Vol.2018-CSEC-83 No.9
2018/12/14

プロセッサ情報を用いたマルウェア検知機構における分類器のサイズ削減手法の検討

高瀬 誉[1]　小林 良太郎[2]　加藤 雅彦[3]　大村 廉[1]

概要：近年，IoT 機器への攻撃が増加していることが問題となっており，IoT 機器でのセキュリティ対策が急務となっている．これまでに，我々はプロセッサ情報を用いたマルウェア検知機構を提案し，プロセッサ情報によってマルウェアが検知できることを確認している．提案機構ではマルウェア分類のために機械学習を使用しているが、分類器のサイズ縮小がハードウェア化の課題となっている．本稿では，機械学習における学習データのサンプリングや特徴量の次元削減を行ない，マルウェア分類器のサイズ削減率や分類精度への影響を調査した．実験の結果から，学習データのサンプリングと特徴量の次元削減を行なうことによって，分類精度をほとんど低下させずに 97%ものサイズ削減できることを確認した．

Hayate Takase[1]　Ryotaro Kobayashi[2]　Masahiko Kato[3]　Ren Ohmura[1]

1. はじめに

近年，IoT(Internet of Things) 機器を狙う攻撃が増加しており，IoT 機器でのセキュリティ対策が重要な課題となっている．2016 年 9 月に，IoT 機器をボットネット化して DDoS(Distributed Denial of Service) 攻撃を行なう “Mirai” と呼ばれるマルウェアが出現し，米 DNS プロバイダの Dyn が攻撃を受けたことで多くの Web サービスに被害が及んだ．さらに，“Mirai” のソースコードが公開されたことにより，“Satori” や “PERSIRAI” などの IoT 機器を狙う多くの亜種マルウェアが作成され，様々な機器やサービスへの被害が拡大している [1], [2]．Kaspersky の調査によると，2018 年の上半期に，約 12 万以上のマルウェアによって IoT 機器が攻撃されたことが分かっている [3]．

IoT 機器が狙われている要因の一つに，ユーザ名やパスワードがデフォルトの状態で使用されていることが挙げられる．デフォルトでは root や admin などのユーザ名やパスワードが多く使用されるため，攻撃者はブルートフォース攻撃によって機器への侵入を試みる．侵入後は機器をボット化するとともに，他の機器にも感染するワーム活動を行なうマルウェアも存在する．さらに，現存する IoT 機器のおよそ 70%に脆弱性が存在していることも，IoT 機器を標的とした攻撃が増加している要因と考えられる [4]．

マルウェアへの対策の一つとして，アンチウイルスソフトウェアの利用が挙げられる．アンチウイルスソフトウェアは，コンピュータに保存されたファイルや実行されているプログラムを監視し，マルウェアや危険なファイルを駆除するものである．しかし，これらのソフトウェアには，あらかじめマルウェアを解析して得られたパターンファイルをもとにマルウェアを検知するシグネチャ型を採用しているものが多い [5]．シグネチャ型のマルウェア検知では，既知の解析されたマルウェアに対して有効なものの，未知のマルウェアを見逃してしまう恐れがある．そこで，未知のマルウェアを検知するために，ヒューリスティック方式やビヘイビア方式を用いたマルウェア検知手法が研究されている [6], [7]．

ヒューリスティック方式は，マルウェアを静的に解析して特徴を抽出し，その情報を基に悪意あるファイルを検知する手法である．未知のマルウェアに耐性がある一方で，プログラムの静的解析を行う必要があるため，難読化が施されたマルウェアの解析が難しいという欠点がある [8]．ビヘイビア方式は，プログラムの挙動をリアルタイムで観察し，悪意のあるプログラムを動的に検知する手法である．この手法はマルウェアの実行ファイルでなく，その挙動を元にルールを作成するため，亜種マルウェアに耐性がある．一方で，ルール作成の難しさに加え，プログラムを実行する必要があるため，検査に時間がかかってしまうという欠

[1] 豊橋技術科学大学 Toyohashi University of Technology
[2] 工学院大学 Kogakuin University
[3] 長崎県立大学 University of Nagasaki

点も持つ [8].

これらの手法を用いたマルウェア検知機構は，ファイルスキャンやプログラムの動作の監視などを行なうため，ソフトウェアとしての実装が前提となっている．しかし，IoT 機器では CPU やメモリに代表されるハードウェアリソースを多く確保することが出来ないため，IoT 機器本来のアプリケーション以外のマルウェア検知機構をソフトウェアとして実装することが難しい [9].

そこで，近年ではセキュリティ対策機構をハードウェアで実装する試みが増えている．例えば，ハードウェアの機能を拡張し，バッファオーバーフローをはじめとするメモリ上のリターンアドレスを書き換える攻撃に対策する研究が行なわれている [10]．スマートフォンなどでは，Google 社の Titan M[11] や ARM の TrustZone[12] のように，ハードウェアでセキュリティ対策を行なう機構も実装されている．このように，ソフトウェア面だけでなく，ハードウェア面でもセキュリティ対策を行なうことが重要になっている．

我々は上記に述べたような IoT 機器上におけるセキュリティの課題を解決するために，プロセッサ情報を用いたマルウェア検知機構を提案している [13], [14], [15], [16]．提案機構はランダムフォレストを用いて分類器を作成しており，学習データ数の多さなどから作成される決定木のノード数が増加し，分類器のサイズが大きくなっている．本研究では，学習済みの分類器をハードウェア実装することを想定している．しかし，IoT 機器では小型なデバイスが多いことから，サイズの大きい分類器が組み込めないことが考えられる．

本稿では，これまでの研究で明らかになった課題である，分類器のサイズ削減手法に焦点を当てる．学習させる命令数のサンプリング方法や特徴量の次元削減方法について検討し，これらの手法を適用した際の分類器のサイズ削減率やプログラムの分類精度への影響を評価する．

第 2 章では提案機構の概要及び課題について述べる．第 3 章では分類器のサイズ削減手法について述べ，第 4 章で評価を行なう．第 5 章で考察を行ない，第 6 章で本論文をまとめる．

2. 提案機構の概要及び課題

本章では，これまでに我々が提案した，プロセッサ情報を用いたマルウェア検知機構の概要と既知の課題を述べる．

2.1 プロセッサ情報の概要

本研究におけるプロセッサ情報とは，プログラム実行時にプロセッサから得られるデータを指す．**表 1** に使用するプロセッサ情報を示す．これらのプロセッサ情報は，オペコードやレジスタ番号など，バイナリを解析することによって得られる静的な情報と，プログラムカウンタやレジスタ値のように，命令実行時のプロセッサから得られる動的な情報を含んでいる．また，我々はプログラムの空間的局所性や時間的局所性に着目し，表 1 の L1_inst_hit_rate から pred_direction のような局所性を含むプロセッサ情報を追加した．dn から do の情報については，得られたオペコードを大きく 5 種類 (NOP, LOAD, STORE, JUMP, Other) に分類し，各命令種別が最後に実行されてから何命令経過したかを格納したものである．

表 1 プロセッサ情報の詳細
Table 1 Detail of the processor information

プロセッサ情報	説明
pc	プログラムカウンタ
insn	命令種別
op	オペコード
addr	ロード/ストアアドレス又は分岐先 pc
cond	条件分岐フラグ
regnum	命令で使用するレジスタ番号
regval	regnum に格納されているレジスタの値
dn	NOP の命令距離
dl	Load の命令距離
ds	Store の命令距離
dj	Jump の命令距離
do	その他の命令距離
L1_inst_hit_rate	L1 命令キャッシュの累積ヒット率
L2_hit_rate	L2 キャッシュの累積ヒット率
btb_hit	BTB による分岐のヒット/ミス
direction	実際に分岐した方向
pred_direction	gshare によって予測された分岐方向

本稿では，プログラムを実行して得られる表 1 に示した一連のデータをまとめて，トレースデータと表記する．

2.2 提案機構の概要

図 1 に提案機構の概要を示す．提案機構は，CPU のプロセッサ情報を取得する機能と，プログラムを機械学習により正常もしくは攻撃と分類する機能を持つ．既存の CPU は，2.1 節に示したプロセッサ情報をバイパスする機能を持たないため，仮想マシンによる提案機構のプロトタイプ実装を行なった．実装したプロトタイプは大きく分けて以下の 4 つのフェーズから構成される．

- 仮想マシン
- データ整形及び各命令距離の計算
- CBP(Cache/Branch Predictor) エミュレータ
- 分類器

仮想マシンにはフリーソフトの QEMU[17] を用い，ソースコードを改変して表 1 に示した pc, op, addr, cond, regnum, regval のデータを出力できるようにした．仮想マシンからプログラム実行時のプロセッサ情報を取得する際に，全て 10 進数の数値データに変換するとともに，各命令距離の計算を行なう．このデータ整形及び命令距離の

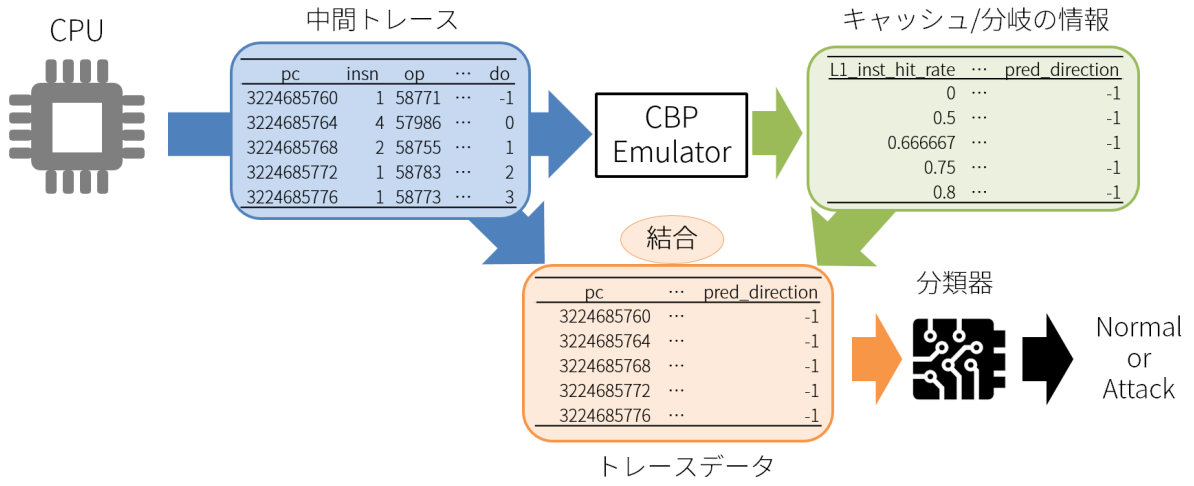

図 1　提案機構の概要
Fig. 1　The summary of our proposed mechanism

計算部分は Python プログラムによって実装し，整形済みの一連のデータを中間トレースとして出力するようにした．CBP(Cache/Branch Predictor) エミュレータは，命令キャッシュと分岐予測機構の動作をエミュレートするものであり，中間トレースの pc や addr などのデータを基に L1_inst_hit_rate から pred_direction までのデータを計算して出力する．上記の中間トレースと CBP エミュレータから得られるデータを全て結合してトレースデータを生成し，機械学習によって分類器の作成を行なう．機械学習のアルゴリズムには，教師有り学習の一つであるランダムフォレストを用いた．我々が特徴量として使用するプロセッサ情報は次元数が多いが，ランダムフォレストは高次元の特徴量においても効率的に学習できる．さらに，プロセッサ情報はカテゴリ値を多く含むが，このような場合でもスケーリング不要で学習できることからランダムフォレストが適していると考えた．なお，このプロトタイプ実装では，Python の scikit-learn ライブラリに含まれるランダムフォレストを用いた．

分類器は，あらかじめ取得した正常プログラム及びマルウェアのトレースデータを学習させて作成する．作成された分類器は，トレースデータを 1 命令単位で「正常」もしくは「攻撃」と分類する．トレースデータ内の全ての命令を分類した後，全命令数に対して攻撃として分類された命令の割合がある閾値以上であれば，そのトレースデータは攻撃として分類される．

評価に利用するトレースデータは QEMU を再起動して取り直しているため，同一プログラムのトレースデータでも，評価用と学習用のトレースデータは完全に一致しない．QEMU から取得できるトレースは対象プログラムだけでなく，OS や他のプロセスの割り込みなど，並行して動作しているプロセスのものも含んでいる．そのため，バックグラウンドで動作しているプロセスの状態に関わらずマルウェアが検知できることを確認するために，評価用と学習用のトレースデータを分けて取得するようにした．

2.3　提案機構における課題

我々は，2.2 節に示した機構をソフトウェアでプロトタイプ実装し，プロセッサ情報によって亜種のマルウェアを検知できることを確認している [14]．しかし，特徴量の次元数や学習データが大きくなることで決定木の数やノード数が増加し，分類器のサイズが大きくなるとハードウェアでの実装に影響を及ぼすことが予想される．これまでの実験では，学習させる正常プログラム及びマルウェアをランダムに選択し，それらのトレースデータの全命令を学習させていた．CPU 使用率の高いプログラムでは，得られるトレースデータの命令数が多くなることから，学習させるプログラム数を増やすごとに分類器のサイズが大きくなることが考えられる．さらに，プログラムによって得られるトレースデータの命令数にばらつきがあるため，正常と攻撃それぞれの学習させる命令数に差が生じ，分類精度に影響を及ぼす恐れもある．

これらの課題を解決するために，本稿では分類器のサイズ削減手法を検討し，その手法を用いた場合の分類器のサイズ削減率と分類精度への影響を評価する．

3.　分類器のサイズ削減手法

本稿では，以下に示す 2 種類のサイズ削減手法を検討し，評価を行なう．

- トレースデータのサンプリングによる学習
- 特徴量の次元削減による学習

本章では 2 種類のサイズ削減手法についての詳細を述べる．

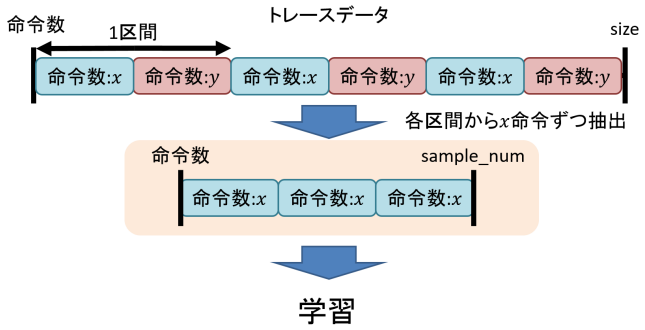

図 2　サンプリング方法
Fig. 2　The method to sample

3.1　トレースデータのサンプリング

学習時において，2.3 節で述べたようにプログラムごとに得られる命令数が異なり，学習させる正常プログラムや攻撃プログラムそれぞれのトレースデータに含まれる命令数に差が生じる．そのため，トレースデータを任意の命令数でサンプリングすることによって，学習させる正常プログラムや攻撃プログラムの命令数を統一するとともに，分類器のサイズを削減する．

サンプリング方法を**図 2** に示す．図 2 において，$size$ はトレースデータに含まれる全命令数を示している．これに加えて，各トレースデータをサンプリングする際に $sample\_num, x, y$ の 3 つのパラメータを設定した．$sample\_num$ はサンプリング数を示し，x は各区間における命令取得数，y は各区間におけるスキップする命令数であり，以下の式により求められる．

$$
\begin{aligned}
y &= size \div \frac{sample\_num}{x} - x \\
y &= \frac{size \times x}{sample\_num} - x \\
y &= x(\frac{size}{sample\_num} - 1) \qquad (1)
\end{aligned}
$$

上記の式について，各トレースデータでサンプリングする区間数を $sample\_num/x$ により求め，$size$ を求めた区間数で割ることによって各区間の命令数を求めている．各区間から x 命令を抽出し，全区間から抽出した合計 $sample\_num$ 命令を学習に用いる．

3.2　特徴量の次元削減

提案機構では，表 1 に示した 17 種類 21 次元の特徴量を使用している．機械学習において，寄与率の低い特徴量を削減することによって，精度の向上が期待される．そのため，本稿では学習用トレースデータのサンプリングに加え，特徴量の次元削減も行なうことで分類器のサイズ削減を目指す．

本研究では，分類器の作成に教師有り機械学習であるランダムフォレストを利用している．ランダムフォレストは，各特徴量の寄与率を計算することができ，特徴選択手法の一つとして広く利用されている．本稿では，ランダムフォレストにより得られた各特徴量の寄与率をもとに次元削減を行ない，分類器のサイズ削減率の評価と分類精度への影響を調査する．

次元削減方法について，全 21 次元の特徴量の寄与率をランダムフォレストにより計算し，以下 2 通りの方法で特徴量の次元削減を行なった．

- 1%未満の特徴量を除外
- 10%未満の特徴量を除外

表 2　評価環境
Table 2　Our evalation environment

Python Version	2.7.11
仮想マシン	QEMU-2.4.11
CPU	ARM1176
OS	Raspbian 2017-04(jessie)

表 3　正常プログラムの一覧
Table 3　The list of benign program

	命令数 (size)	
プログラム	学習用トレース	評価用トレース
cat	19,038	16,655
chmod	24,204	18,978
cp	29,507	32,053

4.　評価

4.1　評価環境

評価環境を**表 2** に示す．IoT 機器には，一般的に ARM アーキテクチャが多く採用されていることから，ARM 環境で実行したプログラムのトレースデータを用いて評価を行なう．そこで，仮想マシンである QEMU を用いて ARM 環境をエミュレートし，各プログラムのトレースデータを取得した．なお，本稿で用いた QEMU は，表 1 に示したプロセッサ情報を取得できるようにソースコードを改変したもので，ベースのバージョンは 2.4.1 である．エミュレートした ARM 環境上で Raspbian を実行し，この環境で正常プログラム及びマルウェアを動作させた．

正常プログラムには，Raspbian に標準でインストールされている 3 種類のコマンドをランダムに選択した．**表 3** に使用したコマンドの一覧を示す．

また，現在蔓延しているマルウェアを用いて評価するために，我々はハニーポットを構築した．ハニーポットには Cowrie[18] を選択し，ARM の環境をエミュレートすることで ARM 向けのマルウェアを捕捉できるようにした．構築したハニーポットにおいて，2018 年 7 月 31 日から 2018 年 8 月 23 日までの期間でいくつかのマルウェアを捕捉した．本評価では捕捉したマルウェアのうち，今回構築した QEMU の環境で動作を確認した 3 種類のバイナ

表 4　マルウェアの一覧
Table 4　The list of malware

Name	VirusTotal Result	Symantec Analysis Result	File Type	Packer	命令数 (size)	
					学習用トレース	評価用トレース
Kaiten	35/59	Linux.Backdoor.Kaiten	elf	none	17,255	22,017
Mirai	23/58	Linux.Mirai	elf	none	16,162	7,561
Trojan	27/59	Trojan.Gen.2	elf	none	15,928	15,373

リファイルを使用する．**表 4** に評価で利用するマルウェアのリストを示す．評価に利用する攻撃プログラムは全て VirusTotal[19] でスキャンしており，各マルウェアのファイル名は，Symantec のアンチウイルスエンジンでの検出名をもとに任意に名前を付けた．表 4 において VirusTotal Result の列は全アンチウイルスエンジンのうちマルウェアとして検知したエンジン数を示しており，Packer はパッキングソフトの種類を示している．今回利用するマルウェアは，全てパッキング処理が施されていないバイナリファイルである．

4.2　評価方法

本稿では，表 3 の正常プログラムと表 4 のマルウェアからそれぞれ 1 種類ずつ学習させた個別マルウェア分類器と，全てのプログラムを学習させた全マルウェア分類器を作成する．個別マルウェア分類器について，本稿では正常プログラムを 3 種類，マルウェアを 3 種類使用するため，合計 9 種類の分類器が作成される．はじめに個別マルウェア分類器の作成で最適なサンプリングのパラメータを調査し，得られたパラメータを用いて全マルウェア分類器を作成した場合の分類精度への影響及びサイズの削減率を評価する．なお，分類器のサイズ削減率は，サイズ削減手法を適用せずに学習した分類器のサイズを基準とし，サンプリングを行なうことによって削減できた割合を計算したものである．

サンプリングの初期パラメータは，**表 5** のように決定した．各 $sample\_num$ において，x を 100 から 10 まで変化させた場合の分類精度への影響を調査する．$sample\_num$ の最大を 15,000 としているのは，表 3 と表 4 に示したように，得られた学習用トレースの命令数から最大でサンプリングできる命令数が 15,000 のためである．各分類器において，分類精度の低下が発生しない最低サンプリング数を調査し，得られた結果をもとに最適なパラメータを設定する．このパラメータを用いてサンプリングによる全マルウェア分類器を作成し，サンプリングせず学習した場合との比較を行ない，分類器のサイズ削減率及び分類精度への影響を評価する．

さらに，サンプリングした状態で特徴量の次元削減を行ない，次元削減前後の分類器のサイズ削減率及び分類精度への影響を評価する．

表 5　サンプリングの初期パラメータ
Table 5　The parameters to sample

$sample\_num$	x
15,000	100, 50, 20, 10
12,000	100, 50, 20, 10
10,000	100, 50, 20, 10
8,000	100, 50, 20, 10
4,000	100, 50, 20, 10
2,000	100, 50, 20, 10
1,000	100, 50, 20, 10
400	100, 50, 20, 10
200	100, 50, 20, 10
100	100, 50, 20, 10

4.3　評価結果

本節では，4.2 節で述べた評価方法をもとに評価した結果を示す．はじめに，分類精度に影響を及ぼさないサンプリングパラメータの決定及び特徴量の次元削減を行ない，その後分類器のサイズ削減率を評価する．

4.3.1　サンプリングパラメータの決定

本稿で使用する $sample\_num, x, y$ の 3 つのサンプリングパラメータのうち，実験から最適な $sample\_num, x$ のパラメータを決定した．表 5 のすべての組み合わせで学習と分類を行ない，正常プログラム及びマルウェアの評価用トレースデータが攻撃として分類された割合を計算した．

各パラメータにおいて，マルウェアの評価用トレースデータが攻撃として分類された割合を**表 6** に示す．また，正常プログラムの評価用トレースデータが攻撃として分類された割合を**表 7** に示す．表 6 及び表 7 の結果は，全ての学習の組み合わせで 10 回ずつ実験を行ない，得られた結果の平均をとったものである．また，サンプリング数で full となっている行は，サンプリングせずに全ての命令を学習させた結果である．今回は，すべての x の場合で誤分類率が 10%以内に収まっているサンプリング数を最適なパラメータとして決定した．これより，表 6 において 90%を下回らず，表 7 において 10%を超えないサンプリング数は 400 であることが分かる．さらに，サンプリング数が 400 の場合に，各区間から取得する命令数を 10 とすることで正常プログラムとマルウェアの誤分類率が最も小さくなっている．そのため，今回はサンプリング数 ($sample\_num$):400，各区間から取得する命令数 (x):10 を最適なパラメータとして使用した．各トレースデータにおいて，y は 3.1 節で述

表 6　各パラメータにおけるマルウェアの分類結果
Table 6 The classification result of Malware in each parameter

	Attack rate[%]			
sample_num	x:10	x:20	x:50	x:100
full	96.395	96.395	96.395	96.395
15,000	96.595	96.049	96.555	96.479
12,000	96.356	96.441	96.780	96.687
10,000	93.261	93.563	93.559	93.672
8,000	89.456	91.836	93.030	91.364
4,000	97.209	97.504	97.843	97.023
2,000	97.675	97.818	97.616	96.921
1,000	97.725	97.353	96.130	98.053
400	97.633	96.019	96.232	98.127
200	96.299	95.766	97.119	87.980
100	96.128	92.478	82.417	62.529

表 7　各パラメータにおける正常プログラムの分類結果
Table 7 The classification result of benign program in each parameter

	Attack rate[%]			
sample_num	x:10	x:20	x:50	x:100
full	2.703	2.703	2.703	2.703
15,000	2.853	2.717	2.948	2.821
12,000	3.258	3.388	3.762	3.803
10,000	3.273	3.312	3.260	3.765
8,000	2.828	3.076	3.204	2.999
4,000	3.049	3.386	3.230	3.456
2,000	2.910	3.675	3.303	2.937
1,000	2.968	2.532	3.295	3.575
400	2.981	3.077	2.757	7.355
200	3.317	2.510	7.619	14.795
100	2.690	4.272	24.633	72.228

べた (1) 式によって計算される．

4.3.2　特徴量の次元削減

得られた最適なサンプリングパラメータを用いて全 9 種類の個別マルウェア分類器を作成し，各分類器作成時に得られる寄与率の平均を計算した．計算した平均の寄与率が高い順に並べた結果を**表 8** に示す．この結果から，寄与率が 1%未満である特徴量を削除した場合，10%未満である特徴量を削除した場合で分類器を作成し，分類精度への影響を評価した．特徴量を削減した分類器による評価用トレースデータの分類結果を**表 9** に示す．表 9 の結果も同様に，9 種類の分類器を作成し，10 回ずつ分類を行なった平均を計算したものである．特徴量を削減することで分類精度が多少低下しているが，精度の低下が 1%未満に収まっていることから，表 8 の上位 3 特徴のみの学習であっても，十分な分類精度が得られることが分かる．

4.3.3　分類器のサイズ削減率及び分類精度の測定

ここまでの実験で得られたパラメータによるサンプリング及び特徴量の次元削減を行ない，全マルウェア分類器を作成した場合の分類器のサイズ削減率を計算した．各削減

表 8　特徴量の寄与率
Table 8 The contribution rate of features

順位	特徴量	寄与率 [%]
1	L1_inst_hit_rate	40.523
2	L2_hit_rate	31.733
3	pc	12.782
4	ds	2.671
5	addr	2.558
6	regval2	2.485
7	dj	1.451
8	regval1	1.376
9	dl	0.915
10	op	0.804
11	regnum2	0.618
12	regnum1	0.614
13	do	0.512
14	insn	0.230
15	regval3	0.220
16	regnum3	0.191
17	bpred_hit_rate	0.129
18	pred_direction	0.083
19	cond	0.062
20	direction	0.042
21	dn	0.000

表 9　次元削減時の各プログラムの分類結果
Table 9 The classification result of each program at dimension reduction

	Attack rate[%]	
特徴量の次元数	正常プログラム	マルウェア
21(全特徴量)	2.981	97.633
8(1%未満削除)	3.107	97.482
3(10%未満削除)	3.462	97.103

手法を適用した場合の分類器のサイズとサイズの削減率を**表 10** に示す．表 10 において，サンプリングや特徴量の次元削減を行わない場合の分類器 (分類器 A) を基準として，サンプリングした場合の分類器 (分類器 B)，サンプリングと特徴量の次元削減の両方を行った場合の分類器 (分類器 C) とのサイズを比較している．なお，分類器 C は，寄与率が 10%未満の特徴量を削除して学習したものである．この結果より，学習する命令のサンプリングだけでも 90%以上のサイズ削減ができ，特徴量の次元を削減することでさらにサイズを半分以下に抑えられることが分かった．

また，各分類器によるプログラムの分類精度を**表 11** に示す．表 11 の全ての結果は，表 6 や表 7 と同様に分類実験を 10 回繰り返した平均を記載している．分類器 A と分類器 C の結果を比較すると，サンプリングと次元削減を行なった分類器 C で誤検知が発生する割合が微増しているが，分類結果に大きな影響を与えないと考えられるため，許容範囲内であるといえる．

表 10　分類器のサイズ削減率
Table 10　The reduction rate of classifier size

分類器の種類	サイズ [KB]	削減率 [%]
分類器 A	5,469	-
分類器 B	352	93.564
分類器 C	158	97.111

表 11　各分類器におけるプログラム分類精度の比較
Table 11　The comparison of program classification accuracy in each classifier

	Attack rate[%]		
プログラム	分類器 A	分類器 B	分類器 C
cat	1.353	1.409	1.157
chmod	2.959	3.680	3.444
cp	0.996	1.750	1.947
Kaiten	90.660	90.761	89.377
Mirai	98.685	99.854	99.310
Trojan	99.058	99.614	98.949

5. 考察

本稿での評価により，サンプリングに加えて特徴量の次元削減をしたトレースデータを学習させることによって，全トレースデータを学習させた場合と比較して，分類器のサイズを 95%以上削減できることが分かった．サンプリング方法については，表 6 と表 7 の結果より，x を小さくすることで精度を低下させずに分類器を作成することができている．このことから，トレースデータの各区間から均等に命令を抽出して学習させることによって，精度の低下を抑制できるといえる．例えば，*sample_num* が 100 で x を 100 に設定した場合，トレースデータの先頭から 100 命令だけを学習させることとなる．この場合，マルウェアのトレースデータは 63%程度しか攻撃として分類されず，正常プログラムのトレースデータも 75%以上攻撃として分類されていることから，正しく分類できていないことがわかる．しかし，*sample_num* が 100 でも，x を 10 に設定することによって精度の低下が抑えられているため，トレースデータを一定の区間に分割し，各区間から均等にサンプリングして学習させることが重要であると分かった．

6. まとめ

本稿では，これまでの我々の研究において課題となっていた，分類器のサイズ削減方法について検討し，評価を行なった．学習させるトレースデータのサンプリングを行なった場合と，サンプリングに加えて特徴量の次元削減を行った場合で，作成された分類器のサイズがどれだけ削減できるかを調査したところ，サンプリングと次元削減の両方を行なった場合で 97%サイズを削減できることが分かった．サンプリングパラメータや特徴量の次元削減方法については，あらかじめ精度に影響の無い最適なパラメータを決定したため，分類精度に影響を及ぼさずに分類器のサイズを削減することができた．

この評価により，我々の提案機構をハードウェアで実現する際のコストが削減できることが期待される．今後は，FPGA による提案機構の実装方法を検討し，ハードウェアでの実現を進める．

謝辞　本研究の一部は，JSPS 科研費 17K00076,16K00071 の支援により行った．

参考文献

[1] Trend Micro: Source Code of IoT Botnet Satori Publicly Released on Pastebin, 入手先 ⟨https://www.trendmicro.com/vinfo/sg/security/news/internet-of-things/source-code-of-iot-botnet-satori-publicly-released-on-pastebin⟩（参照 2018-11-7）.
[2] Trend Micro: Persirai: New Internet of Things (IoT) Botnet Targets IP Cameras, 入手先 ⟨https://blog.trendmicro.com/trendlabs-security-intelligence/persirai-new-internet-things-iot-botnet-targets-ip-cameras/⟩（参照 2018-11-7）.
[3] Kaspersky Lab: New IoT-malware grew three-fold in H1 2018, 入手先 ⟨https://www.kaspersky.com/about/press-releases/2018_new-iot-malware-grew-three-fold-in-h1-2018⟩（参照 2018-10-15）.
[4] P. Maki, S. Rauti, S. Hosseinzadeh, L. Koivunen, and V. Leppanen: Interface diversification in IoT operating systems, 2016 IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC), pp.304–309(2016).
[5] Forbes: Netflix Is Dumping Anti-Virus, Presages Death Of An Industry, 入手先 ⟨https://www.forbes.com/sites/thomasbrewster/2015/08/26/netflix-and-death-of-anti-virus⟩（参照 2018-11-16）.
[6] 村上純一，鵜飼裕司：類似度に基づいた評価データの選別によるマルウェア検知制度の向上，コンピュータセキュリティシンポジウム 2013 論文集，Vol.2013，No.4，pp.870–876(2013).
[7] 笠間貴弘，吉岡克成，井上大介，松本 勉：実行毎の挙動の差異に基づくマルウェア検知手法の提案，コンピュータセキュリティシンポジウム 2011 論文集，Vol.2011，No.3，pp.726–731(2011).
[8] Z. Bazrafshan, H. Hashemi, S. M. H. Fard, and A. Hamzeh: A survey on heuristic malware detection techniques, The 5th Conference on Information and Knowledge Technology, pp.113–120(2013).
[9] Z. K. Zhang, M. C. Y. Cho, C. W. Wang, C. W. Hsu, C. K. Chen, and S. Shieh: IoT Security: Ongoing Challenges and Research Opportunities, 2014 IEEE 7th International Conference on Service-Oriented Computing and Applications, pp.230–234(2014).
[10] 柴田達也，奥野航平，大月勇人，滝本栄二，毛利公一：QEMU を用いた命令拡張によるリターンアドレス書換え攻撃検知手法，研究報告コンピュータセキュリティ，Vol.2015-CSEC-68，No.33，pp.1–8(2015).
[11] Google: Titan M makes Pixel 3 our most secure phone yet, 入手先 ⟨https://www.blog.google/products/pixel/titan-m-makes-pixel-3-our-most-secure-phone-yet/⟩（参照 2018-11-7）.
[12] Arm Developer: TrustZone, 入手先 ⟨https://developer.arm.com/technologies/trustzone⟩

（参照 2018-11-7）.
[13] 小林良太郎，高瀬 誉，大谷元輝，大村 廉，加藤雅彦：機械学習を用いたプロセッサレベルでのプログラム分類に関する予備評価，電子情報通信学会技術研究報告，Vol.117，No.316，pp.5–10(2017).
[14] 大谷元輝，高瀬 誉，小林良太郎，加藤雅彦：プロセッサレベルの特徴量に着目した亜種マルウェアの検知，研究報告コンピュータセキュリティ，Vol.2018-CSEC-80，No.31，pp.1–8(2018).
[15] 小池一樹，小林良太郎，加藤雅彦：Windows におけるプロセッサレベルの特徴量に注目した亜種マルウェアの検知，コンピュータセキュリティシンポジウム 2018 論文集，pp.593–600(2018).
[16] 鈴木庸介，小林良太郎，加藤雅彦：RISC-V におけるプロセッサ情報を用いた動的なアノマリ検知機構，コンピュータセキュリティシンポジウム 2018 論文集，pp.875–881(2018).
[17] QEMU，入手先 ⟨https://www.qemu.org/⟩（参照 2018-11-7）
[18] GitHub: Cowrie SSH/Telnet Honeypot，入手先 ⟨https://github.com/micheloosterhof/cowrie-dev⟩（参照 2018-9-21）
[19] VirusTotal，入手先 ⟨https://www.virustotal.com⟩（参照 2018-9-21）