不審プロセス特定手法の提案

山本 匠　　河内 清人　　桜井 鐘治

三菱電機株式会社 情報技術総合研究所
〒247-8501 神奈川県鎌倉市大船 5-1-1
Yamamoto.Takumi@ak.MitsubishiElectric.co.jp

あらまし 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する. 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.

Proposal of a method detecting malicious process

Takumi Yamamoto　　Kiyoto Kawauchi　　Shoji Sakurai
Information Technology R&D Center, Mitsubishi Electric Corporation
5-1-1, Ofuna, Kamakura, Kanagawa, 247-8501, Japan
Yamamoto.Takumi@ak.MitsubishiElectric.co.jp

Abstract 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. Therefore we propose a method to check whether 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.

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の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.

本稿では，特に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する. 正規プロセスになりすますタイプのマルウェアの多くは，正規プロセス（例えば，explorer.exe）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.

本プロセス解析システムでは，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.

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 章で提案手法であるプロセス解析システムについて述べる. 4 章で考察を行い，5 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それを管理する手間に課題が残る. 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.

本手法では，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て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．

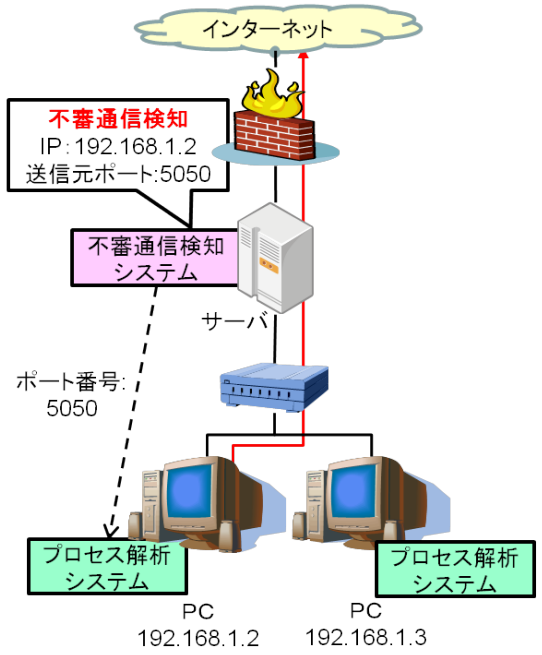

図 1　想定する環境

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．

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．

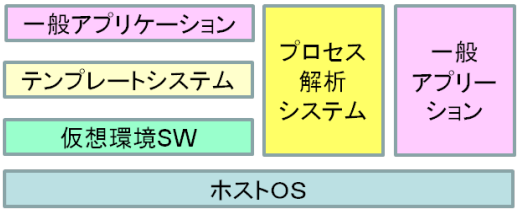

図 2　システム構成

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，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.exe ならば，テンプレートシステム上でも explorer.exe 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．

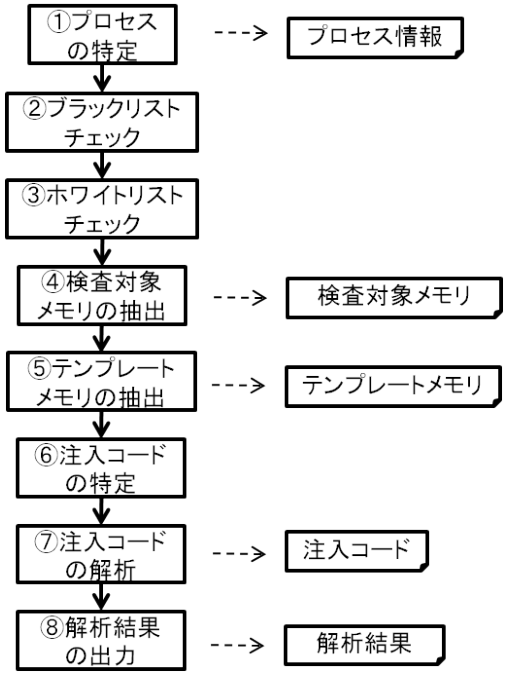

図 3　不審プロセス特定処理の流れ

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．

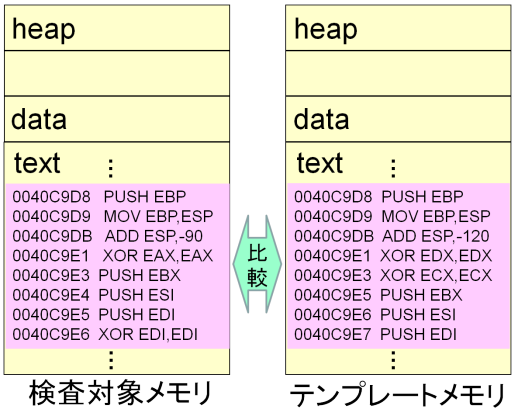

図 4　テキスト領域の比較

●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．

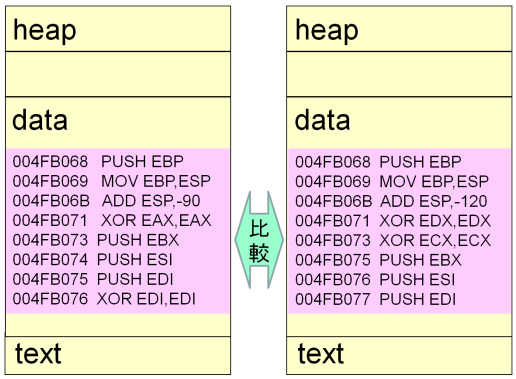

図 5 データ領域の比較

●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.

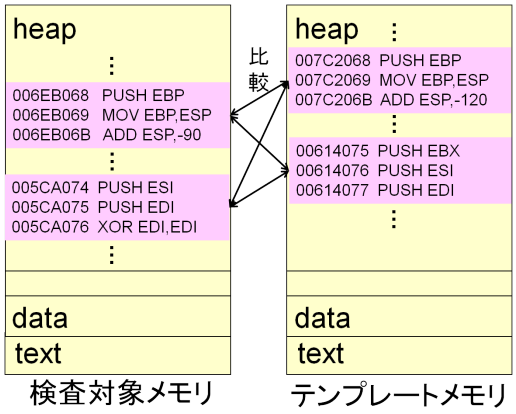

図 6 ヒープ領域の比較

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)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.

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. ここで注目する特徴を以下に示す.

(a) 注入コードへの制御フローの確認

検査対象メモリの注入コード以外の領域から注入コードへのジャンプや関数呼び出しが存在しない場合, 同注入コードは不正なものである可能性が高い.

正規の目的で自プロセスにコードを注入する場合, 自プロセスから同注入コードが実行されるため, 自プロセスから注入コードへの制御の遷移が起こると考えられる(図 7). 一方, 不正に注入された注入コードの場合, 通常外部のプロセスから同コードが実行されるため, 自プロセスから注入コードへの遷移は起こらない.

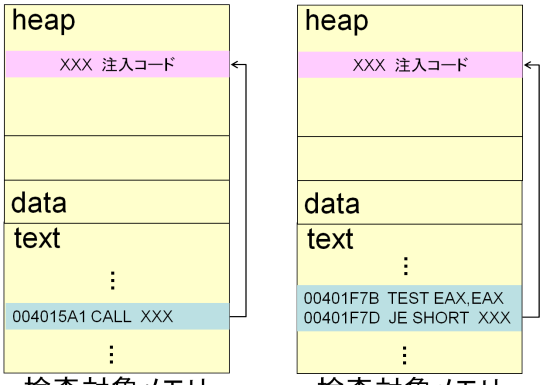

図 7 注入コードへの制御の遷移の例

(b) 注入コード内での API 呼び出しの確認

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利用する API の呼び出しが注入コード内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.

(c) 注入コード内の暗号ロジックの確認

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方法については，既存研究[10,11]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(a)～(c)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を用いて管理する. 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.

テンプレートシステムの更新は，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 考察

検知性能とオーバーヘッドの観点から提案方式について議論する.

4.1 検知性能

●検知漏れ

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.

●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. 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．

4.2 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．

4.3 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．監視対象の PC で動作している 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．

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．

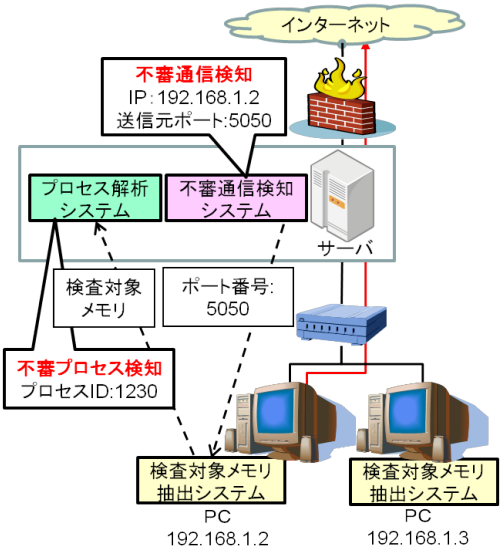

図 8　サーバにプロセス解析システムを導入した環境

5　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．またプロセス解析システムの検知性能とオーバーヘッドに関して考察を行った．

今後はプロセス解析システムのプロトタイプシステムを実装し，実際のマルウェアや正規アプリケーションを用いた検知精度及びオーバーヘッドの評価を行う予定である．

参考文献

[1] シマンテック，"「標的型攻撃」に備える－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（2013 年 8 月 5 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（2013 年 8 月 5 日確認）．

[3] Shawn Denbow, Matasano Security, "pest control: taming the rats", http://www.matasano.com/research/PEST-CONTROL.pdf（2013 年 8 月 5 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] 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.

[11] 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.