実行命令系列の分類に基づいた汎用的なオリジナルエントリポイント特定手法の提案

岸部 功太郎 † 中村 徳昭 † 森井 昌克 † 伊沢 亮一 ‡ 井上 大介 ‡
中尾 康二 ‡

† 神戸大学大学院工学研究科
657-8501 兵庫県神戸市灘区六甲台町 1-1
{kishibe,nakamura.1}@stu.kobe-u.ac.jp,
mmorii@kobe-u.ac.jp

‡ 独立行政法人情報通信研究機構
184-8795 東京都小金井市貫井北町 4-2-1
{isawa,dai,ko-nakao}@nict.go.jp

あらまし マルウェアの多くはパッキング（圧縮/暗号化）が施されており，コード解析の障害となっている．本稿では解析支援を目的として，パッキングのアルゴリズムに依存しない精度の高いオリジナルエントリポイント（OEP）特定手法を提案する．提案手法は主に 2 つの特徴がある．1 つ目は検体の実行命令系列をメモリ上のアドレス距離によりグルーピングする点にある．オリジナルコードとパッキング時に付加されたコード（復号ルーチン）は離れた位置に配置されることが基本であり，それらを分離する．2 つ目は各グループがオリジナルコードか否かを機械学習で判定する点にある．オリジナルコードのうち最初に実行された地点を OEP として出力する．評価実験では，未知のパッカー 20 種類中 18 種類に対して OEP を特定することができた．

Original Entry Point Detection by Classifying Dynamically Generated Instructions

Kotaro Kishibe† Noriaki Nakamura† Masakatu Morii† Ryoichi Isawa‡ Daisuke Inoue‡
Koji Nakao‡

†Graduate School of Engineering, Kobe University.
1-1, Rokkodai-cho, Nada-ku, Kobe-shi, Hyogo 657-8501, JAPAN
{kishibe,nakamura.1}@stu.kobe-u.ac.jp, mmorii@kobe-u.ac.jp
‡National Institute of Information and Communications Technology.
4-2-1, Nukui-Kitamachi, Koganei-shi, Tokyo 184-8795, JAPAN
{isawa,dai,ko-nakao}@nict.go.jp

Abstract Most malware programs spread through the Internet are packed (compressed and/or encrypted) by packers to hamper code analyses. In order to help extract original binaries of packed malware efficiently, this paper presents an accurate original-entry-point detection method featuring two mechanisms. First, it classifies dynamically generated instructions of a packed program based on their memory addresses. Because instructions of the original binary and those of the unpacking routine will be placed away from each other generally, our method can divide them into different clusters. Second, it identifies if each class is composed of the original binary. The first executed point in clusters of original binary will be the original entry point. With experiments using samples packed by 20 packers, we confirmed that our method correctly detected the original entry point against samples packed by 18 packers.

1　はじめに

マルウェアの多くは解析を妨害する目的で難読化（パッキング）が施されている．パッキングされたマルウェアの解析にあたり，まずマルウェア本来のコードを抽出（アンパッキング）する必要がある．アンパッキングはパッカーに対応したアンパッカーを用いるほか，OllyDbg[1] などのデバッガを用い手動で行うことが一般的である．しかし多種多様なパッカーのアンパッキングを個別に行うことは多大な労力を要する．そこで近年では解析の効率化を目指し，パッカーの種類を問わない汎用アンパッキング手法が研究されている．

マルウェアを解析する上でマルウェア本来のコード（オリジナルコード）だけでなく，オリジナルエントリポイント（OEP）を特定する必要がある．OEP はオリジナルコードの最初に実行される命令位置のことを指すが，オリジナルコードを逆アセンブルする際，コードを解釈させる起点となる．x86 などのアーキテクチャでは可変長命令を採用しており，OEP の情報がなければ逆アセンブルを開始する地点を特定できないため，本来とは異なったコードが出力されてしまう．パッカーの中には多段的に展開処理を行うため OEP の候補がメモリ上に複数存在する場合があり，このようなパッカーに対しても OEP を一意に特定することが課題となっている．本稿ではオリジナルコードの取得および OEP の検出を目的とする．

本稿では各命令間の (1) メモリアドレス距離と (2) 実行命令系列の出現順序に着目した，精度の高い OEP 特定手法を提案する．(1) に関して，我々はコードの種類が同じであれば，ある程度連続したメモリ領域で命令の書き込み/実行が行われ，一方でコードの種類が異なれば書き込み/実行されるメモリ領域は一定距離以上離れると考えた．ここで，コードの種類とはオリジナルコードとパッカーが付加したルーチン（復号ルーチン）のことを指す．また (2) に関して，パッカーの動作はマルウェア本体に比べて展開処理に特化しており，実行命令系列の出現パターンが特徴的であると考えた．提案手法では命令間のアドレス距離により実行命令系列のグルーピングを行い，各グループを機械学習によりオリジナルコード，復号ルーチンの 2 種類のクラスに分類する．機械学習には LCS（最長共通部分列）[9] を実行命令系列の特徴として用いる．本稿では機械学習に kNN 法 [10] を用いた．オリジナルコードと復号ルーチンを判別した後，1 メモリページ以上のオリジナルコード群のうち最初に実行された地点を OEP として検出する．評価実験では学習データにないパッカー 20 種類中 18 種類に対して OEP が検出できることを示す．なお我々は [7] で OEP の特定手法を提案しているが，実行命令系列の分割時に PE ヘッダのセクション情報を利用している点で本稿とは差異がある．

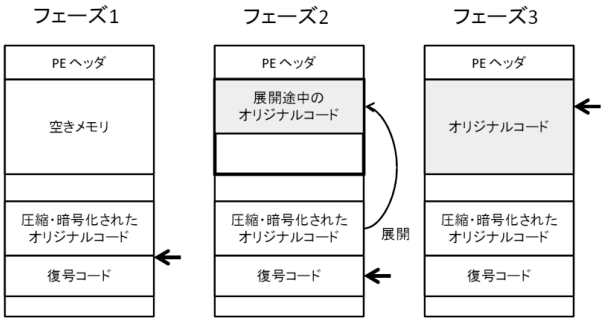

図 1: UPX の動作

2　背景知識

2.1　一般的なパッカーの動作

図 1 は UPX[2] でパッキングされた実行ファイルのセクション構造と処理の流れを示す．UPX を例にしてパッキングされたファイルの基本的な動作を説明する．フェーズ 1 は UPX でパッキングされた直後のセクション構造を示す．UPX は実行ファイルをパッキングするとき，PE ヘッダを除くすべてのセクションを合わせて圧縮･暗号化し，1 つのセクションに格納する．同時に圧縮･暗号化されたオリジナルコードを復号するための復号ルーチンが別のセクションに生成される．このときエントリポイントは復号ルーチンのセクション内を指しており，パッキングされたファイルを実行すると復号ルーチンが実行さ

れる．フェーズ 2 は圧縮･暗号化されたコードが復号されている状態を示している．復号ルーチンは圧縮･暗号化されたコードを読み取りながら復号し，空きメモリ上に展開していく．フェーズ 3 は復号ルーチンによる処理が完了し，オリジナルコードが実行される直前の状態を示している．このときオリジナルコードセクションで最初に実行される地点が OEP である．

図 1 のように，パッキングされた実行ファイルを実行するとメモリ上にオリジナルコードが展開される．そこで提案手法では書き込み/実行が行われたメモリ領域をオリジナルコードが展開される候補とし，オリジナルコードの判別，OEP の検出を行う．

2.2 実行命令系列

PE ファイル（Potable Executable）等の実行ファイルを DBI(Dynamically Binary Instrumention) ツールの一種である PIN[11] 上で実行すると，メモリアドレス，オペコード，オペランドの組を実行順に取得することができる．これを実行命令系列と呼ぶ．提案手法では書き込み/実行が行われたメモリ領域に着目しているが，実行されたメモリ領域は PIN を用いて実行命令系列を取得することで抽出できる．

提案手法では訓練データ（正解データ）として復号ルーチンを抽出する必要があるが，パッキングされていないファイルから抽出した実行命令系列と，パッキングされたファイルから抽出した実行命令系列のメモリアドレスやオペコードを比較し，差分を取ることで抽出できる．

2.3 関連研究

汎用アンパッキング手法に共通するアイデアとして，メモリ上に発生した書き込み/実行に注目していることが挙げられる．これらの研究は目的で大別でき，オリジナルコードのみを取得する方法 [3]，オリジナルコードとオリジナルエントリポイント（OEP）を検出する方法 [4][5][6]，パッキングされたファイルをアンチウイルスソフトで検知する手法 [8] が挙げられる．

Polyunpack[3] は動的に生成されたコードを特定することを目的としている．アンパッキング対象となる実行ファイルを事前に逆アセンブルして静的解析を行う．その後動的解析を行い，静的解析結果と異なるコードの実行が行われた場合，このコードは復号されたオリジナルコードであると判断している．

Renovo[4] は JMP 命令に着目して OEP を検出する．JMP 命令により書き込まれたコードに処理が遷移したとき，EIP のアドレスを OEP とする．対象のファイルを 1 命令ずつ実行し逆アセンブルする．

川古谷らの手法 [5] はメモリアクセスの傾向変化が最も大きい箇所を OEP としてオリジナルコードのメモリダンプを取得している．メモリアクセスと書き込み，読み込み，実行のことであり，式によってこれらの傾向を数値化する．

伊沢らの手法 [6] は，2 つのパラメータを基準にして OEP を判定している．1 つ目はメモリのエントロピによる判定，2 つ目はメモリ上に書き込まれた API アドレスの個数による判定である．前者ではパッキングされたコードに比べて使用される命令には偏りがあり，エントロピの値が低くなる傾向を利用している．後者では，展開処理が進むにつれてメモリ上に記述される API アドレスが増えていくことに着目している．

OmniUnpack[8] はコードの書き込み/実行をページ単位で監視し，そのコードが危険なシステムコールを呼び出したときにアンチウイルスソフトを呼び出す．パッカーで内容が変更されたマルウェアをアンチウイルスソフトで検出することを目的としており，OEP は検出されない．

3 提案手法

3.1 基本アイデア

パッキングされたプログラムがメモリ上で復号されるとき，復号されたオリジナルコードは復号ルーチンとある程度離れた位置に書き込まれるようパッカーが設計されていることが多い．これはアドレス距離を近づけることによる復号ルーチンの上書き等を防ぐためである．我々は

それぞれのアドレス距離が離れる点に着目し，実行命令系列から OEP を求める．

提案手法ではパッキングされたプログラムの実行命令系列をメモリ上のアドレス距離に応じてグルーピングし，複数のグループ（コード群）に分割する．上記で述べたパッカーの設計から，各グループはオリジナルコードの命令群で構成されているか，復号ルーチンの命令群で構成され，理想的には 1 つのグループ内にそれぞれが混ざることがない．このグルーピングの精度が提案手法の精度に影響する 1 つの要因となる．

グルーピングのあと，各グループがオリジナルコードに属するものか復号ルーチンのものかを判定する．提案手法ではある 2 つのグループの組の最長共通部分列（Longuest Common Subsequence）に着目する．オリジナルコードと復号ルーチンのグループは最長一致長が短く，オリジナルコードのグループ同士であれば最長一致長が比較的長くなると仮定している．マルウェアのオリジナルコードはソースの使い回しで作成されることも多く，コンパイラも共通していることが多いため，共通部分も多くなるという考えに基づく．対して，復号ルーチンはパッカーの作成者がアセンブリ言語で書かれていることも多く，オリジナルコードとの共通部分は少なくなる．本稿ではこの判定に k-Nearest Neighbor 法（kNN 法）を用いる．オリジナルコードと判定されたグループのうち，最初に実行されたアドレスを OEP として出力する．この判定結果が提案手法の精度に影響するもう一つの要因である．

3.2 アドレス距離によるグループピング

取得した実行命令系列をアドレス順にソートし，各命令間のアドレス距離 $(d_1,d_2,\cdots,d_n)$ を算出する．メモリアドレス距離の最大値を d_{max} とし，$d_i/d_{max} > \theta$ となる場合，その前後の命令は別グループであると判定する．パッカーによっては復号ルーチンを複数作成するため，単純に距離が最大となる地点のみを分離地点とすることができない．そこで本手法では閾値 θ を設定した．今回の評価実験では $\theta = 0.1$ と設定した．

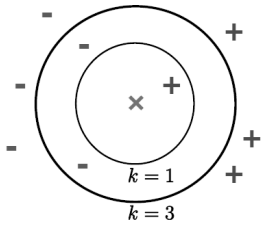

図 2: kNN 法の例

3.3 グループの識別

各グループがオリジナルコードか復号ルーチンのものかを判定する方法を述べる．

3.3.1 LCS(Longest Common Subsequence)

LCS とは 2 つの系列の共通の部分列の中で最長の系列のことであり，例えば‘abcde’と‘abdec’の LCS は‘abde’となる．LCS 長の導出方法としては，2 つの系列を M，N とした場合，計算量 $O(MN)$ で LCS を算出するアルゴリズムが知られている．ここで 2 つの系列をそれぞれ $S = s_1^m$，$T = t_1^n$，S と T の LCS 長を $L(s_1^m, t_1^n)$ とすると，LCS の性質として，以下の式が成り立つ．

$$L(s_1^i, t_1^j) = \begin{cases} 0 & (i = 0 or j = 0) \\ L(s_1^{i-1}, t_1^{j-1}) + 1 & (s_i = t_i) \\ max(L(s_1^i, t_1^{j-1}), L(s_1^{i-1}, t_1^j)) & (s_i \neq t_i) \end{cases}$$

この再帰式を利用することで，動的計画法により $O(MN)$ で LCS の長さを算出することができる．算出した LCS 長は機械学習のパラメータに利用する．

3.3.2 kNN 法 (k-Nearest Neighbor Algorithm)

各コード群の LCS 長をもとに復号ルーチンとオリジナルコードを正しく判定するため，機械学習アルゴリズムの 1 つである kNN 法を用いる．kNN 法は機械学習アルゴリズムの中で最も単純な学習方法である．あるテストデータの分類は，その近傍のデータ群の投票によって決定される．すなわち，最近傍から順に k 個のデータを選び，それらのクラスの多数決によってテストデータのクラスを決定する．kNN 法の

例を図 2 を用いて説明する．図 2 では‘＋’と‘-’，2 つのクラスから構成されており，そこに新たなテストデータ‘x’が入力されたとする．k=3 の場合，最近傍から順に 3 個のデータが用いられるため，‘＋’より 1 つ数の多い‘-’クラスに分類される．このような二項分類の場合，k を奇数にすると同票数で分離できなくなる問題を避けることができる．今回のコード種別判定も二項分類であり，与えられたテストデータをオリジナルコードと復号ルーチンの 2 クラスに分類することを目的とする．

3.3.3 訓練データの作成

kNN 法で使用する訓練データの作成方法を述べる．訓練データはオリジナルコード，復号ルーチンの 2 種類のクラスから構成される．オリジナルコードはパッキングが施されていない実行ファイルを PIN 上で実行することで抽出する．抽出した実行命令系列はそのままオリジナルコードの訓練データとして扱う．復号ルーチンは，パッキングされた実行ファイルと，パッキングされていない実行ファイルの実行命令系列の差分をとることで抽出する．それらを分割してグルーピングし，復号コードの訓練データを作成する．任意のデータ x_s,x_t の一致率 $S(x_s, x_t)$ をシンプソン係数と呼ばれる以下の式で定義する．

$$S(x_s, x_t) = \frac{L(x_s, x_t)}{min(l_{x_s}, l_{x_t})} \quad (1)$$

ただし $L(x_s, x_t)$ は x_s と x_t の LCS 長，l_{x_s},l_{x_t} はそれぞれ x_s と x_t の実行命令系列の長さを表す．また，一致率 $S(x_s, x_t)$ の相関行列を表に示す．この相関行列は kNN 法による機械学習において，訓練データとテストデータをまとめて入力する際に用いる．

4 評価実験

4.1 データセットの作成

評価実験を行うにあたり，kNN 法による機械学習のための訓練データ，実際に OEP を検出するためのテストデータをそれぞれ作成する．

4.1.1 訓練データの作成

訓練データセットの作成のため，27 種類の復号ルーチンと 143 種類のオリジナルコードを用意した．復号ルーチンの訓練データを作成するために用いた 27 種類のパッカーを表 2 に示す．各パッカーで calc.exe，comp.exe，wordpad.exe の順にパッキングし，先に動作したものから復号ルーチンの抽出を行った．これはパッカーと実行ファイルの組み合わせによってはパッキング後動作しないためである．パッカーによる復号処理はマルウェアと電卓などの一般の実行ファイルで差が出ないため，起動及び終了の検証が容易である一般の実行ファイルを使用した．パッカーの種類によっては多段的に復号を行い，複数の復号ルーチンを生成するものも存在するが，その場合は作成されたコード群ごとに訓練データを作成する．また Themida や Armadillo のようなパッカーは復号ルーチンのデータ量が大きく LCS を求めるのが困難であったため，コード群のうち最初に実行された命令系列 10000 行を抽出し訓練データを作成した．

オリジナルコードの訓練データの作成には，WindowsXP のシステムディレクトリから取得できる実行ファイル 106 種類，我々が保有しているマルウェアのうちパッキングが施されていない検体 37 種類を用いた．動作の終了が確認できない実行ファイルに関しては，起動して 1 分後に強制終了することで実行命令系列を取得した．

4.1.2 テストデータの作成

テストデータ作成のため，7 種類のマルウェアと 20 種類のパッカーを用意した．7 種類のマルウェアはいずれもパッキングが施されていないものを選択し，20 種類のパッカーは訓練データ作成時に使用した 27 種類以外から選択した．マルウェアとパッカーの組み合わせが悪い場合を除き，すべての組み合わせで実行命令系列を取得し，それぞれメモリアドレス距離によりグルーピングを行う．

表 1: 一致率 $S(x_s, x_t)$ の相関行列

	x_0	x_1	$\cdots$	x_m	x_{m+1}	$\cdots$	x_{m+n}
x_0	1	$S(x_0, x_1)$	$\cdots$	$S(x_0, x_m)$	$S(x_0, x_{m+1})$	$\cdots$	$S(x_0, x_{m+n})$
x_1	$S(x_0, x_1)$	1					$S(x_1, x_{m+n})$
$\vdots$	$\vdots$		1				$\vdots$
x_m	$S(x_0, x_m)$			1			$S(x_m, x_{m+n})$
x_{m+1}	$S(x_0, x_{m+1})$				1		$S(x_{m+1}, x_{m+n})$
$\vdots$	$\vdots$					1	$\vdots$
x_{m+n}	$S(x_0, x_{m+n})$		$\cdots$	$S(x_m, x_{m+n})$	$S(x_{m+1}, x_{m+n})$	$\cdots$	1

4.2 評価指標

コード種別判定の結果を評価する指標として，Accuracy と Precision，Recall を用いる．あるクラス y_i に着目し，あるデータがそのクラス y_i に分類されたとき ' positive ' と呼び，その他のクラスに分類されたとき ' negative ' と呼ぶ．訓練データのサンプル数を n，P と N をそれぞれ ' positive ' なデータ数と ' negative ' なデータ数とすると，$n = P + N$ が成り立つ．評価に用いる値 $P = TP + FN$ と $N = FP + TN$ を以下のように定義する．

1. TP は ' positive ' なデータのうち，クラス y_i に分類されたデータ数を示す．
2. FP は ' positive ' なデータのうち，他のクラスに分類されたデータ数を示す．
3. TN は ' negative ' なデータのうち，クラス y_i に分類されたデータ数を示す．
4. FN は ' negative ' なデータのうち，他のクラスに分類されたデータ数を示す．

Accuracy は訓練データのうち正しく分類された割合を示し，式 (2) で求められる．FP の割合は式 (3) で求められる．Precision はクラス y_i に分類されたデータのうち ' positive ' なデータの割合を示し，式 (4) で求められる．Recall は ' positive ' なデータのうち，クラス y_i に分類されたデータの割合を示し，式 (5) で求められる．

$$Accuracy = \frac{TP + TN}{n} = \frac{TP + TN}{P + N} \quad (2)$$

$$FPrate = \frac{FP}{N} = \frac{FP}{FP + TN} \quad (3)$$

$$Precision = \frac{TP}{TP + FP} \quad (4)$$

$$Recall = \frac{TP}{TP + FN} \quad (5)$$

4.3 未知パッカーのコード種別判定

本実験では，テストデータに使用されるパッカーが未知という条件のもと，コード種別の判定ができることを示す．そのため，テストデータと訓練データで使用するパッカーは完全に区別する．それぞれのデータ作成に用いたパッカーは表 3 の ' 訓練データ ' および ' テストデータ ' の欄にまとめる．テストデータのグループ化を行い，得られたコード群すべてに対して訓練データとの一致率 $S(x_s, x_t)$ の相関行列（表 1）を作成し，kNN 法による機械学習を行う．

機械学習の結果を表 2 に示す．y_0 はオリジナルコードのクラス，y_1 は復号ルーチンのクラスを表す．最も Accuracy が高いのは k=3 のときで，94.05%であった．k=3 のとき，テストデータの 185 種類中 174 種類が正しいクラスに分類され，残りの 11 種類は正解と異なるクラスに分類された．マルウェアのオリジナルコード群はすべて正しく分類されており，分類に失敗した 11 種類のグループはいずれも復号ルーチンであった．失敗したパッカーは ' Themidav 1.8.5.5 ' と ' yoda's protector 1.02 ' であった．

表 2: 機械学習の結果

k of kNN	Accuracy (%)	Precision(%) クラス y_0	Precision(%) クラス y_1	Recall(%) クラス y_0	Recall(%) クラス y_1	FPrate(%) クラス y_0	FPrate(%) クラス y_1
1	90.81	88.28	100.0	100.0	70.18	29.82	0.0
2	89.73	87.09	100.0	100.0	66.67	33.33	0.0
3	94.05	92.09	100.0	100.0	80.70	19.30	0.0
4	92.43	90.14	100.0	100.0	75.44	24.56	0.0
5	90.27	87.67	100.0	100.0	68.42	31.57	0.0

‘Themidav 1.8.5.5’と‘yoda’s protector 1.02’以外の 18 種類のパッカーに関しては，復号ルーチンをすべて正しく分類することができた．

4.4 考察

メモリアドレス距離による実行命令系列のグルーピングでは，‘ExeCryptor 2.3.5’でパッキングされたマルウェアと‘yoda’s cryptor 1.3’でパッキングされたマルウェア 1 種類（agobot3-priv4.exe）のグルーピングに失敗した．どちらも復号ルーチンで命令間のアドレス距離が大きく離れている箇所が 1 つあったため，余分に分割してしまった．しかし評価実験ではオリジナルコードと復号ルーチンが混ざった状態でグルーピングするケースや，部分的なグループが異なるクラスに分離されるケースは見られなかった．そのため，必要以上に分離されたグループであってもコード種別判定は成功し OEP を検出することができた．

kNN 法による各グループのコード種別判定では，評価実験では k=3 のとき Accuracy が 94.05%と最も高く，20 種類中 18 種類のパッカーに対してコード種別判定に成功した．判定に失敗したのは‘Themidav 1.8.5.5’と‘yoda’s cryptor 1.3’の復号ルーチンであった．この 2 種類のパッカーは他のパッカーとは違い，オリジナルコードらしい命令順序で復号処理が行われたと言える．解決策として，特定の実行命令系列や特定のメモリ領域のみに絞って LCS を取得することが考えられる．

kNN 法では二項分類を扱う際，k を奇数にすることで同票数により分離できないという問題を回避できる．また k の値が小さすぎるとノイズの影響を受けやすく，逆に k が大きすぎると他のクラスのデータを含む可能性が高くなる．その点を考慮しても，k=3 は本手法の最適なパラメータとして決定することができる．

5 まとめ

本稿では OEP を検出するための汎用的なアンパッキング手法を提案した．提案手法では命令間のメモリアドレス距離と命令の出現順序に着目して OEP を検出する．本手法はオリジナルコード領域および OEP を一意に特定することができるため，復号時に多数のコード群を生成する多段パッカーに対して非常に有効である．評価実験では，未知のパッカー 20 種類中 18 種類に対して OEP を特定可能であることを確認した．

参考文献

[1] “OllyDbg,” available at `http://www.ollydbg.de/`

[2] M.F.Oberhumer,L.Molnar,and J.F.Reiser, “UPX: Ultimate Packer for eXecutables,” available at `http://upx.sourceforge.net/`.

[3] P.Royal, M.Halpin, D.Dagon, R,Edmonds, and W.Lee, “Polyunpack: Automating the hidden-code extraction of unpack-executing malware,” Proceedings of the 22nd Annual Computer Security Applications Conference, pp.289-300, ACSAC’06, IEEE Computer Security, 2006.

表 3: パッカー別コード種別判定結果

No.	パッカー名	訓練データ	テストデータ	実験結果（k=3）
1	ASPack 2.12	-	✓	○
2	Asprotect2.1	-	✓	○
3	fsg 2.0	-	✓	○
4	Mew11 SE 1.2	-	✓	○
5	uPack 1.1.300	-	✓	○
6	obsidium 1.3.5.4	-	✓	○
7	PECompact 2.64	-	✓	○
8	RLPack 1.16 FullEdition	-	✓	○
9	Hide PX 1.4	-	✓	○
10	Themidav 1.8.5.5	-	✓	×
11	UPX 3.08	-	✓	○
12	WinUpack 0.39 final	-	✓	○
13	yoda's protector 1.02	-	✓	×
14	yoda's cryptor1.3	-	✓	○
15	exe32pack 1.4.2	-	✓	○
16	JDPack 2.00	-	✓	○
17	Packman 0.0.0.1	-	✓	○
18	simplePack 1.11 Method2	-	✓	○
19	EXECryptor 2.3.5	-	✓	○
20	code protector 0.61	-	✓	○
21	acpr pro 1.32	✓	-	-
22	eXPressor 1.5.0.1	✓	-	-
23	Molebox Pro 2.6.4	✓	-	-
24	NsPack 3.4	✓	-	-
25	NsPack 3.7	✓	-	-
26	UPack 0.39	✓	-	-
27	alloy 4.3.21.2005.40	✓	-	-
28	ARMProtector	✓	-	-
29	Armadillo 4.20	✓	-	-
30	Armadillo 3.60	✓	-	-
31	EnigmaProtector 1.16	✓	-	-
32	ExeStealth 2.73	✓	-	-
33	EXEFog 1.1	✓	-	-
34	Ezip 1.0	✓	-	-
35	Ezip1 1.0	✓	-	-
36	Packman 1.0	✓	-	-
37	PEDiminisher 0.1	✓	-	-
38	PEBundle 2.30	✓	-	-
39	VGCrypt 0.75	✓	-	-
40	PELOCKNT 2.04	✓	-	-
41	PETITE 1.3	✓	-	-
42	PETITE 1.4	✓	-	-
43	Scramble.Upx 1.07w	✓	-	-
44	SVK-Protector 1.32 demo	✓	-	-
45	SVK-Protector 1.43	✓	-	-
46	ExePack 1.0	✓	-	-
47	epprotector 0.3 privatespecialbuild	✓	-	-

[4] M.G. Kang, P. Poosankam, andH. Yin, “Renovo: a hidden code extractior for packerexecutables,” Proceedings of the 2007 ACM workshop on Recurring malcode, pp.46- 53,WORM ’07,ACM New York, NY, USA, 2007.

[5] 川古谷 祐平, 岩村 誠, 伊藤 光恭, “OEP 自動検出によるマルウェアアンパック手法,”信学技報,ICSS,vol.110,no.79,pp.13-18,2010.

[6] 伊沢 亮一, 神園 雅紀, 井上 大介, “データ実行防止機能を用いた汎用的なアンパッキング手法の提案,”信学技報, ICSS, vol.113, no.95, pp.73-78, 2013.

[7] 中村 徳昭, 森井 昌克, 伊沢 亮一, 井上 大介, 中尾 康二“実行命令系列の出現順序に着目したOEP特定手法の提案,”信学技報, ICSS, vol.113, no.288, pp.13-18, 2013.

[8] L, Martignoni, M. Christodorescu, and S. Jha, “Omniunpack: Fast, generic, and safe unpacking of malware,” 23rd Annual Computer Security Applications Conference, pp.431-441, ACSAC’07, IEEE Computer Society, 2007.

[9] Robert A. Wagner and Michael J. Fischer, “The string-to-string correction problem,” J.ACM, vol.21, pp.168-173, January 1974.

[10] Li. Sun, Steven. Versteeg, Serdar. Boztas, and Trevor. Yann ,“Pattern Recognition Techniques for the Classification of Malware Packers,” Proceedings of the 15th Australasian conference on Information security and privacy, pp.370-390, ACISP’10, SpringerVerlag, Berlin, Heidelberg, 2010.

[11] “Pin - A Dynamic Binary Instrumentation Tool,” avaiable at `http://www.pintool.org/`.