IoT 機器上で悪性通信をハードウェアで検知する機構のリソース削減手法の提案

秋山 輝樹[1,a)]　小林 良太郎[1]　加藤 雅彦[2]

概要：近年, IoT 機器の利用数は増加の一途をたどっており, それに伴い様々な攻撃手法で IoT 機器に対してサイバー攻撃が仕掛けられている. それらの攻撃から身を守るために, 従来の境界防御では限界があるためエンドポイントでのセキュリティ対策が IoT 機器にとって重要な課題となっている. しかし, IoT 機器のハードウェアリソースではセキュリティ機構をソフトウェアで実装するのは困難である. そこで我々は, 悪性通信検知機構をハードウェアとしてオフロードする Anti-Malcious Communication Hardware (AMCH) という機構を提案してきた. AMCH では, 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ムカウンタやオペコードなどのプロセッサから得られる情報を利用し, これらをランダムフォレストで学習させることで分類器を作成し, 悪性通信の検知を行う. AMCH は特徴量としてレジスタのヒット率を使用しているが, この特徴量を生成するための演算器を搭載することからハードウェアサイズの肥大化が生じてしまっている. そのため本研究では, アクセス数とヒット率をインデックスとし, それに伴うヒット率を格納したテーブルであるヒットレートテーブル (HRTable) を AMCH 内に実装, 使用し特徴量を生成することで, さらなるハードウェアサイズの削減を行う手法の提案を行う.

キーワード：IoT, プロセッサ情報, 機械学習, 悪性通信

Proposal of a Resource Reduction Method for a Hardware-based Mechanism to Detect Malicious Communications on IoT Devices

Kouju Akiyama[1,a)]　Ryotaro Kobayashi[1]　Masahiko Kato[2]

Abstract: In recent years, the number of IoT devices in use has been steadily increasing, and as a result, various attack techniques have been employed to launch cyberattacks against them. To defend against such attacks, traditional perimeter-based defenses are no longer sufficient, making endpoint security a critical issue for IoT devices. However, due to the limited hardware resources of IoT devices, it is difficult to implement security mechanisms purely in software. To address this challenge, we have proposed a mechanism called Anti-Malicious Communication Hardware (AMCH), which offloads malicious communication detection to hardware. AMCH utilizes processor-derived information such as the program counter and opcodes—data that can be efficiently offloaded to hardware—and trains a random forest classifier on this information to detect malicious communications. Currently, AMCH employs register hit rates as a feature. However, because implementing the arithmetic units needed to generate these features increases hardware size, this poses a scalability issue. Therefore, in this study, we propose a method to further reduce hardware size by implementing and using a Hit Rate Table (HRTable) within AMCH. The HRTable stores hit rates corresponding to access counts and indices, allowing the generation of features without the need for additional arithmetic units.

Keywords: IoT, Processor Information, Machine Learning, Malicious Communications

1. はじめに

Internet of Things (IoT) とは, 様々なモノをインターネットに接続させることで, 相互に情報交換や制御を行う技術である. 遠隔で制御をできるようになり, 場所を選ばずに様々な場面で用いることができるその利便性から, 年々 IoT 機器の数は増加している. しかし, 近年その 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. 2016 年 10 月には, Dyn が Mirai マルウェアに感染した IoT ボットよって DDoS 攻撃を受け, その際 120 万台以上の IoT 機器に感染したという被害が発生している [1]. 加えて近年, IoT 機器の急速な普及に伴い様々な事柄におけるが遠隔化が発展している. それにより, ネットワークを介したデータ通信は従来と比較して一層複雑化しており, ファイアウォールや侵入防止システム (IPS) を代表とした境界型セキュリティのみでは, サイバー攻撃に対する十分な防御手段とはなりえない状況が顕在化している. このような背景から, IoT 機器といったリソース制限のある機器においても実装可能なエンドポイントでのセキュリティの強化が求められる. エンドポイントセキュリティの実装手法は主にソフトウェアベースとハードウェアベースの二つに分けることができるが, 現状ではソフトウェアベースの手法がハードウェアベースの手法と比べて評価され, 広く採用されている. しかし, ソフトウェアによるセキュリティ対策は, 動作に多くのリソース量と必要とするため, 使用することができるリソース量の限られている IoT 機器においては適用することが困難である. 実際, McAfee が提供するアンチウイルスソフトの一つである McAfee+の動作に必要とされる最低限の環境は, RAM2GB, ディスク容量 1.3GB, プロセッサ 1GHz が必要とされている [3]. その他にも Kaspersky が提供する Kaspersky Endpoint Security for Windows は最低限の環境として, RAM が 32-bit OS では 1GB, 64-bit OS では 2GB, ディスク容量 2GB, プロセッサ 1GHz が必要とされている [4]. 加えて, IoT 機器は汎用性の低い OS が用いられている場合も多い. これらのことから, IoT 機器に対して従来型のセキュリティソフトウェアを導入することは極めて困難である. このような IoT 機器のセキュリティ対策におけるリソース不足の観点から, 我々はプロセッサ情報を用いたハードウェアベースのセキュリティ機構として Anti-Malcious Communication Hardware (AMCH) を提案している [2]. AMCH は LSI 上に CPU と混在する形で実装され, 被悪性通信時と通常のプログラム実行時のプロセッサ情報を用いてランダムフォレストで機械学習を行うことによって, 悪性通信を検知する分類器の作成を行う. しかし, 先行研究で作成した AMCH は特徴量として L1 や L2 のキャッシュヒット率を使用することからハードウェア上に割り算器を実装する必要があり, ハードウェアサイズや消費電力量が肥大化してしまっていた. そこで, 我々は出口らの L1, L2 のヒット数とアクセス数をインデックスとしてヒット率を参照するヒットレートテーブル (HRTable) に注目した [5]. 本研究では HRTable のさらなるハードウェアサイズの削減手法の提案を行う.

[1] 工学院大学
Kogakuin University, Shinjuku, Tokyo 163-8677, Japan
[2] 順天堂大学
Juntendo University, Bunkyou, Tokyo 113-8421, Japan
[a)] jx22007@ns.kogakuin.ac.jp

2. 関連研究及び先行研究

HRTable とはヒット数とアクセス数をインデックスとして, それぞれの対応するセルに hit/access のヒット率を格納したテーブルである. 出口らの malware detection mechanism (MDM) には特徴量としてキャッシュのヒット率が使用されていたことから, また, 機構内に除算器を実装する必要があったことから, ハードウェアサイズや消費電力量が肥大化してしまっていた. これらの問題は除算器によって引き起こされていることから, 出口らは除算器の代替として HRTable というヒット数とアクセス数をインデックスとしヒット率を格納した HRTable を LUT として検知機構に実装することで, ハードウェアサイズの削減を行っていた [5]. HRTable に格納する値はヒット率に 2 の 16 乗を乗算してから整数部分を切り取って整数化し, 元の浮動小数点のヒット率の状態での格納を防ぐことでハードウェアサイズの削減と分類器の精度の両立を図っていた.

3. 提案手法

3.1 提案機構

出口らは HRTable のアクセス数が 127 以下の値を削減することによって HRTable 自体のハードウェアサイズの削減を行っていた. しかしこの HRTable には, 通常の動作では使用しないヒット数がアクセス数を上回ったテーブル部分がそのまま使用されている. 提案機構では, この本来使用しないテーブルの部分に出口らが削減したアクセス数が 127 以下の値を格納することによって, HRTable のサイズはそのまま, すべての index に対応できるようにした. それらに加えて, ヒット数が 0 のときの値は hit/access=0 であり, ヒット数とアクセス数が同等のときは hit/access=1 であるため, それらの部分を if 文で出力することによってさらなる HRTable のハードウェアサイズの削減を行う.

3.2 HRTable のハードウェアサイズ削減

既存機構では HRTable の hit>access の部分を使用しない部分としてそのまま実装してしまっていた. 本研究ではその部分に本来使用する部分を折りたたんで格納することによって HRTable そのもののハードウェアサイズの削減を行うことができるような方法を提案する. 折り畳みを行う前に hit==0 のときの値と, hit==access の値は 1 で固

定のため if 文を用いて出力することを想定して削減を行う．

access hit	0	1	2	…	127	128	…	253	254	255
0	-	0	0	…	0	0	…	0	0	0
1	-	1	1/2	…	1/127	1/128	…	1/253	1/254	1/255
2	-	-	1	…	2/127	2/128	…	2/253	2/254	2/255
…	-	-	…	…	…	…	…	…	…	…
126	-	-	-	…	126/127	126/128	…	126/253	126/254	126/255
127	-	-	-	…	1	127/128	…	127/253	127/254	127/255
128	-	-	-	…	-	1	…	128/253	128/254	128/255
…	-	-	…	…	…	…	…	…	…	…
253	-	-	-	…	-	-	…	1	253/254	253/255
254	-	-	-	…	-	-	…	-	1	254/255
255	-	-	-	…	-	-	…	-	-	1

図 1　hit==0, hit==access 削減前の HRTable
Fig. 1　HRTable Before Reduction hit==0, hit==access

access hit	2	3	4	…	128	129	…	253	254	255
1	1/2	1/3	1/4	…	1/128	1/129	…	1/253	1/254	1/255
2	-	2/3	2/4	…	2/128	2/129	…	2/253	2/254	2/255
3	-	-	3/4	…	3/128	3/129	…	3/253	3/254	3/255
…	-	-	…	…	…	…	…	…	…	…
126	-	-	-	…	126/128	126/129	…	126/253	126/254	126/255
127	-	-	-	…	127/128	127/129	…	127/253	127/254	127/255
128	-	-	-	…	-	128/129	…	128/253	128/254	128/255
…	-	-	…	…	…	…	…	…	…	…
253	-	-	-	…	-	-	…	-	253/254	253/255
254	-	-	-	…	-	-	…	-	-	254/255
255	-	-	-	…	-	-	…	-	-	-

図 2　hit==0, hit==access 削減後の HRTable
Fig. 2　HRTable After Reduction hit==0, hit==access

図 1 は hit=0 の値と hit==access の値の削減前のイメージであり，図 2 はそれらの削減後のイメージである．これ等の図を比較すると削減後は HRTable の hit==0 の行，access==0, 1 の列が削減されていることがわかる．

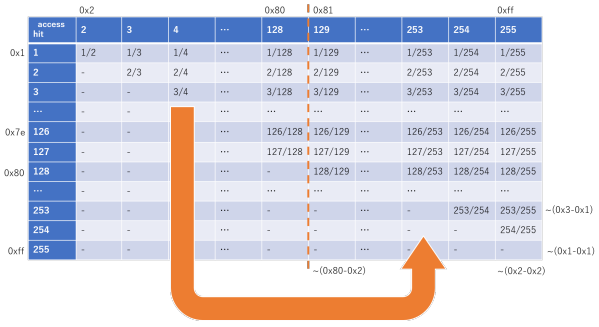

図 3　HRTable のテーブルの反転と格納
Fig. 3　Inversion and Storage of the HRTable

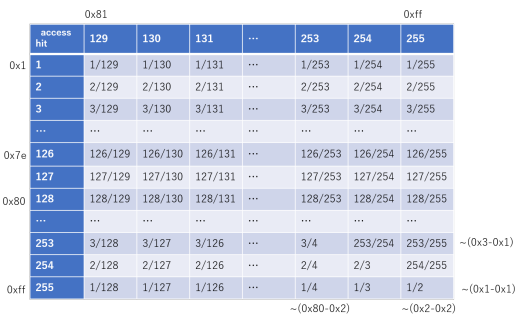

図 4　HRTable のテーブルの反転と格納後
Fig. 4　After HRTable Inversion and Storage

図 3 は図 2 の HRTable のアクセス数が 2~128 までの値を反転させて 129~255 までの値に格納することによってハードウェアサイズの削減を行うイメージ図であり，図 4 はその動作後の図である．

Algorithm 1 Table Value Calculation and Inversion

$MUL \leftarrow 2\text{^}16$
for $i = Access\_Min$ **to** $Access\_Max$ **do**
  for $i = Hit\_Min$ **to** Hit_Max **do**
    $table\_float[hit, access] \leftarrow \frac{hit}{access} \times MUL$
    $table\_int[hit, access] \leftarrow \lfloor table\_float[hit, access] \rfloor$
    if $access < Reduced\_Access\_Min$ **then**
      $Before\_Reversed\_Hit \leftarrow hit - Hit\_Min$
      $Before\_Reversed\_Access \leftarrow access - Access\_Min$
      $Reversed\_Hit \leftarrow\sim Before\_Reversed\_Hit \wedge 0xFF$
      $Reversed\_Access \leftarrow\sim Before\_Reversed\_Access \wedge 0xFF$
      $reduced\_table\_int[Reversed\_Hit, Reversed\_Access] \leftarrow \lfloor table\_float[hit, access] \rfloor$
    else
      $reduced\_table\_int[hit, access] \leftarrow \lfloor table\_float[hit, access] \rfloor$
    end if
  end for
end for

Algorithm 1 は，ヒット数 (hit) およびアクセス数 (access) の組み合わせに基づいた演算結果を，縮小版のテーブルに格納するアルゴリズムである．外側ループではアクセス数を Access_Min から Access_Max まで探索し，内側ループでヒット数を Hit_Min から Hit_Max まで列挙する．Access_Min, Hit_Min は削減前のテーブルのそれぞれの index の最小値であり，Access_Max, Hit_Max 削減前のテーブルのそれぞれの index の最大値である．まず，浮動小数点テーブル要素を (hit/access)*MUL と計算する．その後，整数テーブルに先ほど計算した結果を対応した箇所に格納する．次に，アクセス数が Reduced_Access_Min より小さい場合，縮小テーブルの格納位置を算出するために調整を行う．Reduced_Access_Min は削減後の HRTable のアクセス数のインデックスの最小値である．ヒット数およびアクセス数からそれぞれの最小値を引いた後，それぞれをビット反転することで Reversed_Hit, Reversed_Access を得る．この座標に対して元々の値の格納する．

3.3　ビットシフトによるエントリポイントの変換

HRTable のサイズが 256×256 の関係上，ヒット数とアクセス数をそのまま用いて HRTable でヒット率に変換することはできない．そのためヒット数，アクセス数をそれぞれ HRTable の範囲 0~255 に収まるようにビットシフトを利用して近似を行う．また削減後の HRTable は 255×126 になってしまうが，ビットシフトによる近似化後に HRTable のハードウェアサイズの削減の際に用いた index の反転を

行うことで問題なく変換を行うことができる.

Algorithm 2 Bit-Shift-Based Approximation

```
num1 ← Access_Count
num2 ← Hit_Count
n1 ← num1 ∧ 0xFFFFFFFF
n2 ← num2 ∧ 0xFFFFFFFF
for i = 31 to 0 (step -1) do
    if (n1 ≫ i) ∧ 1 = 1 then
        start ← i
        end ← max(i − 7, 0)
        length ← start − end + 1
        value1 ← (n1 ≫ end) ∧ ((1 ≪ length) − 1)
        value2 ← (n2 ≫ end) ∧ ((1 ≪ length) − 1)
        return value1, value2
    end if
end for
```

Algorithm 2 は, 2 つの 32 ビット整数値, アクセス数とヒット数から最上位の 1 が出現するビットを起点として, その直下 7 ビット分を含む最大 8 ビット幅の部分領域を抽出し, 両入力値から同一のビット範囲を切り出すアルゴリズムである. まず, 入力値 num1, num2 に対して 32 ビットマスク (0xFFFFFFFF) を適用し不要なビットの排除を行う. 次に, ビット位置を最上位ビットから最下位ビットへと順に探索を行い, 初めて 1 が出現した位置を start として記録する. その後, start から最大 7 ビット下までを抽出範囲とし, 範囲の下限を end として定め, 抽出範囲のビット長を length として計算する. このビット長に基づいてマスクを生成し, n1, n2 をそれぞれ end の位置まで右シフトした後にマスクを適用することで, 近似化を行った value1, value2 の結果が返される.

上記のアルゴリズムをフローチャートにしたものを図 5 に示す.

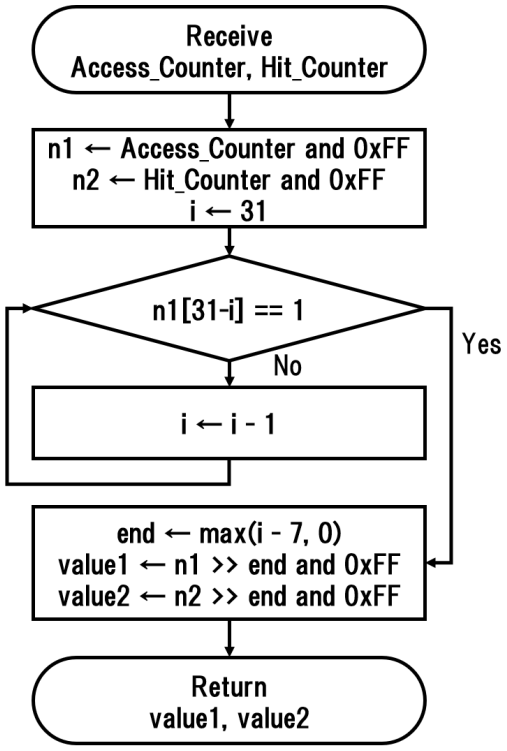

図 5 アクセス, ヒット数の近似化のフローチャート
Fig. 5 Flowchart of Access and Hit Count Approximation

3.4 提案手法の具体例

これらのアルゴリズムに実際の数値を入れたときの動作例を挙げる. 想定するパラメータは次のように設定する.

・Access_Counter[31:0] = 30000
(0b00000000000000000111010100110000)
・Hit_Counter[31:0] = 2000
(0b00000000000000000000011111010000)

まずこれらの数値を HRTable のインデックス内に収まるようにビットシフトによる近似を行う. Access_Counter の最上位ビットから最下位ビットに向けて探索を行うと最初に 1 が現れるビットは 15 ビット目であることがわかる. 探索により, 最初に 1 が現れるビットが分かったため, 15 ビットから 8 ビットの数値を取り出す. 取り出した数値は次のようになる.

・value1[7:0] = 234 (0b11101010)
・value2[7:0] = 15 (0b00001111)

2000/30000 = 0.0666..., 15/234 = 0.0641...
実際のヒット率と近似後のヒット率の差は 0.0025... となることから, ある程度の精度で近似を行うことができていることが確認できる. HRTable に格納する値はこの近似化したヒット率に対して 2^16 を乗算して, 整数部分を切り取ったものになる.

2000/30000 ∗ 2^16 = 4369.066...
15/234 ∗ 2^16 = 4201.025...

格納する値の差は 168 と少し大きくなってしまうが, 2^16 を乗算した後だと考えると充分な近似が行うことができていると考える. またこの index はサイズ削減後の HRTable 内に収まっているため, そのまま使用することができる. 次に index の変換の動作が生じるようなパラメータを想定し動作例を挙げる.

・Access_Counter[31:0] = 100
(0b00000000000000000000000001100100)
・Hit_Counter[31:0] = 20
(0b00000000000000000000000000010100)

これらの値はどちらも 8 ビット内に収まっているため, 近似を行う必要が無いが, この index はサイズ削減後の HRTable 内に存在しないため, index の変換を行う. まずどちらの数値からもそれぞれの index 最小値を引く. 現在もちいている HRTable の場合, Access の index の最小値は 2 であり, Hit の index の最小値は 1 である.

・Before_Reversed_Access = 100 - 2 = 98 (0b01100010)
・Before_Reversed_Hit = 20 - 1 = 19 (0b00010011)

次にビット単位の反転を行う.

・Reversed_Access = 157 (0b10011101)

・Reversed_Hit = 236 (0b11101100)

この index の数値を見ると, 従来使用することのない hit>access の部分に割り当てられていることがわかる. これらのことから, 削減後の HRTable は削減前の HRTable と比べて, 大差なく動作していることが確認できる.

4. 考察

本稿で提案した手法を用いることによって, 既存機構の HRTable からさらなるサイズの削減が可能であると考えられる. IoT 機器のように消費電力や実装面積が厳しく制限される環境においては, 演算器の削減による低消費電力化の効果も期待される. 加えて, 本手法は FPGA 上での検証だけでなく, 将来的に ASIC での実装を見据えた設計最適化にも有効であり, 面積や電力の削減効果を実運用規模で評価できる. さらに, 本手法は他の計算によって特徴量を算出するハードウェアのセキュリティ機構にも適用可能であり, 汎用性の観点からも有用性が高いといえる.

5. まとめ

本研究では, 分類器の割算器を削減することによってハードウェアサイズの削減を図る HRTable のさらなるサイズの削減を行うことによって, 既存機構と比べて IoT 機器に対して実装を容易に行うことができるような機構の提案を行った. 将来的な実験として先行研究で提案された既存機構は, FPGA をもちいたシミュレーション環境で行われたものであり, 作成したコードを動作させることに重点を置いていたため, 実際の ASIC での半導体製造に近い環境で評価を行う必要がある. 特に ASIC 実装においては, 面積効率や電力消費といった実用的な制約条件の下でスケーラブルに動作可能であるかを評価することが, 実運用への移行に向けた重要になる. そのためには, OpenLane を用いたバックエンド評価を実施し, 配置配線段階での配線輻輳やタイミング特性を含めた詳細な検証を行うことで, 提案手法の実効性をさらに明確にできると考えられる. OpenLane とは, 自動化された RTL から GDSII への設計フローであり, OpenROAD, Yosys, Magic, Netgen などの複数のコンポーネントと使用し, 設計の探索および最適化を行うことができるオープンソフトウェアである [6]. これにより, さらなる高精度かつ低リソースな IoT 向けセキュリティ基盤の実現が期待できる.

謝辞

本研究の一部, PS 科研費 23K11108 の支援により行った.

参考文献

[1] Q.-D. Ngo, et al., *"A Survey of IoT Malware and Detection Methods Based on Static Features,"* ICT Express, Vol. 6, No. 4, pp. 280-286, 2020.

[2] K. Fujiwara, et al., *"Hardware Mechanism for Detecting Malicious Communication in IoT Devices by using Time-Series Processor Information,"* Proceedings of the Twelfth International Symposium on Computing and Networking (CANDAR), pp. 149-155, 2024.

[3] McAfee, "System Requirements," `https://www.mcafee.com/en-us/consumer-support/help/system-requirement.html` (Accessed 2025-8-2).

[4] Kaspersky, "Hardware and Software Requirements," `https://support.kaspersky.com/keswin/11/ja-JP/127972.htm` (Accessed 2025-8-21).

[5] M. Deguchi, et al., *"Low Resource and Power Consumption and Improved Classification Accuracy for IoT Implementation of a Malware Detection Mechanism using Processor Information,"* International Journal of Networking and Computing, Vol. 13, No. 2, pp. 149–172, 2023.

[6] M. Gaber, "The-OpenROAD-Project/OpenLane," `https://github.com/The-OpenROAD-Project/OpenLane` (Accessed 2025-8-19).