Vol.2021-MBL-101 No.22
Vol.2021-ITS-87 No.22
2021/12/7

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

佐藤 隼斗[1,a)]　稲村 浩[1]　石田 繁巳[1]　中村 嘉隆[2]

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，プログラムの動作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．本稿では，難読化処理による攻撃の有効性と対処の可能性を示す．難読化なしの収集したマルウェアを用いて学習した画像分類器を作成し，攻撃手法として難読化を施したマルウェア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 が全て誤分類された．この手法に対処するため難読化を施したサンプルを用いたマルウェアの画像分類器を作成し，約 60%以上で分類可能であることを確認した．

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

An Attack using Obfuscator to IoT Malware Image Classification

Hayato Sato[1,a)]　Hiroshi Inamura[1]　Shigemi Ishida[1]　Yoshitaka Nakamura[2]

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 operation of the program as an attack on the image-based classification. In this paper, we show the effectiveness of the attack by obfuscations and the possibility of countermeasures. As an attack attempt, the obfuscated malware families mirai, light aidra and bashlite were all misclassified by an image classifier that was learned using the collected malware without obfuscations. In order to cope with this attack method, we created an image classifier for malware using obfuscated samples and confirmed that it can be classified by at least about 60%.

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

1. はじめに

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 [1]．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薄い．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 マルウェアの分類では，マルウェアの画像化による分類手法が役立つ [2]．多くの IoT マルウェアは静的リン

[1] 公立はこだて未来大学
Future University Hakodate, Hokkaido, 041–8655, Japan
[2] 京都橘大学
Kyoto Tachibana University, Kyoto, 607–8175, Japan
[a)] b1018179@fun.ac.jp

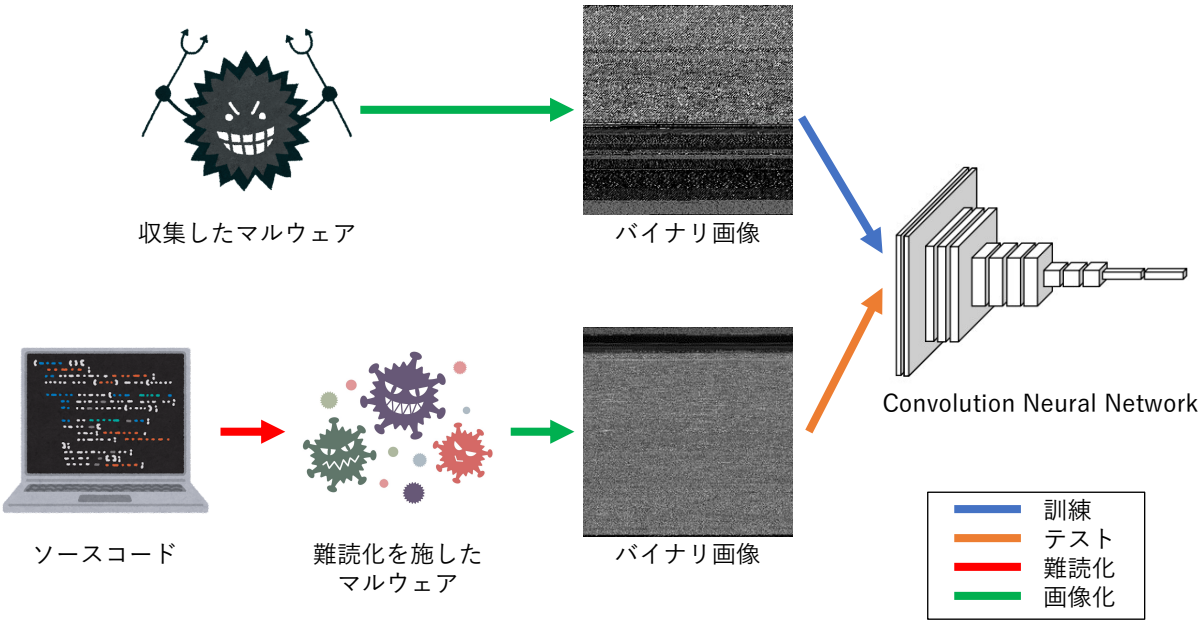


図 1: ソースコードへ施す難読化による画像分類手法への攻撃

クであり，シンボル情報も削除されている [3]．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 [4] 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が得られる [3] 報告があり，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，プログラムの動作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の手段として，ソースコードの難読化処理を検討する．

マルウェアファミリの分類に対して攻撃を受けた場合，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い．

マルウェアの画像化による分類手法の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処を検討しておくことが重要である．

本研究では，ソースコードに施す難読化を攻撃手法として想定し，その効果と対処を検討する．

2. 関連研究

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 [3], [5], [6]．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い．

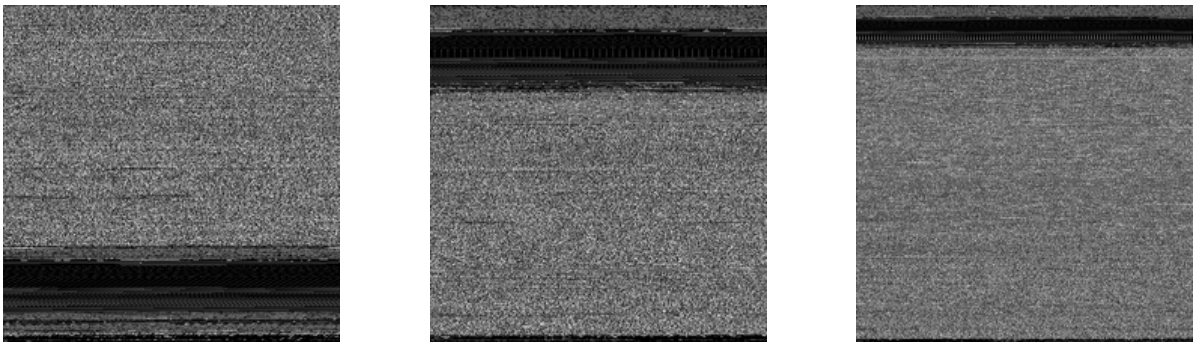
イボットらの研究 [3] では，IoT マルウェアの分類における画像化を用いた手法とシステムコール列を用いた手法の詳細な比較を行っている．彼らの研究によると，検体数が十分であれば，深層学習による画像分類手法は各システムコール命令数を用いた分類手法より約 10%精度が高く，およそ 85%の精度で分類可能であると報告されている．

矢倉らの研究 [5] 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．特有の領域とはファミリの共通の関数などを示し，同じファミリに属するマルウェア間では対応関係があり，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．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．

Nataraj らの研究 [6] 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．パッカーは共通のバイト列を多く含んでいることが知られており [5]，同様の理由から画像分類手法でも分類可能であったと考えられる．本研究では，バイナリ変更の手段としてソースコードへの難読化処理を用いて，画像分類手法に対する攻撃の有効性と対処を検討する．

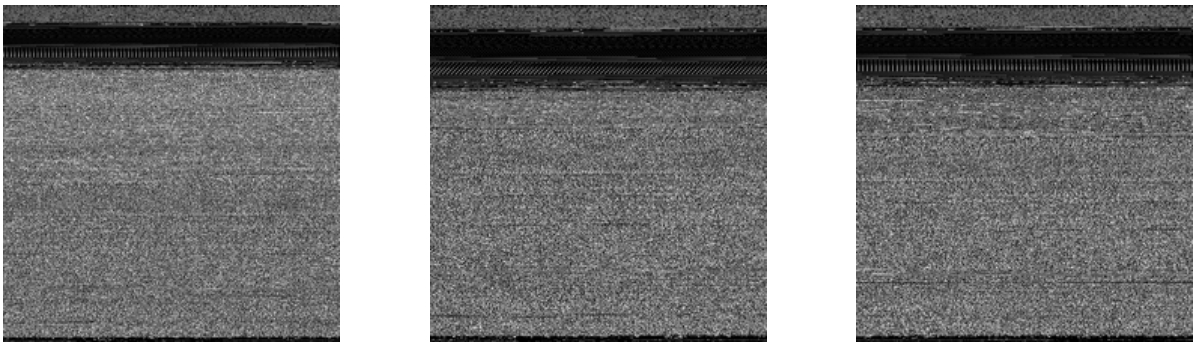
3. ソースコードへ施す難読化処理によるマルウェアの画像分類手法への攻撃及び対処

ソースコードへの難読化処理による画像分類手法への攻撃の概要を**図 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，Obfuscator-LLVM によるソースコードへの難読化処理を用いてマルウェアを生成する．難読化されたマルウェアはバイナリ表現が変化して



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

図 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: 難読化機能別のバイナリ画像

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 $image$

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．攻撃の実装方法では収集したマルウェアのテストデータを難読化されたマルウェアのバイナリ画像と入れ替えることで検証を行う．難読化処理による画像分類手法への攻撃有効性が確認された場合，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を試みる．

3.1 マルウェアの画像化

画像化アルゴリズム [3] を **Algorithm 1** に示す．本研究では，マルウェア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 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がある．

3.2 Obfuscator-LLVM による難読化を用いた攻撃方法

本手法では，Obfuscator-LLVM（oLLVM）[7] 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 は，中間表現である LLVM IR を利用するプログラミング言語に対して IR レベルで難読化を施す．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を行う．

図 2 に，gcc，clang，clang+oLLVM でビルドした IoT マルウェアを 3.1 節で示した手法でバイナリ画像化した例を示す．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．収集したマルウェアより，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．

難読化機能別のバイナリ画像を**図 3** に示す．難読化を施していないバイナリ画像は図 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 2 の clang+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．難読化処理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．

3.3 マルウェアの画像分類手法に対する攻撃の実装方法

難読化なしの収集したマルウェアを用いて学習した画像分類器を作成し，難読化を施したマルウェアを分類させることで，分類結果の違いからマルウェアの画像分類手法への攻撃効果を確認する．

マルウェアの画像分類手法への攻撃として，難読化を施したマルウェアのバイナリ画像を画像分類器のテストデータと入れ替える．このとき，oLLVM による難読化のオプションは，制御フローの平坦化を 1 回，命令置換を 1 回，偽の制御フローの追加を 1 回とする．このビルドオプションを用いるのは，他のマルウェアのファイルサイズから逸脱せず，各機能が有効であり，ビルド時間が短いからである．

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，clang（LLVM）による攻撃と clang+oLLVM による攻撃は分けて実装する．

3.4 難読化を施したサンプルを用いた訓練による対処

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．oLLVM の難読化オプションは，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 回適用する．難読化オプションの組み合わせは 8 通りである．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訓練を行う．1 通りにつき 150 検体用意し，難読化されたマルウェアファミリは 1 ファミリにつき 1200 検体となる．

表 1: 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. 評価

ソースコードが公開されている IoT マルウェアを用いて評価を行った．難読化処理による攻撃有効性の確認では，難読化による攻撃前後での分類結果の比較が必要である．そのため，IoT マルウェアの画像分類器を作成し，難読化を施したマルウェアのバイナリ画像を画像分類器のテストデータと入れ替える攻撃を行った．難読化処理による画像分類手法への攻撃有効性が確認されたため，この攻撃への対処として難読化を施したサンプルを用いた画像分類器の作成を行った．

4.1 実験に用いたマルウェア

評価に用いるデータとして，マルウェア共有サイト VirusShare [8] 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たデータセットを利用した．

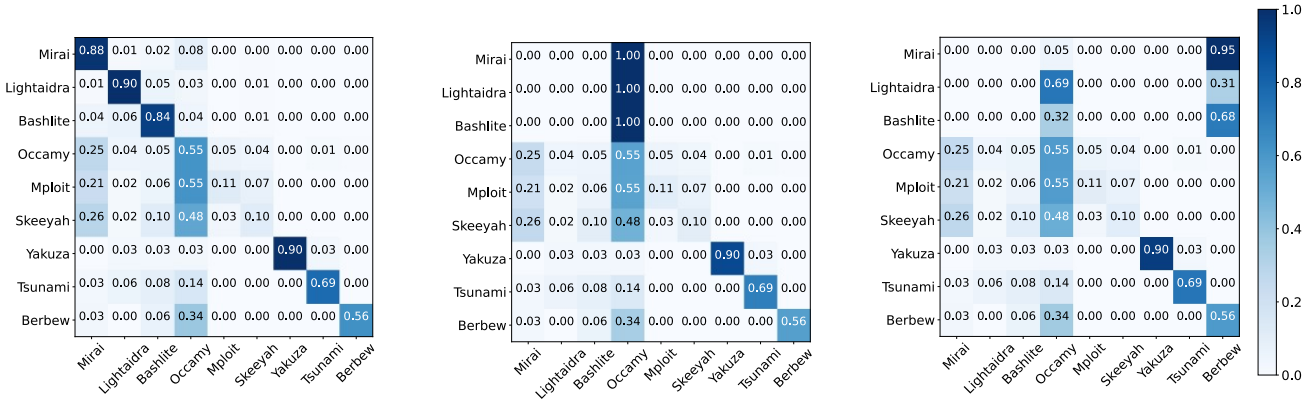
多くの IoT デバイスが ARM アーキテクチャに基づく CPU を搭載しており，IoT マルウェアの多くが静的リンクであることから，データセットの中から ARM アーキテクチャかつ静的リンクの検体のみを抽出しラベリングを行った．

マルウェアファミリのラベリングには VirusTotal [9] 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．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 1** 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 [10]，難読化処理による攻撃の際はこれら 3 種類のマルウェアのソースコードを利用する．

4.2 IoT マルウェアの画像分類器の作成

画像化したマルウェアを入力とし CNN を用いてマルウェアの画像分類器を作成した．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



(a) Original の分類結果　**(b)** clang による攻撃　**(c)** clang + oLLVM による攻撃

図 4: マルウェアファミリ分類結果のヒートマップ

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いた [3]．このとき，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 の作成には TensorFlow 及び Keras を使用した．CNN のモデルは VGG19 [11] を使用し，エポック数は 40，最適化関数は SGD，バッチサイズは 6 として訓練した．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 25 以上のときは 0.001，エポック数が 35 以上のときは 0.0001 へと収縮させた．

収集したマルウェアを用いて訓練した Original の分類器の精度は約 78%であった．先行研究 [3] 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．

4.3 難読化処理による攻撃前後の分類結果の比較

3.3 節で述べた攻撃の実装方法を基に難読化を施したマルウェアのバイナリ画像を画像分類器のテストデータと入れ替えることで攻撃を行った．

難読化による攻撃前後の混同行列をヒートマップ化した結果を**図 4** 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．

Original の分類結果と clang による攻撃及び clang + oLLVM による攻撃を比較すると，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．

clang による攻撃はランダム性がないため全て同じファミリに分類されているが，oLLVM による難読化は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．

本実験では，コンパイル時に適用する難読化処理が 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．

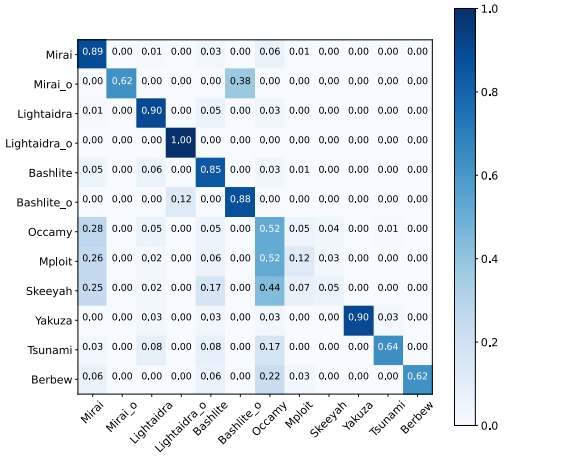


図 5: 難読化を施したサンプルを用いて訓練した分類器の分類結果

4.4 ソースコードの難読化処理に対応した画像分類器

難読化処理による画像分類手法への攻撃有効性が確認されたため，対処として 3.4 節で述べた条件で難読化を施したサンプルを用いたマルウェアの画像分類器の訓練を行った．

混同行列をヒートマップ化した結果を**図 5** に示す．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．サンプルを用いて学習したファミリのヒートマップより，精度に差があるが約 60%以上で分類可能であることがわかる．

分類精度の向上が今後の課題であるが，ソースコードの

難読化処理に対する対処として難読化を施したサンプルを用いることで分類できることが確認された．

5. おわりに

本稿では，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．難読化なしの収集したマルウェアを用いて画像分類器を作成し，攻撃手法として難読化を施したマルウェア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．この手法に対処するため難読化を施したサンプルを用いたマルウェアの画像分類器を作成し，約 60%以上の精度で分類可能であることを確認した．

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処可能なため，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．

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ことが考えられる．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．今後は難読化のオプションの変更や検体数を増加させることで，精度向上を目指した活動を行う予定である．

参考文献

[1] 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)．
[2] 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), 2018, pp. 664-669, doi: 10.1109/COMPSAC.2018.10315.
[3] 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).
[4] 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, 1020–1025. DOI:https://doi.org/10.1145/1774088.1774303
[5] 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，pp1381–1388, Vol.2017，No.2，(2017).
[6] 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, 1–7. DOI:https://doi.org/10.1145/2016904.2016908
[7] 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, 3–9.
[8] CorvusForensics: VirusShare.com, VirusShare, available from 〈https://virusshare.com〉 (accessed 2021-11-10)
[9] Google: VirusTotal - Home, Virustotal, available from 〈https://virustotal.com〉 (accessed 2021-11-10)
[10] Ding, F.: IoT Malware, github, available from〈https://github.com/ifding/iot-malware〉 (accessed 2021-11-10)
[11] 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)