平成25年度　卒業論文

邦文題目

Windows上における危険な処理の承認機構の提案

英文題目

A Proposal of Approval Mechanisms for Dangerous Processing on Windows

情報工学科　渡邊研究室

(学籍番号: 100430100)

早川　顕太

提出日: 平成26年2月12日

名城大学理工学部

内容要旨

マルウェアは多様化が進み，不正インストールやスパムメールの送信，情報漏えいといった様々な活動を行う．これらの活動はバックグラウンドで行われるため，ユーザがその危険な処理に気づくことができないという課題がある．本稿では，Windows 上において危険な処理の動的なユーザへの承認機構を提案する．プログラムが発行する Windows API を提案システムがフックすることにより，危険な処理が行われる直前にユーザへ承認ダイアログを表示する．これにより，マルウェアがバックグラウンドで行う危険な処理を，ユーザは自身の意図していないものとして拒否することが可能となる．本稿では，さらに，提案方式のプロトタイプシステムを実装し，正規ソフトウェアや独自に入手したマルウェアを用いて実験を行った．そして，実験によって得られた結果や既存技術との定性評価から提案方式の有用性を示す．

目 次

第1章　はじめに

近年，インターネットの急速な発展にともない，マルウェアによる被害がますます増加している．初期のマルウェアは自己顕示を目的として開発され，その活動はシステムの破壊や悪意のあるポップアップの表示等，ユーザから目に見えて分かるものが多かった．一方，現在は金銭を目的としたマルウェアの開発に移り変わっている．マルウェアは，不正インストールによりシステムに駐在し，バックドアによる遠隔操作により DDoS 攻撃やスパムメールの送信，情報漏えい等の活動を行う．これらの活動は，初期のマルウェアの活動と異なり，表面化されることがない．攻撃者はこれらのマルウェアを利用して，企業へ強迫を行い身代金を要求したり，クレジットカード等の暗証番号を盗み出すことで，不正に金銭を入手する．これら多様化したマルウェアの活動はバックグランドで行われるため，ユーザは自身が被害に遭う前に，その活動を認識・防止することができないという課題がある．本論文の目的は，これらマルウェアがバックグランドで行う活動を防止することである．

現在，マルウェアを検出する最も一般的な手法としてパターンマッチング法がある．パターンマッチング法は，事前にマルウェアの特徴的なシグネチャを定義しておき，実行ファイルを静的に検査することで，そのシグネチャを含むプログラムをマルウェアとして検出する手法である．このため，既知のマルウェアについては，高い精度で検出することができ，なおかつ誤検知も発生しにくい手法である．しかしながら，シグネチャが定まらない未知マルウェアを検出できないという課題がある．

未知マルウェアを検出する手法として，事前にマルウェアらしい振る舞いを定義して，それを検出するヒューリスティック法がある．ヒューリスティック法は，実行ファイル内のコードを解析することで静的に振る舞いを検出する静的ヒューリスティック法と，実際にマルウェアを動作させた上で動的に振る舞いを検出するビヘイビア法（動的ヒューリスティック法）がある．ビヘイビア法は，静的ヒューリスティック法に比べ暗号化・難読化されたマルウェアを検出できるという利点があるため注目されている．マルウェア開発者はパッカーと呼ばれる実行ファイルを実行可能なまま圧縮するツールを用いることで，既知マルウェアから容易に暗号化・難読化された亜種マルウェアを作成することができる．特に独自に開発されたパッカーによりマルウェアが暗号化・難読化された場合，静的ヒューリスティック法においては，コードの解析が困難となり，これらのマルウェアを検出できない．これに対し，ビヘイビア法においては，コードが暗号化・難読化されても動作上の振る舞いは変化しないため，これらのマルウェアを検出することができる．

しかしながら，ヒューリスティック法の共通の問題として，そもそもマルウェアらしい振

る舞いを定義することが難しいという問題がある．マルウェアは多様化が進み，様々な活動を行うため，全てのマルウェアを網羅的に検出可能な共通した振る舞いを定義することができない．さらに，マルウェアの個々の活動を検出しようとしても，その多くは正規ソフトウェアにおいても類似した活動が存在するため，マルウェアと正規ソフトウェアの分離が難しい．このことが，ヒューリスティック法において誤検知が絶えない原因となっている．

本論文では，Windows 上において危険な処理のユーザへの承認機構を提案する．これにより，マルウェアがバックグラウンドで行う危険な処理を，ユーザは自身の意図していないものとして拒否することが可能となる．提案方式が承認機構という方式を取ることで，正規ソフトウェアにおいても観測される複数の活動を検出対象にできる．さらに，マルウェアと正規ソフトウェアの分離には，ユーザの判断を借りることで誤検知を少なくすることができる．

以下，2 章で既存のビヘイビア法と承認機構に関する既存技術を紹介し，それらの課題を述べる．続いて，3 章で，承認機構である提案方式を述べる．4 章では，提案方式のプロトタイプシステムの実装方法について述べる．5 章で実験による有用性の評価と，既存技術との定性評価を行う．最後に 6 章において，まとめを行う．

第2章　既存技術

承認機構である提案方式をマルウェア検知手法として見た場合，ビヘイビア法に分類される．従って，マルウェア検知手法の既存技術として，ビヘイビア法を取り上げる．続いて，提案方式は承認機構であるため，承認機構に関する既存技術を取り上げる．

2.1　ビヘイビア法に関する既存技術

ビヘイビア法は未知・暗号化（難読化）マルウェアを検出可能であるが，以下に示す問題がある．

- （問題 1）振る舞い定義の難しさ
  表 2.1 にマルウェアによって行われる可能性がある活動を示す．これから分かるように，マルウェアは様々な活動を行うため，全てのマルウェアを網羅的に検出できるような振る舞いを一概に定義することができない．

- （問題 2）正規ソフトウェアとマルウェアとの分離
  表 2.1 にマルウェアの活動に対して，それに類似した正規ソフトウェアによる活動を示す．これから分かるように，マルウェアによる多くの活動は正規ソフトウェアにおいても類似した活動が存在するため，マルウェアと正規ソフトウェアの分離が難しい．

続いて，ビヘイビア法を用いた研究例を示す．ビヘイビア法はマルウェアの振る舞いを定義する方法により，「マルウェアの特徴的な振る舞いを検出する手法」と「機械学習を用いた検出手法」の 2 つの手法に分類できる．以下の 2.1.1 項と 2.1.2 項で，各手法の研究例とその課題を示す．

2.1.1　マルウェアの特徴的な振る舞いを検出する手法

この手法は，研究者の考察と実験により，ある種のマルウェアに特徴的な振る舞いを定式化し，それをビヘイビア法の検出対象の振る舞いとして定義する手法である．表 2.1 にマルウェアの各々の活動に対して，この手法を用いてその活動を検出する既存研究を示す．既存研究として，まずワームの特徴的な振る舞いである自己複製を検出する研究が挙げられる [1] [2]．侵入挙動の反復性を用いたボット検知方式は，ワームは実行環境を復元して再度実行することで，侵入挙動が反復されるため，この特徴的な挙動を検出する手法である [1].

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はワームの自己複製の際，自身の実行ファイルを READ する必要があるため，この挙動を検出する手法である [2]．他の研究例として，動的 API 検査方式によるキーロガー検知方式は，Windows 上においてキーロガーに特徴的であるキー入力を取得するという挙動を網羅的に定式化し，それを検出する [3]．また，近年のマルウェアは，セキュリティソフトを強制終了する等のセキュリティ無効化攻撃を行うものや，デバッカを検知し活動を抑制する等の耐解析機能を持つものがある．これらのマルウェアの機能を逆用して，マルウェアを検知，あるいは活動を防止する研究が存在する [4] [5]．また，既に商用化されたソフトウェアとして Tripwire [6] がある．Tripwire はハッシュ値の比較によって，ファイルの整合性を監視するソフトウェアである．これにより，ファイル改ざんや，アプリ／カーネルの改ざん（あるいは，ウイルスによるファイル感染）を検出することが可能である．また，PC 内のファイル改ざんを行うマルウェアの検知手法はランサムウェアというマルウェアによるファイル改ざんを防止する [7]．ランサムウェアは PC 内のファイルやシステムを勝手に使用不能に（暗号化）し，その復旧と引き換えに金銭を要求するマルウェアである．この論文では，暗号化されたファイルには特徴的なパターンが検出できることを利用して，承認なしにファイル暗号化を行ったプロセスをランサムウェアとして検知する．

これらの既存研究は，いずれもビヘイビア法の問題 1 のために，マルウェアの個々の活動を検出対象にした手法である．そのため，検出対象となるマルウェアの範囲はその活動を行うマルウェアに限定されてしまう．また，研究にもよるがビヘイビア法の問題 2 のために，完全にマルウェアと正規ソフトウェアを分離できているわけではない．

2.1.2　機械学習を用いたマルウェア検出手法

この手法は，ビヘイビア法の振る舞い定義にシステムコールの発行履歴による機械学習を用いる手法である [8] [9]．機械学習を用いることにより，マルウェアに特徴的な全ての振る舞いを検出対象にできるため，ビヘイビア法の問題 1 を解決している．System Service 監視による Windows 向け異常検知システム機構は，ホワイトリスト方式による異常検知であり，事前に正常なソフトウェアにおいてシステムコールの履歴を N-gram 法によって機械学習する．実環境において，システムコールが呼ばれた際，最近 N の長さのシステムコールの履歴を参照しそれが学習されていなければ，それを異常として検知する手法である [8]．危険なシステムコールに着目した Windows 向け異常検知手法は，まず，システムの可用性に影響を与えるクリティカルなシステムコールを定義する．そして，事前に正規ソフトウェアとマルウェアの両方においてクリティカルなシステムコールが呼び出される直前のシステムコールの履歴を N-gram で表現し，それに対して SVM(Support Vector Machine) によって教師あり機械学習を行う．その後，実環境においてリアルタイムでクリティカルなシステムコールを検出し，このときの最近 N の長さのシステムコールの履歴から SVM による識別を行いマルウェアを検知する手法である [9].

表 2.1　マルウェアと正規ソフトウェアによる活動

マルウェアによる活動	正規ソフトウェアによる類似活動	既存研究
不正インストール	インストール，コピー，解凍，等	なし
不正インストール	インストール	なし
スパムメール	メール送信	なし
セキュリティソフト等の強制終了	タスクマネージャ等による強制終了	[4]
掲示板への不正投稿	掲示板への正規投稿	なし
キーロギング	キーバインド	[3]
ファイル破棄・改ざん	ファイル削除・更新	[6], [7]
情報漏えい	アップロード，メール送信，等	なし
バックドア／ボットによる遠隔操作	IRC クライアント等による通信	なし
DDoS 攻撃	サーバへの正規リクエスト送信	なし
悪質な広告表示	試用版ソフトウェア	なし
実行ファイルへの感染	ソフトウェアのアップデート	[6]
自己複製	なし	[1], [2]
アプリ／カーネルの改ざん	なし	[6]

これらの既存研究は，ビヘイビア法の問題 2 のため，いずれもある程度の誤検知が生じてしまうという課題がある．また，機械学習によりマルウェアを検知するため，検出時，なぜそのプログラムがマルウェアとして検出されたのかをユーザに説明できないといった課題がある．

2.2　承認機構に関する既存技術

現在，著者が知る限り Windows 上において動的な承認機構を提供するといった研究はない．承認機構に関連した既存技術としては，Windows Vista 以降に導入されたユーザアクセス制御（UAC；User Access Control）が挙げられる．UAC により，例え管理者のユーザとしてログインしても，昇格プロンプトによるユーザの承認を得ない限り，標準ユーザの権限として動作する．これにより，マルウェアがシステム全体に悪意のある変更を加えることを防止できる．

しかし，UAC はプログラムの起動時に行われる承認機構であり，プログラムの実行中に行われる動的な承認機構ではない．従って，昇格プロンプトを表示した時点では，実際に行われる処理の内容やそのタイミングをユーザが把握することができない．また，標準ユーザの権限で行える，カレントユーザのみへのシステムの変更や，メールの送信などを防ぐことができないといった課題がある．

第3章　提案方式

3.1　概要

本稿では，Windows 上における動的な危険な処理のユーザへの承認機構を提案する．図 3.1 に提案システムの概要を示す．アプリケーションが危険な処理を行うために発行する Windows API を提案システムがフックすることで，その危険な処理が行われる直前に，ユーザへの承認ダイアログを表示する．ユーザは行われていようとしている危険な処理が自分の意図したものであるかどうかによって，その処理の許可／拒否を選択する．ユーザの応答により，提案システムはその処理を続行するか，あるいは処理を中断させアプリケーションにエラーを返す．これにより，マルウェアがバックグラウンドで行う危険な処理を，ユーザは自身の意図していないものとして拒否することが可能になる．

提案方式が承認機構として，満たすべき 4 つの要件は以下の通りである．

＜要件 1＞ 検出対象となる処理は，マルウェアの悪意のある活動であること．

＜要件 2＞ ユーザは処理を許可／拒否するための正しい判断が可能であること．

＜要件 3＞ 承認ダイアログ内に，ユーザの理解の助けとなる情報を提示すること．

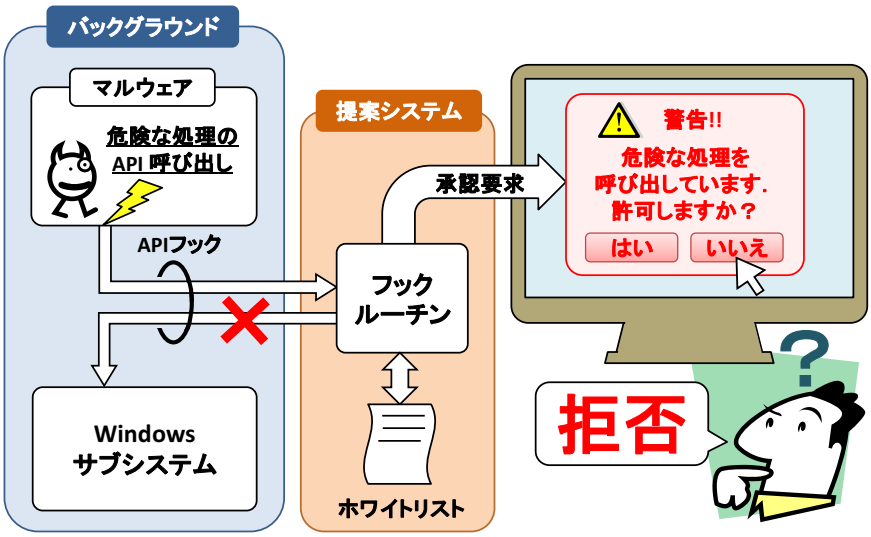


図 3.1　提案方式の概要

＜要件 4 ＞ 承認機構により，ユーザビリティが著しく損なわれないようにすること．

これらの要件を満たすように，提案方式の詳細を以下に述べる．

3.2 危険な処理の定義

提案方式の＜要件 1 ＞と＜要件 2 ＞を満たすように，危険な処理を定義する．＜要件 1 ＞はマルウェアによって悪用されることがない処理に対して，承認を求める必要がないという意味である．＜要件 1 ＞を満たすため，まず，表 2.1 に示したマルウェアが行う活動を危険な処理の候補として挙げることができる．＜要件 2 ＞とは，承認ダイアログが表示された際，その行われようとしている処理がユーザに身に覚えのないものならば，ユーザはそれを拒否できること（すなわち，そのプログラムがマルウェアであること）を意味する．この要件を満たしていれば，提案方式はユーザによる誤検知を少なくできる．この要件を満たすために，検出対象とする処理が次の条件を満たす必要がある．

「危険な処理の条件：任意の正規ソフトウェアは，ユーザが意図したタイミングのみにその処理を行う．」

これは，つまり正規ソフトウェアがバックグランドで行う処理を検出対象とすべきではないという意味である．正規ソフトウェアがバックグラウンドで行う処理を検出してしまうと，ユーザはその処理がマルウェアによるものなのか正規ソフトウェアによるものか判別できずに，誤検知を起こしてしまう可能性がある．

以上の考察から，＜要件 1 ＞の表 2.1 に示したマルウェアが行う活動の内，正規ソフトウェアの類似活動が＜要件 2 ＞の危険な処理の条件を満たしている処理を，最終的な危険な処理として定義できる．

これにより，仮の危険な処理として表 3.1 を定義した．今後，実験により，これらの処理が＜要件 2 ＞の危険な処理の条件を満たしているかを確かめる必要がある．

表 3.1 危険な処理

実行ファイルの作成
自動実行への登録
他プロセスの強制終了
キー入力の取得
HTTP の HOST
メール送信

3.3 承認ダイアログについて

上記に定義した危険な処理をフックし，その処理が行われる直前に，ユーザへの承認ダイアログを表示する．承認ダイアログのイメージ図を図 3.2 に示す．承認ダイアログ内には以下の情報を表示する．

1. プロセスに関する情報
    - プロセス名
    - プロセス ID
    - 実行ファイルの絶対パス
    - 実行ファイルのアイコン
    - 電子署名の発行元
2. 行おうとしている危険な処理の内容
3. プロセスが表示しているウインドウ（複数存在する可能性あり）
4. ユーザの選択肢

提案方式の＜要件 3 ＞を満たすための情報が 1.〜3. である．2. は現在行われようとしている危険な処理の内容であり，ユーザはこれが自身の意図したものであるかどうかによって，その処理の許可／拒否を判断する．1. はユーザが危険な処理を呼び出しているプロセスを一意に特定するための情報である．これに加え，3. を表示することでユーザはそのプロセスを直観的に把握することができる．承認ダイアログが表示されると，ユーザはその後の処理を 4. から選択する．選択肢とその選択場面，選択後の提案システムの動作を表 3.2 に示す．

3.4 ユーザビリティ向上のために

提案方式は承認ダイアログにおいて，「ホワイトリストへ登録」や「この実行に限り常に許可」を選択することより，承認ダイアログを永続的に，あるいは一時的に省略できるようにする．これにより，ユーザビリティを向上させ，提案方式の＜要件 4 ＞を満たすようにする．

提案方式はホワイトリストを導入する．ユーザは承認ダイアログの「ホワイトリストへ登録」を選択することにより，そのプログラムをホワイトリストへ登録する．ホワイトリストへ登録されたプログラムは，OS やプログラムを再起動しても，その処理に関して承認ダイアログが永続的に省略される．ホワイトリストにはプログラムの絶対パスと常に許可したい危険な処理（複数可）が登録される．ホワイトリストの例が表 3.3 である．例えば，エクスプローラはユーザのドラック&ドロップにより実行ファイルをコピーすることがあるため，実行ファイルの作成をホワイトリストに登録する．また，タスクマネージャはユーザの意図

によって他のプロセスを強制終了すことがあるため，他プロセスの強制終了を登録する．インターネットエクスプローラについては，実行ファイルをダウンロードしたり，掲示板へ書き込みを行ったりすることがあるため，実行ファイルの作成や HTTP の POST を登録する．

ユーザは承認ダイアログの「この実行に限り常に許可」を選択することにより，そのプロセスはプロセスが終了するまで，その処理に関して承認ダイアログを一時的に省略することができる．「この実行に限り常に許可」はインストーラでの使用を想定している．インストーラは，通常複数の実行ファイルを作成するため，その都度承認ダイアログを表示していてはユーザビリティを損ねる．しかし，通常，インストーラは一度実行しその後，捨ててしまうためホワイトリストへ登録することも適切ではない．そのため，「この実行に限り常に許可」により適切な許可を与えることができる．

ただし，ホワイトリストに登録したプロセスやその実行に限り処理が許可されたプロセスに対して，マルウェアがスレッドを注入することで，そのスレッドが許可された危険な処理を自由に行えてしまう．従って，これらのプロセスに対してスレッドの注入や仮想メモリの書き込みなどを禁止することで，マルウェアから保護する必要がある．

表 3.2　承認ダイアログにおけるユーザの選択肢

選択肢	選択場面	提案システムの動作
許可	ユーザがその処理を自身で意図した場合	その処理を一度だけ許可
拒否	ユーザがその処理に身に覚えがない場合	その処理を拒否
強制終了	ユーザがそのプログラムをマルウェアと判断した場合	そのプロセスを強制終了
この実行に限り常に許可	ユーザがそのプログラムを安全であると判断した場合	そのプロセスの終了時まで，その処理を常に許可し，承認ダイアログを省略
ホワイトリストへ登録	ユーザがそのプログラムを安全であると判断した場合	OS やプログラムの再起動後も，そのプログラムが行うその処理を常に許可し，承認ダイアログを省略

表 3.3　ホワイトリストの例

登録されたプログラム	許可された処理
C:/Windows/explorer.exe	（実行ファイルの作成）
C:/Windows/System32/taskmgr.exe	（他プロセスの強制終了）
C:/Program Files/Internet Explorer/iexplore.exe	（実行ファイルの作成）＋（HTTP の POST）

図 3.2　承認ダイアログのイメージ図

第4章　実装

提案方式の有用性を評価するため，プロタイプシステムを実装した．そこで，本章ではプロトタイプシステムの実装の詳細を述べる．

4.1　実装方法の検討

提案方式は API フックを用いて実装を行う．API とは Application Programming Interfaces の略で，アプリケーションがオペレーティングシステムの機能を利用するために呼び出すインタフェースである．API フックとは，アプリケーションが呼び出す API を横取り（フック）し，独自の処理を行こなわせることである．マルウェアもアプリケーションの一つであるため，危険な処理を行うには API を呼び出す必要がある．従って，API フックを用いれば，マルウェアが呼び出す危険な処理を検出できる．提案方式は API フックによる独自処理において，ユーザへの承認ダイアログを表示する．

API フックにはいくつかの方法ある．付録 A に，それらの比較・検討を行った結果を示す．その結果，プロトタイプシステムの実装には，実装が容易で比較的にフックの回避が難しい Detours ライブラリを採用することにした．Detours ライブラリとは MicrosoftReserch が提供する API フックライブラリである [10]．Detours ライブラリは，プロセスの仮想メモリ上にロードされたフック対象の API の先頭の命令をフック関数への JMP 命令に書き換えることでフックを行う（以降，この API フックの方法を Detours フックと呼ぶ）．

Detours フックはユーザモードで行われるプロセス単位の API フックであるため，任意のプロセスの API をフックするためには，動作中の各プロセスに対して Detours フックを行う必要がある．プロトタイプシステムでは，これを実現するために DLL インジェクションを採用する．DLL インジェクションとは，DLL を対象プロセスに強制的に注入するこという．Detours ライブラリを用いて API フック用の DLL を作成し，これを DLL インジェクションにより動作中の各プロセスに注入することで，任意のプロセスの API フックを実現する．DLL インジェクションにも，いくつかの方法があるため，これも付録 B において比較・検討を行った．採用した DLL インジェクションの方法とそれを行うタイミングは 4.2 節においてその都度述べる．

また，プロトタイプシステムでは危険な処理を呼び出したプロセスが承認ダイアログを表示するように設計したため，監視対象となるプロセスをカレントユーザが所有するプロセスに限定する．その理由は，Windows Vista 以降，Windows サービスなどのシステムプロセス

は「セッション 0 の分離」により，GUI（つまり，承認ダイアログ）を表示することができないためである．現状では，これらシステムプロセスにはそもそもフック用 DLL を注入しないで，監視対象外とする．

4.2 プロトタイプシステムの構成

プロトタイプシステムは，DllInjector という駐在プロセスと，フック用 DLL（Hooker.dll）の 2 つから構成される．プロトタイプシステムの全体像が図 4.1 になる．なお，プロトタイプシステムは 32 ビット版の Windows OS を対象としており，DllInjector とフック用 DLL は共に 32 ビット版の実行可能ファイルである．次にプロトタイプシステムの各要素を説明する．

- フック用 DLL（Hooker.dll）：フック用 DLL は監視対象となる全てのプロセスで，強制的にロードされる DLL である．フック用 DLL には，検出対象となる API 毎にフックルーチンを実装する．そして，DLL の初期化関数内で Detours ライブラリを呼び出し，検出対象の API をそれに対応するフックルーチンへとフックする．検出対象となる API を以下に述べる．
  - 危険な処理に対する API：表 3.1 に示した危険な処理に対して，それを検出可能な API をフックする．フック対象となる API の詳細は 4.3 節で述べる．これら API に対する各フックルーチン内で，承認ダイアログを表示し，ユーザへの承認機構を提供する．現在はメッセージボックスによる簡易的な承認ダイアログを実装している．
  - 子プロセスを生成する API：親プロセスから子プロセスへ DLL インジェクションを行うために，プロセスを生成する API である CreateProcess API フックする．この API をフックして，CreateProcess API の代わりに Detours ライブラリの DetourCreateProcessWithDll 関数を呼び出す．この関数はプロセスを生成する CreateProcess API の代替となる関数で，指定した DLL を注入した状態でプログラムを起動する関数である．これにより，既にフック用 DLL が注入済みのプロセスから CreateProcess API によりプロセスが生成される場合，子プロセスへ起動時からのフック用 DLL の注入が成功する．例えば，Windows エクスプローラにフック用 DLL の注入が行われれば，それ以降，エクスプローラからダブルクリックにより実行される全てのプロセス（一部例外あり [1]）は，この方法で自動的にフック用 DLL が注入されることになる．
  - その他：マルウェアの他プロセスへの注入を監視するため，CreateRemoteThread API をフックし，メッセージボックスによる警告を出すようにする．

[1] Windows Vista 以降，管理者権限が必要なプログラムをダブルクリックで実行した場合，そのプログラムは最終的に CreateProcess API 以外の API で起動される

- DllInjector：フック用 DLL が注入されていないシステムプロセスから起動されるプロセスや，CreateProcess API 以外の API で生成されたプロセスは，フック用 DLL がロードされていない．DllInjector はこれらのプロセスに強制的にフック用 DLL を注入する役割を持つ駐在プロセスである．DllInjector は OS 起動時から管理者権限でシステムに駐在し，動作中の各プロセスを監視する．そして，定期的に各プロセスにフック用 DLL が注入されているかを調査し，注入されていない場合，そのプロセスへ DLL インジェクションを行う．ただし，前述したシステムプロセスや自身のプロセスへの DLL インジェクションは行わない．このとき，使用する DLL インジェクションは RemoteCreateThread API と LoadLibrary API を組み合わせた方法を用いる．この DLL インジェクションの方法は，実行中のプロセスに DLL を注入する方法であるため，プロセス起動直後から API フックを行えるわけではない．DllInjector のプロセスの監視間隔は長すぎると，API フックを仕掛けるタイミングが遅くなってしまう．一方，短すぎてもシステムに大きな負荷を与えてしまう恐れがある．現在，監視間隔は検討中ではあるが 500 ミリ秒としている．

また，有用性の評価には関係ない部分ではあるが，提案方式のホワイトリストにはレジストリを用いた簡易的な実装を行った．

以上がプロトタイプシステムの構成である．このプロトタイプシステムは，実用を考慮したものではなく，提案方式の有用性を評価するためのものである．従って，現状はプロトタイプシステムが Windows システムに与える負荷や，マルウェア開発者がプロトタイプシステムの実装を理解した上で回避策を立てることを考慮していない．

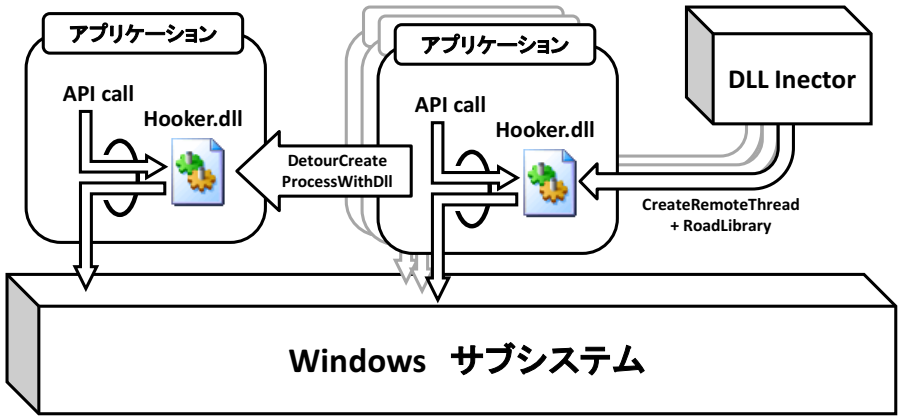


図 4.1　プロタイプシステムの構成図

4.3 危険な処理として検出する API

この節では危険な処理として定義した表 3.1 に対して，実際にどの API をフックするかを述べる．しかし，現在，全ての危険な処理について，その検討を終えているわけではない．現在は表 3.1 の内，実行ファイルの作成と OS 起動時の自動実行への登録が実装済みである．それ以外の危険な処理は，今後，検討・実装を行う予定である．実行ファイルの作成と OS 起動時の自動実行への登録を検出するために，フックする API を表 4.1 に示す．各々の危険な処理について，そのフック対象となる API が指定した引数の条件を満たして呼ばれた際に，承認ダイアログを表示する．OS 起動時の自動実行への登録に関しては，レジストリ，スタートアップフォルダ，タスクスケジューラによるいずれかの方法があるが，プロトタイプシステムではレジストリを用いた方法のみをターゲットとして実装した．また，レジストリを用いた場合にも，自動実行に関するレジストリエントリが複数存在するため，マルウェアがよく利用するレジストリエントリである表 4.2 を検出対象とした．

表 4.1　危険な処理に対して検出対象となる API

危険な処理	フック対象となる API	API の引数の条件
実行ファイルの作成	NtCreateFile	ファイルを新たに生成し，そのファイルの拡張子が”.exe”である場合
	NtSetInformationFile	操作がファイル名のリネームで，リネーム後のファイルの拡張子が”.exe”である場合（ただし，リネーム前のファイルの拡張子が”.exe”の場合を除く）
OS 起動時の自動実行への登録	NtSetValueKey	キー名やエントリ名に表 4.2 示した文字列を含むレジストリエントリへ書き込みを行う場合

表 4.2　検出対象とした自動実行に関するレジストリエントリ

キー名	エントリ名
SOFTWARE/Microsoft/Windows/CurrentVersion/Run	—
SOFTWARE/Microsoft/Windows NT/CurrentVersion/Winlogon	Shell
SOFTWARE/Microsoft/Windows NT/CurrentVersion/Winlogon	Userinit
SOFTWARE/Microsoft/Active Setup/Installed Components	StubPath
SYSTEM/ControlSet001/Services	ImagePath
SYSTEM/ControlSet002/Services	ImagePath

第5章　評価

この章では，提案方式の有用性の評価を行う．まず，5 章で述べたプロトタイプシステムを用いて，正規ソフトウェアとマルウェアの双方で実験を行い，提案方式の有用性を示す．続いて，2 章で述べた既存技術と提案方式で定性評価を行う．

5.1　実験

5.1.1　実験目的と実験環境

まず，正規ソフトウェアとマルウェアの各々における実験の目的を述べる．

- ＜正規ソフトウェアにおける実験の目的＞
  - （目的その 1）
    提案方式の要件として，検出対象となる処理は危険な処理の条件を満たしている必要がある．つまり，任意の正規ソフトウェアにおいてユーザの意図しないタイミングで，その処理が呼び出されてはいけない．実験により，この条件を満たしているかを調査する．
  - （目的その 2）
    危険な処理を監視するために検出対象とした API が，正しく危険な処理を検出し，検知漏れを起こしていないかということを調査する．
- ＜マルウェアにおける実験の目的＞
  実際にどの程度のマルウェアが提案方式で検出可能であるか（つまり，承認ダイアログを表示させることができるか）を調査する．

実験環境は仮想マシン上の 32 ビット版の Windows XP SP3 である．ゲスト OS 上に提案方式のプロトタイプシステムを導入し，正規ソフトウェアとマルウェアのそれぞれで実験を行う．このときのゲスト OS のネットワーク設定は，正規ソフトウェアにおける実験では”NAT”，マルウェアにおける実験では，”ホストオンリー”である．

5.1.2　正規ソフトウェアに関する実験

この実験では，プロトタイプシステムの他に ProcMon というファイルやレジストリ等の処理をリアルタイムで表示するツールを用いた [11]．ProcMon において，実行ファイルの

作成と OS 起動時の自動実行の登録が検出できるようにフィルタを掛け，危険な処理を監視する．

この監視下において，様々な正規ソフトウェアを実行・操作した．ProcMon が実行ファイルの作成あるいは OS 起動時の自動実行の登録を検出した際，それをユーザの行った直前のイベントと関連づけれるかどうかにより，正規ソフトウェアによる実験の（目的その 1）を調査する．また，ProcMon の一部はカーネルで動作し，全てのプロセスの処理を監視できる．そこで，ProcMon において検出対象の処理が，検出されたとき，プロトタイプシステムによる承認ダイアログが表示されたか否かを確認することで，プロトタイプシステムに検出漏れがないかどうか，つまり，正規ソフトウェアによる実験の（目的その 2）を調査する．

実行ファイル作成における実験結果が表 5.1，OS 起動時の自動実行における実験結果が表 5.2 である．それぞれの表で，その検出対象となる処理が呼ばれた際のユーザの直前のイベントと，プロトタイプシステムによる検知漏れの有無を示す．

以下に各々の実験の目的において実験結果の考察を行う．

- （実験目的その 1）の結果
  実験結果から，危険な処理が検出された際，いずれもその検出をユーザの直前のイベントと関連付けることができ，ユーザはそのイベントからその危険な処理を行うことであろうことを推測可能である．例えば，表 5.1 において，Dropbox のインストーラを実行した際に，実行ファイル作成に関して承認ダイアログが表示されている．ユーザは実行ファイルを作成することを想定してインストーラを実行しているため，実行ファイルを作成しようとしていることを通知する承認ダイアログが表示されていても正しく許可を発行できる．このように，今回実験を行った実行ファイルの作成と OS 起動時の自動実行の登録については，ユーザの意図したタイミングのみにその処理が行われている．従って，これらの処理は危険な処理として定義できる．

- （実験目的その 2）の結果
  実験結果から，プロトタイプシステムには検知漏れが存在することが判明した．例えば，表 5.1 の Skype インストーラや Google Chrome インストーラ実行において，実行ファイル作成の検知漏れが存在する．また，表 5.2 の sc コマンドによるサービスの登録においても，自動実行レジストリへの登録の検知漏れが存在する．これらの原因は，プロトタイプシステムによりフック用 DLL が注入されていないシステムプロセス，あるいはサービスがアプリケーションに代わって，危険な処理を行ったからである．従って，マルウェアが同様にシステムプロセス，あるいはサービスを介して危険な処理を行うことで，プロトタイプシステムを回避できてしまう．しかしながら，カーネルモードのフックを用いて提案方式を実装すれば，これらを防止できる．そのため，今後はそのような実装も考慮する必要がある．

表 5.1　実行ファイル作成における実験結果

実行ファイル作成が観測された際のユーザのイベント	検知漏れ
インストーラ (Dropbox) の実行	○
インストーラ (Rainlendar) の実行	○
インストーラ (Lhaplus) の実行	○
インストーラ (Google Chrome) の実行	△
インストーラ (Skype) の実行	×
Lhaplus による実行ファイルを含む ZIP の解凍	○
Web ブラウザ (IE) による実行ファイルのダウンロード	○
Web ブラウザ (Chrome) による実行ファイルのダウンロード	○
エクスプローラによる実行ファイルのコピー (ドラッグ&ドロップ)	○
コマンドプロンプトの copy コマンドによる実行ファイルのコピー	○
Visual Studio C++ 2010(link.exe) によるビルド	○

表 5.2　自動実行への登録における実験結果

自動実行への登録が観測された際のユーザのイベント	検知漏れ
インストーラ (Skype) の実行	○
インストーラ (Rainlendar) の実行	○
インストーラ (Chrome) の実行	○
Skype の設定による自動実行への変更	○
レジストリエディタによる自動実行に関するエントリ値への書き込み	○
reg コマンドによる自動実行に関するエントリ値への書き込み	○
sc コマンドによるサービスの登録	×

5.1.3　マルウェアに関する実験

実験に使用したマルウェアは，マルウェア収集サイトである Offensive Computing [12]　と VX Vault [13] から独自に収集したものの内，ウイルス対策ソフトである Symantec Endpoint Protection により検出された実行可能なマルウェア 53 体である．なお，Symantec 社によるマルウェアの検出名が同名であり，実験結果が等しいものは省いてある．

マルウェアにおける実験結果を表 5.3 に示す．実験の結果，53 体中 31 体ものマルウェアがバックグランドで実行ファイルを作成，あるいは OS 起動時の自動実行への登録を行い，プロトタイプシステムによって承認ダイアログを表示させることができた．従って，これらのマルウェアに対しては，提案方式が有効であると判断できる．また，31 体の内 2 体のマルウェアはエクスプローラにスレッドを注入し，エクスプローラに危険な処理を行わせるものであった．53 体中 12 体のマルウェアは，承認ダイアログが表示されずに実行が続いたが，これらのマルウェアは不正インストールを行わないマルウェアであるといえ，比較的に

安全であるといえる．ただし，この実験では正規ソフトウェアの実験で行った ProcMon による監視を行っていないため，検知漏れが生じているかどうかについては未確認である．その他，53 体中 8 体のマルウェアはプロトタイプシステムを導入したことにより，実行時にエラーが発生した．これについては，今後調査する必要がある．

今回の実験は危険な処理としてインストール（実行ファイルを作成と OS 起動時の自動実行への登録）のみを対象とした．しかしながら，先述のような高い検出率を得ることができた．今後，その他の危険な処理を実装することにより，さらなる検出率の向上が期待できる．

表 5.3　各 API フックによる定性評価

マルウェアの検体数（計 53 体）	結果
31	承認ダイアログが表示される
12	承認ダイアログは表示されず，実行を続ける
2	マルウェアによりシステム全体が操作不能に陥る
8	エラーにより実行できず

5.2　既存技術との定性評価

5.2.1　既存の承認機構との比較

承認機構の既存技術である UAC との定性評価を行う．UAC はプログラム起動時の承認機構であるため，昇格プロンプトを表示した時点では，実際に行われる処理やそのタイミングが分からなかった．しかし，提案方式はプログラムの実行中に行われる承認機構であるため，危険な処理を行う直前に，その処理内容を反映した承認ダイアログを表示できる．また，UAC では防ぐことができなかったカレントユーザへのインストールやメールの送信を，提案方式では防ぐことができる．

5.2.2　ビヘイビア法の既存研究との比較

マルウェアの特徴的な振る舞いを検出する手法と機械学習を用いたマルウェア検出手法に提案方式を加えた 3 手法で定性評価を行った．評価結果を，表 5.4 に示す．

- 誤検知：マルウェアの特徴的な振る舞いを検出する手法では，研究にもよるが基本的にマルウェア固有の振る舞いを検出するため，誤検知は比較的に少ない．機械学習を用いたマルウェア検出手法では，現状ある程度の誤検知が生じてしまう．提案方式は，ユーザ次第であるが，危険な処理を正規ソフトウェアがバックグラウンドで行わない処理に限定したため，ユーザによる誤検知はないと考えられる．

- 振る舞いの検出範囲：マルウェアの特徴的な振る舞いを検出する手法では，その特徴的な振る舞いを持つマルウェアしか検出対象にできないため，検出範囲は狭い．機械学習を用いたマルウェア検出手法では，システムコールの発行履歴に着目するため，検出可能な振る舞いの範囲は広い．また，一般には知られていないようなマルウェア固有の振る舞いを検出対象にできる．提案方式は，全てではないにしろ，危険な処理として定義した複数の振る舞いを検出対象にできる．

- 検出時のユーザへの説明：この評価項目は，その手法によりマルウェアが検出された際，なぜマルウェアとして検出されたのかをユーザに説明できるかという点に着目した評価である．ユーザに説明することができれば，ユーザに一種の安心感を与えることができ，より良いと考えられる．マルウェアの特徴的な振る舞いを検出する手法は，研究にもよるが，検出する振る舞いが特徴的であるがために，PC に詳しくないユーザが検出理由を理解することが難しいと考えられる．機械学習を用いたマルウェア検知手法では，機械学習により識別を行うため，ユーザへ検出理由を説明することができない．提案方式は，ユーザがマルウェアの判断を行うため，そもそもユーザへ説明する必要がない．

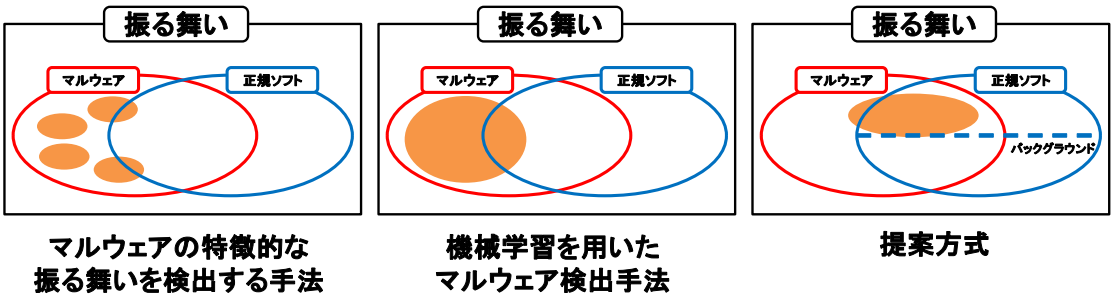
- ユーザビリティ：従来のビヘイビア法（マルウェアの特徴的な振る舞いを検出する手法と機械学習を用いたマルウェア検知手法）はプログラムが自動的にマルウェアの判断を行うため，ユーザビリティは損なわれない．しかし，提案方式は承認機構である．このため，危険な処理を呼ぶ度にユーザへの承認ダイアログが表示するためユーザビリティが損なわれる．しかしながら，提案方式はホワイトリストを導入するため，著しくユーザビリティが損なわれることはない．

図 5.1 に各手法の検出可能な振る舞いの範囲を示す．表中の橙色の楕円が，一つの研究において検出される振る舞いの範囲を示している．この図から分かるように，提案方式は，従来のビヘイビア法では検出できなかった，あるいは検出しても誤検知を起こしてしまう振る舞いを，ユーザの判断を借りることにより誤検知なしに検出対象にできる．従って，提案方式は従来とは異なるマルウェアの振る舞いを検出する新しいアプローチであるといえる．

以上の比較結果から提案方式は，従来のビヘイビア法より有用な手法であるといえる．

表 5.4　提案方式と従来のビヘイビア・ブロッキング法の定性評価

評価項目	マルウェアに特徴的な振る舞いを検出する手法	機械学習を用いたマルウェア検知	提案方式
誤検知	△〜○	△	○
振る舞いの検出範囲	×	○	△
検出時のユーザへの説明	△	×	○
ユーザビリティ	○	○	△



マルウェアの特徴的な振る舞いを検出する手法　　機械学習を用いたマルウェア検出手法　　提案方式

図 5.1　各手法の検出可能な振る舞いの範囲

第6章　まとめ

本論文では，Windows における危険な処理のユーザへの承認機構を提案した．まず，マルウェアの活動と正規ソフトウェアの類似活動を比較することで，危険な処理としてどのような処理を検出すればよいか検討を行った．また，ユーザビリティが損なわれないようにするため，ホワイトリストを導入することを提案した．さらに，提案方式のプロトタイプシステムと危険な処理の一部を実装し，実験を行った．その結果，これらの危険な処理が正規ソフトウェアで行われる場合は，ユーザの判断を借りることにより誤検知少なくできることを示した．さらに，多くのマルウェアがバックグランドで行う危険な処理を，プロトタイプシステムにより検出できることを確認し，提案方式の有用性を示した．今後は，他の危険な処理についても実験と実装を行い，より多くのマルウェアを検出可能にしていく予定である．また，提案システムについての実装方法の再検討を行い提案システムをより強固なものにしていく．

謝辞

本研究にあたり，多大なる御指導と御教授を賜りました，渡邊晃教授には心から感謝いたします．また，本研究を進めるにあたり，御意見ならびに御助言を受け賜りました，鈴木秀和助教，旭健作助教に心から感謝いたします．最後に，本研究を進めるにあたり，数々の有益な御助言や御討論を賜りました，渡邊研究室，鈴木研究室の諸氏に感謝します．

参考文献

[1] 酒井崇裕，竹森敬祐，安藤類央，西垣正勝：侵入挙動の反復性を用いたボット検知方式，情報処理学会論文誌，Vol. 51, No. 9, pp. 1591–1599 (2010).

[2] 松本隆明，鈴木功一，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，情報処理学会論文誌，Vol. 48, No. 9, pp. 3174–3182 (2007).

[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，Vol. 48, No. 9, pp. 3137–3147 (2007).

[4] 松木隆宏，新井　悠，寺田真敏，土居範久：セキュリティ無効化攻撃を利用したマルウェアの検知と活動抑止手法の提案，情報処理学会論文誌，Vol. 50, No. 9, pp. 2127–2136 (2009).

[5] 松木隆宏，新井　悠，寺田真敏，土居範久：マルウェアの耐解析機能を逆用した活動抑止手法の提案，情報処理学会論文誌，Vol. 50, No. 9, pp. 2118–2126 (2009).

[6] : Tripwire, http://www.tripwire.co.jp/.

[7] 澤村隆志，ベッド B．ビスタ，高田豊雄：PC 内のファイル改ざんを行うマルウェアの検知手法，研究報告コンピュータセキュリティ（CSEC），Vol. 56, No. 11, pp. 1–7 (2012).

[8] 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, pp. 420–429 (2006).

[9] 伊波　靖，高良富夫：危険なシステムコールに着目した Windows 向け異常検知手法，情報処理学会論文誌，Vol. 50, No. 9, pp. 2173–2181 (2009).

[10] : Microsoft Research：Detours，http://research.microsoft.com/en-us/projects/detours/.

[11] : Process Monitor, http://technet.microsoft.com/ja-jp/sysinternals/bb896645.aspx.

[12] : Offensive Computing, http://www.offensivecomputing.net/.

[13] : VX Vault, http://vxvault.siri-urz.net/ViriList.php.

[14] : listexp2, http://www.chiyoclone.net/details.html.

[15] : IAT Hooking, http://keicode.com/windows/win09.php.

研究業績

学術論文

なし

研究会・大会等

1. 早川顕太，鈴木秀和，渡邊晃，“インストール時の特性を利用したワーム検出の一手法”，平成 25 年度電気関係学会東海支部連合大会論文集，Sep.2013.
2. 早川顕太，鈴木秀和，旭健作，渡邊晃，“Windows 上における危険な処理の承認機構の提案”, 情報処理学会第 76 回全国大会講演論文集，Mar.2014.

付 録A　APIフックの種類とその比較

API フックは，大別するとカーネルモードで行われるフックと，ユーザモードで行われるフックがある．各 API フックの方法を以下に示す．

- カーネルモードの API フック
  - SDT フック：SDT（Service Descriptor Table）と呼ばれるカーネル領域に存在し，システムコール番号に対するシステムコールのアドレスを保持するテーブルを，書き換える方法．
- ユーザモードの API フック
  - Detours フック：仮想メモリ上の API 先頭の命令をフック関数の JMP 命令に置き換える方法．
  - IAT フック：仮想メモリ上にマップされた実行ファイル内に存在する IAT（Import Address Table）と呼ばれる，API のアドレスを格納したテーブルを書き換える方法．
  - ラッパー DLL：フック対象となる API が実装された DLL と同様のエクスポート関数を持つラッパー DLL を作成し，プログラムにこのラッパー DLL をロードさせる方法．実際にフックを行うには，プログラム本体の実行ファイル内で参照しているフック対象の DLL 名を，ラッパー DLL への名前に書き換える必要がある．

続いて，これらの API フックの定性評価を行う．定性評価の結果が，表 A.1 である．

- フック対象：カーネルモードのフック対象はシステムコールであるのに対し，ユーザモードのフック対象は API である．
- フック単位：SDT フックはカーネルモードのフックであるため，任意のプロセスのシステムコールをフックできる．それに対し，ユーザモードのフックは基本的にプロセス単位のフックとなる．ただし，IAT フックやラッパー DLL は，それぞれイメージファイル [1] とモジュール [2] 内のインポートセクションの情報を書き換えるため，フック単位はモジュールである．

[1] EXE ファイル，または，DLL
[2] イメージファイルが仮想メモリ上にロードされたもの

- API フックの実装難易度：カーネルモードの SDT フックは，ドライバの作成を伴い，Windows のアンドキュメンテッドな部分に触れるため実装が難しい．Detours フックはライブラリが存在するため，実装は容易である [10]．ラッパー DLL も，フック対象の DLL を指定すると，その DLL と同様のエクスポート関数を持つ DLL のソースコードを出力するツールが含まれる，実装は容易である [14]．IAT フックは，サンプルコードを用いた実装が可能である [15].

- フック回避の難易度：SDT フックは API フックの実装難易度が高いため，フックの回避も難しく，また，ユーザモードのみで動作するマルウェアは原理的にフック回避が不可能である．Detours ライブラリは，API の先頭のコードが上書きするため，これを復元する必要があり，フック回避は難しい．ただし，マルウェアは API 先頭を調査することで，それが JMP 命令であれば Detours フックが行われていることを知ることができ，何かしらの対策を取れる．ラッパー DLL や IAT フックは，API のアドレスをプログラムが自力で探索・取得することにより回避することができてしまう．

- システムの実装難易度：提案システムは，任意のプロセスの API をフックする必要がある．この評価項目は，任意のプロセス上で，フック対象となる API の呼び出し全てをフックするための，実装上の難しさを示している．SDT は任意のプロセスのシステムコールをフックできるため，何もしなくてもこの要件を満たしている．Detours フックと IAT フックは，動作中の各プロセスについてフックを仕掛ける必要がある（IAT フックにおいては，各プロセスの各モジュールごとにフックを仕掛ける必要がある）．ラッパー DLL は実行ファイルや DLL を直接書き換える必要があり，Windows 標準の DLL にはそれができないため，事実上，実装が不可能である．

以上の定性評価から提案システムの実装には，実装が容易で比較的にフック回避が難しい Detours ライブラリを採用する．

表 A.1　各 API フックによる定性評価

評価項目	SDT フック	Detours フック	ラッパー DLL	IAT フック
フック対象	システムコール	API	API	API
フック単位	○	△	×	×
API フックの実装難易度	×	○	○	○
フック回避の難易度	○	△	×	×
システムの実装難易度	○	△	×	△

付 録B　DLLインジェクションの種類とその比較

DLL インジェクションにはいくつかの方法があるため以下に各方法の説明を行い，それらの比較を表 B.1 に示す．

1. DetourCreateProcessWithDll(Ex) 関数
   この関数は，Detours ライブラリが提供するプロセスを生成するための関数である．この関数により生成されるプロセスは，引数で指定した DLL を注入した状態で起動させることができる．

2. AppInit_DLLs レジストリ値
   このレジストリ値に DLL 名を登録しておくと，アプリケーションが user32.dll をロードしたタイミングで，自動的に指定した DLL がロードされる．このため，user32.dll をロードしないアプリケーションでは DLL を注入できない．

3. グローバルフック
   グローバルフックとは，Windows が提供するメッセージフックの仕組みである．事前にメッセージのフックルーチンを実装した DLL を作成しておく．あるプロセスが，作成した DLL を指定して SetWindowsHookEx API を呼び出すことにより，それ以降，動作中のプロセスがはじめてメッセージを受信した際に，指定した DLL がロードされる．そして，それ以降メッセージが DLL へフックされることになる．このように，グローバルフックには DLL インジェクションの仕組みが備わっているため，これを利用する方法である．

4. RemoteCreateThread API と LoadLibrary API を組み合わせた方法
   RemoteCreateThread API は指定したプロセスにスレッドを注入する API である．LoadLibrary API は指定した DLL を自プロセスにロードするための API である．この DLL インジェクションの方法は，RemoteCreateThread API により対象プロセスにスレッドを注入し，その注入したスレッドに LoadLibrary API を呼ばせることで DLL を注入する方法である．

DLL インジェクションにより，注入した DLL で API フックを行う場合，DLL の注入のタイミングが，そのまま API フックを仕掛けるタイミングとなる．そのため，なるべくプロセ

スの起動時から DLL の注入が行えた方がよく，また，任意のプロセスへフック用 DLL の注入を行う必要がある．このため，DLL の注入タイミングが不明瞭である 2. や 3. の方法は適さない．提案システムの実装には 1. と 4. の方法を採用して，可能な限りプロセスの起動時から任意のプロセスへの DLL インジェクションを行う．

表 B.1　DLL インジェクションの方法

	DLL インジェクションの方法	注入可能な範囲	DLL の注入タイミング
1.	DetourCreateProcessWithDll(Ex) 関数 (Detours ライブラリ)	この関数により生成される子プロセス	プロセス起動時
2.	AppInit_DLLs レジストリ値	user32.dll をロードする任意のプロセス	user32.dll のード時
3.	グローバルフック	メッセージを受信する任意のプロセス	グローバルフック以降の，プロセスの最初のメッセージ受信時
4.	RemoteCreateThread API ＋ LoadLibrary API	任意の一つのプロセス	プロセス起動後の任意のタイミング