Deep Neural Network多段化による プロセスの挙動に着目したマルウェア推定手法

飛山 駿[1]　山口 由紀子[2]　嶋田 創[2]　秋山 満昭[3]　八木 毅[3]

概要：近年では，標的型攻撃など高度なサイバー攻撃が深刻な問題となっている．このような攻撃では既存の対策による検知が難しい未知のマルウェアが使用されることが多く，感染を完全に防ぐことは難しくなっている．そのため，感染を前提とした対策が必要とされている．本研究では，感染した端末を早期に発見するための手法として，端末上で実行されているプロセスからマルウェアプロセスを推定する手法を提案する．Deep Neural Network を多段に用いることでプロセスから得られた API コール列の特徴抽出/分類を行い，マルウェアプロセスか否かを推定する．分類結果から ROC 曲線と AUC を算出することで提案手法の性能を評価する．

キーワード：マルウェア，感染検知，機械学習

Malware process estimation method by multistage Deep Neural Networks based on process behavior

Shun Tobiyama[1]　Yukiko Yamaguchi[2]　Hajime Shimada[2]　Mitsuaki Akiyama[3]　Takeshi Yagi[3]

Abstract: Increasing malware and advanced cyber-attacks are now becoming a serious problem. An unknown malware is often used in these attacks, and now it's becoming difficult to protect terminals from their infection perfectly. Therefore, it is widely expected to a countermeasure for after infection is required. In this study, we propose the malware process estimation method based on process behavior in order to detect infected terminals. In proposal, we extract features from an API call sequence of a process and classify the process by using Deep Neural Networks in multiple stages. We evaluate the performance of our proposal by calculating ROC curve and Area Under the Curves.

Keywords: malware, infection detection, machine learning

1. はじめに

今日ではインターネットは我々にとって必要不可欠なものとなっている一方で，マルウェアによるサイバー攻撃も深刻な問題となっている．近年では発見されるマルウェアの数は増加の一途を辿っており，サイバー攻撃の手法も高度化，巧妙化している [1]．このような高度な攻撃では，アンチウイルスソフトによる検知を回避するため，しばしば未知のマルウェアが使用される．さらに，シグネチャマッチングを回避するため，実行ごとに動的にコードを変更するようなマルウェアも出現している．このように，現在では攻撃から端末を完全に守り切ることは難しくなっており，侵入を防ぐための対策だけでなく攻撃者の侵入後の対策など多層的な対策が必要とされている．

侵入後の対策手法として，トラフィックデータを利用したマルウェア感染検知手法が研究されている．しかし，近年では正常なトラフィックを模した通信を行うマルウェアが出現している．また，正常通信の量や種類も増加して

[1] 名古屋大学 大学院 情報科学研究科
Graduate school of Information Science, Nagoya University
[2] 名古屋大学 情報基盤センター
Information Technology Center, Nagoya University
[3] NTT セキュアプラットフォーム研究所
NTT Secure Platform Laboratories

いる一方で，継続的な情報窃取を目的として長期間潜伏する，外部との通信頻度の低いマルウェアなども出現している．そのため，膨大な量のトラフィックデータから悪性通信を漏れなく発見することは難しくなってきており，トラフィックによる検知のみでは不十分であると考えられる．

そこで本研究では，Windows 端末上で実行されているプロセスの挙動から悪性プロセスを推定する手法を提案する．本研究の目的は，従来のシグネチャベースの手法で検知しきれないマルウェアや，トラフィックによる検知から漏れて感染してしまった端末を早期に検出することである．提案手法では，異なる種類の Deep Neural Network（DNN）を多段に用いることにより，プロセスの挙動に含まれる特徴を効果的に抽出し，分類する．本手法は大きく学習フェーズと推定フェーズに分かれており，学習フェーズは 4 段階から構成される．1 段階目では，API コール列という形で記録されたプロセスの挙動のログファイルから入力ファイルを作成する．2 段階目では，DNN の 1 種である Recurrent Neural Network（RNN）を使用して API コール列の特徴を学習する．3 段階目では，学習した RNN を使用して入力ファイルから特徴を抽出し，特徴画像を生成する．4 段階目では，Convolutional Neural Network（CNN）を使用して特徴画像の分類学習を行う．推定フェーズでは，学習フェーズで学習した RNN と CNN を用いてプロセスのログからそのプロセスの悪性度を算出することにより悪性プロセスの推定を行う．

2. 関連研究

現在まで，マルウェアを検知するための様々な手法が研究されているが，近年では機械学習や人工知能などの技術を用いたヒューリスティック検知が注目されている．

ヒューリスティック検知で特徴として使用されるものの 1 つに API コールがある．ほとんどのプログラムは OS とのやり取りのために API コールを呼び出すため，プログラムの API コール呼び出しを監視することでそのプログラムの挙動を把握することができる．API コールを用いたマルウェア検知手法として，青木らは検知率を向上させるための API コールの抽出方法を提案している [2]．この手法では，サンドボックス上でマルウェアを動的解析した際に記録された API コール列から N-gram 特徴量を抽出し，決定木を用いて判別する．そして，N の値を変化させたときの検知率の変化を比較している．また，Ahmed らはプログラムの呼び出した API コール列の時系列特徴，および各 API コールの引数の統計量からマルウェアを検知する手法を提案している [3]．この手法では，特徴抽出にマルコフ連鎖モデルを利用し，抽出した特徴を決定木，ナイーブベイズ，サポートベクターマシンなどの機械学習手法を用いて分類することでマルウェアを検出している．

近年，様々な分野で DNN を用いた手法が大きな成果を挙げ話題となっている．現在でも活発に研究が行われており，RNN やその発展形である Long Short-Term Memory（LSTM）Network[4]，CNN など様々な構造の Neural Network が提案されている．

DNN を利用したマルウェア検知手法には次のようなものがある．Saxe らは 2 層の隠れ層を持つ DNN を用いたマルウェア検出手法を提案している [5]．この手法では，バイナリ実行ファイルに含まれる特徴から特徴量を作成し，DNN を用いてマルウェアを検出する．特徴としては，インポートされた DLL の種類や実行ファイルのメタデータなどの 4 種類の特徴が用いられている．抽出した特徴を 2 層の隠れ層を持つ DNN で学習し，得られた出力を用いてバイナリがマルウェアである確率を求める確率密度関数を推定する．この関数を用いてバイナリがマルウェアである確率を計算している．Pascanu らは RNN を用いたマルウェア検知手法を提案している [6]．この手法では API コール列を特徴として使用しており，RNN を用いて API コール列の言語モデルを作成することにより API コールから特徴を抽出している．抽出した特徴を Multi-Layer Perceptron により分類し，マルウェアを判定している．

3. 提案手法

3.1 概要

プロセスの挙動は様々なタスクの組み合わせからなり，それぞれのタスクは複数の動作から構成される．プロセスの挙動を記録する方法は様々なものがあるが，本手法ではプロセスが発行した API コール列をプロセスの挙動として扱い，対象端末上でプロセスが一定期間中に実行した API コールを記録したものを使用する．

記録された API コール列をそのプロセスの挙動としたとき，それぞれのタスクは API コール列の部分集合，個々の動作は 1 つの API コールとして見ることができる．このような構造から，API コール列の出現順序や個々の API コールの出現頻度の情報はプロセスの挙動の特徴量として利用できると考えられる．そこで，RNN を用いて API コール列からこれらの特徴を考慮した特徴抽出を行う．RNN はある時刻の出力が以前の入力に依存するという特徴を持ち，言語処理や音声処理など時系列データの認識でよい成果を出している．そのため，RNN を用いて API コール列から特徴を抽出することで，抽出された特徴列には API コールの出現頻度だけでなく，API コール列の時系列情報，つまり各 API コールの出現順序の情報が含まれると考えた．

RNN により抽出された特徴列は，各タスクの特徴を表す部分列の集合であると考えられる．しかし，常に特徴列の同一部分に良性/悪性プロセス特有のタスクの特徴が現れるとは限らない．そこで提案手法では，まず RNN により抽出された特徴列を画像化する．そして画像化した特徴を CNN を用いて分類する．CNN は画像認識分野で大きな成

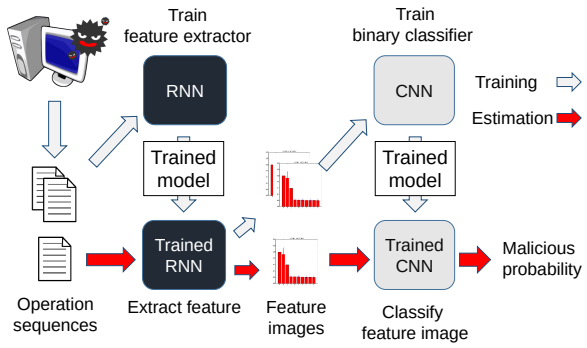

図 1　提案手法の概要
Fig. 1　Overview of proposed method.

果を出しており，特徴の出現位置のずれにも対応できるという特徴を持つ．そのため，CNN を用いて特徴画像を分類することで，良性/悪性プロセス特有の特徴をその出現位置に関わらず認識して分類することができると考えた．

このように，本手法ではプロセスの分類を 2 段階に分割し，それぞれの分類に適した DNN を使用してプロセスを分類する．提案手法の概要を**図 1** に示す．提案手法は，大きく学習フェーズと推定フェーズの 2 フェーズに分けられ，学習フェーズは以下の 4 段階から構成される．

(1) 入力ファイルの作成
(2) 特徴抽出器の学習
(3) 特徴の画像化
(4) 特徴画像分類器の学習

学習フェーズでは，まず端末上で実行されていた全プロセスの API コールを一定時間記録したファイルから，各プロセスの API コール列が記録された入力ファイルを作成する．次に，入力ファイルを用いて RNN を学習させ，API コール列の特徴抽出器を作成する．その後，学習した RNN を用いて API コール列から抽出した特徴列を画像化する．そして，画像化した特徴列を用いて CNN を学習し，特徴画像の分類器を作成する．推定フェーズでは，プロセスの API コール列が記録されたファイルを学習済みの RNN に入力し，特徴列を得る．その後，特徴列を画像化し，学習済みの CNN に入力して分類する．そして，分類結果の出力からそのプロセスの悪性度を計算する．

3.2　学習フェーズ

3.2.1　入力ファイルの作成

本手法では，対象の Windows 端末上で実行される Process Monitor [*1] によって記録されたログから入力ファイルを作成し，学習および推定に使用する．Process Monitor は，プロセスが行ったファイルシステムやレジストリ，スレッド，ネットワークとの通信などの処理をリアルタイムに表示および記録が可能なツールである．処理ごとに記録される情報を**表 1** に示す．

[*1] https://technet.microsoft.com/ja-jp/sysinternals/processmonitor.aspx

表 1　Process Monitor により記録される情報
Table 1　Logged Information by Process Monitor.

項目	内容
Time	処理が実行された時刻
Process Name	処理を実行したプロセス
PID	処理を実行したプロセスの PID
Event	処理の名前（`ReadFile`，`RegSetValue` など）
Path	処理が実行されたパス
Result	処理の結果（`SUCCESS`，`ACCESS DENIED` など）
Detail	処理についての詳細情報（引数の一部など）

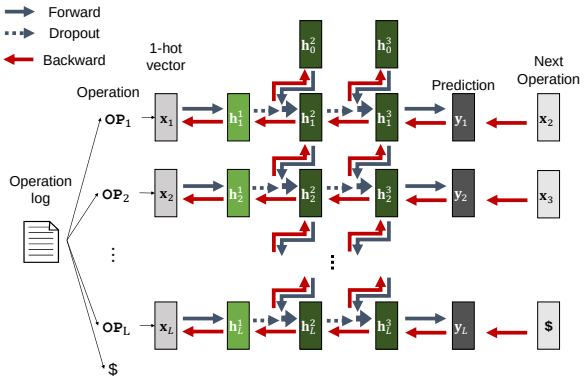

図 2　RNN の構造と学習の流れ
Fig. 2　Flow of RNN training.

提案手法ではプロセスごとにその悪性度を判定する．また，表 1 に示される各処理で記録される内容のうち，Event と Result の値のみを利用する（以降では処理の Event と Result の値の組を Operation とする）．そのため，記録されたログから同プロセスかつ同 PID の処理を時系列順に抜き出して各処理の Operation のみを時系列順に並べた，プロセスごとのログファイルを作成する．

3.2.2　特徴抽出器の学習

この段階では，RNN を用いて Operation 列から特徴を抽出する特徴抽出器を学習する．できる限り長期間に渡る Operation 間の特徴を抽出するため，提案手法では LSTM ユニットを用いた RNN を使用する．学習に使用する RNN は入力層 $\mathbf{x}$，通常の隠れ層 $\mathbf{h}^1$ と LSTM ユニットを使用した隠れ層 $\mathbf{h}^2$，$\mathbf{h}^3$，出力層 $\mathbf{y}$ を持つ．また，時系列方向でない接続では Dropout を使用する．RNN の構造を**図 2** に示す．図 2 に示すように，本手法では 3.2.1 節の方法で作成された，プロセスごとのログファイルに記録されている Operation 列を 1-hot ベクトル列に変換したものを入力として使用する．変換は次のように行われる．

(1) 各 Operation に一意の ID を割り当てる
(2) 各 Operation を ID 番目の位置以外を 0 で埋めた 1-hot ベクトルに変換する

特徴抽出器の学習は次のように行われる．まず，Operation 列 $= \{OP_1, OP_2, ..., OP_L\}$ を 1-hot ベクトル列 $= \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_L\}$ に変換する．その後変換した 1-hot ベ

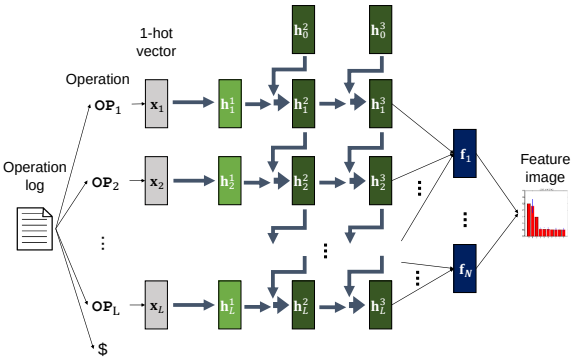

図 3 特徴の抽出と画像化の流れ

Fig. 3 Flow of feature extraction.

クトル $\mathbf{x}_t$ を RNN に順に入力し，次の入力の予測 $\mathbf{y}_t$ を得る．そして得られた予測 $\mathbf{y}_t$ と実際の次の入力 $\mathbf{x}_{t+1}$ を比較し，損失関数を計算する．T 個の Operation を入力後，誤差逆伝搬により RNN の重みを更新する．本手法では T=35 とした．このような学習により入力された Operation 列から次の Operation を予測するモデルを得る．

学習フェーズで使用するファイルに出現しない Operation が推定フェーズに出現する可能性がある．このような場合に対応するため，本手法では学習時に使用するファイルで出現する Operation の一部を次のように匿名化する．まず，あるファイルにおいて出現回数が 10 回以下である Operation をランダムに選択する．次に，ファイル中で出現するその Operation をすべて未知の Operation を表す `UnknownCall` に置き換える．この匿名化を入力ファイルごとに行うことで `UnknownCall` の特徴を学習させる．

全ての入力ファイルを 1 回 RNN に学習させることを 1-epoch と呼ぶ．本手法ではファイルの入力順をランダムに入れ替えて事前に定めた epoch だけ学習を行い，学習後の RNN を特徴抽出器とする．epoch 回数は 3 回とした．

3.2.3 特徴の画像化

この段階では，3.2.2 節で学習した RNN を用いてファイルに記録された Operation 列から特徴を抽出し，特徴画像を生成する．3.2.2 節で学習した RNN は，ある Operation を入力した際それまでの入力を基に次の Operation の予測を出力する．このことから RNN の隠れ層には，それまでの Operation の出現順序などの過去の入力情報が保存されていることが推測される．また，DNN では抽象的な特徴はより深い層に保存されると考えられている．したがって RNN の深い層には，過去の入力情報が圧縮された形で保存されていると推測できる．そこで本手法では，ある Operation を入力した時の RNN の 3 層目の隠れ層 $\mathbf{h}^3$ をその Operation の特徴として抽出する．

特徴画像生成の流れを**図 3** に示す．まず，学習済みの RNN に長さ L の Operation 列 O_L を入力し，各入力における RNN の 3 層目の隠れ層 $\mathbf{h}^3$ を特徴ベクトル列 $H = \{\mathbf{h}^3_1, \mathbf{h}^3_2, ..., \mathbf{h}^3_L\}$ として抽出する．ここで，L は Operation 列によって異なるが，本手法では 3.2.4 節で述べる画像分類器への入力は固定長である．そのため，H を固定長に変換する必要がある．変換後のベクトル列を F，F の長さを N とするとき，$F = \{\mathbf{f}_1, \mathbf{f}_2, ..., \mathbf{f}_N\}$ は H を N 個の部分特徴ベクトル列に分割し，各部分特徴ベクトル列を平均したベクトル列である．ただし，$L \geq N$ とし，Operation 列の長さが N 未満のものは CNN の学習用データ及び推定用データから除外する．平均ベクトル $\mathbf{f}_k$ の計算は，k 個目の部分特徴ベクトル列を $\{\mathbf{h}_{p_{k-1}+1}, ..., \mathbf{h}_{p_k}\}$ と表すとき式 2 のように表される．ただし，$p_0 = 0$ とする．

$$p_k = \lfloor \frac{L+k-1}{N} \rfloor + p_{k-1} \ (1 \leq k \leq N) \tag{1}$$

$$\mathbf{f}_k = \frac{1}{p_k - p_{k-1}} \sum_{j=p_{k-1}+1}^{p_k} \mathbf{h}^3_j \tag{2}$$

また，W を $\mathbf{h}^3$ の次元数としたとき，ベクトル列 F は次のように行列として表すことができる．

$$F = \begin{pmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_N \end{pmatrix} = \begin{pmatrix} f_{11} & f_{12} & \cdots & f_{1W} \\ f_{21} & f_{22} & \cdots & f_{2W} \\ \vdots & \vdots & \ddots & \vdots \\ f_{N1} & f_{N2} & \cdots & f_{NW} \end{pmatrix} \tag{3}$$

このようにして F を算出した後，F の各要素 f_{ij} を *sigmoid* 関数を用いて $[0, 1]$ 空間に写像し，255 倍した後に四捨五入して 0 から 255 までの整数とする．これにより，F は $N \times W$ ピクセルの 256 階調のモノクロ画像として表現できる．これを特徴画像として出力する．

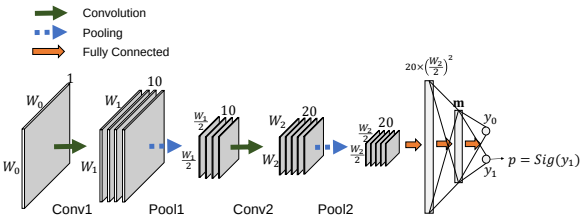

図 4 CNN の構造

Fig. 4 Structure of the CNN.

3.2.4 特徴画像分類器の学習

この段階では，CNN に良性/悪性プロセスの特徴画像を入力し学習させることで分類器を作成する．3.1 節で述べたように，特徴画像には様々なタスクの特徴が含まれていると考えられる．これらの特徴をうまく認識して分類するため，本手法では分類器として CNN を用いる．

使用する CNN の構造を**図 4** に示す．本手法では，畳込み層とプーリング層を 2 層ずつ持ち，全結合の隠れ層を 1 層持つ CNN を使用する．1 層目/2 層目の畳込み層ではそれぞれ 10/20 枚のフィルタを用いて畳み込みを行い，10/20 枚の特徴マップを出力する．また，プーリング層では最大値プーリングにより入力の 1/2 のサイズの画像を出力する．この分類器はプロセスが良性か悪性かを分類する

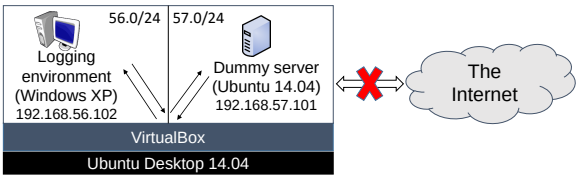

図 5 ログ記録環境
Fig. 5 Logging environment.

ため，出力は 2 次元のベクトルとなる．ベクトルの各要素はそれぞれプロセスの良性度合い，悪性度合いを表す．

学習は次のように行う．まず，3.2.3 節で作成した特徴画像を入力し，プロセスを良性/悪性に分類する．その後，分類結果と実際の分類を比較し，損失関数を計算する．ミニバッチサイズで指定された枚数の画像に関して損失関数を計算後，誤差逆伝搬により全結合層の重みと畳込み層のフィルタの重みを更新する．

3.3 推定フェーズ

推定フェーズでは，学習した RNN および CNN を用いて記録された Operation 列からプロセスの悪性度を算出する．まず，3.2.3 節のように学習済みの RNN を使用して入力ファイルから特徴画像を生成する．その後，生成した特徴画像を学習済みの CNN に入力し，分類結果を得る．分類結果として得られた 2 次元ベクトルのうち悪性度を表す要素 y_m を式 4 のように $Sigmoid$ 関数により $[0, 1]$ 空間に写像することにより，プロセスの悪性度 p を確率値として算出する．算出した悪性度 p と事前に設定した閾値を比較し，閾値以上であれば悪性，閾値未満であれば良性と推定する．

$$p = Sigmoid(y_m) = \frac{1}{1 + \exp(-y_m)} \qquad (4)$$

4. 実験

提案手法の性能を評価するため，仮想環境上に構築されたインターネットから隔離された環境に存在する WindowsXP 端末で動作するプロセスの挙動を記録し，記録したログを用いて実験を行った．

4.1 実験方法

4.1.1 ログ記録環境

ログの取得環境を**図 5** に示す．本実験では，まず仮想環境上で実行されているログ記録環境（WindowsXP）においてマルウェアのバイナリファイルおよび正常プログラムのバイナリファイルを実行し，その際に記録環境上で動作していた全プロセスの挙動を Process Monitor を用いて記録した．記録は 5 分間の記録を 5 分間隔で 10 回行った．仮想環境上には，HTTP，SMTP，DNS など主要なサービスを模擬するソフトウェアである INetSim[*2] を動作させたダミーサーバを用意し，記録環境からの通信は全てダミーサーバへ送信され，ダミー応答を返答するように設定した．

4.1.2 入力データ

4.1.1 節で述べた記録環境で正常バイナリ 32 個及びマルウェアバイナリ 132 個をそれぞれ実行し，プロセスの挙動を記録した．実験に使用したマルウェアバイナリは，NTT セキュアプラットフォーム研究所で 2014 年 4 月から 10 月の間に収集されたものであり，Kaspersky により 36 ファミリに分類される．得られたログを 3.2.1 節の方法でプロセス毎に分割した後，正常バイナリを実行した際に得られたログファイルについて以下の操作を行った．

- プロセス名と PID が重複するログファイルの削除
- 記録された Operation の長さが 400,000 を超えるログファイルの削除

また，マルウェアバイナリを実行した際に記録したログから，以下の条件のいずれかを満たすものを悪性プロセスとして抽出した．

（1）マルウェアバイナリと同名のプロセス
（2）（1）から生成されたプロセス
（3）（1），（2）からコードを注入されたプロセス

(2)，(3) は，Cuckoo Sandbox[*3] を使用してマルウェアバイナリを動的解析した際の結果を参考にして決定した．

上記の方法で記録した 306 個の良性プロセスおよび 238 個の悪性プロセスのログファイル（合計 544 個）を使用して提案手法の学習及び評価を行った．

4.1.3 学習方法

本実験では，5 分割交差検証により約 400 個のログファイルを用いて RNN 及び CNN の学習を行い，学習で使用しなかったログを用いて評価を行った．RNN の学習に使用したログで出現した Operation の種類は平均で 97.8 種類であった．

RNN 及び CNN のパラメータを変化させた時の分類精度を比較するため，RNN の隠れ層 $\mathbf{h}^2$，$\mathbf{h}^3$ の次元，CNN の入力画像サイズおよび全結合層の次元数を変更して学習と評価を行った．変化させたパラメータ及び推定に使用されるデータ数と固定パラメータを**表 2**，**表 3** に示す．表 2 には，画像化されたファイルの数も記載した．RNN の共通パラメータは LSTM を用いた RNN により言語モデルを学習する手法において使用されたもの [7] を参考に決定した．また，CNN のパラメータは 28×28 ピクセルの手書き文字認識で高い認識率を示した LeNet[*4] を参考に決定した．

4.1.4 評価基準

本実験では，ROC（Receiver Operating Characteristic）曲線下の面積である Area Under the Curve（AUC）の値を比較することにより提案手法を評価した．ROC 曲線とは，閾値を変更した時の真陽性率（TPR）と偽陽性率（FPR）

[*2] http://www.inetsim.org
[*3] https://www.cuckoosandbox.org/
[*4] http://deeplearning.net/tutorial/lenet.html

表 2　変化させたパラメータ
Table 2　Variable parameters.

RNN	CNN		画像化された
$\mathbf{h}^2$, $\mathbf{h}^3$ の次元	$N \times W$	全結合層 m の次元	ファイル数
14	14 × 14	28	491
17	17 × 17	34	491
20	20 × 20	40	491
25	25 × 25	50	491
30	30 × 30	60	491
35	35 × 35	70	482
40	40 × 40	80	482
50	50 × 50	100	469
70	70 × 70	140	466
100	100 × 100	200	456
150	150 × 150	300	444
250	250 × 250	500	426

表 3　固定パラメータ
Table 3　Fixed parameters.

RNN		
$\mathbf{h}^1$ の次元	epoch 回数	ミニバッチサイズ
10	3	20

CNN				
層	入力枚数	出力枚数	フィルタサイズ	ストライド
Conv1	1	10	5 × 5	1
Pool1	10	10	2 × 2	2
Conv2	10	20	5 × 5	1
Pool2	20	20	2 × 2	2
全層	epoch 回数		ミニバッチサイズ	
共通	50		20	

表 4　真陽性，偽陽性の関係
Table 4　Relation between TP and FP.

実際の	分類結果	
分類	悪性	良性
悪性（P）	真陽性（TP）	偽陰性（FN）
良性（N）	偽陽性（FP）	真陰性（TN）

の関係をグラフに表したものである．提案手法ではプロセスの悪性度を確率値で表し，値が事前に設定した閾値以上なら悪性，そうでない場合は良性と推定する．悪性である場合を陽性（Positive, P）とし，良性の場合を陰性（Negative, N）としたとき，真陽性（True Positive, TP）と偽陽性（False Positive, FP）の関係は**表 4** で表され，これを用いて TPR=TP/P，FPR=FP/N と表すことができる．評価に用いた ROC 曲線は，各交差検証時に得られたプロセスの悪性度から算出された ROC 曲線の平均とした．

4.2　実験結果

次元ごとの平均 ROC 曲線を**図 6** に示す．横軸は誤って悪性プロセスと判定されたプロセスの割合，縦軸は正しく悪性プロセスと判定できたものの割合である．また，各平均 ROC 曲線の AUC を凡例に表示した．図に示したように，画像サイズが 50 × 50 の時最も高い AUC である 0.95 が得られ，画像サイズが 50 × 50 より大きい場合も小さい場合も AUC が低下するという結果になった．また，どの画像サイズでも 0.85 以上の AUC が得られており，高精度に悪性プロセスを推定することができているといえる．

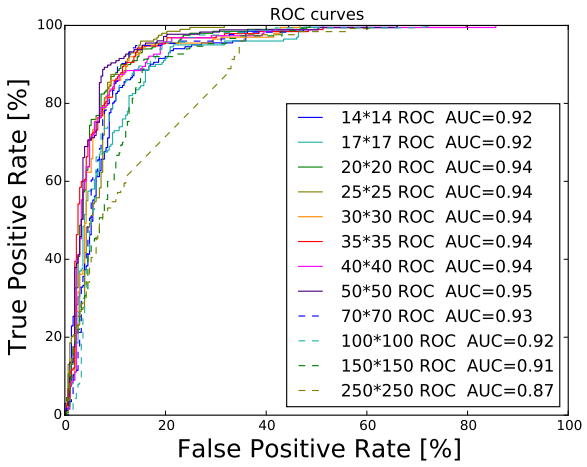

図 6　平均 ROC 曲線と AUC
Fig. 6　Average ROC curves and AUC.

5.　考察

実験の結果から，高い精度で悪性プロセスを推定することができることが示された．本節では，まず実験で使用したデータについて，各プロセスの Operation 列の長さ及び出現する Operation の頻度の点から分析を行う．次に，分析で得られた結果から Operation 列の長さ及び出現頻度が推定精度に与える影響について考察する．

5.1　データの分析

5.1.1　Operation の長さの差異

1 節で述べたように，近年では長期的に情報を窃取するため，潜伏を行うマルウェアが出現している．このようなマルウェアでは，普段は最低限の動作しかしないため，一定期間に発行される API コールの数は少なくなると考えられる．また，攻撃者からの指示を受けて動作するようなマルウェアの場合でも，指示がないときに発行される API コールは良性プロセスに比べ少ないと考えられる．そのため，プロセスの発行する API コールを一定期間記録した場合，良性プロセスと悪性プロセスでは発行される Operation の数に差が生まれると考えられる．そこで，実験において使用したデータについて，それぞれの Operation 列の長さから**図 7** に示すヒストグラムを作成し，良性プロセスと悪性プロセスの Operation 列の長さの差異を調査した．グラフの縦軸は度数を表し，横軸は階級幅 100 における階級を表す．ただし，良性プロセスと悪性プロセスの数の差による影響を防ぐため，度数は正規化した．また，Operation 列の長さが 4500 以上のものは一番右にまとめた．図 7 より，

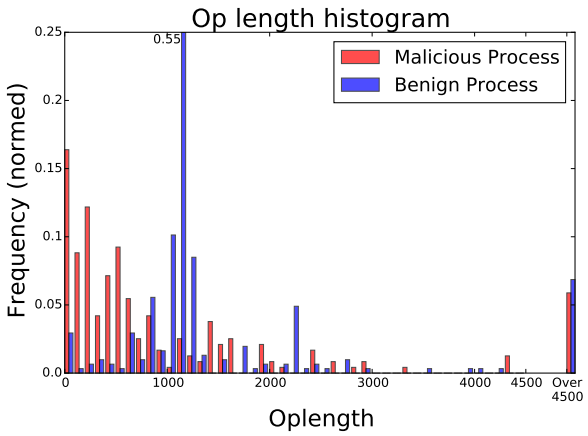

図 7 Operation 列の長さのヒストグラム
Fig. 7 Histogram of Operation length.

良性プロセスでは Operation 列の長さが 1100 から 1300 のものが突出して多く，悪性プロセスは 0 から 500 までのものが 5 割以上を占めていることが判明した．

5.1.2 Operation の出現頻度の差異

マルウェアは解析環境検知を目的とした実行中のプロセス名の読み取りやレジストリ情報の列挙など，良性プロセスでは行われないような不審な挙動を示すことがある．そのため，良性プロセスと悪性プロセスでは，Operation の出現頻度に差が表れると考えられる．そこで，それぞれの Operation の出現頻度に良性プロセスと悪性プロセスの間で差があるかを調査した．全ての Operation の出現頻度のうち，良性プロセス，悪性プロセス，誤推定したプロセスの出現頻度の合計が大きい 20 個の Operation の頻度分布を**図 8** に示す．横軸は Operation の種類を表し，縦軸は出現頻度を表す．ただし，良性プロセスと悪性プロセスの総 Operation 数の差による影響を防ぐため，出現頻度は正規化した．青の棒グラフが良性プロセスにおける Operation の出現頻度，赤の棒グラフが悪性プロセスにおける出現頻度，緑の棒グラフは誤推定したプロセスにおける出現頻度を表す．図 8 より，良性プロセスと悪性プロセスで出現頻度が大きく異なるものが存在することが読み取れる．悪性プロセスで出現頻度が高く，良性プロセスでは低い Operation の例として，`RegEnumValue:SUCCESS` が挙げられる．`RegEnumValue` はレジストリの情報を列挙する API コールである．この Operation の出現頻度が悪性プロセスでのみ高いのは，マルウェアが端末の環境情報を収集するために使用したからであると考えられる．

5.2 誤推定の原因分析

5.1 節より，良性プロセスと悪性プロセスでは Operation 列の長さ，及び出現する Operation の頻度に差異が存在することが判明した．そこで，Operation 列の長さごとに ROC 曲線及び AUC を算出し，Operation 列の長さの推定精度への影響を考察する．また，実験時に誤推定された悪性プロセスについてそれらの Operation の出現頻度を調査し，出現する Operation が推定精度に与える影響を考察する．ただし，誤推定された悪性プロセスとは，推定の結果悪性度が 0.50 未満となった悪性プロセスを指す．

5.2.1 Operation の長さの影響

Operation 列の長さが短いプロセスでは，長さが長いプロセスに比べ特徴抽出時に得られる情報量は少なくなる．そのため，長さの短いプロセスの推定精度は長さが長いプロセスに比べ低下すると考えられる．そこで，4 節の評価結果のデータから，Operation 列の長さが指定の範囲内であるもののみを抽出して ROC 曲線及び AUC を比較し，Operation の長さが推定精度に与える影響を考察した．Operation 列の長さが 0 から 1000 のデータ，1000 から 2000 のデータの ROC 曲線及び AUC を**図 9**，**図 10** に示す．画像サイズが 50×50 の時に抽出されたデータ数を図の上部に示した．図に示したように，長さが 1000 から 2000 のプロセスの ROC 曲線では FPR が 10%になるまでに TPR が 100%となるものも存在するが，長さが 0-1000 までのプロセスの ROC 曲線では FPR が 10%の時の TPR は最大で 80%であり，Operation 列の長さが短いプロセスの方が同 FPR 値における悪性プロセスの誤推定率が高いという結果になった．したがって，Operation 列の長さが推定精度に影響を及ぼしているといえる．

5.2.2 Operation の出現頻度の影響

実験において誤って良性と推定された悪性プロセスでは，Operation の出現頻度に差があると考えられる．そこで，誤推定されたプロセスで出現する Operation の頻度の分布を調査し，良性/悪性プロセスの出現頻度の分布と比較することにより Operation の出現頻度が推定精度に与える影響を考察した．誤推定されたプロセスの Operation の出現頻度の分布を図 7 の緑の棒グラフに示す．図に示したように，誤推定されたプロセスでは，`RegEnumValue:SUCCESS` の出現頻度が悪性プロセスの頻度分布に比べ非常に低く，良性/悪性プロセスの両方で出現頻度が低い `WriteFile:SUCCESS` の出現頻度が非常に高いという結果が得られた．一方で，`RegOpenKey:SUCCESS` などのように，良性/悪性/誤推定されたプロセスのどの場合においても頻度があまり変わらない Operation も存在した．このことから，`RegEnumKey:SUCCESS` のように，その Operation の出現が推定結果に大きな影響を及ぼすものと，`RegOpenKey:SUCCESS` のようにそれだけでは推定結果に影響を及ぼさないものが存在することが推測される．

6. おわりに

本研究では，感染端末の早期検知のため，端末で実行されているプロセスの挙動から悪性プロセスを推定する手法を提案した．提案手法では，異なる種類の DNN を多段に用いることにより，プロセスの挙動に含まれる特徴を抽

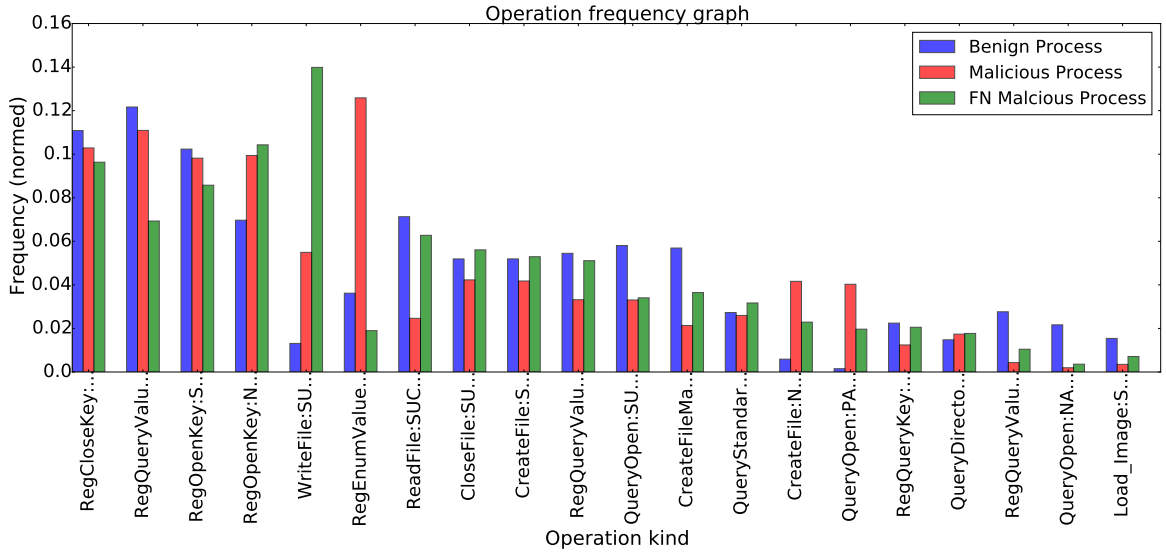

図 8 Operation の出現頻度の分布（上位 20 個）

Fig. 8 Graph of Operation frequency.

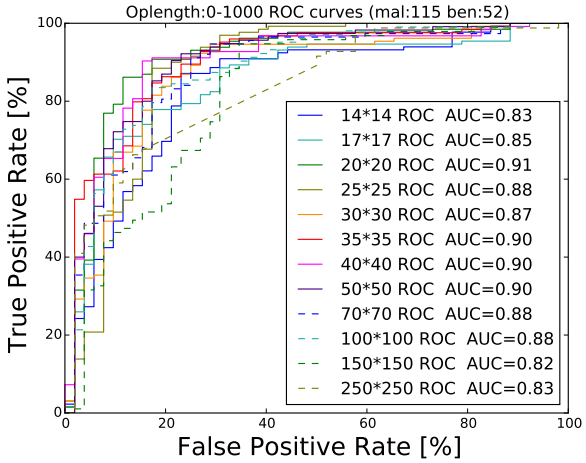

図 9 Operation 列の長さが 0-1000 のプロセスの ROC 曲線

Fig. 9 ROC curves of 0-1000 Operation length.

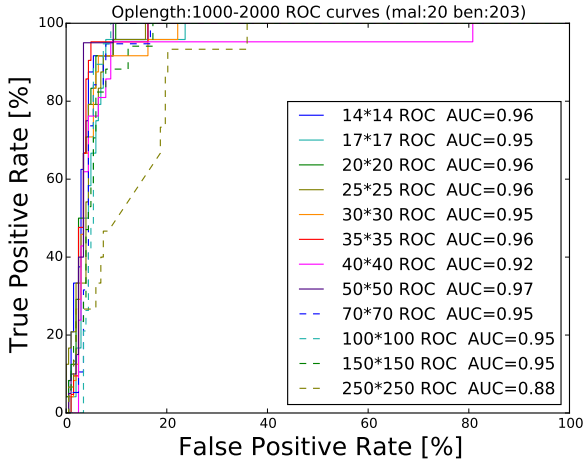

図 10 Operation 列の長さが 1000-2000 のプロセスの ROC 曲線

Fig. 10 ROC curves of 1000-2000 Operation length.

出して分類する．また，提案手法の評価のため，隔離環境上で記録された 544 個のプロセスログを使用して 5 分割交差検証を行い，平均 AUC を求めることで提案手法の性能を評価した．また，RNN および CNN のパラメータを変更した際の AUC の変化を調査した．さらに，実験で使用したデータについて分析を行い，良性/悪性プロセスで Operation 列の長さ，出現頻度に差異が存在することを示した．また，実験において誤推定されたプロセスを分析し，Operation 列の長さや出現頻度が推定精度に影響を及ぼす可能性があることを示した．

本研究では RNN 及び CNN で学習する特徴として，Process Monitor で記録された内容のうち Event 名と Result のみを使用した．しかし，処理が実行した際のパスや引数の情報なども不審なプロセスを検出する際には有効であると考えられる．今後の課題としては，推定で使用する特徴量の追加を行うことが挙げられる．

参考文献

[1] McAfee Labs.: McAfee 脅威レポート 2015 年第 2 四半期 (online), 入手先 ⟨http://www.mcafee.com/jp/resources/reports/rp-quarterly-threat-q2-2015.pdf⟩ (2016.02.10).

[2] 青木 一樹，後藤 滋樹: マルウェア検知のための API コールパターンの分析，電子情報通信学会総合大会講演論文集 D-19-3 (2014).

[3] Ahmed, F., et al.: Using spatio-temporal information in api calls with machine learning algorithms for malware detection, AISec'09, pp. 55-62 (2009).

[4] Gers, F., et al.: Learning to forget: Continual prediction with LSTM, Neural computation, Vol. 12, No. 10, pp. 2451-2471 (2000).

[5] Saxe, J., et al.: Deep Neural Network Based Malware Detection Using Two Dimensional Binary Program Features, MALWARE2015, pp. 11-20 (2015).

[6] Pascanu, R., et al.: Malware classification with recurrent networks, ICASSP2015, pp. 1916-1920 (2015).

[7] Zaremba, W., et al.: Recurrent neural network regularization, ICLR2015 (2015).