単語レベルの言語モデルを用いた悪性PowerShellの検出

田尻 裕貴[1,a)]　三村 守[1,b)]

概要：サイバー攻撃において，攻撃対象の端末にインストールされている正規のツールを利用する傾向が強まっている．特に攻撃ツールとして，Microsoft 社が提供する PowerShell を悪用するケースが年々増加しており問題となっている．先行研究では，文字ベースのディープラーニングを用いた悪性 PowerShell コマンドを検知する手法が提案された．提案された手法は，伝統的な自然言語処理および文字レベルでの畳み込みニューラルネットワークを組み合わせた手法である．しかしながらこの手法では，前処理に動的解析を用いており，解析に時間を要する．そこで本研究では，静的解析のみを用い，単語ベースの言語モデルによって悪性および良性のサンプルから特徴ベクトルを作成し，未知のサンプルを分類する手法を提案する．データセットは，HybridAnalysis から入手した良性および悪性のサンプル，および github から入手した良性サンプルから作成した．検証実験では，F 値は 0.82 となった．また，未知の悪性サンプルを約 5 割検知できることを確認した．

キーワード：PowerShell，潜在意味インデックス，Doc2Vec

Detection of Malicious PowerShell Using a Word Level Language Model

Yui Tajiri[1,a)]　Mamoru Mimura[1,b)]

Abstract: There is a growing tendency for cybercriminals to abuse legitimate tools installed on the target computers for cyberattacks. In particular, the use of PowerShell provided by Microsoft has been increasing every year and has become a problem. In previous studies, a method to detect malicious PowerShell commands using character-based deep learning was proposed. The proposed method combines traditional natural language processing and character-level convolutional neural network. This method, however, requires dynamic analysis for preprocessing, and thereby requires time. This paper proposes a method to classify unknown PowerShell by using only static analysis. Our method uses feature vectors extracted from malicious and benign PowerShell scripts using a word-based language model for classification. Datasets were generated from benign and malicious PowerShell scripts obtained from HybridAnalysis, and benign PowerShell scripts obtained from github. Our experiment shows that the F-measure achieves more than 0.82. Furthermore, we confirmed that almost 50% of unknown malicious PowerShell script could be detected.

Keywords: PowerShell，Latent Semantic Indexing，Doc2Vec

1. はじめに

近年，情報通信技術の発達によって社会生活や経済活動のデジタル化が進んでいる．デジタル化によって業務の効率化や，消費活動の利便性の向上など様々な恩恵が社会にもたらされている．一方で，デジタル化した重要インフラ，企業や金融機関の制御および管理システムはサイバー犯罪者の攻撃対象になり，多くの被害が発生している．このようなサイバー攻撃の手口も年々多様化，巧妙化しており，アンチウイルスソフトや侵入検知システムで検知できない攻撃も増加している [1]．

Symantec のサイバー脅威に関する最新のホワイトペーパーによると，サイバー犯罪者は攻撃対象の端末にインス

[1] 防衛大学校情報工学科
National Defense Academy in Japan
[a)] em58028@nda.ac.jp
[b)] mim@nda.ac.jp

トールされている正規のツールやオペレーティングシステムの機能を利用して攻撃する傾向が強くなっている [2].

特に PowerShell は，Windows のほぼすべての機能にアクセス可能な強力なツールである．そのため，サイバー犯罪と標的型攻撃の両方で主要な手段となっており，悪意のある PowerShell スクリプトは増加傾向にある [2]．また，PowerShell を利用した攻撃は，標的型攻撃やランサムウェア等の異なる技術や攻撃手法と組み合わせて行われることも多く，その脅威はますます増加している．しかしながら，悪性 JavaScript や悪性マクロ等の研究は数多く行われている反面，PowerShell を悪用した攻撃に対する研究はまだ少なく，十分になされているとは言い難い．

悪性の PowerShell の検知に関する先行研究では，Hendler らがディープニューラルネットワークを用いた悪性 PowerShell コマンドの検知手法を提案した [4]．Hendler らが提案した手法は，伝統的な自然言語処理および文字レベルでの畳み込みニューラルネットワークを組み合わせた手法である．この手法では，前処理に動的解析を用いており，解析に時間を要する．そこで本研究では，前処理を含めて静的解析のみを用いた手法を検討する．近年の自然言語処理技術の発達に伴い，マルウェア検知に応用する研究がなされている．しかしながら，悪性 PowerShell の検知の研究に関しては自然言語処理技術に焦点を当てたものはなく，多くはディープラーニングや機械学習に焦点を当てたものである．本論文では，自然言語処理および機械学習を用いた悪性 PowerShell スクリプトの検出手法を提案する．提案手法では，スクリプトを静的解析し，単語レベルの言語モデルを作成する．言語モデルの作成には，伝統的な自然言語処理技術である Bag-of-words に加えて，新しい自然言語処理技術である LSI および Doc2Vec を採用した．PowerShell から，単語ベースの各言語モデルを作成した後，悪性および良性のサンプルから特徴ベクトルを作成し，未知の PowerShell スクリプトの分類をする．データセットは，収集した PowerShell のスクリプトを時系列で，訓練データ及び未知の PowerShell としてテストデータの 2 種類に分割した．分類器は，訓練データから生成した特徴ベクトルで訓練した，SVM および XGBoost を用いた分類器を用いる．分類器の評価は，テストデータの中から悪性 PowerShell スクリプトを検知し精度を評価した．

本研究の貢献は次のとおりである．

(1) 静的解析のみを用い，単語ベースの言語モデルによって悪性および良性の PowerShell スクリプトから特徴ベクトルを作成し，未知の PowerShell スクリプトを分類する手法を提案した．

(2) 時系列のデータセットを作成し，未知の悪性 PowerShell スクリプトに対する検知精度を評価した．

本論文の構成を以下に示す．第 2 節では PowerShell の概要について述べ，第 3 節で関連研究について紹介する．第 4 節では本研究で用いる関連技術について紹介する．第 5 節では提案手法について説明し，第 6 節では検証実験の手法およびその結果について述べる．第 7 節では考察を述べる．最後に第 8 節では，本論文のまとめを述べる．

2. PowerShell

PowerShell は Microsoft 社が提供する拡張可能なコマンドラインシェルおよびスクリプト言語である．オブジェクト指向で.NET を基盤としており，高い柔軟性と強力な機能を持つ．これにより，Windows の主要機能に容易にアクセスすることや遠隔操作が可能である．通常，システム管理者によるオペレーティングシステムとプロセスの管理や自動化のために使用される．

2.1 PowerShell の悪用

今日のサイバー攻撃では，PowerShell の強力な機能がサイバー犯罪者に悪用され，より強力で発見されにくいマルウェアの作成や痕跡を残さない攻撃を可能としている．PowerShell という正規ツールの使用は，攻撃対象の端末にマルウェアを直接送り込まずに PowerShell を使用してマルウェアをダウンロードまたは作成する等攻撃の柔軟性を向上させた．また，攻撃者は不正なタスクを正常なタスクに紛れ込ませることで攻撃を検出困難なものとした．これにより，被害者に攻撃が露呈しにくくすることが可能となった．これらの要因から，PowerShell はサイバー犯罪者の攻撃ツールの一つとして採用されるケースが増加している．

2.2 PowerShell の難読化

悪意のある PowerShell スクリプトの多くは，解析やウイルス対策ソフトによる検知を困難にするために難読化されている．難読化は，処理内容を維持したままバイナリファイルやソースコードを修正し，人間に理解しにくくする処理である．本来はソフトウェアの著作権保護や改竄防止のために用いられる技術である．しかしながら，難読化技術は攻撃者がマルウェアの解析やウイルス対策ソフトの検知を回避するためにしばしば使用される．難読化を解除せずに，コードから実行される内容を把握することは困難である．マルウェア対策として，難読化されたスクリプトが悪性か良性かを区別する際，サンドボックス内で対象のスクリプトを実行し，挙動を確認して判断する手法がよく用いられる．難読化の方法には，変数名やクラス名等の置換，Base64 や ASCII によるエンコーディング，文字列の分割，冗長な空白文字の挿入，実行時にコマンドを生成する等様々な手法がある．代表的な難読化手法の例を表 1 に示す．攻撃者は，表 1 に示した難読化手法を組み合わせて使用することで，容易に難読化を解除できないようにしている．

表 1　難読化手法の例

番号	摘要	例
1	Base64 のデコード・エンコード	$([Convert]::FromBase64String ('7b0HYBxJliUmL23K... 9k9/yGyf/Dw=='))
2	文字列の分割	.('S'+'tart-P'+'roce'+'ss')...
3	冗長な空白の挿入	po w e r s h e l l . e x e - n o p
4	ランダムに大文字・小文字に変換する	$nW=NEW-ObJECt SyS-Tem.NeT.WebCLIeNT;
5	実行時にコマンドを生成	$cmd = "get-"+"Process" Invoke-Expression $cmd
6	エスケープシーケンス (‘) の挿入	$dm=... wI‘N3‘2‘_co‘MPUTE RSYS‘TEm...
7	文字列の ASCII コードへの変換	((...45,72,111,115,116,32,...)| %{ ([Int]$_ -as [char]) }) -Join ”

3. 関連研究

本節では悪意のある PowerShell スクリプトの検知手法に関連する研究を紹介し，本稿の新規性を明らかにする．Hendler らは，ディープニューラルネットワークを用いた悪性な PowerShell コマンドの検知手法 *Deep/Traditional Models Ensemble(D/T Ensemble)* を提案した [4]．*D/T Ensemble* は，畳み込みニューラルネットワークおよびリカレントニューラルネットワーク等のディープラーニングアーキテクチャに基づく検知手法，および n-gram や Bag-of-Words 上の線形分類等による伝統的な自然言語処理に基づく検知手法を比較し，ディープラーニングと伝統的な自然言語処理を用いた手法のそれぞれ最良の検知率であった 4-CNN と 3-gram を組み合わせたものである．Hendler らの提案する手法では，前処理として PowerShell コマンドを抽出するため動的解析をしている．難読化への対策は Base64 のデコードや無作為または冗長に挿入された空白を単純な空白へ置換をしている．また，IP アドレスのような個々の値を持つものは同一の要素として扱うため，数字を他の共通の文字へ置換した．前処理の最後に，PowerShell は大文字小文字を区別しないため，コマンドを一般化するために，すべて小文字に変換している．4-CNN および 3-gram の検知器を使用してコマンドを分類し，2 つの検知結果を比較し，閾値以上であれば最大のスコアを採用し，閾値を下回った場合は 2 つのスコアの平均を採用する．Rusak らは，抽象構文木 (AST) とディープラーニングを組み合わせた悪性の PowerShell の検知手法を提案した [5]．テキストベースではなく，コードの構造情報を使用した検知器であることを特徴としている．AST はプログラミング言語処理系において，コンパイラ等のプログラムの中間表現として，しばしば使われる．提案された手法では，前処理として Base64 でエンコードされたスクリプトまたはコマンドをデコードし，構文解析をして AST に変換する．変換したのち，PowerShell スクリプトのコーパスに基づいて各 AST ノードのルートを構築し，マルウェアファミリの区別を可能としている．しかしながら，[4]，[5] の手法では前処理に関する手法が明示されていないため，再現性が確保されていない．本稿では，全ての段階で再現性を確保したうえで静的解析のみのプログラムを実装した．

Ugart らは PowerShell の難読化の解除をするツール *PowerDrive* を紹介した [6][7]．*PowerDrive* は静的および動的な多段式難読化解除ツールである．Ugart らは，*PowerDrive* を使用した実験において，データセットとして用意したスクリプトのうち 95%を正確に分析した．また，PowerShell の行動モデルの分類法と悪意のあるドメインの包括的リストを提供している [7]．*PowerDrive* は，github で公開されており高い再現性が図られているが，[7] は難読化された PowerShell の解析に重点を置いたものである．そのため悪性の PowerShell の検知に特化したものではない．

本稿では，自然言語処理において単語単位の語順を特徴ベクトルとして得る Doc2Vec およびトピックモデルである LSI を採用し検知器を作成する．そのため，単語の出現頻度だけでなくスクリプト単位で特徴を得ることができる．また，難読化手法には Base64 や ASCII 以外の手法でコマンドの冗長化を図る手法もある．入力文字長を 1024 文字で固定する文字レベル CNN では重要な情報を漏らす可能性がある．本研究では，スクリプト全体を対象とすることで情報の取りこぼしを防いでいる．

4. 関連技術

この節では，本研究で使用する自然言語処理技術および機械学習技術について説明する．

4.1 自然言語処理技術

自然言語処理技術では，自然言語をコンピュータで処理するために数値に変換する．本研究の提案手法では Bag-of-Words，Latent Semantic Indexing および Doc2Vec を用いる．

4.1.1 Bag-of-Words

Bag-of-Words(BoW) は，ある文書を単語の出現頻度を元に表現することでベクトルに変換するモデルである．d を文書，w を単語，n を w に対応した出現頻度として表した場合，文書 d は (1) で表すことができる．

$$d = [(w_1, n_{w_1}), (w_2, n_{w_2}), (w_3, n_{w_3}), ..., (w_j, n_{w_j})] \quad (1)$$

この (1) 式を元に n の位置を固定することで単語 w を省略すると，d を数値で表現することができる．これにより各文書 $\hat{d}_i$ における単語の出現数を表した文書と単語の関係性をベクトル（文書-単語マトリクス）で (2) のように表現できる．

$$\hat{\mathbf{d}}_\mathbf{i} = (n_{w_1}, n_{w_2}, n_{w_3}, ..., n_{w_j}) \quad (2)$$

BoW を用いることで，各文書はユニークな単語数で固定された次元数のベクトルに変換される．

4.1.2 Latent Semantic Indexing

Latent Semantic Indexing(LSI) は，文書群と文書に含まれる単語群の関連性を分析する自然言語処理技術である．LSI では，BoW により求めた文書-単語マトリクスの各成分に重み付けをしたベクトルに対して特異値分解を行うことで，文書の特徴を計算し関連性を分析できる．最後に導き出された関連性は，潜在的意味を示すとされている．

4.1.3 Doc2Vec

Doc2Vec は，Le らによって提案された Paragraph Vector[8] の実装の 1 つである．Paragraph Vector は，Mikolov らによって提案された単語の特徴ベクトルを生成するモデルである word2vec[9] を拡張し，単語ではなく文書の特徴ベクトルを生成するモデルである．Paragraph Vector は，入力文章の長さに制限がないため，入力の長さよって処理を変える必要がない．また構文解析を必要としない．Paragraph Vector の基礎となる word2vec は，文書中の単語同士の関係性をニューラルネットワークを用いて学習させ，その単語の特徴を表すベクトルに変換する．このため，Paragraph Vector では文書間で類似性や関連性を計算が可能である．

4.2 機械学習技術

本研究で用いる機械学習技術は，Support Vector Machine(SVM)，XGBoost および RandomFores の 3 種類である．この 3 種類を選択した理由は，異なる性質を持つ機械学習技術であり，言語モデルと機械学習技術の相性を比較するためである．

4.2.1 Support Vector Machine

SVM は教師あり学習を用いるパターン認識モデルであり，分類問題や回帰問題に適用される．

4.2.2 Random Forest

RandomForest は Leo Breiman によって提案された決定木を弱学習器とする集団学習アルゴリズムであり，ランダムサンプリングされたトレーニングデータによって学習した多数の決定木を使用する．決定木は，葉が分類，枝がその分類に至るまでの特徴の集まりを表す木構造を示し，その学習は，元となる集合を属性値テストに基づいて部分集合に分割することによって行う．

4.2.3 XGBoost

XGBoost は，勾配ブースティングと Random Forests を組み合わせたアンサンブル学習であり，スケーラブルな End-to-End の Tree Boosting システムを持つ分類器である．

5. 提案手法

本節では，本研究で提案する悪性 PowerShell スクリプトの検知手法について詳細を述べる．

5.1 概要

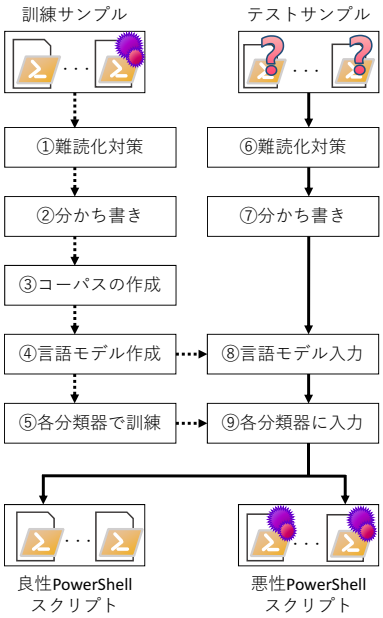

図 1 提案手法

図 1 に提案手法の概要を示す．訓練サンプルの PowerShell スクリプトおよび，テストサンプルの PowerShell スクリプトにはそれぞれ悪性の PowerShell スクリプトと良性の PowerShell スクリプトが含まれている．以下に提案手法の手順を図 1 の番号順に示す．

① 訓練サンプルに対して前処理をする．
② 前処理を実施した訓練サンプルに対して分かち書きを実施する．
③ 分かち書きを実施した訓練サンプルからコーパスを作成する．
④ BoW，Doc2Vec および LSI で各言語モデルを作成する．
⑤ SVM，XGBoost および RandomForest の 3 種類の分類器を訓練する．
⑥ テストサンプルに対して前処理を実施する．
⑦ 前処理をしたテストサンプルに対して分かち書きを実施する．
⑧ 前処理を行ったテストサンプルを④で作成した各言語モデルに入力する．
⑨ テストサンプルを⑤で作成した各分類器に入力し，分類を行う．

以下，各手順の詳細について述べる．

5.2 訓練サンプルの前処理

前処理として，データクレンジング，図 1 中の「①難読化対策」および「②分かち書き」を行う．データクレンジ

ングでは，コメント文，IP アドレスや URL 等様々なバリエーションを持つ特定を文字列は特徴の一つとするため，表 2 に示すとおり置換を実施した．また，PowerShell のエスケープシーケンス (‘a や ‘v) を訓練サンプルのスクリプトファイルから文字列を抽出し，難読化対策ありの場合は表 3 に示す処理をスクリプトに対して施す．その後，正規表現を用いてスクリプトを単語に分割しコーパスを作成する．

表 2 特定の文字列の置換

文字列（例	置換後文字列
IP アドレス 例）255.255.0.26	IPAddress
複数行にまたがるコードを一行で記述	PowerShell の終端文字 ; ／空白文字（空白・タブ・改行）
難読化され冗長となった文字列	置換し特徴の一つにする．
大文字小文字が混合されているスクリプト	小文字に統一する．
IP アドレス・URL	置換し特徴の一つにする．
実行ファイル名	置換し特徴の一つにする．

5.2.1 難読化対策

2.2 で述べた難読化への対策について説明する．正規表現のマッチングを利用して難読化部分を抽出し，表 3 に示す処理をした．

表 3 難読化への対策

番号	対応方法	例
複数行にまたがるコードを一行で記述	PowerShell の終端文字 ; ／空白文字（空白・タブ・改行）	対策前：$id = (Get-Wmi Object Win32...).UUID;$log=$lk+$uuid; 対策後：$id = (Get-WmiObject Win32...).UUID; $log=$lk+$uuid;
Base64 にエンコードされた文字列	置換し特徴の一つにする．	対策前：Base64String(’7b0HYBxJliUmL23K... 9k9/yGyf/Dw==’) 対策後：Base64String(’base64str’)
大文字小文字が混合されているスクリプト	小文字に統一する．	対策前： $mSEnT = nEW-obJEcTsYstEm.io.mEmOrySTReaM 対策後：$msent = new-object system.io.memorystream

5.2.2 分かち書き

分かち書きは，英語と同様の空白に加えて，PowerShell における終端記号，括弧や演算子等のコマンドや変数の境目となる記号 ([ ]{}()+.&”’,:) を，単語の区切りとして文字列を分割した．

5.3 コーパスの作成

まず，分かち書きをした訓練サンプルからユニークな単語の辞書を作成する．ユニークな単語には，出現回数が極端に多いものと，少ないものが存在する．一般的には，共通して現れる単語は分類に影響を与えないとして削除されることが多いが，本研究では文脈を維持するために保持する．出現回数が極端に少ない単語は，特徴に満たないといえる．このため，悪性と良性の分類に影響を与えない単語を gensim の辞書フィルタを用いて削減した．なお，難読化対策した際に置換した文字列については特徴として保持する．

5.4 言語モデルの作成

図 1 中における「④言語モデルの作成」では，訓練サンプルから取得した PowerShell スクリプトを用いて言語モデルを構築する．使用する自然言語処理技術は Doc2Vec, LSI および Bag-of-Words である．言語モデルを作成する際のパラメータは各モジュールの初期値とし，モデルのベクトル空間のサイズは各言語モデルを比較するため固定とした．その他のパラメータは各自然言語処理技術のモジュールの初期設定値とした．

5.5 分類器の訓練

図 1 中における「⑤分類器の訓練」では，訓練サンプルにより SVM，XGBoost および RandomForest の訓練を行う．訓練サンプルは SVM，XGBoost および RandomForest を用い作成した各特徴ベクトルに，悪性または良性のラベルを付したものである．データの詳細については 6.1 で示す．

5.6 未知の PowerShell スクリプトの分類

図 1 中の未知の PowerShell スクリプトを，悪性または良性の PowerShell スクリプトに分類する手順を示す．まず，未知の PowerShell スクリプトを，5.2 と同様の手順で前処理する．訓練サンプルから作成した各言語モデルに前処理した未知の PowerShell スクリプトを入力し，特徴ベクトルに変換する⑧．最後に，訓練サンプルを用いて訓練された各分類器を用いて，悪性または良性の分類を行い，悪性の PowerShell スクリプトを検知する．

5.7 実装

提案手法は Python2.7 を使用し，表 4 に示す環境で実装した．自然言語処理および分類のための機械学習の実装には表 5 に示すライブラリを用いた．

6. 検証実験

6.1 データセット

データセットは HYBRID ANALYSIS[13] から収集した良性と悪性の PowerShell スクリプト 455 件，github[14] か

表 4　環境

CPU	Core i7-8700K 3.70GHz
Memory	16GB
OS	Windows10 Home
使用言語	Python2.7

表 5　使用した主なライブラリ

自然言語処理	Bag-of-Words Doc2Vec LSI	gensim -3.7.3[10]
機械学習	SVM RandomForest	sickit-learn -0.20.4[11]
	XGBoost	xgboost -0.82[12]

ら入手した良性の PowerShell スクリプト 5000 件の合計 5455 件からなる．収集方法は，python を用いたウェブスクレイピングである．HYBRID ANALYSIS からは，公開 API を利用して 1 日 4 回投稿された PowerShell に関する情報の投稿の有無を検索した．投稿がある場合，該当する PowerShell スクリプトのサンプルが入手可能か検索し，可能な場合，サンプルとして入手した．github からは公開されている PowerShell スクリプトを検索し，結果の上位からウェブスクレイピングで入手した．悪性の PowerShell スクリプトは，HYBRID ANALYSIS に投稿された年で分割した．内訳は表 6 のとおりである．検証実験において，未知の悪性 PowerShell スクリプトを検知可能か検証するため，データセットを時系列で分割することにした．そのため，データセットを 2018 年から 2019 年 6 月までのデータを 2018 年と 2019 年の 2 つに分割し，2018 年を既知の PowerShell スクリプト，2019 年を悪性または良性の判定が未知の PowerShell スクリプトとしている．なお github から収集した良性のスクリプトについては時系列情報がないため，2018 年と 2019 年のデータセットに半数ずつ使用する．github から収集した良性の PowerShell スクリプトの分割方法については，スクリプトファイルをファイル名で昇順にソートし，上位から交互に 2 つに分割した．

表 6　データセット内訳

Hybrid Analysis			github
年	悪性	良性	良性
2018	251	118	5000
2019	59	27	

6.2　評価指標

まず，本研究で用いる評価指標の定義について述べる．テストデータを分類器で予測したラベルを予測結果，正答のラベルを真の結果と呼ぶ．予測結果と真の結果の関係を表 7 に示す．ここでは，PowerShell を悪性と予測することを Positive，良性と予測することを Negative と定義する．分類問題の評価制度の指標には一般的に次の 4 種が用いられる．正解率 (Accuracy) は，予測結果と正答がどの程度

表 7　予測結果と真の結果の関係

		真の結果	
		悪性	良性
予測	悪性	True Positive (TP)	False Positive (FP)
	良性	False Negative (FN)	True Negative (TN)

一致しているかを判断する指標であり，(3) で定義される．

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{3}$$

適合率 (Precision) は，悪性と予測したデータのうち，実際に悪性であるデータの割合であり，(4) で定義される．

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

再現率 (Recall) は，実際に悪性であるデータのうち，悪性と予測したデータの割合であり，(5) で定義される．

$$Recall = \frac{TP}{TP + FN} \tag{5}$$

F 値 (F-measure) は，適合率と再現率の調和平均により算出される評価尺度であり，(6) で定義される．

$$F\text{-}measure = \frac{2Recall \times Precision}{Recall + Precision} \tag{6}$$

本稿では，(1) 悪性 PowerShell スクリプトを検知するという目的から検知の取りこぼしがないか，(2) また現実の使用に即するものか を評価する際に着目した．そのため，検証実験では，評価指標として，再現率 (Recall) および F 値 (F-measure) を用いる．そのため評価指標については，(1) については再現率（Recall），(2) については F1 値を用い評価する．

6.3　実験内容

検証実験の環境は実装した環境と同じく，表 4 に示すとおりである．ただし，本実験における未知の悪性 PowerShell スクリプトは，単純に新しい悪性マルウェアを指す．実験は未知の悪性 PowerShell スクリプトを検知可能か検証するためデータセットを 2018 年から 2019 年 6 月までのデータを年で 2 つに分割し，2018 年を既知の PowerShell スクリプト，2019 年を未知の PowerShell スクリプトとしている．

6.4　実験結果

分類器の精度の検証結果を表 8，時系列分析の実験結果を表 9 に示す．なお，分類器の精度検証には 5-Fold ClossValidation を用いた．良性 PowerShell の検知については，いずれの言語モデルにおいても F 値は 0.96 から 0.97，recall 値は 0.96 から 1.00 と高い精度であった．このため，分類器の精度の検証は悪性の F 値および recall 値に重点を置く．分類器の精度は表 8 に示すように，LSI が悪性の F 値が 0.65，悪性の recall 値が 0.57 と，悪性 PowerShell 検知において，最も高い精度の言語モデルであった．また，難読化対策の有無の比較をすると，対策なしよりも対策あり

の方が分類精度が Doc2Vec と XGBoost，Bag-of-Words と RandomForest の組み合わせを除いて，F 値が 0.02 以上，recall 値が 0.03 以上向上した．言語モデルと分類器の組み合わせについては LSI と XGBoost の組み合わせが，難読化対策がありの場合は F 値 0.65，recall 値 0.57，無の場合は F 値 0.60，recall 値 0.50 であり，相性が良いことが分かった．

時系列分析は，悪性の PowerShell の recall 値は表 9 に示す結果となった．各言語モデルの recall の最大値を比較すると，LSI が 0.55，Doc2Vec が 0.36，Bag-of-Words が 0.22 であり，未知の悪性 PowerShell の検知には LSI が有効であることが分かった．また，F 値を見ると，LSI を使用し，難読化対策ありの場合のいずれの分類器を使用した場合も 0.54 以上であり，LSI の有効性を示している．

表 8　分類器の精度 (5-Fold ClossValidation)

言語モデル	分類器	対策	良性		悪性	
			recall	F 値	recall	F 値
Bag-of-Words	SVM	有	0.98	0.96	0.46	0.56
		無	0.98	0.96	0.42	0.50
	XGBoost	有	0.98	0.97	0.54	0.63
		無	0.99	0.97	0.49	0.60
	Random Forest	有	0.98	0.96	0.44	0.55
		無	0.98	0.97	0.45	0.55
Doc2Vec	SVM	有	0.96	0.96	0.50	0.53
		無	0.97	0.96	0.42	0.48
	XGBoost	有	0.99	0.96	0.33	0.46
		無	0.99	0.96	0.38	0.49
	Random	有	1.00	0.96	0.23	0.35
		無	0.99	0.96	0.19	0.30
LSI	SVM	有	0.99	0.97	0.52	0.62
		無	0.99	0.97	0.43	0.56
	XGBoost	有	0.98	0.97	**0.57**	**0.65**
		無	0.99	0.97	**0.50**	**0.60**
	Random	有	0.99	0.97	0.47	0.59
		無	0.99	0.97	0.44	0.57

表 9　時系列分析結果

言語モデル	分類器	対策あり		対策なし	
		recall	F 値	recall	F 値
Bag-of-Words	SVM	0.00	0.00	0.00	0.00
	XGBoost	0.10	0.14	0.09	0.09
	RandomForest	0.06	0.06	**0.22**	0.12
Doc2Vec	SVM	**0.36**	0.40	0.35	0.38
	XGBoost	0.32	0.46	0.31	0.46
	RandomForest	0.15	0.25	0.13	0.21
LSI	SVM	0.42	**0.54**	0.33	0.46
	XGBoost	**0.55**	**0.64**	0.35	0.49
	RandomForest	0.44	**0.57**	0.29	0.43

7. 考察

7.1 分類精度

分類精度は 6.4 節の実験結果でも述べたように，LSI を用いた分類器は，表 8 に示すように，既知の悪性 PowerShell に対する最大の recall 値が 0.57，表 9 に示すように，未知の悪性 PowerShell に対する recall 値が 0.55 であり，最も高い検出率であった．Doc2Vec は既知の悪性 PowerShell に対しては recall 値の最大が 0.50 である一方，未知の悪性 PowerShell に対しては recall 値の最大が 0.36 となり，LSI に劣る結果となった．また，Bag-of-Words は，既知の悪性 PowerShell に対しては recall 値の最大が 0.54 と LSI と大きな差は見られなかった．しかし，未知の悪性 PowerShell に対しては recall 値の最大が 0.22，さらに最低値が 0.00 であり，未知の悪性 PowerShell の検知には不適であるといえる．以上から，使用する言語モデルは LSI が有効であるといえる．言語モデルと分類器の組み合わせ毎の悪性 PowerShell の検知精度（recall 値および F 値）をグラフ化したものを，図 2 に示す．図 2 に示すとおり，言語モデルと分類器の組み合わせによって，精度が大きく異なる．例として言語モデルである Doc2Vec に注目する．SVM と RandomForest を比較すると，同じ言語モデルであるにもかかわらず，RandomForest の場合は，精度検証と時系列分析ともに検知精度に大きな差が表れている．つまり，言語モデルと分類器の組み合わせには相性があり，分類精度に影響を与える．言語モデルと分類器の組み合わせ毎の，分類器の精度検証の結果と時系列分析の結果を比較する．分類器の精度検証において高い精度である反面，時系列分析では精度が低下した組み合わせがほとんどである．一方，分類器の精度検証と時系列分析において分類精度を維持した組み合わせもある．分類精度を維持した組み合わせの中で，最も高い精度の組み合わせは，LSI と XGBoost である．これらから，悪性 PowerShell の検知に適した言語モデルと分類器の組み合わせは，LSI と XGBoost の組み合わせといえる．

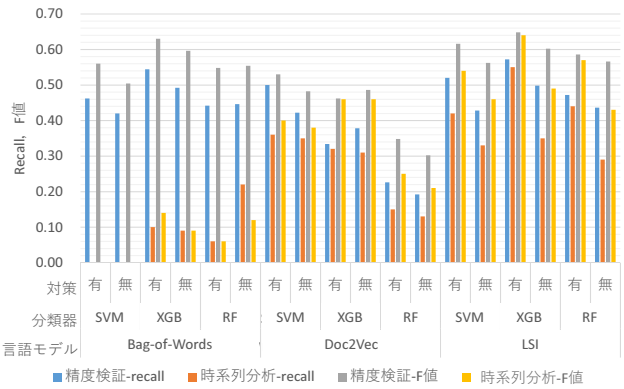

図 2　悪性 PowerShell の検知精度（recall，F 値）

7.2 従来手法との比較

先行研究 [4] では，評価は AUC 及び TPR（recall）を用いてなされた．同じ Bag-of-Words を用いた未知の悪性 PowerShell の分類精度を比較すると，先行研究では TPR が最大 0.87，本研究では recall 値が最大 0.22 という結果となった．しかしながら，本研究における Bag-of-Words は他言語モデルと比較するため次元数が固定されていた．そのため，次元数を固定しない状態であれば，表 10 に示すように最大の recall 値が 0.60 であるため，単純に本提案手法が先行研究に大きく劣るとは言えないことが分かった．また，データセットを比較すると，先行研究では悪性 6290 個，良性 60098 個の計 66388 個の大規模データセットであり，本研究で使用したデータセットは悪性 310 個，良性 5145 個の計 5455 個である．本研究で使用したデータセットの規模は，先行研究で使用されたデータセットの規模の 10%未満であることを鑑みると，同規模のデータセットを用いることでより高い検知率を得ることができると考えられる．

表 10 次元数を固定しない場合の Bag-of-Words の実験結果（難読化対策あり）

分類器	検証法	precision	recall	F 値	accuracy
SVM	5CV	0.81	0.45	0.56	0.94
	未知の分類	0.80	0.42	0.55	0.96
XG Boost	5CV	0.76	0.57	0.65	0.95
	未知の分類	0.80	0.60	0.69	0.97
Random Forest	5CV	0.77	0.57	0.65	0.95
	未知の分類	0.77	0.50	0.61	0.97

7.3 研究倫理

本研究で使用した gensim，scikit-learn 等のモジュールは無償提供されており，コンシューマー用途のコンピュータで使用可能である．また，PowerShell のサンプルの収集はウェブスクレイピングを用いて行ったため，データセットの再現性についても確保できると考える．以上のことから，提案した手法は，一般的なコンピュータでも実装可能であり，再現性が高いといえる．

8. おわりに

本研究では，静的解析のみを用い，単語ベースの言語モデルによって悪性および良性の PowerShell スクリプトから特徴ベクトルを作成し，未知の PowerShell スクリプトを分類する手法を提案した．実験の結果，言語モデルは LSI，分類器は XGBoost の組み合わせが，本稿が提案する手法で最も高い検出率であった．動的解析を用いず，高い再現性を確保できたものの，動的解析を用いる従来手法の精度には至らなかった．今後の課題は次の 5 つである．

1 つ目は，サンプルの入手である．本研究で使用した悪性の PowerShell のサンプルは約 300 個と数が十分ではないため，より多くのサンプルの入手方法の検討が必要である．2 つ目は，マルウェアファミリの調査および新種のマルウェアファミリに対する検知率の調査である．マルウェアファミリの種類を明らかにし，種類ごとの検知率を調査することで検知器の改善に資するものと考える．また，実用に資するものであるか検証するために，新種のマルウェアファミリを検知可能か検証する必要がある．3 つ目は，誤検知の原因の詳細の分析である．4 つ目は，各パラメータの値の調整である．本研究では，言語モデルのサイズを固定したほかは全て各モジュールの初期設定が適用されている．このため，各パラメータを調整することで精度を向上させることができると考える．5 つ目は，静的解析による難読化を含めた前処理の方法の検討である．

参考文献

[1] 株式会社 ICS 研究所 :【産業別】サイバー攻撃事故事例資料, https://www.ics-lab.com/pdf/journal/28/journal-28.pdf (2019)

[2] Symantec :インターネットセキュリティ脅威レポート 第 24 号, p.17 (2019).

[3] McAfee :ファイルレス攻撃解説, https://www.mcafee.com/enterprise/ja-jp/assets/white-papers/fileless-malware-report.pdf (2017).

[4] D. Hendler, S. Kels and A. Rubin :Detecting Malicious PowerShell Commands using Deep Neural Networks, Proceedings of the 2018 on Asia Conference on Computer and Communications Security, pp.187-197 (2018).

[5] G. Rusak, A. Al-Dujaili and U. M. O'Reilly :AST-Based Deep Learning for Detecting Malicious PowerShell, Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp.2276-2278 (2018).

[6] D. Ugarte :Powerdrive, https://github.com/denisugarte/PowerDrive (2019).

[7] D. Ugarte, D. Maiorca, F. Cara, and G. Giacinto :PowerDrive: Accurate De-Obfuscation and Analysis of PowerShell Malware, Proceedings of the DIMVA 2019 16th Conference on Detection of Intrusions and Malware & Vulnerability Assessment, pp.240-259 (2019).

[8] Q. Le, T. Mikolov, : Distributed representations of sentences and documents, Proceedings of the 31st International Conference on International Conference on Machine Learning (ICML'14) Vol32, pp.1088-1196 (2014).

[9] T. Mikolov, K. Chen, G. Corrado and J. Dean :Efficient Estimation of Word Representations in Vector Space, Proceedings of NAACL-HLT, pp.746-751 (2013).

[10] gensim, https://radimrehurek.com/gensim/index.html

[11] scikit-learn, https://scikit-learn.org/stable/

[12] XGBoost, https://xgboost.readthedocs.io/en/latest/index.html

[13] HYBRID ANALYSIS, https://www.hybrid-analysis.com/

[14] GitHub, https://github.co.jp/