情報処理学会論文誌 Vol. 50 No. 9 2118–2126 (Sep. 2009)

マルウェアの耐解析機能を逆用した活動抑止手法の提案

松木隆宏[†1] 新井悠[†1]
寺田真敏[†2] 土居範久[†2]

近年，ウイルス対策ソフトウェアによる検知やパターンファイルの作成に必要となる解析を妨害する機能を有したマルウェアが出現している．特に，ボットの場合にはC&C（コマンド&コントロール）サーバの情報や指令コマンド等の解析作業を妨害するため，解析作業の兆候を検知した場合に，自己の動作を意図的に停止するマルウェアの存在も報告されている．このような耐解析機能は，ウイルス対策ベンダやセキュリティ研究者らによるマルウェアの解析時間を増加させ，結果としてマルウェアによるユーザの被害の拡大につながってしまうことになる．本論文では，耐解析機能を備えたマルウェアによるユーザの被害を低減させることを目的とした新しい対策アプローチを提案する．提案方式は，マルウェアの耐解析機能が動作した際に自己の動作を停止する性質に着目し，これを逆用してマルウェアの動作を抑止する方式である．まず，提案方式の実現例として，耐解析機能の1つであるデバッガ検知機能を逆用し，マルウェアの活動を抑止する手法を示す．次に，デバッガ検知機能を逆用するプロトタイプシステムを実装し，ハニーポットで収集したマルウェア検体を用いた評価を通じて，提案方式の有効性を示す．

Proposal of Malware Activity Control Method Turning Anti-analysis Function to Advantage

Takahiro Matsuki,[†1] Yuu Arai,[†1] Masato Terada[†2] and Norihisa Doi[†2]

In recent years, the malware which hinder antivirus program and obstruct analysis emerge in the wild. Particularly, BOTs make it hard to gather C&C Server's information and analyze their traffic. It is reported that many of them have self-destruction feature against reverse engineering. To analyze these types of malware is cumbersome and time consuming so it makes announcement delay of antivirus vendor's advisories. In this paper, proposes a malware interruption system turning the characteristic to our advantage. First of all, a malware interruption method that beneficially utilizes one of the anti-analysis functions, debugger detection, as an actual example of the proposal. Then the system beneficially utilizing debugger detection is implemented to show efficiency of the proposed method through evaluations using samples collected with honeypot.

1. はじめに

近年，攻撃者からの命令に従ってスパムメール配信やDDoS攻撃を行い，さらに自身のアップデートや別のマルウェアをダウンロードする等の機能を備えたボットの増加は顕著である．

著名なボットの一種であるAgobotは2004年に原作者が逮捕されてはいるが，ソースコードがインターネットに流通しているため，ソースコードを改変した多数の亜種が存在する[1)]．このような多数の亜種の流通は，ウイルス対策ソフトウェアのパターンマッチングによる対策アプローチを困難にしている．さらに，ボットネット実態調査の一環として実施されたソースコードの解析によれば，ボットには自身に対するデバッガ等による解析動作を妨害する機能（以降，耐解析機能）が備わっていることが確認されている[2)]．

具体的には，ボット自身に対するデバッガを用いた解析等，解析作業の兆候を検知すると，自身のプロセスを終了する仕組みである．このような耐解析機能は，ウイルス対策ベンダやセキュリティ研究者らによるマルウェアの解析に必要な時間と労力を増加させ，結果としてマルウェアによる被害を拡大させる．ユーザの被害を低減させるためには，このような耐解析機能に対抗するマルウェア対策が必要な時期にきている．

本論文では，マルウェアが引き起こすユーザ環境での被害を低減させるための新しい対策アプローチを提案する．提案方式は，マルウェアの耐解析機能が動作した際に自己の動作を停止する性質に着目し，耐解析機能を逆用してマルウェアの動作を抑止する方式である．

本論文の構成は次のとおりである．まず，2章で実際のマルウェアのソースコード調査から明らかとなった耐解析機能について述べる．3章では，耐解析機能の仕組みを逆に利用し，これらの機能を備えたマルウェアに解析の兆候を誤認させることで，その動作を抑止させる手法を提案する．また，提案方式の実現例として耐解析機能の1つであるデバッガ検知機能を逆用するシステムの実装について述べる．4章では，実装したプロトタイプシステムを実際のマルウェア検体を用いて評価し，提案方式の有効性を示す．

†1 株式会社ラック サイバーリスク総合研究所
Risk Research Institute of Cyber Space, Little eArth Corporation Co., Ltd.
†2 中央大学
Chuo University

2. ソースコードによる耐解析機能の調査

著者らは，2004 年に出現した著名なボットである Agobot のソースコードをインターネットから入手し，ボット自身の解析を妨害して遅延させる機能，耐解析機能について詳細な調査を行った．本章では，Agobot のソースコードの分析によって明らかとなった耐解析機能の詳細について述べる．

2.1 耐解析機能

ソースコード調査の結果，Agobot には耐解析機能として，デバッガ検知と仮想環境の検知の 2 種類が実装されていることを確認した．これらは，プログラムの初期化処理時に耐解析機能を利用して自身に対する解析の兆候検出を試みる．次に解析の兆候が見られた場合，初期化処理を中断し，自身のプロセスを終了させることで解析作業の妨害を図る．本論文ではこのような，解析を妨害・阻害・遅延させる機能を耐解析機能と定義する．

(1) **デバッガの検知（Anti-Debugging）機能**
- IsDebuggerPresent API による検知
- ブレークポイントの検出
- SoftICE[⋆1] デバッガの検出
- 実行時間の計測

(2) **仮想環境の検知**
- VMware[⋆2] の検知
- Virtual PC[⋆3] の検知

2.2 デバッガの検知

本節では，Agobot に実装されている耐解析機能の 1 つであるデバッガの検知機能について述べる．

2.2.1 IsDebuggerPresent API による検出

Windows[⋆4] では，プロセスが生成されるとカーネルは自動的にプロセス環境ブロック（PEB：Process Environment Block）を割り当てる[3]．PEB 中には，プロセスに関する様々なパラメータが格納されており，その 1 つに BeingDebugged フラグが存在する．BeingDebugged フラグは，プロセス生成時に偽の値（false）が設定され，デバッガがアタッチされると DebugActiveProcess API を介して真の値（true）が設定される．Windows の API である IsDebuggerPresent API は，この BeingDebugged フラグの値を参照する API であり，Agobot ではこの API を利用してデバッガの使用有無を検出する．

2.2.2 ブレークポイントの検出

デバッガは，プログラムにブレークポイントを設定することで，任意の箇所でプログラムの実行を停止することが可能である．このような停止動作は，停止させたいプログラムの実行コード箇所にブレークポイント例外を発生させる機械語命令（INT3 命令）を挿入することで実現されている．プログラムが INT3 命令を実行すると，ブレークポイント例外が発生し，この例外を CPU がハンドリングする．CPU は，デバッガの有無を確認し，例外が発生したプロセスにデバッガがアタッチされていた場合には，以降の処理をデバッガに委ねる．Agobot は，メモリ上に展開された自身の実行コードを定期的にスキャンし，INT3 命令の挿入の有無をチェックすることで，デバッガの使用有無を検知する．

2.2.3 SoftICE の検出

SoftICE は，デバイスドライバ等のデバッグで利用される高性能のカーネルモードデバッガである．SoftICE を，システム上にインストールするとデバッガがカーネルモードにアクセスするための特別な名前空間をファイルシステム（NTFS）上に生成する．Agobot は，この名前空間の存在をチェックすることでシステム上の SoftICE の検出を行う．

2.2.4 コード実行時間（CPU 時間）による検出

一般的なデバッガは，前述のようにプログラムの任意の箇所にブレークポイントを設定することで処理を自由に中断することが可能である．また，シングルステップ実行機能を備えたデバッガを利用することで，CPU における 1 命令ごとにプログラムの実行を中断することもできる．このようにデバッガを用いた解析が行われていた場合には，本来であれば数秒程度で終了する処理の実行に極端に長い時間がかかることになる．Agobot は，耐解析機能の処理の始めと終わりにシステムが起動してからの経過時間を GetTickCount API で取得し，プログラムの 2 点間での値の差分が設定した閾値（Agobot の場合，5,000 ミリ秒）を超えていた場合，デバッガにより解析が行われていると判断する．

2.3 仮想環境の検知

仮想環境の検知は，仮想マシン環境を提供するソフトウェアである VMware ならびに，Virtual PC の環境検知を目的としている．本節では，Agobot の VMware の検知について述べる．

⋆1 商品名称等に関する表示
SoftICE は，米国 Compuware Corp. の米国およびその他の国における登録商標または商標です．
⋆2 VMware は，米国 VMware, Inc. の米国およびその他の国における登録商標または商標です．
⋆3 VirtualPC は，米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です．
⋆4 Windows は，米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です．

VMware において，仮想マシン環境を提供する OS を HostOS，仮想マシン上で稼動する OS を GuestOS と呼ぶ．HostOS と GuestOS は，論理的に分離されていることから，解析の現場でも，安全にマルウェアの解析を行う手法の 1 つとして GuestOS 上でマルウェアを実行するという手法が用いられている．ところが，HostOS と GuestOS は論理的に分離されてはいるが，VMware Backdoor I/O と呼ばれる制御情報を交換するための非公開の機能が存在している[4]．VMware Backdoor I/O は，特定の I/O ポートに対する IN 命令を利用しており，CPU のレジスタに任意のコマンド番号を設定することで様々な制御情報の交換を行っている．Agobot では，VMware Backdoor I/O を用いて VMware のバージョン情報を取得することで，自身が VMware の GuestOS 内で実行されているかどうかを判定する．

3. 耐解析機能を逆用したマルウェアの活動抑止手法の提案

本章では，耐解析機能の仕組みを逆に利用した活動抑止のコンセプトと実現方式について述べる．

3.1 耐解析機能の逆用による活動抑止

耐解析機能について，調査や解析作業を効率化する研究がなされているが，これらはいずれも解析のためである[5)–7)]．本論文では，ユーザのコンピュータ環境においてマルウェアによる被害の低減を目的とした新たな対策手法として，耐解析機能が動作した際に自己の動作を停止する性質に着目し，これを逆に利用することを提案する．このような方法は，研究分野として立ち上がってはいない状況にある．まず，耐解析機能の逆用というコンセプトを，耐解析機能の 1 つである IsDebuggerPresent API によるデバッガ検知機能を例として説明する．

デバッガ検知機能を備えたマルウェアをデバッガで動的解析を行うと，マルウェアは被解析状態であることを検知し，マルウェア自身がプロセスを終了してしまう．このためデバッグによるマルウェアの解析も行えないが，マルウェア自身も悪意のある活動を行うことができない．提案方式は，マルウェアに自らを停止させる状況をユーザのコンピュータ環境において再現することで活動を抑止し，被害の低減を図る手法である．ユーザのコンピュータ環境に侵入するマルウェアをあらかじめ特定することはできないため，デバッガによる解析（アタッチ）はできない．また，すべてのプロセスをデバッガによるデバッグ対象とすることは，リソースの消費が多く現実的ではない．そのため，マルウェアがデバッガ検知に利用する API を，解析状態にあると誤認させる API に置き換えることによって実現する．

3.2 デバッガ検知の逆用による実装

本節では，提案方式の実装について述べる．IsDebuggerPresent API は，この API を呼び出したプロセスが，その時点においてデバッグされているかどうかを真偽値で返す関数である．もし，マルウェアがこの API を呼び出したときにつねに真となる値を返すことができれば，マルウェアに自身が被解析状態であると誤認させることができる．IsDebuggerPresent API に着目した理由は次のとおりである．

(1)　デバッガ検知機能として実装が容易であるため，多数のマルウェアで利用されている可能性が高い．

(2)　他のデバッガ検知手法と比較して，逆用機能の実現が比較的容易である．

まず，提案方式をシステムとして実装する際の要件を以下に示す．ここで，有効性検証のための要件とは，実装したプロトタイプシステムをマルウェア検体を用いて評価するために必要となる条件である．

- **提案方式を実現するための要件**
  - **＜要件 1＞**　IsDebuggerPresent API が返り値として真の値を返すこと
  - **＜要件 2＞**　モジュール（DLL）を動的にロードするマルウェアに有効であること
  - **＜要件 3＞**　プロセスの開始直後のデバッガ検知に対応できること
  - **＜要件 4＞**　新たに生成されるプロセスに対応できること
- **有効性検証のための要件**
  - **＜要件 5＞**　IsDebuggerPresent API の呼び出しの記録が取得できること
  - **＜要件 6＞**　プロセスの終了を検知し，その記録が取得できること

以下では，これらの要件を満たすために解決すべき課題と，その解決策について述べる．

3.2.1 IsDebuggerPresent API のフックとダミー API の実装

＜要件 1＞の IsDebuggerPresent API の返り値の変更は，API フックにより実現する．IsDebuggerPresent API の呼び出しをフックし，代替処理へ遷移させる．Windows ネイティブプログラムに対する API フックの実装手法はいくつか存在する[8)]．代表的な方法は，代替処理を実装した DLL を対象プロセスのメモリ空間にロードし，正規の API による処理を置き換える DLL インジェクションという方式である．

本実装でも，DLL インジェクションをベースとし，正規の IsDebuggerPresent API 呼び出しのキャンセル処理と，つねに真の値を返り値として返す代替処理を行うダミー API を DLL として実装した．

3.2.2 ダミー API アドレスの強制マッピング

通常のプログラムでは，ローダによって DLL 等のモジュールがプロセスのメモリ空間にロードされ，API の実体コードの配置アドレスが，インポートセクションにある Import

```
pfnIsDebuggerPresent IsDebuggerPresent=NULL;
HMODULE hK32=GetModuleHandle("KERNEL32.DLL");
if(!hK32) hK32=LoadLibrary("KERNEL32.DLL");
if(!hK32) {
 IsDbgPresent=(pfnIsDebuggerPresent)GetProcAddress(hK32,"IsDebuggerPresent");
 ...
```

図 1　マルウェア自身によるモジュールのロードと API アドレスの取得
Fig. 1　Dynamic module loading.

Address Table（IAT）に格納される．API フックの実装において，代替処理を実装したダミー API をマッピングする方法として，IAT のアドレス情報をダミー API のアドレスに書き換える IAT Hooking（IAT Patching）がよく用いられる．

しかし，マルウェアには IAT に API のアドレス情報を保持せず，ローダではなく自身で LoadLibrary API により DLL をロードし，GetProcAddress API で動作に必要な API のアドレスの解決（**図 1**）を行うものが存在するため，IAT Hooking では API フックが行えず，**＜要件 2＞**を満たすことができない．

＜要件 2＞を満たすため，ダミー API のマッピング手法を IAT Hooking ではなく，メモリ上にロードされたフック対象 API のコードの先頭にある機械語を JMP 命令に書き換え，ダミー API の存在するアドレスへジャンプさせ，意図する代替処理を実行させる方式（Detours Hook [9]）を用いた．これにより，IAT に API のアドレス情報がないマルウェアに対してもダミー API を強制的にマッピングすることができ，API フックが可能となる．

3.2.3　DLL インジェクション

API フックを行うには，ダミー API を実装した DLL をフック対象プロセスのメモリ空間にインジェクションし，API のアドレスのマッピングを行う必要がある．一般的な方法は，Windows が提供するフック用 API である SetWindowsHookEx API を用いてメッセージフックを行うものである．この手法では，フックの制御は別のプログラムから行う．

しかし，SetWindowsHookEx API を用いるメッセージフックでは，対象プログラムによって最初にウィンドウメッセージが処理されるタイミングで DLL のインジェクションが行われるため，API フック開始のタイミングが遅いという問題がある．このため，通常起動しただけではウィンドウメッセージが発生しないコンソールプログラムやプログラムの開始直後にあるデバッガ検知に対して API フックが間に合わず，**＜要件 3＞**を満たすことができない．実際に，ボット等のマルウェアはコンソールプログラムである場合が多く，デバッガ検知もプログラム開始直後に行われる可能性が高い．

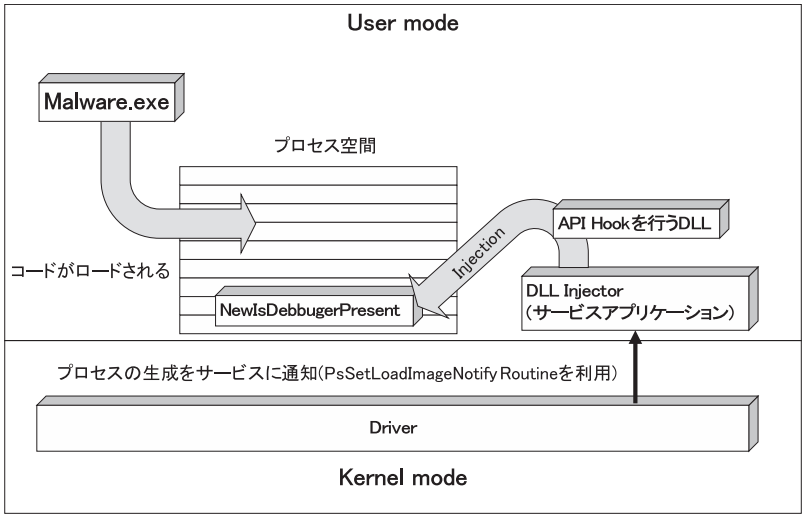

図 2　提案方式の実装システム
Fig. 2　Conception diagram of the system.

この課題については，CreateRemoteThread API を用いて，対象プロセスに LoadLibrary でダミー API を実装した DLL をロードするスレッドを作成，実行させ，ウィンドウメッセージを介さずインジェクションする方法をとることで解決を図った．

3.2.4　カーネルモードでのプロセス生成の監視

実際の環境では，マルウェアがシステムに侵入し，プロセスとして実行されるタイミングは予測不能である．そのため，新たに生成されるすべてのプロセスに対して，ダミー API を実装した DLL をインジェクションし，API フックを適用する仕組み**＜要件 4＞**が必要となる．**＜要件 4＞**について，プロセス生成を監視するカーネルモードで動作するドライバを実装することでこの課題の解決を図った．

ドライバでは，PsSetLoadImageNotifyRoutine API を用いることで，プログラムコードがメモリにロードされ PEB 等の初期化が完了した後，コード実行が開始される前に任意の処理を割り込ませることが可能である．具体的には，ユーザモードで動作するサービスアプリケーションを実装し，生成されたプロセスの情報をこれに通知する．そして，情報を受

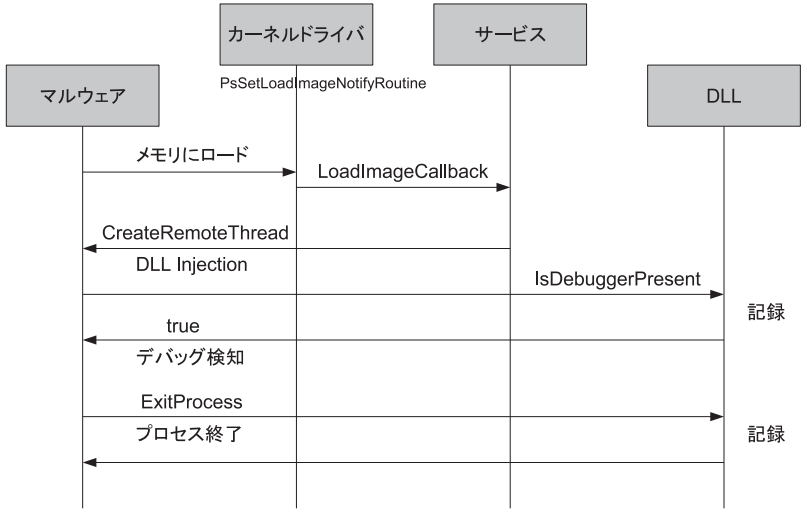

図 3　API フックシーケンス
Fig. 3　API hook sequence.

け取ったサービスアプリケーションが，デバッガ検知を動作させるダミー API を実装した DLL をマルウェアのプロセス空間にインジェクションする形態をとる．この実装の概要を**図 2** に示す．また，DLL をマルウェアのプロセスメモリ空間にインジェクションし，API フックを行うまでの流れを**図 3** に示す．

3.2.5　IsDebuggerPresent API 呼び出しとプロセス終了の記録

提案方式が有効に機能しているかどうかを評価するためには，マルウェアがデバッガ検知のために IsDebuggerPresent API を呼び出したかどうか**＜要件 5＞**と，呼び出した後にマルウェアのプロセスが終了したかどうか**＜要件 6＞**を知る必要がある．

＜要件 6＞のプロセスの終了検知は，マルウェアがプロセス終了時に呼び出す ExitProcess API の呼び出しをフックすることで実現し，API フックの記録をログとして出力する機能をマルウェアにインジェクションする DLL に実装した．ログには，DLL インジェクションの開始と終了，IsDebuggerPresent および ExitProcess の呼び出しとそれぞれが行われた時刻が記録される．

4. 評　　価

本章では，3 章で述べた提案方式のプロトタイプシステムを評価した結果について述べる．

4.1　評 価 項 目

本提案方式を実際のマルウェアに適用した場合にプロセスを終了させて活動を抑止できるかどうかを検証し，さらに有効な検体ファミリの特定と，検体ファミリの亜種における耐解析機能の継承性の観点から評価を行った．

(1)　**プロトタイプシステムの動作検証と有効な検体ファミリの特定**

マルウェアのプロセスを停止させる動作の確認と提案方式が有効であるマルウェアの絞り込みを目的として，プロトタイプシステム作成時の 2006 年時点で可能な限り多くの種類のマルウェア検体を集めた検体群 1 を用いて確認する．

- **検体群 1**
  プロトタイプシステムを実装した 2006 年 5 月までにハニーポットで収集した，ファイルハッシュ値が異なる 6,378 種類のマルウェア検体[⋆1]

(2)　**検体ファミリの亜種における耐解析機能の継承性**

上記の評価から得られた検体ファミリの亜種に対して耐解析機能が継承されていること，それらの検体に対する提案方式の有効性について検体群 2 を用いて確認する．

- **検体群 2**
  2007 年 2 月から 2008 年 10 月までにハニーポットで収集したマルウェアのうち，検体名称が主要なボットファミリに属しており，ファイルハッシュごとの収集数上位 100 に含まれる検体[⋆2]

4.2　手　　順

カーネルドライバ，サービスアプリケーション，DLL から構成されるプロトタイプシステムを VMware 上の GuestOS となる Windows XP SP1 上に構築し，その環境内でマルウェア検体を実行する．評価環境内で 10 秒以内に終了したプロセスに対して，API フックのログを調査し，IsDebuggerPresent API の呼び出し後に ExitProcess API を呼び出し終了しているか否かを集計する．

評価環境は，以下に示すとおりである．

⋆1 2005 年 4 月から 2006 年 5 月までに Telecom-ISAC Japan が運用するハニーポットで収集した検体
⋆2 サイバークリーンセンターが運用するハニーポットで収集した検体

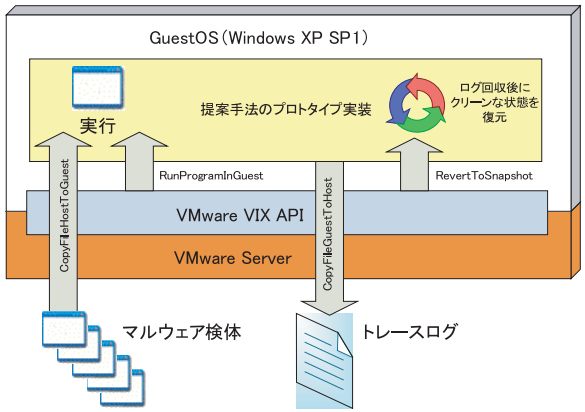

図 4　評価環境
Fig. 4　Evaluation environment.

- ネットワーク接続は行わない（VMware でネットワークインタフェースを切断する）.
- VMware の GuestOS 内であることを隠蔽するファイルやレジストリの変更は加えない.
- 1 つのマルウェア検体を実行するごとに，VMware のスナップショットの復元機能を用いてクリーンな環境に戻す.

なお，大量の検体を利用して評価を行うには下記の処理を繰り返し行う必要があるため，VMware に対する一連の操作を VMware VIX API を用いて自動化するシステム（**図 4**）を構築し評価を実行した.

（1）　評価環境へのマルウェア検体のコピー
（2）　評価環境内でのマルウェア検体の実行
（3）　評価環境からの API フックログのコピー
（4）　評価環境のスナップショット復元

4.3 結　　果

（1）　検体群 1 の評価結果

検体群 1 については，101 種のファイルハッシュを持つマルウェア検体が，IsDebuggerPresent API の呼び出し後に ExitProcess API を呼び出し，10 秒以内に終了した（**表 1**）．このことから，これら 101 種類に対しては提案方式が有効であると判断することができる．ログの分析によって判明したマルウェア検体の挙動の内訳を**表 2** に示す．また，提案方式が有効に作用したマルウェア検体の名称を**表 3** に示す[*1]．

表 1　提案方式が有効に作用した場合のログ
Table 1　Example of API hook log of a successful case.

検体名称：WORM_RBOT.CIU MD5：85c9e71b7a2b716eaca7ffce6d1e3cb2	
時刻（ms）	[イベント] モジュール，API 名
34,038,078	[Init] kernel32.dll, ExitProcess
34,038,671	[Patch] ExitProcess
34,038,703	[Init] kernel32.dll, IsDebuggerPresent
34,038,718	[Patch] IsDebuggerPresent
34,038,968	[Hook] IsDebuggerPresent
34,039,968	[Hook] IsDebuggerPresent
34,040,000	[Hook] ExitProcess
34,040,109	[Unpatch] IsDebuggerPresent
34,040,109	[Term] kernel32.dll, IsDebuggerPresent
34,040,125	[Unpatch] ExitProcess
34,040,125	[Term] kernel32.dll, ExitProcess

表 2　検体群 1 の挙動パターン
Table 2　Behavior of sample - Group1.

プロセスの生存時間	挙動パターン	結果
10 秒以内に終了	IsDebuggerPresent 呼び出しあり	101
	IsDebuggerPresent 呼び出しなし	1,219
	異常終了	1,717
10 秒以上生存	-	3,341
合計		6,378

（2）　検体群 2 の評価結果

検体群 2 については，50 種のファイルハッシュを持つマルウェア検体が IsDebuggerPresent API の呼び出し後に ExitProcess API を呼び出し，10 秒以内に終了した．ログの分析によって判明したマルウェア検体の挙動の内訳を**表 4** に示す．また，提案方式が有効に作用したマルウェア検体名称のリストを**表 5** に示す.

⋆1 検体名称は，トレンドマイクロ社の製品によるもの

表 3　検体群 1 において提案方式が有効に作用した検体
Table 3　Blocked sample list 1.

RBOT ファミリ：56 種	SPYBOT ファミリ：13 種
BKDR_RBOT.DVM	WORM_SPYBOT.A
WORM_RBOT.AJK	WORM_SPYBOT.AHG
WORM_RBOT.ARC	WORM_SPYBOT.AJF
WORM_RBOT.BZR	WORM_SPYBOT.AJG
WORM_RBOT.CAE	WORM_SPYBOT.AJZ
...	...
SDBOT ファミリ：5 種	その他：27 種
WORM_SDBOT.AES	既知検体 6 種
WORM_SDBOT.CIE	未知検体（当時）21 種
WORM_SDBOT.CXL	
WORM_SDBOT.CXO	
WORM_SDBOT.DIJ	

表 4　検体群 2 の挙動パターン
Table 4　Behavior of sample - Group2.

プロセスの生存時間	挙動パターン	結果
10 秒以内に終了	IsDebuggerPresent 呼び出しあり	50
	IsDebuggerPresent 呼び出しなし	36
	異常終了	1
10 秒以上生存	-	13
合計		100

4.4 考　　察

(1) **提案方式の有効性**

検体群 1，検体群 2 の評価結果ともに一部のマルウェアにおいてプロセス生成から 10 秒以内にデバッガ検知機能が動作し，そのプロセスが終了したことから，システム改変や外部への通信による悪意のある活動が抑制され，被害を低減させることができると考える．

活動を抑制できた検体群 1 の 101 種の検体名称（表 3）のうち，74 種がボットファミリの検体であったことと，著名なボットファミリを選択した検体群 2 の 50%という高い割合に有効であることが確認できたことから，提案方式は主にボットに対して有効といえる．また，RBOT，SPYBOT，SDBOT 系のボットは，亜種が非常に多く，被害が多いと報告されている[10]．そのため，これらの検体の活動を抑制できるということは，提案方式の有効性は高いと考える．しかしながら，提案方式は，あくまでマルウェアの活動を抑止するものであり，完全な感染防御，あるいは駆除を行うものではない．また，耐解析機能を持たないマルウェアに関しては有効でないため，パターンマッチングによる従来のマルウェア対策と協調する必要はある．今後，耐解析機能がボットに限らず他の種類のマルウェアに広がった場合，耐解析機能を逆用するという提案方式のアプローチは，既存の対策方法を補強する方法として有効であると考えている．

表 5　検体群 2 において提案方式が有効に作用した検体
Table 5　Blocked sample list 2.

RBOT ファミリ：7 種	SPYBOT ファミリ：7 種
WORM_RBOT.GIH	WORM_SPYBOT.PC
BKDR_RBOT.CJS	WORM_SPYBOT.ADS
WORM_RBOT.QK	WORM_SPYBOT.BEA
WORM_RBOT.GDJ	WORM_SPYBOT.AWL
WORM_RBOT.GLZ	WORM_SPYBOT.RO
WORM_RBOT.CRA	WORM_SPYBOT.ADO
WORM_RBOT.GDS	WORM_SPYBOT.ATH
SDBOT ファミリ：13 種	IRCBOT ファミリ：23 種
WORM_SDBOT.ELL	BKDR_IRCBOT.ABV
WORM_SDBOT.BWZ	BKDR_IRCBOT.AQD
WORM_SDBOT.EYY	BKDR_IRCBOT.ACR
WORM_SDBOT.GAD	WORM_IRCBOT.AHX
WORM_SDBOT.AHI	BKDR_IRCBOT.AFH
...	BKDR_IRCBOT.ABV
	WORM_IRCBOT.JL
	...

(2) **耐解析機能の継承性**

検体群 1（2005 年 4 月から 2006 年 5 月までに収集した検体）と検体群 2（2007 年 2 月から 2008 年 10 月までに収集した検体）による評価結果から，提案方式が有効であった検体に SHA-1 と名称の重複はなかった．これは，2006 年 5 月までに収集されておらず，2007 年以降に収集されたより新しいマルウェアに提案方式が有効な検体を新たに 50 種発見したことを意味する．また，検体群 1 の有効検体のファミリ名が検体群 2 でも確認されたことから，同一ファミリの検体に耐解析機能が備わっていると判断できる．これらのことから，デバッガ検知による耐解析機能は，ボットの同一

ファミリの新しい亜種検体に継承される傾向があり，今後出現することが推測されるボットの同一ファミリの亜種にも提案方式が有効である可能性があると考える．

(3) **耐解析機能の逆用について**

マルウェア自身の機能を逆用して，その活動を抑止するという提案方式は，新しいアプローチの対策である．本論文では，耐解析機能の 1 つのみに着目したが，解明されている他の耐解析機能についても逆用を検討することで有効性のさらなる向上が考えられる．今回実装したデバッガ検知は，マルウェアではない正規のアプリケーションにおいても不正なリバースエンジニアリングからソフトウェアを保護する目的で利用されている．このようなアプリケーションは，提案方式の影響で正常に利用できなくなることが予測されるが，プロセス識別情報をホワイトリストとして保持し，デバッガ検知時に照合を行い，その結果から API の戻り値を変化させることにより回避可能と考える．

また，マルウェア作者が提案方式の存在を知ることによって，提案方式を回避する耐解析機能，逆用が困難な耐解析機能の開発等が考えられる．しかし，マルウェアは様々な観点で高度化が進む傾向にあり，耐解析機能も高度化することが推測される．このことから，継続した耐解析機能の調査ならびに対応が必要であること，提案方式が有効なマルウェアへの対処方法として活用展開する価値があると考えている．

5. おわりに

本論文では，マルウェアの耐解析機能の 1 つである IsDebuggerPresent API を利用したデバッガ検知機能を逆用したマルウェアの動作抑止手法を提案し，評価実験を通して提案方式が有効であることを示した．

また，提案方式については，調査の結果明らかになっている別の耐解析機能についても逆用手法も検討し，それを併用することで活動抑止の効果向上を期待できると考えられる．新たに出現するマルウェアには新しく高度な耐解析機能が実装されている可能性があるため，継続的に耐解析機能の解明を進める必要がある．今後も，耐解析機能の解明とともに，マルウェアの耐解析機能を逆用する対策手法を検討していきたいと考えている．

謝辞　本研究の一部は Telecom-ISAC Japan の支援を受け実施している．本研究を進めるにあたり，有益な助言と協力をいただいた Telecom-ISAC Japan の関係者各位に深く感謝いたします．また論文執筆にあたり，有意義なご指摘をいただいた田中優毅氏に感謝いたします．

参 考 文 献

1) Sophos：ソフォス，Agobot トロイの木馬作成の容疑者ドイツで逮捕と報告 (2004). http://www.sophos.co.jp/pressoffice/news/articles/2004/05/va_agobotarrest.html
2) 高橋正和，村上純一，須藤年章，平原伸昭，佐々木良一：フィールド調査によるボットネットの挙動解析，情報処理学会論文誌，Vol.47, No.8, pp.2512–2523 (2006).
3) NTinternals Teams: PEB. http://undocumented.ntinternals.net/UserMode/Undocumented%20Functions/NT%20Objects/Process/PEB.html
4) Open Virtual Machine Tools: Open Virtual Machine Tools. http://open-vm-tools.sourceforge.net/
5) Honeynet Project: Know your Enemy: Tracking Botnets – Source Code. http://www.honeynet.org/papers/bots/botnet-code.html
6) Nicolas Falliere: Windows Anti-Debug Reference (2007). http://www.securityfocus.com/infocus/1893
7) 株式会社フォティーンフォティ技術研究所：AADebug v0.12 (2007). http://www.fourteenforty.jp/research/freeware.htm
8) Richter, J.（著），長尾高弘，ロングテール（訳）：Advanced Windows 改訂第 4 版，アスキー (2001).
9) Microsoft: Binary Interception of Win32 Functions. http://research.microsoft.com/~galenh/Publications/HuntUsenixNt99.pdf
10) サイバークリーンセンター：平成 19 年度サイバークリーンセンター（CCC）活動報告．https://www.ccc.go.jp/report/h19ccc_report.pdf

(平成 20 年 12 月 1 日受付)
(平成 21 年 6 月 4 日採録)

松木　隆宏（正会員）

（株）ラック サイバーリスク総合研究所研究員．2005 年岡山大学工学部通信ネットワーク工学科卒業．同年（株）ラック入社．ネットワークセキュリティ脅威分析等のセキュリティコンサルティング部門を経て，マルウェアの調査研究，マルウェア対策技術の研究開発に従事．2006 年より，サイバークリーンセンターによるボット対策プロジェクトに参画し，調査研究，ハニーポットの開発，運用支援に従事．2007 年から安心・安全インターネット推進協議会 P2P 研究会構成員．2008 年情報処理学会コンピュータセキュリティ研究会専門委員．CISSP．

新井　悠

（株）ラック サイバーリスク総合研究所所長．2000 年（株）ラック入社．コンピュータセキュリティ研究所にて脆弱性の分析，R&D 部門の統括，ネットワークセキュリティ脅威分析等のコンサルティング業務やセキュリティ・アドバイザを経て，現職．著書・監修書として『ネットワーク攻撃詳解—攻撃のメカニズムから理解するセキュリティ対策』（ソフト・リサーチ・センター），『インシデントレスポンス』（翔泳社）等．2004 年総務省「次世代 IP インフラ研究会」セキュリティWG 構成員，2005 年内閣官房 NIRT（緊急対応支援チーム）研修講師，2006 年経済産業省「ウェブアプリケーションセキュリティガイドライン策定 WG」委員，2007 年総務省「次世代の情報セキュリティ政策に関する研究会」構成員を務める．CISSP．

寺田　真敏（正会員）

（株）日立製作所システム開発研究所主管研究員．1986 年（株）日立製作所入社．システム開発研究所にてネットワークセキュリティの研究に従事．博士（工学）．2002 年から 2006 年慶應義塾大学大学院社会人学生として，JPCERT/CC Vendor Status Notes プロジェクトに参画．2004 年 4 月から JPCERT コーディネーションセンター専門委員，中央大学研究開発機構客員研究員，2004 年 8 月から（独）情報処理推進機構セキュリティセンター研究員，2004 年 10 月から（株）日立製作所 Hitachi Incident Response Team チーフコーディネーションデザイナを兼務．著書に，『基礎からわかる TCP/IP セキュリティ実験』オーム社（2000 年）等．

土居　範久（正会員）

1969 年慶應義塾大学大学院博士課程単位取得退学．慶應義塾大学理工学部教授を経て，2003 年より中央大学理工学部教授，慶應義塾大学名誉教授．工学博士．現在，文部科学省科学技術・学術審議会委員，総務省情報通信審議会委員，世界科学会議（International Council for Science（ICSU））Priority Area Assessment Panel of Scientific Data and Information メンバ，科学技術振興機構（JST）社会技術システムミッションプログラム I「I 情報セキュリティ」研究統括，特定非営利活動法人日本セキュリティ監査協会会長，国際計算機学会（ACM）日本支部長，等．専門はソフトウェアを中心とした計算機科学．