IoTマルウェアの分類方法に関する検討

大迫 勇太郎[1,a)]　山内 利宏[2]　吉岡 克成[3]　藤橋 卓也[1]　渡辺 尚[1]　猿渡 俊介[1]

概要：モノがネットワークに接続されることが一般的となり，我々の生活が便利になった．しかしながら，ネットワーク接続されたモノは攻撃者の標的となり，大規模な攻撃への温床となっているため，マルウェアへの対応が必須である．IoT デバイス向けのマルウェアへの対策は，攻撃方法の高速な分析と多様な CPU アーキテクチャへの対応が求められる．このような観点から，本稿では IoT デバイス向けのマルウェアのクラスタリング手法「String-based Malware Clustering Algorithm (SMCA)」を提案する．SMCA では，マルウェアから文字列を抽出し，階層的クラスタリングによって分類木を作成する．文字列ベースの手法では，CPU プラットフォームが異なる同種のマルウェアを近くにクラスタリングすることもできる．また，説明変数が文字列であるため，ユーザが解析しやすいという特徴がある．SMCA の有効性を横浜国立大学吉岡研究室の IoT マルウェアデータセットの 4000 検体，VirusTotal を用いて評価した結果，異なるプラットフォームでも機能が似ていれば近い距離にクラスタリングされることが分かった．また，解析の過程で 4000 検体の中でとある文字列を含んだ 25 検体を発見し，調べたところ，そのマルウェアのソースコードらしきものにたどり着くことができた．

キーワード：IoT マルウェア，分類，クラスタリング，可視化，YARA Rule

1. はじめに

ネットワーク機器がありとあらゆる場所に設置されて，モノがネットワークに接続されることが一般的となり，我々の生活が便利になった．今や無線 LAN アクセスポイントを設置していない家はほとんど存在しない．さらに，部屋の錠前がネットワークに接続されることで，電子的に鍵を一時的に人に送ることが可能となって Airbnb などのサービスが可能となった．腕時計がネットワークに接続されることで，自分がいつどのくらい睡眠をとったのかを簡単に把握できるようになった．掃除機がネットワークに接続されることで，部屋をいつどのくらい掃除したのかを簡単に把握できるようになった．

ネットワーク機器やネットワーク接続されたモノが攻撃者の標的になることで，これまでにない規模での被害が生じる温床となっているため，マルウェアへの対応が必須である．本稿では，PC やスマートフォン以外のネットワーク機器や IoT デバイスに感染するマルウェアを IoT マルウェアと呼ぶ．2016 年の IoT マルウェア「Mirai [1,2]」による被害では，当時としては最大のトラヒックである 665 Gbps もの DDoS 攻撃が発生した．Mirai はネットワークカメラや家庭用ルータなどの IoT デバイスを対象としており，マルウェアに感染した 10 万台以上の IoT デバイスで botnet を構成して攻撃を行う．

しかしながら，IoT マルウェアは，既存の PC 向けのマルウェアとは以下の 4 つの違いがあるため，どのような性質を持った IoT マルウェアなのかを判別するのが困難であるという現状がある．

- 初期設定のままでの利用などの初歩的な脆弱性
- 放置・非メンテナンス性
- 多様な CPU アーキテクチャ
- 大量の亜種

まず，IoT デバイスのユーザは非ネットワークの専門家であることが多いため，デフォルトのパスワードのままで IoT デバイスをネットワークに接続してしまうなどの初歩的な脆弱性が存在する状態で放置されることが多い．この 2 つの「初期設定のままでの利用などの初歩的な脆弱性」「放置・非メンテナンス性」の組み合わせによって，IoT マルウェアの感染方法はデフォルトパスワードでログインして IoT マルウェアをダウンロードするといったように，単純な場合が多い．

初歩的な脆弱性がある前提に立って，YARA-rules [3] などのマルウェア検知ルールを用いてネットワーク上で IoT マルウェアのダウンロード自体を検出して感染を防ぐ方法が考えられる．しかしながら，IoT マルウェアの「多様な CPU アーキテクチャ」の特徴によって IoT マルウェアの判別も困難な状況にある．まず，PC 向けのマルウェアはオフィスアプリケーションのマクロ，EXE，SYS，COM など

1　大阪大学大学院情報科学研究科
2　岡山大学学術研究院自然科学学域
3　横浜国立大学大学院先端科学高等研究院／環境情報研究院
a)　osako.yutaro@ist.osaka-u.ac.jp

いくつかの形式があるものの，基本的にはあるマルウェアのバイナリは 1 種類しか存在しない．すなわち，PC 向けのマルウェアでは 1 つのバイナリをマルウェア判定することができれば，そのマルウェアはバイナリの特徴的な一部をシグネチャとすることで検出することが可能である．一方で，IoT マルウェアは IoT デバイスが多様な CPU アーキテクチャで開発されているがゆえに，いろいろな CPU アーキテクチャで動作するように 1 つのソースコードからバイナリの特徴が全く異なる複数のバイナリが生成されることが多い．すなわち，PC 向けのマルウェアと異なって，1 つのバイナリを判別できたとしても，他の CPU アーキテクチャ向けにビルドされた IoT マルウェアを検出できるとは限らない．実際，Mirai では，感染時に複数の異なる CPU アーキテクチャのバイナリをダウンロードして片っ端から実行を試みる．

さらに，「多様な CPU アーキテクチャ」の特徴に「大量の亜種」が加わることで，IoT マルウェアの判別がさらに難しくなる．前述したように，IoT マルウェアが攻撃する IoT デバイスの脆弱性は初歩的な脆弱性であるがゆえに，マルウェア自体の挙動は単純なものが多く，改変も容易である．例えば，Mirai の亜種は Miori，Hajime，Owari，SORA，UNSTABLE など今もなお増え続けている．収集した IoT マルウェアを VirusTotal に入力して AV ラベルを取得したとしても，ある特定のマルウェアであると判別されることは少なく，特徴の異なる複数のマルウェアのラベルが出力されることも多い．

以上のことに鑑みると，IoT マルウェアへの対策に向けて，まずは IoT のマルウェアの分類を効率的に行う仕組みを実現することが求められる．IoT マルウェアに新しい亜種が登場した場合には，過去発見されたどのマルウェアに最も機能が近いかを知ることができれば新しいマルウェアにも素早く対応できる．また，CPU や OS などが異なるものでも同じ機能を持った過去のマルウェアを見つけることができれば，分析効率の改善や対策にも寄与できる．文献 [4] でも，IoT マルウェアのライフサイクルを体系化して分析した結果として，既存の PC やスマートフォン向けのマルウェアの対策ツールは部分的には利用可能であるものの，IoT マルウェアに向けた新たな分析ツールが必要であることを示唆されている．

このような観点から，本稿では，IoT デバイス向けのマルウェアのクラスタリング手法「SMCA (String-based Malware Clustering Algorithm)」を提案する．SMCA では，まず，マルウェアから文字列を抽出して説明変数とする．次に，抽出した文字列を用いて階層的にクラスタリングを行いながら分類木を作成する．文字列ベースのクラスタリング手法を取ることで，CPU アーキテクチャの異なる同種のマルウェアを近くにクラスタリングすることもできる．また，説明変数が文字列であるため，ユーザが理解しやすいという特徴がある．SMCA の有効性を横浜国立大学吉岡研究室の IoTPOT [5] で収集した IoT マルウェアデータセット「IoTPOT malware binary dataset A [6]」を用いて評価した結果，異なるプラットフォームでも機能が似ていれば近い距離にクラスタリングされることが分かった．また，解析の過程で 4000 検体の中でとある文字列 (本稿では仮に Tony 文字列と呼ぶ) を含んだ 25 検体を発見した．Tony 文字列について検索エンジンを用いて調べていたところ，とある検索エンジンで 1 つだけページがヒットし，調べたところそのマルウェアのソースコードらしきものを発見することができた．

本稿の構成は以下のとおりである．2 節で，関連研究について述べ，本研究の位置づけを明らかにする．3 節で提案手法である文字列ベースの IoT マルウェアクラスタリング手法「SMCA」について述べる．4 節で提案手法の評価を行い，最後に 5 節でまとめとする．

2. 関連研究

マルウェアの分類・クラスタリング手法に関する研究は，大きく分けると PC 向けのマルウェアを対象としたもの，Android 向けのマルウェアを対象としたもの [7]，IoT マルウェアを対象としたものに分けられる．PC や Android 向けのマルウェアを対象としたものとしては，様々な検討がなされている．これらの手法は，「多様な CPU アーキテクチャ」の特徴を有する IoT マルウェアには適していない．特に，どの手法も検体が十分にあり，かつ検体に対して正解ラベルが存在する「教師あり学習」の前提で研究がなされており，そもそもラベルをどのようにつければよいかが不明確な IoT マルウェアには適用できない．文献 [8] では，Deep Learning を用いてマルウェアを分類する手法について提案している．文献 [9] では，feature hashing [10,11] を用いた分類手法を提案している．文献 [12] では，Android のマルウェアを対象として，対話的なインタフェースを介してクラスタリングする手法を提案している．文献 [13,14] では，fuzzy hashing [15] を用いてクラスタリングする手法を提案している．

IoT マルウェアを対象としたクラスタリング手法は，冒頭で述べた通り，

- 初期設定のままでの利用などの初歩的な脆弱性
- 放置・非メンテナンス性
- 多様な CPU アーキテクチャ
- 大量の亜種

などの特徴に起因して研究段階であることもあって数が少ない [16–20]．例えば，文献 [16] では，IoT マルウェアの opcode をベクトル空間に写像して fuzzy pattern tree を構築して IoT マルウェアを分類する方法を提案している．しかしながら，opcode は CPU アーキテクチャ毎に異なるため，同じソースコードで異なるプラットフォーム向けにコ

表 1 VirusTotal で出力されたラベルの割合

サンプル番号	Mirai	BASHLITE	その他
1	8.1 %	25.7 %	66.2 %
2	4.0 %	28.0 %	68.0 %
3	4.0 %	32.0 %	64.0 %
4	4.0 %	20.0 %	76.0 %
5	1.3 %	33.3 %	65.4 %
6	2.7 %	33.3 %	64.0 %
7	2.7 %	28.0 %	69.3 %
8	0.0 %	0.0 %	100.0 %
9	2.7 %	29.3 %	68.0 %
10	1.3 %	13.3 %	85.4 %

ンパイルした同一マルウェアの分類には用いることができない．文献 [17] では，function call sequence graph を用いたクラスタリング手法が提案されている．特に，関数名の取得に静的解析ツールである IDA Pro [21] のインタフェースを用いることで多様な CPU アーキテクチャに対応している．

マルウェアを分類するツールとしては，VirusTotal が存在する [22]．VirusTotal は 75 種類のアンチウイルスソフトを使ってファイルや URL が悪意のあるものかどうかの判定を行うウェブサービスである．しかしながら，IoT マルウェアの分類に関する知見がまだ整っていないことに起因して，VirusTotal を使った判定結果の信頼性も現時点では高くない．例として，表 1 に IoTPOT malware binary dataset A の 4000 検体から 10 検体をランダムに抽出して VirusTotal に入力した例を示す．Mirai と BASHLITE は共に botnet を構築して DDoS 攻撃を行う点では共通しているものの，BASHLITE の方が歴史が古く，C&C サーバから受け取るコマンド体系も異なるので異なる IoT マルウェアである [23]．しかしながら，表 1 から分かる通り，判定されるマルウェアの種類にばらつきがあることから IoT マルウェアの分類が難しい課題であることが分かる．

本研究のモチベーションと深く関連するのが YARA-rules [3] である．YARA は VirusTotal で開発されているマルウェアの解析・検知ツールである．以下 YARA の github [3] で公開されている Mirai の YARA rule である「MALW_Mirai.yar」の一部を示す．

```
rule Mirai_1 : MALW
{
        meta:
                description = "Mirai Variant 1"
                author = "Joan Soriano / @joanbtl"
                date = "2017-04-16"
                version = "1.0"
                MD5 = "655c3cf460489a7d032c37cd5b84a3a8"
                SHA1 = "4
                     dd3803956bc31c8c7c504734bddec47a1b57d58
                     "

        strings:
                $dir1 = "/dev/watchdog"
                $dir2 = "/dev/misc/watchdog"
                $pass1 = "PMMV"
                $pass2 = "FGDCWNV"
                $pass3 = "OMVJGP"
        condition:
                $dir1 and $pass1 and $pass2 and not
                     $pass3 and not $dir2

}
```

MALW_Mirai.yar より，Mirai の検出には文字列 `/dev/watchdog` 等を用いていることが分かる．

3. 提案手法

3.1 全体像

IoT マルウェアの多様な CPU アーキテクチャと，大量の亜種を教師なしで効率的に分類することを目的として，文字列を用いた IoT マルウェア分類手法「String-based Malware Clustering Algorithm (SMCA)」を提案する．SMCA はマルウェアのバイナリに含まれる文字列のみを使用するため，CPU アーキテクチャに依らず分類することができる．また，SMCA は教師なし学習に分類されるため，正解ラベルが存在しないような状況でも利用することができる．さらに，YARA rules では IoT マルウェアの判別に文字列を使用していることが多いため，SMCA の出力結果で特定のクラスタの特徴を表す文字列が取得できた場合に，新たな YARA rule を生成するのに使用することもできる．

図 1 に提案手法の全体像を示す．提案手法は，各マルウェアからの文字列抽出 (extract strings)，ベクトル生成 (convert strings to vector)，階層的クラスタリング (hierarchical clustering) から構成される．N はクラスタリング対象のマルウェアのバイナリファイルの個数，μ_i は各マルウェアのバイナリファイル，$\boldsymbol{S}_i$ は各マルウェア μ_i から文字列抽出によって抽出した文字列のテーブル，$\boldsymbol{T}$ は全てのマルウェアから抽出して重複を削除した文字列の集合，$\boldsymbol{V}_i$ は各マルウェア μ_i に対応付けられたベクトル，$\boldsymbol{R}$ は $\boldsymbol{V}_1, \boldsymbol{V}_2, \cdots, \boldsymbol{V}_N$ から階層的クラスタリングアルゴリズムによって生成した樹形図データを意味する．

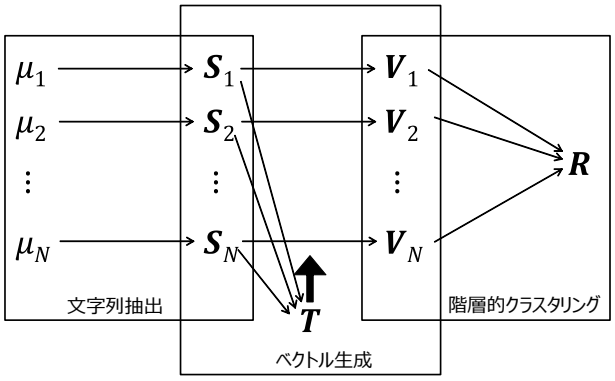

図 1 提案手法の全体像

Algorithm 1 Extract strings

Input: μ: a malware sample
Output: $\boldsymbol{S}$: a set of strings

```
S is an empty array
buffer ← empty string
for each byte B ← μ do
    if B is printable then
        append B to buffer
    else
        add buffer to S
        buffer ← empty string
    end if
end for
```

3.2 文字列抽出

Algorithm 1 に文字列抽出アルゴリズムを示す．Algorithm 1 は，入力があるマルウェアのバイナリ μ，出力がバイナリに含まれる文字列の集合 $\boldsymbol{S}$ である．バイナリの先頭から 1 バイトずつ読み込んで，連続した印刷可能文字を文字列と判定して文字列の集合 $\boldsymbol{S}$ に追加する．文字列抽出アルゴリズムの計算量は，各マルウェア毎にマルウェアのバイナリのバイト数に比例した $O(\mathrm{size}(\mu))$ となる．

3.3 ベクトル生成

ベクトルの生成では，まず，全てのマルウェア検体に含まれている重複なしの文字列の集合である文字列テーブル $\boldsymbol{T}$ を Algorithm 2 を用いて生成して，各マルウェア検体それぞれのベクトル $\boldsymbol{V}_i$ を Algorithm 3 を用いて生成する．Algorithm 2 に，文字列テーブル $\boldsymbol{T}$ の生成アルゴリズムを示す．Algorithm 2 は，入力が各マルウェアから Algorithm 1 を用いて抽出した文字列集合の集合 $\boldsymbol{S}_1, \boldsymbol{S}_2, \cdots, \boldsymbol{S}_N$，出力が文字列テーブル $\boldsymbol{T}$ である．各文字列集合 $\boldsymbol{S}_i$ からそれぞれ文字列 s を取り出して，文字列テーブル $\boldsymbol{T}$ にその文字列 s が含まれていない場合には s を文字列テーブル $\boldsymbol{T}$ に追加する．Algorithm 2 の計算量は，各マルウェアの文字列の数の和になるため，$O(|\boldsymbol{S}_1|+|\boldsymbol{S}_2|+\cdots+|\boldsymbol{S}_N|)$ となる．

Algorithm 3 に，文字列テーブル $\boldsymbol{T}$ を用いてあるマルウェアに含まれる文字列の集合 $\boldsymbol{S}$ からベクトル $\boldsymbol{V}$ に変換するアルゴリズムを示す．文字列テーブル $\boldsymbol{T}$ のサイズのベクトル $\boldsymbol{V}$ を用いて，文字列テーブルの各文字列それぞれに対してその文字列 s が各マルウェアから抽出した文字

Algorithm 2 Generate a string table $\boldsymbol{T}$

Input: $\boldsymbol{S}_1, \boldsymbol{S}_2, \cdots, \boldsymbol{S}_N$: sets of strings
Output: $\boldsymbol{T}$: a string table for generating vectors

```
T is an empty array
for each S in S_1, S_2, ..., S_N do
    for each s in S do
        if s is not in T then
            append s to T
        end if
    end for
end for
```

Algorithm 3 Convert strings to a vector

Input: $\boldsymbol{S}$: strings extracted from a malware
Input: $\boldsymbol{T}$: a string table generated by Algorithm 2
Output: $\boldsymbol{V}$: a vector

```
V is an empty array
for each s in T do
    if s is in S then
        v ← 1
    else
        v ← 0
    end if
    append v to V
end for
```

Algorithm 4 Hierarchical clustering

Input: $\boldsymbol{V}_1, \boldsymbol{V}_2, \cdots, \boldsymbol{V}_N$: a list of vector
Input: $\boldsymbol{R}$: results of clustering Algorithm 4

```
for each (i, j) ∈ [0, N) × [0, N) do
    d[i, j] = distance(V_i, V_j)
end for
L ← (0, 1, ..., N − 1)
chain ← []
while L.length > 1 do
    if chain.length ≤ 3 then
        a ← (x ∈ L)
        b ← (x ∈ L \ {a})
        chain ← [a]
    else
        a ← chain[chain.length − 4]
        b ← chain[chain.length − 3]
        remove last 3 elements from chain
    end if
    while True do
        c ← argmin_{x∈L\{a}} d[x, a]
        append c to chain
        a ← c
        b ← a
        if chain.length < 3 or a ≠ chain[chain.length−3] then
            break
        end if
    end while
    append (a, b, d[a, b]) to R
    remove a, b from I
    n ← (last label) + 1
    for all x ∈ I do
        d[n, x] = d[x, n] = (d[a, x] + d[b, x]) / 2
    end for
    append n to L
end while
```

列の集合 $\boldsymbol{S}$ に含まれていた場合には 1 を，含まれていない場合には 0 としたビットマップを構築することでベクトル $\boldsymbol{V}$ を生成する．Algorithm 3 の計算量は，各マルウェアに対して文字列テーブル $\boldsymbol{T}$ のサイズに比例するため，$O(|\boldsymbol{T}|)$ となる．

3.4 階層的クラスタリング

Algorithm 4 に，階層的クラスタリングの疑似コードを示す．まず，N 個のベクトル $\boldsymbol{V}_1, \ldots, \boldsymbol{V}_N$ に対して，任意の 2 ベクトル間の距離をテーブル d として事前に計算する．ここで，ベクトル間の距離関数 distance にはハミング距離を用いている．次に，nearest-neighbor chain algorithm [24] を利用して，順次新たなクラスタを生成する．このアルゴリズムは貪欲に近傍を探索するよりも効率良くクラスタを探索でき，貪欲法と同一の出力が得られる．Algorithm 4 の計算量については，ベクトル全体を走査してクラスタを統合するループで $O(N)$ かかり，2 つのクラスタを統合したクラスタの距離を更新するためにさらに $O(N)$ かかるため，全体では $O(N^2)$ となる．

4. 評価

4.1 評価で用いたデータセット

提案手法である SMCA を実際の IoT マルウェア検体のデータセットを用いて評価した．データセットとしては，横浜国立大学吉岡研究室の IoTPOT [5] で 2016 年 10 月 2 日から 2017 年 10 月 2 日に収集した IoT マルウェア 4000 検体の「IoTPOT malware binary dataset A [6]」と，github で公開されている Mirai のソースコードからコンパイルした 9 検体の合計 4009 検体を用いた．図 2 に file コマンドを用いて抽出した 4009 検体の CPU アーキテクチャの内訳を示す．組み込み Linux で多く用いられる ARM や MIPS が多いものの，Renesas SH なども 200 検体以上存在することが分かる．4009 検体は各検体毎にバイナリ全体に MD5 をかけたハッシュ値で区別しており，バイナリが異なることを確認できている．また，4009 検体中 1407 検体が strip 処理でシンボル情報が削除された検体であった．

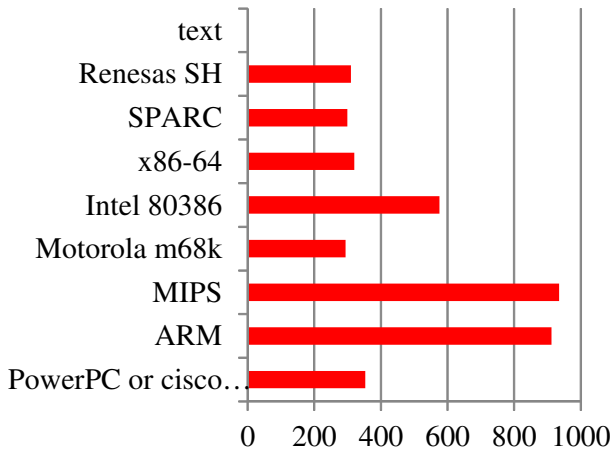

図 2 データセットの CPU アーキテクチャの内訳

4.2 可視化結果

図 3 に提案手法でクラスタリングした結果を可視化したものを示す．中心が分類木の根，一番外側の円が分類木の葉となっている．葉の部分には各検体に振られた 0〜4008 の ID が表示されている．図 3 の可視化結果を基に，手元でコンパイルした Mirai，文字列`/dev/watchdog` を含んだ Mirai の可能性のあるもの，文字列 `fucknet` を含んだ BASHLITE の可能性のあるもの，検証の過程で見つけた Tony 文字列を含んだもの，マルウェア間の距離が近い最大のクラスタのものの 5 つの観点で検証した，葉の数が多くて ID 自体は見えないため，以下では ID に色付けして分析する．

手元コンパイル Mirai

同じソースコードで異なる CPU アーキテクチャが提案手法によってどのように分類されるかを調査するために，ARM 静的リンク版，ARM 動的リンク版，Motorola m68k 版，MIPS MSB 版，MIPS LSB 版，PowerPC 版，Renesas SH 版，SPARC 版，x86 版の計 9 種類の CPU 環境でコンパイルした Mirai の分類木上での位置を確認した．図 3 の赤文字の ID が手元でコンパイルした Mirai を意味している．提案手法では，ソースコードが同じであれば異なる CPU アーキテクチャのバイナリであっても近い位置に分類できることが確認できた．

/dev/watchdog

2 節に示したように，Mirai 用の YARA rule では，`/dev/watchdog` という文字列を使用している．Mirai である可能性のある検体と手元コンパイル Mirai との分類木上での位置関係を調べることを目的として，`/dev/watchdog` を含む検体の分類木上での位置を可視化した．図 3 の紫色文字の ID が`/dev/watchdog` を含んだ検体を意味している．`/dev/watchdog` が含まれている検体は，214 検体存在した．手元コンパイル Mirai にも`/dev/watchdog` が含まれているが，214 検体からは手元コンパイル Mirai は除外している．図 3 より，手元コンパイルの Mirai と`/dev/watchdog` が含まれている検体で構成されるクラスタが重なっていることが分かる．また，一番大きなクラスタには MIPS，x64，x86，ARM，Motorola m68k など様々な CPU アーキテクチャを含んでいることが確認できた．図 3 において「`/dev/watchdog` クラスタ」と記載されているクラスタは Mirai で構成されている可能性が高いと考えている．

BASHLITE

以下に，YARA の github で公開されている BASHLITE の YARA rule である「MALW_Gafgyt.yar」の一部を示す．

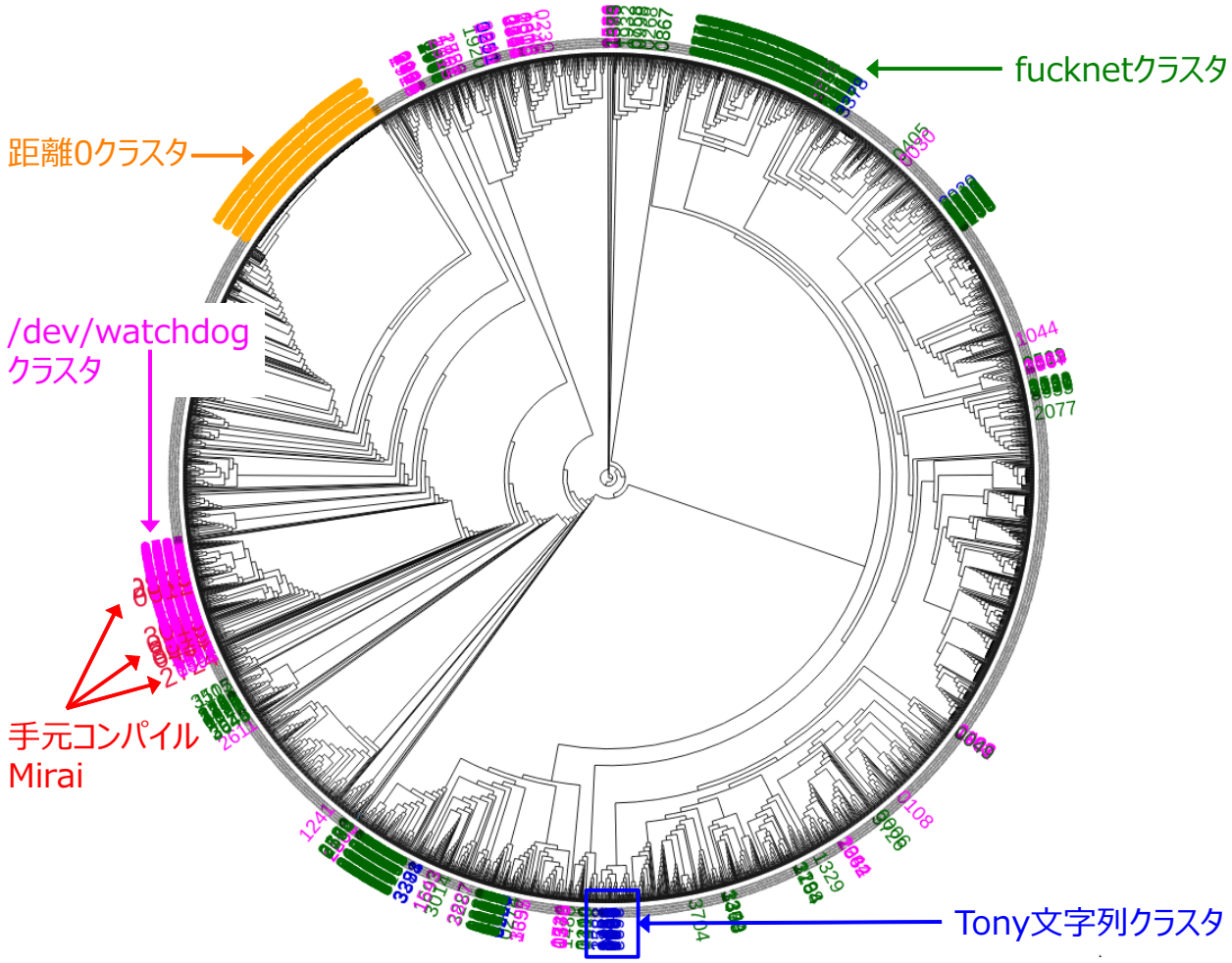

図 3 可視化結果

```
rule Gafgyt_Botnet_oh : MALW
{
meta:
description = "Gafgyt Trojan"
author = "Joan Soriano / @joanbtl"
date = "2017-05-025"
version = "1.0"
MD5 = "97f5edac312de349495cb4afd119d2a5"
SHA1 = "916a51f2139f11e8be6247418dca6c41591f4557"

    strings:
            $s1 = "busyboxterrorist"
            $s2 = "BOGOMIPS"
            $s3 = "124.105.97.%d"
            $s4 = "fucknet"
    condition:
            $s1 and $s2 and $s3 and $s4
}
```

上記から分かるように，BASHLITE 用の YARA rule では，`fucknet` という文字列を使用している．

BASHLITE である可能性の検体が提案手法によってどのように分類されているかを検証するために，`fucknet` を含む検体の分類木上での位置を可視化した．図 3 の緑文字の ID が `fucknet` を含んだ検体を意味している．`fucknet` を含んだ検体は，4009 検体中 394 検体であった．いくつかのクラスタに分割されているものの，`fucknet` を含んだ検体同士が近い位置に分類されていることが分かる．また，図 3 における「`fucknet` クラスタ」と記載している大きなクラスタには MIPS，x64，x86，ARM，Motorola m68k など様々な CPU アーキテクチャを含んでいることが確認できた．さらに，2 検体のみ`/dev/watchdog` と `fucknet` の両方の文字列を含む検体が存在した．

Tony 文字列

提案手法の評価を進める過程で，偶然的にとある文字列を含む 25 検体を見つけた．その文字列を本稿では Tony 文字列と呼ぶ．Tony 文字列について調べていたところ，いくつかの検索エンジンではヒットが 0 であった．しかしながら，とある検索エンジンで 1 ページだけ Pastebin 関連のページがヒットした．そのページは大阪大学のネットワークでは情報セキュリティ上の問題があると判定されてアクセスできなかった．とあるネットワークを介してアクセスしたところ，Tony 文字列を含んだマルウェアのソースコードらしきものを発見することができた．

図 3 の青文字の ID が Tony 文字列を含む検体の分類木上

での位置である．大まかに 6 つのクラスタに分かれており，1 番大きいクラスタが 12 検体，2 番目に大きいクラスタが 4 検体，3 番目に大きいクラスタが 3 検体，残り 3 クラスタが 1 検体であった．1 番大きいクラスタには PowerPC，x86，MIPS，SPARC など異なる CPU アーキテクチャの検体が含まれていた．

マルウェア間の距離が近い最大のクラスタ

提案手法の評価を進める過程で，検体間の距離が 0 の 263 検体から構成される巨大なクラスタが 1 つ生成された．図 3 の「距離 0 のクラスタ」と記載された黄文字の ID が本巨大クラスタに含まれている検体である．本クラスタの検体は，文字列を全く含まないか，ほとんど文字列を含まない検体であった．パッカーを用いて圧縮・暗号化された検体であると考えられる．

4.3 計算コスト

3 節の提案手法の計算コストを評価することを目的として，マルウェア検体数を変えた場合の処理時間の評価を行った．評価では，CPU が動作周波数 2.60 GHz の Intel Core i7-9750H，RAM が 32 GB の Lenovo ThinkPad X1 Extreme を用いた．評価に用いた 4009 検体では，検体毎に含んでいる文字列数 $|\boldsymbol{S}_i|$ の平均は約 734，ベクトルの次元数と一致する $|\boldsymbol{T}|$ は 52913 であった．

図 4 にマルウェア検体数を変えた場合の文字列抽出 (Algorithm 1) とベクトル生成 (Algorithm 2，Algorithm 3) に要する処理時間を示す．x 軸がマルウェア検体数，y 軸が処理時間 (秒) である．図 4 より，処理時間は検体数に対して一次関数的に増加していることが分かる．3.3 節で述べたように，計算量が $O(|\boldsymbol{S}_1|+|\boldsymbol{S}_2|+\cdots+|\boldsymbol{S}_N|)$ となるからだと考えられる．また，マルウェア検体数が 4000 のときの処理時間は約 160 秒であった．

図 5 にマルウェア検体数を変えた場合の階層的クラスタリング (Algorithm 4) に要する処理時間を示す．x 軸がマルウェア検体数，y 軸が処理時間 (秒) である．図 5 より，処理時間は検体数に対して二次関数的に増加していることが分かる．3.4 節で述べたように，計算量が $O(N^2)$ となるからだと考えられる．また，マルウェア検体数が 4000 のときの処理時間は約 303 秒であった．

5. おわりに

本稿では，IoT マルウェアの分類方法として，文字列を用いた階層的クラスタリングアルゴリズムを提案した．横浜国立大学吉岡研究室が提供している IoTPOT malware binary dataset A を提案手法で分類・可視化したところ，共通のマルウェアと思われるものが近くにクラスタリングされることが確認できた．また，特定のマルウェアに

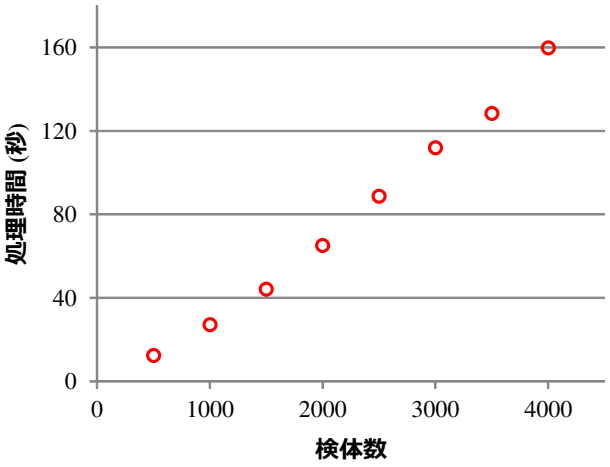

図 4 マルウェア検体数を変えた場合の文字列抽出とベクトル生成に要する処理時間

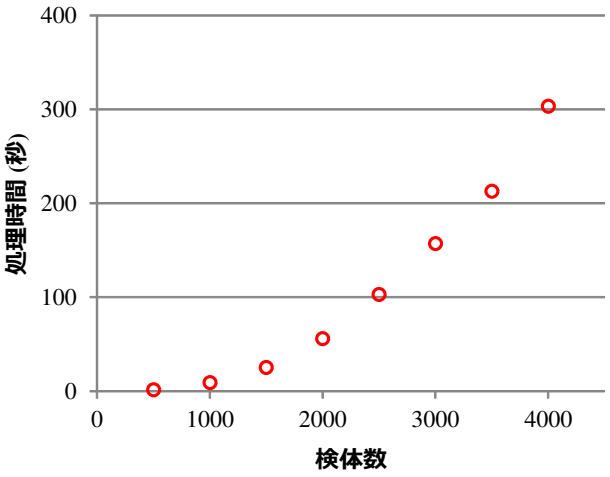

図 5 マルウェア検体数を変えた場合の文字列抽出とベクトル生成に要する処理時間

含まれていると思われる Tony 文字列も見つけることができた．現在，strip 処理済みの検体をどのように扱うか，VirusTotal の出力結果との詳細な比較，パック処理された検体の扱い方など改良方法を模索している．特にパック処理された検体に関しては，2021 年現在でもまだ数は少ないことが観測されているが，今後 PC 向けのマルウェアのようにパック処理されたマルウェアが増えていくことが予想される [25].

謝辞 本研究は JST さきがけ (JPMJPR2032，JPMJPR1938)，NTT の支援の下で行った．

参考文献

[1] Kolias, C., Kambourakis, G., Stavrou, A. and Voas, J.: DDoS in the IoT: Mirai and Other Botnets, *Computer*, Vol. 50, No. 7, pp. 80–84 (2017).
[2] Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J., Durumeric, Z., Hal-

derman, J. A., Invernizzi, L., Kallitsis, M., Kumar, D., Lever, C., Ma, Z., Mason, J., Menscher, D., Seaman, C., Sullivan, N., Thomas, K. and Zhou, Y.: Understanding the Mirai Botnet, *Proceedings of the USENIX Security Symposium (USENIX Security'17)*, Vancouver, BC, pp. 1093–1110 (2017).

[3] VirusTotal: YARA: The Pattern Matching Swiss Knife, `https://github.com/VirusTotal/yara`.

[4] Alrawi, O., Lever, C., Valakuzhy, K., Court, R., Snow, K., Monrose, F. and Antonakakis, M.: The Circle of Life: A Large-Scale Study of The IoT Malware Lifecycle, *Proceedings of the USENIX Security Symposium (USENIX Security'21)*, pp. 3505–3522 (2021).

[5] Pa, Y. M. P., Suzuki, S., Yoshioka, K., Matsumoto, T., Kasama, T. and Rossow, C.: IoTPOT: Analysing the Rise of IoT Compromises, *Proceedings of the USENIX Workshop on Offensive Technologies (USENIX WOOT'15)* (2015).

[6] Yokohama National University: IoT Honeypot: Malware Binaries Dataset A, `https://sec.ynu.codes/iot/available_datasets#1`.

[7] Zhou, Y. and Jiang, X.: Dissecting Android Malware: Characterization and Evolution, *Proceedings of the IEEE Symposium on Security and Privacy (IEEE SP'12)*, pp. 95–109 (2012).

[8] : HYDRA: A Multimodal Deep Learning Framework for Malware Classification, *Computers & Security*, Vol. 95.

[9] Jang, J., Brumley, D. and Venkataraman, S.: BitShred: Feature Hashing Malware for Scalable Triage and Semantic Analysis, *Proceedings of the ACM Conference on Computer and Communications Security (ACM CCS'11)*, pp. 309–320 (2011).

[10] Shi, Q., Petterson, J., Dror, G., Langford, J., Smola, A. and Vishwanathan, S.: Hash Kernels for Structured Data, *Journal of Machine Learning Research*, Vol. 10, pp. 2615–2637 (2009).

[11] Weinberger, K., Dasgupta, A., Langford, J., Smola, A. and Attenberg, J.: Feature Hashing for Large Scale Multitask Learning, *Proceedings of the Annual International Conference on Machine Learning (ACM ICML'09)*, pp. 1113–1120 (2009).

[12] Atzeni, A., Díaz, F., Marcelli, A., Sánchez, A., Squillero, G. and Tonda, A.: Countering Android Malware: A Scalable Semi-Supervised Approach for Family-Signature Generation, *IEEE Access*, Vol. 6, pp. 59540–59556 (2018).

[13] Li, Y., Sundaramurthy, S. C., Bardas, A. G., Ou, X., Caragea, D., Hu, X. and Jang, J.: Experimental Study of Fuzzy Hashing in Malware Clustering Analysis, *8th Workshop on Cyber Security Experimentation and Test (CSET'15)* (2015).

[14] Naik, N., Jenkins, P., Gillett, J., Mouratidis, H., Naik, K. and Song, J.: Lockout-Tagout Ransomware: A Detection Method for Ransomware using Fuzzy Hashing and Clustering, *Proceedings of the IEEE Symposium Series on Computational Intelligence (IEEE SSCI'19)*, pp. 641–648 (2019).

[15] JesseKornblum: Identifying Almost Identical Files Using Context Triggered Piecewise Hashing, *Digital Investigation*, Vol. 3, pp. 91–97 (2006).

[16] Dovom, E. M., Azmoodeh, A., Dehghantanha, A., Newton, D. E., Parizi, R. M. and Karimipour, H.: Fuzzy Pattern Tree for Edge Malware Detection and Categorization in IoT, *Journal of Systems Architecture*, Vol. 97, pp. 1–7 (2019).

[17] 川添玲雄，韓　燦洙，伊沢亮一，高橋健志，竹内純一：IoT マルウェアの機能差分調査手法の改善及びクラスタに対する分析，電子情報通信学会技術研究報告 (ICSS)，pp. 78–83 (2021).

[18] Alhanahnah, M., Lin, Q., Yan, Q., Zhang, N. and Chen, Z.: Efficient Signature Generation for Classifying Cross-Architecture IoT Malware, *Proceedings of the IEEE Conference on Communications and Network Security (IEEE CNS'18)*, pp. 1–9 (2018).

[19] Ngo, Q. D., Nguyen, H. T., Le, V. H. and Nguyen, D. H.: A Survey of IoT Malware and Detection Methods based on Static Features, *ICT Express*, Vol. 6, No. 4, pp. 280–286 (2020).

[20] 何　天祥，韓　燦洙，伊沢亮一，高橋健志，来嶋秀治，竹内純一：高速な系統樹構成アルゴリズムにおけるスケーラブルなクラスタリング評価，電子情報通信学会技術研究報告 (ICSS)，pp. 72–77 (2021).

[21] Hex Rays: IDA Pro: A Powerful Disassembler and a Versatile Debugger, `https://hex-rays.com/ida-pro/`.

[22] Hispasec Sistemas: VirusTotal: Analyze Suspicious Files and URLs to Detect Types of Malware, Automatically Share Them with the Security Community, `https://www.virustotal.com`.

[23] Marzano, A., Alexander, D., Fonseca, O., Fazzion, E., Hoepers, C., Steding-Jessen, K., Chaves, M. H. P. C., Ítalo Cunha, Guedes, D. and Meira, W.: The Evolution of Bashlite and Mirai IoT Botnets, *Proceedings of the IEEE Symposium on Computers and Communications (IEEE ISCC'18)* (2018).

[24] Murtagh, F.: *Multidimensional Clustering Algorithms, Lectures in Computational Statistics (COMPSTAT LECTURES 4)*, Vol. 4, chapter 3.7, pp. 86–87, Physica-Verlag (1985).

[25] Isawa, R., Ban, T., Tie, Y., Yoshioka, K. and Inoue, D.: Evaluating Disassembly-Code Based Similarity between IoT Malware Samples, *Asia Joint Conference on Information Security (AsiaJCIS)*, pp. 1–6 (2018).