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

岩村　誠 †‡　　伊藤　光恭 †　　村岡　洋一 ‡

†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

{iwamura.makoto,itoh.mitsutaka}@lab.ntt.co.jp

‡ 早稲田大学
169-8555 東京都新宿区大久保 3-4-1

{iwamura@muraoka.info.waseda.ac.jp,muraoka@waseda.jp}

あらまし　本論文では，パッキングされた実行ファイルからオリジナルコードを抽出する新たなアンパッキング手法を提案する．従来のページフォールトハンドラを用いた手法では，メモリアクセス監視の際に TLB を利用していたためその実装が不十分な仮想マシン上で動作させることは不可能であった．これに対し提案手法では，TLB への依存を無くし仮想マシン上での動作を実現した．さらに，多重にパッキングされた実行ファイルであっても，確率モデルを用いてコンパイラが出力したバイナリとしての尤もらしさを算出することで，オリジナルコードを特定可能にした．研究用データセットCCC DATAset 2008のマルウェア検体による実験では，複数のオリジナルコードの候補を抽出するとともに，その候補からオリジナルコードを特定可能なことを示した．

Unpacking based on the likelihood of the compiled code model

Makoto Iwamura†‡　　Mitsutaka Itoh†　　Yoichi Muraoka‡

†NTT Information Sharing Platform Laboratories
9-11, Midori-Cho 3-Chome Musashino-Shi, Tokyo 180-8585 Japan

{iwamura.makoto,itoh.mitsutaka}@lab.ntt.co.jp

‡Waseda University
3-4-1 Ohkubo, Shinjuku-ku, Tokyo, 169-8555 Japan

{iwamura@muraoka.info.waseda.ac.jp,muraoka@waseda.jp}

Abstract　In this paper, we propose a novel unpacking tool which can automatically extract the original code of a packed malware. A conventional tool hooking page-fault handler cannot run on the virtual machine which does not correctly implement TLB. Compared with it, our tool can run on it by more sophisticated monitor, which do not need TLB, for memory access. Additionally, even if the malware were multiple-packed i.e., two or more candidates of the original code were extracted by the previous process, our tool can specify the original code of a malware by calculating the likelihood of the compiled code model from given candidates. Experimental results with the malware sample of CCC DATAset 2008 show that our tool could extract some code regions as the candidates and specify the original code from them.

1　研究の背景と動機

近年マルウェアの種類数が増加の一途を辿っており，国内でも一日当たり数千検体の新種が収集されているとの報告がある [1]．この要因としてはポリモーフィック型マルウェアや，対象プログラムを圧縮しオリジナルコードを隠蔽できるパッカーが多数存在することが挙げられる．特にランタイムパッカーは，ソースコードや開発環境に手を入れることなく，容易にそのプログラムコードを隠蔽することができるため，解析を困難にすることを目的としてマルウェアに適用されるケースが多い．このようにマルウェアが氾濫している状況においては，それら全てに対策を打つことはおろか，対策の優先度を決めることさえ困難になっている．故に膨大な種類のマルウェアを，如何に効率的に管理するかが大きな課題となっており，その解決策のひとつとしてマルウェアの分類技術の研究が進んでいる．マルウェアの分類技術には大きく分けて二つのアプローチが存在する．ひとつはマルウェアの挙動によりそれらを分類する方法である．しかしながら，マルウェアの昨今のトレンドが攻撃者からの指令無しには動かないボットであることを鑑みると，挙動によるマルウェアの分類は非常に難しい．こうした課題を解決するために，マルウェアのプログラムコードに基づく分類技術も研究されている．これはマルウェアのプログラムコードより抽出した N-perms やコールツリー，コントロールフローグラフなどからマルウェア間の類似度を算出し，クラスタリングする手法である [2][3]．ただし，これらの従来研究は何れもアンパッキングや逆アセンブル等の解析が済んだマルウェアを対象としており，プログラムコードに基づくマルウェアの分類を全自動化するためには，以下のアンパッキングの工程を自動化することが必要となる．

- オリジナルエントリーポイントの特定
- オリジナルコードの抽出
- インポートテーブルの再構築
- 実行ファイル用ヘッダの再構築

本論文では，この中でも特にプログラムコードに基づく分類の要となるオリジナルコードの抽出に焦点を当てる．

2　従来技術

実行ファイルがどのパッカーでパッキングされたかを識別し，隠蔽されたオリジナルコードを抽出するツールはいくつも存在する．例えば，PEiD[4] は単純なパターンマッチングによりパッカーを識別することができる．こうしてパッカーを特定することができれば，個別に開発されたアンパッカーを利用し隠蔽されたオリジナルコードを抽出することができる．このアプローチは非常に高速かつ正確にアンパッキングが可能であるが，パッカーを識別でき，尚且つ対応するアンパッカーが存在している必要があるため，新しいアルゴリズムを備えるパッカーが利用された場合には適用できない．

一方，パッカー毎のアルゴリズムに依存しない動的なアンパッキング手法も提案されている．ランタイムパッカーでパッキングされたプログラムは，実行されるとオリジナルコードがメモリ上に展開され，通常 OS に備わるローダが行う処理（動的ライブラリのリンク，リロケーションなど）を行った後に，オリジナルコードが実行される．ほとんどの動的なアンパッキング手法はこの事実を前提として開発されている．OllyBonE[5] や Universal PE Unpacker[6] は事前にオリジナルコードのエントリポイントが含まれるであろう領域に対して，実行ブレークポイントを張り対象プログラムを実行することで，オリジナルコードのエントリポイントを発見する．ただしオリジナルコードのエントリポイントはマルウェアをリンクする際に任意に設定可能であるため，広大なアドレス空間に対してこれを事前に予測することは非常に難しい．この問題を解決するために Saffron[7] や Renovo[8] は，書き込みが生じた領域がその後実行された場合に，当該領域をオリジナルコードとして出力する．しかしながら，この手法にもふたつの課題が残っている．

一つ目は実装上の問題である．Renovo は TEMU と呼ばれる一種のエミュレーションエンジン上に構築されており，そのエミュレーション精度の問題からアンチデバッグ機構により解析できない状況も報告されている．またエミュレーションの性質上，解析速度の低下も免れない．一方，

Saffron は OS のページフォールトハンドラを独自のハンドラに置き換えることで，解析速度の向上を果たしている．具体的には解析対象プロセス空間内の全メモリページに関して PTE (Page Table Entry) 内のスーパーバイザービットを設定することで，全メモリアクセスをフックし，メモリに対する書き込みおよび実行アクセスを監視している．メモリアクセスをフックした後に処理を続行させるためには，PTE の専用キャッシュである TLB (Translation Lookaside Buffer) を用いることで解決している．IA-32 における TLB と PTE の同期処理は，通常システムプログラマに任されており，Saffron ではこの同期処理を敢えて省略することで PTE にはスーパーバイザービットを設定しつつも，TLB ではスーパーバイザービットが設定されていない状態を作り出し，メモリアクセスをフックした後の処理を続行させている．ただしこれにも大きな問題が存在する．一般的に仮想マシンにおける TLB は実機のそれを忠実に再現しているとは限らない．実際，最も著名な仮想マシンである VMware[9] においても命令用 TLB が特権モードを跨ぐ際にフラッシュされるとの報告もある [10]．命令用 TLB が特権モードを跨ぐ際にフラッシュされてしまうと，Saffron の実装では常に PTE に設定したスーパーバイザービットが有効になってしまうため，ユーザプロセスとページフォールトハンドラを行き来する無限ループに陥ってしまう．そのため，マルウェア解析に非常に有効である仮想マシン上での解析が実質不可能となってしまう．

二つ目は多重にパッキングされたバイナリを解析するときに，各層のアンパッキングが完了するたびにオリジナルコード（の候補）が抽出されてしまうことにある．このため事前にタイムアウトを設定しておく等の対処も提案されているが，オリジナルコードを特定するための根本的な対処には至っていない．

3 提案手法

本章では，TLB の実装に依存せずに，ページフォールトハンドラを用いて高速にメモリアクセスを監視しつつ，さらに多重にパッキングされたバイナリであっても，オリジナルコードを特定可能なアンパッキング手法を提案する．提案手法は大きく分けて二つのモジュールから構成される．一つ目はメモリアクセス監視モジュールである．これは Saffron と同様，カーネルドライバとして実装され，独自のページフォールトハンドラを用いることで,「書き込まれた後に実行される」メモリページを，オリジナルコードの候補として出力する．二つ目はオリジナルコード特定モジュールであり，メモリアクセス監視モジュールから出力されたオリジナルコードの候補からオリジナルコードを特定するモジュールである．以下に各モジュールの詳細について述べる．

3.1 メモリアクセス監視モジュール

本モジュールは Saffron と同様の機能を持っている．まず対象プロセス空間内のすべてのメモリページに関して，書き込み監視状態とする．その後，書き込みが発生したメモリページに関して，実行監視状態に遷移させる．そして実行監視状態であるメモリページが実行された際に，そのメモリページが含まれる領域をファイルへ出力し，当該領域を再び監視状態に戻す．メモリアクセス監視機構において提案手法と Saffron とが大きく異なる点は監視方法にある．まず書き込み監視するべきメモリページに対しては，対応する PTE の writable-bit を 0 に設定する．これにより書き込みが発生したときのみページフォルトを引き起こすことができる．一方，実行を監視するべきメモリページに対しては，対応する PTE の XDbit を 1 に設定することで実現する．これには PAE(Physical Address Extensions) が有効になっており，かつ MSR レジスタ内の 11 ビット目を設定する必要があることに注意する．一方，PTE を操作し該当する監視状態に移行させた後に，当該メモリページがページアウトしてしまうと，PTE へ書き込んだ情報が失われてしまう．さらに動的に確保されたメモリページに対しても，それを検出し書き込み監視のために writable-bit を 0 に設定する必要がある．これらの問題は，MiAddValidPageToWorkingSet(カーネル内部の関数) の呼び出しを監視しておくことで，物理メモリへのマッピングが行われる際に，PTE に該当する操作

を行うようにすることで解決できる．また，仮想アドレスが動的に解放される場合は，MiRemoveVad(カーネル内部の関数) が呼び出されるため，この関数の呼び出しを契機に，監視状態を解除することができる．

こうした機能により TLB を用いることなく，監視する必要があるアクセスのみでページフォールトを発生させることができるようになったため，TLB の実装が実機のそれとは異なる VMware 等の仮想マシンであっても，問題なく動作可能となった．また，不要なページフォールトが発生しなくなったことにより速度面のパフォーマンス向上も見込むことができる．

3.2 オリジナルコード特定モジュール

ここでは，アンパッキングの処理ルーチンはコンパイラ出力コードとして尤もらしくなく，オリジナルコードはコンパイラ出力コードとして尤もらしいことを前提として，コンパイラ出力コードを確率モデルにより表現することで，その尤度を算出しオリジナルコードを決定することを目的とする．コンパイラ出力コードのモデル化については逆アセンブルに関する研究において，隠れマルコフモデルを用いた確率モデルが提案されている [11]．提案手法ではこれを参考に命令およびデータを表す二つの状態を用意し，各々において一命令または一バイトデータを出力するモデルを利用する．次にいくつかの記号を定義する．

- N: 入力バイト数.
- $X = x_1^N$: 入力バイト系列.
- $T = t_1^N$: HMM の状態系列.
- $S = \{I, I', D\}$: t_i がとりうる値の集合 S. I は命令の先頭，I' は命令の 2 バイト目以降，D はデータを表す.
- M_θ: パラメータ θ を持つ HMM

逆アセンブルに関する研究では，Viterbi アルゴリズムにより $\mathrm{argmax}_T P(X,T|M_\theta)$ を求めることで，最も尤もらしい逆アセンブル結果を得ていた．一方，Forward アルゴリズムを用いることで，$P(X|M_\theta) = \sum_{all\ T} P(X,T|M_\theta)$ を効率的に求めることも可能である．これは M_θ を一般的なコンパイラで学習しておくことで，入力バイト系列 X が与えられたときの M_θ の尤度，つまりコンパイラ出力コードとしての尤もらしさと捉えることができる．こうして得られた $P(X|M_\theta)$ と，事前の知見なしに求まる $P(X)$[1]の比 $\frac{P(X|M_\theta)}{P(X)}$（尤度比という）を算出し，その値が 1 以下であればコンパイラ出力コードらしくない，1 より大きければコンパイラ出力コードらしい，といった判断が可能となる．ただし，このモデルはあくまで命令の出現頻度および各状態間の遷移確率のみをモデル化しているに過ぎないため，アンパッキングの処理ルーチンであっても尤度が高くなる状況が考えられる．

アンパッキングの処理ルーチンには，コンパイラ出力コードには見られない特徴を持つものがある．そのひとつとして分岐命令の宛先が命令の先頭からずれた場所を指すように見える，といった状況（ブランチバイオレーションと呼ぶ）が挙げられる．この意図は，先頭から順に逆アセンブルを行った際に本来の命令列を隠す（つまり解析を困難にする）ためであり，アンパッキングの処理ルーチンにはこういった分岐命令が頻繁に出現する．一方，通常のコンパイラ出力コードには，宛先が命令の先頭ではない分岐命令は存在しない．したがって，コンパイラ出力コードの尤もらしさを算出する際には，このブランチバイオレーションが起こる逆アセンブル結果を排除するべきである．

ここで分岐命令の先頭と解釈できる入力バイト値を x_i，その宛先を x_j とした場合に，入力系列 X に存在する $\{i,j\}$ の集合を B_X とすると，入力バイト列 X を出力し，ブランチバイオレーションを全く起こさない確率は

$$
\begin{aligned}
&P(X, \forall\{i,j\} \in B_X (t_i \neq I \ or\ t_j = I)|M_\theta) \\
&= P(X|M_\theta) P(\forall\{i,j\} \in B_X (t_i \neq I \ or\ t_j = I)|X, M_\theta)
\end{aligned}
$$

と表現できる．しかし，これを直接算出するには全ての分岐を展開する必要があり，その展開数は分岐のオーバーラップ数に応じて指数関数的に増加するため，計算量の観点から困難である．そこで,「ある分岐命令がブランチバイオレーションを起こさない確率は，その他の分岐命令がブランチバイオレーションを起こさない確率と独立であり，分岐元の状態確率と分岐先

[1] ヌルモデルと呼ばれる．ここでは $P(X) = \frac{1}{256^N}$ とする．

の状態確率もまた独立である」と仮定する．これにより前式は以下のように近似することができる．

$$
\begin{aligned}
& P(X|M_\theta)P(\forall\{i,j\} \in B_X(t_i \neq I \ or \ t_j = I)|X, M_\theta) \\
\approx \ & P(X|M_\theta) \prod_{\{i,j\}\in B_X} P(t_i \neq I \ or \ t_j = I|X, M_\theta) \\
\approx \ & P(X|M_\theta) \prod_{\{i,j\}\in B_X} (1 - P(t_i = I|X, M_\theta)P(t_j \neq I|X, M_\theta))
\end{aligned}
$$

ここで出てくる $P(t_i|X, M_\theta)$ はForward / Backward アルゴリズムにより求められる行列を用いることで容易に算出することができる．また尤度比を求める際に必要となるヌルモデルに関しては，全てのバイト値をデータとして解釈することで，ブランチバイオレーションは全く起こらないことになるため，最終的には以下の尤度比をコンパイラ出力コードの判断基準とする．

$$
\frac{P(X|M_\theta)\prod_{\{i,j\}\in B_X}(1 - P(t_i = I|X, M_\theta)P(t_j \neq I|X, M_\theta))}{P(X)}
$$

具体的には上式が 1 より大きい場合にはコンパイラが出力したコードとし，1 以下の場合はコンパイラが出力したコードではない，と判断する．

4 実験

本章では，研究用データセットCCC DATAset 2008のマルウェア検体を提案手法でアンパッキングした結果について述べる．

アプリケーション	Firefox 3.0.1
コンパイラ	Microsoft C++ Compiler Ver. 15.00.21022.08

表 1: 学習用プログラムとコンパイル環境

提案手法におけるコンパイラ出力コードとしての尤度を算出するために必要な HMM のモデルパラメータに関しては，表 1 によって生成された xul.dll [2]のアセンブリコードを学習データとし，命令状態における出力確率および状態遷移確率を算出した．またデータ状態における出力確率は，プログラムによって大きく異なると考え全シンボルに関して等確率 $\frac{1}{256}$ とした．

実験環境は表 2 の通りであり，マルウェアの検体をゲスト OS 上でアンパッキングした．

[2]Firefox において最も大きな実行バイナリ (約 8.5MB) であり，学習データとして十分なサイズである．

ホスト OS	Windows XP SP3
ホスト CPU	Intel Xeon 2.66GHz
ホストメモリ	3GB
仮想マシン	VMware Server 1.0.4
ゲスト OS	Windows XP SP1
ゲスト CPU 数	1
ゲストメモリ	512MB

表 2: 実験環境

図 1 は，オリジナルコードの候補を抽出した後に，各候補に関してコンパイラ出力コードモデルとヌルモデルの尤度比を算出した結果である．各点はオリジナルコードの候補であり，全部で 230 程度の候補が抽出された．横軸はその候補が抽出された時間（秒），縦軸はその候補の尤度比の対数をとった値となっている．当該検体は起動した後に三回の再起動を繰り返すため，オリジナルコードの候補を表すマーカーをプロセス毎に変えている．図 1 において対数尤度比が 0 を超えた点は，コンパイラ出力コードとして尤もらしいことを表している．

当該検体において最初に対数尤度比が 0 を超えた候補は図 1 における (*1) であり，その領域を逆アセンブルした結果，ボットとしてのプログラムコードが確認された．またその他の対数尤度比が 0 を超える領域は，全て (*1) と同じくボットとしてのプログラムコードを持ち，対数尤度比が 0 以下の領域は，全てアンパッキング処理中のルーチンであることも確認された．

図 1 からは他にもいくつかの興味深い結果を読み取れる．まず一番目のプロセス（PID=364H）には，全くオリジナルコードが出現していないことが分かる．このマルウェアに用いられたパッカーによりパッキングされた実行ファイルは，デバッガによる追跡等を困難にするため，アンパッキング前に一度再起動を行うプログラムに変更されると考えられる．また二番目のプロセスでオリジナルコードが出現した後に，再びプロセスが再起動しているが，これはマルウェア自身がシステムディレクトリへコピーされ，そこから再起動されているためである．つまり一度目の再起動はパッカーによりもたらされたものであり，二度目の再起動はアンパッキングされたマルウェアによりもたらされたものであることが分かる．その後，システムディレクトリ内で

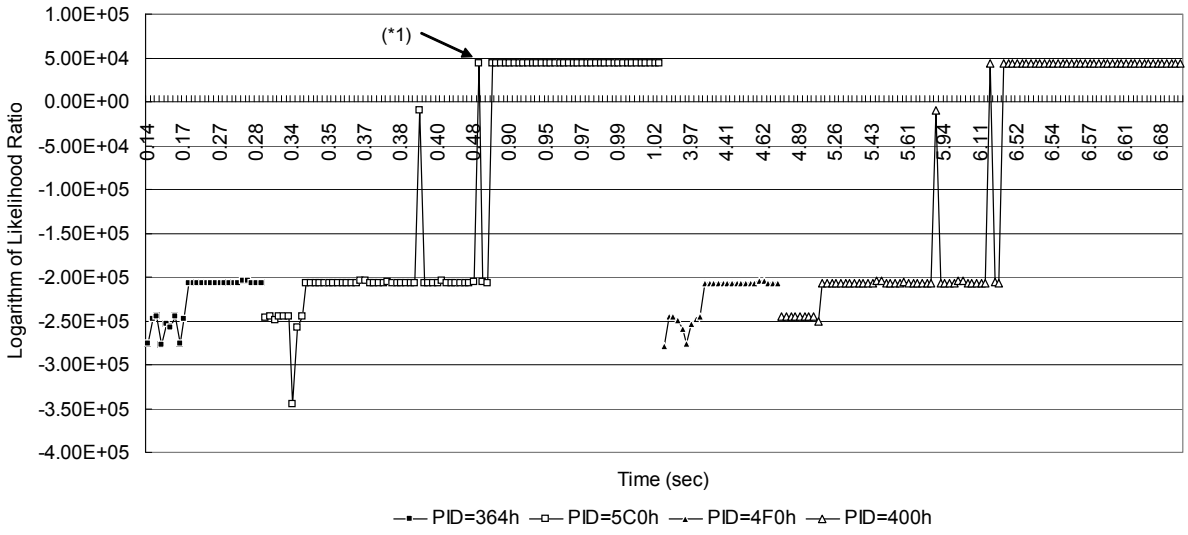

図 1: オリジナルコード候補の対数尤度比

三番目のプロセスが起動し，さらにパッカーの機能により四番目のプロセスが起動している．実際，この四番目のプロセスではじめてボットとしての本来の活動が開始されていることも別途確認できた．

5 まとめ

本論文ではパッキングされた実行ファイルからオリジナルコードを自動的に抽出可能なアンパッキング手法を提案した．従来の OS のページフォールトハンドラを用いたアンパッキング手法では，TLB の実装が不十分な仮想マシン上で動作させることはできなかった．これに対し，提案手法では PTE に存在する XDbit を利用することで TLB への依存をなくし，その結果，仮想マシン上での動作を可能にした．また，従来技術では「書き込まれた後に実行された」領域をオリジナルコードとして抽出していたが，多重パッキングされた実行ファイルの場合，オリジナルコードではないアンパッキング中のコードが抽出されてしまう問題があった．これに対して本論文では，得られたオリジナルコードの候補に関して，確率モデルによってコンパイラ出力コードの尤もらしさを算出し，オリジナルコードを特定できる新たなアンパッキング手法を提案した．本手法を利用することで，マルウェア解析作業の簡略化はもとより，プログラムコードに基づくマルウェア分類の全自動化に大きく貢献できるものと期待している．

参考文献

[1] Cyber Clean Center, https://www.ccc.go.jp /report /200807 /0807monthly.html.

[2] Md. Enamul Karim, Andrew Walenstein, Arun Lakhotia, Laxmi Parida, Malware phylogeny generation using permutations of code. European Research Journal of Computer Virology 1, 1-2 (Nov. 2005) 13–23.

[3] E. Carrera and G. Erdelyi, Digital Genome Mapping - Advanced Binary Malware Analysis, In Proc. Virus Bulletin Conf., 2004, pp. 187–197.

[4] PEiD, http://peid.has.it/.

[5] OllyBonE, http://www.joestewart.org/ollybone/.

[6] Using IDA to deal with packed executables, http://www.hex-rays.com/idapro/unpack_pe/.

[7] D. Quist, V. Smith, Covert Debugging: Circumventing Software Armoring, Blackhat USA 2007 / Defcon 15, Las Vegas, NV.

[8] Min Gyung Kang, Pongsin Poosankam, Heng Yin, Renovo: a hidden code extractor for packed executables, In Proc. WORM '07: Proceedings of the 2007 ACM workshop on Recurring malcode, 2007, pp. 46–53.

[9] VMware: Virtualization via Hypervisor, Virtual Machine & Server Consolidation - VMware, http://www.vmware.com/.

[10] Tron: He Fights for the User Alan Bradley, http://www.openrce.org /repositories /users /AlanBradley /Tron-TC8.pdf.

[11] 岩村誠, 伊藤光恭, 村岡洋一, 隠れマルコフモデルに基づく新規逆アセンブル手法, In Proceedings of the 2008 IEICE General Conference.