Computer Security Symposium 2019
21 - 24 October 2019

動的シンボル情報を用いたLinuxマルウェアの検知

イボット アリジャン[1,a)]　大山 恵弘[1]

概要：近年，IoT や組み込みデバイスの普及により Unix 系 OS を狙う ELF 形式のマルウェアが増加している．一方で ELF ファイルは対象となるアーキテクチャの多さなどターゲットの多様性が高く，解析が難しいという側面がある．本研究ではアーキテクチャに依存しない解析手法として動的リンク情報，とりわけ再配置されるシンボルの名前を使用した Linux マルウェアの検知を試みる．先行研究として import API の fuzzy hash を使用する Windows マルウェアの分類手法に注目し，ELF ファイルから得られたシンボル名に fuzzy hash 関数を適用する．さらに得られたハッシュ値をベクトル化することで，機械学習を用いたマルウェア検知手法を提案する．

キーワード：マルウェア，表層解析，ELF，Linux

Detection of Linux Malware Using Dynamic Symbol Information

Ibot Arijan[1,a)]　Yoshihiro Oyama[1]

Abstract: Recently, with the popularization of IoT and embedded devices, the number of ELF formatted malware targeting Unix-like operating systems is increasing. On the other hand, ELF files are difficult to analyze because of the high target diversity such as the large number of target architectures. In this paper, we attempt to detect Linux malware using information on dynamic linking, particularly the names of relocated symbols, as an architecture-independent analysis method. We focus on classification methods of Windows malware in previous studies in which import APIs are used as inputs of fuzzy hashing, and applies fuzzy hash functions to symbol names obtained from ELF files. In addition, we propose a malware detection method using machine learning by vectorization of obtained hash values.

Keywords: Malware, static analysis, ELF, Linux

1. はじめに

近年，IoT デバイスの流行に伴いセキュリティが貧弱な IoT デバイスが増加し，それを狙うマルウェアが増加している．Kaspersky Lab の調査レポート [1] によると，同社によって収集された IoT マルウェアは 2016 年時点で 3219 検体だったが，2018 年においては前半期のみで 12 万検体以上にのぼる．また IoT デバイスだけでなく，Unix 系 OS が使用されたサーバを狙ったマルウェアも増加している [2]．

2016 年には Mirai [3] と呼ばれるマルウェアが発見され，Mirai により形成されたボットネットを使用した大規模な DDoS 攻撃が行われた [4]．その後 Mirai のソースコードが作成者によって公開されたことによって，さらに大量の亜種マルウェアが作成されている．このような新たな脅威に対応するため，より高精度なマルウェア検知手法が必要とされている．

IoT デバイスやサーバーで多く使用される Linux は，実行ファイル形式に ELF を採用している．しかし ELF ファイルのマルウェア検知に関する研究は，Windows を対象とした PE ファイルのマルウェアと比べて少ない．その理由の一つとして挙げられるのが，Linux が動作するターゲットの多様性である．たとえば実行コードを解析する場合，Windows を狙ったマルウェアでは x86 や x86-64 のアセンブリ言語が使用されるのが一般的である．一方で Linux は

[1] 筑波大学
University of Tsukuba
[a)] ibot@syssec.cs.tsukuba.ac.jp

数多くのアーキテクチャに対応しているため，同じ動作をする実行ファイルであっても全く異なるコード領域を保有している事がある．さらにデータ領域を解析する際も，ビッグエンディアンとリトルエンディアンのアーキテクチャが存在することから，同じ数値であるにも関わらずバイナリレベルでは表現が異なる場合がある．

また，Linux マルウェアを IoT デバイスで扱う場合は，IoT デバイス固有の貧弱な計算資源を考慮する必要がある．たとえばマルウェア検知手法としてよく使用されるビヘイビア法は，実行中のプログラムの挙動を監視して，悪意のある処理が行われたかどうかを調べる動的解析手法である．しかし IoT デバイスでは CPU やメモリなどのハードウェア資源が少なく，動的解析のようなコストの大きい解析手法を行うのは難しいと考えられる．さらに IoT デバイスの記憶容量は小さいことが多く，サイズの大きなシグネチャファイルは保存できない可能性もある．

このような状況で使える軽量なマルウェア検知手法として，パターンマッチングやシグネチャマッチングがあげられる．JPCERT/CC の調査レポート [5] では，fuzzy hash 関数の入力に PE ファイルの import API を使用して得られた fuzzy hash 値が，マルウェア分類に有効であったと報告されている．以後この fuzzy hash 値をハッシュ値，fuzzy hash 関数をハッシュ関数と記述する．これは同じマルウェアファミリーならば，インポートする関数にも重複が多いという理論に基づき，import API からマルウェア間の類似度を判定する手法である．

本研究ではこれを Linux マルウェアの検知に応用することで，軽量かつアーキテクチャに依存しない検知手法を提案する．動的リンク用のシンボルテーブルから得られるシンボル名からハッシュ値を取得し，ベクトル化することで機械学習を用いたマルウェア検知モデルを作成する．

提案手法の特徴として，動的リンクされるシンボル名を使用するためアーキテクチャやエンディアン，32 ビットや 64 ビットの違いに左右されないことがあげられる．これは Mirai のように，多種多様な環境に対応するマルウェアに対して効果的である．また，シンボル名のみをハッシュ関数の入力とするため計算コストが低く，シグネチャも 100 バイト程度のハッシュ値で表現できることから，計算機資源が貧弱な IoT デバイスにおいて有用である．

2. 関連研究

伊藤らの研究 [6] では，本研究と同じく文字列情報を使用したマルウェア検知を行っている．彼らの研究では GNU strings を用いてマルウェア全体から文字列情報を取得した後に，複数の自然言語処理ライブラリから言語モデルを構築し，マルウェア分類器を作成している．一方で文字列情報が少ないファイルや，サイズが大きいファイルのバイナリデータ全体から文字列を探す事は非効率である．本研究はヘッダから動的リンク情報をたどるため，より低いコストで動的リンクに使用される文字列のみを抽出する事ができる．この文字列情報にはシンボル名が含まれており，マルウェアと良性ソフトウェアを区別するための特徴となる．また，ハッシュ値の計算は言語処理ライブラリを使ったベクトル化より，計算コストが低いと推測できるため，本研究で提案する手法は計算機資源が少ない IoT デバイスに有効である．

愛甲の研究 [7] では，本研究と同じく fuzzy hash 関数を用いたマルウェア検知手法を提案している．論文ではファイル全体のハッシュ値と Suffix Tree を利用したアルゴリズムを使用して，ある時点までのデータから学習を行い，それ以降のデータに対してマルウェアの検知を行っている．本研究ではファイル全体ではなく，シンボル名からハッシュ値を取得している．これは Linux マルウェアでファイル全体を使用すると，対象アーキテクチャに大きく左右されてしまう一方で，シンボル名はアーキテクチャに依存しない部分が多いからである．

Linux マルウェアのシンボル情報を用いたマルウェア検知手法として，Jinrong らの研究 [8] がある．彼らの研究ではシンボル情報から 100 回以上使用されるシステムコールを抽出して，高い検知率のマルウェア検知モデルを作成している．一方で使用されたデータはマルウェアの検体が 763 件と少なく，756 件の良性ソフトウェアについても，どの Linux ディストリビューションから取得したか明確でない．プログラムによって使用される関数やシステムコールは，インターネット通信の有無で大きく異なることが推測される．また，良性ソフトウェアには通信を行わないプログラムも存在するが，マルウェアの大半は通信を行うという違いがある．よって，良性ソフトウェアを扱う時はどこから取得したのか，どれほどインターネット通信を行うかという点を明示する事が重要である．本研究では良性ソフトウェアがインターネット通信を行う場合でも，誤検知が少ないマルウェア検知手法を提案する．

IoT デバイスのマルウェア対策については，貧弱な計算資源を考慮して可能な限り計算コストを削減することが求められている．大谷らの研究 [9] ではマルウェア検知機構を外部にオフロードすることで，IoT デバイスの計算コストを減らす手法が示されている．ここで表現されるオフロードとは，ARM の TrustZone [10] などに代表されるハードウェア実装されたセキュリティ機構を指している．彼らの研究ではレジスタの値やキャッシュヒット率など，動的なプロセッサ情報を使用したマルウェア判別器のハードウェア実装を提案している．本研究では，動的解析でなく計算コストの低い特徴量を使用した表層解析によって，計算資源が貧弱な IoT デバイスにも有効なマルウェア検知手法を提案する．

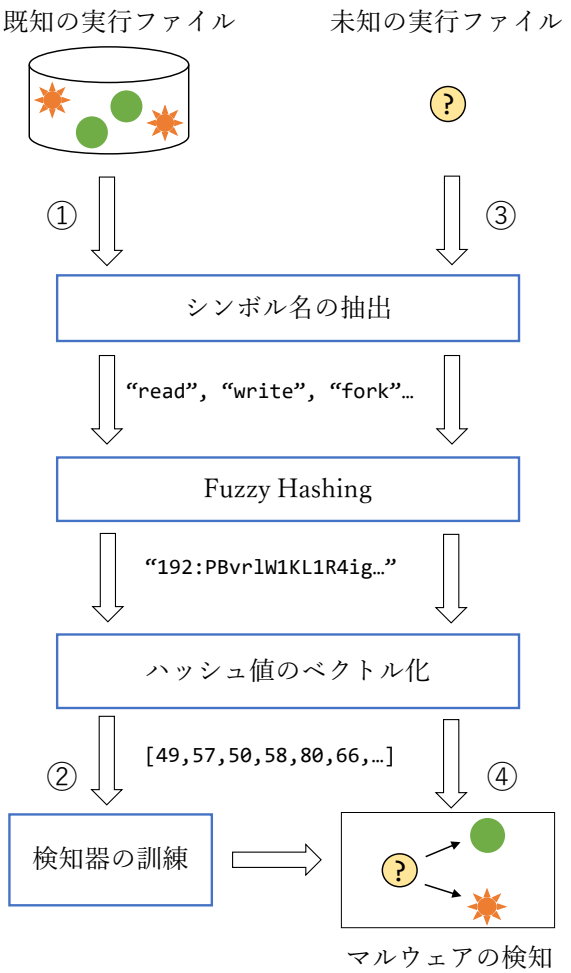

図 1　提案手法の概要

3. 提案手法

本研究では図 1 のような，動的リンクされるシンボルの名前を使った Linux マルウェアの検知手法を提案する．

3.1 動的リンク情報

ELF ファイルには静的リンクされたものと，動的リンクされたものが存在する．静的リンクとは，ライブラリなどを実行ファイルに組み込むことで，必要なコードが全て揃った実行ファイルを生成できる手法である．一方で動的リンクは，プログラムを実行するとインタプリタが OS によって起動され，ライブラリなどの共有オブジェクトが実行ファイルにリンクされる手法である．このインタプリタがリンクを行う際に使われる情報が，動的リンク情報である．

ELF の実行ファイルにおいて，インタプリタが動的リンク情報をどのように使用するか，その概要を以下に示す．なお，遅延リンクなどは考慮しないものとする．

カーネルにより起動されたインタプリタはプログラムヘッダーを探索し，動的リンクに関する情報が含まれている DYNAMIC セグメントを取得する．このセグメントは以下のデータへのアドレスやサイズなどが含まれている．

- JMPREL : 再配置が必要なアドレスとシンボルのペア
- PLTGOT : 実際に書き換えが行われる領域
- SYMTAB : 動的リンクされるシンボルのテーブル
- STRTAB : 動的リンクに関係した文字列

ここで JMPREL が所有するシンボルは SYMTAB へのインデックスであり，SYMTAB が所有するシンボルの名前は STRTAB のオフセットである．インタプリタは JMPREL からシンボルを取得し，PLTGOT を書き換えることで動的リンクを実現する．実行プログラムが動的リンクされたコードを呼び出すには，リンカーによって自動的に生成される関数を経由する．この関数は PLTGOT からアドレスを取得して，そこへジャンプするというラッパー関数のようなものである．

なお，ELF ファイルの構造については Linux Programmer’s Manual の ELF (5) や，複数の書籍 [11,12] で詳しく説明されている．

3.2 シンボル名の抽出

シンボル名を得るためには動的リンク用のシンボルテーブルを取得する必要があるが，これはセクションヘッダから探す方法と，プログラムヘッダから探す方法が存在する．ELF ファイルには「再配置可能オブジェクト」「共有オブジェクト」「実行可能ファイル」という 3 つのタイプがある．実行可能ファイルは，プログラムヘッダが必須となっている代わりに，セクションヘッダが省略可能である．一方でオブジェクトファイルは，セクションヘッダが必須である代わりに，プログラムヘッダが省略可能となっている．実験では共有オブジェクトや再配置可能オブジェクトも検体として含めるため，2 種の方法を使い分ける．

ここで動的リンク用のシンボルテーブルや文字列は，参照しない領域ならば自由に書き換え可能なため，検知アルゴリズムから回避するために偽のシンボル情報を挿入する可能性が考えられる．その偽造されたシンボル名を，さらに回避するためにアドレスの再配置リストをたどる方法もあるが，参照されないシンボルの存在は明らかに不自然であるから，ファイルの改変を検知できる．もっともエラー処理を行う関数などは，実行終了まで使われない場合もあり，リンクされていることはその関数が実際に呼ばれていることと異なる．よって偽のシンボル情報に対抗する必要は薄いと判断し，直接シンボルテーブルから取得する方法を用いる．

シンボルテーブルから得られたシンボル名には名前マングリングされている物も含まれているため，必要に応じてデマングルを行う必要がある．また同じシンボル名の集合であっても，順番が異なれば生成されるハッシュ値が異なってしまうため，シンボル名のソートを行う．

3.3 特徴ベクトルの作成

機械学習によってマルウェアの検知モデルを作成するため，得られたシンボル名を特徴ベクトルに変換する．この時点で得られたシンボル名は数百キロバイトになることも多いため，一度ハッシュ関数によって 100 バイト程度のハッシュ値にする．これにより直接シンボル名をベクトル化する手法と比べて，特徴の次元を減らすことができる．

ハッシュ値の計算には ssdeep [13] を使用し，生成されたハッシュ値は ASCII コードでベクトル化を行う．ASCII コードを使用すると，似た文字列から生成される特徴ベクトルのユークリッド距離が短くなるため，ハッシュ関数の特性を失わないという利点がある．なお，機械学習を行うためにはベクトル長を統一する必要がある．ssdeep によって生成されるハッシュ値は可変長であるが最大値は 148 であるため，ベクトル化する際にゼロパディングを行い，ベクトル長を 148 に固定する．

3.4 マルウェア検知モデルの作成

機械学習のアルゴリズムにはランダムフォレストを選択した．ランダムフォレストは少しずつ異なる決定木を複数集め，その平均を取ることで過剰適合を減らす事ができるアンサンブル法の一種である．並列化が容易であるため特徴ベクトルが高次元であっても効率的に処理できるほか，スケーリングが不要であり多くの場合デフォルトパラメータで十分に機能する点から，ランダムフォレストが適していると判断した．

3.5 実装

実装には Python 3 (3.7.4) を使用した．プログラムを作成するにあたり，利用したモジュールとその用途について以下に示す．

- lief (0.10.0.dev0)：シンボル名の抽出
- cxxfilt (0.2.0)：シンボル名のデマングル
- ssdeep (3.3)：ハッシュ値の計算
- scikit-learn (0.21.3)：ランダムフォレストによる分類

ランダムフォレストがデフォルトで使用する決定木は 10 個であるが，scikit-learn はバージョン 0.22 から決定木を 100 個にすると告知 [14] している．そのためランダムフォレストのハイパーパラメータは決定木の個数のみ 100 に設定し，後はデフォルト値を使用する．

4. 実験

4.1 データセット

マルウェアの検体は VirusShare [15] から 2019 年 2 月までに集められた ELF 形式のマルウェアのうち，動的リンクされる検体のみ抽出した．「DYNAMIC セグメント」が存在し「要求される共有オブジェクトが 1 つ以上存在する」場合に動的リンクと判断している．なお収集されたマルウェア検体の中には VirusTotal [16] で全くマルウェア判定されないバイナリが存在した．これらは未知のマルウェアである可能性も秘めているがマルウェアでない可能性もあり，学習用のデータとしては不適切と判断したため除外した．

表 1　収集したマルウェア検体のファミリ

ファミリ名	検体数	割合 (%)
Mirai	276	8.7
RST	220	6.9
Tsunami	138	4.3
Osf	115	3.6
Jiagu	99	3.1
Grip	42	1.3
CoinMiner	35	1.1
Gafgyt	33	1.0
Ramen	31	1.0
(others)	2198	69.0

表 2　収集したマルウェア検体の対象アーキテクチャ

アーキテクチャ	検体数	割合 (%)
x86	2335	73.3
ARM	539	17.0
x86-64	169	5.3
SPARC	58	1.8
MIPS	44	1.4
(others)	42	1.3

VirusTotal のスキャン結果から得られたマルウェア検体のファミリを**表 1** に記載する．Mirai に分類された検体が 1 割弱を占めており，次いで RST (Remote Shell Trojan) や Tsunami といったファミリが多い．その他にはジェネリックとして検知されたファイルや，マルウェアのダウンローダー，ルートキットなども含まれていた．また，収集したマルウェア検体の対象アーキテクチャを**表 2** に示す．x86 が最も多かったが，ARM や SPARC, MIPS といったアーキテクチャを対象とするマルウェアの存在も確認できる．よってこれらマルウェアを検知するためには，複数のアーキテクチャに対応できる手法が必要となる．

マルウェアの検知モデルを作成するには良性ソフトウェアも用意する必要がある．しかし提案手法では動的リンクされる関数名を含むシンボル名を使用するため，インターネット通信の有無によって大きく結果が異なることが推定される．そのため良性ソフトウェアは，ペネトレーションテスト用のツールが複数存在する Kali Linux から取得した．x86-64 を対象とした Kali Linux 2019.2 の Live イメージから/usr/bin と/usr/sbin ディレクトリにある動的リンクされた ELF ファイル 2222 個を抽出した．

より正確な学習モデルを生成するために，3187 個のマルウェア検体から無作為に 2222 個選択し，マルウェアと非マルウェアの検体数を揃えた．また，学習用としてデータの 75%，テスト用として残りの 25%を使用した．

表 3　混同行列 (Confusion Matrix)

		真の結果	
		悪性	良性
予測結果	悪性	True Positive (TP)	False Positive(FP)
	良性	False Negative(FN)	True Negative(TN)

$$
\begin{aligned}
\text{精度} &= \frac{TP+TN}{TP+FP+FN+TN} \\
\text{適合率} &= \frac{TP}{TP+FP} \\
\text{再現率} &= \frac{TP}{TP+FN} \\
\text{F 値} &= 2\cdot\frac{(\text{適合率})\cdot(\text{再現率})}{(\text{適合率})+(\text{再現率})}
\end{aligned}
$$

図 2　評価指標の計算式

4.2　評価指標

評価指標の計算には**表 3** に示す混同行列の結果を使用する．検知モデルによってマルウェアと予測された検体は Positive，そうでない場合は Negative とする．そして予測結果が正しい場合は True，間違っていれば False を先頭に付ける．たとえば，良性ソフトウェアを誤ってマルウェアと予測した場合，その検体は False Positive に分類される．

本研究では評価指標として精度・適合率・再現率・F 値を用いる．各評価指標の計算式は**図 2** に示すとおりである．

4.3　予備実験

Linux マルウェアの検知モデルを作成する前に，ハッシュ値をベクトル化する提案手法の有用性を，FFRI Dataset 2018 [17] を用いて確認する．FFRI Dataset 2018 は PE ファイルの表層解析ログであり，マルウェア 29 万件と良性ソフトウェア 21 万件が収められている．

様々なフィールドがデータセット内に存在するが，提案手法を再現するために fuzzy hash 関数によって生成された値として impfuzzy と ssdeep のフィールドを用いる．ここで impfuzzy は import API のハッシュ値を，ssdeep はファイル全体のハッシュ値を示す．

正確な結果を得るためにマルウェアデータ内から無作為に 21 万件抽出して，良性ソフトウェアと数を揃えた．そしてデータセットを学習用とテスト用に分割し，ベクトル化されたハッシュ値を元にマルウェア検知モデルを作成した．その結果を**表 4** に示す．ssdeep により生成された検知モデルは高い適合率を実現しているが精度・再現率は低い．一方で impfuzzy により生成された検知モデルは，全ての評価指標において ssdeep で生成したモデルより優れており，特に再現率と精度においては大きく上回っている．この結果から，ベクトル化されたハッシュ値はマルウェア検知に有効であり，さらに import API のハッシュ値はファイル全体のハッシュ値と比較して，より有効であることが示された．

表 4　FFRI Dataset 2018 から作成した検知モデル

入力データ	精度	適合率	再現率	F 値
impfuzzy	0.944	0.977	0.915	0.945
ssdeep	0.805	0.957	0.733	0.830

表 5　各手法におけるハッシュ関数の入力データ

提案手法 1	動的リンクされる全てのシンボル名
提案手法 2	動的リンクされる関数の名前
比較手法 1	ファイル全体のバイナリデータ
比較手法 2	動的リンクに使用される文字列テーブル

表 6　ELF ファイルから作成した検知モデル

	精度	適合率	再現率	F 値
提案手法 1	0.941	0.952	0.928	0.940
提案手法 2	0.919	0.985	0.851	0.913
比較手法 1	0.768	0.837	0.665	0.741
比較手法 2	0.912	0.950	0.869	0.908

4.4　実験内容

Windows マルウェアにおいて import API のハッシュ値がマルウェア検知に有効であったことから，Linux マルウェアにおいても同様に，シンボル名のハッシュ値がマルウェアの検知に有効だと推測される．

マルウェア検知の性能を評価するために，複数の手法を用いて比較実験を行う．実験手法の一覧を**表 5** に示す．本研究が提案する手法ではハッシュ関数の入力にシンボル名を使用するが，動的リンクされるシンボルにはオブジェクト・関数・ファイルなど複数のタイプが存在する．そのため，全てのシンボル名を使用する「提案手法 1」，関数タイプのシンボル名のみを使用する「提案手法 2」として複数の実験を行う．また，性能評価のためにハッシュ関数の入力にファイル全体のバイナリデータを使用する「比較手法 1」と，動的リンクに使用される文字列テーブル全体を使用する「比較手法 2」の実験も行う．なお，提案手法 1 と提案手法 2 についてはシンボル名のデマングルとソートが行われているが，比較手法 1 と比較手法 2 では行われていない．

5.　評価

各手法により作成されたマルウェア検知モデルの性能を**表 6** に示す．ファイル全体のハッシュ値を用いた比較手法 1 は，FFRI Dataset 2018 の ssdeep フィールドを用いた実験と同様に，一定の適合率を保ちつつも精度・再現率が他の手法より劣っていた．

また提案手法 1 によって生成されたモデルは，比較手法 2 で生成したモデルより全ての評価指標において上回った．ここで生じた両者の差は重要である．動的リンクに使用さ

れる文字列テーブルは大半がシンボル名で埋められており「リンクされる共有オブジェクトの名前」や「自分が共有オブジェクトである場合の名前」など，他の文字列はごく一部にすぎない．それにも関わらず検知率において差が生じているのは，シンボル名のデマングルとソートが，ハッシュ値を用いた分類に有効であったためと推測できる．もっともデマングルやソートは一定の計算資源を使うため計算量では比較手法 2 の方が優れており，手法として完全に上位互換であるとは言えない．

興味深い結果として，提案手法 1 と提案手法 2 によって生成された検知モデルの関係があげられる．関数シンボル名は全シンボル名の中から抽出しているため，提案手法 1 は提案手法 2 より多くの情報量を持っている．その余分な情報が負の側面で働けばモデルの検知性能は低くなり，正の側面で働けば高くなると予想していた．しかし実験結果を確認すると提案手法 1 は精度・再現率・F 値が優れており，提案手法 2 は適合率が優れていた．特に適合率と再現率において両者の差は顕著である．この実験から，ハッシュ関数の入力として全シンボル名の代わりに関数名を使用することで，検知できないマルウェア検体が 2 倍に増えるが，誤検知を半分に抑える事が出来るという結果を得られた．

なぜこのような結果になったのかは今後の研究課題としたいが，どちらの手法を使うかについては議論の余地がある．F 値や精度を指標としてマルウェア検知の性能を評価するならば提案手法 1 の方が優れている．単一の特徴量を使用したモデルとしては精度・適合率・再現率ともに高い水準でまとまっており，アンサンブル学習への応用も考えられる．一方でマルウェアの検知率を大きく下げても，良性ソフトウェアの誤検知を避けたい場合は提案手法 2 を使用する事が望ましい．たとえば複数のマルウェア検知手法を組み合わせる場合，互いにカバーすることで検知率を高めることは出来るが，各手法の誤検知が多いと結果として全体の正答率が上がらない．結論として提案手法 1，提案手法 2 ともにマルウェアの検知に有効であり，状況に応じて使い分ける事が望ましいと考えられる．

6. おわりに

本研究では ELF ファイルの Linux マルウェアに対して，動的リンクされるシンボル名と fuzzy hash 関数を用いたマルウェア検知方法を提案した．

複数の手法を比較するために，「全シンボル名」「関数シンボル名」「文字列テーブル」「ファイル全体」から生成したハッシュ値を使用し，機械学習によって得られたマルウェア検知モデルの性能を比較・評価した．その結果，全シンボル名を用いた手法では F 値が 0.94 となるマルウェア検知モデルの作成に成功した．また同じシンボルテーブルから得られる情報であるにも関わらず，全シンボル名を使用する場合と関数シンボル名のみを使用した場合において，前者は総合的な検知精度で優れており，後者は誤検知の少なさで優れているという実験結果を得られた．

次に今後の課題を述べる．提案手法では動的リンクされた ELF ファイルの分類を行ったが，実際には静的リンクされたファイルも多く存在する．しかし IDA の F.L.I.R.T. [18] をはじめとして，リンクされた関数を特定する手法が考案されており，これらを静的リンクされたマルウェアに応用することで提案手法を再現することが可能である．また，提案手法では取得したシンボル名をベクトル化するために fuzzy hash を用いたが，word2vec [19] など他の手法によって作成された特徴ベクトルを使用した比較実験を行う予定である．さらに，マルウェア検知モデルの精度を上げるため，分類に失敗したサンプルについても検証を行う必要があると考えている．

謝辞 FFRI Datasets を提供して下さった（株）FFRI および MWS 組織委員会に感謝する．本研究の一部は JSPS 科研費 17K00179 の助成を受けている．

参考文献

[1] Kaspersky, “New IoT-malware grew three-fold in H1 2018”, https://www.kaspersky.com/about/press-releases/2018_new-iot-malware-grew-three-fold-in-h1-2018/

[2] ZDnet, “Google’s VirusTotal puts Linux malware under the spotlight”, https://www.zdnet.com/article/googles-virustotal-puts-linux-malware-under-the-spotlight/

[3] Manos, A., Tim, A., Michael, B. et al.: Understanding the Mirai Botnet, USENIX Security Symp., (2017).

[4] Krebs, Brian (2016-09-21).: KrebsOnSecurity Hit With Record DDoS, KrebsOnSecurity, https://krebsonsecurity.com/2016/09/krebsonsecurity-hit-with-record-ddos/

[5] 朝長 秀誠.: Import API と Fuzzy Hashing でマルウエアを分類する ～impfuzzy～(2016-05-09), JPCERT/CC Eyes, https://blogs.jpcert.or.jp/ja/2016/05/impfuzzy.html

[6] 伊藤 良, 三村 守, 田中 秀磨.: 出現頻度の高い可読文字列を用いた未知のマルウェアの検知手法，コンピュータセキュリティシンポジウム 2018 論文集，pp. 454-461

[7] 愛甲健二.: Concept Drift と Scale-free の性質を持つデータセットに対する検知の自動化手法，コンピュータセキュリティシンポジウム 2018 論文集，pp. 778-784

[8] Jinrong, B., Yanrong, Y., Shiguang, M., Yu, M.: Malware Detection Through Mining Symbol Table of Linux Executables, Information Technology Journal Volume 12 (2013), pp. 380-384

[9] 大谷 元輝，高瀬 誉，小林 良太郎，加藤 雅彦.: プロセッサレベルの特徴量に注目した亜種マルウェアの検知，研究報告コンピュータセキュリティ，2018-CSEC-80(31), pp. 1-8

[10] Arm Developer, https://developer.arm.com/ip-products/security-ip/trustzone/

[11] Jhon R. Levine (1999), “Linkers and Loaders”, Morgan Kaufmann

[12] Ryan “elfmaster” O’Neill (2016), “Learning Linux Bi-

nary Analysis", Packt
[13] ssdeep Project, https://ssdeep-project.github.io/ssdeep/index.html
[14] sklearn.ensemble.RandomForestClassifier, scikit-learn API Reference, https://scikit-learn.org/0.21/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[15] VirusShare.com, https://virusshare.com
[16] VirusTotal, https://virustotal.com
[17] 高田 雄太，寺 真敏，松 隆宏，他.: マルウェア対策のための研究用データセット 〜 MWS 2018 Datasets 〜, 研究報告コンピュータセキュリティ, 2018-CSEC-82(38), pp. 1-8
[18] Hex-Rays: IDA F.L.I.R.T. Technology: In-Depth, https://www.hex-rays.com/products/ida/tech/flirt/in_depth.shtml
[19] word2vec, Google Code Archive, https://code.google.com/archive/p/word2vec/