Computer Security Symposium 2018
22 - 25 October 2018

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

小池 一樹[1]　小林 良太郎[1]　加藤 雅彦[2]

概要：コンピュータの発展による利便性の向上と同時に，コンピュータへの攻撃や情報の窃取を目的としたマルウェアの数も増加しており，その種類は多岐にわたる．また，2017 年 5 月に大流行した WannaCry を始め，マルウェアは従来よりも容易に亜種を作り出すことが可能となっている状況もマルウェア増加の一因である．その対策として，既存のマルウェア検知手法でも，その特徴的な挙動に着目したアンチウイルスソフトが多くのコンピュータで利用されている．本研究ではそれら検知手法に加わる新たな手法として，プロセッサ情報を特徴量とした機械学習による亜種マルウェアの検知を提案する．検知機構の構成要素は，プロセッサ情報の取得を行うハードウェア部と，機械学習による学習・判別を行うソフトウェア部の 2 つに分けられる．本稿では検知機構の実装へ向けた予備評価として，提案機構のハードウェア部のエミュレーションを行い，取得したプロセッサ情報の有用性を確認した．

キーワード：プロセッサ情報，亜種マルウェア，マルウェア検知

1. はじめに

近年，コンピュータに危害を加える存在であるマルウェアは多種多様なものとなっている．マルウェアとは，攻撃対象であるコンピュータに感染することで，そのコンピュータ自身を破壊したり，内部に保存されている機密情報を窃取したりする，悪意のある（Malicious）ソフトウェア（Software）である．マルウェアは，侵入経路や活動内容によって幾つかの種類に分類することができる．例えば，既にコンピュータに存在するプログラムにとりつくことで自己複製を繰り返しながら「感染」していくウイルス，独立したソフトウェアとして自己複製をするワーム，正常なプログラムに擬態してコンピュータに侵入するトロイの木馬，などが挙げられる．近年では 2017 年 5 月以降，「WannaCry」と呼ばれるランサムウェアが企業や官公庁で甚大な被害をもたらしている．WannaCry は，特定の拡張子を持つファイルを暗号化し，それらのファイルを復号することを条件に身代金を要求するワームである．また，ランダムに IP アドレスのスキャンを行い，ネットワーク上のコンピュータに侵入することで感染を拡大させる．結果として，組織の規模が大きくなるほ，どその被害も大きくなる．

昨今では，亜種マルウェアの開発が盛んに行われている．亜種マルウェアとは，既存のマルウェアのソースコードを一部改変，もしくは機能拡張して作成されたマルウェアを指す．また，バイナリファイルを暗号化して，形を変化させることで静的解析を妨害したり，アンチウイルスエンジンの検知を回避する手法を施されたものも亜種マルウェアとする．

マルウェア対策の 1 つとしてアンチウイルスソフトの利用が挙げられ，シグネチャ方式，ヒューリスティック方式，ビヘイビア方式などに分類される．シグネチャ方式では既知のマルウェアからハッシュを取得し，定義ファイルを作成する．そして，定義ファイル内のハッシュと，疑わしいファイルのハッシュを比較しマルウェアの検知を行う．この方式では正常プログラムの誤検知を低く抑えることが可能である．しかし，検知対象のマルウェアや，その亜種が増加していくことで次第に定義ファイルのサイズが膨大になり，ハッシュの比較に時間を要する．また，大量に亜種が作られるため，定義ファイルの作成が追い付かないといった問題がある．

シグネチャ方式に比べ亜種に強い手法として，ヒューリスティック方式とビヘイビア方式が挙げられる．ヒューリスティック方式は，検査対象のプログラムを静的に解析し

[1] 工学院大学
Kogakuin University 1-24-2, Nishi-shinjuku, Shinjuku-ku, Tokyo, Japan
[2] 長崎県立大学
University of Nagasaki 123, Kawashimo-chou, Sasebo-shi, Nagasaki, Japan

© 2018 Information Processing Society of Japan

て特徴を抽出し，その情報からマルウェアを検知する手法である．未知のウイルスに耐性があるが，プログラムの静的解析を行う必要があるため，難読化を施されたマルウェアについては解析が困難であるといった問題がある．一方，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．しかし，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問題がある [1].

また，これらのソフトウェアで実装される検知手法では，マルウェアの自己防衛機能が問題となる．自己防衛機能には，マルウェアが，セキュリティソフトの監視そのものを回避する機能と，自身が監視されている状況下にあることを検知し，攻撃活動を停止する機能が存在する．これらの機能は，アンチセキュリティツール，アンチサンドボックス，アンチアナリストなどを組み合わせることで実現している [2]．アンチセキュリティツールは，ウイルス対策ソフト，ファイアウォールといったツールによる検知を回避する．アンチサンドボックスは，サンドボックスと呼ばれる，コンピュータに実害が及ばない検証用の仮想環境を検知し，活動を停止する．アンチアナリストは，Process Explorer や Wire Shark といった解析ソフトウェアを検知すると同時に，プロセス監視やパッカーを使用することでリバースエンジニアリングを回避する．

本研究では既存の手法において課題となる，亜種マルウェアへの耐性，マルウェアの持つ自己防衛機能への対策手法を提案する．提案手法では，特徴量としてプロセッサから得られるハードウェア情報に注目する．そして，プロセッサ情報を入力とした機械学習を行い，マルウェアと正常プログラムの判別を動的に行う．プログラム内の 1〜数百命令単位で判別を行うため，ごく僅かなプログラムの実行時間で判別に必要なデータを取得可能である．また，ソフトウェアを介さずプロセッサから直接特徴量を得るため自己防衛機能による検知を抑制できる．さらに，1〜数百命令単位での振る舞いに着目して機械学習を行うため，パッカーなどによるバイナリレベルでの変更に対応することができる．なお，本手法は，Windows マシン上での実装を想定しているが，本稿では予備評価として，仮想マシンを用いた評価環境上で実装および評価を行う．

第 2 章では関連研究について，第 3 章では提案機構について述べる．第 4 章ではエミュレータを用いた評価環境について説明し，第 5 章で評価を行う．第 6 章で考察を行い，第 7 章で本論文をまとめる．

2. 関連研究

機械学習を用いてマルウェアを発見する手法が既にいくつか提案されている．Arvind らは，Android に対するマルウェアが実行時に要求する権限に注目し，それを特徴量とすることでマルウェアを検知する手法を提案している [3]．また複数の機械学習手法による分類精度の比較も行っている．

村上らは，プログラムを動的解析して API コールを抽出し，その名前の n-gram を用いて，マルウェアと正常プログラムの分類を行っている [4]．n-gram とは，隣り合う N 個の単語を一塊として文章を分解する手法である．さらに，評価用のデータを学習用のデータとの類似度に基づいて選別し，選別後の評価データの検出精度を向上させる手法についても提案している．

一方，マルウェアの静的解析によって抽出した特徴を用いて，機械学習によってマルウェアを分類する方法も広く行われている [5][6].

これらの機械学習を用いた手法では，API コールや権限を監視しており，ソフトウェアとしてマシン本体に実装する必要がある．これらの手法は，前述の自己防衛機能を持ったマルウェアへの耐性が低いといった問題がある．そこで，提案手法では，ソフトウェア的な情報ではなくハードウェアから取得可能な情報を用いてプログラムの特徴を抽出することで，自己防衛機能の問題を解決する．

セキュリティ機構をハードウェアとして実装する手法としては ARM プロセッサに実装されている TrustZone が挙げられる [7]．TrustZone は実行環境を Secure World と Normal World の 2 つに分けている．Normal World から Secure World へのアクセスは制限されるため，Secure World 上の認証情報や支払い情報をマルウェアから守ることが可能となる．また，TrustZone を利用したセキュリティ機構の研究も行われている [8][9]．この手法は，マルウェアから，重要なデータを守るための機構であり，マルウェア実行の検知を目的とした本研究とは異なる．

特定の処理を専用のハードウェアにオフロードすることで，高速な処理を実現する製品も存在する．SmartNIC はネットワークのパケットフィルタリングや暗号化などといった処理を，専用ハードウェアによって処理する技術である [10]．これによって CPU のリソースを他のアプリケーションに充てることが可能となる．また，OpenSSL の暗号化について，FPGA ベースにの専用ハードによってパフォーマンスの向上を図る研究も行われている [11][12].

また，本研究は，大谷，高瀬らが先行して行った，IoT 機器における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に関する研究 [13][14] の継続研究である．先行研究では，プロセッサ情報が亜種マルウェアの検知に有用であることが実証されており，本研究はその有用性に基づいて行われる．大谷，高瀬らが行った研究では，計算資源に制約のある IoT 機器での実装を想定し，検知機構全体のハードウェアへのオフロードを提案していた．それと比

© 2018 Information Processing Society of Japan

較して本研究では，計算資源に余裕のある Windows マシンでの実装を想定し，新たに追加するハードウェアはプロセッサ情報の取得部のみの最小限に留めることで，ハードウェアとソフトウェアを組み合わせた検知機構を提案する．

3. 提案手法

提案手法はプロセッサに追加される専用ハードウェアと，プロセッサ上で実行される専用ソフトウェアで構成される．

3.1 メインアイデア

専用ハードウェアをプロセッサ内部の物理コアに追加し，プロセッサ情報を取得する．取得したプロセッサ情報は複数コア間で共有される共有キャッシュに出力される．一方，専用ソフトウェアは，プロセッサ情報を取得したコアとは異なるコアで，機械学習による正常プログラムとマルウェアの判別を行う．

本稿で使用するプロセッサ情報とは，プロセッサ内部で得られる静的な情報と動的な情報で構成される．静的なプロセッサ情報とは，プログラムカウンタやオペコード，レジスタ番号など，バイナリにエンコードされている情報を指す．動的なプロセッサ情報とは，レジスタの値など，命令実行時のプロセッサの内部状態を示す情報である．これらは，静的なプロセッサ情報とは異なり，実際に命令を実行しないと取得することが出来ない．本稿ではプロセッサ情報を命令実行順に並べたものをトレースと表記する．トレースはプログラム固有のデータとして学習や判別に利用する．

提案手法では，動的なプロセッサ情報が持つ，プログラムの空間的局所性と時間的局所性に注目した．空間的局所性とは，一度参照された命令アドレスの付近が再び参照されやすいという特性であり，時間的局所性とは，一度参照された命令アドレスは短時間のうちに再び参照されやすいという特性である．空間的局所性と時間的局所性はプログラムごとに特徴が異なり，命令キャッシュや分岐予測機構の動作の違いとなって現れる．そこで，動的なプロセッサ情報を特徴量として採用することとした．

この提案手法には以下の利点が挙げられる．

- 類似した挙動の亜種マルウェアの検知
- マルウェアの挙動の高速な取得
- 自己防衛機能による逆検知の防止

1 つ目の亜種マルウェアの検知について示す．亜種マルウェアには，派生元のマルウェアと類似した関数，同じライブラリの参照が記述されている．本研究の提案手法では，1～数百命令単位で機械学習での判別を行うため，プロセッサを流れる命令列が類似する，亜種マルウェアの検知が可能である．このことは，既知の亜種マルウェアだけでなく，未知の亜種マルウェアに対しても言えることである．

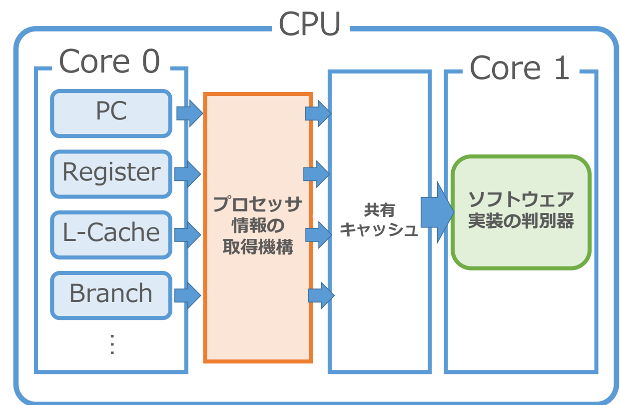


図 1: 提案機構の概要

表 1: 特徴量として使用するプロセッサ情報

プロセッサ情報	説明
insn	命令種別
op	オペコード
cond	コンディショナルフラグ
len	命令長
dn	NOP の命令距離
dj	JUMP の命令距離
do	OTHER の命令距離
inst_cache_hit_rate	命令キャッシュの累積ヒット率
L2_cache_hit_rate	L2 キャッシュの累積ヒット率
btb_hit	BTB のヒット/ミス
direction	分岐命令の分岐方向
pred_direction	分岐命令の分岐予測方向

2 つ目のマルウェアの挙動の高速な取得について示す．専用ハードウェアを用いた検知機構では，ソフトウェア実装のセキュリティソフトと比較して，マルウェアの挙動がコンピュータに反映されてからの検知ではなく，命令が実行されたその場でプロセッサ情報として命令を取得する．そのため，瞬時にマルウェアの挙動を捉えることができる．例えば，1GHz で動作するプロセッサであれば，100 μ s 程度の実行で判別に必要な特徴量を抽出できる．

3 つ目の逆検知の防止について示す．第 1 章では，マルウェアは，自己防衛機能によってマルウェアに不利益な環境を検知し，回避や活動停止をすると述べた．提案手法では，マルウェアの挙動は専用ハードウェアを通じて取得されるため，API コールやレジストリから専用ハードウェアの存在は確認することができない．

図 1 に提案機構の概要図を示す．

3.2 プロセッサ情報の概要

表 1 に特徴量として使用したプロセッサ情報を示す．

表 1 における，insn から do までは，静的なプロセッサ情報である．静的なプロセッサ情報は新たに追加する取得

© 2018 Information Processing Society of Japan

機構から取得できる情報と，そこから計算した情報に分けられる．insn は NOP，LOAD，JUMP，OTHER の 4 種類の命令種別を表す．dn から do は，取得機構から得られた情報から計算した情報であり命令の距離を表す．これらは，NOP，JUMP，OTHER の 3 種類の命令が最後に実行されてから何命令経過したかをあらわしており，それぞれ dn，dj，do に対応する．

inst_cache_hit_rate から pred_direction は，実際に命令を実行することで得られる動的なプロセッサ情報である．inst_cache_hit_rate は，命令キャッシュの累積ヒット率である．命令キャッシュは，使用頻度の高い命令を保存するためのキャッシュメモリである．また，累積ヒット率とは，処理された全命令に対してヒットした命令の割合をあらわす．L2_cache_hit_rate は，命令キャッシュの下位キャッシュである L2 キャッシュの累積ヒット率である．btb_hit は Branch Target Buffer（BTB）のヒット/ミスである．BTB とは，過去の分岐命令の実行結果を保持し，次回以降の分岐命令の実行時にフェッチすべき命令アドレスを予測する機構である．ヒットの場合は 1，ミスの場合は 0，それ以外の場合は-1 となる．pred_direction は，gshare によって予測された分岐方向，direction は実際に分岐した方向である．gshare は過去の分岐命令の結果から，分岐命令が成立となるか不成立となるかを予測する機構である．分岐成立の場合は 1，分岐不成立の場合は 0，それ以外はどちらの値も-1 となる．

3.3　判別器

判別器では，機械学習によって，あるトレースが正常プログラムのものであるか，マルウェアのものであるか判別する．機械学習のアルゴリズムにはランダムフォレストを選択した．ランダムフォレストは，複数の決定木を組み合わせて，各決定木の予測結果の多数決によって結果を得る，アンサンブル学習の一種である．高次元のデータ分類においても効率的に動作し，高次元の特徴においても効率的な学習が可能である特徴を持つ．プロセッサ情報にはカテゴリ値が多いが，このような場合でもスケーリング不要である点からランダムフォレストが適していると考えた．判別器の動作は，学習フェーズと判別フェーズに分けられる．学習フェーズでは，あらかじめ取得した正常プログラムの学習用トレースと，マルウェアの学習用トレースを使用し，判別器を作成する．判別フェーズでは判別を行いたいプログラムの判別用トレースを判別器に与える．

トレースは，先頭から順に複数の区間に分けられ，区間ごとに学習や判別を行う．区間ごとに学習することによって，各区間に対応する学習器が作成される．これらの学習器を用いて，区間ごとに判別することによって，各区間の判別結果が出力される．

図 2 に，1 つの区間における判別フェーズの流れを示す．

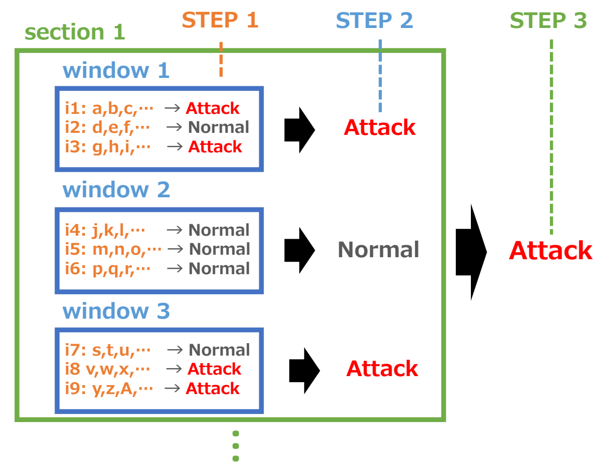


図 2: 判別フェーズの流れ

判別フェーズでは，3 つのステップを経て、最終的な正常/攻撃プログラムの判別結果を出力する．図において，緑で囲まれた領域は区間を示す．区間内において，青で囲まれた領域はウィンドウを示す．ウィンドウ内において，橙色はプロセッサ情報を示し，プロセッサ情報内の i1〜i9 は，それらのプロセッサ情報がそれぞれ命令 i1〜i9 に対応していることを示す．つまり，1 つの区間は複数のウィンドウで構成され，1 つのウィンドウは複数のプロセッサ情報で構成される．

以下に，判別フェーズの各ステップの詳細を示す．

STEP 1　命令単位での判別

機械学習による命令単位の判別を行う．プログラムの実行によって動的に得られる判別用トレース内には複数のプロセッサ情報が含まれているが，それら 1 つ 1 つを正常であるか攻撃 (マルウェア) であるか判別する．1 プロセッサ情報が 1 命令に対応している．つまり，1 命令単位で判別が行われている．

STEP 2　ウィンドウ単位での判別

1 ウィンドウ内の全命令に対する判別が終了すると，攻撃命令率を計算する．攻撃命令率とは，1 つのウィンドウに含まれるプロセッサ情報のうち，攻撃と判別されたプロセッサ情報の割合を意味する．攻撃命令率が，あらかじめ定めた閾値を超えると当該ウインドウを攻撃と判別し，下回ると正常と判別する．ウィンドウの概念を導入する目的は，機械学習による命令単位の判別の際に生じる False Positive と False Negative を吸収し，最終的な判別結果を安定させることにある．なお，STEP 2 は判別フェーズの途中である．あるウインドウが攻撃と判定されたとしても，それだけでは当該プログラムをマルウェアであると判断しない．

STEP 3　区間単位での判別

1 区間内の全ウインドウに対する判別が終了すると，攻撃ウインドウ率を計算する．攻撃ウインドウ率とは，

© 2018 Information Processing Society of Japan

1 つの区間に含まれるウィンドウのうち，攻撃と判別されたウインドウの割合を意味する．ある区間において攻撃ウィンドウ率が一定の閾値以上であれば，マルウェアが動作していると判別し，そうでなければ正常プログラムが動作していると判別する．

4. エミュレータ

この章では，使用するエミュレータについて述べる．

第 3 章までは新規にプロセッサ内に追加するハードウェア機構の存在を前提にアイデアを述べてきたが，既にあるプロセッサに対して物理的に改造を施すことは現実的ではない．そこで，本研究では提案手法におけるハードウェア部をエミュレートすることで，プロセッサ情報の取得機構を含めた環境を再現し，評価を行う．

実験環境の構成要素には，

- 仮想マシン
- CBP エミュレータ
- 判別器

が含まれる．この内, 仮想マシンについては 4.1 節で，CBP エミュレータを用いたトレースの取得については 4.2 節で説明する．

4.1 仮想マシン

仮想マシン部で取得できる，静的なプロセッサ情報のみで構成されるトレースを，中間トレースとする．この中間トレースに CBP エミュレータを利用して動的なプロセッサ情報を付加したトレースをトレースデータとする．学習・判別に用いるのは，このトレースデータである．

仮想マシンでは，プログラムを動作させ，その際の中間トレースを取得する．仮想マシンにはオープンソースのエミュレータである QEMU[15] を使用した．QEMU の-d in_asm オプションを利用することで QEMU 上で実行されている命令のアセンブリコードを取得出来る．しかし，既存のデバッグ機能では，CBP エミュレータを動作させるために必要な，命令長の情報を取得することができない．そこで QEMU のソースコードを改変し，命令長の情報を取得できるようにすると共に，データとして扱いやすいようにログの取得部分を改変した．また仮想マシンによる中間トレースの取得は python プログラムによって管理される．仮想マシン動作中に，外部の python プログラムによってシグナルを送信することで中間トレースの取得が開始される．

4.2 CBP エミュレータ

CBP エミュレータでは，仮想マシンで取得した中間トレースを使用し，命令キャッシュと分岐予測機構の動作を再現することで，仮想マシンでは取得できない動的なプロ

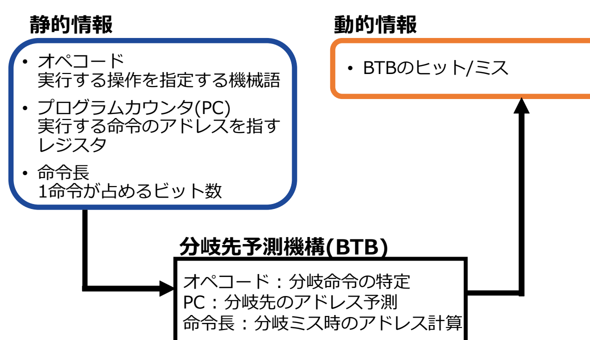


図 3: 動的なプロセッサ情報の計算（BTB の場合）

セッサ情報を付加する．CBP エミュレータは以下の 3 つの要素によって構成される．

- 命令キャッシュ
- Branch Target Buffer(BTB)
- gshare

これらのプロセッサの機構について, C 言語で実装を行った．

中間トレースに含まれる静的なプロセッサ情報から，CBP エミュレータで付加される動的なプロセッサ情報を計算する例を**図 3** に示す．動的なプロセッサ情報として，分岐先予測機構である BTB のヒット/ミスを挙げる．BTB のヒット/ミスを計算するための静的情報として，プログラムカウンタ，オペコード，命令長がある．まず，オペコードによって分岐命令を判別し，BTB を動作させるかを決定する．分岐命令であった場合，プログラムカウンタを元に BTB を動作させ，分岐先のアドレスを予測する．命令長は分岐ミスをした際のアドレスの計算に使用する．最終的に分岐先のアドレスが予測したものであれば BTB ヒット，そうでなければ BTB ミスの情報を得る．このような分岐予測機構の内部情報は実際に CPU を動作させないと得ることができないため，エミュレータを作成して CPU の動作を再現することで対応している．計算された動的なプロセッサ情報の例を**図 4** に示す．図 4 において，プロセッサ情報は命令実行順に並んでおり，1 行が 1 命令に対応している．

5. 評価

この章では評価方法及び，評価結果について述べる．

5.1 トレースの取得

本研究では，正常プログラムから取得されるトレースデータを正常トレース，マルウェアから取得されるトレースデータを攻撃トレースと呼ぶ．この正常トレースと攻撃トレースを学習させ，同じトレースをテストデータとしたとき，正しく判別できることを確認する．

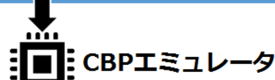
正常トレースの取得方法について示す．正常プログラムとして Microsoft Office から Word，Excel，PowerPoint，Publisher，Access，Outlook の中間トレースを取得する．

© 2018 Information Processing Society of Japan

静的なプロセッサ情報

命令実行順↓

pc	insn	op	tpc	cond	len
135121635	4	137	0	0	2
135121637	3	235	135121658	0	2
135121658	4	139	0	0	2



動的なプロセッサ情報

L1_inst_hit_rate	L2_hit_rate	btb_hit	direction	pred_direction
0.790000	0.190476	−1	−1	−1
0.792079	0.190476	0	1	1
0.794118	0.190476	−1	−1	−1

図 4: 動的なプロセッサ情報の例

また，インターネットブラウザとして Internet Explorer, Google Chrome の中間トレースを取得する．仮想マシンの OS には Windows7 32bit SP1 を選択した．ランサムウェアが暗号化するファイルを用意するため，デスクトップ上に 500MB のファイルを用意した．このファイルは，コマンドプロンプトの fsutil コマンドを用いて作成したファイルである．中間トレースは，仮想マシンの電源を入れ，デスクトップ画面の表示が完了し，正常プログラムを起動した時点で取得開始する．また，中間トレースは 5 つの区間に分けて作成され，1 区間の命令数は 500K 命令とする．区間と区間の間に 500K 命令のインターバルを挟み，これを計 5 回繰り返す．よって，動的情報を加えた正常トレースは，１つの正常プログラムにつき 5 つ存在する．

攻撃トレースの取得方法を示す．攻撃プログラムとして，マルウェアの検体を配布している Web サイトから入手した Ransomware.WannaCry と，Ransomware.Jigsaw を使用した．この 2 つのマルウェアは，ウイルス検証サイト [16] にて当該マルウェアであることを確認した．トレースの取得方法は，基本的に正常プログラムと同じである．しかし，マルウェアの判別結果をより詳細に評価するため，区間数を 5 から 10 へ増加させた．取得した正常，攻撃プログラムの中間トレースは，CBP エミュレータを通して，動的なプロセッサ情報が特徴量として追加される．この特徴量が追加された攻撃データが学習，判別に利用される．

5.2 評価方法

学習フェーズでは，マルウェア 1 種類につき 10 個の判別器を作成する．この判別器はそれぞれ 10 区間のトレースに対応しており，1 つの判別器に 1 つの区間を学習させる．正常トレースは，8 種類のプログラムから 2 区間ずつ選択し，すべての判別器に学習させる．全体で 20 個の判別器を作成する．判別器と判別するトレースについて以下に示す．

10 個ずつ作成した WannaCry と Jigsaw の判別器に対して，

(1) WannaCry のトレースデータ（10 区間）

(2) Jigsaw のトレースデータ（10 区間）

(3) 正常トレース（8 種× 5 区間）

これらのトレースデータを判別させる．(1)，(2) では，10 個の判別器に対して 10 個のトレースデータが存在するため，100 通りの結果が出力される．(3) では判別器に対してトレースデータが 40 個存在するため 400 通りの結果が出力される．判別時の基準となる攻撃命令率と攻撃ウィンドウ率は，前者で 90%，後者で 10%の閾値を設けた．また，ウィンドウサイズは 500 とした．

5.3 評価結果

WannaCry のトレースデータの判別結果を**表 2(a)**，**表 2(b)** に，Jigsaw のトレースデータの判別結果を**表 3(a)**，**表 3(b)** に，また，正常トレースデータの判別結果の一部を**表 4(a)**，**表 4(b)** に示す．

表 2(a) と表 3(b) を例に表の見方を説明する．表 2(a) では，WannaCry の区間 2 を学習した判別器（判別器 2）は，区間 1 に対して 62.7%という結果を示している．また，判別器 8 は区間 6 に対して 31.7%という結果を示している．一方，表 3(b) は，判別器 7 は区間 5 に対して 11.7%という結果を示している．表の数値について，黒字での表記は，攻撃ウィンドウ率が 10%未満であり正常と判別された区間である．赤字での表記は，攻撃ウィンドウ率が 10%以上であり，攻撃と判別された区間である．

学習済みマルウェアの判別では，WannaCry を学習した判別器（表 2(a)），Jigsaw を学習した判別器（表 3(a)）どちらも検知に成功している．また，異なるマルウェアの判別では，WannaCry を学習した判別器で 2 区間（表 3(b)），Jigsaw を学習した判別器で 4 区間（表 2(b)），検知に成功している．表 4(a)，表 4(b) は，正常プログラムにおいて誤検知された区間を含む例である．この他に，スペースの都合上表の掲載を見送るが，WannaCry の判別器において 2 区間，Jigsaw の判別器において 1 区間，誤検知された区間が存在した．

6. 考察

初めに，表 2(a)，表 3(a) について述べる．この 2 つの表では，対角線上に 100%が並んでいる．これは，学習済みの区間において全てのウィンドウが攻撃と判別されたということである．このことから，ウィンドウの導入によって，機械学習による検知ミスを抑制できたことが分かる．また，対角線周辺の多くの区間において，未学習であるにもかかわらず，10%以上の数値が得られている．この原因として，区間を跨いで，類似または同じ命令が繰り返し実行されることが考えられる．表 2(a) において，WannaCry

© 2018 Information Processing Society of Japan

表 2: WannaCry の判別結果

(a) 判別結果（判別器: WannaCry）

区間	判別器 0	1	2	3	4	5	6	7	8	9
0	100	70.0	59.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	13.0	100	62.7	55.6	31.4	17.9	0.0	0.0	0.0	0.0
2	38.5	91.5	100	45.3	18.2	1.8	4.4	1.6	0.0	0.0
3	0.2	79.7	68.4	100	47.5	0.0	0.0	1.4	0.0	0.0
4	0.0	53.2	35.7	43.8	100	16.0	49.4	0.5	27.7	36.9
5	0.0	28.3	2.1	1.5	39.7	100	56.9	0.2	8.3	20.1
6	0.0	14.8	0.0	0.0	43.1	64.6	100	0.0	31.7	47.8
7	0.0	0.0	0.0	5.1	0.0	4.4	1.0	100	0.0	0.0
8	0.0	0.0	0.0	0.0	58.2	78.8	88.6	0.0	100	69.8
9	0.0	0.0	0.0	0.0	40.8	75.9	88.7	0.0	46.2	100

数値: 攻撃ウインドウ率 [%]
色: 区間に対する判定結果（黒: Normal, 赤字: Attack）

(b) 判別結果（判別器: Jigsaw）

区間	判別器 0	1	2	3	4	5	6	7	8	9
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.7	0.0	0.0
1	3.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	21.7	21.7	25.0	21.7	0.0	0.0	0.0	0.0	0.0	0.0
3	3.3	0.0	1.7	1.7	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	3.3	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	1.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

数値: 攻撃ウインドウ率 [%]
色: 区間に対する判定結果（黒: Normal, 赤字: Attack）

表 3: Jigsaw の判別結果

(a) 判別結果（判別器: Jigsaw）

区間	判別器 0	1	2	3	4	5	6	7	8	9
0	100	15.4	52.4	16.5	0.0	0.0	0.0	0.0	0.0	0.0
1	26.2	100	6.7	1.0	0.0	0.0	0.0	7.2	1.9	0.8
2	23.3	22.9	100	75.4	0.0	9.2	0.0	0.0	1.6	0.0
3	12.3	4.8	23.9	100	30.8	58.3	0.0	8.7	4.7	0.3
4	0.0	0.0	0.0	87.9	100	14.3	0.0	0.0	0.0	0.0
5	0.0	0.0	3.9	68.1	9.2	100	0.0	0.0	0.3	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	100	0.0	4.1	0.0
7	0.0	6.7	0.1	0.2	0.0	0.0	0.0	100	43.0	2.2
8	0.0	3.6	0.1	0.2	0.0	0.0	0.0	18.8	100	12.2
9	0.0	2.6	0.0	0.0	0.0	0.0	0.0	1.0	62.6	100

数値: 攻撃ウインドウ率 [%]
色: 区間に対する判定結果（黒: Normal, 赤字: Attack）

(b) 判別結果（判別器: WannaCry）

区間	判別器 0	1	2	3	4	5	6	7	8	9
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	6.7	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	20.0	1.7	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	11.7	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

数値: 攻撃ウインドウ率 [%]
色: 区間に対する判定結果（黒: Normal, 赤字: Attack）

表 4: 正常プログラムの判別結果（一部）

(a) 判別結果（判別器: WannaCry, 判別トレース: IE）

区間	判別器 0	1	2	3	4	5	6	7	8	9
0	11.7	3.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

数値: 攻撃ウインドウ率 [%]
色: 区間に対する判定結果（黒: Normal, 赤字: Attack）

(b) 判別結果（判別器: Jigsaw, 判別トレース: Outlook）

区間	判別器 0	1	2	3	4	5	6	7	8	9
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	10.0	3.3	3.3	0.0	0.0	0.0	0.0	0.0	0.0

数値: 攻撃ウインドウ率 [%]
色: 区間に対する判定結果（黒: Normal, 赤字: Attack）

© 2018 Information Processing Society of Japan

を学習した判別器 4 は，10 区間中 8 区間で最終的に攻撃と判別している．そこで，区間 4 のトレースデータを観察してみると，ADC 命令が 8 回続いた後，TEST 命令，OR 命令，NOT 命令，JNP 命令の順で，ルーチンを終始繰り返していることが確認できた．このルーチンを取得している時点でのデスクトップ上の変化では，デスクトップに配置したファイルの暗号化を確認している．このことから，WannaCry の動作フェーズの 1 つである暗号化活動の特徴を捉えられており，暗号化活動がトレース取得の終了まで広範囲に渡って行わていると推測できる．

続いて表 4(a)，表 4(b) について述べる．この 2 つの表は，正常トレースの判別結果の一部であるが，今回定めた攻撃ウィンドウ率の閾値では，攻撃と判別される区間が存在している．考えられる原因として，意識的に実行したプログラムとは別に，バックグラウンドで動作する OS プログラムの存在がある．OS プログラムは常に動作しており，この命令が混ざった攻撃トレースを学習した判別器が，正常トレース内の OS プログラムの命令に反応したと考えられる．対策として，攻撃ウィンドウ率に対する閾値を上方修正する，または該当判別器の利用を避ける，といった方法が挙げられる．特に後者については，表 2(a)，表 3(a) の結果も考慮すると，誤検知が少なく，広い範囲をカバーできる判別器を数個用意すればマルウェアの検知が可能である，という事が言える．実際に利用を考えたとき，例えば表 2(a) の判別器 1 ならば，同プログラムの 10 区間中 7 区間の検知が可能であり，かつ，正常トレースに対して誤検知も起こらないため，マルウェアの検知に有効である．

7. まとめ

多種多様に増加していく亜種マルウェアに対する従来の検知手法の問題点を補うべく，Windows マシンにおける専用ハードウェアとソフトウェアを組み合わせたマルウェア検知手法を提案した．専用ハードウェアでは，プロセッサ内部の情報を直接取得し，ソフトウェア部分で機械学習による判別を想定した．本稿では提案手法の予備評価という位置づけで，ハードウェア部のエミュレートを行い，機械学習による正常プログラムとマルウェアの判別が可能であることを確認した．今後は，今回用意できなかったマルウェアの亜種検体を入手し，先行研究で実証されている亜種マルウェアに対する提案手法の有用性を確認することが課題となる．また，プロセッサ情報の取得を手動で行っていたため，より現実に近い環境を想定した，リアルタイムに判別可能な機構の実装を検討する．

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] 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.

[2] McAfee Blogs, An Overview of Malware Self-Defense and Protection, https://securingtomorrow.mcafee.com/mcafee-labs/overview-malware-self-defense-protection/ (参照 2018-08-03).

[3] A. Mahindru and P. Singh. Dynamic Permissions based Android Malware Detection using Machine Learning Techniques, Proceedings of the 10th Innovations in Software Engineering Conference, pp.202-210, 2017.

[4] 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精度の向上, コンピュータセキュリティシンポジウム 2013 論文集 2013(4), pp.870-876, 2013.

[5] F. Adkins, L. Jones, M. Carlisle, and J. Upchurch. Heuristic malware detection via basic block comparison, 2013 8th International Conference on Malicious and Unwanted Software: "The Americas" (MALWARE), pp.11-18, 2013.

[6] P. Khodamoradi, M. Fazlali, F. Mardukhi, and M. Nosrati. Heuristic metamorphic malware detection based on statistics of assembly instructions using classification algorithms, 2015 18th CSI International Symposium on Computer Architecture and Digital Systems (CADS), pp.1-6, 2015.

[7] TrustZone - Arm Developer, https://developer.arm.com/technologies/trustzone (参照 2018-01-29).

[8] A. M. Azab, P. Ning, J.Shah, Q. Chen, R. Bhutkar, G. Ganesh, J. Ma, and W. Shen. Hypervision Across Worlds: Real-time Kernel Protection from the ARM TrustZone Secure World, Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pp.90-102, 2014.

[9] L. Guan, P. Liu, X. Xing, X. Ge, S. Zhang, M. Yu, and T. Jaeger. TrustShadow: Secure Execution of Unmodified Applications with ARM TrustZone, Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services, pp.488-501, 2017.

[10] G. Sabin and M. Rashti. Security offload using the SmartNIC, A programmable 10 Gbps ethernet NIC, 2015 National Aerospace and Electronics Conference (NAECON), pp.273-276, 2015.

[11] A. Thiruneelakandan and T. Thirumurugan. An approach towards improved cyber security by hardware acceleration of OpenSSL cryptographic functions, 2011 International Conference on Electronics, Communication and Computing Technologies, pp.13-16, 2011.

[12] J. K. T. Chang, S. Liu, J. L. Gaudiot, and C. Liu. Hardware-assisted security mechanism: The acceleration of cryptographic operations with low hardware cost, International Performance Computing and Communications Conference, pp.327-328, 2010.

[13] 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, 情報処理学会研究報告 Vol.2018-CSEC-80, No.31, pp.1-8, 2018.

[14] 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, 電子情報通信学会信学技報 ICSS, Vol.117, No.316, pp.5-10, 2017.

[15] QEMU. https://www.qemu.org/ (参照 2018-01-09).

[16] VirusTotal. https://www.virustotal.com (参照 2018-08-15).

© 2018 Information Processing Society of Japan