マルウェア対策技術の精度向上を目的とした コンパイラおよび最適化レベルの推定手法

碓井 利宣 †　　松浦 幹太 †

† 東京大学生産技術研究所
153-8505　東京都目黒区駒場 4-6-1
{tusui, kanta}@iis.u-tokyo.ac.jp

あらまし マルウェアによる脅威が増加している．これらに対して効果的な対策を実施していくには，マルウェアの実行ファイルが持つ様々な情報を最大限に利用することが必要である．機械語命令列は，マルウェアの動作に関わる重要な情報である．しかし，コンパイラの種類や最適化レベルなどの要素によって性質を大きく変え得るため，マルウェア対策に用いることが難しい．本研究では，実行ファイルの生成に用いられたコンパイラの種類および最適化レベルを推定する手法を提案する．実装したシステムを用いてマルウェア検体に対して実験を行う．その結果を通して，本研究がコンパイラがマルウェア対策に与える影響をどのように削減できるかについて，論じる．

Compiler and Optimization Level Estimation for Improving Anti-malware Technologies

Toshinori Usui†　　Kanta Matsuura†

†Institute of Industrial Science, The University of Tokyo
4-6-1 Komaba, Meguro-ku, Tokyo 153-8505, JAPAN
{tusui, kanta}@iis.u-tokyo.ac.jp

Abstract For effective countermeasures against malwares, we need to utilize all the utilizable information which belongs to malwares' executable files. Machine instruction sequence is an important information which is relevant to malwares' behavior. However, utilizing the information is difficult because varieties of compilers and optimization levels tend to affect strongly. In this paper, we propose a method to specify the compiler and optimization level that are adopted to generate the executable files of malwares. Additionally, experimentations are conducted with the datasets of malware samples. This paper also discusses how the proposed method decreases ill effects against anti-malware technologies caused by compilers.

1　はじめに

インターネットの普及にともなって，利用者に対する脅威も増加してきた．近年，悪意を持ったソフトウェアである，マルウェアによる脅威が顕在化している．マルウェアの数は多く，ウイルス対策ソフトウェアベンダでは，2013 年 3 月時点で延べ 1 億 2800 万以上の種類のマルウェアが保有されていると報告している [4]．また，その増加も速く，毎月 500 万もの新たなマルウェアが観測されているとも報告している．したがって，マルウェアは継続して増加していることが

分かる．マルウェアの解析には，次の 2 つの手法がある．1 つは，マルウェアを実際に実行させて，その動作を監視することによって解析する動的解析である．もう 1 つは，マルウェアを実行させずに，マルウェアの実行ファイルから情報を抽出することによって解析する静的解析である．しかし，動的解析では，マルウェアを実際に動作させて監視する必要があるという特性から，一定の時間を必要とし，多くのマルウェアを一度に解析することは難しい．そのため，一日に数万もの新たなマルウェアが出現する現状では，動的解析の手法のみによる解析は限界を迎えていると言える．したがって，ソフトウェアによる静的解析のように，高速で一度に多くのマルウェアを解析できる手法が必要不可欠である．マルウェアの脅威に対して，静的解析で得られた情報に基づいて，効果的な対策を効率よくとっていくには，マルウェアの持つ様々な特徴を捉えた手法を用いることが必要である．近年の研究では，次のような特徴量を用いた研究が行われている．API 呼び出しおよびコールグラフ，文字列，ファイルヘッダ，ファイル全体および一部のハッシュ値などである．このことから，マルウェアのファイルに含まれる情報の多くが特徴量として用いられていることが分かる．一方で，現在では特徴量として十分に用いるのが難しい情報も存在する．マルウェアの機械語命令列がその一つである．機械語命令列は，マルウェアの実行ファイルに含まれていて，その命令の流れに基づいてマルウェアが実行される．したがって，機械語命令列はマルウェアの動作に大きく関わっており，重要な特徴の一つであると考えることができる．しかし，機械語命令列には，同じソースコードから作成されたものであっても，利用したコンパイラの種類や設定された最適化レベルによって大きく変化するという特徴がある．前者は，コンパイラの種類によって，用いられる最適化やコード生成のアルゴリズムが異なるためである．後者は，最適化レベルによって，どの種類の最適化が実施されるかが異なるためである．これらによって，マルウェア対策技術に機械語命令列を特徴量として用いることが難しくなっている．たとえば，機械語命令列に基づいてマルウェアを検知しようとした場合を考える．同じソースコードから生成された，同じ動作をするマルウェアであっても，異なるコンパイラや最適化レベルによって機械語命令列は異なってくる．したがって，これらの違いに影響を受けて，正しくマルウェアの持つ機械語命令列の特徴を捉えることができず，検知に失敗する可能性が考えられる．また，機械語命令列に基づいてマルウェアを分類しようとした場合を考える．検知の場合と同様の理由によって，同種のマルウェアであっても，別の種類として分類されてしまうことが考え得る．

このことについて，岩村らの研究 [3] で実験が行われている．岩村らは，機械語命令列の最長一致部分列を用いて，マルウェア同士の類似度を計算し，分類する手法を提案した．その中で，コンパイラの種類や最適化レベルの違いが類似度計算にどのように影響を与えるかの実験をしている．実験の結果から，コンパイラや最適化レベルの違いはマルウェアの種類の違いよりも，類似度の算出に大きく影響する，と結論づけられている．実験の内容については，第 2.1 節に詳細を記述する．本来，分類結果にはマルウェアの種類の違いが表れることが目的であり，このようなコンパイラや最適化レベルによって受ける影響は一つの解決されるべき問題であると言える．

本研究では，コンパイラの種類および最適化レベルを推定する手法を提案する．本手法では，まず，一般に用いられているコンパイラのシグネチャとなる特徴を抽出する．それを用いたパターンマッチングによって，コンパイラの種類を推定する．また，各コンパイラの最適化レベルを用いた実行ファイルからそれぞれを表す特徴となり得る値を抽出し，それらを特徴量として機械学習の手法を適用する．実行ファイルがどの最適化レベルを用いているクラスに属するかの分類を実施することで，最適化レベルを推定する．これらの手法を実装し，マルウェア検体に対して実験を行った．

本手法の成果によって，2 つの利点が考えられる．1 つめは，コンパイラの種類や最適化レベ

ルを推定し，マルウェアを分けることによって，同一のコンパイラ，最適化レベルを用いたマルウェアの中で検知や分類などを考えるようにすることができる．これによって，コンパイラの種類や最適化レベルの差違による大きな影響を受けることなく，機械語命令列を用いた検知や分類などのマルウェア対策技術を適用することができるようになる．2つめは，さらに一歩踏み込んだ手法を考える．最適化レベルにあわせて再最適化することで，最適化レベルの差違をなくすことができるというものである．たとえば，最適化を全くしていない，あるいは最適化をほとんどしていないマルウェアに対して，最適化レベルが最高の水準となるまで最適化を施す．これによって，すべてのマルウェアをあらかじめ最大の最適化レベルでコンパイルされたマルウェアと同等の機械語命令列を持つような状態にする．結果として，最適化レベルの差違による影響を受けることなく，機械語命令列を用いた検知や分類を行えるようにするというものである．これらによって，コンパイラの種類や最適化レベルの変更といった，攻撃者による対策を防ぐことにより，現状では十分に用いられていない機械語命令列を利用したマルウェア対策技術を有用なものとすることが期待される．

2 関連研究

2.1 マルウェア対策技術に与える影響に関する研究

岩村らの研究 [3] で行われた実験では，あるマルウェアのソースコードを，2種類のコンパイラ，それぞれ2種類の最適化レベルの4種類の環境でコンパイルし，生成された4種類のマルウェアの実行ファイル間で類似度を計算している．その結果，異なるコンパイラで生成された実行ファイル同士では類似度は5%から10%の間の低い値に収まっていることが観測されている．また，同じコンパイラ同士であっても，最適化レベルが異なれば18.46%という低い値をとる場合も見られた．これらは，他の実験で観測された他種のマルウェア同士との類似度の値よりも低くなっている．

2.2 コンパイラの種類の推定に関する既存技術

コンパイラの種類の推定については，特に，ツールの開発が進んでいる．PEiD[2] や AT4RE FastScanner[1], RDG Packer Detector[6] などのツールが存在する．いずれも，マルウェアの PE ヘッダのダンプやパッカーの種類の検出を主要な機能としている．したがって，コンパイラの種類の推定は，開発の途中で付加的に追加された機能であり，パッカーの判定を行う中で，コンパイラの種類についても得られた情報があれば提供するという形態のものが多い．そのため，必ずしも十分な精度があるとは言えない．実際，本研究で行った実験では，コンパイラの種類が推定されなかったマルウェア検体も多く存在した．

2.3 最適化に関する研究

コンパイラの最適化レベルの推定に関する研究は現在のところ見られない．そのため，最適化レベルの推定に関わる，コンパイラの最適化手法に関する研究を挙げる．機械語命令列に大きな影響を与える最適化の手法の一つとして，ループ変換がある．ループ変換とは，動作自体は変えずにループの形態を変換することで，効率化を計る手法である．Wolf らの研究 [7] では，並列度を高める観点からループ変換の手法について議論している．また，McKinley らの研究 [5] では，データ局所性の向上の観点から，ループ変換の手法を挙げている．それぞれのループ変換の詳細な手法については，第 5.1 節にて記述している．

3 コンパイラによるコード生成の仕組み

コンパイラは，一般に次のような流れでコード生成を行う．(1) 読込み (2) 字句解析 (3) 構文解析 (4) 意味解析 (5) 中間語生成 (6) 中間語レベル最適化 (7) コード生成 (8) コードレベル最適化，である．読み込みから意味解析にか

けてが入力されたソースコードの解析を行う解析フェーズであり，中間後の生成からコードレベルの最適化にかけてがコード生成を行う組立フェーズである．解析フェーズでの動作においては，言語ごとに定められた規格があり，それに基づいた動作をする場合が多い．一方で，組立フェーズにおいては，特に各最適化やコード生成において，コンパイラによって動作に異なりが生じやすい．したがって，本研究では，最適化やコード生成の部分を扱うことになる．具体的には，最適化に影響を与える最適化レベルのオプションの設定や，コード生成に影響を与えるコンパイラの種類を研究の対象とする．

4 コンパイラの種類の推定

コンパイラの種類の推定は，実行ファイル内に含まれる情報に基づいてシグネチャを抽出し，それらに基づいたパターンマッチングによって行う．シグネチャとしては，以下の 2 つを用いる．1 つめは，PE ヘッダに含まれるリンカのバージョンを示すフィールドである．2 つめは，コンパイラの種類ごとに特有の文字列や API/DLL の名前である．

4.1 リンカのバージョンに基づいたシグネチャ

PE ヘッダには，MajorLinkerVersion と MinorLinkerVersion という 2 つのフィールドが存在し，それぞれが 16 進数の値を保持している．これらは，2 つのフィールドが合わさって，コンパイルの際に用いられたリンカのバージョンを示すものである．たとえば，MajorLinkerVersion が 0x08 であり，MinorLinkerVersion が 0x00 であれば，リンカのバージョンは 8.0 ということになる．多くのコンパイラについて，リンカのバージョンとは一対一に対応しており，このフィールドを参照することによって，どのコンパイラを用いて生成された実行ファイルであるかを推定することができる．一般的なマルウェアに見られるコンパイラの種類とそのリンカバージョンの対応を表 4.1 に示す．

リンカ バージョン	Hex	コンパイラ
2.25	0x0219	Delphi
2.2x	0x021x	gcc/g++（Cygwin/MinGW 含む）
5.0	0x0500	Borland C/C++
5.12	0x050c	MASM/TASM/FASM
6.0	0x0600	Microsoft Visual C/C++ 6.0
7.0	0x0700	Microsoft Visual C/C++ 7.0
7.10	0x070a	Microsoft Visual C/C++ 7.1
8.0	0x0800	Microsoft Visual C/C++ 8.0
9.0	0x0900	Microsoft Visual C/C++ 9.0

しかし，この手法では，マルウェアにおいては必ずしも確実にコンパイラの種類を特定できるとは限らない．その理由として，前述の MajorLinkerVersion と MinorLinkerVersion の 2 つのフィールドは書き換えることが可能であるためである．すなわち，このフィールドを任意の値に書き換えても，その実行ファイルの動作には影響を与えないのである．したがって，一部の攻撃者は，このフィールドを実在しないリンカのものに書き換えている．また，一部のパッカーによっても，このフィールドは書き換えられる．たとえば，WinUpack によってパッキングされたマルウェアは，MajorLinkerVersion に 4c を，MinorLinkerVersion に 6f を示す．したがって，リンカのバージョンのみに依存しないシグネチャも必要となる．

4.2 特徴的な文字列に基づいたシグネチャ

コンパイラには，その種類によって特徴的な文字列を実行ファイルに残す場合がある．また，インポートセクションに含まれる，実行ファイルがインポートする API や DLL の名前についても，コンパイラの種類を推定したり，絞り込んだりする要素となる．たとえば，Microsoft Visual C/C++を用いた場合，”This program cannot be run in DOS mode”という文字列が含まれる．一方，Delphi を用いた場合，”This program must be run under Win32”という文字列が含まれる．これを利用して，コンパイラを識別することができる．

4.3 シグネチャを用いたコンパイラの種類の推定

これらのシグネチャを用いて，パターンマッチングでコンパイラの種類を推定する．まず，リンカのバージョンをシグネチャとしたパターンマッチングを実施し，コンパイラを推定する．次に，特徴的な文字列をシグネチャとしたパターンマッチングを実施してコンパイラを推定し，リンカのバージョンによるパターンマッチングの結果と突合する．これらの結果が一致すればそれを最終的な結果とし，2つの結果に差違があれば，MajorLinkerVersion および MinorLinkerVersion が書き換えられている可能性があるとして，後者の結果を採用する．また，リンカのバージョンとしてあり得ない値が得られた場合も，後者の結果を採用するものとした．

5 最適化レベルの推定

最適化レベルの推定は，機械学習の手法を用いて行う．コンパイラの最適化アルゴリズムを示す．それらのうち，機械語命令列に自明に影響を与えるものに焦点を当て，それに基づいて特徴量を決定する．特徴量を基に，機械学習によって最適化レベルを推定する．

5.1 コンパイラの最適化アルゴリズム

一般的なコンパイラの最適化アルゴリズムについて議論する．コンパイラによる最適化は，(1) 命令の実行回数を減らす (2) より速い命令を用いる (3) 並列度を向上させる，の3つの方法によって実施される．

命令の実行回数を減らす方法には，定数の計算を先に実施する，冗長性を除去する，ループ不変な命令をループ外へ移動させる，といったものがある．特に機械語命令列に影響を与えると考えられるのが，ループ変換である．多重ループの条件を書き換えることでループを一重にするループつぶしや，複数の連続したループを1つのループにまとめるループ融合，複数周分のループを一周にまとめるループ展開などがある．これらはいずれも，ループ制御の命令やその実行回数を減少させることを目的としている．

より速い命令を用いる方法には，特殊な命令を利用する，演算の強さを軽減する，といったものがある．特殊な命令とは，複数の命令の機能を一つにまとめたより高速な命令である．たとえば，加算を行った上でその結果を比較して分岐するという動作を一命令で実施するような命令のことである．演算の強さの軽減とは，演算命令をより高速な演算命令で置き換えるものである．たとえば，べき乗の $x**2$ を乗算の $x*x$ で，乗算の $x*2$ を加算の $x+x$ で，除算の $x/2.0$ を乗算の $x*0.5$ で置換するような方法である．それぞれ，前者の演算命令よりも後者の演算命令の方が一般的に速いため，高速化が見込める．

並列度を向上させる方法には，スーパースカラ方式のプロセッサなどにおいて，命令間の依存関係を軽減することで並列に実行できる命令を増加させるなどの方法がある．これらの手法では，命令の順序を変更させることで並列度を向上させる場合が多い．したがって，命令数や命令の種類は大きく変化しないことが多く，その他の最適化アルゴリズムと比較して，機械語命令列に与える影響は大きくないと考えられる．

5.2 特徴量の決定

コンパイラの最適化アルゴリズムに基づいて，機械学習に用いる特徴量を決定する．本研究では，以下の特徴量を用いる．全命令数，全命令数に対する分岐命令の割合，全命令数に対する特殊命令の割合，べき乗命令に対する乗算命令の割合，乗算命令に対する加算命令の割合，除算命令に対する乗算命令の割合，である．まず，最適化レベルによって，実行ファイル全体の総命令数は大きく変化することは自明である．分岐命令については，ループ変換を行うと，ループつぶしおよびループ融合による分岐の減少や，ループ展開による分岐命令あたりの命令の増加が発生する．また，特殊命令については，通常の命令を特殊命令に置き換える最適化が実施されれば，特殊命令の割合は大きくなることが予測される．さらに，各種演算命令については，演算の強さを軽減する方向で演算命令を置換す

れば，それらの割合には偏りが見られると考えられる．したがって，以上のような特徴量を用いる．

5.3 機械学習による分類を用いた最適化レベルの推定

機械学習によって最適化レベルを推定する．まず，最適化レベルの推定の問題について，詳細を示す．一般に，コンパイラは複数の種類の最適化レベルを持つ．利用されたコンパイラが既知のある実行ファイルが存在したとき，そのコンパイラの最適化レベルの種類がn種類である場合，その実行ファイルはそのn種類の最適化レベルのいずれかに属していると考えることができる．したがって，nクラスのクラス分類問題として捉えることができる．本研究では，教師あり学習の機械学習手法を用いて，分類を行う．最適化レベルが既知の実行ファイルの特徴量を学習し，分類器を生成する．それを用いて，最適化レベルが未知の実行ファイルの特徴量を入力として，最適化レベルを得る．また，機械学習のアルゴリズムは，実験的な手法を用いて選定する．複数のアルゴリズムに対して実際に学習と認識を実行し，それらの結果を解釈し，議論を展開する．それに基づいて，本問題に適した機械学習のアルゴリズムを選出する．本研究では，機械学習のアルゴリズムとして，以下の4つを用いた．k-Nearest Neighbor（以下，k-NN），Artificial Neural Network（以下，ANN），Decision Tree（以下，決定木），Support Vector Machine（以下，SVM）である．

6 実験

6.1 コンパイラの種類の推定

実験環境

2種類の方法によって準備されたマルウェア検体に対して，実験を行った．1つめは，ハニーポットを用いて収集した一般に流通しているマルウェアである．1292検体を収集したものであり，いずれの検体も特有のSHA-1のハッシュ値を保持している．これらは，2009年から2010年にかけて，大学のネットワークに設置された収集環境によって収集されたものである．この収集環境は，サーバ型ハニーポットとクライアント型のハニーポットを含み，インターネット上からの攻撃を受動的に待ち受けるとともに，怪しいWebページのリンクに対して能動的に確認にいく．収集されたマルウェア検体は，実行されずに保存されている．したがって，ダウンローダによる新たなマルウェアのダウンロードなどの，攻撃の過程で追加的に発生するマルウェアは含まれていない．2つめは，CCCDATAsetにファイルのハッシュ値が含まれているマルウェア検体である．本実験では，それらの中から50検体を抽出した．

実験結果

まず，ハニーポットに収集された一般に流通しているマルウェアに対して実験を行った．リンカのバージョンによるシグネチャのみを用いてコンパイラの推定を行ったところ，1292検体中，1073検体を推定することができた．精度は0.83である．コンパイラの種類ごとの推定された検体数を表6.1に示す．さらに，それに加えて，特徴的な文字列によるシグネチャを用いてコンパイラの推定を行った．その結果，表6.1で未識別となっていたもののうち，87検体をMicrosoft Visual C/C++，23検体をDelphiと推定することができた．したがって，最終的に，1292検体中，1183検体について推定することができた．精度はリンカのバージョンによるシグネチャのみを用いた場合から上昇して，0.91となった．

コンパイラ	検体数
Microsoft Visual C/C++ 6.0	419
Microsoft Visual C/C++ 8.0	192
Delphi	160
Microsoft Visual C/C++ 7.1	119
MASM/TASM/FASM	73
Microsoft Visual C/C++ 9.0	63
Borland C/C++	37
Microsoft Visual C/C++ 7.0	10
未識別	173

次に，CCCDATAsetのマルウェア検体に対し

て実験を行った．リンカのバージョンによるシグネチャのみの推定では 50 検体中，45 検体を推定することができた．精度は 0.9 である．コンパイラの種類ごとの推定された検体数を表 6.1 に示す．それに加えて，特徴的な文字列によるシグネチャを用いてコンパイラの推定を行ったところ，未識別のもののうち，4 検体を Microsoft Visual C/C++，1 検体を Delphi と新たに推定することができた．したがって，最終的に，全検体について推定することができた．精度はリンカのバージョンによるシグネチャのみを用いた場合から上昇して，1.0 となった．

コンパイラ	検体数
Microsoft Visual C/C++ 6.0	13
Delphi	13
Microsoft Visual C/C++ 9.0	7
Microsoft Visual C/C++ 8.0	4
Microsoft Visual C/C++ 7.1	3
Borland C/C++	3
Microsoft Visual C/C++ 7.0	1
MASM/TASM/FASM	1
未識別	5

6.2 最適化レベルの推定

実験環境

教師あり機械学習の手法を用いるにあたって，学習データが必要となる．本実験では，最適化レベルをクラス分類問題として推定するため，分類対象のクラスとなる最適化レベルを用いた実行ファイルが必要となる．すなわち，ソースコードを複数の最適化レベルでコンパイルして生成した実行ファイルを必要とする．しかし，学習データとして十分な量のマルウェアのソースコードを入手するのは容易ではない．したがって，本実験では，最適化レベルがマルウェアの機械語命令列に与える影響の特徴が，最適化レベルが悪性でないソフトウェアの機械語命令列に与える影響の特徴と差違がないと仮定した．この仮定にしたがって，悪性でないソフトウェアのソースコードから生成された実行ファイルを学習データとして用いた．学習データを生成するソースコードは，ソフトウェア開発プロジェクトのための共有 Web サービスである，GitHub に存在するものを用いた．GitHub 上の C/C++によるプロジェクトを，トレンドのランキング上位のものから，100 プロジェクト分のソースコードを収集した．ただし，最適化レベルの設定のため，Makefile 内で CFLAGS のマクロを指定しているものを選択した．これらを Windows 上で gcc/g++によってコンパイルして，実行ファイルを生成した．その際に，CFLAGS の値に-O0，-O1，-O2，-O3 の 4 つの最適化レベルをオプションとして与え，1 つのプロジェクトについてそれぞれの最適化レベルを用いた 4 つの実行ファイルを生成した．したがって，分類先となるクラスは 4 クラスあり，それぞれのクラスに対してサンプル数 100 の学習データが存在する．これらの学習データを用いて，k-NN，ANN，決定木，SVM のそれぞれの学習アルゴリズムで分類器を生成した．

2 種類のマルウェア検体に対して分類を行った．1 つめは，10 種類のマルウェアのソースコードを-O0，-O1，-O2，-O3 の 4 つの最適化レベルでそれぞれコンパイルした 40 検体のマルウェア検体である．2 つめは，CCCDATAset にファイルのハッシュ値が含まれているマルウェアのうち，抽出した 10 検体のマルウェア検体である．

実験結果

まず，ソースコードを-O0，-O1，-O2，-O3 の 4 つの最適化レベルでそれぞれコンパイルした 40 検体のマルウェアに対して実験を行った．k-NN，ANN，決定木，SVM のそれぞれによる分類器で分類した際の平均再現率と平均適合率は表 6.2 のようになった．

分類器	平均再現率	平均適合率
k-NN	0.15	0.311
ANN	0.275	0.55
決定木	0.25	0.5
SVM	0.375	0.752

次に，CCCDATAset のマルウェア検体に対して実験を行い，結果として，-O0 が 2 検体，-O1 が 1 検体，-O2 が 4 検体，-O3 が 3 検体となった．

7　考察

本論文での手法によって，実験では，コンパイラの種類をいずれも 90%を超える高い割合で推定することができた．今回の実験では，既にコンパイル済みの検体に対して推定を行った．推定の正しさ，確実性を保証することは難しいが，2 種類のシグネチャを突合し，整合性を確認することによって，推定の誤りを最小限に抑えていると考えることができる．また，実際に一般に流通しているマルウェアに対してもコンパイラの種類の推定を試み，それらについても実際に高い割合での推定が可能であることが分かった．これは，現時点ではマルウェアを作成している攻撃者のほとんどが，コンパイラの種類を特定されることに対して十分な対策をとっていないと考えることができる．実際，リンカのバージョンについては書き換えが可能であり，コンパイラの種類の推定を妨害することも可能であるが，それが行われているマルウェアはごく一部であった．仮に攻撃者が本手法への対策を実施したとしても，特徴的な文字列に基づいたシグネチャによって推定できる可能性は残されている．また，特徴的な文字列を巧妙に書き換えることができたとしても，それによって攻撃者に負担を与えることができると考えられる．

一方，最適化レベルの推定については，現時点では十分な精度を確保できているとは言えない結果となった．SVM による推定は 40%程度の平均再現率を示しているが，その他の機械学習のアルゴリズムにおいてはランダムネスによる分類と同等か，それよりも低くなっている．原因の一つとしては，現在設定している特徴量では十分に各最適化レベルによって生成された実行ファイルの特徴を捉えることができていないことが考えられる．今後はより適した特徴量を模索していく必要があると言える．

8　まとめ

本論文では，マルウェアの実行ファイルに対してコンパイラの種類および最適化レベルを推定する手法を提案した．実験を通して，コンパイラの種類を推定することが可能であることを示した．また，最適化レベルについても，十分な精度を確保しているとは言えないものの，ランダムネスよりも高い割合で推定が可能であることを示した．本研究の成果は，コンパイラの種類や最適化レベルの差異が機械語命令列に基づいたマルウェア対策技術に与える影響を低減するための第一歩となる．今後は，最適化レベルの推定の精度を向上させるとともに，実際に影響を低減するために，コンパイラの種類や最適化レベルにあわせて再最適化を行う手法の確立を目指す．

参考文献

[1] AT4RE. At4re fastscanner. http://www.at4re.com/.

[2] Jibz, Qwerton, snaker, and xineohP. Peid. http://www.softpedia.com/get/Programming/Packers-Crypters-Protectors/PEiD-updated.shtml.

[3] Iwamura M., Itoh M., and Muraoka Y. Towards efficient analysis for malware in the wild. In *IEEE International Conference on Communications (ICC), 2011*, pages pp 1–6, Jun 2011.

[4] McAfee. Mcafee threats report: First quarter 2013. Technical report, McAfee Labs, Jun 2013.

[5] Kathryn S. McKinley, Steve Carr, and Chau-Wen Tseng. Improving data locality with loop transformations. In *ACM Transactions on Programming Languages and Systems (TOPLAS)*, pages pp 424–453, Jul 1996.

[6] RDGMax. Rdg packer detector. http://www.rdgsoft.8k.com/.

[7] M.E. Wolf and M.S. Lam. A loop transformation theory and an algorithm to maximiz parallelism. In *IEEE Transactions on Parallel and Distributed Systems*, pages pp 452–471, Oct 1991.