情報処理学会論文誌 Vol.62 No.5 1317–1327 (May 2021)

推薦論文

単語ベースの機械学習モデルによる未知の悪性PowerShellスクリプトの検知手法

田尻 裕貴[1,a)] 三村 守[1,b)]

受付日 2020年5月11日, 採録日 2021年2月2日

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脅威となっている．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ベース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，動的解析を用いずに，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，AnyRun および github から入手した良性および悪性のサンプルから作成した．検証実験では，未知のサンプルに対する最大 recall 値は 0.98 となった．また，新種のマルウェアファミリを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，XGBoost

Using a Word-based Machine Learning Model to Detect Unknown Malicious PowerShell Script

Yui Tajiri[1,a)] Mamoru Mimura[1,b)]

Received: May 11, 2020, Accepted: February 2, 2021

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 threat. 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-based convolutional neural network. This method, however, requires dynamic analysis for preprocessing, and thereby requires time. This paper proposes a method to classify unknown PowerShell without dynamic analysis. Our method uses feature vectors extracted from malicious and benign PowerShell scripts using a word-based language model for classification. Our dataset was generated from benign and malicious PowerShell scripts obtained from HybridAnalysis, VirusTotal, and github. Our experiment shows that the maximum recall achieves 0.98 against unknown samples. Furthermore, we confirmed that new malware families could be detected.

Keywords: PowerShell, latent semantic indexing, Doc2Vec, XGBoost

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等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

[1] 防衛大学校情報工学科
Department of Computer Science, National Defense Academy of Japan, Yokosuka, Kanagawa 239–8686, Japan
[a)] em58028@nda.ac.jp
[b)] mim@nda.ac.jp

本論文の内容は 2019 年 10 月のコンピュータセキュリティシンポジウム 2019（CSS2019）で報告され，コンピュータセキュリティ研究会主査により情報処理学会論文誌ジャーナルへの掲載が推薦された論文である．

者の攻撃対象となり，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は年々多様化，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ホワイトペーパによると，サイバー犯罪者は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．このような脅威に対し，機械学習を用いて悪意あるスクリプトを検知する手法が研究されている．しかしながら，悪性 JavaScript [3], [4] や悪性マクロ [5], [6], [7] 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いい難い．

PowerShell 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ている [8]．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．また，Rubin らは，Microsoft が提供する Antimalware Scan Interface（AMSI）をベースに使用し，単語の意味をベクトル表現する技術を用いた悪性 PowerShell の検知手法を提案している [9]．

これらの手法では，前処理に動的解析を用いており，解析にサンドボックス等専用の環境やスクリプトの実行が必要である．また，動的解析を行って PowerShell スクリプトを入手する際に難読化を解除している．本研究では，前処理を含めて動的解析を用いず，難読化が解除されていない状態でも高速にマルウェアを検知する手法を検討する．

近年の自然言語処理技術の発達にともない，マルウェア検知に応用する研究がなされている [10], [11], [12]．しかしながら，悪性 PowerShell の検知に自然言語処理技術を適用したものは少なく，多くはディープラーニングや他の機械学習技術を用いている．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，スクリプトを分割して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，比較的新しい自然言語処理技術である LSI および Doc2Vec を採用した．分類器には，SVM，RandomForest および XGBoost を用いた．

本研究は，先行研究 [13] を基に発展させたものである．本研究では，[13] の課題であったマルウェアファミリの分析を実施し，新種のマルウェアに対する検知精度を明らかにした．また，サンプルのマルウェアファミリの分析の過程で悪性と疑われる，または誤検知の可能性のあるサンプルを除外しデータセットの質を向上させている．さらに，事前実験を実施し，文献 [13] では言及できていなかったコーパスの設定方法を明確にした．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．
(3) 悪性 PowerShell スクリプトのうち新種のマルウェアファミリ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．2 章では PowerShell の概要について述べ，3 章では関連研究を紹介する．4 章では本研究で用いる関連技術を紹介する．5 章では提案手法について説明し，6 章では検証実験の手法およびその結果について述べる．7 章では実験結果について考察する．最後に 8 章では，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．攻撃者は正規のツールを悪用することで，攻撃対象の端末にマルウェアを直接送り込まずにマルウェアをダウンロードまたは作成し，検知を回避しようと試みる．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，攻撃の検出を困難なものとしている．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 1 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

表 1 難読化手法の例
Table 1 Examples of typical obfuscation techniques.

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 ''

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．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ることで，容易に難読化を解除できないようにしている．

3. 関連研究

本章では機械学習を用いた悪意のある PowerShell スクリプトの検知手法に関連する研究を調査し，本研究の新規性を示す．

Hendler らは，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 *Deep/Traditional Models Ensemble*（*D/T Ensemble*）を提案した [8]．*D/T Ensemble* は，ディープラーニングアーキテクチャに基づくモデルおよび伝統的な自然言語処理の検知手法に基づくモデルから，それぞれ検知率が高いモデルを組み合わせた手法である．*D/T Ensemble* が使用するモデルは 4-CNN および 3-gram を組み合わせたものである．使用したディープラーニングアーキテクチャは，文字ベースの畳み込みニューラルネットワーク（CNNs）およびリカレントニューラルネットワーク（RNN）であり，自然言語処理は文字 n-gram および Bag-of-words である．*D/T Ensemble* における前処理では，PowerShell コマンドの難読化への対策として Base64 のデコード，数字の記号（*）への置換，冗長な空白の単純な空白への置換やコマンドの正規化をしている．4-CNN および 3-gram のモデル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とる．評価指標には，適合率（TPR）および偽陽性率（FPR）を用いている．

Rubin らは，Microsoft が提供する Antimalware Scan Interface（AMSI）をベースに使用し，単語の意味をベクトル表現する技術を用いた悪性 PowerShell の検知手法を提案した [9]．AMSI は Microsoft が開発した Windows10 から標準で搭載されているマルウェア対策用インタフェースである．AMSI は，実行されるスクリプトをバックグラウンドで実行して動的解析を行う．そして，解析結果は Windows Defender に送られてスキャンされ，悪意のあるスクリプトを検知する．提案手法では，AMSI の動的解析の結果から得られたスクリプトを利用する．これらのスクリプトおよび単語の意味をベクトル表現する手法である Word2Vec および FastText を使用して単語埋め込みモデルを作成する．単語埋め込みモデルおよびディープラーニングアーキテクチャを使用して分類を行う．

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 PowerShell の検知手法を提案した [14]．この手法は，コードの構造情報を使用したモデルであることを特徴としている．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・コマンドをデコードし，構文解析をして AST に変換してから PowerShell プログラム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．

Ugart らは PowerShell の難読化を解除するツール *PowerDrive* [15] を提案した [16]．*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，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．多層化された難読化解除によりデータセットの 95%が正確に分析されている．

このように先行研究では，スクリプトを自然言語として分析するアプローチの有効性が示されているものの，いずれも動的解析を必要としている．また，マルウェアファミリの分析も十分に実施されておらず，新種のマルウェアファミリに対する検知精度も明らかではない．そこで本研究では，動的解析を用いずに，自然言語ベースの悪性 PowerShell の検知手法を提案する．提案手法では，PowerShell をスクリプト単位で扱うことで，より現実的なマルウェアへの対応を可能とし，コマンドに分ける工程を削減する．さらに，新種のマルウェアファミリに対する検知精度を明らかにする．

4. 関連技術

この章では，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た場合，文書 d は式 (1) で表すことができる．

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

この式 (1) 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連性をベクトル（文書–単語マトリクス）で式 (2) のように表現できる．

$$\hat{d}_i=(n_{w_1},n_{w_2},n_{w_3},\ldots,n_{w_j}) \quad (2)$$

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できる．

4.1.2 TFIDF

TFIDF は，文書に含まれる単語の重要性を評価する手法である．TFIDF は式 (3) のように，Term Frequency（TF）および Inverse Document Frequency（IDF）に基づいて計算される．

$$tfidf=tf*idf \quad (3)$$

TF は，文書内のある単語の出現頻度を表す．TF は式 (4) のように $tf(t,d)$ で表し，t は文書内の単語を，d は文書を意味する．$n_{t,d}$ は文書 d における単語 t の出現回数を，$\sum_{s\in d} n_{s,d}$ は文書 d におけるすべての単語の出現回数の和である．

$$tf(t,d)=\frac{n_{t,d}}{\sum_{s\in d} n_{s,d}} \quad (4)$$

IDF は，ある単語が出現する文書の頻度の逆数である．ある単語 t の IDF は $idf(t)$ で表され，式 (5) で計算される．式 (5) 中の N は文書の総数，$df(t)$ は，ある単語 t が出現する文書の数である．

$$idf(t)=\log\frac{N}{df(t)}+1 \quad (5)$$

4.1.3 Latent Semantic Indexing

Latent Semantic Indexing（LSI）[17] 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て関連性を分析する．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．

この重み付けには，一般的に，式 (6) に示す TFIDF が使用される．式 (6) 中の，$n_{i,j}$ は文書 d_j 内の単語 t_i の出現頻度，$|D|$ は文書の総数，$\{d:d\ni t_i\}$ は，単語 t_i を含む文書の総数を意味する．

$$tf_{i,j}*idf_i=\frac{n_{i,j}}{\sum_k n_{k,j}}*\log\frac{|D|}{\{d:d\ni t_i\}} \quad (6)$$

次に，TFIDF によって重み付けされたベクトルを特異値分解する．文書群の行列 X の要素 $x_{i,j}$ は，文書 d_j 内の単語 t_i の TFIDF を表す．行列の各行は 1 つの文書に対応するベクトルを示し，各要素は各文書との関係を示す．同様に，行列の列はある文書に対応するベクトルを示し，各要素は各単語との関係を示す．X の特異値分解は，直交行列 U および V，対角行列 Σ に分解される．行列式は次式で表す．

$$\begin{aligned}X&=\begin{bmatrix}x_{1,1}\ldots x_{1,j}\\ \vdots\ \ddots\ \vdots\\ x_{i,1}\ldots x_{i,j}\end{bmatrix}\\&=U\Sigma V^T\\&=\begin{bmatrix}u_{1,1}\ldots u_{1,r}\\ \vdots\ \ddots\ \vdots\\ u_{i,1}\ldots u_{i,r}\end{bmatrix}*\begin{bmatrix}\sigma_{1,1}\ldots\ 0\\ \vdots\ \ddots\ \vdots\\ 0\ \ \ldots\sigma_{r,r}\end{bmatrix}*\begin{bmatrix}v_{1,1}\ldots v_{1,r}\\ \vdots\ \ddots\ \vdots\\ v_{j,1}\ldots v_{j,r}\end{bmatrix}\end{aligned} \quad (7)$$

4.1.4 Doc2Vec

Doc2Vec は，Le らによって提案された Paragraph Vector [18] 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 [19] 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，文書中の単語どうし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数値で表すことが可能である．

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 RandomForest 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．

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．

RandomForest は Breiman [20] 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．

XGBoost [21], [22] は，勾配ブースティングと RandomForest を組み合わせたアンサンブル学習である．拡張性があり，大規模データにも対応可能である．また，値の欠損や 0 が多い疎なデータにも対応することができるアルゴリズムを有する．

5. 提案手法

本章では，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 に提案手法の概要を示す．訓練サンプルおよびテストサンプルの PowerShell スクリプトには，それぞれ悪性および良性 PowerShell スクリプトが含まれている．

以下に提案手法の手順を示す．

(1) 訓練サンプルに対して前処理（難読化対策，データクレンジングおよび分かち書き）を実施する．
(2) 前処理を実施した訓練サンプルからコーパスを作成する．
(3) BoW, Doc2Vec および LSI で各言語モデルを作成する．
(4) SVM，XGBoost および RandomForest の 3 種類の分類器を訓練する．
(5) テストサンプルに対し前処理を実施する．
(6) 前処理をしたテストサンプルを各言語モデルに入力する．
(7) 各言語モデルから得た特徴ベクトルを各分類器に入力し，分類を行う．

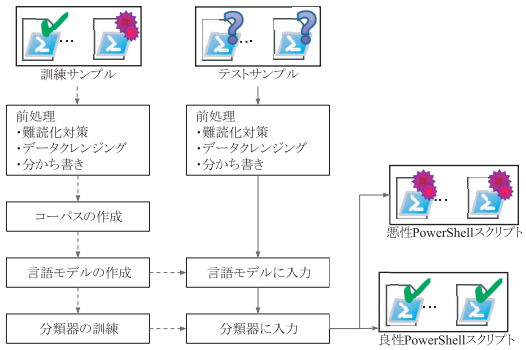

図 1 提案手法
Fig. 1 Overview of the proposed method.

以下，各手順の詳細について述べる．

5.2 サンプルの前処理

サンプルの前処理として，難読化対策，データクレンジングおよび分かち書きを実施する．まず，サンプルに対し難読化対策を実施する．詳細は 5.2.1 項で述べる．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 1 つとするため，**表 2** に示すとおり置換を実施する．加えて，マルチバイト文字列については特徴の 1 つとなるように同一の記号に置換する．その後，正規表現を用いてスクリプトを分かち書きしてコーパスを作成する．分かち書きは，空白文字に加え，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する．

5.2.1 難読化対策

2.2 節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** に示す処理を実施する．PowerShell では大文字と小文字を区別しないため，最後にアルファベットを小文字に統一する．

表 2 特定の文字列の置換
Table 2 Replacing specific patterns.

文字列	置換後文字列
複数行にまたがるコードを 1 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 1 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 1 つにする．
実行ファイル名	置換し特徴の 1 つにする．
マルチバイト文字列	置換し特徴の 1 つとする

表 3 難読化への対策
Table 3 Obfuscation countermeasures.

番号	対応方法	例
複数行にまたがるコードを 1 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+$uuid;
Base64 にエンコードされた文字列	置換し特徴の 1 つにする．	対策前：Base64String(’7b0HYBxJliUmL23K... 9k9/yGyf/Dw==’) 対策後：Base64String(’base64str’)
大文字小文字が混合されているスクリプト	小文字に統一する．	対策前：$mSEnT = nEW-obJEcTsYstEm.io.mEmOrySTReaM 対策後：$msent = new-object system.io.memorystream

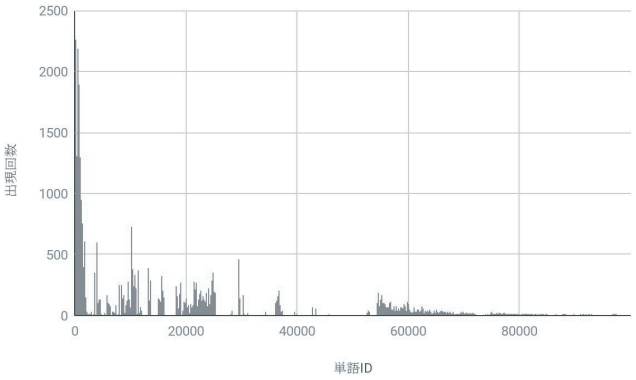

図 2 各単語の出現回数
Fig. 2 Total number of occurrences of each word in the dataset.

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．また，文書全体における出現頻度が高く，どの文書にも出現するような一般的な単語も存在する．このような単語は，文書の特徴の表現に貢献しないことが多いためコーパスから除外する．コーパスから除外する単語を調査するため，本研究で使用するデータセットにおけるこのような単語の出現頻度を調査する．データセット内に存在する単語の出現回数を**図 2** に表す．図 2 から，出現回数が極端に少ない単語と極端に多い単語の種類が非常に多いことが分かる．図 2 に現れるごく稀にしか出現しない単語を出現回数，一般的な単語を全文書における出現頻度を基準に，gensim の辞書フィルタを用いて文書の特徴の表現に貢献しないと考えられる単語を除外し，コーパスを作成する．コーパスの設定は事前実験を実施し，最もその結果が良い設定のコーパスを本実験に使用する．事前実験については 6.3.1 項で詳細を述べる．

5.4 言語モデルの作成

図 1 中における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 BoW 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，言語モデルの特徴ベクトルの次元数は BoW を除き，比較のため 200 に固定した．BoW の特徴ベクトルの次元数は出現する単語数と同じになる．そのため，現実世界に適用した場合，次元数が非常に大きくなるとともに計算コストも増大する恐れがある．そこで，計算コストを抑えるために次元削減を行った場合の言語モデルも作成した．

BoW の次元削減には主成分分析（PCA）を用いた．PCA は機械学習をする際の特徴ベクトルの次元数を削減する一般的な手法の 1 つである．PCA は元の情報を失わないよう，冗長な情報を削減し，特徴的な情報を抽出するため，計算コストの削減に有意と考え，この手法を用いた．

表 4 環境
Table 4 Experiment environment.

CPU	Core i7-7700HQ 2.80 GHz
Memory	8 GB
OS	Windows10 Home
使用言語	Python3.7

表 5 使用した主なライブラリ
Table 5 Main Python libraries used for experiments.

自然言語処理	Bag-of-Words Doc2Vec LSI	gensim -3.8.1 [23]
機械学習	SVM RandomForest XGBoost	sickit-learn -0.22.2.post1 [24] xgboost -1.0.2 [22]

5.5 分類器の訓練

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，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．データセットの詳細については 6.1 節で示す．

5.6 テストサンプルの分類

図 1 中のテストサンプルを，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，テストサンプルを，5.2 節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テストサンプルを入力し，特徴ベクトルに変換する．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 PowerShell スクリプトを検知する．

5.7 実装

提案手法は Python3.7 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 HybridAnalysis [25] から収集した PowerShell スクリプト 589 体，AnyRun [26] から収集した PowerShell スクリプト 355 体，github [27] から入手した良性の PowerShell スクリプト 4,901 体の合計 5,845 体のサンプルから作成した．検体の収集期間は，2019 年 1 月から 2020 年 3 月の間である．

収集方法は，Python を用いたウェブスクレイピングである．HybridAnalysis からは，公開 API を利用して投稿

表 6 主要マルウェアファミリ（上位 10 種類）
Table 6 Top 10 major malware families in the dataset.

マルウェアファミリ名	検体数	悪性検体に占める割合
Presenoker	95	19.8%
Ploty	63	13.1%
Uwasson	26	5.4%
Occamy	25	5.2%
Powersploit	22	4.6%
Injector	22	4.6%
Generic	15	3.1%
PsAttack	15	3.1%
Splitfuse	14	2.9%
Bynoco	14	2.9%

された検体情報のリストを 1 日に 4 回取得し，その情報の中から，検体のファイルタイプが PowerShell である情報の有無を検索した．検索の結果，検体が公開されている PowerShell の情報がある場合，該当する PowerShell スクリプトを，サンプルとして入手した．AnyRun からは，Public submissions からフィルタ機能利用して，拡張子に .ps が含まれるファイルを検索し，手動でサンプルを入手した．github からは，まず公開されているリポジトリのうち，言語に PowerShell が選択されている，または「PowerShell」という単語を含むリポジトリを検索した．検索結果の並べ替え順序はデフォルトの「Best match」とし，上位 1,000 件のリポジトリをダウンロードした．そして，ダウンロードしたリポジトリから，Python を用いて拡張子に.ps が含まれるファイルを検索し，PowerShell ファイルをサンプルとして抽出した．

入手した検体は，VirusTotal [28] を使用して検体の情報を調査した．悪性 PoweShell スクリプトはベンダによって扱いが異なり，ベンダによってはマルウェアとして検知しない場合もある．そのため，主要ベンダ 5 社（Kaspersky，McAfee，Microsoft，Symantec，TrendMicro）のうち 2 社以上がマルウェアと判定した場合にそのサンプルを悪性とした．主要マルウェアファミリ（上位 10 種類）を**表 6** に示す．良性については，マルウェアと判定したベンダが 0 件の場合とし，悪性と良性どちらの条件にもあてはまらない検体は悪性疑いとして除外した．github から収集した検体についても，同様に調査し，マルウェアと判定したベンダが 0 件の検体のみを良性サンプルとして使用した．

本研究で使用するデータセットは，未知の悪性 PowerShell スクリプトを検知可能か検証するためにデータセットを時系列で分割することにした．データセットの分割は 2019 年 6 月以前の検体を既知の PowerShell スクリプト，2019 年 7 月以降の検体を未知の PowerShell スクリプトとして扱う．内訳は**表 7** のとおりである．

なお，良性スクリプトは現実における一般的なアプリケーションであると考えることができる．マルウェアに

表 7 データセット内訳
Table 7 Structure of dataset.

AnyRun, HybridAnalysis			github
データセットの種別	悪性	良性	良性
既知の PowerShell スクリプト	309	232	4,901
未知の PowerShell スクリプト	171	92	

表 8 予測結果と真の結果の関係
Table 8 Confusion matrix.

		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）

は，技術的な動向に応じてトレンドが発生する傾向があるのに対し，良性スクリプトではその傾向は低いと考えられる．そのため，検体を時系列で分割する際に使用した日付は，VirusTotal に投稿された日付と検体を入手した日付の古い方を採用した．しかし，github からは，初投稿日等の時系列分析に必要な時間情報を得ることができなかった．そのため，github から収集した良性のスクリプトについてはランダムに 2 分割した．

6.2 評価指標

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 8** に示す．本研究では，PowerShell を正しく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．

分類問題の評価精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一致しているかを判断する指標であり，式 (8) で定義される．

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{8}$$

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，式 (9) で定義される．

$$Precision = \frac{TP}{TP + FP} \tag{9}$$

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，式 (10) で定義される．

$$Recall = \frac{TP}{TP + FN} \tag{10}$$

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，式 (11) で定義される．

$$F\text{-}measure = \frac{2Recall \times Precision}{Recall + Precision} \tag{11}$$

本研究では，実際の悪性 PowerShell スクリプトを検知する

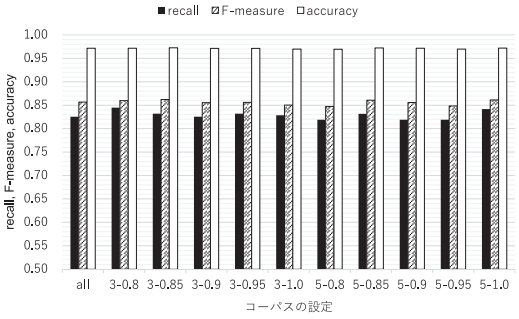

図 3 コーパスごとの 5 分割交差検証の結果

Fig. 3 The performance of 5-fold cross-validation for each corpus.

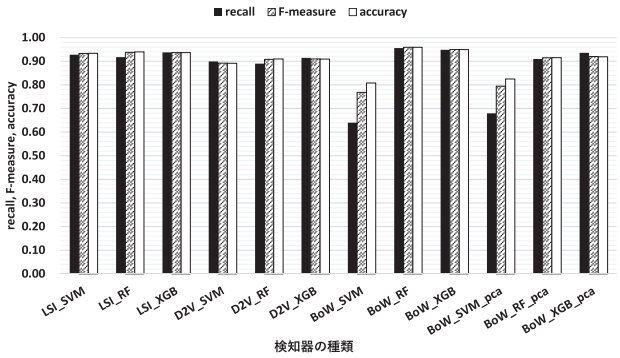

図 4 5 分割交差検証の結果

Fig. 4 The performance of 5-fold cross validation.

という目的から，悪性 PowerShell をどの程度検知できているかに注目する必要がある．したがって，評価指標のうち，再現率（Recall）に焦点を当てる．加えて，モデル全体の性能のバランスを確認するために F 値（F-measure）および正解率（Accuracy）を用いる．

6.3 実験内容

6.3.1 事前実験

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．事前実験では既知の PowerShell スクリプトから複数のコーパスを作成し，LSI と XGBoost を組み合わせた分類器を用いて 5 分割交差検証を実施した．コーパスの設定は，図 2 において出現回数が顕著な単語の数を目安にして設定した．設定は「最小出現回数－最大出現頻度」で表し，最小出現回数（3，5），最大出現頻度（0.8，0.85，0.9，0.95，1.0）の各組合せおよび全単語を使用したコーパス「all」からなる．5 分割交差検証の結果を**図 3** に示す．事前実験の結果，最も適当なコーパスの設定は「3-0.8」となった．本実験では，このコーパスを使用する．

6.3.2 本実験

本研究では，汎化性能の評価と時系列分析の 2 つの実験を実施した．モデルの構築は，既知の PowerShell スクリプトを用いた．言語モデルには LSI，Doc2Vec または BoW を使用し，機械学習モデルには XGBoost，SVM または RandomForest を使用した計 6 種類の組合せを用いた．汎化性能の評価では，既知の PowerShell スクリプトを使用し，5 分割交差検証を実施する．時系列分析では，既知の PowerShell スクリプトで訓練を実施し，未知の PowerShell スクリプトで分類精度を評価する．加えて，新種のマルウェアファミリを検知可能かを調査する．

また，本研究では，データセットが不均衡であるため，ダウンサンプリングを行って均衡なデータセットにして各実験を行った．ダウンサンプリングは，まず各データセットに含まれる良性スクリプトを，それぞれのデータセット内の悪性スクリプトと同数になるようにランダムに分割した．この分割した良性スクリプトと悪性スクリプトを用いて，ダウンサンプリング済みの既知および未知のデータセット群をそれぞれ作成した．なお，分割した際に生じた，端数となった良性サンプルは実験から除外した．ダウンサンプリングした各データセットを用いてそれぞれ実験し，実験の結果は検知器ごとに全データセットの平均をとった．

6.4 実験結果

本節では，汎化性能と未知の PowerShell スクリプトの検知精度を示す．

5 分割交差検証の結果を**図 4** に示す．各指標において最も高い精度であったのは，BoW と RandomForest を組み合わせたモデルであり，recall 値は 0.96 であった．次いで高い精度となったモデルは，BoW と XGBoost の組合せで recall 値が 0.95，LSI と XGBoost の組合せが 0.94 であった．

図 4 において言語モデルに着目すると，興味深い特徴が得られた．言語モデルに BoW を使用しているモデルは，組み合わせる機械学習手法によって精度が大きく変化した．一方で，言語モデルに LSI および D2V を使用したモデルは，組み合わせる機械学習手法が異なっても安定して高い精度を得られた．また，BoW は次元削減を行っていない場合と行った場合を比較した結果，わずかに精度が低下するが大きな変化は見られなかった．

時系列分析の結果を**図 5** に示す．時系列分析の結果においては LSI と XGBoost を組み合わせたモデルが最も高い精度となり recall 値が 0.98 であった．次いで，BoW と RandomForest を組み合わせたモデルが 0.97，LSI と SVM を組み合わせたモデルが 0.96 となった．

次元削減を行った BoW の結果を見ると，時系列分析の結果が著しく低下し，最も精度が低下したモデルでは recall 値が 0 となった．このため，BoW を用いたモデルの精度は，特徴ベクトルの次元数に大きく依存することが分かった．各言語モデルに着目すると，LSI を使用したモデルの精度は，いずれの機械学習手法を使用した場合でも高い結果となった．

これらの結果から，言語モデルと分類器の組合せについ

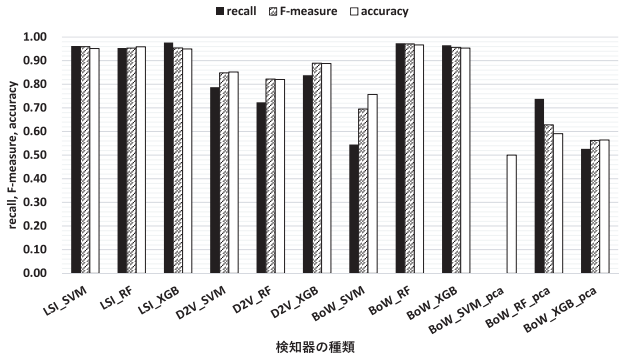

図 5 時系列分析の結果
Fig. 5 The performance of the time series analysis.

ては相性が存在することが分かった．加えて，BoW を使用したモデルは特徴ベクトルの次元数（単語数）に大きく依存するため，性能が不安定である．以上を考慮すると，言語モデルに LSI を使用することで比較的安定した性能を得られることから，LSI が有用であるといえる．

次に，時系列分析の結果から新種のマルウェアファミリの検知の可否を調査した．この調査では，時系列分析において recall 値が高い LSI-XGBoost および BoW-RandomForest の 2 種類のモデルの検知結果を使用した．未知の PowerShell スクリプトに含まれる新種のマルウェアファミリは，15 種類の計 24 体であった．LSI-XGBoost の組合せでは，これら新種のマルウェアファミリ全体のうち 91%を，BoW-RandomForest の組合せではこれら新種のマルウェアファミリ全体のうち 88%を検知した．

したがって，これらのモデルで新種のマルウェアファミリを検知することが可能といえる．

7. 考察

7.1 検知精度

検証実験の結果，5 分割交差検証では 96%のマルウェアが検知可能であった．したがって，提案手法は亜種のマルウェアに対しては有効であると考えられる．さらに，時系列分析では新種のマルウェアファミリの 91%が検知可能であった．よって，提案手法は完全に新種のマルウェアに対してもある程度は有効であると考えられる．ただし，新種の各マルウェアファミリの検体数については 15 種類中 11 種類が 1 件であった．そのため，検知精度には偏りがある可能性がある．そこで，新種のマルウェアファミリのうち最も数が多い Powdown は検体数に注目すると，提案手法では 100%を検知することが可能であった．したがって，提案手法で新種のマルウェアファミリを検知することは可能であると考えられる．

7.2 言語モデルの影響

本実験では LSI が分類器の種類を問わず高精度に検知をすることができた．本結果は，LSI の特徴である「文書と単語の関連性」という要素が貢献していると考えられる．この「文書と単語の関連性」という要素が存在することにより，表記が異なる場合でも類似した意味の単語として言語モデルを構築することが可能であると考えられる．したがって，LSI はいずれの分類器を使用した場合でも，分類器の学習に十分な情報を提供しうる言語モデルであると考えられる．

一方で，BoW は単語の出現頻度，Doc2Vec は文書と単語の前後関係から言語モデルを構築する．このため，分類器によっては言語モデルから与えられる情報では，分類を行うには不足しているのではないかと考えられる．また，時系列分析において Doc2Vec の精度は他の言語モデルに比べて低い結果となった．Doc2Vec の言語モデルの学習は文書と前後の単語に挟まれる単語を予測する形で行われ，特徴ベクトルは単語ではなく文書単位で生成される．特にスクリプト言語は，一般的な文章に比べて記述の自由度が高く，同じ機能を持つスクリプトを作成しても，スクリプトそのものは異なる記述となることがある．そのため，同じ機能を持つスクリプトである場合でも，そのスクリプト間の関連性が低いと学習する可能性があり，低い精度となったと考えられる．

7.3 従来手法との比較

本研究と先行研究 [8] および [9] では，データセット等の実験環境が異なるため，単純に精度を比較することはできない．そのため，あくまでも参考情報として比較結果を示す．

まず，Hendler らが提案した手法 [8] のうち最も精度が高い手法では True Positive Rate（TPR）が最大 0.92 であった．なお BoW を用いた手法では 0.87 という結果であった．そして Rubin らが提案した手法 [9] のうち最も精度が高い手法では，TPR は 0.894 であった．

本論文で提案した手法において，recall 値の最大値は 0.98 であった．本結果は，先行研究と実験条件が異なるため精度を比較できないものの，きわめて良好な結果を示すことができたと考える．一方で，先行研究ではマルウェアファミリの分析が十分に実施されておらず，新種のマルウェアファミリに対する精度は検証されていない．本研究では，マルウェアファミリの詳細を分析し，新種のマルウェアファミリも検知可能であることを示した．

7.4 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ていると考える．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．

7.5 研究の限界

本研究では，各マルウェアファミリに対する検知精度を詳細に分析することができていない．なぜならば，悪性PowerShellの検体数が少なく，マルウェアファミリあたりの検体数が非常に少ないためである．つまり，特定の新種のマルウェアファミリに対する検知精度を検証するには，検体数が不十分である．各マルウェアファミリごとの検体をさらに集めることで，各マルウェアファミリに対する正確な検知精度を得ることが可能と考えられる．

8. おわりに

本研究では，動的解析を用いずに，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．また，新種のマルウェアファミリに対する検知の可否を評価した．検証実験の結果，言語モデルにLSIを使用したモデルでは，機械学習の手法を問わず高い検出精度が得られた．さらに，新種のマルウェアファミリに対する検知の可否を検証し，検知可能であることを示した．

今後の課題としては，より多くのサンプルを入手することがあげられる．本研究で使用した悪性のPowerShellのサンプルは約500体と数が十分ではないため，マルウェアファミリごとの検知率や，誤検知の原因を分析するための特徴等が適切に分布しているとは言い難い．より多くのサンプルを入手することで提案手法の特徴を明らかにし，モデルを改善することが可能となるものと考えられる．

さらに，前処理において難読化に用いられる手法の1つであるInvoke-Expressionへの対策の検討があげられる．Invoke-Expressionは，文字列をコマンドとして実行するコマンドである．難読化に用いられる場合，実行するコマンドの文字列部分が動的に生成されるため，静的な手法を用いて最終的に生成されるコマンドを得ることは困難である．この問題に対し，有効な対策を実装することができれば，よりサンプルごとの特徴を明確にし，精度の向上につながるものであると考える．

参考文献

[1] 株式会社ICS研究所：サイバー攻撃の事例集，入手先 ⟨https://www.ics-lab.com/pdf/journal/28/journal-28-20200127.pdf⟩（参照 2020-03-30）．

[2] Symantec: Symantec 2019 Internet Security Threat Report (2019), available from ⟨https://docs.broadcom.com/docs/istr-24-2019-en⟩.

[3] Mimura, M. and Suga, Y.: Filtering Malicious JavaScript Code with Doc2Vec on an Imbalanced Dataset, *2019 14th Asia Joint Conference on Information Security* (*AsiaJCIS*), pp.24–31 (online), DOI: 10.1109/AsiaJCIS.2019.000-9 (2019).

[4] Ndichu, S., Kim, S., Ozawa, S., Misu, T. and Makishima, K.: A machine learning approach to detection of JavaScript-based attacks using AST features and paragraph vectors, *Applied Soft Computing*, Vol.84, p.105721 (online), DOI: 10.1016/j.asoc.2019.105721 (2019).

[5] Miura, H., Mimura, M. and Tanaka, H.: Macros Finder: Do You Remember LOVELETTER?, *Information Security Practice and Experience – 14th International Conference, ISPEC 2018*, pp.3–18 (online), DOI: 10.1007/978-3-319-99807-7_1 (2018).

[6] Mimura, M. and Miura, H.: Detecting Unseen Malicious VBA Macros with NLP Techniques, *JIP*, Vol.27, pp.555–563 (online), DOI: 10.2197/ipsjjip.27.555 (2019).

[7] Mimura, M. and Ohminami, T.: Towards Efficient Detection of Malicious VBA Macros with LSI, *Advances in Information and Computer Security – Proc. 14th International Workshop on Security, IWSEC 2019*, Attrapadung, N. and Yagi, T. (Eds.), Lecture Notes in Computer Science, Vol.11689, pp.168–185, Springer (2019).

[8] Hendler, D., Kels, S. and Rubin, A.: Detecting Malicious PowerShell Commands using Deep Neural Networks, *Proc. 2018 on Asia Conference on Computer and Communications Security, AsiaCCS 2018*, pp.187–197 (online), DOI: 10.1145/3196494.3196511 (2018).

[9] Rubin, A., Kels, S. and Hendler, D.: AMSI-Based Detection of Malicious PowerShell Code Using Contextual Embeddings, arXiv e-prints, arXiv:1905.09538 (2019).

[10] Ito, R. and Mimura, M.: Detecting Unknown Malware from ASCII Strings with Natural Language Processing Techniques, *2019 14th Asia Joint Conference on Information Security* (*AsiaJCIS*), pp.1–8 (online), DOI: 10.1109/AsiaJCIS.2019.00-12 (2019).

[11] Nagano, Y. and Uda, R.: Static analysis with paragraph vector for malware detection, *IMCOM*, p.80, ACM (2017) (online), available from ⟨http://dl.acm.org/citation.cfm?id=3022306⟩.

[12] Wang, J., Ma, S., Zhang, Y., Li, J., Ma, Z., Mai, L., Chen, T. and Gu, D.: NLP-EYE: Detecting Memory Corruptions via Semantic-Aware Memory Operation Function Identification, *22nd International Symposium on Research in Attacks, Intrusions and Defenses, RAID 2019*, pp.309–321 (2019) (online), available from ⟨https://www.usenix.org/conference/raid2019/presentation/wang-0⟩.

[13] Tajiri, Y. and Mimura, M.: Detection of Malicious PowerShell Using Word-Level Language Models, *Advances in Information and Computer Security – Proc. 15th International Workshop on Security, IWSEC 2020*, Aoki, K. and Kanaoka, A. (Eds.), Lecture Notes in Computer Science, Vol.12231, pp.39–56, Springer (online), DOI: 10.1007/978-3-030-58208-1_3 (2020).

[14] Rusak, G., Al-Dujaili, A. and O'Reilly, U.: AST-Based Deep Learning for Detecting Malicious PowerShell, *Proc. 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018*, pp.2276–2278 (online), DOI: 10.1145/3243734.3278496 (2018).

[15] PowerDrive, available from ⟨https://github.com/denisugarte/PowerDrive⟩.

[16] Ugarte, D., Maiorca, D., Cara, F. and Giacinto, G.: PowerDrive: Accurate De-obfuscation and Analysis of PowerShell Malware, *Proc. 16th International Conference Detection of Intrusions and Malware, and Vulnerability Assessment, DIMVA 2019*, pp.240–259 (online), DOI:

10.1007/978-3-030-22038-9_12 (2019).
[17] Deerwester, S.C., Dumais, S.T., Landauer, T.K., Furnas, G.W. and Harshman, R.A.: Indexing by Latent Semantic Analysis, *JASIS*, Vol.41, No.6, pp.391–407 (online), DOI: 10.1002/(SICI)1097-4571(199009)41:6<391::AID-ASI1>3.0.CO;2-9 (1990).
[18] Le, Q.V. and Mikolov, T.: Distributed Representations of Sentences and Documents, *Proc. 31th International Conference on Machine Learning, ICML 2014*, pp.1188–1196 (2014) (online), available from ⟨http://proceedings.mlr.press/v32/le14.html⟩.
[19] Mikolov, T., Chen, K., Corrado, G. and Dean, J.: Efficient Estimation of Word Representations in Vector Space, *Proc. 1st International Conference on Learning Representations, ICLR 2013* (2013) (online), available from ⟨http://arxiv.org/abs/1301.3781⟩.
[20] Breiman, L.: Random Forests, *Mach. Learn.*, Vol.45, No.1, pp.5–32 (online), DOI: 10.1023/A:1010933404324 (2001).
[21] Chen, T. and Guestrin, C.: XGBoost: A Scalable Tree Boosting System, *Proc. 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp.785–794 (online), DOI: 10.1145/2939672.2939785 (2016).
[22] XGBoost, available from ⟨https://xgboost.readthedocs.io/en/latest/index.html⟩.
[23] gensim, available from ⟨https://radimrehurek.com/gensim/index.html⟩.
[24] scikit-learn, available from ⟨https://scikit-learn.org/stable/⟩.
[25] HYBRID ANALYSIS, available from ⟨https://www.hybrid-analysis.com/⟩.
[26] AnyRun, available from ⟨https://app.any.run/⟩.
[27] GitHub, available from ⟨https://github.co.jp/⟩.
[28] Virus Total, available from ⟨https://www.virustotal.com/⟩.

推薦文

PowerShell を用いた攻撃が増加しているため，重要な研究対象である．複数の自然言語処理方法と機械学習の組合せにおける性能を比較しており，データセットもオープンかつ合理的な選定をしている．難読化解除等の細かい工夫も含めて 1 つのシステムとして実装・評価までできており，論文も明快に書かれている．以上のことから，推薦論文に値するため，推薦いたします．

（コンピュータセキュリティ研究会主査　山内 利宏）

田尻 裕貴

2015 年岐阜大学工学部応用情報学科卒業．同年海上自衛隊入隊．2021 年防衛大学工理工学研究科前期課程修了．航空プログラム開発隊勤務．

三村 守　（正会員）

2001 年防衛大学校情報工学科卒業．同年海上自衛隊入隊．2008 年防衛大学校理工学研究科前期課程修了．2011 年情報セキュリティ大学院大学博士後期課程修了．博士（情報学）．同年情報セキュリティ大学院大学客員研究員．2011～2013 年の間内閣官房情報セキュリティセンター出向．2015 年内閣サイバーセキュリティセンター併任，2017 年から防衛大学校准教授．