メモリ上に展開されたコードを使うウイルス解析支援システム

市川 幸宏[†,☆] 伊沢 亮一[†,☆☆]
白石 善明[††,☆☆☆] 森井 昌克[†††]

コンピュータウイルスによる被害を軽減させるためには，ネットワーク上において早期に検知し，いち早く廃棄する必要がある．コンピュータウイルスを検知するためには，まずそのコンピュータウイルスを解析する必要がある．通常，その解析はアンチウイルスベンダに所属する技術者によって，基本的にそのウイルスコードを１行１行解析する手法がとられている．亜種も含めて，大量にコンピュータウイルスが発生する現在，その解析能力は飽和状態にあり，ウイルス解析者を支援するシステムの開発が希求されている．本論文では，既知のコンピュータウイルスだけでなく，未知のコンピュータウイルスを解析することを目的として，ウイルス解析者を支援するシステムを提案している．提案システムは，ウイルスコードを直接解析するのではなく，実行時に動作するメモリ上に展開されたコードを解析し，難読化が施されたコードであっても解析が可能となっている．

Unknown Virus Analysis Support System Using Code Loaded on Memory

Sachihiro Ichikawa,[†,☆] Ryoichi Isawa,[†,☆☆]
Yoshiaki Shiraishi[††,☆☆☆] and Masakatu Morii[†††]

This paper presents a design and implementation of automatic virus analysis support system. It is expected that not a binary or disassemble code but a suspicious code expanded on memory is helpful for automating of virus analysis. We take the following approach: 1) execution of a doubtful code on a virtual environment; 2) dumping the object code on memory; 3) disassemble of the dumped code; 4) analysis of the assembly code. By the above approach, we realize a system for supporting computer virus analysis. Although it has been difficult to automate analysis, the system can automatically output a result as same as the technical details of security response issued by anti-virus vendor. In addition, the system can analysis of new virus including one with a difficult analysis.

1. はじめに

コンピュータネットワーク，いわゆるインターネット基盤は社会システムの重要インフラとして定着し，その安定した稼動，および運用が強く求められている．しかしながらその重要性が増すに従って，その安全性を脅かす行為，特にサイバーテロに代表される組織的，および個人の恣意的な不正アクセスが問題となっている[1)]．また，インターネット基盤の出現以前から，コンピュータウイルスの脅威が問題視され，近年のネットワークによる急速かつ広域への拡散は不正アクセス以上の問題となっている[2)]．最近の不正アクセス，特に特定のサーバやサイトを機能不全に陥れる DoS（Denial of Service，サービス拒否）攻撃では，直接的に人手によるコンピュータの操作からの攻撃ではなく，bot と呼ばれる不正なソフトウェアによる攻撃が増加している[3),4)]．この不正なソフトウェアはコンピュータウイルスによって拡散される場合が多く，またバックドアと呼ばれる故意に作られた脆弱性を利用する場合も報告されている．バックドアもコンピュータウイルスによって作成される場合が多い．このように不正アクセスはコンピュータウイルスとの連携の下に行われるようになり，コンピュータウイルスの脅威は著し

† 徳島大学工学部
Faculty of Engineering, The University of Tokushima
†† 近畿大学理工学部
School of Science and Engineering, Kinki University
††† 神戸大学工学部
Faculty of Engineering, Kobe University
☆ 現在，三菱電機株式会社情報技術総合研究所
Presently with Information Technology R&D Center, Mitsubishi Electric Corporation
☆☆ 現在，神戸大学工学部
Presently with Faculty of Engineering, Kobe University
☆☆☆ 現在，名古屋工業大学大学院工学研究科
Presently with Nagoya Institute of Technology

く増加している．

コンピュータウイルスによる被害を回避するためには，ネットワーク上において早期に検知し，いち早く廃棄する必要がある．しかしながら，コンピュータウイルスの拡散機能の高度化だけでなく，インターネット基盤の利用の一般化，広域化，さらにその高速広帯域化から，きわめて短い時間によって広い範囲に拡散が行われる．「ゼロデイアタック（zero-day attack）」とは，OS やシステムの脆弱性が発見されたとき，その対策が十分でない短い期間を衝いて不正アクセスを行うことであるが，コンピュータウイルスでは「ゼロアワーアタック（zero-hour attack）」と呼ばれるように，その最初の発見から 1 時間程度の対策の遅れが，広域への拡散を促し，深刻な被害を及ぼすことになる．

コンピュータウイルスを検知するためには，まず第 1 に，そのコンピュータウイルスを解析する必要がある．特に被害を最小に抑えるためにも，そのコンピュータウイルスの感染機構や発症機構，さらにその症状，つまり発症した場合の具体的な振舞いを明らかにする必要がある[5)]．通常，その解析はアンチウイルスベンダに所属する数少ない熟練した技術者によって，基本的にそのウイルスコードを 1 行 1 行解析する手法がとられている．一部，ウイルスコードの解析の自動化も試みられているものの詳細な報告はない．このように技術者のノウハウ的な手作業による解析が主であり[6)]，亜種も含めて，大量にコンピュータウイルスが発生する現在，その解析能力は飽和状態にあるといえる．したがって，熟練した技術者になりかわって，ウイルスを自動的に解析するシステムの開発が希求されている．

本論文では，亜種を含めた既知のコンピュータウイルスだけでなく，いまだ解析されたことがない未知のコンピュータウイルスを解析することを目的として，そのウイルスの感染機構，発症機構，および発症後の振舞いを推定するために必要な情報を効率良く提示する，ウイルス解析支援システムを提案する．本システムを用いることにより，従来の逆アセンブラやデバッガの機能のみに頼ることに比較して，効率良く短時間でコンピュータウイルスを解析するために必要な情報を抽出，および提示することが可能となる．さらにこれらの出力からコンピュータウイルスの感染，発症，振舞いの概要を陽に得ることが可能となり，それらの結果は，アンチウイルスベンダから適時報告されているセキュリティレスポンス[7)] と同等の内容であることが確かめられる．

提案するシステムの要点は，ウイルスコードを直接解析するのではなく，実行時に動作するメモリ上に展開されたコードを解析することにある．さらに JMP 命令に着目し，大幅にコードを解析する時間を短縮する手法，および実行順に抽出された Windows API と ASCII 文字列を手がかりに，挙動を解析する手法にある．

2. ウイルス解析者のワークフロー

アンチウイルスベンダでは，疑わしいコード（suspicious code）を入手すると，そのコードがウイルスか否かを確認する．そして，ウイルスと判断すれば，ウイルス検知・駆除ソフトに配布するシグネチャと呼ばれるウイルスの特徴情報を作成する．ウイルスによる被害を軽減するためには，シグネチャの配布開始が早くなされなければならない．疑わしいコードの解析を迅速に行うことができれば，シグネチャの作成/配布を早くすることができる．

一般的なウイルス解析者のコード解析は次のような手順となる．

1) バイナリ形式の疑わしいコードから可読な ASCII 文字列を抽出する．抽出した文字列から Windows API/DLL の一部や，ウイルスの特徴的な文字列を確認する．このとき，コードが UPX 圧縮などの既知の難読化処理を施している場合には復号して処理を行う．

2) 疑わしいコードをリバースエンジニアリングツールを使って，逆アセンブルしたコードを 1 ステップごとに実行しながら，発症条件（以下，トリガと呼ぶ）を調べる．

3) 確認したトリガの情報を基に，実環境を整え，疑わしいコードを実行する．

1) は，strings などのバイナリファイルから ASCII 文字を抽出するツールを用いれば容易に行える．抽出された情報は，2) のリバースエンジニアリングを行う際の手がかりとして使用される．

2) は，IDA[8)] などの逆アセンブラとデバッガが組み合わさった，ステップ実行機能とブレークポイント機能などを持つリバースエンジニアリングツールが使われる．解析者は，ステップ実行をしながら，挙動解析を行う．たとえば，ウイルスの 1 つである Netsky をこのようなツールで読み込むと，逆アセンブルコードは 1 万行行に及び，コード量が数万行にのぼるウイルスも珍しくはない．コードの先頭からステップ実行していては解析に時間がかかるために，適当な箇所にブレークポイントを設定し，ある程度の時間の短縮を試みる必要がある．ブレークポイントは，悪意ある操作が行われる箇所の付近に設定される．このブレークポ

イントの選定に，1) で得られた情報が利用される．しかしながら，1) において，著者らが収集した複数のウイルスでは Windows API の名前，ファイルやレジストリの名前およびパス，そしてネットワークアドレスなどの可読な ASCII 文字列はたかだか半分ほどしか取得できず，さらに暗号化/復号機能を含めた難読化機構を有するウイルスでそのような文字列を取得することは困難である．

アンチウイルスベンダが公表しているセキュリティレスポンスに記述されている内容のうち，ウイルスの行為に関わる部分が自動的に抽出できれば，2) を今よりも迅速に行えると期待できる．ウイルスの発生件数が日々増加し，解析に長けた技術者の負荷が増大している状況において，解析に長けていない技術者も手がかりとなる詳細な情報があれば解析しやすくなり，ゼロディアタックの脅威を低減できる．

以下では，暗号化/復号機能を含めた難読化機構を有していても，疑わしいコードが利用する Windows API や操作するレジストリを抽出できる，セキュリティレスポンスと同等の内容を出力するウイルス解析支援システムについて述べる．

3. 提 案 手 法

3.1 仮想環境と不正プロセスのメモリダンプ

コード解析に使用するホストで疑わしいコード（以下では，ターゲットコードと呼ぶ）を実行すると，もしもウイルスであればホストが感染してしまう．そこで，VMware[9] による仮想環境を使ってコード解析を行う．

VMware をインストールしたコード解析を行うホストのことをホストマシン，VMware のゲスト OS をインストールした仮想マシンのことをゲストマシンと以下では呼ぶ．

ゲストマシンでターゲットコードを実行する前のプロセス情報を取得し，ウイルスを実行した後のプロセス情報と比較を行う．ターゲットコードを実行した際に起動した新たなプロセスを不正プロセスと見なす．なお，ターゲットコードの実行中に生成され，終了するプロセスについては，プロセス情報の比較による差分情報では捕捉できないために，提案手法の解析対象からは外れる．そして，解析対象となった不正プロセスをメインモジュールのみメモリダンプし，ダンプしたコードを逆アセンブルする．

メインモジュールだけをダンプするのは，不正プロセスに関わるすべてのメモリ領域をダンプすると，Windows であれば最大で 4 Gbyte の仮想メモリすべてをダンプする可能性があり，その時間を削減するためである．メインモジュールのコードだけで，不正プロセスが使用する Windows API や，操作したレジストリなどの行動を把握する解析方法を 3.2，3.3 節で述べる．

3.2 Windows API の実行手順の解析

ターゲットコードで起動された不正プロセスの Windows API の実行手順の解析は，メモリダンプされたメインモジュールのバイナリコードを逆アセンブルしたアセンブリコードに対して行う．アセンブリコードには，呼び出される Windows API の名前は直接書かれておらず，その Windows API が格納されている番地が書かれている．そこで，次のような手順でアセンブリコードの解析を行う．

(1) Windows API 呼び出し箇所の検出
(2) 呼び出された API の名前を特定

まず，アセンブリコードを先頭からプログラムの実行順に走査し，Windows API が呼び出されている箇所を検出する．このとき，検出箇所を検出順に記録しておく．次に，API 呼び出しを検出した箇所ではどのような API が呼び出されているのかを特定する．

以下で，これらの詳細について述べる．

3.2.1 Windows API 呼び出し箇所の検出

Windows API の呼び出し箇所を検出するために，アセンブリ言語の CALL 命令に着目する．CALL 命令は，Windows API とサブルーチンの呼び出しを行う．CALL 命令のオペランドが，メインモジュール外の番地であれば，それは Windows API を呼び出していると判断し，その CALL 命令が実行される番地とオペランドで指定されている Windows API の番地を記録しておく．メインモジュール内の番地であれば，サブルーチンへ処理が移ると判断する．サブルーチン内に Windows API の呼び出しがあるかもしれないので，メインルーチン内だけでなく，サブルーチン内も逐次，解析を行う．

CALL 命令のオペランドがメインモジュール内の番地の場合，すなわちサブルーチンへ処理が移った場合には，解析処理も呼び出し元の番地に戻ってこなければならない．また，移動先のサブルーチンでさらに CALL 命令による移動がある場合も考慮しなければならない．そこで，**図 1** に示したような次の処理を行う．

step1 現在の番地をスタックに入力し，CALL 命令のオペランドが指す番地に移動する．

step2 1 行ずつ解析処理を行う．RET 命令があった場合，スタックから値を取り出し，取り出した番

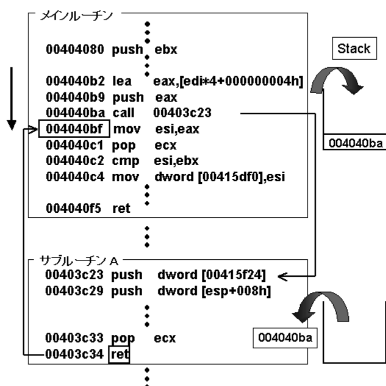

図 1 CALL 命令による同一サブルーチン外へのジャンプ命令出現時の処理

Fig. 1 Logic transform at jump somewhere in other subroutine by call instruction.

地に移動する．CALL 命令があった場合，step1 に戻る．そうでなければ，次の行に進み，step2 を続ける．

このように，再帰的に CALL 命令に従った解析を行っていく．

解析処理を進めていく際に，同一ルーチン内の別の番地に処理が移る分岐命令が出現する．条件分岐処理，および繰返し処理のための JMP 命令がそれにあたる．CALL 命令に対してはスタックを用意して呼び出し元に戻ってくるように処理していたが，これらの処理はともに，スタックは使わずに JMP 命令を無視して解析を進める．

まず，**図 2** のような条件分岐処理を例にあげて説明する．解析を進めていくと，00404878 番地の JZ 命令が出現する．そのオペランドで指定されている 00404887 番地へ処理を移動すると，0040487a〜00404885 番地の解析がなされないので，その命令を無視して解析処理を進める．0040487c, 00404885 番地も同様に，JNZ 命令を無視をすることで，すべての行の解析処理を行える．

次に，**図 3** のような繰返し処理を例にあげて説明する．00404a67 番地で JNB 命令を検出する．そのオペランドで指定されている移動先は 00404a7d 番地であるが，命令に従ってジャンプすると，00404a69〜00404a7b 番地の解析がなされないので，その命令を無視して解析を進める．解析を進めていくと，00404a7b 番地で JMP 命令が検出される．そのオペランドで指定されている移動先は 00404a65 番地と繰返し処理になっていることが分かる．この命令に従ってジャンプしても，すでに解析を行った命令文の解析を再度行うことになる．ウイルス解析システムは，ターゲットコー

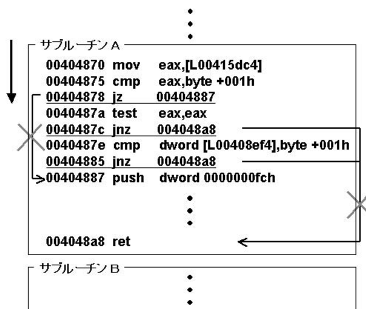

図 2 同一サブルーチン内の条件分岐処理に利用されたジャンプ命令出現時のロジックの変更

Fig. 2 Logic transform at jump somewhere in same subroutine for conditional branch.

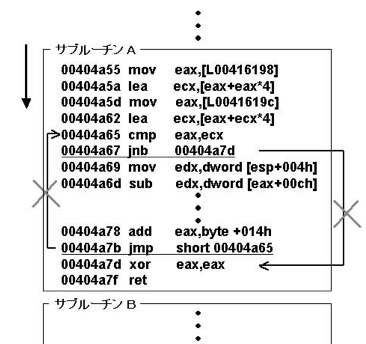

図 3 同一サブルーチン内の繰返し処理に利用されたジャンプ命令出現時のロジックの変更

Fig. 3 Logic transform at jump somewhere in same subroutine for loop.

ドの振舞いの概要を把握することが目的であるので，このような繰返し処理はトレースせず，処理時間の短縮化を図ることとした．

以上の処理で，一般的なソフトウェアのアセンブリコードに対して解析ができる．しかしながら，通常ウイルスの作成者はウイルスの解析を困難にするために，JMP 命令を使って別のサブルーチンへ処理を移動させるという手法をとる．そこで，別のサブルーチンへ処理が移動する JMP 命令が出現したときには，CALL 命令による別のサブルーチンへの移動時の処理と同様に，スタックを用いて解析処理を進めていく．ここでは，**図 4** の例で説明する．00401f1f 番地の JMP 命令で別のサブルーチンである 00403202 番地へ移動する．このとき，現在の番地である 00401f1f をスタックに入力する．00403202 番地から始まるサブルーチンでは，さらに 00404174 番地から始まるサブルーチンに JMP 命令で移動する．したがって，現在の番地である 00403203 をスタックに入力する．00404174 番地から始まるサブルーチンを解析していくと，00404385

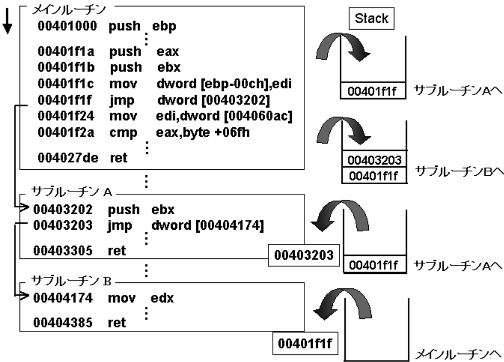

図 4 同一サブルーチン外へのジャンプ命令出現時の処理
Fig. 4 Logic transform at jump somewhere in other subroutine by jump instruction.

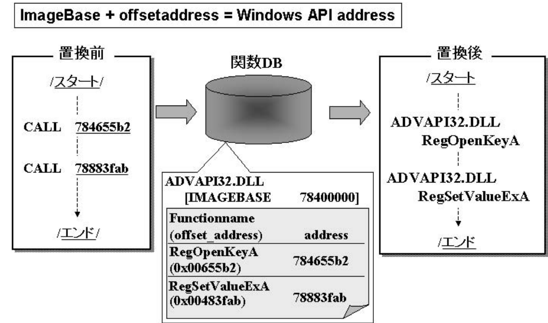

図 5 関数 DB を利用した Windows API の名前の取得
Fig. 5 Getting name of Windows API from ImageBase and offsetaddress by function DB.

番地に RET 命令があるので，スタックから値を 1 つ取り出し，00403203 番地へ戻って，その次の行から処理を進める．すると，00403305 番地の RET 命令を見つけ，スタックから値を 1 つ取り出し，00401f1f 番地に戻り，処理を続ける．

以上のような解析を行うことで，Windows API が呼び出される順番，すなわちターゲットコードの起動した不正プロセスが Windows API を実行する順番を大まかに把握でき，そして解析時間を短縮することができる．

3.2.2 呼び出された API の名前の特定

ターゲットコードの起動した不正プロセスで利用される Windows API が含まれている DLL は，メインモジュールとは異なるメモリ位置に展開されている．メインモジュールから呼び出される Windows API の名前をメインモジュールだけで直接取得することはできないため，次のような方法をとる．

まず，ターゲットコードを実行する．そして，システムが利用している DLL が展開されているメモリをダンプし，実行中のプロセスがロードしている DLL や API に関する下記の情報を取得する．

(1) DLL の名前とその ImageBase
(2) API の名前とその offsetaddress

ある DLL が展開されている先頭番地が ImageBase，DLL に含まれる API の先頭番地は ImageBase の値から相対的に指定され，その番地が offsetaddress である[10)]．したがって，ある API の先頭番地は，その API が含まれている DLL の ImageBase と offsetaddress の和で得られる．たとえば，実験に使用したホストであれば，ADVAPI32.DLL の ImageBase は 78400000 で，この DLL に含まれる RegOpenKeyA という API は offsetaddress が 00655b2 であった．RegOpenKeyA という API は 784655b2 番地から格納されていることが分かる．

次に，すべての DLL とそれらに含まれる API の ImageBase と offsetaddress から，API の名前と先頭番地を記録したデータベースを構築する．このデータベースを関数 DB と呼ぶこととする．そして，**図 5** に示すように，メインモジュール内で呼び出されている番地を関数 DB で検索し，呼び出された Windows API の名前を取得する．

3.2.1 項の解析結果と，以上の処理で特定された Windows API の名前により，ターゲットコード内での実行の順に，Windows API の名前と呼び出したメインルーチンもしくはサブルーチン内の番地を知ることができる．

3.3 文字列の抽出

ターゲットコードの起動した不正プロセスで利用される Windows API が特定できるだけでは，そのプロセスが具体的に何をするのかがつかめない．そこで，不正プロセスのアセンブリコードに埋め込まれているファイルやレジストリの名称を含めた文字列を抽出する．

アセンブリコードでは，文字列は基本的に db 命令でメモリ上に格納される．コード内の db 命令を探し，そのオペランドを取り出すことでアセンブリコードに含まれている文字列が抽出できる．

抽出した文字列がどのような Windows API と組み合わせて使われるかを知ることができれば，ターゲットコードから起動された不正プロセスの振舞いを予想する手助けとなる．アセンブリコードが Windows API を呼び出す直前にレジスタに何らかの文字列を代入することは，その文字列を Windows API の引数として利用する可能性が高いと考えられる．そのような文字列と API を同時に出力するために，文字列

がレジスタに格納される番地を調べ，3.2 節で調べた Windows API の呼び出し番地をもとに対応付ける．

文字列がレジスタに格納される番地を調べる方法について述べる．まず，アセンブリコードの先頭から最後までを走査し，出現した db 命令のオペランドを ASCII コードとして抽出し，その db 命令の番地とあわせて記録しておく．次に，再度アセンブリコードを先頭から走査し，先に記録された db 命令の番地をオペランドとしている命令を持つ番地を検索する．これで，どの番地で文字列がレジスタに格納されたのかを検出できる．

そして，レジスタに格納された文字列を引数に持つ Windows API の呼び出しは，文字列がレジスタに格納された直後の番地にあるので，次の走査で Windows API の名前と引数である文字列を対応させ，出力する．

以下の擬似コードを用いて説明する．

```
00402d08  mov esi,dword 00408034
00402d68  call dword 004060c0
004060c0  077e5e4c1  (GetAtomName を
                      呼び出している)
00408034  dd  00408ae0
00408ae0  db  'dolly_buster.jpg.pif',0
```

まず，1 度目の走査で，004060c0 番地の dd 命令によって格納されている 077e5e4c1 は，3.2.2 項に従い，関数 DB を利用して Windows API の名前とその呼び出し番地が記録される．この例では，004060c0 番地と GetAtomName という名前が記録される．あわせて，00408ae0 番地と dolly_buster.jpg.pif が記録される．次の走査で，00408ae0 という値をオペランドとして持つ命令の行を探す．このとき，00408034 番地のような dd 命令による 4 バイト単位でメモリに格納する命令行がマッチする場合がある．このようなときには，さらに 00408034 という値をオペランドとして持つ命令行を探す．そして，00402d08 番地が文字列をレジスタに格納していると判断し，その番地もあわせて記録しておく．この走査中，00402d68 番地の call 命令のオペランドに Windows API の呼び出し（ここでは GetAtomName となる）が直下にあり，この API と引数を対応付けて出力をする．以上の方法で，Windows API の呼び出し箇所に加えて，その引数を抽出でき，アセンブリコード内でどのように Windows API を実行するのかを解析者が把握しやすくなる．

4. ウイルス解析支援システム

3 章で述べた手法を使ったウイルス解析支援システムについて説明する．

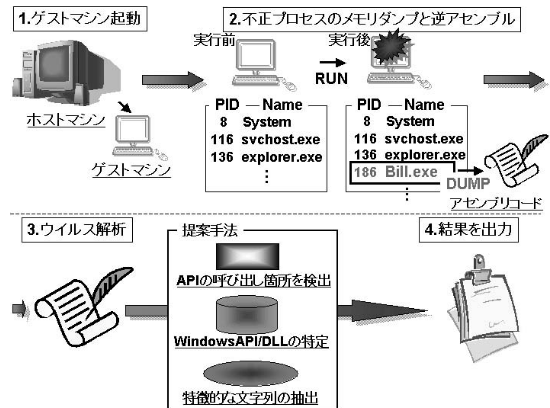

図 6 提案システムの動作手順
Fig. 6 Flow of the proposed system.

提案システムは**図 6** に示した，次のような手順で動作をする．

step1 ホストマシンでターゲットコードを実行するゲストマシンを起動する．

step2 ターゲットコードの実行の前後でプロセス情報を取得し，比較を行い不正プロセスを検出する．そして，検出した不正プロセスのメインモジュールをメモリダンプし，そのバイナリコードを逆アセンブルする．

step3 提案手法により step2 で出力されたアセンブリコードを解析する．

step4 解析結果を出力する．

step2 では，3.1 節で述べた手法で不正プロセスを検知する．現在動作しているプロセス情報の取得は次のように行う．

step2-1 CreateToolhelp32Snapshot という Windows API を利用し，システムのスナップショットを取得する．

step2-2 PROCESSENTRY32 クラスを使って，プロセスエントリ取得の前処理を行う．

step2-3 スナップショットとプロセスエントリを引数として，Process32First 関数を実行し，1 番目のプロセス ID を取得する．以下，取得できるプロセス情報が NULL になるまで，Process32Next 関数を実行し，プロセス ID を取得する．

以上で，現在動作しているプロセスの ID をすべて取得できる．

このプロセス情報の取得を，ターゲットコード実行の前後で行い，比較をする．2 度目の情報取得時に，1 度目に取得されなかったプロセスがあれば，それを不正プロセスと判断する．そのプロセス ID を IDA [8] などの逆アセンブラに渡し，メモリダンプと逆アセン

表 1 注釈のためのデータベースの例
Table 1 Example of database for annotation.

Windows API の名前や文字列	推測される行動や特徴
SOFTWARE￥Microsoft￥Windows￥CurrentVersion￥Run	ウイルスを自動実行する場合に用いられるレジストリキー
SOFTWARE￥Microsoft￥Windows￥CurrentVersion￥RunServices	ウイルスを自動実行する場合に用いられるレジストリキー
MIME-Version:	独自の SMTP エンジンコード
From，To	メールのフィールド
RegSetValueEx	レジストリ改ざん
RegQueryValueEx	レジストリキーから指定した値のデータを取得する
RegEnumKey	サブキーを取得する
RegCreateKeyEx	指定されたレジストリキーを作成する
CreateFile	ファイルを作成する
CopyFile	ファイルをコピーする
DeleteFile	ファイルを削除する
GetWindowsDirectory	Windows ディレクトリのパス名を取得する
GetSystemDirectory	Windows のシステムディレクトリのパス名を取得する
WriteFile	ファイルを改ざんする
GetTickCount	Windows 起動後の経過時間を取得する
CreateMutex	ミューテックスオブジェクトを作成する
CreateFileMapping	指定されたサイズのファイルマッピングオブジェクトを作成する
GetTempFileName	テンポラリファイルの作成
CreateCompatibleDC	指定されたデバイスコンテキストに関連するデバイスと互換性のあるメモリデバイスコンテキストを作成する
WritePrivateProfileString	ファイルに文字列を書き込む
WinExec	指定されたアプリケーションを実行する
CreateWindowEx	新しいウインドウを作成する
RegisterServiceCtrlHandler	関数を呼び出して，自らのサービス制御要求を処理する関数を登録する
InternetGetConnectedState	ローカルシステムの接続状態を確認する
DnsQuery	宛先メールアドレスのドメインについてデフォルトのメールサーバ情報（MX レコード）をクエリする
GetNetworkParams	宛先メールアドレスのドメインについてデフォルトのメールサーバ情報（MX レコード）をクエリする

ブルを行い，そのアセンブリコードをファイルに保存する．

step3 では，3.2，3.3 節で述べた，Windows API の実行手順の解析，文字列の抽出を次のように行う．

step3-1 関数 DB を作成する．

step3-2 アセンブリコードに対して，メインルーチン，およびサブルーチンの始点，終点を調べ，それらの番地を記録する．

step3-3 アセンブリコードから，Windows API 呼び出し箇所の検出，および文字列の抽出を行う．Windows API を呼び出している番地，その API が格納されている番地，文字列とそれを格納している番地が分かったときには，その情報を記録する．

step3-4 文字列をレジスタに格納している番地を記録する．

step3-5 記録した Windows API が格納されている番地から，関数 DB により Windows API の名前を特定し，記録する．

ここで，step3-1 の関数 DB の作成には，Dependency Walker[11] と呼ばれる，プロセスが利用する DLL の名前，ImageBase，API の名前，offsetaddress を出力するツールを用いる．step3-2 の各ルーチンの始点と終点を調べるのは，3.2.1 項で述べた，step3-3 での Windows API 呼び出し箇所の検出で，JMP 命令によるジャンプ先が同一サブルーチン内か否かを判断するために必要だからである．

step4 では HTML 形式で，次の情報を含んだ解析結果をアセンブリコードの実行順に出力する．

(1) Windows API
 (a) 呼び出された API の名前
 (b) メインルーチンでその API を呼び出している番地
 (c) サブルーチンでその API を呼び出して

　　いる番地
　(d)　処理の中でのその API の出現順
(2)　文字列
　(a)　格納される文字列
　(b)　文字列を取り出してレジスタに格納している番地
　(c)　文字列を格納している番地

Windows API を呼び出している番地と文字列をレジスタに格納している番地が近いものを 1 セットとして出力をし，ターゲットコードがどのような動作を行うのか理解しやすい結果を出力する．

このとき，ウイルスがよく用いる Windows API の名前や文字列と，それらから推測される行動や特徴をデータベースに保存しておき，そのデータベースに登録されている情報が取得された場合には，注釈を付記する．たとえば，**表 1** のような情報をデータベースに格納しておく．

もしも，Windows API がメインルーチン内で呼び出された場合には，(1) - (c) は出力せず，(1) - (b) のみを出力する．サブルーチン内で呼び出された場合には，(1) - (c) を出力し，あわせて (1) - (b) としてメインルーチン内のそのサブルーチンを呼び出す番地を出力する．このようなメインモジュール内での Windows API を直接的，間接的に呼び出す番地は，解析者がリバースエンジニアリングツールによるブレークポイントの設定を助ける情報となる．

5. 実　　験

提案システムを用いて複数のウイルスを対象とした実験を行った．実験環境を**表 2** に示す．

表 2　実験環境
Table 2　Specification of hardware and software.

ホストマシン	
OS	Windows2000 Professional
CPU	Pentium4 3 GHz
Memory	512 Mbyte（160 Mbyte はウイルス実行ホストで使用）
HD	Ultra160 SCSI
仮想環境ソフトウェア	VMware Workstation4.0
ゲストマシン	
OS	Windows2000 Professional
仮想マシン用に割り当てた Memory	160 Mbyte

まず，よく知られた 8 つのウイルスを提案システムに入力し，結果が得られるまでの時間を測定した．W32.Welchia.Worm，W32.Netsky.B@mm，W32.Netsky.P@mm，W32.Yaha.B@mm，W32.Beagle.H@mm，W32.KwbotF.Worm は約 20 秒，W32.Mydoom.A@mm，Backdoor.SubSeven は約 30 秒で結果が得られた．

本システムの解析においては，3.2.1 項で示した，CALL 命令と JMP 命令による別のサブルーチンへのジャンプの際にはスタックを用い，同一ルーチン内の条件分岐処理と繰返し処理の JMP 命令は無視するという手法をとっている．すべての命令文に対して 1 ステップごとに解析を行うと，複雑に振る舞うウイルスではその処理時間は膨大なものとなる．CALL 命令と JMP 命令によるジャンプ先が，同一ルーチン内か否かで処理の方法を分けることによって，短時間で解析結果が出力でき，提案手法は有効であるといえる．

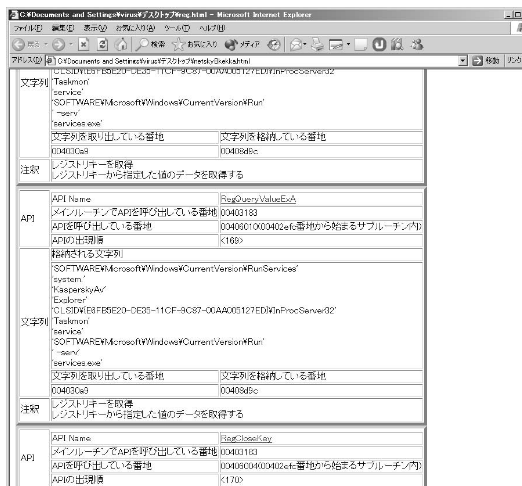

図 7　W32.Netsky.B@mm のレジストリに関連する出力結果の一部
Fig. 7　A part of output of W32.Netsky.B@mm about registry.

次に，近年被害が大きかった W32.Netsky.B@mm を使った実験結果について述べる．

W32.Netsky.B をターゲットコードとして入力し実行させたところ，bill.exe という名前の不正なプロセスが起動した．したがって，bill.exe が不正プロセスと見なされ，そのアセンブリコードが解析対象となる．

一般にウイルスは，レジストリやファイルを操作することで，感染ホストに被害を与える．マスメイル型ウイルスであれば，独自の SMTP エンジンを利用して感染を拡大させる．**図 7** はレジストリ操作に関連する出力結果，**図 8** はファイル操作に関連する出力結果，**図 9** は SMTP に関連する出力結果の一部である．解析結果はターゲットコードが起動した不正プロセスの実行する Windows API と，その Windows API が使用する文字列と注釈があわせて表示される．実行され

表 3 提案システムの出力結果の例（図 7 から抜粋）
Table 3 An example of output from the proposed system.

API	API 名	RegQueryValueExA
	メインルーチンで API を呼び出している番地	00403183
	API を呼び出している番地	00406010（00402efc 番地から始まるサブルーチン内）
文字列	格納される文字列	
	’SOFTWARE￥Microsoft￥Windows￥CurrentVersion￥RunServices’ ’system’ ⋮	
	文字列を取り出している番地	文字列を格納している番地
	004030a9	00408d9c
注釈	レジストリキーを取得	

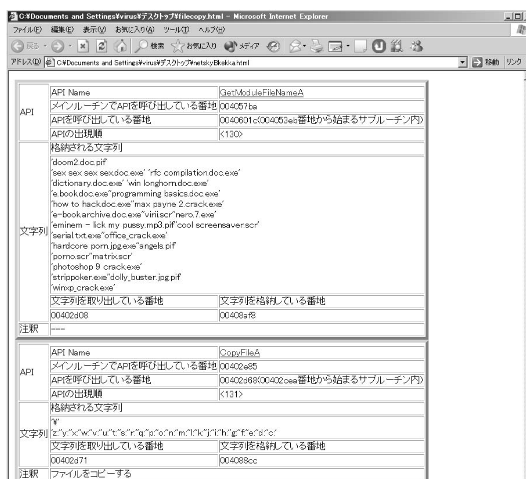

図 8 W32.Netsky.B@mm のファイル操作に関連する出力結果の一部
Fig. 8 A part of output of W32.Netsky.B@mm about file operation.

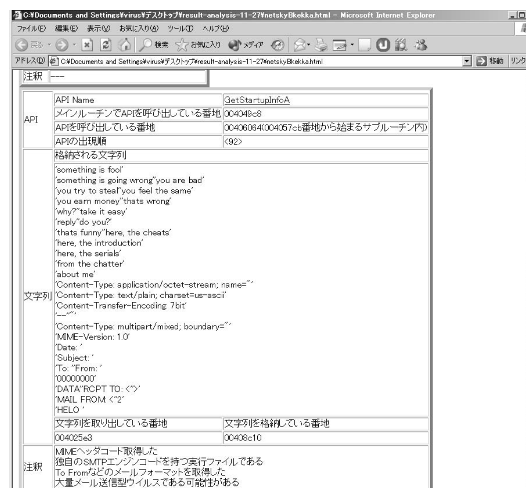

図 9 W32.Netsky.B@mm の SMTP に関連する出力結果の一部
Fig. 9 A part of output of W32.Netsky.B@mm about SMTP.

る Windows API ごとに分割して表形式に整理し，時系列に表示をさせることで，ウイルス解析者はその不正プロセスの大まかな動作を知ることができる．**表 3** は，図 7 から抜粋した出力例である．

結果の表には，不正プロセスで使用される Windows API を呼び出している，メインルーチン内の番地を表示している．その行動内容とメインルーチン内の番地が分かることで，ウイルス解析者のリバースエンジニアリングツールによる手動での解析において，重点的に解析をするためのブレークポイントの設定が容易に可能となる．

また，実験に用いた 8 つのウイルスについて，本システムの出力結果をアンチウイルスベンダが公開しているセキュリティレスポンスのウイルスの動作に関する情報と比較したところ，すべて出力されていることを確認した．

6. おわりに

本論文では，ウイルス解析者の支援を目的としたシステムを提案した．従来の疑わしいコードのファイルを解析する方法と異なり，コードを実行してメモリに展開したコードを解析する手法について述べた．メモリに展開したコードを解析することにより，難読化が施されている未知のウイルスに対しても解析が可能となった．そして，ターゲットコードの解析にかかる処理時間を短縮する方法について述べ，実験によりその有効性を示した．さらに，システムから出力された解析結果は，アンチウイルスベンダのセキュリティレスポンスと同等の内容を含んでいること，抽出された Windows API と文字列を組み合わせて出力させることが可能なことを確認した．

Windows API の呼び出しをトレースする点では，Rabek らの手法[12] に提案手法は類似している．Rabek らは利用者のクライアント上でプログラムを実行中に，Windows API の呼び出し手続きに注目し，異常をホストベースで検知することを目的としている．提案システムは異常検知ではなく，ウイルス解析者を支援するために，セキュリティレスポンスと同等の内容となるべく，呼び出される順に Windows API の名前と引数を出力する．

コンピュータウイルスによる被害を回避するためには，利用者のクライアントに導入されているウイルス検知・駆除ソフトに対して迅速にシグネチャを配布する必要がある．大量のコンピュータウイルスが発生する中で，アンチウイルスベンダによるシグネチャの作成は未知のウイルスに対して迅速に行われなければならないために，文献 13) などの手法を使った解析の自動化が試みられている．しかしながら，すべてのウイルスを自動的に解析できないために，ウイルス解析者の手作業に頼る部分も残されている．コード内で使用される Windows API が呼び出されるメインルーチン内の番地を示すことで，ウイルス解析者のリバースエンジニアリングによる解析の支援につながる．

実験において，バッファオーバフローの脆弱性を利用する W32.Welchia.Worm も解析できることを確認しているが，巧みにスタックを操作しながらフローを制御するコードに対応できる方法について，さらなる検討を行うことが今後の課題としてあげられる．

謝辞 本研究は総務省からの委託研究「コンピュータウイルス等に関する研究基盤の構築」の一環である．本委託研究の関係者各位に謝意を表する．特に，有益なご討論をいただいた KDDI（株）中尾康二氏をはじめとする（株）KDDI 研究所ネットワークセキュリティグループの関係各位,（株）セキュアブレイン星澤祐二氏に感謝する．なお，本研究の初期の段階において有益な議論をしていただいた，当時徳島大学大学院工学研究科博士前期課程学生，現在（株）日立製作所神薗雅紀氏に謝意を表する．

参考文献

1) 総務省：情報通信白書 平成 13 年版 (2001). http://www.johotsusintokei.soumu.go.jp/whitepaper/ja/h13/html/D1281000.htm
2) 情報処理推進機構：2004 年度ウイルス届出状況 (2005). http://www.ipa.go.jp/security/txt/2005/documents/2004all-vir.pdf
3) 総務省：次世代 IP インフラ研究会 第二次報告書「情報セキュリティ政策 2005」への提言 (2005).
4) Hayasi, K: Open Malicious Source, AVAR Conference 2004 (2004). http://www.aavar.org/2004web/AVAR2004/pc01.htm
5) 藤長昌彦，中尾康二，森井昌克：ウイルス分析のためのテストベッドの構築，2005 年暗号と情報セキュリティシンポジウム予稿集，pp.1189–1194 (2005).
6) ITmedia：Level3 ウイルスの発見 そのときのベンダーは? (2004). http://www.itmedia.co.jp/enterprise/articles/0410/04/news075.html
7) Symantec，セキュリティレスポンス. http://www.symantec.com/region/jp/sarcj/
8) DataRescure, IDA Pro Disassembler and Debugger. http://www.datarescue.com/idabase/
9) Networld, VMware. http://www.networld.co.jp/products/vmware/
10) MSDN Online Japan. http://www.microsoft.com/japan/msdn/default.asp
11) Steve P. Miller, Dependency Walker. http://www.dependencywalker.com/
12) Rabek, J.C., Khazan, R.I., Kewandowski, S.M. and Cunningham, R.K.: Detection of Injected, Dynamically Generated, and Obfuscated Malicious Code, *Proc. the 2003 ACM Workshop on Rapid Malcode* (*WORM '03*), pp.76–82 (2003).
13) Symantec: Understanding Heuristics: Symantec's Bloodhound Technology. http://www.symantec.com/avcenter/reference/heuristc.pdf

(平成 17 年 12 月 1 日受付)
(平成 18 年 6 月 1 日採録)

市川 幸宏

2004 年徳島大学工学部知能情報工学科卒業．2006 年同大学大学院博士前期課程修了．同年三菱電機（株）入社．情報セキュリティに関する研究に従事．

伊沢 亮一

2004 年徳島大学工学部知能情報工学科卒業．2006 年同大学大学院博士前期課程修了．現在，神戸大学大学院博士後期課程在学中．情報セキュリティに関する研究に従事．

白石　善明（正会員）

1995 年愛媛大学工学部情報工学科卒業．2000 年徳島大学大学院工学研究科博士後期課程修了．博士（工学）．2002 年近畿大学理工学部講師，2006 年名古屋工業大学大学院工学研究科助教授．情報セキュリティ，コンピュータネットワーク等の研究，教育に従事．2002 年電子情報通信学会オフィスシステム研究賞，2003 年暗号と情報セキュリティシンポジウム（SCIS）20 周年記念賞，2006 年 SCIS 論文賞．電子情報通信学会，IEEE 各会員．

森井　昌克（正会員）

1989 年大阪大学大学院工学研究科通信工学専攻博士課程修了．工学博士．同年京都工芸繊維大学工芸学部助手．1990 年愛媛大学工学部講師，1992 年同助教授，1995 年徳島大学工学部教授を経て，2005 年神戸大学工学部教授．情報セキュリティ，代数的符号理論，離散数学，コンピュータネットワーク等の研究，教育に従事．IEEE，電子情報通信学会各会員．