2015年度 修士論文

階層成長型自己組織化マップによる
マルウェアのクラスタリング

提出日：2016年2月1日

指導：後藤滋樹教授

早稲田大学 基幹理工学研究科 情報理工・情報通信専攻
学籍番号：5114F002-2

青木　一樹

目次

図一覧

表一覧

第 1 章

序論

1.1 研究の背景

マルウェア (コンピュータウイルスなどの不正なソフトウェア) の検出数が増加の一途をたどっている．そのマルウェアの多くが既存のマルウェアの亜種であることがわかっており, 4 秒に 1 つのペースで新しい亜種が発見されている [1]．この問題の主な要因はマルウェアの自動作成ツールの拡大により容易にマルウェアの亜種を生み出せるようになったことである．また, マルウェアの亜種の中にはアンチウイルスソフトによる検知の回避や難読化機能を有したものがある．本来, ソフトウェアの難読化技術の普及は知的財産権を保護するために重要である．しかし, その一方で検知困難なマルウェアの作成に悪用されている．このようなマルウェアの増加・難読化に対抗するために, より短時間で難読化にも対抗できるマルウェア解析の研究が進んでいる．マルウェアの解析技術として動的解析と静的解析の 2 つがある．動的解析とはマルウェアを動作させ, 感染活動を明らかにする方法である．静的解析とは逆アセンブラやデバッガを使用しマルウェアを分析する方法である．動的解析と静的解析にはそれぞれ長所と短所がある [2].

動的解析

- 長所：短時間で感染動作が調査でき, 要求される技術のレベルが比較的低く, 一部の難読化に対応できる
- 短所：デバッガやエミュレータの環境, OS 環境を検査し, 環境に応じて動作を変えるマルウェアの解析が困難である．

静的解析

- 長所：マルウェアを動作させないので感染することがなく, 詳細な動作解析ができる
- 短所：コードを分析するのに時間がかかり, 要求される技術のレベルが比較的高い

マルウェアの増加・難読化の問題には, 短時間でマルウェアの感染活動を調査でき, 一部の難読化に対応できる動的解析が有効である．また, 短時間でマルウェア解析を行うために, クラスタリングを用いる手法がある．クラスタリングとは, データの集合を類似性の高いデータの部分集合 (クラスタ) に分けるデータ解析の手法である．解析したいマルウェアがどのようなマルウェアと類似性が高いかをあらかじめ確認することで, より効率的なマルウェアの解析を行うことが可能となる．マルウェアの亜種には既存のマルウェアと同じ特徴を持つものが多いため, クラスタリングはマルウェア解析において有効な手法である．以上のような技術が日々生み出されており, マルウェアの脅威に対抗する動きが広まっている．今後も続くマルウェアの増加に備え, 更なる研究が求められている．

1.2　研究の目的

本研究では, 動的解析ログから取得できる情報を用いた, マルウェアのクラスタリングを提案する．具体的には, Cuckoo Sandbox [3] を用いて収集した動的解析ログに含まれている API コール情報を特徴量とした, マルウェアのクラスタリングを行う．API コール情報とは, 検体実行時のプロセス ID, API の関数名, 引数, 返り値などの API に関係する情報である．本研究では, API コール情報から API の関数名のみを抽出して使用する．API の関数名のみを使用することで, 特徴量の種類が爆発的に増えることがなくなり, クラスタリングの時間を削減することが狙いである．また, マルウェアのクラスタリングには階層成長型自己組織化マップ (GHSOM, Growing Hierarchical Self Organizing Map) を用いる．階層成長型自己組織化マップ [4] は, 入力データに対して自動的にクラスタリングの規模を調整するため, 入力データの大小に関わらないクラスタリングを行うことができる．本研究は, API の関数名を特徴量とした階層成長型自己組織化マップによるマルウェアのクラスタリングの有効性を示す．

1.3　本論文の構成

本論文は以下の章により構成される.

第 1 章 序論

本論文の概要を述べる.

第 2 章 GHSOM

GHSOM について紹介する.

第 3 章 Cuckoo Sandbox

Cuckoo Sandbox について紹介する.

第 4 章 関連研究

関連研究を紹介する.

第 5 章 提案手法

本研究の提案手法を説明する.

第 6 章 実験

提案手法を用いた実験について述べる.

第 7 章 結論

本論文の結論を述べ, 今後の課題を示す.

第 2 章

階層成長型自己組織化マップ

本研究の提案手法では, 階層成長型自己組織化マップ (Growing Hierarchical Self-Organizing Map, 以下 GHSOM と略記) を利用する．GHSOM は自己組織化マップ (Self-Organizing Map, 以下 SOM と略記) を階層化した手法である．本章では, 2.1 節で SOM の概要, 2.2 節で SOM のアルゴリズム, 2.3 節で SOM の問題点について解説する．そして, 2.4 節で GHSOM の概要, 2.5 節で GHSOM のアルゴリズム, 2.3 節で GHSOM の特徴について説明する．

2.1 自己組織化マップの概要

SOM は, T. Kohonen [5] により提案された教師なしのニューラルネットワークアルゴリズムで, 高次元データを 2 次元平面上へ非線形写像するデータ解析方法である．ニューラルネットワークモデルの中ではフィードフォワード型に分類される (Feedforward Neural Network)．SOM は, 入力層と出力層により構成された 2 層のニューラルネットワークである．出力層は競合層とも呼ばれている．

今, 入力層には分析対象となる個体 $j\,(j = 1, 2, \cdots, n)$ の特徴ベクトルを $\boldsymbol{x}_j\,(x_{j1}, x_{j2}, \cdots, x_{jp})$, 出力層には $k\,(i = 1, 2, \cdots, k)$ 個のユニット $\boldsymbol{m}_i$ があるとする．図 2.1 の (a) で示すように, 出力層における任意の 1 つのユニットは, 入力層における特徴ベクトルのすべての変数とリンクしている．初期段階では乱数により各変数との間に図 2.1 の (b) に示すように重み $\boldsymbol{m}_i\,(m_{i1}, m_{i1}, \cdots, m_{ip})$ が付けられている．

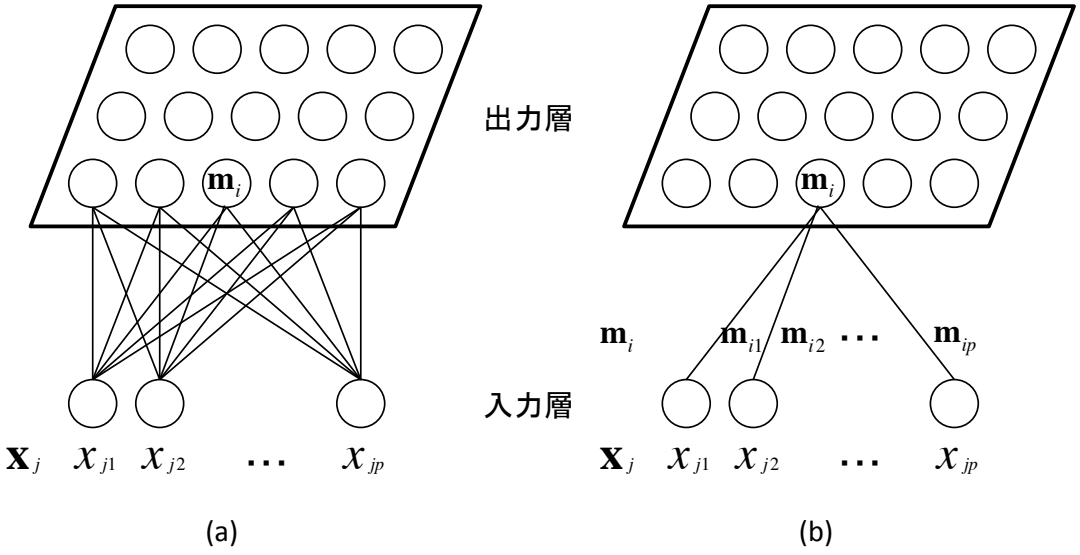

図 2.1: 自己組織化マップの基本構造

2.2　SOM のアルゴリズム

SOM のアルゴリズムについて概説する [6].

1. 下記の式 (2.1) のように入力 $\boldsymbol{x}_j$ と出力層のすべてのユニットの距離を比べ, 最も距離が近いユニット $\boldsymbol{m}_c$ を探し出し, そのユニットを勝者とする.

$$\|\boldsymbol{x}_j - \boldsymbol{m}_c\| = min_i \{\|\boldsymbol{x}_j - \boldsymbol{m}_i\|\} \tag{2.1}$$

2. 探し出したユニットおよびその近傍のユニットの重みベクトル $\boldsymbol{m}_i$ を更新する．更新は下記の式 (2.2) によって行われる.

$$\boldsymbol{m}_i(t+1) = \begin{cases} \boldsymbol{m}_i(t) + h_{ci}(t)[\boldsymbol{x}_j(t) - \boldsymbol{m}_i(t)] & (i \in N_c) \\ \boldsymbol{m}_i(t) & (i \notin N_c) \end{cases} \tag{2.2}$$

$$h_{ci}(t) = \alpha(t) exp\left(-\frac{\|r_c - r_i\|^2}{2\sigma^2(t)}\right)$$

式の中の $h_{ci}(t)$ は近傍関数であり, ユニット c とその近傍のユニット i の近さによって x_j の影響を調整する．式 $h_{ci}(t)$ の中の $\alpha(t)$ は学習率の係数であり, r_c と r_i はユニット c と i の 2 次元上の座標位置ベクトルである．$\sigma^2(t)$ はユニット c の近傍領域 N_c の半径を調整

する関数である．$\sigma(t)$, $\sigma^2(t)$ は学習回数 (あるいは時間) を変数とする単調減少関数である．学習回数を変数とする最も簡単な単調減少関数は $1 - \frac{t}{T}$ である．この t は学習回数 (あるいは時間 1,2,3,⋯,) T は事前に設定した学習の総回数である．

3. すべての入力の特徴ベクトル $\boldsymbol{x}_j\ (j = 1, 2, \cdots, n)$　に対して 1〜2 を繰り返し実行する．

SOM は上記アルゴリズムにより, 多次元空間上の分類対象を 2 次平面に射影する．SOM の結果の出力画面のユニットは, 格子状 (正方形) , 蜂の巣状 (六角形) などが提案されているが, 蜂の巣状が多く用いられており, 本研究においても蜂の巣状の画面を用いる．蜂の巣状というのは, 文字通り蜂の巣のように正六角形のユニットを並べ, 出力層の画面を構成する．出力層の画面は, 上述のアルゴリズムにより, 似ているもの同士を同じユニット, あるいはその近辺のユニットに配置する．

2.3　SOM の特徴

SOM の利点と欠点を下記にあげる．

- 利点

1. 入力データの数を圧縮して, 解析の効率を上げる．
   SOM の各ユニットは, 入力データ (学習したデータ) と同じ形式の参照ベクトルを持つ．これは, 入力データの数に対してユニット数は少なくなる場合に, 一種のデータ圧縮として解釈できる．つまり, ユニットごとの解析をすることで, 入力データをそのまま解析するよりも作業数を減らすことができる．
2. マップ上に表現するユニット数を調節できる．
   ユニット数を調節することによって, 特徴を把握しやすいレベルのクラスタリングが行える．
3. マップ上で隣接するユニット同士は類似する [7].
   特徴が類似したユニット同士が集合することにより, 複数のユニットをまとめた集合をクラスタとして分類が可能である．
4. 出力層の各ユニットが参照ベクトルを持つので, 各要素ごとに可視化可能である [8].
   各要素ごとに可視化することにより, マップの各領域の特徴を調査したり, 変数間の複雑な (非線形な) 関係を分析することができる．

- 欠点

  SOM では人間の経験則から, データに適したマップのサイズをあらかじめ決める必要がある．そのため巨大な入力データを扱う場合などに, 入力データに適したユニット数を調整することが困難になる．

2.4　階層成長型自己組織化マップの概要

GHSOM は, A. Rauber [9] により提案された SOM の階層化手法であり, マップサイズや階層構造を自動で決定する．GHSOM の概要図を図 2.2 に示す．

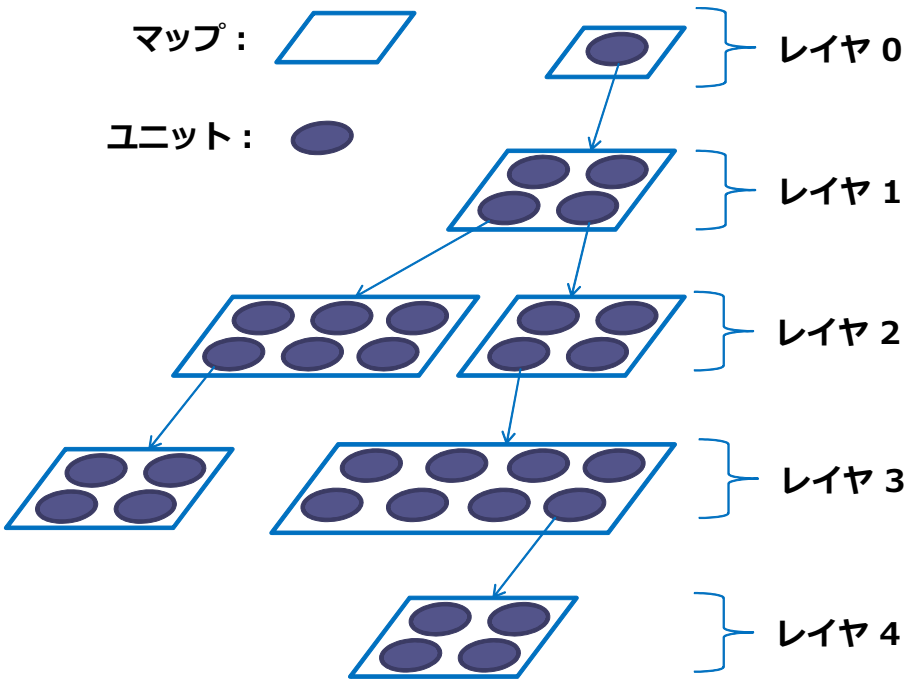

図 2.2: GHSOM の概要図

GHSOM では図 2.2 のように SOM がツリー構造を形成しており, 各入力データはレイヤ 0 以外のユニットに分類される．GHSOM のクラスタリングは適切なマップサイズと階層構造をある規則に従い決定することで行われる．マップサイズと階層構造を決定する規則をそれぞれ成長規則, 階層化規則と呼ぶ．この規則により, 入力データに適した全体マップが構築できる．次に GHSOM のアルゴリズムについて述べる．

2.5　GHSOM のアルゴリズム

GHSOM は図 2.2 のように複数のマップにより構成され, 各レイヤの各マップで独立して SOM による学習が行われる．レイヤ 0 では SOM による学習とマップの成長は行われず, 単一のユニットのみ存在する．また, レイヤ 0 において全入力データの平均誤差を求め, これを GHSOM の成長規則で用いるパラメータの初期値, 階層化規則の条件として用いる．レイヤ 0 はこの初期値を求めるために存在する仮想レイヤである．レイヤ 1 以降は, あらかじめ決めたマップサイズ (今回の実験では 2×2) から始まり, 成長規則・階層化規則に従いながらマップの成長・階層化を行う．ここから, マップの成長及び階層化の手順について説明する．まず, 全入力データの平均を取ったベクトルである $\boldsymbol{m}_0$ を以下の式 (2.3) で表す．ここで, d は入力データの個数である．

$$\boldsymbol{m}_0 = \frac{1}{d} \cdot \sum_{i=1}^{d} \boldsymbol{x}_i \tag{2.3}$$

次に, 式 (2.4) に従いレイヤ 0 の平均誤差である mqe_0 を求める．

$$mqe_0 = \frac{1}{d} \cdot \sum_{i=1}^{d} ||\boldsymbol{x}_i - \boldsymbol{m}_0|| \tag{2.4}$$

レイヤ 1 以降のユニット i に関しては章 2.2 で述べた SOM の重みベクトル $\boldsymbol{m}_i$ を用いて平均誤差 mqe_i を表す．各ユニットに分類された入力データの個数を d_i としたとき, mqe_i を以下の式 (2.5) で表す．

$$mqe_i = \frac{1}{d_i} \cdot \sum_{j=1}^{d_i} ||\boldsymbol{x}_j - \boldsymbol{m}_i|| \tag{2.5}$$

mqe_i が小さいユニットは, そのユニットとそのユニットに分類されたデータの類似性が高いことを示している．一方, mqe_i が大きいユニットは, そのユニットとそのユニットに分類されたデータの類似性が低いことを示している．次に, 成長規則について説明する．

2.5.1　成長規則

あるマップ m 上にある全ユニットの mqe_i の合計 MQE_m を以下の式 (2.6) で表す．u_m はマップ上にあるユニットの数である．

$$MQE_m = \frac{1}{u_m} \cdot \sum_{i=1}^{u_m} mqe_i \tag{2.6}$$

各マップは以下の式 (2.7) を満たす場合にユニットの挿入を行い成長する．T_m は閾値であり, この値で成長の頻度を制御する．mqe_u はマップ m の上位レイヤにある階層化元のユニットの平均誤差である．

$$MQE_m \geq T_m \cdot mqe_u,\ (0 \leq T_m \leq 1) \tag{2.7}$$

式 (2.7) を満たす場合, 以下の方法でユニットを挿入してマップを成長させる．ここから, マップの成長の方法について説明する．まず, 最も平均誤差が大きいユニット e を式 (2.8) で表す．d_i は各ユニットに分類された入力データの個数である．

$$e = \arg\max_i \left(\sum_{j=1}^{d_i} ||\boldsymbol{x}_j - \boldsymbol{m}_i|| \right) \tag{2.8}$$

次に, ユニット e と隣接しているユニットの中で, 式 (2.9) を満たすユニットを d とする．m_e はユニット e の重みベクトル, m_i はユニット d の重みベクトルを表す．

$$d = \arg\max_i \left(||\boldsymbol{m}_e - \boldsymbol{m}_i|| \right) \tag{2.9}$$

これまでに求めたユニット e とユニット d の間に新しいユニットを挿入する．マップの行に挿入する場合は図 2.3 のようになり, 列に挿入する場合は図 2.4 のようになる．

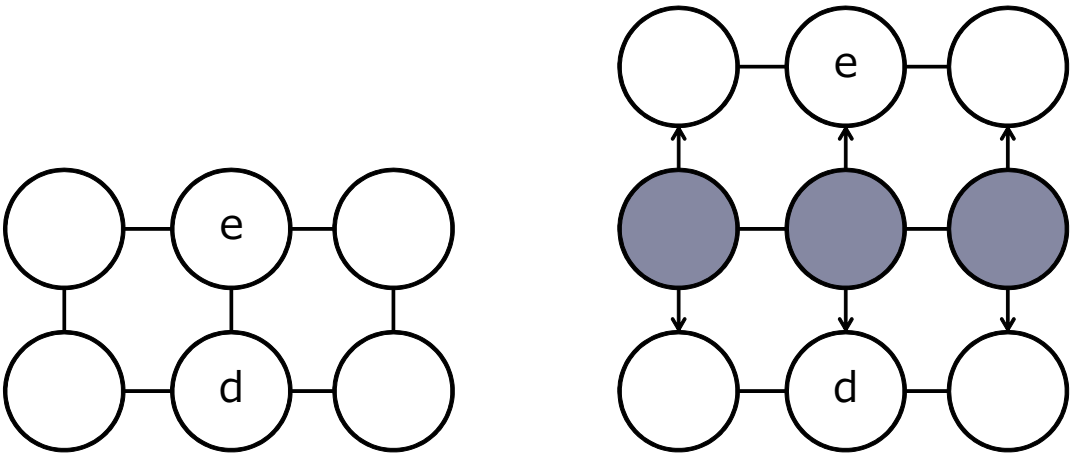

図 2.3: マップの行へのユニットの挿入

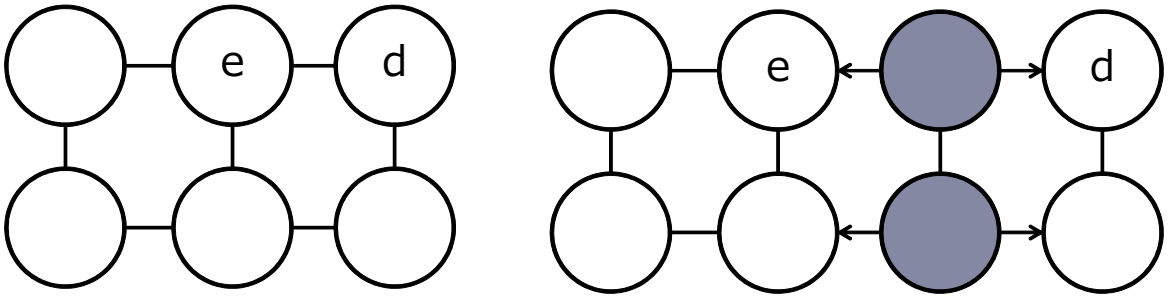

図 2.4: マップの列へのユニットの挿入

挿入する各ユニットの重みベクトルは, 挟み込む 2 つのユニットの平均値となる．その他の各ユニットの重みベクトルはそのままで再び SOM アルゴリズムによる学習を行い, 式 (2.7) を満たさなくなるまで繰り返す．以上の規則で各マップのマップサイズを入力データに最適な大きさにする．各マップの成長が終った時点で, 階層化を行うかどうかの判定を行う．次に階層化規則について説明する．

2.5.2　階層化規則

各ユニットは以下の式 (2.10) を満たす場合に以下の階層化を行う．T_u は閾値であり, この値で階層化の頻度を制御する．

$$mqe_i > T_u \cdot mqe_0,\ (0 \leq T_u \leq 1) \tag{2.10}$$

式 (2.10) は各マップ中のユニットの中から, レイヤ 0 の平均誤差に Tu を掛け合わせた値より, 大きな平均誤差を持つユニットに対して階層化を行うことを示している．階層化によりできる下層レイヤのマップの初期サイズはあらかじめ決めたサイズとなる．以上が GHSOM のアルゴリズムの説明である．

2.6　GHSOM の特徴

GHSOM は成長規則, 階層化規則に従い自動で入力データに適したマップのサイズを決定することができる．そのため, 2.3 節で述べた SOM の利点を残したまま, 大量の入力データを扱う場合にも入力データに適したユニット数の調整を自動で行うことができる．

第 3 章

Cuckoo Sandbox

本章では, 3.1 節で Cuckoo Sandbox の概要, 3.2 節で本研究で使用するデータについて解説する.

3.1 Cuckoo Sandbox の概要

Cuckoo Sandbox は動的解析を自動化するオープンソフトウェアである．実行ファイルを Cuckoo Sandbox 上で実行させることで動的解析ログを json ファイル形式で得ることができる．また, 閉鎖された仮想環境で実行させるため, 実環境のプロセスやシステムファイルへの不正アクセスを防ぐことができる．Cuckoo Sandbox により取得できる具体的なデータ項目とその内容について表 3.1 に示す [10].

表 3.1: Cuckoo Sandbox により取得できる具体的なデータ項目

項目	内容
info	解析の開始, 終了時刻, id など
yara	yara (OSS のマルウェア検知・分類エンジン) の標準ルールとの照合結果
signatures	ユーザー定義シグニチャとの照合結果
virustotal	VirusTotal の検査履歴との照合結果
static	検体のファイル情報 (インポート API, セクション構造等)
dropped	検体が実行時に生成したファイル
behavior	検体実行時の API ログ (PID, TID, API の関数名, 引数, 返り値等)
processstree	検体実行時のプロセスツリー (親子関係)
summary	検体が実行時にアクセスしたファイル, レジストリ等の概要情報
target	解析対象検体のファイル情報 (ハッシュ値等)
debug	検体解析時の Cuckoo Sandbox のデバッグログ
strings	検体中に含まれる文字列情報
network	検体が実行時に行った通信の概要情報

3.2　使用するデータ

本研究では, Cuckoo Sandbox から取得できるデータの中から本研究で使用データを抽出する．特徴量として API の関数名, マルウェア名を表すラベルとして Kaspersky の検知結果の 2 つのデータを用いる．それぞれのデータについて 3.2.1 節と 3.2.2 節で説明する．

3.2.1　API の関数名

Cuckoo Sandbox で収集した動的解析ログから begavior に含まれる API の関数名のみを抽出する．behavior の構造例を図 3.1 に示す．


```
"behavior": {
  "processes": [
    {
      "parent_id": "324",
      "process_name": "8158df23e651708c03a2cf0929d306e3.exe",
      "process_id": "336",
      "first_seen": "2013-10-26 07:44:56,830",
      "calls": [
        {
          "category": "registry",
          "status": false,
          "return": "0x00000002",
          "timestamp": "2013-10-26 07:44:56,840",
          "thread_id": "616",
          "repeated": 0,
          "api": "RegOpenKeyExW",
          "arguments": [
            {
              "name": "Registry",
              "value": "0x80000002“
            },
            {
              "name": "SubKey",
              "value": "System¥¥WPA¥¥Starter“
            },
            {
              "name": "Handle",
              "value": "0x00000000“
            }
          ]
        },
        {
          "category": "registry",
          "status": true,
          "return": "0x00000000",
          "timestamp": "2013-10-26 07:44:56,850",
          "thread_id": "616",
          "repeated": 0,
          "api": "LdrLoadDll",
          "arguments": [
            {
              "name": "Flags",
              "value": "457048"
            },
            {
              "name": "FileName",
              "value": "NETMSG"
            },
            {
              "name": "BaseAddress",
              "value": "0x71a70000"
            }
          ]
        },
                    ⋮
      ]
    }
  ]
}
```


図 3.1: behavior の構造例

図 3.1 のような json ファイル形式のデータに対して behavior, processes, calls, api の順番でキーをたどることで API の関数名のみを抽出することができる．

3.2.2　Kaspersky の検知結果

Cuckoo Sandbox で収集した動的解析ログから virustotal に含まれる VirusTotal [11] の検知結果を用いる．virustotal の構造例を図 3.2 に示す．


```
"virustotal": {
    "scan_id": "cedcaf79c28f8c760d28a91724d19a7e375971a17bab776bb9bb5232ed73d6ec-1368986773",
    "sha1": "e94d91a454e783b604ae2c2f0a980c085a1a2006",
    "resource": "8158df23e651708c03a2cf0929d306e3",
    "response_code": 1,
    "scan_date": "2013-05-19 18:06:13",
    “permalink”: “https://www.virustotal.com/file/cedcaf79c28f8c760d28a91724d19a7e375971a17bab･･･",
    "verbose_msg": "Scan finished, scan information embedded in this object",
    "sha256": "cedcaf79c28f8c760d28a91724d19a7e375971a17bab776bb9bb5232ed73d6ec",
    "positives": 0,
    "total": 47,
    "md5": "8158df23e651708c03a2cf0929d306e3",
    "scans": {
    "McAfee": {
          "detected": true,
          "version": "5.400.0.1158",
          "result": "W32/Eggnog.worm.gen",
          "update": "20121122"
       },
    "Kaspersky": {
          "detected": true,
          "version": "9.0.0.837",
          "result": "P2P-Worm.Win32.Eggnog.f",
          "update": "20121122"
       },
                                                    ⋮
    }
}
```


図 3.2: virustotal の構造例

本研究では Virustotal の検知結果の中から, 最も検知率が高かった Kaspersky [12] の検査結果を使用する．この検知結果は API と GHSOM による分類の有効性を検証するために用いる．図 3.2 ような json ファイル形式のデータに対して virustotal, Kaspersky, result の順番でキーをたどることで Kaspersky の検知結果を抽出することができる．

続いて, Kaspersky の検知結果から亜種を表す部分を取り除き, 亜種を集約してマルウェアの種別を行う．Kaspersky の命名規則は文献 [13] より, 図 3.3 の通りとなっている．

[Prefix:]Behaviour.Platform.Name[.Variant]

図 3.3: Kaspersky の命名規則

Prefix はヒューリスティックな方法でマルウェアであると検知したこと, またはマルウェアではないアドウェア系の検体であることを表す．Behaviour, Platform, Name, Variant はそれぞれ検体の動作, OS 環境, 検体のファミリー名, マルウェアの亜種名を表す．Kaspersky の命名規則から亜種を表す Variant の部分を取り除き, マルウェアの種別に用いる新しいラベルとする．以降, このラベルを用いて論述を行う．

第 4 章

関連研究

本章では, まず 4.1 節において, 本研究に関連のあるマルウェア解析の研究を紹介する．その後 4.2 節において, 本研究の着眼点について述べる．

4.1 マルウェア解析

マルウェアには多くの種類が存在するが, 同じ機能を持ったマルウェアや同じツール, 同じ作成者によって作られたマルウェアには似た特徴があることが多い．そのため, この性質を利用してさまざまな特徴量を用いたマルウェア解析の研究が行なわれている．

史ら [14] は, マルウェアが使用する dll ファイルの数を特徴量として GHSOM を用いたクラスタリングを行っている．マルウェアの自動的なクラスタリングの結果とアンチウイルスベンダー 3 社の検知結果との比較によりその有効性を示している．

藤野ら [15] は, マルウェアが使用する API とその引き数の組み合わせを持つか否かを特徴量として, k-means アルゴリズムを使用したマルウェア分類を行っている．クラスタリングの結果や特徴量の分析によりその有効性を示している．

中村ら [16] は, マルウェアが使用した API の関数名を出現順の N-gram で抽出し, マルウェアごとの Kullback-Leibler 情報量を用いた亜種の判別を行っている．

筆者である青木 [17] は, マルウェアが使用した API の関数名を出現順の N-gram で抽出して特徴量とすることで, マルウェアの種類ごとに有効な N-gram がそれぞれ存在することを示している．

4.2　本研究の着眼点

本研究では, 史らと同様に GHSOM を用いたクラスタリングを行う．GHSOM は 2.6 節で述べたように大量の入力データがある場合にも自動でマップサイズを決めることができる．そのため, 検体数や特徴量の数に関わらず解析ができ, 柔軟性がある．また, 特徴量として API の関数名のみを用いることで特徴量が爆発的に増えることがないので, 他の研究の特徴量と組み合わせることが可能である.

第 5 章

提案手法

本章では, まず 5.1 節において, 提案手法の概要について説明する. その後 5.2 節において, 本研究で使用する特徴量について説明する.

5.1 提案手法の概要

本研究では, 動的解析ログに含まれる API コール情報を特徴量とした GHSOM によるクラスタリングを提案する. 提案手法の概要は図 5.1 の通りである.

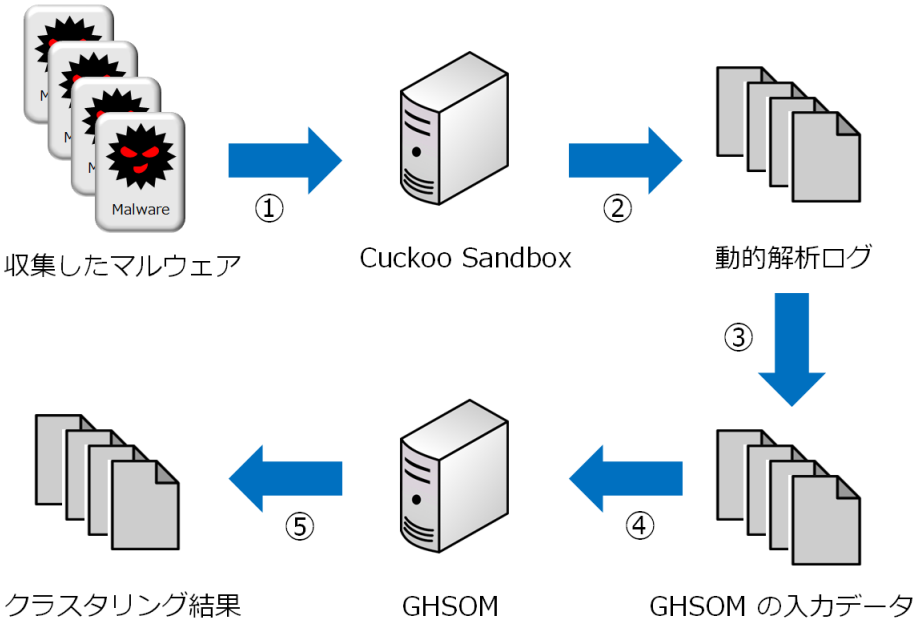

図 5.1: 本手法の概要

図 5.1 に示したように, 動的解析からクラスタリングまでを以下の手順で行う.

1. 収集したマルウェアを Cuckoo Sandbox により動的解析する.
2. 動的解析ログを取得する.
3. GHSOM の入力データを作成する.
4. GHSOM でクラスタリングを行う.
5. クラスタリングの結果を取得する.

次に, 本手法で使用する特徴量について説明する.

5.2　使用する特徴量

本手法では, 動的解析ログごとに 3.2.1 節で説明した API の関数名を抽出し, 各 API の使用回数を特徴量とする. 一つのマルウェアを実行したときに使用される API の関数名を順に並べた一例を図 5.2 に示す.

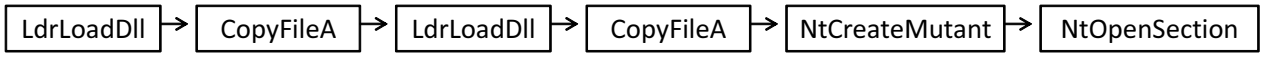

図 5.2: API の関数名を出現順に並べた一例

図 5.2 を例にすると, 本手法の特徴量は以下の表 5.1 のようになる. この特徴量を用いてクラスタリングを行う.

表 5.1: 特徴量の一例

API の関数名	出現回数
LdrLoadDll	2
CopyFileA	2
NtCreateMutant	1
NtOpenSection	1

第 6 章

提案手法の有効性評価

本章では, 実験を行った後, 本手法の有効性を評価する．まず, 6.1 節で実験準備について説明する．次に, 6.2 節で実験結果について述べる．最後に, 6.3 節で考察について述べる．

6.1 実験準備

5 章で述べた提案手法を実験データに適用し, 本手法の有効性を示すための実験を行う．実験データについて 6.1.1 節で述べ, GHSOM のパラメータについて 6.1.2 節で述べる．

6.1.1 実験データ

MWS データセット [18] の一部として, FFRI 社から提供された FFRI Dataset 2013, 2014, 2015 を実験に用いる．これらのデータセットには, マルウェア検体の動的解析ログが収録されている．それぞれ, 2012 年 9 月から 2013 年 3 月に収集された 2650 個, 2014 年 1 月から 2014 年 4 月に収集された 3000 個, 2015 年 1 月から 2015 年 4 月にかけて収集された 3000 個のマルウェア検体に対して Cuckoo Sandbox を用いて動的解析した際のログである．マルウェア検体の総集は 8650 検体, 動的解析の OS は Windows 32bit, ログファイルはすべて JSON 形式である．

本研究では, 最新版のデータセットである FFRI Dataset 2015 を対象とした実験データ 1 と FFRI Dataset 2013, 2014, 2015 のすべてを対象とした実験データ 2 を用いて実験を行う．実験データ 1 と実験データ 2 を用いた 2 パターンの実験の比較により, 入力データの大きさに適したクラスタリングが行われていることを確認する．まず, 実験データ 1 について説明する．3.2.2 節で述べた亜種を集約するためのラベルを基に, FFRI Dataset 2015 からラベル 1 種類につき

動的解析ログが 30 個以上あるマルウェアを対象としてそれぞれ 30 個を無作為に抽出する．この動的解析ログから特徴量を抽出して, 実験データ 1 とする．実験データ 1 には合計で 510 個のデータがあり, 特徴量の数 (API の種類) は 119 であった．さらに, クラスタリングの有効性を示すために, 30 個のデータを 20 個のトレーニングデータと 10 個のテストデータに分ける．次に実験データ 2 について説明する．FFRI Datase 2013, 2014, 2015 からラベル 1 種類につき動的解析ログが 50 個以上あるマルウェアを対象としてそれぞれ 50 個を無作為に抽出する．この動的解析ログから特徴量を抽出して, 実験データ 2 とする．実験データ 2 には合計で 950 個のデータがあり, 特徴量の数は 147 であった．さらに, 50 個のデータを 40 個のトレーニングデータと 10 個のテストデータに分ける．以上の手順により取得したデータの内訳, 実験データ 1 のラベル一覧, 実験データ 2 のラベル一覧をそれぞれ表 6.1, 図 6.1, 図 6.2 に示す．

表 6.1: 実験データ

	実験データ 1	実験データ 2
対象データセット	FFRI Dataset2015	FFRI Dataset2013, 2014, 2015
ラベル数	17	19
特徴量の数	119	147
トレーニングデータ総数	340	760
テストデータ総数	170	190
合計データ数	510	950

Backdoor.Win32.Androm,Backdoor.Win32.Hlux,Backdoor.Win32.Simda,
Hoax.Win32.ArchSMS,Trojan.Win32.Agent,Trojan.Win32.Inject,
Trojan.Win32.Jorik.Vobfus,Trojan.Win32.Neurevt,Trojan.Win32.Reconyc,
Trojan.Win32.VB,Trojan.Win32.VBKrypt,Trojan.Win32.Yakes,
Trojan-Downloader.Win32.Agent,Trojan-PSW.Win32.Tepfer,
Trojan-Ransom.Win32.Foreign,Trojan-Spy.Win32.Zbot,Worm.Win32.VBNA,
Worm.Win32.Vobfus,Worm.Win32.WBNA

図 6.1: 実験データ 1 のラベル一覧

Backdoor.Win32.Androm,Backdoor.Win32.Hlux,Backdoor.Win32.Simda,
Trojan-Banker.Win32.Banker,Trojan-Banker.Win32.Tinba,
Trojan-Downloader.Win32.Cabby,Trojan-Dropper.Win32.Necurs,
Trojan-Dropper.Win32.SFX,Trojan-PSW.Win32.Fareit,
Trojan-Ransom.Win32.Foreign,Trojan-Spy.Win32.Zbot,Trojan.Win32.Agent,
Trojan.Win32.Inject,Trojan.Win32.Neurevt,Trojan.Win32.Staser,
Trojan.Win32.Yakes,Worm.Win32.Ngrbot

図 6.2: 実験データ 2 のラベル一覧

6.1.2　GHSOM のパラメータ

GHSOM ではあらかじめ設定するパラメータがある．各パラメータの意味と, その値を以下に示す.

- INITIAL LEARNRATE: 0.8
  2.2 節で説明した SOM の学習係数 $\alpha(t)$ の初期値を表す.
- INITIAL X SIZE: 2
  階層化を行ったときにできる新しいマップの行数の初期値を表す.
- INITIAL Y SIZE: 2
  階層化を行ったときにできる新しいマップの列数の初期値を表す.

- NUMITERATION: 10000

  2.2 節で説明した SOM の学習回数 T を表す.

- Tm: 0.03

  2.5.1 節で説明したマップの成長頻度を制御するための閾値 T_m を表す.

- Tu: 0.001

  2.5.1 節で説明したマップの階層化頻度を制御するための閾値 T_u を表す.

6.2　実験結果

GHSOM では, マップ単位をクラスタと定義する方法と, ユニット単位をクラスタとして定義する方法がある．今回, マップ単位のクラスタをクラス, ユニット単位のクラスタをサブクラスとして, それぞれの各クラスタに ClassID, SubClassID を割り当てた．ClassID はレイヤ 1 のマップを 1 として, 上位レイヤから順番に割り当てた．SubClassID に関しては, レイヤ 1 のマップの 1 行 1 列目のユニットから順番にマルウェアが分類されているか否かを確認し, マルウェアが分類されているユニットのみに割り当てた．SubClassID についても上位レイヤから順番に割り当てている．実験データ 1 のテストデータに対して行ったクラスタリングの結果としてクラス, サブクラスに分類された各マルウェアの数をそれぞれ図 6.3, 図 6.4 に示す．同様に, 実験データ 2 のテストデータに対して行ったクラスタリングの結果としてクラス, サブクラスに分類された各マルウェアの数をそれぞれ図 6.5, 図 6.6 に示す.

各クラスタに分類されたマルウェアの数を色の濃淡で表しており, 分類された数が多い場合は色が濃くなっている．また, 図 6.3 と図 6.5 上の数字は各クラスタに分類されたマルウェアの数を表している．加えて, レイヤの区切りに実線を引いている．さらに, サブクラスの結果には同じクラス (マップ) に属するサブクラス (ユニット) がわかるように破線を引いている.

次に, クラスタリングによってできた分類器を用いてテストデータのマルウェア判別を行う. このマルウェア判別は Kaspersky のラベルを基に行うが, Kaspersky の検知結果に近づいたことを確認するためではなく, API の関数名を特徴量とした GHSOM でもクラスタリングの有効性を確認するために行う．各マルウェアのラベルごとに判別できた確率を Accuracy として, その結果を図 6.7 と図 6.8 に示す.

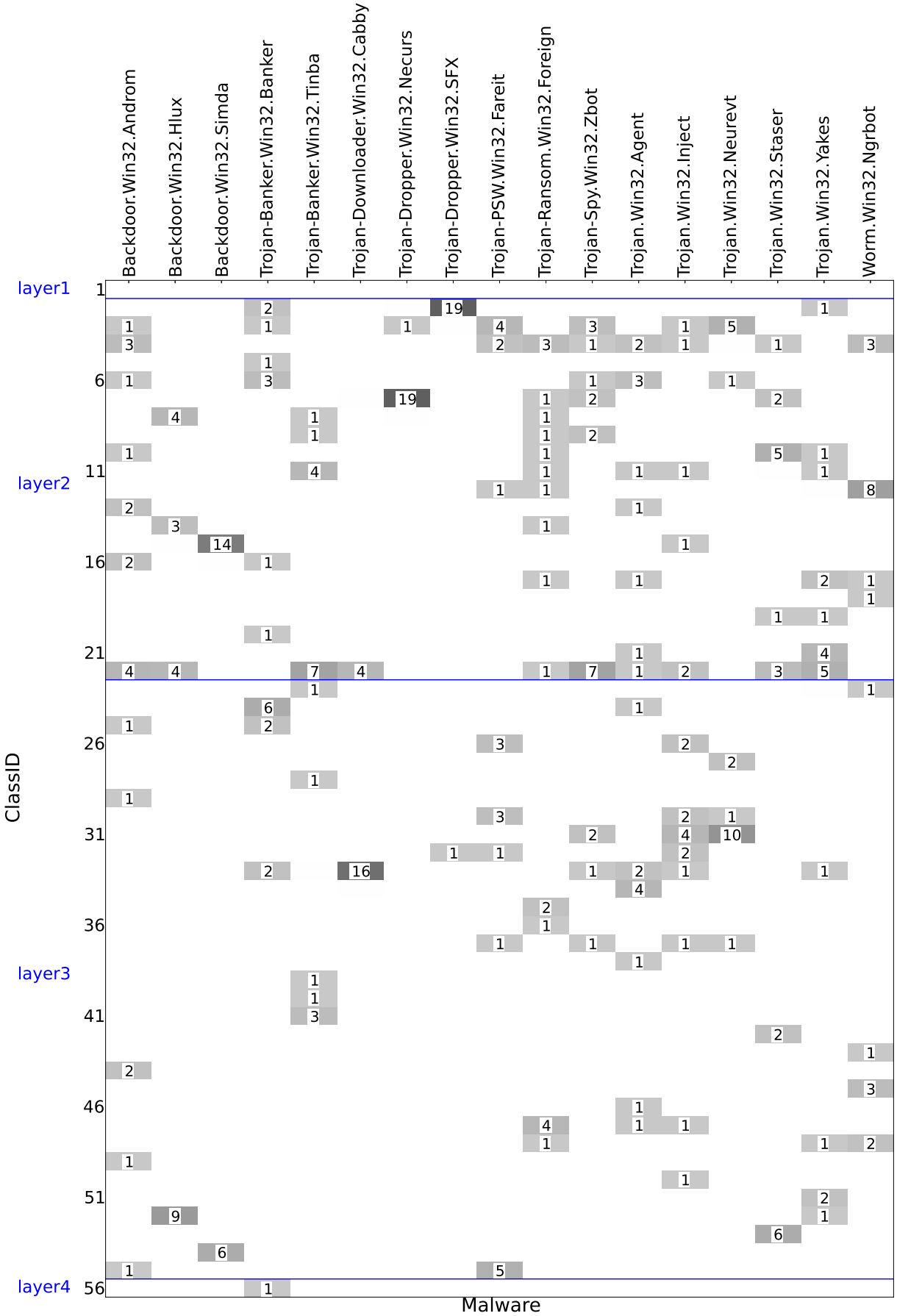

図 6.3: 各クラスに分類されたマルウェア数 (実験データ 1)

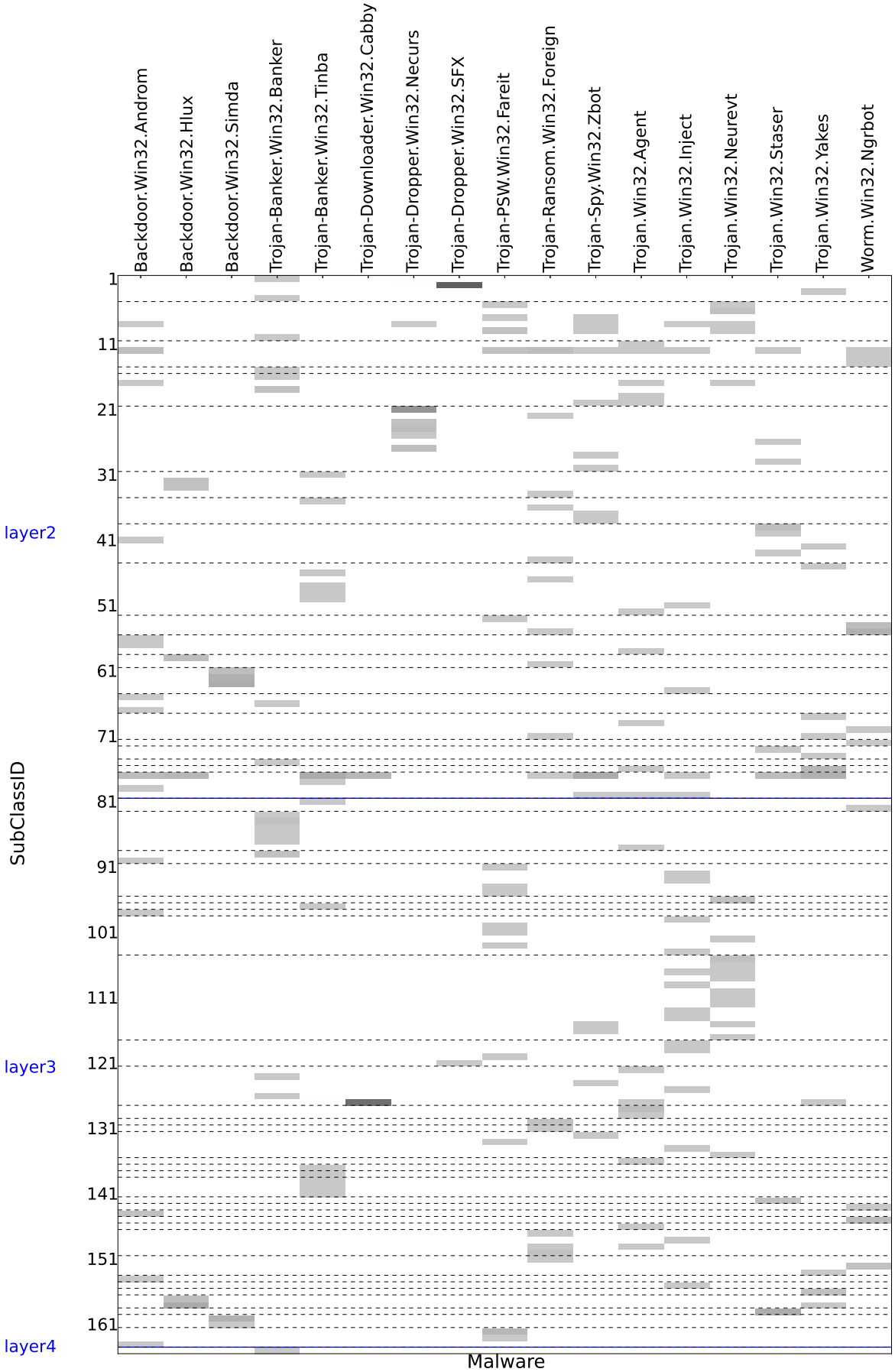

図 6.4: 各サブクラスに分類されたマルウェア数 (実験データ 1)

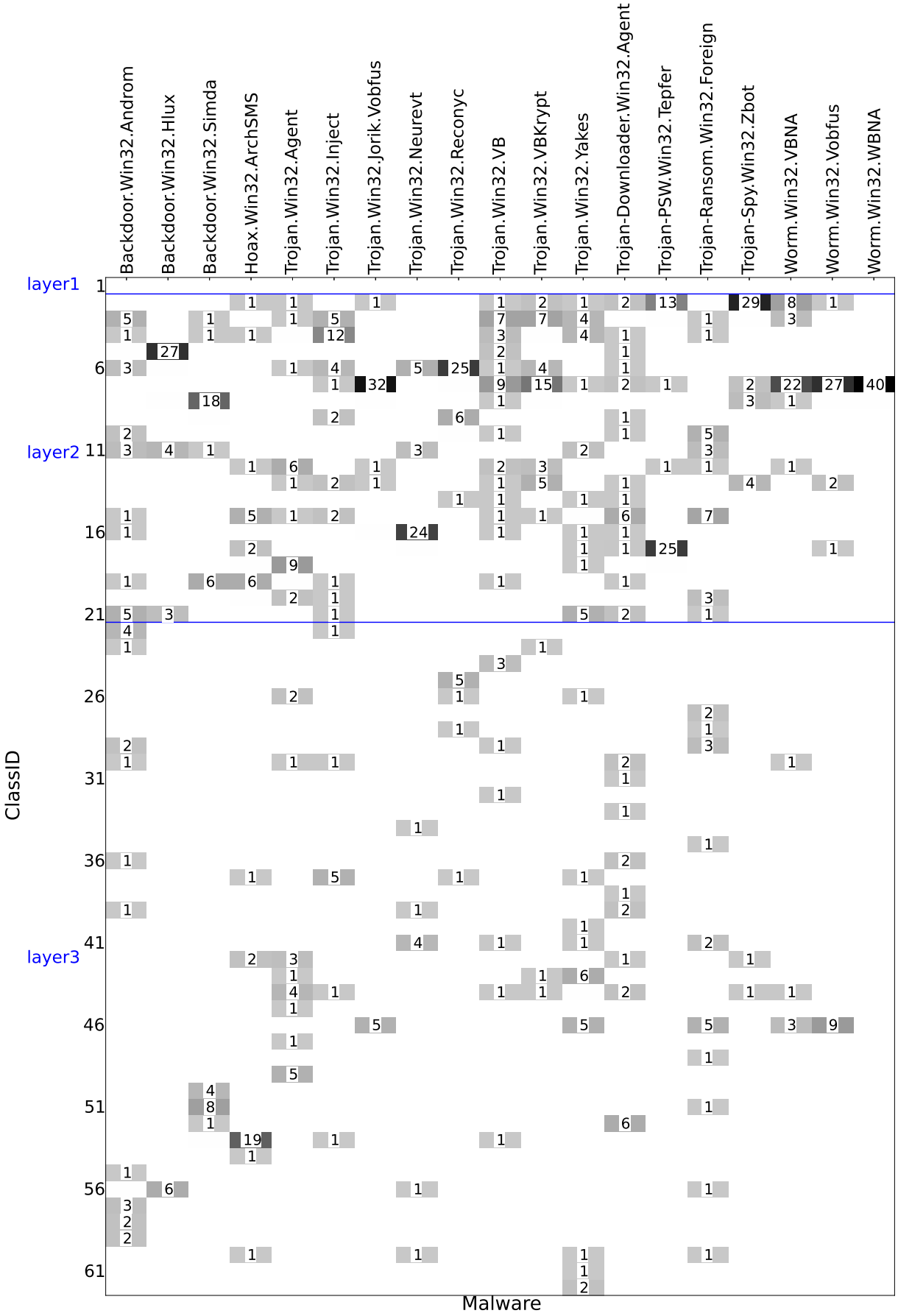

図 6.5: 各クラスに分類されたマルウェア数 (実験データ 2)

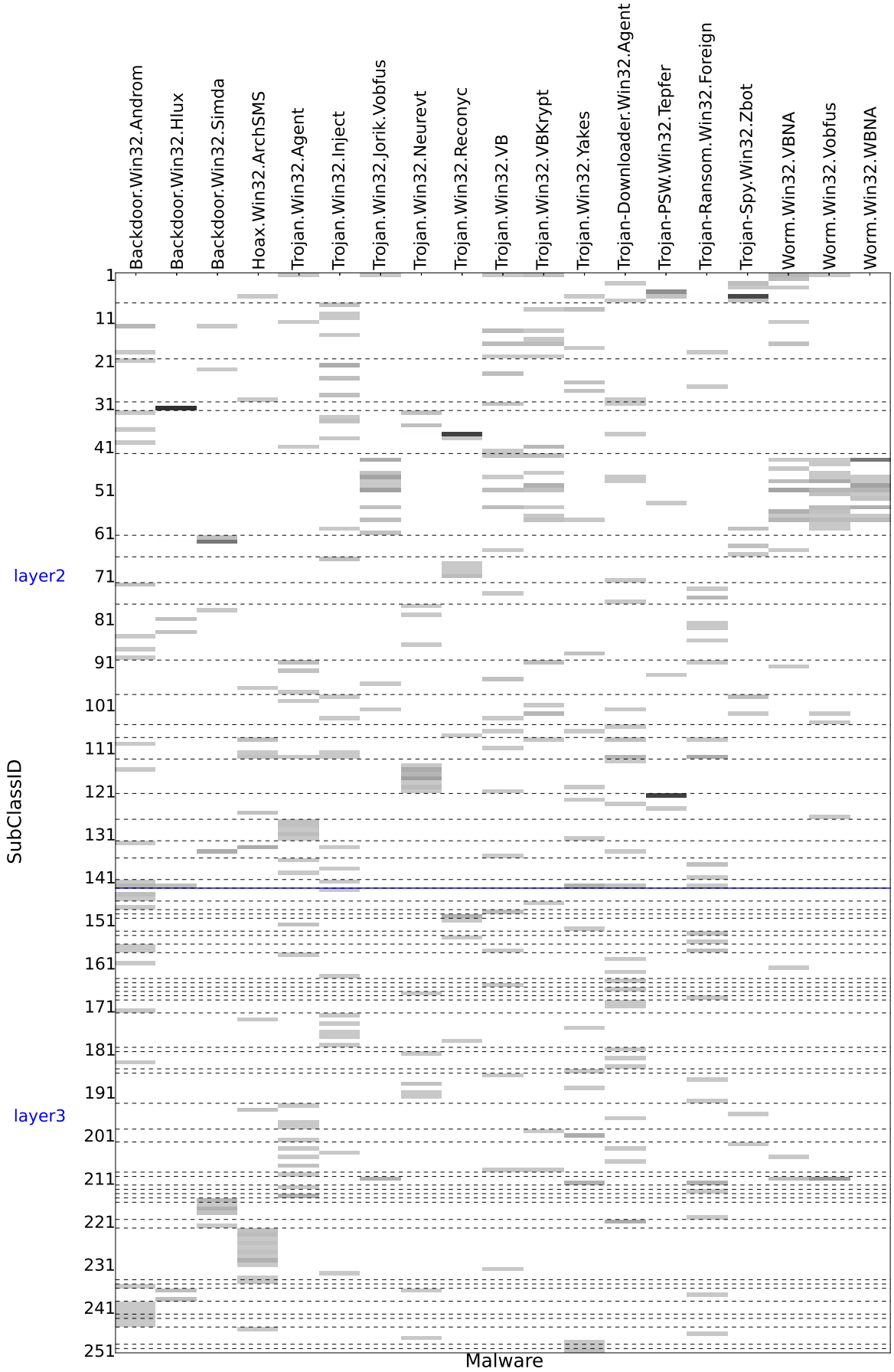

図 6.6: 各サブクラスに分類されたマルウェア数 (実験データ 2)

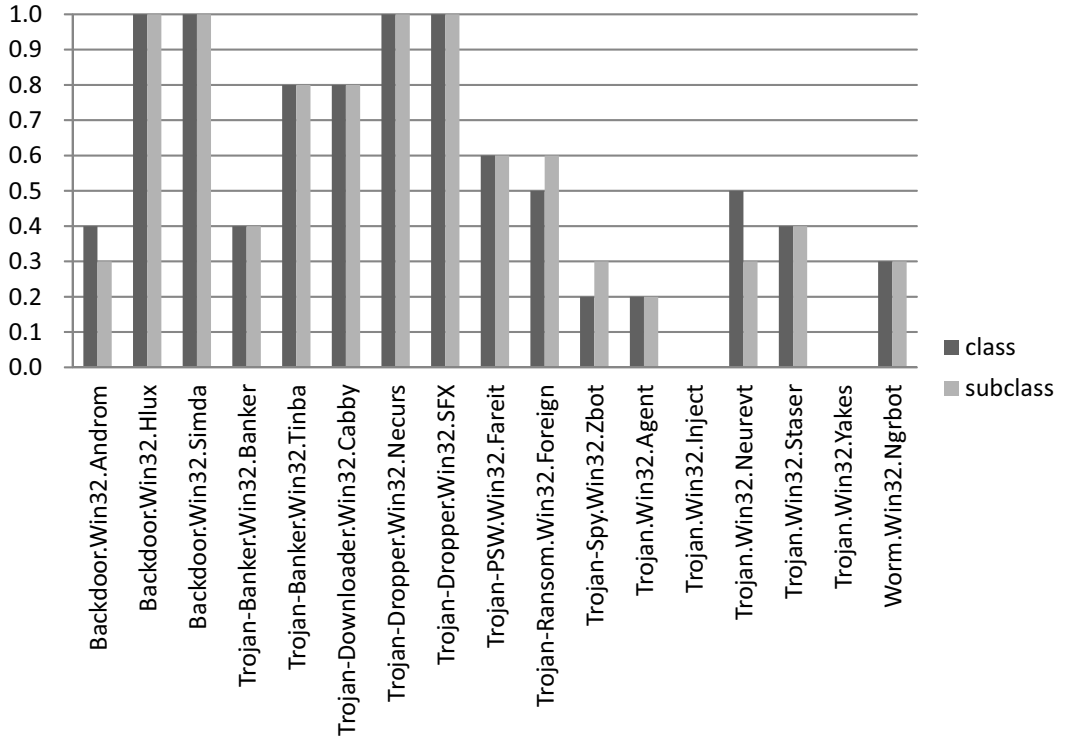

図 6.7: Accuracy (実験データ 1)

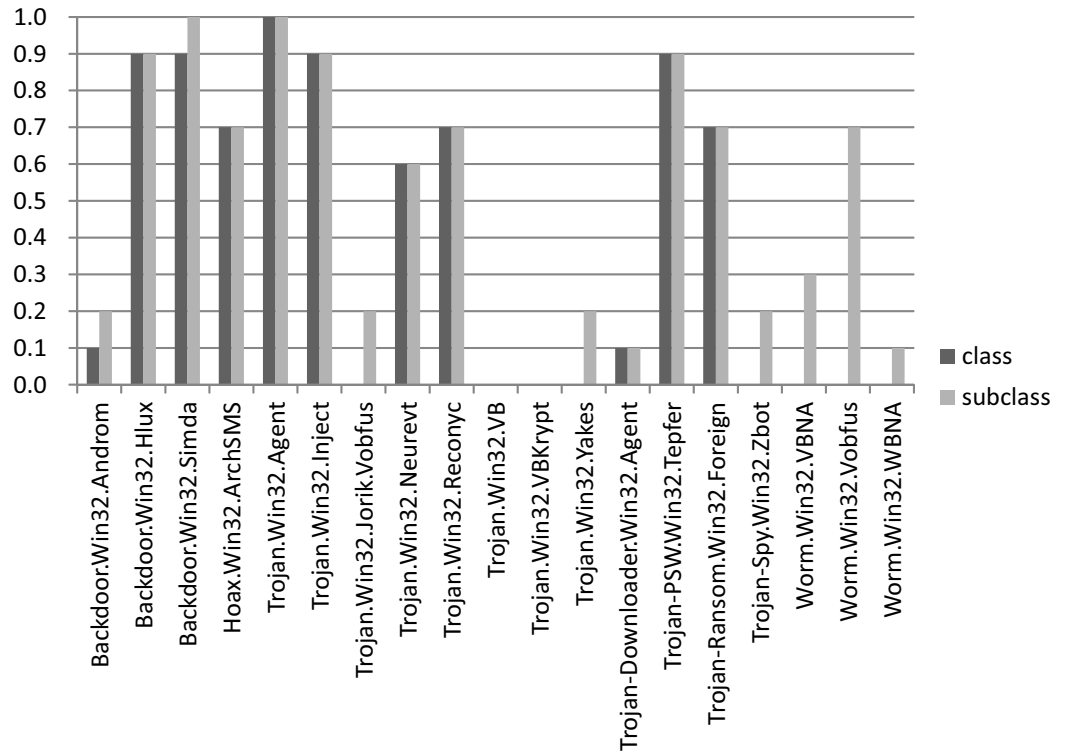

図 6.8: Accuracy (実験データ 2)

6.3　考察

6.3.1　本手法の有効性

図 6.7 と図 6.8 から考察を行う．実験データ 1 では，Backdoor.Win32.Hlux，Backdoor.Win32.Simda，Trojan-Banke.Win32.Tinba，Trojan-Downloader.Win32.Cabby，Trojan-Downloader.Win32.Necurs，Trojan-Dropper.Win32.SFX の 6 種類のマルウェアについて，クラス・サブクラスどちらの場合も Accuracy が 0.8 以上であった．実験データ 2 では，Backdoor.Win32.Hlux，Backdoor.Win32.Simda，Trojan.Win32.Agent，Trojan.Win32.Inject，Trojan-PSW.Win32.Tepfer の 5 種類のマルウェアについて，クラス・サブクラスどちらの場合も Accuracy が 0.9 以上であった．Accuracy が高いマルウェアについては亜種を集約したクラスタリングができている．このことから，一部のマルウェアに対して本手法が有効であることがわかる．また，クラス・サブクラスどちらの手法も有効であることから，より詳細な分類を行うときはユニット単位でクラスタを定義し，大まかな分類を行うときにはマップ単位でクラスタを定義するなど，必要に応じた使い分けができることがわかる．

6.3.2　有効なクラス・サブクラス

図 6.3，図 6.4，図 6.5，図 6.6 から考察を行う．Accuracy が高いマルウェアを見ると，一つのクラスタに多くのトレーニングデータが分類され，なおかつ他の分類されているマルウェアの数が少ないことがわかる．例えば図 6.3 では，Backdoor.Win32.Simda が ClassID 15，52 のクラスにそれぞれ 14，9 個分類されており，なおかつ他のマルウェアの数が少ない．同様のマルウェアについて図 6.6 で見ると，SubClassID 61，62，63，160，161 のサブクラスに分類されており，他のマルウェアは 1 つも分類されていない．このようなクラス・サブクラスが各マルウェアの判別率向上に貢献している．

6.3.3　GHSOM の柔軟性

実験データ 1 と実験データ 2 のクラスタリングの結果を比較すると，実験データ 2 を用いた場合の方がクラス・サブクラスの数が多くなっている．これは，実験データ 2 の方が入力データの数，特徴量の数のどちらも実験データ 1 より多かったため，GHSOM が自動的にクラスタ

数を調節したためである．このことから, GHSOM が入力データの大きさに関わらず柔軟にクラスタリングができる手法であることがわかる．

6.3.4　その他の分類結果

実験データ 2 のクラスタリング結果を見ると，Trojan.Win32.Jorik.Vobfus, Worm.Win32.VCNA，Worm.Win32.Vobfus，Worm.Win32.WBNA の 4 つのマルウェアのほとんどが同じクラスタに分類されている．したがって, これらのマルウェアには同じ API を同じ程度の回数使用する機能が内在していることを表している．ここで, 上記の 4 つのマルウェアが分類されたクラスタについて, 各クラスタを決定する際に重要になった特徴量を調査する．そうすると, API の LdrLoadDll が重要な特徴量であることがわかった．この API は DLL ファイルの読み込みのときに用いられる．このことから, 上記の 4 種類のマルウェアには DLL ファイル扱うという共通の特徴があることがわかった．以上のように, 同じクラスタに分類できたマルウェアについては共通の特徴が何であるのかを調査することができた．また, 今回のように DLL ファイルを扱う特徴があるマルウェアについては, 史らが用いた特徴量を組み合わせることでより詳細なクラスタリングができる．

第 7 章

結論

7.1 まとめ

本研究では, 動的解析ログに含まれる API コール情報を特徴量としたマルウェアのクラスタリングを提案した．具体的には, 動的解析ログから API の関数名のみを抽出して, これを特徴量とした GHSOM によるクラスタリングを行う手法である．次に, 本手法の有効性を示すための実験を行った．まず, データセットをトレーニングデータ, テストデータに分けて, トレーニングデータに対してクラスタリングを行った．さらに, クラスタリングの結果を基にテストデータのマルウェア判別を行った．この判別結果から本手法の有効性を示した．また, クラスタリングの結果から GHSOM が自動的にクラスタ数を調節したことを確認した．これにより, GHSOM は入力データの数や特徴量の数に関わらない柔軟なクラスタリングができることを示した．さらに, 同じクラスタに分類できたマルウェアについては共通の特徴を発見できることがわかった.

7.2 今後の課題

7.2.1 特徴量の選定

本研究では動的解析でマルウェアが用いたすべての API の使用回数を特徴量としたが, API の中にはクラスタリングのノイズとなる API も存在する．そのため, マルウェアごとにどのような特徴量が有効に作用しているのかを調査して, 特徴量の選定を行うことでより精度の高いクラスタリングが可能になる.

7.2.2　特徴量の連携

今回の実験では API の関数名のみを特徴量としたクラスタリングを行った．その他にも関連研究で紹介したようにさまざまな特徴量を用いた研究が行われている．その特徴量の中から, 実用性を考慮した量の特徴量を選んで連携を行うことにより, クラスタリングの精度をさらに向上させることができる.

7.2.3　有効に作用した特徴量の調査

6.3.4 節で述べたように有効な特徴量を調査することができる．本稿では各クラスタの詳細な調査を行っていない．各クラスタを形成する際にどのような特徴量が重要であったかを調査することで API ごとの重要度がわかり, 特徴量の選定に役立てることができる.

謝辞

本修士論文を作成するにあたり, 日頃よりご指導いただいた早稲田大学基幹理工学研究科の後藤滋樹教授に深く感謝いたします．また, 本研究を進めるにあたり, OG 史虹波氏には, 実験方法, 参考文献, その他有益な情報のご提供と, 多大なご協力をいただき大変感謝いたします. 最後に, 研究室で共に過ごした後藤滋樹研究室の諸氏に感謝いたします.

参考文献

[1] “G DATA MALWARE REPORT,” G DATA SECURITYLABS, https://public.gdatasoftware.com/Presse/Publikationen/Malware_Reports/GData_PCMWR_H2_2014_EN_v1.pdf

[2] 新井 悠, 岩村 誠, 川古谷 裕平, 青木 一史, 星澤 裕二, “アナライジング・マルウェア―フリーツールを使った感染事案対処,” オライリージャパン, pp.1–17, December 2010.

[3] Cuckoo Sandbox, http://www.cuckoosandbox.org

[4] The GHSOM Project, http://www.ifs.tuwien.ac.at/~andi/ghsom/download.html

[5] T. コホネン, “自己組織化マップ,” 丸善出版, 東京, 2012.

[6] 金 明哲, “Rによるデータサイエンス - データ解析の基礎から最新手法まで,” 森北出版, 東京, 2007.

[7] 田中 雅博, 古河 靖之, 谷野 哲三, “自己組織化マップを利用したクラスタリング,” 電子情報通信学会論文誌, vol.J79-D-2, no.2, pp. 301–304 1986, Feb. 1996.

[8] マインドウエア総研, “SOM 活用のメリットとは,” http://www.mindware-jp.com/basic/faq3.html, Dec. 14, 2013.

[9] A. Rauber, D. Merkl, M. Dittenbach, “The Growing Hierarchical Self-Organizing Map: Exploratory Analysis of High-Dimensional Data,” IEEE Transactions on Neural Networks Vol.13 (6) , pp.1331–1341, 2002.

[10] FFRI Dataset 2013, http://www.iwsec.org/mws/2013/files/FFRI_Dataset_2013.pdf

[11] Virus Total, https://www.virustotal.com/

[12] Kapersky, http://www.kaspersky.com

[13] “Rules for naming detected objects,” SECURELIST, http://www.securelist.com/en/threats/detect?chapter=136

[14] Hongbo Shi, Tomoki Hamagami, Katsunari Yoshioka, Haoyuan Xu, Kazuhiro Tobe, Shigeki Goto, “Structural Classification and Similarity Measurement of Malware,” IEEJ Transactions on Electrical and Electronic Engineering Volume 9, pp.621–632, November 2014.

[15] 藤野 朗稚, 森 達也, “自動化されたマルウェア動的解析システムで収集した大量の API コールログの分析,” コンピュータセキュリティシンポジウム 2013 論文誌, pp.618–625, 2013.

[16] 中村 燎太, 松宮 遼, 高橋 一志, 大山 恵弘, “Kullback-Leibler 情報量を用いた亜種マルウェアの同定,” コンピュータセキュリティシンポジウム 2013 論文誌, pp.877–884, 2013.

[17] 青木 一樹, 後藤 滋樹, “マルウェア検知のための API コールパターンの分析,” 電子情報通信学会総合大会講演論文集 2014 年, pp.179, 2014.

[18] 神薗雅紀, 秋山満昭, 笠間貴弘, 村上純一, 畑田充弘, 寺田真敏, “マルウェア対策のための研究データセット〜MWS Datasets 2015〜,” 情報処理学会 Vol.2015-CSEC-70 No.6, pp.1-8, 2015.

[19] Yahui Yang, Dianbo Jiang, Min Xia, “Using Improved GHSOM for Intrusion Detection,” JOURNAL OF INFORMATION ASSURANCE AND SECURITY (JIAS) 2010 Vol.5, pp.232–239, 2010.

[20] Faraz Ahmed, Haider Hameed, M.Zubair Shafiq, Muddassar Farooq, “Using Spatio-Temporal Information in API Calls with Machine Learning Algorithms for Malware Detectioni,” 16th ACM Conference on Compurter and Communications Security, pp.55–62, 2009.

[21] Ashkan Aami, Babak Yadegari and Hossein Rahimi, Naser Peiravian, Sattar Hashemi, AliHamze, “Malware Detection Based on Mining API Calls,” Proceedings of the 2010 ACM Symposium on Applied Computing, pp. 1020–1025, 2010.

[22] 戸部 和洋, 森 達哉, 千葉 大紀, 下田 晃弘, 後藤 滋樹, “実行ファイルに含まれる文字列の学習に基づくマルウェア検出方法,” マルウェア対策研究人材育成ワークショップ 2010, 2010.

[23] 高須 雄一, 後藤 滋樹, “ダークネット観測に基づく攻撃の時間変化の可視化,” 早稲田大学 2011 年度修士論文, February 2012.

[24] “マルウェアの進化と脅威の状況 10 年間の振り返り,” Microsoft Security Intelligence Report, February 2012

[25] “ワームウイルスとは,” urlhttps://japan.norton.com/worm-4330