平成20年度　　卒業研究論文

Windowsにおけるシステムコール履歴に基づく異常検知システムの実現

指導教官
齋藤 彰一 准教授

名古屋工業大学　情報工学科
平成17年度入学　17115072番

下平 哲也

目 次

第1章

はじめに

近年，マルウェアによる攻撃は増加の一歩をたどり，その攻撃手法も多種多様なものとなってきている．マルウェアの攻撃に対し，主な既存対策手法ではパターンマッチングによるブラックリスト方式を採用している．ブラックリスト方式とは，既存のマルウェアの特徴をあらかじめデータベースに登録しておき，それらと比較することによって異常を検知するというものである．しかし，この方式はデータベースに登録されていない新種のマルウェアに対しては十分な効力を発揮する事はできない．また，マルウェアの種類の増加によりブラックリストの検索コストは増大し，パフォーマンスの低下やデータベースの肥大化が生じるなどの問題がある．これらの理由により，ブラックリスト方式にはいづれ限界が訪れると考えられている．

これらの問題を解決するための手法として，Forrest ら [1] を先駆けとするシステムコール履歴に基づく異常検知がある．これは正常動作時のプログラムのシステムコール履歴をあらかじめ記録しておき，実際の実行履歴と比較することにより異常を検知するというものである．この方法であれば，マルウェアの種類に依存しないため，上記の問題は生じない．この異常検知手法は槙本らによる研究 [2] 等，盛んに研究が行われている．しかし，これらの研究対象は Linux を対象としたものであり，現在最も普及している Windows に関してはほとんど対象とされていなかった．これは Windows が Linux と違いソースが公開されていないため，システムコール履歴の取得が困難であったことなどが考えられる．よって本論文ではマルウェアの種類によらない異常検知を目標とし，Windows におけるシステムコール履歴に基づく異常検知システムを提

案する．

本提案手法では対象 OS を WindowsXP する．また，システムコール発行時に実行される sysenter 命令に着目し，この命令をフックすることによってシステムコール履歴の保存を実現した．動作履歴の記録には N-gram を用いる．異常と判断する基準として違反動作パターンの発生数に基づく方法と，システムコールの群れを識別しその内の動作パターンの発生内訳に基づく方法の二つの方式の実装を行った．

以降，二章にて関連研究に関して述べ，三章にて提案手法の説明を行い，四章にて性能評価，五章にて結論と今後の課題について説明する．

第2章

関連研究

本提案手法にて用いた関連研究に関する説明を行う．

2.1 システムコール

通常，オペレーティングシステムの機能の一部はシステムコールとしてユーザプログラムへ提供される．それはファイルに対する読み書きや，ハードウェアへのアクセス等であり，アプリケーションにとってはなくてはならない存在である．しかし，システムコールはそれ単体ではプログラマにとって，扱いにくいものである．なぜならシステムコールは機能としての最小単位である場合が多く，ひとつファイルを読み書きするだけでも多くのシステムコールを発行する必要があるためである．そのため，プログラマはより抽象度の高い API を用いてシステムコールを扱う．API はシステムコールを組み合わせてよりプログラマが用いやすいように機能をパッケージしたものである．プログラマは API を用いることによりシステムコールの存在を意識することなく，システムコールを扱うことが可能となる．例えばファイルアクセスを扱う場合，プログラマは API として fopen や fread を用いるが，fopen の内部では “NtCreateFile” や，“NtOpenFile”，“NtReadFile” といったシステムコールを組み合わせて機能を構築している．また，システムコールはオペレーティングシステムと同じくカーネルモードにて実行されるが，API はユーザーモードで実行されるという違いがある．これは API が DLL としてユーザに提供されているためである．

ユーザアプリケーションはシステムコールなしにはオペレーティングシステムが管

理する情報へアクセスすることはできない．これはユーザアプリケーションとして活動するマルウェアに関しても同様である．マルウェアによって情報の取得や改竄が行われる際には必ずシステムコールが発行される．本提案手法ではこの点に着目し，システムコールの発行履歴に基づく侵入検知の実装を行った．

なお，システムコールはスーパーバイザコールと呼ばれたり，Windows においてはシステムサービスと呼ばれることもあり，機能と定義に若干の違いが生じるが，OS の機能をプログラムに提供するという点でシステムコールと同義であるため，以降システムコールと名称を統一する．

2.2 システムコールフック

Linux では他プロセスの発行するシステムコールを取得するためのインタフェースとして ptrace システムコールがある．しかし，Windows においてはそのようなインタフェースは公式には提供されていない．類似したものとしてメッセージフックがあるが，これは Linux のシグナルに相当するウィドウメッセージをフックするものである．ウィンドウメッセージは主にユーザアプリケーション間の通信を行うためのものである．このウィンドウメッセージをプログラムの動作様相と見なすこともできる．しかし，マルウェアはウィンドウメッセージを発行することなく情報の取得や改竄が可能である．このためウィンドウメッセージを用いた方法では目的とする異常検知システムを実現することはできない．よって，システムコールフックを実現するため，本提案手法では自作のフック機構を導入する．

Windows においてシステムコールフックを実現する方法として主に次の 2 つの方式がある．SSDT とは各システムコールへのアドレスが格納されているテーブルであり，sysenter 命令とは WindowsXP(x86) にてユーザーモードからカーネルモードへ切り替えるための CPU 命令である．

- SSDT(System Service Descriptor Table) パッチング
- sysenter 命令フック

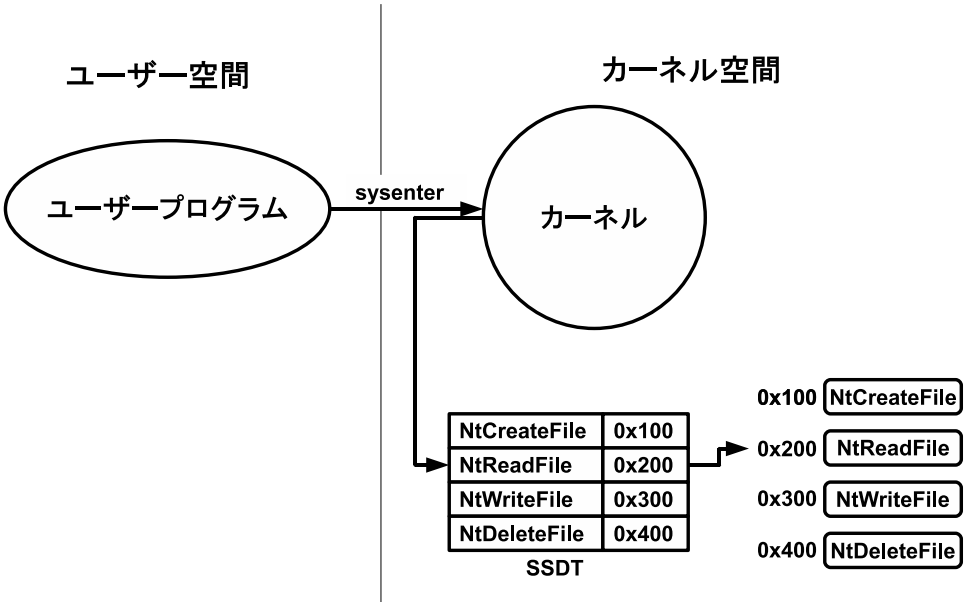

図 2.1: システムコール呼び出し時の動作

2.2.1 SSDT パッチング

SSDT パッチングについて説明する．通常，システムコールが発行された際，システムコールはシステムコール ID としてカーネルに通知される．通知を受け取ったカーネルは，渡されたシステムコール ID を基に SSDT よりエントリの検索を行う．カーネルは SSDT より該当システムコールへのアドレスを取得後，それを実行する．NtReadFile を呼び出した場合の例を図 2.1 に示す．SSDT パッチングでは，この SSDT 内の目的システムコールに該当するエントリをフック用のコードの先頭アドレスへと書き換えることでフックを実現する．フック対象システムコールが発行されると，フック用のコードはあたかも，そのシステムコール自体であるかのようにカーネルに呼び出される．そして処理を行った後，本来のシステムコールの呼び出しを行う．場合によっては本来のシステムコールを呼び出さず，エラーを返すことによって，システムコールを制限する事も可能である．この方法はシステムコールフックによるオーバーヘッドの影響が，フック対象とするシステムコールのみに限定されるため，全体に対するパ

表 2.1: Windows におけるシステムコールフック手法

手法	利点	欠点
SSDT パッチング	オーバーヘッド小	該当エントリを全て書き換える必要性
sysenter 命令フック	管理が容易	カーネルモードプログラムは対象外

フォーマンスの低下が小さいという利点がある．しかし，フック対象とするシステムコールに該当するエントリをすべて書き換えなければならないという欠点もある．

2.2.2 sysenter 命令フック

sysenter 命令フックについて説明する．ユーザーモードで実行されているプログラムがシステムコールを実行するためには，必ずカーネルモードへ移行する必要がある．その際に，WindowsXP(x86) では sysenter 命令が用いられ，カーネルモードへの切り替えが行われる．sysenter 命令は実行モードをカーネルモードへと切り替えた後，次の処理へのアドレスを特殊なレジスタ MSR より読み取り処理を行う．sysenter 命令フックではこの MSR の値を書き換えることによってフックを実現する．これにより，sysenter 命令実行後にはフック用のコードへと処理が移る．通常，書き換え対象の MSR の値は変更されないため，MSR の書き換えはフック開始時に一度行うだけでよい．この方法は，ユーザモードが発行するシステムコールが全て同一のフックコードを通過することになるため，管理が容易であるという利点がある．しかし，フック可能な対象がユーザモードプログラムに限られるという欠点がある．これはカーネルモードで動作するプログラムは sysenter 命令を発行しないためである．なお，sysenter 命令は PentiumII より実装された CPU 命令であり，カーネルモードへの切り替え命令は Windows のバージョンによって異なるものが使用されている．Windows2000 では int 2E 命令，WindowsXP(x64) では syscall 命令が用いられている．これらは sysenter と同様の処理を行うものであり，それぞれの命令に対しても同等の処理を行うことでフックが可能となる．システムコールフックを実現する二つの方式の特徴を表 2.1 にまとめた．本提案手法では，次の二つの点から sysenter 命令フックを採用するものとした．

- 複数の種類のシステムコールをフックする必要がある
- 監視対象はユーザーモードプログラム

2.3 N-gram

本提案手法は正常時のシステムコール発行履歴を記録し，実際の動作と比較し異常の検知を行う．しかし，GUI プログラムや複雑なプログラムでは，ユーザー入力や環境変数によって発行されるシステムコールの種類や順番は常に変化し，単純に順番を記録しただけでは期待する侵入検知を実現することはできない．よって本提案手法では，N-gram を用いて正常なシステムコール順の記録を行った．

N-gram とは検索エンジンなどで用いられる順番を記録するための記録方式であり，入力列中に存在する一定数 N 個分の順番を全て記録する．例えば入力として「A B C D E F」があったとき，N-gram を用いて定数 N を 3 として記録すると「ABC」「BCD」「CDE」「DEF」の四つの順番の組が記録される．以降，N-gram によって生成される順番の組を動作パターンと呼び，動作パターンとして保存するものを動作規則と呼ぶ．

一般的に，順番に意味のある列に対し，任意の位置 K を注目点として見た時，K は K より前方の，ある一定の N 個の値に影響をうけている場合が多い．例えば検索エンジンが対象とする言語であれば,「今日　の　天気　は　○○」という列を対象としたとき，○○に入る単語を想像すると「晴れ」や,「雨」と予想することができる．これは○○が前方 2 個の「天気　は」という列に影響を受けていると予想できるためである．検索エンジンではこの法則を利用し，ユーザーが入力した文字列から次に入力すると予想される単語を候補として表示する機能を実現している [3]．プログラムにおいても，この例は当てはまる．通常，プログラムは実現する機能を関数という形で小さなモジュールに分割し，それらを組み合わせて構築する．この関数内において，条件分岐を除いた部分では命令は常に一定の順番で実行されるため，ある命令は一定の N 個以前の命令に依存していると考えることができる．本提案手法ではこの法則を利用し，N-gram によってプログラムの動作様相を動作規則として記録した．

第3章

提案手法

提案手法は学習モードと検知モードの二つのモードがあり，学習モードでは動作規則の作成を行い，検知モードでは記録された動作規則を基に異常の検知を行う．

本手法では異常と判断するための基準として，プロセスの動作規則に対する違反度合いを示す違反度というパラメータに基づく方法と，短時間にまとまって発行されたシステムコールによって構成されるシステムコール群に基づく方法の二つの方式の提案する．

3.1 システムコール群

プログラムが発行するシステムコールには，その発行履歴に時間的偏りがある．その理由はシステムコールが API としてパッケージされていること，プログラム自体が関数としてモジュール化されていることが考えられる．また，GUI プログラムなどのユーザー対話型のプログラムではユーザーの入力待ちなどによって処理は中断され，発行されるシステムコールは時間的に局所に集中し，群れのようになって発行されることが多い．以降，時間的に集中し，まとまりとなったシステムコールの群れをシステムコール群という．ひとつのスレッドが発行したシステムコール数の推移を元に，システムコール群を区別した例を図 3.1 に示す．

システムコール群はそのプログラム中の特定のモジュールが発行するシステムコールの群れと考えることができる．モジュールとモジュールにまたがる動作パターンは，その順番の意味は薄く，動作規則として保存することは避けるべきである．よって本

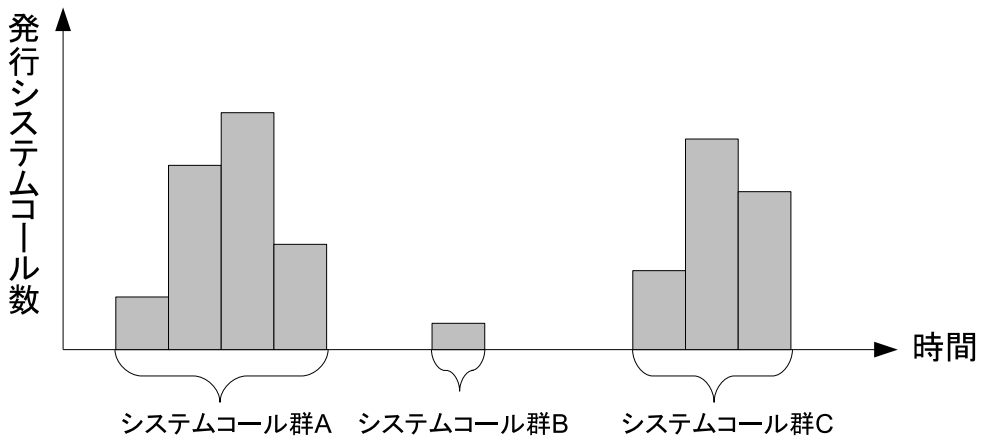

図 3.1: システムコール群の例

提案手法では，動作規則の精度の向上のため，このシステムコール群の判別を行った．また，システムコール群をモジュールとしてみた場合，モジュールが発行する動作パターンの内訳はパターン化することができる．システムコール群 A は動作パターン A が 80 パーセント，動作パターン B が 15 パーセント，動作パターン C が 5 パーセントといった具合である．本提案手法ではこの特性に着目し，システムコール群中における動作パターンの内訳をパターン化し記録を行い，実際の動作と比較することによって，異常の判断基準としても利用する．

3.1.1 システムコール群の判別

監視しているスレッドそれぞれに対し，管理プログラムはシステムコール群の判別を行う．管理プログラムは対象スレッドの単位時間当たりの発行システムコール数を監視し，発行システムコールが途切れた時点をシステムコール群の分け目として判別する．

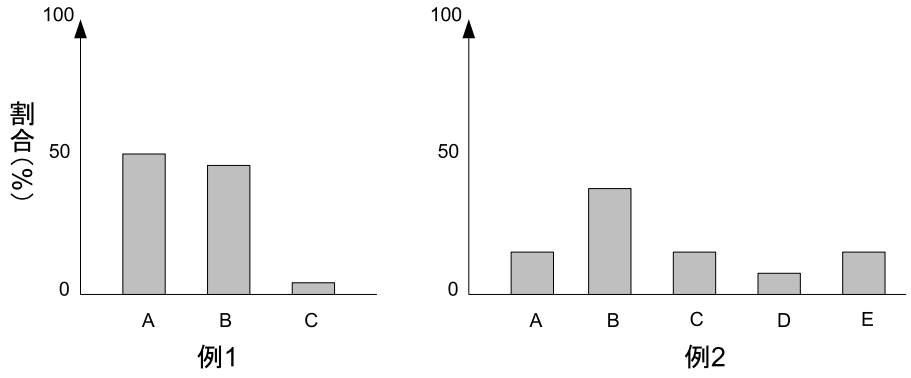

図 3.2: システムコール群中の動作パターンの内訳の例

3.1.2 システムコール群のパターン化

学習モードにてシステムコール群が決定されると，そのシステムコール群を構成する動作パターンの内訳の算出を行い，システムコール群パターンとして記録を行う．しかし，システムコール群の動作パターンの内訳を厳密に記録すると不都合が生じる．システムコール群はその時々の実行環境によってその内訳は微妙に変化し，厳密に記録を行った場合では，システムコール群パターンが膨大な数になるためである．システムコール群パターンが増大すると，その比較にかかるオーバーヘッドは大きくなる．また，誤検知が多くなるという問題もある．そのため，システムコール群の動作パターンの内訳の記録には，微妙に変化する部分を除くため，ある程度緩和した大まかな内訳のみを記録する必要がある．この大まかな内訳への変換をシステムコール群のパターン化という．

システムコール群の内訳の例を図 3.2 を示す．パターン化の際には全体に対して占める割合が相対的に小さいものを切捨てる．図 3.2 中の例 1 の場合，内訳として記録する対象は動作パターン A と動作パターン B の二つであり，動作パターン C は誤差と考え切捨てを行う．この際の切り捨てる動作パターンの判断基準を,「占める割合が平均割合の 30 %以下のもの」とする．例 1 の場合，一つの動作パターンが占める割合の平均値は約 33 %であり，割合が 9.9 %以下である C は切り捨てられる．そして，動作

パターンＡ－55％，動作パターンＢ－45％として記録する．また，例2の場合では，平均は20％であり，6パーセント以下のものが切捨てとなる．しかし，該当する動作パターンはないないためそのまま記録する．以上のようにしてシステムコール群のパターン化を行う．以降，システムコール群のパターン化によって保存される規則をシステムコール群規則という．

3.1.3　システムコール群の比較

システムコール群同士の比較には含まれる動作パターンの内訳の比較を行う．まず，3.1.2節に示した方法に基づき，比較する双方のパターン化を行う．そして，含まれる動作パターンそれぞれに対し割合の比較を行い，全ての差が許容誤差範囲内であれば，同一のシステムコール群とみなす．本提案手法では許容誤差範囲を5％とした．

3.2　異常判断基準

本提案手法では異常と判断するための基準として違反度に基づく方法と，システムコール群における動作パターンの発生割合に基づく方法の二つの方式を設ける．

3.2.1　違反度に基づく方法

実行されているプロセスそれぞれに対し，本提案手法では違反度というパラメータを設ける．違反度の初期値は0であり，そのプロセスが動作規則にない動作パターンを行うたび加算が行われる．違反動作パターンは学習モードの精度や，監視対象プログラムの特性によって少なからず発生してしまうということを考慮し，違反度は定期的に0へと減少するものとした．違反度が一定値に達すると異常として判断する．

3.2.2　システムコール群に基づく方法

学習モードにてパターン化され記録されたシステムコール群に基づき異常を判断する．3.1.3節に示した比較方法によって比較を行い．学習モードにて記録されたシステ

ムコール群に含まれないと判断された場合，直ちに異常と判断する．

第4章

実装

本章では実際に実装したシステムの説明を行う．

4.1 全体構成

提案システムの全体構成を説明する．提案システムはカーネルモードで動作する監視デバイスドライバと，ユーザーモードで動作する管理プログラムの二つによって構成される．監視デバイスドライバではシステムコール履歴の取得を行い，管理プログラムでは動作規則の作成や異常検知を行う．本提案手法の動作図を図 4.1 に示す．監視対象ユーザプログラムがシステムコールを発行すると，sysenter 命令が発行されカーネルモードへの切り替えが行われる．その後，本来であればカーネルによってシステムコールの実行が行われるが，代わりに本提案システムの監視デバイスドライバへと制御が移行する．監視デバイスドライバではシステムコール履歴の保存を行った後，カーネルへ処理の引き渡しを行う．管理プログラムは定期的にデバイスドライバへとアクセスし，システムコール履歴を受け取る．管理プログラムはうけっとたシステムコール履歴を解析し，学習モードであれば動作規則の保存を行い，検知モードであれば異常の検知を行う．異常が検知された際には該当ユーザプログラムに対し停止の処理を行う．

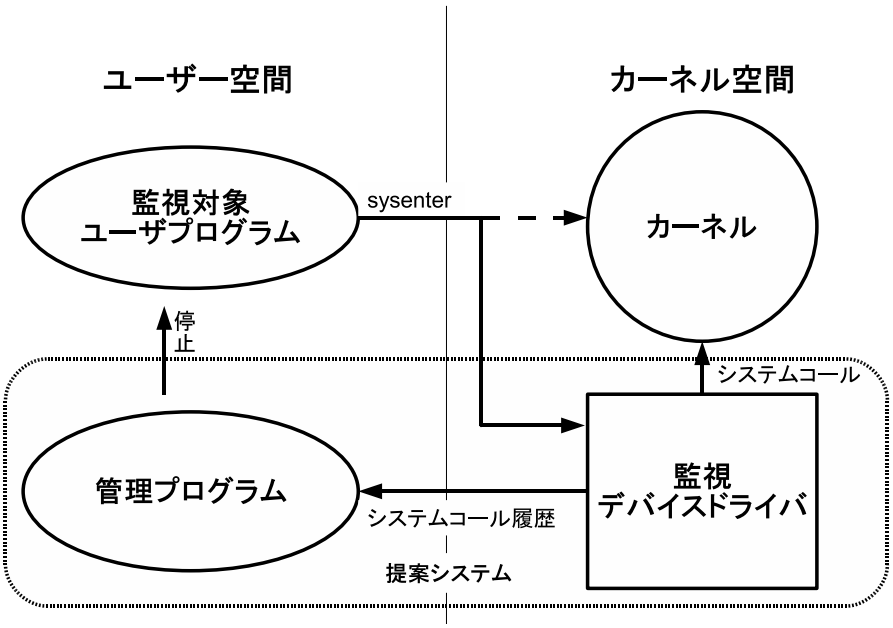

図 4.1: 提案手法の動作図

4.2 監視デバイスドライバ

sysenter 命令フックを実装するにあたり，カーネルモードにて実装を行う必要があるため，監視部分をデバイスドライバにて実装を行う．デバイスドライバとして実行されるが，特にハードウェアを制御するといったことはなく，ユーザープログラムとカーネルとの橋渡しを行うのみである．

4.2.1 sysenter フック

本提案手法ではシステムコールフックを実現するための方法として sysenter 命令フックを用いる．sysenter 命令が実行されると，カーネルモードへ移行するため，大まかに次の処理が実行される [4]．MSR とは CPU 固有の情報を保持するための特殊なレジスタであり，sysenter ではカーネルモードへ移行する際の各レジスタの値を保持させている．

1. セグメントセレクタを SYSENTER_CS_MSR(MSR 174H) から CS レジスタにロード
2. 命令ポインタを SYSENTER_EIP_MSR(MSR 176H) から EIP レジスタにロード
3. SYSENTER_CS_MSR の値に 8 を加算し，その合計を SS レジスタにロード
4. スタックポインタを SYSENTER_ESP_MSR(MSR 175H) から ESP レジスタにロード
5. 特権レベル 0 に切り替え，カーネルモードへ切り替える．

ここで手順 2 に注目する．ここでは sysenter 命令実行後の命令アドレスを MSR のうち，176H 番目の SYSENTER_ESP_MSR より EIP レジスタにロードしている．SYSENTER_ESP_MSR は，常に一定の値であり，sysenter 命令実行後は常に同じ箇所へ実行が遷移している．この SYSENTER_ESP_MSR をこちらで用意したフック用のコードへのアドレスへと書き換えることによって sysenter 命令フックが可能となる．

MSR を書き換えるためには RDMSR 命令と WRMSR 命令を用いる [5]．しかし，これらの命令はカーネルモードでのみ実行可能な CPU 命令であり，ユーザーモードプログラムからは実行することができない．そのためカーネルモードで任意のコードを実行させることが可能なデバイスドライバにて実装を行う．フック開始時に RDMSR によって本来のジャンプ先を MSR より取得と記録を行い，WDMSR によってデバイスドライバ内のフック用のコードへと SYSENTER_ESP_MSR を書き換える．また，マルチコアプロセッサでは MSR はコアごとに独立しているため，全てのシステムコールをフックするために全てのコアにて書き換えを行う．そして，フック用のコードでは，呼び出されるシステムコール ID を記録後，本来のジャンプ先へとジャンプする．

4.2.2 システムコール履歴の記録

フック用のコードが実行される際，システムコール ID は eax レジスタに格納されている．フックコードでは eax の値を取得後，スレッド別にシステムコールの発行履歴を保存する．具体的には，発行プロセス名，プロセス ID，スレッド ID を取得し，それらをキーとしたデータ構造によって履歴の保存を行う．その際に形成されるデータ構造の例を図 4.2 に示す．この際，プロセス名とプロセス ID の両方を取得するのは，同名のプロセスが複数実行される場合があるためであり，また，検知モードではプロセス名でデータの検索を必要があるためである．このデータ構造により，スレッド毎に履歴の保存が可能となり，マルチスレッドプログラムに対しても有意義なシステム

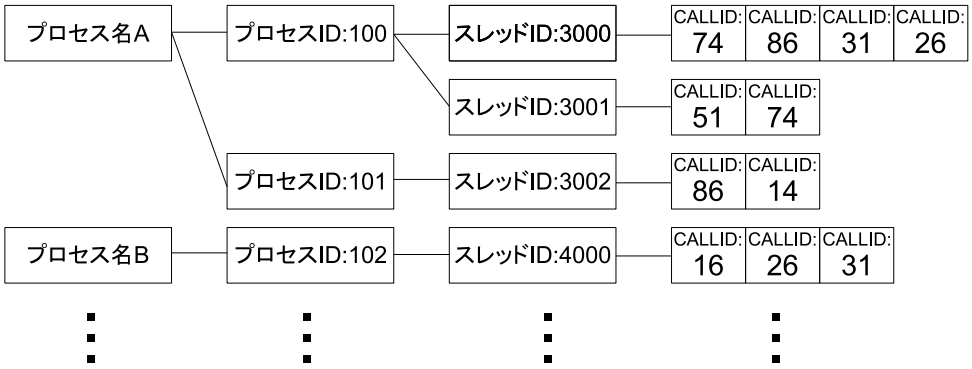

図 4.2: システムコール履歴を保存するデータ構造

コール履歴の取得が可能となる．

4.3 管理プログラム

管理プログラムは定期的にデバイスドライバへとアクセスし，システムコール履歴を受け取り解析を行う．また，システムの停止ボタンなどのユーザインタフェースを提供する．以下，管理プログラムについて説明する．

4.3.1 動作パターンの作成

管理プログラムはデバイスドライバから受け取ったシステムコール履歴を基に，N-gram による動作パターンの作成を行う．動作履歴より順番に動作パターンの取り出しを行うが，システムコール群をまたぐ場合には動作パターンとしては扱わない．既存の動作規則群から取り出した動作パターンの検索を行い，学習モードと検知モードのそれぞれに応じた処理を行う．

4.3.2 学習モード　動作規則の保存

学習モードにおいて既存の動作規則にないものと判断された動作パターンは，動作規則として追加の処理が行われる．動作規則はプロセス名毎に管理され，スレッド別の記録は行われない．これはスレッド ID は環境によって変化し，常に同じスレッド ID として生成される保証はなく，スレッドを区別することができないためである．動作規則はプロセス名ごとにファイルに分けられ，次の情報を記録する．検知モードではこの動作規則ファイルより動作規則を読みとり，異常検知を行う．

- N-gram による動作規則
- システムコール群規則

N-gram による動作規則の保存方法について説明する．WindowsXP におけるシステムコールの総数は約 1000 種類である．それらには別々のシステムコール ID が割り当てられており，それぞれは 10bit あれば表現することが可能である．本提案手法ではこの 10bit の値を動作規則の順番に並べ，動作規則をひとつの値として表現するものとした．具体的には 64bit の変数を用い，最大 6 つまでのシステムコール ID の記録を行う．この方法を用いると，変数の値比較を行うことで動作規則の比較をすることが可能となり，高速な検索が可能となる．N-gram の定数 N を 6 と制限し，この方法による記録を行った．

4.3.3 検知モード　異常検知

学習モードにて記録した動作規則とシステムコール群に関する情報を基に，3.2 項に示す異常判断基準に基づき，異常の判断を行う．それにより異常と判断された場合，ただちに対象プロセスを停止し，ユーザーへ通知を行う．

第5章

性能評価

提案手法の実装を行い，性能評価を行った．評価は次の性能の計算機にて行った．

- OS:WindowsXP SP3
- CPU:PentiumD 3.0GHz
- メモリ:DDR2-800 2G

システムコール履歴の取得間隔は 0.5 秒，記録対象としたシステムコールを表 5.1 に示す．記録対象としたシステムコールはプログラムの動作様相として特徴的なものと，マルウェアが用いやすいものから選別した．

5.1 オーバーヘッド

本提案手法に生じるオーバーヘッドの内，システムコールフックによるオーバーヘッドと動作規則との比較に必要なオーバーヘッドの測定を行った．

5.1.1 システムコールフックによるオーバーヘッド

本提案手法では監視対象に直接影響を及ぼすオーバーヘッドとしてシステムコールフックによるオーバーヘッドがある．ユーザープログラムがシステムコールを呼び出す際には必ず本提案手法のデバイスドライバを通過するためである．このオーバーヘッ

表 5.1: 記録対象としたシステムコール

システムコール名	ID
NtWriteFile	0x112
NtReadFile	0xB7
NtDeleteKey	0x3F
NtCreatePort	0x2E
NtListenPort	0x60
NtReplyPort	0xC2
NtRequestPort	0xC7
NtCreateProcess	0x2F
NtCreateProcessEx	0x30
NtSuspendProcess	0xFD
NtTerminateProcess	0x101
NtCreateThread	0x35
NtSuspendThread	0xFE
NtTerminateThread	0x102

表 5.2: システムコールフックによるオーバーヘッド

提案手法	実行速度 (秒)
なし	22.58
あり	23.03
倍率	1.02 倍

※記録対象システムコール平均発行数：31000 回

ドはシステムコールを呼び出す回数が多いプログラムほど影響が大きく，オーバーヘッドが大きくなってしまう．

この性能評価ではシステムコールフックによるオーバーヘッドの測定を行った．まず，観測のための性能評価用のプログラムを作成した．性能評価用のプログラムは，スレッド 10000 個を順に作成するプログラムである．それぞれのスレッドではファイルの新規作成を行い，ランダムな値を 1000 回追記する．この性能評価用プログラムを提案手法によって監視し，その平均実行速度を測定した．結果を表 5.2 に示す．

結果は 1.02 倍と比較的小さくすることができた．また，この性能評価用のプログラ

表 5.3: 動作規則検索に必要な時間

検索対象	規則数	検索パターン数	必要時間 (ミリ秒)
N-gram による動作規則	306	301	3
システムコール群規則	315	1	2

ムはその動作のほとんどがシステムコールを発行するものであり，オーバーヘッドは多く見積もってもこの倍率に収まると見ることができる．この評価より，提案手法にはさらに詳細な情報の取得などを行う余裕があるということを確認した．

5.1.2 動作規則検索に必要な時間

動作規則検索に必要な時間の測定を行った．InternetExplorer を対象とし，N-gram 数の定数 N は 6 として測定を行った．動作規則は定数 N の値によらず 64bit の値としてパッケージされている．そのため，ひとつの規則の比較に必要な時間は定数 N に依存しない．測定は InternetExplorer に対し十分な学習をさせ，動作規則として 306 種，システムコール群パターンとして 315 種の記録がなされた状態で行った．システムコールが約 2000 個発行された際の結果を表 5.3 に示す．

この検証では動作履歴の監視間隔を 0.5 秒としている．また，0.5 秒間の内にひとつのプログラムから発行される監視対象システムコールの総数は多くとも 2000 程度であるということを確認してる．これにより，ひとつのプログラムの監視に必要なオーバーヘッドは 5 ミリ秒程度ということがわかる．監視間隔が 0.5 秒の場合，全ての動作規則との照合は 0.5 秒の内に収まる事が理想である．そして，この検証結果より，本提案手法では 100 プロセスまでの監視が可能であるということがわかる．通常，一般の計算機にて稼動しているプロセスは 50 程度であることから，本提案手法は一般の計算機に十分適応可能であるということが確認できる．今後，プログラムの最適化や閾値の調整により，動作規則を少なくすることができると考えているため，更なる実装を期待することができる．

表 5.4: 学習モードによって作成された動作規則

監視対象	行った主な動作	観測 システムコール数	動作規則数	システムコール群 規則数
explorer	新規ファイルの 作成及び削除	96527	87	109
IE7	Web ページの回覧	293725	113	306
Outlook Express	電子メールの送受信	87291	65	85

5.2 検知精度

次に示すプログラムに対し，提案手法によって学習と監視を行い，検知制度の測定を行った．

- explorer
- Internet Explorer 7(IE7)
- Outlook Express

学習モードにて 2 時間動作規則の学習を行わせ，検知モードにて測定を行う．N-gram に用いる定数 N は 3，違反度の検出基準値は 5 とし，違反度は 0.5 秒毎に 1 減少するものとした．学習モードにて作成された動作規則に関する詳細を表 5.4 に示す．

5.2.1 検知率

検知率の測定を行うため，擬似的にマルウェアに侵入された状況を再現し，測定を行った．提案手法に対し，意図的に改変を施した虚偽動作履歴を渡すことによって侵入の再現を行う．その際に渡す虚偽動作履歴は，マルウェアの挙動を模したプログラムの動作履歴より作成を行った．虚偽動作履歴作成に用いたプログラムは次の動作を行うものである．

- レジストリの読み取りと改竄
- 他のプロセスの起動
- スレッドの作成

表 5.5: 検知率

監視対象	違反度に基づく方法	システムコール群に基づく方法
explorer	検知	検知
IE7	検知せず	検知
OutlookExpress	検知	検知

- 外部に対してパケットの送信

作成した虚偽動作履歴を，監視対象プロセスが発行したものとして提案手法に引き渡すことにより，侵入の再現を行う．結果を表 5.5 に示す．

監視対象とした三つのプログラム全てに対し，システムコール群に基づく方法では正常に検知を行うことができた．しかし，違反度に基づく方法では IE7 に対して正常に検知を行うことはできず，他２つの検知のみにとどまる結果となった．この結果となった理由として，違反度に基づく方法の検知率は監視対象プログラムの発行するシステムコールの種類に依存しているということが考えられる．今回の検証に用いたマウルェアの挙動を模したプログラムでは，プロセスの起動やパケットの送信などの動作を行うが，これらの動作は全て IE7 も行う動作である．違反度に基づく方法では，監視対象のプログラムが発行するシステムコールの種類とマルウェアが発行するシステムコールの種類が類似する場合，検知が難しくなる．そして，IE7 は他二つと比べて発行するシステムコールの量が多いという特徴があり，動作規則として多くの動作パターンを許容することとなった．この二つが今回の結果となった理由と考えられる．逆に explorer では外部に対するパケットの送信が少なく，OutlookExpress ではプロセスやスレッドの起動が少ないという点が，検出できた理由である．

システムコール群に基づく方法も，違反度に基づく方法と同じく監視対象が発行するシステムコールの種類に依存する．しかし，システムコール群規則は，発効されるシステムコールの順番に関する規則に加え，その発生頻度に関しても規則として監視している．そのため，システムコール群に基づく方法では全て検知を行うことができたと考えられる．

表 5.6: 誤検知率

監視対象	動作パターン数	違反動作パターン数	違反度に基づく方法	システムコール群に基づく方法
explorer	2815	5	誤検知なし	誤検知なし
IE7	7240	1	誤検知なし	誤検知
OutlookExpress	1404	0	誤検知なし	誤検知なし

5.2.2 誤検知率

誤検知率の測定を行うため，検知モードで 30 分間動作させ測定を行った．誤検知率の指標として，各検出法によって誤検知が生じたかどうかの確認と，動作規則に違反する動作パターンの発生数の測定を行った．結果を表 5.6 に示す．

この検証ではシステムコール群に基づく方法でのみ誤検知が生じる結果となった．動作規則にない動作パターンとして違反動作パターンの数の記録を行ったところ，explorer にて 5 つ，IE7 にて 1 つ発生している．explorer にて発生した違反動作パターン数は違反度の判定基準である 5 と等しいが，これらの違反はそれぞれ別々に発生したものであり，0.5 秒毎の違反度の減少により違反度が 5 となる瞬間は発生しなかった．そのため，explorer では誤検知は発生しなかったと考えられる．

一方，システムコール群に基づく方法では，IE7 に対して誤検知という結果となった．この理由としては IE7 が他二つと比べて動作が複雑であること，IE7 の動作は表示する Web ページに依存することが考えられる．システムコール群は，対象とするプログラムの動作が複雑な場合，正しくモジュールとして分割されない可能性が高くなる．複数の処理が短時間に行われるとそれらをまとめてひとつのモジュールと見なしてしまうためである．また，表示する Web ページにより，動作の変動は大きい．今回の検証では学習モードにて表示を行っていないページの表示も行っている．これらの理由から IE7 のようなプログラムは正しく動作規則を作成することが難しいといえる．システムコール群に基づく方法では，システムコール群の判別に更なる工夫が必要であるということが確認できた．

第6章

まとめと今後の課題

Windows においてシステムコール履歴に基づく異常検知システムの提案を行った．本提案では sysenter 命令フックを用いてシステムコール履歴の取得を行い，異なる二つの判断基準によって異常を判定する．違反度に基づく判定方法は，誤検知を小さくすることが可能であるが，発行システムコール数が多いプログラムに対しては検知が難しい．システムコール群に基づく方法では，プログラムの動作自体を規則化できるため，プログラムの乗っ取りには高い検知率を出すことができる．しかし，プログラムの動作自体が外部の情報に依存していたり，動作が複雑なプログラムに対しては，正しいシステムコール群規則を生成することが難しく，誤検知率が高くなるということが性能評価により確認できた．

今後考えられる実装案として，これらの二つの方式を組み合わせる方法が考えられる．具体的にはシステムコール群をひとつのモジュールとみなし，モジュールの実行順番を N-gram によって動作規則として保存する方法である．この方法はプログラムをより大域的な流れで見ることが可能となり，プログラム乗っ取りに対して更に高い検知率を期待することができる．しかし，この方法もシステムコール群の区別方法が重要な鍵となるため，更なる調整と新たな区別方法が必要である．

改良すべき点として，プログラムの動作規則を実行ファイル名でのみ記録しているという点がある．本提案手法では，実行ファイル名が同じプログラムに対しては全て同一の動作規則を適応する．しかし，システムのサービスの実行を担う「svchost」などのプログラムは，同一の名前で複数同時に実行を行う．それぞれの実行ファイルは

別々の機能を有したものであり，本来別々の動作規則を適応する事が望ましい．この問題の解決案としては，実行ファイルに対しハッシュ値をとり，そのハッシュ値によって管理するという方法が考えられる．

マルウェアの脅威は依然増加する一方であり，その攻撃方法は複雑なものとなってきている．本提案手法はマルウェアの種類に依存せず検知を行うことが可能である．

謝辞

本研究のために，多大なご尽力を頂き，ご指導を賜った名古屋工業大学の齋藤彰一准教授，松尾啓志教授，津邑公暁准教授，松井俊浩助教に深く感謝します．また，本研究の際に多くの助言，協力をして頂いた，齋藤研究室ならびに松尾・津邑研究室の皆様に深く感謝致します．

参考文献

[1] Stephanie Forrest, Steven A. Hofmeyr, Anil Somayaji, and Thomas A. Longstaff: A Sense of Self for Unix Processes, Proceedings of the 1996 IEEE Symposium on Security and Privacy, p. 120. IEEE Computer Society(1996).

[2] 槙本裕司, 齋藤 彰一, 松尾 啓志: システムコールの実行順と実行位置に基づく侵入検知システムの実現, 情報処理学会研究報告 Vol.2007, No.83, pp. 23–30(2007).

[3] 工藤智行:検索エンジンを作る, http://gihyo.jp/dev/serial/01/make-findspot.

[4] インテル R: IA-32 アーキテクチャー・ソフトウェア・デベロッパーズ・マニュアル, http://www.intel.co.jp/jp/download/index.htm.

[5] Microsoft: MSDN Library, http://msdn.microsoft.com/en-us/library/default.aspx.