2008－CSEC－41 (8)
2008／5／23

# 自己組織化マップを用いたWindowsシステムサービスコールの分類によるマルウェア検出手法

柿本 圭介，田中 英彦
情報セキュリティ大学院大学

**あらまし** 自己組織化マップは、データマイニング手法の一つであるが、多次元のデータの統計的性質を学習し、類似した性質を持ったものどうしが近接するように低次元化を行い、2次元平面上への写像を可能とする手法である。このような特性をもった自己組織化マップは、予測を行う問題において有効であると考えられている。本研究では、未知のマルウェアを検出することを目的とした動的解析による検出手法として自己組織化マップを適用し、解析対象とするデータとしてマルウェア実行時に呼び出されるWindowsシステムサービスの呼び出し列および入出力パラメータに設定される文字列を用いた手法を提案する。

# Malware Detection Method by Windows System Service Call Classification using Self-Organizing Maps

Keisuke KAKIMOTO, Hidehiko TANAKA
Institute of Information Security

**Abstract** Self-Organizing Maps is one of the data mining methods, which is able to project the multidimensional data onto two-dimensional plane by learning the statistical nature of the data and producing a low-dimensional representation so as to locate the data which has same nature near each other. Because of such characteristics, Self-Organizing Maps is effective to prediction problems. In this research, we propose an dynamic analysis method to detect unknown malware by Self-Organizing Maps, and our target of analysis are windows system service call sequence which is caused by malware and input/output parameters of the system service.

## 1 はじめに

ブロードバンドインターネット環境の普及に伴って、情報通信基盤がますます重要な社会インフラとなる一方、ワームやウィルスなど（以下、悪意のあるプログラムを総称してマルウェアと呼ぶ）によるネットワークへの被害の拡散はより広範囲、かつ急速なものとなっている。マルウェアの数、種類は年々増加しており、オリジナルの一部を改変した亜種の急速な感染拡大や、自らプログラムの一部を変更するマルウェアの出現によって、データのバイト列を攻撃のシグネチャと単純にマッチングする方式によるセキュリティシステムでは検出できないケースが増えてきている。このような攻撃を検出するための有効な対策の一つにプログラムの動作の監視による異常検知がある。

侵入検知システムにおいて、分析手法によって不正検知と異常検知に分類されるが、不正検知が既知の攻撃のシグネチャをパターンマッチングすることにより検知を行うのに対し、異常検知はシステムやユーザの通常の状態を定義しておき、通常でない状態を閾値により判別して攻撃を検知するため、未知の攻撃に対して有効な手法である。

本研究では、未知の攻撃を検知することを目的とした異常検知の考え方をベースとして、マルウェアの検出に自己組織化マップを適用し、分析対象とするデータとして、保護対象となるWindowsマシン上でプログラムを動作させた際に呼び出されるシステムサービスの情報を用いて検出を行う方式についての検討を行った。

## 2 自己組織化マップとは

### 2.1 概要

自己組織化マップ（Self-Organizing Maps、以下SOMとする）は、データマイニング手法の一つであるが、多次元のデータの統計的性質を学習し、類似し

た性質を持ったものどうしが近接するように低次元化を行い、2 次元平面上への写像を可能とする手法である。このような特性をもった SOM は、予測を行う問題において有効であると考えられている。1981 年に Kohonen により教師なしのニューラルネットワークアルゴリズムとして提案されたのが最初であるが、その後教師ありのクラス分類が行えるように拡張したものや、時系列情報を扱うことができない性質を改善したものなど、様々な拡張版が提案されている。

## 2.2 マルウェアの検出への適用

本研究では、既知のマルウェアだけではなく未知のマルウェアについても検出可能な方式を実現するための手法として SOM を適用する。

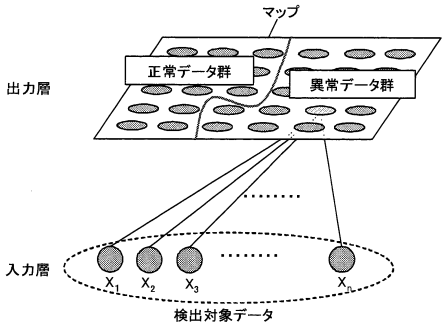


図 1 マルウェア検出方法概念図

SOM をマルウェアの検出に適用する場合、学習フェーズと検出フェーズの 2 段階に分けられるが、それぞれにおいて以下のような操作を行う。

- 学習フェーズ
  正常なデータと異常なデータを用いて学習を行い予めマップを作成する。このとき、正常なデータ同士、異常なデータ同士でデータの性質が類似していて、それぞれが近接するようなマップが作成されることが望ましい。
- 検出フェーズ
  学習フェーズで生成したマップに対して、検出対象となる入力データを与えて検出を行う。検出する方法は、入力データがマップ上の正常なデータ群と異常なデータ群のどちらに分類されるかによって判別する方法とする。

# 3 関連研究

SOM をホスト型の異常検知に適用した研究として、Wei ら[1]の研究がある。Wei らは、正常時に呼び出されるシステムコールを入力データとして、SOM と隠れマルコフモデルを適用し、検知の正確性とパフォーマンスの両面で比較を行っている。SOM を適用した場合、システムコールの頻度情報が入力データとなるのに対し、隠れマルコフモデルを適用した場合はシステムコールの呼び出し列情報を入力データとしている。実験の結果、隠れマルコフモデルに比べて SOM は検知の正確性ではやや劣っているが、パフォーマンスにおいては勝っているため、リアルタイムでの検知には SOM の方が適しているとしている。

システムコールの呼び出し列を侵入検知に用いる研究は、主に UNIX 系 OS を対象として多数行われてきているが、その先駆的な研究として Forrest ら[2]の研究が挙げられる。Forrest らは、特定のアプリケーションの通常時の動作を N-gram 法により定義しておき、不正な攻撃を受けた際の動作との比較により異常検知を行う手法を提案した。

N-gram 法でシステムサービス(UNIX におけるシステムコールに相当)の呼び出し列を特徴化する異常検知を Windows に適用した研究としては島本ら[3]の研究がある。島本らは、カーネルレベルで呼び出されるシステムサービスを監視して得られた呼び出し列を N-gram 法により特徴化する手法を用いているが、実験による評価の結果、効果的な異常検知が実現可能であることを示す測定結果が得られている。

Wei らの研究と本研究の差異としては、入力データとして、システムコールの呼び出し列だけでなく入出力パラメータに設定される文字列を加えた点である。これにより、mimicry attack のようにダミーのシステムコールを発行して正常なように偽装するような攻撃に対しても検出が可能な手法となり、検出精度の向上が期待できる。また、Forrest らの研究や島本らの研究との差異は、前述の入出力パラメータを加えたことと、検出するための手法として SOM を適用する点である。SOM を適用することにより、検出を行うのと同時に検出対象となるデータの性質をマップ上の近接するデータから類推することが可能となる。

# 4 提案手法

## 4.1 概要

本研究では、マルウェアを検出するための手法として SOM を適用し、解析対象とするデータとして、マルウェア実行時に呼び出されるシステムサービスの呼び出し列および入出力パラメータの文字列を用いる方式を提案する。

提案手法は、学習フェーズと検出フェーズの 2 段階に分けられる。まず、学習フェーズにおいて正常なデータと異常なデータを用いてマップを生成する。正常なデータとして、正規のアプリケーションをインストールした際に呼び出されるシステムサービスのログを

使用し、異常なデータとして、マルウェアを実行した際に呼び出されるシステムサービスのログを使用する。これら 2 種類のデータそれぞれから、システムサービスの呼び出し列と入出力パラメータ文字列に関する特徴ベクトルを生成し、それを入力データとして SOM による学習を行い、正規のアプリケーション同士、マルウェア同士が近接するようにマップを生成する。

次に検出フェーズにおいて、生成したマップに対して検出対象となるデータの特徴ベクトルを入力データとして与え、マップ上の全てのノードの中から特徴ベクトルと最も類似した参照ベクトルをもつノードを見つけ出す。そのノードが学習フェーズにおいて正常なデータが写像されたものなのか異常なデータが写像されたものなのかを判別することによりマルウェアの検出を行う。

## 4.2 学習フェーズにおける操作手順

学習フェーズでは、学習用の検体を動作させて取得したシステムサービスのログからシステムサービスの呼び出し列と入出力パラメータの文字列を抽出して特徴ベクトルを生成し、生成した特徴ベクトルを入力データとして SOM を実行しマップを生成する。

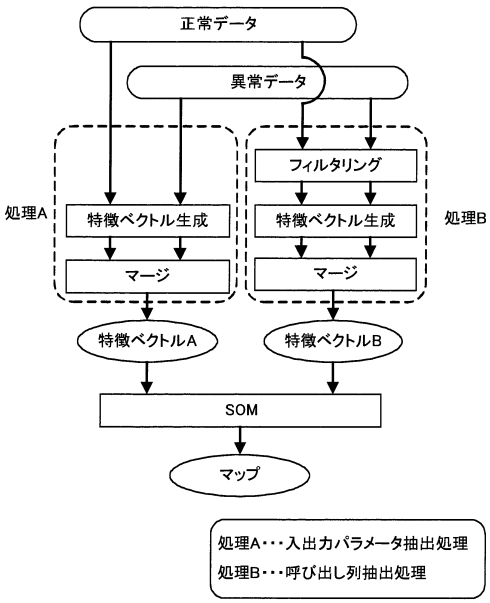


図 2 学習フェーズの流れ

### 4.2.1. システムサービス入出力パラメータ文字列の抽出手順

システムサービス入出力パラメータ文字列を抽出する手順は以下のようになる。

(1) 特徴ベクトル生成

正常なデータと異常なデータから別々に、以下の 3 つの条件を指定して入出力パラメータ文字列を抽出する。

【条件 1 】抽出対象とする入出力パラメータのリスト

どのシステムサービスのどの入出力パラメータから文字列を抽出するのかを指定する。例えば、NtCreateKey()の第 2 入力パラメータ及び、NtDeviceIoControlFile()の第 6 入力パラメータのように複数の指定を可能とする。

【条件 2】抽出対象文字

どの文字を抽出対象とするのかを指定する。例えば、アルファベットの a～z、半角スペース、カンマのように指定する。

【条件 3】連続出現文字数

抽出対象文字で指定した文字のいずれかが、何文字以上連続して出現する文字列を抽出対象とするのかを指定する。

このような方法により抽出した文字列を属性とし、属性値をその文字列の出現可否とする特徴ベクトルを生成する。

(2) マージ

正常なデータと異常なデータのそれぞれから生成した特徴ベクトルをマージする。マージの方法は、異常なデータから生成した特徴ベクトルにのみ存在する属性を残し、正常なデータから生成した特徴ベクトルにのみ存在する属性および両方に存在する属性を削除する方法とする。

### 4.2.2. システムサービス呼び出し列の抽出手順

システムサービスの呼び出し列を抽出する手順は、以下のようになる。

(1) フィルタリング

システムサービスのログから、以下に示すシステムサービスの呼び出しに関する行を残し、それ以外の行を削除する。

- ファイル操作関連
  NtCreateFile(),NtOpenFile(),NtReadFile(),NtWriteFile(), その他 9 種類
- レジストリ操作関連
  NtCreateKey(),NtDeleteKey(),NtOpenKey(),NtQueryKey(), その他 6 種類
- ネットワーク接続関連
  NtDeviceIoControlFile()

(2) 特徴ベクトル生成

正常なデータと異常なデータ対して別々にギブスサンプリング[1]を適用し、正常なデータの間で類似している呼び出し列と異常なデータの間で類似している呼び出し列を抽出する。次に、抽出した呼び出し列を属性とし、属性値をその呼び出し列の出現可否を表す値とする特徴ベクトルを生成する。

(3) マージ

正常なデータと異常なデータのそれぞれから生成した特徴ベクトルをマージする。マージの方法は、正常なデータから生成した特徴ベクトルにのみ存在する属性および異常なデータから生成した特徴ベクトルにのみ存在する属性を残し、両方に存在する属性を削除する方法とする。

#### 4.2.3. マップの生成方法

4.2.1、4.2.2 述べた方法により呼び出し列の特徴ベクトルと入出力パラメータ文字列の特徴ベクトルを生成し、これらを入力データとして SOM を実行してマップを生成する。

SOM により学習を行った結果、各データは、マップ上のいずれかのノードに写像され、言い換えると、写像が行われたノードは SOM に与えた学習用データのいずれかを表している状態となる。SOM では、各ノードがどのデータの写像であるかという「ラベル付け」の情報を保持しているが、マップ上の全てのノードがラベル付けされているとは限らないため、ラベル付けが行われていないノードも存在する。正常なデータが写像されたノードを正常ノードとし、異常なデータが写像されたノードを異常ノードとすると、学習後に生成されたマップは図 3 のようになる。

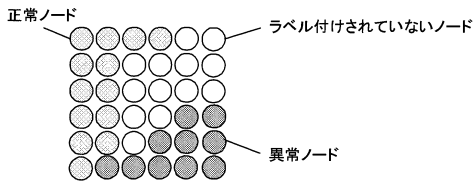


図 3 学習後のマップ

### 4.3 検出フェーズにおける操作手順

検出フェーズでは、検出対象となる検体を動作させて取得したシステムサービスのログから学習フェーズにて抽出したシステムサービスの呼び出し列と入出力パラメータの文字列が出現するか否かの情報を抽出して特徴ベクトルを生成する。次に学習フェーズにて生成したマップの中から、生成した特徴ベクトルと最も類似している参照ベクトルをもつノードを見つけ出し、そのノードが正常なデータの写像が行われたノードなのか異常なデータの写像が行われたノードなのかを判別することにより検出を行う。

#### 4.3.1. 検出方法

呼び出し列の特徴ベクトルと入出力パラメータ文字列の特徴ベクトルを入力データとして SOM を実行して以下の手順により検出を行う。

(1) 類似ノードの選出

学習フェーズにて生成したマップ上のノードから、検出対象となるデータから生成した特徴ベクトルと最も類似している参照ベクトルを持つノードを選出する。具体的には、ある特徴ベクトルとマップ上の全ての参照ベクトルとのユークリッド距離を算出し、その中から最小の距離となる参照ベクトルをもつノードが類似ノードとなる。

(2)ラベル付け判定

類似ノードが学習フェーズにてラベル付けされたノードかどうかの判定を行う。

(3)最近接ノード選出

ラベル付け判定でラベル付けされたノードでなかった場合、最も近くにあるラベル付けされたノードを選出する。

(4)正常・異常判定

(1)または(3)で選出したラベル付けされたノードが、正常ノードか異常ノードかに従って判定を行う。なお、(3)で選出した最近接ノードが複数存在し、その中に正常ノードと異常ノードが混在している場合は異常と判定する。

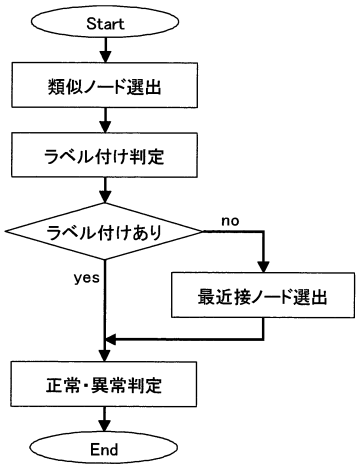


図 4 検出手順フロー

1 確率分布から偏りなくランダムにサンプルの列を得る手法で、与えられた複数の配列から互いにできるだけ類似している部分文字列集合を求めることができる。

## 5　提案方式の検証

### 5.1　検証環境の構成

検証環境は、以下の 3 つのシステムからなる。

・マルウェア収集システム

・マルウェア自動実行システム

・マルウェア解析システム

マルウェア収集システムでは、ローインタラクション型のハニーポットである Nepenthes ver.0.1.7 を用いて検体の収集を行った。

マルウェア自動実行システムでは、収集したマルウェアを実行するための仮想環境を構築し、実行後にマルウェアから呼び出されるシステムサービスをトレースした。マルウェアを実行した仮想環境における OS は WindowsXP で、システムサービスのトレースには、Strace for NT を用いた。

マルウェア解析システムでは、マルウェアから呼び出されたシステムサービスコールのログから、呼び出し列および入出力パラメータ文字列に関する特徴ベクトルを生成し、SOM を実行する。SOM は、ESOM を用い、学習フェーズでは特徴ベクトルを入力としてマップを生成し、検出フェーズでは特徴ベクトルに対する類似ノードの選出を行う。

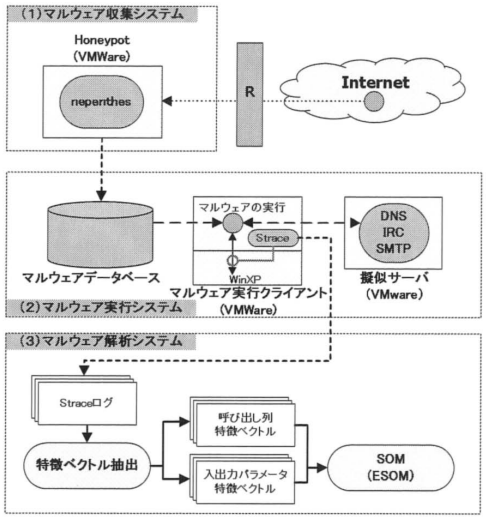


図 5　検証環境の構成

### 5.2　入力データセット

正常データ、異常データとして、以下の方法により取得したデータを使用した。

・正常データ

拡張子が「.exe」である実行形式のアプリケーションを download.com[2]から合計 100 個ダウンロードした。次にダウンロードした全てのアプリケーションを一つずつ手動で実行してインストールを行い、この際に呼びだされるシステムサービスを Strace for NT を使用して記録した。

・異常データ

5.1 節で述べた検証環境にて収集したマルウェアの中からランダムに合計 100 個選出して実行し、システムサービスを記録した。なお、検証環境におけるマルウェアの実行時間は約 30 秒である。

次に検証結果の信頼性を上げるために、正常データ、異常データとして取得したシステムサービスコールのログから以下の手順により 4 つのパターンのデータを生成した。

① 正常データを 4 つのグループ N1、N2、N3、N4 にそれぞれ 25 個ずつランダムに振り分け、同様に異常データを 4 つのグループ M1、M2、M3、M4 にそれぞれ 25 個ずつランダムに振り分けた。

② 次に学習用データ、検出用データとして、表 1 に示すように 4 つのパターンを用意した。

表 1　検証用データパターン

| パターン No. | 学習用データ | | 検出用データ | |
|---|---|---|---|---|
| | 正常データ | 異常データ | 正常データ | 異常データ |
| 1 | N2、N3、N4 | M2、M3、M4 | N1 | M1 |
| 2 | N1、N3、N4 | M1、M3、M4 | N2 | M2 |
| 3 | N1、N2、N4 | M1、M2、M4 | N3 | M3 |
| 4 | N1、N2、N3 | M1、M2、M3 | N4 | M4 |

これら 4 つのパターンを用いて検証を行い、各々の検証結果の平均を求めた。

### 5.3　検証結果

#### 5.3.1.　検出率

表 2 に示す 4 つのパターンで特徴ベクトルを生成して、特徴抽出率および検出率の測定を行ったところ、表 3 のような結果になった。なお、特徴抽出率とは、特徴抽出したデータのうち、特徴ベクトルの属性値が 1 つでも 1 となっているデータの割合を表す。特徴抽出率が 100%に満たないということは学習時に生成した特徴ベクトルの属性を 1 つも持たないデータが存在するということであり、そのデータの検出は行えないため 100%であることが望ましい。

表 3 に示す検出率は、特徴が抽出できたデータの中で正しく検出できたデータの割合を表しているため、特徴が抽出できなかったデータを検出できなったものとみなすと、検出できたことを表す指標としては、特徴抽出率×検出率を用いる必要がある。

2　http://www.download.com/

表 2　特徴ベクトル生成パターン

| パターン No. | 呼び出し列長 N | | 入出力パラメータ抽出対象システムサービス | |
|---|---|---|---|---|
| | 10 | 50 | NtOpenKey | NtQueryAttributesFile |
| 1 | ○ | ○ | − | − |
| 2 | ○ | ○ | ○ | − |
| 3 | ○ | ○ | − | ○ |
| 4 | ○ | ○ | ○ | ○ |

表 3　提案手法の検出率

| パターン No | 特徴抽出率 | | 検出率 | | 特徴抽出率×検出率 | |
|---|---|---|---|---|---|---|
| | 正常 (%) | 異常 (%) | 正常 (%) | 異常 (%) | 正常 (%) | 異常 (%) |
| 1 | 99 | 96 | 100 | 99 | 99 | 95 |
| 2 | 99 | 100 | 99 | 100 | 98 | 100 |
| 3 | 99 | 100 | 99 | 100 | 98 | 100 |
| 4 | 99 | 100 | 95 | 100 | 94 | 100 |

この結果から、異常データの検出を確実に行うことを最優先に考えた場合、パターン 2 または、パターン 3 において最も良い結果が得られたといえる。

### 5.3.2.　既存研究との検出精度の比較

Zhang ら[4]の研究では、未知のマルウェアを検出することを目的として、サポートベクターマシン（SVM）を用いた検出手法を提案しており、実際のマルウェアを用いた手法の評価を行っている。Zhang らの手法と本研究の提案手法との大きな違いは、解析対象とするデータとして Zhang らの手法では API コール（本研究ではシステムサービスコール）の呼び出し列のみを用いているのに対して、提案手法では呼び出し列に加えて、入出力パラメータの文字列を用いている点である。

表 4　既存手法との検出精度の比較

| 手法 | 解析対象データ (※) | | 検出率 | |
|---|---|---|---|---|
| | ① | ② | 正常 (%) | 異常 (%) |
| Zhang ら | ○ | − | 94.34 | 96.79 |
| 提案手法 | ○ | − | 99.00 | 95.00 |
| | ○ | ○ | 98.00 | 100.00 |

(※)①:API コール呼び出し列,　②:入出力パラメータ文字列

提案手法において入出力パラメータ文字列を加えたパターンは、正常データの検出率とともに異常データの検出率も Zhang らの手法を上回っており、提案手法により Zhang らの手法を検出精度の面で改善可能であることを示している。

## 6　今後の課題

今後の課題は、以下のようになる。

- データの数や種類を増やして検証を行う。特に正常なデータの数を増やすことと、5.1 節で述べた検証環境以外から取得した検体を使用して種類を増やすことが必要であると考えている。
- 仮想環境を検出して動作しないようなマルウェアや、実行したファイルから生成されたプロセスからさらに別のプロセスを生成してそのプロセスで不正な動作をするようなマルウェアに対して、システムサービス記録方法の改善策を検討する必要がある。
- 使用するリソースが日々変化するマシン上での適用を考えた場合、学習フェーズで正常な動作を網羅的に抽出するのは難しいため、教師ありの SOM を適用し、誤検出したデータを教師情報として与えることによりマップを自動更新していくような方法を検討していく必要がある。
- 本研究では、検出フェーズにおけるマルウェアの実行から検出に至るまでの過程を全て自動化したシステムを構築できていないため、全て自動化したシステムの実装と処理時間についての評価が今後の課題となっている。

## 参考文献


[1] Wei Wang, Xiaohong Guan, Xiangliang Zhang, Liwei Yang, "Profiling program behavior for anomaly intrusion detection based on the transition and frequency property of computer audit data, " Computer & Security 25, pp. 539-550(2006).

[2] Forrest S., Hofmeyr S.A., Somayaji A., Longstaff T.A., "A sense of self for Unix processes, " In: Proc. 1996 IEEE Symp. On Research in Security and Privacy, pp. 120-128(1996).

[3] 島本 大輔, 大山 恵弘, 米澤 明憲, "System Service 監視による Windows 向け異常検知システム機構, " 情報処理学会論文誌 コンピューティングシステム, Vol. 47, No. SIG 12(ACS 15), pp. 420-429(2006).

[4] Bo-yun Zhang, Jian-ping Yin, Jin-bo Hao, Ding-xing Zhang, Shu-lin Wang, "Using Support Vector Machine to Detect Unknown Computer Viruses," International Journal of Computational Intelligence Research, Vol2, No.1, pp.100-104(2006).