コンパイラ変更に対して頑強なマルウェア分類手法

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大している．その多くは亜種マルウェアによるものである．これらを効率的に解析し，対策を実施するには，プログラムによってあらかじめマルウェアを種族ごとに分類しておくことが効果的である．その手法の一つとして，機械語命令列の特徴に基づいて分類するものが有効とされる．しかし，この手法はコンパイラや最適化レベルの変更の影響で精度が低下し得る問題を持つ．本研究では，コンパイラおよび最適化レベルを推定する手法を提案する．また，推定結果に基づいて，マルウェア分類におけるコンパイラの影響を削減する手法も提案する．実験を通して，提案手法が様々な既存の分類手法の精度を向上させることを確認した．

Malware Classification Robust against Compiler Modification

Toshinori Usui†　　Kanta Matsuura†

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

Abstract To take effective countermeasures against malware variants, automatic family classification by programs is desirable. Classifying malware based on the features of the machine instruction sequence is one of the effective methods. However, this method has a problem that its precision drops down affected by compiler and optimization level. In this paper, we propose the method to estimate compiler variety and optimization level, as well as the method to mitigate compiler's ill effects on malware classification depends on the estimation results are also proposed. Through the experiments, we confirmed that our proposing method can improve various kinds of existing malware classification methods.

1　はじめに

今日，マルウェアが継続的に高速に生み出されており，すべてのマルウェアを手動で解析することは限界を迎えていると言える．そのため，マルウェアの解析の全部または一部を自動化しようとする研究が多数行われている．新たに出現するマルウェアに着目すると，新しく発見されたマルウェアの多くは，全く新しく作成されたものではない．その多くは，既存のマルウェアに改変や機能の追加を施した亜種である．亜種のマルウェアは，機能的に近いもの同士が多いため，亜種であると判明した場合，新たに解析する必要がなかったり，注力して解析する部分を特定できたり，といった利点が生まれる．また，数理モデルや機械学習などの手法による自動分類で亜種を把握できる場合が多いなど，亜種マルウェアへの対処は，自動化に適していると言える．したがって，自動化によって亜種マルウェアを知ることは，多くのマルウェアが生み出される現状に対して効果的である．

マルウェアの解析には，大きく分けて次の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り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り解析する静的解析である．自動化する際に，動的解析は，実行時間を必要とすることから，静的解析と比較して大きな時間がかかる．そのため，すべての新たに発見されたマルウェアに動的解析を適用することは，現実的ではない．したがって，まずは，プログラムによって自動化された静的解析のように，高速で一度に多くのマルウェアを解析できる手法を用いること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ることが必要である．近年の研究では，次のような特徴を用いることについての研究が行われている．API 呼び出し [9] およびコールグラフ [4]，文字列 [15]，ファイルヘッダ [11]，ファイルヘッダのハッシュ値 [14] などである．一方で，マルウェアの機械語命令列を用いた手法には，問題がある．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生成されたものであっても，利用したコンパイラ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．これが，マルウェア対策技術の精度に影響

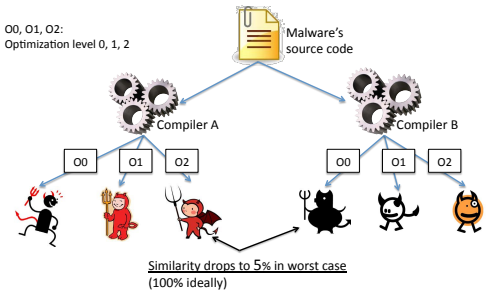

図 1: コンパイラの差異による問題のモデル

を与えている．この問題をモデル化したものを図 1 に示す．一つのマルウェアのソースコードを基にして，様々なコンパイラや最適化レベルを用いて生成されたそれぞれの実行ファイルは，機械語命令列の観点から見ると，異なった特徴を示す．いずれの実行ファイルも，一つのソースコードを基にして生成された，機能が同じマルウェアであるため，理想的には 100%の類似度を示すことが期待される．しかし，例えば，後述する Iwamura ら [3] の類似度算出手法では，最悪の場合でその類似度は 5%程度まで低下していることが実験で確認されている．

この問題によって，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と，精度が低下してしまう可能性がある．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．たとえ亜種や，同種の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が用いられることによって，機械語命令列には大きな差異が生まれる．したがって，この差異に影響を受けて，マルウェアの持つ機械語命令列の特徴を正しく捉えることができず，全く別の種族のマルウェア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，Iwamura らの研究 [3] で実験が行われている．Iwamura 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，実験している．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族の違いよりも，類似度の算出に大きく影響する，と結論づけられている．本来，分類結果にはマルウェアの種族の違いが表れることが目的であるため，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，マルウェア対策技術において，コンパイラの差異による影響を削減し，頑強な精度を維持する手法を提案する．特に，機械語命令列に基づいたマルウェアの分類に着目して，頑強化する手法を提案している．また，それを達成するための要素技術として，実行ファイルの生成に用いられたコンパイラおよび最適化レベルを推定する手法も提案する．

コンパイラおよび最適化レベルを，機械学習の手法を用いて推定する手法を提案する．コンパイラおよび最適化レベルの推定を分類問題として捉え，機械学習を適用した．その際に，本手法では，実行ファイルのうち，コード領域に着目し，オペコードに基づいて特徴抽出の手法を行っている．まず，自然言語処理で用いられる手法を適用し，その有効性を確認した．さらに，それを改善するために，実際にコンパイルした多くのファイルを静的解析し，どの機械語命令がコンパイラや最適化レベルによって大きく変化するかを調べた．それに基づいて，コンパイラ，最適化レベル推定の文脈に合わせた，より優れた特徴量を提案した．

また，コンパイラおよび最適化レベルの推定結果に基づいて，機械語命令列に基づくマルウェア分類システムに，コンパイラ変更に対する頑強性を付与する手法を提案した．この手法は，マルウェアの種族分類に先立って，コンパイラと最適化レベルで事前に分類しておき，同じコンパイラ，最適化レベルで生成されたマルウェア同士で種族分類を行うことで，コンパイラによる影響を削減するものである．

コンパイラおよび最適化レベルを推定する手法を実装し，マルウェア検体に対して実験を実施し，手法の有効性を確認した．さらに，Iwamura らのマルウェア分類システムおよび Rad らのマルウェア分類システムを実装し，コンパイラ変更に対する頑強性を付与する前後での，マルウェア分類の精度を調べた．それらの結果について，本稿で議論している．

本研究の貢献をまとめると，以下の通りである．

- 実行ファイルの機械語命令列の特徴から，コンパイラおよび最適化レベルを推定する手法を提案した．
- コンパイラおよび最適化レベルの推定結果を基に，コンパイラの影響を受けることなくマルウェアを分類する手法を提案した．
- 実験を通して，提案手法の有効性を示した．

2 関連研究

2.1 コンパイラが機械語命令列に与える影響に関する研究

Walenstein らの研究 [12] では，マルウェアの類似度を計測する上での問題についてまとめ，今後の研究するべき問題について述べている．その中で，コンパイラや最適化レベルの違いによって生み出される変化をどう扱うか，という問題を挙げている．

Iwamura らの研究 [3] では，その際に，特筆するべき点として，コンパイラや最適化レベルの違いによって機械語命令列が変化し，類似度の算出結果に影響を与える可能性について，実験を通して議論していることが挙げられる．実験では，マルウェアである sdbot のソースコードを用意し，2 種類のコンパイラ，それぞれ 2 種類の最適化レベルを用いて，4 種類のマルウェアの実行ファイルを生成している．それらに対して，すべての組み合わせでの類似度を算出し，比較している．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，最適化オプションが異なれば，18.46%という低い値になっている場合も見られた．この実験の結果から，コンパイラの種類や最適化レベルの違いは，マルウェアの種族の違いよりも，類似度に対して強く影響を与えると述べている．したがって，コンパイラの差

異は，機械語命令列を用いたマルウェア分類に対して，負の影響を与える可能性が十分にあると言える．

2.2 コンパイラの推定に関する研究

過去の我々の研究 [16] では，シグネチャを用いたパターンマッチングにより，コンパイラを推定する手法を提案した．シグネチャには，PE ヘッダに含まれるリンカのバージョン情報や，実行ファイルに含まれる，コンパイラに特有の文字列を用いた．この手法によって，高い精度でコンパイラを推定可能である．しかし，これらのシグネチャの多くは，プログラムの動作に影響を与えずに書き換えが可能であるという欠点がある．よって，マルウェアのような，作成者ができる限りプログラムに関する情報を隠蔽しようとする敵対的環境においては，容易に推定を回避されてしまうため，十分な手法であるとは言えない．

2.3 機械語命令列に基づいてマルウェアを分類する研究

検知や分類などのマルウェア対策技術における機械語命令列の有用性は，以前から議論されてきている．マルウェアの機械語命令列についての統計的解析は，Weber ら [13]，Li ら [5]，Bilar[1] などによって行われている．これらはいずれも，機械語命令列やその中のオペコードが，マルウェアの検知や分類に有効であることを示している．以下では，それらを特徴としてマルウェアを分類している研究について述べる．

Iwamura らの研究 [3] では，マルウェア間の類似度を算出し，それをデンドログラムとして描画し，分類している．閾値を設定し，デンドログラム上で類似度がその閾値を超えた部分を同じ種族として分類している．このとき，分類対象のマルウェア群のうち，取り得るすべての 2 つのマルウェアの組み合わせで類似度を算出する．類似度の算出では，機械語命令列から最長一致部分列 (LCS) を抽出し，2 マルウェア間での LCS の Jaccard 係数を算出し，類似度として用いている．このとき，LCS の算出を高速化するために，縮約命令を構築し，ビットベクトル化した上で，SSE2 命令を利用した動的計画法での算出を行っている．この縮約命令は，IA-32 命令における，Prefix，オペコード長，Opcode，ModR/M，SIB に当たる情報で構成されている．

Rad らの研究 [7][6] では，マルウェアからオペコードの並びを抽出し，その 1 バイト目に着目して，その出現頻度をヒストグラムにして分類のための特徴として用いている．ここで，オペコードの 1 バイト目は 0x00 から 0xFF の 256 通りの値を取り得るため，そのヒストグラムは 256 のビン数となる．このヒストグラムを用いて，以下のようにマルウェアを種族ごとに分類する手法を提案している．まず，あらかじめ，既知のマルウェアに対して，種族ごとにオペコードの平均出現頻度のヒストグラムを作成しておく．そして，分類対象のマルウェアに対してもヒストグラムを作成する．このヒストグラムと，それぞれの種族のヒストグラムとの間での類似度を Minkowski 距離を用いて算出し，この距離があらかじめ設定した閾値よりも近ければ，その種族に所属するマルウェアとして分類する．

Shabtai らの研究 [10] では，マルウェアのオペコードの N-gram[2] に基づいて分類する手法を提案している．同じくマルウェアのバイト列の N-gram に基づいて分類する研究を先行研究としている．バイト列の N-gram による表現よりも，オペコードの N-gram による表現の方が，より意味のある特徴となると主張している．N-gram から TF-IDF[8] を算出し，それに基づいて分類している．

以上のような研究は，いずれもマルウェアの持つ機械語命令列にその精度が強く依存している．しかし，どのコンパイラや最適化レベルを用いて生成されたかによって，機械語命令列は大きく異なってくる．したがって，これらのマルウェア分類手法は，本研究の適用対象となると言える．

3 コンパイラおよび最適化レベルの推定手法

3.1 コンパイラの推定

まず，実行ファイル中のどの情報を用いて推定するかについて述べる．実行ファイルのどの領域にコンパイラの影響が強く現れるかを考える．実行ファイルには，実行コードの含まれるコード領域 (.text)，初期値を持つグローバル変数が含まれるデータ領域 (.data，.rdata)，インポート関数とエクスポート関数の情報がそれぞれ含まれるインポート領域 (.idata) とエクスポート領域 (.edata)，例外の情報が含まれるエクセプション領域 (.pdata，.xdata)，再配置情報が含まれるリロケーション領域 (.reloc)，リソース情報が含まれるリソース領域 (.rsrc) などがある．これらの中で，.data，.rdata，.idata，.edata，.pdata，.xdata，.rsrc などの領域に含まれる情報は，コンパイラよりも，生成元となるソースコードなどに強く依存する．そのため，コンパイラや最適化レベルの推定に用いる情報としては有効ではない．一方，コード領域である.text に含まれる実行コード，すなわち機械語の命令列などは，コンパイラの実装によって変化する部分が大きい．したがって，コンパイラや最適化レベルの推定には，この領域の情報を利用するのが有効であると考えられる．

次に，実行ファイル中の情報を用いて，どのように推定を行うかについて述べる．コンパイラの推定は，分類問題と考えることができる．したがって，本研究では，機械学習の手法を用いて分類を行うことで，コンパイラの推定を行う．機械学習をコンパイラの推定に適用する上で重要なこととして，実行ファイル中の機械語命令列のどのような特徴に基づいて推定するか，ということが挙げられる．すなわち，機械学習における特徴選択である．機械語命令は，オペコードとオペランドに分けられる．このうち，オペランドには即値やアドレスなどが含まれる．これについて，Iwamura らは，アドレスは環境によって変化する可能性があり，即値にもアドレスが含まれる場合がある [3] ことから，分類のための情報として用いていない．また，Rad らはオペランド部分は，たとえばメタモーフィックエンジンなどの難読化手法によって改変されうる [6] ため，やはり分類のための情報として用いないようにしている．そのため，提案手法でも，オペランド部分の利用はせず，オペコード部分に着目して，推定を行う．ここで，考えられる特徴量として，以下のようなものがある．1 つめは，オペコードの出現頻度である．コンパイラご

とに用いるオペコードの種類や用い方の傾向には異なりがあるため，それぞれのオペコードの出現頻度は有効な情報となる．自然言語処理の分野では，語の出現頻度を Bag-of-Words と呼称するため，本稿では，オペコードの出現頻度を，Bag-of-Opcodes と呼称する．2 つめは，オペコードの N-gram[2] である．N-gram とは，N 個の連続した要素をまとまりとして抽出したものであり，そのまとまりごとの出現頻度を用いる手法である．オペコードの出現頻度のみでは，オペコード同士の前後関係，すなわち，出現順序を踏まえることができない．しかし，N-gram であれば，特徴量にオペコードの出現順序の情報を組み込むことができる．ここまでで，Bag-of-Words, N-gram による特徴抽出について述べた．しかし，これらの単純な手法のみでは，効率よくコンパイラを推定することは難しい．コンパイラを推定する上で効果的な特徴を選択的に組み込むことができていないためである．したがって，効率のよい推定のためには，コンパイラの実装の違いによって，出力される機械語命令列がどのように異なるかということを明らかにする必要がある．コンパイラの設計と実装によって，出力される機械語命令列がどのように異なるかを把握するために，さまざまなプログラムのソースコードを複数のコンパイラでコンパイルし，静的解析した．その結果として，以下のような点で機械語命令列に差異が生じることが分かった．

サブルーチン呼び出しの引数の指定方法　サブルーチン呼び出しの際に，cdecl，stdcall などの呼び出し規約のサブルーチンは，引数の受け渡しにスタックを用いる．このとき，コンパイラの実装によって，スタック上にどのように引数を配置するかが異なる．具体的には，push 命令を用いて配置する方法と，mov 命令を用いて配置する方法の 2 通りが存在する．したがって，どちらの方法を採用しているコンパイラかによって，push 命令と，mov 命令および pop 命令の数に偏りが生まれる．これは，サブルーチン呼び出しの数に応じて偏りが増し，コンパイラを指し示す大きな特徴の一つである．

条件分岐命令の選択　分岐命令には様々な種類の条件を利用するものが用意されている．具体的には，ここで，ソースコードに記述された条件を，オブジェクトコード生成時にどのように解釈するかによって，用いる条件分岐命令も異なってくる．たとえば，実際，コンパイラによって，どの条件分岐命令を積極的に利用するかが異なることが分析により分かった．多くのプログラムが条件分岐を行うため，これもコンパイラを推定する上で重要な特徴の一つであると考えられる．

特殊命令の使用　コンパイラによっては，生成したコード内で特殊な命令を用いる場合がある．ここでの特殊な命令とは，ビットシフト命令，FPU 命令，複合命令である．いずれも，高速化に用いられることがある命令である．これらの命令は，積極的に用いるコンパイラとほとんど用いないコンパイラに分かれ，その間で使用頻度が大きく異なる．したがって，コンパイラを推定する重要な手掛かりとなると言える．

反対に，上記以外の命令については，コンパイラの違いに依存して変化していることが明確な命令を見出すことが難しかった．コンパイラの違いにによる変化が明確でない命令を用いると，ノイズとなる可能性が生まれる．そのため，これらの変化の大きい命令に注目して，特徴ベクトルを構成することを提案する．具体的には，Bag-of-Opcodes から，mov, push，pop，条件分岐命令，特殊命令の出現頻度のみを抽出して特徴ベクトルとした．Bag-of-Opcodes, N-gram との比較を，実験によって示した．

3.2　最適化レベルの推定

最適化レベルの推定手法は，基盤となる部分は前節のコンパイラ推定の手法と同様である．マルウェアのコード領域に存在する機械語命令列を情報として用いる．最適化レベルの推定を分類問題として捉え，機械学習を用いて推定する．その際の特徴量は，命令の中でも，オペコードに焦点を当てて設計する．また，特徴量の基盤として，Bag-of-Opcodes や N-gram が適用可能なことも，同じである．一方で，Bag-of-Opcodes や N-gram などの手法のみでは，効率よく最適化レベルを推定することが難しいと思われるのも同様である．最適化レベルを推定する上で効果的な特徴を，選択的に組み込むことができていないためである．したがって，効率のよい推定のためには，最適化レベルの違いによって，出力される機械語命令列がどのように異なるかということを明らかにする必要がある．

最適化レベルによって，出力される機械語命令列がどのように異なるかを把握するために，様々なプログラムのソースコードを，複数のコンパイラ，最適化レベルでコンパイルし，静的解析した．結果として，コンパイラの違いによる変化のような，最適化レベルに依存して明確に変化している命令はあまり見られなかった．また，コンパイラやコンパイル対象のプログラムにより，最適化レベルが近いもの同士の間では，あまり命令列に変化が見られない場合も存在した．一方で，特殊命令の数には違いが表れやすいということが，解析を通して見てとれた．そのため，特殊命令は，出現頻度の値は他の命令と比較して大きくなりにくいものの，重要な特徴であると言える．したがって，これに対して重み付けをするために，TF-IDF[8] を用いることを提案する．具体的には，Bag-of-Opcodes に対して，TF-IDF による重み付けを行ったものを，特徴ベクトルとする．

4　マルウェア分類の頑強化

4.1　頑強性の定義

マルウェア分類システムとは，おもに，多数のマルウェアを，分類結果に種族が現れるようにして分類するものである．これによって，亜種のマルウェア同士がまとめられることにより，未知のマルウェアがどの種族に属すかを知ることができる．その結果，未知のマルウェアが既知の種族に属するものであるものであることが分かれば，新たに解析する作業が不要であったり，どの部分に注力して解析するべきかを把握できたりといった恩恵を得られる．機械語命令列を特徴としてこの分類を実施する場合，Iwamura ら [3] や Walenstein ら [12] が主張するように，コンパイラや最適化レベルの違いによって生成される命令列が異なり，同種や亜種のマルウェアであっても正しく分類できない場合が生じる．すなわ

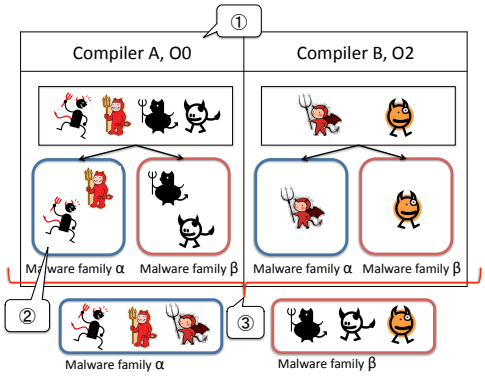

図 2: コンパイラ変更に対する頑強性を持ったマルウェア分類

ち，分類結果が，コンパイラによる悪影響を受けるのである．本研究でのコンパイラ変更に対する頑強性とは，このコンパイラによる悪影響を削減することにより，異なるコンパイラや最適化レベルによってコンパイルされた亜種マルウェア同士であっても，正しく同じ種族に属するものとして分類できる性質である．

4.2 頑強性の付与

機械語命令列に基づくマルウェア分類システムに，コンパイラ変更に対する頑強性を付与する手法について述べる．頑強性の付与は，通常のマルウェア分類の前と後に，コンパイラによる分類への影響を削減する処理を挿入することによって実現される．そのため，頑強性を付与されたマルウェア分類システムによる種族分類は，以下の 3 つの手順で行われる．また，これらの手順を図 2 に示す．

1 つめの手順は，コンパイラおよび最適化レベルによる事前分類である．まず，分類対象となるマルウェア群に対して，コンパイラおよび最適化レベルを推定する．推定結果を用いて，同じコンパイラと最適化レベルで生成された実行ファイルが同じクラスとなるように，分類する．

2 つめの手順は，マルウェア分類システムによる種族分類である．頑強性の付与対象となるマルウェア分類システムのアルゴリズムに基づいて，分類対象のマルウェア群を，その種族ごとに分類する．その際に，種族分類は，1 つめの手順で事前分類したクラスごとに実施する．これによって，同じコンパイラ，最適化レベルによって生成されたマルウェア同士で種族分類を実施できる．そのため，種族分類にコンパイラや最適化レベルの差異が引き起こす影響を削減できる．

3 つめの手順は，分類結果の統合である．2 つめの手順において，1 つめの手順で事前分類したクラスごとに種族分類の結果が得られている．これを統合することで，最終的な種族分類の結果とする．結果の統合には，それぞれのコンパイラおよび最適化レベルのクラスごとに，種族分類の結果のクラスがどの種族であるかを特定し，同じ種族のクラス同士をコンパイラ，最適化レベルをまたがって統合することで実施する．種族分類の結果がどの種族であるかが不明な場合は，たとえば，代表的ないくつかのマルウェアを解析することで，種族を特定する．

以上の 3 つの手順で，マルウェア分類システムにコンパイラ変更に対する頑強性を付与した上で種族分類を行う．この手順からも分かる通り，この提案手法はマルウェア分類手法の前後にコンパイラによる影響を削減する手順を追加することで頑強性を付与している．そのため，マルウェアの種族分類をする部分には変更を加える必要がない．したがって，様々な種類の既存の機械語命令列に基づくマルウェア分類手法にそれぞれ適用可能であると考えられる．

4.3 適用要件

先に述べた，コンパイラ変更に対する頑強性の付与手法が適用可能となる要件を記述する．適用対象は，マルウェアの分類システムである．また，コンパイラや最適化レベルの変更の影響を強く受けるのは，機械語命令列を特徴として分類に用いている場合である．そのため，特に機械語命令列に基づいて分類を行うマルウェア分類システムとなる．ただし，前述の通り，提案手法は特定のマルウェア分類の手法に依存するものではないため，コンパイラの影響を受けやすい，機械語命令列を用いた分類システムであれば，システムを選ばず効果を発揮することが期待できる．また，分類対象のマルウェアについての要件を，以下に述べる．コンパイラや最適化レベルの推定の際に，ランタイムパッカー（以下，パッカー）や，クリプターなどによって機械語命令列が圧縮，暗号化されていると，正しく推定できない可能性が高い．したがって，アンパック，復号などの処理によってオリジナルコードが得られるものである必要がある．

5 実験と評価

5.1 コンパイラおよび最適化レベルの推定に関する実験

5.1.1 実験データの準備

提案手法では，あらかじめ学習によって分類モデルを構築しておくために，一定数の学習データが必要となる．学習データには，分類結果となるクラスが含まれていなければならない．そのため，本実験では，生成に用いたコンパイラおよび最適化レベルが既知の実行ファイルが必要となる．しかし，コンパイラの種類が既知のマルウェアはほとんど存在しない．また，マルウェアのソースコードを収集して自分でコンパイルすることを考えた場合でも，学習データとして十分な量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，コンパイラの出力する実行ファイルに現れる，コンパイラや最適化レベルを指し示す特徴が，良性ソフトウェアとマルウェアの間で大きな差異がないと仮定した．この仮定に基づいて，良性の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，分類モデルを構築するための学習データとして用いた．学習データの実行ファイル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でトレンドのラン

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．これらを実験に用いるコンパイラ，最適化レベルによってそれぞれコンパイルして，実行ファイルを生成した．今回の実験では 3 種類のコンパイラを使用しており，4 種類の最適化レベルを持つコンパイラが 2 つ，3 種類の最適化レベルを持つコンパイラが 1 つであるため，サンプル数 1100 の学習データが存在する

5.1.2 実験の概要

提案しているコンパイラおよび最適化レベルの推定手法に関して，2 つの実験を行った．それぞれの実験では，学習および推定の対象となるデータを，以下の 3 つのコンパイラによって生成された実行ファイルとする．Microsoft Visual C++ 2010（以下，MSVC++），MinGW-gcc 4.7.2（以下，MinGW），Borland C++ Compiler 5.5（以下，Borland）である．実験データとして収集したソースコードを，これらの 3 つのコンパイラによってコンパイルしたものを用いる．また，それぞれのコンパイラでコンパイルする際には，それぞれ複数の最適化レベルでコンパイルする．コンパイラごとの最適化レベルを，表 5.1.2 に示す．

表 1: 各コンパイラに対して用いる最適化レベル

コンパイラ	最適化レベル	数
MSVC++	Od，O1，O2，Ox	4
MinGW	O0，O1，O2，O3	4
Borland	Od，O1，O2	3

これらに対して，提案手法を実装したシステムを用いて，それぞれ正しく推定できるかを実験する．

1 つめは，生成に用いられたコンパイラが既知である良性ソフトウェアに対して，コンパイラの推定を試みるものである．これにより，提案手法によるコンパイラの推定の精度を得ることを目的とする．精度を計測するためには，多くのデータを用いた実験が必要となる．しかし，5.1.1 節に述べた通り，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，本研究では，5.1.1 節での仮定に基づいて，良性のソフトウェアの実行ファイルを基にして，コンパイラの種類推定の精度を計測するものとした．計測の際には，5.1.1 節で挙げた 1100 件の学習データに対して，5 分割交差検定を実施する

2 つめは，生成に用いられた最適化レベルが既知である良性ソフトウェアに対して，最適化レベルの推定を試みるものである．これにより，提案手法による最適化レベルの推定の精度を得ることを目的とする．

いずれの実験においても，特徴量に，Bag-of-Opcodes，N-gram，提案手法の特徴量の 3 つを用いて推定し，比較を行った．なお，N-gram については，一般に有効性が高いとされる，2-gram と 3-gram を用いた．また，学習アルゴリズムには，SupportVectorMachine (SVM)，RandomForest (RF)，Boosted NaiveBayes (boost+NB)，Boosted DecisionTree (boost+DT) の 4 つを用いて推定し，それぞれ比較を行った．これらの学習アルゴリズムは，高次元のデータを扱いやすく，様々な分類問題に対して用いられるものから選択した．

5.1.3 実験結果と考察

1 つめの実験である，実行ファイルの生成に用いられたコンパイラが既知の良性ソフトウェアに対する，コンパイラの推定の結果を示す．表 2 に，推定の結果から精度を計算したものを示す．ここで，精度とは，正しく推定できたデータ数を全データ数で除算したものである．

結果から，提案手法を用いた RandomForest により，最も優れた精度が得られることが分かる．

まず，特徴量について詳しく議論する．提案手法を用いることで，他の特徴抽出の手法と比較して，良い結果が得られている．この要因を分析するために，各コンパイラで生成された実行ファイルから抽出したデータを比較した．MinGW によってコンパイルされた実行ファイルは，push，pop 命令による引数の受け渡しの代わりに，可能な限り mov 命令を用いていた．そのため，push，pop 命令を用いる他の 2 つのコンパイラによる実行ファイルと比較して，それらの割合は有意に異なっていた．それにより，提案した手法は，この偏りを基に，MinGW とその他のコンパイラが高い精度で分離できたと考えられる．また，MSVC++と Borland の間では，用いられる特殊命令の数が異なっていた．Borland は，MSVC++と比較して，より積極的に特殊命令を用いている．さらに，用いられる条件分岐命令の種類の傾向も異なりがあった．これによって，MSVC++と Borland を分離することも可能である．以上の点から，提案手法によって，重要な特徴を選択的に捉えられたことで，ノイズを乗せることなく，他の特徴抽出と比較して，高い精度が得られたと考えられる．また，Bag-of-Words と比較して 2-gram，3-gram が良い結果を示しているのは，オペコードの連なりを特徴として捉えられているためと考えられる．3-gram は 2-gram より若干高い精度を示しているが，3-gram は 2-gram よりも特徴ベクトルの次元数が非常に大きくなる弊害もある．これにより，学習にかかる時間が増大するため，実用を考えると，2-gram の方が有用性が高いと考えられる．また，学習アルゴリズムによっては，次元数が大きくなると，学習に必要な学習データが急激に増加する「次元の呪い」の影響を受ける可能性もある．学習アルゴリズムとしては，SVM よりも，集団学習であるその他の 3 つのアルゴリズムの方が概ね良い結果を示している．この結果から，実行ファイルの生成に用いられたコンパイラは，高い精度で推定可能であると言える．

表 2: コンパイラ推定の精度

	Bag-of-Opcodes	2-gram	3-gram	提案手法
SVM	0.76	0.82	0.83	0.86
RF	0.77	0.85	0.88	0.91
boost+NB	0.73	0.83	0.85	0.87
boost+DT	0.76	0.83	0.87	0.89

次に，2 つめの実験である，生成に用いられた最適化レベルが既知である良性ソフトウェアに対する，最適化レベルの推定の結果を示す．表 3 に，推定の結果から精度を計算したものを示す．

結果から，RandomForest に提案手法を用いたものと 3-gram を用いたものが同程度の精度で，実験に用いた組み合わせの中では優れた精度が得られることが分かった．

まず，特徴量について考察する．提案手法で良い精度が得られたのは，最適化レベルの差異によって

現れる特殊命令の出現頻度の違いをうまく捉えることができているためと考えられる．一方，2-gram や 3-gram は，やはりオペコードの連なりを特徴として捉えられる分，Bag-of-Opcodes と比較すると高い精度が出ている．単純に精度で見た場合，3-gram が提案手法をわずかに上回る．その一方で，3-gram は提案手法と比較して特徴ベクトルの次元数が非常に大きいため，先に挙げたような，学習時間の問題や次元の呪いの問題が発生し得る．したがって，これらの問題がない提案手法は，特徴抽出の手法としては，十分に有用なものだと考えられる．また，学習アルゴリズムとしては，やはり SVM よりも，集団学習のアルゴリズムの方が良い結果を示している．このことから，オペコードを基にコンパイラや最適化レベルを推定する際には，集団学習を用いた場合の方が良い結果が出やすい可能性が考えられる．

また，コンパイラの推定と比較すると，精度が低めになっている．これは，コンパイラや最適化レベル，コンパイルする対象のプログラムによって，最適化レベルが異なっても，出力する機械語命令列に大きな差異が生じない場合があることが一因となっている．たとえば，MinGW において，あるプログラムを O2 の最適化レベルを用いてコンパイルした場合と O3 の場合とでは，出力された実行ファイルの機械語命令列にはほとんど違いが見られなかった．このような実行ファイルの最適化レベルを推定する場合，誤推定してしまうことも考えられる．しかし，機械語命令列にほとんど差異がないのであれば，マルウェア分類に与える影響も少ないと考えられる．したがって，実際に頑強性を付与する際には，本実験での精度以上の結果が期待できる．

表 3: 最適化レベル推定の精度

	Bag-of-Opcodes	2-gram	3-gram	提案手法
SVM	0.58	0.67	0.68	0.67
RF	0.61	0.69	0.71	0.74
boost+NB	0.59	0.68	0.71	0.72
boost+DT	0.60	0.69	0.70	0.71

5.2 コンパイラ変更に対する頑強性付与に関する実験

5.2.1 実験データの準備

提案手法を用いてコンパイラ変更に対する頑強性を付与するためには，コンパイラおよび最適化レベルの推定が必要となる．そのため，学習データとして，先に用いた良性ソフトウェアの 1100 の実行ファイルを用いる．また，実際にマルウェアの種族分類をする対象の要件として，亜種の関係にあるマルウェアが一定数存在することが挙げられる．したがって，CCC DATAset 2013 を採用した．実験の結果を明確にするために，Conficker，RAHack，sdbot に属するマルウェアのみを抽出した．抽出には，McAfee による検知名に上述の種族名が含まれているかで判断した．シグネチャを用いたコンパイラ推定 [16] により，それぞれの種族において，複数の異なるコンパイラで生成された検体が存在することを確認した．これにより，コンパイラの差異による影響が発生することが見込まれる．これが，提案手法によって変化することで，分類精度にどのように影響を与えるかを確認する．

5.2.2 実験の概要

提案手法を用いて，既存のマルウェア分類システムに対し，コンパイラ変更に対する頑強性を付与する実験を行った．既存のマルウェア分類システムとして，Iwamura らのシステム [3] および Rad らのシステム [6] を実装した．はじめに，提案手法を適用しない環境下で，これらのシステムを用いて，マルウェアの種族分類を行った．次に，提案手法を適用した状態で，これらのシステムを用いて，改めてマルウェアの種族分類を行った．これらの 2 つの結果を比較することで，頑強性を付与する前後で，実験データに対するマルウェア分類の精度がどのように変化するかを確認する．この実験によって，マルウェア分類の際に，コンパイラ変更に対する頑強性を付与することの有用性を評価する．なお，分類精度の算出については，McAfee によるマルウェアの検知名と照らし合わせ，一致したものの割合で行った．

本実験における，コンパイラおよび最適化レベルの推定について記述する．推定のための学習データとして，先の実験で用いた 1100 件の良性ソフトウェアを用いた．この際，学習データが MSVC++，Borland，MinGW で構成されることから，推定結果もこの 3 種類のコンパイラのみに限定される．しかし，シグネチャを用いたコンパイラ推定による調査から，9 割以上のマルウェアがこの 3 種類のいずれかによって生成されていると推定されるため，大きな問題とはならないと考えられる．

次に，マルウェア分類システムの運用について述べる．Rad らのシステムにおいては，既知のマルウェアの学習に基づいた分類となるため，5 分割交差検定を用いた．Iwamura ら，Rad らの両システムにおいて必要となる閾値は，いずれも，実験に用いるデータを最も正しく分類できた値を採用した．

5.2.3 実験結果と考察

表 5.2.3 に，提案手法を用いてコンパイラに対する頑強性を付与する前後での，それぞれのマルウェア分類システムの精度を示す．いずれのシステムにおいても，頑強性の付与による精度向上が見られた．Iwamura らのシステムにおける精度向上が，Rad らのシステムにおける精度向上よりも大きいのは，Iwamura らのシステムが特徴として用いている LCS が，Rad らのシステムが特徴として用いているオペコードのヒストグラムよりも，コンパイラの影響を受けやすいためと考えられる．このコンパイラの影響を削減できたことにより，精度が向上しているものと考えられる．また，Iwamura らのシステムでは，精度のほかに，亜種に共通する機能を抽出することにも焦点を当てている．しかし，コンパイラや最適化レベルの差異によって，本来的には多くの共通機能を持つマルウェアを，うまく抽出できない可能性がある．一方，同じコンパイラ，最適化レベルのもの同士であれば，コンパイラの影響を受けないため，問題なく機能を抽出できる．したがって，精度以外にも，提案手法の寄与する部分は存在するのではないかと考えられる．

本実験においては，データセットのマルウェアを用いているため，コンパイラおよび最適化レベルの推定の正しさを示すことは難しい．また，前述のコンパイラや最適化レベルの推定をする実験において一定の推定誤りを避けることができていないため，本実験においても，一定の推定誤りは起こっている

と考えられる．それにも関らず，一定の精度が向上していることから，現時点でのコンパイラおよび最適化レベルの推定精度で，十分にコンパイラ変更に対する頑強性を付与できると考えられる．一方で，コンパイラ，最適化レベルの推定誤りが増えれば，精度向上の鈍化に繋がる．したがって，コンパイラや最適化レベルの推定精度を向上させることは，今後の継続的な課題であると言える．

表 4: 頑強性付与の前後での分類精度の比較

	頑強性付与前	頑強性付与後
Iwamura らのシステム	0.82	0.88
Rad らのシステム	0.60	0.63

6 まとめ

本稿では，機械語命令列に基づいたマルウェア分類手法に対して，コンパイラ変更に対する頑強性を付与する手法を提案した．コンパイラおよび最適化レベルが同じもの同士を分類によってまとめ，その分類したクラスの中でマルウェアの種族分類をすることで，コンパイラの影響を受けずにマルウェアの種族分類を実施する．また，そのために必要な要素技術として，コンパイラおよび最適化レベルを，機械学習を用いて推定する手法を提案した．その際に，効果的に推定するために，コンパイラ推定において有効な特徴抽出の手法について述べた．これらの提案手法に対して実験を行い，その精度と有効性を示した．まず，良性ソフトウェアに対するコンパイラおよび最適化レベルの推定実験により，ランダム選択と比較して有意に高い精度で推定可能なことを示した．また，CCC DATAset2013 を用いた実験では，提案手法によって既存の 2 つのマルウェア分類システムの精度を向上させることに成功している．これにより，要素技術であるコンパイラや最適化レベルの推定も，コンパイラ変更に対する頑強性の付与に可能な水準まで高めることができていると考えられる．また，マルウェアの亜種間に共通する機能の抽出など，種族分類の精度以外にも本研究が寄与する部分があることが期待される．今後は，コンパイラおよび最適化レベルの推定の精度を高めることにより，マルウェアの種族分類の精度をより向上させることを目指す．

7 謝辞

本研究の一部は JSPS 科研費 25280045 の助成を受けた．

参考文献

[1] Daniel Bilar. Opcodes as predictor for malware. *International Journal of Electronic Security and Digital Forensics*, 1(2):156–168, 2007.

[2] Peter F Brown, Peter V Desouza, Robert L Mercer, Vincent J Della Pietra, and Jenifer C Lai. Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–479, 1992.

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

[4] Joris Kinable and Orestis Kostakis. Malware classification based on call graph clustering. *Journal in computer virology*, 7(4):233–245, 2011.

[5] Wei-Jen Li, Ke Wang, S.J. Stolfo, and B. Herzog. Fileprints: identifying file types by n-gram analysis. In *Information Assurance Workshop, 2005. IAW '05. Proceedings from the Sixth Annual IEEE SMC*, pages 64–71, June 2005.

[6] Babak Bashari Rad, Maslin Masrom, and Suahimi Ibrahim. Opcodes histogram for classifying metamorphic portable executables malware. In *e-Learning and e-Technologies in Education (ICEEE), 2012 International Conference on*, pages 209–213. IEEE, 2012.

[7] Babak Bashari Rad, Maslin Masrom, Suhaimi Ibrahim, and Subariah Ibrahim. Morphed virus family classification based on opcodes statistical feature using decision tree. In *Informatics Engineering and Information Science*, pages 123–131. Springer, 2011.

[8] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

[9] Ashkan Sami, Babak Yadegari, Hossein Rahimi, Naser Peiravian, Sattar Hashemi, and Ali Hamze. Malware detection based on mining api calls. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, pages 1020–1025. ACM, 2010.

[10] Asaf Shabtai, Robert Moskovitch, Clint Feher, Shlomi Dolev, and Yuval Elovici. Detecting unknown malicious code by applying classification techniques on opcode patterns. *Security Informatics*, 1(1):1–22, 2012.

[11] M Zubair Shafiq, S Momina Tabish, Fauzan Mirza, and Muddassar Farooq. Pe-miner: Mining structural information to detect malicious executables in realtime. In *Recent Advances in Intrusion Detection*, pages 121–141. Springer, 2009.

[12] Andrew Walenstein and Arun Lakhotia. The software similarity problem in malware analysis. In Rainer Koschke, Ettore Merlo, and Andrew Walenstein, editors, *Duplication, Redundancy, and Similarity in Software*, number 06301 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2007. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

[13] Michael Weber, Matthew Schmid, Michael Schatz, and David Geyer. A toolkit for detecting and analyzing malicious software. In *Computer Security Applications Conference, 2002. Proceedings. 18th Annual*, pages 423–431. IEEE, 2002.

[14] Georg Wicherski. pehash: A novel approach to fast malware clustering. In *2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2009.

[15] Yanfang Ye, Lifei Chen, Dingding Wang, Tao Li, Qingshan Jiang, and Min Zhao. Sbmds: an interpretable string based malware detection system using svm ensemble with bagging. *Journal in computer virology*, 5(4):283–293, 2009.

[16] 碓井 利宣 and 松浦幹太. 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　よび最適化レベルの推定手法. In **マルウェア対策人材育成ワークショップ** *2013* **論文集**, pages 885–892, Oct 2013.