静的解析によるマルウェアのAPI推移の抽出とクラスタ解析

岩本 一樹 †　　和崎 克己 ‡

† 日本コンピュータセキュリティリサーチ株式会社
422-8052 静岡県静岡市駿河区緑が丘町 1-1
iwm@maid.org

‡ 信州大学大学院工学系研究科
380-8553 長野県長野市若里 4-17-1
wasaki@cs.shinshu-u.ac.jp

あらまし 大量のマルウェアが作られている今日ではマルウェアの分類はマルウェア対策において重要である．未知のマルウェアの分類ができれば，どの既知のマルウェアと類似しているのかわかり，またマルウェアの機能を推定することができる．しかしマルウェアのバイナリコードを単純に比較しただけでは分類は不可能であり，マルウェアの構造に基づく特徴を抽出する必要がある．本研究では大量のマルウェアを効率よく自動的に分類する手法を提案する．

具体的にはマルウェアに対して制御フロー解析を行うことでAPIの推移を抽出し，マルウェアの特徴とした．またそれらのマルウェアの特徴に基づいてクラスタ解析を用いて検体を分類を行った．

API Sequence Extraction and Cluster Analysis for Malware Codes based on the Static Analysis

Kazuki Iwamoto†　　Katsumi Wasaki‡

†Japan Computer Security Research Center
1-1 Midorigaoka-cho, Suruga-ku, Shizuoka-city, Shizuoka 422-8052 Japan
iwm@maid.org

‡Graduate School of Science and Technology, Shinshu University
4-17-1 Wakasato, Nagano-city, Nagano 380-8553 Japan
wasaki@cs.shinshu-u.ac.jp

Abstract A classification for malware is important in anti-malware, because lots of malwares are released nowadays. If we can classify unanalysed malwares, we can know the nearest known malware and estimate functions. But malwares aren't classified by comparing their binaries only. So we have to extract features based on the structure of malware. In this paper, we propose the method to classify lots of malwares automatically.

Concretely, we extracted API sequences by using control flow analysis from malware samples and defined that API sequence is the feature of malware. Also we classified malwares based on the feature of malware by cluster analysis.

1 はじめに

今日，マルウェアは利益目的で作成され大量の亜種が存在する．それらのマルウェアは既存マルウェアを少し改変しただけのものが多い．少し改変しただけとはいえ，大量にあるそれらのマルウェアを何の前提知識もなく解析するのは困難である．

あるマルウェアを解析するとき，同じ種類のマルウェアを解析した経験のある技術者が解析

を行うことが最も効率が良い．しかたってマルウェアの分類ができば解析を行う適任な技術者を決定できる．またマルウェアが既知のどのマルウェアに似ているかを知ることができれば，解析対象となるマルウェアの動作や構造を推定でき，マルウェアの解析を行う上で手がかりとなる．さらにマルウェアの動作の推定は，ある程度のマルウェア対策として役立つ．

マルウェアを解析する方法には動的解析と静的解析がある．動的解析ではプログラムを実行したときにそのプログラムが呼び出した API やシステムコールなどから検出を行う．この方法の問題点は，検査対象のプログラムがマルウェアとして動作するまで検出ができないことである．

一方，マルウェアのコードを逆アセンブルするなどの方法で解析する静的解析がある．静的解析では検査対象がマルウェアとして動作する条件には依存しない．この方法ではマルウェアの特徴をどのように捉えるのかが問題となる．

本研究ではマルウェアを静的解析することで特徴を抽出し，クラスタ解析を用いて分類を行う方法を提案する．この方法の利点はマルウェア作成者側によるわずかな改変では変わることのない特徴を抽出することと，分類の結果をわかりやすく示すことが可能なことである．

提案する方法では，対象となる検体を逆アセンブルし，それに対して制御フロー解析を行った．制御フロー解析の結果のグラフから，ある API が呼び出されたときに，次に呼び出される可能性のある API を求めてグラフ化した．ある API とその後に呼び出される API の組が現れる頻度（API 推移）を，本研究では検体の特徴とした．API 推移はプログラムの構造を変えない限り再コンパイルしても変化が少ない特徴と言える．本研究では抽出した特徴に対してクラスタ解析を行った．クラスタ解析は教師なし学習の一種なので，これにより抽出した特徴に基づく検体間にある関係を明らかにできた．

結果，本研究で提案する方法によりマルウェアの分類は可能であった．しかしその結果には既存の分類とは一致しない検体もあった．

2 関連研究

マルウェアの解析方法は主に動的解析と静的解析に分かれる．動的解析ではプログラムを実行したときにそのプログラムが呼び出した API やシステムコールなどから検出を行う．星澤裕二らの研究 [1] ではマルウェアを実行して API の呼び出しとそれに伴う引数や戻り値のログからマルウェアの分類を試みている．Michael Bailey らの研究 [2] ではマルウェアの動作から正規化圧縮距離を求め階層型クラスタ分析を行うことでマルウェアの分類を行っている．Konrad Rieck らの研究 [3] ではマルウェアの動作ログの文字列を特徴として抽出し，サポートベクターマシンを用いて分類を行っている．藤原将志らの研究 [4] ではマルウェアのネットワーク接続をマルウェアの特徴として捉えようと試みている．堀合啓一らの研究 [5] ではマルウェアを実行したときに生じるファイルやレジストリの変化やプロセスの起動・停止，ネットワークアクセスなどを特徴として捉え，それらを固定長の配列に変換し，ハミング距離を求めることで分類を行っている．

一方，プログラムを実行するのではなく，プログラムに対して逆アセンブルなどを行いプログラムの構造からマルウェアの解析を試みる静的解析がある．Halvar Flake はマルウェアのコールグラフや制御フロー解析の結果のグラフを比較することでマルウェアの分類を行う提案を行っている [6]．安本幸希らの研究 [7] では，あるマルウェアのコードからランダムに一定の長さのバイト列を抽出し，同一のバイト列がもう一方のマルウェアに含まれている確率を 2 つのマルウェアの類似度と定義している．勝手壮馬らの研究 [8] では 2 つの静的解析による特徴抽出の結果を用いて分類を行っている．

3 本研究の提案

3.1 特徴抽出

マルウェアをクラスタ解析するためには，まずマルウェアの特徴を抽出する必要がある．コンパイラのコード生成オプションを変更しただけ

```
#include <windows.h>

int WINAPI
WinMain (HINSTANCE hInstance,
         HINSTANCE hPrevInstance,
         LPSTR     lpszCmdLine,
         int       nCmdShow)
{
  int i, s = 0;

  for (i = 1; i < 10; i++)
    s += i;
  return 0;
}
```

表 1: コンパイラのテスト用ソースコード

でも生成されるコードは大きく異なるので，マルウェアのコードをそのまま特徴とすることはできない．例えば表 1 のソースコードをコンパイラのオプションを変えてコンパイルした場合，

最適化なし 55 8B EC 83 EC 08 C7 45 FC 00 00 00 00 C7 45 F8 01 00 00 00 EB 09 8B 45 F8 83 C0 01 89 45 F8 83 7D F8 0A 7D 0B 8B 4D FC 03 4D F8 89 4D FC EB E6 33 C0 8B E5 5D C2 10 00

サイズで最適化 6A 01 33 C9 58 03 C8 40 83 F8 0A 7C F8 33 C0 C2 10 00

速度で最適化 33 C9 B8 01 00 00 00 03 C8 40 83 F8 0A 7C F8 33 C0 C2 10 00

となる．生成されたコードを見るとコンパイラのオプションを変えただけで，まったく別のコードになってしまう．ゆえにコードの変化に影響されない特徴抽出を行う必要がある．

3.1.1 制御フロー解析

制御フロー解析を用いてプログラムを解析して特徴を抽出する．制御フロー解析では関数の内部の分岐やループなどを解析するので，関数内部でのコードの順番が異なっていても同一の結果が得られる．表 2 のプログラムを制御フロー解析した場合には図 1 になる．表 2 は C 言語だが，実際にはマルウェアの逆アセンブルリストを元に制御フロー解析を行う．

3.1.2 API 推移の抽出

制御フロー解析の結果はあまりにも大きいグラフになるので扱いづらい．制御フロー解析の

```
int main (int argc, char *argv[])
{
  int i;

  if (argc == 0) {
    ApiA ();
    if (!argv[]) {
      ApiB ();
      return -1;
    }
  } else {
    if (!ApiA ())
      goto loop;
  }
  for (i = 1; i < argc; i++)
    if (!ApiC ())
      break;
  loop:
  ApiD ();
  return 0;
}
```

表 2: 制御フロー解析のソースコード

結果のグラフからはある API が呼び出された後に呼び出される可能性がある API が何であるかを読み取ることができる．

具体的には制御フロー解析の結果を利用して API 呼び出しを行わないノードを API 呼び出しを行うノードに統合する．例えば図 1 の場合にはノード a，c，e は消滅する．ノード b からはノード d（b → c → d）とノード f（b → c → e → f）とノード h（b → c → e → h）へ，ノード f からはノード f 自身（f → e → f）とノード h（f → h）とノード h（f → e → h）へ，ノード g からはノード f（g → e → f）とノード h（g → e → h）とノード i（g → i）へエッジが作られる．さらにノード b とノード g，ノード h とノード i は同一の API を呼び出しているので統合される．結果，図 2 のようになる．エッジの数字は API 推移の頻度を示す．

3.2 分類

API の数は有限なので API の数を n とすると，検体から抽出した API 推移は長さが $l = n^2$ の固定長の配列となり配列の各要素 $s = (a_1, a_2, \ldots, a_l)$ は 0 以上の整数になる．この配列を各検体の特徴として分類を行う．分類を行う上で，まず各検体間の距離（2 つの検体

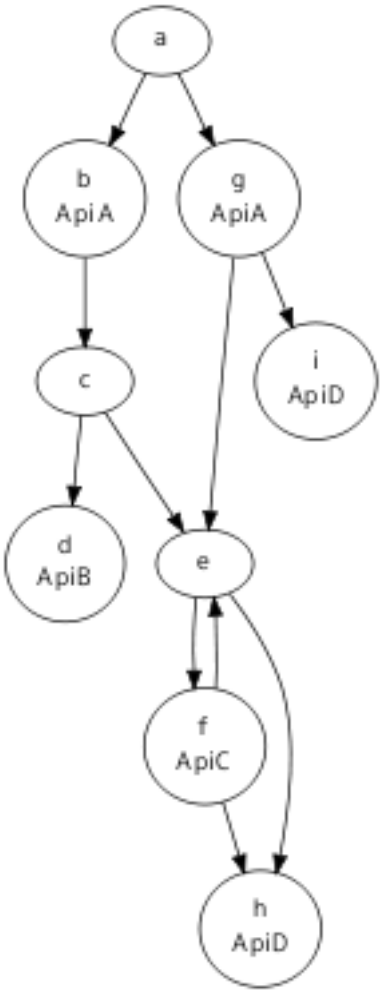

図 1: 制御フロー解析の結果

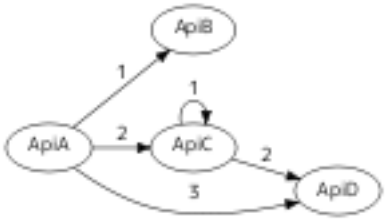

図 2: API 推移の解析の結果

がどの程度似ていないのか）を決める方法を定義する．その後，検体間の距離を基づいてクラスタ解析を行う．

3.2.1 API 推移における検体間の距離の定義

共通するエッジの割合を求めることで API 推移の距離を求めることを考える．共通するエッジが多いほど検体は似ていると言える．そこで

図 3: API 推移の比較例

図 4: 共通する API 推移

比較する 2 つの検体のすべての API 推移のうち，一致しない API 推移の比率を求めることで 2 つの検体の距離（不一致度）求める．ある 2 つの検体 s_i と s_j の距離 d_{ij} を

$$DIS(s_i, s_j) = d_{ij} = \frac{\sum_{k=1}^{l} |(a_i)_k - (a_j)_k|}{\sum_{k=1}^{l} (a_i)_k + \sum_{k=1}^{l} (a_j)_k}$$

と定義する．

例えば図 2 と図 3 に共通するエッジは図 4 になる．この場合には図 2 の API 推移の数の合計が 9，図 3 の API 推移の数の合計が 7，共通する API 推移が 4 なので一致しない API 推移の数の合計は 12，よって 2 つの検体の距離は $12 \div (9 + 7)$ で 0.75 となる．

96 種類のマルウェアの API 推移を求めたところ 9 種類は他のマルウェアの API 推移と同じであった．そのため 87 種類（科名では 17 種類）の各検体間の距離を求めてた．表 3 は 87 種類の各検体間の距離の表の一部である．表 3 では値を 10000 倍しているので，2 つの API 推移が完全に一致するときには距離は 0 となり，まったく一致しないときには 10000 になる．

3.2.2 階層型クラスタ分析による分類

階層型クラスタ分析では検体数を N とすると，

1. 検体集合 $S = \{s_1, s_2, \ldots, s_N\}$ の中で最も距離が近い 2 つの検体 s_i，s_j を探す

	BagleA	KorgoA	KorgoB	MydoomA	MydoomB	MytobB
Autorun180	9644	9774	9772	9757	9659	9790
BagleA		9477	9471	9056	9191	9116
KorgoA			1098	9052	9123	8932
KorgoB				9136	9156	8958
MydoomA					3927	8047
MydoomB						8185

表 3: 各検体間の距離

2. s_i，s_j の平均となる c を S に加え，s_i と s_j を S から取り除く

を S の要素が目的のクラスタ数になるまで繰り返す．図 5 は階層型クラスタ分析を行った結果である．しかしながら，どのクラスタ数で分類したときが最も特徴をあらわしているのかという問題がある．最適なクラスタ数を求めるため，ベイズ情報量規準

$$BIC = -2\ln(L) + K\ln(N)$$

（K はクラスタ数,N は検体数）を用いることができる．BIC が最小となる K が最適なクラスタ数であるといえる．対数尤度は

$$\ln(L) = \sum_{i=1}^{K}(-\frac{n_i}{2}\ln(2\pi) - \frac{n_i l}{2}\ln(\hat{\sigma}_i^2) - \frac{n_i - K}{2} + n_i\ln(n_i) + n_i\ln(N))$$

（n_i はクラスタの検体数）とする．分散 $\hat{\sigma}_i^2$ は

$$\hat{\sigma}_i^2 = \frac{1}{n}\sum_{j=0}^{n}(DIS(g_i, s_j))^2$$

である（g_i はクラスタの平均）．図 6 はベイズ情報量規準が最小となるクラスタ数 36 で階層型クラスタ分析を行った結果である．

4 考察

本研究で提案した方法でコードの特徴からマルウェアを分類することはできた．マルウェアのコードを静的に解析して API の推移を特徴として抽出し，それに基づいて分類を行うという方法の 1 つを提案できたと言える．これから解析を行おうとする検体が，どの既知のマルウェアに近いのかを知ることができれば，マルウェアの解析を行う上で大変有益な情報となる．またコードの類似性からマルウェアの機能を推定することも可能である．

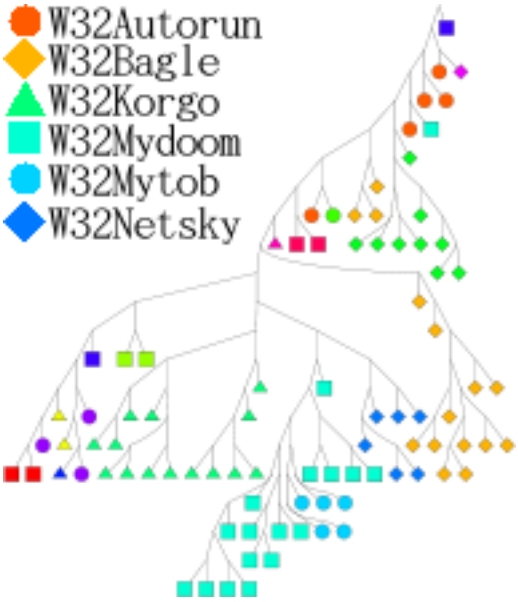

図 5: 階層型クラスタ分析による分類

しかし必ずしも現状のマルウェアの分類と同じ分類ができた訳ではなかった．現状の分類ではコードの関連性による分類だけではなく，Autorun のようにある機能があるか否かでその科名が構成されている場合がある．このような分類基準に基づく分類を行う場合には，コードの特徴を前提となる知識なしで分類する本研究で提案した方法では対応できない．

階層型クラスタ分析による分類の結果からは Koobface と Korgo では検体が 1 つの枝の下にまとまっているのがわかる．Bagle は Bagle.C, Bagle.D，Bagle.X が別の枝を作っているが，これらは他の Bagle に対して距離が大きくなっている．同様に Mydoom#67 も他の Mydoom とは離れた位置にある．興味深いのは Mydoom と

図 6: 階層型クラスタ分析によるクラスタ数 36 の分類

Mytob の関係で，Mydoom の枝の中に Mytob が含まれる形になっている．Mydoom はオリジナルのソースコードが公開されており，Mytob は Mydoom から派生したと考えられている．ゆえにこのような結果になったと推測できる．

Autorun は木の根元の方に配置されている．これは距離が近い検体が見つからなかったため，木を作る最後の段階で相次いで木に追加されたことが原因である．木の根元付近に集まっているからといって，これらの検体が似ているという訳ではない．

5 今後の課題

本研究の結果は既存の分類と一致しない部分もあった．それは既存の分類が必ずしもコードの特徴から分類が行われているとは限らないからである．もし，既存の分類に沿うようにマルウェアを分類したいならば，本研究のようなクラスタ解析（教師なし学習）は不適切である．ゆえに本研究で抽出した特徴を，Konrad Rieck らの研究 [3] のように教師あり学習によって分類することも検討できると思われる．

実際のシステムの運営では，新たに発見されたマルウェアを逐次分類することになる．本研究で作成したシステムは，現存する検体すべてに対して分類を行っているので，この部分を改良する必要がある．

参考文献

[1] 星澤裕二，太刀川剛，山村元昭，マルウェアの亜種等の分類の自動化, 電子情報通信学会技術研究報告 ISEC2007-53, pp.63-70, July 2007

[2] Michael Bailey, Jon Oberheide, Z. Morley Mao, Farnam Jahanian, Jose Nazario, `Automated Classification and Analysis of Internet Malware`, Recent Advances in Intrusion Detection 2007 10th International Symposium (RAID '07), pp.178-197 (2007)

[3] Konrad Rieck, Thorsten Holz, Carsten Willems, Patrick Düssel, Pavel Laskov, `Learning and Classification of Malware Behavior`, DIMVA2008, Detection of Intrusions and Malware and Vulnerability Assessment, Vol.5137/2008, pp.108-125 (2008)

[4] 藤原将志, 寺田真敏, 安部哲哉, 菊池浩明, マルウェアの感染動作に基づく分類に関する検討, 情報処理学会研究報告. CSEC, [コンピュータセキュリティ] IPSJ SIG Notes 2008(21) pp.177-182 20080306

[5] 堀合啓一, 今泉隆文, 田中英彦, マルウェア亜種の動的挙動を利用した自動分類手法の提案と実装, 情報処理学会論文誌，Vol.50, No.4

[6] Halvar Flake, `Automated Unpacking and Malware Classification`, Black Hat Japan 2007 pp.61-88, 25th October 2007

[7] 安本幸希, 森井昌克, 中尾康二, マルウェアコードの類似度判定による機能推定, 電子情報通信学会技術研究報告 ISEC2007-93, pp.31-36, November 2007

[8] 勝手壮馬, 安本幸希, 伊沢亮一, 森井昌克, 中尾康二, マルウェアの分類方法とその応用に関する考察, 電子情報通信学会技術研究報告 ICSS2009-27, pp.103-108, June 2009