API呼び出しとそれに伴う経過時間とシステム負荷を用いたマルウェア検知手法

佐藤 順子[1]　花田 真樹[2]　面 和成[3]　山口 崇志[2]　鈴木 英男[2]　布広 永示[2]　折田 彰[4]　関口 竜也[4]

概要：マルウェアは，実行環境の検知，セキュリティサービスの無効化，自分自身を削除するなどの機能を用いて，検知の回避やユーザからの発見を免れようとしている．また，近年，マルウェアの自身の隠蔽が巧妙化しており，未知マルウェアを高精度で検知する手法が求められている．先行研究 [1] では，マルウェアの API 呼び出しパターン，API 呼び出しによる経過時間とシステム負荷に関する分析を行い，マルウェアと正規プログラムでは，同じ API 呼び出しパターンでもその API 呼び出しによる経過時間とシステム負荷に特徴が表れることが報告されている．本研究では，先行研究 [1] の結果を基に，API 呼び出しパターン，API 呼び出しによる経過時間とシステム負荷に関する特徴量を用いたマルウェア検知手法を提案し，評価する．

キーワード：マルウェア検知，API，システム負荷

Malware Detection Method based on API Call Patterns, Elapsed Time and System Load between API Calls

Junko Sato[1]　Masaki Hanada[2]　Kazumasa Omote[3]　Tatakachi Yamaguti[2]　Hideno Suzuki[2]
Eiji Nunohiro[2]　Akira Orita[4]　Tatsuya Sekiguchi[4]

Abstract: Malware has functions which detect the system environment, disable the security protection and remove myself. Using these functions, the malware tries to avoid detection. From these backgrounds, malware detection method is needed to provide high-accuracy detection because mechanism for hiding myself is getting sophisticated. In [1], the authors analysed the elapsed time and system load between API calls of malware, and reported that the elapsed time and system load between API calls of malwares are different from these of regular programs. In this paper, we propose a malware detection method based on API call pattern, elapsed time and system load between API calls.

Keywords: Malware detection method, API, System load

1. はじめに

近年，企業や政府機関の組織の情報漏えいや金銭の搾取を目的とした，マルウェアを用いたサイバー攻撃が多く報告されている [2]．その中でも，特にランサムウェアの検知数が，2015 年の 9.8 倍に増加したという報告 [3] もあり，今後，脅威が増大する可能性がある．侵入したマルウェアは，セキュリティサービスの無効化やでデバッガなどの検知，自分自身を削除するなどの機能を用いて，検知の回避

[1] 東京情報大学 総合情報学研究科
Graduate School of Informatics, Tokyo University of Information Sciences
[2] 東京情報大学 総合情報学部
Department of Information Sciences, Tokyo University of Information Sceinces
[3] 筑波大学大学院 システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba
[4] 株式会社日立システムズ サイバーセキュリティリサーチセンタ
Hitachi Systems, Ltd. Cyber Security Research Center

やユーザからの発見を免れようとしている．また，近年，マルウェアの自身の隠蔽が巧妙化しており，Nemesis[4] というマルウェアは，ボリューム・ブート・レコードを改変を行い，発見だけでなく削除も困難にしている．そのため，未知マルウェアを高精度で検知する手法が求められている．

マルウェアの検知手法は大きく 2 つに分けられ，ヒューテック法とビヘイビア法に分かれる．ヒューテック法は，静的解析でマルウェアの挙動を把握する手法であり，ソースコード上から悪意のある特徴的な動作を検出する．暗号化や難読化により検出が困難になる問題点がある．ビヘイビア法は，動的解析でマルウェアの挙動を把握する手法であり，実際に動作させるため，具体的な挙動を把握することが可能である．しかし，実行毎に挙動が変化するものや環境によって挙動が変わってしまう場合，常に観測させるとは限らない．先行研究 [1] では，マルウェアの身を隠すための機能に着目し，マルウェアと正規プログラムでは，同じ API 呼び出しパターンでもその API 呼び出しによる経過時間とシステム負荷に特徴が表れることが報告されている．

本研究では，先行研究 [1] の結果を踏まえ，API 呼び出しパターン，API 呼び出しによる経過時間とシステム負荷に関する特徴量を用いたマルウェア検知手法の提案を行い，評価する．

以下，2 章では関連研究について述べる．3 章では提案手法について述べる．4 章で評価実験を行う際に用いた使用検体，実験環境，評価指標，実験結果を述べる．5 章ではまとめと今後の課題について述べる．

2. 関連研究

近年，マルウェアの検知手法や分類手法に関して，API を特徴として用いた研究が広く行われている．API 呼び出しパターンに着目した検知手法 [5] では，API 呼び出しパターンの特徴を用いて，マルウェアの種類判定を行っている．マルウェアは，その種類によって固有の API 呼び出しパターンを持っており，検知に有効な API 呼び出しパターンの連鎖数が異なっていることが報告されている．API を用いた機能に基づく検知手法 [6] では，API がマルウェアの機能を決定しているという点に着目し，API の特徴を用いてマルウェアの有している機能ごとの分類を行っている．複数の機械学習を用いた提案・評価を行っており，マルウェアの機能を高精度で分類している．実行毎に動作を変える挙動に着目した検知手法 [7] では，パターンマッチング法や C&C サーバーの特定を困難にするマルウェアに着目している．特定の API の引数に注目しており，実行毎に引数が異なれば，マルウェアとして検知する．先行研究 [1] は，マルウェアの API 呼び出しパターン，API 呼び出しによる経過時間とシステム負荷に注目している．マルウェアは，API 呼び出しにより，システム負荷が高くならないようにするなどのユーザから身を隠すための特徴を有しているという想定に基づいている．先行研究 [1] では，マルウェアと正規プログラムでは，同じ API 呼び出しパターンでもその API 呼び出しによる経過時間とシステム負荷に特徴が表れることが報告されている．その他にも，システムの振る舞いに着目した研究として，システムフォルダなどの別のフォルダに新たなファイルを生成する挙動に着目した検知手法 [8] が提案されている．ユーザからの発見を免れるために行うボットの身を隠すため動作と侵入後消去されないようにする挙動に着目しており，侵入後生成されたファイルを侵入前の環境で実行したときに，侵入挙動が観測された場合にボットであると検知する．また，通信の時間的な振る舞いに着目した研究として，ボットの通信の挙動から検知する手法 [9] が提案されている．

本研究では，マルウェアと正規プログラムでは，同じ API 呼び出しパターンでもその API 呼び出しによる経過時間とシステム負荷に特徴が表れるという先行研究 [1] の結果を基に，従来の API 呼び出しパターンに加えて，API 呼び出しによる経過時間やシステム負荷を考慮した検知手法を提案する．

3. 提案手法

本研究では，API 呼び出しパターン，API 呼び出しによる経過時間とシステム負荷の特徴量を用いたマルウェアの検知手法を提案する．提案手法では，1 検体あたり 10 秒動作させ，その間に得られた API とそれに伴う経過時間とメモリ使用量を用いる．動的解析ログから，API 呼び出しとそれに伴う経過時間とメモリ使用量の情報を抽出し，マルウェアの特徴抽出を行い，機械学習アルゴリズムを用いてマルウェアか正規プログラムかの検知を行う．

3.1 想定動作環境

マルウェアを動作させるツールとして，CuckooSandbox[10] を用いる．マルウェア検知には，機械学習アルゴリズムを用いる．なお，CuckooSandbox については，経過時間とメモリ使用量の取得が困難であったため，変更を行っている．

3.2 特徴抽出

API 呼び出しによる経過時間やシステム負荷の特徴を表す手法は様々なものが考えらえる．本研究では，以下 2 つの手法（特徴抽出手法 1，特徴抽出手法 2）を用いる．

3.2.1 特徴抽出手法 1

API 呼び出しによる経過時間とシステム負荷を，統計的な手法を用いて特徴づける．本手法では，各検体における API 呼び出しパターンの出現回数，API 呼び出しによる経過時間の平均，最大値，最小値，分散を特徴量とする．例えば，遷移数 2 の API に着目した場合，API1 から API2

のパターンの出現回数が 2 で，API1 から API2 までの経過時間の平均が 4，最大値が 9，最小値が 1，分散が 6 であるとすると，$API1 \rightarrow API2 : (2, 4, 9, 1, 6)$ のように，出現回数，平均，最大値，最小値，分散の順のベクトル表記となる．API 呼び出しパターンが複数存在する場合は，この表記を連続で結合して表記する．

3.2.2 特徴抽出手法 2

API 呼び出しによる経過時間やシステム負荷をより正確に特徴づけるために，本手法では，区分けを行う値を用いて，経過時間やシステム負荷を複数区間に区分けする．例えば，2 区間を想定した場合，API1 から API2 への経過時間が 400 μs であり，区分けを行う値が 300μs の場合，300μs 未満を 0，300μs 以上を 1 で表記すると，$API1 \rightarrow API2 : 1$ という表記となる．これにより，マルウェアと正規プログラムの API 呼び出しによる経過時間やシステム負荷をより正確に特徴づけることが可能となる．

3.3 CuckooSandbox の変更

精度の高い経過時間が取得できるようにするためとメモリ使用量を取得するために，CuckooSandbox に変更を行っている．

API 呼び出しの経過時間については，CuckooSandbox には記録する機能が備わっている．しかし，timeGetTime 関数で取得しているため，ミリ秒単位の精度しか取得できないため，API 呼び出しは，マイクロ秒単位の精度が必要なため，本研究では用いることができない．本研究では，QueryPerformanceCounter 関数 [14] や QueryPerformanceFrequency 関数 [15] を用いて時間取得を行う．timeGetTime 関数より細かい時間取得が期待できるが，CPU カウンタを使用しているため，他の環境で同じような精度で取得できるとは限らないことになる．メモリ使用量については，CuckooSandbox には，調査環境のメモリ使用量を調べる機能はないため，CuckooSandbox の API フックの処理に GetProcessMemoryInfo 関数を用いたメモリ使用量を記録する機能を追加する．

3.4 提案手法の流れ

提案手法の検知の流れを以下に記す．

(1) 検体を CuckooSandbox を用いて動的解析し，API の呼び出しと呼び出しによる経過時間，その時のメモリ使用量の変動をログに記録する．
(2) 動作を確認した検知のうち，学習用と評価用の検体に分ける．学習用と評価用の検体で重複が生じないようにランダムに抽出する．
(3) 3.2 節の特徴抽出方法に沿って，データを加工する
(4) ナイーブベイズ法で判別，検知を行う．学習用の検体データは，トレーニングデータとして利用し，評価用の検体データは，マルウェアの判別に使用する．

4. 評価実験

比較実験の概要と使用検体，評価指標，実験結果について下記に記す．

4.1 比較実験の概要

API 呼び出しパターン（以降，遷移と呼ぶ）のみを考慮した実験と遷移と時間間隔を考慮した実験，遷移とメモリ使用量を考慮した実験を比較する．遷移のみを考慮した実験の特徴抽出方法は，（1 検体あたりの）API 遷移の回数ではなく，出現の有無とする．

4.2 使用検体

正規プログラムの検体は，Vector[11] から収集したファイルのうち，zip や tar.gz などで圧縮されていたファイルは解凍し，実行形式のファイルを用いた．収集時の検体は 3289 個であったが，そのうち，WindowsXP で動作した検体は 2710 個であり，Windows7 で動作した検体は 875 個であった．マルウェア検体は，独自にマルウェアが配布されているサイトを紹介しているサイトから収集したものと CCCDataset2013[12] で使われているマルウェアと同じものを用いた．独自で収集したマルウェアを使用した理由は，CCCDataset2013[12] で使われているマルウェアの多くは WindowsXP のみで動作する検体であるため，Windows7 で動作する検体数を確保し，検証を行うためである．収集時の検体は 2313 個であったが，Windows7 で動作した検体は 238 個である．独自で収集したマルウェアに関しては，収集後，VirusTotal[13] を用いた．したがって，CCCDataset2013 で使われているマルウェアを動作させる環境は WindowsXP，独自で収集したマルウェアを動作させる環境は Windows7 を用いた．実験では，評価用検体として 70%，学習用検体として 30%用いている．マルウェア・正規プログラムともに 50%の割合で使用している．CCCDataset2013 で使われているマルウェアの特徴抽出手法は，3.2.2 節で述べた手法，独自で収集したマルウェアの特徴抽出手法は 3.2.2 節で述べた手法を用いる．区分けを行う値は，経過時間が 300 μs，システム負荷は 4000 Byte を用いる．この値は，検体の事前調査において，経過時間やシステム負荷に関して，マルウェアと正規プログラムで最も差の生じる適当な値として抽出した値である．

4.3 評価指数

実験の評価指数として，Accuracy，TPR（True Positive Rate），TNR（True Negative Rate），FPR（False Positive Rate），FNR（False Negative Rate）を用いる．Accuracy は、正しく判別できた割合である．TPR はマルウェアをマルウェアとして判別した割合，TNR は正規プログラムを正規プログラムとして判別した割合，FPR はマルウェアを正

規プログラムとして判別した割合，FPR は正規プログラムをマルウェアとして判別した割合である．なお，Accuracy は TPR と TNR の平均値となる．Accuracy，TPR，TNR，FPR，FNR を下記に示す．

$$Accuracy = \frac{\text{正しく分類された検体数}}{\text{検体総数}} \quad (1)$$

$$TPR = \frac{\text{マルウェア検体をマルウェアとして分類した数}}{\text{マルウェア検体の総数}} \quad (2)$$

$$TNR = \frac{\text{正規プログラムを正規プログラムとして分類した数}}{\text{正規プログラムの総数}} \quad (3)$$

$$FNR = \frac{\text{マルウェア検体を正規プログラムとして分類した数}}{\text{マルウェア検体の総数}} \quad (4)$$

$$FPR = \frac{\text{正規プログラムをマルウェアとして分類した数}}{\text{正規プログラムの総数}} \quad (5)$$

4.4 実験結果

CCCDataset と正規プログラムを判別した実験と独自に収集したマルウェアと正規プログラムを判別した実験を行った．

4.4.1 CCCDataset2013 の検知精度

CCCDataset と正規プログラムを用いて，マルウェアの検知を行った．既存手法の遷移のみを考慮した場合の実験と提案手法の遷移とそれに伴う時間間隔を考慮した実験，提案手法の遷移とそれに伴うシステム負荷を考慮した実験を行った．既存手法の遷移のみを考慮した場合の実験結果を表 1 に示す．また，提案手法の遷移とそれに伴う時間間隔を考慮した実験結果を表 2 に，提案手法の遷移とそれに伴うシステム負荷を考慮した実験結果を表 3 に示す．各表中の遷移の項目は，API 呼び出しパターンの遷移数を表しており，例えば，$API1 \rightarrow API2$ のように，ある API 呼び出し後に次の API を呼び出したパターンの遷移数は 2 となる．また，$API1 \rightarrow API2 \rightarrow API3$ のように，ある API 呼び出し後に次の API を呼び出し，その後に次の API を呼び出したパターンの遷移数は 3 となる．なお，提案手法では，ある API 呼び出し後に次の API を呼び出すまでに経過する時間を想定しているため，遷移数 1 は考えないこととする．既存手法の遷移のみを考慮した状況下において，TNR は 100.0%，TPR は 96.7%と高い精度となっており，誤検知の検体数は，2 個のみであった．

理由の 1 つとして，CCCDataset の API 遷移と正規プログラムの API 遷移がほとんど重複していないことが考えられる．そのため，API 遷移がどの程度，正規プログラムとマルウェアで重複しているのかの調査を行った．調査対象は，遷移 2 の既存手法と提案手法である．API 遷移のみを考慮した既存手法では，1823 個の遷移中，102 個が重複していた．API 遷移とそれに伴う経過時間を考慮した提案手法では，1888 個の遷移中，102 個が重複しており，遷移とそれに伴うメモリ使用量を考慮した提案手法では，1930 個の遷移中，143 個が重複していた．前述のように，

表 1　遷移のみ考慮した検知結果（CCCDataset）
Table 1　Detection results considering API call patterns (CCCDataset)

遷移	TPR	TNR	FPR	FNR	Accuracy
1	96.7%	100.0%	0.0%	33.0%	98.4%
2	96.7%	100.0%	0.0%	33.0%	98.4%
3	96.7%	100.0%	0.0%	33.0%	98.4%
4	96.7%	100.0%	0.0%	33.0%	98.4%

表 2　遷移と経過時間を考慮した検知結果（CCCDataset）
Table 2　Detection results considering API call patterns and elapsed time (CCCDataset)

遷移	TPR	TNR	FPR	FNR	Accuracy
2	96.7%	100.0%	0.0%	33.0%	98.4%
3	96.7%	100.0%	0.0%	33.0%	98.4%
4	96.7%	100.0%	0.0%	33.0%	98.4%

表 3　遷移とメモリ使用量を考慮した検知結果（CCCDataset）
Table 3　Detection results considering API call patterns and memory usage (CCCDataset)

遷移	TPR	TNR	FPR	FNR	Accuracy
2	96.7%	100.0%	0.0%	33.0%	98.4%
3	98.3%	91.7%	8.3%	1.7%	95.0%
4	98.3%	90.0%	10.0%	1.7%	94.2%

CCCDataset の API 遷移と正規プログラムの API 遷移はほとんど重複していなかったため，API 遷移のみを考慮した既存手法でも高い精度の検知率となっている．今後は，正規プログラムの API 遷移がある程度重複しているマルウェア検体を用いた実験が必要であると考えられる．

もう 1 つの理由としては，CCCDataset はボット検体のデータセットのため，多種多様な挙動は行っていないため，多くのマルウェア検体で同じ API 遷移が多く発生していたためと考えられる．今後は，検体の動作時間を延ばし，より多くの API 遷移が動作するマルウェア検体での実験・評価が必要であると考えられる．また，CCCDataset におけるボット検体の API 遷移の特徴に関しては，今後詳細な分析を実施する予定である．

4.4.2 独自収集マルウェアの検知精度

独自に収集したマルウェアと正規プログラムを用いて，マルウェアの検知を行った．既存手法の遷移のみを考慮した場合の実験と提案手法の遷移とそれに伴う時間間隔を考慮した実験，提案手法の遷移とそれに伴うシステム負荷を考慮した実験を行った．既存手法の遷移のみを考慮した場合の実験結果を表 4 に示す．また，提案手法の遷移とそれに伴う時間間隔を考慮した実験結果を表 5 に，提案手法の遷移とそれに伴うシステム負荷を考慮した実験結果を表 6 に示す．既存手法の遷移のみを考慮した場合の Accuracy の実験結果は，遷移 1 と遷移 3 が高い精度となっており，それぞれ 69.0%と 68.4%である．一方，提案手法の遷移と

表 4　遷移のみを考慮した検知結果
Table 4　Detection results considering API call patterns

遷移	TPR	TNR	FPR	FNR	Accuracy
1	47.1%	90.8%	9.2%	52.9%	69.0%
2	34.5%	96.6%	3.4%	65.5%	65.6%
3	39.1%	97.7%	2.3%	60.9%	68.4%
4	37.9%	97.7%	2.3%	62.1%	67.8%

表 5　遷移と時間経過を考慮した検知結果
Table 5　Detection results considering API call patterns and elapsed time

遷移	TPR	TNR	FPR	FNR	Accuracy
2	43.7%	93.1%	6.9%	56.3%	68.4%
3	55.2%	93.1%	6.9%	44.8%	74.2%
4	75.9%	52.9%	47.1%	24.1%	64.4%

表 6　遷移とメモリ使用量を考慮した検知結果
Table 6　Detection results considering API call patterns and memory usage

遷移	TPR	TNR	FPR	FNR	Accuracy
2	49.4%	93.1%	6.9%	50.6%	71.3%
3	49.4%	90.8%	9.2%	50.6%	70.1%
4	43.7%	95.4%	4.6%	56.3%	69.6%

それに伴う時間間隔を考慮した場合の Accuracy の実験結果は，遷移 3 が最も高い精度となっており，74.2%であり，提案手法の遷移とそれに伴うシステム負荷を考慮した場合の Accuracy の実験結果は，遷移 2 が最も高い精度となっており，71.3%である．既存手法で最も値を示す遷移 1 より，提案手法の遷移とそれに伴う時間間隔を考慮した遷移 3 は 5.2%改善され，提案手法の遷移とそれに伴うシステム負荷を考慮した遷移 2 は 2.3%改善されている．また，TPR の値が 50%程度であり，マルウェア検体を高い精度で検知できていないことが分かる．正規プログラムの検知については，TNR の値が高く，正規プログラムを高い精度で検知ができている．

検知の精度は，マルウェアの API 遷移と正規プログラムの API 遷移の回数，最大値，最小値，平均，分散の差に依存すると考えられるが，十分な分析結果が得られていない．今後は，詳細な分析を行い，高精度を示す要因を特定する予定である．

5. まとめと今後の課題

本研究では，先行研究 [1] の結果を基に，API 呼び出しパターン，API 呼び出しによる経過時間とシステム負荷に関する特徴量を用いたマルウェア検知手法を提案し，評価した．CCCDataset2013 の検知精度では，API 遷移がほとんど重複していなかったため，提案手法と既存手法の両手法において高い精度の検知率となった．今後は，検体の動作時間の延ばし，正規プログラムの API 遷移がある程度重複しているマルウェア検体を用いた実験が必要であると考えれる．また，独自収集のマルウェアの検知精度では，提案手法を用いることで，Accuracy が 5%程度改善された．しかし，検知精度に関する分析が十分に行えていないので，今後は詳細な分析を行い，高い精度を示す要因の特定を行う予定である．加えて，マルウェアの種別や機能の分類に関する提案も行う予定である．

参考文献

[1] 佐藤 順子 , 三須 剛史 , 花田 真樹 , 鈴木 英男 , 布広 永示 , “API 呼び出しとシステム負荷を用いたマルウェアの特徴抽出に関する一検討,” コンピュータセキュリティシンポジウム 2016 論文集 , No.2 pp. 305–309, 2016-10-04.

[2] 平成 25 年 情報通信白書 総務省, http://www.soumu.go.jp/johotsusintokei/whitepaper/ja/h25/html/nc132110.html, 参照 July 28, 2017.

[3] 情報セキュリティ 10 大脅威 2017 情報処理推進機構, https://www.ipa.go.jp/security/vuln/10threats2017.html, 参照 July 28, 2017.

[4] Thriving Beyond The Operating System: Financial Threat Group Targets Volume Boot Record FireEye, https://www.fireeye.com/blog/threat-research/2015/12/fin1-targets-boot-record.html, 参照 July 28,2017

[5] 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 情報・システム , Vol.2, No.179, 2014-03-04.

[6] 川口 直人 , 面 和成 “動的解析ログの API を用いた機能に基づくマルウェア分類” 暗号と情報セキュリティシンポジウム 2015.

[7] 笠間 貴弘 , 吉岡 克成 , 井上 大介 , 松本 勉 , “実行毎の挙動の差異に基づくマルウェア検知手法,” コンピュータセキュリティシンポジウム 2011 論文集 , No.3 pp. 726-731, 2011-10-12.

[8] 酒井 崇裕 , 竹森 敬祐 , 安藤類央 , 西垣 正勝 , “侵入挙動の反復性を用いたボット検知手法,” コンピュータセキュリティシンポジウム 2009 論文集, Vol.51 No.9 pp. 1591-1599, 2010-09-15.

[9] 溝口 誠一郎 , 笠原 義晃 , 堀 良彰 , 櫻井 幸一 , “機械的通信挙動モデルに基づく階層型クラスタリングによるボット検知手法,” 情報処理学会論文誌, Vol.54 No.3, pp. 1087-1098, 2013-03-15.

[10] CuckooSandbox https://www.Cuckoosandbox.org/

[11] Vector http://www.vector.co.jp/

[12] 高田雄太, 寺田真敏, 村上純一, 笠間貴弘, 吉岡克成, 畑田充弘, “マルウェア対策のための研究用データセット～MWS 2016 Datasets ～,” 情報処理学会, Vol.2016-CSEC-74, No.17, 2016-07-07.

[13] VirusTotal https://www.virustotal.com/ja/

[14] QueryPerformanceCounter https://msdn.microsoft.com/ja-jp/library/cc410966.aspx, 参照 Aug 19, 2017.

[15] QueryPerformanceFrequency https://msdn.microsoft.com/ja-jp/library/cc410968.aspx, 参照 Aug 19, 2017.