実行命令トレースに基づく動的パッカー特定手法

川古谷 裕平 †　　岩村 誠 †　　針生 剛男 †

†NTT 情報流通プラットフォーム研究所
180-8585 東京都武蔵野市緑町 3-9-11
{kawakoya.yuhei,iwamura.makoto,hariu.takeo}@lab.ntt.co.jp

あらまし 世の中のマルウェアの多くは静的解析を妨害するためパッカーと呼ばれるツールにより難読化が施されている。このようなマルウェアを解析するためには難読化を解きオリジナルコードを取り出す必要がある。難読化アルゴリズムはパッカー毎に異なるため、正確にオリジナルコードを取り出すにはパッカー毎に対応するのが望ましい。しかし、パッカーには数多くの種類が存在し、既存技術のシグネチャベースでの静的なパッカー特定手法では誤検知が多く発生し正確に特定することが難しい。そこで本論文では、隔離された環境内でマルウェアを実行し、そのマルウェアの実行する特徴的な命令列を元にパッカーを特定する動的なパッカー特定手法を提案する。パッカーを特定することにより利用されている難読化アルゴリズムを把握することができ、効率的にオリジナルコードを取り出すことが可能となる。本提案手法を 8 個の Windows に標準に含まれる実行ファイルを 10 種類のパッカーで難読化を施した 80 個の検体を元にクロスバリデーション方式で評価を行ったところすべての検体のパッカーの特定に成功した。また CCC Dataset2011 の検体に対して本手法を提案した結果も併せて示す。

Dynamic Packer Identification Based on Instruction Trace

Yuhei Kawakoya†　　Makoto Iwamura †　　Takeo Hariu†

†NTT Information Sharing Platform Laboratories
9-11,Midori-Cho 3-Chome Musashino-Shi, Tokyo 180-8585 JAPAN
{kawakoya.yuhei,iwamura.makoto,hariu.takeo}@lab.ntt.co.jp

Abstract Most of the malware in the real world is obfuscated by a packer which is designed for disturbing static analysis. In order to analyze such malware, we first have to unpack the malware and extract the original code from it. The obfuscation algorithm is different for each type of packer so it is better to take a different approach suited for each packer type. However, the fact that a large number of different types of packers exist makes it difficult to correctly identify the type of packer by signature-based identification, which is a common existing method. So, in this paper, we propose a dynamic packer identification method, which executes malware in an isolated environment, collects instruction trace logs, and compares them with signatures of packer-specific instruction sets. Identifying the types of packers and understanding their packing algorithm allow us to effectively unpack obfuscated malware. To evaluate our proposed method, we have conducted a cross validation against 80 sample executables which are generated by packing 8 standard Windows executables with 10 common packers. As a result, we succeeded to identify the types of packers for all 80 executables. Lastly, we show the result of the experiment with CCC Dataset 2011.

1 はじめに

マルウェアの持つ機能の全体像を把握するには逆アセンブラやデバッガなどを利用してマルウェアを静的解析するのが有効である。しかし、近年のマルウェアの多くは静的解析を妨害するためパッカーと呼ばれるツールにより、難読化、解析妨害機能の具備が施されている。これらのマルウェアを解析するためには、解析妨害機能を回避しつつ難読化を解き、マルウェアの本来のプログラムコードであるオリジナルコードを取り出す作業、アンパッキング、が必要になる。アンパッキングには複数の方法がある。

1. デバッガなどを利用し手動で実施
2. パッカーオリジナルのアンパッカーを利用
3. 他の解析者が公開しているアンパッカーツール、デバッガスクリプトを利用
4. パッカー特有のアンパッキング手法を利用
5. ジェネリックアンパッカーを利用

1. は最も柔軟性に富むが人手を介して行うため時間と労力を要し、大量のマルウェアを捌くのには適していない。2. はたとえば UPX などのパッカーの場合、-d オプションを利用することでアンパッキングをツールが行ってくれる。しかし、UPX のようにアンパッキング機能を持つパッカーは稀である。3. は使われているパッカーさえ特定できれば、OpenRCE[1] のフォーラム等で公開されている OllyScript 等を利用することが出来る。4. も同様にパッカーの特定が必要になるが、パッカーを正確に特定することが出来れば、そのパッカーの種類によっては [2] で提案されている展開コードがオリジナルエントリポイントにジャンプする直前のコードを捉える手法などを利用して正確にオリジナルエントリポイントを検出することが可能である。5. はパッカーの種類にとらわれずにアンパッキングを行う手法であり、多くの研究が行われている [3][4][5]。しかし、それぞれの手法で一長一短があり得意とするパッカーの種類は異なっている。

3、4 の手法ではマルウェアに利用されているパッカーの正確な特定が必要とされ、2、5 の手法に関してもパッカーを正確に特定することで既存ツールのアンパッカーの有無を調べたり、パラメータの値を調整することでジェネリックアンパッカーの精度を高めることが可能だと考えられる。しかし、既存技術のシグネチャベースで静的に検知する手法では誤検知が発生してしまい、正確にパッカーを特定することが難しい。

そこで本論文では、検査対象の実行ファイルを隔離された環境で実行し、その実行命令トレースと、予め作成しておいた各パッカーを利用してパッキングした実行ファイルの実行時に現れる特徴的なコードブロックをまとめたシグネチャとを比較することにより、その実行ファイルで利用されているパッカーを特定する動的なパッカー特定手法を提案する。

本提案手法の実験のため、Windows の標準的な実行ファイル 8 種をマルウェアによく利用されるパッカー 10 種でそれぞれパッキングした 80 種類の検体を作成し、クロスバリデーション方式で評価を行った。また同時にこの 80 検体を静的なシグネチャマッチングによるパッカー特定ツールである PEiD[7] を利用し比較を行った。結果として、PEiD ではフォールスポジティブが発生しているのに対して、提案手法では全てのパッカーを正確に特定できた。また、CCC Dataset2011[6] のマルウェア検体のうち 9 種を選び出し、同じく PEiD と提案手法での比較を行い、その結果を示した。

本提案手法を利用することで、マルウェアに利用されているパッカーを正確に特定することが可能となる。これにより、アンパッキングツールやデバッグスクリプトが既に他の解析者により作成され、公開されている場合はそれらの利用を可能とし、パッカー特有のアンパッキング手法の利用も可能とする。さらには、ジェネリックアンパッカーの精度の向上等が見込めるものと考えられる。

2 関連研究

パッカーの種類を特定する手法としてはシグネチャベースで静的に行うパッカー特定手法が存在している。代表的なものとしては PEiD がある。PEiD は特定のバイト列で構成されるシグネチャによりパターンマッチングでパッカーの特定を行っている。特定能力はシグネチャの作りに大きく依存するが、一般的に Web から

手に入るシグネチャを利用すると [8]、誤検知が発生し、精度はあまり高くないと考えられる。

Donghwi らは PEiD のシグネチャがオリジナルエントリポイント付近やセクションヘッダの情報に基づいて作成されている問題点を指摘し、特定のパッカーでパッキングされたバイナリのアンパッキングコード部分のバイト列をシグネチャとする手法を提案している [9]。この提案手法ではアンパッキングコード部分のバイト列と検査対象の検体のバイト列の一致する割合を計算することでパッカーを特定しようと試みている。

2.1 関連研究の問題点

既存研究はシグネチャベースで静的にパッカーの種類を特定する手法であるが、この手法ではシグネチャとして利用できるバイト列候補を取得できる領域が狭いといった問題がある。パッキングされた実行ファイルは主として展開コードとデータにより構成される。データ部分は検体により変動が激しい部分であるのでシグネチャを作成する上での候補バイト列とするには向いていない。そのため、エントリポイント付近の展開コード内のバイト列のみでシグネチャを作成する必要がある。これはシグネチャとして利用できるバイト列の範囲が狭く精度の高いシグネチャを作成するのが困難であると考えられる。

3 提案手法

本章では前章で述べた静的なシグネチャマッチング手法の問題点を解決すべく、検査対象の検体を隔離された環境上で動作させ、その実行命令結果をシグネチャの比較対象とする動的なパッカー特定手法を提案する。検査対象の検体を動作させることで、隠されていたコードの命令列をシグネチャの対象とすることができ、検知精度を向上させることができる。また検査対象と比較するシグネチャは、「同じパッカーでパッキングされた実行ファイルの展開コード部分の命令列は類似する」といったパッカーの特徴を利用し、複数のバイナリを特定のパッカーでパッキングしその展開コード部分の共通部分のコードブロックから作成する。

以下、シグネチャの作成方法、実行命令トレースの取得方法、シグネチャと実行命令トレースを比較してスコアを算出する方法について述べる。

3.1 シグネチャの作成

パッカーの特定に利用するシグネチャはそのパッカーに特有のコードブロックのハッシュ値の集合で構成される。このハッシュ値はコードブロックのオペコード部分のみを利用し、一つのコードブロックに含まれるオペコードの列から MD5 ハッシュ値を計算して求めた。

以下に、シグネチャ作成の式を述べる。まず式に使用する記号を定義する。

- n:パッカーの数
- m:実行ファイルの数
- B:コードブロックの集合
- $P = \{q_1, q_2, ..., q_n\}$:パッカーの集合

パッカー q におけるシグネチャは、パッカー q でパッキングした m 個の検体のコードブロックの共通部分からパッカー q 以外でパッキングした検体の共通のコードブロックの和集合との差分から構成される。パッカー $q_j(1 \leq j \leq n)$ のシグネチャは以下の式から求められる。ここで B_q はパッカー q における展開コード部分のコードブロックの集合とする。

$$SIG_{q_j} = \bigcap_{1 \leq i \leq m} B_{q_1,i} - \bigcup_{q \neq q_j \wedge q \in P} \left(\bigcap_{1 \leq i \leq m} B_{q,i} \right)$$

この SIG_{q_j} を $j = 1...n$ の n 個のパッカーに関してそれぞれ求め、これらをそれぞれのパッカー特有のコードブロックの集合、つまりパッカー特定のためのシグネチャとして利用する。

3.2 実行命令トレースの取得

パッカーの多くは、対象の実行ファイルを難読化、暗号化、圧縮すると共に解析妨害機能を付与することが多い。解析妨害機能にはデバッガなどの解析ツールの存在を検知し挙動を変化させたり、ツールに対して動作を誤魔化したりするため、解析対象の正確な動作を追うことが出来なくなってしまう。そこで本提案手法では、パッキングされた実行ファイルの実行命令トレースを取得するため、耐解析妨害機能を持ったステルスデバッガを利用する [2]。ステルスデバッガでは仮想マシンを利用し仮想的に表現された

ハードウェアの中にトレースを取るための機能を埋め込む。このため、この仮想マシン上で動作するプロセスからは解析機能を持つハードウェアも通常のハードウェアにしか見えず、解析妨害機能から存在を隠蔽することが出来る。またログを取得するために発生してしまうファイル書き込み等のオーバヘッドも仮想マシン上のプロセスには計測させない機能を持っているため、時間を利用した解析妨害機能も回避することができる。

3.3 スコアの算出

調査対象の実行ファイルで利用されているパッカーを調べる際は、この実行ファイルの実行命令トレースを取得し、あらかじめ作成しておいた各パッカーのシグネチャと比較しスコアを算出する。パッカー q と、調査対象の実行ファイルの実行命令トレースのコードブロックの集合 T のスコアは以下の式で計算する。

$$SCORE_{(T,q)} = \frac{T \cap SIG_q}{SIG_q}$$

これはシグネチャを構成するコードブロックの何割程度が調査対象の実行ファイルの実行命令トレースに含まれるかをあらわしている。上記の $SCORE$ を全てのパッカーに対して算出し、この値が最も高いシグネチャのパッカーを調査対象の実行ファイルで利用されているパッカーとする。

4 実験

本章では提案手法の有効性を示すため行った二つの実験について述べる。一つ目の実験はWindows の標準的な実行ファイルの 8 つを選び出し、世の中のマルウェアに比較的頻繁に使われると考えられるパッカー 10 種類を利用してパッキングを施した。この計 80 検体に対して正確にパッカーの特定が可能かをクロスバリデーション方式で実験した。また併せて PEiD でも同様の検体を調査しその結果を比較した。

二つ目の実験は CCC Dataset2011 で提供されている 50 検体のうち 9 個の検体に関して、PEiD と提案手法で検査しその結果を比較した。尚、PEiD のシグネチャは実験 1、2 共に [8] から入手したものを利用した。

4.1 実験 1

実験 1 では 10 種類の異なるパッカーでパッキングを施したオリジナルコードが同じ実行ファイル 10 個を 1 セットとしてクロスバリデーション方式で評価を行った。具体的には、検査対象の実行ファイルのセット以外の検体を利用して 10 種類のパッカーに対するシグネチャを作成し、そのシグネチャを利用して検査対象セットの実行ファイルのパッカーの特定を行った。実験 1 のクロスバリデーションの結果を表 1、表 2 に示す。表 1 は正解シグネチャのスコアであり、表 2 は正解シグネチャ以外で最も高いスコアとそのパッカー名である。

表 1 の結果から各パッカー共に 0.90 以上の高い値を示しておりフォールスネガティブが発生していないことがわかる。また表 2 からは各パッカーの正解シグネチャ以外は低いスコアを示しておりフォールスポジティブが発生していないことを示している。以上のことから本提案手法では、フォールスネガティブ、フォールスポジティブ共に発生しておらず 10 種類のパッカーを正確に特定することが出来たといえる。

表 3 に各パッカーによりパッキングした実行ファイルを PEiD で調査した結果を示す。調査には calc を利用したが他の実行ファイルでも結果は変わらなかった。Aspack に関しては、Aspack と ASProtect の両方のシグネチャが反応しており、フォールスポジティブが発生してしまっている。nPack に関しては 1.x 系のシグネチャが実験に利用したシグネチャセットに含まれていたがここでは反応を示さなかった。

4.2 実験 2

実験 2 では CCC Dataset2011 のマルウェア検体 50 種のうち、実験 1 で作成した 10 種類のパッカーのいずれかの名前が PEiD で検出された 9 検体を利用し、PEiD と提案手法でそれぞれ調査し、結果を比較した。マルウェア検体はステルスデバッガ上で実時間で 3 分程度実行し、実行命令トレースを取得した。各パッカーのシグネチャは実験 1 で利用した 80 検体から作成した。実験 2 の結果を表 4 に示す。

PEiD で UPX だと判別された 10db5...、2641c...、b8f0a... は提案手法でも UPX のシグネチャと完

表 1: 実験 1 の結果 (正解シグネチャのスコア)

パッカー名 バージョン	UPX 3.07	Aspack 2.2	nPack 2.0	FSG 2.0	MEW 11SEv1.2	WinU 0.39final	PEComp 3.02	EXECrypt 2.3.9	Yoda 1.03.3	ASProt 1.5
calc	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.96	1.00	1.00
cmd	0.96	1.00	1.00	1.00	1.00	1.00	0.98	0.91	1.00	0.99
freecell	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.96	1.00	1.00
msiexec	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99
mspaint	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.91	1.00	1.00
telnet	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99
notepad	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99
winmine	1.00	0.97	1.00	1.00	1.00	1.00	1.00	0.94	1.00	0.99

WinU:WinUpack PEComp:PECompact EXECrypt:EXECryptor Yoda:Yoda's Protector ASProt:ASPRotect

表 2: 実験 1 の結果 (正解シグネチャ以外で最も高いスコア)

	UPX	Aspack	nPack	FSG	MEW	WinU	PEComp	EXECrypt	Yoda	ASProt
calc	0.0 N/A	0.0 N/A	0.0 N/A	0.0 N/A	0.023 PEComp	0.0 N/A	0.029 Aspack	0.015 ASProt	0.0 N/A	0.017 EXECrypt
cmd	0.0 N/A	0.0 N/A	0.0 N/A	0.0 N/A	0.023 PEComp	0.0 N/A	0.029 Aspack	0.032 UPX	0.0 N/A	0.029 Aspack
freecell	0.0 N/A	0.0 N/A	0.0 N/A	0.0 N/A	0.0 N/A	0.0 N/A	0.033 UPX	0.033 UPX	0.0 N/A	0.029 Aspack
msiexec	0.0 N/A	0.0 N/A	0.0 N/A	0.0 N/A	0.023 PEComp	0.0 N/A	0.033 UPX	0.033 UPX	0.0 N/A	0.036 EXECrypt
mspaint	0.0 N/A	0.0 N/A	0.0 N/A	0.0 N/A	0.023 PEComp	0.0 N/A	0.029 Aspack	0.033 UPX	0.0 N/A	0.033 UPX,EXECrypt
telnet	0.0 N/A	0.0 N/A	0.0 N/A	0.0 N/A	0.023 PEComp	0.0 N/A	0.033 UPX	0.033 UPX	0.0 N/A	0.029 Aspack
notepad	0.0 N/A	0.0 N/A	0.0 N/A	0.0 N/A	0.142 FSG	0.0 N/A	0.033 UPX	0.033 UPX	0.0 N/A	0.033 UPX
winmine	0.0 N/A	0.0 N/A	0.0 N/A	0.0 N/A	0.023 PEComp	0.0 N/A	0.033 UPX	0.033 UPX	0.0 N/A	0.017 EXECrypt

N/A:該当なし セル内の数値 (上段):スコア セル内の数値 (下段):シグネチャ名

全一致した。このため、PEiD では UPX の 2.X 系だと判別されているが、これらの検体は提案手法で利用したパッカーと同じ 3.07 またはそれに近いバージョンでパッキングされたものだと考えられる。一方で、f6166... の検体では PEiD では 2.X 系だと判別され、提案手法では UPX のスコアは 0.33 とそれほど高い値を示していない。よって、UPX の 2.X 系や UPX の亜種によってパッキングされたものと考えられる。一方、775b8...、a854a...、ee718... の 3 検体に関しては、PEiD では EXECryptor と PECompact の二つのパッカーを示しているが、提案手法では EXECryptor で高いスコアを示している。bc2cd... 検体に関しては、EXECryptor でのスコアが高いが、PECompact での値も 0.41 と比較的高い。ddf5b... の検体では提案手法では EXECryptor で最も高いスコアだが 0.45 程度である。おそらくシグネチャを作成した EXECryptor とはバージョンの異なる EXECryptor、またはその亜種が利用されたものと考えられる。同じ EXECryptor として検出された検体であっても、775b8...、a854a...、ee718... の 3 検体はおそらく同じバージョンであり、bc2cd... と ddf5b... は異なるバージョンの EXECryptor でパッキングされたものだと考えられる。

5 まとめ

本論文ではパッカーの特定手法に関して、既存技術の静的なシグネチャマッチング方式の問題点を指摘し、動的にパッカーの特定を行う手法の提案を行った。本提案手法を 80 種類のパッキングした検体でクロスバリデーション方式で評

表 3: 実験 1 の検体を PEiD で調査した結果

パッカー	シグネチャ
UPX	UPXv20,UPXV200V290,UPX290
Aspack	ASPackv212,ASProtectV2XDLL
nPack	N/A
FSG	FSGv20
MEW	MEW11SEv12,Mew11SEv12
WinUpack	Upackv039final, UpackV037
Yoda	yodasProtectorV1033,yodasProtectorv1033
PECompact	PECompactV2X,PECompactv25,PECompact2xx
EXECrypt	EXECryptor2223,EXECryptor224(devel1),EXECryptor224(devel2) EXECryptor224(devel3),EXECryptor2xx,EXECryptorV22X,EXECryptor2223IAT
ASProt	ASProtect13321,ASProtectv12x

N/A:該当なし

表 4: 実験 2 の結果

検体	シグネチャ名	提案手法
10db5...	UPXv20,UPXV200V290,UPX290	UPX:1.0,ASProt:0.01
2641c...	UPXv20,UPXV200V290	UPX:1.0,EXECrypt:0.05
775b8...	EXECryptor2223,EXECryptor224,EXECryptor2xx EXECryptorV22X, EXECryptor2223IAT,PECompactv2xx	EXECrypt:0.78 UPX:0.06
a854a...	EXECryptor2223,EXECryptor224,EXECryptor2xx EXECryptorV22X,EXECryptor2223IAT, PECompactv2xx	EXECrypt:0.81 UPX:0.06
b8f0a...	UPXv20,UPXV200V290,UPX290	UPX:1.0,ASProt:0.01
bc2cd...	EXECryptor2223,EXECryptor224,EXECryptor2xx EXECryptorV22X,EXECryptor2223IAT, PECompactv2xx	EXECrypt:0.80 PEComp:0.41
ddf5b...	EXECryptor2223,EXECryptor224,EXECryptor2xx EXECryptorV22X,EXECryptor2223IAT, PECompactv2xx	EXECrypt:0.45 UPX:0.03
ee718...	EXECryptor2223,EXECryptor224,EXECryptor2xx EXECryptorV22X,EXECryptor2223IAT, PECompactv2xx	EXECrypt:0.76 UPX:0.06
f6166...	UPXv20,UPXV200V290,UPX290	UPX:0.33,yoda:0.007

左列:検体ハッシュ値の先頭 5 文字　右列のセル内 (上段):最もスコアが高いシグネチャ名と値 右列のセル内 (下段):2 番目にスコアが高いシグネチャ名と値

価を行ったところ全てのパッカーを正確に特定することができた。また CCC Dataset2011 の 9 検体を利用して PEiD との結果の比較を行った。

今後の予定としてより多種のパッカーで評価を行う。その際、バージョン違いやオプションの違い、パッカーの亜種なども評価対象にいれ精度高く検出可能かを評価する。

参考文献

[1] OpenRCE http://www.openrce.org/

[2] 川古谷裕平ら、”ステルスデバッガを利用したマルウェア解析手法の提案” マルウェア対策研究人材育成ワークショップ 2008、2008

[3] 岩村誠ら、 ”コンパイラ出力コードモデルの尤度に基づくアンパッキング手法” マルウェア対策研究人材育成ワークショップ 2008、2008

[4] Yuhei Kawakoya. et al. ”Memory Behavior-Based Automatic Malware Unpacking in Stealth Debugging Environment” 5th IEEEE International Conference on Malicious and Unwanted Software,2010

[5] 塩治榮太朗ら、 ”メモリアクセス値のエントロピーに基づく自動アンパッキング” 信学技法 vol.110, no.475、2011

[6] 畑田充弘ら、”マルウェア対策のための研究用データセット ～MWS2011 Datasets～”、マルウェア対策研究人材育成ワークショップ 2011、2011

[7] PEiD http://www.peid.info/

[8] Bob / Team PEiD signatures http://code.google.com/p/reverse-engineering-scripts/

[9] Donghwi Shin. et al. ”The new signature generation method based on an unpacking algorithm and procedure for a packer detection”, International Journal of Advanced Science and Technology vol.27, 2011