Vol.2018-CSEC-83 No.11
2018/12/14

プログラムカウンタのアドレス空間の履歴に着目したCNNによるマルウェア検知

関野 翔太[1]　石川 亮太[1]　小林 良太郎[1]　加藤 雅彦[2]

概要：本研究ではプログラムカウンタに着目し，その履歴を CNN を用いたディープラーニングによって学習することで，実行中のマルウェアを検知する新しい手法を提案する．本手法は，プログラムカウンタの履歴を画像化することで CNN による学習を可能にし，動的な検知を可能とするものである．本稿では，本手法の有用性を示すため，実際にプログラムカウンタの履歴を用いて CNN による学習を行い，マルウェアが検知可能であることを検証した．

キーワード：マルウェア検知, ディープラーニング,CNN, 画像化

Detection of malware by CNN focusing on history of program counter address space

Sekino Shota[1]　Ishikawa Ryota[1]　Kobayashi Ryotaro[1]　Kato Masahiko[2]

1. はじめに

2016 年頃から急激に増加したランサムウェアによる被害は早くも大幅に減少し，代わりに 2017 年後期からはコインマイナーなどによる不正マイニングの被害が大幅に増加している [1] ように，マルウェアの攻撃方法は非常に速く変化しており，その対応が求められている．ランサムウェアとは ransom（身代金）の名前の通り，コンピュータに保存されたファイルなどを暗号化してアクセスできない状態にし，復号と引き換えに身代金の支払いを要求するマルウェアである．不正マイニングは，コインマイナーなどに感染した機器の計算資源を用いて，仮想通貨の採掘（マイニング）を行う攻撃である．ランサムウェアの被害は，攻撃手法の周知，アンチウイルスソフトによる対策，復号ツールの開発などにより大幅に減少したが，標的型攻撃のように特定の組織を狙った攻撃は継続している．

[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 1-1-1 Manabino, Nagayo-cho, Nishi-Sonogi-gun, Nagasaki, Japan

また，流行したマルウェアの一部を改変したり，機能を追加したりした亜種が作られるケースが多く見られる．代表的なマルウェア検知方式であるシグネチャ方式では，既知のマルウェアのハッシュ値を保存し，同じハッシュ値を持つソフトウェアをマルウェアとして検知する．亜種を作る際に行われた改変によって，ハッシュ値も変化するため，シグネチャ方式では亜種を検知できなくなることがある．こうした亜種に対抗するために，プログラムの不正な挙動を検出するビヘイビア方式がある．しかしながらビヘイビア方式は，不正な挙動であると判断する条件の設定が困難である [2].

そこで本研究では，ビヘイビア方式の実装として，プログラムカウンタ [*1] の履歴を画像化したデータを学習データとして用い，CNN（Convolutional Neural Network）によってマルウェアを検知する新しい手法を提案する．提案手法では，不正な挙動を判定するために，プログラムカウンタの履歴を用いる．そのため，条件の設定は閾値などに限られるため，条件の設定が容易である．また，新種や亜種

[*1] インストラクションポインタ，命令ポインタなどとも呼ばれる．本稿ではプログラムカウンタという呼称で統一する．

図 1　手法の概略図
Fig. 1　Schematic diagram of the method

が出現した場合にも，そのマルウェアを動作させプログラムカウンタの履歴を新たに学習データに追加することで，検知することが可能になる．さらに，プログラムの一部を変更しても，その基本的な動作が変わらなければ同じプログラムの可能性が高いと判断できるので，追加の学習を行わなくても亜種の検出が可能である．

2. 関連研究

既に，機械学習やディープラーニングを用いて，マルウェアを分類，検知する手法が複数提案されている．

鈴木らは，Android マルウェアおよび PDF 文書型マルウェアに対して，バイナリデータを画像化し機械学習によりマルウェアを分類する手法の検証を行っている [3]．

Nataraj らは，複数のマルウェアのファミリーの分類を，バイナリを画像化することで行う手法を検証している [4]．

Su らは，IoT 機器のマルウェアを検知する手法として，畳み込みニューラルネット画像分類機器を用いた手法を提案している [5]．

これらの手法は，いずれもマルウェアのバイナリを解析対象としており，バイナリの内容が大きく変化するような亜種や，圧縮などによってバイナリの変化したマルウェアを検出できない可能性が高い．また，ファイルレスマルウェアと呼ばれる，ソフトウェアを介さず，Windows や Linux の機能を用いたり，直接メモリにマルウェアをロードするなどの方法で攻撃を試みるマルウェア [6] も存在しており，このようなタイプのマルウェアはバイナリを取得できないため，前述の手法では原理的に検出が不可能である．

それに対し，提案手法ではプログラムカウンタの履歴を用いるため，バイナリが変化していても同じ動作を行えばマルウェアを検出することが可能である．また，プログラムカウンタはコンピュータが何らかの動作をする限り変化するので，ファイルレスマルウェアに対しても一定の効果が見込める．

3. 提案手法

提案手法は，プログラムカウンタの履歴を取得して画像化し，マルウェアの判定を CNN を用いたディープラーニングで行うものである．手法の概略図を図 1 に示す．

概略図のうち，プログラムカウンタの履歴を画像化するプロセスと，マルウェアの判定を行うプロセスの詳細を次節で示す．

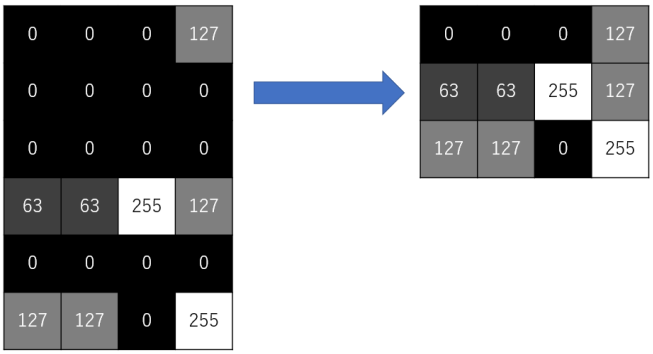

図 2　画像から行を削除する例
Fig. 2　Example of delete row from image

3.1 画像化のプロセス

プログラムカウンタの履歴を取得し，画像に変換する．画像化の方法は 4 通り考案し，それぞれについて検討を行った．なお，システムは 32bit，配列は zero-based index であり，配列が確保された時点でその配列の全ての要素が 0 であることを前提としている．

3.1.1 画像化方法 1

各要素が符号なし 8bit で，$256 \times \frac{2^{32}}{256}$ の 2 次元配列 $\mathrm{image}(x, y)$ を確保する．プログラムカウンタの値（以下，式中では PC で示す）が変化するたびに，式 (1)(2) に従い対応する要素の色を変更する．

$$\mathrm{changecolor}(x) = x \leftarrow 255 \tag{1}$$

$$\mathrm{changecolor}(\mathrm{image}(mod(\mathrm{PC}, 256), \lfloor \frac{\mathrm{PC}}{256} \rfloor)) \tag{2}$$

こうして得られた 2 次元配列 $\mathrm{image}(x, y)$ を，0 から 255 の 256 階調のグレースケール画像とみなして画像化する．こうして得られた画像から，横方向（x 方向）の要素が全て黒（値が 0）である行を削除する．例を図 2 に示す．例では，4 × 6 ピクセルの画像を用いている．また，見た目では 1 階調の違いを見分けることは困難なので，ピクセルごとの数値を表示している．

最後に，画像の高さ（y 方向）を幅と同じ 256 ピクセルになるように拡大，または縮小する．本研究では，いずれの画像化方法でも拡大，および縮小の際の補完に Bicubic 法を用いている．

3.1.2 画像化方法 2

各要素が符号なし 8bit で，$256 \times \lceil \frac{2^{32}}{256 \times 3} \rceil \times 3$ の 3 次元配列 $\mathrm{image}(x, y, rgb)$ を確保する．プログラムカウンタの値が変化するたびに，式 (1)(3) に従い対応する要素をインクリメントする．

$$\begin{aligned}\text{changecolor}(\text{image}(&mod(\text{PC}, 256\times 3),\\ &\lfloor\frac{\text{PC}}{256\times 3}\rfloor,\\ &mod(\text{PC}, 3)))\end{aligned} \tag{3}$$

こうして得られた 3 次元配列 image(x, y, rgb) を，RGB 成分がそれぞれ 0 から 255 の 256 階調，16777216 色のカラー画像とみなして画像化する．こうして得られた画像から，横方向（x 方向）の要素が全て黒（RGB の値が全て 0）である行を削除する．

最後に，3.1.1 の画像化方法 1 と同じ手順で拡大，または縮小処理を行い，256 × 256 ピクセルの画像に変換する．

3.1.3 画像化方法 3

各要素が符号なし 8bit で，$256\times\lceil\frac{2^{32}}{255\times 256}\rceil$ の 2 次元配列 image(x, y) を確保する．プログラムカウンタの値が変化するたびに，式 (4)(5) に従い対応する要素をインクリメントする．

$$\text{increment}(x) = \begin{cases} x \leftarrow x+1 & (x < 255)\\ x \leftarrow x & (otherwise)\end{cases} \tag{4}$$

$$\begin{aligned}\text{increment}(\text{image}(&mod(\lfloor\frac{\text{PC}}{255}\rfloor, 256),\\ &\lfloor\frac{1}{256}\lfloor\frac{\text{PC}}{255}\rfloor\rfloor))\end{aligned} \tag{5}$$

こうして得られた 2 次元配列 image(x, y) を，3.1.1 の画像化方法 1 と同じ手順で行の削除，拡大，または縮小処理を行い，256 × 256 ピクセルの画像に変換する．

3.1.4 画像化方法 4

各要素が符号なし 8bit で，$256\times\lceil\frac{2^{32}}{256\times 255\times 3}\rceil\times 3$ の 3 次元配列 image(x, y, rgb) を確保する．プログラムカウンタの値が変化するたびに，式 (4)(6) に従い対応する要素をインクリメントする．

$$\begin{aligned}\text{increment}(\text{image}(&mod(\text{PC}, 256\times 255\times 3),\\ &\lfloor\frac{\text{PC}}{256\times 255\times 3}\rfloor,\\ &mod(\lfloor\frac{\text{PC}}{255}\rfloor, 3)))\end{aligned} \tag{6}$$

こうして得られた 3 次元配列 image(x, y, rgb) を，3.1.2 の画像化方法 2 と同じ手順で行の削除，拡大，または縮小処理を行い，256 × 256 ピクセルの画像に変換する．

3.2 判定のプロセス

3.1 で述べた方法により生成した画像を入力データとして，CNN を用いたディープラーニングによりマルウェアの判定を行う．判定のために，まず学習データを用いて，マルウェアを判別するための学習を行う必要がある．通常のディープラーニングでは，初期の重みをランダムに決定し学習を開始するが，本研究では学習時間の短縮と精度の向上のため，Fine tuning を利用する．Fine tuning とは，予め別の目的のために学習済みのモデルと重みのうち，上位の層を固定し特徴量の抽出に利用し，下位の層を再学習することによって，既存のモデルを別の用途に転用する技術である．本研究では，VGG16[*2] を用いて Fine tuning を行い，マルウェア検知を行うことができるように変更を行った．作成したモデルを図 3 に示す．図のモデルのうち，input と block1 から block5 までは VGG16 のものであり，最下層の sequential は独自に作成したものである．sequential の内容についても，図 3 の右側に示している．学習にあたっては，モデルのうち，block4 までの層を固定し，block5 とその下位の層で再学習を行う．

あるソフトウェアがマルウェアであるか判定を行うには，判定対象のプログラムを動作させ，プログラムカウンタの履歴を取得して画像化し，上記の手順で学習したモデルを用いて判定を行えばよい．

4. 実装

この章では，実装について述べる．

まず，提案手法ではプログラムカウンタの値を用いるため，プログラムカウンタの値を取得する必要があるが，ソフトウェアからプログラムカウンタの値を取得することは，通常の環境ではできない．この問題を解決するために専用のハードウェアを用いるといった方法も提案されている [7] が，本稿の目的は提案手法が有効であることを示すことであるので，仮想マシンを利用して，仮想マシンのプログラムカウンタをホストマシンで取得する方法を用いる．本研究では，仮想マシンとして，オープンソースの QEMU[8] をプログラムカウンタの値を取得できるよう一部改変して用いている．

また，ディープラーニングによって学習を行うにあたり，多くの計算資源を要求される．そこで，学習には Google Colaboratory[9] を用いた．Google Colaboratory は無料で利用できる Jupyter ノートブック環境で，設定からランタイムを GPU に変更すると，クラウド上の GPU（NVIDIA Tesla K80）を用いて計算を行うことができる．

実装に用いた環境の一覧を表 1 に示す．Local は物理マシンで，画像化の処理と，仮想マシンのホストとして用いた．Virtual は Local 上で動作している仮想マシンで，実際にソフトウェアやマルウェアを動作させた．Colaboratory は Google Colaboratory が提供するサーバ上で動作している仮想マシンで，モデルの学習に用いた．

5. 評価

この章では評価を行う方法と，その結果について述べる．

[*2] ILSVRC で提案された 1000 クラスの画像を判別可能な学習済みモデル

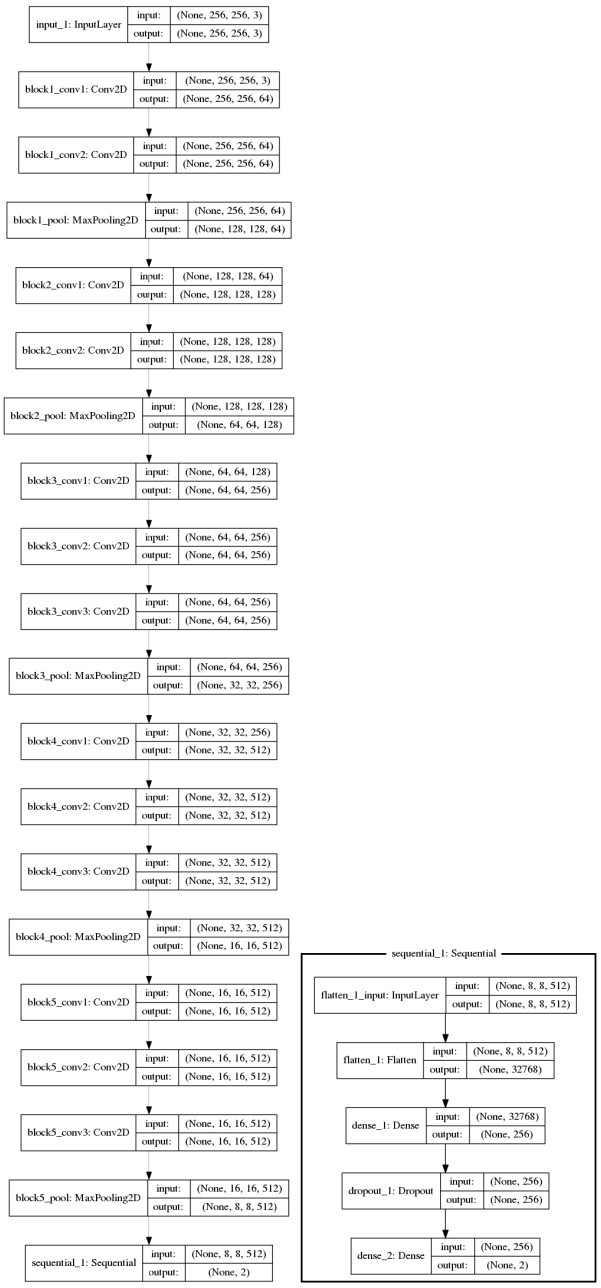

図 3 Fine tuning を用いて作成したモデル
Fig. 3 Created model using Fine tuning

表 1 使用したマシンの一覧
Table 1 List of used machines

	Local	Virtual	Colaboratory
OS	Ubuntu16.04	Windows7	Ubuntu17.10
Memory	7.7GiB	4GiB	13GB
CPU	Intel Core i7	QEMU Virtual	Intel Xeon
GPU	-	-	Tesla K80

5.1 評価方法

マルウェアでないソフトウェアと，マルウェアをそれぞれ準備し，それらを仮想マシンで動作させ，ホストマシンでプログラムカウンタの値の履歴を取得する．今回は，対象のソフトウェアおよびマルウェアの実行から 1000000 回のプログラムカウンタの変化を取得した．取得した履歴を，3.1 で示した方法によりそれぞれ画像化する．今回は，画像化したデータのうち 70%を学習データ，30%を判定データとして，無作為に振り分ける．こうして得られた学習データを用いて，マルウェアを検知するモデルの学習を行う．学習が完了したら，学習したモデルを用いて，判定データに対してマルウェアの判定を行う．判定の正解率，検知率，誤検知率の指標を根拠として，その手法の有用性を評価する．

マルウェアでないソフトウェアを取得するにあたり，様々な種類を幅広く収集し，また収集するソフトウェアを作為的に選択していないことを示すために，窓の杜 [*3][10] に掲載されているソフトウェアを収集する．収集するソフトウェアは，窓の杜のソフトウェアの各サブカテゴリから 1 つのソフトウェアを収集する．収集するソフトウェアは次の手順で決定する．

(1) 対象のサブカテゴリから，フリーソフトのみを抜き出す．
(2) 抜き出したフリーソフトを，無作為の順番に並び替える．
(3) 並び替えた結果，先頭に来たフリーソフトを対象とし，収集する．
(4) ただし，対象のカテゴリにフリーソフトが存在しない場合は，収集対象から除外する．

この手順で，156 個のマルウェアでないソフトウェアを収集した．それらのソフトウェアのうち，ランタイムや OS のバージョンの関係などで動作しなかったソフトウェア 30 個を除き，126 個のソフトウェアのプログラムカウンタの値の履歴を取得した．

また，マルウェアについては，GitHub で公開されている theZoo[10] を用いた．theZoo には，マルウェア解析用に実際に動作するマルウェアが含まれている．それらのマルウェアのうち 30 個のマルウェアのプログラムカウンタの値の履歴を取得した．30 個のうち，11 個はランサムウェア，14 個はトロイの木馬，残り 5 個はスパイウェアやワームなどのマルウェアである．

5.2 評価結果

まず，画像化方法 1 と画像化方法 2 について画像化の処理を行った．しかしながら，これらの方法は縮小処理を行う前の画像サイズがおよそ 4.3GB と非常に大きくなってしまい，学習と判定に大きな支障が出ることが分かった．画像サイズが大きいため，学習時には大量の記憶領域を消費し，判定時には時間を要してしまう．そのため，この 2 つの画像化方法は実装が現実的でないと判断し，評価は行わないこととした．

[*3] フリーソフト紹介サイト

表 2　判定結果
Table 2　Judgment result

	正解率	検知率	誤検知率
画像化方法 3	80.85%	62.50%	15.38%
	85.11%	30.00%	0.00%
	87.23%	60.00%	5.41%
画像化方法 4	87.23%	97.37%	55.56%
	80.85%	87.18%	50.00%
	76.00%	100.00%	100.00%

続いて，画像化方法 3 と画像化方法 4 について画像化の処理を行った．これらの画像化方法では，画像サイズが画像化方法 3 で 50MB，画像化方法 4 で 17BM ほどなので問題ないことが分かった．それぞれの画像化方法で学習と判定を行った結果を表 2 に示す．学習データと判定データの分割をランダムに行っているため，選ばれたデータによっては極端な外れ値が出る可能性もある．そこで，学習を 3 回行い，それぞれの結果について表記した．

なお，正解率とは，全てのデータのうちマルウェアであるか否かを正しく判定された割合であり，検知率とは，マルウェアのうちマルウェアであると判定された割合であり，誤検知率とは，マルウェアでないデータのうちマルウェアと判定された割合である．

また，表 2 のデータは学習に用いていないデータ（未知のデータ）の判定結果であるが，学習に用いたデータ（既知のデータ）を判定した場合，正解率はいずれの場合でも 100%である．

6. 考察

まず，画像化方法 3 ではいずれも 80%以上の正解率が得られた．また，検知率は 30%から 60%ほどと高くないが，誤検知率はかなり低く抑えられている．実際に実装するにあたり，誤検知率が高いとユーザがマルウェア検知のアラートに慣れてしまい，実際の感染時の対応が遅れるといったケースがある．そのため，誤検知率が低い点も大きな利点であると考えられる．

反対に，画像化方法 4 では検知率は最低でも 87%と高いが，同時に誤検知率も 50%以上と高くなっている．この方法は，セキュリティを高く保つ必要のある決済システムや個人情報を取り扱うシステムなどの，誤検知が多くてもマルウェアを確実に検出したい環境での利用に適していると言える．

どちらの画像化方法についても，特徴を考慮して使用する場面を適切に選択する必要がある．

また，今回は画像化方法を独立に評価したが，複数の画像化方法による判定を組み合わせて用いることで，検知率や誤検知率を改善できるのではないかと考えられる．例えば，誤検知率の高い画像化方法 4 でマルウェアと判断されたデータも，誤検知率の低い画像化方法 3 でマルウェアでないと判断されれば，最終的にマルウェアでないと判定する方法などが考えられる．

さらに，今回は学習データと判定データを無作為に振り分けていたが，これを無作為ではなく適切に振り分けることで，検知率を向上したり，誤検知率を低減するといったことが可能であると推測できる．表 2 に示したように，同じデータでも振り分けによって検知率や誤検知率に差が出ており，これは振り分けの方法によって生じているのではないかと考えられる．そこで，例えば各マルウェアファミリーから数個ずつを学習データとして用いるようにすれば，そのファミリーのマルウェアを検出できる可能性が高いと考えられるので，検知率の向上を期待できる．また，今回は網羅的に収集したマルウェアでないソフトウェアを分割して利用したが，分割するのではなく全てのデータを学習に用いて，判定用には別途用意したデータを用いればより網羅的に学習ができるので，誤検知率の低減を期待できる．いずれの場合でも，どのような振り分けが最適であるかは検討の余地がある．

7. まとめ

日々出現するマルウェアやその亜種に対抗するため，CNN を用いた新しいマルウェア検知手法を提案した．本稿では，画像化方法と学習の方法について検討を行った．画像化方法については 4 種類の方法を考案した．考案した画像化方法のうち，問題のなかった 2 種類の方法で学習と判定を行い，その有効性の評価を行った．今後は，検知率と誤検知率の改善をすると共に，実際に専用ハードウェアを用いてリアルタイムに判定を行うことを想定し，判定時間なども考慮して検討を行う必要がある．

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] トレンドマイクロ:サイバー犯罪の狙いは「ランサムウェア」から「不正マイニング」へ、2018 年第 1 四半期の脅威動向を分析，入手先 ⟨http://blog.trendmicro.co.jp/archives/17460⟩ (2018.11.01).
[2] 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.
[3] 鈴木 貴之，笠間 貴弘，宮保 憲治. バイナリデータの画像化を活用したマルウェア分類法の検討，平成 27 年度電子情報通信学会東京支部学生会研究発表会，pp.205，2016.
[4] L. Nataraj, S. Karthikeyan, G. Jacob, B. S. Manjunath. Malware Images: Visualization and Automatic Classification, VizSec '11 Proceedings of the 8th International Symposium on Visualization for Cyber Security Article No. 4.
[5] J. Su, D. V. Vargas, S. Prasad, D. Sgandurra, Y. Feng, K. Sakurai. 画像分類を用いる IoT マルウェアの検知手法についての提案，Information Processing Society of

Japan，2018.
[6] IPA: 長期感染の実態と攻撃証跡の分析，入手先 ⟨https://www.ipa.go.jp/files/000062281.pdf⟩ (2018.11.06).
[7] 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，Information Processing Society of Japan，2018.
[8] QEMU 入手先 ⟨https://www.qemu.org/⟩ (2018.11.16)
[9] Google Colaboratory 入手先 ⟨https://colab.research.google.com/⟩ (2018.11.16)
[10] 窓の杜 入手先 ⟨https://forest.watch.impress.co.jp/⟩ (2018.11.16)
[11] theZoo 入手先 ⟨https://github.com/ytisf/theZoo⟩ (2018.11.09)