情報処理学会論文誌 Vol.54 No.3 1199–1210 (Mar. 2013)

静的解析により抽出された API推移に基づくマルウェアの分類

岩本 一樹[1,†1,a)] 和崎 克己[2,b)]

受付日 2012年6月17日, 採録日 2012年12月17日

概要：本論文では，対象とする多数の検体を静的解析することで特徴を抽出し，ソースコードの構成に基づいた，精度の高いマルウェアの自動分類法を提案する．特徴抽出に関する提案手法は，検体の実行コードに対して，API 推移依存グラフの，ある API とその後に呼び出される API の組の有無を定義し，マルウェアの検体の特徴量とする．検体間の類似度の定義として Dice 係数を適用した．特徴が似ている検体群の可視化のため，抽出した特徴量に基づいた階層型クラスタ分析を行う．分析結果は科名ごとに着色された樹形図で提示する．提案手法を評価するため，逆アセンブラ，制御フロー解析器，API 推移特徴抽出器，Dice 係数生成器，階層型クラスタ分析処理プログラムを制作し，自動マルウェア静的解析システムを構築した．実験として，4,684 種類のマルウェアの検体を用意し，API 推移抽出に成功した 1,821 種類の検体に対して，類似度比較による自動分類を実行した．その結果，短い時間で階層型クラスタ分析まで自動処理を実施し，亜種グループを形成する多数の有意なクラスタを得た．

キーワード：マルウェア，静的解析，制御フロー解析，API 推移，特徴抽出

Malware Classification Based on Extracted API Sequence by Static Analysis

Kazuki Iwamoto[1,†1,a)] Katsumi Wasaki[2,b)]

Received: June 17, 2012, Accepted: December 17, 2012

Abstract: In this paper, we propose highly accurate automatic malware classification method, by extracting features by using static analysis of malware samples, with the structure of malware source code. In the proposal extracting method, existence and non-existence of a particular pairs of API and its subsequent API in API sequence graph is compared with the executable code of a sample, with which feature of malware sample is defined. To determine the degree of similarity between samples, Dice's coefficient has been applied. To visualize the grouping of similarly-featured samples, we have used hierarchical cluster analysis based on the extracted features. The analysis results are presented in dendrogram with colored nodes to each family name. In order to assess the proposed method, we have set up the automatic malware static analysis system with combination of disassembler, control flow analyzer, API sequence extractor, similarity calculator and hierarchical cluster analyzer. We have acquired 4,684 malware samples, and 1,821 of those samples successfully extracted from API sequence have been put to our proposal classification method. As a result, automatic processing has been executed to hierarchical cluster analysis in a short time, and significant clusters of variant groups have been obtained.

Keywords: malware, static analysis, contorl flow analysis, API sequence, feature extraction

1. はじめに

マルウェアとは，計算機上において不正で有害な動作を行う「悪意のある」ソフトウェアやコードの総称である．マルウェアは，メタモーフィック型であること [1]，マル

1 日本コンピュータセキュリティリサーチ株式会社
Japan Computer Security Research Center, Shizuoka 422–8052, Japan
2 信州大学大学院総合工学系研究科
Interdisciplinary Graduate School of Science and Technology, Shinshu University, Matsumoto, Nagano 390–8621, Japan
†1 現在，独立行政法人情報処理推進機構
Presently with Information-technology Promotion Agency, Japan
a) iwm@maid.org
b) wasaki@cs.shinshu-u.ac.jp

© 2013 Information Processing Society of Japan

ウェアを作成するツールの存在 [2] などにより，異なるバイナリイメージのマルウェアが，大量に作られることが大きな問題となっている．新規マルウェアの発生数には時期により増減があるものの，どの時期においても，多数の亜種マルウェアが発生している状況にある [3]．

マルウェアは計算機が提供するサービスやセキュリティ上での脅威であることが多く，感染防止や除去対策援用プログラムの開発上，マルウェアの解析のコスト低減が重要な課題である．ここで，未知のマルウェアの検体が，すでに解析されているどのマルウェアに似ているかを知ることができれば，検体の動作や構造を推定できるため，マルウェアの解析を行ううえで手がかりとなる．

マルウェアを手動あるいは自動分類するうえで障害となっている主な原因は，マルウェアの元になるソースコードが共通の基盤として広く流通し，配布バイナリイメージは異なっているが，似た機能・動作を有するマルウェアが存在することである．具体的には，ソースコードの改変やコンパイル環境の違いなどが原因で実行コードが異なった亜種が制作されている．さらに実行コードは，パッカと呼ばれる圧縮・暗号化プログラムを用いて難読化（パック）した後に配布されることがあり，同じマルウェアであっても異なるバイナリイメージが作られる．

上記の背景に対して，本研究ではマルウェアの解析のコスト低減を目的とし，対象とする多数の検体を静的解析することで特徴を抽出し，元となったソースコードの構成に基づいた，精度の高いマルウェアの自動分類法を提案する．まず，本研究で取り扱う検体の前提として，パック済みマルウェアの検体については，我々の従来研究 [4] によって開発された自動アンパッカを用いて，アンパックが成功している実行コードが得られていることとする．

本論文で提案する手法では，アンパックされた検体に対して逆アセンブルを行った後，制御フロー解析を行いすべての関数のグラフを取得する．制御フロー解析の結果は巨大なグラフとなるため，現実的な時間で比較することはできない．そこで提案する手法では，API（Application Program Interface）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 を探査することにより，巨大なグラフを API 推移依存グラフに収縮させる．次に，マルウェアの検体から抽出した API 推移（特徴量）を比較することで，検体間の類似度を求める．類似度の算出ではグラフの比較は行わず，Dice 係数を適用することで実行時間の短縮を図る．最後に，特徴が似ている検体群の可視化のため，抽出した特徴量に基づいた階層型クラスタ分析を行う．

提案手法を評価するため，我々は，従来研究のアンパッカに加え，逆アセンブラ，制御フロー解析器，API 推移特徴抽出器，Dice 係数生成器，ならびに階層型クラスタ分析器の処理プログラムを制作し，自動マルウェア静的解析システムを構築した．耐性評価実験として，ソースコードが公開されている，あるマルウェアに対して，コンパイラと最適化オプションを変化させた場合の類似度抽出に関する耐性を調査した．性能評価実験としては，Windows OS を攻撃対象とした 4,684 種類のマルウェアの検体を用意し，本システムにより分類した．結果，逆アセンブルとフロー解析に成功し，API 推移を抽出できた検体は 1,821 種類，科名では 113 種類の，類似度比較可能な検体を短い実行時間内で得ることができた．このデータセットからランダムに 500 種類を取り出し，階層型クラスタ分析を行い，係数 0.8 以上の類似度を有する検体でクラスタを作成したところ，有意な 56 クラスタを得た．しかしながら，検体の科名（名付け）との関係においては，マルウェアの実際の動作に基づく命名がなされているという現状があり，静的解析のみに頼る手法の限界や課題も明らかになった．

本論文は以下で構成される．まず 2 章で，従来の動的解析・静的解析ならびにアンパックを用いたマルウェアの分類手法について比較する．3 章では，我々が提案するバイナリコードの特徴抽出と類似度分類を用いた静的解析手法について述べる．4 章では，自動分類プログラムの実装と評価対象検体，ならびに分類結果について説明する．5 章では，提案手法と実験結果に対する評価と議論を行い，最後の 6 章でまとめと今後の検討課題を述べる．

2. 関連研究

マルウェアの分類を行う研究は，主に動的解析と静的解析に分かれる．多くの研究では，まず初めに何らかの方法で各検体から特徴を抽出している．その特徴の抽出で実際に検体を実行させるケースを動的解析，そうでないならば静的解析といえる．その後，その特徴を比較することで各検体間がどの程度同じか（あるいはどの程度違うのか）を数値化する．

2.1 動的解析

動的解析の場合にはアンパックの必要がなく，パッカが付加したバイナリコードも含めて動的解析を行う．Bailey らの研究 [5] 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．

Christodorescu らは，システムコールのログの引数の値からシステムコールの依存関係を求めて，グラフにして比較する方法 [6] を提案している．

2.2 静的解析

静的解析の研究としては，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 [7]．この方法はコンパイラの設定を変えるなどで，見かけのバイナリコードが大きく異なるような場合に対して耐性がある．

© 2013 Information Processing Society of Japan

Kruegel らが提案するネットワークの通信からマルウェアを見つける方法 [8] では，マルウェアのバイナリコードを静的に制御フロー解析することで特徴をグラフとして表し，その部分グラフの比較を行うことで，マルウェアの検出を試みている．この方法では，グラフのノードに対応するバイナリコードの種類をグラフの属性（色）として与え，それを含めて一致するグラフを見つけることでマルウェアの検出を行っている．

Zhang らはメタモーフィック型のマルウェアを検出する方法 [9] を提案している．この提案では，逆アセンブルして得られた機械語の命令を，その関数呼び出しからブロックに分けて一致する部分を探すことで，マルウェアのパターンと検体の類似度を求めている．

Han らはマルウェア検体から API 呼び出し関係グラフを抽出し，Jaccard 指数を用いて類似度を求めることでマルウェアを検出する方法 [10] を提案している．また Han らはマルウェアが実行されるときに呼び出される API の順序と IAT（Import Address Table）が類似するという仮定に基づき，IAT から API リストを作成し，無害なプログラムから作成した API リスト（White List）の API を除いたうえ，重複した半順序関係を再帰的に除去することで検体間の類似度を求める方法 [11] を提案している．

Altaher らはマルウェアの検体がインポートしている API を抽出し，ECM（Evolving Clustering Method）による分類する方法 [12] を提案している．

また我々はマルウェアの検体を逆アセンブルし制御フロー解析の結果から API 推移を抽出して検体を比較する方法 [13] を提案した．

文献 [7], [10], [11], [12], [13] では何らかの方法で検体はアンパックされ，マルウェアのパックされる前のバイナリイメージが入手できていることを前提としている．

2.3　アンパックと静的解析

アンパックと特徴抽出・分類を組み合わせた提案がある．岩村らの研究 [14] では，動的なアンパックの後，マルウェアの検体を逆アセンブルし，その機械語の命令の並びを Jaccard 係数を用いて比較することでマルウェアの類似度を求めている．

Ye らの研究 [15], [16] ではマルウェアの検体から API を対応する数値（ID）に変換して抽出している．文献 [15] では OOA（Objective-Oriented Association）による分類を行っており，他の分類方法と比較している．文献 [16] では Jaccard 係数を用いて階層型クラスタ分析を行っており，教師ありの学習により実際の名前付けを反映させている．またこの研究では他者の名前付けとの比較も行っている．

3.　提案手法

3.1　特徴抽出

コンパイラのバイナリコード生成オプションを変更した場合，生成されるバイナリコードが変化するため，バイナリコードを見ただけでは同じソースコードから作られたマルウェアであると推測することが困難である．本研究のマルウェアの分類の目的は同じソースコードから作られたマルウェアに対して高い類似度を与えることである．ゆえにバイナリコード生成時の環境変化に影響されない特徴を抽出する必要がある．

本研究ではマルウェアの特徴抽出として，制御フロー解析の結果から API 推移の抽出を行い，マルウェアの検体間の類似度を求めて分類を試みる．既存のソースコードを利用して新しいマルウェアが作成されたときには，新たに改変されたバイナリコードの割合が多いほど，類似度は小さくなる．また 2 つ以上のマルウェアのソースコードを組み合わせて 1 つのマルウェアが作成されたときには，元になったマルウェアのコードが新しいマルウェアに占める割合が多いほど，類似度は大きくなる．

3.1.1　逆アセンブル

多くのマルウェアではソースコードが公開されていない．そのためマルウェアの検体を逆アセンブルすることで静的解析を行う必要がある．本研究では我々が作成した逆アセンブラを使用する．逆アセンブルの手法は主に線形掃引法（Linear sweep method）と再帰走査法（Recursive traversal method）に分かれる [17]．高い精度の結果を得るために実行時間は長くなるが，本研究の逆アセンブラでは再帰走査法を用いる．

3.1.2　制御フロー解析

3.1.1 項で取得した逆アセンブルリストに対して制御フロー解析を行う．エントリーポイントから制御フロー解析を行い，呼び出されている関数があるならば，その内部も再帰的に制御フロー解析を行う．

たとえば**表 1** の逆アセンブルリストに示す命令列の場合，エントリーポイントに相当する 401014 から再帰的に制御フロー解析を行う．制御フロー解析においては，jmp や jnz，loop，call，ret などの分岐命令が必要である．例として，表 1 から分岐命令を取り出すと**表 2** となり，これを制御フロー解析した場合には**図 1** のグラフを得る．

3.1.3　API 推移の抽出

制御フロー解析の結果グラフからは，ある API が呼び出された後に呼び出される可能性のある API を判別可能である．制御フロー解析の結果グラフから API の推移を求めるために，グラフの中の 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．

たとえば図 1 の場合には**図 2** で点線で示したノードとエッジは消滅し，太線で示したエッジが作られる．**図 3** は

© 2013 Information Processing Society of Japan

表 1 逆アセンブルリスト
Table 1 Disassemble list.

```
401000 xor  eax,eax
401002 cmp  [esp+04],eax
401006 jz   401012
401008 push [esp+04]
40100c call [402040] ; lstrlenA
401012 inc  eax
401013 ret
;
401014 call [402038] ; GetVersion
40101a test eax,eax
40101c jns  401022
40101e or   eax,ff
401021 ret
401022 cmp  [esp+04],01
(abbr.)
401075 xor  eax,eax
401077 pop  esi
401078 ret
```

表 2 分岐命令リスト
Table 2 Branch instruction list.

```
401006 jz   401012
40100c call [402040] ; lstrlenA
401013 ret
;
401014 call [402038] ; GetVersion
40101c jns  401022
401021 ret
401029 jg   40103a
40102b call [40203c] ; GetTickCount
401033 jz   401067
401038 jmp  40106b
401041 jle  401067
40104a call 401000
401052 call [40203c] ; GetTickCount
40105e jz   401035
401065 jl   401043
40106e call [4020ac] ; MessageBoxA
401078 ret
```

API 呼び出しを行わないノードを削除した結果である．さらに 40102b と 401052 は同一の API を呼び出しているので統合され，最終的には図 **4** になる．

3.2 分類

マルウェア検体は 3.1.3 項で抽出した API 推移を要素とする集合と見なすことができる．この集合を各検体の特徴とし，各検体間の類似度を決める方法を定義する．本研究では同一の科名（マルウェアの名付け）の検体間の類似度および異なる科名の検体間の類似度を求め，これを比べることで定義した類似度について検証する．その後，検体間

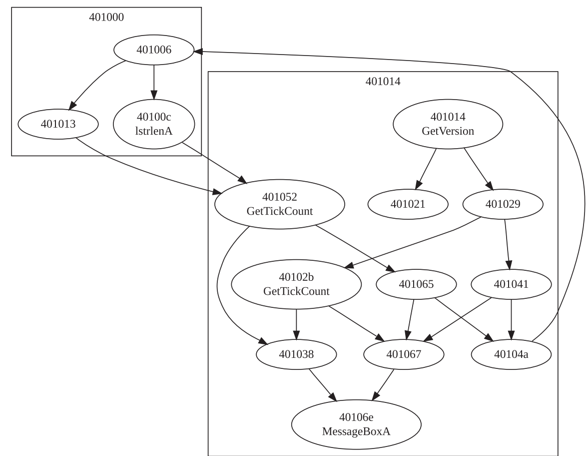

図 1 制御フローグラフ
Fig. 1 Control flow graph.

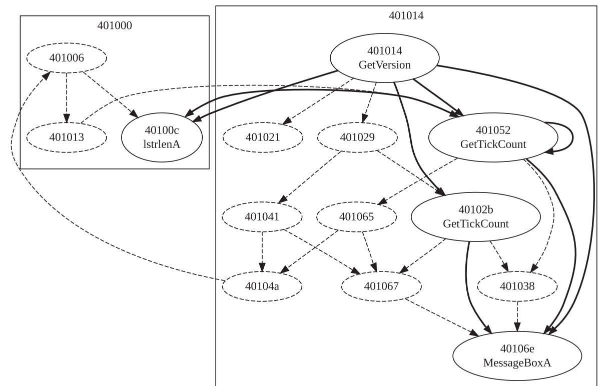

図 2 削除されるノードと追加されるエッジ
Fig. 2 Removed nodes and appended edges.

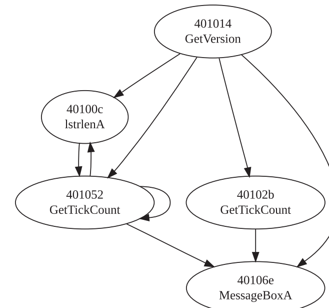

図 3 ノード統合前
Fig. 3 Before merging.

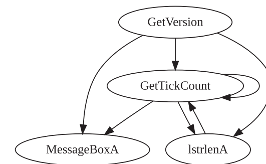

図 4 API 推移
Fig. 4 API sequence.

の類似度に基づいて階層型クラスタ分析を行う．

3.2.1 API 推移における検体間の類似度の定義

検体から抽出した API 推移の集合を X，Y とすると，こ

© 2013 Information Processing Society of Japan

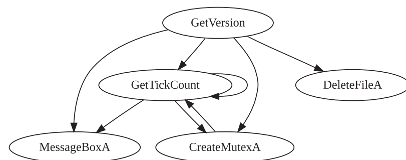

図 5 API 推移の比較例
Fig. 5 Example for API sequence.

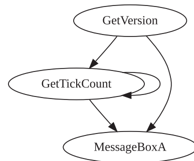

図 6 共通する API 推移
Fig. 6 Common API sequence.

れらの集合の類似度 S は Dice 係数を用いて

$$S = \frac{|X \cap Y| \times 2}{|X| + |Y|}$$

と定義する．$|X|$ は集合 X の要素数である．完全に一致するならば類似度は 1 になり，まったく一致しないならば類似度は 0 になる．

たとえば，図 4 と**図 5** に共通するエッジは**図 6** になる．図 4 の API 推移の数の合計が 7，図 5 の API 推移の数の合計が 8，共通する API 推移が 4 なので，2 つの検体の類似度は 0.53 となる．

3.2.2 階層型クラスタ分析による分類

検体間の類似度を求めた結果は，類似度の数値が並ぶだけの大きな表になるため，全体像を把握することが困難になる．そこで類似度に基づき階層型クラスタ分析を行い検体間の関係の可視化を試みる．

3.3 提案手法と関連研究の比較

本論文で提案する分類手法は，上で述べた既存研究と同様に，検体から特徴抽出を行い類似度を求めるという方法である．我々が過去に提案した研究 [13] に対して，本研究では検体数を増やしており，過去の提案で不足していた評価実験を加え，また階層型クラスタ分析の結果に対する検証も行った．文献 [13] では API 推移の頻度を用いて独自に定義した式による検体間の類似度（距離）を求めているが，本研究では API 推移を集合の要素と見なして Dice 係数を適用した．

静的解析で必要となるマルウェアの，パック前のバイナリイメージを得るアンパックの手法については，我々の研究 [4] を含めて様々な研究が行われている [18], [19], [20], [21], [22], [23]．本研究では，我々が行ったアンパッカに関する研究 [4] によりすでにアンパックを試みた検体を用いるが，他のアンパッカに関する研究で得られた成果を用いることも可能である．ゆえに文献 [7], [13] のようにアンパックの方法は本研究の対象とせず，本研究では「マルウェアのパックされる前のバイナリイメージが入手可能である」ことを前提とする．

文献 [6] では API（システムコール）の前後関係あるいは依存関係を動的解析を用いて抽出しているが，本研究の提案はこれを静的解析により API 推移として抽出するものである．

文献 [7], [8] と同様に本研究でも制御フロー解析を行いグラフを取得しているが，本研究では API を呼び出すに至るバイナリコードを削除して，制御フロー解析のグラフを API 推移のグラフに収縮させている．本研究では同じソースコードから作られたマルウェアであれば，コンパイラなどの違いにより変わることはない API の呼び出しに注目する．文献 [7] では API を扱わないので，API を実行時に取得する場合にも対応できるが，API を呼び出すに至るバイナリコードの影響をより大きく受ける可能性がある．

また文献 [10] は本研究と同様に API 呼び出しをグラフ化しており，またグラフを直接比較していないところも本研究とも類似している．しかし本研究とは異なり文献 [10] では API に関するブロックとエッジの数を比べることで類似度を算出している．

一方，文献 [8], [9], [14] はバイナリコードの違いも特徴ととらえて分類している．これらの方法では同じソースコードから作られたマルウェアであっても，コンパイラや最適化法の違いなどにより異なるバイナリコードが生成されたときには，類似度が小さくなる可能性がある．そのため，ソースコードを基に分類を行うためには，本研究の提案する方法が有効である．しかしバイナリコードそのものの違いを重要視するならば，文献 [8], [9], [14] などの方法が有効である．

文献 [11], [12], [15], [16] ではグラフを扱わないので本研究よりも高速な処理が期待できるが，API 呼び出しをグラフ化する本研究は API 呼び出しの前後関係をより正確にとらえることができる．

本研究では文献 [5], [13], [14], [16] と同様に類似度を定義して階層型クラスタ分析を行う．

4. 実験

4.1 実験環境とマルウェア検体

提案手法の評価実験のため**表 3** の実験環境と**表 4** の我々が作成したプログラム，耐性評価実験のためにバージョンの異なる同種のマルウェアのソースコード，性能評価実験のために 4,684 種類のマルウェアの検体を準備した．

表 4 の disw32 [*1] は 3.1.1 項で説明した逆アセンブラである．ctlflw.pl は制御フロー解析を行いグラフを Graphviz [*2] の DOT 言語で出力する．apiseq.pl は ctlflw.pl

*1 IWM Utilities http://gtklab.sourceforge.jp/iwmutlis/
*2 Graphviz - Graph Visualization Software http://www.graphviz.org/

© 2013 Information Processing Society of Japan

表 5 評価用コンパイラ
Table 5 Compilers for evaluation.

Compiler (abbr.)	Version	Optimization Option
Microsoft Visual C++ (msvc)	16.00.40219.01	/Od (Disable Optimization) /Ox (Maximum Optimization)
MinGW (mingw)	4.6.1	-O0 (Disable Optimization) -O3 (Maximum Optimization)

表 3 実験環境
Table 3 Environment.

CPU	Pentium M 1.20 GHz
Memory	1 GB
OS	Ubuntu 10.04 LTS

表 4 プログラム
Table 4 Programs.

Name	Language	Description
disw32	C	Disassembler
ctlflw.pl	Perl	Control flow analysis
apiseq.pl	Perl	Extract API sequence
ascomp	C	Calculate similarity
dendro	C	Draw dendrogram with Graphviz
setup.pl	Perl	Classification

の出力結果から API 推移を表すグラフを DOT 言語で出力する．ascomp は 2 つ以上の apiseq.pl の出力結果から，それぞれの検体間の類似度を求める．dendro は apiseq.pl の出力結果に対して階層型クラスタ分析を行いグラフを DOT 言語で出力する．setup.pl はこれらのプログラムを検体に対して実行し，その出力結果から指定した類似度を閾値としてクラスタを生成する．

Microsoft Visual C++ 2010 の SDK に含まれる，インポートライブラリで定義されている 18,095 種類の API を対象とする．ただし API 呼び出しの間に挿入される可能性がある GetLastError，Sleep，SleepEx は対象としない．

4.1.1 耐性評価実験の検体

耐性評価実験のために，**表 5** の 2 種類のコンパイラと，それぞれ 2 種類の最適化オプションを指定した合計 4 種類のコンパイル方法で，**表 6** のソースコード[*3]をコンパイルしたバイナリイメージを準備した．これらのバイナリイメージを用いて，下記の 2 点の類似度を取得することで，提案手法の類似度の精度を評価する．

(1) 同じコンパイラ・最適化オプションでコンパイルされた，別種・亜種のマルウェア間の類似度
(2) 異なるコンパイラ・最適化オプションでコンパイルされた，同じマルウェア間の類似度

表 6 評価用マルウェア
Table 6 Malware for evaluation.

Name (abbr.)	Supported Compiler
sdbot v0.4b (sdbot04b)	msvc, lcc-win32
sdbot v0.5a (sdbot05a)	msvc, lcc-win32
sdbot v0.5b (sdbot05b)	msvc, lcc-win32, mingw
rxBot v0.7.7 Sass (rxbot)	msvc

表 7 sdbot のソースコード行数
Table 7 The number of lines for sdbot.

Name	LOC
sdbot04b	1,601
sdbot05a	1,902
sdbot05b	2,173

表 8 共通する行数
Table 8 The number of common lines.

	sdbot05a	sdbot05b
sdbot04b	1,309	1,188
sdbot05a		1,661

表 9 diff コマンドに基づく類似度
Table 9 Similarity matrix based on diff command.

	sdbot05a	sdbot05b
sdbot04b	0.75	0.63
sdbot05a		0.82

表 7 は 3 種類の科名 sdbot のソースコードの行数，**表 8** は各ソースコードの共通する行数である．**表 9** はソースコードの行数を集合の要素数，共通行の行数を積集合の要素数と見なし，Dice 係数に基づいて算出した各ソースコード間の類似度である．これらの検体は文献 [14] の 5.4 節の考察[*4]に相当する．

4.1.2 性能評価実験の検体

性能評価実験のための 4,684 種類の検体は，我々が行ったアンパッカに関する研究 [4] によりすでにアンパックを試みた検体である．検体は 32 ビット Windows の実行可能ファイルであり，同研究により Microsoft Visual Basic で

*3 rxbot のソースコードは for 文ループ変数に関する古い C++の仕様に基づいて記述されているなどの問題があり，表 5 のコンパイラでコンパイルできない．そのため本研究では元の意味を変えることがない範囲でソースコードを修正してコンパイルした．

*4 sdbot04b には msvc 版と lcc-win32 版の 2 つの異なるソースコードがあり，文献 [14] では lcc-win32 版のソースコードの行数が提示されている．しかし本研究と文献 [14] では msvc で作成されたバイナリイメージが使われているので，本論文では msvc 版のソースコードの行数を表 7 に提示している．

© 2013 Information Processing Society of Japan

表 10 類似度行列，最適化オプション ‘Od’
Table 10 Similarity matrix, optimization option ‘Od’.

	sdbot05a	sdbot05b	rxbot
sdbot04b	0.67	0.61	0.05
sdbot05a		0.81	0.08
sdbot05b			0.08

表 11 類似度行列，最適化オプション ‘Ox’
Table 11 Similarity matrix, optimization option ‘Ox’.

	sdbot05a	sdbot05b	rxbot
sdbot04b	0.71	0.67	0.08
sdbot05a		0.79	0.09
sdbot05b			0.11

表 12 ‘sdbot05b’ に関する類似度
Table 12 Similarity matrix for ‘sdbot05b’.

	mingw (O3)	msvc (Od)	msvc (Ox)
mingw (O0)	0.85	0.80	0.79
mingw (O3)		0.74	0.75
msvc (Od)			0.85

作成されていないことが確認されている．

一般にマルウェアの名称は，科名（Family Name）と亜種名（Variant Name）で構成される．たとえば W32/Mydoom.A ならば Mydoom が科名，A が亜種名である．本研究では世界各国のマルウェア研究者の報告に基づいて作成されている WildList *5による命名を採用する．検体はすべて 32 ビット Windows の実行可能ファイルなので接頭辞の W32 は以後省略する．

4.2 実験結果

4.2.1 耐性評価実験の結果

表 10，**表 11** は表 5 に示す同じコンパイラ・最適化オプションで，表 6 のソースコードをコンパイルしたバイナリイメージの間の類似度である．**表 12** は表 5 に示す異なるコンパイラ・最適化オプションで表 6 の亜種 sdbot05b のソースコードをコンパイルしたバイナリイメージの間の類似度である．

4.2.2 性能評価実験の結果

4,684 種類の検体に対して，逆アセンブル，制御フロー解析，API 推移抽出を行ったところ，API 推移を抽出できた検体数は 1,821 種類，科名では 113 種類であった．1,821 種類について表 3 に示す計算機上で分類プログラムを実行した結果，逆アセンブルに要した時間は 5 時間 26 分 3 秒，制御フロー解析に要した時間は 43 分 38 秒，API 推移の抽出に要した時間は 3 時間 7 分 15 秒であった．**図 7**，**図 8**，**図 9** はそれぞれ逆アセンブル，制御フロー解析，API 推移

*5 The WildList Organization International
http://www.wildlist.org/

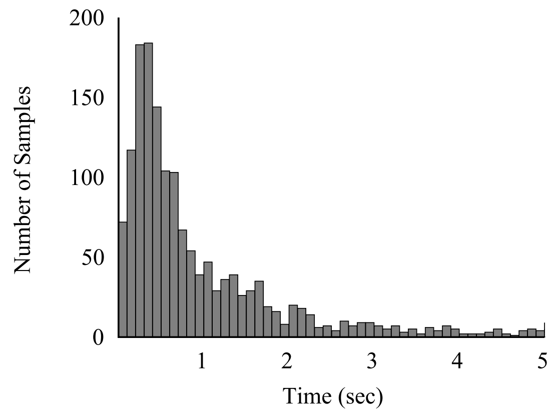

図 7 逆アセンブルの時間と検体数
Fig. 7 Disassemble time vs. samples.

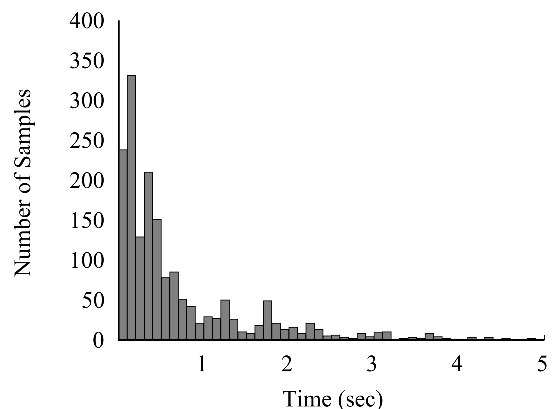

図 8 制御フロー解析の時間と検体数
Fig. 8 Control flow analysis time vs. samples.

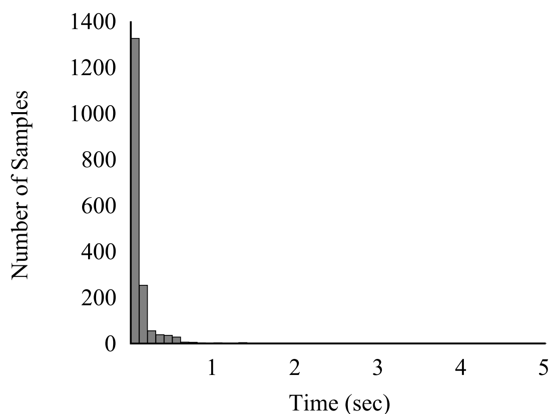

図 9 API 推移の抽出の時間と検体数
Fig. 9 API sequence time vs. samples.

抽出に要した時間と検体数の関係を表すグラフである．グラフは横軸が 0.1 秒ごとの時間，縦軸が検体数を示している．時間が 5 秒以上であった検体は逆アセンブルでは 231 種類，制御フロー解析では 92 種類，API 推移抽出では 55 種類であった．最も時間がかかった検体は，逆アセンブルでは 41 分 9 秒，制御フロー解析では 50 秒，API 推移抽出では 1 時間 3 分であった．**図 10** の横軸は抽出できた API 推移の数，縦軸は検体数である．136 種類の検体は抽出で

© 2013 Information Processing Society of Japan

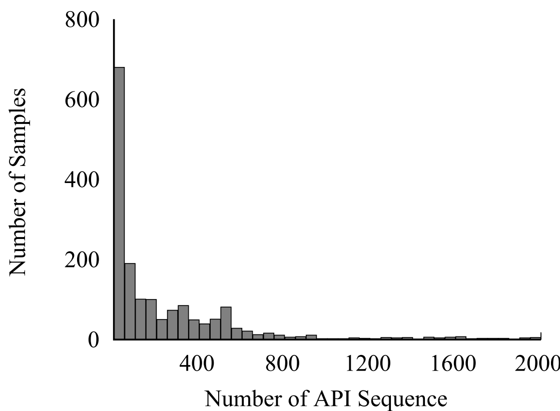

図 10 API 推移の数と検体数
Fig. 10 Number of API sequence vs. samples.

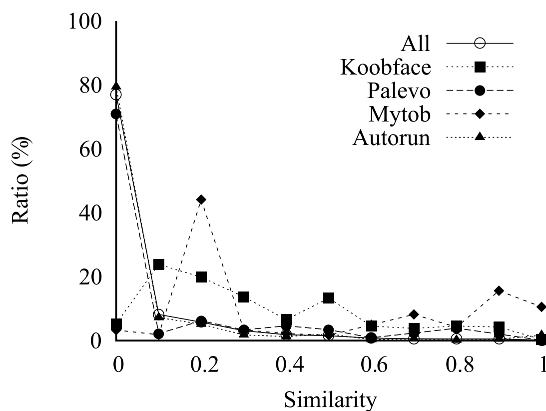

図 11 類似度の分布
Fig. 11 Distribution of similarity.

表 13 科名の上位 10 種類
Table 13 Top 10 families.

	Family	Num.
1	Koobface	208
2	Palevo	129
3	Mytob	92
4	Autorun	79
5	Ircbot	42
6	Kolab	29
7	Sdbot	27
8	Yahos	24
9	Areses	21
10	Agent	20

きた API 推移の数が 2,000 を超えていた．

検体から抽出した API 推移に基づいて検体間の類似度を求めた．検体間の類似度を求めるのに要した時間は 1 時間 41 分 36 秒であった．**図 11** は，検体間全体の類似度および**表 13** に示す科名に属する検体数が多い上位 4 種類について，検体間の類似度の分布を示すグラフである．横軸は 0.1 ごとの類似度，縦軸はその範囲の類似度が対象に占める割合を表す．検体数が多いためすべての類似度を表にすることはできないので，それらの中から検体を選びその検体間の類似度を**表 14** に示す．

図 12 は 1,821 種類の検体のうちランダムに選んだ 500 種類について，階層型クラスタ分析による可視化を行った結果である．この樹形図ではノードは検体を表し，類似度が高いノードは近くに配置される．表 13 の検体はそれぞれ固有の形のノードで表し，それ以外の検体は同じ形のノードで表す．今回のケースでは，類似度 0.8 以上の検体でクラスタを作成したとき，56 のクラスタが作成され，172 種類の検体がクラスタに属さなかった．**表 15** はクラスタに属する検体数が多い 10 のクラスタの，そのクラスタで最も多い科名，その科名の検体数，クラスタの検体数を示す．同様に類似度 0.6 以上では**表 16** になり，40 のクラスタが作成され，119 種類の検体がクラスタに属さなかった．

5. 議論

5.1 耐性評価実験の評価

表 10 と表 11 の双方で 3 種類の sdbot 間の類似度は高い値になっている．一方，sdbot と rxbot の類似度は低いことから，亜種と別科の関係が類似度に反映されている．また表 9 では sdbot05a と sdbot05b の類似度が最も高く，次に sdbot04b と sdbot05a，sdbot04b と sdbot05b の類似度が最も低い．表 10 と表 11 でもこの大小関係は変化していない．

表 12 では，同じソースコードであってもコンパイラや最適化オプションによって制御フローが変化すること，コンパイラがランタイムライブラリの初期化やスタックのチェックのために，固有のコードをバイナリイメージに付け加えること，などが原因で類似度は 1 にはならなかった．しかし文献 [14] とは異なり類似度は十分に高い値になっており，ソースコードに基づいてマルウェアの分類を行うという，本研究の目的を達成している．

表 10 と表 11，表 12 を比べると，同じソースコードで異なるコンパイラにおける類似度よりも，異なるソースコードで同じコンパイラ・最適化オプションにおける類似度の方が高くなっている事例がある．通常はソースコードの改変はその部分にのみ影響を与えるが，コンパイラの変更はバイナリコード全体に影響を与えることが原因である．

5.2 性能評価実験の評価

5.2.1 API 推移抽出が困難なマルウェア

今回のケースで，API 推移を抽出できた検体は 4,684 種類中 1,821 種類（約 39%）であった．API 推移を抽出できない原因は 2 つ考えられる．1 つは本研究では検体がアンパック済みであることが求められており，それが十分でなかったことである．実際に我々がいくつかの検体を解析したところ，IAT（Import Address Table）の再構築が不完全であったり，アンパックの途中で終了していた検体があった．この場合，研究の前段階であるアンパッカを改善する

© 2013 Information Processing Society of Japan

表 14　マルウェアの検体間の類似度

Table 14 Similarity matrix between malware samples.

	Koobface.026	Koobface.064	Mydoom.60	Mytob.120	Netsky.AA	Palevo.186	Palevo.241	Scar.04	Slenfbot.47
Autorun.019	0.10	0.10	0.07	0.08	0.03	0.02	0.02	0.01	0.01
Koobface.026		0.99	0.03	0.07	0.02	0.02	0.02	0.00	0.02
Koobface.064			0.03	0.07	0.02	0.02	0.02	0.00	0.02
Mydoom.60				0.63	0.09	0.02	0.02	0.00	0.03
Mytob.120					0.08	0.01	0.01	0.00	0.03
Netsky.AA						0.37	0.52	0.16	0.19
Palevo.186							0.45	0.25	0.26
Palevo.241								0.19	0.23
Scar.04									0.47

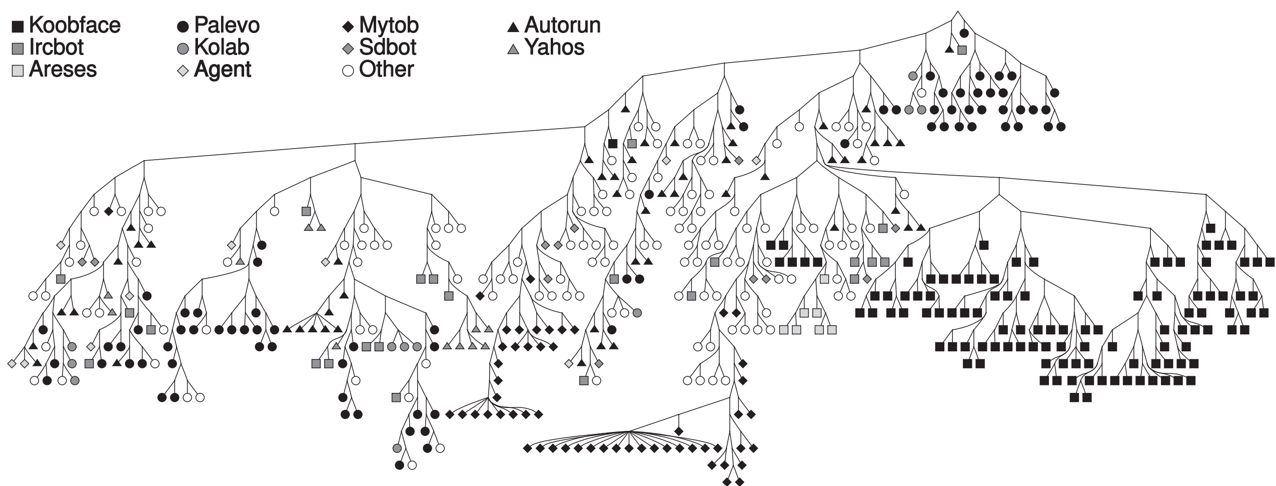

図 12　階層型クラスタ分析木

Fig. 12 Hierarchical clustering tree.

表 15　類似度 0.8 以上のときの上位 10 クラスタ

Table 15 Top 10 clusters ($Similarity \geq 0.8$).

	Family	Num.	Total	Ratio (%)
1	Koobface	34	34	100.0
2	Palevo	13	30	43.3
3	Mytob	28	28	100.0
4	Koobface	26	26	100.0
5	Palevo	18	21	85.7
6	Autorun	4	18	22.2
7	Koobface	12	12	100.0
7	Mytob	12	12	100.0
9	Sdbot	2	8	25.0
10	Koobface	8	8	100.0

表 16　類似度 0.6 以上のときの上位 10 クラスタ

Table 16 Top 10 clusters ($Similarity \geq 0.6$).

	Family	Num.	Total	Ratio (%)
1	Koobface	74	74	100.0
2	Palevo	13	56	23.2
3	Palevo	10	36	27.8
4	Mytob	28	34	82.4
5	Palevo	18	25	72.0
6	Mytob	23	23	100.0
7	Koobface	20	20	100.0
8	Sdbot	2	9	22.2
9	Areses	7	7	100.0
10	Looked	7	7	100.0

か，文献 [5], [6] のような動的解析で分類する必要がある．

もう 1 つの原因は実行時に API のアドレスが取得されることである．通常，Windows では API のアドレスはプログラムが実行される前に IAT の記述に基づいて OS によって解決される．しかし Stration などの一部のマルウェアは，マルウェアが実行されているときに必要な API のアドレスを随時取得している．そのため本研究のようにマルウェア検体を実行しない静的解析では，実行時に API のアドレスが取得される API の呼び出しを抽出できない．この API 抽出に関する問題は静的解析のみでは解決できないた

© 2013 Information Processing Society of Japan

め，別途動的解析の結果に基づいたAPI呼び出しアドレスの特定など，他の手法の併用を検討するか，文献 [7], [14] のように API に依存しない別の特徴を抽出する方法を検討する必要がある．

5.2.2　実行時間

図 7，図 8，図 9 より，多くの検体は数秒で解析を終えているものの，一部のバイナリコードのサイズが大きい検体では，解析に時間がかかっていることが分かった．図 10 において，API 推移を抽出できた検体の中では，その数が 100 未満が多数を占めていた．一方で少数ではあるが 2,000 を超える検体があった．これは Downloader のよう単一の目的に特化したマルウェアと，Bot のような複数の機能を持つ汎用的なマルウェアの違いであると考えられる．分類プログラムの実行時間の期待値は，対象とするマルウェアの検体数に比例して増大するので，並列実行可能なプロセッサあるいはクラスタリング分散処理によって，必要な実行速度が得られる．

5.2.3　類似度の分布

図 11 において，全体を見ると求めた類似度の 80%以上は 0.1 未満であった．113 種類の科名であることから考えると，同じ科名の検体間で類似度が求められるのは 1%に満たないので，この結果は妥当である．

同一の科名の中では，Koobface と Mytob は全体の傾向とは異なる結果となった．Koobface の類似度の分布は 0.1 以上 0.2 未満，Mytob の類似度の分布は 0.2 以上 0.3 未満が最大となっている．さらにそれ以上に大きな類似度の分布も多くなっており，同一の科名では類似度が高くなることを示している．

一方，同一の科名であるが Palevo は全体よりも若干類似度が高くなっており，Autorun は全体の傾向とほとんど違いがなかった．これは Autorun はソースコードの関連性による命名ではなく，メディアの自動実行機能を利用して広まるマルウェアに対して Autorun という科名が与えられていることが原因である．我々が Autorun の検体をいくつか解析したところ，異なる C 言語のソースコードから作成された検体や，本研究の実験で用いた検体ではないが Microsoft Visual Basic で作成された検体もあった．また他の科名でもこのようにソースコードの特徴以外の基準で命名されることがあるため，マルウェアの動作として命名した科名と，静的解析を行った結果としての類似度分布には，差異が認められる．

5.2.4　検体間の類似度と検体の関連性

表 14 では多くの組合せでは類似度が 0.1 以下であり，これらの検体は関連性がないと推測が可能である．科名が同じ Koobface.026 と Koobface.064，Palevo.186 と Palevo.241 の類似度は高くなった．Koobface.026 と Koobface.064 はともに Facebook を利用して広まるマルウェアであり，Palevo.186 と Palevo.241 はともにメディアの自動実行機能や P2P ファイル共有，インスタントメッセンジャで広まるマルウェアである．また Mydoom.60 と Mytob.120 の類似度は 0.63 であった．図 11 より類似度が 0.2 以上となるのは稀であるので，これらの検体は何らかの関連性がある．我々が Mydoom や Mytob の亜種を解析したとき，制御フローや使われている API，関数の呼び出し関係の類似性から Mytob は Mydoom のソースコードを流用した可能性が高いと推測していた．特にメールの送信エンジンは共通しており，両者とも DNS に問い合わせて送信先の SMTP サーバに直接接続してメールを送信するが，それが失敗したときにはレジストリから標準の SMTP サーバを取得してメールを送信する．この実験の結果により，解析者の推測に根拠が与えられた．同様に科名が異なっているが Scar.04 と Slenfbot.47 はともにメディアの自動実行機能で広まるマルウェアであり，先頭部分のコードが類似していた．

我々は準備した検体のすべてを解析してはいないが，Netsky.AA と Palevo.241 のように類似度が高くなっているならば，何らかの関連性があると推測が可能である．

5.2.5　階層型クラスタ分析

図 12 の樹形図では Koobface や Mytob などは 1 つまたは複数の枝の下にグループを形成した．これらの亜種は同一の起源を持つソースコードから作成されているため，同じ科名の検体間での類似度が高くなったことが原因である．一方，Autorun はグループを形成せず全体に散らばった．表 15 と表 16 では Koobface と Mytob，Areses，Looked のクラスタでは同一の科名で占められた．一方，Autorun や Palevo，Sdbot が多数を占めるクラスタでは他の科名の検体がクラスタに含まれた．これらは前述の命名の方法や基準のあいまいさが原因である．

6.　まとめ

本研究で提案した方法でソースコードの特徴からマルウェアを分類することは可能であった．類似度からマルウェアの機能を推定し，マルウェア解析や対策に寄与すると考えられる．しかし必ずしも既存のマルウェアの分類と同じ分類結果ではなかった．これは Autorun のようにソースコードの特徴以外の理由で命名が行われている検体があることが原因である．

また本研究では共通する API 推移の割合を類似度として定義した．しかし既存のソースコードを元に新たなマルウェアが作られるならば，あるマルウェアの API 推移を別のマルウェアがどれだけ含んでいるかを見ることで，マルウェアの包含関係を数値化することが可能である．この場合，元のマルウェアに機能を追加したならば，新たなマルウェアは元のマルウェアの特徴をすべて持っていることになる．複数のマルウェアのソースコードから新たなマルウェアが作られたならば，それぞれの元のマルウェアに対

© 2013 Information Processing Society of Japan

して，本研究で定義した類似度よりも高い値となる．類似度をどのように定義するかも検討する必要がある．

本研究の提案は Microsoft Visual Basic で作成されたマルウェアを除く Windows で動作するマルウェアを対象としている．しかし制御フロー解析を行い API 呼び出し（あるいは何らかの API に相当する機能）に注目するという方法自体は異なる環境にも適用可能である．

本研究の実験には 10 時間以上の時間を要した．この中で時間がかかった処理は逆アセンブルと API 推移の抽出であった．特に API 推移の抽出は Perl のスクリプトで行っているため，この処理を C 言語で再実装し，ネイティブコンパイラで生成したバイナリプログラムに置き換えれば，所要時間は改善されると考えられる．また一部の検体において，解析に長い時間を要した原因を分析する必要がある．

謝辞　本研究を行うにあたって日本コンピュータセキュリティリサーチ株式会社主席研究員の遠藤基氏の協力を得た．

参考文献

[1] Mell, P., Kent, K. and Nusbaum, J.: Guide to Malware Incident Prevention and Handling, National Institute of Standards and Technology (online), available from ⟨http://csrc.nist.gov/publications/nistpubs/800-83/SP800-83.pdf⟩ (accessed 2012-04-01).

[2] 独立行政法人情報処理推進機構：脆弱性を狙った脅威の分析と対策について Vol.2，独立行政法人情報処理推進機構（オンライン），入手先 ⟨http://www.ipa.go.jp/security/vuln/report/documents/newthreat200907.pdf⟩（参照 2012-04-01）．

[3] Bu, Z., Dirro, T., Greve, P., Lin, Y., Marcus, D., Paget, F., Schmugar, C., Shah, J., Sommer, D., Szor, P. and Wosotowsky, A.: McAfee Threats Report: First Quarter 2012, McAfee Labs (online), available from ⟨http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q1-2012.pdf⟩ (accessed 2012-06-10).

[4] 岩本一樹，和崎克己：マルウェアアンパッキングにおけるランタイムライブラリのコード比較によるオリジナルエントリーポイント検出，電子情報通信学会技術研究報告，ICSS，情報通信システムセキュリティ，Vol.111, No.82, pp.57–62 (2011).

[5] Bailey, M., Oberheide, J., Andersen, J., Mao, Z.M., Jahanian, F. and Nazario, J.: Automated classification and analysis of Internet malware, *Proc. 10th international Conference on Recent Advances in Intrusion Detection*, Berlin, Heidelberg, pp.178–197, Springer-Verlag (2007).

[6] Christodorescu, M., Jha, S. and Kruegel, C.: Mining specifications of malicious behavior, *Proc. 1st India Software Engineering Conference*, New York, NY, USA, ACM, pp.5–14 (2008).

[7] Flake, H.: Automated Unpacking and Malware Classification, *Black Hat Japan*, Tokyo, Japan, pp.61–88 (2007).

[8] Kruegel, C., Kirda, E., Mutz, D., Robertson, W. and Vigna, G.: Polymorphic worm detection using structural information of executables, *Proc. 8th International Conference on Recent Advances in Intrusion Detection*, Berlin, Heidelberg, pp.207–226, Springer-Verlag (2006).

[9] Zhang, Q. and Reeves, D.S.: MetaAware: Identifying Metamorphic Malware, *Computer Security Applications Conference, Annual*, pp.411–420 (2007).

[10] Han, K.S., Kim, I.K. and Im, E.G.: Detection Methods for Malware Variant Using API Call Related Graphs, *Proc. International Conference on IT Convergence and Security 2011*, Suwon, Korea, Vol.120, pp.607–611, Springer, (2012).

[11] Han, K.S., Kim, I.K. and Im, E.G.: Malware Classification Methods Using API Sequence Characteristics, *Proc. International Conference on IT Convergence and Security 2011*, Suwon, Korea, Vol.120, pp.613–626, Springer (2012).

[12] Altaher, A., Supriyanto, ALmomani, A., Anbarm, M. and Ramadass, S.: Malware detection based on evolving clustering method for classification, *Scientific Research and Essays*, Vol.7, No.22, pp.2031–2036 (2012).

[13] 岩本一樹，和崎克己：静的解析によるマルウェアの分類と結果の検討，マルチメディア，分散，協調とモバイル（DICOMO2010）シンポジウム，pp.477–491 (2010).

[14] 岩村　誠，伊藤光恭，村岡洋一：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，情報処理学会論文誌，Vol.51, No.9, pp.1622–1632 (2010).

[15] Ye, Y., Wang, D., Li, T. and Ye, D.: IMDS: Intelligent malware detection system, *Proc. 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, NY, USA, ACM, pp.1043–1047 (2007).

[16] Ye, Y., Mei, Y. and Peng, R.: MCNS: Intelligent Malware Categorization and Naming System, *AVAR 2009 Conference*, Kyoto, Japan, pp.15–25 (2009).

[17] Schwarz, B., Debray, S. and Andrews, G.: Disassembly of Executable Code Revisited, *Proc. IEEE 2002 Working Conference on Reverse Engineering* (*WCRE*), IEEE Computer Society, pp.45–54 (2002).

[18] Josse, S.: Secure and advanced unpacking using computer emulation, *AVAR 2006 Conference*, Auckland, New Zealand, pp.174–190 (2006).

[19] Royal, P., Halpin, M., Dagon, D., Edmonds, R. and Lee, W.: PolyUnpack: Automating the Hidden-Code Extraction of Unpack-Executing Malware, *Proc. 22nd Annual Computer Security Applications Conference*, Washington, DC, USA, IEEE Computer Society, pp.289–300 (2006).

[20] Kang, M.G., Poosankam, P. and Yin, H.: Renovo: A hidden code extractor for packed executables, *Proc. 2007 ACM Workshop on Recurring Malcode*, New York, NY, USA, ACM, pp.46–53 (2007).

[21] Martignoni, L., Christodorescu, M. and Jha, S.: Omniunpack: Fast, generic, and safe unpacking of malware, *Proc. Annual Computer Security Applications Conference* (*ACSAC*) (2007).

[22] Kim, H.C., Inoue, D., Eto, M., Takagi, Y. and Nakao, K.: Toward Generic Unpacking Techniques for Malware Analysis with Quantification of Code Revelation, *Joint Workshop on Information Security 2009*, Kaohsiung, Taiwan (2009).

[23] Lungu, C. and Botis, M.: CJ-Unpack: Efficient Runtime Unpacking System, *19th EICAR Annual Conference*, Paris, France, pp.235–253 (2010).

© 2013 Information Processing Society of Japan

岩本 一樹 （学生会員）

1998年東京電機大学理工学部情報科学科卒業．同年日本コンピュータセキュリティリサーチ株式会社入社，マルウェア解析に従事．2012年独立行政法人情報処理推進機構非常勤研究員．1999年 Anti Virus Asia Researchers（AVAR）会員，2011年 AVAR 理事．2008年信州大学大学院工学系研究科修士課程情報工学専攻修了，2010年同大学院総合工学系研究科システム開発工学専攻博士課程入学．

和﨑 克己 （正会員）

1991年信州大学工学部情報工学科卒業，1993年同大学大学院工学系研究科博士前期課程情報工学専攻修了，1994年同博士後期課程システム開発工学専攻退学，同年長野工業高等専門学校電子制御工学科助手，1998年信州大学工学部情報工学科助手，2001年同大学工学部助教授．現在，信州大学工学部教授．博士（工学）．2003年，2005年カナダ・アルバータ州立大学計算科学科客員研究員．2008年独立行政法人新エネルギー・産業技術総合開発機構（NEDO）技術評価委員．並列分散システムのモデル化と解析，非同期システムの数学モデルと形式検証，モデル検査系向けハードウェアコンパイラに関する研究に従事．IEEE，電子情報通信学会，電気学会，教育システム情報学会各会員．

© 2013 Information Processing Society of Japan