IAT エントリ格納場所の特定方法

岩村 誠 †　　川古谷 裕平 †　　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

{iwamura.makoto,kawakoya.yuhei,hariu.takeo}@lab.ntt.co.jp

あらまし　本論文では，アンパッキング後のマルウェアにおけるインポート・アドレス・テーブル（IAT）のエントリ格納場所を特定する手法を提案する．従来の手法は，マルウェアの逆アセンブル結果から IAT を利用する機械語命令を探し出すことで IAT 格納場所を推定していた．しかし Windows 用コンパイラは可変長の機械語命令とデータが混在するバイナリを出力する傾向にあるため，正確な逆アセンブル結果を得ることは難しい．こうした問題に対し提案手法は，IAT エントリ格納場所の候補について真の IAT エントリ格納場所である確率を算出し，当該確率が十分に高いアドレスを抽出する．これにより従来技術と比較し，精度よく IAT エントリの格納場所を特定することを可能にした．

Specifiying the Addresses of IAT Entries

Makoto Iwamura†　　Yuhei Kawakoya†　　Takeo Hariu†

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

{iwamura.makoto,kawakoya.yuhei,hariu.takeo}@lab.ntt.co.jp

Abstract　We propose a novel approach, which accurately specifies the addresses of Import Address Table (IAT) entries in unpacked malware. Existing approaches specify the IAT entry addresses by finding the machine code instructions that uses an IAT entry in the result of disassembly. However, since a compiler for Windows tends to output a binary mixing variable length instructions and data, it is difficult to correctly disassemble unpacked malware. For solving the problem, our approach calculates the probabilities that each address in malware points to an IAT entry, and then finds highly probable IAT entry addresses.

1　はじめに

マルウェアの機能を把握するためには，マルウェアが利用する外部関数（Win32 API 等）の特定が重要となる．Windows 用の実行ファイルにおいて，暗黙的なリンクにより外部関数を利用する際には，外部関数のアドレスが IAT（Import Address Table）に格納される．通常，IAT は PE（Portable Executable）ヘッダによりその場所を特定することができる．

一方パックされたマルウェアには，パック以前のプログラムコード（以下，オリジナルコード）用の PE ヘッダが含まれない場合がある．これは，PE ヘッダが OS の備えるローダに対する情報であり，パックされたマルウェアのロードを担うのは，独自のローダであることに起因する．つまり，マルウェアのオリジナルコードのロードに必要な情報は，必ずしも PE ヘッダ

のフォーマットに従う必要はなく，パッカーが用意する独自の構造で管理しておけばよい．実際，オリジナルコード用の PE ヘッダを削除する難読化手法 [1] も存在する．こうした状況では，オリジナルコードが利用する IAT エントリの格納場所を特定することが困難になる．

本稿ではこうした課題を解決するために，オリジナルコードの PE ヘッダが存在しない状態であっても，IAT エントリの格納場所を精度よく特定する手法を提案する．

2 従来技術

ここでは，オリジナルコードが利用する IAT エントリ格納場所を特定する従来手法について説明する．

2.1 0xFF, 0x15/0x25 の探索

通常，IAT エントリは間接分岐命令により利用される．この手法は，間接分岐命令と解釈できるバイト列を網羅的に探し出し，当該命令の分岐先が格納されているアドレスを，IAT エントリ格納場所として抽出する [2, 3]．コンパイラが出力するプログラムコードにおいて，IAT エントリを参照する機械語命令は，主に以下の間接分岐命令である．

- JMP [IAT エントリのアドレス]
- CALL [IAT エントリのアドレス]

これらの間接分岐命令は，0xFF, 0x25（JMP の場合）もしくは 0xFF, 0x15（CALL の場合）から始まり，その直後に 4 バイトの IAT エントリのアドレスが続く．そこでこの手法では，オリジナルコードから 0xFF, 0x25 もしくは 0xFF, 0x15 を探し出し，後続の 4 バイトを IAT エントリ格納場所として出力する．当然のことながら，0xFF, 0x25（もしくは 0xFF, 0x15）が間接分岐命令ではなくオペランド部やデータとして存在する場合は，IAT エントリ格納場所ではないアドレスが誤って出力されてしまう（この事象を False positive とする）．通常のプログラムの場合，この False positive の割合は単純計算で $\frac{1}{2^{16}} \times 2$ 程度（32KB に 1 つの割合）と非常に低いが，ランタイムパッカーの実装方法によっては，False positive を意図的に誘発することが可能となる．

その説明の前に，まずランタイムパッカーの特徴を整理しよう．ランタイムパッカーは，オリジナルの実行ファイルを入力として受け取り，実行ファイル全体（もしくはコードセクション等）に対し何らかのエンコードを行う．その後，エンコードされたバイナリとこれを復元する特殊なローダをつなぎ合わせ，新たな実行ファイルを生成する．このようにランタイムパッカーは実行ファイルを入力として受け付けるため，マルウェア作者は自身の開発環境を変更することなく，マルウェアのオリジナルコードを隠ぺいすることができる．一方，この実行ファイル生成（コンパイル）とパッキングの分業は，ランタイムパッカーによるオリジナルコードの変異（命令の置換や順序入れ替え等）を困難にする側面もある．これは実行ファイルの正確な逆アセンブルが困難であることに依る．この理由として，Windows 向けのコンパイラは実行ファイルの削減やキャッシュヒット率の向上を見込み，機械語命令とデータを混在させる傾向にあることが挙げられる．また IA-32[4] の機械語命令が可変長であることも，逆アセンブルを難しくしている一因である．このためオリジナルコードの変異は，基本的にオリジナルエントリポイントから辿れる範囲等，非常に限定的であると考えられる．

一方で，Windows の実行ファイルにはリロケーション・テーブルと呼ばれる情報を持たせることができる．これは，想定されるベースアドレスではない場所に実行ファイルがロードされる場合に，実行ファイル中の変更すべき箇所を記録しておくテーブルである．具体的には，このテーブルの内容は絶対アドレスを利用する機械語命令のオペランドを指している．ローダは，このリロケーション・テーブルを参照するだけで，絶対アドレスの参照先を実際にロードされるベースアドレスに合わせた値に補正することができる．これは，正確な逆アセンブル結果を必要とせずに，機械語命令のオペランド部

分を変更することができることを意味する．

ここで，False positive 誘発の説明に戻ろう．まず，実行ファイルのベースアドレスが0x00400000 のときの，以下の機械語命令について考えてみる．

```
MOV EAX, DWORD PTR DS:[0x0040BEEF]
```

この機械語命令のバイト列表現は，0xA1, 0xEF, 0xBE, 0x40, 0x00 となる．次に，実行ファイルのベースアドレスが 0x15FF0000 に変更されたときの，上記の機械語命令を見てみよう．

```
MOV EAX, DWORD PTR DS:[0x15FFBEEF]
```

この機械語命令のバイト列表現は，0xA1, 0xEF, 0xBE, 0xFF, 0x15 となり，機械語命令中に 0xFF, 0x15 が出現していることが分かる．つまり，実行ファイルのベースアドレスを 0x15FF0000（もしくは 0x25FF0000）とすることで，実行ファイルの先頭 64KB の領域（たとえばグローバル変数）にアクセスする機械語命令のオペランド内に，0xFF, 0x15（もしくは 0xFF, 0x25）を出現させることが可能になる．これは正確な逆アセンブル結果を把握していないランタイムパッカーであっても，IAT エントリの格納場所を錯乱させることが可能になることを意味する．また，リロケーション･テーブルも PE ヘッダの一部であるため，パッカーは独自の構造でこれを管理することで，解析者からリロケーション情報を隠ぺいすることができる．

2.2 逆アセンブラによる間接分岐命令の探索

この手法は事前に逆アセンブラを利用することで，間接分岐命令を探索し，そこで利用される IAT エントリを抽出する．逆アセンブラとしては，IDA Pro[5] 等が存在する．ただ，前述したように Windows の実行ファイルを正確に逆アセンブルすることは難しい．特に，パックされたマルウェアの場合，逆アセンブルのヒントとなる PE ヘッダが消失している場合もあり，正確な逆アセンブル結果は望めない．このため IAT エントリ格納場所の抽出精度にも課題が残る．

表 1: プログラムコード

アドレス	ニーモニック
0x0005	CALL 0x000F
0x000F	CALL 0x1004
0x001F	JMP 0x1004
0x002F	CALL [0x2004]
0x1004	JMP [0x2004]
0x1100	CALL [0x20AA]

3 提案手法

オリジナルコードが利用する IAT エントリには，以下の特徴がある．

- IAT エントリの参照元は複数存在する場合がある
- IAT エントリの参照元は Thunk とよばれる間接 JMP 命令の場合があり，Thunk は直接 CALL 命令により呼び出される．

また，隠れマルコフモデル（HMM）に基づく確率的逆アセンブル手法 [6, 7] では，Forward/Backward アルゴリズムにより，対象バイナリ中の各バイト値が命令の先頭になる確率を算出することができる．本稿ではこうした IAT エントリの特徴と，確率的逆アセンブル手法により求められる各分岐命令が機械語命令と解釈される確率を利用することで，IAT エントリの抽出精度を向上させる手法を提案する．

図 1 に本手法の処理概要を示す．本手法は，

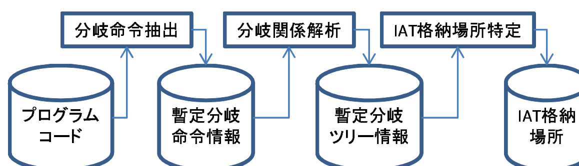

図 1: 提案手法の処理概要

まずプログラムコード中に含まれる分岐命令と解釈できるバイト列を探し出す．そして，確率的逆アセンブル手法により当該バイト列が機械語命令である確率を算出し，暫定分岐命令情報を作成する．例えば，表 1 の分岐命令と解釈できるバイト列を含むプログラムコードがあったとしよう．CALL/JMP X は X へ分岐する直接分岐命令で，CALL/JMP [X] は X に格納され

表 2: 暫定分岐命令情報

アドレス	種別	命令確率	アドレス
0x0005	直接 CALL	0.2	0x000F
0x000F	直接 CALL	0.4	0x1004
0x001F	直接 JMP	0.6	0x1004
0x002F	間接 CALL	0.8	0x2004
0x1004	間接 JMP	0.1	0x2004
0x1100	間接 CALL	0.4	0x20AA

たアドレスへ分岐する間接分岐命令を表す．このプログラムコードに対する暫定分岐命令情報は，例えば表 2 のようになる．次に暫定分岐命令情報を参照し，間接分岐命令からその呼び出し元となる分岐命令を探索することで，例えば図 2 のような暫定分岐ツリー情報を生成する．この暫定分岐ツリーは，間接分岐命令の分岐先

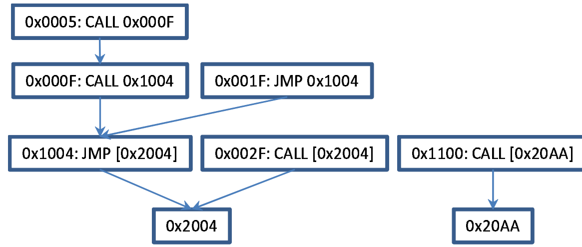

図 2: 暫定分岐ツリー情報

格納アドレス（つまり IAT エントリの候補）を根とする．

こうして得られた暫定分岐ツリーから，分岐先格納アドレスの候補（例えば図 2 では 0x2004 と 0x20AA）に着目し，それらが真の分岐先格納アドレスである確率を算出する．ここで，図 2 において「0x2004 が真の分岐先格納アドレスではない」ときを考えてみよう．0x2004 が分岐先格納アドレスでないためには，その参照元となる JMP [0x2004] および CALL [0x2004] が機械語命令として解釈されてはいけない．また，JMP [0x2004] が機械語命令として解釈されない状況では，JMP 0x1004 と CALL 0x1004 も機械語命令として解釈されてはいけない．これは，機械語命令として解釈されない箇所を分岐先とする分岐命令が存在しないためである．したがって,「0x2004 が真の分岐先格納アドレスではない」ということは，0x2004 と同じ暫定分岐ツリー内に含まれるすべての分岐命令は，機械語命令として解釈されないことを意味する．

ここで，入力となるオリジナルコードのバイト列を X，確率的逆アセンブル手法で用いる HMM のモデルパラメータを θ，分岐先格納アドレスの候補 r が真の分岐先格納アドレスである事象を E_r，その確率を $P(E_r)$ とする．また r と同じ分岐命令ツリーに含まれる分岐命令を $b_{r,i}(1 \leq i \leq N_r, N_r$ は r と同じ暫定分岐ツリーに含まれる分岐命令数$)$ とし，$b_{r,i}$ が機械語命令として解釈されない事象を $F_{b_{r,i}}$，その確率を $P(F_{b_{r,i}})$ とする．前述の暫定分岐ツリーの特徴から，X, θ のもとで r が真の分岐先格納アドレスである確率は，以下の通りである．

$$
\begin{aligned}
P(E_r|X,\theta) &= 1 - P(\bigcap_{i=1}^{N_r} F_{b_{r,i}}|X,\theta) \\
&= 1 - \frac{P(X, \bigcap_{i=1}^{N_r} F_{b_{r,i}}|\theta)}{P(X|\theta)} \qquad (1)
\end{aligned}
$$

ここに出てくる $P(X|\theta)$ は，X が与えられた際に Forward アルゴリズムで算出し，全ての r について再利用できる．一方，$P(X, \bigcap_{i=1}^{N_r} F_{b_{r,i}}|\theta)$ の算出は，全ての $b_{r,i}(1 \leq i \leq N_r)$ に割り当てる状態を，データもしくは機械語命令の 2 バイト目以降とした上で，Forward アルゴリズムにより X の出力確率を計算することに他ならない．つまり $\{b_{r,i}|1 \leq i \leq N_r\}$ が変化するたびに（分岐先格納アドレスの候補 r ごとに），対象となる X の出力確率を算出する必要がある．確率的逆アセンブル手法の計算量は，対象となるプログラムコードサイズを $|X|$ とすると $O(|X|)$ となる．そのため，全ての分岐先格納アドレスの候補数を $|R|$，暫定分岐ツリーの平均ノード数を $\bar{N}$ とすると，全ての $P(E_r|X,\theta)$ を求めるには $O(|X|\bar{N}|R|)$ の計算量が必要となってしまう．ここで，計算量を削減するために $F_{b_{r,i}}(1 \leq i \leq N_r)$ が互いに独立であると仮定しよう．すると $P(E_r|X,\theta)$ は以下のように算出できる．

$$
P(E_r|X,\theta) \approx 1 - \prod_{i=1}^{N_r} \frac{P(X, F_{b_{r,i}}|\theta)}{P(X|\theta)} \qquad (2)
$$

式 2 中の $P(X, F_{b_{r,i}}|\theta)$ は，X に関する Forward/Backward アルゴリズムの実行結果を保持しておくことで，O(1) で算出できる．よってこの近似により，全ての $P(E_r|X,\theta)$ を求める計算量を $O(|X| + \bar{N}|R|)$ に抑えることができる．

本提案手法では，こうして r ごとに算出された $P(E_r|X,\theta)$ が 0.5 以上のときに，当該 r を IAT エントリ格納場所として出力する．

表 3: ベースアドレス 0x00400000

手法	TP	TN	FP	FN	PPV	NPV	MCC
Exhaustive	343	0	18	0	0.9501	NaN	0.0000
IDA Pro	293	18	0	50	1.0000	0.2647	0.4755
Probabilistic	343	18	0	0	1.0000	1.0000	1.0000

表 4: ベースアドレス 0x15FF0000

手法	TP	TN	FP	FN	PPV	NPV	MCC
Exhaustive	343	0	2,839	0	0.1078	NaN	0.0000
IDA Pro	293	2,839	0	50	1.0000	0.9827	0.9162
Probabilistic	343	2,829	10	0	0.9717	1.0000	0.9840

4 実験

ここでは以下の三つの手法について，IAT エントリ格納場所の特定精度を評価する．

- 0xFF, 0x15/0x25 の網羅的な抽出による手法（Exhaustive）
- IDA Pro の逆アセンブル結果から間接分岐命令を特定する手法（IDA Pro）
- 提案手法（Probabilistic）

評価対象とするプログラムコードは，アンパック後のメモリイメージを想定し，メモリ上に展開され PE ヘッダが存在しない状態のオリジナルコードとする．正解データは，パックされていない状態の実行ファイル（PE ヘッダを含んだ状態）を IDA Pro 6.0 で逆アセンブルした結果から生成しておく．この正解データを生成するため，MWS 2011 Datasets[8] のマルウェア検体のうち，パックされていない 5 つの検体（ハッシュ値先頭 2 バイトが 5e2d, 15a4, 542a, 7586, ae58 の検体）を利用した．また，提案手法で用いる HMM のモデルパラメータは，Firefox 3.0.1 に含まれる xul.dll を Microsoft C++ Compiler(Ver. 15.00.21022.08) でコンパイルした結果から学習した．

まず表 3 に，対象プログラムコードのベースアドレスが 0x00400000 であった場合の評価結果を示す．表の列は左から順に True positive 数 (TP)，True negative 数 (TN)，False positive 数 (FP)，False negative 数 (FN)，Positive predictive value(PPV)，Negative predictive value(NPV)，Matthews correlation coefficient(MCC) となっている．ここで True negative とは，0xFF, 0x15/0x25 の網羅的な抽出により得られた IAT エントリ格納場所の候補のうち，IAT エントリ格納場所ではない箇所を特定できたことを示す．また TP, TN, FP, FN は，5 検体に関する総計である．PPV/NPV/MCC は，それぞれ以下の式で表すことができる．

$$PPV = \frac{TP}{TP + FP}$$

$$NPV = \frac{TN}{TN + FN}$$

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

MCC は二値分類器の評価指標として用いられ，TP, TN, FP, FN の相関係数である．具体的には 1 が完全に正しい予測，0 がランダムな予測，-1 が全く逆の予測を意味する．Exhaustive アプローチは，全ての IAT エントリ格納場所の候補を出力する．このため False negative は存在せず，正解となる 343 箇所の IAT エントリ格納場所を全て網羅できている．一方で 0xFF, 0x15/0x25 というバイト列が，プログラムコード中のデータ部分等に 18 箇所存在するため，その数だけ False positive が発生している．ただ，正解数と比較すると十分少ない数であるため，PPV は 95.01%と高い値を示している．対照的に IDA Pro を用いた手法では，False negative が 50 箇所で発生している．IDA Pro は Recursive traversal と呼ばれるアルゴリズムに基づいている．この Recursive traversal はプログラムのエントリポイントや，頻出命令列を起点とし

表 5: ベースアドレス 0x25FF0000

手法	TP	TN	FP	FN	PPV	NPV	MCC
Exhaustive	343	0	2,839	0	0.1078	NaN	0.0000
IDA Pro	293	2,839	0	50	1.0000	0.9827	0.9162
Probabilistic	342	2,837	2	1	0.9942	0.9996	0.9951

てプログラムの制御フローを辿りながら逆アセンブルを進める．このため動的に分岐先が決まる状況では，その分岐先が機械語命令として解釈されない場合がある．こうした理由から，IDA Pro は False negative を生じさせやすい傾向にあると考えられる．一方提案手法では，18 箇所の 0xFF, 0x15/0x25 というバイト列に惑わされずに，完全に正確な予測を行えていることが分かる．

次に，対象となるプログラムコードのベースアドレスを 0x15FF0000 としたときの評価結果を表 4 に示す．このデータセットでは，0xFF, 0x15 というバイト列が機械語命令のオペランド部分に頻出しており，その数は 2,839 箇所に及ぶ．このため Exhaustive アプローチの PPV は 10.78%まで急落している．これに対し IDA Pro はベースアドレスの影響を全く受けず，0x00400000 のときと同じ False positive/negative 数となっている．また提案手法では 10 箇所の False positive が発生しているが，IDA Pro との MCC の比較では十分に優れた抽出精度となっている．

さらに表 5 はベースアドレスを 0x25FF0000 とした場合である．Exhaustive アプローチと IDA Pro はベースアドレスが 0x15FF0000 のときと全く変わらない抽出精度である．一方，提案手法では 0x15FF0000 のときと比較し，MCC が 98.40%から 99.96%へと向上していることが分かる．

5 まとめ

パックされたマルウェアに関して，オリジナルコードの IAT エントリ格納場所を特定することは，その機能を把握するために重要な作業となる．従来の手法では，不正確な逆アセンブル結果に基づき IAT エントリを特定していたため，その特定精度に課題があった．本稿では，IAT エントリ格納場所の参照元が複数ある場合等に着目し，確率的逆アセンブル手法により算出される機械語命令の確率を利用することで，精度よく IAT エントリ格納場所を特定する方法を提案した．また実験により，提案手法が従来手法を上回る特定精度であることを示した．今後は IAT エントリが指す外部関数を特定することで，IAT 再構築を自動化する手法を検討していく．

参考文献

[1] Masaki Suenaga, A Museum of API Obfuscation on Win32, white paper, Symantec, November, 2009.

[2] Paul Royal, Mitch Halpin, David Dagon, Robert Edmonds, Wenke Lee, PolyUnpack: Automating the Hidden-Code Extraction of Unpack-Executing Malware, In Proceedings of ACSAC '06, Washington, DC, USA, 289-300.

[3] Daniel Quist, Lorie Liebrock, and Joshua Neil, Improving antivirus accuracy with hypervisor assisted analysis, J. Comput. Virol. 7, 2 (May 2011), pp.121–131.

[4] Intel Corporation, Intel 64 and IA-32 Architectures Software Developer's Manual, `http://www.intel.com/products/processor/manuals/`.

[5] Hex-Rays, IDA Pro Disassembler, `http://www.hex-rays.com/idapro/`.

[6] 岩村誠, 伊藤光恭, 村岡洋一, コンパイラ出力コードの尤度に基づくアンパッキング手法, MWS2008, 2008, pp.103–108.

[7] Makoto Iwamura, Mitsutaka Itoh, Yoichi Muraoka, Towards Efficient Analysis for Malware in the Wild, In Proceedings of IEEE ICC 2011, June 2011.

[8] 畑田充弘, 中津留勇, 秋山満昭, マルウェア対策のための研究用データセット〜MWS 2011 Datasets〜, MWS2011, 2011.