マルウェアのコードの類似度を用いた分類手法に関する一考察

東 結香 †‡　　中津留 勇 ‡　　猪俣 敦夫 †　　砂原秀樹*　　藤川 和利 †

† 奈良先端科学技術大学院大学情報科学研究科
630-0192 奈良県生駒市高山町 8916-5
yuka-h@is.naist.jp
{atsuo, fujikawa} @itc.naist.jp

‡ 株式会社ラック
102-0093 東京都千代田区平河町 2-16-1
you.nakatsuru@lac.co.jp

*慶應義塾大学大学院メディアデザイン研究科
223-8526 横浜市港北区日吉 4-1-1

あらまし 近年、様々なマルウェアが攻撃において使用されており、マルウェアの挙動を正確に把握するための手法を確立することは急務である。しかし、詳細な挙動や機能を把握するのに適した静的解析は時間とコストだけでなく経験や技術も必要であり、膨大にマルウェアが発生する状況においては困難である。そこで本論文では、機械的にマルウェアの挙動を抽出するためコードの類似度を用いた分類手法を提案する。マルウェアの類似度をコード全体から導出する手法と関数のコードの類似度を用いて求める手法を比較した後、関数のコードと類似度の関係ついても分析し、考察を与える。そして、これらの結果よりコードの類似度を算出する際の問題点を整理する。

Consideration on Malware Classification by the Function based on the Malware's Code

Yuka Higashi†　　You Nakatsuru‡　　Atsuo Inomata†　　Hideki Sunahara*
Kazutoshi Fujikawa †

†Graduate School of Information Science, Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, NARA 630-0192 JAPAN
yuka-h@is.naist.jp, {atsuo, fujikawa} @itc.naist.jp
‡Little eArth Corporation Co., Ltd
6F Hirakawacho Mori Tower 2-16-1 Hirakawa-cho, Chiyoda-ku Tokyo 102-0093 Japan
you.nakatsuru@lac.co.jp
*Graduate School of Media Design, Keio University
4-1-1 Hiyoshi Kohoku-ku Yokohama-city Kanagawa, Japan 223-8526

Abstract Recently various kinds of security incidents occur and what is more the majority of attacks in such the incidents have been used Malware, there is an urgent need to analysis and measures against for unknown future Malware. Some of the static analysis of malware are suitable for understanding their behavior, however it needs more skills and more times. In this paper, we compare a method of calculation from the function with method of complete Malware code. Furtheremore we explore the relationship between function code and degree of its similarity.

1　はじめに

インターネットや計算機の急激な普及に伴い、情報セキュリティインシデントの発生件数は増加の一途を辿っている。これらのインシデントで使用された攻撃は、最終的にマルウェアをユーザにインストールさせるものが多い [1]。このような攻撃で使用されるマルウェアの振る舞いはユーザが感染しても気がつかないことが多く、知らぬ間に加害者になる可能性もある。そのため、可能な限り感染を事前に食い止める必要がある。一方、マルウェアを作成する攻撃者らは集団や組織でマルウェア作成を行っているだけでなく、マルウェア作成の自動化ツールを使用して大量の亜種を作り出すことが可能となっており、新しいマルウェアは 1.5 秒以内に 1 種発生しているのが現状である [2]。マルウェアの持つ機能や発現する挙動を詳細に把握するには、一般に静的解析を用いる。しかし、静的解析には経験や知識を持った人材が必要であり、時間がかかるため、感染を未然に防ぐ事は難しい。

また、現在のマルウェアは、プログラムのハッシュ値や、感染方法等を基にアンチウィルスベンダによって識別・命名・対応されている。しかし、亜種生成の高速化・巧妙化により、解析にかかるコストが増加し、全ての種類への迅速な対応が困難となっている。このように大量のマルウェアに対応するためには、今後これらの識別に加えマルウェアの持つ機能に着目した識別が必要であり、その識別を高速に行う必要があると考えられる。

そこで、マルウェアのコードを根拠として分類し、その分類結果に機能の共通点を持たせる事を目的とする。本論文ではマルウェアの関数に着目した分類を提案する。また、マルウェアの類似度をコード全体から導出する手法と関数のコードの類似度を用いて求める手法を比較、及び類似度と解析結果の関係についても述べる。

2　関連研究

マルウェアのコードの類似度を根拠とした分類に関する研究について、N-gram と LCS を用いた手法について述べる。

N-gram とは一般に連続する N 個の要素がそれぞれ何回出現したかを示す。Enamul らは、発生系統的な分類をこの N-gram(N-perm) を用いて行っている [3]。マルウェアを逆アセンブルした結果を用い、N 個のオペコードの出現回数を特徴ベクトルとし、TF-IDF で重み付けを行っている。また、N-gram ではオペコードの順序は保存されており、順序が違うと違う物と認識され出現回数にカウントされない。そこで、Enamul らは加えて N-perm という順序を考慮しない特徴抽出の方法も提案を行った。

LCS とは、最長共通部分列のことを指す。岩村らはアンパッキング、逆アセンブル、類似度算出の 3 つのモジュールからなるマルウェア自動分類システムを開発し、その要となる類似度算出の部分で LCS を特徴量として使用している [4]。LCS を算出するためには直前の結果を保存しておかなければならず、計算コストが高くなる。岩村らの研究では、マルウェアを逆アセンブルしたコードを、元のデータのもつ情報が失われないように加工し、サイズを削減した縮約命令のモデルを提案した上で、LCS の適用が行われている。そして、各検体の縮約命令列を集合とし、LCS を両検体の共通部分と見なしたときの Jaccard 係数を類似度として使用している。

3　提案と検証

前述の関連研究はコードに着目し、検体のコード全体の類似度を比較する事により、コードの発生系統的に分類が可能だと言う事を示した。また、岩村らの研究では既知のマルウェアとの差分を求め解析をスムーズに行うことを可能にした。しかし、機能や挙動に関する分類に関しては課題が残る。ここでは、マルウェアの機能に着目した分類手法を提案し、本論文で行う検証について述べる。

3.1　検体

本論文で用いた検体の持つ主な機能について表 1 にまとめる。

表 1: 検体の識別番号と主な機能

	機能
1	ブラウザのスタートページ変更,IRC 機能
2	耐解析機能,IRC 機能
3	DLL ファイルの作成とインストール
4	キーロガー, パスワード盗聴, バックドア
5	ファイアウォールの設定解除 感染端末の情報送信, 各種プロキシ機能
6	他のプログラムの削除, ファイルの ダウンロードと実行,Network 感染機能
7	他のプログラム削除,Network 感染機能 パッチのダウンロードと実行
8	感染端末内の HTML ファイルの改ざん DoS 攻撃機能 (実行はされない)
9	ファイルのダウンロードと実行 IRC 機能
10	Autorun.inf の作成,Network 感染機能 ブラウザのスタートページ変更

3.2 提案概要

類似した機能を持つ検体同士、機械語レベルで何らかの共通点があると考え、コードの類似性から機能の類似を示すこと目指す。

本研究ではマルウェア全体ではなく、関数単位のコードに注目する。一般にプログラムは複数の関数から構成されており、関数がプログラムの持つ機能や挙動を決定するので、類似した関数を持つ検体同士の持つ機能は類似すると考える。そこで異なる複数のマルウェアを逆アセンブルしたものを用意し、以下に述べる提案手法を適用する。

1. 関数間の類似度の算出

   マルウェアを逆アセンブルしたものを関数単位で切り出し、異なるマルウェアの関数同士を総当たりで比較し、類似度を算出する。

2. 検体間の類似度の算出

   関数間の類似度から検体間の類似度を算出する。重み付けをした類似度の代表値を用いて検体間の類似度を求める。

3. 算出した類似度を用いた分類化

   算出した検体間の類似度を用いてクラスタリングを行い、デンドログラム (樹状図) を作成することにより分類する。

既に著者らは、具体的な類似度算出方法については言及している [5]。

3.3 類似度算出対象の比較

本研究では、類似度の算出対象をマルウェアの関数コードとしている。マルウェアの機能や挙動を推測するために関数のコードの類似度を用いた場合とマルウェア全体のコードを用いた場合との違いについて検証する必要があると考える。以下の手順で検証を行う。以後、関数毎に求めた類似度を【関数間の類似度】、マルウェア全体に対して求めた類似度を【全体間の類似度】と表記する。

1. 関数間の類似度及び全体間の類似度を算出
2. 検体間の類似度を算出
   - ・関数間の類似度では代表値
   - ・全体間の類似度では自身の持つ値
3. コードの差分とそれぞれの手法の検体類似度値を比較し、関数間類似度と全体間類似度の特徴を考察

3.4 関数間の類似度と機能の関連性

関数間の類似度をもとにマルウェアの機能や挙動に着目した分類を行うためには、関数間の類似度と機能の関連性について検証する必要があると考える。そこで、関数間毎に算出した類似度を、10 %毎に区切り分析を行う。具体的には、関数のコード同士の diff をとり、コードの内容と類似度の違いをまとめ考察を与える。以下、言葉の定義を記載する。尚、解析済みコードとは、解析を行い関数名や変数名を統一した状態を指す。

・コードの一致

解析済みのコードが完全に一致する場合

(Nop 命令やオペランドの入れ替わり、及び使用するレジスタの違いのみの場合一致に含める)

・コードの類似

解析済みのコードに部分一致があり、全体に対する部分一致が 50 %を超えている場合

4　類似度算出対象の比較

本章では関数間の類似度と全体間の類似度をそれぞれ求めた結果に対し、考察及び問題点を述べる。類似度算出方法はLCSとN-gramを用い、関数間の類似度の平均値を代表値とし検体間の類似度を算出した。

4.1　LCS

LCSの場合は検体間類似度はすべて関数毎に区切った場合の値が高くなった。コード全体ではLCSとして算出されなかった部分一致のものが関数毎に区切る事により反映された結果だと考えられる。つまり、関数毎に区切って類似度を算出する手法が、全体に対して類似度を算出する手法よりコードの類似を反映する事が分かった。しかし、現状の算出方法では代表値の取り方として平均値をとっているため、関数サイズを問わず値の重みは同じである。コードを実行する際に与える影響が大きいのは関数サイズが大きいものである可能性が高いので、サイズを根拠とした重み付けは必要だと考える。同時に、APIやライブラリ関数を呼び出して使用している場合も、考慮した重み付けの設計が必要である。

また、関数間の類似度は全体間の類似度より平均して8％高く類似度が算出されていた。IRCボットであり、検体の構造も類似している検体同士の結果は3％高く算出された。一方、ツールで作成されたとされる検体同士の結果が19％高く算出された。コードを精査した所、ツールで作成されたとされる検体は、共通で用いられている機能のコードは一致していた。主な機能の部分はサイズの大きな関数になる事が多い。その部分が異なるため、コード全体での一致する比率は少ないが、関数毎で見ると類似度が高く算出されたと考えられる。

4.2　N-gram

N-gramの場合、算出した検体間類似度の3分の2で全体間の類似度が関数間の類似度を上回るという結果が出た。ツールで作成されたもののみ、関数間類似度が検体間類似度を10％程度上まっていたが、その他の検体は－5％～＋3％の差が生じた。同じモジュールを用いて算出しているため、出現した要素と出現回数は同一であるが、検体毎にこのような差が生まれた。原因の一つとしてLCSでも上げた、関数サイズが上げられる。関数毎に区切る事により、短い関数の影響が強まると考えられる。

また、N-gramを求める際に、コード全体に対して適用する時に比べ関数毎では特徴となる要素とその出現回数が少ない。そのため、類似度を比較するための特量として十分なサンプル数とは言えない可能性がある。つまり、関数といった小さな単位にはN-gramは不適当だということが明らかになった。

5　関数間の類似度と機能の関連性

5.1　結果

解析データと類似度の関連性を表2にまとめる。比較した2つの関数を1ペアとし、コードの一致及びコードの類似を、それぞれ一致・類似と表記する。

表 2: 関数間類似度と解析結果の関係

類似度	特徴	
60％以上	一致	112ペア中112ペア
50％～60％	一致	1 4ペア中12ペア
	類似	1 4ペア中2ペア
40％～50％	一致	4 3ペア中1ペア
	類似	4 3ペア中39ペア
	上記以外の3ペア 一致及び類似：無	
30％～40％	一致	118ペア中36ペア
	類似	ペア中ペア
	上記以外の82ペア 一致及び類似：無	
10％～30％	一致及び類似：無 関数の挙動の類似は有り	
10％～30％	一致及び類似：無	

表 3: コードの類似例

関数 A	関数 B
push ebp mov ebp esp push ecx lea eax [ebp+hKey] push eax	push ebp mov ebpesp sub esp 8 lea eax [ebp+hKey] push eax
push offset aSoftwareCvc push HKEY_CURRENT_USER call ds:RegCreateKeyA	push KEY_ALL_ACCESS push 0 push offsetSubKey push HKEY_CURRENT_USER call ds:RegOpenKeyExA test eax eax jnz loc_402E72 mov dword ptr [ebp+Data] 0
push 4 lea ecx [ebp+Data] push ecx push 4 push 0 push offsetaDl mov edx [ebp+hKey] push edx call ds:RegSetValueExA mov eax [ebp+hKey] push eax call ds:RegCloseKey mov esp ebp pop ebp retn	push 4 lea ecx [ebp+Data] push ecx push 4 push 0 push offsetValueName mov edx [ebp+hKey] push edx call ds:RegSetValueExA mov eax [ebp+hKey] push eax call ds:RegCloseKey mov esp ebp pop ebp retn

5.2 考察

類似度が 60 %以上のものは 112 ペア存在し、その全てで関数のコードの一致が見られた。コードが一致する場合、同じ挙動をとる。類似度にばらつきがある原因は、命令とオペランドの入れ替わりとレジスタの違いの他に、分岐命令や mov 命令のオペランドだと考える。例えば、call 命令で API を呼び出す際、検体や解析するコンピュータによってアドレスが異なる。それらの命令が多い検体は類似度が下がる事が分かった。

類似度 50 %台は 14 ペア存在し、そのうち 2 ペアを除く 12 ペアのコードが一致した。コードが一致しなかった 1 ペア (表 3) では一部のコードの固まりが一致している。2 ブロック目のコード関数 A ではレジストリキーを新しく作成する API が呼び出され、関数 B では既存のキーを開く API が呼び出されている。どちらの関数もレジストリキーをセットする機能の関数であった。

類似度 40 %台は、43 ペア存在した。そのうちコードの一致は 1 ペアのみであった。39 ペアは表 3 のような一部コードの一致があり、使用する API も一致していた。レジスタが異なる部分、デコードや実行環境や検体依存のサブルーチンをもつ部分があったため、コードの一致ではなく類似に留まったと考えられる。残り 3 検体はコードの類似は見られなかった。類似が見られなかったペアは十数 byte の短い関数であり、mov,push,pop 命令から構成されていたためにこのような類似度が算出されたと考える。

類似度 30 %台では、コードが類似している関数が見られたが、コードの類似がない関数も多

く見られた。コードの類似さえ無い関数の多くは 40 %台の考察で述べたように十数 byte, 最も大きいものでも 30byte 程度であり、この関数サイズに対する push や mov 命令の数が近かったため、それらの値が影響したものと考えられる。

一方、30 %以下の類似度を持つものにコードの一致や類似は見られなかったが、類似度が低い値であっても発現する機能が似ているものも存在する。例えば、検体 4 と 5 はお互いバックドア機能を有している。その機能を持つ関数同士を比較してみると類似度はわずか 16 %であったが、総当たりで類似度を算出した中でバックドアの機能を持つ関数同士の類似度が最も高かった。検体 1 と 2 に関しても、IRC の機能を持つ関数同士が総当たりの関数の中で一番類似度が高かったが値は同じく 16 %程度であった。これらの関数サイズは、同一検体の関数の数倍〜10 倍である。関数サイズが大きいという事は、類似度を算出する際の分母が大きくなる。同時に関数サイズが大きいからこそ機能に直接関係のない共通の特徴が存在し、類似度算出の分子が大きくなることが考えられる。現状の類似度算出方法では、類似度が低いものに関して、機能的に類似しているものと類似していないものの区別が難しいと言える。

6 おわりに

本論文では、提案手法に関して 2 点の検証を行い、考察を与えた。はじめに、類似度を算出する対象について、関数間の類似度と全体間の類似度を比較した。その結果、LCS・N-gram のどちらの手法も、関数サイズによって左右されている可能性がわかった。N-gram は関数の比較には適さない事が明らかになった。LCS については、関数間で算出するほうがコードの一致をより多く抽出できるが、重複して一致をカウントしている可能性もある。今後は、LCS 長のみをとるのではなく、何が一致しているかを記録する事により、重複を防ぐ機構が必要である。

また、関数毎に類似度を算出すれば、関数サイズや関数間類似度の値に応じた重み付けを行うことが可能となる。重み付けの設計次第で、主な機能が一致または類似していることを類似度として反映することが可能かもしれない。今後は検体数を増やし傾向を抽出し、そこから重み付けの設計を目指す。

次に関数間の類似度と機能の関連性について考察した。現状の類似度算出方法では、誤差はあるものの、コードが一致している部分が多いものは機能が一致していることが確認できた。しかし十数%の類似度であってもそのコード内で検体の核となる共通の機能を持っているものも存在した。今後の課題として、「機能が一致または類似しているならばコードの類似度が高くなる」という命題を満たすことが上げられる。現状では、「コードが一致いている部分の機能は一致している」という部分しか満たせていない。命令の順序や使用するレジスタが変わるだけで異なるコードとなる。今後、そのような差を吸収するような機構の設計を目指す。

参考文献

[1] 独立行政法人情報処理推進機構, 2010 年版 10 大脅威, http://www.ipa.go.jp/security/vuln/10threats2010.html

[2] McAfee,“ 2010 年第 3 四半期脅威レポート ”, http://www.mcafee.com/japan/

[3] Md. Enamul Karim, Andrew Walenstein, ArunLakhotia, Laxmi Parida, “ Malware phylogeny generation using permutations of code. ”, Journal in Computer Virology vol.1, pp.13-23, 2005.

[4] 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, pp1-11, 2010.

[5] 東結香, 中津留勇, 真鍋敬士, 猪俣敦夫, 藤川和利, 砂原秀樹,”コードに基づいたマルウェアの機能推定に関する研究”, 2011 年暗号と情報セキュリティシンポジウム予稿集,No.3B4-4,Jan,2011