Paragraph Vectorを用いたマルウェアの亜種推定法

佐藤 拓未[1]　後藤 滋樹[1]　武部 嵩礼[1]

概要：マルウェアの亜種はツールを用いることで容易に作成可能である．亜種の発見数が顕著に増加している．マルウェアの種別に応じた適切な対策のためには，効率よく亜種の判定ができることが望まれる．本研究は，FFRI Dataset のマルウェアの動的解析の結果に含まれる API コール群を用いて，マルウェアが指定の亜種であるかどうかを推定する．具体的には，マルウェアが呼び出す API 群およびその引数に着目し，それらを自然言語の一連の文章のように捉えて Deep Learning の技術を用いて Paragraph Vector を作成する．自然言語における文脈の特徴に相当する情報を動的解析から導く．提案手法により人間による特徴の選定および抽出を行わない自動的なマルウェアの亜種判定を実現する．

キーワード：亜種マルウェア, 動的解析, API コール, Deep Learning, Paragraph Vector

Detectiong Malware Variants using Paragraph Vector

Takumi Sato[1]　Shigeki Goto[1]　Takanori Takebe[1]

Abstract: Subspecific malware, or a malware variant, can be easily made by tools. The number of subspecific malware is increasing remarkably. For appropriate countermeasures for a malware family, it is desirable to realize an effective method for deciding subspecific malware in a family. This paper proposes a new method for determine malware variants. We use log files of dynamic analysis of malware in FFRI Dataset. We focus on API calls and their arguments. Our method treat them as a natural language sentence. We apply Deep Learning technology, and convert log files into Paragraph Vectors. The paragraph vector represents a feature vector, which characterize malware variants. The proposed method does not depend on manual extraction of malware features by human observation.

Keywords: Subspecific malware, Dynamic Analysis, API Call, Deep Learning, Paragraph Vector

1. はじめに

1.1 背景：マルウェアの亜種

マルウェアの発見数は増加の一途を辿っている．大手セキュリティベンダである McAfee のレポート [1] によれば，新しく発見されたマルウェアの種類は 2016 年第 1 四半期だけで約 4000 万種類にものぼる．

マルウェアの発見数が膨大な数になる原因の 1 つとして，**亜種**の存在がある．ある既存のマルウェアに対して，大筋の動作は変更せず，一部に変更を加えたものをそのマルウェアの亜種と呼ぶ．亜種はツールを使うことによって容易に作成が可能であり，これがマルウェアの発見数の増加につながっていることは容易に想像できる．2015 年の G DATA SECURITY のレポート [2] によるとマルウェアの亜種が 1 分間に 12 種類という速度で新しく生まれている．このようにマルウェアの亜種が大量に生まれている現在の状況において，あるマルウェアがどのマルウェアの亜種であるのかを判別することができれば，対象に応じた対策をとることができる．

1.2 方法：機械学習

前項で述べたとおり，マルウェアの増加に対して亜種の推定を行うことが望まれる．既存研究には機械学習を用いるものがあるが，そこで使われる特徴量の選定には人手を

[1] 早稲田大学
Waseda University

介在させるものが多い．一方で，マルウェアの解析方法には表層解析，静的解析，動的解析がある．その中でも動的解析はマルウェアを実行する単純な方法であるが，実際の挙動という特徴的な情報を取り出すことが可能である．

本論文は人手による特徴量の選定が不要な方法を提案する．選定された特徴量を用いて機械学習を行い，マルウェアがどのマルウェアの亜種であるかを判別する．本論文はマルウェアの動的解析の結果の中の API 呼び出しに注目し，Deep Learning による自然言語の解析に用いられている Paragraph Vector を応用して API 呼び出しの一連の流れを特徴量として扱う．つまりプログラムの動的解析の結果を，あたかも自然言語の文章のように取り扱って，マルウェアの亜種判別を行う．

以下に本論文の構成を示す．まず，2 章で本論文で用いる手法である Paragraph Vector について説明する．次に，3 章では既存研究を示す．4 章で本論文の提案手法について述べ，5 章にて提案手法を用いた評価実験の概要と結果を示す．最後に 6 章でまとめを述べる．

2. Paragraph Vector

本論文で用いる Paragraph Vector について説明する．Paragraph Vector は Le と Mikolov [8] によって提案された自然言語処理における Deep Learning の手法であり，ニューラルネットワークに基づく．

1 つの文章を文中の単語を用いてベクトルとして定義することで，意味の近い文章がベクトル空間でも近くに位置するという手法である．なお，この手法は Word Vector [9] と呼ばれる単語の意味をベクトルとして解釈する手法を拡張したものである．以下に Word Vector のアルゴリズムにおいて重要な役割を果たす CBOW と Skip-gram モデルおよび Paragraph Vector のアルゴリズムにおいて重要な PV-DM と PV-DBOW モデルについて述べる．

2.1 CBOW モデル

Word Vector における CBOW モデルおよび後述する Skip-gram モデルは [9] で Word Vector を計算する際に用いられているモデルである．まず，CBOW (Continuous Bag-of-Words) モデルを説明する．Bag-of-Words とは文章における単語の出現回数を要素とするベクトル表現である．例えば A, B, C, D の 4 語のみが存在する文章群において，“A, B, C, B” というという文章は { 1,2,1,0} と表せる．この表現は各単語が含まれているかどうかのみを表し，その順序を考慮することができない．また，N-gram の出現回数に対して Bag-of-Words を適用したものを Bag-of-n-gram と呼ぶ．この場合ベクトルの各要素が n-gram の出現回数となる． CBOW モデルのニューラルネットワークの各層を図 1 に示す．CBOW モデルは前後の文脈から出現する単語を予測するモデルである．ある文

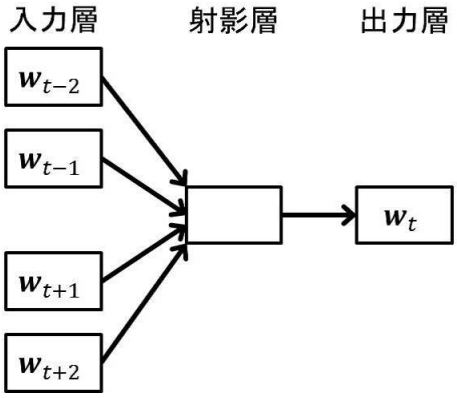

図 1 CBOW モデル

脈で t 番目に出現する語 w_t について，その前後の $2k$ 語 $(w_{t-k}, ..., w_{t-1}, w_{t+1}, ..., w_{t+k})$ の文脈の Bag-of-Words をニューラルネットの入力として，w_t が出力となるように Word Vector を Deep Learning で学習していく．図に示した射影層は，ニューラルネットワークにおける中間層であり，入力層から中間層，中間層から出力層への伝達の際の重みが学習によって最適な値に変化する．Word Vector の学習では，どの語に対しても同じ中間層を用いる．

2.2 Skip-gram モデル

Skip-gram モデルのニューラルネットワークの各層を図 2 に示す．Skip-gram モデルは CBOW モデルと逆向きの予測をするモデルである．つまり，ある文脈で t 番目に出現する語 w_t について，w_t を入力として，その前後の語である $w_{t-k}, ..., w_{t-1}, w_{t+1}, ..., w_{t+k}$ を出力できるように Deep Learning で Word Vector を学習していく．CBOW モデルと Skip-gram モデルの比較が Mikolov [9] に述べられている．Skip-gram モデルの方が語の予測精度が高いという結果が出ている．

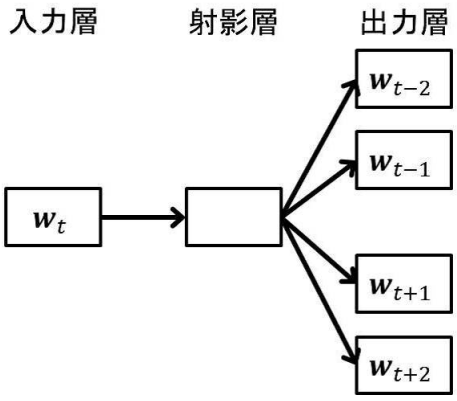

図 2 Skip-gram モデル

2.3 PV-DM モデル

PV-DM (Distributed Memory version of Paragraph

Vector) モデルは前述の CBOW モデルを Paragraph Vector の学習をするために拡張したものである．基本的な構造は CBOW モデルと似ている．相違点は入力層への入力に Paragraph Vector が加わる点である．CBOW モデルにおける入力となっている前後の文脈の $2k$ 語に加えて，Paragraph Vector を入力とする．出力層では w_t が推定できるように Deep Learning で学習する．この時に，学習は Paragraph Vector に対して行っているため，Word Vector は値を固定して学習を行う．

2.4 PV-DBOW モデル

PV-DBOW (Distributed Bag of Words version of Paragraph Vectors) モデルは Skip-gram モデルと類似している．Skip-gram モデルでは文脈の前後の語を推定したい語 w_t を入力としていたが，PV-DBOW モデルでは Paragraph Vector を入力として与え，文中の語 $w_{t-k}, ..., w_t..., w_{t+k}$ が出力層で推定されるように Deep Learning で Paragraph Vector を学習していく．PV-DM モデルと PV-DBOW モデルの比較については Le と Mikolov [8] では説明されていない．双方のモデルから学習したベクトルの連接 (concatenation) を用いることを推奨している．

3. 既存研究

亜種マルウェアの推定を目的とする既存研究が幾つかある．

中村 [11] は本研究の対象と同じ API コールに着目し，API コールの N-gram のディリクレ分布をもとに，Kullback-Leibler 情報量を用いて亜種マルウェアの推定を行っている．

堀合 [12] はマルウェアの動的解析の前後での解析環境のログの差分を マルウェアの挙動を表す情報としてベクトル表現に変換して，ベクトルのハミング距離を計算して距離の近いものを亜種であると判定するマルウェアの亜種推定法を提案している．

我々は次の点に留意した．N-gram や Bag-of-N-gram の手法では API コール列中の一部分のみの順序と出現回数に着目しており，文脈全体における呼び出される位置について考慮していない．本論文で提案する手法では，Paragraph Vector を用いることによって API コール列全体を一連の文章として見ている．このようにして文脈における出現位置を考慮した特徴ベクトルの計算が可能となる．さらに本論文の手法では，API コール列をそのまま Paragraph Vector の計算に用いて特徴ベクトルとする．人手による技巧的な処理を必要としない．さらに，本論文では既存研究では触れられることの少なかった API の引数にも着目して，API の引数名および引数値を文脈に含めて扱う．この効果を評価実験で明らかにする．

4. 提案手法

4.1 提案手法の手順

本論文は動的解析に含まれる API コール列を Paragraph Vector で表して，マルウェアの亜種判定が可能となる特徴を作成する．Deep Learning を応用して，人手による特徴抽出を行わずに自動的に特徴量を作成できる．提案手法は次の 3 つの手順からなる．API コール列の作成, Paragraph Vector の作成，亜種であるか否かの判定の 3 段階である．

4.2 API コール列の作成

API コール列を次のように作成する．動的解析の結果から，API の関数名およびその引数を実行された順番に並べる．これが API コール列となる．ここで，引数の値にはヌル文字のように印字不可能なものも含まれる．以後の処理のために前処理を施して印字可能にする．具体的には，もし引数の値が設定されていない場合は “NULL” に置換する．引数の文字列中の印字不可能な文字列に対応できるように文字列を hex エンコードする．図 4.1 から図 4.3 には，同じ検体について API 名のみ取り出した場合，API 名・API の引数名・API の引数値を取り出した場合，API 名・API の引数値の長さを取り出した場合の例をそれぞれ示す．ここで引数値の長さに着目する理由は，引数の値自体に特徴が現れる場合に長さを用いると特徴が弱まってしまうが，Paragraph 中に出現する単語数を減少させることができる．これにより Word Vector を計算する際のコストを抑えられるからである．

4.3 Paragraph Vector の作成

前項で作成した API コール列を Paragraph Vector へ変換する方法を述べる．まず Paragraph (API コール列) に現れるすべての Word (API 名，API の引数名，引数値) について Word Vector を求める．求めた Word Vector と Paragraph (API コール列) を用いて Paragraph Vector を計算する．対象とする Paragraph の長さはまちまちであるが，Paragraph Vector は一定の次元数に揃う．

4.4 亜種であるか否かの判定

前項で計算したマルウェアの API コール列を Paragraph Vector 化したものを特徴ベクトルとして，亜種であるか否かを判別したいファミリーの検体とそれ以外のファミリーの検体の 2 種類のラベル付けを行う．ラベル付けした特徴ベクトルを教師データとして入力して後述する学習器に学習させる．学習の後に，亜種かどうかを判定したいマルウェアの Paragraph Vector をテストデータとして学習済みの学習器に入力して，マルウェアが亜種であるか否かを

LdrGetDllHandle LdrLoadDll NtDelayExecution LdrLoadDll NtProtectVirtualMemory NtFreeVirtualMemory NtFreeVirtualMemory NtFreeVirtualMemory NtFreeVirtualMemory NtProtectVirtualMemory NtFreeVirtualMemory LdrLoadDll LdrLoadDll LdrLoadDll LdrLoadDll LdrLoadDll LdrLoadDll LdrLoadDll LdrLoadDll NtProtectVirtualMemory ...

図 3 API 名のみ取り出した場合

LdrGetDllHandle ModuleHandle 30783030316639376338 FileName 3078623234376239663372e646c6c LdrLoadDll Flags 31363335333434 BaseAddress 30783735376430303030 FileName 6b65726e656c33322e646c6c NtDelayExecution Status 536b6970706564 Milliseconds 3736 ...

図 4 API 名・引数名・引数値を取り出した場合

LdrGetDllHandle 10 14 LdrLoadDll 7 10 12 NtDelayExecution 7 2 LdrLoadDll 7 10 8 NtProtectVirtualMemory 10 10 10 10 10 NtFreeVirtualMemory 10 10 10 10 NtFreeVirtualMemory 10 10 10 10 NtFreeVirtualMemory 10 10 10 10 NtFreeVirtualMemory 10 10 10 10 NtProtectVirtualMemory 10 10 10 10 10 ...

図 5 API 名・引数値の長さを取り出した場合

判定する．

本論文では教師あり学習器として SVM とランダムフォレストを採用して比較した．ここで注意するべき点は，ランダムフォレストは特徴ベクトルから特徴をランダムに選択して分類器を作成する．一方で Paragraph Vector は文章自体を特徴ベクトルとする．この両者を併用すると相互の特徴が発揮されるのか、それとも相互の特徴が打ち消し合ってしまうのかを調べることは有意義である．手法の一連の手順を図 6 に示す．まず，動的解析結果から API コール列を抽出する．次に API コール列を用いて Deep Learning を行い，Word Vector および Paragraph Vector を求める (図中の学習 1)．そして Paragraph Vector にラベル付けしたものを教師あり学習器である SVM およびランダムフォレストにそれぞれ学習させ分類器を作成し (図中の学習 2)，テストデータを入力しマルウェアの亜種判定を行う．

5. 評価実験

5.1 実験に使用するデータ

5.1.1 FFRI Dataset

今回の実験には FFRI Dataset 2013, 2014, 2015, 2016 [4], [5], [6], [7] に含まれるマルウェアの動的解析の結果を用いた．このデータセットは株式会社 FFRI が独自に収

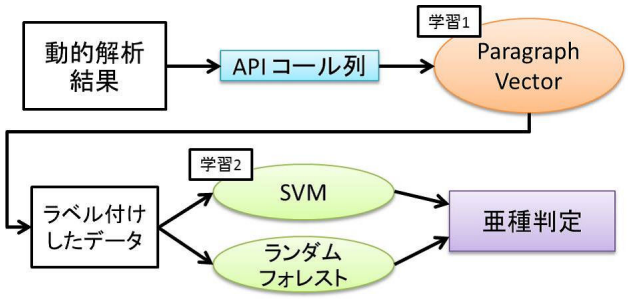

図 6 提案手法の概要図

[Prefix:]Behaviour.Platform.Name[.Variant]

図 7 Kaspersky の命名規則

集した PE (Portable Executable) 形式かつ Windows プラットフォーム上で実行可能なマルウェアの動的解析結果である．この動的解析はオープンソースのマルウェア解析ツールである Cuckoo Sandbox [13] を用いて行い，1 検体あたり 90〜120 秒間 (データセットによって異なる) 実行した結果が json 形式でデータセットに記載されている．FFRI Dataset 2014 には Cuckoo Sandbox での解析結果に加えて，FFRI 社のマルウェア自動解析ツールである FFR yarai analyzer Professional [14] を用いて動的解析を行った結果も含まれるが，本論文では解析方法の統一のため Cuckoo Sandbox での動的解析結果を使用した．また，FFRI Dataset 2016 には Windows 10 と Windows 8.1 での動的解析結果が含まれている．本論文の実験では Windows 10 での動的解析結果を使用した．

本研究では動的解析結果中の API 呼び出しに着目する．そこで，データセットに含まれる 16887 検体のうち，1 つ以上の API 呼び出しログが取得できている 16759 検体を実験に用いた．なお，本研究におけるマルウェアの名称は Kaspersky の検知結果を使い，亜種推定の実験における正解を Kaspersky の分類とした．Kaspersky の命名規則は文献 [3] にある．フォーマットを図 5.1.1 に示す．Prefix と Variant は必須な部分ではなく，存在しないこともある．Prefix はマルウェアを検知したサブシステムを，Variant は亜種をそれぞれ示す．Bahaviour は検出されたマルウェアが何をするか動作を表現し，Platform は実行された環境，Name はマルウェアのファミリー名を指す．本論文では Behaviour から Name までが一致しているマルウェア同士を亜種であると定義する．ここで Kaspersky を採用した理由は，検知率の高さから先行研究 [10], [11] において既に利用されているからである．

5.1.2 サンプリングしたデータセット

本論文では，マルウェアがあるマルウェアの亜種であるか否かを判定するために，データセット中に亜種が多く含まれているファミリーを実験に用いる．FFRI Dataset 2013 – 2016 において，亜種の数が 100 検体以上あるファミリー 24

種類を表 1 に示す．表 1 に挙げられているもののうち，Tro-

表 1 主要な検体名とデータセットに含まれる数

検体名	検体数
Trojan.Win32.Generic	2215
DangerousObject.Multi.Generic	801
Trojan.Win32.Waldek	687
Trojan-Spy.Win32.Zbot	581
Trojan.Win32.Yakes	577
Backdoor.Win32.Androm	445
Trojan.Win32.Agent	384
Trojan.Win32.Inject	351
Worm.Win32.WBNA	308
Hoax.Win32.ArchSMS	304
Trojan-PSW.Win32.Fareit	272
Trojan-PSW.Win32.Tepfer	218
Backdoor.Win32.DarkKomet	206
Trojan-Ransom.Win32.Foreign	204
Trojan-Dropper.Win32.Injector	185
Trojan.Win32.Jorik	171
Packed.Win32.Tpyn	163
Trojan.Win32.Kovter	145
Trojan.Win32.Scar	127
Downloader.Win32.LMN	123
Worm.Win32.Vobfus	120
Backdoor.Win32.Matsnu	110
Trojan-Downloader.Win32.Upatre	108
Trojan.Win32.Llac	100

jan.Win32.Generic，DangerousObject.Multi.Generic の 2 種類は，Generic という名前の通り特定の種類ではなく「その他」として扱われているため，今回の亜種判定の実験の対象から除く． さらに，正確な精度を算出するために，亜種かを判定する検体とそれ以外の検体の数が等しくなるようにランダムにサンプリングしたデータセットを作成する．例えば表 1 において，前述の 2 種類の「その他」を除くとファミリーは 22 種類存在する．Trojan.Win32.Waldek について判定するためのデータセットとして Trojan.Win32.Waldek とそれ以外の 21 種類のファミリーに属するマルウェアが 21:1 の比率で存在するデータセットを作成して，学習する際の教師データおよびテストデータとして用いる．

5.2 実験の方法

実験の手順を以下に示す．まず，対象の 16759 検体の動的解析の結果から，API 名・API の引数名・API の引数値をすべて抽出して，Word Vector を作成するためのコーパスを作成する．今回のコーパスは以下に示す 3 通りを作成し，それぞれのコーパスを用いて Paragraph Vector を作成して，マルウェアの亜種推定を行う手法を手法 1，手法 2，手法 3 と呼ぶ．name は API の引数名，value は API の引数値を hex エンコードしたもの，len(value) は API の引数値を文字列として見た時の長さをそれぞれ表す．次

手法 1. API1 API2 API3 ...
手法 2. API1 name1 value1 API2 name2 value2 ...
手法 3. API1 len(value1) API2 len(value2) ...

図 8 各手法における API コール列の抽出形式

に，作成したコーパスから Word Vector を作成する．さらに，コーパスと作成した Word Vector を用いてコーパス中の各 Paragraph (API コール列) の Paragraph Vector を計算する．Paragraph Vector の計算には sentence2vec [15] を用いた．作成した全検体分の Paragraph Vector のうち，表 1 で示したファミリーから名前が Generic であるものを除いた 22 ファミリーに属するマルウェアの Paragraph Vector を用いる．前述したように亜種判定したいファミリーとそれ以外の 21 ファミリーの比率が 21:1 になるよう Paragraph Vector をランダムに抽出し，データセットを作成する．ファミリーごとにデータセット作成を行うため，合計 22 種類のデータセットができることになる．各データセットの Paragraph Vector に 亜種と判定したいファミリーのであるか，そうでないかを 1 または −1 のラベルで設定する．このように作成した教師データを学習器 (本実験では SVM とランダムフォレストを使用する) に入力して学習させる．学習の終わった学習器にテストデータを入力し，亜種であるか否かの二値分類を行い，判定の精度を測定する．Word Vector および Paragraph Vector は 100 次元で作成した．亜種推定は検体数の都合で 4 分割交差検証を行った．SVM は Libsvm [16] を使用し，SVM のパラメータはグリッドサーチにより最適な数値を求めた．ランダムフォレストは scikit-learn [17] 内のランダムフォレストの実装を使用し，木の数を 4000，各木の特徴数は 20 とした．

5.3 実験結果と考察

最初に実験結果を述べる．各マルウェアファミリーごとに亜種であるかを判定した SVM による精度を表 2 および図 9 に示す．手法 1 から 3 については図 8 で示したとおり，手法 1 は API 名のみを抽出したもの，手法 2 は API 名・引数名・引数値を抽出したもの，手法 3 は API 名・引数値の長さを抽出したものに、それぞれ提案手法を適用したものである．ここで精度というのは，正しく亜種であるもしくは亜種でないと判定できたマルウェアの割合である．

表 2 を見ると，全体的に高い精度で判定ができている．90% 以上の精度で判定できているファミリーも多くある．特に Worm.Win32.Vobfus や Hoax.Win32.ArchSMS のファミリーは 94〜96% の精度で正しく亜種判定ができ

表 2 手法ごとの SVM による精度

ファミリー名＼手法 [%]	手法 1	手法 2	手法 3
Trojan.Win32.Jorik	90.48	89.29	89.29
Worm.Win32.WBNA	91.87	92.26	92.66
Trojan.Win32.Agent	72.02	73.36	70.68
Worm.Win32.Vobfus	94.05	94.05	92.26
Trojan.Win32.Yakes	81.25	78.47	80.56
Trojan-PSW.Win32.Tepfer	75.60	71.43	78.27
Trojan-Spy.Win32.Zbot	74.80	72.72	76.39
Trojan-Dropper.Win32.Injector	70.24	69.35	69.05
Hoax.Win32.ArchSMS	94.84	96.43	95.24
Trojan.Win32.Inject	69.64	73.81	72.77
Trojan-Ransom.Win32.Foreign	86.31	82.14	85.12
Trojan.Win32.Scar	75.00	75.60	77.38
Trojan.Win32.Llac	76.19	70.24	72.62
Backdoor.Win32.DarkKomet	81.25	76.19	79.46
Downloader.Win32.LMN	89.29	93.45	92.26
Backdoor.Win32.Androm	79.88	80.12	80.95
Trojan-PSW.Win32.Fareit	79.37	79.17	77.18
Trojan-Downloader.Win32.Upatre	76.79	80.95	82.74
Packed.Win32.Tpyn	79.76	75.00	79.17
Trojan.Win32.Kovter	90.48	89.88	88.69
Trojan.Win32.Waldek	91.59	88.84	90.55
平均値	81.94	81.08	82.06
最高値	94.84	96.43	95.24
最低値	69.64	69.35	69.05

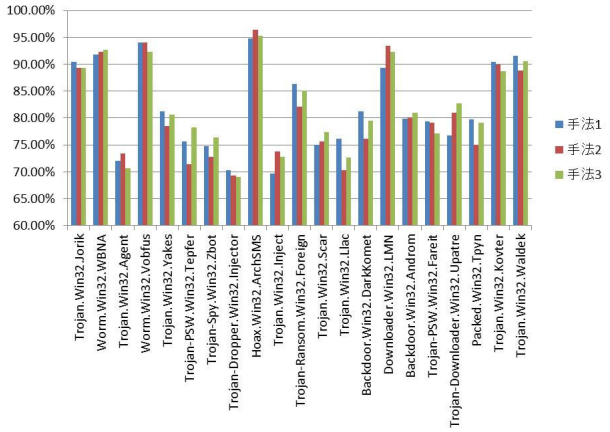

図 9 SVM による亜種推定結果 (Accuracy)

ている．提案手法の Paragraph Vector によるマルウェアファミリーの特徴の表現が有効であることを示している．一方で，命名規則の “Behaviour” が Trojan-XX であるマルウェアファミリーの多くが 75% 前後の精度となった．提案手法では判別の精度が悪いファミリーもある．

手法 1, 2, 3 による精度の変化については，図 9 に示す通り，全体的な傾向としては手法 1 と手法 3 が競っている．Downloader.Win32.LMN のように手法 2 が他の 2 つよりも有効となるファミリーも存在する．以上のことから亜種推定において API の引数も含めて Paragraph Vector を求めることが有用である．また API の引数値は何らかの変換を行うと特徴を表しやすくなることが分かる．

ランダムフォレストを用いた場合にも SVM と同様の結果が得られた．ランダムフォレストの場合の手法 1, 2, 3 の精度，TPR，FPR の平均値，最高値，最低値を表 3, 4, 5 にそれぞれ示す．これらの表に示す通り，TPR，FPR 双

表 3 ランダムフォレストにおける精度

数値＼手法 [%]	手法 1	手法 2	手法 3
平均値	83.28	81.33	82.33
最高値	96.63	95.44	97.02
最低値	70.98	69.20	67.26

表 4 ランダムフォレストにおける TPR

数値＼手法 [%]	手法 1	手法 2	手法 3
平均値	82.39	80.52	81.75
最高値	95.24	93.25	95.24
最低値	66.96	67.26	57.14

表 5 ランダムフォレストにおける FPR

数値＼手法 [%]	手法 1	手法 2	手法 3
平均値	15.83	17.86	17.10
最高値	31.55	29.46	32.14
最低値	1.98	1.59	1.19

方の平均値が極端に高いあるいは低いわけでない．TPR, FPR の両方が精度に影響している．ランダムフォレストは複数の特徴量からランダムに選んだ特徴量を用いて決定木を作成して，判別における特徴量の貢献度や重みを学習する．ランダムフォレストでも SVM と同等の結果を得られたことから，今回用いた Paragraph Vector はベクトルの全体で 1 つの文脈の特徴を表すものであるが，分類をする際には要素に重み付けすることも有効であると分かる．今回の目的が**人の手を介さず特徴量を作成し亜種判定を行う**ことであるから、本論文では自動的に作成された Paragraph Vector に重み付けなどの処理を加えることは範囲外と考えている．

6. まとめ

6.1 結論

本論文は，マルウェアの動的解析の結果に含まれる API 名と API の引数の列を 1 つの文章のように取り扱い，Paragraph Vector で表現することにより，人手によらず自動的に特徴ベクトルを作成し，それによってマルウェアの亜種判定が可能であることを示した．判別精度はファミリーによって異なるが，SVM を用いることで平均 81.69% ，最大 96.43% の精度が実現可能であった．また，API の引数の取り扱い法は，引数値自体を用いる場合，引数値の長さを用いる場合でそれぞれ API 名のみを用いる場合よりも判定精度が高くなるファミリーがある．この結果から引数も亜種判定に有用であることを示した．

6.2 今後の課題

本論文で提案した手法では亜種判定としては不十分な精度となるファミリーが存在した．本論文では API の引数

に着目したが、さらに返り値等の他のパラメータも含めることで精度の向上が望まれる．

また，本論文では Paragraph Vector 化するコーパスの作成に API 名のみからなるもの，API 名だけでなく引数名，引数値を含めたものを用いた．引数名および引数値を用いることで精度が向上したマルウェアファミリーも存在したが，API のみの方が亜種判定が高精度で行えるファミリーも存在する．引数名や引数値にマルウェアファミリーの特徴が出ないことは考えにくい．本論文で行った引数値の長さを引数値自体の代わりに用いるという前処理が有効なファミリーも存在する．Paragraph Vector で表現する際に特徴を消さないような単純な前処理ができるかどうか検討してみることは意味がある．

本論文においては Word Vector ならびに Paragraph Vector を用いて API コール列の特徴をベクトル表現で表した．具体的にベクトル表現のどの要素が元の API コール列のどの要素と強く関係があるか，というようなベクトルで表す前後の関連性についての考察が必要となるだろう．自然言語における Word Vector や Paragraph Vector の 成功例として良く引用されるベクトル化した後のベクトル演算の意義が本手法の場合にはどの程度あるのかという考察も，本手法のさらなる発展のために有効であろう．

謝辞 本研究の一部は JSPS 科研費 JP16H02832 の助成を受けたものです．MWS データセットを提供頂いた情報処理学会 MWS コミュニティのメンバー諸氏に感謝いたします．

参考文献

[1] “McAfee 脅威レポート 2016 年第 1 四半期,” http://www.mcafee.com/jp/resources/reports/rp-quarterly-threats-may-2016.pdf, 2016.

[2] “G DATA SECURITYLABS MALWARE REPORT HALF-YEAR-REPORT JANUARY - JUNE 2015,” https://public.gdatasoftware.com/Presse/Publikationen/Malware_Reports/G_DATA_PCMWR_H1_2015_EN.pdf, 2015.

[3] “Rules for naming,” https://securelist.com/threats/rules-for-naming/

[4] 神薗 雅紀, 畑田 充弘, 寺田 真敏, 秋山 満昭, 笠間 貴弘, 村上 純一, “マルウェア対策のための研究用データセット〜MWS datasets 2013〜,” 情報処理学会, マルウェア対策研究人材育成ワークショップ 2013 (MWS2013), Oct. 2013.

[5] 秋山 満昭, 神薗 雅紀, 松木 隆宏, 畑田 充弘, “マルウェア対策のための研究用データセット〜MWS datasets 2014〜,” 情報処理学会, Vol.114, No.118, CSEC, pp.125 – 131, Jul. 2014.

[6] 神薗 雅紀, 秋山 満昭, 笠間 貴弘, 村上 純一, 畑田 充弘, 寺田 真敏, “マルウェア対策のための研究用データセット〜MWS datasets 2015〜,” 情報処理学会, Vol.115, No.121, CSEC, pp.37 – 44, Jul. 2015.

[7] 高田 雄太, 寺田 真敏, 村上 純一, 笠間 貴弘, 吉岡 克成, 畑田 充弘, “マルウェア対策のための研究用データセット〜MWS datasets 2016〜,” 情報処理学会, 2016-CSEC-74, CSEC, pp.1 – 8, Jul. 2016.

[8] Q.V. Le, T. Mikolov, “Distributed Representations of Sentences and Documents”, Proceedings of the 31st International Conference on Machine Learning, pp.1188 – 1196, Jun. 2014.

[9] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient Estimation of Word Representations in Vector Space”, Proceedings of Workshop at ICLR, pp.1 – 12, May 2013.

[10] Akinori Fujino, Junichi Murakami, Tatsuya Mori, “Discovering similar malware samples using API call topics,” 2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC), pp. 140 – 147, Jan. 2015.

[11] 中村燎太, 松宮遼, 高橋一志, 大山恵弘, “Kullback-Leibler 情報量を用いた亜種マルウェアの同定,” コンピュータセキュリティシンポジウム 2013 論文誌, pp.877 – 884, Oct. 2013.

[12] 堀合啓一, 今泉隆文, 田中英彦, “マルウェア亜種の動的挙動を利用した自動分類手法の提案と実装,” 情報処理学会論文誌 50(4), pp.1321 – 1333, Apr. 2009.

[13] “Cuckoo Sandbox,” http://www.cuckoosandbox.org/

[14] “FFR yarai analyzer Professional,” http://www.ffri.jp/products/yarai_analyzer_pro/

[15] “sentence2vec,” https://github.com/klb3713/sentence2vec

[16] “Libsvm,” https://github.com/arnaudsj/libsvm

[17] “scikit-learn,” http://scikit-learn.org/stable/ .