機械語命令列の類似性に基づく自動マルウェア分類システム

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　本稿では，機械語命令列の類似性に基づく自動マルウェア分類システムを提案する．本システムは，ページフォールトハンドラを利用した汎用アンパッキング手法，アンパッキングされたマルウェアのメモリイメージから機械語命令列を特定する確率的逆アセンブル手法，そして機械語命令列の類似性に基づくマルウェアの分類手法により構成され，アンパッキングから分類までの一連の作業を全自動化する．実験では，京都大学において収集されたマルウェア検体および研究用データセット CCC DATAset 2009 におけるマルウェア検体を分類し，マルウェアの全体像を把握するために要する解析作業量を大幅に削減できることを明らかにした．

Automatic Malware Classification System based on Similarity of Machine Code Instructions

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　We propose an automatic malware classification system based on similarity of machine code instructions. Our classification system has three modules, which are the unpacker using customized page-fault handler, the probabilistic disassembler and the classifier based on a similarity of machine code instructions. Experimental results with the malware samples provided by Kyoto University and CCC DATAset 2009 malware samples show that our system can drastically reduce the quantity of reverse-engineering work to reveal the entire view of malware.

1　研究の背景と動機

昨今の多くのマルウェアは，一般的なソフトウェアと同様，バグ改修や機能改良といった工程を円滑に実施するために，C や C++といった高級言語で開発されている [1]．また，ウィルス対策ソフトによる検出から逃れるために，パッカーと呼ばれる一種の圧縮ツールにより，同じ機能を持ちながらも外観が異なる形式で大量に生産されている [2]．このような開発サイクルにより，近年のマルウェア数は増加の一途を辿り

[3]，その解析作業量も急増，さらにはマルウェアの脅威やトレンドを把握することすら困難にしている．こうした課題の解決策として，マルウェアの分類技術の研究が進んでいる．これには大きく分けて 2 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分類する方法 [4] である．たとえば，システムコールの呼び出し履歴や，ネットワーク・ファイルシステム等のシステムリソースへのアクセス履歴を収集できる環境上でマルウェアを動作させ，得られた動作履歴を元にマルウェアを分類する．この方法は，マルウェアの挙動を把握できる実行環境さえ用意できればよく，リバースエンジニアリング等の高度な作業を必要としない．一方，ボットのような攻撃者からの指令無しには動作しないマルウェアや，ある決まった時刻にしか動作しないマルウェアに関して，その挙動を網羅的に抽出することは非常に難しい．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から抽出した特徴に基づきマルウェア間の類似度を算出し，マルウェアを分類する手法 [5][6][7] も存在する．これは挙動による分類とは異なり，マルウェアが潜在的に備える機能を踏まえつつ分類できる．しかしながらこれを全て自動化するには，アンパッキングや逆アセンブルといった技術的な課題も多い．またプログラム構造の類似性を正確に算出するには多大な計算コストを要する．

本稿では，プログラムコードの類似性を機械語命令単位で高速に算出する手法を提案する．また提案手法に加えて，これまで我々が研究開発してきたアンパッキング手法および逆アセンブル手法を組み合わせて構築した自動マルウェア分類システムについて述べる．

2　従来研究

マルウェアのプログラムコードに基づく分類に関する従来研究は，プログラムコードの特徴として何に着目しているかで整理することができる．N-gram に基づくアプローチ [5] では，まずマルウェアの逆アセンブル結果からオペコード列を抽出し，連続する N 個のオペコード列（N-gram）がそれぞれ何回出現したかを特徴ベクトルとする．この特徴ベクトルをマルウェアの特徴とし，マルウェアの非類似度をコサイン距離として定義する．N-gram は N 個のオペコード列の順序が保存され，命令単位での並べ替えが生じると異なる N-gram として扱われてしまう．そのため，N-perm と呼ばれるオペコード列の順序関係を考慮しない特徴を利用する方法も提案されている．N-gram/N-perm を利用した手法の特長は，ベクトル間のコサイン距離としてマルウェアの類似度が定義されるため非常に高速に処理可能なことである．一方でマルウェア間の比較対象が N-gram/N-perm の出現回数という一種の統計情報であるため，実際に変化のあった箇所を抽出することは難しくなる．他にもプログラムのベーシック・ブロックに着目した手法 [6] も提案されている．ベーシック・ブロックとは，分岐命令･分岐先命令を含まない連続した命令列である．まずマルウェアの逆アセンブル結果から，プログラムコードをベーシック・ブロックに分割する．そしてベーシックブロック単位でレーベンシュタイン距離を算出し，それをマルウェアの非類似度とする．またベーシックブロック単位での並べ替えに対応するために，転置インデックスまたはブルームフィルタを用いる手法も提案されている．具体的には，まずサンプルとなるマルウェアの全ベーシックブロックをデータベースへ格納する．その後，対象となるマルウェアのベーシックブロックがデータベース内に存在したか否かのビットベクトルを求め，それをマルウェアの特徴とする．この手法もまた高速に類似度を算出することが可能であるが，動的に分岐先が決まる場合などは，そもそもベーシック･ブロックの抽出が困難な場合もある．また，レジスタのカラーリング程度の変化であっても，全く異なるベーシック･ブロックとして識別されてしまうため，必要以上にマルウェア間の距離が長くなる可能性が残る．一方でプログラムのコールツリーを特徴とし，類似性を求める手法 [7] も存在する．コールツリーは，ソースコード上のプログラム構造が反映されるため，コンパイラの変更に強いと言われている．しかしながら，インライン展開等の関数を跨ぐ最適化が施されると，大幅にマルウェア間の距離が変化してしまう．またツリー構造の厳密な比較は計算コストが非常に高い．このため，マルウェアの IAT(Import Address Table) を再構築した状態で，コールツリーの葉となる外部関数から近似的にコールツリーの類似度を算出する等の工夫が必要となる．加えてベーシック･ブロックと同様，関数呼び出しの際

に動的に呼び出し先が決まるプログラム (C++ や Delphi を用いた際に散見される) の場合はコールツリーを構築することは困難になってしまう．

3　提案手法

前述のようにプログラムコードに基づく分類技術に関して様々な研究がなされており，ベーシックブロックやコールツリー等の各々の着眼点に関する類似度を算出することができる．しかしその一方で，計算量の観点から機械語命令単位での比較は行われてこなかった．ここでは 2 つのマルウェアが与えられた際に，機械語命令を単位として共通する機械語命令列およびその数を高速に算出し，類似性を求める手法を提案する．これにより機械語命令の追加や削除に応じた類似度を算出できるようになるとともに，実際に変更のあった箇所も機械語命令単位で提示することが可能になる．また，この際に必要となるのはマルウェアのオリジナルコードの逆アセンブル結果のみであり，コールツリーやベーシックブロックの分割，IAT の再構築などは必要としない．

3.1　LCS(最長共通部分列)

提案手法では，機械語命令を 1 要素とし 2 つのマルウェアから得られた各機械語命令系列の LCS およびその長さを算出することを目指す．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 $dbca$ と $bdca$ の LCS は dca と bca となる．LCS 長の導出方法としては，2 つの系列の長さを 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 [8] 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 の性質として次式が成り立つ．

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

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．この作業は行と列にそれぞれに S, T を割り当てた LCS の行列（以下，DP 行列と呼ぶ）を算出することに他ならない．例えば $S =$ "$dbca$", $T =$ "$bdca$" としたときの DP 行列は図 1 になる．

一方，これまでの調査から 100,000 を超える命令数のマルウェアも多数確認されており，こ

	b	d	c	a
d	0	1	1	1
b	1	1	1	1
c	1	1	2	2
a	1	1	2	3

図 1: DP 行列

うしたマルウェア同士の比較には相当な時間を要する．またマルウェア数も日々増加し，それらの組み合わせの数も膨大になっているため，高速に機械語命令系列から LCS およびその長さを算出する手法が望まれる．

3.2　ビットベクトル化

Crochemore ら [9] は LCS を算出する際に，DP 行列の各マスを 1 ビットとして扱い，and, or, not および add の 4 種類の演算で演算ワード長分のマス目をまとめて処理する手法を提案した．ここでもまた $S =$ "$dbca$", $T =$ "$bdca$" の 2 系列を例に説明する．まず T に出現するアルファベットが S のどの位置に出現するかを現すビット行列 M（図 2）を作成する．

	a	b	c	d
d	0	0	0	1
b	0	1	0	0
c	0	0	1	0
a	1	0	0	0

図 2: ビット行列 M

これに基づき下記演算を繰り返すことで，加算（下線部）で発生したキャリーの総和として LCS 長を算出することができる．

$$V_j = \begin{cases} j = 0 \rightarrow 1111 \\ 1 \leq j \leq n \rightarrow \underline{(V_{j-1} + (V_{j-1} \& M_{y_j}))} | (V_{j-1} \& M'_{y_j}) \end{cases}$$

本例では DP 行列における 4 マス分がまとめて処理されることになるが，IA-32 アーキテクチャにおける演算用レジスタのビット長は 32 ビットであるため，32 マス分ずつ処理できることになる．さらに SSE2 命令を用いることで andnot/or に関しては 128 ビット単位，add に関しては 64 ビット単位（このうち 1 ビットはキャリー用とする）で処理可能となる．

ただしこの際に事前に必要となる M は機械語命令の種類数分のメモリスペースを要する．IA-32 アーキテクチャにおける機械語命令は最長で 16 バイトにも及ぶため，そのまま機械語命令を一要素として扱うとメモリ領域が枯渇する恐れがある．

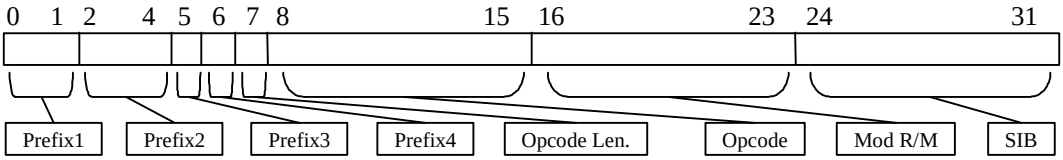

図 3: 縮約命令

3.3 縮約命令

ここでは機械語命令としての情報量を残しつつ，アルファベットサイズを削減することを目的として新たに縮約命令を提案する．IA-32 機械語命令は命令プレフィックス，オペコード，ModR/M，SIB，ディスプレースメント，即値の 6 つの部位から構成される [10]．このうちディスプレースメントには分岐命令の分岐先情報が含まれ，分岐元と分岐先の間に新たな命令が追加されることで変化してしまう．また，メモリアクセスに必要となる絶対アドレスもディスプレースメントとして指定される．一方マルウェアが動的リンクライブラリとして実装されている場合，ロードされるアドレスは一定でない．つまり，環境によってこの絶対アドレスは変化する可能性がある．したがって，ディスプレースメントも類似性算出の情報としてしまうと，上記のような状況において必要以上に類似性が失われてしまう．そのため，縮約命令ではディスプレースメントの情報を含めないこととした．また即値に関しても同様にアドレス情報が含まれる場合があるため縮約命令には盛り込まない．以上を踏まえ，縮約命令は図 3 のエンコーディング規則に従い，1 つの機械語命令を 32 ビット値に変換する．Prefix1〜Prefix4 は命令プレフィックスを表しており，命令プレフィックスの各グループにおいて指定された値を格納する．また Opcode Len はオペコードの長さを表しており，オペコードが 1 バイトの場合は 0，それ以外の場合は 1 を格納する．Opcode は実際のオペコードを格納する変数であるが，オペコードが 2 バイト以上のときは一番最後のオペコードのバイト値を格納する．ModR/M および SIB は機械語命令に含まれる場合はその値を，含まれない場合は 0 を格納する．こうして定義された縮約命令の種類数に関して事前に調査したところ，1 マルウェア当たりの機械語命令は多いもので約 50,000 種類あったのに対し，縮約命令の種類数は約 8,000 種類に抑えることができた．これはビットベクトル化の際に必要なメモリ量に換算すると約 100MB [1] であり，複数の CPU コアで並列処理したとしても，現在の計算機性能で十分に処理可能なアルゴリズムとなる．

3.4 類似度算出

本提案手法では，2 つのマルウェアが与えられた際にまず各機械語命令列を縮約命令列 A, B に変換し，それらの LCS 長（$|L|$ とする）を求める．こうして得られた $|L|$ を元にマルウェア間の類似度 S および非類似度（距離）D を以下のように定義する．

$$S = \frac{|L|}{|A| + |B| - |L|}$$
$$D = 1 - S$$

S は各縮約命令列 A,B を集合，LCS をそれらの積集合とみなしたときの Jaccard 係数に相当し，0 から 1 の値をとる．0 は 2 つのマルウェアに共通部分がないことを意味し，1 は 2 つのマルウェアが全く同一であることを意味する．

4 自動マルウェア分類システム

我々はこれまでにマルウェアの分類を全自動化するために，汎用アンパッキング手法 [11] および確率的逆アセンブル手法 [12] の研究開発を行ってきた．本章では，これらの成果に 3 章の提案手法を加えた自動マルウェア分類システムについて述べる．本システムはアンパッキングモジュール，逆アセンブルモジュール，類似度算出モジュールより構成される．まず本システムに対してマルウェアが与えられると，アンパッキングモジュールにおいて，パッキングされているマルウェアからオリジナルのプログラムコードを含むメモリダンプが出力される．アンパッキングモジュールにより出力されたメモリダンプは逆アセンブルモジュールにおいて逆アセンブルされ，機械語命令列が得られる．この際，1 つのマルウェアが複数のコード領域を持つと複数の機械語命令列が得られるが，本稿では特

[1] 比較対象の縮約命令数を 100,000 とした場合．

に明示しない限り最長の機械語命令列を当該マルウェアの機械語命令列とする．類似度算出モジュールは，各マルウェアの機械語命令列を元に類似度を算出し，マルウェアの全組み合わせ関する類似度行列を作成する．これにより入力されたマルウェアを系統別に分類することが可能となる．

5 実験

本章では，我々が開発した自動マルウェア分類システムによる分類結果およびその考察について述べる．分類対象としては 2 つのデータセットを用いた．1 つ目は 2009 年 7 月に京都大学において収集した 702 種類の SHA1 ハッシュ値が重複しないマルウェア検体である．収集には我々が開発したハニーポットを利用している．このハニーポットは Windows XP SP0 をベースとしたハイインタラクション型の受動的ハニーポットであり，内部では一種のサンドボックスによってマルウェア感染を防いでいる．このため，ダウンローダー等を介したマルウェアは収集対象となっていない．2 つ目のデータセットは CCC DATAset 2009[13] のマルウェア検体 10 種である．各マルウェア間の非類似度が算出された後は，メディアン法を利用した階層的クラスター分析を実施した．

5.1 京都大学におけるマルウェアの分類結果

本システムを利用し京都大学で収集されたマルウェア 702 検体を分類した．ここで類似度 S が 0.8 以上のマルウェアをひとつのクラスタとしてまとめた場合，クラスタ数はわずか 7 つであることが確認された．この中で最も大きいクラスタに含まれるマルウェアは 679 種類にものぼり，リバースエンジニアリングの結果，これは Conficker.B および Conficker.B++であった．また類似度 S を 0.91 を閾値とすると，当該クラスタは Conficker.B（446 検体）と Conficker.B++（233 検体）の 2 つのクラスタに正確に分割されることも確認できた．さらに Conficker.B と Conficker.B++の LCS を求め当該検体同士の差分を解析したところ，Conficker.B++には新たに CreateNamedPipeA 等の名前付きパイプに関連する処理等が追加されていることを確認できた．SRI International[14] は Conficker に関して，B++には遠隔から実行ファイルをアップデートするために，新たにバックドアが追加されたと報告しているが，これはまさしく名前付きパイプを利用したものである．

5.2 CCC DATAset 2009 の分類結果

CCC DATAset 2009 のマルウェア検体の分類結果をデンドログラムとして図 4 に示す．各葉はマルウェア検体におけるオリジナルコードの縮約命令列を表し，縦軸は非類似度となっており，下方で繋がっていれば類似した検体（クラスタ）同士であることを意味している．縮約命令列の名前は”HASH_ハッシュ値の先頭 4 文字”とし，複数のオリジナルコードが存在したものには名前の末尾に識別番号（00 など）を付けている．当該データセットにおける 393F,84E9,1D23（グループＡと呼ぶ）は何らかの関連性を持つとされており，7190,CD91（グループＢと呼ぶ）もまた何らかの関連性を持つとされている．図 4 では，実際にグループＡの 393F および 84E9 に関して類似度 $S = 0.47$ 程度の一致がみられた．またグループＢの 7190 および CD91 に関しても類似度 $S = 0.29$ 程度の一致がみられ，その一致箇所には TCP 139/445 番を用いた通信ロジックが含まれていた．一方で，7190 の検体には IRC を利用したボットと思われるロジックが確認されたが，CD91 の検体には書式指定文字列 *’[SCAN]: IP: %s:%d, Scan thread: %d, Sub-thread: %d.’* を元に文字列を作成する部分以外に IRC 関連の処理はほとんど存在していない．その代わりに autorun.inf 作成等の処理が追加されていた．また，上述の書式指定文字列を元に作成された文字列は，プログラム上使用されることがないため 7190 の検体の残骸とも解釈できる．

6 まとめ

本稿では，プログラムコードの類似性を機械語命令単位で高速に算出する手法を提案するとともに，これまで我々が研究開発してきたアンパッキング手法および逆アセンブル手法を組み合わせた自動マルウェア分類システムについて述べた．また本システムを利用し京都大学で収集されたマルウェア 702 検体を分類したところ，類似度 S が 0.8 以上のマルウェアをひとつのクラスタとしてまとめた場合，クラスタ数はわずか 7 つであることが確認された．加えて LCS に基づく機械語命令単位の解析により，効率的に

図 4: CCC DATAset 2009 の分類結果

亜種間の共通機能・差分機能を把握できることも分かった.

今後は，コンパイラの変更やその最適化がプログラムコードに対して及ぼす影響を踏まえつつ，マルウェアの分類技術を洗練化していく.

謝辞

本研究を進めるにあたり，ハニーポット設置環境及びマルウェア検体をご提供いただいた京都大学学術情報メディアセンター高倉弘喜准教授に深く感謝いたします.

参考文献

[1] Halvar Flake, マルウェアの分類とアンパッキングの自動化, Black Hat Japan 2007.

[2] サイバークリーンセンター, https://www.ccc.go.jp/report/h20ccc_report.pdf.

[3] Computer Security Research - McAfee Avert Labs Blog, http://www.avertlabs.com/ research/ blog/ index.php/ 2009/ 07/ 22/.

[4] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario. Automated Classification and Analysis of Internet Malware. In RAID, 2007.

[5] 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.

[6] Marius Gheorghescu. An automated virus classification system. In Virus Bulletin Conference, October 2005.

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

[8] Robert A. Wagner, Michael J. Fischer, The String-to-String Correction Problem, Journal of the ACM (JACM), v.21 n.1, p.168-173, Jan. 1974

[9] Maxime Crochemore, Costas S. Iliopoulos and Yoan J. Pinzon, Speeding-up Hirschberg and Hunt-Szymanski LCS algorithms, String Processing and Information Retrieval, 2001, pp. 59–67.

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

[11] 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

[12] 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.

[13] 畑田充弘, 他, マルウェア対策のための研究用データセットとワークショップを通じた研究成果の共有, MWS2009（2009 年 10 月）

[14] SRI International, An Analysis of Conficker, http://mtc.sri.com/Conficker/.