プログラムの振る舞いに着目したマルウェア検知方式の研究

静岡大学　博士論文

プログラムの振る舞いに着目したマルウェア検知方式の研究

平成19年6月

大学院　理工学研究科
設計科学専攻

松本　隆明

はじめに

昨今のインターネットの爆発的な普及に伴って、我々の社会経済活動は、インターネットを前提としたコンピュータシステムに大きく依存しつつある状況にある。このため、ひとたびこうしたコンピュータシステムが障害となるとその影響は計り知れないものとなる。また、コンピュータシステムがビジネス的に大きな価値を持つに従って、そうした価値の盗用を目的として、コンピュータウィルスなどのコンピュータシステムを狙いとしたサイバー攻撃が最近急激に増加しており、大きな社会的問題へと発展している。サイバーモールへの攻撃によるサービス停止、ATM ネットワークの停止、オンライン予約システムの停止、ID やパスワードの盗用による窃盗行為など社会活動に甚大な影響を与える事例が多数発生するようになってきており、こうした攻撃に対する対策は社会的にも急務となっている。

最近ではコンピュータウィルスの活動内容もきわめて複雑化してきており、いわゆる狭義のコンピュータウィルスという定義だけでは当てはまらないものが増えてきている。そこで現在は、コンピュータシステムに障害を与えるような悪意を持ったソフトウェアであるウィルス、ワーム、トロイの木馬の 3 種類のソフトウェアは、総称してマルウェアと呼ばれている。

マルウェアに対する対策として現在主流となっているのはパターンマッチング法と呼ばれる方式であり、マルウェアプログラムの特徴的なコード部分を抽出して「定義ファイル」として定義しておき、検査対象プログラムの中に、定義ファイルにあるコード部分と一致する箇所が存在するかどうかによってマルウェアを検出するという方式である。この方式は、既知のマルウェアに対しては比較的簡単に、かつマルウェアが実行される前（感染の前）に検知が行えるため非常に有効な方式であるが、特徴的なコード部分が分かっていない未知のマルウェアに対しては当然ながら無力である。未知のマルウェアは、基本的に、感染前にこれを完全に検知することはできない。そこで、たとえ感染は許してしまったとしても、マルウェアが引き起こす異常動作を検出し、その時点でマルウェアの動作を停止させることにより、他システムへのさらなる伝染といった被害の拡大を防止する方式が提案されている。こうした方式のうち最も検知精度が高いものがビヘイビアブロッキング法と呼ばれる方式であり、システムファイルの変更、レジストリの書き換え、他コンピュータへの感染などのマルウェアらしい振る舞いを検出してマルウェアを検知するという方式である。

しかしながら、ビヘイビアブロッキング法に基づく既存の方式では、どこまでの動きならば異常動作と見なすかの閾値を正しく設定することが難しく、正常なプログラムまでマルウェアと見なしてしまうという誤検知の発生が多いという問題がある。

本論文では、真にマルウェアらしい振る舞いを定式化することで、従来方式に比べ、誤検知の発生を抑え、かつより高精度にマルウェアの検知を可能とする方式の提案とその有効性の検証を行う。

具体的には、他コンピュータへのさらなる感染を引き起こすウィルスとワームに対しては、感染行為の過程で必要となる自分自身のプログラムファイルを READ するという動作を真にマルウェアらしい振る舞いとして規定し、OS のファイルシステムにおいてそれぞれのプロセスが行うファイル READ を監視することでマルウェアの検知を行う方式を提案する。さらに、ビヘイビアブロッキング法においては、感染状態のコンピュータ上では検知動作そのものがマルウェアによって妨害されることもあるため、本方式を補完する方式として、他コンピュータへの感染時に送信するデータと既に受信したデータとの相関関係を検査することで、マルウェアによる感染行為を検知する手法を考案した。本手法は、検査対象のコンピュータを外部から監視することも可能となるため、これをファイル READ の検知と組み合わせて適用することでより堅牢な監視が行えることとなる。さらに、送受信データ相関監視では、感染時に送信するデータを擬似的な定義ファイルとして利用することで、リアルタイムにかつ効率的に監視システム全体の運用が行えるようになる。

一方、感染行為を伴わないトロイの木馬に対しては、その社会的ならびに金銭的な被害の大きさに比べて対策技術が進んでいない、キーロガーを対象として対策方式の提案を行う。キーロガーが行うキー情報の取得行為を、キーロガーが使用する OS の API の観点からマルウェアらしい振る舞いとして定式化し、その API の動作を監視することでマルウェアの検知を行う。従来の多くのアンチスパイウェア製品と呼ばれるものでは、多種多様な攻撃者の意図を持つスパイウェア（トロイの木馬）を同一の方法で検知しようとするため、アンチウィルス製品に比べてどうしても検知精度が低下するという問題があった。本論文では、キーロガーというトロイの木馬に特定して、その振る舞いをより厳密に規定することができたため、従来の製品では検知できなかったキーロガーの検知を可能とした。

本論文では、ウィルス／ワームとトロイの木馬のそれぞれに対して、提案した方式が、正規のプログラムに対する誤検知が少なく、かつマルウェアに対して極めて検知精度が高い方式であることを基礎実験により検証している。特に、これまで検知が困難であった未知のマルウェアならびに、感染に際して自らの形を変化させて他システムへの感染を行う「ミューテーション型」と呼ばれるマルウェアについても検知が可能であることを検証している。また、実際に検知を行う際のオーバヘッドについても実測を行い、リアルタイムでの検知を行っても実用上も問題とはならない方式であることを確認している。

目次

第1章　序論
1．1　背景 …… 5
1．2　研究の目的 ……11
1．3　構成 ……15

第2章　自己ファイルREADの検出によるマルウェア検知方式
2．1　従来技術における課題 ……16
2．2　真にマルウェアらしい振る舞いの規定 ……24
2．3　提案方式 ……28
2．3．1　検証実験 ……29
2．3．2　考察 ……37
2．4　まとめ ……46

第3章　送受信データ間の相関に基づくマルウェア蔓延防止方式
3．1　従来技術における課題 ……48
3．2　提案方式 ……50
3．2．1　マルウェアによる感染行為の規定 ……50
3．2．2　検証実験 ……52
3．2．3　考察 ……62
3．3　まとめ ……68

第4章　動的API検知方式によるキーロガーの検知方式
4．1　従来技術における課題 ……70
4．1．1　対象とするトロイの木馬……70
4．1．2　従来方式の限界……73
4．2　真にキーロガーらしい振る舞いの規定 ……77
4．3　提案方式 ……82
4．3．1　検証実験 ……82
4．3．2　考察 ……90
4．4　まとめ ……93

第5章　結論

５．１　総括 ……94
５．２　今後の課題 ……98

謝辞 ……99
参考文献 ……100
関連論文 ……108

第１章
序論

1.1 背景

1946 年の米国ペンシルバニア大学での世界最初のコンピュータである ENIAC[1]の開発以来、この 50～60 年間におけるコンピュータシステムの急速な発展にはめざましいものがある☆1。当初、コンピュータシステムは、それまで人間が人手で計算していた定型的で大量のデータを単純に処理する数値計算の電子データ処理（Electronic Data Processing）に適用することが主な利用方法であった。その後、MIS（Management Information System）や DSS（Decision Support System）と呼ばれるような、経営管理や意思決定といった企業の管理面への利用が広まってきた。さらに、1980 年代に入ると、パーソナル・コンピュータやワードプロセッシング・システムなどの登場によって、企業内の業務の効率化への適用という、いわゆる OA（Office Automation）化が進展することとなる。1990 年代以降は、こうしたオフィス内の WS（Work Station）と、企業管理用の基幹システムとを組み合わせた統合管理システムとしての利用が広がり、また、EDI（Electronic Data Interchange）[2]や CALS（Continuous Acquisition and Life-cycle Support）[3]という標準化の考え方により、各企業同士のシステムを相互に接続した企業間連携が進むようになり、最近では当たり前のようになってきた電子商取引（EC：Electronic Commerce）の時代へと進んできている。

一方、1969 年に軍事用の指揮・統制管理用のネットワークとして開発された ARPANET[4]は、その後のコンピュータシステムの発展にとって重要な意味を持つネットワークとなった。それまでのネットワークは、電話網に代表されるようにネットワークの中心に位置する交換機が通信の処理の全てを管理する形態であったが、これでは中央交換機が故障（戦時下においては、破壊）してしまうと通信の全てがマヒしてしまうため、ARPANET では自立分散的にパケット単位で通信する形態が考案された。さらに、1980 年代に入りこの ARPANET が今日のインターネットとして民間用にも開放されるとともに、1990 年にはネットワーク上で簡単に文書を交換できる仕組みとして

☆1 ENIAC は、いわゆるノイマン型の計算機ではないため、世界で最初のコンピュータであるという説には異論もあるが、一般的には幅広く電子計算機という意味で ENIAC が世界最初のコンピュータであるという考え方が定説となっている。

WWW（World Wide Web）が発明され、インターネットを利用したデータ交換が一気に世界的な規模で加速されることとなった（図 1)。

EC への適用が進んできたコンピュータシステムも、2000 年以降は、こうしたインターネットの普及を受けて、Web サービスと呼ばれるインターネットベースのサービスやシステム連携が急速に進みつつある。また、携帯電話や PDA（Personal Digital Assistant）に代表されるモバイル端末の普及、さらには RFID（Radio Frequency Identification）のようなセンサ機器の広がりによって、こうした端末や機器を業務システムに連携させたり、インターネットへ接続させたりする、いわゆるユビキタスサービスが実用化されるようになってきた。さらに、最近では、次世代の Web と呼ばれる Web2.0 [5]という概念が台頭しつつある。Web2.0 はきわめて幅広い概念を含んでおり、例えば、扱うコンテンツが単なるテキスト情報からマルチメディア情報に変わるとともに、情報の提供形態もこれまでのように情報提供者から利用者へという一方向ではなく、だれでもが情報提供者であり利用者にもなるといった利用形態の変化などが含まれている。

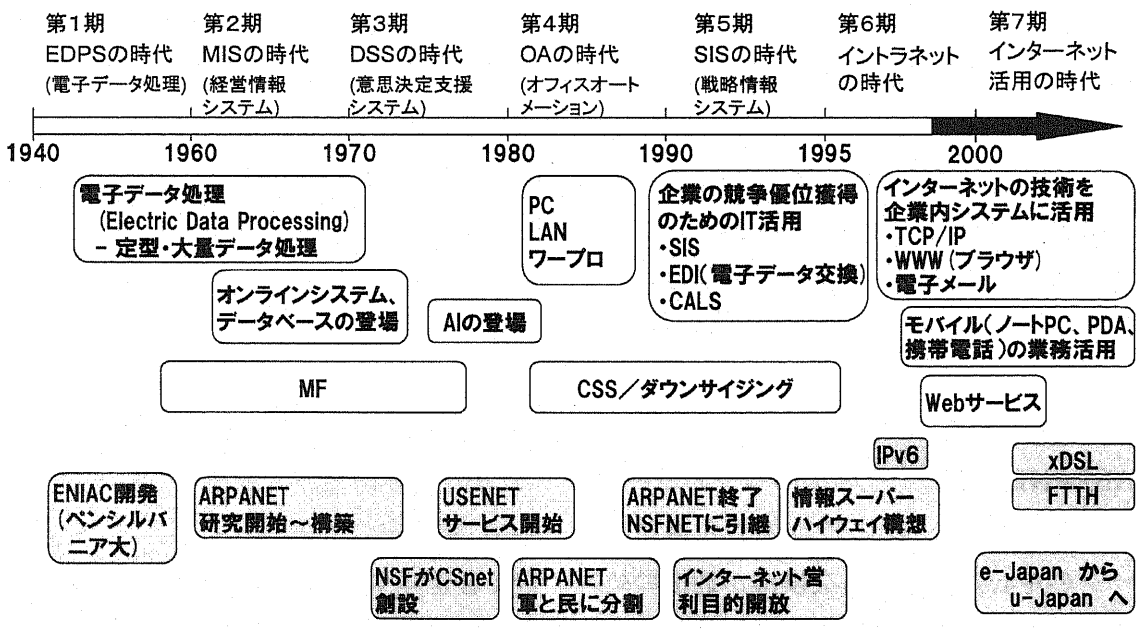

図 1　情報システムの変遷

Fig. 1　Development of the information processing system

こうした、情報システムの世界的な進展に呼応して、わが国においてもインターネッ

トの普及と EC の適用は爆発的な勢いで伸びている。総務省による平成 18 年版情報通信白書[6]によれば、2005 年のわが国のインターネット利用人口は 8,529 万人に達しており、全人口における普及率は 66.8%と実に日本人の 2/3 はインターネット利用者であるとの報告がなされている（図 2)。また、世帯、企業および事業所におけるインターネットの普及率も年々増加しており、2005 年末時点の世帯普及率は 87.0%、事業所普及率は 85.7%であり、企業普及率に至っては 99.1%とほぼ 100%に近づきつつある状況である[7]。

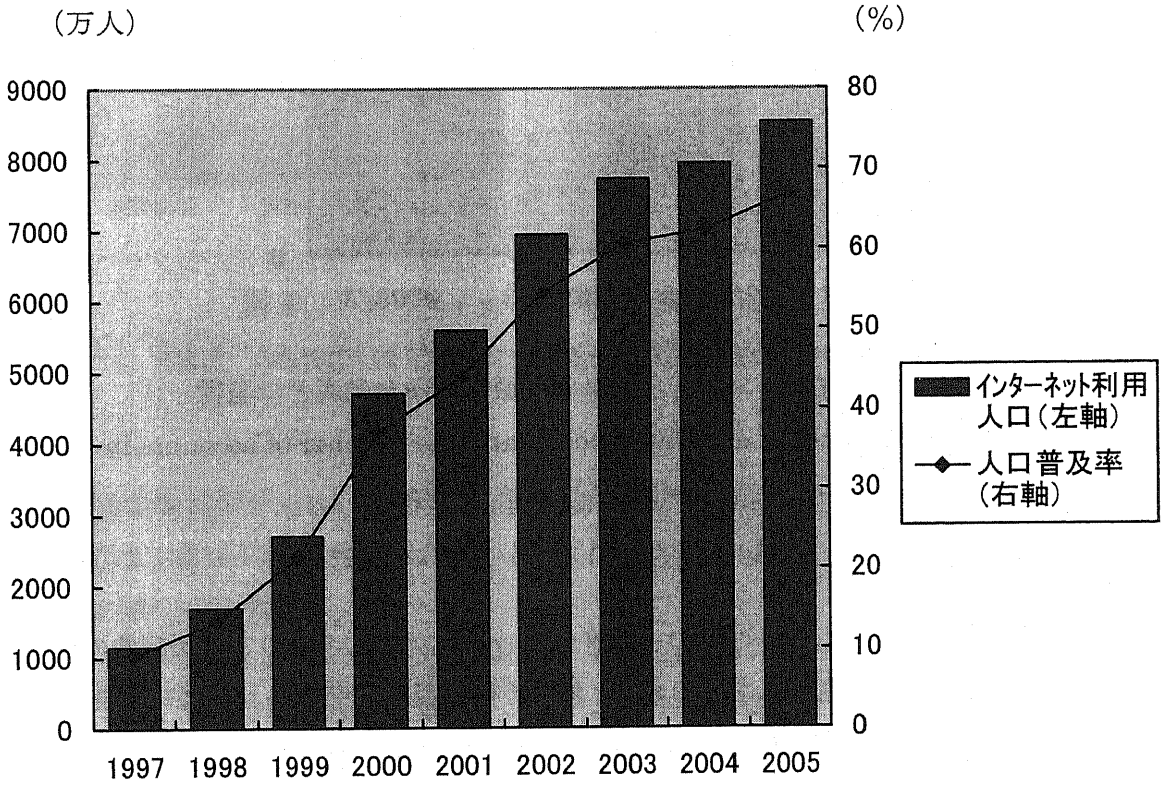

図 2　わが国のインターネット利用状況

Fig. 2　The diffusion of the Internet in Japan

さらに、こうしたインターネットの普及に伴って、インターネットを前提としたコンピュータシステムは社会経済活動の基盤をなすインフラストラクチャとして無くてはならないものとなりつつある。前出の平成 18 年版情報通信白書によれば、例えば金融分野では、代表的なインターネット専業銀行 4 行の口座数及び預金残高は 2005 年 3 月末時点で前年度に比べそれぞれ 33.5%と 47.0%の増加と驚異的な伸びを示しており、口座数にして 250 万口座、全体の預金残高は 1 兆円を超えている（図 3)。

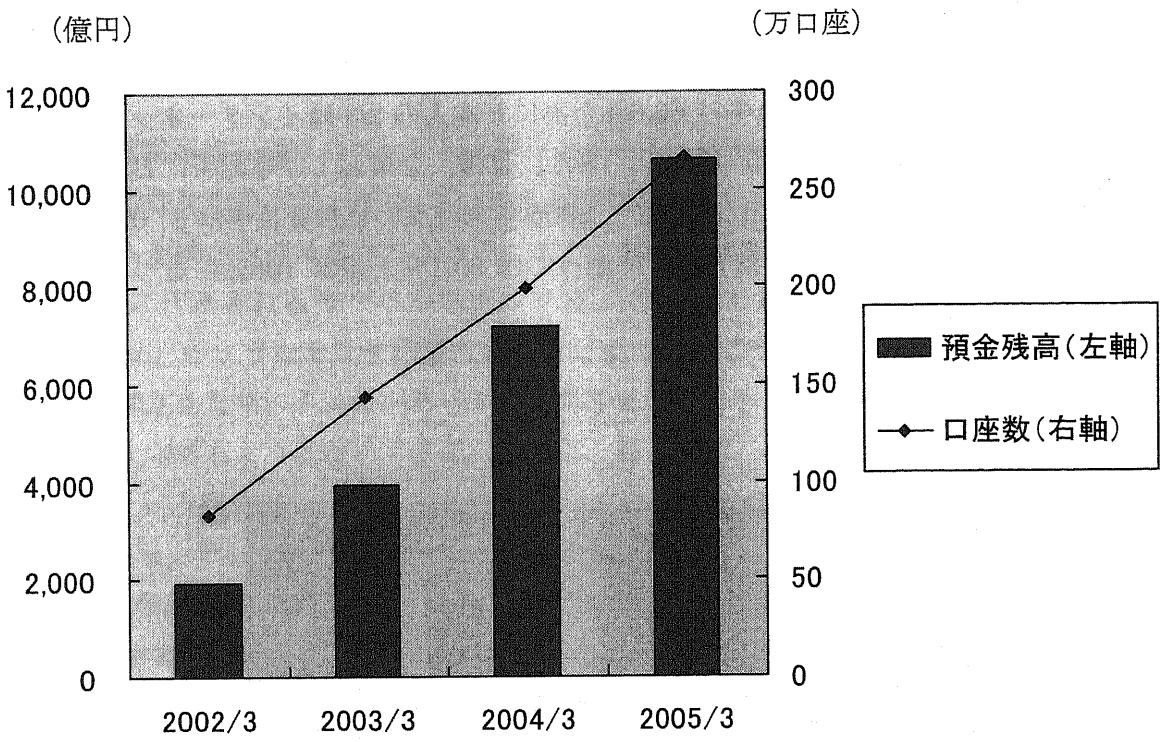

図 3　主なインターネット専業銀行の預金残高と口座数

Fig.3　The balance of account deposits and the number of accounts in major Internet banks in Japan

証券業界においても証券取引のインターネット化が進んでおり、有価証券の購入・売却を行っている利用者のうち、49.8%とおよそ半数がインターネット経由で取引を行っており、その年間取引代金の総額は 2005 年 9 月末時点で 160 兆円にも達していることが報告されている。対前年比で比較してもその伸び率は 38.1%に達している（図 4)。

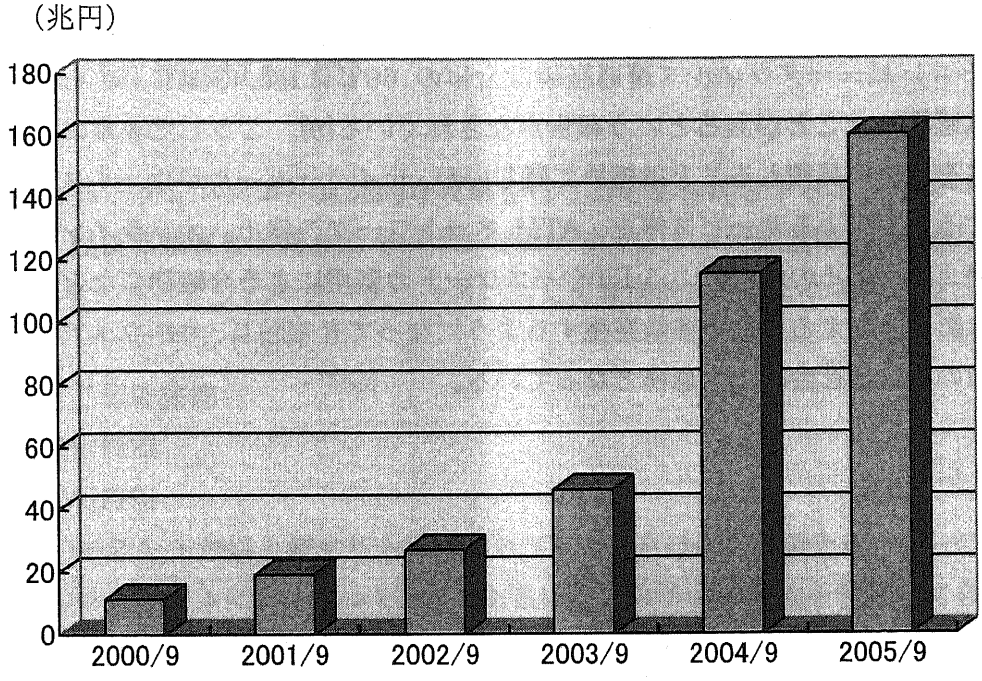

図 4　インターネットによる証券取引金額

Fig.4　Amount of stock trading through the Internet

また、企業間取引であるいわゆる BtoB 市場も EC 化が進んでおり、その市場規模は 2004 年では 102 兆円と見積もられている。EC による BtoB 市場はこの 6 年間でおよそ 12 倍に膨れている。

このように、我々の社会経済活動はインターネットを前提としたコンピュータシステムに大きく依存しつつある状況にあり、ひとたびこうしたシステムが障害となるとその影響は計り知れないものとなる。また、コンピュータシステムが金銭の取引に使われるようになりビジネス的に大きな価値を持つに従って、そうした価値の盗用を目的として、コンピュータウィルスなどのコンピュータシステムを狙いとしたサイバー攻撃が急激に増加しており、社会的にも大きな問題となっている。一般的にコンピュータシステムにより構築される現代の情報化社会には、

① 匿名性：行動者の特定が困難であること
② 無痕跡性：電子的な情報であるため物理的な痕跡がないこと
③ 時間的・場所的無限定性：24 時間社会であり、どこでも利用できること
④ 超高速性：操作一つで一瞬にして処理が完結すること

といった特徴があり、これらの特徴が犯罪者にとっても好都合な条件となり、犯意を高める要因ともなっている[8]。

情報処理推進機構（IPA）セキュリティセンターの調査によれば、日本国内のコンピ

ュータシステム利用者へのアンケート結果において、2005 年 1 月から 12 月の 1 年間におけるコンピュータウィルス遭遇経験は全体の 69%に達しており、さらに全体の 15.6%は感染したことがあるという報告がなされている[9]。こうした攻撃による被害の大きさを厳密に見積もることは極めて難しいが、サイバーモールがウィルス／ワームに感染することによるサービス停止、ATM ネットワークの停止、オンライン予約システムの停止、トロイの木馬による ID やパスワードの盗用による金銭窃盗など社会活動に甚大な影響を与える事例が多数発生するようになってきており、マルウェアに対する対策技術の開発は社会的にも急務となっている。

1.2 研究の目的

コンピュータシステムへ意図的な攻撃を行うことを目的とするコンピュータウィルスとは、経済産業省告示第 952 号「コンピュータウィルス対策基準」の定義[10]によると、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あり、

① 自己伝染機能
② 潜伏機能
③ 発病機能

の 3 つの機能を 1 つ以上有するものとされている。自己伝染機能とは、自らの機能によって他のプログラムに自らをコピーし、あるいはシステムの機能を利用して自らを他のシステムにコピーすることにより、他のプログラムやシステムに伝染する機能である。潜伏機能とは、発病するための特定時刻、一定時間、処理回数等の条件を記憶させて、発病するまで症状を出さない機能である。発病機能とは、プログラム、データ等のファイルの破壊を行ったり、設計者の意図しない動作をする等の機能のことであると定義されている。

これに対して、最近ではコンピュータウィルスの活動内容もきわめて複雑化してきており、前述のいわゆる狭義のコンピュータウィルスという定義だけでは当てはまらないものが増えてきている[11][12]。そこで、最近では、コンピュータシステムに障害を与えるような悪意を持ったソフトウェアのことを総称してマルウェアと呼ぶようになってきている。マルウェアとは、悪意のあるソフトウェアという意味の malicious software の省略形から来ているもので、以下の 3 種類のソフトウェアの総称である[13]。

① ウィルス
② ワーム
③ トロイの木馬

ウィルスとは、人間に感染するウィルスのように宿主になるプログラムに自プログラムを添付させて感染するマルウェアであり、宿主となっているプログラムが実行されると、添付されたウィルスも実行されて、ファイル破壊や情報改ざんなどの悪意を持った動作を行う。また、感染範囲を広げるためにさらに他の新しい宿主に自プログラムを添付させようと試みるウィルスも存在する。例えば、電子メールクライアントを宿主として、電子メールの添付ファイルとして自分自身を感染させていくウィルスはその典型的な例である。また、こうしたファイル感染ウィルス以外にも、コンピュータのブートセ

クタに存在するブートプログラムに感染して、コンピュータの立ち上げ時に活動するものや、Microsoft Word や Excel といったプログラムのマクロとして悪意の動作を行うマクロウィルスなどが存在する。

ワームとは、ウィルスとは異なり、宿主を必要とせず、単独で動作を行うマルウェアであり、コンピュータシステムに存在するセキュリティホール（セキュリティ上の弱点となるソフトウェアのバグ）を悪用して他コンピュータに自分自身のプログラムのコピーを送り自動的に悪意を持った動作を開始する。ワームは基本的には利用者の操作を介さずに独自に伝染行為を行っていくため、早急に感染行為を阻止しないと、ねずみ算的に被害の範囲が広がっていくという危険がある[14]。

トロイの木馬とは[15]、利用者から見ると正しいプログラムのように偽装されたマルウェアであり、例えば、正常なソフトウェアのアップデートプログラムの配信を装い、更新センタのサーバを騙ってのダウンロードサービスのような形で、または更新センタを騙っての電子メールの添付ファイルあるいはCD-ROMのような形で、利用者に送信、郵送される場合が多い。トロイの木馬は、ウィルスのように正常なプログラムに寄生することもなく、またワームのように自己伝染を行うこともないが、利用者が正しいプログラムだと誤認して起動するとファイルの破壊、情報の窃盗、あるいはバックドア（悪意を持った攻撃者がそのコンピュータシステムを乗っ取るための隠れた入り口）の設定などの行為を行う。特に、最近では、バックドアを設定したコンピュータに対して攻撃者が外部から指示を送り、その指示に従って他のコンピュータを攻撃するなどの動作を行わせるケースが増えており、こうした動作を行うマルウェアはボットと呼ばれている。また、こうしたボットをインターネット内で1万台レベルで用意し、それらをボットネットと呼ばれる組織化されたネットワークにして、一斉に組織的な攻撃を仕掛けるような世界的なスケールでの攻撃も出てきており、社会的にきわめて大きな問題となっている[16]。

こうしたマルウェアに対する対策として、これまでさまざまな方式が考案され、実用に供せられるようになってきた。既にマルウェアとして発見されている既知のマルウェアについては、そのプログラムの特徴や動作があらかじめ分かっているため、コンピュータシステムに感染する前に検出して駆除することが原理的には可能であるはずである。しかしながら、ネットワークのブロードバンド化の進展とも相まって、最近のマルウェアは感染していくスピードが極めて速く、2003年にATMネットワークの障害や航空機予約システムの停止など全世界的に甚大な被害をもたらしたワームであるSlammerワームのケースでは、ワームの発生から10分以内に脆弱性を有するホストの90%以上が感染したとの報告がなされている[17][18]。これだけ感染のスピードが速い

と、マルウェアの発生が報告されて、それに対する検出用の定義ファイルまたは対処用のアップデートファイルを作成し、その後ネットワーク内のホストにて個別に検出／対処を行うのでは被害の拡大を抑えられないという問題が顕在化してきている[19]。

さらには、セキュリティホールが公開されると、当日のうちにその脆弱性を悪用したマルウェアが出現するという、いわゆる「ゼロデイアタック」といった攻撃も出現するようになり、マルウェアの特徴を抽出して対処策を作成する時間的な余裕が無いという問題も発生している[20]。実際、前述の Slammer ワームの場合には、脆弱性情報が発表されてからワームの出現まで 6 ヶ月であったものが、Nimda ワームの場合には 4 ヵ月後、Slapper は 6 週間後、さらに MSBlaster は 3 週間後と、脆弱性情報発表からマルウェア出現までの期間がどんどん短くなってきている。また、最近の傾向としてマルウェアが出現するとほとんど間髪を入れずに、そのプログラムを一部変更した「亜種」と呼ばれるマルウェアが多数発生してくるようになってきており、たとえ既知のマルウェアであってもその対策は容易ではないという状況になっている。もちろん、まだ見つかっていない未知のマルウェアに対しては、その特徴も分からないために、感染前にマルウェアを完全に検出して駆除することは事実上不可能であると言わざるを得ない。

以上の状況から、たとえマルウェアのコンピュータシステムへの感染を事前に 100% 防げなくとも、感染後にそのマルウェアが悪意を持った動作を起こす時点でその動作を検知し、他システムへのさらなる伝染やコンピュータシステム内の秘密情報の窃盗といった実被害の発生前にその動作を防止することが強く求められている。

そこで、本論文では、未知ならびに亜種のマルウェアを対象とし、たとえ感染は許してしまったとしても、そのマルウェアが悪意を持った動作を開始した時点で、その動作をマルウェアらしい振る舞いとして検知して、コンピュータシステムへの実際の被害が発生する前にそれを防止する方式の研究を行う。

マルウェアは、大きく API ベースのマルウェアとカーネルベース（または Rootkit 型とも呼ばれる）のマルウェアに大別される[89]。前者は、OS のユーザモードで動作するタイプのマルウェアであり、OS が提供する標準的な API を利用して悪意を持った動作を行う。後者は、OS のカーネルモードで動作するタイプのマルウェアであり、デバイスドライバやフィルタドライバとして OS に組み込まれた状態で悪意を持った動作を行う。

カーネルベースのマルウェアは、これまではトロイの木馬として正しいプログラムを偽装した形で送り込まれるケースがほとんどであったが、最近では、まだその数は極めて少ないがウィルスやワームも出現するようになってきている。しかしながら、カーネルベースのマルウェアの場合には、利用者が意識的にデバイスドライバやフィルタドラ

イバをインストールすることにより感染するものであり、利用者の意識が十分に高く、かつ、正しい動作の保証されたデバイスドライバやフィルタドライバしかインストールを許さないという厳格な運用[116]がなされていれば防止できる。そこで、本論文では、通常のユーザプログラムとして利用者の気づかないうちにインストールされてしまう、API ベースのマルウェアの検知方式について検討する。

1.3 構成

本論文では、1.2 節で述べた 3 種類のマルウェアの検知方式として 3 つの方式を提案する。

第 2 章では、ウィルスとワームに対する対策技術として、マルウェアによる自己複製行為をマルウェアらしい振る舞いとして規定することによりマルウェア検知を行う方式として「自己ファイル READ の検出によるマルウェア検知方式」を提案し、その有効性を検証する。

第 3 章では、コンピュータシステムとしての安全性、効率性をより高めるため、第 2 章で提案した方式をシステム運用的に補完する方式として、コンピュータシステム全体を外部からも監視できる「送受信データ間の相関に基づく蔓延防止方式」を提案し、その有効性を検証する。

第 4 章では、トロイの木馬に対する対策技術として、近年その被害の大きさが問題視されていながら対策技術の進んでいないキーロガーを対象とし、キー入力情報の取得行為をマルウェアらしい振る舞いとして規定することによりマルウェア検知・被害発生防止を行う方式として、「動的 API 検査方式によるキーロガー検知方式」を提案し、その有効性を検証する。

第 5 章では、全体のまとめとして、未知および亜種のマルウェアに対する対策技術としての上記提案方式の有効範囲について考察するとともに、残された課題について論じる。

第2章
自己ファイルREADの検出によるマルウェア検知方式

2.1 従来技術における課題

これまでのマルウェア対策技術と呼ばれるものには大きく分類して以下の 3 種類の方式がある[21][22][23]。

① パターンマッチング法
② ヒューリスティックスキャン法
③ ビヘイビアブロッキング法

パターンマッチング法とは、マルウェアプログラムの特徴的なコード部分を抽出して「定義ファイル」として定義しておき、検査対象プログラムに、定義ファイルにあるコード部分と一致する箇所が存在するかどうかによってマルウェアを検出するという方式である。パターンマッチング法は、コード部分の単純なパターンマッチングだけで実現でき、比較的簡単に検査が行えるため、現在の多くの商用アンチウィルスソフトウェアで採用されている方式であるが、当然ながら、特徴的な部分がまだ分かっていない未知のマルウェアに対してはこの方式では検知ができない。また、たとえ既知のマルウェアであっても、前述したゼロデイアタックのような攻撃に対しては定義ファイルの更新が間に合わないという問題や、マルウェアの亜種に対してはその定義ファイルが機能しない場合が多いという問題がある。

さらに、最近では、ポリモーフィック型ワームやメタモーフィック型ワームと呼ばれる、コンピュータに感染のつど自分自身のプログラムを変異させていくミューテーション型（突然変異型）のワームが増えてきており、こうしたマルウェアに対してはたとえ既知のものであってもパターンマッチング法により検知することは困難である。

ポリモーフィック型ワームとは、図5に示すように、感染のたびに異なる暗号化／復号ルーチンを生成して、自分自身のプログラムをランダムに暗号化するワームのことを指す[21]。ポリモーフィック型ワームが起動されると、まず復号ルーチンが実行されて、暗号化されているワーム本体の復号を行う。その後、復号されたワーム本体に制御が移り、ワームの動作を開始する。暗号化／復号に使用する鍵は、ランダムに生成されるた

め、暗号化されたワーム本体を、パターンマッチング法により一意に検知することは不可能である[23]。

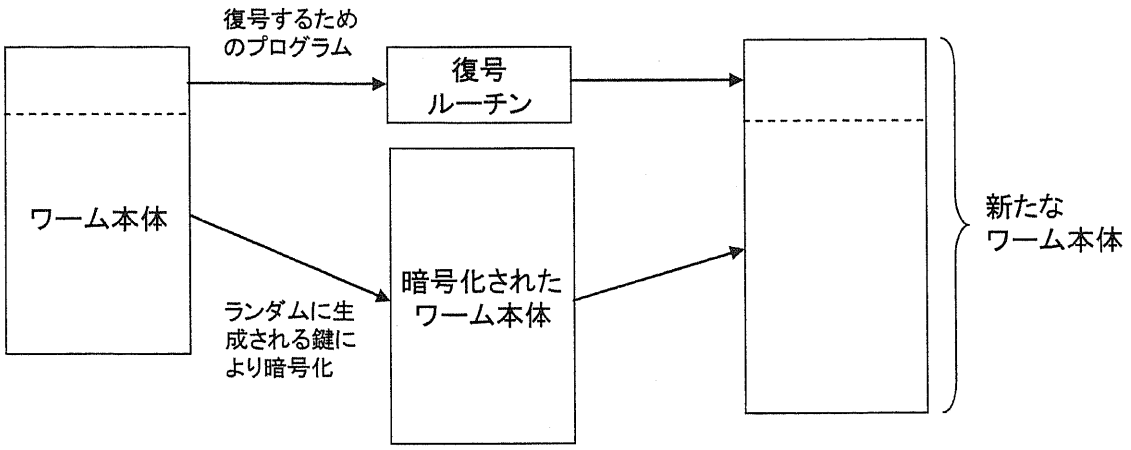

図5　ポリモーフィック型ワームの構造例

Fig. 5　Structure of polymorphic warm

メタモーフィック型ワームとは、図6に示すように、感染のたびに自身のプログラムの順番を入れ替えたり、同じ動作を行う異なる命令パターンを利用して自分自身のプログラムを書き換えたりして自分自身を難読化するワームのことを指す[21]。メタモーフィック型ワームの場合には、新たに生成されるワーム本体は、元のワームとは異なるプログラムの形をしているため、パターンマッチング法により検出することは困難である。

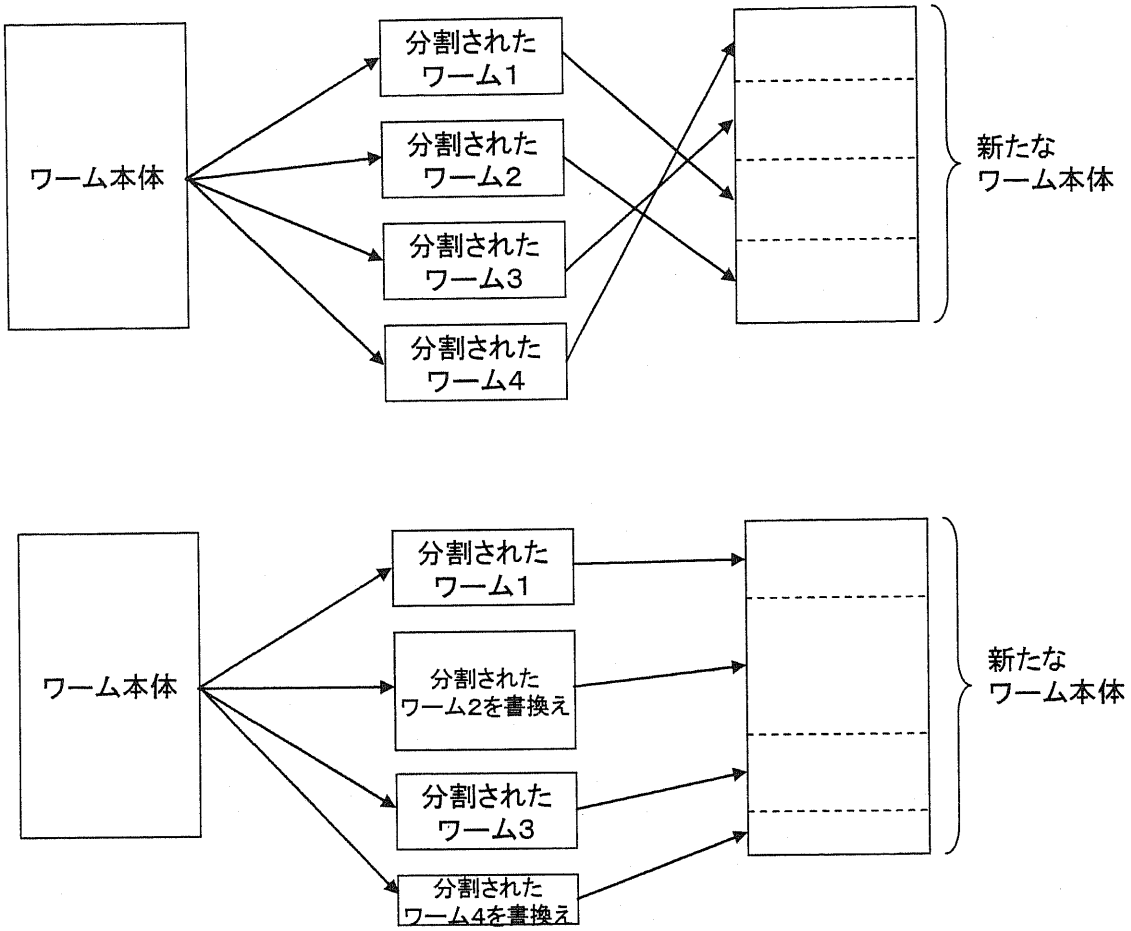

図6　メタモーフィック型ワームの構造例

Fig. 6　Structure of metamorphic warm

ヒューリスティックスキャン法には、さらに静的ヒューリスティックスキャン法と動的ヒューリスティックスキャン法の2種類があり、既知のマルウェアだけでなく未知のマルウェアも検知できるという特長を持つ。

静的ヒューリスティックスキャン法[24]は、検査対象となるプログラムを実行する前に、マシン語レベルのコード解析を行い、システムファイルへの書き込みやレジストリへの登録といったマルウェアらしい振る舞いを行うコードが含まれていないかを検査する方法である[25][26]。プログラムの実行前に検査するので、マルウェアがコンピュータシステム内に送り込まれて感染している状態でも実被害に至る前にマルウェアの検出をすることが可能であるが、前述のメタモーフィック型ワームのように、1つの不正動作を行わせる際にもプログラムの書き方は多種多様であるので、コードを完全に解析することは容易ではない。また、ポリモーフィック型ワームの場合には、コード部分が暗号化されているため、マルウェアを実行させないことには復号ができず、そのまま

ではコード内容の解析が行えないという問題がある。さらに、最近では、コンピュータシステムに入り込んだメモリ常駐のマルウェアが、自己のプログラムファイルへのアクセスを監視して、アクセスが行われると、当該プログラムからマルウェアらしい振る舞いを行うコード部分を削除して無害なプログラムのように見せかける巧妙なマルウェアも出現している。このようなマルウェアはステルス型とも呼ばれている。こうしたマルウェアは静的ヒューリスティックスキャン法やパターンマッチング法などのコード分析による方法では検知できない。

動的ヒューリスティックスキャン法[27]は、仮想マシン（VM）などの仮想環境で検査対象となるプログラムを実行させ、プログラムがマルウェアらしい振る舞いを行うかどうかを検知する方法である[28][29][30]。感染している場合には実際にマルウェアを実行して、マルウェアらしい振る舞い（レジストリ改ざん、システムファイル変更、感染活動など）が発生したことにより検知を行うため、基本的には、ミューテーション型を含むすべての既知のマルウェアと未知マルウェアを検知することが可能であるが、後述するビヘイビアブロッキング法と同様、マルウェアらしい振る舞いの特定が難しく、正常なプログラムをマルウェアだと検知してしまうという誤検知が多発する傾向にある。また、最近では、マルウェア自身が仮想環境下で走行しているのか実環境下で走行しているのかどうかを識別して、自身が仮想環境下で走行している場合には発病しないという巧妙なマルウェアも出てきており、仮想環境下では検知できず、実環境で動作させたときに発病するといったマルウェアも出現している。さらに、エンドユーザのコンピュータ上で検出する場合には、仮想マシンをエミュレートするためのモニタであるVMware[31]などの仮想環境を常駐的に稼動させるため、モニタによるオーバヘッドが発生してCPU負荷が大きくなりすぎるといった問題も存在する。

ビヘイビアブロッキング法[20]は、実環境下において検査対象となるプログラムが発行するシステムコールなどを検査することにより、当該コンピュータ上で動作しているプログラムの動きを監視し、レジストリ改ざん、システムファイル変更、感染活動などマルウェアらしい振る舞いをした場合に、そのプログラムをマルウェアとして検知する方法である。実際にプログラムを実環境下で動作させて監視するという点から3種類の方法の中では最も検知精度が高く有効な方法である[23][32][33]が、「マルウェアらしい振る舞い」というものを確実に規定することが難しいという問題があり、マルウェアと類似した動作を行う正常なプログラムを誤検知してしまう可能性が高いという誤検知の問題がある。例えば、プログラムをコンピュータシステムにインストールするためのプログラムであるインストーラの場合には、新たにインストールしたプログラムをシステムに登録するためにレジストリの変更動作を行うが、このような動作をマルウェアによる動作と誤認識してしまう可能性がある。

さらに、これら3種類の方法を組み合わせて検出するという方法もいくつか提案されているが[34][35]、OR条件で検知しようとするとFalse Positiveとなり誤検知の発生が多くなり、逆にAND条件で検知しようとするとFalse Negativeとなって検知漏れが発生しやすくなるという問題があり、各々をどのような比率で組み合わせて検知すれば最適となるかを規定することが難しいという問題が残る。

いずれの方法においても、「真にマルウェアらしい振る舞い」を誤検知なく正しく規定できるかどうかが方式の有効性の鍵となる。マルウェアらしい振る舞いとしては、これまで以下のような動作を規定する方法が提案されている。

① OS起動時の自動実行[28][36][37]

「マルウェアは、できるだけ長い期間、クライアントに感染し続けようとするため、OS 再起動後にも自身が自動実行されるようにする」という動作をマルウェアらしい振る舞いとして規定する。具体的には、OS の自動実行に関わるレジストリや、スタートアップファイルへの書き込みを行う動作を検知する。しかしながら、前述したように、インストーラはインストールした常駐型プログラムをOS 起動時に自動実行するためにレジストリの変更を行うので、インストーラをマルウェアとして誤検知する。また、CodeRed[38]のように、再起動がほとんど行われないサーバへの感染を目的として作成されたメモリ常駐型のマルウェアについては検知できない[11]。

② 起動直後のファイルコピー[39]

「マルウェアは、自身のプログラムをいつでも再実行できるように、OSのシステムフォルダ以下に自身のコピーの書き込みを行う」という動作をマルウェアらしい振る舞いとして規定する。具体的には、システムフォルダ以下のファイルへの書き込み動作の有無を監視する。しかしながら、インストーラはプログラムをインストールする際に新規DLLなどをシステムフォルダ以下にコピーするので、前項と同様、インストーラをマルウェアとして誤検知してしまうという問題がある。

③ 別プロセスの起動[40]

「マルウェアは、自身とは別のプロセスを起動してそのプロセスに悪意のある動作をさせることによって自身の行動を隠そうとする」という動作をマルウェアらしい振る舞いとして規定する。具体的には、利用者が起動していないプロセスがバックグラウンドで起動されたかどうかを監視する。しかしながら、クライア

ントブラウザやアンチウィルスソフトのインストーラは自身とは別のプロセスを起動する場合があるため、これらをマルウェアであると誤検知してしまう。一方、マルウェアの中には別プロセスを起動せずに自プロセスの中で悪意を持った動作を行うものも存在するため、このようなマルウェアは検知できない。

④ トラフィック量の異常変動[41][42][43][44] [45][46][47][48][49]

「マルウェアはネットワークを介して他の多数のコンピュータへの伝染行為を行う」ということをマルウェアらしい振る舞いとして規定する。具体的には、ネットワークを流れるトラフィックを監視して、トラフィック量の急激な上昇や通常とは異なるパケットの送信などをマルウェアによる伝染行為として検知する。この方法は、感染したコンピュータを外部から監視することができるため、これまで数多くの方式が提案されている。しかしながら、FTP などを用いた正常なファイル転送の場合にもネットワークトラフィックは上昇するため、どのくらいの期間にどのくらいトラフィック量が増加したら異常とみなすのかの閾値の設定が一般的には難しいという問題がある。閾値を低く設定してしまうと誤検知が、また閾値を高く設定してしまうと検知漏れが発生する可能性が生じる。さらに、最近のマルウェアの中には、一度に自分自身のファイルを伝染させるのではなく、外部への送信を時間的に分散させて行うことで、ネットワークの急激な負荷上昇を抑え通常のネットワーク送信状態を装うものもあり、そのようなマルウェアはこの方法では検知できない。

⑤ CPU 利用率の上昇[50]

「マルウェアは侵入、発病、感染などの動作をなるべく速やかに行おうとする」という動作をマルウェアらしい振る舞いとして規定する。具体的には、システムの CPU 使用率を監視し、特定のプロセスによる CPU 使用率の上昇が見られた場合にはマルウェアとして検知する。しかしながら、一般的には大規模シミュレーションプログラムのように膨大な計算処理を伴うアプリケーションの場合には、CPU が寡占されることとなり、このようなプログラムをマルウェアとして誤検知する可能性がある。従って、トラフィック量の監視を行う方法と同様、CPU 使用率がどの程度の期間にどの程度上昇したらマルウェアと判断するかの閾値を設定することが難しいという問題がある。

⑥ バッファオーバフロー攻撃[51]

一部のマルウェアに見られる、「マルウェアはバッファオーバフローを用いて不正動作を行う」という動作をマルウェアらしい振る舞いとして規定する。具体的には、関数呼び出しが起こる際に、その呼び出しをすべてフックして、呼び出し

側のパラメータサイズと呼び出される側のバッファサイズを動的に調べ、バッファオーバフローが生じるかどうかをチェックする。バッファオーバフローを利用した攻撃とは、図7に示すように、関数呼び出し時にバッファサイズを超えたパラメータを送ることにより、バッファを意図的にあふれさせ、バッファ領域の後ろにあるデータを破壊してプログラムが正しく動作しないようにしてしまう攻撃のことである。また、通常バッファ領域の後ろには、関数呼出し後のリターンアドレスが設定されていることが多く、そのアドレスをあふれたデータで上書きすることで、プログラム実行番地を攻撃者の設定した番地に強制的に飛ばして、攻撃者の思い通りの動作をさせるような攻撃も数多く存在する[52]。

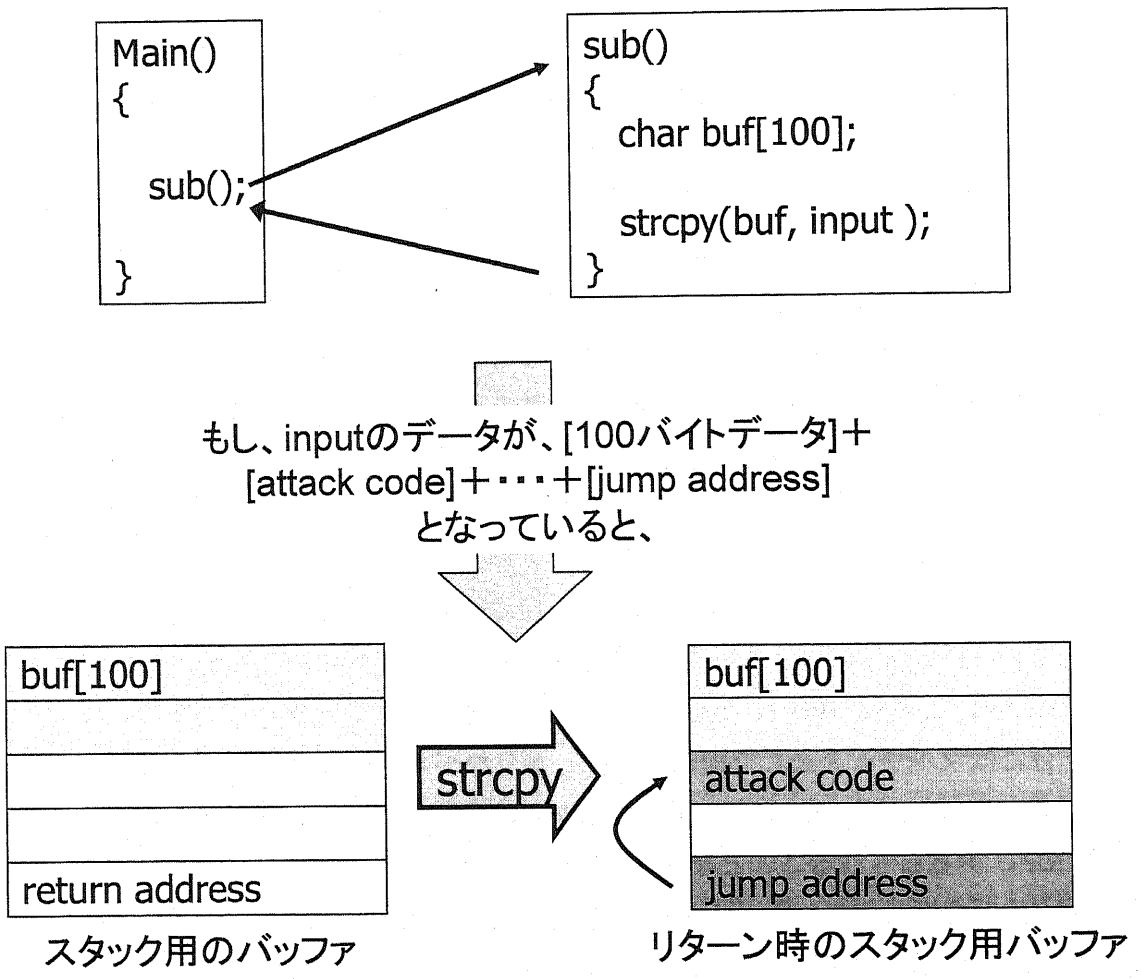

図7　バッファオーバフロー攻撃

Fig. 7　Buffer overflow attack

バッファオーバフロー攻撃を防ぐためには、文献[51]や[52]に示されているよう

に、呼び出された関数に対して常にパラメータのサイズを調べておかなければならない。しかし、こうしたバッファオーバフローに対する対策が行われていることは少なく、バッファオーバフローの脆弱性は世の中に存在する全てのセキュリティホールのうち約半数を占めると言われている。なお、文献[51]に示されている方法はバッファオーバフロー攻撃のような特定の脆弱性を狙った攻撃に対しては有効であるが、その他の攻撃に対しては検知することができない。

⑦ 脆弱性が発見されたポートへの通信[53]

「マルウェアは脆弱性が発見されたポートから侵入を試みる」という動作をマルウェアらしい振る舞いとして規定する。具体的には、セキュリティホールが発見された特定のポートを定常的に監視し、当該ポートへ外部からアクセスがあった場合には、さらにアクセスを行ったプロセスの挙動を監視して、特定ディレクトリにファイルコピーが行われたかどうかで、当該セキュリティホールを狙った未知マルウェアを検出する。これは、セキュリティホールが公表されてから定義ファイルが配布されるまでの間を防御するための対策となる。しかしながら、この方法では、例えばメールへの添付ファイルにより感染するワームなどの、特定ポートへの侵入という手段によらないマルウェアには対処できない。また、特定ディレクトリへのファイルコピーを監視するだけでは正常なプログラムと見分けることは困難であるという問題がある。更に、ディレクトリへのアクセスの監視だけでは、ディレクトリへの書き込みを伴わないメモリ常駐型のマルウェアに対しては検知できない。

以上述べてきたように、これまで提案されてきた方法では、ミューテーション型の既知マルウェアや未知マルウェアを含め、誤検知を生じさせずに一意にマルウェアを検知することができないという問題がある。

2.2 真にマルウェアらしい振る舞いの規定

マルウェアの検知方式として最も有効な方式であるビヘイビアブロッキング法で未知マルウェアやミューテーション型マルウェアの検知を行う場合、真にマルウェアらしい振る舞いを一意に規定することが方式の有効性の鍵となることは既に述べたとおりである。そこで、本節ではウィルスとワームに関して、真にマルウェアらしい振る舞いの定式化を行っていく。

従来技術で提案されている各種の振る舞いの規定を改めて整理してみると、マルウェアらしい振る舞いは、以下の3つの振る舞いのいずれかに集約できる。

① マルウェアは悪意を持った動作を行うために、マルウェア自身をいつでも実行可能な形にしておく。OSの起動直後、あるいは任意のタイミングで自身が実行できるようにするためには、実行可能なプログラムの形で一旦ファイルに保存し、さらにプログラムとしてシステムから起動可能とするためにシステムのレジストリファイル等にその実行形式ファイルの場所を登録しておくという動作を行う。ここでは、実行中のプロセスが自分自身のプログラムを保存・登録しておくということであり、インストーラが自分自身のファイルの中に格納されているインストールすべきプログラムを個別の実行形式ファイルとして保存・登録する動作とは異なることに注意されたい。

【挙動1：プログラムの実行に関する振る舞い】

自分自身のプログラムをファイルとして保存し、プログラムとして起動可能な形にしておく

② マルウェアは自身の影響範囲を拡大するために、ネットワークを経由して他のコンピュータに侵入しようと試みる。このため、自分自身の複製を作成して、メール送信やファイル転送などの何らかの通信手段を用いて他のコンピュータに複製を送り込む動作を行う。

【挙動2：感染行為に関する振る舞い】

自身の複製を他のコンピュータに送信するという感染行為を行う

③ マルウェアは、駆除されないように、自身がコンピュータシステム内に存在していることをなるべく利用者には気づかせないようにする。このため、自分自身がマルウェアとして容易に見つからないようにプログラムの形を変える、あるいは悪意を持った動作を行うにあたって自分自身のプロセスではなく、他のプロセスに依頼して自身の行為を隠すという動作を行う。

【挙動3：隠蔽行為に関する振る舞い】

自分自身の行為やプログラムをシステムや利用者から隠蔽する

さらに、挙動3の行為は、以下の2つの振る舞いのいずれかに分けられる。

【挙動 3-1】自分自身のプログラムを検出されないように変異させる
【挙動 3-2】他のプロセスに、挙動 1～3 の行為を依頼する

挙動 3-2 は、マルウェアがメーラなどの既存のプロセスあるいは自ら新たに生成したプロセスに、挙動 1、挙動 2、あるいは挙動 3-1 を行わせるという動作の再帰動作である。従って、これらの挙動に基づくウィルスとワームの検知は図 8 のフローチャートのように定式化することができる。

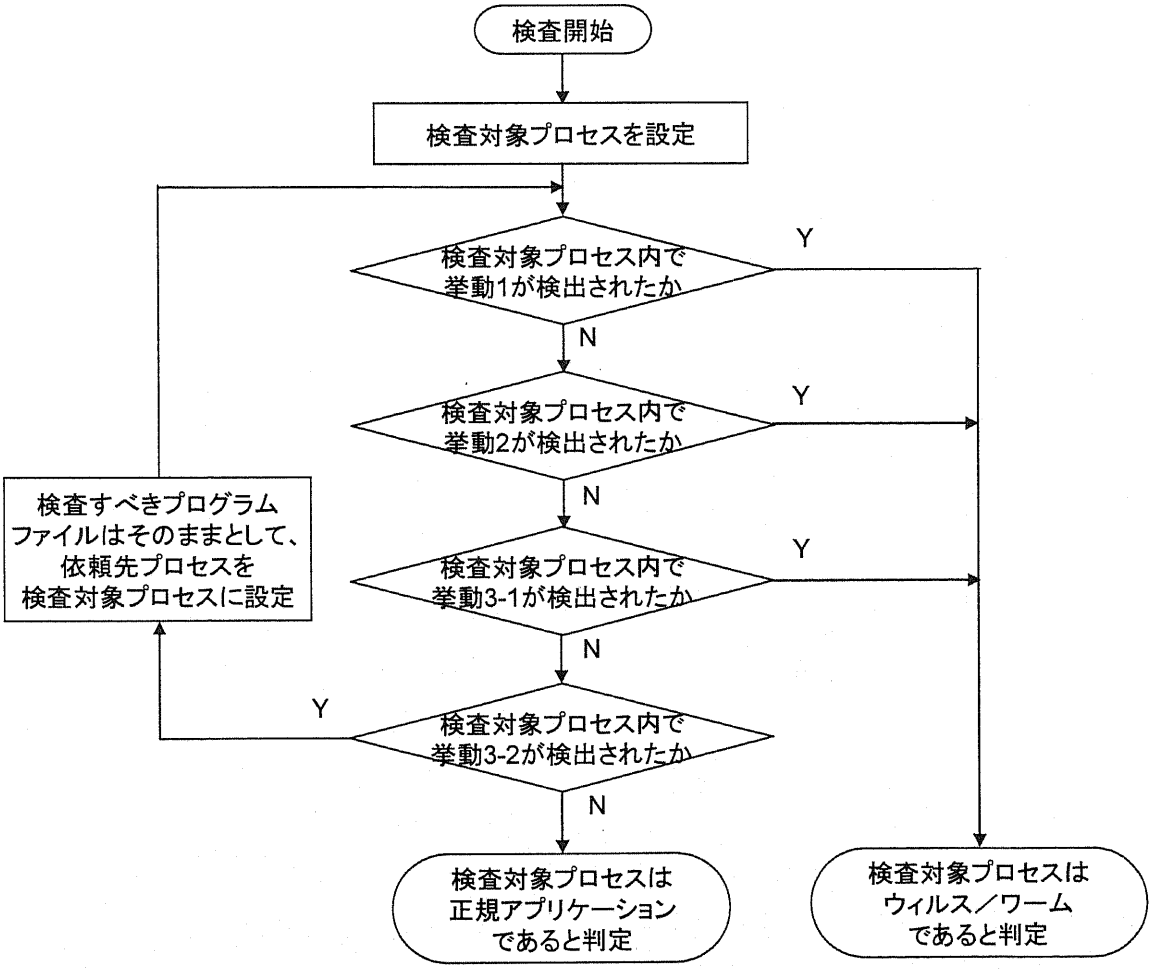

図 8　ウィルスとワームの検知フロー

Fig. 8　Process flow of virus and warm detection

一方、挙動 1、挙動 2、挙動 3-1 の各動作は、プロセスを制御している OS の観点から見てみると、以下の一連の動作に分解できる。

【挙動 1】

① マルウェアのプログラムをデータとしてファイルから読み込む（READ 動作）

② 読み込んだデータをいずれかのフォルダにファイルとして書き込む

③ 保存したファイルをシステムのレジストリ等に登録する

【挙動 2】

① マルウェアのプログラムをデータとしてファイルから読み込む（READ 動作）

② 読み込んだデータを通信用 API のパラメータに設定する

- ウィルスの場合には、SMTP などのメール送信用の API に対し、読み込んだデータを送信メールの添付ファイルとして設定する
- ワームの場合には、FTP などのファイル転送用の API に対し、読み込んだデータを転送ファイルとして設定する

③ 通信用の API をコールすることによりネットワークを介して他コンピュータにウィルスやワームのプログラムを転送する

- ウィルスの場合には、ウィルス付きメールとして送信される
- ワームの場合には、他コンピュータに存在する脆弱性を突いて、ファイル転送などの形で送り込む

【挙動 3-1】

① マルウェアのプログラムをデータとしてファイルから読み込む（READ 動作）

② 読み込んだデータを変形する

- ポリモーフィック型ワームの場合には、読み込んだデータをランダムに生成する暗号化ルーチンを用いて暗号化する
- メタモーフィック型ワームの場合には、読み込んだデータをある単位で分割しその順番を入れ替えたり、他の命令コードに置き換えたりする

このように、いずれの挙動においても、「メモリ上で実行しているマルウェアのプロセスが、ファイルシステム上にある自分自身の実行ファイル（マルウェア本体のファイル）を読み込む」という READ 動作が発生している。以降、本論文では、実行中のマルウェアが自分自身のプログラムが格納されているファイルを読み込むという動作を「自己ファイル READ」と呼ぶことにする。挙動 1、挙動 2、挙動 3-1 という真にマルウェアらしい振る舞いにおいて、必ず自己ファイル READ が発生しているため、自己ファイル READ の有無を検出することができればミューテーション型や未知のものも含めてマルウェアの検知が可能となる。

ネットワークを介して宿主や自らの力で増殖を行うウィルスやワームのようなマルウェアに対する既存の未知マルウェア検知方式においては、マルウェアの自己複製をマルウェアらしい振る舞いとして利用している例は、調べた限りでは存在していない。そ

の理由は以下のように考えられる。

- ファイルコピーは利用者のコンピュータにおいて日常的に行われる操作であるため、マルウェアによる自己複製と正常のファイルコピーを見分けることが難しい。
- ファイル単位でその複製を検知する方法は、ファイルに寄生するタイプの（狭義の）ウィルスに対しては無力である。

これに対し、自己ファイル READ の検査では、単なるファイルのコピーではなく、自己のプログラムが格納されているファイルを読み込むという限定された動作に着目しているという点において、単なるファイルコピーの検査で発生する正常ファイルコピーの誤検知の問題を回避することを狙いとする。また、狭義のウィルスに対しても、添付ファイルとして作成する際には、一旦自己ファイル READ を行ってからメール送信文に添付（寄生）するという動作となるため、必ず自己ファイル READ を伴うこととなる。

一方、正常なプログラムで自己ファイル READ を行うものが存在する可能性も考えられる。例えば、従来のビヘイビアブロッキング方式において誤検知されるケースの多かったインストーラの場合には、インストーラ自身のファイルの中にインストールすべきプログラムが格納されているため、これを読み込むという自己ファイル READ 動作が発生する。しかしながら、3 つのマルウェアらしい振る舞いの①で述べたように、インストーラが読み込むのはインストールすべきプログラムが格納されている領域の部分だけであり、インストーラ自身のプログラムが格納されている領域を読み込む必要はないはずである。従って、自身のプログラム領域の全てを読み込むことを自己ファイル READ とすれば、インストーラが誤検知されることはないと期待できる。

これまで述べてきたように、自己ファイル READ の有無を検出すれば、従来は検知が困難であったミューテーション型や未知のもの含めたあらゆるウィルスやワームの振る舞いが特定できると考えられる。そこで、次節では、自己ファイル READ の検出による新たなマルウェア検知方式を提案し、その有効性を検証することとする。

2.3 提案方式

提案方式では、マルウェアによる「自己ファイル READ」をビヘイビアブロッキング法におけるマルウェアらしい振る舞いとして利用する。具体的には、OS のファイルシステムをフックすることにより、エンドユーザのコンピュータにおける全プロセスのファイルアクセスを常時監視し、自己ファイル READ を行ったプロセスをリアルタイムで検知する。よって、マルウェアが活動を開始した瞬間にこれを発見することが可能となる。自己ファイル READ の検出は、以下の両方の検査項目の検出で行う。

- 実行しているプロセスの実行ファイルが存在するパスと、そのプロセスが READ しようとしているファイルのパスが一致する
- 実行しているプロセスの実行ファイルの内容と、そのプロセスが READ しようとしているファイルの内容が一致する

通常、実行プログラムはその一部分が欠落するだけで正常に動作をしなくなるため、マルウェアが自己ファイル READ を行うにあたっては基本的に自分自身のプログラムをそっくりコピーすることになる。そこで本方式では、自分自身の本体のファイルのヘッダー領域を除くプログラム（以下、プログラム領域と呼ぶ）のすべてを READ したプロセスが検出された時点でアラートを上げることとする。

ファイルシステムのフックを行うにあたっては、フィルタドライバを改造することで比較的容易に実装が可能である。また、ファイルシステムのフックは、動的ヒューリスティックスキャン法[54]のように仮想環境を稼動させるような方法と比べて十分低負荷であるため、エンドユーザのコンピュータの性能を鑑みてもリアルタイム検知が十分可能であると想定される。

提案方式を組み込んだフィルタドライバの処理の流れを図 9 に示す。

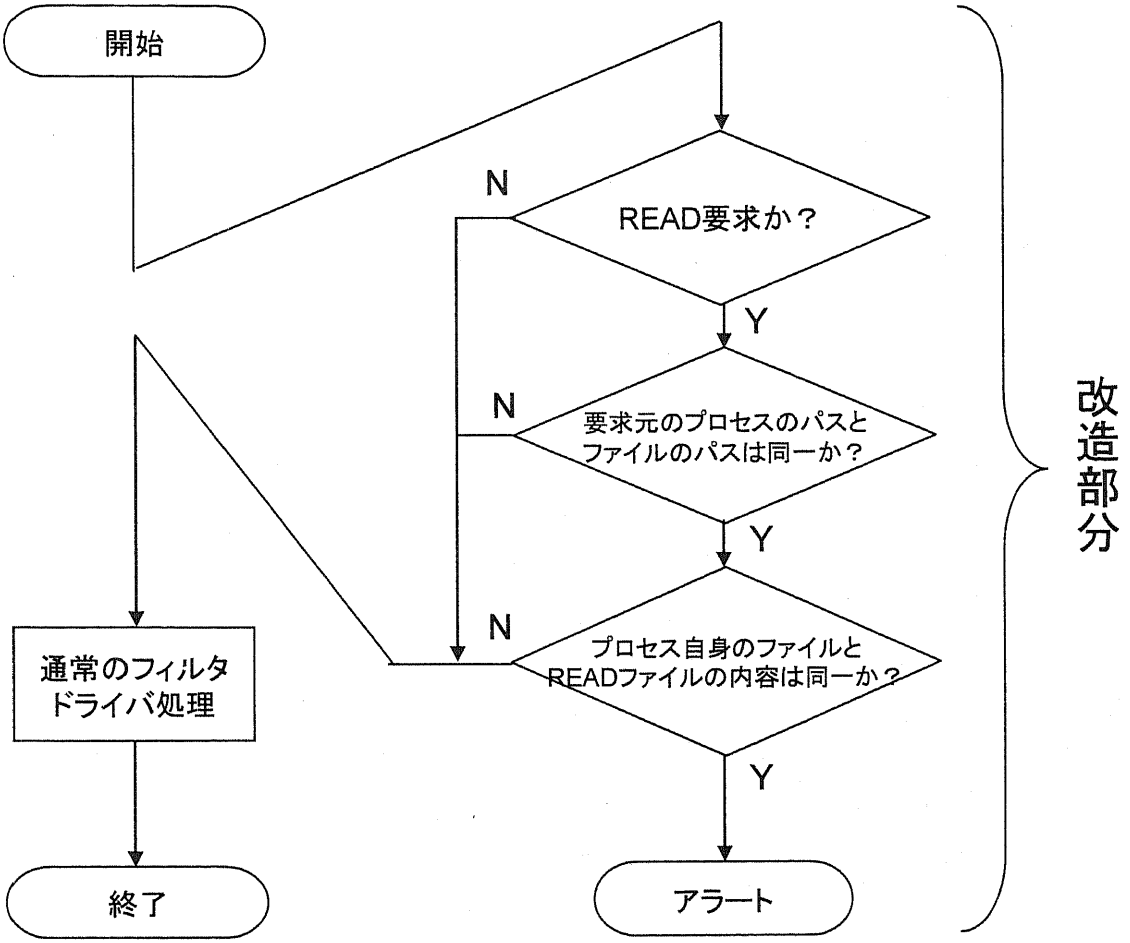

図 9　提案方式を組み込んだフィルタドライバの処理フロー

Fig. 9　Process flow of filter driver attached with malware detection mechanism

2. 3. 1　検証実験

(1) 検証の要件

ビヘイビアブロッキング法に基づく未知マルウェアの検知方式の要件としては以下のものが挙げられる。

【機能要件】

① 「ミューテーション型マルウェアを含む、未知マルウェアの検知が可能であること」

未知マルウェアならびにミューテーション型のマルウェアは従来方式では検知が困難であった。こうしたマルウェアに対する検知精度が従来方式に比べて十

分に高い方式である必要がある。

② 「正規プロセスを誤検知しないこと」

未知マルウェアに対して最も有効なビヘイビアブロッキング法には、誤検知が多いという問題があった。こうした誤検知の発生が従来方式に比べて十分に低い方式であることが求められる。

【実装要件】

① 検知速度が速い（リアルタイム検知ができる）こと

② オーバヘッドが少ないこと

本論文では、基本的な方式の提案と基礎実験による本方式の有効性の報告に注力することとして、上記要件の内、機能要件に焦点を当てて検証実験を行う。すなわち、本方式でミューテーション型マルウェアや未知マルウェアの検知が方式的に可能であること、さらには正規のプロセスを誤検知しないことを示すことに重点を置く。また、同じ理由で、今回は再帰動作となる挙動 3-2 の検出に関しては、実装を割愛する。

なお、上記以外の実装上の要件としては、既存のアプリケーションには手を加えずに、OS レベルで実現する方式とする点が上げられる。宿主を必要とするウィルスの場合には、宿主となるアプリケーション側で対処を行うことも考えられるが、極めて多種多数の既存アプリケーションのすべてに対処を行うことは非現実的であることから、既存のアプリケーションには手を加えずに検知できる方式が求められる。

(2) 検証方法

前項に示した要件のうちの機能要件のみの検証であれば、フィルタドライバを改造して実際に OS システムを実装する必要はないため、ファイルシステムのフックの代わりに OS（Windows）のファイルアクセスをリアルタイムで監視可能なモニタツールである FileMon[55]を用いて自己ファイル READ の検出を行うことにより、本方式の検証を行った。具体的には、FileMon によりコンピュータ内で発生したすべてのファイルアクセスを記録し、プロセスが自分自身の本体のファイルのプログラム領域のすべてを READ するかどうかをチェックする。すなわち、パス名 A のファイルを実行することによって生起されたプロセスが、パス名 A のファイルのプログラム領域のすべてを READ した場合に、マルウェアの疑いありと判断することとする。

自己ファイル READ の検知方法としては、OS のファイル READ の仕組みを鑑み、以下の 2 種類の方法で自己ファイル READ を行っているかどうかを検査することとした。

- シーケンシャル READ

  ファイルの中身全体をシーケンシャルに READ して自己のプログラム領域全体を READ しているかどうかを検査する。

- ブロック READ
  自分自身のファイルをブロック（例えば 1024 バイト）ごとに次々と READ して自己のプログラム領域全体を READ しているかどうかを検査する。

検知実験では、実際には未知マルウェアの入手が困難であるため、ここでは代表的な既知のマルウェアのファイルアクセスを見ることで仮想的に未知マルウェア検知が可能であるかを確認することとした。

本実験は、物理的に隔離されたネットワーク上で行った。隔離されたネットワーク上のコンピュータでマルウェアを実行し、FileMon でマルウェアのファイルアクセスの状況を観測する。なお、実験に使用したコンピュータの OS はセキュリティパッチの当たっていない Windows2000 である。

(3) 実験結果

表 1 に、提案方式によるマルウェアの検知実験の結果を示す。○は自己ファイル READ が検出されたことを表し、×は検出されなかったことを表す。検知結果の欄は、シーケンシャル READ、ブロック READ のいずれかが○になっていれば提案方式によってマルウェアが検知されたことになるとした場合の検知結果であり、○は提案方式により検知されたことを表す。

表 1　検査対象マルウェアおよび検知結果

Table 1　Malware detection by the proposed scheme

マルウェア	型	シーケンシャル READ	ブロック READ	検知結果
Sasser.C	ワーム	○	×	○
Blaster.C	ワーム	○	×	○
Beagle.X	ウィルス	○	×	○
Netsky.B	ウィルス	×	○	○
Netsky.D	ウィルス	○	×	○
Netsky.Z	Zip圧縮型ウィルス	×	○	○
Beagle.AG	鍵付きZip圧縮型ウィルス	○	×	○
Mimail.Q	自己変異型ウィルス	○	○	○

以下に、シーケンシャル READ とブロック READ のそれぞれの場合について、マルウェアによる具体的な自己ファイル READ の動作を説明する。

- シーケンシャル READ

  Sasser.C、Blaster.C、Beagle.X、NetSky.D 、Beagle.AG、Mimail.Q においては、自分自身のファイル（マルウェア本体）のファイルサイズなどを調べた上で、ファイルの中身全体をシーケンシャルにコピーするという動作が捕らえられた。「Open Sequential Access」オプション付きのファイル OPEN、ファイル CREATE、ファイル WRITE の処理の中で自分自身をシーケンシャルに READ している。FileMon によるファイルアクセスの観測結果（の一部）を表 2 に示す。今回の実験に用いたマルウェアにおいては、シーケンシャル READ により、自分自身のファイルをシステム管理下のフォルダにコピーする挙動が見られた。

表 2　シーケンシャル READ を行うマルウェア（Sasser.C）の観測結果の一部

Table 2　File access log for sequential-READ obtained with Sasser.C

コマンド	オペランド	結果	オプション
OPEN	C:¥avserve2.exe	SUCCESS	Options: Open Sequencial Access: All
QUERY INFORMATION	C:¥avserve2.exe	SUCCESS	FileAttributeTagInformation
QUERY INFORMATION	C:¥avserve2.exe	SUCCESS	Length: 15872
QUERY INFORMATION	C:¥avserve2.exe	SUCCESS	Attributes: RA
QUERY INFORMATION	C:¥avserve2.exe	SUCCESS	FileStreamInformation
QUERY INFORMATION	C:¥avserve2.exe	SUCCESS	Attoributes: RA
QUERY INFORMATION	C:¥avserve2.exe	SUCCESS	FileEaInformation
CREATE	C:¥WINNT¥avserve2.exe	SUCCESS	Options: OverwriteIF Sequencial Access: Al
SET INFORMATION	C:¥WINNT¥avserve2.exe	SUCCESS	Length: 15872
QUERY INFORMATION	C:¥avserve2.exe	SUCCESS	Length: 15872
WRITE	C:¥WINNT¥avserve2.exe	SUCCESS	Offset: 0 Length: 15872
SET INFORMATION	C:¥WINNT¥avserve2.exe	SUCCESS	FileBasicInformation
CLOSE	C:¥avserve2.exe	SUCCESS	

- ブロック READ

NetSky.B、NetSky.Z、Mimail.Q においては、自分自身のファイル（マルウェア本体）をブロック（例えば NetSky.Z では 1024 バイト）ごとに次々と READ していることがわかった。FileMon によるファイルアクセスの観測結果（の一部）を表 3 に示す。今回の実験に用いたマルウェアにおいては、ブロック READ により、自分自身のファイルをシステム管理下のフォルダにコピーする挙動や、自分自身のファイルを（WINSOCK 等の通信用の API へ引き渡すために）メモリ領域に読み出す挙動が見られた。

表 3　ブロック READ を行うマルウェア（NetSky.Z）の観測結果の一部

Table 3　File access log for block-READ obtained with NetSky.Z

コマンド	オペランド	結果	オプション
READ	C:¥Informations.txt(大量のスペース)*.exe	SUCCESS	Offset: 0 Length: 1024
READ	C:¥Informations.txt(大量のスペース)*.exe	SUCCESS	Offset: 1024 Length: 1024
READ	C:¥Informations.txt(大量のスペース)*.exe	SUCCESS	Offset: 2048 Length: 1024
READ	C:¥Informations.txt(大量のスペース)*.exe	SUCCESS	Offset: 3072 Length: 1024
:	:	:	:
READ	C:¥Informations.txt(大量のスペース)*.exe	SUCCESS	Offset: 20480 Length: 1024
READ	C:¥Informations.txt(大量のスペース)*.exe	SUCCESS	Offset: 21504 Length: 1024
READ	C:¥Informations.txt(大量のスペース)*.exe	SUCCESS	Offset: 22016 Length: 1024

＊ NetSky.Z は、プログラム名に大量のスペースが含まれるが、表では（大量のスペース）と表した。

次に、提案方式によって正規のプログラムがマルウェアとして誤検知されることがないかを代表的なプログラムを用いて調べる実験を行った。表 4 に本実験で用いた正規プログラムと実験結果を示す。×は提案方式において誤検知が発生しなかったことを表す。

表 4　検査対象正規プログラムおよび検知結果

Table 4　False detection of the proposed scheme

正規プログラム	検知結果
MS WORD	×
MS EXCEL	×
Adobe Reader	×
Symantic Client Firewall	×
Internet Explorer	×
インストーラ (sint 1-4-7-0.exe)	×
インストーラ (memcl)	×
インストーラ (Norton AntiVirus)	×

MS WORD、MS EXCEL においては、プログラムを起動させた後、しばらくの間ファイルアクセスをモニタリングしたが、自己ファイル READ は観測されなかった。

インストーラ（sinst1-4-7-0.exe [56]、memcl [57]、Norton AntiVirus）においては、稼動（インストール実行）中に自分自身のファイルを READ するイベントが観測された。しかし、インストーラは、インストールされるプログラム群に関するデータは READ されるが、インストールを制御するプログラム部分については READ されなかった。よって、プログラム領域のすべてが READ されることはなく、提案方式によってマルウェアと検知されることはなかった。

Internet Explorer、Adobe Reader、SymantecClientFirewall においても、プロセスを起動させた直後に、自分自身のファイルの一部を READ するイベントが観測された。Internet Explorer などのソースファイルが公開されていないため推測の域を出ないが、これは、自身のコードの中に書き込まれているバージョン情報などを読み込んでいるのではないかと思われる。しかし、いずれのプログラムにおいても、自己ファイルを READ する対象がファイルのプログラム領域の数%にしか満たないもの、あるいは、ファイルのヘッダー領域の一部のみを READ しているもののみであったため、本方式で誤検知

されることはなかった。

本方式の目的は、ビヘイビアブロッキング法における検知漏れあるいは誤検知の発生頻度を減らし、その検知精度を高めるというものである。そこで、本方式と既存のビヘイビアブロッキング法について、検知漏れと誤検知の発生状況を比較した。評価に際し、用意したアプリケーションは、表4に示した正規プログラムの誤検知実験で使用したアプリケーション以外にもいくつかのアプリケーションを加え、常駐型アプリケーションのインストーラ（memcl)、メーラ（Edmax)、FTP クライアント（Smart FTP)、ブラウザ（Internet Explorer)、Ethereal、アンチウィルスソフト（Norton AntiVirus）のインストーラで評価を行った。また、マルウェアについては、表1に示した検知実験で使用したマルウェアを用いた。

本実験では、既存のビヘイビアブロッキング法で規定されている「マルウェアらしい振る舞い」として、2.1節で挙げた挙動のうち代表的な以下の5つを採り上げた。

- タイプ1：OS 起動時の自動実行
- タイプ2：起動直後のファイルコピー
- タイプ3：別プロセスの起動
- タイプ4：トラヒック量の異常変動
- タイプ5：CPU 使用率の上昇

表5に比較結果を示す。表中の○はマルウェアが検知されたことを、×は正規プログラムが誤検知されなかったことを表す。「誤検知」は、正規プログラムをマルウェアだと誤検知したことを表している。「検知漏れ」は、一部検知できないマルウェアがあったことを表す。

表 5　誤検知および検知漏れの比較結果

Table 5　Experimental results

	マルウェアらしい振る舞い(監視対象)					
	タイプ 1	タイプ 2	タイプ 3	タイプ 4	タイプ 5	本方式
インストーラ（常駐型）	誤検知	誤検知	×	×	×	×
メーラ	×	×	×	×	×	×
FTPクライアント	×	×	×	誤検知	×	×
ブラウザ	×	×	誤検知	×	×	×
Etherreal	×	×	×	×	誤検知	×
インストーラ（アンチウィルスソフト）	誤検知	誤検知	誤検知	誤検知	×	×
マルウエア	○	○	検知漏れ	検知漏れ	検知漏れ	○
ミューテーション型マルウェア	○	○	検知漏れ	検知漏れ	検知漏れ	○

2.3.2 考察

(1) 提案方式の検知精度

今回の FileMon を用いた基本的な機能確認実験では、検査対象としたすべてのマルウェアにおいて、自分自身のファイルのプログラム部分のすべてが READ されていること（すなわち自己ファイル READ が行われていること）が確認された。

実験で使用したマルウェアのうち、NetSky.Z は感染の際に自分自身を Zip 圧縮したものを相手に送りつけるマルウェアである。同じく、Beagle.AG は自分自身を鍵付き Zip 圧縮したものを送る（鍵は常に変化する）マルウェアである。Mimail.Q は感染の度に自分自身を変異させるマルウェアである。表 1 の結果から明らかなように、本方式によれば、シーケンシャル READ あるいはブロック READ のいずれかの検出により、これらミューテーション型マルウェアを含むすべてのマルウェアの検知が可能であることが確認できた。

また、表 4 に示す代表的な正規プログラムを用いた誤検知検査実験の結果により、本方式によって正規プログラムが誤検知されることもなかった。従って、自己ファイル

READ が検知された際に、「ファイルのプログラム領域のすべて」を READ しているかどうかということをチェックすることにより、マルウェアとその他の正常なプログラムを切り分けることが可能であることが確認された

表 5 より、それぞれの「マルウェアらしい振る舞い」を個別に見た場合に、本方式は、従来の方法と比べて、検知漏れが起こらず、かつ、誤検知が生じないというきわめて優れた方式であるということが分かる。特に、タイプ 1 からタイプ 5 までの既存方式を AND 条件で組み合わせたとしても、本方式の特徴であるミューテンション型マルウェアの検知能力を備えることはできないことに注意されたい。

(2) 提案方式の適用範囲

本方式は、プロセスが自分自身の本体のファイルのプログラム領域のすべてを READ するかどうかをチェックするというものである。よって、本方式によって、検知可能なマルウェアの要件としては以下の 3 つを備えている必要がある。

- 自プロセスから自分自身を READ する
- プログラム領域をすべて READ する
- READ の対象はファイルである

2.3.1 節の検証実験で用いたマルウェアは以上の要件を全て満たすものであったため、本方式により検知が可能であったが、これらの要件を満たさない以下のようなマルウェアが存在した場合には、本方式ではそのままでは検知することができない。

- 他のプロセスにマルウェアのプログラムが含まれているファイルを READ させるマルウェア【要件 1】
- プログラム領域の一部のみを READ するマルウェア【要件 2】
- ファイルからの読み込みは行わず、メモリ上にのみ存在するマルウェア【要件 3】

以下では、これらのマルウェアの検知についての考察を行う。

【要件 1】他のプロセスに自身のファイルを READ させるマルウェア

他のプロセスに自身のファイルを READ させるマルウェアとは、2.2 節で示したマルウェアらしい振る舞いの挙動 3-2 に示した動作を行うマルウェアのことであり、マルウェアとしての振る舞いを隠蔽するために、自らのファイルの流布を他のプロセスに依頼して実行させるマルウェアのことである。原理的には、2.2 節の検知処理フローで説明したように、検査対象とするプロセスを依頼先のプロセスに設定しなおして本方式による検査を再帰的に行えば、このようなマルウェアについても検知可能であるが、再帰的検査を具体的に実装するにあたっては、READ を行うプロセスと検査対象とすべきファイルのプロセスとが異なるため、単なるファイルシステムのフィルタドライバの監視

だけでは検知できず、それ以外のアクセスを監視する必要が生じる。
このタイプのマルウェアはさらに以下の3つに分類できる。

① 既存のプロセスの機能を用いて自身のファイルのREAD依頼をするマルウェア

既存のプロセスの機能を用いて自身のファイルのREAD依頼をするマルウェアとは、例えばMicrosoft Outlook Expressのようなメーラに自身のファイルをREADさせ、メール送信を行わせることにより自身を流布させるマルウェアである。Microsoft Outlook Expressはメール送信用のインタフェースとしてMAPI（Messaging Application Program Interface）をサポートしている。そこで、本方式を拡張して、MAPIのメール送信APIであるMAPISendMailを監視すれば、マルウェアからMicrosoft Outlook Expressへの自己ファイルREADの依頼（マルウェアを添付ファイルとしてメールに添付させるための指示）を捉えることができる。既存のプロセスであれば、マルウェアによる自身のファイルの送信依頼はこのような汎用的なAPIを通じて依頼するしか方法がないため、こうしたAPIを監視すれば、フィルタドライバでの監視による検知と同様の検知が可能である。

② 自ら新しいプロセスを起動し、自身のファイルのREAD依頼をするマルウェア

新しいプロセスを起動するマルウェアとは、マルウェアが別の実行プロセスを新たに起動し、起動したプロセスに自分自身のファイルのREADを依頼するマルウェアである。このようなマルウェアに対しては、上記と同様に、本方式を拡張して、プロセスを起動するAPIであるCreateProcessを監視し、CreateProcessによって起動されたプロセスを検査対象プロセスとして、当該プロセスがCreateProcessを発行したプロセスの実行ファイルをREADした場合にアラートをあげるようにすれば検知が可能である。

③ 2つのプロセスが互いのファイルを送信し合うマルウェア

マルウェア自体が2つのプログラムから構成されており、それぞれのプロセスが互いのファイルをREADし、送信を行うというマルウェアが考えられる。マルウェアが2つ以上に分割されていて、互いにREADしあうというような場合も考えられる。このようなマルウェアに対しては、それぞれが独立に活動するため、マルウェアが自身のファイルのREADを依頼することも、他のプロセスを起動することもない。よって、これまで述べてきたような方法では対処できない。こうしたマルウェアに対しては、OSに対して強制アクセス制御（MAC：Mandatory Access Control）機能を付加し、本方式とMAC機能とを併用することにより対処可能である。MAC機能を用いた対策方式については、本節(3)で考察する。

【要件2】プログラム領域のすべてをREADしないマルウェア

プログラム領域のすべてをREADしないマルウェアとは、自身のファイルの一部のみをREADして全体を再構成するマルウェアである。このタイプのマルウェアは以下の2つのタイプに大別できる。

① ジャンクコードを付加するマルウェア

ある程度のサイズのジャンクコード（プログラムの実行上は意味のない不要なコード）をマルウェア本体に付加することにより、「ファイル全体の中の一部に本体が隠されている」というマルウェアが作成可能である。このようなマルウェアは、感染にあたっては、

a. ファイル全体の中からマルウェア本体の部分のみをREADし、
b. 新たなジャンクコードを生成した上で、
c. a.のマルウェア本体とb.のジャンクコードを合わせたものを他のコンピュータに送信する

という動作をする。この場合には、感染したコンピュータでは、マルウェアは自身のファイルの一部しか読み込まないため、提案方式そのままでは検知できない。この問題を回避するために、自身のプログラムファイルのある比率（閾値）以上を読み込んだ場合には自己ファイルREADとみなすという閾値制御を取り入れることも考えられるが、その場合には、閾値次第ではマルウェアの検知漏れや正規プログラムの誤検知が発生することになり、これまで従来方式で課題となっていた閾値の最適化が難しいという問題が生じてしまうこととなる。

この問題は特にインストーラの誤検知において顕著に現れるが、自己ファイルREADという観点からインストーラの動作を考えた場合、インストールを実行するプログラム部分は読み込まない。一方、上述のマルウェアにおいては、感染にあたってマルウェアの本体であるプログラム部分のファイルREADが発生する。このため、走行しているプロセスのプログラム部分を読み込んでいるかどうかが検出できれば、誤検知を生じさせずにこうしたマルウェアの検知が可能となる。そこで、本方式を拡張して、自身のファイルが一部でも読み込まれた場合に、走行しているプロセスのプログラムのエントリポイントがREADされたか否かを検査することで、このようなマルウェアに対処できる。すなわち、ファイルの一部分のREADであっても、プログラムのエントリポイントも含めてファイルREADを行っている場合に自己ファイルREADが発生したと判断する。

② データ領域のデータから自分自身を再構成するマルウェア

任意のマルウェア（マルウェアAとする）に対し、

a. そのマルウェアを複製し（これをマルウェアBとする）、

b. マルウェアAのデータ領域にマルウェアBを挿入する

というマルウェア（これをマルウェアCとする）が作成可能である（図10）。マルウェアCは、データ領域の中のマルウェアBの部分のみをREADするだけで、自分自身を再構成できる。このようなマルウェアは、①に示した自己のプログラムのエントリポイントの読み込みチェックを行うタイプの自己ファイルREAD（プログラム領域をREADしたかどうか）の検査であっても検知できない。こうしたマルウェアは形としてはインストーラと何ら変わるものではなく、インストーラがマルウェアであった場合に他ならない。よって、こうしたマルウェアに対しては、自己ファイルREADの有無といった振る舞いをチェックするだけではなく、自身がREADしようとしているデータの内容を分析して、その中にマルウェアCのプログラム部分が含まれていないかどうかを調べる必要がある。

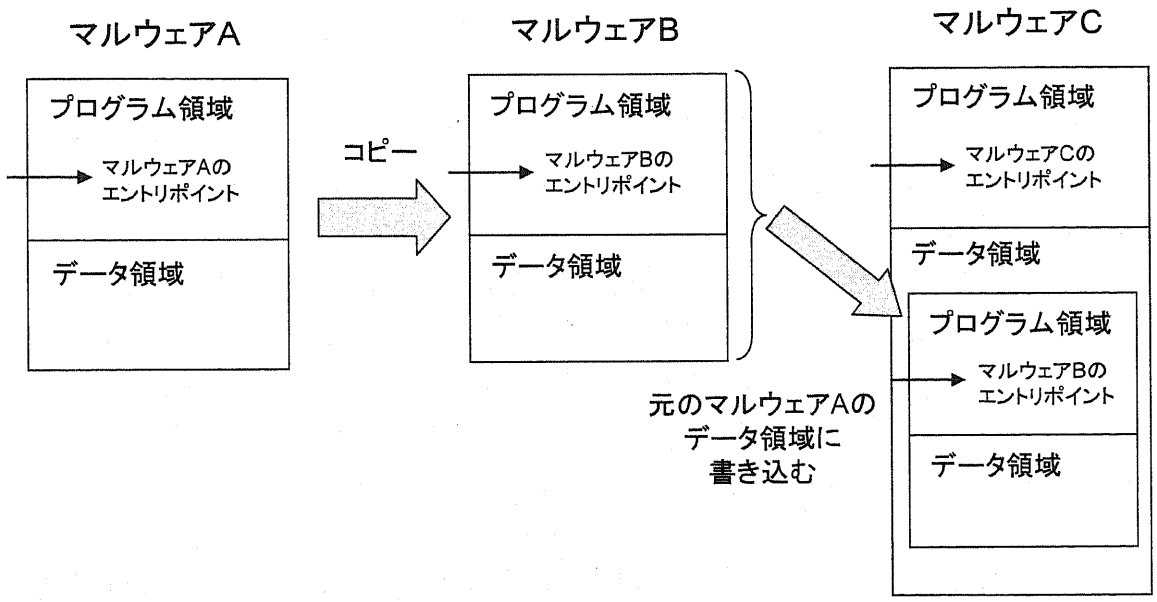

図10　自分自身を再構成するマルウェア

Fig. 10　Self-organizing malware

【要件3】メモリ上にのみ存在するマルウェア

コンピュータのメモリ上でのみ動作し、ファイルシステムの中にマルウェア本体のデータを書き込むことがないCodeRed[11]のようなメモリ常駐型のマルウェアは、自己ファイルREADのイベントが発生しないため提案方式での検知は不可能である。しかしながら、基本的には、メモリ常駐型のマルウェアであっても、感染活動を行おうとする

際には、メモリ上にある自分自身のプログラムを、実行しているプロセスの作業領域に一旦読み込むという動作は必ず行うはずである。従って、自己ファイル READ に基づく検知方式を拡張して、「自己メモリ READ」の検査まで実施できれば、2.3 節で示した方法と同様の考え方でこのようなマルウェアを検知することが可能であると期待できる。ただし、そのためには、OS のメモリ管理機能にモニタするためのフック機能を組み込む必要がある。Windows 等の商用 OS では、OS の中枢機能であるメモリ管理機能をモニタする機能は、現時点では提供されていないため、自己 READ の検査対象をメモリまで広げた方式の有効性を検証するところまで確認することは困難である。

(3) 強制アクセス制御による検知方式の強化

前項で述べたように、2 つ以上のプロセスが相互に連携しあいながら自己複製、感染を行うようなマルウェアに対しては、特定のプロセスに限定した監視だけでは対処できず、システム全体としてプロセス間の独立性を保障する仕組みが必要となる。こうした仕組みとして有効な方法が強制アクセス制御（MAC : Mandatory Access Control）である。

強制アクセス制御は、Microsoft Windows や UNIX／Linux 等が標準で備える自由裁量のアクセス制御機構（DAC : Discretionary Access Control）が抱える脆弱性およびそこから生じるリスクを解消するものとして考案されたものであり、当初は、軍事用途を含めた特殊な要件を持つシステムに適用、実装されたが、2004 年 11 月にオープンソースの Linux に強制アクセス制御を実現するためのフレームワークである LSM [58]およびそれに対応した SELinux [59][60]が組み込まれたことにより一気に身近なものとなった。

強制アクセス制御が働くシステムにおいては、各利用者がアクセスできるファイル、プロセス、デバイスといったすべてのコンピュータ資源はシステム管理者の定めたポリシーに従って制限される[61][62]。例えば、図 11 に示すように、ファイルαに対してはユーザ A のみが read/write 可能であり、ファイルβに対しては、ユーザ B のみが read/write 可能とポリシーに設定しておくことにより、ユーザ A からファイルβへのアクセスやユーザ B からファイルαへのアクセスを禁止することが可能となる。また、ユーザ A が自身のファイルαをユーザ B が管理するファイルβへコピーしようとする動作も禁止される。

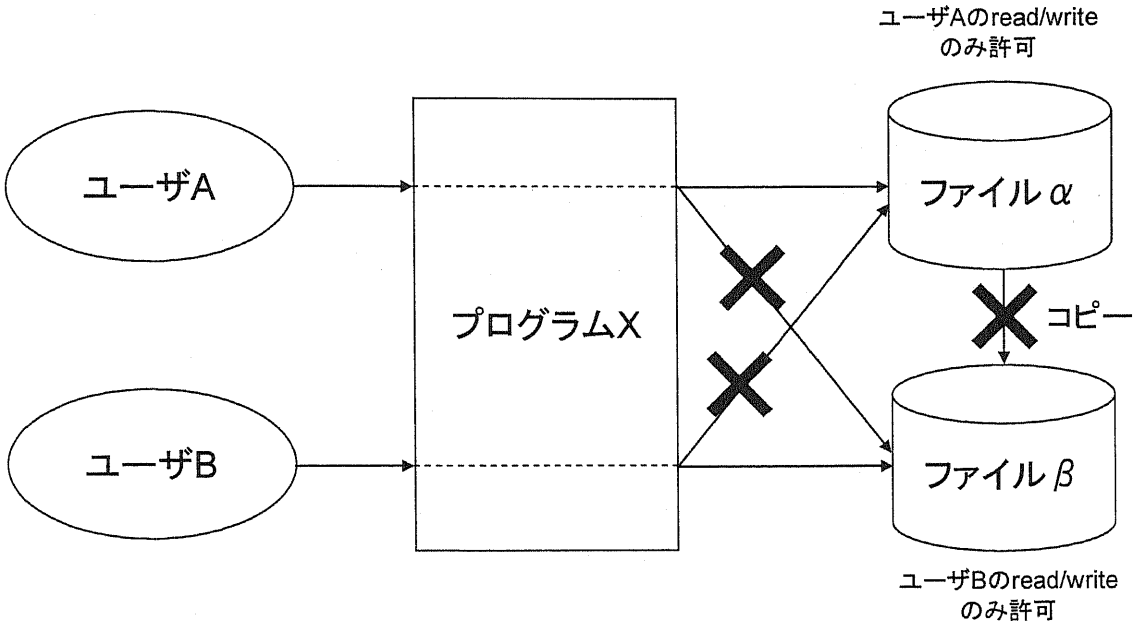

図 11　強制アクセス制御

Fig. 11　Mandatoru Access Control

こうした機能を用い、「通常のプログラムに対しては、自分以外の実行ファイルへのアクセスを禁止する」というポリシーを設定することよって、マルウェア自身も勝手に他プロセスの資源にアクセスすることができなくなるため、2 つ以上のプロセスが互いに相手のファイルを送信することを制限することができる。

強制アクセス制御は、もともと汎用の用途に供することを目的に設計された OS に対し、アクセス機能を制限することにより「必要でない」機能の実行を禁止することを可能とするものである。それは全てのプロセス・全てのユーザに対して例外無く適用され（標準の Linux においてはシステム管理者である root に対しては適用されない）、プロセスやユーザがアクセス可能なファイルやディレクトリ等の資源を詳細に制限できる。一般に、強制アクセス制御を導入した OS をセキュリティ強化 OS と呼ぶ。セキュリティ強化 OS としては、SELinux が有名であるが、ポリシーの記述にきわめて専門的な知識を必要とするため使いづらいという問題がある。そこで、最近では、ポリシーの生成を学習しながら動的に行うことでシステム管理者の負担を減らす工夫がなされるようになっている。筆者もその開発に携わった TOMOYO Linux[63][64]は、一般的なファイル名を用いてアクセス制御の記述ができ、かつ自動学習機能を有するセキュリティ強化 OS であり、専門的な知識がなくてもセキュリティを強化したシステムが構築できるという特長を持つ。

こうしたセキュリティ強化 OS を搭載したシステムでは、適切なポリシーさえ策定さ

れていれば、バッファオーバフロー等によりマルウェアにプロセスを乗っ取られ、そこからシステム管理者権限を付与したプロセスを起動されても無制限に被害を受けることはない。ただし、正規の手順（例えば管理者権限のユーザ ID とパスワードによるログイン認証）を経れば、システム管理者のプロセスとしてシステムにログインできるため、システム管理者のユーザ ID とパスワードが漏洩した場合、図 11 のようなユーザ単位でのアクセス制御ができなくなってしまうという問題がある。SELinux についても、root 権限を無力化したポリシーは策定できても[65]、そのポリシーを変更、反映するためのインタフェースは残っており、その部分を守っているのは結局昔ながらの ID、パスワードによる認証である。

著者らは、このような、正規の手順を用いて管理者権限でログインして強制アクセス制御を無効化するという脅威に対して、強制アクセス制御を応用して防御する新たな方式を提案している[66][67]。以下、本方式の説明を行う。

ユーザを認証するために一般的に用いられている方法は、ユーザが入力するパスワードによるログイン認証である。ログイン認証は通常は、ユーザがシステムにログインする時点において一度だけ行われるのみである。そのため、不正者が辞書攻撃等でパスワードを割り出したり、認証プログラムの脆弱性を攻撃して認証を回避することによって、ログイン認証のルーチンを一点突破しさえすれば、なりすましに成功するという脅威を常に抱えている。

従来のログイン認証には、以下のような課題がある。

- 認証は一度しか行われない
- 多くの環境では認証手段はパスワードに基づいている
- パスワードが漏洩しても、漏洩したことに気づかず被害の拡大を防止できない
- 認証プログラムに脆弱性が存在すると認証そのものが無力化する

ログイン認証が持つ課題を克服して、その脆弱性を防止する基本的な考え方は、ログイン認証を多重化することである。認証の多重化自体は強制アクセス制御を採用していない OS でも可能であるが、強制アクセス制御を採用したセキュリティ強化 OS においては、その多重化した認証の実行を強制することができる点が最大の特長となる。

ログイン認証が多重化できるようになることで、以下のようなメリットが生じる。

① 好きなだけログイン認証を強制させることが可能となる

保護したいコンピュータ資源の重要度に応じて、任意の回数だけログイン認証を強制させることができる。

② 認証プログラムの脆弱性に左右されない方式とすることができる

認証が 1 回しか行えない場合は、認証プログラムの脆弱性は致命傷となる。しかし、複数の異なる認証プログラムを用いて認証を強制させることができるので、

認証プログラムの１つに脆弱性があったとしても問題にならない。

③ パスワード以外の認証手段も利用することができる

通常のログイン認証では、パスワードを入力する過程等をシステム側が知ることはできず、認証プログラムは「入力結果」に基づいてのみ、本人かどうかを判断する。しかし、ログイン認証が多重化されていれば、一段目のログイン認証後にはシステムとの確認環境が提供されるので、二段目以降の認証プログラムはユーザの挙動を詳細に知ることができる。すなわち、それまでに行った操作内容等の「入力過程」も認証の判断基準に使えるので、パスワード以外の要素を認証に用いることが可能である。その他にも、特定のファイルが存在するかどうかや、ファイルの内容そのものをパスワードとして利用することもできる。このように、（二段目以降の）認証プログラムを通常のアプリケーションと同じ感覚で作成できるため、認証を実現できる組み合わせを無限に構成し得る。

④ 全ての認証を突破されない限り被害を受けない

全てのログイン認証に成功するまでは重要なコンピュータ資源へのアクセスを許可しないようなアクセスポリシーを策定することができる。具体的には、あるログイン認証プログラムからは、次のログイン認証に必要な最低限の操作のみを行えるようにアクセスポリシーを策定する（すなわちアクセスポリシーの変更などの行為はこの時点ではまだ許可されないようにする）ことで、侵入をより困難にさせることが可能である

ログイン認証を連続して複数回行わせることは強制アクセス制御機能を持たないOSでも可能である。例えば、図11の例において、プログラムX自身が各ユーザに対する認証を多段階で行うように作り込むことで実現できる。しかし、プログラムの振る舞いをポリシーにより制限できない場合、それぞれのプログラムが自分自身で外部からの不正に対処できるようにしておく必要があり、認証プログラムの作成にあたって、抜け道が発生しないように細心の注意を払わなければいけない。強制アクセス制御機能があれば、外部からの不正な操作をポリシーにより排除できるので、認証プログラム自身の抜け道を心配する必要が無く、プログラムを容易に作成できるという利点がある。

2.4 まとめ

自分自身のファイルを READ して自己複製を行うというマルウェア特有の振る舞いに着目し、プロセスのファイルアクセスを監視することにより、リアルタイムに未知マルウェアの検知を行う方式を提案した。本方式は、従来の方式では検知が困難であった、ポリモーフィック型マルウェアやメタモーフィック型マルウェアなどのミューテーション型マルウェアの検知に対しても有効な方式である。OS のファイルアクセスを観測するモニタツールを用いた基礎実験の結果から、本方式の有効性が確認できた。今後は、本方式を OS 上で実装した上で各種のマルウェアに対する実証実験を行うことにより、実際の検知率、誤検知率、リアルタイム検知を行うにあたってのオーバヘッドなどを計測していく必要がある。

以上、自己ファイル READ の検出に基づく検知方式により、原理的には全てのウィルスとワームの検知が可能となる。しかしながら、1 章の背景で述べたように、コンピュータシステムに根ざした社会活動や企業活動の BCP（Business Continuity Plan）を確保するという意味においては、提案した方式によるマルウェア検知のみでは不十分であり、運用面も含めたシステムトータルな対策とする必要がある。特に、以下の 2 点の観点から、さらなる対策技術を検討する必要がある。

① コンピュータがマルウェアに感染した状態での検知

ビヘイビアブロッキング法に基づく未知マルウェアの検知は、利用者のコンピュータは既にマルウェアに感染している状態で、その動作を監視することで感染の有無を検知しようとするものである。特に、本論文で提案した方式ではマルウェアらしい振る舞いとして伝染行為に着目している。通常、マルウェアに感染したコンピュータは、感染、発病、伝染という段階を踏むため、自己ファイル READ に基づく方式のようにクライアントコンピュータ上のみでの監視では、既に発病した状態で正しく監視が行えるかという問題が残る。このため、単なるクライアントコンピュータ上での監視だけでなく、守るべきコンピュータが接続される LAN ネットワーク内に監視装置を設置して、外部ネットワークとの送受信状況を監視する等、外部からマルウェアによる感染状態を監視する仕組みも必要となる。

② リアルタイム検知に伴うコスト

理想的には、LAN 内の全てのコンピュータに自己ファイル READ に基づく方式を搭載し、常にリアルタイムの監視を行うことが望ましい。しかしながら、全てのコンピュータでリアルタイムに監視を行うことには、オーバヘッドの発生などの相応のコストが発生することになる。そこで、LAN 内においてリアルタイム監視を行

うコンピュータはその数を限定し、そのうちの１台でも未知のマルウェアを検知すれば、その情報を、パターンマッチング法における定義ファイルのように、擬似的な定義ファイルとして直ちにLAN内の残りのコンピュータに通知するようにする。これにより、多くのコンピュータは単に（疑似）定義ファイルとのマッチング動作のみを行えばマルウェアの LAN 内への蔓延を防止できるようになるので、システム全体として実運用上リーズナブルなコストでBCPが確保できることになる。

そこで、次章では、上記①および②を実現するための方式を検討する。

第3章
送受信データ間の相関に基づくマルウェア蔓延防止方式

3.1 従来技術における課題

外部ネットワークとの送受信状況を監視してマルウェアをブロックする方法としては、ファイアウォールによる方法が最も一般的な方法である。特に、クライアントコンピュータ上でも検知・防御が可能な方式としてパーソナルファイアウォールが広く普及している。

パーソナルファイアウォールでは、利用者の指示に従って、コンピュータから外部への通信ならびに外部からコンピュータへの通信を制御する[68]。パーソナルファイアウォールを用いれば、利用者が許可していないポートやプロトコルを用いた通信は遮断できる。また、アプリケーションごとに通信を遮断するかどうかを設定することも可能である。従って、外部からのマルウェアの侵入（1次感染）を防ぐことができ、また、万一、コンピュータがマルウェアに感染しても、他のコンピュータへさらに感染（2次感染）しようとして外部へ通信を始める段階で阻止することができる。すなわち、未知のマルウェアによる蔓延を食い止めることができる。

しかしながら、HTTP や SMTP など、一般的に使われ、ユーザが許可しているポートやプロトコルあるいはアプリケーションを用いた通信は当然遮断することはできない。従って、それらを利用して侵入してくるマルウェアに対しては1次感染を阻止することはできない。また、2次感染の防止の際にも、感染したマルウェアによる通信がパーソナルファイアウォールに検出された時点で利用者にアラートが上がるが、一般レベルの利用者にとってはそれがマルウェアによる異常な発信なのか正常な発信なのかの識別が難しく、概して通信の許可を与えてしまいがちであるという問題も残る。

ネットワーク上を流れる通信パケットを監視して、ルール上許されない挙動を行う通信パケットを検出してブロックするという方法も提案されている。例えば、以下のような方法が提案されている。

- ICMP（Internet Control Message Protocol）で送信先が見つからないというパケットを調べポートスキャンが行われているかどうかを検出する[69]

- DNS で名前解決を正しく行った IP アドレスへの送信は許可し、それ以外は一旦内部のキューにつなぎ、キューの長さがある閾値を越えた場合にマルウェアによる攻撃とみなす[70]
- ネットワークモニタリング装置（プローブ）を用いて、ネットワークを流れるプロトコルの割合比率を監視する。割合比率が通常の値と著しく異なる場合に、マルウェアによる感染があったと検知する[71][72]
- IDP（侵入検知防御）装置において、ネットワークのレイヤ 7（アプリケーション）層でアクセス情報の統計分析を行う。例えば、HTTP リクエストのパラメータ長や文字セットなどをチェックして、統計的に見て異常値であればマルウェアによる攻撃が行われているとみなす[73]
- 通信の挙動とプロセスの対応をルール化し、あるプロセスからルールファイル上許されない通信が行われるとそれを検出して、当該プロセスを停止してマルウェアの外部への感染を防止する [74]

しかしながら、これらは、いずれの方法も、ルールや閾値（ネットワーク負荷の急激な上昇、プロトコル比率の急激な変化、ポートスキャン動作の有無等）の決め方が性能を左右する鍵となる。そして、それらルールや閾値については経験的に決めていかざるを得ないために、どうしても検知漏れや誤検知の問題が避けられない。また、文献[74]の方式は現時点では正常なプロセスになりすまして外部への通信を行うようなマルウェアに対しては対処がなされていない。さらに、この方式は、エンタープライズネットワーク内のあるコンピュータに感染した未知マルウェアが他のコンピュータに 2 次感染を試みる時点でこれを止めるものであり、未知マルウェアに感染したコンピュータがエンタープライズネットワークの外部にあり、そこからエンタープライズネットワーク内のコンピュータ群に次々と攻撃が仕掛けられるような場合には、その感染を防ぐことはできない。

3.2 提案方式

本節では、「ネットワークを経由してコンピュータに侵入し、自己の複製をネットワークで送信する」というウィルスやワームのネットワークを介した自己増殖機能に着目し、送受信データ間の相関を検査することにより感染を検知するとともに、新たに発見されたマルウェア本体を擬似的なウィルス定義ファイルとして直ちに使用することにより、エンタープライズネットワーク内で最初に感染したコンピュータ以外の、他のコンピュータへのマルウェアの蔓延を自動的に防止する方式を提案する。本方式は、クライアントコンピュータが受信するデータと送信するデータとの相関を見ることによりマルウェアによる感染の有無を検出するため、監視はクライアント上で行っても良いし、また外部のネットワーク監視装置でクライアントコンピュータへの送受信データを監視する形で行うこともできる。

また、本方式は、マルウェアが添付されたメールを大量に送信して感染を広げる「マスメーリング型マルウェア」（いわゆる狭義のウィルス）と、ネットワーク経由で脆弱性を狙って感染を広げる「ネットワーク感染型マルウェア」（いわゆるワーム）の両方に対応可能な未知マルウェア検知ならびに蔓延防止方式となっている。

3.2.1 マルウェアによる感染行為の規定

(1) 送受信データ間の相関の検査

ネットワークを介して宿主や自らの力で増殖を行うウィルスやワームのようなマルウェアは、ネットワークを経由してコンピュータに侵入し、コンピュータ内で自己の複製を作成し、これを外部へ送りつけるという動作を繰り返す。このような感染活動の場合には、感染したコンピュータが受信したデータと更なる感染のために外部へ送信するデータとの間には何らかの相関が存在する。提案方式は、この相関関係を検査することによりマルウェア感染の有無を検知しようとするものである。

送受信データ間の相関の検査に基づくマルウェア検知は、具体的には、Windows OSの通信機能を司るWinsock API [75]をフックすることにより利用者のコンピュータの送受信データを監視し、「外部から受信したデータ」と「外部に送信したデータ」が類似しているか否かを検査して、送受信データ間に類似したものがあった場合にマルウェア感染の疑いがあるとしてアラートを上げる。

すなわち、単なる自己複製の検査ではなく、「外部からネットワークを介して利用者のコンピュータに侵入し、自分の複製をネットワークを通じて外部に発信する」という

振る舞いを検査することにより、単なるファイルコピーの検出で発生する誤検知の問題を解消する。また、3.1 節で述べたネットワーク送信状況の異常有無による検知方法における検知漏れや誤検知の問題も避けることができる。

なお、本方式は具体的には「受信データと送信データが十分に一致した」ことを検出する方法であるため、コンピュータに侵入するときと外部に伝染するときでその姿が変異するミューテーション型マルウェアについては検知することは困難である。これらのマルウェアに対する検知は、2 章で提案したクライアント上での自己ファイル READ の検出による検知に委ねる。

(2) 擬似定義ファイルによる蔓延防止

本方式では、ネットワークを介して「外部から受信したデータ」と「外部に送信したデータ」が類似しているか否かを測ることによって、ネットワークを介したマルウェアの自己増殖活動を捕らえる。よって、ある利用者のコンピュータで未知マルウェアが新たに発見された場合、以下のように、そのマルウェア自身を擬似的な定義ファイルとして利用することが可能である。

① ある利用者のコンピュータで新たに検知された未知マルウェアを（必要に応じて無毒化処理を施した上で）エンタープライズネットワーク内の他のコンピュータに渡す。
② 未知マルウェアを受け取ったコンピュータは、これを擬似的に自身の「外部に送信したデータ」として登録する。
③ 未知マルウェアを受け取ったコンピュータは、その後、通常通り「外部から受信したデータ」と「外部に送信したデータ」の類似度を定常的にチェックする。
④ 未知マルウェアを受け取ったコンピュータに実際に当該マルウェアが外部から侵入した時点で、③のチェックの結果が陽性となり、当該未知マルウェアが検出される。

このように、エンタープライズネットワーク内の 1 台のコンピュータに未知マルウェアが感染し、外部に伝染行為を行おうとした時点でこれを検出し 2 次感染を防ぐとともに、直ちに他のコンピュータに擬似定義ファイルを配布することが可能であるため、エンタープライズネットワーク内の 2 台目以降のコンピュータに感染する前に当該マルウェアを自動的にブロックすることができる。

本方式は、既存の技術に比べ、エンタープライズネットワーク内の感染コンピュータからの未知マルウェアの 2 次感染を阻止するだけでなく、ネットワークの外部にある感染コンピュータからエンタープライズネットワーク内のコンピュータ群に未知マルウ

ェアが次々に送られてくることによる感染蔓延をも防止できるという特長をもつ。特に、アンチウィルスベンダの提供するウィルス定義ファイルが正式にリリースされる前に、とりあえず擬似定義ファイルをエンタープライズネットワーク内に瞬時に配布することで疑わしい未知マルウェアをブロックできるというメリットの持つ意味は大きい。

本方式では、マルウェアの自己増殖活動を捕らえるために、単なる自己複製を検出するのではなく、「ネットワークを介して受信されたファイルが再び外部に送信される」という事象を監視することになる。これは、Windows OS の通信機能を司る Winsock API [75]をフックすることにより、エンドユーザのコンピュータにおける送受信データを常時モニタリングすることで達成される。よって、本方式は OS の DLL をラッピングするだけで実装が可能であり、ウィルス型（マスメーリング型）の未知マルウェアの検出はもちろん、エンドユーザのコンピュータでのリアルタイム検知が求められるワーム型（ネットワーク感染型）の未知マルウェアの検出にも適応する。

ただし、「外部から受信したデータ」と「外部に送信したデータ」の類似度をチェックするというオーバヘッドは少なからず発生するため、その性能を評価する必要がある。オーバヘッドについては、次節で考察する。

3. 2. 2 検証実験

(1) 検知システムの実装

本方式では、Winsock API をフックすることにより、エンドユーザのコンピュータにおける「外部から受信したデータ」と「外部に送信したデータ」をそれぞれのバッファにコピーし、両者の中に類似度の高いデータが存在するか否かをチェックする。

Winsock API の機能は、wsock32.dll および ws2_32.dll に内包されている[76]。今回の実装では、Windows2000 を対象とし、ws2_32.dll のラッパーDLL を作成した。送信データの取得のために send、WSASend、sendto、WSASendTo、closesocket の 5 つの API を、受信データの取得のために recv、recvfrom、WSARecv、WSARecvFrom、closesocket の 5 つの API をフックする☆2。

本ラッパーDLL により、送受信データの取得、およびデータ間の類似度の検査を行う。ラッパーDLL の動作の詳細を以下に記す。開発環境には、Microsoft Visual C++ 6.0 を用いた。

【データ受信時】

☆2 closesocket は送受信において同一の API なので、計 9 つの API をフックしている。

ラッパーDLL を介して受信バッファにコピーされたデータは、一旦ファイル単位で格納される。ファイル全体をすべてバッファリングする方法は、ストレージ容量の観点からもデータを比較する際のコストの観点からも非現実的であるため、効果的なデータ管理法が必要となる。本論文では、以下の 2 つの方法を試した。

① 分割ハッシュ比較法

a. 各ファイルを先頭から 1024 バイト単位で区切った上で、各データブロックをハッシュ化する。なお、ファイルkのデータブロックの総数をB_kとする。

b. 各ファイルに対するすべてのブロックのハッシュ値（以降、「ハッシュ値集合」と呼ぶ）を受信バッファに蓄える。ハッシュアルゴリズムは SHA-1 を採用しており、各ハッシュ値は 20 バイト長である。

② 部分データ比較法

a. 各ファイルを先頭から 1024 バイト単位で区切った上で、各データブロックの先頭 20 バイトを「部分データ」として取り出す。なお、ファイルkのデータブロックの総数をB_kとする。

b. 各ファイルに対するすべての部分データ（以降、「部分データ集合」と呼ぶ）を受信バッファに蓄える。

なお、受信バッファは全てのプロセスで共有されるため、共有メモリ空間上に実装している。

【データ送信時】

ラッパーDLL は、Winsock API を介して送信されるデータを一時的に送信バッファに隔離する。そして、送信バッファ中のデータを、その時点までに受信バッファに格納されている各データ 1 つ 1 つと比較する。類似のデータがなければ、隔離していたデータを通信相手に送信する。送信データと類似度の高い受信データが検出された場合には、送信完了を意味する closesocket API を強制的に発行することにより通信先に通信のキャンセルを通知する。

送受信データ間の類似度チェックは、データ受信時の受信バッファにおけるデータ管理方法に合わせて、以下のような 2 通りの方法で検査を行う。

① 分割ハッシュ比較法

a. すべての受信ファイル（受信バッファ内にハッシュ値集合が蓄えられているファイル）に対して、ハッシュ値が一致したデータブロック数をカウントするための変数を用意する。受信ファイルkに対する変数をz_kとする。すべてのz_kの

値を0に初期化する。

b. 送信バッファ内に送信ファイルの先頭から1024バイト分のデータ（送信ファイルの第1データブロック）がたまった時点で、この第1送信データブロックのハッシュ値を計算する。$i=1$として、以下の手順c～fを実行する。

c. すべての受信ファイルのハッシュ値集合におけるそれぞれの第iブロックのハッシュ値に対して、第i送信データブロックのハッシュ値と一致するものがあるか否かを検査する。

d. 一致するものがなければ、第i送信データブロックを実際に送信する。手順gに進む。

e. 第i送信データブロックのハッシュ値が受信ファイルkの第iブロックのハッシュ値と一致した場合には、z_kをインクリメントした上で、送信データと受信ファイルkとの一致度$\frac{z_k}{B_k}$を計算する。

f. $\frac{z_k}{B_k}$がある閾値を越えたならば、送信完了を意味するclosesocket APIを強制的に発行することにより通信をキャンセルする。$\frac{z_k}{B_k}$が閾値を越えていなければ、第i送信データブロックを実際に送信する。

g. 送信バッファ内に次の1024バイト分のデータ（送信ファイルの第2データブロック）がたまった時点で、この第2送信データブロックのハッシュ値を計算する。$i=2$として、手順c～fを実行する。

h. 以下、送信バッファに次の1024バイト分のデータがたまるごとに、iをインクリメントしながら、手順c～fを実行する。

② 部分データ比較法

a. すべての受信ファイル（受信バッファ内に部分データ集合が蓄えられているファイル）に対して、部分データが含まれるデータブロック数をカウントするための変数を用意する。受信ファイルkに対する変数をz_kとする。すべてのz_kの値を0に初期化する。

b. 送信バッファ内に送信ファイルの先頭から1024バイト分のデータ（第1送信データブロック）がたまった時点で、$i=1$として、以下の手順c～fを実行する。

c. すべての受信ファイルの部分データ集合からそれぞれのi番目の部分データを取り出し、それと完全に一致するビット列が第$i-1$送信データブロック☆3およ

☆3 $i=1$の場合に限っては、第$i-1$送信データブロックはnullデータとなる。すなわち、各受信ファイルの先頭からの20バイト分のビット列と同じものが、送信ファイルの先頭から1024バイト目までの中に含まれるかが検査される。

び第i送信データブロックの中に含まれるか(すなわち、各受信ファイルの1024i バイト目から20バイト分のビット列が、送信ファイルの$1024(i-1)$バイト目から$1024(i+1)$バイト目までの中に含まれるか) を検査する。

d. 一致するものがなければ、第i送信データブロックを実際に送信する。手順gに進む。

e. 第$i-1$送信データブロックまたは第i送信データブロックの中に受信ファイルkのi番目の部分データが含まれていた場合には、z_kをインクリメントした上で、送信データとファイルkとの一致度$\frac{z_k}{B_k}$を計算する。

f. $\frac{z_k}{B_k}$がある閾値を越えたならば、送信完了を意味するclosesocket APIを強制的に発行することにより通信をキャンセルする。$\frac{z_k}{B_k}$が閾値を越えていなければ、第i送信データブロックを実際に送信する。

g. 送信バッファ内に次の1024バイト分のデータ（第2送信データブロック）がたまった時点で、 $i=2$として、手順c～fを実行する。

h. 以下、送信バッファに次の1024バイト分のデータがたまるごとに、iをインクリメントしながら、手順c～fを実行する。

なお、分割ハッシュ比較法は各ブロックのハッシュ値をそのマルウェアに特徴的なコード部分として、部分データ比較法は各ブロックの先頭20バイト分のデータをそのマルウェアに特徴的なコード部分としてそれぞれを定義ファイルに規定し、「ファイルの類似度を定義ファイルに基づくパターンマッチングで判定している」という意味では、上述のどちらの比較法も一般の商用アンチウィルスソフトで用いられている定義ファイルによるマルウェア判定と同様であると言える。しかし、商用アンチウィルスソフトでは、専門家がマルウェアプログラムの特徴的な一部分を抜き出すことにより定義ファイルを作成するものであるのに対し、本方式では、すべての受信データの内容を擬似的な定義ファイルとして自動的にかつリアルタイムに作成できるという特徴がある。すなわち本方式は、定義ファイルの生成に時間および人手をかける必要がないので、ゼロデイアタックのような攻撃に対しても有効であると言える。本方式では、このリアルタイム性を確保するために、受信データと送信データを 1024 バイトのブロックに分割し、各ブロックのハッシュ値や先頭20バイトを機械的に作成し、ブロックごとにマッチング検査を実施している。

【擬似定義ファイルによるブロッキング】

3.2.1 項(2)で述べたように、本方式においては、送信データと受信データを比較する

というコンセプトによって、未知マルウェア自身を擬似定義ファイルとして利用することが可能である。ただし、未知マルウェアの検知が「今までに受信したデータ」と「今から送信するデータ」の類似度を検査するのに対して、擬似定義ファイルによる当該マルウェアのブロッキングは「今、受信したデータ」と「今までに外部に送信したデータ（として格納されている擬似定義ファイル）」の類似度を測ることになる。

具体的には、送信バッファ、受信バッファに加え、定義ファイルバッファを用意した上で、ラッパーDLL に以下の機能を実装する。

a. 他のコンピュータから届けられた擬似定義ファイルは、定義ファイルバッファに格納する。
b. ラッパーDLL は、Winsock API を介して受信されるデータを、一旦受信バッファに隔離する。
c. ラッパーDLL は、受信バッファに隔離されたファイルを、定義ファイルバッファに格納されている各データ（擬似定義ファイル）一つ一つと比較する。比較の方法には、前述した分割ハッシュ比較法あるいは部分データ比較法などを用いる。
d. 類似のデータがなければ、受信データの隔離を解除する。擬似定義ファイルと類似度の高いデータの受信が確認されたならば、隔離した受信データを廃棄する。

(2) 検知システムによる検証実験

本方式の有効性を検証するため、本節(1)で説明したラッパーDLL を実装し、実際にマルウェアの検知を行った。ただし、未知マルウェアの入手が不可能であるため、ここでは本方式によって、代表的な既知マルウェアを検出することが可能であるかを測定することとした。また実験にあたっては、試行的に、両比較法ともデータの一致度 $\frac{z_k}{B_k}$ の閾値を 0.9 とした。すなわち、データを送信する際に、そのデータと 90%以上類似しているデータがすでに受信されている場合に、当該データを「マルウェアの疑いあり」と判断し、送信をブロックする。

本実験で使用したネットワーク構成を図 12 に示す。サーバ PC のスペックは OS：Fedora Core 2、CPU：Pentium4 1.6GHz、メモリ：384MB、マシン A の PC のスペックは OS：Windows 2000、CPU：Pentium2 400MHz、メモリ：128MB、マシン B の PC のスペックは OS：Windows 2000、CPU：Pentium2 400MHz、メモリ：384MB である。

サーバにはDNS名が割り当てられており、ドメイン名がinfected.net、ルータ、DNSサーバ、Mailサーバはそれぞれgw、ns、mailという名前である。ただし、実際にはルータ、DNSサーバ、Mailサーバは1台のPCで実装している。サーバPCのOSはFedora Core 2である。

クライアントのPCは2台であり、すでにマルウェアに感染しているマシンAがマルウェアには未感染のマシンBに攻撃をしかける。マシンAもマシンBもOSはセキュリティパッチが当たっていないWindows 2000、メーラはOutlook Expressである。なお、@infected.netドメイン宛てのメールは全てマシンBのメーラにて受信可能なように設定した。そして、マシンBに本節(1)で説明したラッパーDLLを装備した。

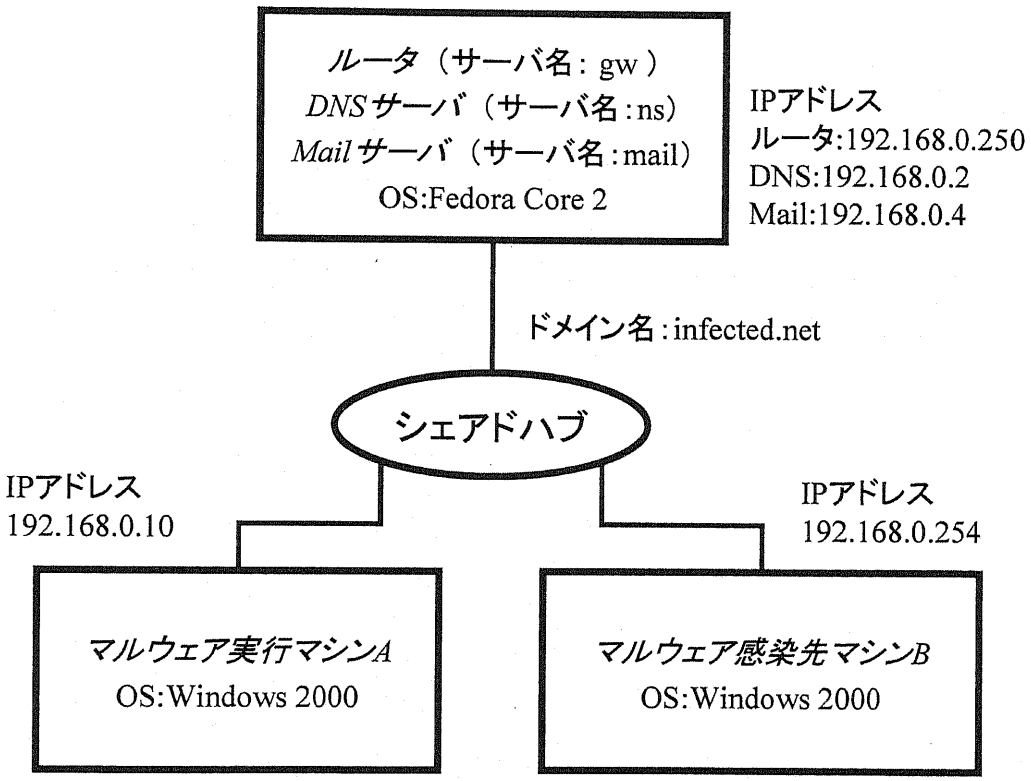

図12　隔離ネットワーク構成図

Fig.12　Experimental network.

マルウェアの検知実験は、図12の環境において、以下のように行った。

① ウィルスの検知

マシンAのマルウェアを実行し、マシンBにマルウェア付きメールを送信させ

る。マシンBは、マシンAからのマルウェア付きメールを受信し、メールの添付ファイル（マルウェア本体）を利用者が故意に実行することにより自らもマルウェアに感染し、マルウェア付きメールを発信し始める。代表的な既知マルウェアの感染に対し、マシンBに装備されているラッパーDLLが、これを検知できるかどうかを確かめた。

② ワームの検知

マシンAのマルウェアを実行し、マシンBに対してマシンBの脆弱性を突いた攻撃を行わせる。マシンBは、マシンAからの攻撃を受け、自らもマルウェアに感染し、他のマシンを攻撃し始める。代表的な既知マルウェアの感染に対し、マシンBに装備されているラッパーDLLが、これを検知できるかどうかを確かめた。

(3) 実験結果

ウィルス、ワームともに、分割ハッシュ比較法、分割データ比較法のそれぞれの方法により検知実験を行った結果を表6に示す。表中、○は、マルウェアが検知できたことを示し、×は検知できなかったことを表す。なお、3.2.1項(1)で述べたように、Netsky.Zのように感染前後でデータ内容が変化するミューテーション型のマルウェアは本方式では検知することができないため、検知実験の対象からは除いた。

表6　マルウェア検知実験の結果

Table 6　Malware detection by the proposed scheme.

名前	型	分割ハッシュ比較	部分データ比較
Beagle.X	ウィルス	○	○
Mydoom.Q		○	○
Sobig.F		○	○
Netsky.B		○	○
Netsky.D		○	○
Blaster.C	ワーム	×	○
Sasser.C		×	○

本方式の目的は、エンタープライズネットワーク内において、感染したコンピュータからの2次感染や同一マルウェアによる外部からの連続攻撃から、エンタープライズネットワークを守ることである。従って、安全側に倒してエンタープライズネットワークを守ることを基本的な考え方としている。しかしながら、疑わしきはマルウェアという方向に倒しすぎると、適用される環境によっては従来方式に比べて誤検知のケースが増加して実運用上問題となる場合があるため、その評価を行った。ここでは、本方式と2.1節で述べた既存のビヘイビアブロッキング法について、検知漏れと誤検知の発生状況を実測した。

評価に際し、表6に示した検知実験で用いたマルウェアの他に、いくつかの一般的なアプリケーションを用意した。用意したアプリケーションは、常駐型アプリケーションのインストーラ（memcl[57]）、メーラ（Edmax）、FTPクライアント（Smart FTP）、ブラウザ（Internet Explorer）、Ethereal、アンチウィルスソフト☆4（Norton AntiVirus）のインストーラである。これらに対し既存のビヘイビアブロッキング法で規定されている「ワームらしい振る舞い」と本方式による検知を行って、その結果を比較した。

本実験では、既存のビヘイビアブロッキング法で規定されている「ワームらしい振る舞い」としては2.1節で挙げた挙動のうち代表的な以下の5つを採り上げた。また、本方式としてはマルウェアの検知実験において検知率が高かった部分データ比較法を採用した。

- タイプ1：OS起動時の自動実行
- タイプ2：起動直後のファイルコピー
- タイプ3：別プロセスの起動
- タイプ4：トラヒック量の異常変動
- タイプ5：CPU利用率の上昇

実験の結果を表7に示す。表7中、○はマルウェアが検知できたことを示し、×は正規プログラムをマルウェアとして誤検知しなかったことを示す。また、「検知漏れ」は、マルウェアの一部が検知できなかったことを示し、「誤検知」は、正規アプリケーションをマルウェアとして誤検知してしまったことを表す。

☆4 アンチウィルスソフトも常駐型アプリケーションのカテゴリに含まれるが、ここでは特に個別に評価した。

表 7　誤検知および検知漏れの実験結果

Table 7　Experiment results.

	マルウェアらしい振る舞い(監視対象)					
	タイプ 1	タイプ 2	タイプ 3	タイプ 4	タイプ 5	本方式
インストーラ (常駐型)	誤検知	誤検知	×	×	×	×
メーラ	×	×	×	×	×	誤検知
FTPクライアント	×	×	×	誤検知	×	×
ブラウザ	×	×	誤検知	×	×	×
Ethereal	×	×	×	×	誤検知	×
インストーラ (アンチウィルス)	誤検知	誤検知	誤検知	誤検知	×	×
マルウェア	○	○	検知漏れ	検知漏れ	検知漏れ	○

表 7 から分かるように、タイプ 1 とタイプ 2 では、検査対象としたマルウェアの検知は行えたが、レジストリの変更やシステムファイルへの書き込みを行う常駐型のインストーラやアンチウィルスソフトをマルウェアとして誤検知した。タイプ 3 では、自ら子プロセスを生成するブラウザやアンチウィルスソフトをマルウェアとして誤検知するとともに、一部のマルウェアでは他プロセスの生成を行わずに感染行為を行うものがあり、そのようなマルウェアは検知できなかった。タイプ 4 では、FTP クライアントにおける FTP 通信時にトラフィックが上昇ししたため、これをマルウェアとして誤検知した。ネットワークを利用するアプリケーションは、実行時に少なからずトラフィックが上昇する[☆5]ため、どの程度トラフィックが変動したら異常とみなすかの閾値設定が困難である。また、一部のマルウェアでは転送ファイルサイズが大きくなかったため検知できなかった。タイプ 5 では、多大な計算処理を行う Ethereal 実行時に誤検知が発生した。タイプ 4 と同様に、どの程度 CPU 使用率が上昇したら異常とみなすかの閾値設定が難しい。また、一部のマルウェアは CPU をそれほど消費しなかったため検知できなかった。

☆5 今回の実験では、メーラにおいては、送受信を行ったメールの本数やファイルサイズが多大ではなかったため、誤検知には至らなかった。

これに対し、本方式では、利用者が自分宛に届いたメールをそのまま他者に転送するケースをマルウェアとして誤検知した。

ラッパーDLL が行う送受信ファイル間の相関チェックには相応のオーバヘッドが見込まれる。そこで、ws2_32.dll のラッパーDLL を用いた際のエンドユーザの PC における処理時間のオーバヘッドを測定した。エンドユーザの PC のスペックは、OS：Windows 2000、CPU：Pentium4 2.8GHz、メモリ：512MB である。

今回は、ブラウザ(Mozilla Firefox 1.0)によるネットサーフィンおよびメーラ(Becky! Internet Mail 2) によるメールの送受信を行う際のオーバヘッドを測定した。まず、ある利用者にエンドユーザの PC にて、およそ 2 時間に渡って約 300 サイトへのネットサーフィンと、約 10 通のメールの受信を行ってもらった。その間に受信バッファに格納されたファイル数は 445 個（18,911,428 バイト）であり、1 秒当たり約 2626 バイトの受信量であった。そしてその上で、任意のサイトに Web アクセスをし、134,735 バイトのファイルを受信した際のオーバヘッドと、2,129,838 バイト（Base64 エンコード後は 2,914,516 バイト）のファイルを添付したメールを送受信した際のオーバヘッドを測定した。その結果を表 8 に示す。表 8 中の各項の意味は以下の通りである。

- データ受信時：ファイルを受信してから、ファイルのハッシュ値集合／部分データ集合を受信バッファに格納するまでの時間（ms）
- データ送信時：ファイルを送信する際に、受信バッファに格納されているすべてのハッシュ値集合／部分データ集合との比較を行うのに要した時間（ms）
- ファイル数：受信したファイル数
- データ数：受信バッファに格納されたハッシュ値または部分データの総数

表 8　オーバヘッドの測定結果

Table 8　Overheads

	データ受信時 (ms)		データ送信時 (ms)		ファイル数	データ数
	ブラウザ	メーラ	ブラウザ	メーラ		
分割ハッシュ比較	47	264	93	940	445	18697
部分データ比較	47	217	125	1880		

3.2.3 考察

(1) 提案方式の検知精度

表6に示した検知実験の結果によれば、分割ハッシュ比較法による比較を行うだけで検査対象としたウィルスはすべて検知することができた。ワームである Blaster.C、Sasser.C は、先頭数バイトに通信用制御コードが含まれており、それ以降がワーム本体となっていた。よって、分割ハッシュ比較法では検出できなかった。これに対し、部分データ比較法による比較では、ワームも含め検査対象としたすべてのマルウェアを検知することができた。また、従来ビヘイビアブロッキング法においては、閾値の設定が難しいという問題があったが、本方式では、類似度の判断閾値については熟慮せずに適度な値（90%）としてやれば十分な検知精度が得られた。これにより、マルウェアのネットワークを介した自己増殖をマルウェアらしい振る舞いとして検出する本方式の有効性が確認できた。

表7に示した従来方式との比較結果によれば、本方式は、他の振る舞いを監視する従来の方式と比べて、検知漏れが起こりにくく、かつ、誤検知が少ない方式であるということが分かる。

しかしながら、本方式では、メールの転送を誤検知するため、一般のオフィス業務のようにメールの転送が日常茶飯事的に発生するエンタープライズネットワーク環境においては、本方式の誤検知発生率は相当のものになることが予想される。このためこうした環境においては、本方式を既存の方式と併用することにより誤検知の発生を防止する必要があるだろう。表7の結果から、例えば、タイプ4の方式と本方式とを組み合わせて、両方の方式でアラートが上がった場合にマルウェアと判別する。単なるメールの転送であれば、ネットワークの負荷はそれほど急激には上昇しないため、たとえ本方式でアラートが上がったとしてもマルウェアとは見なされない。一方、感染行為の場合には、多数の PC への感染を行おうとするため同一の内容を含む送信データが増加し、ネットワーク負荷も上昇するため、両方の方式でアラートが上がり、マルウェアと判別することが可能となる。

以上のように、メールの転送が定常的に行われるようなエンタープライズネットワーク環境では本方式と既存方式を併用することで、メール転送を誤検知することなくマルウェアを検知することができる。

Slammer ワームの際に問題となった ATM ネットワークやオンライン予約サービスをはじめとする重要インフラなどに見られるような、エンタープライズネットワーク内のクライアント間でメール転送が生じないような環境下では、本方式だけの適用であっ

ても十分に有用である。

(2) 提案方式によるオーバヘッド

表8に示したオーバヘッドの測定結果によれば、分割ハッシュ比較法と部分データ比較法の比較方式によってオーバヘッドはやや異なるが、ブラウザを使用したネットサーフィンにおいては、最大で130ms程度のオーバヘッドであり、利用者の感じるストレスはほとんどないと思われる。メーラにおけるデータ送信においては、およそ2Mバイト（Base64エンコード後はおよそ3Mバイト）のメールに対して類似度の検査が行われるため、2秒程度のオーバヘッドが生じている。しかし、実効速度1Mbpsのネットワーク上で約3Mバイトのメールを送受信するのにおよそ23秒を要することを鑑みると、本方式を実装した場合のファイルの送信時間のオーバヘッドは約8%（$\frac{2秒}{23秒+2秒}$）に過ぎない。また、今回の実装はプロトタイプ段階に留まっており、今後、送受信データ比較のアルゴリズムの改善検討を行うことで更なる高速化が達成される余地は残されている。

以上より、本方式では、過去2時間程度の受信ファイルを保持して、データを送信する度に送信ファイルと受信ファイルの比較を行ったとしても、パフォーマンスの劣化はさほど高くはならないことが確認できた。

(3) 擬似定義ファイルによる蔓延防止の効果

3.2.1項(2)で述べたように、本方式では、LANの中の1台目のコンピュータにおける未知マルウェアの感染が検出された時点で、未知マルウェア自身を疑似定義ファイルとして他のコンピュータに配布してLAN内での蔓延を防止することが可能である。すなわち、LAN内の2台目以降のコンピュータにおいては、当該マルウェアの侵入時に、これをブロックすることができる。ここでは、擬似定義ファイルの有効性を検証する。

理想的には、LAN内の全てのコンピュータが本方式を実装し、送受信データの比較を常時行うことが望ましい。LAN内のすべてのコンピュータが本方式を実装していれば、未知マルウェアがLANの中の1台目に感染した時点でこれが検知され、他のコンピュータに擬似定義ファイルが配布されることとなる。すなわち、LAN内のコンピュータの総数をN、1台のコンピュータがマルウェア被害にあった場合の損害額の平均値をD円とすると、未知マルウェアに感染してしまうコンピュータはN台中の最初の1台のみであり、未知マルウェアによる被害額の期待値はLAN全体でD円に抑えられる。

しかしながら、本節(2)でも述べたように、本方式では常に送信データと受信データの類似を検査する必要があり、送受信データのサイズに応じて相応のオーバヘッドが発生する。このため、LAN 内に一世代前のコンピュータが存在しているような場合など、CPU パワーが低いコンピュータに本方式を実装することが躊躇されることもあり得る可能性がある。このような場合に有効であると考えられるのが、本方式の擬似定義ファイルのみを活用する方法である。すなわち、3.2.2 項の【擬似定義ファイルによるブロッキング】で示した機能のみをラッパーDLL に実装する。以降では、両者を区別するために、すべての送受信データの類似度検査を実装した方式を「送受信データ比較方式」、擬似定義ファイルの検査のみを実装した方式を「擬似定義ファイル方式」と呼び分けることにする。

ここでの被害額とは、簡単のため、そのコンピュータが提供するサービスがストップすることによるビジネス機会損失や復旧に要する費用などを包含したものを指すこととする。実際には、1 台でもエンタープライズネットワーク内でのコンピュータが感染すると顧客への賠償や企業イメージのダウンなどによる損失も発生するが、ここでは損害額を厳密に見積もることが目的ではなく、擬似定義ファイルを用いた未知マルウェア検出方式がコスト的に意味があるかどうかを検証することが目的であるので、モデルを簡単化することとする。

受信データを擬似定義ファイルと照合するという作業が発生するのは、あるコンピュータに未知マルウェアが侵入して他のコンピュータに擬似定義ファイルが配布されてからアンチウィルスベンダによって正規のウィルス定義ファイルがリリースされるまでの間に限られ、通常時には行われない。よって、送受信データ比較方式のオーバヘッドをC_1、擬似定義ファイル方式のオーバヘッドをC_2とした場合、

$$C_1 \gg C_2 \qquad (1)$$

$$C_2 \approx 0 \qquad (2)$$

が成立する。

この前提に基づき、以下では、 LAN 内に送受信データ比較方式を実装しているコンピュータと擬似定義ファイル方式を実装しているコンピュータが混在している場合を想定して、両方式の導入率とその効果に関して検討していく。

LAN 内で送受信データ比較方式を実装しているコンピュータの割合をx（すなわち、擬似定義ファイル方式を実装しているコンピュータの割合は（$1-x$））とすると、LAN 内で送受信データ比較方式を実装しているコンピュータの数n_1は

$$n_1 = xN \qquad (3)$$

であり、擬似定義ファイル方式を実装しているコンピュータの数n_2は

$$n_2 = (1-x)N \qquad (4)$$

となる。

擬似定義ファイル方式を実装しているn_2台のコンピュータは未知マルウェアを新たに発見する機能はないので、新種の未知マルウェアがLAN内に侵入した場合、当該マルウェアが送受信データ比較方式を実装しているn_1台のコンピュータの内のいずれかに感染した時点で、それが検出されることになる。すなわち、確率的には、当該マルウェアがLAN内でm台のコンピュータに感染した時点で、それが検知されると期待される。ここで、

$$m = \frac{N}{n_1} = \frac{1}{x} \qquad (5)$$

である。

未知マルウェアが送受信データ比較方式を実装しているコンピュータに発見された直後に、擬似定義ファイルが生成され、LAN内のすべてのコンピュータに配布される。よって、これ以降、LAN内のコンピュータが当該マルウェアに感染することはない。ここで、擬似定義ファイルの生成および配布は機械的に実行可能なため、それに要する時間は無視できるとすると、結局、この時点までにおいて当該マルウェアに感染したLAN内のコンピュータはm台であり、その損害額$f_1(x)$は次式となる。

$$f_1(x) = mD = \frac{D}{x} \qquad (6)$$

一方、LAN内のn_1台のコンピュータにおいては送信データと受信データを常時比較するためにC_1のオーバヘッドが、n_2台のコンピュータにおいては受信データと擬似定義ファイルを比較するためにC_2（≈ 0）のオーバヘッドが発生する。このオーバヘッドはコンピュータの作業効率の悪化として現れるため、これも一種の損害であると算定すると、その損害額$f_2(x)$はLAN全体で次式のように導かれる。

$$f_2(x) = n_1 C_1 + n_2 C_2 = NC_1 x + NC_2(1-x) \quad (7)$$

以上より、LAN内で送受信データ比較方式を実装するコンピュータの割合xについては、全体の損害

$$f(x) = af_1(x) + bf_2(x) \quad (8)$$

を最小にするように設定するのが最も効率的であるということが分かる。なお、式(8)のaとbは適当な加重係数である。

簡単のために、$a:b=1:1$として計算してみると、擬似定義ファイル方式のオーバヘッドC_2はほぼゼロであるとしたので、式(8)は

$$f(x) = \frac{D}{x} + NC_1 x \qquad (9)$$

となる。式(9)を解析すると、

$$f'(x) = -\frac{D}{x^2} + NC_1 = 0 \quad (10)$$

を満たす点で$f(x)$が最小値を得ることがわかる。 $0 \le x \le 1$であるので、その解は、

$$x = \sqrt{\frac{D}{NC_1}} \qquad (11)$$

である。

1台のコンピュータがマルウェアに感染した際の損害額や、送受信データ比較方式や擬似定義ファイル方式のオーバヘッドが関与する損害額を見積もることは、実際には非常に難しい問題だが、式(11)の結果は、以下のことを意味する。

- Case 1 : $C_1 >> D$のとき

$$x = \sqrt{\frac{D}{NC_1}} \approx 0$$

これは、「マルウェアに感染した際の損害額が小さければ、送受信データ比較方式を導入する必要はなく、すべてのコンピュータに擬似定義ファイル方式を実装すれば十分である」ということを意味する。ただし、この場合は擬似定義ファイルを生成するコンピュータが存在しないため、LANの外部から擬似定義ファイルの供給を受ける必要性がある。

- Case 2 : $C_1 << D$のとき

$$x = \sqrt{\frac{D}{NC_1}} \approx \infty$$

$0 \le x \le 1$より、$x = 1$である。従って、「マルウェアに感染した際の損害額が大きい場合には、相応のオーバヘッドがあろうとも、すべてのコンピュータに送受信データ比較方式による未知マルウェア検知を導入する必要がある」ということを意味する。

- Case 3 : $C_1 = D$のとき

$$x = \sqrt{\frac{D}{NC_1}} = \sqrt{\frac{1}{N}}$$

上記の式より、$n_1 = xN = \sqrt{N}$となる。また、$m = 1/x = \sqrt{N}$である。よって、送受信データ比較方式のオーバヘッド（本方式の導入コスト）とワームに感染した際の損害額が等価である場合には、N台中のコンピュータの中の$\sqrt{N}$台に送受信データ比較方式を導入すると効率がよい」ということを意味する。

$C_1 = D$のときを例に取ると、例えばLAN内のコンピュータが50台である場合、送受信データ比較方式を導入すべきコンピュータは7台であり、残り43台は擬似定義フ

ァイル方式でよい。そして、損害が生じるコンピュータ（擬似定義ファイルが生成されるまでに未知マルウェアに感染する可能性があるコンピュータ）は平均で 7 台となる。また、LAN 内のコンピュータが 1,000 台である場合であっても、送受信データ比較方式を導入すべきコンピュータは 32 台であり、残り 968 台は擬似定義ファイル方式でよい。そして、損害が生じるコンピュータは平均で 32 台となる。よって、マルウェア被害の際の損害額や本方式のオーバヘッドが関与する損失を非常におおまかに見積もった限りでは、大規模な LAN 上であっても、少ない台数のコンピュータに比較的オーバヘッドの大きい送受信データ比較方式を導入することで、効果的に未知マルウェア対策を展開することが可能であるという示唆が得られた。

3.3 まとめ

マルウェアのネットワークを介した自己増殖機能に着目し、送受信データ間の相関を見ることにより未知マルウェアを検知し蔓延防止を行う方式を提案した。検知率および処理コストを評価する基礎実験から、本方式の有効性が確認できた。

本方式では、ネットワークを介して「外部から受信したデータ」と「外部に送信したデータ」が類似しているか否かを測ることによって、未知マルウェアを検知する。従って、あるコンピュータで未知マルウェアが新たに発見された場合、そのマルウェア自身を擬似的な定義ファイルとして利用することができるため、エンタープライズネットワーク内で最初に感染したコンピュータを除き、他のコンピュータへのマルウェアの蔓延を防止することができる。また、3.2.3 項(3)での考察結果から、コスト的には、ネットワーク内のすべてのコンピュータに「送受信データ比較方式」を導入する必要はなく、大部分のコンピュータはそれほどコストのかからない「擬似定義ファイル方式」を導入すれば十分であることが確認できた。

本方式は、Winsock API をフックすることによりエンドユーザのコンピュータにおける送受信データを常時監視するという実装が可能であり、ウィルスおよびワームの両方の未知マルウェアのリアルタイムな検出が可能である。

さらに、本方式はコンピュータからの送信データと受信データとの相関をチェックする方式であるため、自コンピュータ内での監視ではなく、ネットワーク内につながった他のコンピュータ（例えば、専用のネットワーク監視装置等）によって外部から送受信データを監視するようにすることも可能である。

以上述べてきたように、本章で提案した「送受信データ間の相関に基づくマルウェア検知方式」を 2 章で提案した「自己ファイル READ の検出によるマルウェアの検知方式」と併用することにより、2.4 節で述べた、

- コンピュータがマルウェアに感染した状態でのクライアント検知の困難さ
- リアルタイム検知に伴うコスト増

の 2 つの課題に対処することができる。さらに、両方式の併用にあたっては、「自己ファイル READ の検出によるマルウェアの検知方式」でマルウェアが検知された際に、そのマルウェアから「送受信データ間の相関に基づく蔓延防止方式」での擬似定義ファイルを作成して蔓延防止に利用することが可能となる。従って、「送受信データ間の相関に基づく蔓延防止方式」を「自己ファイル READ の検出によるマルウェアの検知方式」と併用する際にも、エンタープライズネットワーク中の一部のコンピュータにのみ「自己ファイル READ の検出によるマルウェアの検知方式」を実装し、それ以外のコンピュータにおいては「疑似定義ファイル方式」によってマルウェアの感染を予防する、

という形態を採用することによって運用コストを削減することもできる。

以上、2章で提案した、「自己ファイルREADの検出によるマルウェアの検知方式」と、本章で提案した「送受信データ間の相関に基づく蔓延防止方式」の2つを組み合わせることで、システム全体としてコンピュータシステムを未知マルウェアから守るという目的が達成できることとなる。

第 4 章
動的 API 検査方式によるキーロガーの検知方式

4.1 従来技術における課題

4.1.1 対象とするトロイの木馬

トロイの木馬に分類されるマルウェアで代表的なものを以下に記す[77][78]。

① アドウェア
利用者が予期しない広告や求めていない広告などを利用者に強制的に表示する。

② トラッキングクッキー
Web サイトの閲覧 URL 履歴などの情報を収集する。

③ キーロガー
利用者がキーボードから入力した情報を取得する。

④ ブラウザハイジャッカー
スタートページや検索ページなどのクライアントブラウザの設定を変更する。

⑤ ダイアラー
アダルトサイトなどの有料または特定の番号に強制的に接続する。

⑥ ボット
バックドアを作成して、外部からの指示に従った動作を強制的に行わせる。

これらのトロイの木馬のうち、アドウェア、トラッキングクッキー、ブラウザハイジャッカーは、攻撃者にとって実効的な利益がそれほど大きくないことから、最近ではあまり見かけなくなってきている。また、最近の利用者のインターネットへの接続は、ダイアルアップで行われることはほとんど少なくなり、常時接続が一般的となってきたことから、ダイアラーによる影響もほとんど見られなくなってきた。これに対し、キーロガーは、近年オンラインバンキングの不正送金事件[79]など経済的な被害を伴う深刻な問題となっている[80]。最近のキーロガーによる主な犯罪事例には以下のようなものがある。

- キーロガーをインターネットカフェのコンピュータに仕込み、オンラインバンキ

ングの ID とパスワードを取得して、現金 1,600 万円を不正に送金（2002.9）

- キーロガーをインターネットカフェのコンピュータに仕込み、オンラインバンキングの ID とパスワード、さらにはネットオークションの ID とパスワードを取得して、現金 36 万円を不正に送金（2004.7）
- 苦情メールを装ってキーロガーを送付し、オンラインバンキングの ID とパスワードを入手して、金融機関 4 行から計 1,140 万円を不正に送金（2005.7）
- キーロガーを組み込んだ CD-ROM を銀行の郵便物を装って送付し、キーロガーをインストールさせ、オンラインバンキングの ID とパスワードを入手して、現金数 100 万円を不正に送金（2005.10）

このように、犯罪件数の増加もさることながら、相当な金額の金銭的な被害が生じていることが大きな問題となっている。

最近のキーロガーの被害が急速に拡大している原因の一つとして、トロイの木馬のようなスパイウェアに対する利用者の認識の低さが挙げられる。キーロガーがインストールされている可能性のある環境（インターネットカフェ等）でのオンラインバンキングやクレジットカードによるオンラインショッピングの利用により、キーロガーの被害に遭うケースが後を絶たない。さらにスパイウェアは、ウィルスやワームのように感染活動を行わず、利用者に気付かれないように行動するという特徴を持つ。そのため利用者は自分のコンピュータがキーロガーに感染していることにすら気付かず、それがキーロガーによる被害を拡大する要因ともなっている。

米国Webroot社の調査によると、2006年第1四半期時点で、一般消費者のPCの87%がスパイウェアに感染しており、2005年第4四半期から18%も増加していると報告されている[81]。また、PC1台あたりにインストールされている平均のスパイウェア数も2006年第1四半期時点で29.5個という驚異的な数字となっている。さらに、McAfeeの調査によっても、2004年1月から2006年5月のおよそ2年間にキーロガーの数は2.5倍に増えており、Anti-Phishing Working Groupに報告されているキーロガーによるアラート数は100倍にも増加している[82]。こうした状況から、トロイの木馬のようなスパイウェア、特にキーロガーに対する有効な対策技術の開発が社会的にも大きく望まれてきている。

これに対して、ボットも、DoS（Denial of Service）攻撃の踏み台として利用されることで、オンラインショップのサービス停止などの大きな社会的問題を引き起こしている。ボットについては、1.2 節で述べたように、ボットが仕掛けられたコンピュータを 1 万台を超えるような世界的規模でネットワーク化し、攻撃者の指示に従って組織的かつ極めて大規模な攻撃を行うという、ボットネットと呼ばれる犯罪形態に移行してきている。ボットネットによる主な攻撃としては、以下のようなものがある[83][84]。

- DDoS（Distributed DoS）攻撃
  特定サイトに一斉にアクセスすることで、当該サイトのサービスを停止させる、あるいはネットワークの帯域を占有して他のサービスが行えないようにしてしまう攻撃。競業企業による意図的な攻撃も最近では出現している。
- スパムメール送信
  人を不快にさせるようなメールや宣伝メールをボットネットを通じて大量にばら撒く。また、最近社会問題化しているフィッシングメールをボットネットを通じてばら撒くことも行われている。
- トラフィック盗聴
  感染している端末から送信されるユーザ ID やパスワードなどの個人に関わる重要情報を盗み取る。
- Google AdSense の悪用
  Google の提供する広告に対して、ボットネットを利用して多数のクリックを行わせることにより金銭を取得する。
- オンライン投票・ゲームの操作
  異なる大量のクライアントから意図的な投票をさせることにより、投票結果を恣意的に変える。

犯行が組織的に行われ、かつその影響範囲が全世界的に及ぶこともあって、ボットネットに対しては最近様々な対策技術が考案されている。例えば、ボットは、通常以下の動作を踏む。

① 添付ファイルや脆弱性を突いた侵入、あるいは CD-ROM の郵送などを通じてクライアント端末に微小なマルウェアを感染させる。
② マルウェアに感染した端末は、自ら、FTP などのファイル転送プロトコルや HTTP などの通常使われるプロトコルを用いてボットプログラムの本体をボットを配布するサイトからダウンロードする。
③ ボットプログラムは、攻撃者からの指示を受け取るために IRC（Internet Relay Chat）[85]サーバとの接続を確立する。その後、指示が来るまで自らを隠蔽し、利用者から見つからないようにする。
④ IRC サーバから指示を受け取ると、ボットプログラムが自動的に起動し、DoS 攻撃等の指示された内容の動作を開始する。

そこで、守るべきエンタープライズネットワーク内に IRC 接続要求をトラップするためのハニーポットを設置し、クライアントが IRC 接続を行おうとした時にはクライアントに対してそれを許可してよいか必ず問い合わせを行うことで、ボットによる乗っ取りを防止する[83]。さらに、ボットプログラムの本体をダウンロードするアクセス先として既に分かっているサイトをブラックリストとして登録しておき、そうしたサイト

へのアクセスが発生した場合にはこれをブロックするという“Malware Block List”[86]という仕組みもできてきている。ただし、こうしたサイトは数日のうちに変更される可能性があるため、リストの更新は頻繁に行っていく必要がある。

また、本来 IRC は人間と人間がインターネット上で簡単に会話（チャット）するために使われるものであることから、ボットのようにプログラムが自動的に会話しようとすることを防ぐために、IRC サーバへの接続時に、人間にしか分からない認証方式を使うことでこれを防ぐという方法も有効である。例えば、カーネギーメロン大学の CAPTCHA（Completely Automated Public Turing to Tell Computer and Human Apart）プロジェクトでは、人間にしか分からないように画面上に歪んだ文字や一部隠された文字など表示して、それを読み取って入力してもらうことで人間と機械とを識別するという方式を用いている[87]。

このように、ボットに関しては様々な対策技術が考案されて、その有効性を示しつつある。一方、ボットと同様に深刻な問題を引き起こすキーロガーについては、一部対策技術を組み込んだ製品も出てきているが、後述するように、検知精度はあまり高くないのが実情である[88]。

以上述べてきた状況から、本論文では、トロイの木馬に対する検知技術として、キーロガーを対象に検討を行うこととする。

4.1.2 従来方式の限界

現在のキーロガー（をはじめとしたトロイの木馬全般）の検知方式の主流はパターンマッチング法である。パターンマッチング法は、2.1 節で述べたウィルスやワームへの対策技術と同様に、キーロガーの特徴的なコード部分をパターンとして定義ファイルに予め登録しておき、検査対象となるプログラムと比較することでキーロガーの検知を行う方式である[91]。この方式は、ウィルスやワームと同様に、定義ファイルに特徴的なコード部分が登録されている既知のキーロガーに対しては確実かつ容易に検知することができるが、未知のキーロガーを検知することができないという本質的な問題が残る。さらに、たとえ既知のキーロガーであっても、いわゆるゼロデイアタックのような攻撃に対しては対処できない。

また、ウィルスやワームの場合であれば、広範囲に感染活動を行うため比較的検体が入手しやすく、アンチウィルスベンダもウィルスやワームの定義ファイルを比較的容易に作製することができる。そのため、ウィルスやワームの対策とパターンマッチング法

は（完全な検知ができないとはいえ）比較的相性が良いともいえる。事実、情報処理推進機構（IPA）の調査では、ウィルス届出件数はここ数年増加傾向にあるものの、実害のあったケースが毎年減少してきている理由を、ウィルス対策ソフトウェアの導入の増加による効果であると分析している[92]。

しかしながら、トロイの木馬の場合には、ウィルスやワームとは異なり、感染活動を行わないため、定義ファイルを生成するための検体を入手することが困難であり[93]、パターンマッチング法の効果はウィルスやワームほどには期待できない。特にキーロガーはその目的から、不特定多数を幅広く攻撃するのではなく特定のユーザやグループを狙う場合も多く、その場合、検体の入手そのものがきわめて困難である。

そこで、未知のウィルスやワームに対して最も検知精度が高い方式であるビヘイビアブロッキング法を、キーロガーなどのトロイの木馬の検知に適用できないかという研究が最近になって始まっている。4.1.1 項で述べたように、トロイの木馬の場合には攻撃者の意図がある程度明確である。攻撃者の意図とは、例えば、キーロガーであれば、キーボードから入力された情報を盗むことであり、ダイアラーであれば、攻撃者の意図するサイトに強制的に接続することであり、アドウェアであれば、攻撃者の意図する表示を利用者のコンピュータの画面に表示することである。すなわち、トロイの木馬の場合、その種類ごとに個別に見てやれば、ウィルスやワームと比べて、マルウェアらしい振る舞いがより特定されていると言える。ビヘイビアブロッキング法の場合は、マルウェアらしい振る舞いが特定できるかという点が鍵となることから、攻撃者の意図が多様であるウィルスやワームに比べて、ビヘイビアブロッキング法はトロイの木馬の検知に対して、より相性の良い方式であると考えられる。

こうした背景から、最近になって、ビヘイビアブロッキング法に基づいて、キーロガーの挙動を捉えることにより、キーロガーらしい振舞いをするプログラムを検出する方法が提案され、商用に供せられるようになってきた[94][95][96]。こうした方法であれば、パターンマッチング法では検知することができない未知キーロガーに対しても対処が可能である。

しかし、筆者らの調べた限り、現在までに「真にキーロガーらしい振る舞い」の定義を明確に示した文献や製品は見当らない。実際、未知キーロガー検知機能を有するとされる商用製品である Keylogger Hunter [94] 、Keylogger Stopper [95]、Anti-Keylogger [96]を用いて各種キーロガーの検知を実施したところ、表 9 の結果が得られた。表 9 において、○は当該キーロガーが検知されたことを、×は検知されなかったことを示す。また、△は、本来検知すべき Casper のプロセスではなく、それを起動した explorer.exe のプロセスがキーロガーであると誤検知されたことを示す。なお、表 9 では、参考までにパターンマッチング法に基づいたトロイの木馬の検知ソフトウェアの代表である

Spybot – Search & Destroy 1.4[97]（2007.3.14 現在の定義ファイル）を用いた検知実験結果についても追記した。

表 9　既存製品のキーロガー検知実験

Table 9　Experimental results for keylogger detection with the existing products

検査対象キーロガー	パターンマッチング法	ビヘイビアブロッキング法		
	Spybot	Keylogger Hunter 2.1	Keylogger Stopper 2.0	Anti-Keylogger Elite 3.3.3
SpyAnywhere [98]	○	○	○	○
Perfect Keylogger Lite [99]	○	○	○	○
Activity Logger [100]	○	○	○	○
XPCSpy [101]	×	○	○	○
きいろがぁ [102]	×	×	×	×
キーロガー[103]	×	×	×	×
Parasite [104]	×	○	○	○
WingKEY [105]	×	○	○	○
キーのログをとる者 [106]	×	×	×	×
Casper [107]	×	×	×	△

表 9 に示すように、商用の製品群においても「真にキーロガーらしい振る舞い」を完全に定式化できておらず、検知漏れが発生していることがわかる。特に、「きいろがぁ」、「キーロガー」、「キーのログをとる者」、「Casper」については、いずれの検知ソフトウェアによっても検知することはできなかった。

さらに、上記の製品群を用いて、いくつかの代表的なアプリケーションに対して、正規のプログラムを誤検知することがないか実験を行った結果を表 10 に示す。ここで、「誤検知」は正規プログラムをキーロガーとして誤検知してしまったことを、×はキーロガーとして誤検知はされなかったことを示す。

表 10　正規プログラムの誤検知実験

Table 10　Experimental results for false detection with the existing products

検査対象正規プログラム	種類	パターンマッチング法	ビヘイビアブロッキング法		
		Spybot	Keylogger Hunter 2.1	Keylogger Stopper 2.0	Anti-Keylogger Elite 3.3.3
Mozilla Firefox 2.0	WEBブラウザ	×	×	×	×
Microsoft Word 2000	ワードプロセッサ	×	×	×	×
Microsoft Excel 2000	表計算ソフト	×	×	×	×
Microsoft PowerPoint 2000	プレゼンツール	×	×	×	×
Internet Explorer 6	WEBブラウザ	×	×	×	×
Outlook Express 6	メーラ	×	×	×	×
Mozilla Thunderbird 1.5.0.7	メーラ	×	×	×	×
Orchis [108]	ランチャツール	×	誤検知	誤検知	×
Xkeymacs [109]	キーバインドツール	×	誤検知	誤検知	誤検知
AltIME [110]	キーバインドツール	×	誤検知	誤検知	誤検知

表 9 ならびに表 10 からわかるように、従来の既存キーロガー検知方式では、既知のキーロガーに対しても検知漏れが数多く発生しているとともに、正規のプログラムを誤検知してしまうという検知精度上多くの問題がある。従って、真にキーロガーらしい振る舞いをより精度よく定式化して検知することが必要となる。

また、API ベースのキーロガーの検知を目的として、全ての API コールのフック状況をチェックして検出するという方法も提案されているが[111]、全 API フックの状態をリアルタイムに監視することは相当のオーバヘッドを伴うことになるため、実用的に適用が難しい。実用的な方式とするためには、真にキーロガーらしい振る舞いを特定して、その振る舞いに特化した監視を行う必要がある。

4.2 真にキーロガーらしい振る舞いの規定

本節では、APIベースのキーロガーの振る舞いの定式化を試みる。

Windows 環境下において、API を利用してキーボード入力を取得する方法は、筆者らが確認した限りでは二つ存在する。キー判定 API を用いる方法と、フック API を用いる方法である。従って、これらの動作を検出できれば、キーロガーの完全な検知が可能となる。以下、これらの仕組みとその検出方法について述べる。

- キー判定 API を用いる方法

キー判定 API である GetAsyncKeyState は、呼び出し時に指定したキーが押されているか、また前回の呼び出し時以降に指定したキーが押されていたかどうかを判定する API である。GetAsyncKeyState は、単に指定したキーが押されているか否かを判定する API であるため、任意のプロセスへのキーボード入力を取得することが可能である。

キーロガーがキー判定 API を用いてキーボード入力を取得する場合、ID やパスワードなどを盗み出すために、少なくとも A～Z と 0～9 のキーボード入力を取得しなければならない。またキーロガーは、キーロガー自身のプロセスへのキーボード入力だけでなく、他プロセスへのキーボード入力も取得する。そのため、他プロセスがキーボードフォーカスを所持している場合でも、キーロガーは GetAsyncKeyState を用いてキーボード入力を取得できなければならない。

そこで本方式では、キー判定 API を用いたキーロガーの振る舞いを以下のように規定する。

【挙動 1：キー判定 API に関する振る舞い】

他プロセスがキーボードフォーカスを所持しているときにも、GetAsyncKeyState によって A～Z、0～9 の全てのキーボード入力を取得する。

- グローバルフック API を用いる方法

フックとは、Windows OS が各プロセスへ送信するメッセージを SetWindowsHookEx という API を用いて取得する方法であり、自身のプロセスへのメッセージのみを取得するローカルフックと、全プロセスへのメッセージを取得することが可能なグローバルフックが存在する[112]。

例えばキーロガーがフックを用いて Web ブラウザ上で利用者により入力される ID やパスワードを盗むためには、Web ブラウザへのキーボード入力メッセージを取得する必要がある。キーロガーから見て Web ブラウザは他プロセスであるため、キーロガーは必然的にグローバルフックを用いることになる。なお、フックには様々なタイプが

存在し、その中にはキーボード入力メッセージをフックできないタイプも存在する。キーロガーは、他プロセスへのキーボード入力メッセージを取得することが目的であるため、キーボード入力メッセージをフックできるフックタイプを指定する必要がある。

そこで本方式では、フックを用いるキーロガーの振る舞いを以下のように規定する。

【挙動 2：グローバルフック API に関する振る舞い】

キーボード入力メッセージをフックできるフックタイプのいずれかを指定し、SetWindowsHookEx によるグローバルフックを用いてキーボード入力を取得する。

これまでは、キーロガー自らが他プロセスへのキーボード入力を取得する方法を説明した。しかしながら、キーロガーは、自身の存在をユーザに知られないように、他プロセスに寄生してキーロギングを行う場合もある。以下では、他プロセスに寄生してキーボード入力を取得する方法とその検知方法について説明する。

- 他プロセスに寄生する方法

Windows 環境下において、他プロセスに寄生する方法は、筆者らが確認した限りでは二つ存在する。CreateRemoteThread を用いる方法と SetWindowsHookEx を用いる方法である[113]。

CreateRemoteThread は、他プロセスのメモリ空間にスレッドを生成する API であり、この API を用いれば、例えばキーボード入力を取得するスレッドを他プロセスに生成することも可能である。

SetWindowsHookEx は前述の通り、フックを行うための API であるが、グローバルフックを行う場合には、フック先のプロセス全てにフックプロシージャ（フックしたメッセージを処理するルーチン）が含まれている DLL をアタッチする。よって、キーボード入力を取得するためのルーチンを、フックプロシージャを含む DLL に仕込むことにより、任意のプロセスにキーロガー機能を有する DLL をアタッチさせることが可能となる。

そこで本方式では、キーロガーが他プロセスに寄生するという振る舞いを以下のように規定する。

【挙動 3：寄生 API に関する振る舞い】

CreateRemoteThread または SetWindowsHookEx のグローバルフックを発行する。

- 寄生先プロセスにキーロガー行為を行わせる方法

一旦、他プロセスに寄生すれば、キーロガーはそのプロセスの一部として動作することが可能となるため、寄生したプロセスへのキーボード入力を取得することが OS によ

り許可される。また、寄生したプロセス以外のプロセスへのキーボード入力も取得したい場合は、寄生したプロセスに挙動1あるいは挙動2の二つの方法のいずれかを行わせることで、他のプロセスの利用者のキーボード入力を取得することが可能となる。具体的には、以下の方法によりキーボード入力を取得する。

① 寄生先プロセスへのキーボード入力を取得する方法

寄生したプロセスへのキーボード入力を取得する方法としては、GetKeyStateを用いる方法、GetKeyboardState を用いる方法、そして SetWindowsHookExによるローカルフックを用いる方法が挙げられる。GetKeyStateは自プロセスに対して、あるキーが押されているか否かを判定する API である。GetKeyboardState は自プロセスに対して、全てのキーの状態（押されているか否か）を取得できるAPIである。SetWindowsHookExによるローカルフックは、前述の通り、自プロセスへのメッセージのみを取得する方法である。

そこで本方式では、寄生先プロセスのキーボード入力を取得するキーロガーの振る舞いを以下のように規定する。

【挙動4:寄生先プロセスへのキーボード入力を取得するAPIに関する振る舞い】
寄生先プロセスが自プロセスへのキーボード入力を取得する。

② 寄生先プロセスにて他プロセスへのキーボード入力を取得する方法

寄生したプロセス以外のプロセスへのキーボード入力を取得するには、寄生先プロセスにおいて、挙動1あるいは挙動2で示したキー判定APIを用いる方法か、あるいはグローバルフック API を用いる方法を実行すればよい。

そこで本方式では、全プロセスへのキーボード入力の取得を寄生先プロセスに代行させるというキーロガーの振る舞いを以下のように規定する。

【挙動5：寄生先プロセスにて他プロセスへのキーボード入力を取得するAPIに関する振る舞い】
寄生先プロセスが挙動1もしくは挙動2の行動を示す。

- 寄生を繰り返す方法

他プロセスへの寄生は多重に行うことも可能である。巧妙な攻撃者は、「寄生先プロセスにさらに別のプロセスへの寄生行為を行わせ、そのプロセスにキーロガー行為を行わせる」という方法を必要な回数だけ繰り返すことも考えられる。このような挙動を検出するためには、検査対象プログラムにおける寄生行為を再帰的に検査していく必要が

ある。

そこで本方式では、寄生行為を繰り返すというキーロガーの振る舞いを以下のように規定する。

【挙動 6：多重寄生に関する振る舞い】

寄生先プロセスが挙動 3 の行動を示す。

以上のように規定した挙動1から挙動6に基づき真にキーロガーらしい振る舞いの定式化を行う。

挙動 5 は、キーロガーが寄生活動（挙動 3）を行った際における挙動 1 および挙動 2 の再帰検査である。また、挙動 6 が見受けられた際には、寄生先プロセスを基点として更に挙動 3～挙動 6 の検査を繰り返す必要がある。これらに注意すると、挙動 1～挙動 6 に基づくキーロガーの検査は図 13 のフローチャートのように定式化することができる。

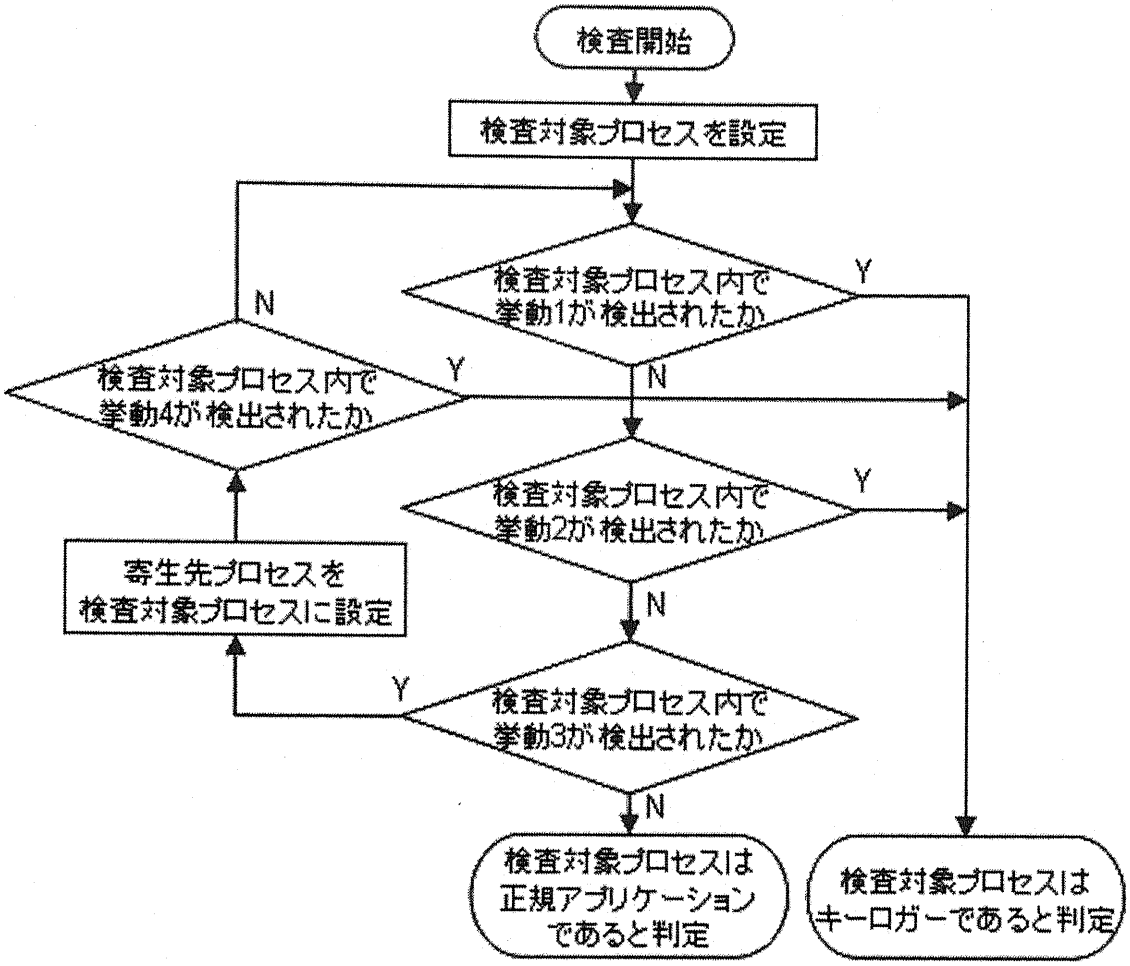

図 13　キーロガー検知フロー

Fig. 13　Flow of keylogger detection

4.3 提案方式

提案方式では、キーロガーによる「キー判定 API の利用」、「グローバルフック API の利用」、ならびに「他プロセスにキーロガー行為を行わせる」ことをビヘイビアブロッキング法におけるキーロガーらしい振る舞いとして利用する。具体的には、OS（Windows）の提供する API である、GetAsyncKeyState、SetWindowsHookEx、CreateRemoteThread のそれぞれをエクスポートしている DLL を改造して、エンドユーザのコンピュータにおける全プロセスのキー入力の取得行為を監視し、キー入力取得行為を行ったプロセスをリアルタイムで検知する。さらに、CreateRemoteThread あるいはSetWindowsHookExによって他プロセスに寄生する行為が検出できた場合には、寄生先プロセスにおいて GetKeyState、GetKeyboadState、あるいは SetWindowsHookEx が発行されたかどうかを、それぞれの API をエクスポートしている DLL を改造してチェックルーチンを組み込むことにより検知する。よって、キーロガーが活動を開始した瞬間にこれを検出することが可能となる。このように、キーロガーに関連する DLL を変更することにより、当該 API の発行を動的に監視し、キーロガーの検知を行うことから、本提案方式を、「動的 API 検査方式によるキーロガー検知方式」と呼ぶことにする。

4.3.1 検証実験

本方式の有効性を検証するため、検知システムを実装して基礎的な実験を行った。

(1) 検証の要件

ビヘイビアブロッキング法に基づくキーロガーの検知方式の要件としては以下のものが挙げられる。

【機能要件】

① 図 13 のフローチャートに従って、他プロセスにキーロガー行為を行わせるものも含め、キーロガーの検知が可能であること

② 正規プロセスを誤検知しないこと
ビヘイビアブロッキング法には、本質的に、誤検知が多いという問題があるため、提案方式においては誤検知の発生が十分に低いことが求められる

【実装要件】

③ 検査対象となるプログラムに対する改変は不要であること

④ リアルタイムにキーロガーの検知が可能であること

⑤ 検査対象となるプログラムに対して、OS の起動直後からの検知が可能であること

実装要件の③、④、⑤も実用的な検知方式という観点からは基本的に備えておかなければならない要件である。機能要件の②については、必須の要件である。機能要件の①に関しては、本論文では、基本的なコンセプトの提案に重点を置くこととして、実装の手間を鑑み、挙動 4 と挙動 6 の検出処理に関しては実装を割愛することとした。このため、検証実験での実装におけるキーロガー検知フローは、以下の挙動 A～C の検出となる。

【挙動 A】

他プロセスがキーボードフォーカスを所持しているときにも、GetAsyncKeyState によって A～Z、0～9 の全てのキーボード入力を取得する（4.2 節の挙動 1)。

【挙動 B】

キーボード入力メッセージをフックできるフックタイプのいずれかを指定し、SetWindowsHookEx によるグローバルフックを用いてキーボード入力を取得する（4.2 節の挙動 2)

【挙動 C】

CreateRemoteThread、SetWindowsHookEx を用いて他プロセスに寄生した（4.2 節の挙動 3）後、寄生先プロセスが挙動 1 もしくは挙動 2 を行う（4.2 節の挙動 5)

(2) 検証方法

本実装では、本節(1)で示した要件を満たす方式として、Microsoft Research の提供するライブラリである Detours[114]を用いた。

Detours は、x86 マシン上で任意の Win32 関数にインターセプト用のコードを挿入するための API フックライブラリ関数群である[115]。API α をインターセプトするにあたり、Detours はメモリ上の API α の先頭アドレスに、インターセプト用コードの先頭アドレスへの無条件ジャンプ命令を挿入する。プログラムの中で API α が呼び出されると、無条件ジャンプ命令によって制御がインターセプト用コードに移行するため、API α をインターセプトすることが可能となる。インターセプト用コードの実行が終了すると、API α に制御が渡され、API の本来の動作が実行される。

本論文では、Detours を用いて挙動 A～C に関連する 3 つの API（GetAsyncKeyState、SetWindowsHookEx、CreateRemoteThread）をインターセプトし、それぞれの API においてキーロガーらしい挙動（挙動 A～C）が検出されるか否かを監視することでキ

ーロガーを検知するための検査用 DLL を実装する。検査用 DLL の開発環境には Microsoft Visual C++ 6.0 を用いた。

図 14 が、Detours ライブラリを用いることにより作成したキーロガー検査用 DLL である。ここで、DetourAttach (A, B)は Detours のライブラリ関数であり、第一引数 A に指定された API をインターセプトし、第 2 引数 B に指定されたインターセプト用のコードを挿入する。検査用 DLL は、自身がプロセスにロードされた際に、関数 DetourAttach (A, B)を実行することにより GetAsyncKeyState、SetWindowsHookEx、CreateRemoteThread の 3 つの API をインターセプトする。また検査用 DLL は、自身がプロセスからアンロードされた場合に、インターセプトを自動的に解除するように

```
検査用 DLL　{
        if （DLL_load）{                          // DLL ロード時
                DetourAttach ( Real_GetAsyncKeyState,
                                Mine_GetAsyncKeyState ) ;
                DetourAttach ( Real_SetWindowsHookEx,
                                Mine_SetWindowsHookEx ) ;
                DetourAttach ( Real_CreatRemoteThread,
                                Mine_CreatRemoteThread ) ;
                DetourAttach ( Real_CreatProcess,
                                Mine_CreatProcess ) ;
        }       else if （DLL_unload）{          // DLL アンロード時
                DetourDetach ( Real_GetAsyncKeyState,
                                Mine_GetAsyncKeyState ) ;
                DetourDetach ( Real_SetWindowsHookEx,
                                Mine_SetWindowsHookEx ) ;
                DetourDetach ( Real_CreatRemoteThread,
                                Mine_CreatRemoteThread ) ;
                DetourDetach ( Real_CreatProcess,
                                Mine_CreatProcess ) ;
        }
}
```

図 14　検査用 DLL

Fig. 14　DLL for inspection

実装されている。インターセプトの解除には、Detours のライブラリ関数である DetourDetach (A, B)を用いた。DetourDetach (A, B)は、第 1 引数 A に指定された API から、第 2 引数 B に指定されたインターセプト用のコードを取り除く関数である。

そして図 15 が各 API のインターセプト用コードである。各 API のインターセプト用コードは、挙動 A～C のいずれかを満たす場合にアラートが発生するように実装されている。また、処理を正常に継続させるために、各インターセプト用コードの最後で本物の当該 API を呼び出している。

```
Mine_GetAsyncKeyState ( 引数 ) {
        if ( 他プロセスがキーボードフォーカスを所持 ) {
                if ( 引数 ==A～Z, 0～9 )
                        alert ;
        }
        return Real_GetAsyncKeyState ( 引数 ) ;
}

Mine_SetWindowsHookEx ( 引数 ) {
        if ( 引数 == グローバルフック) {
                if ( キーボード入力メッセージ取得用フックタイプ )
                        alert ;
                else
                        // 寄生先プロセスに DLL をアタッチ
                        AttachParasiteProcess(検査用 DLL);
        }
        return Real_SetWindowsHookEx ( 引数 ) ;
}

Mine_CreateRemotoThread ( 引数 ) {
        // 寄生先プロセスに DLL をアタッチ
        AttachParasiteProcess(検査用 DLL);
        return Real_CreateRemotoThread ( 引数 ) ;
}
```

図 15　各 API のインターセプト用コード

Fig. 15　Interception code for each API

上記の検査用 DLL を検査対象プロセスにロードさせれば、これら 3 つの API の挙動が監視されるため、対象プロセスがキーロガーか否かを検査することが可能となる。なお、SetWindowsHookEx と CreateRemoteThread の発生は他プロセスへの寄生（の可能性があること）を意味するため、検査対象プロセスにてこれらの API が呼び出された場合には、寄生先プロセスに検査用 DLL を更にアタッチさせる必要がある。この機能を有する関数として AttachParasiteProcess 関数を自作した。

また、本節(1)⑤の要件に関しては、今回は、Windows の起動後にユーザのクリックにより実行される全てのアプリケーションに対して、アプリケーションの起動直後からのキーロガーの検知を行うという形態の実装とした。

ユーザのクリックにより実行されるアプリケーションは、Windows のファイルマネージャである explorer.exe が子プロセス生成用 API である CreateProcess を呼び出すことにより実行される。例えばユーザがデスクトップ上の Internet Explorer のアイコンをダブルクリックした場合には、explorer.exe は内部で CreateProcess を呼び出し、Internet Explorer を子プロセスとして生成することにより、Internet Explorer が実行されるという仕組みになっている。

そこで、CreateProcess に対しても Detours によるインターセプトを行い、explorer.exe が生成する子プロセスに対して、子プロセスが生成される瞬間に検査用 DLL をロードさせるようにする。CreateProcess のインターセプト用コードを図 16 に示す

```
Mine_CreateProcess ( 新規プロセス ) {
        return DetourCreateProcessWithDll ( 新規プロセス, 検査用 DLL ) ;
}
```

図 16　CreateProcess のインターセプト用コード

Fig. 16　Interception code for CreateProcess

ここで、DetourCreateProcessWithDll (A, B)は Detours のライブラリ関数であり、第 2 引数 B に指定された任意の DLL をロードさせた上で、第 1 引数 A に指定された新規プロセスを生成する。

キーロガーの検知にあたっては、Windows 起動時に、図 14～図 16 に示したキーロ

ガー検査用 DLL を explorer.exe にロードさせておけば、図 16 で説明した CreateProcess に関するインターセプトの機構によって、利用者によって起動される全てのプログラムに検査用 DLL を自動的にロードさせることが可能である。

キーロガー検査用 DLL を explorer.exe にロードさせるには、図 14 のインターセプトコードを利用して「Mine_CreateProcess (explorer.exe)の API コールを発行するプログラム」を作成しておき、一旦、explorer.exe を終了した上で、当該プログラムを走行させ、その後、explorer.exe を再起動すれば、キーロガー検査用 DLL がロードされた explore.exe が立ち上がることとなる。

この作業は、一般ユーザ権限でも実行可能である。また、利用者が各自のスタートアップメニューにこの作業を登録することで、本検知システムを Windows の起動時から常に起動させておくことも可能である。

更に、管理者権限を有する利用者であれば、この作業を Windows のレジストリの Run エントリに登録することもできる。例えばインターネットカフェの管理者が、インターネットカフェ内のパソコンに本検知システムを導入(検査用 DLL を explorer.exe にロードさせる作業をレジストリに登録) しておけば、インターネットカフェ利用者をキーロガーの脅威から自動的に保護することが可能となる。

以上の Windows 起動時処理を行っておけば、explorer.exe にキーロガー検査用 DLL がロードされるため、その後、利用者のクリックにより起動される全てのアプリケーションプログラムには、プログラム起動時に検査用 DLL が自動的にロードされる。検査用 DLL はアプリケーションプログラムを常時監視し、挙動 A～C が発生した場合には、その時点でアラートを上げる。

実装した検知システムを用いて、キーロガーの検知実験を行った。実験では、本節(1)で説明した挙動 A、B、C のそれぞれによってキーロガーが検知できるかどうかを調べた。また、本方式で規定した挙動の検出によって、正規のプログラムを誤検知するかどうかについても実験を行った。誤検知実験でも同様に、挙動 A、B、C のそれぞれについて、正規のプログラムが検知されてしまうかどうかを調べた。更に、実装した検知システムのオーバヘッドについても測定を行った。

(3) 実験結果

検知実験にあたっては、表 9 に示す 10 種類のキーロガーを用いた。未知のキーロガーは検体の入手が非常に困難であるため、実際には商用のキーロガーやインターネット上で公開されているキーロガーを対象とした。なお、本実験は、物理的に隔離されたネ

ットワーク内のWindows 2000 Professional SP4がインストール済みのPC上で行った。実験に使用したキーロガーとその実験結果を表11に示す。

表11において、○はそれぞれの挙動が検出できたことを示し、×は検出できなかったことを示す。検知結果の欄は、挙動A、挙動B、挙動Cのいずれかでキーロガーが検出されれば○が記され、本方式によってマルウェアが検知されたことを表す。

表11　キーロガー検知実験

Table 11　Experimental results for Keylogger detection

検査対象	挙動A	挙動B	挙動C	検知結果
SpyAnywhere	×	○	×	○
Perfect Keylogger Lite	×	○	×	○
Activity Logger	×	○	×	○
XPCSpy	×	○	×	○
きいろがぁ	○	×	×	○
キーロガー	○	×	×	○
Parasite	×	○	×	○
WingKEY	×	○	×	○
キーのログをとる者	○	×	×	○
Casper	×	×	○	○

正規のアプリケーションを用いて、検知システムの誤検知実験を行った。誤検知実験にあたっては、表10に示す10種類のアプリケーションを検査対象とした。アプリケーションとしては、Microsoft Office製品やWebブラウザ、メーラ等のアプリケーションに加え、誤検知の可能性のあるアプリケーションも重点的に選択した。しかしこれらのアプリケーションは多機能であり、全ての機能を網羅して実験を行うことは非常に困難である。そこで本実験では、基礎実験として各アプリケーションの基本機能（WEBブラウザであればブラウジング、メーラであればメール送受信などの機能）を中心に実行した場合に対してのみの挙動監視を行った。なお、本実験は、物理的に隔離したネットワーク内のWindows 2000 Professional SP4がインストール済みのPC上で行った。本実験に使用したアプリケーションとその結果を表12に示す。

表12において、×は正規プログラムに対して挙動A、B、Cを検出しなかったこと

を示し、○は対象となる挙動が検出されたことを示す。挙動 A、挙動 B、挙動 C のいずれかが検出されれば本方式によってマルウェアが検知されたことになる。表 11 と同様に、この結果を「誤検知結果」の欄に示した。×は本方式により正規プログラムがキーロガーがとして検知されなかったことを表し、「誤検知」は正規プログラムをキーロガーとして誤検知してしまったことを表す。

表 12　誤検知実験

Fig. 12　Experimental results for false detection

検査対象	種類	挙動A	挙動B	挙動C	誤検知結果
Mozilla Firefox 2.0	WEBブラウザ	×	×	×	×
Microsoft Word 2000	ワードプロセッサ	×	×	×	×
Microsoft Excel 2000	表計算ソフト	×	×	×	×
Microsoft PowerPoint 2000	プレゼンツール	×	×	×	×
Internet Explorer 6	WEBブラウザ	×	×	×	×
Outlook Express 6	メーラ	×	×	×	×
Mozilla Thunderbird 1.5.0.7	メーラ	×	×	×	×
Orchis	ランチャツール	×	○	×	誤検知
xkeymacs	キーバインドツール	×	○	×	誤検知
AltIME	キーバインドツール	×	○	×	誤検知

本検知システムによるオーバヘッドを測定した。オーバヘッドの測定に用いた実験環境は、Windows をインストールした直後の PC ではなく、ある程度の期間、利用者が実際に使用していた Windows PC を用いた。これは、測定環境をより実環境に近づけるためである。測定に用いた Windows PC のスペックは、OS：Windows 2000 Professional SP4、CPU：Athlon 900MHz、メモリ：128MB である。

ここでは、特に、監視対象プロセスにおいて挙動 A～C を検査するために要するオーバヘッドを測定する。オーバヘッドは、GetAsyncKeyState、SetWindowsHookEx、CreateRemoteThread の各 API に対して、オリジナルの API を呼び出した際の処理時間と、検査用 DLL 付き API を呼び出した際の処理時間の差として求めた。各 API に対して、それぞれを単独で 10,000 回呼び出したときの処理時間の平均とそのオーバヘッドを表 13 に示す。

表 13　キーロガーらしい挙動の検出に伴うオーバヘッド

Fig.13　Overhead for checking keylogger behavior

	オリジナル [μs]	検査用DLL付き [μs]	オーバヘッド [μs]
GetAsyncKeyState の呼び出し	2.9	15.8	12.9
SetWindowsHookEx の呼び出し	2.8	3489.4	3486.6
CreateRemoteThread の呼び出し	178.1	191.2	13.1

4.3.2 考察

(1) 提案方式の検知精度

表 11 から分かるように、今回の実験で用いた全てのキーロガーに関して、本論文にて規定したキーロガーらしい振る舞い（挙動 A、挙動 B、挙動 C の全て）を監視することにより確実に検知できることが確認された。特に、キーロガー検知のための商用製品でも検知できなかった「きいろがぁ」、「キーロガー」、「キーのログをとる者」といったキーロガーも挙動 A の監視により、そして、商用製品では誤検知となった「Casper」に対しても挙動 C の監視により、漏れなく検知することができた。

さらに、表 12 に示すように、基本的な機能を使用する限りの検査において、一部のアプリケーションを除いて正規のアプリケーションを本方式によって誤検知することはないことが確認された。xkeymacs、AltIME、Orchis の 3 つのアプリケーションについては、挙動 B が確認されたため、誤検知が発生した。これら 3 つのアプリケーションで誤検知が発生した理由は以下のように考えられる。

xkeymacs と AltIME は、グローバルフックを用いてキーボード入力を取得しキーバインドを置換するアプリケーションである。また、Orchis はランチャ（プログラムのショートカットの管理ツール）であり、特定のキーを押下した場合にランチャがアクティブ化する機能が実装されている。これらの機能を実現するにあたって、これらのアプリケーションでは、グローバルフックを用いてキーボード入力を取得していたため、本方式はこれらについてはキーロガーと誤検知した。

本実験において誤検知となった xkeymacs、AltIME、Orchis の 3 つのアプリケーションは、実際にキーロガーとしての機能を果たし得る可能性を有しており、悪用されれ

ばコンピュータからキー入力情報を取得できてしまう。従って、利用者に対して、警告の意味も込めて、その可能性を通知することは有用であるという考え方も当然成り立つが、このようなアプリケーションとキーロガーを切り分けるためには、更に詳細な「キーロガーらしい挙動」を規定する必要が生じる。キーロガーの場合には、取得したキー入力情報を悪用するため、クライアントコンピュータ外にその情報を持ち出す行為を必ず伴うはずである。こうした行為は、例えば、ネットワークを経由して取得したキー入力情報を犯罪者のコンピュータに送信したり、ユーザやシステムから隠蔽したログ情報として一旦クライアントコンピュータ内に蓄積しておいて、後日犯罪者が回収するなどによって行われる。キーバインドツールやランチャは、特定のキー入力をある機能に結びつけることが目的であり、入力されたキー情報を外部へ送信したり、蓄積しておくようなことは行わない。従って、こうしたキー入力情報を「外部に送信する」あるいは「ログ情報として蓄積する」といった動作を、キーロガーらしい挙動として更に規定すれば、誤検知を低減できると考えられる。

以上述べてきたように、提案方式は、従来のキーロガー検知方式に比べより高精度にキーロガーを検知することができ、また、利用者への警告の有意性を鑑みれば、誤検知もほとんど発生しない方式であると言える。

(2) 提案方式のオーバヘッド

表 13 に示すように、本方式によるオーバヘッドは、API 呼び出し 1 回に対して、最大でも約 3.5 ミリ秒とごくわずかであることが確認された。

CreateRemoteThread は、ひとたびスレッドを生成すれば、そのスレッドが終了するまでスレッドは実行され続けるため、CreateRemoteThread を何度も発行し続ける必要はない。よって、CreateRemoteThread の API コールが呼ばれる頻度は少ないことが一般的である。従って、表 13 に示したオーバヘッドであれば、実用上問題となることはほとんどないと考えられる。

GetAsyncKeyState は、API コールを発行し続けなければキーボード入力を取得し続けることができない。そのため GetAsyncKeyState の API コールが呼ばれる頻度は多くなるが、表 13 に示した程度のオーバヘッドであれば、同様に実用上の問題はほとんど生じないと考えられる。

SetWindowsHookEx は、ひとたびフックを開始すれば、そのフックを解除するまでフック処理が継続するため、CreateRemoteThread と同様に、呼ばれる頻度が比較的少ない API コールであるといえる。ただし、表 13 から、SetWindowsHookEx の API コールにおけるオーバヘッドは、他の API コールと比べ桁違いに大きいことが分かる。このため、一般的なアプリケーションにおいて SetWindowsHookEx の API コールが

どれくらいの頻度で発行されているのか調査を行った。調査に用いたアプリケーションは、4.3.1 項の実験にて誤検知となった xkeymacs と AltIME である。調査結果を図 17 に示す。

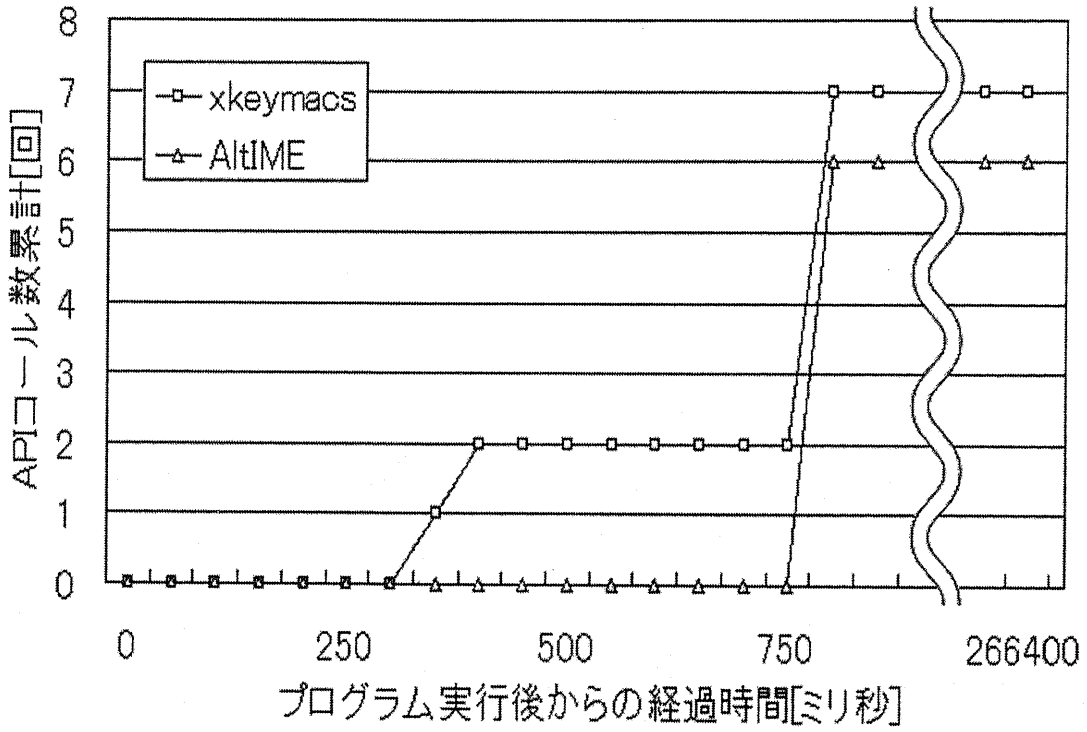

図 17　SetWindowsHookEx の API コールの発行数

Fig. 17　The number of API calls to SetWindowsHookEx

図 17 から分かるように、SetWindowsHookEx の API コールはプログラム起動時に集約されており、またその発行総数も 10 回以下と極めて少ないため、表 13 に示したオーバヘッドであれば、まったく問題とはならないことが確認できた。

4.4 まとめ

本章では、キーボード入力を取得する API を悪用してキーロギング行為を行うキーロガーに対して「キーロガーらしい挙動」を定式化し、これを利用したキーロガー検知方式を提案した。検知実験、誤検知実験およびオーバヘッド測定実験を通じ、本方式がキーロガー検知に有効であることを示した。

本方式はパターンマッチング方式に依らないキーロガー検知方式であるため、たとえ特定の相手を狙うようにカスタマイズされたキーロガーであっても、これを検知することが可能であると期待できる。

第5章
結論

5.1 総括

本論文では、これまで未知マルウェアの検知に最も効果があるとされているビヘイビアブロッキング方式の考え方に基づき、3つの新たなマルウェア検知方式を提案した。

ウィルスとワームに対しては、ウィルスとワームが行う「自己ファイル READ」という行為を、真にマルウェアらしい振る舞いとして定式化し、OSのファイルシステムにファイルアクセスを監視する仕組みを組み込むことによりウィルスとワームの検知を行う、「自己ファイルREADの検出によるマルウェアの検知方式」を提案した。本方式を用いれば、従来方式では検知が困難であったミューテーション型マルウェアや未知のマルウェアを精度良く検知することができる。また、従来のビヘイビアブロッキング方式では、閾値の制御が経験的にならざるを得ないために正規プログラムを誤検知する場合が多いという根本的な問題があったが、提案方式では、「マルウェアは他コンピュータへの感染などのため自分自身のファイルを READ して自己複製を行う」という行為を真にマルウェアらしい振る舞いとして厳密に定式化することによって、誤検知の問題を大きく解消することができた。本方式では、OSのファイルシステムのみの改造で実現可能なため、既存のアプリケーションプログラムには一切変更を加える必要がないという特長も持つ。

一方、ビヘイビアブロッキング法は「利用者のコンピュータが実際にマルウェアに感染した際に、その発病時の症状を検知する」という考え方に基づく検知方法である。マルウェアに感染した状態では検知システムが正しく動作しない恐れがある。実際いくつかのマルウェアでは、商用のアンチウィルスソフトウェアのリアルタイム監視動作を妨害して、マルウェアの検知を無効化するという動作を行うものも出現している。また、企業内のエンタープライズネットワーク内の全てのクライアントコンピュータ上で常時全てのファイルアクセスを監視することは相当のオーバヘッドになる。よって、「自己ファイル READ の検出によるマルウェアの検知方式」を全てのクライアントコンピュータにて運用することは、コンピュータシステム全体としてのコスト増につながり、結果として本方式の導入がためらわれてしまう恐れがある。

そこで、本論文では、「自己ファイルREADの検出によるマルウェアの検知方式」を補完する方式として、「送受信データ間の相関に基づくマルウェアの蔓延防止方式」を提案した。後者の方式では、マルウェアらしい振る舞いとして、ネットワークを介した他コンピュータへの感染行為に着目し、クライアントコンピュータへの受信データと送信データ間の類似性を監視することで感染行為を検出し、マルウェアを検知する。送信データと受信データの監視は、必ずしもクライアントコンピュータ上で行う必要はなく、エンタープライズネットワーク内に設置するネットワーク監視装置上で行うこともできるため、既に感染状態にあるコンピュータの挙動も正しく監視することが可能となる。もちろん、こうした監視装置がマルウェアに集中的に狙われる可能性が考えられるが、監視装置は通常の業務用の汎用的なコンピュータである必要はなく、セキュリティを強化した専用ハードウェアやソフトウェアを用いて防御することが可能である。

さらに、「送受信データ間の相関に基づくマルウェアの蔓延防止方式」では、マルウェアの可能性が高い送受信データが検出された段階で、そのデータ（擬似定義ファイル）を瞬時にネットワーク内の他のクライアントコンピュータに配布して通知することで、当該マルウェアによるさらなる感染の拡大を食い止めることができる。また、「自己ファイルREADの検出によるマルウェアの検知方式」で検知されたマルウェアを、同様に擬似定義ファイルとして利用することも可能である。単なるデータの類似性比較は大きなオーバヘッドなく実行することができるため、エンタープライズネットワーク内で自己ファイルREADの発生監視や送受信データ間の相関監視をリアルタイムに行うクライアントコンピュータはその数を限定し、その他のクライアントコンピュータはマルウェアの疑いありとして配布されるデータ（疑似定義ファイル）と受信データとの比較のみを行えばよいため、システム全体として低コストで未知マルウェアに対する対策がとれることとなる。

OSのファイルシステムそのものを改造することは許されていないため、本論文では、「自己ファイルREADの検出によるマルウェアの検知方式」については、OSが行うファイルアクセスのモニタリングツールであるFileMonを用いて、提案する方式で実際にマルウェアの検知が行えるかの基礎実験を行った。実験の結果、従来方式では検知が困難であったポリモーフィック型マルウェアやメタモーフィック型マルウェアなどのミューテーション型マルウェアも正しく検知することができた。さらに、正規プログラムに対する誤検知についても実施し、代表的に良く使われるプログラムについては一切誤検知が発生しないことを確認した。

「送受信データ間の相関に基づくマルウェアの蔓延防止方式」については、送受信動作を行うOSのAPIをフックして監視する機能を組み込むことにより、検知実験を行

った。その結果、本方式が従来のビヘイビアブロッキング法に基づく検知方式に比べ検知精度も高く、誤検知の発生頻度も低いことが確認できた。また、検知に伴うオーバヘッドも実用上問題とならない程度であることが確かめられた。ただし、本方式では、利用者が受信したメールをそのまま送信するようなメール転送を、マルウェアとして誤検知してしまった。ATM ネットワークのような基幹システムの場合にはメール転送が生じることはなく、また最近のオフィス環境でもセキュリティ上の問題から単なるメール転送は禁止するケースが増えているとは言え、メール転送を誤検知するとなると実用上、幅広いシステムへの適用が難しくなる。メール転送が発生するような環境においては、3.2.3 項(1)で述べたように他方式と組み合わせて検知精度を高める必要がある。

一方、「送受信データ間の相関に基づくマルウェアの蔓延防止方式」では、擬似定義ファイルをマルウェア検出と同時に配布することで、システム全体として効率よく監視が行える。そこで、本論文では擬似定義ファイルを用いた蔓延防止方式の有効性についても検証した。3.2.3 項(3)に示すように、コンピュータ 1 台当りの「被害発生時に想定される損害コスト」と「対策に要する投資コスト」が等しいと見積もった場合には、ネットワークに接続されたクライアントコンピュータが 1,000 台のシステムのうち、968 台のコンピュータはオーバヘッドの少ない擬似定義ファイル方式のみの実装で対策効果が最大となるとの結果が得られた。

一方、トロイの木馬に対しては、その社会的ならびに金銭的な被害の大きさに比べて対策技術の進んでいないキーロガーを対象として、キー入力 API の検出に基づく検知方式である「動的 API 検査方式によるキーロガー検知方式」を提案して、その有効性を検証した。従来の多くのアンチスパイウェア製品と呼ばれるものでは、多種多様な攻撃者の意図を持つスパイウェア(トロイの木馬)を同一の方法で検知しようとするため、アンチウィルス製品に比べて検知精度が低下するという問題があった。真にマルウェアらしい振る舞いを特定するためには、マルウェアの持つ目的(意図)に応じてその振る舞いを規定することが肝要である。本論文では、キーロガーというトロイの木馬を検知対象として限定し、キー入力情報を取得するというキーロガーが持つ本来の攻撃者の意図を、OS が提供する API レベルで抽出し、これをキーロガーの振る舞いとして定式化することで精度良く検知を行う方式を考案した。

OS が提供するキー入力取得に関する API をフックして監視する仕組みをシステムに組み込むことで、本方式の有効性を検証する実験を行った。実験の結果、今回検査対象としたキーロガーは全て検知することができた。特に、キーロガー検知に特化したアンチスパイウェア商用製品でさえも検知ができなかったキーロガーに対しても、本方式であれば検知することができるという結果が得られた。また、正規プログラムの誤検知に関する実験でも、実用上問題ないことが確認できた。4.3.2 節で考察したように、Orchis、xkeymacs、AltIME の 3 つのプログラムは誤検知が発生したが、これらのソフトウェ

アは実際にキーロガーと同様の動作を行わせることが可能であるソフトウェアであり、これらがコンピュータ上で走行していることを利用者に注意喚起することは意味があると考えることもできる。さらに、本方式の適用により発生するオーバヘッドも実用上問題ないレベルであることが確認できた。

「動的 API 検査方式によるキーロガー検知方式」では、OS 起動時に explore.exe プログラムに検査用 DLL をロードして API をインターセプトするという方法を用いている。このため、その後、マルウェアによってさらに API が書き換えられて（上書きされて）しまうと本方式による検知が無効となってしまうという問題がある。これを防ぐためには、2.3.2 項(3)で説明した SELinux や TOMOYO Linux 等で導入されている強制アクセス制御（MAC）機能を用いて、当該 API の書き換えを特定利用者にしか許可しないようにするなどの方策が必要となるだろう。強制アクセス制御機能を用いれば、ファイルやプロセス、デバイスといったコンピュータ資源へのアクセスを、設定したポリシーに基づいて利用者単位で制約することができるため、API の利用権限ポリシーを適切に設定することで、システム管理者以外の利用者による API の書き換えを阻止することができる。更に、たとえ強制アクセス制御機能を実装したシステムであっても、従来の認証方法では、システム管理者のパスワードが漏洩すると、ポリシーファイルの変更が可能となり、強制アクセス制御機能そのものを無効化されてしまう恐れがある。こうした脅威に対しては、筆者らが提案したセキュリティ強化 OS における利用者認証の強化方式[77]を用いて防御することができる。例えば、キーロガーによる被害として顕在化しているインターネットカフェのような環境であれば、インターネットカフェの管理者のみが認識できる任意の認証機構を付加することで、カフェの利用者による API の勝手な書き換えを防止することが可能となる。

なお、本論文で提案した方式は、パターンマッチング方式など既存のマルウェア検知方式に取って代わるものではなく、既存方式における検知精度の低下や誤検知などの欠点を補う方式であることを申し添えておく。「送受信データ間の相関に基づくマルウェアの蔓延防止方式」においても、コンピュータシステムとして効率的なマルウェア検知のためには、既知のマルウェアに対しては広く普及している既存のパターンマッチング方式による検知を行い、未知のマルウェアに対しては、提案した方式で作成した擬似定義ファイルを当面活用するという運用を想定している。さらに、「自己ファイル READ の検出によるマルウェアの検知方式」や「動的 API 検査方式によるキーロガー検知方式」においても、UNIX や Linux 等で実現されている既存の強制アクセス制御方式と併用することにより、より強固な安全性を実現できるのである。

5.2 今後の課題

ミューテーション型や未知のものも含む、ウィルスとワームに対する検知方式として、「自己ファイル READ の検出によるマルウェアの検知方式」と、それを運用的に補完する方式として「送受信データ間の相関に基づくマルウェアの蔓延防止方式」を提案し、その基本的な有効性を検証した。OS のファイルシステムの改造ができないため、現時点ではモニタツールによる基本検知動作の検証に留まっているため、今後は実際にファイルシステム内に検知機能を組み込んで動作検証、オーバヘッド検証を行う必要がある。特に、「自己ファイル READ の検出によるマルウェアの検知方式」では、システムで走行中の全プロセスのファイルアクセスを全てチェックすることになるため、そのオーバヘッドに関しては特に十分な検証を行う必要がある。

さらに、より完全な検知方式とするためには、強制アクセス制御機能との併用が必要となるが、現状、強制アクセス制御機能が基本機能として搭載されている OS は UNIX や Linux などのサーバ系の OS であり、Windows のようなクライアント系の OS では強制アクセス制御機能はまだ十分に装備されているとは言えない。今後のさらなる検知精度向上のためには、OS レベルでの対応が必須である。

トロイの木馬に対する検知技術として、キーロガーの検知を目的とした「動的 API 検査方式によるキーロガー検知方式」を提案し、その有用性を検証した。今回の検証実験では、基本的な方式の有効性の確認を優先したため、寄生先のプロセスにキーロガー行為を行わせたり、寄生を繰り返すような再帰的な動作を行うキーロガーの検知までは確認していない。再帰的な動作ゆえ、監視する API を増やすなど提案した方式の拡張により対応可能であると考えられるが、それに伴うオーバヘッドなどについては今後の検証が必要である。

また、本方式で誤検知した Orchis、xkeymacs、AltIME の 3 つの正規プログラムについては、キーロガーと同様の行為が行えるという意味で、本来利用者に警告すべきという考え方もあるが、さらに厳密に検知しようとするならば、4.3.2 項(1)で言及したような、「外部への通信行為」や「ログ情報としての蓄積行為」などを検知するといった更なる工夫が必要となる。

「動的 API 検査方式によるキーロガー検知方式」においても、提案方式をさらに安全かつ強固なものとするためには、ウィルスやワームへの対処と同様に、強制アクセス制御機能と組み合わせて、キー入力やフックのための API の書き換えを管理者にしか行わせないという仕組みを採り入れる必要があり、同様に OS レベルでの対応が望まれる。

謝辞

本研究を行うにあたり、指導教授である静岡大学 水野忠則先生ならびに西垣正勝先生には、進捗が遅れがちになるにもかかわらず、終始温かくご指導いただき深く感謝申し上げます。また、本研究の遂行にあたって、夜遅くまで様々な角度から有意義な議論を熱心に行って頂いた西垣研究室の室員の皆様には厚く御礼申し上げます。特に、方式の検討から実証実験の実施まで幅広く支援いただいた、株式会社富士通ソーシアルサイエンスラボラトリ 杉村友幸様、東芝ソリューション株式会社 鈴木功一様、静岡大学 高見知寛様の協力なくしては本論文として纏めることはできなかったものと深く感謝致します。株式会社 NTT データ稲田勉様、馬場達也様、前田秀介様、角将高様、東川淳紀様には、論文の作成に当たって、専門家の視点から何度も議論を重ね、示唆に富む助言を多数頂いた。株式会社 NTT データ 田中一男様、原田季栄様には強制アクセス制御に関する知見や助言を頂いた。この場を借りて、深く感謝致します。また、株式会社 NTT データエンジニアリングシステムズ会長（前 株式会 NTT データ代表取締役副社長）中村直司様には、学位取得の機会を支援していただき厚く御礼申し上げます。

最後に、私の研究を裏から支え励ましてくれた私の家族、妻の久恵と娘の千鶴にも深く感謝する次第である。

参考文献

[1] 大駒誠一著、“コンピュータ開発史－歴史の誤りをただす「最初の計算機」をたずねる旅”、共立出版、2005.11、ISBN: 978-4320121386

[2] 流通システム開発センター編、“EDI の知識”、日本経済新聞社、1997.5、ISBN: 978-4532107505

[3] 水田浩著、“CALS の実践”、共立出版、1997.11、ISBN: 978-4320028883

[4] Peter H. Salus 著、“Casting the Net: From Arpanet to Internet and Beyond (Unix and Open Systems Series)”、Addison-Wesley、1995.3、ISBN: 978-0201876741

[5] Tim O'Reilly、“What is Web 2.0 － Design Patterns and Business Models for the Next Generation of Software”、http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html（2007.6.17 確認）

[6] 総務省編、“情報通信白書平成 18 年版”、ぎょうせい、2006.7、ISBN: 978-4324079935

[7] 総務省、“平成 17 年「通信利用動向調査」”、http://www.soumu.go.jp/s-news/2006/pdf/060519_1_bt1.pdf（2007.6.17 確認）

[8] 社会安全研究財団・情報セキュリティビジョン策定委員会編、“情報セキュリティビジョン策定委員会報告書：安全なネットワーク社会の実現を目指して”、東京法令出版、1998.3

[9] 情報処理推進機構セキュリティセンター、“国内におけるコンピュータウィルス被害状況調査”、2005.11、http://www.ipa.go.jp/security/fy17/reports/virus-survey/index.html（2007.6.17 確認）

[10] 経済産業省、 告示第 952 号、“コンピュータウィルス対策基準”、http://www.meti.go.jp/policy/netsecurity/CvirusCMG.htm（2007.6.17 確認）

[11] 第一 I/O 編集部編、“コンピュータ・ウィルス事典”、工学社、2002.6、ISBN: 978-4875933489

[12] 水野貴明著、“コンピュータウイルスの謎”、ソーテック社、2004.11、ISBN: 978-4881664285

[13] Microsoft、“対ウィルス多層防御ガイド”、http://www.microsoft.com/japan/technet/security/guidance/serversecurity/avdind_0.mspx（2007.6.17 確認）

[14] 小畑直祐、宮地玲奈、川口信隆、重野寛、岡田謙一、“ウィルス感染アルゴリズムの違いによる伝播状況のシミュレーション”、情報処理学会、研究報告、CSEC-22、Vol.2004、pp75-80、2004.3

[15] 御池鮎樹著、“インターネット個人情報防衛マニュアル”、工学社、2004.11、ISBN: 978-4777510832

[16] Craig A. Schiller、Jim Binkley、David Harley、Gadi Evron、Tony Bradley、“Botnets”、Syngress Media Inc、2007.2、ISBN: 1597491357

[17] David Moore、et al.、“The Spread of the Sapphire/Slammer Worm”、http://www.cs.Berkeley.edu/~nweaver/sapphire

[18] 杉田誠、片山勝、塩本公平、山中直明、“SQL Slammer ワームウィルスに対する Distributed Active Firewall の性能評価”、情報処理学会、研究報告、CSEC-22、Vol.2003、pp105-112、2003.7

[19] 寺田真敏、長井康彦、倉田盛彦、“Web サービスを対象とするワーム流布対策方式の検討”、情報処理学会、研究報告、CSEC-18、Vol.2002、pp89-96、2002.7

[20] INTERNET Watch、 ““ゼロデイアタック”の脅威が増している”、http://internet.watch.impress.co.jp/cda/special/2004/03/19/2498.html（2007.6.17 確認）

[21] 情報処理推進機構、 “未知ウィルス検出技術に関する調査”、http://www.ipa.go.jp/security/fy15/reports/uvd/index.html（2007.6.17 確認）

[22] 馬場達也、“ウィルス・ワームの生態学”、NETWORKWORLD、pp140-143、2005.1

[23] 菅原啓介、千石靖、八島一司、地下雅志、西川弘幸、岡本英司、“未知ウィルス検出技術に関する一考察”、電子情報通信学会、SCIS2004、pp 1269-1274 、2004.1

[24] Taras Malivanchuk、“The Win32 worms: classification and possibility of heuristic detection”、Virus Bulletin VB2002 Conference、2002.9

[25] Mihai Christodorescu、Somesh Jha、“Static Analysis of Executables to Detect Malicious Patterns” 、Proceedings of the 12th USENIX Security Symp.、pp169-186、2003.8

[26] 阿部洋丈、大山恵弘、岡瑞起、加藤和彦、“静的解析に基づく侵入検知システムの最適化”、情報処理学会論文誌、Vol.45、No.SIG3(ACS5)、pp.11-20、2004.3

[27] 高本勉、“商業サイト改ざん事件から何を学ぶか”、Virus Conference For Enterprise 2005、2005.7

[28] 神薗雅紀、森井昌克、白石善明、“仮想ネットワークを使った未知ウィルス検知システム”、情報処理学会、研究報告、CSEC-22、vol.2003、pp113-120、2003.7

[29] Ian Whalley、Bill Arnold、David Chese、John Moara、Alla Segal、Morton

Swimmer、"An Environment for Controlled Worm Replication and Analysis"、Virus Bulletin VB2000 Conference、2000.9

[30] Kurt Natvig、"Sandbox Technology inside AV Scanners"、Virus Bulletin VB2001 Conference、pp475-488、2001.9

[31] 初野文章、"はじめてのVMware－最新版「Workstation5」の仕組みと利用法"、工学社、2005.9、ISBN: 978-4777511600

[32] Carey Nachenberg、"Behavior Blocking: The Next Step in Anti-virus Protection"、http://www.securityfocus.com/print/infocus/1557（2007.6.17確認）

[33] 島本大輔、大山恵弘、米澤明憲、"System Service監視によるWindows向け異常検知システム機構"、情報処理学会論文誌、Vol.47、No.SIG12(ACS15)、pp.420-429、2006.9

[34] 三宅崇之、森井昌克、白石善明、"仮想サーバを使った未知ウィルス検知システムの提案"、情報処理学会、研究報告、CSEC-18、Vol.2002、pp45-52、2002.7

[35] 森彰、澤田寿実、泉田太宗、井上直、"未知ウィルス検知のための新手法と実装"、情報通信研究機構季報 Vol.51、Nos.1/2、pp73-87、2005.3/6

[36] Mary Landesman、"What is Behavior Blocking?"、Antivirus Software、http://antivirus.about.com/b/a/257711.htm（2007.6.17確認）

[37] Frank Apap、Andrew Honig、Shlomo Hershkop、Eleazar Eskin、Sal Stolfo、"Detecting Malicious Software by Monitoring Anomalous Registry Accesses"、Proceedings of the Recent Advances in Intrusion Detection (RAID 2002): 5th International Symposium、Volume 2516、pp36-53、2002.10

[38] Cliff Changchun Zou、Weibo Gong, Don Towsley、"Code Red Worm Propagation Modeling and Analysis"、Proceedings of the 9th ACM Conference on Computer and Communications Security、2002.11

[39] クワンタム・リープ・イノヴェーションズ・インコーポレーテッド/シュヌラー,「コンピュータ・ウィルス・トラップ装置」、特表平10-501354、1998

[40] 日本ネットワークセキュリティ協会：IDS研究WG報告、"ホストベースのIDSの概要と適用について"、http://www.jnsa.org/active/houkoku/IDSBasic.pdf（2007.6.17確認）

[41] 株式会社東芝/高橋俊成、" コンピュータウィルス発生検出装置、方法、およびプログラム"、特開 2003-241989、2003

[42] 武蔵泰雄、松葉龍一、杉谷賢一、"DNSトラヒックとメールサーバのログ解析"、情報処理学会、研究報告、CSEC-18、Vol.2003、pp185-190、2003.2

[43] 鈴木和也、馬場俊輔、田中貴志、金山卓矢、"トラフィック監視による新出ワームの検出システム"、電子情報通信学会、信学技報、IA2004-14、pp7-12、2004.10

[44] 面和成、下山武司、鳥居悟、“未知ワームを遮断すべきタイミングについて”、情報処理学会、研究報告、CSEC-32、Vol.2006、pp281-286、2006.3

[45] 中谷直司、小池竜一、厚井裕司、吉田等明、“メール型未知ウィルス感染防御ネットワークシステムの提案”、情報処理学会論文誌、Vol.45、No.8、pp.1908-1920、2004.8

[46] 武蔵泰雄、松葉龍一、杉谷賢一、“Mass Mailing Worm と DNS/SMTP トラフィック解析”、情報処理学会、研究報告、CSEC-122、Vol.2002、pp19-24、2002.12

[47] 前田秀介、馬場達也、大谷尚通、角将高、稲田勉、“感染プロセスに着目したワーム拡散防止システムの実装と評価”、情報処理学会、研究報告、CSEC-32、Vol.2006、pp287-292、2006.3

[48] Xinzhou Qin, Wenke Lee、“Statistical Causality Analysis of INFOSEC Alert Data”、Proceedings of the Recent Advances in Intrusion Detection (RAID 2003): 6th International Symposium、Volume 2820、pp73-93、2003.9

[49] 今野徹、“HTTP リクエストの未知攻撃検出における精度向上”、情報処理学会研究報告、マルチメディア通信と分散処理研究会報告、Vol.2004、No.22、pp121-126、2004.3

[50] 東日本電信電話株式会社/鈴木晃、”電子メール中継システム及び電子メール中継方法”、特開 2002-314614、2002

[51] 岩村誠、柏大、“バイナリプログラムにおけるバッファオーバフロー攻撃検知法と攻撃痕跡抽出法”、情報処理学会、研究報告、CSEC-22、Vol.2004、pp187-192、2004.3

[52] 脇田建、“バッファ溢れ攻撃とその防御”、日本ソフトウェア科学会、コンピュータソフトウェア、Vol.19、No.1(20020115) 、pp49-63、2002.1

[53] 西川弘幸、太田良二、八島一司、千石靖、岡本栄司、“セキュリティホールを狙うワーム検出の実験” 、電子情報通信学会、SCIS2004、pp1275-1280 、2004.1

[54] Andrew P. Kosoresow、Steven A. Hofmeyr、“Intrusion Detection vis System Call Traces”、IEEE Software、Vol.14、No.5、pp35-42、1997.9

[55] デビット・ソロモン、マーク・ルシノビッチ著、豊田孝訳、“インサイド Microsoft Windows 第 4 版(下)”、日経 BP ソフトプレス、2005.10、ISBN: 978-4891004743

[56] サクラエディタ、http://sakura_editor.at.infoseek.co.jp/ (2007.6.17 確認)

[57] メモリの掃除屋さん、http://www6.plala.or.jp/amasoft/soft/soft1/memcl.html (2007.6.17 確認)

[58] Chris Wright、Crispin Cowan、Stephen Smalley、James Morris、Greg Kroah-Hartman、“Linux Security Modules: General Security Support for the Linux Kernel” 、Proceedings of the 11th USENIX Security Symposium、pp17-31、2002.8

[59] 中村雄一、上野修一、水上友宏著、“SELinux 徹底ガイド”、日経 BP 出版センター、2004.3、ISBN: 978-4822221119

[60] Peter A. Loscocco、Stephen D. Smalley、“Meeting Critical Security Objectives with Security-Enhanced Linux”、Proceedings of the 2001 Ottawa Linux Symposium、2001.7

[61] 情報処理推進機構、“アクセス制御に関するセキュリティポリシーモデルの調査報告書” 、2004 情財第 736 号、2005.3

[62] 内閣官房情報セキュリティセンター、“電子政府におけるセキュリティに配慮した OS を活用した情報システム等に関する調査研究”、http://www.nisc.go.jp/inquiry/pdf/secure_os_2004.pdf（2007.6.17 確認）

[63] 原田季栄、保理江高志、田中一男、“使いこなせて安全な Linux を目指して”、Linux Conference 2005、第 3 巻、CP-09、2005.6

[64] 原田季栄、保理江高志、田中一男、“TOMOYO Linux-タスク構造体の拡張によるセキュリティ強化 Linux”、Linux Conference 2004、第 2 巻、CP-06、2004.6

[65] Russell Coker、“Root for All on the SE Linux Play Machine”、LINUX Journal、2003.8

[66] 原田季栄、松本隆明、“セキュリティ強化 OS によるログイン認証の強化方式”、情報学研究第 11 巻、pp93-102、2005.9

[67] 日本ネットワークセキュリティ協会、“セキュリティ・スタジアム 2004 盛況のうちに終了”、JNSA Press 第 12 号、pp25-26、2004.12

[68] 阿部一義著、“パーソナル・ファイアウォール導入ガイド”、工学社、2002.3、ISBN: 978-4875933144

[69] Vincent Berk、George Baakos、Robert Morris、“Designing a Framework for Active Worm Detection on Global Networks”、Proceedings of the First IEEE International Workshop on Information Assurance、pp13、2003.3

[70] 岡本剛、“DNS の正引き応答連動型パケットフィルタリングによるワーム増殖”、情報処理学会、研究報告、CSEC-23、Vol.2003、pp19-24、2003.12

[71] 中村信之、中井敏久、“トラフィック内部状態変化を利用したネットワーク異常検知”、電子情報通信学会、信学技報、NS2005-5、pp17-20、2005.4

[72] 中村信之、中井敏久、“複数プローブによる異常トラフィック検知システム”、情報処理学会、研究報告、CSEC-32、Vol.2006、pp269-274、2006.3

[73] 今野徹、楯岡正道、“ネットワークに対する未知攻撃の検知・防御技術とその応用”、東芝レビュー、Vol.60、No.6、pp32-35、2005.6

[74] 面和成、東角芳樹、鳥居悟、武仲正彦、“セキュアな企業内ネットワークの実現～メールウィルスによるゼロデイアタックの防御～”、コンピュータセキュリティシンポジウム 2004 論文集、Vol. I 、pp.19-24、2004.10

[75] Lewis Napper 著、江村豊訳、“Winsock2 プログラミング 改訂第 2 版”、ソフトバンククリエイティブ、2004.12、ISBN: 978-4797330441

[76] 澤川渡、綱島明宏著、“TCP/IP 解析とソケットプログラミング”、オーム社、200.2、ISBN: 978-4274063534

[77] 与那原亨、大谷尚通、馬場達也、稲田勉、 “トラフィック解析によるスパイウェア検知の一考察”、情報処理学会、研究報告、CSEC-30、Vol.2005、pp23-29、2005.7

[78] Anti-Spyware Coalition, “Glossary”、http://www.antispywarecoalition.org/documents/GlossaryJune292006.htm（2007.6.17 確認）

[79] 渡部章著、“恐怖のスパイウェア”、三交社、2006.7、ISBN: 978-4903559001

[80] Cormac Herley、Dinei Florencio、 “How To Login From an Internet Cafe Without Worrying About Keyloggers” 、Symposium On Usable Privacy and Security、2006.7

[81] Webroot Software, Inc.、“STATE OF SPYWARE”、Q1 2006

[82] Francois Paget、 “Identity Theft” 、http://www.mcafee.com/us/local_content/white_papers/wp_id_theft_en.pdf（2007.6.17 確認）

[83] B. McCarty、 “Botnets: big and bigger” 、Security and Privacy Magazine、IEEE、Volume 1、Issue 4、pp87-90、2003.7

[84] 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、pp2512-2523、2006.8

[85] J. Oikarinen、D. Reed、“RFC 1459: Internet Relay Chat Protocol”、1993.5

[86] Malware Block List、http://www.malware.com.br/index.shtml（2007.6.17 確認）

[87] Luis von Ahn、Manuel Blum、Nicholas J. Hopper、John Langford、 “CAPTCHA: Using Hard AI Problems for Security”、Proceedings of the EUROCRPYT 2003: International Conference on the Theory and Applications of Cryptographic Techniques、Volume 2656、pp646、2004.2

[88] Kishore Subramanyam、Charles E. Frank、Donald H. Galli、“Keyloggers: The Overlooked Threat to Computer Security”、First Midstates Conference for Undergraduate Research in Computer Science and Mathematics、2003.10

[89] 愛甲健二著、“ハッカー・プログラミング大全 攻撃編”、データハウス、2006.3、ISBN: 978-4887188679

[90] Kimmo Kasslin、“Kernel Malware: The Attack from Within”、Association of

anti-Virus Asia Researchers (AVAR) 2006、2006.12

[91] トレンドマイクロ、“ウィルス検出技術”、http://www.trendmicro.com/jp/security/general/tech/overview.htm（2007.6.17 確認）

[92] 情報処理推進機構、“2005 年のコンピュータウィルス届出状況”、http://www.ipa.go.jp/security/txt/2006/documents/2005all-vir.pdf（2007.6.17 確認）

[93] 御池鮎樹著、“インターネット個人情報防衛マニュアル”、工学社、2004.11、ISBN: 978-4777510832

[94] STYOPKIN Software、“Keylogger Hunter”、http://www.styopkin.com/keylogger_hunter.html（2007.6.17 確認）

[95] Citisoft Development、“Keylogger Stopper”、http://www.chithai.com/keystop.htm（2007.6.17 確認）

[96] ISecSoft, Inc.、“Anti-Keylogger Elite”、http://www.remove-keyloggers.com/index.php（2007.6.17 確認）

[97] Spybot Search & Destroy、http://www.safer-networking.org/en/index.html（2007.6.17 確認）

[98] Symantec Security Response、“Remacc.SpyAnywhere”、http://www.symantec.com/region/jp/avcenter/venc/data/jp-remacc.spyanywhere.html（2007.6.17 確認）

[99] Symantec Security Response、“Spyware.Perfect”、http://www.symantec.com/region/jp/avcenter/venc/data/jp-spyware.perfect.html（2007.6.17 確認）

[100] Symantec Security Response、“Spyware.ActivityLog”、http://www.symantec.com/enterprise/security_response/writeup.jsp?docid=2004-062311-5929-99（2007.6.17 確認）

[101] Symantec Security Response、“Spyware.XpcSpy”、http://www.symantec.com/region/jp/avcenter/venc/data/jp-spyware.xpcspy.html（2007.6.17 確認）

[102] Vector、“きいろがぁ”、http://rd.vector.co.jp/soft/win95/util/se322072.html（2007.6.17 確認）

[103] Symantec Security Response、“Infostealer.Wowcraft”、http://www.symantec.com/ja/jp/security_response/writeup.jsp?docid=2005-073115-1710-99（2007.6.17 確認）

[104] Vector、“Parasite”、http://www.vector.co.jp/soft/winnt/util/se327656.html（2007.6.17 確認）

[105] Vector、"WingKEY"、http://www.vector.co.jp/soft/winnt/util/se263226.html (2007.6.17 確認)

[106] Vector、"キーのログをとる者"、http://www.vector.co.jp/soft/win95/util/se369025.html (2007.6.17 確認)

[107] S-CENTER、"Casper"、http://www.s-center.net/index.php (2007.6.17 確認)

[108] Vector、"Orchis"、http://www.vector.co.jp/soft/win95/util/se127007.html (2007.6.17 確認)

[109] Vector、"Xkeymacs"、http://www.vector.co.jp/soft/win95/util/se196436.html (2007.6.17 確認)

[110] Vector、"AltIME"、http://www.vector.co.jp/soft/win95/util/se027730.html (2007.6.17 確認)

[111] Muhammad Aslam、Rana Naveed Idrees、Mirza Muzammil Baig、Muhammad Asif Arshad、"Anti-Hook Shield against the Software Key Loggers"、Proceedings of the National Conference on Emerging Technologies (NCET) 2004、pp189-191、2004.12

[112] MSDN、"SetWindowsHookEx"、Microsoft、http://msdn2.microsoft.com/en-us/library/ms644990.aspx (2007.6.17 確認)

[113] Jefferev Richter 著、長尾高弘、ロングテール訳、"Advanced Windows 改訂第4版"、アスキー、2001.5、ISBN: 978-4756138057

[114] Microsoft Research、"Detours"、http://research.microsoft.com/sn/detours/ (2007.6.17 確認)

[115] Galen Hunt、Doug Brubacker、"Detours: Binary Interception of Win32 Functions"、Proceedings of the 3rd USENIX Windows NT Symposium、pp135-43、1999.7

[116] Rubin A. D.、Geer D. E., Jr.、"Mobile code security"、Internet Computing、IEEE、Volume 2、Issue 6、pp30-34、1998.11

関連論文

A　論文

1）　松本隆明、杉村友幸、鈴木功一、前田秀介、馬場達也、水野忠則、西垣正勝：送受信データ間の相関に基づく未知ワーム検知を利用した蔓延防止手法の提案、情報処理学会論文誌、Vol.47、No.6、pp.1941-1953、2006

2）　松本隆明、鈴木功一、高見知寛、馬場達也、前田秀介、水野忠則、西垣正勝：自己ファイル READ の検出による未知ワームの検知方式、情報処理学会論文誌（採録決定）

3）　松本隆明、高見知寛、鈴木功一、馬場達也、前田秀介、水野忠則、西垣正勝：動的 API 検査方式によるキーロガー検知方式、情報処理学会論文誌（採録決定）

B　国際会議

1）　Takaaki Matsumoto: The Prospects for Cyber Security, World Summit on the Information Security Asian Regional Conference, 13 Jan., 2003.

2）　Takaaki Matsumoto: The Role of the Technology in Supporting Information Systems and Network Security and Trust, OECD Global Forum on Information Systems and Network Security, 13-14 Oct., 2003.

3）　Takaaki Matsumoto: The Role of Technology in Supporting Information Security, CIAJ-TIA Broadband Forum, 10 Jan., 2004.

4）　Osamu Dosaka, Takaaki Matsumoto: Biometrics for Border Control Systems, 11th German-Japanese Symposium, 13-16 Sep., 2005

研究会など

1）　鈴木功一、松本隆明、高見知寛、馬場達也、前田秀介、西垣正勝：自己ファイル READ の検出による未知ワーム・変異型ワームの検知方式の提案、2006 年情報処理学会研究報告、2006-CSEC-32、pp.275-280、2006.3

2）　高見知寛、鈴木功一、馬場達也、前田秀介、松本隆明、西垣正勝：動的 API 検査方式によるキーロガー検知方式の提案、2006 年情報処理学会研究報告、Vol.2006、No.26、2006-CSEC-32、pp.209-214、2006.3

3）　鈴木功一、松本隆明、高見知寛、馬場達也、前田秀介、水野忠則、西垣正勝：自己ファイル READ の検出による未知ワームの検知方式の提案（その２）、2007 年暗号と情報セキュリティシンポジウム、2007.1

4）　高見知寛、鈴木功一、馬場達也、前田秀介、松本隆明、西垣正勝：動的 API 検

査方式によるキーロガー検知方式の提案（その２）、2006 年情報処理学会研究報告、2007-CSEC-36、pp.147-152、2007.3

5） 原田李栄、松本隆明：セキュリティ強化 OS によるログイン認証の強化手法、情報学ワークショップ 2005

6） 松本隆明、岡本龍明：未来ネット技術シリーズ「情報セキュリティ技術」、オーム社、2000