Vol.2014-DPS-158 No.33
Vol.2014-CSEC-64 No.33
2014/3/7

不審プロセス特定手法の実装及び評価

山本 匠† 河内 清人† 桜井 鐘治†

通信の特徴を基にマルウェアによる通信を特定する技術が提案されている．しかし，マルウェアの巧妙化や正規アプリケーションの通信の多様化により，通信の特徴からだけではマルウェアの通信と正規アプリケーションの通信とを正確に切り分けることが困難となっている．そこで著者らは，プロセスのメモリイメージを解析することで，不審な通信を生成するプロセスがマルウェアかどうかを判定するプロセス解析システムを提案している．提案システムでは，正規プロセスに不正なコードを注入することで同プロセスになりすますタイプのマルウェアに注目し，感染していないことが保証された仮想マシン上のプロセスと検査対象のプロセスのメモリイメージを比較することで注入されたコードを特定する．さらに同コードの特徴を解析し，同コードが不正なものかを判定する．本システムでは，正規プロセスの正常な動作に関する情報をあらかじめ用意する必要が無い．プロトタイプシステムを実装し，マルウェアと正規プロセスをそれぞれ 4 種類ずつ用いて評価を行った．結果として 1 件ずつ検知漏れと誤検知が発生した．またヒープの抽出に時間を要してしまっていることが判明した．今後は，検知精度及び処理性能の改善を行っていく．

Implementation and evaluation of a malicious process detection method

Takumi Yamamoto† Kiyoto Kawauchi† Shoji Sakurai†

Malwares' communication detection methods based on communication characteristics have been proposed. However as malwares are getting more sophisticated and legitimate SWs' communication is getting diverse, it becomes harder to correctly tell malwares' communication and legitimate SWs' communication apart. Thus we have proposed a method to check if a process generating suspicious communication is malicious or not. This method focuses on malwares which impersonate a legitimate process by injecting malicious codes into the process. This method extracts two process images. One is obtained from the target process generating suspicious communication. The other is obtained by executing the same executable as the target process in a clean Virtual Machine. Then the two process images are compared to extract injected codes. Finally the codes are verified whether the codes are malicious or not. This method does not require defining characteristics of legitimate processes in advance. In this paper, we implemented a prototype system of the method and carried out a preliminary experiment with four malwares and four legitimate SWs. One false positive and one false negative were happened. In addition, we found it took considerably long time in heap extraction. We will devise the accuracy and the speed of prototype system as a future work.

1. はじめに

近年，新しいセキュリティ脅威として，特定企業や組織をねらい，執拗に攻撃を行う Advanced Persistent Threat（APT）と呼ばれる“標的型攻撃”が顕在化している[1,2]. APT では，標的とする組織の PC にメールによってマルウェアを感染させ，感染したマルウェアは外部の攻撃者のサーバと通信を行い，新しい攻撃プログラムのダウンロードや組織の IT システム内の機密情報の送信を行う．

このような APT への対策として侵入検知システム（IDS）によるマルウェア通信の検知が挙げられる．IDS では，既知のマルウェアに特有の通信パターン（シグネチャ）が，監視対象の通信の中に含まれているかを確認する. しかし，昨今のマルウェアは，通信を暗号化する[3]ことでシグネチャによるマッチングを回避するため，IDS での検知が困難となっている．

そのため，ヘッダ情報や通信トラフィックなど様々な情報を活用し，マルウェアによる不審な通信を特定する手法（以降，不審通信検知システムと呼ぶ）が提案されている[4-7]．ただし正規アプリケーションの通信の多様化により，マルウェアとよく似た通信を出す正規アプリケーションが存在するため不審な通信を出しているプロセスがマルウェアかどうかをさらに検査する必要がある．

不審通信検知システムによって不審な通信を出している PC と送信元ポート番号が特定されるため，PC 上で不審な通信を出しているプロセスを特定することができる．厳密に運用された正規プロセスに関するホワイトリストを用いることで，ホワイトリストに含まれないプロセスを不正なプロセスとして特定することが可能である．しかし，ホワイトリストに含まれる正規プロセスになりすますタイプのマルウェアを見逃す可能性があり, 検知漏れの課題が残る．

文献[8,9]では，正規プロセスの正常な動作に関する情報をあらかじめ収集しておき，正常な動作に反した処理を行ったプロセスを，正規プロセスになりすますタイプのマルウェアと判定する．同手法では，正規プロセスになりすますタイプのマルウェアも効果的に特定することができるが，正規プロセスの正常な動作に関する情報をあらかじめ解析し収集しておかなければならず，手間がかかる．

このような問題に対して，著者らは，正規プロセスになりすますタイプのマルウェアに注目し，既存の不審通信検知システムにおける検知漏れの課題を解決するプロセス解析システムを提案している[10]．正規プロセスになりすま

† 三菱電機株式会社
Mitsubishi Electric Corporation

Vol.2014-DPS-158 No.33
Vol.2014-CSEC-64 No.33
2014/3/7

すタイプのマルウェアの多くは，正規プロセス（例えば，Explorer）に外部から不正コードを注入しそれを実行させることで，同プロセスに不正を行わせる．

文献[10]で著者らが提案したシステムでは，不審通信検知システムが出すアラートをトリガとして，不審な通信を出している PC とプロセスを特定する．アラートには，不審な通信を出している PC の IP アドレスと送信元ポート番号などの情報が含められる．特定したプロセスのメモリ上に不正なコードが注入されていないかを解析することで，アラートのあがった通信がマルウェアによるものなのかを判定する．なお同システムでは，文献[8,9]のように，正規プロセスの正常な動作に関する情報をあらかじめ解析し収集する必要はない．

本稿では，文献[10]で著者らが提案したシステムのプロトタイプを実装し，簡単な評価を行った結果を報告する．以下，2 章でプロセスのメモリを解析することで正規プロセスになりすますタイプのマルウェアを特定する既存研究について紹介し，3 章でプロセス解析システムについて述べる．4 章と 5 章で，提案方式の評価及び考察についてそれぞれ示す．6 章で本稿をまとめる．

2. 関連研究

本章では，プロセスのメモリを解析することで，正規プロセスになりすますタイプのマルウェアを特定する既存研究を紹介する．

文献[8]では，あらかじめプログラムを静的に解析し，同プログラムのコールグラフを作成しておく．同プログラムを実行し起動されたプロセスの中で，同コールグラフに反した関数呼び出しがあれば，同プロセスを正規プロセスになりすますタイプのマルウェアと判定する．

本手法では，正確なコールグラフを作成するコストやプログラム更新に伴いコールグラフを管理するコストに課題が残る．さらに Packer によって圧縮された正規プログラムのプロセスを誤検知する恐れがある．

文献[9]では，プログラムを静的及び動的に解析し，あらかじめ同プログラムから呼び出される関数のテーブルを作成しておく．同プログラムを実行し起動されたプロセスの中で，テーブルに含まれない関数が呼び出された場合に，同プロセスを正規プロセスになりすますタイプのマルウェアと判定する．

本手法は，あらかじめテーブルを作成しておく必要があるため手間がかかる．またテーブルに記録された関数を悪用し不正を実行するマルウェアは特定することができない．

3. 提案システム

3.1 コンセプト

本稿で提案するプロセス解析システムは，図 1 に示すように，不審通信検知システムと連携して動作することを想定している．不審通信検知システムは組織のネットワークを監視する場所，例えば IDS と同じ場所に設置される．プロセス解析システムは組織内の PC 全てに導入される．

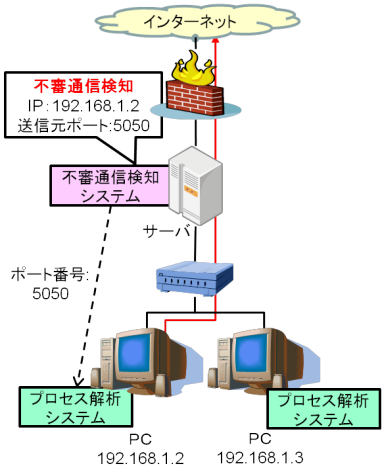

図 1　想定する環境

不審通信検知システムが不審な通信を生成している PC の IP アドレスと送信元ポート番号を特定し，同 IP アドレスに対応する PC にアラートをあげる．アラートには不審な通信の送信元ポート番号を含む．

アラートのあがった PC 上のプロセス解析システムは，送信元ポート番号から不審な通信を生成しているプロセスを特定する．さらに，同プロセスに対応する実行ファイルを，マルウェアに感染していないクリーンな仮想マシン（テンプレートシステム）上で実行しプロセスを起動する．2 つのプロセスのメモリを比較し，同通信がマルウェアによるものなのかを判定する．

図 2 にプロセス解析システムの構成を示す．プロセス解析システムは検査対象の PC（検査対象 PC）のホスト OS 上で動作し，ホスト OS 上の一般アプリケーションのメモリを解析する．テンプレートシステムは仮想マシンとして用意され，テンプレートシステム上でもホスト OS と同じアプリケーションが実行される．

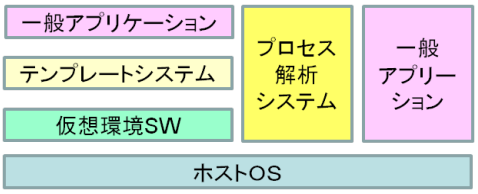

図 2　システム構成

3.2 提案方式の詳細

プロセス解析システムは不審プロセス特定処理とテンプレートシステム更新処理から構成される．

3.2.1 不審プロセス特定処理

不審プロセス特定処理の流れを図 3 に示す．各処理については以下で説明する．

① プロセスの特定

OS の標準コマンドの netstat を利用し，アラートで通知されたポートを利用しているプロセスを特定する．特定し

Vol.2014-DPS-158 No.33
Vol.2014-CSEC-64 No.33
2014/3/7

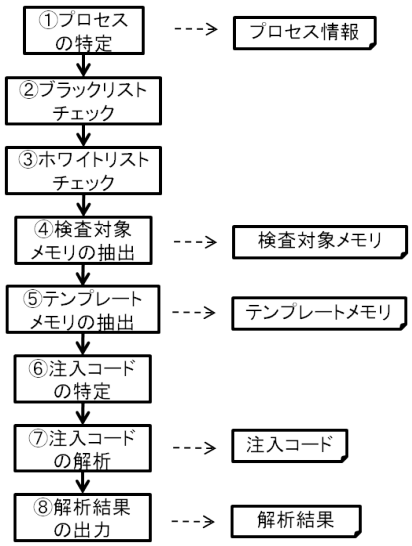

図 3　不審プロセス特定処理の流れ

たプロセスの実行ファイル名，パス，実行ファイルのハッシュ値，バージョン情報などのプロセスに関する情報を抽出しておく．

② ブラックリストチェック

①で抽出したプロセスに関する情報がブラックリストに含まれているかどうかを確認する．ブラックリストはアンチウィルスベンダやセキュリティの専門機関が公表する情報を基にあらかじめ用意しておく．

ブラックリストに含まれていた場合，同プロセスがマルウェアであることをユーザに通知する．含まれていない場合，以降の処理を行う．

③ ホワイトリストチェック

プロセスの名前がホワイトリストに含まれているかどうかを確認する．ホワイトリストは組織で利用可能な正規アプリケーションのプロセス名のリストである．

ホワイトリストに含まれていない場合，同プロセスがマルウェアの可能性が高いことをユーザに通知する．含まれている場合，以降の処理を行う．

④ 検査対象メモリの抽出

①で特定したプロセスのメモリイメージ（検査対象メモリ）を抽出する．検査対象メモリには，同プロセスの PE ヘッダに含まれる情報，テキスト領域（機械語），データ領域及びヒープ領域を含む．メモリのアクセス保護属性に関する情報についても抽出しておく．

⑤ テンプレートメモリの抽出

①で特定したプロセスに対応する実行ファイルを，テンプレートシステム上で実行しプロセスを起動する．テンプレートシステム上の同プロセスのメモリイメージ（テンプレートメモリ）を抽出する．テンプレートメモリに含まれる情報は検査対象メモリと同じである．

例えば①で特定したプロセスが Explorer ならば，テンプレートシステム上でも Explorer を起動し，起動されたプロセスのメモリイメージを抽出する．

なお，テンプレートメモリの抽出は，仮想環境ソフトウェア（ハイパバイザ）のインターフェースを介して行われる．テンプレートシステム上でテンプレートメモリを抽出するためのスクリプトをホスト OS から実行する．それ以外にも，テンプレートメモリを抽出するための機能をハイパバイザに追加することでも対応可能であると考える．

⑥ 注入コードの特定

● テキスト領域

検査対象メモリとテンプレートメモリにおける対応するテキスト領域を比較する（図 4）．対応するアドレス同士でバイナリの差分を計算する．検査対象メモリにおける差分を注入コードとして抽出する．本稿において，注入コードはアクセス保護属性が同一の連続するメモリ領域にあると仮定している．

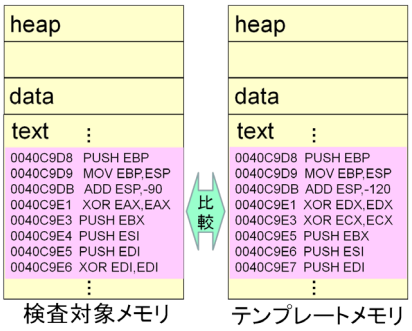

図 4　テキスト領域の比較

●データ領域

検査対象メモリとテンプレートメモリにおける対応するデータ領域を比較する（図 5）．比較対象とする領域は，アクセス保護属性が実行可能な領域とする．対応するアドレス同士でバイナリの差分を計算する．検査対象メモリにおける差分を注入コードとして抽出する．

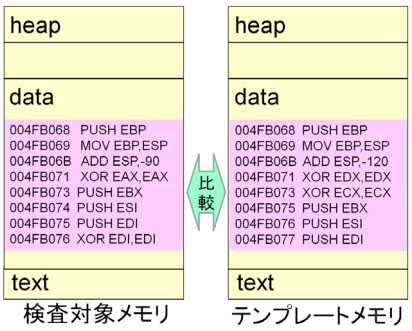

図 5　データ領域の比較

●ヒープ領域

検査対象メモリとテンプレートメモリにおける対応するヒープ領域を比較する（図 6）．比較対象とする領域は，アクセス保護属性が実行可能な領域とする．

ヒープ領域は確保する度に割り当てられるアドレスが変わるため，アドレスの対応だけで直接メモリを比較することはできない．ヒープは割り当てられた領域ごとブロック（ヒープブロック）として OS が管理している．

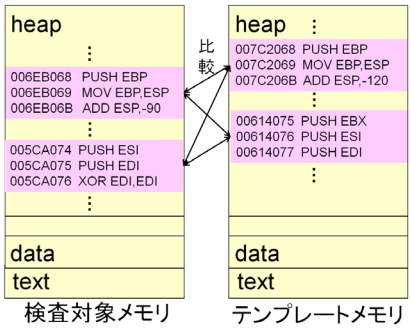

図 6　ヒープ領域の比較

ここで検査対象メモリとテンプレートメモリのヒープブロックの集合をそれぞれ Hc と Ht とする．全ての Hc の要素と Ht 要素について距離 dist(c,t)(c∈Hc, t∈Ht)を計算する．dist は入力された 2 つのブロックの距離を計算する関数である．距離 dist(c,t)の計算には，距離関数(例えば編集距離やハミング距離)を利用する．距離の計算には，ヒープブロック中のオペコードのみを用いる．これは，オペランドにはヒープブロックが配置されるアドレスに伴い変化する情報が含まれうるからである．

距離 dist(c,t)が既定の閾値以下となるブロックのペア(c,t)を抽出し，距離の昇順にペアリスト P に登録する．登録時に，ペア(c,t)のどちらかの要素(c または t)を持つペアが，ペアリスト P に既に登録されている場合，ペア(c,t)をペアリスト P に登録しない．

登録処理終了後，ペアリスト P の中にペアとして登録されていない要素が Hc の中にあれば，同要素（検査対象メモリのヒープブロック）を注入コードとする．

なお計算処理を減らすために，テキスト領域，データ領域，ヒープ領域の注入コードの特定において，サイズが異なる実行可能な領域同士の比較は，省略している．

⑦ 注入コードの解析

注入コードの特徴を解析し，同コードが不正なものかを調査する．提案方式で利用する不正コードの特徴の一部を記載する．

(a) 注入コード内での API 呼び出し

通信，レジストリ操作，プロセス操作などのマルウェアが不正によく使う API の呼び出しが注入コードで行われている場合，同注入コードは不正なものである可能性が高い．

(b) 注入コード内の暗号ロジック

注入コードに独自の暗号ロジックが含まれている場合，同注入コードは不正なものである可能性が高い．

正規のプログラムであれば，OS が提供する暗号 API や公開されている暗号ライブラリを活用して暗号機能を実装することが一般的である．一方，マルウェアには独自に作成された暗号ロジックが利用されていることがある．暗号ロジックの特定については，文献[11,12]を参照されたい．

⑧ 解析結果の出力

⑥で発見された注入コードに，⑦の特徴のいくつかが含まれていれば，同コードは不正なコードの可能性が高い．そのため，同コードが注入されているプロセスは，マルウェアの可能性が高いことを，ユーザに通知する．

3.2.2 テンプレートシステム更新処理

テンプレートシステムにインストールされているアプリケーションやそのバージョンは，検査対象 PC におけるそれら同じものである必要がある．また，テンプレートシステムは，マルウェアが感染していないクリーンなシステムでなければならない．

プロセス解析システムは，テンプレートシステムにインストールされているアプリケーションのプログラム情報をアプリケーションリストで管理する．プログラム情報にはアプリケーションの名前，実行ファイルの名前，実行ファイルのパス，実行ファイルのバージョン，実行ファイルのハッシュ値などの情報が含まれる．

プロセス解析システムは定期的に，検査対象 PC にインストールされているアプリケーションに対してプログラム情報を抽出し，アプリケーションリスト内のプログラム情報と照らし合わせる．アプリケーションリストの中に合致するプログラム情報が無ければ，プログラム情報に合致するアプリケーションをダウンロードする．

ダウンロードしたアプリケーションに対してセキュリティチェックを行い，問題が無ければ，同アプリケーションをテンプレートシステムにインストールする．セキュリティチェックとは，アンチウィルスソフトウェアやウィルス検査サイトを用いたチェックのことを言う．

既に同名のアプリケーションがテンプレートシステムにインストールされている場合，同名のアプリケーションをアンインストールした後に，ダウンロードしたアプリケーションをインストールする．アプリケーションをインストールした後，同アプリケーションのプログラム情報をアプリケーションリストに追加する．

更新処理は，テンプレートメモリの抽出と同様に，ハイパバイザのインターフェースを介して行う．

4. 評価実験

提案方式の実現可能性を確認するため，3.2.1 節の不審プロセス特定処理の一部をプロトタイプ実装し実験を行った．

4.1 評価項目

今回の評価では，検査対象メモリとテンプレートメモリとの比較により，どの程度の数の注入コードが抽出されるかを確認した．また，正規プロセスから得られた注入コードに不正コードの特徴が含まれているかを解析し，提案方式の誤検知の有無を確認した．同様に，マルウェアのプロセスから得られた注入コードに不正コードの特徴が含まれているかを解析し検知漏れの有無を確認した．メモリイメージの抽出，注入コードの特定及び注入コードの解析に要する時間を計測し，提案方式の処理性能について確認した．

4.2 実験環境

3.2.1 節における④～⑦の機能をプロタイプ実装した．実験を簡略化するために，検査対象システム及びテンプレートシステムは共に仮想マシンとした．テンプレートシステム上で抽出したテンプレートメモリについては，手作業で検査対象システムにコピーし，注入コードの特定及び注入コードの解析を行う．仮想マシンのホスト及びゲストの環境を表 1～3 に示す．実験環境上の OS やアプリケーションは，評価の時点で最新となるようにアップデートしている．

表 1　ホストの構成

OS	Windows 7 Professional 64 bit Service Pack 1
CPU	Intel(R) Core(TM) i7-3770 CPU@3.40GHz
メモリ	16.0GB

表 2　ゲストの構成　テンプレートシステム

OS	Windows 7 Professional 32 bit Service Pack 1
プロセッサ数	8
メモリ	4.0 GB
解析環境	Python 2.7

表 3　ゲストの構成　検査対象システム

OS	Windows 7 Professional 32 bit Service Pack 1
プロセッサ数	8
メモリ	4.0 GB
解析環境	Python 2.7

4.3 プロトタイプ

3.2.1 節における④～⑦の機能を，Python を使ってプロトタイプ実装した．テンプレートシステム上で抽出したテンプレートメモリについては，手作業で検査対象システムにコピーし，注入コードの特定及び注入コードの解析を行う．そのため，④，⑤は実装上同じ機能となる．距離関数にはハミング距離を利用した．

プロセスのデバッグや逆アセンブルについては，pydbg や pydasm などのライブラリ[13]を利用した．プロセスのメモリイメージとして，実行可能なテキスト領域，データ領域，ヒープ領域に加え，同プロセスのベースアドレス，サイズ，インポートテーブル，ロードしている DLL の情報（名前，ベースアドレス，サイズ）を抽出した．

4.4 評価手順

表 4 に本評価で用いたマルウェアと同マルウェアがなりすます正規プロセスについて記載する．TSPY_LLAC.SM 及び TROJ_DELF.SMA は，それぞれ Cybergate Rat 及び Cerberus Rat と呼ばれるマルウェアである．Rat サーバ生成ツールを利用し，正規プロセスになりすますトロイの木馬を用意した．その際，Cybergate Rat と Cerberus Rat がそれぞれ Explorer 及び calc になりすますよう設定した．

TROJ_STABUNIQ.A は，正規の iexplore と同様，iexplore の親プロセスと子プロセスから構成される．TROJ_STABUNIQ.A は，親と子の両プロセスに同じコードを注入していた．なお，テンプレートシステム及び検査対象システムにインストールされている iexplore はバージョン 11 である．

誤検知の有無を確認するために，同マルウェアがなりすます正規プロセスに対しても評価を行った．すなわち，誤検知の評価に用いた正規プロセスは，表 4 のなりすまし先の正規プロセスである．

表 4　評価用マルウェアとなりすまし先の正規プロセス

マルウェア	なりすまし先の正規プロセス
TSPY_LLAC.SM	Explorer
TROJ_DELF.SMA	calc
TROJ_STABUNIQ.A	iexplore（子プロセス）
TROJ_STABUNIQ.A	iexplore（親プロセス）
TROJ_SASFIS.NG	Services

表 4 に記載した正規プロセスを，テンプレートシステム上で実行し，テンプレートメモリを抽出した．同様に，検査対象システム上で，表 4 に記載した正規プロセス及びマルウェアを実行し，検査対象メモリを抽出した．1 つのプロセスのメモリイメージを抽出する度に，テンプレートシステムまたは検査対象システムはクリーンな状態に戻した．

正規プロセスについては，起動後に同プロセスをしばらく利用した後に検査対象メモリを抽出した．例えば，iexplore の場合，JavaScript を含む動的なページを閲覧し，その状態で検査対象メモリを抽出した．

マルウェアのプロセスについては，同マルウェアから通信が生成されたことを TCPView[14]及び Wireshark[15]を用いて確認した上で，検査対象メモリを抽出した．同じプロセス名同士の検査対象メモリとテンプレートメモリとを比較し，注入コードを特定した．1 つのプロセスに親プロセスと子プロセスを持つプロセスの場合，親は親，子は子同士で比較を行った．

特定した注入コードについて，不正な特徴が含まれるかどうかを確認した．比較する 2 つのプロセスの選択は手作業であるが，それ以外はスクリプトで自動化した．

4.5 評価結果

表 5～7 に，テンプレートメモリ及び検査対象メモリの抽出に要した時間を記載する．表中，「インポートテーブル」，「ロードしている DLL の情報」，「実行可能なメモリ領域（ヒープ以外）」，「実行可能なメモリ領域（ヒープ）」は，それぞれが示す情報を抽出するために要した時間である．

表 8，9 に不正な注入コードの特定についての評価結果を記載する．表中，「特定した注入コードの数」は本プロトタイプにて特定された注入コードの数である．括弧書きの値は，マルウェアが実際に同プロセスにコードを注入しているところを，著者がデバッガを用いて目視で確認し，計測した注入コードの数である．「特定した不正な注入コードの数」は，特定した注入コードに対し，プロトタイプによっ

表 5　テンプレートメモリの抽出に要した時間[秒]

テンプレート	インポートテーブル	ロードしている DLL の情報	実行可能なメモリ領域（ヒープ以外）	実行可能なメモリ領域（ヒープ）	合計
Explorer	0.163	0.119	0.914	694.334	695.535
calc	0.087	0.009	0.198	5.513	5.808
iexplore（子プロセス）	0.044	0.034	1.037	1975.132	1976.249
iexplore（親プロセス）	0.034	0.035	0.458	16.368	16.896
services	0.110	0.009	0.100	15.345	15.566

表 6　検査対象メモリ（正規プロセス）の抽出に要した時間[秒]

正規プロセス	インポートテーブル	ロードしている DLL の情報	実行可能なメモリ領域（ヒープ以外）	実行可能なメモリ領域（ヒープ）	合計
Explorer	0.244	0.138	1.291	1263.502	1265.177
calc	0.131	0.008	0.181	5.560	5.881
iexplore（子プロセス）	0.031	0.051	1.090	6317.984	6319.159
iexplore（親プロセス）	0.089	0.074	1.218	49.188	50.572
services	0.105	0.008	0.161	10.713	10.990

表 7　検査対象メモリ（正規プロセスになりすますマルウェアのプロセス）の抽出に要した時間[秒]

マルウェアのプロセス	インポートテーブル	ロードしている DLL の情報	実行可能なメモリ領域（ヒープ以外）	実行可能なメモリ領域（ヒープ）	合計
Explorer	0.653	0.142	5.078	935.458	941.348
calc	0.148	0.015	1.639	3.801	5.609
iexplore（子プロセス）	0.093	0.012	0.252	0.283	0.643
iexplore（親プロセス）	0.105	0.055	1.216	2.391	3.770
services	0.103	0.032	1.632	2.453	4.225

表 8 テンプレートメモリ及び検査対象メモリ（正規プロセス）の比較と注入コードの解析

正規プロセス	特定した注入コードの数[個]	特定した不正な注入コードの数[個]	注入コード特定の時間[秒]	注入コード解析の時間[秒]	合計時間[秒]
Explorer	0	0	0.159	0.000	0.159
calc	0	0	0.076	0.000	0.076
iexplore（子プロセス）	5	0	0.330	1.770	2.101
iexplore（親プロセス）	1	0	0.171	0.002	0.173
services	0	0	0.061	0.000	0.061

表 9 テンプレートメモリ及び検査対象メモリ（マルウェアのプロセス）の比較と注入コードの解析

マルウェアのプロセス	特定した注入コードの数[個]	特定した不正な注入コードの数[個]	注入コード特定の時間[秒]	注入コード解析の時間[秒]	合計
Explorer	131(131)	8	0.240	18.005	18.245
calc	103(103)	5	0.078	11.748	11.826
iexplore（子プロセス）	1(1)	0	0.029	0.407	0.436
iexplore（親プロセス）	1(1)	0	0.049	0.408	0.457
services	1(1)	1	0.064	0.385	0.449

て⑦に示す特徴の内，少なくとも 1 つが特定されたコードの数である．また，「注入コード特定の時間」及び「注入コード解析の時間」は，それぞれが示す処理を実行するために要した時間である．

5. 考察

5.1 評価結果について

5.1.1 メモリ抽出の速度

表 5〜7 より，ヒープの抽出に非常に時間がかかっていることがわかる．他の処理は長くても 5 秒程度であるにも関わらず，ヒープの抽出には長いもので 2 時間弱要している．

本プロトタイプでは，ヒープの抽出に，Heap32ListNext 及び Heap32Next を利用してヒープマネージャが管理するヒープを列挙する，よく知られた方法を用いている．しかし，文献[16]によると Windows 7 においては，Heap32Next の処理が非常に遅いことが報告されている．文献[17]では，高速なヒープの列挙の手法が紹介されている．同手法を実装した ProcHeapViewer[17]を試使用したところ，iexplore のヒープを 10 分弱で列挙した．今後は同手法を導入し，ヒープの抽出の高速化を図っていく．

5.1.2 テンプレートメモリ及び検査対象メモリの比較

表 8,9 より，テンプレートメモリ及び検査対象メモリの比較にはそれほど時間を要していないことがわかる．これは，サイズが異なる実行可能な領域同士の比較を省略することで，無駄な距離計算を減らしているためである．

正規プロセスについては，iexplore から注入コードが特定された．これは iexplore が，自身のプロセスに動的にコードを注入し実行するためであった．メモリを抽出するタイミングや状況に応じて同プロセスに展開されるコードが変化するため，テンプレートメモリとの差分が生じる．

マルウェアのプロセスについては，全ての検査対象プロセスについて，マルウェアが注入したコードを特定できた．

5.1.3 注入コードの解析

正規プロセスの iexplore から特定された注入コードを解析したところ，3.2.1 節の⑦で示すような不正な特徴は見られなかった．一方，マルウェアのプロセスについては，iexplore になりすますマルウェア以外からは，注入コードのいずれかに不正な特徴を見つけることができた．

しかし iexplore になりすますマルウェアの注入コードからは不正な特徴を見つけることはできなかった．これは，3.2.1 節の⑦における解析ルールが，同マルウェアの注入コードをカバーしていなかったためと考えられる．今後は同マルウェアの注入コードを解析し，不正な特徴に関するルールの改良を行っていく．

解析時間については，解析対象となる注入コードの数に応じて，時間が長くなっている．マルウェアの注入コードによっては，初期化状態の実行可能領域が保持されているだけの場合がある．解析時間短縮のためにも，そのような領域に対しては，解析を行わないなどの工夫が必要である．

5.2 運用上の課題について

5.2.1 テンプレートの汚染

テンプレートシステムの更新時に，マルウェアに感染したアプリケーションがダウンロードされたとする．セキュリティチェックが同アプリケーションをマルウェアとして判定しない場合，テンプレートシステムにマルウェアが感染する．

テンプレートシステムが汚染されている場合，テンプレートシステムから得られるテンプレートメモリには既に不正なコードが注入されている可能性があり，検査対象メモリとテンプレートメモリの比較では同コードを特定することができない恐れがある．

この問題については現状，ダウンロード先及びダウンロードしたアプリケーションのセキュリティチェック，セキュリティチェックのアップデートなどの対策を徹底することでしか対応する術が無い．

5.2.2 誤検知

今回の提案では，検査対象システムとテンプレートシステムとで実行されるプロセスの同期（プロセスの実行時間，状態，入力の再現）までは行わない．iexplore のように，プロセスによっては起動時または起動後の入力によって，動的に状態を変えるものもある．そのため，たとえ検査対象システムと同一の実行ファイルをテンプレートシステムで実行したとしても，検査対象メモリとテンプレートメモリとの間で差異が生じる可能性があり，注入コードとして特定される恐れがある．

また，Packer や JIT (Just In Time) コンパイラなどを使ったアプリケーションでは，自プロセスのデータ領域やヒープ領域にコードの注入を行う．検査対象メモリとテンプレートメモリを抽出するタイミングによっては，2 つのメモリイメージの間に差異が生じるため，注入コードとして特定される恐れがある．

これらの問題については，マルウェアが注入する不正コードの特徴について調査し，3.2.1 節の⑦における解析ルールに反映していくことで，正規の目的で注入されたコードの誤検知を減らすことが可能であると考える．プロセスの同期については今後の課題として取り組む予定である．

5.2.3 オーバーヘッド

プロセス解析システムは，テンプレートシステムを常時起動させておく，または必要なタイミングで起動させることが求められる．そのため，テンプレートシステムによるオーバーヘッドが課題となる．テンプレートシステムによるオーバーヘッドについては，テンプレートメモリの DB 化により軽減することができると考える．

テンプレートメモリの DB 化では，あらかじめ DB に登録しておいたテンプレートメモリを用いて注入コードの特定を行う．DB に登録されているテンプレートメモリを抽

出したアプリケーションの情報については，3.2.2 節で説明したアプリケーションリストを用いて管理する．検査対象のプロセスの情報（実行ファイルの名前，実行ファイルのパス，実行ファイルのバージョン，実行ファイルのハッシュ値）がアプリケーションリストのプログラム情報に合致しない場合のみ，対応する実行ファイルをテンプレートシステム上で実行し，テンプレートメモリを抽出する．

これにより，テンプレートシステムからテンプレートメモリを抽出する頻度を減らすことができるため，プロセス解析システムのオーバーヘッドを減らすことが可能である．

5.2.4 その他の運用形態

図 1 で示した想定環境では，監視対象の PC 全てにテンプレートシステムを用意する必要がある．監視対象 PC の OS の種類によっては，テンプレートシステムの数だけ OS のライセンスを追加で必要とするため現実的ではない．

この問題については，PC の環境（OS のバージョン，パッチの導入状態，インストールされているアプリケーション）が統一されているような組織において，組織の IT マネージャが管理する 1 つのテンプレートシステムとプロセス解析システムを組織のサーバ上に配置する形態（図 8）をとることで解決することができると考える．本形態では，指定されたプロセスのメモリイメージを抽出する検査対象メモリ抽出システムが，組織内の PC 全てに導入される．

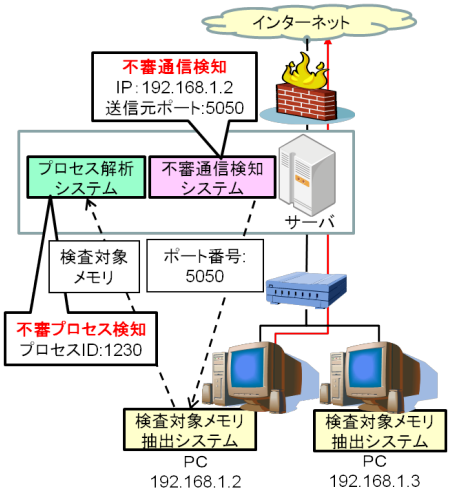

図 8　サーバにプロセス解析システムを導入した環境

本形態では，不審通信検知システムが不審な通信を生成している PC の IP アドレスと送信元ポート番号を特定し，同 IP アドレスに対応する PC にアラートをあげる．アラートのあがった PC 上の検査対象メモリ抽出システムは，不審な通信を生成しているプロセスを特定し，検査対象メモリを抽出する．同システムは検査対象メモリをサーバに送信し，サーバ上のプロセス解析システムが検査対象メモリの解析を実施する．

この形態により，各 PC 上でテンプレートシステムを動作させるオーバーヘッドやライセンスの課題を解決することができると考える．

6. おわりに

本稿では，プロセスのメモリイメージを解析することで，不審な通信を生成するプロセスがマルウェアかどうかを判定するプロセス解析システムを提案した．一部の機能についてプロトタイプを実装し簡単な評価を行った．今後は，評価で得られた課題を基に，提案方式の改良を行い，より多くのマルウェアに対して評価を実施していく予定である．

参考文献

1) シマンテック，"「標的型攻撃」に備えるーサイバー攻撃: 標的型攻撃とは，APT とは"，http://www.symantec.com/ja/jp/theme.jsp?themeid=apt_insight （2014 年 2 月 3 日確認）．
2) kaspersky，"Advanced Persistent Threat（APT）攻撃：今までにない高度なマルウェア"，http://www.kaspersky.co.jp/downloads/pdf/advanced-persistent-threats-not-your-average-malware_kaspersky-endpoint-control-white-paper_jp.pdf （2014 年 2 月 3 日確認）．
3) Shawn Denbow, Matasano Security, "pest control: taming the rats", http://www.matasano.com/research/PEST-CONTROL.pdf （2014 年 2 月 3 日確認）．
4) 鳥居 悟，清水 聡，森永 正信，"RAT 通信監視手法の提案"，コンピュータセキュリティシンポジウム 2012 予稿集．
5) Areej Al-Bataineh and Gregory White, "Analysis and Detection of Malicious Data Exfiltration in Web Traffic", Proceedings of the 7th International Conference on Malicious and Unwanted Software, 2012.
6) J Zhang, C Chen, Y Xiang and W Zhou, "Internet Traffic Classification by Aggregating Correlated Naive Bayes Predictions", IEEE Transactions on Information Forensics and Security, 2013.
7) W Ren and X Wu, "Intelligent Detection of Network Agent Behavior Based on Support Vector Machine", Proceedings of the International Conference on Advanced Computer Theory and Engineering, 2008.
8) C. M. Linn, M. Rajagopalan, S. Baker, C. Collberg, S. K. Debray, J. H. Hartman, “Protecting Against Unexpected System Calls”, Proceedings of the 15th conference on USENIX Security Symposium, 2005.
9) David Wagner, Drew Dean, "Intrusion detection via static analysis", Proceedings of the IEEE Symposium on Security and Privacy, 2001.
10) 山本 匠，河内 清人，桜井 鐘治，“不審プロセス特定手法の提案”，コンピュータセキュリティシンポジウム 2013，2013.
11) Xin Li, Xinyuan Wang, Wentao Chang, "CipherXRay: Exposing Cryptographic Operations and Transient Secrets from Monitored Binary Execution", IEEE Transactions on Dependable and Secure Computing(preprint), 2012.
12) Joan Calvet, Jose M. Fernandez, Jean-Yves Marion, "Aligot: Cryptographic Function Identification in Obfuscated Binary Programs", Proceedings of the 19th ACM Conference on Computer and Communications Security, 2012.
13) Justin Seitz 著，安藤慶一訳，“リバースエンジニアリング Python によるバイナリ解析技法”，O'REILLY.
14) Wireshark, http://www.wireshark.org/download.html（2014 年 2 月 3 日確認）．
15) TCP View, http://technet.microsoft.com/ja-jp/sysinternals/bb897437.aspx（2014 年 2 月 3 日確認）．
16) The Old New Thing, "Why is the Heap32Next function incredibly slow on Windows 7?", http://blogs.msdn.com/b/oldnewthing/archive/2012/03/23/10286665.aspx（2014 年 2 月 3 日確認）．
17) SecurityXploded, "Faster Method to Enumerate Heaps on Windows", http://securityxploded.com/enumheaps.php（2014 年 2 月 3 日確認）．
18) SecurityXploded, "ProcHeapViewer : Fastest Process Heap Viewer Tool", http://securityxploded.com/ProcHeapViewer.php（2014 年 2 月 3 日確認）．