未知コンピュータウイルスを駆除するUSBフラッシュメモリの開発

小池竜一† 中谷直司† 厚井裕司†

近年，コンピュータウイルス感染により機密情報が流出する事件等が頻繁に起こっており，ウイルスの被害は社会的なリスクとしてとらえるべきものとなっている．そこで本論文では，未知ウイルスに感染済みのPCに挿入することで，人手を介することなく自動的に未知ウイルスを駆除するUSBフラッシュメモリを開発した．USBフラッシュメモリにはブート可能なLinuxが内蔵されているため，適当なPCに挿入すると独自環境が立ち上がり，デバッグ機能を利用した動的ヒューリスティック手法により未知ウイルスを検出する．未知ウイルスが検出された際には，ウイルスを検出・駆除するソフトウェアが自動的に生成される．本システムを用いて実際のウイルスに対する駆除ソフトウェアを自動生成し，ウイルスが駆除可能か検証を行ったところ，通常のファイルを誤って削除することなく未知ウイルスを駆除することができ，システムの有効性を確認することができた．

Development of an USB Flash Memory for Detecting Unknown Computer Viruses

Ryuiti Koike,† Naoshi Nakaya† and Yuuji Koui†

Recently cases that confidential information leaks out by computer virus infections have happened frequently. Therefore we should think damage of viruses as a social risk. In this paper, we developed an USB flash memory which automatically exterminates unknown viruses by only inserted into a PC's USB port. The USB flash memory includes bootable Linux in itself. When the USB flash memory is inserted into a PC's USB port, an unique environment boots up, and detects unknown viruses by using a heuristics method with debugger. If an unknown virus was found by this system, a virus extermination software is created automatically. We created virus extermination softwares for real viruses with this system, and tried whether the softwares exterminate viruses. In the result, the softwares could exterminate viruses without deleting benign executables.

1. はじめに

インターネットをはじめとするネットワークが急速に発展するにつれて，コンピュータウイルス（以下，ウイルス）による被害は年々深刻なものとなっている．これらのウイルスを検出・駆除するためには，ウイルス対策ソフト（以下，アンチウイルス）を利用する方法が原則である．アンチウイルスはウイルスの特徴を収めたシグネチャと，対象となるファイルをパターンマッチングすることでウイルスを検出している（以下，パターンマッチング方式）．そのため，シグネチャに情報が存在しないウイルスは，アンチウイルスにとっては未知のものであり検出することはできない．しかし，近年では新種の未知ウイルスが1日に約30種類発生している[1]にもかかわらず，アンチウイルスメーカはシグネチャ生成に平均10時間を要している[2]．そのため，シグネチャに含まれない未知ウイルスを検出するための様々な研究が行われてきた．しかしながら，完璧な検出手法というものは存在せず，ユーザの一定数が未知ウイルスに感染してしまう事態は避けることができない．

そこで本論文では，未知ウイルスに感染済みのPCに挿入することで，人手を介することなく自動的に未知ウイルスを駆除するUSBフラッシュメモリを開発した．開発したUSBフラッシュメモリ内にはブート可能なLinuxが内蔵されているため，それをUSBポートに挿入しPCの電源を入れると，独自の検証環境が起動しハードディスク内の未知ウイルスの検出・駆除が自動的に行われる．具体的には，仮想PCを用いた動的ヒューリスティック手法と，実行ファイルのヘッダ情報からシグネチャを自動生成する手法とを組み合わ

† 岩手大学大学院工学研究科
Graduate School of Engineering, Iwate University

せ，未知ウイルスを検出・駆除するソフトウェア（以下，ワクチン）の自動生成を行う．開発したシステムを用いて実際のウイルスに対するワクチンを自動生成し，ウイルスが駆除可能か検証を行ったところ，システムの有効性を確認することができた．

以下，2章ではウイルスについて述べ，3章では，未知ウイルス駆除ソフトウェア自動生成システムの概要ついて述べる．また，4章では実装の詳細について述べ，5章では，実験を行い本システムの有効性について述べる．6章では本システムと関連研究との比較を行う．

2. コンピュータウイルス

経済産業省の定義によると，コンピュータウイルスとは，第三者のプログラムやデータベースに対して意図的に何らかの被害を及ぼすように作られたプログラムで，自己伝染機能，潜伏機能，発病機能のうち1つ以上を有するものとされている．初期のコンピュータウイルスは，これら3つの機能をすべて有し，感染，潜伏，発病というサイクルを繰り返すものが主であった．しかし，最近では潜伏期間を持たず活動するウイルスや，発病することなく感染を繰り返すウイルス等が出現している．そこで，現在では狭義のウイルスの定義では含まれなかったワームやトロイの木馬といったものも含めた，不利益をもたらす不正プログラム全体をウイルスと呼んでいる．ただし，実際のウイルスのほとんどがMicrosoft Windows OS（以下，Windows）を対象としていることを考慮して，本システムでもWindows上で動作する実行ファイル形式のウイルスのみを対象とする．

3. 未知ウイルス駆除ソフトウェア自動生成システム

本システムは何らかの原因で未知ウイルスに感染してしまったPC（以下，感染済みPC）からウイルスを検出し，さらにそのウイルスの駆除ソフトウェアを自動生成するためのシステムである．本システムはUSBフラッシュメモリ内に格納されており，これを用いて感染済みPCをブートさせ独自の検証環境を構築する．次に感染済みPC内からウイルス候補ファイルを探し出し，それらを順次検証環境内で実行する．そして，その際の振舞いからウィルスを検出し，ワクチン自動生成およびウイルス駆除を行う．この一連の動を，検証環境構築，ウイルス候補ファイル検索，未知ウイルス検出，ワクチン自動生成と定義し以下で概要を述べる．

検証環境構築 本システムは独自の検証環境内でウイルスを実際に実行し，その振舞いによって未知ウイルスの検出を行う．そこで，本システムはウイルスにとって活動しやすい環境を用意するとともに元の環境を即座に復元させる機能を持たせた．これにより，つねにシステムが用意したウイルス未感染の環境でウイルスを動作させることが可能となり，ウイルス活動の痕跡を発見することが容易となる．

ウイルス候補ファイル検索 一般的にウイルスはOS起動時に自動実行が行われメモリへの常駐を試みる．そのため，感染済みPC内のOS起動時に自動実行されるような実行ファイルはウイルス候補としてあげることが可能である．そこで，本システムでは検証環境の構築後，感染済みPC内より自動実行が行われる実行ファイルを検索し，すべてのウイルス候補を決定する．

未知ウイルス検出 本システムはウイルス候補を1つ選択し，それを検証環境内で実際に実行する．そして，一定時間の間の実行ファイルの動作をすべて監視し記録する．1つの実行ファイルの行動を記録した後は，検証環境を元の状態に復元し新たなウイルス候補を実行する．これをすべてのウイルス候補に対して行う．上記の処理中に危険箇所を書き換える実行ファイルが存在した場合，ウイルスであるとの判断がなされる．このときにおける危険箇所とは，OS起動時に実行ファイルを自動実行させるような設定やシステムフォルダのことを指す．つまり本システムにおけるウイルスの定義とは，ユーザの同意を得ずに自動実行の設定を書き換えOSが起動するたびにメモリへの常駐を試みるようなプログラムであるといえる．なお，シマンテック社のウイルス情報[3]をもとに集計を行ったところ，自動実行の設定改ざんとシステムフォルダの書き換えとの少なくとも一方の動作を行うウイルスは，ネットワーク上に存在が確認された実行ファイル形式ウイルスの約80%を占めている．

また，実行ファイルの監視を行う時間は3分とした．これは手持ちのウイルスに対する5章以降で行われる実験において，監視時間をこれ以上に増加させても検出結果に変化がなかったことより経験的に導き出された値であり，ウイルスにとってOSの危険箇所を書き換えるのに十分な時間であると考えた．

ワクチン自動生成 本システムは未知ウイルス検出機能によってウイルス検出を行った場合，ウイルス駆除プログラムであるワクチンを生成する．ワクチンは記録されたウイルス動作を単純に逆にした動作を行うプログラムである．つまり，仮にウイルスが自分自身を

ある箇所へコピーし，自動実行の設定を行った場合，ワクチンは単純にウイルスのコピーと自動実行の設定とを感染済み PC から削除する．ただし，ウイルスの動作にランダムな要素が含まれる場合，単純にウイルスの動作の逆を行うことができなくなる．そこで，ウイルスを一意に識別するためにシグネチャの生成も行う．

なお，シマンテック社のウイルス情報をもとに集計を行ったところ，他のファイルへを改竄する破壊的な行動を行う実行ファイル形式のウイルスは約 9%存在した．ただし，ワクチンはウイルス駆除にのみ特化しているため，このようなウイルス活動によって破壊されたファイルの復旧等は行わない．

4. 実装詳細

ここでは，本システムの実装詳細について述べる．

4.1 検証環境構築

本システムはユーザが普段利用している PC 上に，仮想 PC である VMware 5.0[4] を用いて図 1 のような検証環境を構築する．以下で検証環境の特徴を述べる．

- 検証対象の PC を Linux が格納された USB フラッシュメモリを用いて起動させ，Linux を Host OS として VMware を実行する．
- 本システムのターゲットである未知ウイルスは最新の Windows 環境を対象としていると考えられるので，VMware 上には Gest OS として Windows XP SP2 を標準インストールする．この Windows が実行ファイルが実際に実行される環境となる．
- VMware のスナップショット機能により，ウイルスに破壊された環境を即座に復元できるように設定する．
- VMware 上の WindowsXP は Host OS のみと通信可能とし，閉じたネットワークを構成する．

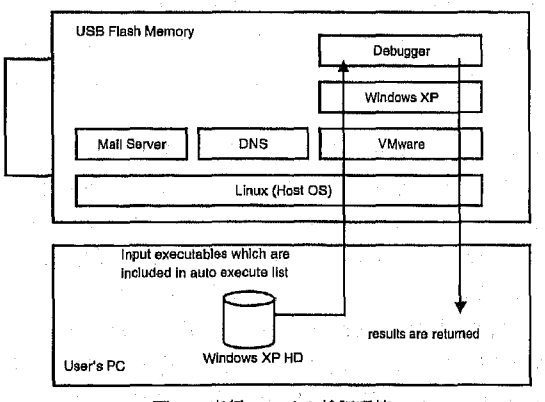

図 1 実行ファイル検証環境
Fig. 1 Verification environment of executables.

- ウイルスのネットワークを使った感染活動が阻害されないように，Host 側に DNS を用意し，あらゆる名前解決に対して Host OS の IP アドレスを返すよう設定する．
- ウイルスのメールによる感染活動が阻害されないように，Host 側にメールサーバを用意し，あらゆるあて先のメールもローカルに配信するように設定する．

すなわち，たとえ VMware 上の Windows がウイルスに感染しても，閉じたネットワーク構成により感染が外に広がることはないが，メールをはじめとしたネットワークを介した感染は，Host OS 側に用意した DNS によりすべて Host OS に誘導されるため，ウイルスにとってはネットワークが閉じているとは認識し難い状況になっている．

なお，本システムを起動させるための記録媒体として USB フラッシュメモリを選択した理由は，本システムを利用可能な PC の数を最大化するためである．その他の記録媒体としては SD メモリカードやメモリースティック等も広く利用されている．しかし，これらの利用には専用のリーダを必要とする．一方，USB ポートを装備していない PC は皆無である．また，ウイルスの動作ログを記録するため，CD-R/DVD-R 等の 1 度の書き込みが基本である記録媒体は本システムには不向きである．さらに，HDD は上記の記録媒体に比べ高価であるため積極的に選択する必要はない．

本システムの主要な構成要素のファイルサイズは，それぞれ Linux が 100 MByte，VMware が 20 MByte，Windows XP が 1700 MByte の合計 1820 MByte であり，一般的な 2 GMByte の USB フラッシュメモリ内に格納が可能である．ただし，今後 Windows の容量が増加した際には，さらに大容量の USB フラッシュメモリを用いる可能性は存在する．

4.2 ウイルス候補ファイル検索

本システムは，ユーザが普段利用している PC の Windows（以下，ユーザ用 Windows）上からウイルス候補を検索し，VMware 上の Windows（以下，検証用 Windows）に入力を行うことで未知ウイルスを検出する．以下で一連の流れを示す．

(1) Linux はユーザ用 Windows のハードディスクをマウントし，レジストリから自動実行が行われる実行ファイルの検索を行う．そして，それらの実行ファイルをウイルス候補としてリスト化する．

(2) リスト中から 1 つ実行ファイルを選び，Linux 側から検証用 Windows に渡す．

(3) 検証用 Windows 内で実行ファイルが実行され，未知ウイルス検出機能により活動の記録がなされる．

(4) 得られた動作情報を検証用 Windows 側から Linux 側に渡す．また，次の検証に備えて検証用 Windows の環境を元の状態へ復元する．
(5) Linux 側において，渡された動作の記録を解析し，ウイルスであるかの判定を下す．
(6) 操作(2)へ戻り，リスト中のすべての実行ファイルに対して繰り返す．

4.3 未知ウイルス検出

未知ウイルスの検出にはシグネチャによるパターンマッチングは使えないため，本研究ではヒューリスティック手法を用いることとする．ヒューリスティック手法は次の2つに分類することができる．

静的ヒューリスティック手法 チェック対象のファイルのコード内の命令を解析して，実行するとどのような行動をするのかを予測する．そして，その動作がウイルスの動作と認められる場合にはウイルスと判定をする[5),6)]．

動的ヒューリスティック手法 特殊な環境下で実際に実行ファイルを動作させる．そして，その動作がウイルスの動作と認められる場合にはウイルスと判定をする[7)~11)]．

今回のシステムは未知ウイルスの検出のみではなく，ワクチンの自動生成を目的としている．そこで，ワクチン生成に利用可能なウイルスの動作情報を得るのに都合がよい動的ヒューリスティック手法をデバッグ機能を用いて実装した．

4.3.1 デバッガを用いた動的ヒューリスティック手法

Windows 上で動的ヒューリスティック手法を行う場合，重要になるのが API (Apprication Programing Interface) の呼び出しである．API は Windows で使用される関数や構造体，マクロ等の集合体であり，DLL (Dynamic Link Library) により提供されている．基本的に Windows の実行ファイルは API を経由してすべての処理を行っている．すなわち，API 関数の呼び出しを取得することができれば，検証対象の動作をすべて検出することが可能となる．この API 関数の呼び出しを取得するには，特定の実行ファイルからの API 関数の呼び出しを横取りし，任意の動作を行わせる技術である API フックを用いる．API フックの手法はいくつか知られているが，本システムではデバッガを用いた手法を実装した．

デバッガを用いた手法は Windows が用意しているデバッグ機能を利用する．この機能を用いれば，デバッグ対象の API 関数の呼び出しを検出し，引数を抽出することが可能となる．実際にデバッグ機能を用いて API 関数の使用を監視するには，以下の手順を踏む必要がある．

(1) 監視する API 関数のブレークポイント設定に使用するアドレス情報を取得する．
(2) デバッグ対象の実行ファイルをデバッグ可能なフラグを立てて実行する．
(3) 監視する API 関数を含む DLL がロードされたら，取得したアドレス情報を用いて該当する API 関数にブレークポイントを設定する．
(4) ブレークポイントを検出したら，該当する API 関数の引数を取得する．
(5) (3)，(4)を実行ファイルが終了，もしくは3分間経過するまで繰り返す．

これにより，実行ファイルのすべての動作を監視することが可能になる．本システムの目的である未知ウイルスの駆除のためには，次項で述べる動作を監視する必要がある．

4.3.2 監視対象 API 関数

本システムでは**表1**に示した API 関数をフックすることでレジストリの自動実行に関する設定と Windows システムフォルダの書き換えを監視している．なお，ウイルス以外にも Windows の危険箇所への変更を行う実行ファイルは普遍的に存在する．代表的なものとしてはインストーラ等があげられる．ただし，実際のインストーラは変更を行う前にユーザにメッセージを表示し，クリック等の確認作業を要求することが一般的である．よって，本システムのように純粋に実行ファイルの自動実行しか行わない環境では，ユーザによる確認作業が行われないため，インストール作業は実際には行われずウイルスと判定されることもない．

4.4 ワクチン自動生成

前章で述べたように未知ウイルス検出に動的ヒューリスティック手法を用い，ウイルスが Windows に感染するための動作であるファイルの作成やレジストリの変更情報を正しく得ることができれば，ワクチンを自動生成することは難しいことではない．基本的には，ウイルスの動作を逆にすればよく，作成されたファイルを削除し，変更されたレジストリを元に戻せばウイルスは駆除可能である．ただし，ウイルスの動作にランダムな要素が含まれる場合，単純にウイルスの動作の逆を行うことができなくなる．たとえば作成するファイル名をランダムに決めるウイルスの場合，本システムの検証環境で実行したときに作成されたファイル名と，ワクチンを動作させたい環境で作成されたファイル名が一致しなくなるため，単純にファイル名の一致したファイルを削除するという駆除はできなくなる．

表 1 監視する動作と API 関数
Table 1 Relations between monitored operations and API functions.

監視する動作	提供する DLL	API 関数	
ファイルの作成	Kernel32.Dll	CreateFileA	CreateFileW
		CopyFileA	CopyFileW
		CopyFileExA	CopyFileEzW
		WriteFile	
		WriteFileEx	
		CreateDirectoryA	CreateDirectoryW
		CreateDirectoryExA	CreateDirectoryExW
レジストリの変更	ADVAPI32.Dll	RegOpenKeyA	RegOpenKeyW
		RegOpenKeyExA	RegOpenKeyExB
		RegCreateKeyA	RegCreateKeyW
		RegCreateKeyExA	RegCreateExW
		RegSetValueA	RegSetValueW
		RegSetValueExA	RegSetValueExW

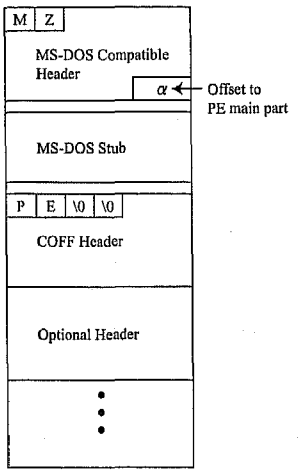

図 2 PE 形式の構造
Fig. 2 The structure of the PE format.

そこで，ウイルスを識別する何らかのシグネチャを，自動生成することが必要となってくる．

4.4.1 シグネチャの自動生成

シグネチャを自動生成する場合，専門知識を持った人間が生成するのとは違い，現在のウイルスの多くが共通して持っており，なおかつ固有な情報を抽出し，それをもとにシグネチャを生成する必要がある．そこで，著者らは論文 12) において Windows の実行ファイルのヘッダ情報から，シグネチャを自動生成する手法を提案した．ここでは概要を述べる．

Windows の実行ファイルは，PE (Portable Executable) 形式[13] と呼ばれるフォーマットに従って構成されており，ウイルスも Windows という OS 上で実行される以上，このフォーマットに従っている．

図 2 に PE 形式の構造を示す．このうち，MS-DOS 互換ヘッダ/スタブは下位互換のためのものであり本質的には必要がない．そのため，実行ファイルによってはこれらのヘッダが存在しない可能性が考えられ，シグネチャ生成に用いる部分としては適さない．すべての実行ファイルに存在するヘッダは COFF (Common Object File Format) ヘッダとそれに続くオプション・ヘッダのみである．そこで，実際の実行ファイルにおいて，この 2 つのヘッダ部分の各項目ごとに値が一致する確率を調査したところ，オプション・ヘッダ中の "Import Table" の値が最も一致しなかった (一致率 0.02%)．そこで，この値をもとにシグネチャを生成する．Import Table までのファイルの先頭からのオフセットは，MS-DOS ヘッダの最後の部分に記述されている，PE 形式であることを表す「PE\0\0」までのオフセット (可変:α byte とする) と，「PE\0\0」の先頭から Import Table までのオフセット 128 byte の和となり，$(128+\alpha)$ byte となる．そして，Import Table のサイズは 8 byte なので，生成されるシグネチャ (以下，IT (Import Table) シグネチャ) は「ファイルの先頭から $(128+\alpha)$ byte 目に続く 8 byte がウイルスの同部分と等しい」となる．

4.4.2 ワクチンの構造

前項で述べた IT シグネチャを用いることで，作成するファイル名がランダムなウイルスでもウイルスを識別可能となった．そこで，この IT シグネチャを用いたワクチンの構造について述べる．

本システムによってウイルスが発見された場合には，Linux 上で動作するワクチンと Windows 上で動作するワクチンとが同時に生成される．前者のワクチンはユーザ用 Windows に発見された未知ウイルスを Linux で動作する本システム側から即削除を行うためのものである．また，後者のワクチンは他の PC 向けに配布を行うためのものである．

ワクチンは個々のウイルスに合わせて自動生成されるものの，その場合でもファイルを削除しレジストリ

を元に戻すというワクチンの基本動作は，ウイルスによらず共通のものとなる．したがってワクチンを，ファイルを削除しレジストリを元に戻すという基本動作を実行する部分と，その基本動作を制御する定義部分とに分け，定義部分を個々のウイルスに合わせることでワクチンを構成する．さらに，ワクチンの取扱いを容易にするために定義部分を別ファイルには分けずに，ワクチンの実行処理部分の直後に定義部分を連結し1つの実行ファイルとする．定義部分には動的ヒューリスティック手法で得られた情報を整理し，以下に示す内容を持たせる．

IT シグネチャ　ヘッダ内の Import Table をベースとしたシグネチャ．Windows 側のハードディスク上でウイルスを特定するために用いる．

削除すべきレジストリ　（複数可）ウイルスにより追加された Windows 起動時の自動実行設定等がこれにあたる．

削除すべきファイルのあるフォルダ　（複数可）ウイルスにより追加されたファイル（ウイルスのコピー）があるフォルダを示す．ファイル名はランダムであることが考えられるため，このフォルダ内を IT シグネチャを用いたパターンマッチングで検索し，実際の削除すべきファイルを特定する．

ワクチンは実行されると定義部分からこれらの情報を読み出し，次項で述べる手順でウイルスの駆除を行う．

4.4.3 ワクチンの動作

ここではワクチンの一連の動作を示す．なお，4.4.2 項で述べたように，Linux 用と Windows 用との2種類のワクチンがあり，Linux 上で動作するワクチンは以下の動作フロー中の (3a) と (5) とを行わないものとする．これは，ユーザ用 Windows が動作していないため，ウイルス駆除が純粋にファイル操作のみで完了するためである．

(1) 実行部分の直後に添付された定義部分から，駆除を行うウイルス固有の情報を読み出す．
(2) レジストリの削除
  (a) “削除すべきレジストリ” の情報をもとに，追加されたレジストリを削除する．
  (b) “削除すべきレジストリ” の情報がある間，(2a) に戻り繰り返す．
(3) ファイルの削除
  (a) “削除すべきファイルのあるフォルダ” 内を，“IT シグネチャ” を用いて検索し削除すべきファイルを特定する．
  (b) (3a) で特定されたファイルを使用しているプロセスを検索し，存在した場合はそのプロセスを停止する．
  (c) (3a) で特定されたファイルを削除する．
  (d) “IT シグネチャ” が一致するファイルがある間，(3a) に戻り繰り返す．
(4) “削除すべきファイルのあるフォルダ” の情報がある間，(3) に戻り繰り返す．
(5) Windows を再起動する．

5. 実　　験

本システムの有効性を確認するため，実際のウイルスを用いた実験を行った．実験の対象としたウイルスを表 2 に示す．これらのウイルスは大学の総合情報処理センターのアンチウイルスにて捕獲されたもので，日本の組織における一般的なウイルス受信状況を反映しているものと考えられる．また，ウイルス名の判定には Symantec 社の Norton Antivirus を用いた．

5.1 未知ウイルス検出実験

まずはじめに，デバッガを用いた動的ヒューリスティック手法での未知ウイルス検出について実験を行う．対象とした実行ファイルは表 2 に示したウイルスと，Web サイト等から無作為に選択した，インストーラを含む 50 の通常の実行ファイルである．本システムに入力し，ファイルの作成とレジストリの変更の検出状況，およびウイルスかどうかの判定結果を確認する．結果を表 3 に示す．表中の File はファイルの作成が検出できたとき○，できなかったとき×，Registry はレジストリの変更が検出できたとき○，できなかったとき×，ウイルス判定はウイルスと判定されたとき○，されなかったとき×としている．

以下では結果について詳細に述べる．表ではほとん

表 2 対象としたウイルス
Table 2 Target viruses.

ウイルス名	総数
Erkez.B@mm	10
Mydoom.M@mm	52
Mydoom.BO@mm	149
Mydoom.BT@mm	16
Mytob.B@mm	11
Mytob.C@mm	69
Mytob.M@mm	57
Mytob.U@mm	245
Mytob.V@mm	14
Mytob.AF@mm	36
Mytob.AG@mm	75
Mytob.AH@mm	11
Mytob.AP@mm	136
Mytob.AS@mm	17
Mytob.CH@mm	53

表3 未知ウイルス検出結果
Table 3 Results of unknown virus detection.

ウイルス名	File	Registry	ウイルス判定
Erkez.B@mm	○	○	○
Mydoom.M@mm	○	○	○
Mydoom.BO@mm	○	○	○
Mydoom.BT@mm	×	○	○
Mytob.B@mm	○	×	○
Mytob.C@mm	×	×	×
Mytob.M@mm	○	×	○
Mytob.U@mm	○	×	○
Mytob.V@mm	○	×	○
Mytob.AF@mm	○	×	○
Mytob.AG@mm	○	×	○
Mytob.AH@mm	○	×	○
Mytob.AP@mm	○	×	○
Mytob.AS@mm	○	×	○
Mytob.CH@mm	○	○	○
通常の実行ファイル	×	×	×

表4 ITシグネチャの作成結果
Table 4 Results of generated IT signatures.

ウイルス名	総数	ITシグネチャ	MD5
Erkez.B@mm	10	1	7
Mydoom.M@mm	52	1	51
Mydoom.BO@mm	149	1	1
Mydoom.BT@mm	16	1	1
Mytob.B@mm	11	1	1
Mytob.C@mm	69	2	3
Mytob.M@mm	57	1	3
Mytob.U@mm	245	1	20
Mytob.V@mm	14	1	2
Mytob.AF@mm	36	2	4
Mytob.AG@mm	75	1	11
Mytob.AH@mm	11	4	4
Mytob.AP@mm	136	1	1
Mytob.AS@mm	17	2	2
Mytob.CH@mm	53	1	1

どのウイルスを正しくウイルスと判定できている一方，Mytob.C@mmに関してはいっさいの動作が検出できず，ウイルスと判定できなかった．他のMytobの亜種においてもMytob.CH@mm以外はレジストリの変更が検出できなかった．これは，最近のウイルスはUPXをはじめとするPacker[14]と呼ばれるソフトウェアによって圧縮がなされ，その際にアンチデバッグ機能が付加されることがあるためである．そのためデバッガが混乱させられ，レジストリ関連のAPIに正しくブレークポイントを設置できなかった．よって，アンチデバッグ機能の妨害を最小限にするためにはDLL Injection[15]等の技術により，ウイルスをデバッグモードに移行させずにAPIトレースを行うことを検討している．

Mytobとは逆にMydoom.BT@mmでは，ファイルの作成を検出することができなかった．レジストリの変更は検出したためウイルス判定には問題がなかったが，ワクチンの自動生成のための情報としては十分なものが得られなかった．この点については，ワクチンの自動生成実験で述べる．

最後に，今回の実験では通常の実行ファイル50個すべてに対し，いっさい誤検出することはなくウイルスと誤判定されることもなかった．通常の実行ファイルの中にはインストーラも含まれていたが，予想どおりメッセージを表示しユーザに確認を要求する段階で実行が停止するため，Windowsシステムにおいて重要な部分への変更が行われることはなかった．ただし，3章で述べたように，実際の運用ではWindows起動時に自動実行が行われる実行ファイルのみが本システムに入力される．そのため，本来インストーラが検査されること自体が希であると考えられる．また，通常の実行ファイルの中には，ファイルを生成したりレジストリを変更したりするものもあったが，それらは自動実行の設定やシステムフォルダに対するものではなかったため，検出対象とはならなかった．

5.2 ワクチンの自動生成実験

ここではワクチンの自動生成に関する実験について述べる．はじめに，4.4.1項で述べたITシグネチャが実行ファイル固有の情報となっているかを確認する．次に，生成されたワクチンで実際にウイルスに感染したPCから，ウイルスが駆除可能かを実験する．

5.2.1 ITシグネチャに関する実験

本システムで生成されるワクチンはウイルスの識別をITシグネチャに依存している．したがって，ITシグネチャが実行ファイルにとって十分に固有な値でなければ，ウイルス以外のファイルを誤検出し，正常なファイルを削除してしまう可能性がある．そこで，表2のウイルスからITシグネチャを作成し，互いに異なる1,000個の通常の実行ファイルに対しパターンマッチングを行い，誤検出の有無を確認する．なお，通常の実行ファイルは複数のWindows PCから無作為に選択されたものである．

表4に各ウイルスから作成したITシグネチャのパターン数を示す．たとえば，表中のMytob.U@mmは手元に245個の実行ファイルが存在し，そのすべてからITシグネチャを作成しても1パターンとなり，すべて同様のITシグネチャが得られたことを意味している．一方，同様に245個のMytob.U@mmすべてからMD5でハッシュ値をとると，20パターンのハッシュ値が得られたことを意味している．これはMytob.U@mmが，おそらくはハッシュ等による

同一ファイル検索を回避するために，自身をコピーするときに特に意味のない情報をランダムに付加しているためである．しかし，IT シグネチャで検索を行えば，Mytob.U@mm のようなランダム性は無視できるため確実な駆除が期待できる．

Mytob.C@mm，Mytob.AF@mm，Mytob.AH@mm，Mytob.AS@mm の 4 つのウイルスは，IT シグネチャのパターン数が 1 つにはならなかった．これは Mytob.U@mm 等の場合とは異なり，感染時にランダムな変更を加えた結果ではなく，ウイルスのバイナリそのものにパターンが存在しているためである．アンチウイルスメーカーは同一機能を持っているものを同種類のウイルスとして分類を行っているため，実験でも示されるようにバイナリとしては複数パターンが存在している場合がある．つまり，この例では Symantec 社の分類では同一のウイルスとされているが，IT シグネチャを識別に用いる本システムとしては異なるウイルスと認識されたと考えられる．これは Symantec 社の分類における Mydoom.BO は Trend Micro 社では Mytob.EC，Mytob.ED という種類の異なる 2 つのウイルスであると判別される事例と同様である[16]．

各ウイルスから作成した合計 21 パターンの IT シグネチャそれぞれで，1,000 個の通常の実行ファイルに対しパターンマッチングを行ったところ，いずれの IT シグネチャも通常ファイルをウイルスとして誤検出することはなかった．しかし，21 パターンの IT シグネチャで，表 2 に示す 951 個のウイルスに対しパターンマッチングを行ったところ，Mytob.M@mm と Mytob.AF@mm から作成したシグネチャで誤検出が発生した．それぞれの IT シグネチャで発生した誤検出ファイル数を表 5 に示す．表 5 では Mytob.M@mm，Mytob.AF@mm 1，Mytob.AF@mm 2 の IT シグネチャを用いてウイルスの検出を試みたところ，Mytob.M@mm の IT シグネチャで 1 個の Mytob.AF@mm を誤検出し，Mytob.AF@mm 2 の IT シグネチャが 57 個の Mytob.M@mm を誤検出したことを表している．これは，Mytob.M@mm の 57 個すべてと Mytob.AF@mm の 36 個中 1 個の IT シグネチャが，まったく同様のものになっているのが原因である．

表 5 IT シグネチャによる誤検出ファイル数
Table 5 Results of false positive detection by IT signatures.

IT シグネチャ	スキャン対象ウイルス	
	Mytob.M	Mytob.AF
Mytob.M@mm		1
Mytob.AF@mm 1		
2	57	

この誤検出の発生原因や発生頻度は，Mytob.AF@mm 側に誤検出したウイルスが 1 個しか存在しないためはっきりしない．しかし，ウイルスが互いに誤検出する限りでは，片方のウイルスを駆除する際に，もう片方のウイルスが駆除されるにすぎない．しかも，その状況は両方のウイルスに同時に感染している場合にだけ起こりうるため，確率的にはかなり低い値になるものと思われる．

5.2.2 ウイルス駆除実験

自動生成されたワクチンを用いて，感染済み PC からのウイルス駆除実験を行う．実験の対象としたウイルスは同じく表 2 に示すものである．ただし，Mytob.C@mm だけは未知ウイルス検出段階で失敗しているため，ワクチンが生成された残りの 14 ウイルスについて実験する．実験は実際に PC をウイルスに感染させた後，本システムで PC をブートし未知ウイルスのワクチン自動生成および，そのワクチンによるウイルス駆除を行う．次にユーザ用 Windows を起動させウイルスが駆除されたかどうかを確認する．

表 6 に実験結果を示す．まず，未知ウイルス検出の際に，ファイルの作成とレジストリの変更の両方を検出できた，Erkez.B@mm，Mydoom.M@mm，Mydoom.BO@mm，Mytob.CH@mm について述べる．この 4 つのウイルスについては，作成されたファイルと追加されたレジストリがともに削除され，Windows 起動後にウイルスのプロセスが存在しないことを確認した．すなわち，完全にウイルスを駆除することができたといえる．

次に，ファイルの作成は検出できた一方，レジストリの変更を検出できなかった Mytob.CH@mm を除く Mytob の亜種に関して述べる．検出そのものができ

表 6 ワクチンによる駆除結果
Table 6 Result of virus extermination.

ウイルス名	File	Registry	駆除判定
Erkez.B@mm	○	○	○
Mydoom.M@mm	○	○	○
Mydoom.BO@mm	○	○	○
Mydoom.BT@mm	−	○	×
Mytob.B@mm	○	−	○
Mytob.C@mm	−	−	−
Mytob.M@mm	○	−	○
Mytob.U@mm	○	−	○
Mytob.V@mm	○	−	○
Mytob.AF@mm	○	−	○
Mytob.AG@mm	○	−	○
Mytob.AH@mm	○	−	○
Mytob.AP@mm	○	−	○
Mytob.AS@mm	○	−	○
Mytob.CH@mm	○	○	○

ていないため，削除すべきレジストリに関する情報もなく，レジストリの削除は行われなかった．しかし，ファイルの削除は正常に行われたため，Windows 起動時に自動実行するようレジストリに設定は残っていても，実行すべきウイルスが存在しないという状況になった．実行ファイルの存在しない自動実行設定は無効になるので，Windows 起動後にウイルスのプロセスが起動することはなくなり，ウイルスは駆除されたと判断できる．

Mytob とは逆に，レジストリの変更は検出できたが，ファイルの作成を検出できなかった Mydoom.BT@mm は正しく駆除が行えなかった．ワクチンにより自動起動に関するレジストリは削除できるため，Windows 起動時にウイルスが動作することはなくなった．ただし，ウイルス本体は PC 内に残っているため駆除できたとはいえない．今回実装したワクチンは，「ウイルス動作の逆を行うことが駆除である」との仮定に基づく単純なものである．そのため，この事例ではレジストリからは自動実行の設定削除のみを行っている．しかし，検証環境内でウイルス判定がなされているため，IT シグネチャの生成そのものはすでに行われている．よってレジストリ情報と組み合わせることでウイルス本体を探し出すことは理論上可能である．

最後に，Erkez.B@mm について述べる．図 6 に示されるように，ワクチンは Erkez.B@mm を完全に駆除することができた．これは IT シグネチャによるウイルス検索が有効に機能したためである．Erkez.B@mm は自身のコピーを作成する際に，そのファイル名をランダムに決定するウイルスである．そのため，ファイル名による単純な検索では検出することはできない．また，自身をコピーする際にほんの数 Byte ではあるが変更を加えるため，ハッシュを用いた検索でも検出は不可能である．よって，Erkez.B@mm の駆除が行えたことは，本システムで用いた IT シグネチャの有効性を示す例であると考える．

6. 関連研究との比較

6.1 動的ヒューリスティック手法

本システムでも採用している動的ヒューリスティック手法では大きく分けると以下の 2 つの概念でウイルスを検出しようとしてきた．

- 悪意ある行動をしたものがウイルス
- 通常見られない例外行動をしたものがウイルス

ただし現実的にはユーザが普段利用するプログラムであっても，インストーラ等は悪意ある行動に似た振舞いをする．そのため，これまでは論文 7)～11) に示されるように例外行動検知に研究の重点が置かれてきた．

そのため，論文 8)～10) 等で提案された手法は平常状態を定義するために学習期間を必要とする．学習期間においては，個々の PC 所有者の振舞いを API のトレース等を通じて収集し，平常状態を定義する．検出期間に移行してからは，定義された平常状態から外れた動作を例外行動として検出する．すなわち，このときの学習とは PC 稼働時におけるユーザが行う通常の行動を無視できるようにする過程に等しい．一方，本システムではユーザによる操作をいっさい排除した仮想 PC が用いられるため学習期間は必要なく，多くのウイルスが一般的に行う行動を単純に監視するだけで，未知ウイルスを正しく検出している．

また，静的ヒューリスティック手法を組み合わせた例外行動検出手法も論文 11) で提案されている．この手法ではプログラムを実行する前に，呼び出される API とその際にジャンプする仮想アドレスの一覧を得る．その後プログラムを実行し，API 呼び出しと仮想アドレスとの組合せが事前に得たリストと一致しなかった場合に例外行動として検出を行う．これにより，実行中プロセスの構造がメモリ内で大きく変わるような動作を検出する．これはウイルスのポリモーフィック機能が発動した場合等に見られる現象である．しかし，現実世界では広く拡散するウイルスほど，単純な技術で実装されているものが多く，実行されると単にシステムを改ざんするようなウイルスを検出することはできない．一方，本システムでは仮想 PC 内での改ざん活動さえ確認されれば，ウイルスがどのような技術で作成されているのかに関係なく検出が可能である．

6.2 シグネチャ生成

自動的にシグネチャを生成する手法は論文 17) において提案されている．前述の論文で対象としているのは，他の実行ファイルに感染するような種類のウイルスであり，事前に用意した囮実行ファイルにウイルスを感染させ，感染前後の状態を比較することで異常を検知している．さらに，この際の実行ファイルの差分をウイルス本体であると判断し，シグネチャの自動生成を行っている．ただし，一般の実行ファイルに対する誤検出を生じさせたシグネチャの大部分が，C や Pascal 等の高級言語で記述されたウイルスから生成されたものであると報告されている．

論文 17) の手法は，ウイルスがアセンブリ言語で記述され実行ファイル内に作者が残した特有の特徴が含まれることを前提としている．しかし，高級言語で記述されたウイルスはコンパイル時に最適化が行われ

るため，ソースコードの記述の仕方によらず，同様の処理はだれが記述しても同様の機械語が生成されることが多い．そのため，ウイルスと一般の実行ファイルとの両方に含まれるような情報を持つシグネチャが生成され誤検出を引き起こした．現在のウイルスは高級言語で記述されるウイルスが大部分を占めるため，論文17)の手法では一般の実行ファイルに対する誤検出が多発することが予想される．一方，本システムでは高級言語で書かれた実行ファイルにおけるPE形式のユニーク部分を明示的に選択しているためウイルスを正確に検出可能である．

7. おわりに

本研究では，クライアントPCにおける未知ウイルス検出の最終防御層としてワクチンを自動生成しウイルス駆除を行うUSBフラッシュメモリを開発した．開発したUSBフラッシュメモリ内にはブート可能なLinuxが内蔵されており，それをUSBポートに挿入しPCの電源を入れると独自の検証環境が起動する．次に，デバッグ機能を利用した動的ヒューリスティック手法により，未知ウイルスを検出すると同時にLinuxとWindowsで動作するワクチンを生成する．実際にこれらのワクチンでウイルスが駆除可能か検証を行ったところ，通常のファイルを誤検出し誤って駆除することもなく，実験した15種類のウイルスのうち13種類のウイルスを駆除することが可能であった．

今後は，未知ウイルス検出のための動的ヒューリスティック手法の精度を向上させるとともに，本システムを未知ウイルス検出専用のLinuxディストリビューションとして一般配布可能な形へと改良を行っていきたい．

参考文献

1) Online Publication: Symantec Internet Security Threat Report VIII, Technical report, Symantec Corporation (2005).
2) Marx, A.: Anti-Virus Outbreak Response Testing and Impact, *Virus Bulletin 2004 Conference Presentation*, Chicago (2004).
3) Symantec Corporation: Security Response. http://securityresponse.symantec.com/
4) VMware, Inc.: VMware — Virtualization Software. http://www.vmware.com
5) Sung, A.H., Xu, J., Chavez, P. and Mukkamala, S.: Static Analyzer of Vicious Executables (SAVE), *ACSAC*, pp.326–334 (2004).
6) Christodorescu, M. and Jha, S.: Static analysis of executables to detect malicious patterns, *12th USENIX Security Symposium*, Washington, DC, Advanced Computing Systems Association (2003).
7) Forrest, S., Hofmeyr, S.A., Somayaji, A. and Longstaff, T.A.: A Sense of Self for Unix Processes, *Proc. 1996 IEEE Symposium on Research in Security and Privacy*, pp.120–128, IEEE Computer Society Press (1996).
8) Feng, H.H., Kolesnikov, O.M., Fogla, P., Lee, W. and Gong, W.: Anomaly Detection Using Call Stack Information, *Proc. 2003 IEEE Symposium on Security and Privacy*, Oakland, IEEE, pp.62–75 (2003).
9) Sekar, R., Gupta, A., Frullo, J., Shanbhag, T., Tiwari, A., Yang, H. and Zhou, S.: Specification-based anomaly detection: A new approach for detecting network intrusions, *Proc. 9th ACM conference on Computer and communications security*, pp.265–274, ACM Press (2002).
10) Ghosh, A.K., Schwartzbard, A. and Schatz, M.: Learning Program Behavior Profiles for Intrusion Detection, *Proc. Workshop on Intrusion Detection and Network Monitoring*, USENIX, pp.51–62 (1999).
11) Rabek, J.C., Khazan, R.I., Lewandowski, S.M. and Cunningham, R.K.: Detection of Injected Dynamically Generated and Obfuscated Malicious Code, *Proc. Workshop on Rapid Malcode (WORM'03)*, ACM, pp.76–82 (2003).
12) Nakaya, N., Ryuiti, K., Koui, Y. and Yoshida, H.: The Network System Defended from Infection of Unknown E-Mail Viruses, *IPSJ Journal*, Vol.45, No.8, pp.1908–1920 (2004).
13) Microsoft Corporation: *Microsoft Portable Executable and Common Object File Format Specification* (1999).
14) Oberhumer, M.F.X.J., Molnar, L. and Reiser, J.F.: UPX the Ultimate Packer for eXecutables. http://upx.sourceforge.net/
15) Wikipedia: DLL Injection. http://en.wikipedia.org/wiki/DLL_Injection
16) Symantec, Inc.: security response, W32.Mydoom.BO@mm. http://securityresponse.symantec.com/region/jp/avcenter/venc/data/jp-w32.mydoom.bo@mm.html
17) Kephart, J. and Arnold, B.: Automatic Extraction of Computer Virus Signatures, *the 4th Virus Bulletin International Conference*, England, Virus Bulletin Ltd., pp.178–184 (1994).

(平成18年6月27日受付)
(平成19年1月9日採録)

小池 竜一（学生会員）

2003年岩手大学工学部情報工学科卒業．2005年同大学院工学研究科博士前期課程修了，同年同大学院工学研究科博士後期課程入学，現在に至る．ネットワークセキュリティに関する研究に従事．電子情報通信学会学生員．

中谷 直司

1994年埼玉大学工学部電子工学科卒業．1996年同大学院博士前期課程修了．1999年同大学院博士後期課程修了．同年岩手大工学部情報システム工学科教務職員．2001年同科助手，現在に至る．進化型アルゴリズム，ネットワークセキュリティに関する研究に従事．博士（学術）．電子情報通信学会会員．

厚井 裕司（正会員）

1970年東京理科大学理学部応用物理学科卒業．同年三菱電機（株）入社．2001年岩手大学工学部情報システム工学科教授，現在に至る．主として，マルチメディアネットワーク，ネットワークセキュリティ，RF-IDタグに関する研究に従事．工学博士．IEEE，電子情報通信学会各会員．