マルウェアに使用されるパッカーの経年変化に関する考察

岩田 吉弘[1,a)]　大坪 雄平[1,2]　萬谷 暢崇[1]

概要：パッカーはマルウェアの静的解析を困難にするものであり，マルウェアの多くはパッキングされていると言われている．我々は過去に機械学習を用いてパッカーを推定する方法を提案した．本稿では，この手法を用いて，2012 年から 2019 年までに観測されたマルウェアの分類を行った．学習に使用したパッカーは 2010 年から 2011 年頃のものであるため，正しい分類結果を期待できないものの，パッカーの経年変化については観察できるものと思われる．実験の結果，マルウェアに使用されるパッカーの傾向は毎年大きく変化していることが明らかとなった．

キーワード：パッカー, 静的解析, 機械学習

A study on trend of packer used by malware

Yoshihiro Iwata[1,a)]　Yuhei Otsubo[1,2]　Nobutaka Mantani[1]

Abstract: Packers make static analysis difficult, and most malware is said to be packed. We proposed a method to identify packers with machine learning in the past. In this paper, we used this method to classify the malware samples observed from 2012 to 2019. Although correct classification results could not be expected because the packed malware samples used used for our learning process were created from 2010 to 2011, trends of packers used by the malware samples could be observed. As a result of the experiment, it became clear that the trends were changing significantly every year.

Keywords: packer, static analysis, machine leaning

1. はじめに

マルウェアの多くはパッカーが使用され，検知を回避したり解析を困難にさせていると言われている．パッカーには，無償で使用できるもの，有償で使用できるもの，オリジナルで作成したものなど様々な種類が存在するため，パッカーの種別を特定することは困難である．従って，攻撃に使用されるマルウェアの傾向は毎年のように変化するため，マルウェア解析者は新しい解析妨害機能と日々対峙し続けるという状況が続いている．

上記のような状況は，解析者の実感としてはあるものの，客観的な調査は少なく，実態は明らかではない．そこで本稿では，マルウェアに使用されるパッカーに着目し，その経年変化を観測することで，マルウェア解析者の実感が実態に即しているかを調査した．

具体的には，2012 年から 2019 年 5 月末頃までのマルウェアについて，パッカー推定を行った．調査の結果，収集したマルウェアの 60%から 80%程度がパッキングされており，その内訳についても年ごとに大きく異なることが明らかとなった．

2. 関連研究

パッカー推定によく使われるツールとして PEiD [*1]があげられる．PEiD は非常に高い精度でパッカーを検知できるものの，パターンマッチングを用いていることから，誤検知率を抑えたシグネチャの作成に手間がかかる上に，シ

[1] 警察庁
National Police Agency
[2] 情報セキュリティ大学院大学
Institute of information security
[a)] iwata@post.cyberpolice.go.jp

[*1] 最新のバージョン 0.95 は次の Web サイトからダウンロード可能：`https://www.softpedia.com/`

グネチャが複雑になる傾向がある．また，新しいパッカーに対応するためには，新しいパッカー用のシグネチャの作成だけでなく，既存のシグネチャを修正する必要もあり，性能の維持にコストがかかる．

上記の課題を解決するため機械学習を用いたパッカー推定について，様々な研究がされている．この中で本論文と最も関連の深いものは伊沢らの手法 [1] である．伊沢らは，プログラムのエントリポイント部分（EP）にパッカーの特徴が最も現れると考え，EP から 15 バイト分のバイナリデータを SVM で学習させた．26 種類のパッカーと 3 種類のパッキングされていないプログラムの計 29 クラスを含む約 4,000 のサンプルを用いた実験では，99.46 %という非常に高い Accuracy で分類できることが確認されている．

3. 予備知識

3.1 パッカー

パッカーとは，プログラムをパッキング（暗号化／圧縮）するソフトウェアツールの総称である．マルウェアは，静的解析[*2]を妨害するために，パッキングという技術を用いている．そのため，解析をする際は，解析者はパッキングツールの種類を特定してから，パッキングを解除するための手順を考え，マルウェア本来のコード（以下，オリジナルコード）を抽出しなければならない．一方攻撃者は，UPX[2] や Themida[3] などのパッカーを使用するだけでマルウェアをパッキングすることができる．そのため，攻撃者の労力はほぼない．加えて，多くの種類のパッカーが流通しているため，あるマルウェアに対してどのパッカーが使用されたかはマルウェア取得時点では解析者側には分からない．そのため，取得のたびに使用されているパッカーの特定やアンパッキングをしなければならない．そこで，パッカーの種類を自動的に特定することができれば，オリジナルコードを取り出すまでにかかる手間が削減でき，解析の効率化につながる．また，オリジナルコードを自動で抽出するツールの作成にもつながる可能性がある．

パッカーは対象のプログラムをパッキングし，それにアンパッキングルーチンを付加する．このアンパッキングルーチンにより，自己解凍及び実行が可能となっている．例として，UPX でパッキングされたプログラムのファイル構造を図 1 に示す．

UPX でパッキングされたプログラムは，PE ヘッダと空のセクション，パッキングされたオリジナルプログラム，アンパッキングルーチンで構成されている．プログラムが実行されると，アンパッキングルーチンが先に実行され，パッキングされたプログラムをアンパッキングし，メモリ上に展開する．アンパッキングされたオリジナルプログラムは空のセクションに書き込まれた後，実行される．

[*2] マルウェアを動作させることなく，プログラミングコードから情報を得る解析

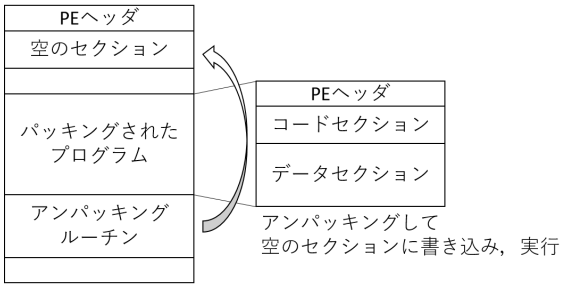

図 1 UPX でパッキングされたプログラムのファイル構造と挙動

UPX は非常に単純なパッカーだが，Themida や ASProtect[4] などは複雑なパッキングアルゴリズムを有しており，パッキングに使用されたパッカーによりプログラムの動きは異なってくる．

3.2 o-glasses

過去に我々が提案した o-glasses[5] について述べる．この手法は，入力データを x86/x86-64 実行コードとみなし，これを固定長命令に変換したものを畳み込みニューラルネットワーク（CNN [6]）に入力している．CNN の局所受容野と重み共有の仕組みを固定長命令に適用させることで，実行コードの分類が 99%という精度で実現されている．

入力データは，2048bit 値配列で，128bit 固定長命令に変換された実行コードを 16 命令分としている．1 層目は 1d-CNN で，フィルタサイズを 128，ストライドを 128 とし，1 命令分の局所受容野を形成している．出力されるチャンネルの深さは 96 である．2 層目も 1d-CNN で，フィルタサイズを 2，ストライドを 1，チャンネルの深さを 256 としている．この層により，2 つの命令間の関係の特徴を得ることを期待している．3〜5 層目は全結合で，ノード数は 400，400，n としている．ここで，n は分類するクラス数である．また，3〜4 層目の入力部分には，ネットワークの学習プロセスをより安定化させ高速化するため，Batch Nomalization[7] を取り入れている．中間層の活性化関数は ReLU を，出力層の活性化関数は Softmax 関数を使用している．

この手法は，実行コードか否かの認識に留まらず，様々な分野に適用できる可能性がある．例えば，この手法を用いて実行コードを生成したコンパイラの推定できるか実験を行ったところ，コンパイラの種類，対象アーキテクチャ，最適化レベル等の 15 クラス分類について，16 命令というコード断片から 95%以上の精度で分類できることが確認されている [8]．

4. 転移学習を用いたパッカー推定 [9]

我々が過去に提案したパッカー推定手法 [9] について以下に述べる．

これはパッカーの推定手法として，伊沢らの手法 [1] と

同様にプログラムの EP に格納されている機械語命令列に着目したものである．具体的には，EP を起点として L 個の命令を入力データとして o-glasses に入力し，パッカー推定を行う．この EP の機械語命令列に着目したパッカー推定は，前述のコンパイラ推定と比較して学習用データセットの作成について課題がある．コンパイラ推定の場合，1 つのプログラムに含まれる機械語命令すべてをデータセットに組み込めるため，データセットの大規模化は容易である．一方，EP の機械語命令列に着目したパッカー推定の場合，1 つのプログラムから作成できるサンプルは 1 つだけである．加えて，攻撃者が使用しているパッカーの入手が困難な場合がある．その場合，実際の攻撃で観測されたマルウェアからサンプルを作成することとなる．一般的なプログラムと比較してマルウェアの入手はより困難であることから，データセットの大規模化は更に困難なものとなる．

そこで我々の既存手法では o-glasses の学習フェーズで転移学習を行っている．転移学習は，ある領域で学習させたモデルを別の領域に適応させる技術のことであり，少ないデータで精度の高い学習結果を得ることができるというメリットがある．転移学習を用いた手法の概要を図 2 に示す．

具体的には以下の手順で学習を行う．まずは，o-glasses のネットワーク全体でコンパイラ推定用のデータセットを用いて学習を行う．我々はこの学習済みネットワークに，以下の特徴を有していることを期待している．

- 1 層目の CNN で 1 命令分の局所受容野の形成
- 2 層目の CNN で 2 つの命令間の特徴を得る
- プログラムとして成立する命令列か否かの判別

この学習済みネットワークの第 1 層と第 2 層の CNN における重み（パラメータ）を固定し，パッカー推定用のデータセットの学習を行う．パッカー推定用のデータセットはコンパイラ推定用のデータセットと比較して非常に小規模なものであるが，転移学習を用いることで効率的に学習を行うことができた．

5. マルウェアに使用されるパッカーの経年変化

本稿では，前節で述べた転移学習を適用した o-glasses を用いてマルウェアに使用されるパッカーの経年変化の観測を行う．具体的には，以下の手順で行う．

- 事前学習（5.1）
  コンパイラ推定用のデータセットを用いて o-glasses の全層を更新
- 転移学習（5.2）
  パッカー推定用のデータセットを用いて o-glasses の 3〜5 層目を更新
- データセットの準備（5.3）
  未知のマルウェアの収集
- パッカー推定（5.4）
  o-glasses でパッカー推定を行い，パッカーの経年変化について観測
- 経年変化の数値化（5.5）
  コサイン類似度を用いて経年変化を数値化

各フェーズにおける詳細を以下に述べる．

表 1　コンパイラ推定用データセットのサンプル数．(Opt.): 最適化レベル

Label	Compiler	Arch.	Opt.	File	Code
Program	VC2017	32bit	None	1,170	369,605
			Max	1,147	255,143
		64bit	None	1,456	540,568
			Max	1,242	542,020
	VC2003	32bit	None	1,350	292,277
			Max	1,306	270,743
		64bit	None	-	-
			Max	-	-
	GCC	32bit	None	2,111	227,004
			Max	1,844	239,821
		64bit	None	1,582	283,276
			Max	1,580	287,775
	Clang	32bit	None	1,205	101,024
			Max	1,196	86,521
		64bit	None	1,892	332,278
			Max	1,883	246,500
	ICC	32bit	None	1,761	1,494,677
			Max	1,724	1,161,499
		64bit	None	1,796	1,419,705
			Max	1,728	1,046,958
Others				130	912,958
Total				28,103	10,110,352

5.1 事前学習

コンパイラ推定用のデータセットの概要を表 1 に示す．

このデータセットは，オープンソースから誰でも収集することができ，大規模なデータセットを容易に作成できる．具体的には以下の手順で作成した．GitHub[*3] 上でオープンソースソフトウェアを収集し，複数のコンパイラ（VC[*4], GCC[*5], Clang[*6], ICC[*7]）で最適化レベルを「なし」と「最大」に変えながらコンパイルを行った．VC2003 は 32bit のみでコンパイルし，それ以外は 64bit および 32bit の両方でコンパイルした．コンパイルに成功したオブジェクトファイルのヘッダ情報を元に実行コード部分のみを取り出し，先頭から順番に L 個の命令ずつ切り出すことでデータセットを作成した．表 1 は $L = 16$ のときのデータセットの概要を示している．また，入力データが実行コードとみなせない機械語命令列にも対応するため，「Others」とラベル付けしたデータセットも作成した．これは，文書ファイルを元に作成した．文書ファイルは，文章，画像，メタデータなど様々な種類のデータが含まれている．その一方

*3 https://github.com/
*4 Microsoft Visual C++: 2003 and 2017
*5 the GNU Compiler Collection: 6.3.0
*6 a C language family frontend for LLVM: 5.0.2
*7 Intel C++ Compiler: 19.0.0.117 Build 20180804

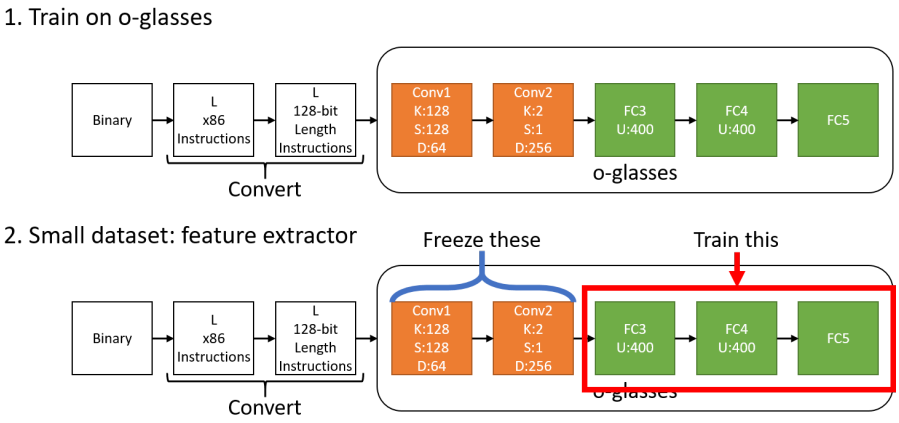

図 2 転移学習を用いた手法の概要

で，この文書ファイルがマルウェアでなければ，実行コードが含まれている可能性は極めて少ないと思われる．そこで，我々は検索エンジンを用いて「rtf」,「doc」,「docx」および「pdf」の 4 種類のファイルを収集し，VirusTotal[*8]を利用し，ウイルス対策ソフトでそのファイルがマルウェアとして検知されないことを確認した．その上で，文書ファイルを機械語命令列の集合とみなして，先頭から順番に強制的に逆アセンブルして L 個の命令ずつデータを切り出すことで，「Others」とラベル付けしたデータセットを作成した．

$L = 16$ の場合．19 クラスで延べ 28,103 個のファイルから合計 10,110,352 個の訓練データができた．このデータセットに含まれる命令の長さの平均を調べたところ，「Program」カテゴリで 3.69 バイト，「Others」カテゴリで 2.38 バイトであった．また，各クラス毎のサンプル数に偏りがあるため，学習に使用する各クラス毎のサンプル数の上限を $10^5 \times 16 \div L$ に設定し，$L = 16$ のときは，19 クラスで約 190 万のサンプルを使用した．

なお，本稿における事前学習では，分類精度が最も良かった $L = 4$ としたため，19 クラスで約 760 万のサンプルを使用し，epoch 数は 50 を選択した．

5.2 転移学習

コンパイラ推定用のデータセットで学習した o-glasses のネットワークの 1 層目と 2 層目の重みを固定し，パッカー推定用データセットによる転移学習を行った．パッカー推定用データセットについては伊沢ら [1] が使用したものと同じで，その内訳は表 2 のとおりである．また，この転移学習を用いたパッカー推定手法について評価実験を行った結果を表 2 に示す．

なお，本稿における転移学習では，$L = 4$，epoch 数は 500 を選択した．

[*8] https://www.virustotal.com/

表 2 クラス分類結果．(Num.): サンプル数, (P): precision, (R): recall

			L=16		L=4	
No.	Class	Num.	P	R	P	R
1	Unpacked(64bit)	475	99.8	100.0	100.0	99.8
2	Unpacked(32bit)	212	96.8	93.8	96.3	95.3
3	Unpacked(32bit&.NET)	144	100.0	98.6	100.0	98.6
4	Armadillo 4.00.0053	203	99.0	100.0	99.0	100.0
5	MoleboxPro 2.6.4	203	99.1	100.0	100.0	100.0
6	Themida 1.8.5.5	195	100.0	100.0	100.0	100.0
7	PESpin 1.330	194	100.0	100.0	100.0	100.0
8	obsidium 1.3.5.4	191	98.0	100.0	99.0	100.0
9	ASProtect 2.100	188	99.5	100.0	100.0	100.0
10	yoda's protector 1.020	170	100.0	100.0	100.0	100.0
11	obsidium 1.4.5	165	100.0	100.0	100.0	100.0
12	nPack 1.1.300	201	100.0	100.0	100.0	100.0
13	eXPressor 1.5.0.1	182	100.0	100.0	99.5	100.0
14	ASPack 2.12	194	100.0	100.0	100.0	100.0
15	PECompact 2.64	201	100.0	100.0	100.0	100.0
16	NsPack 3.7	203	100.0	100.0	100.0	100.0
17	RLPack 1.2	200	100.0	100.0	100.0	100.0
18	FSG 2.0	203	100.0	100.0	100.0	100.0
19	UPX 3.08	194	99.5	100.0	98.5	100.0
20	ASPack 2.11	147	99.3	100.0	100.0	100.0
21	Mew11SE 1.2	189	100.0	100.0	100.0	100.0
22	ACProtect pro 1.32	75	87.1	92.1	92.8	97.8
23	WWPack32 1.2	108	100.0	100.0	100.0	100.0
24	AntiCrack 1.32Pro	64	91.0	77.9	98.6	90.7
25	PETITE 2.2	90	98.2	97.8	99.0	97.8
26	exe32pack 1.4.2	97	100.0	98.9	100.0	98.9
27	tElock 0.98	74	98.9	100.0	100.0	100.0
28	PKLITE 32	20	93.3	90.0	93.3	90.0
29	Upack 0.39	201	96.7	97.0	97.2	96.5

表 3 パッカー推定を行った検体数

年	2012	2013	2014	2015	2016	2017	2018	2019	計
検体数	58	162	161	96	120	106	235	61	999

5.3 データセットの準備

パッカー推定を行う未知のマルウェアとして，VirusTotal からランダムサンプリングで 999 検体を収集した．これらのマルウェアの観測された時期を調べたところ，2012 年 1 月から 2019 年 5 月末頃までのものであった．その検体数の内訳を表 3 に示す．

5.4 パッカー推定

学習済みの o-glasses を用いて前節の未知のマルウェ

表 4　推定結果（パッカーごとの推定された検体数）

No.	Class	2012	2013	2014	2015	2016	2017	2018	2019
1	Unpacked(32bit)	12	33	15	14	34	20	44	9
2	Unpacked(64bit)	0	7	11	5	6	8	16	3
3	Unpacked(32bit&.NET)	0	14	18	9	7	9	72	3
4	Armadillo 4.00.0053	29	40	34	11	11	10	10	2
5	MoleboxPro 2.6.4	0	0	0	0	1	1	0	0
6	Themida 1.8.5.5	1	0	0	0	0	0	0	0
7	PESpin 1.33	0	0	0	1	0	0	0	0
8	obsidium 1.3.5.4	1	6	5	6	10	8	3	0
9	asprotect 2.100	0	0	2	0	5	0	4	1
10	yodaprotector 1.020	0	1	1	1	0	0	1	0
11	obsidium1.4.5	0	0	1	0	0	2	0	0
12	nPack 1.1.300	1	6	4	4	1	0	0	0
13	eXPressor 1.5.0.1	3	8	9	4	3	1	1	0
14	ASPack 2.12	1	5	14	10	23	9	2	3
15	PECompact 2.64	0	0	1	0	0	0	0	0
16	NsPack 3.7	0	3	5	3	2	1	0	0
17	RLPack 1.2	0	1	0	0	0	3	0	0
18	fsg 2.0	0	1	4	2	6	1	0	1
19	UPX 3.08	1	15	3	6	4	3	10	3
20	ASPack 2.11	0	2	4	3	0	3	1	0
21	Mew11SE 1.2	0	5	7	1	2	2	45	26
22	ACProtect pro 1.32	0	2	0	0	0	22	1	1
23	WWPack32 1.2	8	10	12	13	2	2	2	0
24	AntiCrack 1.32Pro	0	0	0	0	1	0	0	1
25	PETITE 2.2	0	1	7	3	2	0	1	1
26	exe32pack 1.4.2	0	0	4	0	0	1	11	6
27	tElock 0.98	0	2	0	0	0	0	7	1
28	PKLITE 32	0	0	0	0	0	0	0	0
29	Upack 0.39	1	0	0	0	0	0	4	0

表 5　パッキングされた検体の割合

	2012	2013	2014	2015	2016	2017	2018	2019
Packed	79.3%	66.7%	72.7%	70.8%	60.8%	65.1%	43.8%	75.4%
Unpacked	20.7%	33.3%	27.3%	29.2%	39.2%	34.9%	56.2%	24.6%

アに使用されているパッカーの推定を行った．推定した結果を表 4 及び図 3 に示す．推定を行った検体は未知のマルウェアであり，パッキングの有無及びパッカーの種別は不明である．

表 4 の推定結果で示された，パッキングされた検体とパッキングされていない検体の割合を表 5 に示す．パッキングされた検体とは，表 4 で示した「Unpacked(32bit)」，「Unpacked(64bit)」及び「Unpacked(32bit&.NET)」以外の検体を指す．パッキングされた検体の割合は，各年，60%から 80%程度となっていた．

また，2013 年には 64 ビットプログラム及び.NET プログラムが出現し始め，2018 年には.NET プログラムの割合が急激に増加したことが確認できた．

5.5　経年変化の数値化

表 4 における各年の推定結果を特徴ベクトルとみなし，マルウェアに使用されたパッカーの経年変化の指数をコサイン類似度を用いて計算する．$\vec{a_i}$ を，i 年に観測されたマルウェアに使用されたパッカーの傾向を示す特徴ベクトルとすると，i 年と j 年とのコサイン類似度は次式で定義される．

$$\cos(\vec{a_i}, \vec{a_j}) = \frac{\vec{a_i} \cdot \vec{a_j}}{|\vec{a_i}||\vec{a_j}|} \quad (1)$$

この場合，コサイン類似度は 0 から 1 の値をとり，数値が大きくなるほど類似度が高いことを示す．

表 6　各年のコサイン類似度

	2012	2013	2014	2015	2016	2017	2018	2019
2012	1.000	-	-	-	-	-	-	-
2013	0.881	1.000	-	-	-	-	-	-
2014	0.810	0.892	1.000	-	-	-	-	-
2015	0.669	0.855	0.880	1.000	-	-	-	-
2016	0.529	0.746	0.697	0.813	1.000	-	-	-
2017	0.480	0.690	0.631	0.682	0.734	1.000	-	-
2018	0.265	0.586	0.600	0.572	0.538	0.544	1.000	-
2019	0.182	0.373	0.389	0.322	0.388	0.350	0.684	1.000

各年のコサイン類似度を表 6 に示す．

6.　考察

6.1　マルウェアの経年変化観測へのパッカー推定結果の活用

我々が過去に提案したパッカー推定手法では，転移学習時に使用したデータセットに含まれているパッカーに対してのみ，パッカー種別の推定を行うことができる．また，転移学習時に使用したデータセットは 2010 年から 2011 年頃のパッカーを使用したものであるため，パッカーのバージョンが異なるものや学習していない未知のパッカーに対しては正しく推定することができない．

そこで，本論文では，パッカーを正しく推定できていることを期待してはいない．推定結果を年ごとにまとめ，その年のマルウェアの傾向を示す特徴ベクトルとして扱うことで，マルウェアの経年変化の考察に推定結果を活用している．

6.2　マルウェアの傾年変化について

表 6 から，各年の傾向がどの程度変化したか読み解くことができる．例えば，直近の年とのコサイン類似度に着目すると，2017 年と 2018 年を比較した値が 0.544 で最小であり，2018 年に使用されるパッカーの傾向が大きく変化したことが分かる．また，コサイン類似度の値が最小なものは，2012 年と 2019 年を比較したもので 0.182 となっており，7 年間でパッカー傾向が大きく変化したことが分かる．

6.3　解析の困難さ

表 5 から，マルウェアの 60%から 80%程度はパッキングされていることが確認できた．また，表 6 から，年数が経つほどパッカー傾向は大きく変化していることが確認できた．よって，マルウェア解析者は過去の知見が活かしにくくなっており，解析の困難さを示す結果となった．

6.4　提案手法の限界

我々が過去に提案したパッカー推定手法では，.NET プログラムについて推定を行った場合，パッキングの有無にかかわらず「Unpacked(32bit&.NET)」と推定される．これは，.NET プログラムの EP の機械語命令列は，パッキ

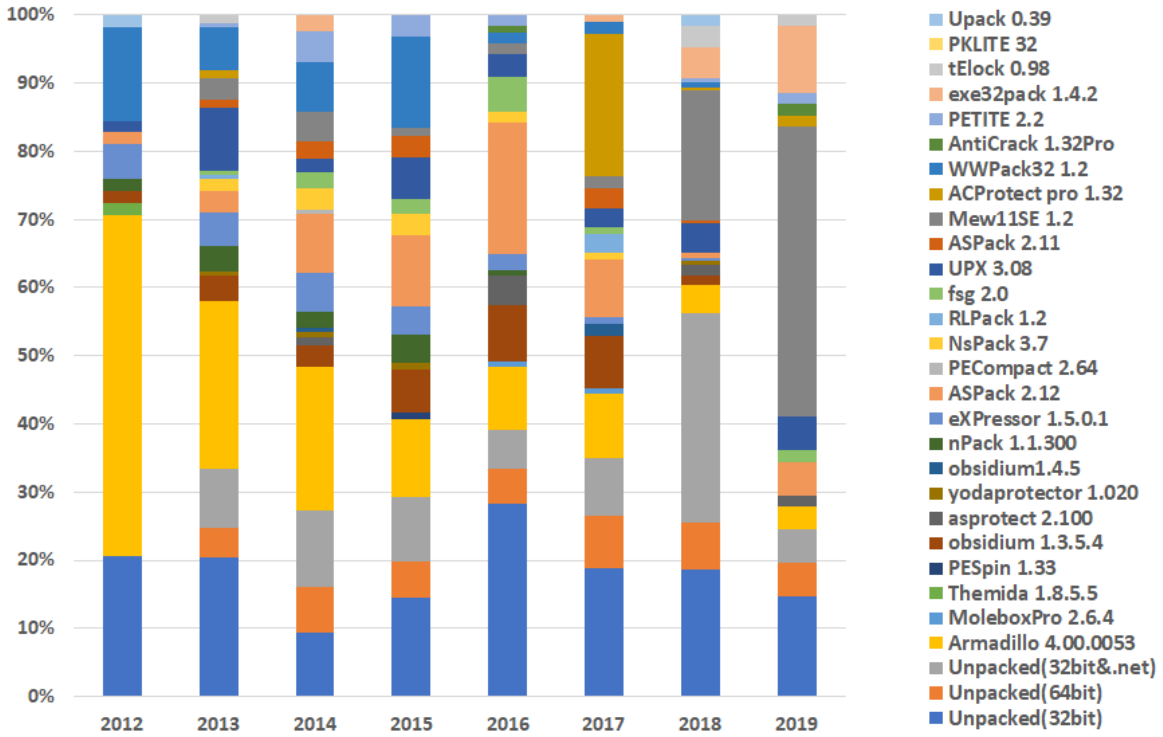

図 3 推定結果（年ごとの各パッカーの割合）

ングの有無にかかわらずほぼ同じ命令から始まっていることが原因である．そのため，.NET プログラムのパッカーを推定するためには，EP の機械語命令列に依らない別の手法と組み合わせる必要がある．

7. おわりに

本稿では，2012 年から 2019 年に観測された未知のマルウェアに対してパッカーの推定を行った．パッカー推定の学習用のデータセットは，2010 年から 2011 年頃のパッカーを使用したものであり最近のマルウェアに使用されているパッカーのものではない．そのため，各々のパッカー推定結果については正しく推定されることが期待されない．しかしながら，推定結果を特徴ベクトルとみなしコサイン類似度を計算することで，マルウェアの経年変化の指標として数値化した．その結果，マルウェアのパッカーの傾向は年々変化しており，年数が経つほど過去の解析の知見が活かしにくくなっているという解析者の実感が正しいことを客観的に示すことができた．

謝辞

本研究では，パッカーのデータセットを国立研究開発法人情報通信機構サイバーセキュリティ研究所の伊沢亮一博士に提供いただいた．ここに感謝する．

参考文献

[1] Isawa, R., Ban, T., Guo, S., Inoue, D. and Nakao, K.: An Accurate Packer Identification Method Using Support Vector Machine, *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. 97, No. 1, pp. 253–263 (2014).

[2] Markus, F. O., Lszl, M. and John, F. R.: the Ultimate Packer for eXecutables, `https://upx.github.io/` (1996).

[3] Oreans Technologies: Themida, `https://www.oreans.com/themida.php`.

[4] StarForce Technologies Ltd.: ASPack Software, `http://www.aspack.com/asprotect32.html`.

[5] Otsubo, Y., Otsuka, A., Mimura, M., Sakaki, T. and Goto, A.: o-glasses: Visualizing x86 Code from Binary Using a 1d-CNN, *ArXiv e-prints*, p. arXiv:1806.05328 (2018).

[6] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. and Jackel, L. D.: Backpropagation applied to handwritten zip code recognition, *Neural Computation*, Vol. 1, No. 4, pp. 541–551 (1989).

[7] Ioffe, S. and Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift, *arXiv preprint arXiv:1502.03167* (2015).

[8] 大坪雄平，大塚　玲，三村　守，榊　剛史，受川　弘，岩田吉弘：コード断片からのコンパイラ推定手法，コンピュータセキュリティシンポジウム 2018(CSS2018)，pp. 4C2–1 (2018).

[9] 大坪雄平，大塚　玲，岩田吉弘，三村　守，榊　剛史：転移学習による機械語命令列分類における学習の効率化，人工知能学会全国大会（第 33 回）(JSAI2019) (2019).