色が濃ければ濃いほど，対応する推測が行われた検体が多い．各手法の攻撃対象ファミリの精度を**表 5.2** にまとめる．

表 5.2: 各手法の攻撃対象ファミリの精度

	Mirai	Lightaidra	Bashlite
評価基準の分類器	0.90	0.90	0.85
clang による攻撃	0.00	0.00	0.00
oLLVM による攻撃	0.00	0.00	0.00

5.5 考察

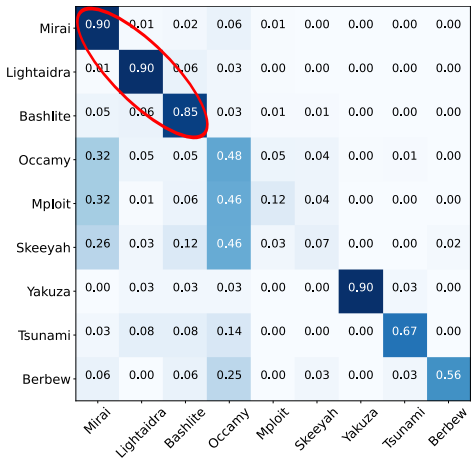
表 5.2 より，ベースライン分類器と clang による攻撃，oLLVM による攻撃のターゲットとなるマルウェアファミリの精度を比較すると，ベースライン分類器は約 88% の精度で分類可能であるが，攻撃後はどちらの手法も精度が 0% になったおり，攻撃したマルウェアファミリである Mirai，Lightaidra，Bashlite が全て誤分類されていることが分かる．

clang を用いた攻撃効果が認められることから，LLVM を通したセクション再配置の効果が大きいと考えられる．しかし，**図 5.1** において，clang による攻撃は全て同じファミリに分類されている．clang によるビルドはランダム性がないため同じバイナリしか生成できず，単体での攻撃性能は低いと推測される．

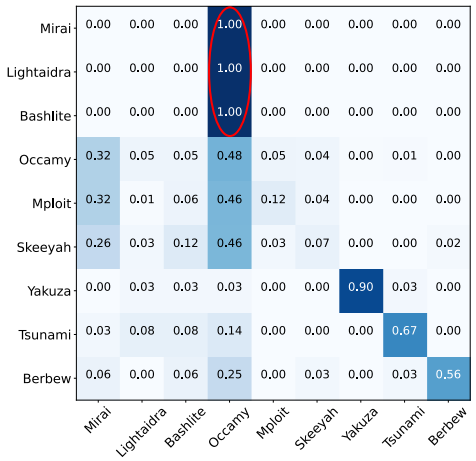
oLLVM によるビルドは難読化により毎回異なるバイナリが生成されるため，バイナリによって分類されるファミリが変化している．バックエンドコンパイラとして LLVM を用いており，clang による攻撃効果も内包しているため，難読化との相乗効果で攻撃性能が高まっていると考えられる．

clang による攻撃と oLLVM による攻撃の多くが Occamy と誤分類している．Occamy，Mploit，Skeeyah が上手く分類できないため，難読化のような静的解析を阻害するための手法を用いている可能性が高まった．

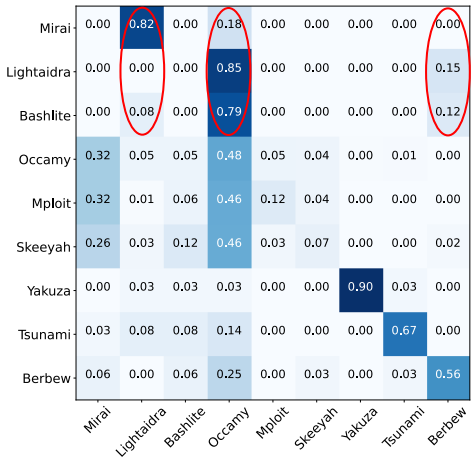
本実験により，コンパイル時に適用する難読化処理がマルウェアの画像分類手法への単純な攻撃として一定の効果を持ち得ることが確認できた．マルウェアの画像化による分類手法の恒常的な精度向上のためには，このような想定される攻撃への対処が必要である．



(a) ベースライン分類器の分類結果



(b) clang による攻撃の分類結果



(c) oLLVM による攻撃の分類結果

図 5.1: 難読化による攻撃前後の分類結果の混同行列

第 6 章

難読化サンプルを学習させた画像分類器による防御

難読化による画像分類手法への攻撃有効性が確認されたため，対策として難読化を施したサンプルを用いて訓練したマルウェアの画像分類器の作成を行う．まず，収集したマルウェアと難読化を施したサンプルを学習させた分類器を生成し，提案手法による難読化されたサンプルの正しいマルウェアファミリへの分類の可能性を評価する．次に，提案手法の結果により誤分類されたサンプルにおいて，有効であった難読化機能を確認する．本研究では，clang と oLLVM という 2 つのコンパイラを用いているため，コンパイラの差異の影響も考慮する．2 つ目の分類器では，難読化を施したサンプルのみを学習させ，収集したマルウェアがサンプルを用いた訓練に影響を及ぼさないことを確認する．最後に，3 つ目の分類器は，難読化を施したサンプルのみの訓練において，難読化機能毎に細分化を行うことで，サンプル分類の精度向上を目指す．難読化による攻撃への対策前ではサンプルは全て誤分類されていたが，難読化を施したサンプルを用いて単純に訓練をするだけで 85% 以上の精度で分類できた．誤分類されていたサンプルに施した難読化機能は，命令置換と偽の制御フローの追加であった．更に，難読化機能毎に細分化を行うことで，難読化機能によって変化する画像特徴を明確にし，誤分類されていたサンプルも分類できた．難読化を施したサンプルは，難読化機能毎に細分化を行い論理和を取ることで，100% に近い精度で分類可能であった．

6.1 実験環境

第 4 章と同様に，機械学習用の PC 及びマルウェア操作用の PC を使用した．マルウェア操作用の PC では，Linux の仮想環境においてサンプル生成及び画像化を行った．機械学習用の PC では，難読化を施したサンプルを用いた訓練を行い，新たなマルウェアの画像分類器を 3 つ作成した．

6.2 実験に用いるデータセット

第 4 章においてラベリング済みのデータセット及び難読化を施したサンプルを使用し，第 5 章と同様にサンプルのビルドには Mirai，Lightaidra，Bashlite のソースコードを用いた．oLLVM の難読化機能は，制御フローの平坦化，命令置換，偽の制御フローの追加を各自 0〜1 回適用した．難読化機能の組み合わせは 8 通りである．難読化なしの clang による攻撃にも対応できるように，難読化機能を 0 回適用したサンプルも含めた．1 通りにつき 150 検体の難読化されたサンプルを用意したため，難読化されたサンプルはマルウェア種別に 1200 検体である．攻撃及び対処に使用する IoT マルウェアは 3 種類であるため，難読化を施したサンプルの総数は 3600 検体であった．

6.3 実験手順

収集したマルウェア及び難読化を施したサンプルのバイナリ画像を入力とし，CNN を用いてソースコードの難読化に対して 3 つのマルウェアの画像分類器を作成した．

1 つ目の分類器は，収集したマルウェアと難読化を施したサンプルを単純に訓練させたものである．訓練により，分類可能なマルウェアファミリの拡大が期待できる．

2 つ目の分類器は，難読化を施したサンプルのみを訓練させたものである．収集したマルウェアを用いないため，評価対象のマルウェアファミリは 9 種類から 3 種類へ絞られている．LLVM を用いた大域的な変化により，収集したマルウェアと難読化を施したサンプルには大きな違いがある．そのため，難読化を施したサンプルを用いて訓練した場合，サンプルの分類は既存のマルウェアの分類に影響を及ぼさないと考えられる．難読化されたサンプルの分類精度を上げるためには，様々なパターンでの分類を試みる必要がある．その度に収集したマルウェアも学習させていては時間がかかるため，サンプルの分類が既存のマルウェアの分類に影響を及ぼさないことを検証するために行う．

3 つ目の分類器は，難読化を施したサンプルのみを用いて，難読化機能毎に細分化を行って訓練したものである．難読化サンプルは Mirai, Lightaidra, Bashlite の 3 種類であり，各々に難読化機能の組み合わせである 8 通りの細分化を行うため，細分化されたファミリは 24 種類となる．細分化されたファミリにおいて学習を行い，分類する際に論理和演算によりマルウェア種別にまとめることで精度向上を目指す．2 つ目の分類器では，難読化を施したサンプルを 1 つのファミリとして扱っており，大雑把な学習を行っていた．難読化機能を細かく学習させていくことで，正確に特徴量を捉え精度が上がると考えられる．

データセットの分割方法と CNN モデル及び訓練方法は第 4 章で作成した評価基準となるベースライン分類器と同じである．機械学習アルゴリズムに与える検体はマルウェアファミリごとに分割し，訓練用を 7 割，テスト用を 3 割，訓練データの 1 割を検証データとして用

表 6.1: 訓練方法別の攻撃対象ファミリの精度

	Mirai_o	Lightaidra_o	Bashlite_o
ベースライン分類器	0.00	0.00	0.00
サンプルを用いた訓練	0.87	1.00	0.87
細分化を用いた訓練	0.99	1.00	0.99

いた．この時，難読化を施したサンプルは新しいファミリとして扱った．CNN のモデルは VGG19 をファインチューニングして使用し，ハイパーパラメータの調整を行った．エポック数は 30，最適化関数は SGD，バッチサイズは 6 である．学習率はエポック数に合わせて変化させ，学習の初めは 0.01，エポック数が 20 以上のときは 0.001 へと収縮させた．

6.4 評価

評価基準となるベースライン分類器，難読化を施したサンプルを用いた訓練を行った分類器，難読化機能の細分化を用いた訓練を行った分類器に対して，サンプルを分類させた際の精度を**表 6.1** に記載する．細分化を用いた訓練を行った分類器の精度は，マルウェアの種類毎に論理和演算したものである．

各モデルの分類結果の混同行列を**図 6.1**，**図 6.2**，**図 6.3** に示す．縦軸が真のマルウェアファミリであり，横軸が推測されたマルウェアファミリである．真のファミリに対する推測されたファミリの比率を数値と色で表現している．青色が濃ければ濃いほど，対応する推測が行われた検体が多い．Mirai_o，Lightaidra_o，Bashlite_o が難読化を施したサンプルを用いて学習した新しいマルウェアファミリである．**図 6.3** において，ラベル名の 3 桁は難読化機能の有無を表している．百の位は制御フローの平坦化，十の位は命令置換，一の位は偽の制御フローの追加であり，0 が難読化機能の無効化，1 が難読化機能の有効化を示す．

6.5 考察

表 6.1 で示すように，ベースライン分類器では難読化されたサンプルを全て誤分類しているが，サンプルを用いた訓練や細分化を行った訓練を行うことで高精度で分類可能であった．難読化機能毎に細分化を行い，マルウェアの種類別に論理和演算を行うことで，100% 近い推論を行うことができた．

図 6.1 より，Mirai_o が約 86%，Lightaidra_o が 100%，Bashlite_o が約 87% の精度で分類されていることがわかる．高精度で分類されているため，難読化されたサンプルを用いた訓練は対策として効果的である．しかし，Mirai_o と Bashlite_o の分類が混乱しているため，誤分類しているサンプルを調べたところ，Mirai_o の場合は偽の制御フローの追加のみ，

Bashlite_o の場合は命令置換のみで難読化を施したサンプルが誤分類していた．これらの難読化機能は学習しても誤分類する場合があるため，攻撃効果が大きいと考えられる．

図 6.1 と**図 6.2** を比較すると，分類結果が概ね同じであることがわかる．**図 6.1** の内，収集したマルウェアの分類結果において評価基準となるマルウェアの画像分類器と大きな差異がないことから，難読化を施したサンプルを使用して訓練を行った場合はサンプルの分類は既存のマルウェアの分類に影響を与えないことが示される．

図 6.3 では, 難読化機能ごとに細分化して学習させることで, **図 6.1** で誤分類していた偽の制御フローの追加のみで難読化した Mirai(Mirai001) と命令置換のみで難読化した Bashlite（Bashlite010）も高精度で分類できた．制御フローの平坦化のみで難読化した Bashlite（Bashlite100）と制御フローの平坦化と命令置換で難読化した Bashlite（Bashlite110）が誤分類しているが，誤分類の割合が高くないため制御フローの平坦化の攻撃効果はあまり大きくないと考えられる．命令置換と偽の制御フローの追加は全体に冗長な処理が追加されるためファミリの特有の領域が上手く獲得できない時があるが，制御フローの平坦化ではバイナリ形状が変化するだけであるため一度学習してしまえば対応できるようになり，誤分類したとしても同じマルウェア種に留まっていると推測される．

難読化による攻撃には，同様のサンプルを用いて訓練を行うことで対応可能である．多様な難読化手法を考慮する必要があるが，画像分類手法を用いる際にはサンプルを用いた事前対策を施すべきである．

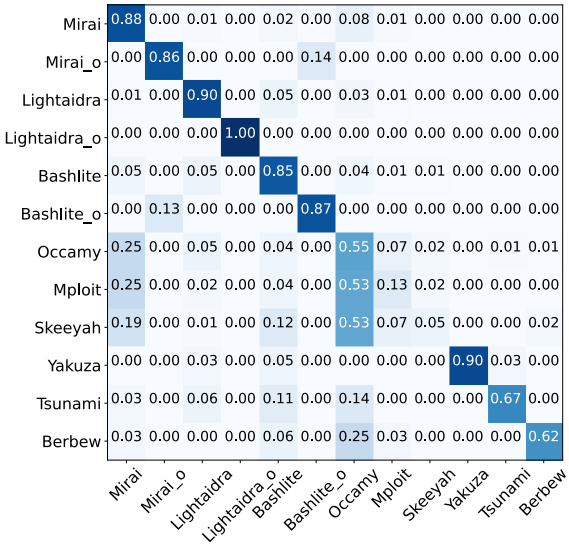


図 6.1: 難読化を施したサンプルを含めて訓練した分類器の分類結果

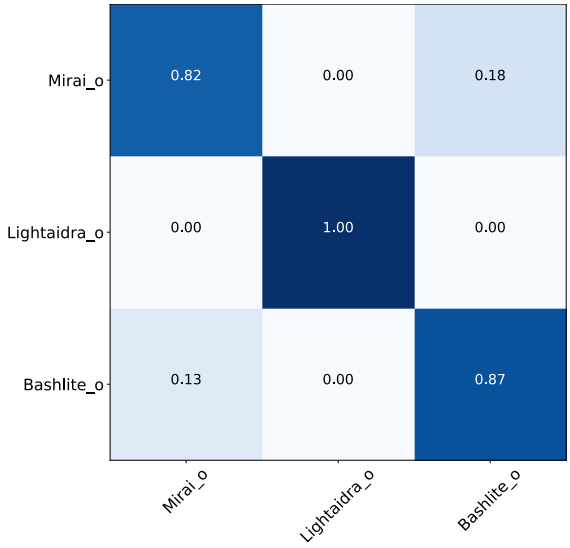


図 6.2: 難読化を施したサンプルのみ訓練した分類器の分類結果

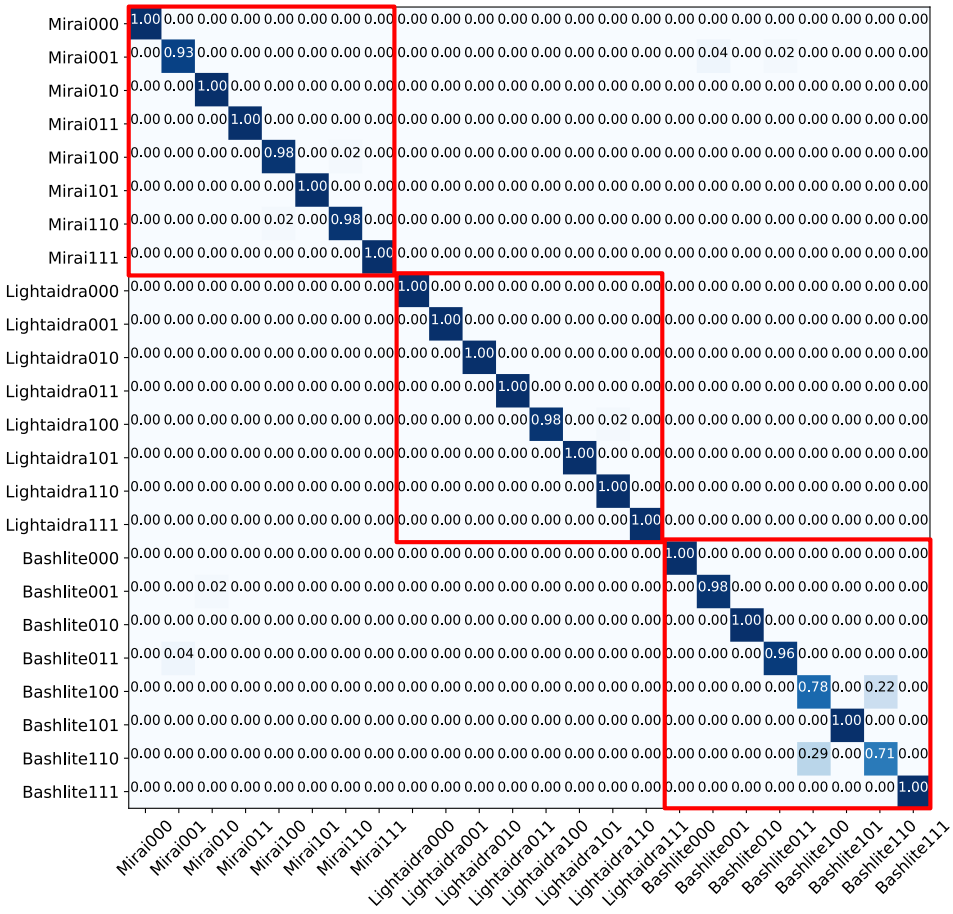


図 6.3: 難読化機能ごとに細分化して訓練した分類器の分類結果

第 7 章

結言

7.1 まとめ

本研究では，ソースコードに施す難読化処理によるマルウェアの画像分類手法への攻撃効果及び難読化を施したサンプルを用いた訓練による対処を示した．

収集したマルウェアを用いて評価基準となる画像分類器を作成し，攻撃手法として oLLVM で生成したサンプルを分類させたところ，攻撃対象のマルウェアファミリである Mirai，Lightaidra，Bashlite は全て誤分類された．難読化処理による攻撃手法への対策として，難読化を施したサンプルを含めた訓練を行ったところ，85% 以上の精度でサンプルを分類可能であることが確認できた．更に，難読化機能毎にマルウェアファミリを細分化し論理和を取ることで，難読化されたサンプルの分類は 100% に近い精度で分類可能であることを示した．

コンパイル時に適用する難読化処理は IoT マルウェアの画像分類手法への単純な攻撃として一定の効果を持ち得た．この攻撃は難読化を施したサンプルを用いた訓練により対策可能であり，難読化機能毎にファミリを細分化し論理和を取ることで精度を高めることができる．画像分類に基く分類器を用いる際には予め難読化を施したサンプルを用いて訓練させることを考慮すべきである．

7.2 今後の課題

今後の課題として，学習していない難読化機能の分類を可能にする必要がある．機器性能に制限があったため，本研究では各難読化機能を 1 度ずつ適用した検体までしかビルドしていない．マルウェアのファイルサイズを考えると妥当ではあるが，oLLVM の難読化機能は 2 度 3 度の適用など柔軟に変更できるため，高性能機器を用いることができる場合は多種多様なサンプルを用意した方が良い．サンプル数を増やすことで，学習していない難読化機能の組み合わせも分類可能になると考えられる．

本研究の目的は難読化処理による攻撃効果の検証と対策であったため，Occamy，Mploit，Skeeyah が分類できない理由は明確にすることができなかった．マルウェアの画像分類器を実用化する場合は解決する必要があるため，取り組むべき課題である．

oLLVM による難読化はソースコードを変更することなく容易に実装できるため，同様のマルウェアが増加する危険性がある．このような想定される攻撃では，先んじた対処を施すことで攻撃による被害を減らすことが可能であると考えられる．

謝辞

本研究を進めるにあたり，研究テーマの選定から論文の校閲や発表資料の添削，ゼミでのアドバイスなど貴重なご指導とご助言を賜りました公立はこだて未来大学システム情報科学部情報アーキテクチャ学科の稲村浩教授と石田繁巳准教授，京都橘大学工学部情報工学科の中村嘉隆准教授に深く感謝申し上げます．また，日頃から大変お世話になりました稲村・石田研究室とセンシングゼミの皆様，発表に際して大変有益なご指導，ご鞭撻を賜りました学生と教員の皆様に深く感謝申し上げます．

発表・採録実績

研究会（査読無し）

[1] 佐藤隼斗，稲村浩，石田繁巳，中村嘉隆：IoT マルウェアの画像分類手法への難読化による攻撃の試み，情報処理学会 研究報告モバイルコンピューティングと新社会システム（MBL）．Vol.2021-MBL-101．No.22，pp.1-6（2021）．**（WiP 奨励賞）**．

参考文献

[1] 総務省：IoT デバイスの急速な普及，入手先〈https://www.soumu.go.jp/johotsusintokei/whitepaper/ja/r03/html/nd105220.html〉（参照 2022-01-18）.

[2] トレンドマイクロ：ワーム戦争 IoT 分野におけるボットネット間の戦い，入手先〈https://bit.ly/3jUkQkk〉（参照 2021-11-10）.

[3] 総務省：IoT 機器調査及び利用者への注意喚起の取組「NOTICE」の実施，入手先〈https://www.soumu.go.jp/menu_news/s-news/01cyber01_02000001_00011.html〉（参照 2022-01-18）.

[4] IPA：情報セキュリティ 10 大脅威 2020 [組織編]，入手先〈https://www.ipa.go.jp/files/000081291.pdf〉（参照 2022-01-18）.

[5] Su, J., Vasconcellos, D.V., Prasad, S., Sgandurra, D., Feng, Y. and Sakurai, K.: Lightweight Classification of IoT Malware Based on Image Recognition, Proc. 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), pp.664-669, DOI: 10.1109/COMPSAC.2018.10315 (2018).

[6] イボットアリジャン，大山恵弘：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，情報処理学会 研究報告マルチメディア通信と分散処理（DPS），Vol.2021-DPS-186，No.25，pp.1-8（2021）.

[7] Sami, A., Yadegari, B., Rahimi, H., Peiravian, N., Hashemi, S. and Hamze, A.: Malware detection based on mining API calls, Proc. 2010 ACM Symposium on Applied Computing (SAC '10). Association for Computing Machinery, New York, NY, USA, pp.1020 – 1025, DOI: 10.1145/1774088.1774303 (2010).

[8] Junod, P., Rinaldini, J., Wehrli, J. and Michielin, J.: Obfuscator-LLVM – Software Protection for the Masses, Proc. the 1st International Workshop on Software Protection (SPRO '15), IEEE Press, pp.3 – 9 (2015).

[9] Kalash, M., Rochan, M., Mohammed, N., Bruce, N.D., Wang, Y. and Iqbal, F.: Malware Classification with Deep Convolutional Neural Networks, Proc. 2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS), pp.1-5, DOI: 10.1109/NTMS.2018.8328749 (2018).

[10] 矢倉大夢，篠崎慎之介，西村礼恩，大山恵弘，佐久間淳：CNN と注意機構による画像

化されたマルウェアの解析手法，情報処理学会 コンピュータセキュリティシンポジウム 2017 論文集， Vol.2017，No.2，pp.1381-1388（2017）．

[11] 小寺建輝，房安良和，泉隆：画像特徴量による自己防衛機能を有したマルウェアの検知に関する検討，情報処理学会 研究報告コンピュータセキュリティ（CSEC），Vol.2019-CSEC-84，No.15，pp.1-6（2019）．

[12] Nataraj, L., Karthikeyan, S., Jacob, G. and Manjunath, B.S.: Malware images: visualization and automatic classification, Proc. the 8th International Symposium on Visualization for Cyber Security (VizSec '11). Association for Computing Machinery, New York, NY, USA, Article 4, pp.1 – 7, DOI: 10.1145/2016904.2016908 (2011).

[13] llvm.org: The LLVM Compiler Infrastructure, available from〈https://llvm.org/〉(accessed 2022-01-18).

[14] Xiaolu, Z., Frank, B., Engelbert, L. and Stephen O'S.: ndroid application forensics: A survey of obfuscation, obfuscation detection and deobfuscation techniques and their impact on investigations, Forensic Science International: Digital Investigation, Vol.39, p.301285, DOI: 10.1016/j.fsidi.2021.301285. (2021).

[15] llvm.org: Clang: a C language family frontend for LLVM, available from〈https://clang.llvm.org/〉(accessed 2022-01-18).

[16] CorvusForensics: VirusShare.com, VirusShare, available from〈https://virusshare.com〉(accessed 2021-11-10).

[17] Google: VirusTotal - Home, Virustotal, available from〈https://virustotal.com〉(accessed 2021-11-10).

[18] Ding, F.: IoT Malware, github, available from〈https://github.com/ifding/iot-malware〉(accessed 2021-11-10).

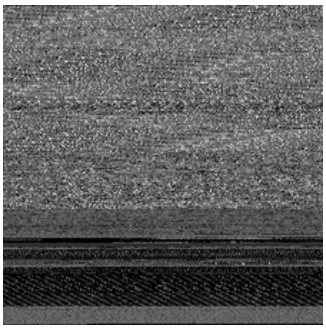
[19] Google: Help protect the Great Barrier Reef with TensorFlow on Kaggle, available from〈https://www.tensorflow.org〉(accessed 2022-01-21).

[20] Google: Keras — TensorFlow Core, available from〈https://www.tensorflow.org/guide/keras〉(accessed 2022-01-21).

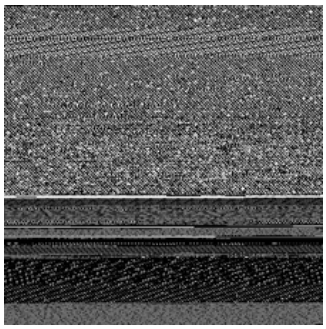
[21] Simonyan, K. and Zisserman, A.: Very deep convolutional networks for large-scale image recognition, arXiv, available from〈https://arxiv.org/abs/1409.1556〉(accessed 2021-11-10).

[22] Aurélien, G. 長尾高広（訳）：scikit-learn，Keras，TensorFlow による実践機械学習 第 2 版，オライリー・ジャパン（2020）．

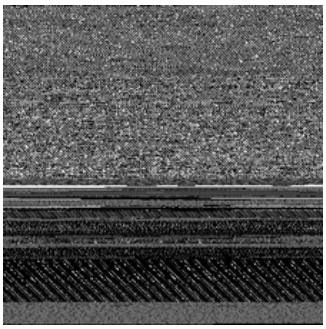
付録 A 収集したマルウェア（gcc）



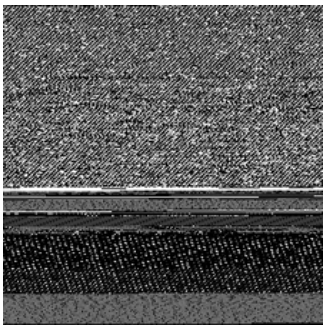
(a) Mirai



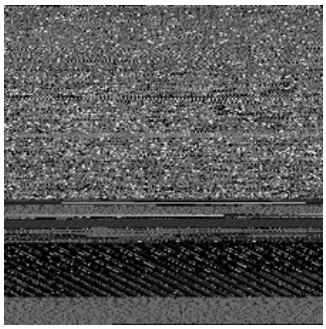
(b) Lightaidra



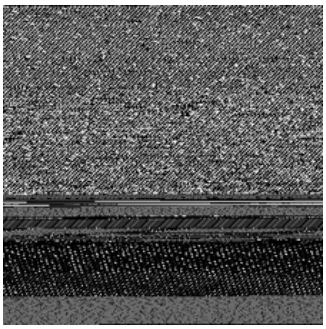
(c) Bashlite



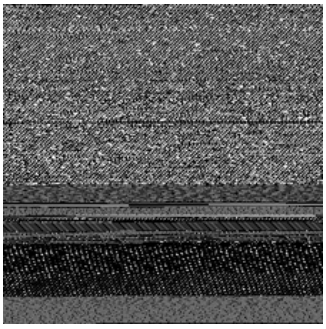
(d) Occamy



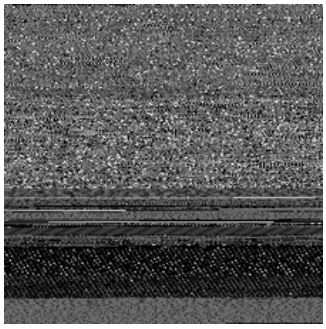
(e) Mploit



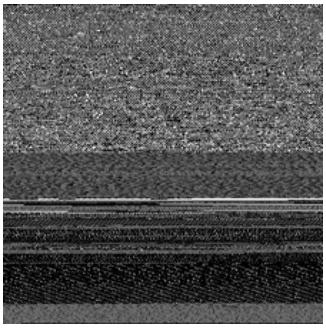
(f) Skeeyah



(g) Yakuza

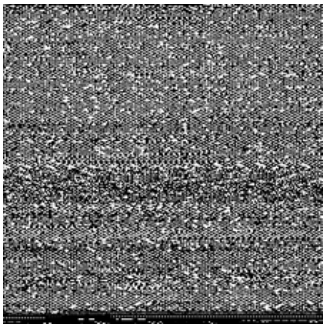


(h) Tsunami

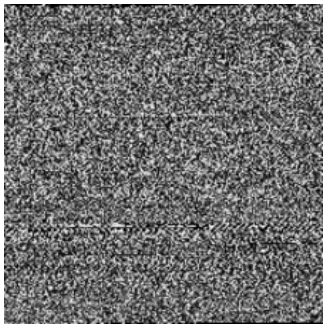


(i) Berbew

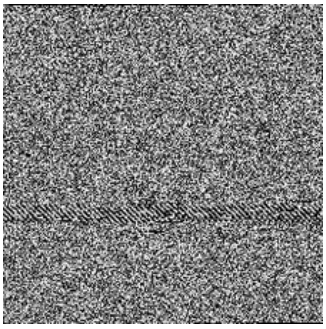
付録 B 収集したマルウェア（妨害）



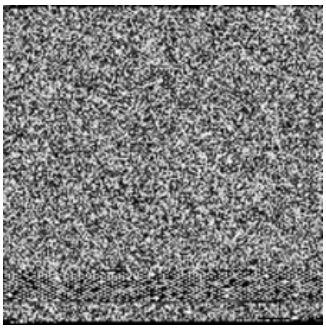
(a) Mirai



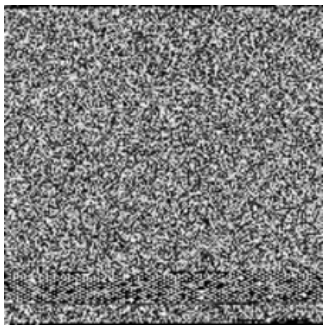
(b) Lightaidra



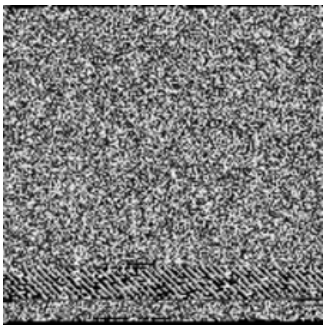
(c) Bashlite



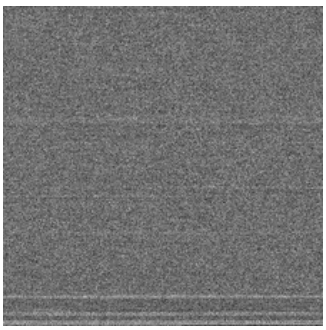
(d) Occamy



(e) Mploit



(f) Skeeyah



(g) Berbew

図目次

表目次