ハードウェアベースでのマルウェア検知における bit 分割と幅削減による分類器のサイズと生成時間の削減

豊島 千尋[1,a)]　小林 良太郎[1]　加藤 雅彦[2]

概要：近年，新種や亜種といった形でマルウェアは増加し，その被害も増え続けている．既知のマルウェアであれば，パターンマッチングなどで過去のデータと照合し特定のパターンが存在するかどうかで検知することができる．ただこの手法では，増加している新種や亜種のマルウェアの検知は難しい．そこで，本稿では機械学習を使ったマルウェアの検知を採用している．機械学習を使ったマルウェア検知は，過去のマルウェアや正常なプログラムのデータを学習させ分類器を作成し，その分類器を使って判定を行う．この手法はデータのパターンではなく特徴から検知を行うため，既知のマルウェアだけでなく新種や亜種のマルウェアの検知にも有効である．本稿では機械学習を用いてマルウェア検知を行い，学習するファイルの増量に伴い発生する，データ量の増加への対処法として計算による特徴量の bit 幅削減と，特定の bit 数で特徴量を区切り必要な特徴量を選択する手法を提案し，それを評価した．評価した結果，ほとんど精度を維持したまま分類器のサイズを約 67% 削減することができた．

キーワード：機械学習, マルウェア, IoT

Reduction of Classifier Size and Generation Time by Bit Partitioning and Width Reduction in Hardware-Based Malware Detection

Chihiro Toyoshima[1,a)]　Ryotaro Kobayashi[1]　Masahiko Kato[2]

Abstract: In recent years, the number of new types and variants of malware has been increasing, and the damage caused by such malware is also increasing. If there is known malware, we can detect it by comparing it with past data using pattern matching or other methods so that we can determine whether a specific pattern exists or not. However, it is difficult to detect the new types and variants of malware using this method. Therefore, in this article, we use machine learning to detect malware. Malware detection using machine learning involves learning data from known malware and normal programs, and creating a classifier to make a decision. This method is effective not only for detecting known malware, but also for detecting new types and variants of malware, because detection is based on features rather than patterns of data. In this article, we propose and evaluate a method for detecting malware using machine learning. To solve the problem of growing data volume caused by increasing the number of files to be trained, we propose a method for reducing the bit width of features by calculation and a method for selecting necessary features by separating features with a specific bits. As a result, we were able to reduce the size of the classifier by approximately 67% while maintaining nearly the same accuracy.

Keywords: Machine Learning, Malware, IoT

1. はじめに

近年，マルウェアによる被害は増加しており，様々な種類や亜種などが存在している．マルウェアが攻撃の対象

[1] 工学院大学
Kogakuin University
[2] 順天堂大学
Juntendo University
[a)] j121212@ns.kogakuin.ac.jp

とするのはパソコンだけではなく，IoT 機器が対象となる事もある．IoT とは従来インターネットに接続されていなかった様々なモノがネットワークを通じてサーバーやクラウドサービスに接続され，相互に情報交換をする仕組みである．IoT により遠隔で機器を操作できるようになった一方で，この特性を悪用し IoT 機器に対して様々な手法のサイバー攻撃が仕掛けられている．IoT 機器に侵入することを目的としたマルウェアは，侵入に成功した IoT 機器を遠隔操作できるようにし，そこから LAN でつながっている他の IoT 機器に対しても管理者権限を奪うなど感染を広げる性質をもつものが存在する．様々な技術が生まれ IT や機械が進化している中，IoT 機器の供給も増加しており，その台数は 2023 年には 370 億台以上の数を記録している [1]．またこの数は今後さらに増加していくことが予想されており，IoT 機器を狙ったマルウェアへの対策がさらに求められるようになると考えられる．Guarnizo らの研究 [2] では，9 か国 16 都市の 39 のワームホールインスタンスを使用した大規模な実験を行い，提案した SIPHON アーキテクチャを実証した．実証では，6 台の物理 IP カメラ，1 台の NVR，および 1 台の IP プリンターを用い，その結果，2 か月間で 1 日あたり 700 MB ものトラフィックを集めた．その一部から Mirai マルウェアの資格情報が発見された．Pour らの研究 [3] では，誤った構成のトラフィックをサニタイズし観察を行った結果，1,787,718 個のスキャナの内，441,766 個が感染した IoT 機器から発信されたことが分かった．また，その内 41.9% がポート 23 番，23.9% がポート 80 番，19.7% がポート 8080 番を狙ったものであると分かった．我々はこのような IoT マルウェアに対してハードウェアベースのセキュリティとして Anti-Malware Hardware (AMH) を提案している [4]．一般的なマルウェア検知機構はソフトウェアベースで行う事を前提にしており，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できない IoT 機器では，機器本来のアプリケーションと並行して実装することが難しい．AMH は IoT 機器のハードウェアリソース不足によるこの問題の解決策として提案されている．AMH は LSI 上にプロセッサコアと一緒に実装され，プログラムを実行した際のプロセッサ情報をもとに機械学習ベースでのマルウェア検知を行う．機械学習において大量のデータの学習が精度向上に有効であるが，学習データ量が増加することで，分類器サイズと生成時間は増加していく．そこで本研究では分類器サイズと生成時間の増加を抑える手段として，既存特徴量の bit 幅削減と bit 分割による新規特徴量の作成を行う．まず第 2 章で関連研究について述べる．第 3 章で提案手法について述べ，第 4 章で提案手法を用いた評価について述べる．それをもとに第 5 章で結果及び今後の課題について述べる．

2. 関連研究

ハードウェアでセキュリティ機構を実装する手法として TrustZone [5] や Control-flow Enforcement Technology (CET) [6] などが挙げられる．ARM プロセッサで使用される TrustZone は，コンピュータリソースを非セキュアな環境とセキュアな環境に分割して安全な実行環境を提供することを目的としている．センシティブなデータの取り扱いはセキュアな環境で行い，非セキュアな環境からセキュアな環境に不正にアクセスできないように操作することで，データの取り扱いを安全に行うことを可能としている．また，CET は Intel のチップレベルでのセキュリティ機能であり，CPU のジャンプテーブルを使用するソフトの機能の制限や，アプリケーションソフトが目的とする制御フローのコピーを作り CPU の安全な領域に格納し，ソフトの実行順序で不正が行われないようにする事でデバイスを保護する．ただし，これらはデータの保護を目的としておりマルウェアの検知は目的としていない．藤原らの研究では既存機構である AMH に対して，DDoS 攻撃や Brute-Force 攻撃の検知も行えるように変更を行っている [7]．また動的プロセッサ情報として平均特徴量や空間特徴量を作成しこれらを利用してマルウェアを検知する Anti-Malicious Communication Hardware（AMCH）を提案している．本稿では既存機構である AMH を利用してマルウェア検知を行い，その機械学習部分を精度向上させることを目的としている．この既存機構ではハードウェアベースでマルウェア検知を行う．この時，ハードウェアにオフロードしやすい情報としてプログラカウンタやオペコードといった静的なプロセッサ情報と，レジスタの値やキャッシュヒット率といった動的なプロセッサ情報を用いてトレースを作成し，RandomForest を利用した機械学習を行う．

マルウェアを機械学習で検知するにあたって，大量のデータ学習が精度向上に向けて有効である．この際に課題となるのが大量のデータを扱うことで分類器のサイズや生成時間が増加することであり，これらを解決するために，判定精度を維持しつつ，増加するデータ量を効率的に処理することが求められる．n-gram は連続した n 個の項目のグループを指し，テキストやコードから特徴を抽出する方法であり，この抽出したデータからパターンや傾向を分析することができる．Liu らの研究ではマルウェアの検知において n-gram モデルを利用してマルウェアプログラムからオペコード機能を抽出し特徴量を作成している [8]．連続した n 個のオペコードがどのように出現するのかに基づきマルウェアの特性を捉え，従来のシグネチャベースの方法よりも高い精度でマルウェアが分類できることを示している．連続している文字列を抽出することでデータを分類できるという点から，本研究では既存機構の特徴量 pc に

ついて，その数値を決まった bit ずつで区切り新規特徴量を作成する．

3. 提案手法

3.1 特徴量 bit 幅削減

機械学習において学習させるデータが多くなると，データ処理の時間が増加し分類器の作成にかなりの時間がかかる事や，メモリ不足によって分類器の作成ができなくなる事が考えられる．学習データの削減にはサンプリングという手法があり，分類器作成時に学習に利用するデータ量の上限を設けたり，データで使用する命令列を規則に削減したりすることができるが命令列を削除することで判定精度に大きく影響する命令列を除いてしまう可能性がある．そこで本研究では特徴量の bit 幅の削減を行う．小数でない特徴量の bit 幅削減には以下の式を利用する．なお，a には元のデータの bit 数を，n には削減後の bit 数を代入し，これを対象の特徴量のデータ x に掛ける．また，特徴量が 0 から 1 の間の小数である場合は値が小さすぎるため上記公式をそのまま用いると判定精度が低下してしまう．そこで，特徴量に一定の値を乗算することとした．なお，本研究では実験により乗算する値を 10^{16} と定めた．

$$y = \frac{2^n - 1}{2^a - 1} \cdot x \tag{1}$$

例として削減対象 x の値が 3226793112 である場合を挙げる．この場合 x を 2 進数で示すと 32 bit の数値に変換され，$n = 32$ となる．削減後の bit 数 a を 16 bit として計算し，bit 数が削減された値を求める．

$$\begin{aligned} x &= 3226793112 \\ &= (11000000010101001111010010011000)_2 \\ y &= \tfrac{2^{16}-1}{2^{32}-1} \cdot 3226793112 \\ &\approx 49236 \text{（小数点以下切り捨て）} \\ &= (1100000001010100)_2 \end{aligned}$$

3.2 特徴量の bit 分割と部分抽出

特報量 pc（プログラムカウンタ）は実行する命令のメモリアドレスを示し，同じメモリ領域で命令が実行される際には領域内で小さな数値の変化が繰り返され，実行するメモリ領域が変化することで数値の上位部分が変化する．これはプログラムが通常，連続したメモリアドレスに配置された命令を順に実行するため，上位部分は変化せず，下位部分が実行する命令のバイト数分だけ増加するためである．このように上位部分と下位部分に変動の差がある事を利用して，**図 1** の手順で 8 bit ずつに数値を区切り，4 つの新規特徴量 pc1，pc2，pc3，pc4 を作成する．特徴量 pc の数値が 32 bit 以下の場合は，足りない桁を上位の bit から 0 埋めし 32 bit に変換する．毎命令ごとに数値が変化する pc3 や pc4（下位部分）と比較し，pc1 や pc2（上位部分）はメモリ領域が移動するタイミングで変化する傾向にあり，領域のアドレスの特徴を継承する．また，作成した 4 つの特徴量を比較し，寄与率の低い特徴量を排除し，高い寄与率の特徴量を利用することで，使用する特徴量の bit 幅を削減し分類器サイズや演算時間を削減する．

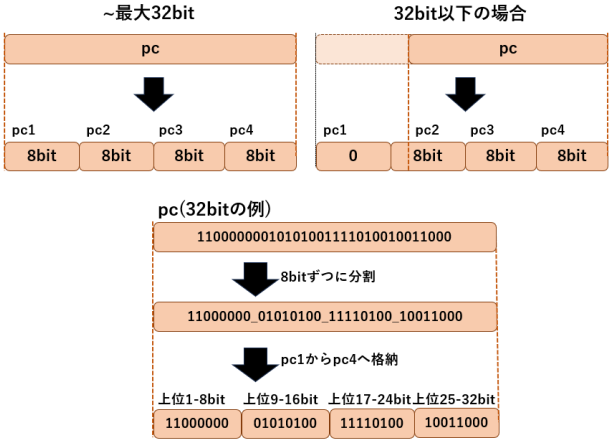

図 1: 特徴量 pc の bit 分割方法

Fig. 1 Bit Partitioning Method for Feature PC

4. 評価

4.1 評価環境

シミュレーションを行うハードウェア，ソフトウェアのスペックは以下のとおりである．

- ホスト OS: Ubuntu 22.04.4-amd64
  - CPU: Intel Core i5-12450H
  - メモリ: DDR4-3200 64 GB
- エミュレータ: QEMU 5.0.0
  - ゲスト OS: 2020-05-27-raspios-buster-lite-armhf.img
  - マシン: Versatilepb
  - CPU: Arm1176
  - メモリ: 256 MB

本研究では先行研究と同様に，QEMU [9] というオープンソースのプロセッサエミュレータを使用し実機を想定したシミュレーションを行った．QEMU でゲスト OS を起動し，起動したゲスト OS でマルウェアや正常プログラムを実行しトレースの採取を行った．なお，その際に実行したプログラムは sftp を用いてホスト OS から QEMU に送信し使用した．各ソフトウェアのバージョンは Python 3.10.2，QEMU 5.0.0 を使用している．また，QEMU では次の分岐命令までの一連の命令シーケンスを TransLation Block（TB）と呼ばれるブロックで管理していて，プロセッサ情報として出力する際にも TB が適用されているため，適用前のものを利用した．

4.2 評価手順

本研究の評価は，Scikit-learn ライブラリの Random Forest を利用し分類器を作成して行う．本研究ではプログラムを 2 クラスに分類し，正常プログラムを Normal クラスに，悪性プログラム（マルウェア）を Attack クラスに分類する．正常プログラムと悪性プログラム（マルウェア）共に，ゲスト OS にて実行しトレースを採取する．各クラスの実行方法は表 1 に示すように，Normal クラスに分類するトレースは複数のコマンドを実行して採取し，Attack クラスに分類するトレースは IoTPOT のデータセットから実行できるマルウェアを調査した後，実行し採取する [10], [11]．各プログラムを 10 回ずつ実行し，1 プログラム当たり 10 個のトレースを採取する．10 個のトレースのうち 9 個を学習に使用し 1 個を判定に使用する．また，表 2 には学習時に用いる特徴量を示す．なお，既存機構で使用している特徴量は pc，op，addr，cond，pc_mean，op_mean，addr_mean，gshare_predict，gshare_hit_rate，L1_inst，L1_data，L2 の 12 種であり，本研究ではそれに追加して新規特徴量として pc1，pc2，pc3，pc4 と，すべての既存特徴量に対して bit 削減を行った特徴量を合わせた 24 種を使用する．また，この特徴量において寄与率の高い特徴量を抜粋して，判定精度を維持しながらデータ量の削減を目指す．評価において使用する各トレースの命令数は上限を設けず，サンプリングする場合は採取したデータの最初の命令列から 4 命令おきに抽出する．学習及び判定において各トレースにはあらかじめラベルを付与し，正常トレースには 0 を，マルウェアトレースには 1 を付与し実行した．

表 1: 既存機構での分類器作成と判定結果
Table 1 Classifier Creation and Decision Results with Existing Mechanisms

クラス	実行方法 (ゲスト OS で実行)
Normal	ls
	bison
	cp
	ifconfig
	mkdir
	rm
	flex
	tar
	gcc
	tar
	chmod
	ip a
Attack	./でプログラムファイルを実行

4.3 評価指標

判定スコアは 0 から 1 までの数値で表し，トレース内で判定した結果の攻撃命令率を表す．判定スコアが 0 に近い値であればより正常プログラムであることが予測され，悪性プログラム（マルウェア）は 1 に近いスコアになる．RandomForest を使用した分類器の作成と判定には各プログラムトレース 20 回分を使用し，実施内容に合わせてその割合を調節する．また閾値を 0.5 で設定し，判定スコアが 0.5 以上のトレースに対して「悪性（マルウェア）」判定を，0.5 を下回るトレースに対しては「正常」判定を行う．判定に使用するトレースは初めに 25 種類のプログラムにて，それぞれ 1 トレースずつ抜き出し，本研究の全実施内容において，この抜き出した 25 個のトレースを利用した．

表 2: 使用する特徴量
Table 2 Features Used

特徴量名	内容
pc	プログラムカウンタ
op	オペコード
addr	ロードストアアドレス，又は分岐先アドレス
cond	条件分岐フラグ
pc_mean	pc の平均
op_mean	op の平均
addr_mean	addr の平均
gshare_predict	分岐予測された方向
gshare_hit_rate	分岐予測ヒット率
L1_inst	L1 命令キャッシュの累計ヒット率
L1_data	L2 データキャッシュの累計ヒット率
L2	L2 キャッシュの累計ヒット率
pc1	pc の上位 8bit
pc2	pc の上位 9bit 目から 16bit 目
pc3	pc の上位 17bit 目から 24bit 目
pc4	pc の上位 25bit 目から 32bit 目
pc_red	bit 削減を行った pc
op_red	bit 削減を行った op
addr_red	bit 削減を行った addr
cond_red	bit 削減を行った cond
pc_mean_red	bit 削減を行った pc_mean
op_mean_red	bit 削減を行った op_mean
addr_mean_red	bit 削減を行った addr_mean
gshare_predict_red	bit 削減を行った gshare_predict
gshare_hit_rate_red	bit 削減を行った gshare_hit_rate
L1_inst_red	bit 削減を行った L1_inst
L1_data_red	bit 削減を行った L1_data
L2_red	bit 削減を行った L2

5. 結果

5.1 既存機構の特徴量による評価

既存機構の特徴量として，表 2 の pc，op，addr，cond，pc_mean, pc_mean, op_mean, addr_mean, gshare_predict, gshare_hit_rate，L1_inst，L1_data，L2 の 12 種を用いて分類器の作成と判定を行った．この結果表 3 の 1 行目の結

果が得られ，各特徴量の寄与率は表 4 のようになった．さらに，分類器作成にかかる時間やデータ量を減らすことを目的として，表 4 から上位 6 種の特徴量（pc_mean，pc，op_mean，addr_mean，L1_inst，L2）を選択し，分類器の作成と判定を行い，表 3 の 2 行目の結果が得られた．これにより，正解率・適合率の精度を維持しつつ，使用する特徴量を選抜し，分類器サイズや分類器生成時間の削減を行うことができた．

表 3: 既存機構での分類器作成と判定結果
Table 3 Classifier Creation and Decision Results with Existing Mechanisms

特徴量数	分類器サイズ	生成時間	正解率	適合率
12	105,357,025	578.95 s	94.40 %	92.31 %
6	102,538,016	490.81 s	97.60 %	95.38 %

表 4: 既存機構での特徴量寄与率
Table 4 Contribution Ratio

特徴量名	寄与率	特徴量名	寄与率
pc_mean	33.07 %	cond	3.69 %
pc	21.29 %	gshare_hit_rate	3.25 %
op_mean	11.03 %	gshare_predict	2.69 %
addr_mean	7.74 %	addr	2.69 %
L1_inst	7.32 %	op	1.38 %
L2	5.76 %	L1_data	0.10 %

5.2 特徴量の bit 削減と bit 分割後の評価

表 4 に示した特徴量のうち寄与率が高い 6 つに対し 3.1 節で提案した手法で bit 削減を行った．本稿では，削減前に最大 32 bit あった特徴量データを 16 bit に削減して新規特徴量を作成し，分類器の作成を行った．その結果，各特徴量の寄与率の変化は表 5 のようになり，bit 削減前に 1 番寄与率の高かったプログラムカウンタの特徴量 pc の値が大幅に低下し，addr_mean の寄与率が上昇した．また，分類器作成時のサイズと時間，正解率，適合率は表 6 の結果となり，分類器のサイズは表 3 の bit 削減前と比べて約 85% 削減され，分類器生成にかかる時間も約 33 % 短縮された．適合率は bit 削減前から数値は変化せずその精度を維持できたが，正解率の低下がみられた．

3.1 節で示した手法ではプログラムカウンタの特徴量 pc の寄与率が大幅に低下するため，特徴量 pc のみに関して 3.2 節の手法を使った bit 削減に変更し分類器の作成を行った．この時，新規特徴量 pc1，pc2，pc3，pc4 を作成し，それぞれの寄与率を比較するため，4 つすべてを適用し元の特徴量 pc を抜いた状態で実施した．その結果，各特徴量の寄与率は表 7 のようになり，新規特徴量 4 つのうち pc3，pc4 の寄与率は低く，pc1，pc2 の寄与率が高いことが分かった．また，pc2 の寄与率は他の特徴量を含めても 1 番高い値になった．

表 5: bit 幅削減前後の特徴量寄与率
Table 5 Feature Contribution Ratio before and after Bit Width Reduction

特徴量名	bit 削減前の寄与率	bit 削減後の寄与率
pc_mean	30.37 %	29.45 %
pc	32.85 %	9.99 %
op_mean	13.48 %	9.38 %
addr_mean	4.23 %	26.93 %
L1_inst	8.91 %	0.13 %
L2	10.15 %	11.01 %

表 6: bit 削減前後の分類器作成と判定結果
Table 6 Classifier Creation and Decision Results before and after Bit Reduction

特徴量数	分類器サイズ	生成時間	正解率	適合率
6	102,538,016	490.81 s	97.60 %	95.38 %
6	14,970,397	329.77 s	94.40 %	95.38 %

表 7: 新規特徴量を使用した学習における寄与率
Table 7 Contribution Rate in Learning Using New Features

特徴量名	寄与率
pc_mean_red	6.81 %
pc_red	-
op_mean_red	7.28 %
addr_mean_red	16.48 %
L1_inst_red	9.06 %
L2_red	8.98 %
pc1	11.14 %
pc2	30.07 %
pc3	6.74 %
pc4	3.44 %

表 8: 特徴量 pc 分割後の分類器作成と判定結果
Table 8 Classifier Creation and Decision Results After Feature PC Segmentation

特徴量	分類器サイズ	生成時間	正解率	適合率
pc	14,970,397	329.77 s	94.40 %	95.38 %
pc1-pc4	15,671,856	423.43 s	95.20 %	92.31 %
pc1	18,224,987	323.04 s	93.60 %	95.38 %
pc2	15,336,315	313.25 s	96.80 %	93.85 %
pc3	19,058,555	353.68 s	92.00 %	92.31 %
pc4	25,100,219	388.46 s	92.80 %	93.85 %

5.3 考察

3.1 節の式を用いた特徴量の bit 幅の削減では，削減によって扱うデータ量や計算量が減少し，演算処理の高速化や分類器サイズの削減に繋がったと考えられる．また特徴量 pc の bit 幅分割では，使用する特徴量の bit 幅を小さく区切ることで有効な数値の部分を抽出し，精度が向上し演算時間も削減されたと考える．pc1 から pc4 にかけて特徴量の寄与率に違いが生まれた原因としては，特徴量 pc に使用されているプログラムカウンタの数値が実行する命令のアドレスを示しており，メモリマップなどで割り当てられている区間ごとに変化する上位の桁がある程度の傾向を持ち合わせているのに対し，下位の桁はその数値の変動が激しく，ばらついているためであると考えられる．上位の bit であるほどその数値に変動が起きにくく，ある程度の規則性を持つようになると考えられるが，最上位の bit の塊を要素に持つ pc1 に比べ pc2 の方が寄与率が高くなったのは，24 bit 以下の数値がありpc1 の値が 0 となった命令があったためであると考えられる．実際に特徴量 pc2 で学習判定を行った際に，pc1-4 を使用した場合よりも正解率や適合率の数値が高くなっており，bit 幅を区切り必要な特徴量のみを使用することで精度を維持しつつ分類器のサイズや生成時間が削減されたと考える．

6. まとめ

本論文では，既存機構である AMH を利用し，その精度向上に向けて新規特徴量の作成と同一プログラムの複数のトレースを使った学習を行った．機械学習において，学習データを増やすと判定精度が向上することは周知の事実であり，扱うデータ量の増加によって起こる分類器のサイズと生成時間の増加に対して新規特徴量の作成を行うことで対処する手法を提案した．bit 幅の削減では式を利用し特徴量の bit を 32 bit から 16 bit へ変換することで分類器サイズや生成時間を削減することができた．またこの際に寄与率が下がった既存特徴量 pc に関しては 4 つの新規特徴量に分割することで，新規特徴量の中で寄与率の高い部分と低い部分が明確化し，不必要な特徴量を削除することができた．これにより精度を維持しつつ，分類器のサイズと生成時間を削減することができた．

謝辞 本研究の一部は，JSPS 科研費 23K11108 の支援により行った．

参考文献

[1] Ministry of Internal Affairs and Communications, “世界の *IoT* デバイス数の推移及び予測,” https://www.soumu.go.jp/johotsusintokei/whitepaper/ja/r06/html/datashu.html (Accessed 2024-8-8).

[2] J.D. Guarnizo, et al., *“SIPHON: Towards Scalable High-Interaction Physical Honeypots*,” Proceedings of the 3rd Workshop on Cyber-Physical System Security, pp. 57-68, 2017.

[3] M.S. Pour, et al., *“On Data-Driven Curation, Learning, and Analysis for Inferring Evolving Internet-of-Things (IoT) Botnets in the Wild*,” Computers & Security, Vol. 91, Article 101707, 2020.

[4] K. Koike, R. Kobayashi and M. Kato, *“IoT-Oriented High-Efficient Anti-Malware Hardware Focusing on Time Series Metadata Extractable from Inside a Processor Core*,” International Journal of Information Security, Vol. 21, pp. 757-775, 2022.

[5] Arm, *“TrustZone for Cortex-M*,” https://www.arm.com/en/technologies/trustzone-for-cortex-m (Accessed 2024-8-8).

[6] V. Shanbhogue, D. Gupta and R. Sahita, *“Security Analysis of Processor Instruction Set Architecture for Enforcing Control-Flow Integrity*,” Proceedings of the 8th International Workshop on Hardware and Architectural Support for Security and Privacy, No. 8, pp. 1-11, 2019.

[7] K. Fujiwara, R. Kobayashi and M. Kato, *“Evaluation of Mechanisms to Detect Malicious Communications Using Average and Spatial Features of Processor Information*,” Computer Security Symposium, pp. 375-382, 2023.

[8] L. Liu, B. Wang, B. Yu and Q. Zhong, *“Automatic Malware Classification and New Malware Detection Using Machine Learning*,” Frontiers Inf Technol Electronic Eng, Vol. 18, pp. 1336-1347, 2017.

[9] *“QEMU*,” https://www.qemu.org/ (Accessed 2024-8-8).

[10] Y.M.P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama and C. Rossow, *“IoTPOT: A Novel Honeypot for Revealing Current IoT Threats*,” Journal of Information Processing, Vol. 57, No. 4, 2016.

[11] Y.M.P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama and C. Rossow, *“IoTPOT: Analysing the Rise of IoT Compromises*,” 9th USENIX Workshop on Offensive Technologies (USENIX WOOT 2015), 2015.