Vol.2019-CSEC-87 No.7
2019/12/4

画像処理ベースのプログラム識別を目的としたプログラムの挙動の可視化に関する検討

石川 亮太[1]　小林 良太郎[1]　加藤 雅彦[2]　嶋田 創[3]

概要：近年，情報窃取やサービス不能攻撃を代表とする組織内外への攻撃など，悪意のある活動を行うマルウェアと呼ばれるプログラムによる被害が，情報セキュリティにおいて大きな問題となっている．マルウェアを機械学習や深層学習を用いて検知する研究が広く行われているが，その検知率にはまだまだ課題が残る状況である．一方で，画像処理における機械学習や深層学習による画像の分類や検出の技術は著しく進歩している．そこで，マルウェアの検知に，それらの画像処理技術を用いることを提案する．本稿では画像処理技術を用いる前段階として，プログラムの動作特徴を画像にするための可視化を検討した．具体的には，DBI (Dynamic Binary Instrumentation) を用いてプログラムがアクセスしたメモリアドレスのログを取得し，空間に 3 次元でプロットする．そして，各メモリアドレスのアクセスの頻度に応じて，プロットしている点の形状・色を変化させる．本手法の有用性を示すため，実際にプログラムがアクセスしたメモリアドレスのログを用いて可視化を行い，画像を比較することでプログラムが視覚的に分類可能であることを検証した．

キーワード：動的解析, メモリアクセス, 可視化

Visualization of Program Behavior for Image Processing Based Program Identification

Ishikawa Ryota[1]　Kobayashi Ryotaro[1]　Kato Masahiko[2]　Shimada Hajime[3]

Abstract: In recent years, there is a major problem in information security caused by malicious programs (malware) that perform malicious activities. Malware performs many malicious activities to both internal and external side such as information theft and denial of service attacks. There are many researches to detect these malicious programs using machine learning and deep learning, but the detection rate are still insufficient. On the other hand, there are remarkable advancements in image classification and detection techniques by machine learning and deep learning. Therefore, we propose to use these image processing techniques for program classification and identification. In this paper, we examined the visualization process to make behavioral characteristics of the program into an image or a motion picture as a first step to utilize image processing technology. As a detail process, we use DBI (Dynamic Binary Instrumentation) to obtain a memory access log of a program and plot it into 3D space. The shape and color of the plotted points are changed according to the access frequency. In order to show the usefulness of this method, we performed visualization examples using a memory access log of some programs and verified that they can be classified visually.

Keywords: dynamic analysis, memory access, visualization

[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, Nishi-Sonogi-gun, Nagasaki, Japan
[3] 名古屋大学
Nagoya University Furo-cho, Chikusa-ku, Nagoya, Aichi,

1. はじめに

近年，画像処理における深層学習は著しく進歩している．深層学習の技術は，人間の神経細胞（ニューロン）の仕組みを模したシステムであるニューラルネットワークがベースになっている．多層にしたニューラルネットワーク（DNN）を用いることで，データに含まれる特徴を段階的により深く学習することが可能になっている．多層構造のニューラルネットワークに大量の画像などを入力することで，コンピュータのモデルはデータに含まれる特徴を各層で自動的に学習している．この構造と学習の手法が深層学習の特徴であり，極めて高い精度を誇っている．MathWorks の調査によると，近年の進歩により，画像認識などのタスクにおいて，人間の認知能力を超えるまでになっている [1]．

一方で，情報窃取やサービス不能攻撃を代表とする組織内外への攻撃など，悪意のある活動を行うマルウェアと呼ばれる悪性のプログラムによる被害が，情報セキュリティにおいて大きな問題となっている．マルウェアの対策の 1 つに，アンチウイルスソフトの利用が挙げられる．シグネチャ型のアンチウイルスソフトでは，マルウェアに現れる共通のコードやハッシュ値などをあらかじめ解析し，対象のファイルにそれらが含まれるかどうかを検査している．この手法を利用することで，データベースに登録されているシグネチャを持つマルウェアは検知でき，誤検知を大きく抑えることができる．しかし，既知のマルウェアのコードを難読化したものや，パッキングを施した亜種は誤検知を起こす恐れがある．またマルウェアのシグネチャを抽出するために，検体を解析しなければならない．近年ではマルウェアが大幅に増加しており，AV-TEST の調査によると，毎日 35 万を超えるマルウェアが出現している [2]．このことから，マルウェアを全て解析することが困難になっている．

そこで，シグネチャに依存しない方法として，プログラムの振る舞いに着目する，ビヘイビア法を用いたマルウェア検知の研究が広く行われている．ビヘイビア型マルウェア検知の研究では，マルウェアの実行時に現れる特徴を収集し，深層学習を用いた分類を行うことによって，未知のマルウェア検知を目指している．

我々は，プログラムの挙動を可視化したものを静止画像や動画像にし，それらを特徴量とした深層学習を用いることで，マルウェアを判別する手法を提案する．挙動としてはプログラムによるメモリアクセス[*1]に着目する．この提案手法の概略図を図 1 に示す．本稿では，図 1 のうち，メモリアクセスログを取得し，メモリアドレス毎のアクセス頻度を可視化する部分に焦点を当てる．

Japan

[*1] プログラムカウンタの値，ロード命令，ストア命令が示すメモリ空間上のアドレスへのアクセス

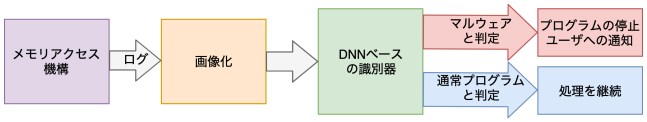

図 1 提案手法の概略図
Fig. 1 Schematic diagram of the proposal

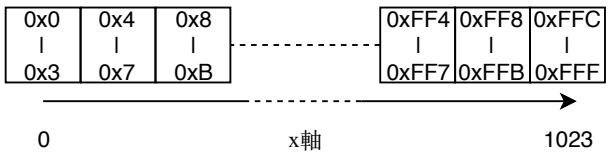

図 2 メモリアドレスに対応する x 軸座標
Fig. 2 x axis corresponding to the memory address

2 章では提案手法について述べる．3 章では実装方法について述べ，4 章で評価環境と結果を示す．5 章で考察を述べ，6 章で関連研究との比較を述べ，7 章で本論文をまとめる．

2. 提案手法

本章では，メモリアクセスを可視化する空間と，プロットの色や大きさの変化のプロセス，可視化プロセスの詳細を示す．

2.1 メモリアクセスを可視化する 3 次元空間の構成法

提案手法は，メモリへのアクセス頻度を描画する．プログラムが実行する際にアクセスするメモリ空間は広大であるため，各辺を 1024 個に区切った 3 次元空間を用意し，この空間内にプロットする．また，本稿では暫定的に 1 座標あたり 4Byte を割り当てる．x 軸，x-y 平面，x-y-z 立体がそれぞれ，4KB 分の軸，4MB 分の平面，4GB 分の立体に相当している．メモリアドレスに対応する x 軸方向の座標を図 2，立体における座標を図 3 に示す．描画対象のプログラムが使用しない空間も情報に含めることで，空間上の距離や位置も情報量として維持できる．メモリアドレスと用意したメモリ空間の対応は，メモリアドレスを $addr$，3 次元空間座標を (x, y, z) として式 (1)(2)(3) により求められる．

$$x = \left(\frac{addr}{4} \ \% \ 1024^2\right) \ \% \ 1024 \tag{1}$$

$$y = \left(\frac{addr}{4} \ \% \ 1024^2\right) \frac{1}{1024} \tag{2}$$

$$z = \frac{addr}{4} \cdot \frac{1}{1024^2} \tag{3}$$

2.2 メモリへのアクセス頻度を基にしたプロットの変化

提案手法では，アクセス頻度を基にしてメモリアクセスを可視化する．図 4 にメモリアクセスの可視化例を示す．図 4 では z 軸方向の奥行きを省略している．図 4 のように

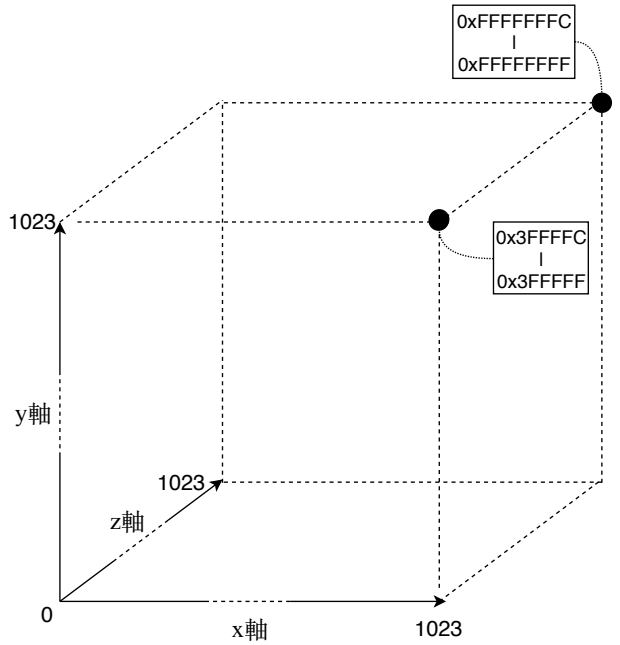

図 3　メモリアドレスに対応する x-y-z 立方体
Fig. 3　x-y-z cube corresponding to the memory address

空間のプロットとして，色や大きさをヒートマップのように変化させ，メモリアクセスを可視化する．

色の変化はヒートマップに相当するものを用いて，アクセス頻度が低い場合は青になり，徐々に頻度が上がるに連れて赤に変化していく．ある頻度 f の色は，f_{max} を頻度の最大値，f_{min} を頻度の最小値として式 (4) により正規化し，R,G,B を色の成分として式 (5)(6)(7) により求める．式 (5)(6)(7) と色の成分の関係を図 5 に示す．f_{max} と f_{min} は全プロットのうちの頻度の最大値と最小値の場合，1 度しかアクセスされない巨大なメモリ空間がある場合に全体的に青に寄ることや，ループカウンタ変数の膨大なアクセスがある場合に全体的に赤に寄ることが考えられる．色が赤や青に寄る場合，プロット同士の色差が小さくなり，アクセス頻度差が視覚的に認識しづらくなることが考えられる．そのため本稿では，色がアクセス頻度の偏りに影響しにくくなるように全プロットの頻度の第 1 四分位数を f_{min}，第 3 四分位数を f_{max} とした．

プロットの大きさはヒートマップを模して，頻度が低い場合は小さく，徐々に頻度が上がるにつれて大きく変化していく．ループカウンタ変数などによる局所的な頻度の爆発的な上昇がある場合，他のプロットに対して巨大化することが考えられる，そのため，頻度が上がるにつれて非線型[*2]に大きさを変化させる．また，プロットの大きさに比例して透過度を上げることで，プロットが大きくなった際にも周りのプロットが隠れない．

$$f' = \frac{f - f_{min}}{f_{max} - f_{min}} \tag{4}$$

[*2] 大きさは 3^{-1} 乗で減衰する

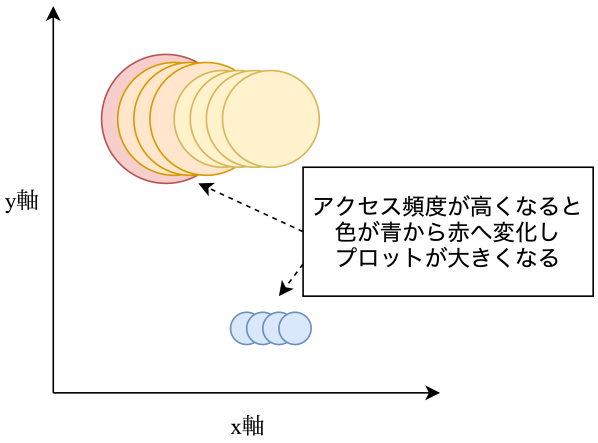

図 4　メモリアクセスの可視化例
Fig. 4　An example of memory access visualization

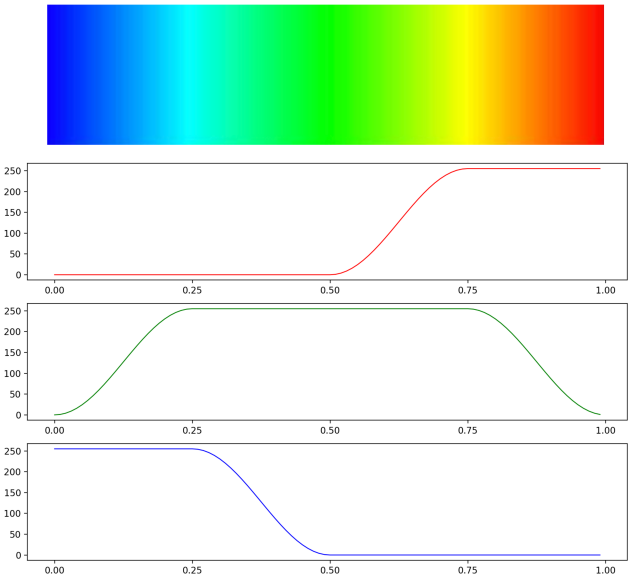

図 5　アクセス頻度と色の関係
Fig. 5　Relationship between access frequency and color

$$R = \begin{cases} 255 & (f' \geq \frac{3}{4}) \\ 255\sin^2 2\pi f' & (\frac{3}{4} > f' \geq \frac{1}{2}) \\ 0 & (f' < \frac{1}{2}) \end{cases} \tag{5}$$

$$G = \begin{cases} 255 & (\frac{3}{4} > f' \geq \frac{1}{4}) \\ 255\sin^2 2\pi f' & (1 > f' \geq \frac{3}{4},\ \frac{1}{4} > f' \geq 0) \\ 0 & (f' \geq 1,\ f' < 0) \end{cases} \tag{6}$$

$$B = \begin{cases} 255 & (f' < \frac{1}{4}) \\ 255\sin^2 2\pi f' & (\frac{1}{2} > f' \geq \frac{1}{4}) \\ 0 & (f' \geq \frac{1}{2}) \end{cases} \tag{7}$$

2.3　可視化プロセス

メモリアドレスに対応する 3 次元空間の各座標の頻度は 0 から始まり，0 の時にはプロットは表示せず，0 以上の値をとる．メモリアクセスのログを 1 アクセス分読み込み，対応する座標の要素を増加させる．座標ごとの要素を座標ごとの頻度として 2.2 節の変化方法に従い，対応する

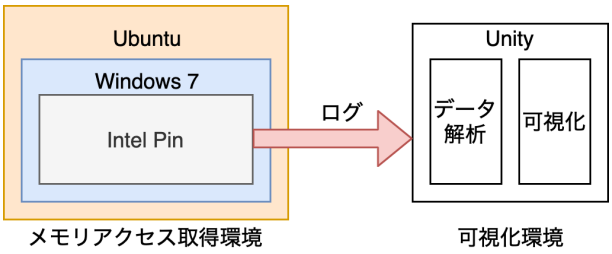

図 6　メモリアクセス可視化システム
Fig. 6　Memory access visualization system

プロットの色や大きさを変化させる．最後に空間の全要素を減少させ，次の 1 アクセス分を読み込み同じ動作を繰り返す．増加や減少させる大きさを調整することにより，任意の時間間隔においての頻度を求めることができる．

3. 実装

本章では，実装について述べる．本システムはメモリアクセス取得環境と可視化環境の 2 つからなっている．概要を図 6 に示す．メモリアクセス取得環境では，Ubuntu 18.04 上にゲスト OS として Windows 7 Professional を解析環境として用意し，解析環境上においてプログラム検体を実行し，Intel Pin[3] を用いて解析することによってメモリアクセスの実行トレースを取得する．可視化環境では，Unity を用いてプロット，頻度計算による色や形の変形を行い描画する．可視化環境は Unity Editor を用いて実装を行い，解析者は，頻度変化度合，色のバランス，大きさの変化率，位置，描画リセットなどの設定をインターフェイス上で変更できるようにした．可視化結果は動画像で表示される．

4. 評価

本章では評価を行う環境，その可視化した結果について述べる．

4.1 評価環境

評価は行う環境別に評価 1，評価 2，評価 3 とする．本稿では，プログラムを可視化し，それらの挙動が視覚的に認識できることを根拠に，提案手法の有用性を評価する．ただし，解析者が設定できるパラメータが多く，全ての組み合わせは膨大であるため，注目すべき結果のみ示す．また，今後深層学習を用いることを想定しているため，可視化結果では，任意の時間や角度で切り出した静止画像を示す．

4.1.1 評価 1

評価 1 では 2 種類のプログラムを用いる．1 つ目は Windows コマンドの ipconfig である．2 つ目はコンソール上に “Hello, world!” と出力する自作のプログラム（以下，hello と呼ぶ）である．これらのプログラムに対して，提案手法により可視化を試みる．

4.1.2 評価 2

評価 2 では 4 種類のプログラムを用いる．用いるプログラムの情報を表 1 に示す．for_normal は 100×100 の 2 次元配列を確保し，その配列に対し for 文で順に 0 で初期化する自作のプログラムである．その他の 3 種類は for_normal を UPX(Ver. 3.91)，UPX(Ver. 3.95w)，ASPack(Ver. 2.43) の 3 種類のパッカーを使用し，パッキングしたものである．これらのプログラムに対して，提案手法により可視化を試みる．バージョン違いの UPX については，それらを用いてパッキングしたプログラムのハッシュ値が異なるため，亜種を生成するために用いる．

4.1.3 評価 3

評価 3 では，2 種類の Potentially Unwanted Application（以下，PUA と呼ぶ）に対して，提案手法により可視化を試みる．表 2 に評価 3 で用いた PUA の情報を示す．用いる PUA は VirusTotal[4] によると，sham1 は全 69 個のセキュリティソフトの内，60 個でマルウェアとして検出され，sham2 は 59 個でマルウェアとして検出される．また Symantec[5] によると，どちらも “SMG.Heur!gen” というマルウェアとして検出される．

4.2 可視化結果

評価 1 の結果として，図 7 に ipconfig と hello のメモリアクセスの全体部分を可視化した結果を示す．

評価 2 の結果として，図 8 に for_normal，for_upx1，for_upx2，for_aspack のメモリアクセスの全体部分を可視化した結果を示す．図 9 に for_normal のオリジナル処理の候補部分を可視化した結果を示す．オリジナル処理は，for_normal の 100 × 100 の 2 次元配列を確保し，その配列に対し for 文で順に 0 で初期化するメイン関数の処理とする．また，動画像で出力された可視化結果からオリジナル処理に相当すると判断した部分を，オリジナル処理の候補部分として抽出した．図 10 に for_upx1，for_upx2，for_aspack のアンパック処理の候補部分とオリジナル処理の候補部分を可視化した結果を示す．図 10 では，「プログラム名：処理名」の形式でプログラム名と，そのプログラムの処理候補名を示す．

評価 3 の結果として，図 11 に sham1，sham2 のメモリアクセスの全体部分を可視化した結果を示す．

なお，各図における赤青緑の軸はそれぞれ，x 軸，y 軸，z 軸である．

5. 考察

まず，図 7 の可視化結果より，プログラムの動作により異なる可視化結果が得られることがわかった．さらに，図 8 の可視化結果より，for_upx1，for_upx2，for_aspack の可視化結果の中に for_normal の可視化結果に酷似している部分があることがわかるが，その箇所は x 軸に平行に移

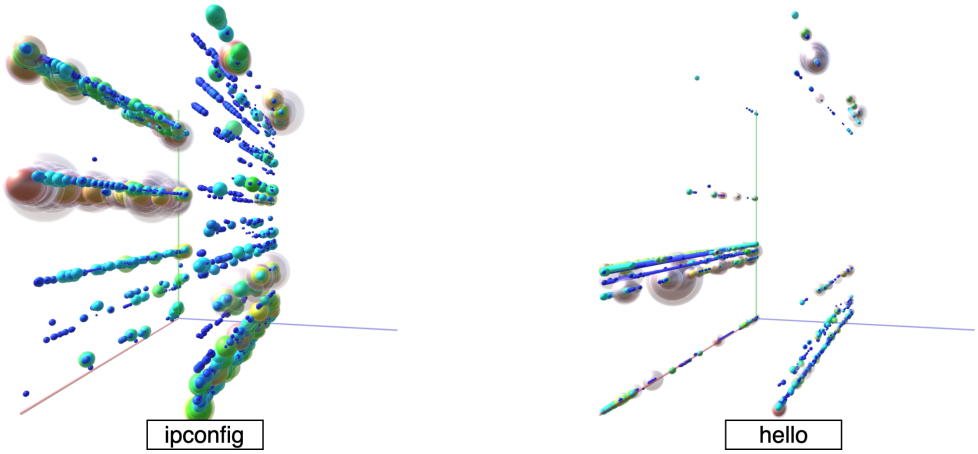

図 7　ipconfig, hello の可視化例
Fig. 7　Visualization example of ipconfig and hello

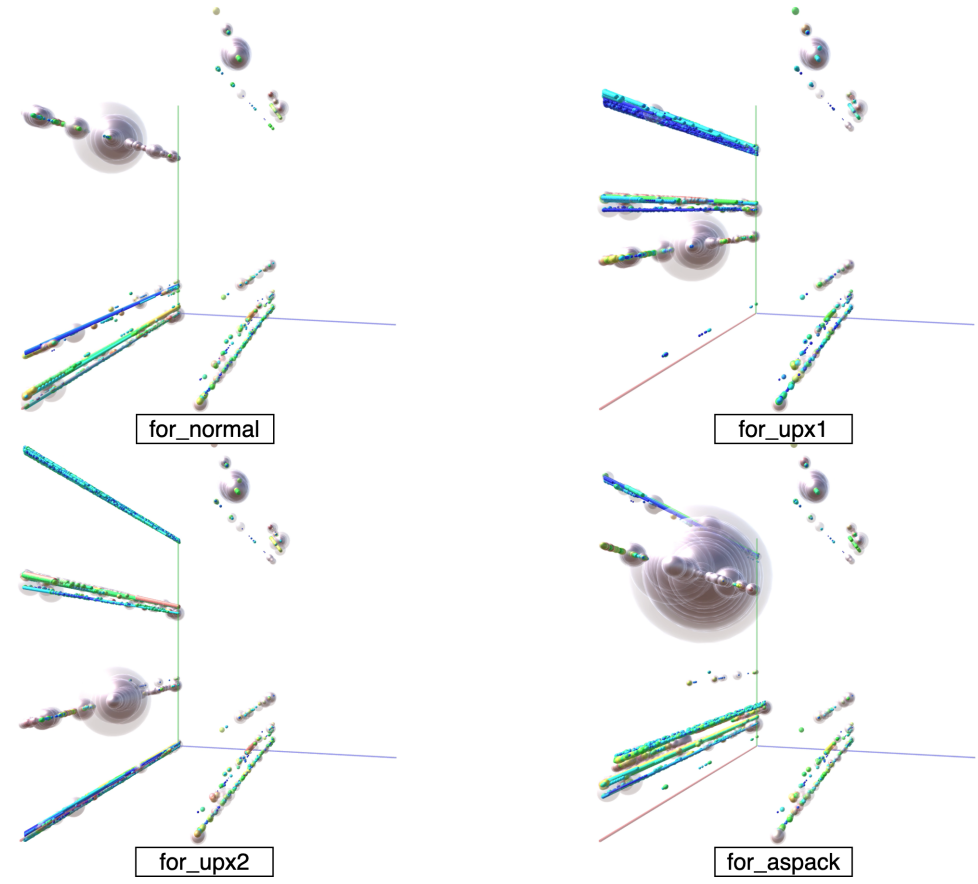

図 8　for_normal, for_upx1, for_upx2, for_aspack の可視化例
Fig. 8　Visualization example of for_normal, for_upx1, for_upx2, and for_aspack

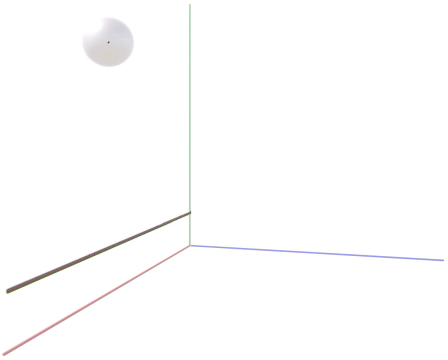

図 9　for_normal のオリジナル処理部分の可視化例
Fig. 9　Visualization example of the original processing part of for_normal

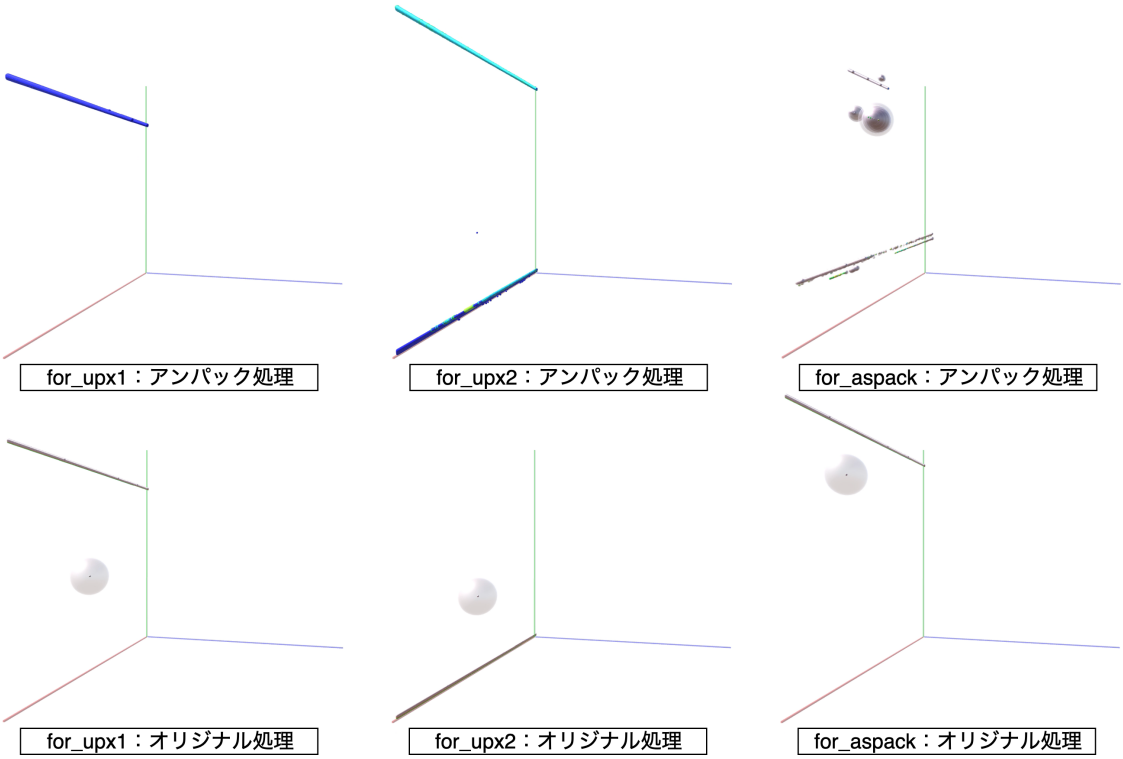

図 10　for_upx1, for_upx2, for_aspack の各処理の可視化例
Fig. 10　Visualization example of each processing of for_upx1, for_upx2, and for_aspack

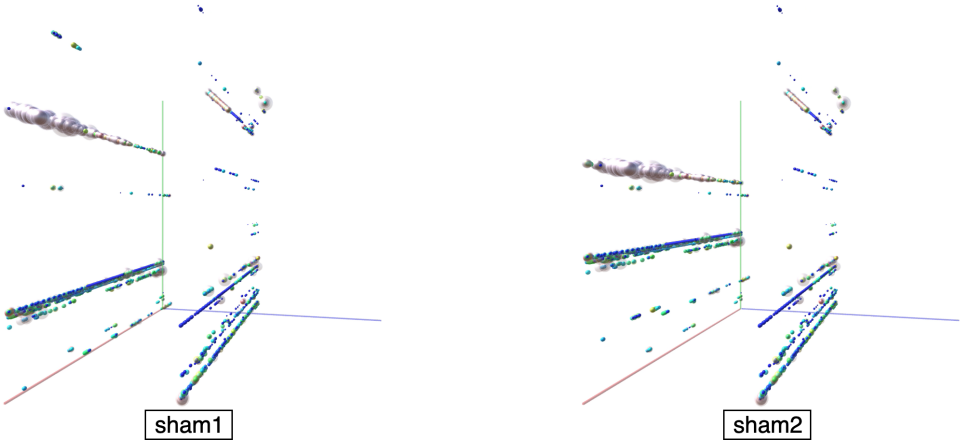

図 11　sham1, sham2 の可視化例
Fig. 11　Visualization example of sham1 and sham2

表 1　評価に用いたプログラムの情報
Table 1　Information of evaluated programs

プログラム名	パッカー	パッカーのバージョン	ハッシュ値 (MD5)
for_normal	-	-	770a6db3bc43d1ac9594c7fbfb488c76
for_upx1	UPX	3.91	bbceab2c267bcbe03fd31c3d5446a535
for_upx2	UPX	3.95w	8536daa095bd331112d2849265644b7f
for_aspack	ASPack	2.43	30f1908e37f8d4415a0a90d1b99c8e3a

表 2　評価に用いた PUA の情報
Table 2　Information of evaluated PUAs

PUA 名	マルウェア名 (Symantec[5])	ハッシュ値 (MD5)
sham1	SMG.Heur!gen	f66636c8bf793f678fe25e4f1a5f806c
sham2	SMG.Heur!gen	755f60c05ebab028ed9d5be80eaac21e

動していた．これは，動的に割り当てられたメモリ空間が異なることが原因であると考えられる．メモリ空間が平行移動する可能性を考慮すると，for_normal と for_upx1，for_upx2，for_aspack の可視化結果は多くの酷似点があると言える．反対に，酷似していない部分もいくつか存在していることがわかる．また，図 9 と図 10 のオリジナル処理候補の可視化結果から，パッキングされたオリジナルのプログラムの処理部を可視化できることがわかった．for_upx1，for_upx2，for_aspack のオリジナル処理のメモリ空間は図 8 において酷似していた部分と一致することも確認できる．図 10 のアンパック処理候補の可視化結果から，パッカーぞれぞれのアンパック処理の候補となる可視化が得られ，図 8 において酷似していなかった部分と一致している．これらのことから，本提案手法で，プログラムの処理の特徴を視覚的に認識でき，パッキングされたプログラムにおいてもオリジナルのプログラムの処理の特徴を視覚的に認識できる可能性があると言える．

また，図 11 の sham1 と sham2 の可視化結果より，ハッシュ値違いの PUA に対しても同じ特徴を得られたので，マルウェアに対しても用いることができる可能性があると考えられる．

6. 関連研究

既に，プログラムやマルウェアの動作を可視化して，プログラムを分類，検知する手法が複数提案されている．

関野らは，プログラムカウンタに着目し，その履歴を画像化することで CNN を用いた深層学習を可能にし動的な検知を可能とする手法の検証を行なっている [6]．この手法によるマルウェアの検知率は比較的低く，可視化段階で得られる画像は視覚的に認識できる可能性が低いこと，プログラムカウンタを用いていることの 2 つが要因であると考えられる．それに対し，提案手法による可視化はこの手法に対して認知できる可能性が高く，プログラムカウンタに加えロードストア命令を含んでいるメモリアクセスを用いるので，さらなる効果が見込める．

塩谷らは，メモリアクセスに着目し，マルウェアの自己書き換え動作を可視化する手法の検証を行なっている [7]．この手法は，メモリ空間を 2 次元的に再現し，使用しないアドレスを省いていることから，プログラムがアクセスする膨大な空間を可視化した際に認知できなくなる可能性が高い．それに対し，提案手法では，メモリ空間を 3 次元的に全て再現するため，比較的認知できなくなる可能性が低いと考えられる．

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なっている [8]．Nataraj らは，バイナリを画像化することで複数のマルウェアのファミリーの分類を行う手法を検証している [9]．大坪らは，機械学習による直感バイナリ分類・可視化（目 grep）の手法の検証を行なっている [10]．これらの手法は，いずれも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．それに対し，提案手法ではメモリアクセスログを用いるため，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．

7. まとめ

画像処理ベースの深層学習によるマルウェアの検知を行うため，その前段階として，プログラムの動作を可視化する新しい手法を提案した．本稿では，可視化についての検討を行った．考案した可視化方法を用いてプログラムを可視化し，その有効性の評価を行なった．今後は，実際に本提案手法を用いた深層学習による検知を行うことを想定し，各種パラメーターの調整や，自動調整機能などの検討を行う必要がある．

謝辞　本研究の一部は，JSPS 科研費 17K00076，19K11968，19H04108 の支援により行なった．

参考文献

[1] ディープラーニング これだけは知っておきたい 3 つのこと, https://jp.mathworks.com/discovery/deep-learning.html (2019.10.16)
[2] Malware Statistics & Trends Report | AV-TEST, https://www.av-test.org/en/statistics/malware/ (2019.10.10)
[3] Pin - A Dynamic Binary Instrumentation Tool, https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool (2019.10.15)
[4] VirusTotal, http://www.virustotal.com/ja/ (2019.10.16)
[5] Symantec, https://www.symantec.com/ (2019.10.16)
[6] 関野 翔太, 石川 亮太, 小林 良太郎, 加藤 雅彦. “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,” 情報処理学会研究報告, 2018-CSEC-83, pp. 1-6, 2018.
[7] 塩谷 正治, 森 博志, 吉岡 克成, 松本 勉. “マルウェアの自己書換え動作をメモリアクセスに着目して可視化する方法,” コンピュータセキュリティシンポジウム 2012, pp. 720-727, 2012.
[8] 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, p.205, 2016.
[9] L. Nataraj, S. Karthikeyan, G. Jacob, B. S. Manjunath. “Malware Images: Visualization and Automatic Classification,” In Proceedings of the 8th International Symposium on Visualization for Cyber Security (VizSec ’11), Article No. 4, pp. 1-7, 2011.
[10] 大坪 雄平, 三村 守, 榊 剛史, 後藤 厚宏. “機械学習による直感バイナリ分類・可視化 (目 grep) の実装,” コンピュータセキュリティシンポジウム 2017, pp. 1217-1224, 2017.