Computer Security Symposium 2017
23 - 25 October 2017

Seq2Seqモデルを用いたプロセスの悪性度推定手法

飛山 駿[1]　山口 由紀子[1]　長谷川 皓一[1]　嶋田 創[1]　秋山 満昭[2]　八木 毅[2]

概要：近年の巧妙なサイバー攻撃では，実行後のマルウェアバイナリの削除や既存プロセスへのインジェクションなど，バイナリファイルに依存した検知では見つかりにくい攻撃も存在する．そこで，本研究では端末上に悪性なプロセスが存在するかを推定することにより感染を検出する手法を提案する．本手法では，Windows 上で動作するプロセスについて，Process Monitor で記録したプロセスの挙動を Seq2Seq モデルを用いて学習して特徴ベクトル列を抽出する．さらに，抽出された特徴ベクトル列を Seq2Seq モデルで分類することでプロセスの悪性度の推定を行う．交差検証により分類性能の評価を行った結果を報告する．

キーワード：マルウェア，感染検知，機械学習，Seq2Seq

Malicious Process Estimation Method Using Seq2Seq Model

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

Abstract: In recent sophisticated cyber-attacks, it is hard to detect malware with binary-dependent method because some malware utilize countermeasures such as post execution binary elimination or process injection. In this study, we propose an infection detection method by estimating maliciousness of existence processes in Windows machines. In proposal, we extract feature vector sequence from process behavior captured by process monitor with Seq2Seq model. Then, we estimate the maliciousness of the process by categorizing with an another Seq2Seq model. We evaluated the performance of our proposal by cross validation.

Keywords: Malware, Infection detection, Machine Learning, Seq2Seq

1. はじめに

インターネットの普及によりネット経由で利用可能なサービスが増え，我々の生活がより便利になる一方で，サイバー攻撃を始めとするネットを利用した犯罪による被害も深刻化している．近年のサイバー攻撃は金銭や機密情報の窃取を目的としたものが多く，特に，標的型攻撃のような巧妙な攻撃では，標的に気付かれないよう長期間侵入を続け，継続的に情報を窃取するなど，その攻撃手法も巧妙なものとなっている．標的型攻撃では，標的を攻撃する前に入念な調査を行い，標的型メールや水飲み場型攻撃などにより標的内部に侵入する．侵入段階では，悪性なマクロを含む文書ファイルや悪性スクリプトを実行させ，囮用ドロッパやトロイの木馬とともに本命の高機能マルウェアを感染端末にインストールするなど，複数のマルウェアが使用される．標的内部に侵入した攻撃者は，マルウェアだけでなく ifconfig, net などの OS 標準のコマンドやツールを用いて端末内の情報収集，認証情報の窃取，感染の拡大を試みる [1]．ここで利用される本命のマルウェアは，特にアンチウイルスソフトで検出できないよう細工され，保存場所にメモリやレジストリなどを利用してファイルとして存在しないもの，正規プロセスのメモリ領域に悪性コードを注入して動作することで自身の存在を隠匿するものなどが存在する．また，自身は最低限の機能しか持たず，コマンドにより C&C サーバからモジュールをダウンロードしメモリ上に展開することで機能を追加するマルウェアも出現

[1] 名古屋大学
Nagoya University
[2] NTT セキュアプラットフォーム研究所
NTT Secure Platform Laboratories

している [2]．このようなマルウェアをバイナリファイルに依存したシグネチャベースの既存の対策で完全に防ぐことは困難になっているとともに，現在では侵入の防止だけではなく侵入された後の対策も重要となっている．

そこで本研究では，ファイルではなくプロセスの振る舞いに着目し，深層学習を用いて悪性なプロセスを推定する手法を提案する．これにより，バイナリファイルに依存しない検知を実現すると同時に，侵入後のマルウェアの挙動の変化に追従した検知を実現できる．提案手法では，Seq2Seq モデルと呼ばれる Deep Neural Network（DNN）を用いてプロセスの挙動から時系列を考慮した特徴を学習する．そして，学習済みの Seq2Seq モデルを用いて特徴を抽出し，さらに別の Seq2Seq モデルを用いた分類器で分類して悪性度を算出することで悪性プロセスを推定する．

2. 関連研究

端末で動作中のプロセスに着目した悪性なプロセスの検出手法には次のようなものがある．中里らは，プロセスの出現頻度や，外部との通信状態から特徴量を抽出し，不審なプロセスを判定する手法を提案している [3]．この手法では，標的型攻撃では標的組織内部の端末で通常見られないプロセスが現れることと外部との通信を行うという特徴に着目し，プロセスの出現頻度，利用者数，外部通信の頻度などの特徴量からプロセスの不審度を算出し，不審プロセスを判定している．山本らは，正規プロセスに悪性コードを注入するマルウェアに着目し，感染が疑われるプロセスのメモリイメージと非感染環境の同プロセスのメモリイメージを比較して注入された悪性コードを検出することにより，悪性プロセスを特定する手法を提案している [4]．

近年，画像認識や言語処理など様々な分野で Deep Neural Network（DNN）を用いた手法が他手法を圧倒する成果を出し話題となっている．DNN のマルウェア検出への応用として次のような研究が存在する．Pascanu らは，Recurrent Neural Network（RNN）を用いて API コール列から特徴を抽出してマルウェア分類を行う手法を提案している [5]．この手法では，API コール列を言語として捉え言語モデルを構築することで分類に効果的な特徴の抽出を試みている．Wang らは，Seq2Seq モデルのマルチタスク学習によりマルウェア分類を行う手法を提案している [6]．この手法では，エンコーダにより API コール列を固定長の特徴ベクトルに圧縮し，圧縮されたベクトルから 2 つのデコーダを用いてマルウェアの分類結果とマルウェアによるファイルアクセスの要約を出力する．

我々は，RNN を用いてプロセスの挙動から特徴ベクトル列を抽出し，抽出した特徴ベクトル列を画像化して Convaolutional Neural Network（CNN）により分類することで悪性プロセスを推定する手法を提案してきた [7]．この手法は，プロセスの挙動に含まれる特徴を RNN を用いて可変長の特徴ベクトル列として抽出することで，挙動の時間的変化を含む特徴を抽出する．さらに，抽出した特徴ベクトル列を画像化して CNN を用いて分類することで，特徴ベクトル列に含まれる正常/悪性プロセス特有の挙動を認識して分類させるという，異なる DNN を多段に適用することで効果的に特徴を認識し，分類するものであった．しかしながら，この手法では抽出される可変長の特徴ベクトル列の長さは入力系列と等しい一方，画像化の際の画像サイズは一定である．そのため，入力系列が非常に長い場合には，画像化の際に特徴の情報が失われる可能性があるという問題があった．そこで，本研究では 2 つの Seq2Seq モデルを用いて入力系列を段階的に圧縮することで，特徴情報の損失を防ぎつつ分類を行うことを試みた．

3. 提案手法

3.1 概要

本手法では，プロセスが発行したイベント列をプロセスの挙動として扱い，イベント列に含まれる言語的な特徴を捉えて分類することによって悪性プロセスを推定する．

悪性プロセスと正常プロセスでは，その挙動には差異が表れることが推測される．これは，悪性プロセスの持つ機能と正常プロセスの持つ機能の違いのためである．悪性プロセスは，自身の存在隠匿や入力履歴の収集など，正常なプロセスにはあまり見られない機能を有する場合が多い．このような機能の違いにより，ファイルやレジストリの読み書き頻度やアクセス試行，プロセスの操作などの挙動に差異が表れることが予想される．そのため，イベント列から適切な特徴を抽出することにより悪性プロセスを推定することが可能であると考える．

また，イベント列は一種の文章のようなものとして扱うことが可能である．そのため，イベント列に言語処理における特徴抽出手法を適用することで，適切な特徴を抽出できると推測する．イベント列では，それぞれのイベントはある操作を表し，イベントの組み合わせによって連続した操作，つまり挙動を表すことができる．例えば，あるイベント列 $W = [\texttt{CreateFile}, \texttt{SUCCESS}, \texttt{WriteFile}, \texttt{SUCCESS}]$ があったとき，`CreateFile`，`WriteFile` はそれぞれファイルを開く，ファイルに書き込むという操作を表し，イベント列 W はファイルを作成して書き込みを行ったという挙動を表す．言語において，単語を文法に従って並べることで意味をなす文を構成することが可能であるのと同様に，プロセスにおいてもある目的に従って行われたイベントの列によって挙動を表すことが可能であると考える．

そこで本手法では，まず言語翻訳などで使用されている Seq2Seq モデルを用いてイベント列の言語的特徴を学習し，学習済みの Seq2Seq モデルを用いて特徴を抽出する．次に，別の Seq2Seq モデルを用いた分類器を用いて抽出した特徴を分類し，悪性度を算出することで悪性プロセスを

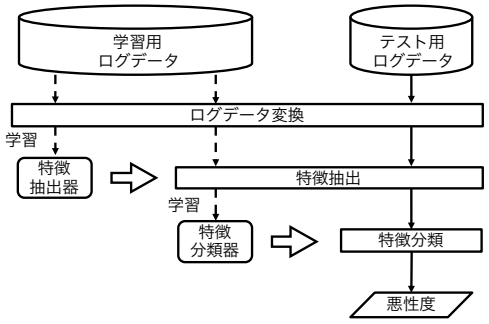

図 1 提案手法の概要
Fig. 1 Overview of proposed method.

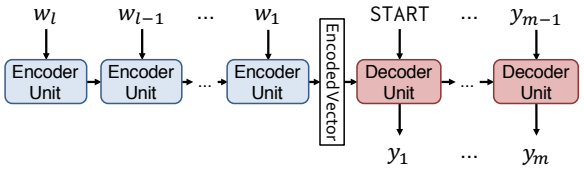

図 2 Seq2Seq モデルの構造
Fig. 2 Architecture of Seq2Seq.

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	操作が発生した時刻
Process Name	操作を発生させたプロセス
PID	操作を発生させたプロセスの PID
Event	操作名（`ReadFile`, `RegSetValue` など）
Path	操作が実行されたパス
Result	操作の結果（`SUCCESS`, `ACCESS DENIED` など）
Detail	操作についての詳細情報（引数の一部など）

推定する．提案手法の流れを**図 1** に示す．まず，ログデータをモデルに入力可能な形式へ変換する．その後，学習段階では学習用のログデータを用いて特徴抽出器，特徴分類器の順に学習を行う．この時，特徴抽出器は入力データ列と同じ系列を出力させるよう学習させる．特徴分類器では特徴ベクトル列の悪性度を 2 値分類により学習させる．つまり，悪性プロセスの悪性度は 1 に近く，正常プロセスの悪性度は 0 に近くなるよう学習させる．そして，推定段階では得られた学習済みの特徴抽出器と特徴分類器を用いてプロセスのログデータを分類することで，そのプロセスの悪性度を推定する．以降の節では，まず本手法で使用する Seq2Seq モデルについて説明した後，各段階について解説する．

3.2 Seq2Seq モデル

Seq2Seq モデルは Sutskever らによって提案された手法 [8] で，入力系列をエンコード（圧縮）するためのエンコーダとエンコードされた系列をデコード（解釈）するデコーダから構成される．Seq2Seq モデルの基本構造を**図 2** に示す．エンコーダ，デコーダはそれぞれ別の RNN で構成される．Seq2Seq モデルでは，長さ l の入力系列 $[w_1, w_2, ..., w_l]$ をエンコーダ RNN に逆順に入力していき，入力系列の情報を圧縮したベクトル（圧縮ベクトル）を得る．次に，得られた圧縮ベクトルをデコーダ RNN の隠れ層の初期値とし，開始記号（START）を入力することにより出力を開始する．そして，長さ m' の正解系列 $[v_1, v_2, ..., v_{m'}]$ に対する長さ m の予測系列 $[y_1, y_2, ..., y_m]$ を得る．このとき，m は l に依存せず，人手もしくは Seq2Seq モデル自身により決定される．Seq2Seq モデル自身が系列の長さを決定する場合，入力列に終端記号を追加して学習させることで出力終了を決定させる．この場合，m は m' にも依存しない．また，デコーダに入力される系列は学習時と予測時で異なる．学習時には，正解系列をデコーダへの入力として使用する．一方，予測時には，出力された予測をデコーダの次の入力として使用する．図 2 は，予測時の流れを示している．

3.3 ログデータの変換

本手法では，対象の Windows 端末上で実行される Process Monitor[9] によって記録されたログデータを用いて悪性度を推定する．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へのアクセス，スレッド作成，ネットワークとの通信などの操作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．各イベントごとに記録される情報を**表 1** に示す．本手法では，表 1 に示す情報のうち，Event および Result の情報のみを用いる．以降では Event および Result をイベントと表し，イベントを時系列順に並べたものをイベント列と表す．イベント列はプロセスごとに作成され，ベクトル列に変換された後特徴抽出器に入力される．ベクトル列は，各イベントを 1-hot ベクトルに変換することにより得る．ここで，推定時には学習時には現れなかったイベントが出現する可能性がある．そのため，そのようなイベントが出現した場合，ベクトル変換前に未知単語を表す `UnknownCall` に置換し，その後ベクトル変換を行う．ベクトル変換では，各イベントに一意な ID を割り当てた後，あるイベントに対し，そのイベントの ID 番目の要素を 1 とし，それ以外の要素を 0 で埋めた 1-hot ベクトルを生成する．

3.4 特徴抽出器の学習

特徴抽出器は，入力したイベント列と同一のイベント列を出力するよう Seq2Seq モデルを学習させることにより得る．3.2 節に示したように，Seq2Seq モデルでは入力系列の情報が圧縮され，固定長のベクトルとして保存される．そのため，Seq2Seq モデルを適切に使用することでプロセスの挙動情報を圧縮することができると考えられる．

Wang らの手法 [6] では，入力 API コール列と同一の API コール列を出力させるような Seq2Seq モデルを学習

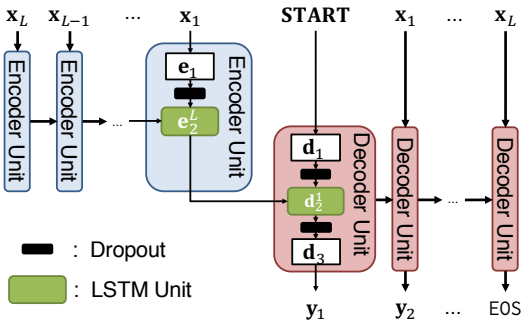

図 3 特徴抽出器の構造
Fig. 3 Architecture of feature extractor.

することにより，入力された API コール列の情報を圧縮している．しかし，Seq2Seq モデルでは入力系列が非常に長い場合，初期に入力された系列の情報が失われてしまうという問題がある．本手法で扱うイベント列は，自然言語に比べ系列長が非常に長くなるため，この問題に対処する必要がある．そこで，本手法ではあらかじめイベント列を分割し，分割した部分イベント列のそれぞれについて特徴の圧縮を行うことでこの問題を回避する．すなわち，あるプロセスの実行によって得られた長さ N のイベント列 $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N]$ を長さ L で分割する．このとき s 個の部分イベント列が得られた場合，$\mathbf{X} = [\mathbf{X}_1, \mathbf{X}_2, ..., \mathbf{X}_s]$ となる．本研究では，学習に使うイベント列の最大個数 s を指定し，s が S より大きい場合は，先頭から S 個の部分イベント列のみを学習する．このようにイベント列を分割し，分割された部分イベント列から特徴ベクトルをそれぞれ抽出することで，イベント列が長い場合でも先頭付近の情報を失うことなく保持できると考えられる．

特徴抽出のための Seq2Seq モデルの構造を**図 3** に示す．エンコーダは通常の隠れ層（$\mathbf{e}_1$），および Long-Short Term Memory（LSTM） Unit を用いた隠れ層（$\mathbf{e}_2$）の 2 層の隠れ層を持つ RNN により構成される．デコーダは通常の隠れ層 2 層（$\mathbf{d}_1$, $\mathbf{d}_3$），および LSTM Unit を用いた隠れ層（$\mathbf{d}_2$）の 3 層の隠れ層を持つ RNN により構成される．また，デコーダの出力として入力されたイベントの次に各イベントが出現する確率がベクトルとして得られる．このベクトルを確率ベクトルと表す．$\mathbf{e}_1$ と $\mathbf{e}_2$，$\mathbf{d}_1$ と $\mathbf{d}_2$，$\mathbf{d}_2$ と $\mathbf{d}_3$ 間の接続には Dropout と呼ばれる汎化性能向上手法を用い，過学習を防いでいる．

学習は次のように行う．ただし，学習データに出現したイベントの集合を $W = \{w_1, w_2, ..., w_n\}$ とし，$p_i(w_j)$ をイベント列の i 番目にイベント w_j が出現する確率を出力する関数とする．まず，エンコーダに長さ L のイベント列 $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_L]$ を逆順に入力していく．エンコーダに L 個のイベントを入力し終えた際の第 2 隠れ層 $\mathbf{e}_2^L$ をデコーダの LSTM Unit の初期値 $\mathbf{d}_2^1$ とする．そして，デコーダに開始記号を表すベクトル **START** を入力することで出力を開始する．デコーダには，$\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_L]$ を順に入力する．$i(i \leq L)$ 番目のイベント $\mathbf{x}_i$ をデコーダに入力した際の出力は，イベント列の $i+1$ 番目に各イベントが出現する確率を表すベクトル $\mathbf{y}_{i+1} = [p_{i+1}(w_1), p_{i+1}(w_2), ..., p_{i+1}(w_n)]$ となる．ここで，$\mathbf{x}_{i+1}$ がイベント w_j の 1-hot ベクトル表現であるとき，$\mathbf{x}_{i+1}$ はイベント w_j が出現する確率が 100%でその他のイベントが出現する確率が 0%の確率ベクトルとみなすことができる．そのため，$\mathbf{y}_{i+1}$ と $\mathbf{x}_{i+1}$ を比較して誤差が小さくなるようパラメータを調整することで，次に $\mathbf{y}_{i+1}$ が出力されるときの $p_{i+1}(w_j)$ を大きくすることができる．本手法では，デコーダから長さ L の確率ベクトルの系列を出力し，入力イベント列と比較して誤差を計算し終えた後，その誤差が小さくなるようパラメータを調整する．以上の操作を，学習データの各イベント列について行う．これを終了条件を満たすまで繰り返す．

以上の手順で学習した結果として得られる特徴抽出の際には，部分イベント列 $\mathbf{X_i}(0 < i \leq s)$ のそれぞれについて，L 個のイベントを入力し終えた時の第 2 の隠れ層 $\mathbf{e}_2^L$ を特徴ベクトルとして抽出する．この操作を s 回行い，プロセスのイベント列 $\mathbf{X}$ に対する特徴量として，s 個の特徴ベクトルを抽出する．

3.5 特徴分類器の学習

特徴分類器は，プロセスを実行して得られたイベント列 $\mathbf{X}$ が正常/悪性のどちらに属するかを分類するよう Seq2Seq モデルを学習させることにより得る．3.4 節で述べたように，特徴抽出器では $\mathbf{X}$ の先頭 $s(s \leq S)$ 個の部分イベント列からそれぞれ特徴ベクトルを抽出して特徴ベクトル列を得る．各特徴ベクトルには挙動の一部の情報が保存されていると考えられるため，特徴ベクトル列を適切に解釈することができれば，挙動の全体像を把握することができると推測する．そこで，Seq2Seq モデルを用いて特徴ベクトル列を解釈することで挙動の特徴を認識し，正常/悪性のどちらであるかを分類する．Seq2Seq モデルは入力系列の順序を考慮して学習を行うことができるため，Seq2Seq モデルを用いて分類学習を行うことで，特徴ベクトル列の時間的な変化を考慮した分類が可能であると考える．

特徴分類のための Seq2Seq モデルの構造を**図 4** に示す．エンコーダは通常の隠れ層（$\mathbf{e}_1$）と，LSTM Unit を用いた隠れ層（$\mathbf{e}_2$）の 2 層の隠れ層を持つ RNN で構成される．デコーダは通常の隠れ層 2 層（$\mathbf{d}_1$, $\mathbf{d}_3$）と，LSTM Unit を用いた隠れ層（$\mathbf{d}_2$）の 3 層の隠れ層を持つ RNN で構成される．デコーダは，特徴ベクトル列の正常度，および悪性度を 2 次元のクラスベクトルという形で出力する．$\mathbf{e_1}$ と $\mathbf{e_2}$，$\mathbf{d_1}$ と $\mathbf{d_2}$，$\mathbf{d_2}$ と $\mathbf{d_3}$，$\mathbf{d_3}$ と出力層の間の接続には Dropout を用い，過学習を防いでいる．

学習は次のように行う．ただし，$\mathbf{F} = [\mathbf{f}_1, \mathbf{f}_2, ..., \mathbf{f}_s]$ はイベント列 $\mathbf{X}$ を長さ L で分割して得られた部分イベント列の先頭 s 個について，3.4 節で述べた特徴抽出を行うこと

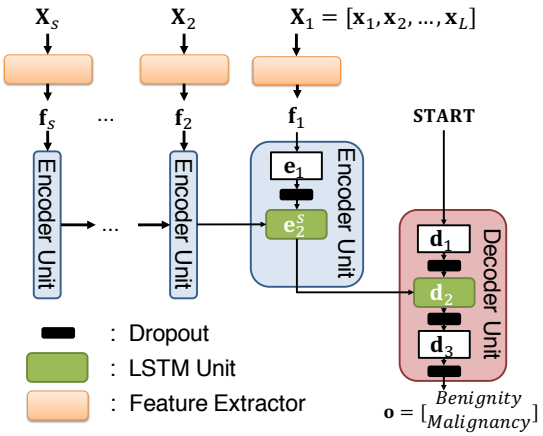

図 4 特徴分類器の構造
Fig. 4 Architecture of feature classifier.

で得たものとする．まずエンコーダが入力として特徴ベクトル列 $\mathbf{F}$ を受け取り，エンコーダに各特徴ベクトルを逆順に入力する．エンコーダに s 個の特徴量で構成される特徴ベクトル列 $\mathbf{F}$ を入力し終えた後の $\mathbf{e}_2^s$ をデコーダの $\mathbf{d}_2$ の初期値とし，デコーダに開始記号を表すベクトル **START** を入力することで特徴ベクトル列の分類を行う．特徴ベクトル列の分類結果は，2 次元のクラスベクトル $\mathbf{o}$ という形で得られる．$\mathbf{o}$ と特徴ベクトル列の真のクラスベクトル（正常なら [1, 0]，悪性なら [0, 1]）を比較して誤差を計算した後，誤差が小さくなるようパラメータを更新する．この操作を，学習データの各イベント列について行う．これを終了条件を満たすまで繰り返す．

3.6 悪性度の推定

プロセスの悪性度の推定は，学習済みの特徴抽出器および特徴分類器を用いて行う．推定時も，学習時と同様，入力としてイベント列 $\mathbf{X}$ を分割して得られた部分イベント列の先頭 $s(s \leq S)$ 個を用いる．まず，特徴抽出器を用い，各部分イベント列 $\mathbf{X_i}(0 < i \leq s)$ の特徴を $\mathbf{f}_i$ として抽出し，s 個の特徴ベクトルからなる特徴ベクトル列 $\mathbf{F}$ を得る．次に，抽出された特徴ベクトル列 $\mathbf{F}$ を特徴分類器に入力してプロセスの悪性度を推定する．出力された分類結果 $\mathbf{o}$ の要素のうち，悪性クラスの値がそのプロセスの悪性度となる．

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 Windows7 の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 ログ記録環境

ログ記録環境を**図 5** に示す．本実験では，入力データとして各プロセスのログデータを用いる．そのため，感染ホスト端末（Victim）上でマルウェアまたは正常バイナリを実行させ，その際に Victim 上で動作している全プロセスの挙動を Process Monitor を用いて記録した．バイナリは Cuckoo Sandbox[*1] を用いて端末に送り込んで実行し，ログの記録時間はバイナリ実行開始から 30 分間とした．また，ログ記録環境上には模擬インターネットサーバ（Fake Internet）およびルータ（Router）を配置した．Fake Internet では，HTTP, DNS などの主要サービスのリクエストに対するダミー応答を返す Inetsim[*2] を動作させた．Victim からの通信は全て Router を通して Fake Internet に送信され，ダミー応答が Victim に返されるよう設定した．

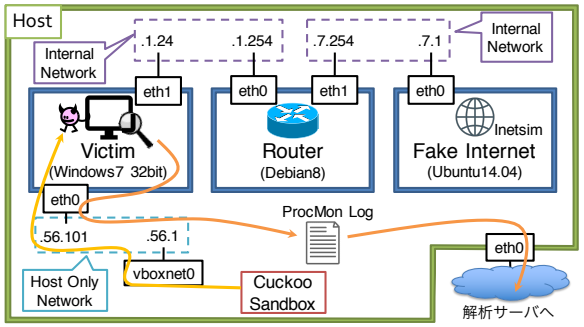

図 5 ログ記録環境
Fig. 5 Logging environment.

4.2 入力データ

4.1 節の実験環境上で正常バイナリ 37 検体，およびマルウェアバイナリ 320 検体をそれぞれ実行し，ログを記録した．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る．得られたログをプロセスごとに分割した後，正常バイナリのログについて，システムプロセスなどの，異なるバイナリの実行時に得られた重複するプロセスのログの一部を除外した．また，マルウェアバイナリのログについて，以下の条件を満たすプロセスのログを悪性プロセスのログとして抽出した．

(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と思われるプロセス

なお，(2) および (3) は，事前に Cuckoo Sandbox を用いて行った各マルウェアの動的解析の結果を参考にして決定した．上記の方法で得た 1000 個の正常プロセスのログと 641 個の悪性プロセスのログの合計 1641 個を用いて提案手法の学習および評価を行った．

4.3 評価方法

本実験では 5 分割交差検証により約 1300 個のプロセスのログ（学習データ）を用いて特徴抽出器，特徴分類器の学習を行い，学習で使用されなかったプロセスのログ（テストデータ）を用いて悪性度を算出した．そして，結果から得

[*1] https://cuckoosandbox.org/
[*2] http://www.inetsim.org/

表 2 真陽性，偽陽性の関係
Table 2 Relationship between TP and FP.

実際の分類	分類結果	
	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）

られた ROC（Receiver Operating Characteristic）曲線下の面積（AUC: Area Under the Curve）を求め，他の手法を用いた場合の AUC と比較することで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）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，そうでない場合は正常と推定する．悪性である場合を陽性（P: Positive）とし，正常の場合を陰性（N: Negative）としたとき，真陽性（TP: True Positive）と偽陽性（FP: False Positive）の関係は**表 2** で表される．また，このとき TPR=TP/P，FPR=FP/N と表される．

本実験では 5 分割交差検証により，各検証ごとに 5 つの ROC 曲線が算出される．これらの ROC 曲線を平均したものを平均 ROC 曲線とし，評価に使用した．本実験ではアベレージ法を用いて平均 ROC 曲線を算出した．

4.4 比較手法

提案手法の有効性を示すため，既存のマルウェア検知手法でしばしば使用される手法を用いた比較実験を行った．比較手法では，イベント列から uni-gram 特徴を抽出し，抽出された特徴を Support Vector Machine（SVM）を用いて分類する．特徴抽出および特徴分類方法を以下に示す．

4.4.1 特徴抽出

単語の出現頻度は，系列から特徴抽出を行う際の最も単純な特徴の 1 つである．そこで本実験では，提案手法との比較手法として，イベント列の uni-gram 特徴を特徴ベクトルとして抽出する手法を用いる．比較手法では，イベントの出現有無のみ考慮する方法，およびイベントの出現頻度を考慮する方法の 2 種類の方法で特徴を抽出する．以降では，出現有無のみを考慮する特徴抽出方法を Bag of Words（BoW），出現頻度を考慮する特徴抽出方法を Term Frequency（TF）とする．特徴抽出では，まず学習データで出現したイベント集合 $\{w_1,\ w_2,\ ...,\ w_n\}$ を作成し，このイベント集合を用いて各イベント列を特徴ベクトル化する．BoW の場合，i 番目のイベント列 X_i の特徴ベクトルは，$f(w)$ をイベント列におけるイベント w の出現有無をバイナリ値として出力する関数として，$\mathbf{V}_{\mathbf{X_i}} = [f(w_1),\ f(w_2),\ ...,\ f(w_n)]$ という n 次元のベクトルで表される．TF の場合，i 番目のイベント列 X_i の特徴ベクトルは，$g(w)$ をイベント列におけるイベント w の出現頻度を出力する関数として，

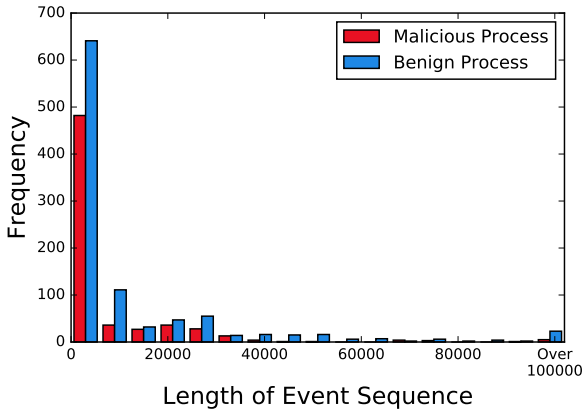

図 6 各プロセスのイベント列の長さの分布
Fig. 6 Distribution of event sequence length.

$\mathbf{V}_{\mathbf{X_i}} = [g(w_1),\ g(w_2),\ ...,\ g(w_n)]$ と表される．

4.4.2 特徴分類

比較手法では，SVM を用いて特徴を正常/悪性に分類する．SVM は 2 値分類に使用される機械学習手法の 1 つで，特徴を最もよく分離する識別面を求めることで分類を行う．識別面は，識別面と最も近い学習サンプルとの距離を最大化することにより求める．また，カーネル関数により特徴を高次元に写像することにより，非線形な識別面を構成することが可能である．SVM では，学習時にハイパーパラメータ C, γ および使用するカーネル関数を予め指定する必要がある．パラメータ C はコストパラメータと呼ばれ，誤分類の許容度を決定する．C の値が大きいほど誤分類が許容される．γ は RBF カーネルで用いられるパラメータで，γ を大きくするほど識別面が複雑な形となる．本実験では，カーネル関数に RBF カーネルを用い，$C = 1000$, $\gamma = 0.001$ とした．

4.5 実装方法

本実験で使用したプログラムは全て Python3.5 で実装し，特徴抽出器および特徴分類器は Chainer[*3]を用いて構築した．本実験で使用した特徴抽出器および特徴分類器のハイパーパラメータを**表 3** に示す．Vocabulary は各交差検証時の学習データで現れたイベントの種類を表す．また，1 epoch は，学習データに含まれる全イベント列を 1 度ずつモデルに入力し，学習させることを指す．イベント列の最大長 L および特徴ベクトル列の最大個数 S は，**図 6** に示す各プロセスのログに記録されているイベント列の長さの分布および学習時間を考慮して決定した．ただし，図 6 の横軸はイベント列の長さを示し，縦軸は出現回数を示す．また，階級幅は 6000 とした．

4.6 実験結果

得られた平均 ROC 曲線，および x 軸を対数軸とした

[*3] https://chainer.org/

表 3 特徴抽出器および特徴分類器のハイパーパラメータ
Table 3 Hyper parameters for feature extractor and classifier.

Vocabulary	108 〜 110
$\mathbf{e}_1$, $\mathbf{e}_2$, $\mathbf{d}_1$, $\mathbf{d}_2$, $\mathbf{d}_3$ の次元	75
イベント列の最大長: L	100
特徴ベクトル列の最大個数: S	60
Optimizer	Adam
Minibatch size	35
Gradient Clipping	5
終了条件	特徴抽出器: 15 epoch
	特徴分類器: 25 epoch

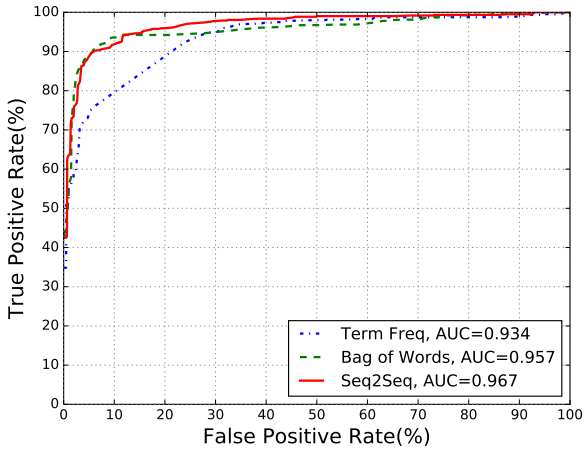

図 7 平均 ROC 曲線と AUC
Fig. 7 Average ROC curves and AUC.

表 4 各交差検証時の AUC
Table 4 AUCs in each cross validation.

	AUC					
手法	1	2	3	4	5	平均
Seq2Seq	0.933	0.979	0.967	0.971	0.983	0.967
BoW	0.916	0.982	0.952	0.948	0.982	0.957
TF	0.904	0.955	0.930	0.922	0.957	0.934

ROC 曲線を**図 7**，**図 8** に示す．また，各 ROC 曲線の AUC を凡例に表示した．図 7 に示したように，提案手法の AUC が最も高い 0.967 で，高精度に悪性プロセスを推定できていると言える．BoW，TF を抽出方法として使用した手法と比較してそれぞれ 0.010, 0.033 高い結果を得られた．また，図 8 に示したように，提案手法では FPR が 1%における検出率が 60%を超えており，低い誤検知率でも比較的高い検出率を得られていると言える．各検証時の AUC の値を**表 4** に示した．表 4 より，提案手法はほぼ全ての検証で比較手法より高い AUC を得ることができた．

5. 考察

5.1 抽出された特徴に関する考察

提案手法では，特徴の学習に Seq2Seq モデルを用い，入力イベント列と出力イベント列が同じになるよう学習を行

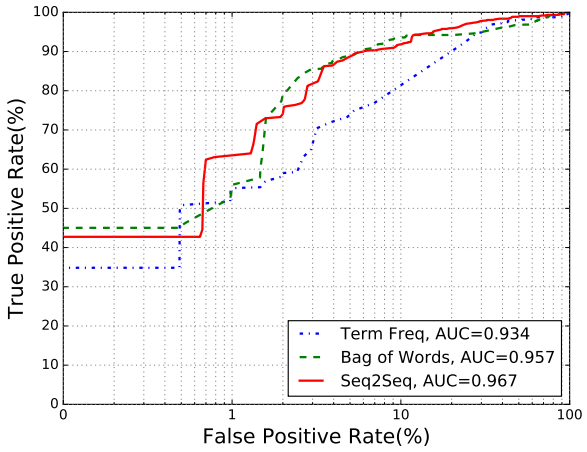

図 8 平均 ROC 曲線 (対数軸)
Fig. 8 ROC curves with logarithmetic axis.

表 5 特徴抽出器によるイベント予測結果
Table 5 Predicted event sequence.

位置	入力	予測
1	Process Start	Process Start
2	SUCCESS	SUCCESS
3	Thread Create	Thread Create
4	SUCCESS	SUCCESS
5	Load Image	Load Image
6	SUCCESS	SUCCESS
7	Load Image	Load Image
8	SUCCESS	SUCCESS
…	…	…
27	RegOpenKey	RegOpenKey
28	NAME NOT FOUND	SUCCESS
29	RegOpenKey	RegQueryValue
30	REPARSE	NAME NOT FOUND
31	RegOpenKey	RegCloseKey
32	NAME NOT FOUND	SUCCESS
…	…	…
97	CreateFileMapping	RegOpenKey
98	FILE LOCKED WITH ONLY READERS	NAME NOT FOUND
99	CreateFileMapping	RegCloseKey
100	SUCCESS	SUCCESS

うことでイベント列の時系列を考慮した特徴の抽出を試みた．ここでは，学習済みの特徴抽出器にイベント列を入力した際に出力される予測イベント列から，抽出される特徴がイベント列の時系列を考慮しているかどうかを考察する．

入力されたイベント列と出力された予測イベント列の例を**表 5** に示す．表 5 に示したように，イベント列の先頭付近のイベントは正しく予測できている．また，位置 27〜32 の予測から，「RegOpenKey → レジストリ関連の操作→ RegCloseKey」のような複数のイベントからなる挙動についても認識できていると考えられる．これらの結果から，特徴抽出器から抽出される特徴は時系列を考慮したものであると言える．

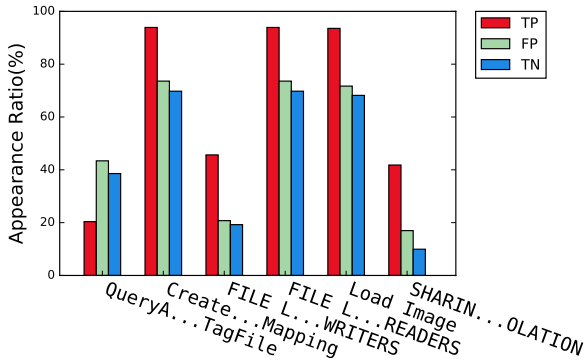

図 9 TP, TN, FP 集合におけるイベント出現割合
Fig. 9 Event appearance ratio between TP, TN and FP sets.

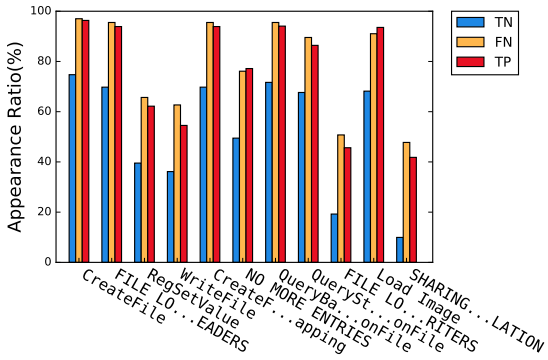

図 10 TN, TP, FN 集合におけるイベント出現割合
Fig. 10 Event appearance ratio between TN, TP and FN sets.

5.2 誤推定の原因に関する考察

実験では，提案手法の AUC が 0.967 と非常に高い検出性能が得られた一方，BoW を用いた比較手法でも 0.957 とほとんど同等の検出性能が得られており，提案手法でも BoW と同様にイベントの出現有無が推定に大きな影響を及ぼす可能性が予想される．ここでは，正常/悪性プロセスが生成するイベント列において，各イベントの出現の割合を比較することで，イベントの出現有無が推定に与える影響を考察する．ただし，悪性度が 0.5 以上の正常/悪性プロセスの集合をそれぞれ FP/TP 集合とし，悪性度が 0.5 未満の正常/悪性プロセスの集合をそれぞれ FN/TN 集合とする．各交差検証時のテストデータの推定結果を用いたとき，各集合に属するプロセスの数は，TP 集合，FN 集合，TN 集合，FP 集合の順に 546，67，947，53 であった．

図 9 に TP 集合と FP 集合，または TN 集合と FP 集合で，出現する割合が 20 ポイント以上異なるイベントとその割合を，**図 10** に TN 集合と FN 集合，または TP 集合と FN 集合で，出現する割合が 20 ポイント以上異なるイベントとその割合を示す．横軸はイベント名を示し，縦軸は各イベントが，各集合に属するプロセスのイベント列に出現する割合を示す．例えば，図 10 において `CreateFile` というイベントに注目すると，TP 集合のプロセスの 9 割以上で出現しているが，FP 集合のプロセスでは 7 割程度しか出現しない．図 9，図 10 より，いずれも正常プロセスの集合である TN および FP 集合，悪性プロセスの集合である TP および FN 集合のプロセスにおけるイベントの出現割合が類似している一方，TP 集合と FP 集合，TN 集合と FN 集合ではイベントの出現割合は類似していない．この結果から，提案手法ではイベントの出現有無が推定に与える影響は小さく，5.1 節の結果より，提案手法ではイベント列の時系列的な特徴を捉えているため，この特徴が検出性能に影響を与えている可能性が高いと推測される．

6. 終わりに

本論文では，端末のプロセスの挙動に着目し，挙動の特徴を抽出および分類することで悪性プロセスを推定する手法を提案した．本手法では，2 つの Seq2Seq モデルを用いて可変長のイベント列を効果的に圧縮し，圧縮された特徴を分類することにより悪性プロセスを推定した．1641 個のプロセスのログを用いた 5 分割交差検証を行い，ROC 曲線から AUC を求めることで，比較のため BoW および TF を用いた手法とともに提案手法を評価した．また，実際に特徴抽出器において時系列を考慮した特徴が学習されていることの提示，および，誤推定されたプロセスについてイベントの出現有無の観点から原因を分析した．

今後は，より実環境に近い環境で得られたログを用いた実験を行い，感染端末を検出可能であるか評価したいと考えている．また，path 情報など特徴抽出の際に用いる情報を追加し，より複雑なデータを用いて性能を評価することで提案手法の有効性を示したいと考えている．

参考文献

[1] JPCERT/CC: インシデント調査のための攻撃ツール等の実行痕跡調査に関する報告書 (PDF), 入手先 ⟨https://www.jpcert.or.jp/research/20160628ac-ir_research.pdf⟩ (2017.8.21).
[2] JPCERT/CC: Cookie ヘッダーを用いて C&C サーバとやりとりするマルウエア ChChes(2017-01-26), 入手先 ⟨https://www.jpcert.or.jp/magazine/acreport-ChChes.html⟩ (2017.8.21).
[3] 中里ら: プロセスの出現頻度や通信状態に着目した不審プロセス判定, 信学技報, Vol. 115, No. 488, pp. 77-82 (2016).
[4] 山本ら: 不審プロセス特定手法の提案, CSS2013, pp. 634-641 (2013).
[5] Pascanu, R., et al.: Malware classification with recurrent networks, ICASSP2015, pp. 1916-1920 (2015).
[6] Wang, X. and Yiu, S. M. : A multi-task learning model for malware classification with useful file access pattern from API call sequence, arXiv:1610.05945 (2016).
[7] 飛山ら: 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, CSS2016, pp.310-317 (2016)
[8] Sutskever, I., et al.: Sequence to sequence learning with neural networks, NIPS2014, pp. 3104-3112 (2014).
[9] TechNet-Microsoft: Process Monitor, 入手先 ⟨https://technet.microsoft.com/ja-jp/sysinternals/processmonitor.aspx⟩ (2017.08.21).