Vol.2019-DPS-178 No.14
Vol.2019-CSEC-84 No.14
2019/3/4

プロセッサ情報を用いたマルウェア検知における アルゴリズムの高速化の検討

小池 一樹[1]　小林 良太郎[1]　加藤 雅彦[2]

概要：近年，IoT 機器でのセキュリティ対策が急務となっているのは，IoT 機器への攻撃が増加していることが原因である．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．将来的には，提案機構をハードウェアにオフロードすることによって，IoT 機器における軽量なマルウェア検知機構の実現を目指している．提案機構ではマルウェア検知のために機械学習を使用しているが，取得できる全てのプロセッサ情報に対して分類を行っていた．本稿では，分類器へ投入するプロセッサ情報の削減，分類器の出力の扱い方，という視点に基づいて，検知アルゴリズムの高速化を検討する．

1. はじめに

近年，IoT（Internet of Things）機器を対象としたサイバー攻撃が増えており，IoT 機器での重要な課題としてセキュリティ対策が急務となっている．マルウェアの一種である “Mirai”は，2016 年 9 月に米 DNS プロバイダの Dyn へ攻撃を行い，多くの Web サービスに被害を及ぼすことでその認知度を高めた．それをもとに “Satori”や “Persirai” といった多くの亜種が作成され，被害が様々な機器やサービスへ拡大している [1], [2]．また，2018 年の上半期には，約 12 万以上のマルウェアが IoT 機器への攻撃を行っていることが，Kaspersky の調査で判明した [3]．

ユーザ名やパスワードが初期設定のまま私用されていることが，攻撃者が IoT 機器を狙う原因の一つである．ユーザ名やパスワードでは root や admin などといった値がデフォルトで多用されることから，ブルートフォース攻撃を用いることで攻撃者は IoT 機器への侵入を試行する．侵入後に IoT 機器を bot 化し，ワーム活動によって他の機器へ感染を広げるマルウェアも存在する．さらに，現存する約 70%の IoT 機器が脆弱性を抱えていることも，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]．

マルウェア対策の一環としてアンチウィルスソフトウェアの導入がある．マルウェアや危険なファイルを駆除するアンチウィ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中のプログラムを監視することで実現する．しかし，これらのソフトウェアには，シグネチャ方式と呼ばれる，事前にマルウェアのハッシュ値を集めたパターンファイルをもとに検知する方式を採用しているものが多い [5]．シグネチャ型は，パターンファイルに存在しない未知のマルウェアを見逃してしまう可能性がある．そこで，ヒューリスティック方式やビヘイビア方式といった未知のマルウェアを検知する手法が研究されている [6], [7]．

ヒューリスティック方式は，悪意あるファイルを検知するために，マルウェアを静的に解析し，その特徴を抽出する方式である．未知のマルウェアへの対応が可能だが，検査対象の静的解析が必要なため，難読化されたマルウェアの解析が難しいという問題がある [8]．ビヘイビア方式は，リアルタイムでプログラムの挙動を監視し，動的にマルウェアを検知する方式である．この方式は，実行ファイルではなくその挙動を基に検知ルールを作成するため，亜種マルウェアに耐性がある．しかし，プログラムを実行する必要から検査に時間を要する，ルールの作成が難しいなどの問題を持つ [8]．

これらのマルウェア検知機構は，ソフトウェアとしての実装を前提とした，プログラムの動作の監視やファイルスキャンを行う．しかし，CPU やメモリといったハードウェアリソースは IoT 機器では確保しづらく，機器に標準搭載されているアプリケーションに加えて，新たにマルウェア検知機構を実装することが難しい [9]．

近年では，ハードウェアでセキュリティ対策機構を実装する試みが増加している．例えば，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して，ハードウェアの機能を拡張することで対策する研究が行われている [10]．Google 社の Titan

[1] 工学院大学 Kogakuin Uniersity
[2] 長崎県立大学 University of Nagasaki

M[11] や ARM の TrustZone[12] のように，スマートフォンにおいても，セキュリティ対策機構がハードウェア実装されている．このように，ソフトウェアとハードウェアの両面からのセキュリティ対策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].

また，高瀬らは，本稿の研究に先立って従来の提案機構において課題であった，分類器のサイズ削減について，サンプリングを用いた分類器のサイズ削減手法を提案している [17].

本稿では，高瀬らが行った学習フェーズにおける分類器のサイズ削減に続き，分類フェーズにおける分類アルゴリズムの高速化に焦点を当てる．分類させる命令数の削減方法や，分類結果の扱い方について検討し，これらの手法を適用した際の分類器の実行時間の削減率やプログラムの分類性能への影響を評価する．

第 2 章では提案機構の概要と課題について述べる．第 3 章では分類アルゴリズムの高速化方法について述べ，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. 提案機構の概要と課題

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，バイナリを解析することで得られるオペコードやレジスタ番号などの静的な情報と，プロセッサを実際に動かすことで得られるプログラムカウンタやレジスタ値のなどの動的な情報を含んでいる．また，我々は表 1 の L1_inst_hit_rate から pred_direction のような，プログラムの空間的局所性や時間的局所性に着目できるプロセッサ情報を追加した．dn から do の情報は，オペコードを 5 種類 (NOP, LOAD, STORE, JUMP, Other) に大別し，各命令種別を最後に実行してから何命令経過したかを格納したものである．

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，仮想マシンを用いた提案機構のプロトタイプ実装を行う．大きく分けて以下の 4 つのフェーズをもってプロトタイプを構成する．

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

フリーソフトの QEMU[18] を仮想マシンとして採用し，表 1 に示した pc, op, addr, cond, regnum, regval のデータをソースコードを改変することで出力できるようにした．仮想マシンから取得されるプロセッサ情報は，すべて 10 進数の数値データに変換され，各命令距離の計算を行う．データ変換と命令距離の計算は Python によって実装され，整形済みのプロセッサ情報の集合を中間トレースとして出力する．また，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ために CBP(Cache/Branch Predictor) エミュレータを実装し，中間トレースのプロセッサ情報を基に L1_inst_hit_rate から pred_direction までのデータを計算して出力する．中間トレースと CBP エミュレータが出力したデータを結合することでトレースデータを生成し，機械学習による分類器の作成を行う．

分類器の作成には，ランダムフォレストと呼ばれる教師あり学習を採用した．ランダムフォレストは高次元の特徴量でも効率的に学習できることから，特徴量の次元数が多いプロセッサ情報を扱う提案機構には適していると考えた．さらに，カテゴリ値を多く含むプロセッサ情報に対して，ランダムフォレストはスケーリングを必要とせずに学習可能であることも採用した理由の一つである．このソフトウェアでのプロトタイプ実装では，Python のライブラリである scikit-learn で機械学習を実行した．

表 1 プロセッサ情報の詳細

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

Vol.2019-DPS-178 No.14
Vol.2019-CSEC-84 No.14
2019/3/4

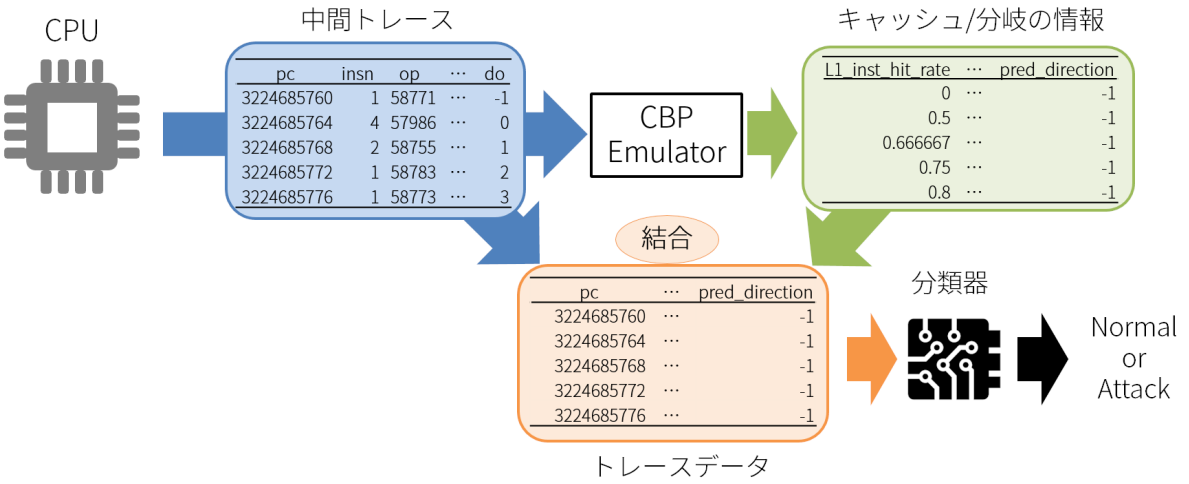

図 1 提案機構の概要

分類器は，あらかじめ取得した正常プログラムと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．

評価用のトレースデータは取り直しのため QEMU を再起動しており，同一プログラムのトレースデータでも，学習用と評価用のトレースデータは完全に一致しない．また，取得できる中間トレースは，OS や他のプロセスの割り込みなどの，対象プログラムと並行して動作しているプロセスのものも含まれる．そのため，評価用と学習用のトレースデータを分けて取得し，動作するバックグラウンドプロセスに関わらずマルウェアが検知できることを確認する．

2.3 提案機構における課題

我々は，ソフトウェアでプロトタイプ実装した 2.2 節に示した機構から，プロセッサ情報に基づく亜種のマルウェア検知が可能であることを確認している [14]．これまでの実験では，ランダムに選択した正常プログラムとマルウェアを用いて，それらのトレースデータに含まれる全ての命令を分類させていた．しかし，CPU 使用率の高いプログラムでは，単位時間あたりに分類しなければならない命令数が増加することから，プログラムによっては実行される命令の分類が追いつかなくなることが予想される．また，同様の理由から，取得した全ての命令を分類し終えるまで，トレースデータの正常/攻撃を決定することができなかった．迅速な検知が求められるマルウェア検知機構においては，アルゴリズムの動作も可能な限り高速である事が望ましい．これらの課題を解決するために，本稿では分類アルゴリズムにおける分類回数の削減手法を検討し，その手法を用いた場合の分類性能，実行時間を従来手法との比較を交えながら評価する．

3. 分類アルゴリズムの高速化手法

本稿では，以下に示す 2 種類の分類回数の削減手法（以下，削減手法と言う）を検討し，評価を行なう．

削減手法 1 トレースデータのサンプリング

削減手法 2 攻撃命令の継続に基づく予測検知

本章ではこれらの削減手法についての詳細を述べる．

3.1 トレースデータのサンプリング

1 章で述べた高瀬らの先行研究では，学習時のトレースデータのサンプリングが有効であることが示されている．これは，適切なサンプリングを行うことでプログラムの特徴を捉えるのに必要な命令数を削減することができると言い換えられる．そこで，トレースデータの分類時でも，任意の命令数でサンプリングすることで，分類器への投入命令数を削減し，分類アルゴリズムの高速化を図る．

サンプリング方法を**図 2** に示す．トレースデータに含まれる全命令数が $size$ である．加えて，任意のサンプリングパラメータとして，$sample\_num, x, y$ の 3 つを設定した．

$sample\_num$ ： サンプリングする命令の総和．

x ： 各区間での取得命令数．

y ： 各区間におけるスキップする命令数．

y は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

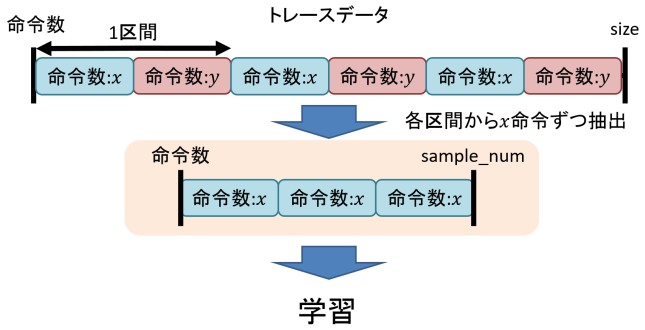

図 2　サンプリング方法

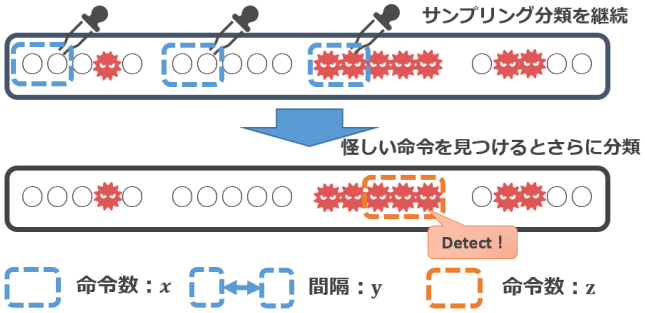

図 3　予測検知方法

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　攻撃命令の連続に基づく予測検知

1 章では，従来手法ではトレースデータ全体を分類し終えるまでプログラムの正常/攻撃を決定することできないとも述べた．提案手法では，トレースデータに含まれる攻撃命令を予測することで，トレースデータを最後まで分類することなく攻撃の検知を目指す．

予測検知の方法を**図 3** に示す．本研究では，予測検知は削減手法 1 をベースに 2 ステップの分類によって攻撃の検知を行う．ステップ 1 は，1 区間分命令を抽出し，その区間について分類を行い，また次の区間を抽出する，という動作を繰り返す．分類結果に攻撃反応があったとき，ステップ 2 に分類が進む．ステップ 2 では，ステップ 1 での当該区間に続く命令を追加で分類し，そこに含まれる攻撃命令数を求める．攻撃命令が規定値以上であるとき，攻撃が連続，集中しているとしてマルウェアの検知に至る．

本研究では各区間の命令数 x やスキップする命令の数である y は，削減手法 1 と同様の値を使用する．次章では，ステップ 2 で追加で分類する命令数 z を求めていく．

表 2　トレース取得環境

Python Version	3.5.3
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

	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

トレースデータの取得時の環境を**表 2** に示す．一般的に ARM アーキテクチャが IoT 機器で採用されていることから，プログラムのトレースデータは ARM 環境で実行したものを用いて評価を行なう．そのため，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，ソースコードを改変することで表 1 に示したプロセッサ情報を取得できるようになっており，ベースのバージョンは 2.4.1 である．QEMU 上で Raspbian を実行し，正常プログラムおよびマルウェアを動作させた．

正常プログラムには，3 種類の Raspbian 標準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，現在のマルウェア情勢を評価に反映するために，我々はハニーポットを構築した．ハニーポットには Cowrie[19] 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．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本ハニーポットで収集した．評価では収集したマルウェアのから，今回構築した QEMU 環境で動作を確認した 3 種類を使用する．**表 4** に評価で利用するマルウェアの一覧を示す．マルウェアは全て VirusTotal[20] でスキャンされており，Symantec のアンチウイルスエンジンでの検出名をもとに各マルウェアのファイル名を名付けた．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．評価には全てパッキングされていないバイナリファイルを使用する．

続いて，トレースデータの分類精度および分類アルゴリズムの実行時間の評価環境を**表 5** に示す．実行時間の計測については，実機の Raspberry Pi 上で Python で記述された分類プログラムを起動し，計測した．トレースデータや分類器のロードなどを除いた，分類器への命令投入か

表 4　マルウェアの一覧

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

表 5　分類アルゴリズムの評価環境

Python Version	3.5.3
物理マシン	Raspberry Pi Model B+
CPU	ARM1176
OS	Raspbian 2017-04(jessie)

表 6　サンプリングの初期パラメータ

$sample\_num$	x
7,000	5, 10, 50, 100
4,000	5, 10, 50, 100
2,000	5, 10, 50, 100
1,000	5, 10, 50, 100
800	5, 10, 50, 100
400	5, 10, 50, 100
200	5, 10, 50, 100
100	5, 10, 50, 100

ら検知結果の出力までを計測対象とし，ライブラリ関数 time.perf_counter() 用いて時刻の取得を行った．

4.2　評価方法

本稿では，表 3 の正常プログラム 3 種と，表 4 のマルウェア 3 種を学習させたマルウェア分類器を作成する．ここで作成する分類器は学習時にサンプリングを行っておらず，取得した命令全てを学習している．はじめに，削減手法 1（サンプリング），削減手法 2（サンプリング+予測）についてパラメータ毎の分類精度への影響を調査し，評価に適したパラメータを決定する．パラメータが決定した時点で各削減手法を適用した分類アルゴリズムの実行時間を計測し，従来手法との比較によってその削減率を評価する．

削減手法 1 におけるサンプリングの初期パラメータは，**表 6** のように決定した．各 $sample\_num$ において，x を 5 から 100 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 7,000 としているのは，表 3 と表 4 に示したように，得られた評価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 7,000 のためである．各パラメータにおいて，分類精度の低下が発生しない最低サンプリング数と各区間の命令取得数を調査し，得られた結果からサンプリング分類の評価用パラメータを決定する．

削減手法 2 における予測検知のパラメータは，削減手法 1 で採用した x 並びに $sample\_num$ を採用する．ここで調査を行う，攻撃を確定させるために追加で分類する命令数 z は，初期パラメータを 5 から 100 で変化させていく．いずれのパラメータでも，検知を確定するための攻撃命令数は，追加で分類した z 命令中，暫定的に 90% と設定する．なお，予測検知では，

- マルウェアを少ない分類で検知すること
- 正常プログラムを誤検知しないこと

これら 2 点が求められる．この 2 点を考慮しながらパラメータの分類精度への影響を評価する．

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，削減手法 1 と削減手法 2 の両方において，分類精度に影響を及ぼさないパラメータの決定を行ない，その後各削減手法を適用した分類アルゴリズムの実行時間とその削減率を評価する．

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 6 のすべての組み合わせで学習と分類を行ない，正常プログラムおよ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 7** 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 8** に示す．表 7 および表 8 の結果は，全ての分類の組み合わせで実験を行なったものである．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評価用パラメータとして決定した．これより，表 7 において 90%を下回らず，表 8 において 10%を超えないサンプリング数は 800 であることが分かる．さらに，サンプリング数が 800 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 $sample\_num$:800，各区間の取得命令数 x:10 を最適なパラメータとして使用した．また，y は 3.1 節で述べた (1) 式によって計算される．

4.3.2　予測検知パラメータの決定

4.3.1 で得られたパラメータ x:10 ならびに $sample\_num$:800 を予測検知においても採用する．**表 9** は，z の値を x 同様に 5 から 100 までで変化させていったとき，分類器が命令を分類した回数を集計したもの

表 7　各パラメータにおけるマルウェアの分類結果

	Attack rate[%]			
sample_num	x:5	x:10	x:50	x:100
7000	95.319	95.381	95.638	95.629
4000	95.408	95.275	95.308	96.400
2000	95.150	95.033	96.017	95.333
1000	94.733	95.133	94.200	93.667
800	95.500	95.167	93.167	91.625
400	95.083	95.750	89.583	93.250
200	95.333	94.000	88.167	86.500
100	93.667	93.000	76.333	73.000

表 8　各パラメータにおける正常プログラムの分類結果

	Attack rate[%]			
sample_num	x:5	x:10	x:50	x:100
7000	1.924	1.876	1.919	1.933
4000	1.975	1.817	2.517	1.425
2000	1.967	2.000	2.000	1.650
1000	2.033	2.867	2.733	2.333
800	2.000	2.458	2.333	2.917
400	2.583	2.417	4.417	5.833
200	2.667	3.667	8.833	11.667
100	4.000	5.000	17.667	23.333

表 9　予測検知における各プログラムの分類結果

	分類回数					
z	kaiten	mirai	trojan	cat	chmod	cp
100	220	220	120	800	900	900
50	120	120	70	800	850	70
10	20	40	30	800	20	30
5	15	15	25	800	15	25

である．この表から，マルウェアの分類では回数がより少なく，正常プログラムの分類では回数が $sumple_num$ の値が 800 以上かつ分類が最後まで完了する z を求める．実験結果では，z:100 以外の場合にで，誤検知に至り早期に分類が完了する結果となった．従って，誤検知が発生しなかった z:100 を予測検知でのパラメータとして採用した．なお，表中で 850 や 900 とどの分類回数が記録されているが，分類がステップ 2 に進んだものの，攻撃が確定しなかった場合このような値をとる．

4.3.3　各分類アルゴリズムの実行時間および削減率

ここまでの実験で得られたパラメータを用いてサンプリングおよび予測検知を行ない，分類アルゴリズムの実行時間と削減手法毎の削減率を計算した．その結果を**表 10** に示す．表中の削減率は，各削減手法において削減された時間を，基準となる従来手法で実行時間で割り，100 倍した値である．サンプリングのみを行った削減手法 1 では，マルウェアと正常プログラム問わず分類アルゴリズムの実行時間を半分以下に削減できた．また，サンプリングに予測検知を加えた削減手法 2 では，マルウェアの分類については削減手法 1 ほどではないが実行時間を削減できており，正常プログラムでは著しく実行時間が増加していた．

5.　考察

削減手法 2 を適用した正常プログラムの分類時，実行時間が著しく増加している原因について考察する．Python で記述したプログラムを行単位で調べたところ，引き渡す命令数にかかわらず，分類器を呼び出す毎に約 0.1 秒かかっていることが判明した．また，削減手法 2 では，1 区間分サンプリングし分類器へ投入，結果を検査する，という動作を $sample\_num/x$ 回反復することで予測検知を実現している．4.3.1 で，x:10，*sample_num*:800 としたため，反復回数は少なくとも 80 回であると求められ，正常プログラムの判別では単純計算で 8 秒以上の実行時間を要している．さらに，予測検知特有の分岐条件判定などの処理も加わることで，従来手法に対して大幅に実行時間が増加したと考えられる．マルウェアの予測検知においても表 9 中 z:100 の行にある通り，何れも 120〜220 回の分類が行われていることから 3，4 回分類器が呼び出されて入ることが読み取れる．そのため，分類する命令数は減少しているものの，削減手法 2 が削減手法 1 の削減率を上回ることがなかった．このことから，予測検知を組み合わせた高速化という視点では，分類器に投入する命令数よりも，単位時間あたりに分類器を呼び出す回数を抑えることが重要であると言える．対応策としては，現状 3，4 回の分類器の起動でマルウェアを予測検知できていることから，サンプリングの頻度はそのままに，サンプリング数回分の命令をまとめて分類器に投入するといった構想がある．また，動的にサンプリング間隔の変更，より高速な言語でのプログラムの記述，などが挙げられる．

これらの評価結果は，リソースに制限がある IoT 機器へのハードウェア実装へ向けた予備評価という位置づけである．一般的には，同じ処理であればソフトウェアよりもハードウェア実装の方が高速であり，Raspberry Pi 実機を用いたソフトウェアでの計測は，ハードウェア実装を見据えた見積もりとなる．速度の改善が示された削減手法 1 の結果からは，機構をハードウェア実装することで，CPU のスループットに対する分類のオーバーヘッドを十分に解消できると推測できる．また，削減手法 2 における実行時間の増加についても，ソフトウェアほど分類器の起動にコストのかからないハードウェア実装では，分類器の起動回数に起因する問題を解決できることが期待される．

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，分類アルゴリズムの実行時間の削減方法について検討し，評価を行なった．分類対象のトレースデータをサンプリングして分類器へ投入した場合と，サンプリングに加えて予測検知による分類アルゴリズムの早期打ち切りを

表 10　分類アルゴリズムの実行時間および削減率

分類アルゴリズム	実行時間 [sec]（削減率 [%]）					
	kaiten	mirai	trojan	cat	chmod	cp
従来手法	1.063 (0.00)	0.505 (0.00)	0.791 (0.00)	0.808 (0.00)	0.901 (0.00)	1.329 (0.00)
削減手法 1	0.258 (75.72)	0.257 (49.00)	0.257 (67.47)	0.256 (68.34)	0.260 (71.12)	0.256 (80.73)
削減手法 2	0.428 (59.78)	0.424 (15.90)	0.428 (45.85)	11.214 (-1288.64)	11.162 (-1139.06)	11.489 (-764.75)

行った場合で，それぞれパラメータが及ぼす分類精度への影響を調査し，分類精度に影響を及ぼさないパラメータを求めた．そのパラメータを採用することで，サンプリング分類では正常プログラム，マルウェアともに最大 80% 程度，マルウェアの予測検知では最大 59% 程度削減することができた．一方，正常プログラムの予測検知では，分類器への投入命令数は削減できたものの，分類器の呼び出し回数が増加したことから実行時間が大幅に増加した．この評価から，サンプリング分類の有用性は認められたものの，予測検知を併せた場合には分類器の呼び出し回数の低減や，より高速で動作する言語での実装が課題として残った．今後は，サンプリングの有用性や課題を考慮しながら，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 2019-1-31）.

[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 2019-1-31）.

[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 2019-1-31）.

[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 2019-1-31）.

[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 2019-1-31）.

[12] Arm Developer: TrustZone, 入手先 ⟨https://developer.arm.com/technologies/trustzone⟩（参照 2019-1-31）.

[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] 高瀬 誉，小林良太郎，加藤雅彦，大村 廉：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，研究報告コンピュータセキュリティ，Vol.2018-CSEC-83，No.9，pp.1–8 (2018).

[18] QEMU, 入手先 ⟨https://www.qemu.org/⟩（参照 2019-1-31）

[19] GitHub: Cowrie SSH/Telnet Honeypot, 入手先 ⟨https://github.com/micheloosterhof/cowrie-dev⟩（参照 2019-1-31）

[20] VirusTotal, 入手先 ⟨https://www.virustotal.com⟩（参照 2019-1-31）