Computer Security Symposium 2020
26 - 29 October 2020

プロセッサ情報を用いたマルウェア検知機構の FPGA 実装のための予備評価

出口 睦樹[1]　加藤 雅彦[1]　小林 良太郎[2]

概要：通信機器に対するセキュリティ対策は，ソフトウェアが主流となっている．しかし，ハードウェア資源が少ない IoT 機器ではソフトウェアでの対策は資源を圧迫し，現実的ではない．そこで我々はプロセッサ情報を用いたハードウェアによるマルウェア検知機構を提案し，シミュレーションによって有効性を確認している．本研究では提案機構を FPGA 実装するための予備評価をおこなう．従来の提案機構で使用している割り算を用いた実装に加え，割り算を使用しない軽量で高精度な実装もおこなう．最後に FPGA 上に RISC-V コアと提案機構を搭載し，全体のリソース使用量を評価する．

キーワード：機械学習，IoT，RISC-V ,マルウェア検知

Preliminary evaluation for FPGA implementation of malware detection mechanism using processor information

Mutsuki Deguchi[1]　Masahiko Kato[1]　Ryotaro Kobayashi[2]

Abstract: Software is standard security measures for computers. However, IoT devices has few hardware resources. So, it is unrealistic that software measures put pressure on them. Therefore, we have proposed a hardware malware detection mechanism with processor information and already verified effectiveness by simulation. We perform preliminary evaluation for FPGA implementation of the proposed mechanism. We implement not only the conventional proposed mechanism with division but also light and accurate mechanism without division. We implement RISC-V core and the proposed mechanism on FPGA, evaluate resource utilization.

Keywords: Machine Learning, IoT, RISC-V, malware detection

1. はじめに

近年，IoT（Internet of Things）機器に対するサイバーセキュリティの発生率は増加の傾向にある．特に 2016 年の Mirai による攻撃は有名であり，それに伴った Mirai のソースコードのインターネット上への公開により，Mirai の亜種となるマルウェアも生まれている[1]．また IoT 機器の普及も進んでおり，車や医療，家電製品での高成長が見られる[2]．だが，IoT 機器には多くの脆弱性があり，上記で述べた Mirai などがその脆弱性をついた攻撃にあたる[3]．このことから，IoT 機器へのセキュリティ対策は急務を要しているが，現状として見送られる場合が多い．

IoT 機器の脆弱性の原因として，ハードウェアリソース量の不足が挙げられる．通信機器に対するセキュリティ対策はソフトウェアが主流となっているが，IoT 機器では，PC やサーバ用の機器のように豊富なハードウェア資源は内蔵されておらず，長いスパンで常時活動するための最小限のハードウェアリソースしか持ちあわせていない．IoT 機器でソフトウェアによるアンチウイルス機構を常時動作させることは大量のリソースを使用する事となり，基本的に低コスト，低消費電力で活動する IoT 機器の特徴を活かせず，セキュリティ対策の実装はほとんどされていない[4].

我々は上記で述べた IoT 機器へのセキュリティ対策として低資源で運用が可能な，プロセッサ情報を用いたハードウェアレベルでのマルウェア検知機構を提案している[5][6][7]．本研究ではこのマルウェア検知機構が実際に IoT 機器上で動作することを確認するための予備評価として，FPGA に搭載可能な判別機と格納表のサイズ調整，及びマルウェア検知精度に影響が出ない範囲でヒット率のビット幅を調整し，FPGA 実装時の速度を計測，評価する．

2 章で従来の提案機構とその課題について述べたのち，3 章では従来の提案機構の課題に取り組むための本研究での提案機構の概要を，さらに 4 章では本研究の各機構の課題とその対策について述べる．5 章では，4 章で述べた対策についての評価手法を示し，測定結果を述べる．6 章では考察を行い，7 章で本論文をまとめる．

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

2. 先行研究とその課題

本章では，先行研究にて提案しているマルウェア検知機構の概要と課題について述べる．

2.1 従来の提案機構の概要

従来の提案機構の概要を図 1 に示す．提案機構では，LSI 上に CPU とマルウェア検知機構を混載している．CPU はプログラムを実行しながら 1 命令ごとにプロセッサ情報を生成し，マルウェア検知機構に送信する．プロセッサ情報とは，キャッシュのヒット率，Program Counter（PC），レジスタ番号等，CPU 内部で得られる情報で構成される．マルウェア検知機構は 1 命令ごとに良性，もしくは悪性の判別を行っていき全命令数における悪性と判別された割合が，ある閾値以上であれば悪性のプログラムとして判別する．マルウェア検知機構の一部である判別機は，ランダムフォレストと呼ばれる教師あり学習を用いて生成する．学習には，すでに良性，もしくは悪性と判断されているプログラムを実行し得られたプロセッサ情報を用いる．なおランダムフォレストは，学習データの特徴量が増加しても効率的に学習が可能であるため，特徴量が多いプロセッサ情報を扱う従来の提案機構では適していると考えられる[8].

CPU は通常，プロセッサ情報を生成する機能を有していないため，これまでの研究では仮想マシンを用いたシミュレーション上で提案機構の実装を行っている．また，評価により，キャッシュヒット率が特徴量として重要であることがわかっている．そのため本研究では特徴量としてキャッシュヒット率に着目する．ヒット率については，L1 命令キャッシュ，L1 データキャッシュ，L2 キャッシュ，それぞれのキャッシュから得られるアクセス数とヒット数を除算器にわたすことで，除算器内で計算が行われ各ヒット率が算出される．

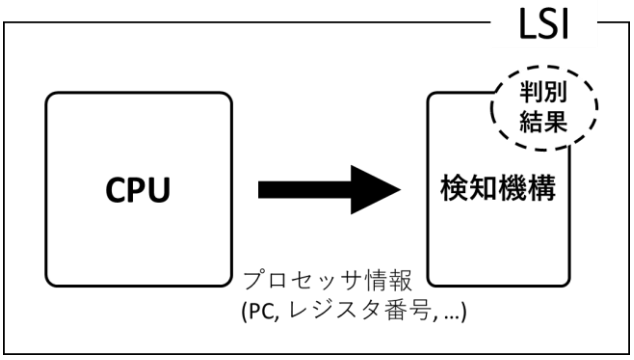

図 1　従来の提案機構の概要

2.2 FPGA 実装における課題

従来の提案機構では，ハードウェアリソースが大量に用意されている LSI を前提として，ソフトウェアでシミュレーションを行っている．しかし本研究ではリソース制限の厳しい IoT 機器への実装を前提とするため，ハードウェアリソースの少ない FPGA を用いて提案機構の実装を試みる（図 2）．そこでまず，図 1 に示している提案機構を，FPGA 上に実装する検討を行う．提案機構で使用する除算器での浮動小数点型のデータを使用した除算処理は，ハードウェアリソースを大量に消費する．本研究で実装の対象としている Digilent 社製の FPGA「Zedboard Zynq-7000」は，実装可能な回路規模が小さいため，除算器をそのまま実装すると，回路が収まらない．

そこで本研究では，従来の提案機構のハードウェア構造に変更を加え，FPGA が保有している資源内に収まる回路規模の機構を提案する．

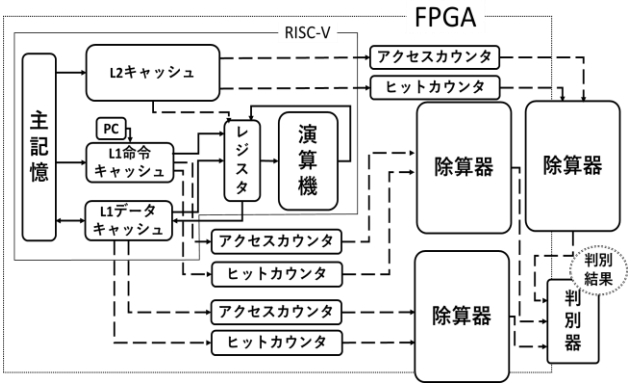

図 2　従来提案機構の FPGA 実装の概要

3. 格納表の導入

3.1 格納表の概要

本研究では，提案機構のハードウェア構造を変更するため，格納表と呼ばれる機構を導入する．

ハードウェア上で演算処理を行う場合，特に乗算や除算の処理では，他の処理に比べると多量にハードウェアリソースを消費してしまう．本研究では，マルウェア検知機構での判別に用いる特徴量としてヒット率を扱う．従来の提案機構では図 2 に示すように，除算器を用いることで CPU から得られるアクセス数とヒット数を除算し，ヒット率を算出しており，その値を検知機構に渡している．しかし除算器での浮動小数点型のデータを使用した除算処理は，ハードウェアリソースを消費している原因となっている．そのため，提案機構の FPGA 実装をすすめる上で，除算器のサイズを縮小する必要がある．この課題に対し，本研究では除算器を用いる代わりに格納表を使用する（図 3）．格納表は，各エントリにヒット率を保持しており，アクセス数とヒット数を入力とすると，それに対応するヒット率を出力する．これによりハードウェア上で除算器を用いずにヒット率を算出することができ，提案機構の軽量化につながる．

・hit=ヒット数，access=アクセス数

hit \ access	1	2	3	…
1	1.0	0.5	0.33..	…
2	…	1.0	0.66..	…
3	…	…	1.0	…
…	…	…	…	…

図 3　格納表

3.2 全体の概要

本章で述べる提案機構のハードウェア実装時における概要を図 4 に示す．提案機構は CPU 部にあたる RISC-V コア[9]，アクセスカウンタ，ヒットカウンタ，格納表，判別器にて構成されている．まずそれぞれのキャッシュに接続されているアクセスカウンタとヒットカウンタが，プログラム実行中にキャッシュへアクセスした回数とその際にヒットした回数を 1 命令ごとに取得し，格納表へ出力する．格納表は，入力として得られるアクセス数とヒット数に対応するヒット率を判別器へ出力する．判別器では従来の提案機構のように全てのプロセッサ情報を入力として得るかわりに，各キャッシュのヒット率を受け取る．1 命令ごとに，良性，もしくは悪性の判別を行っていき全命令数における悪性と判別された割合がある閾値以上であれば悪性のプログラムとして判別し，OS などの上位層に異常を知らせる．

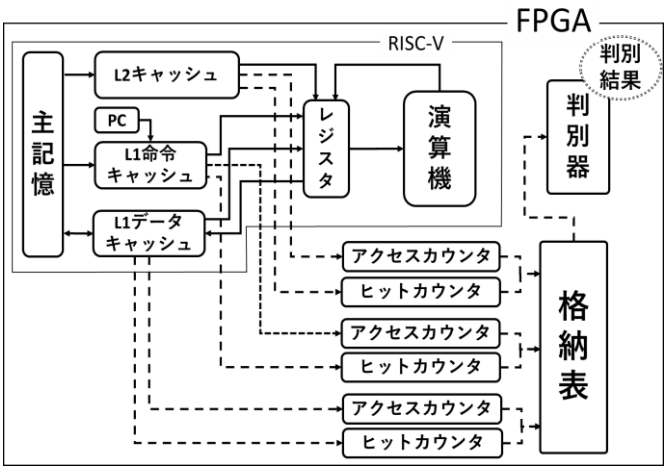

図 4　本研究の提案機構の FPGA 実装の概要

4. 各機構におけるハードウェア量削減

4.1 削減手法

本研究で，FPGA への実装を試みる格納表は命令のアクセス数，ヒット数の値の増加に比例して表のサイズが大きくなる．本研究で機械学習の際に用いるプロセッサ情報が収集されている学習データでは，キャッシュへのアクセス数が最大で約 10 万にまで増加する．この場合，ヒット数の値も最大 10 万となる．表への入力となる二値の組み合わせの総数は 10 万*10 万の 100 億通りとなり，この内の約半分の 50 億個の組み合わせがマッチする可能性のある値となる（図 5 点線内）．これら全てをメモリに格納するのは，ハードウェアリソースが少ない FPGA では困難である．そこで，格納するアクセス数，ヒット数の最大値を制限し格納表のサイズを FPGA への実装が可能となる範囲まで縮小する（図 6）．なお，格納表を縮小した場合，表への入力となる二値も縮小する必要がある．

また，従来のマルウェア検知機構の構造を基にハードウェアへの実装を行う場合，検知機構では判別のために保持しているヒット率の値について，ひとつあたり浮動小数点型 64 ビット幅のサイズを有している．これはハードウェアのリソースを消費する要因であると考える．そこで，良性，悪性命令への判別精度に影響を与えないよう，マルウェア検知機構の保持するヒット率をそれと一対一に対応する整数に変換することでハードウェアリソースの縮小をおこなう．例えば，浮動小数点型である 0.953218…を，整数である 9532 へと変換する．なお，この変換により，ビット幅を削減することができるが表現可能な値の範囲も削減されてしまうため，判別精度に影響を与えない範囲で変換を行う必要がある．なお，ヒット率のビット幅を削減することによって，ヒット率を保持している格納表のサイズも削減される（図 7）．

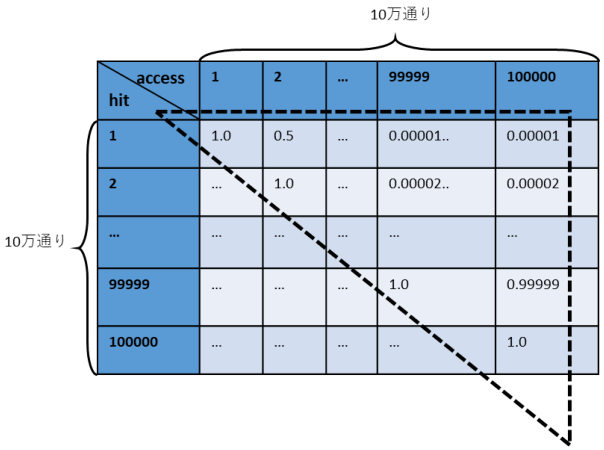

図 5　格納表の初期のサイズ

約50億通り

hit \ access	1	2	…	126	127	…	99999	100000
1	1.0	0.5	…	0.00793..	0.00787..	…	0.00001..	0.00001
2	…	1.0	…	0.01587..	0.01574..	…	0.00002..	0.00002
…	…	…	…	…	…	…	…	…
126	…	…	…	1.0	0.99212..	…	0.00099..	0.00099
127	…	…	…	…	1.0	…	0.001..	0.001
…	…	…	…	…	…	…	…	…
99999	…	…	…	…	…	…	1.0	0.99999
100000	…	…	…	…	…	…	…	1.0

約8000通り

hit \ access	1	2	…	126	127
1	1.0	0.5	…	0.00793..	0.00787..
2	…	1.0	…	0.01587..	0.01574..
…	…	…	…	…	…
126	…	…	…	1.0	0.99212..
127	…	…	…	…	1.0

図 6　表のサイズ縮小案

hit \ access	1	2	…	126	127
1	10000	5000	…	79	78
2	…	10000	…	158	157
…	…	…	…	…	…
126	…	…	…	10000	9921
127	…	…	…	…	10000

図 7　表の各エントリ整数化

5.　評価手法，及び測定結果

本章では第 4 章で述べた削減手法を適応させ，測定を行う．評価項目は，ハードウェア量と判別精度である．5.1 節では格納表のサイズを縮小し，開発ツールを用いて，FPGA 実装を行った際のサイズを測定する．5.2 節では，提案機構の精度評価をおこなう．評価では悪性プログラムとして Mirai，良性プログラムとして ls, mkdir, chmod を用いる．提案機構は機械学習を用いているため，上記の悪性・良性プログラムを用いて学習用データと評価用データを用意する．

5.1 ハードウェア量の縮小効果

5.1.1　評価環境

格納表のサイズ縮小をおこなうにあたり，アクセス数とヒット数の値に最大値を設定し，最大値を超える値にはシフト演算の要領を用いて値の縮小を行う．これにより，格納表のアクセス数とヒット数の取りうる組み合わせの総数を減らすことができる．なお，各エントリのヒット率のビット幅は 64 ビットとする．縮小の方法としてシフト演算を模しているのは，実際に FPGA 上で提案機構を動かすときに，CPU 内では除算などを用いて値を制限内に縮小するよりも，シフト演算を用いた縮小をおこなう手法が演算時のハードウェアリソース使用量を削減できるためである．縮小のパターンとして，他のハードウェアからのアドレスの参照の容易性の観点から，

①縮小前 0<hit, access<100000（$10^5 * 10^5$，約 50 億通り）

②0<hit, access<65536（$2^{16} * 2^{16}$，約 20 億通り）

③0<hit, access<256（$2^8 * 2^8$，約 3 万通り）

の 3 パターンに分け，Xilinx 社の統合開発ツール Vivado での論理合成，及び実装の結果をもとに実際に対象機器内に収まるか，またリソースをどれほど使用しているかを測定する．対象機器の詳細を表 1 に示す．Zedboard のリソースのうち，メモリとして認識される BlockRAM の値を参考に評価を行う．LUT は Look Up Table の略である．

次に，上記の 3 パターンの測定結果の中で，FPGA が持つハードウェアリソース内に収まり，かつ縮小幅が最も大きいパターンの格納表に対し，各エントリが保持するヒット率のビット幅を，64 ビット，32 ビット，16 ビット，8 ビットに制限した際の格納表のハードウェアリソース使用量についての測定もおこなう．

測定ツール	Vivado 2018.2.2 HL WebPACK Edition
対象機器	Zedboard Zynq-7000
RAM メモリ	512MB
LUT 個数	53200
BlockRAM 個数	140

表 1　測定環境

5.1.2　評価結果

格納表のアクセス数とヒット数の取りうる値を制限した際の測定結果を図 8 にしめす．①，②，③の 3 パターンのうち，①と②では Zedboard の BlockRAM の最大値を超えており，論理合成をおこなうことができなかった．唯一，③のパターンが論理合成可能であり，この場合 Zedboard のリソースの BlockRAM の使用率は小容量に抑えられている．このため，格納表への入力値となる二値の縮小は格納表のサイズ縮小の点では効果的であると思われる．測定結果において評価可能なパターンが少なかったため，新しく評価基準としてパターン④を追加する．④では BlockRAM を最大値近くまで使用する場合の格納表のサイズを測定したものである．結果として，$2^8 * 2^8$ < ④ <$2^{10} * 2^{10}$ のサイズとなった．次に，この結果から二値の範囲はパターン③の 0<hit,access<256 が適切であると判断し，このときの格納表が保持する各エントリのビット幅を現状の 64 ビットから，32 ビット，16 ビット，8 ビットに縮小した際の実装結果を図 9 にしめす．

格納表のサイズ	①$10^5$ x 10^5	②$2^{16}$ x 2^{16}	③$2^8$ x 2^8	④550x550
LUT	x	x	129	439
BlockRAM	x	x	58	135
BRAM使用率	x	x	41%	96%

図 8　格納表を削減した際のハードウェア量

ヒット率のビット幅	64ビット	32ビット	16ビット	8ビット
LUT	129	59	58	43
BlockRAM	58	30	16.5	8.5
BRAM使用率	41%	21%	11%	6%

図 9　格納表とビット幅を削減した際のハードウェア量

5.1.3 コアとマルウェア検知機構のハードウェア量

本項では，RISC-V コア及び，5.1 節でハードウェア量を縮小したマルウェア検知機構とで構成される提案機構全体(図 4)を FPGA へ実装した際のハードウェアリソース使用量を測定する．格納表のサイズは，FPGA に実装可能な 2^8 * 2^8 とする．各エントリのビット幅については，8 ビットから 64 ビットの組み合わせが存在するが，ハードウェア削減量が高く，かつ判別精度への影響がほぼない 16 ビットの場合のみ結果を示す．なお，判別精度については次節で述べる．

上記で説明したマルウェア検知機構を FPGA 実装した際のハードウェアリソース使用量の測定結果を図 10 に示す．図 10 では，図 8 や図 9 に対し，FF（Flip Flop），IO，BUFG（Global Buffer）が追加されている．マルウェア検知機構の判別機の部分は python で記述されており，判別精度の評価はシミュレーション上で行っている．そのため，実装時には記述内容を論理合成可能なファイルへと変換し，FPGA への実装をおこなう[10].

最後に，FPGA 上での動作の確認ができている RISC-V コアのハードウェアリソース使用量を図 11 に示す．なお，DSP は Digital Signal Processor, MMCM は Mixed Mode Clock Manager である．

	使用数	使用可能数	使用率
LUT	71	53200	0.13%
FF	37	106400	0.03%
BRAM	30	140	21.42%
IO	108	200	54%
BUFG	1	32	3.13%

図 10　マルウェア検知機構のハードウェア量

	使用数	使用可能数	使用率
LUT	31944	53200	69.4%
LUTRAM	1217	17400	6.7%
FF	15556	106400	15.59%
BRAM	24	140	17.14%
DSP	15	220	6.82%
IO	1	200	5.5%
BUFG	1	32	12.5%
MMCM	1	4	50.0%

図 11　RISC-V コアのハードウェア量

5.2 判別精度への影響

判別精度の評価は，Python 3.6.9 の Scikit-Learn をもちいてシミュレーション上でおこなう．まず学習用データを用いて機械学習を行って作成した判別モデルに，評価用データを 1 命令ずつ与え，良性，もしくは悪性の判別を行っていき，全命令における悪性と判別された割合を算出する．

5.2.1 評価環境

格納表のハードウェアリソース使用量を削減した結果を踏まえ，判別モデルが評価用データの判別の際に，判断材料としているビット幅の範囲を探すために，ビット幅のサイズを変更していき，その際の精度について測定をおこなう．本節ではヒット率のデータ型を，

①浮動小数点型 64 ビット幅 (縮小前，Float64)
②浮動小数点型 64 ビット幅 (縮小後，Float64)
③整数型 64 ビット幅 (縮小後，Int64)
④整数型 32 ビット幅 (縮小後，Int32)
⑤整数型 16 ビット幅 (縮小後，Int16)
⑥整数型 8 ビット幅 (縮小後，Int 8)

の 6 パターンに分ける．上記のパターンに分けた理由は以下の通りである．まず，格納表のサイズを 0<hit,access<10^5 とし，ヒット率のデータ型を Float64 としたものを①とする．①の格納表の縮小を行っていないパターンでは，論理合成ができないことが 5.1 節で確認できているので，他のパターンと精度比較をする際の参考までに測定する．

次に，格納表を縮小した場合について検討すると，0<hit,access<550 の範囲まではハードウェアリソース内に収まることが確認できている．そこで，他のハードウェア部分からのアドレス参照の容易性の観点から，アクセス数とヒット数のビット幅をそれぞれ 8bit まで（0<hit,access < 256）と制限する．この制限した格納表について，保持しているヒット率の値を縮小させない場合を②とし，それに対して，Int64 から Int8 の間で変化させたものを③～⑥とする．ヒット率のビット幅を縮小させるほどハードウェア量が少なくなるが，表現できる値の範囲は狭くなる．上記の 6 パターンについて，それぞれ判別精度を測定し，精度の変化を確認する．

5.2.2 評価結果

各パターンに基づき，判別精度の測定をおこなった結果を図 12 に示す．図の②～⑤のパターンをみると，ヒット率が 64 ビットから 16 ビットの範囲内では判別精度に変化が見られなかった．しかし，⑥のパターンで，8 ビットにまでビット幅を縮小すると，判別精度に若干のブレが生じているのが確認できる．この結果から，ヒット率のビット幅は，精度を保つためには，64 ビットの内，最上位ビットから 16 ビット分の値までは保持する必要があると考える．

データ型 / 命令	①縮小前	②Float64	③Int64	④Int32	⑤Int16	⑥Int8
悪性						
Mirai②	99.999	95.873	95.873	95.873	95.873	100.000
良性						
ls -l	95.144	96.694	96.694	96.694	96.694	99.913
mkdir test	99.663	99.678	99.678	99.678	99.678	99.804
chmod -x	99.708	99.708	99.708	99.708	99.708	99.831

図 12　各パターンでの判別精度

6. 考察

5.2 節にて，今回の測定では，全体を通して精度の変化が微量であり，ヒット率のビット幅の削減が，判別精度に与える影響は小さいことが確認できた．この評価結果についての考察をおこなう．今回の研究での判別精度の測定方法として，浮動小数点型から整数型への変換を行う場合，もとのヒット率に対し 10^n をかけ，小数点以下を切り捨てることで，各ヒット率の値を整数に変換し，その値を機械学習時の学習データとした．しかしこの場合，もととなるヒット率の値が小さい時，整数化をおこなうと少数のままの値については切り捨てられるので，結果として判別に用いられるデータの差異が少なくなる．

（例）10000 倍した場合，

0.004586… → 45.86… → 45

0.004513… → 45.13… → 45

この対策として全体へ一律に同じ値をかけるのではなく，1 命令ずつデータを調べ，少数の位で 0 以外の数字が初めてでた位からの桁数を確保する手法を提案する．この場合，はじめの 0 を全て省くことができるので，より下位の数字まで整数化が可能となるため，データ同士の重複を減すことができる．

（例）初めての 0 以外の箇所から 4 桁を切り出す場合，

0.004586… → 4586

0.004513… → 4513

これにより，判別器の機械学習時の有効なデータの数が増え，判別精度が向上し，格納表のサイズ縮小，及びヒット率のビット幅の削減を行った際の精度の変化が，今回の測定結果よりも大きくなると期待できる．

また，5.3 節で測定した結果のうち，図 10 と図 11 で示している測定結果を足し合わせたものが，本研究の提案機構全体を FPGA へ実装した際のハードウェアリソース使用量となる．したがって，マルウェア検知機構を備えた RISC-V は，FPGA に搭載可能な規模であり，また，ハードウェアリソース使用量全体のうち，RISC-V コアが占める割合が高いことがわかる．

7. まとめ

本研究では，先行研究で扱われた従来の提案機構を，FPGA へ実装するための予備調査をおこなった．従来の提案機構に対し，ハードウェアリソース使用量の消費が激しい除算器を用いての実装は困難であったため，除算器の代わりとなる格納表を提案し，FPGA 内に収まるようにサイズの縮小をおこなった．また，マルウェア検知機構での判別精度についての評価もおこない，精度への影響が最小限に収まる範囲でのヒット率のビット幅の測定もおこなった．さらに，RISC-V コアとマルウェア検知機構からなる提案機構のハードウェアリソース使用量が FPGA 内に収まることも確認できた．今後は，より高速な動作が可能な RISC-V コアである boom を用いた FPGA 実装や，ヒット率以外の特徴量を利用したマルウェア検知機構の検討などをおこない，より実用的な提案機構の作成を進める．

参考文献

[1] トレンドマイクロセキュリティブログ，Web アプリの脆弱性を利用する「Miori」などの複数の「Mirai」亜種の拡散を確認，https://blog.trendmicro.co.jp/archives/20045（accessed 2020/04）

[2] 総務省，令和元年版　情報通信白書第 1 部第 1 章デジタル経済を支える ICT の動向，https://www.soumu.go.jp/johotsusintokei/whitepaper/ja/r01/pdf/01honpen.pdf （accessed 2020/04）

[3] トレンドマイクロセキュリティブログ，ルータや IoT 機器に危険をもたらす管理用機能の放置，https://blog.trendmicro.co.jp/archives/14195（accessed 2020/04）

[4] NEC, 2020 年 4 月　IoT 機器想像業者はセキュリティ対策の具備が義務化へ，https://www.nec-nexs.com/sl/security/it/38.html（accessed 2020/02）

[5] 小池一樹，小林良太郎，加藤雅彦：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，Vol2019-DPS-178 No.14（2019/3）

[6] 赤松諒，小林良太郎： IoT 機器におけるマルウェア検知機構の精度緩和によるハードウェア料削減，2019 年度工学院大学卒業論文（2019/3）

[7] 永井雄也，小林良太郎，加藤雅彦，島田創：プロセッサ情報によるマルウェア検知における特徴量のビット数削減手法の検討，CSS2019（2019/10）

[8] AI Academy Media，ランダムフォレストとは ，https://aiacademy.jp/media/?p=252（accessed 2020/02）

[9] Rocket Chip on Zynq FPGAs，https://github.com/ucb-bar/fpga-zynq （accessed 2019/12）

[10] FPGA implementation of SKLearn Random Forest, https://github.com/johnbensnyder/FPGA_random_forest (accessed 2020/06)