平成27年度　修士論文

邦文題目

動的なユーザ承認によるWindows向けマルウェア対策手法

英文題目

Anti-malware Technique for Windows using Dynamic User Approval

情報工学専攻 渡邊研究室

(学籍番号: 143430021)

早川　顕太

提出日: 平成28年1月28日

名城大学理工学研究科

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．本研究では，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システムコール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．また，ホワイトリスト／ブラックリストを導入や許可／拒否される危険な処理を子のプロセスへと引き継ぐことにより，ユーザビリティの向上や承認要求を回避する手段を防止する事を検討した．本研究では，さらに，提案方式のプロトタイプシステムを実装し，マルウェアの検知実験やフック処理のオーバーヘッドの評価を行った．そして，実験によって得られた結果や既存技術との比較から提案方式の有用性を示す．

目 次

第1章　はじめに

インターネットの急速な発展にともない，マルウェアの種類・数が日々増加しており．大きな脅威となっている [1]．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常駐し，バックドアによる遠隔操作により DDoS（Denial of Service attack）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．本研究の目的は，これらマルウェアがバックグランドで行う不正な活動を検出し，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により，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により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．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により，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という利点がある．ビヘイビア法を用いた研究例としては，考察と実験によりマルウェアの振る舞いを定義した研究 [2–6] や機械学習を用いる研究 [7,8] が挙げら

れる．

しかしながら，ヒューリスティック法の共通課題として，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課題がある．マルウェアの多様化により，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，ビヘイビア法の一環として危険な処理に対するユーザへの動的な承認機構を提案する．ユーザの判断を借りることによりマルウェアと正規ソフトウェアの振る舞いを分離する．承認機構という方式を取ることで，正規ソフトウェアとマルウェアの共通する振る舞いを検出対象とすることができ，従来のビヘイビア法では検出できなかったマルウェアを検出できる可能性がある．

承認機構の既存技術として，スマートフォン向け OS の Android では，危険なパーミッションの利用時にユーザへ承認の問い合わせを行う研究がある [9–13]．しかし，これらの研究は，パーミッションやサンドボックスなど Android の特有の仕組みに依存しているため，その他の OS に流用することが出来ない．一方，デスクトップ OS の主流である Windows では，標準搭載されている承認機構としてユーザアクセス制御が存在するが，これはプログラムの起動時に行われる承認機構であるため，危険な処理が行われるタイミングやその内容を把握することはできない．

そこで，本論文は Windows を対象として，危険な処理のユーザへの承認機構を提案する．提案システムは，Windows のシステムコールをフックすることにより，危険な処理を検知し，ユーザへ問い合わせを行う．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ないものとして拒否することが可能となる．

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について述べる．5 章で，プロトタイプを用いたマルウェアの検知実験やフック処理にかかるオーバーヘッドの評価を行い，さらに，既存技術との比較を行う．6 章では提案方式の今後の課題を述べ，最後に，7 章でまとめる．

第2章　既存技術

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．さらに，提案方式は承認機構であるため，Windows を対象とした承認機構の既存技術を取り上げる．また，Windows 以外の OS として Android を対象とした承認機構の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，以下に示す課題がある．

- （課題 1）振る舞い定義の難しさ
  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．
- （課題 2）正規ソフトウェアとマルウェアとの分離
  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．

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．以下に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．文献 [14] はワームが自己複製を行う際に自身の実行ファイルを READ する必要があるため，このワームに特徴的な挙動を検出する手法である．文献 [3] は，PC へインストールされたワーム（ボット）は実行環境を復元して再度実行することにより，インストール挙動（実行ファイルの作成と自動実行への登録）を反復するという特徴的な挙動をとるため，これを検出する手法である．文献 [4] 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検出する．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 [5, 6]．文献 [15] では，マルウェアが解析者による解析を妨害するために，生成するファイル名などをランダムに変化させるという特徴的な挙動を，実行毎の API ログを比較することにより検出する．

これらの既存研究は，いずれもビヘイビア法の課題 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る．また，研究にもよるがビヘイビア法の課題 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．文献 [7] 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 法で表現し機械学習する．実環境において，システムコールが呼ばれた際，最近の N 個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．文献 [8] は，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しており，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（N-gram により表現される）を SVM(Support Vector Machine) によって教師あり機械学習する．実環境では，クリティカルなシステムコールの呼び出しをトリガーとして SVM による識別を行いマルウェアを検知する手法である．

これらの既存研究は，ビヘイビア法の課題 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

2.2.1 Windows における承認機構

Windows 上においては，動的な承認機構を提供するといった研究はなされていない．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．

2.2.2 Android における承認機構

スマートフォン向け OS の Android を対象に，ユーザへ動的な承認要求を行う研究がある．Android では，パーミッションと呼ばれる権限によって，位置情報やアドレス帳などの重要な情報の利用やネットワーク接続などの特定の機能の利用を制限している．アプリケーションはインストール時に，利用するパーミッションをユーザに承認してもらうことによって，その重要な情報や機能を利用できる．しかし，アプリケーションをインストールするためには，ユーザはアプリケーションが要求する全てのパーミッションを承認する必要がある．さらに，アプリケーションがパーミッションを利用するタイミングをユーザが把握することができない．従って，与えられた権限の範囲内で悪意を働くマルウェアにユーザは気づくことができないという課題がある．文献 [9–12] はパーミッション利用時にユーザへの動的な承認要求を行うことによりこの課題を解決する．制御対象とするパーミッショは研究の方針によって多少異なるが，情報漏えいを防止するという観点から選ばれることが多い．例えば，SMS の送信やネットワークの接続など外部と通信するためのパーミッションや，アドレス帳や位置情報，ネット閲覧履歴の取得等に関するパーミッション等が挙げられる．Android6.0 では，一部パーミッションを対象に，この仕組みが Runtime Permission [**?**] として標準搭載されている．

また，Andorid にはサンドボックスと呼ばれる仕組みがある．アプリケーション毎に異なる UID を割り当てることで，各アプリケーションが独立して実行され，互いのリソースに直接アクセスすることができない．このため，Android における承認機構の利点として，マルウェアがホワイトリストに記載された他のアプリケーションを利用することにより，承認を回避するということが比較的困難である点が挙げられる．しかしながら，複数のマルウェアを連携させることで，承認を回避することが可能である．文献 [11] では，SEAndroid の TagPropagation というアプリケーションに付与されるタグをアプリケーション間の通信により伝播させる機能を利用して，複数のアプリケーションの連携による情報漏えいを防止する．

これらの研究はいずれも Android のパーミッションやサンドボックスなど特有の仕組みに依存しており，その他の OS に流用することが出来ない．

第3章　提案方式

3.1　概要

本論文では，Windows を対象に動的な危険な処理のユーザへの承認機構を提案する．図 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システムコールを提案システムがフックすることにより，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ないものとして拒否することが可能になる．

以降の節で，検出対象とする危険な処理についての検討と承認機構として求められる機能要件を整理した．

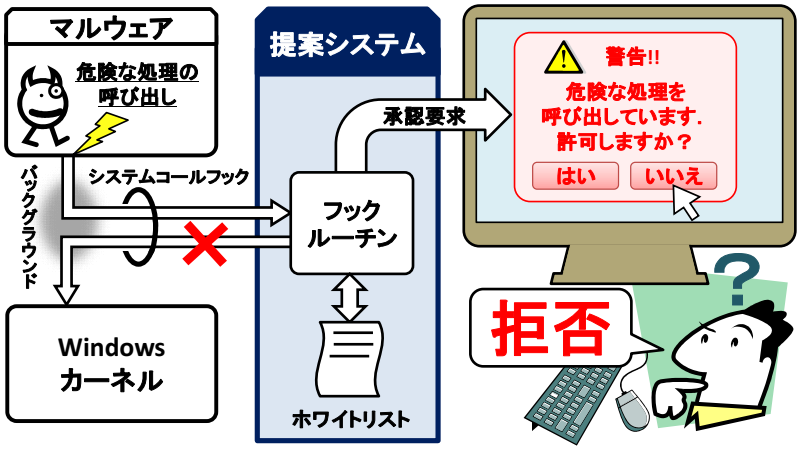

図 1　提案方式の概要

3.2 危険な処理の定義

危険な処理は，マルウェアによって悪用される可能性があり，ユーザがアプリケーションを実行する目的となる処理である．ユーザはその処理を行うことを目的として，そのアプリケーションを実行するため，その処理に関する承認ダイアログが表示されても正しく許可を発行することができる．

例えば，メール送信はほとんどの場合，ユーザがメーラーによって意図して行う．これに対し，マルウェアはバックグラウンドでスパムメールを送信する．従って，メール送信時に，承認ダイアログを表示することにより，それが正規なものかをユーザが判断することができる．逆に，ユーザの目的とならないような処理，あるいはユーザの目的となっても，その処理を行うことをユーザが把握していない場合は，例えマルウェアに悪用される恐れがあっても，ユーザの許可／拒否の判断が難しいために，承認を行うのに適切ではない．このような方針によって，マルウェアが行う危険な処理と正規アプリケーションが行う危険な処理をユーザが区別することができる場合がある．

ビヘイビア法の研究などを参考に危険な処理として表 1 を定義した．前述したメール送信のほかに，実行ファイルの作成や自動実行への登録などは，通常，ユーザがソフトウェアをインストールする時などに観測される．しかし，トロイの木馬やワームといった多くのマルウェアがその初動時に，マルウェア自身をバックグラウンドでインストールする．これは，OS 再起動後も自身が PC に常駐するためである．また，マルウェアの中にはセキュリティソフトや解析ツールから自身を防衛するために，それらのプロセスの動作を検知し強制終了するものが存在する．これに対し，ユーザはソフトウェアが不具合でフリーズした場合など，意図的に強制終了を行う．提案方式はホワイトリストを導入するが，ホワイトリストに登録された正当なプロセス／プログラムに寄生することで，マルウェア自身が危険な処理を行わずに承認を回避する方法が考えられる．これを防止するため，他のプロセスへ感染する手段であるスレッドの注入や，プログラムへ感染する手段である実行ファイルの改ざんについても危険な処理として定義し検出する．

表 1　危険な処理の一覧

危険な処理	悪用された場合の被害
実行ファイルの作成	不正インストール
自動実行への登録	不正インストール
他プロセスの強制終了	セキュリティの無効化
メール送信	スパムメールの送信
スレッドの注入	他プロセスへの感染
実行ファイルの改ざん	プログラムへの感染

3.3　承認機能

提案方式の承認機能の動作を以下にまとめる．

(1) スレッドの危険な処理の呼び出しを検出し，ユーザの承認応答を得るまでそのスレッドを一時停止状態にする．

(2) スレッドが一時停止している間に，デスクトップ上に承認ダイアログを表示し，ユーザへ承認要求を行う．

(3) ユーザの応答を得たら，停止していたスレッドを再開させユーザの応答に応じて処理を続行させたり，中断させたりする．

(2) における承認ダイアログのイメージを図 2 に示す．承認ダイアログ内には，そのプログラムがマルウェアであるかどうかをユーザが判断できる十分な情報を表示する必要がある．従って，ユーザがプロセスを一意に特定することができるようにプロセスに関する情報や，行おうとしている危険な処理の内容，ユーザの選択肢を表示する．特に対象のプロセスが表示しているウインドウを示すことで，ユーザは対象のプロセスを直観的に把握することができる．

図 2　承認ダイアログのイメージ図

3.4 ホワイトリスト／ブラックリスト機能

危険な処理を行う度に承認要求を行うと，ユーザの利便性が損なわれる恐れがある．これを緩和するために，ホワイトリスト／ブラックリスト機能を導入する．ホワイトリスト／ブラックリストには，ユーザが安全あるいは危険であると判断したプログラムを登録し，プログラムごとに許可／拒否される危険な処理を記載する，承認要求の際に，ユーザは任意にその応答をホワイトリスト／ブラックリストに記憶することができ，再度，同じプログラムから危険な処理が呼び出されたときは，ホワイトリスト／ブラックリストを参照することにより，ユーザへの承認要求なしで許可／拒否の決定を自動化することが出来る．また，登録後もホワイトリスト／ブラックリストをユーザ自身が編集することもできるようにする．さらに，提案方式では以下の 2 つの機能を設ける．

- （機能 1）信頼済みプログラム
  事前調査（附録 A）により，OS 自身が通常の動作において，表 1 で示す処理を実行することが分かっている．このため，OS が行う正常な処理をユーザが誤検知してしまう可能性や，OS の動作に支障をきたす可能性がある．また，これら OS のプログラムをユーザが適切にホワイトリストへ登録する保証はない．そこで，Windows 標準のプログラムによる処理は基本的に許可する．以降，Windows 標準のプログラム，すなわち，Microsoft のデジタル署名を含むプログラムを信頼済みプログラムと呼び，信頼済みプログラム以外のソフトウェア，すなわち，ユーザプログラムを信頼できないプログラムと呼ぶこととする．なお，Windows 標準ソフトにはコマンドプロンプトや PowerShell，Office アプリケーションのマクロのように，スクリプトを実行する機能を持ったプログラムが存在する．そのため，これらのプログラムは悪意のあるスクリプトが実行される可能性があるため，信頼できないプログラムとして扱う必要がある．

- （機能 2）許可／拒否された危険な処理の子プロセスへの継承機能
  ホワイトリスト／ブラックリストはプログラム単位で管理されるため，マルウェアがホワイトリストに登録されたプログラムを子のプロセスとして生成し，処理を依頼することで承認要求を回避できる．そこで，プロセス単位で許可／拒否された危険な処理を管理するオンメモリデータベースを別途設け，その情報を子のプロセスに引き継ぐことにより，これを解決する．

各プロセスの許可／拒否される危険な処理を決定する仕組みを図 3 に示す．以後の説明のために「信頼済みプロセス／信頼できないプロセス」という用語を導入する．信頼済みプロセスとは，信頼済みプロセスから起動される信頼済みプログラムのことであり，OS 起動時に起動されるルートの Windows プログラムは信頼済みプロセスとする．信頼済みプロセスについては，全ての危険な処理を信頼できるものとして常に許可する．信頼できないプロセスとは，信頼できないプロセスから起動したプログラム，あるいは，信頼済みプロセスから

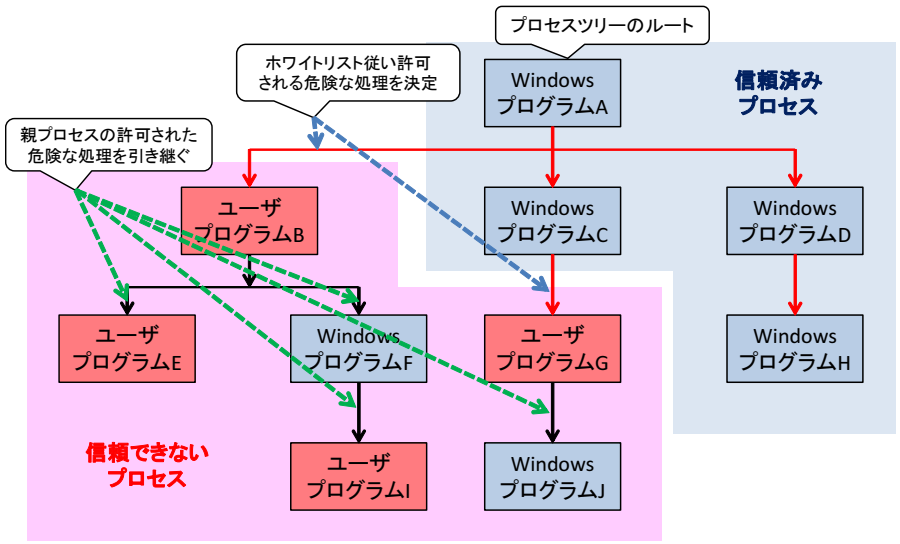

図 3　各プロセスの許可／拒否される危険な処理を決定する仕組み

起動した信頼できないプログラムである．信頼済みプロセスから起動した信頼できないプログラムは，ホワイトリスト／ブラックリストを参照して許可／拒否される危険な処理を決定する．信頼できないプロセスから起動したプログラムについては，親のプロセスの許可／拒否される危険な処理を引き継ぐ．ただし，起動したプログラムがブラックリストに記載されていれば，対応する危険な処理は拒否する．このような仕組みにより，ホワイトリスト／ブラックリストと（機能 1），（機能 2）が混在しながら機能させることが出来る．

また，プログラム単位のホワイトリスト／ブラックリストに登録せずとも，承認要求時においてオンメモリデータベースに直接，危険な処理の許可／拒否を登録することができる必要がある．この機能は，以下に示すようにインストーラにおいて有用である．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，ユーザへ承認要求を行っていては利便性を損ねる．しかし，通常インストーラは一度実行し，その後削除してしまうため，ホワイトリスト／ブラックリストへ登録することも適切ではない．また，ソフトウェアのインストールでは，インストールしたソフトウェアがさらに別のソフトウェアをインストールすることがあり得る．そこで，プロセスに対して許可／拒否を自動化できるようにすることで，ユーザがより適切な選択を行えるようにする．

3.5　ログ機能

提案方式は危険な処理の呼び出しのログを取る機能を導入する．たとえホワイトリスト／ブラックリストに登録済みのプログラムであったとしても，ログを残すことが望ましい．ユーザはログを確認することによって，ホワイトリストに登録したプログラムが登録後も，悪さを働いていないかどうかを確認することができる．ただし，信頼済みプロセスの全ての

処理は，信頼できるものとしてログは取らないことにする．また，ホワイトリストへ登録後，定期的に（1 週間後，1 か月後，半年など）ユーザへ強制的にログ情報を報告することで，ユーザに確認を促す．これにより，ユーザのセキュリティ意識の向上も期待できる．

第4章　実装

提案方式の有用性を評価するため，プロトタイプシステムを実装した．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　実装構成と機能

32 ビット版 Windows7 を対象に以下のような実装を行った．プロトタイプシシステムでは，ユーザモードで動作するマルウェアが回避不可能であるシステムコールフックを用いて危険な処理を検出することとした．システムコールフックには SSDT フックを採用した．SSDT（System Service Descriptor Table) とは，システムコール番号に対応するシステムサービスルーチンのアドレスを管理するテーブルである．SSDT フックは，フック対象のシステムコールの SSDT のエントリをフック関数のアドレスで上書きすることでフックする方法である．

図 4 にプロトタイプシステムの構成図を示す．プロトタイプはシステムコールをフックするためのデバイスドライバと常駐プロセスとなる監視アプリケーションの 2 つから構成される．以下に各要素の機能を説明する．

(1) ドライバ

ドライバには以下の 3 つの機能を実装した．

1. フック関数

   危険な処理に対応するシステムコールが呼ばれると，SSDT フックによりフック関数へと処理が移る．フック関数のフローチャートを図 5 に示す．フック関数では，まず，プロセスの PID を元にデータベースを検索し，そのプロセスが信頼済みプログラムであるかどうかをチェックする．信頼済みプログラムであれば，危険な処理を許可し処理を再開する．信頼済みプログラムでなければ，システムコールの引数などをチェックすることにより，それが危険な処理に該当するかどうかをチェックする．危険な処理であると判断した場合，危険な処理の情報を格納した承認要求データを循環キューに格納する．承認要求データには，危険な処理を示す番号やタイムススタンプ，プロセス ID，スレッド ID，危険な処理毎に固有な情報などが格納される．次に，前述のデータベースの検索結果を参照し，その危険な処理の許可／拒否が自動化されているかどうかをチェックする．自動化されていれば，それに応じて処理を再開する．自動化されていなければ，ユーザへの承認要求を行うために，スレッドを待機させる．監

視アプリケーションによるユーザへの承認要求の後，待機していたスレッドが再開され，ユーザの応答に応じて処理を分岐する．

2. プログラムの起動検出

PsSetCreateProcessNotifyRoutine 関数にコールバック関数を設定することで，プログラムの起動を検出する．本来は，ここで信頼済みプログラムであるかを検証し，ホワイトリスト／ブラックリストと親プロセスのデータベース情報を参照することで，子プロセスの許可／拒否される危険な処理を決定する．しかし，信頼済みプログラムの検証とワイトリスト／ブラックリストが未実装であるため，データベースにはダミーデータを登録した．また，データベースは PID をキーとしたハッシュテーブルにより実装した．

3. 監視アプリケーションとのインタフェース

ドライバには IRP_ MJ_ DEVICE_ CONTROL コールバック関数を実装し，独自のコントロールコードにより監視アプリケーションとドライバの間で循環キューに溜まったデータの受け渡しや，通知イベントによるフック関数内のスレッドの再開を行う．

(2) 監視アプリケーション

監視アプリケーションは，危険な処理の呼び出しが発生すると，その情報をドライバから受け取り，ユーザへ承認ダイアログによる承認要求を行ったり，ログを出力する常駐プログラムである．監視アプリケーションはコンソールアプリケーションとして実装し，受け取ったデータの詳細情報は CSV 形式でログファイルに出力するようにした．

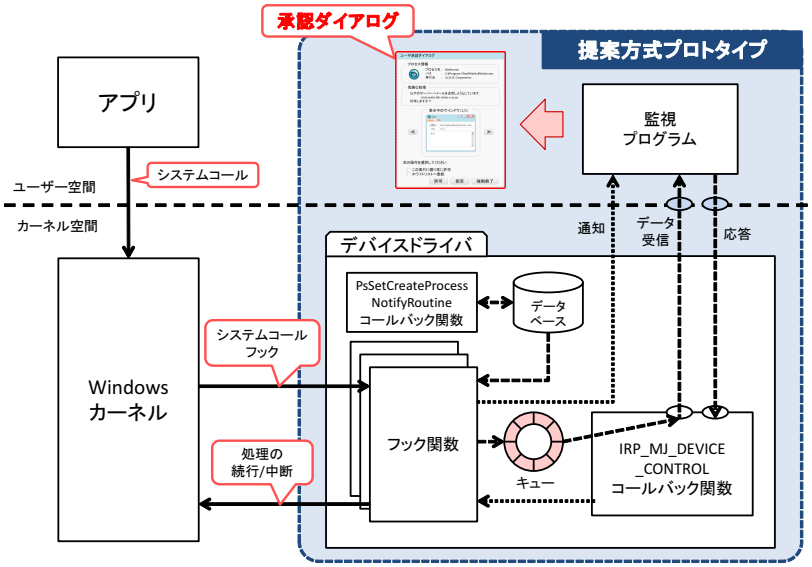

図 4　プロタイプシステムの構成図

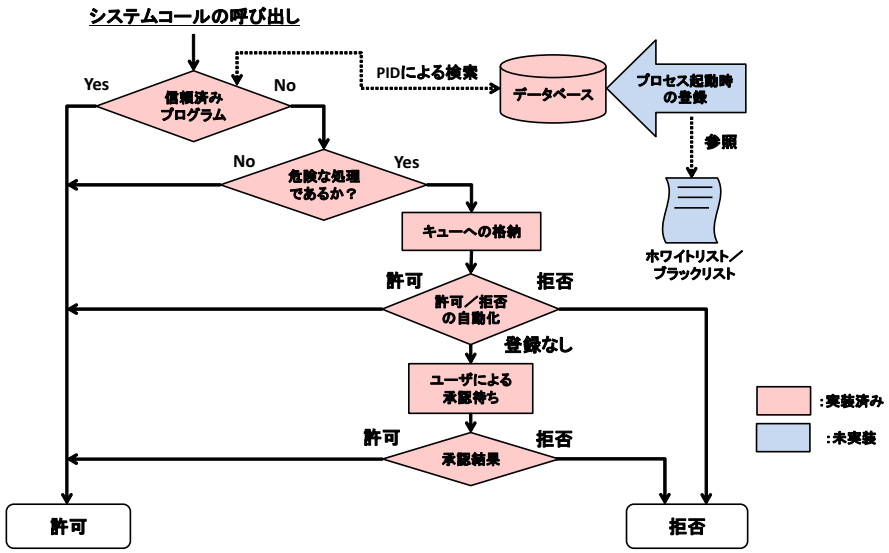

図 5　フック関数のフローチャート

4.2　危険な処理として検出するシステムコール

本節では危険な処理として定義した表 1 に対して，実際にどのシステムコールをフックするかを述べる．危険な処理に対して検出対象となるシステムコールを表 2 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システムコールが指定した引数の条件を満たして呼ばれた際に，危険な処理として検出する．実行ファイルの作成は，Windows における実行ファイルの拡張子 “.exe”，“dll”，“.sys” を持つファイルの作成を検出する．OS 起動時の自動実行への登録は，その方法にレジストリ，スタートアップフォルダ，タスクスケジューラによる複数の方法がある．プロトタイプではマルウェアによく利用されるレジストリを用いた方法のみをターゲットとし，表 3 に示す自動実行に関するレジストリエントリを検出対象とした．他プロセスの強制終了については，マルウェアが自らセキュリティソフトや解析ツールを起動して強制終了するということは考えられないため，親プロセスが子プロセスを強制終了する場合を許可する．メールの送信は，Windows のソケットインタフェースがドライバ（Ancillary Function Driver）として実装されているため，アプリケーションからドライバを操作するためのシステムコールである ZwDeviceIoControlFile をフックすることで検出する．他プロセスへの感染は，自身以外のプロセスへのスレッドを作成を検出する．なお，実行ファイルの改ざんについては今回，実装を行っていない．

表 2　危険な処理に対して検出対象となるシステムコール

危険な処理	システムコール	システムコールの引数の条件
実行ファイルの作成	ZwCreateFile	ファイルを新たに生成し, そのファイルの拡張子が実行ファイルの拡張子（“.exe”, “dll”, “.sys” のいずれか）である場合
	ZwSetInformationFile	操作がファイル名のリネームで，リネーム後のファイルが実行ファイルの拡張子である場合（ただし，リネーム前のファイルが実行ファイルの拡張子の場合を除く）
OS 起動時の自動実行への登録	ZwSetValueKey	キー名やエントリ名に表 3 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
他プロセスの強制終了	ZwTerminateProcess	自身あるいは子プロセス以外のプロセスに対して強制終了する場合
メール送信	ZwDeviceIoControlFile	コントロールコードが connect 関数に該当する場合で, 接続先ポート番号がメール送信に関するポート番号 SMTP(25), SMTPs(456), サブミッションポート (587) の場合
スレッドの注入	ZwCreateThreadEx	自身以外のプロセスにスレッドを作成しようとした場合
実行ファイルの改ざん	—	—

表 3　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

提案方式のプロトタイプシステムを用いて，マルウェアの検知実験，フック処理のオーバーヘッドの評価を行った．また，既存技術と提案方式の比較を行った．

5.1　マルウェア検知実験

仮想マシン上に提案方式のプロトタイプシステムを導入しマルウェアを動作させた．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 [16] と VX Vault [17] 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 49 体である．

マルウェアの検出結果を表 4 に示す．それぞれの危険な処理について，その処理を呼び出したマルウェアの検体数を示しており，マルウェア 1 体による複数の危険な処理も呼び出しも勘定に含んでいる．全体では，約 59%（29/49 体）のマルウェアで危険な処理を検出することが出来た．詳しく見ると，検出したマルウェアのほとんどは，実行ファイルの作成により検出されていることが分かる．実行ファイルの作成は，マルウェアが PC に常駐するための常套手段であるためだと考えられる．他プロセスの強制終了については，実験時に，マルウェアが削除されないようにセキュリティソフトを停止しており，さらに，解析のためのツールも用いていなかったことから，マルウェアによる他プロセスの強制終了は行われなかったと考えられる．危険な処理を一度も検出できなかったマルウェアの理由としては，マルウェアを実行した時間が短かったこと，あるいは，マルウェアが想定した環境や動作条件に合わなかった，仮想マシン上での動作であることを察知されマルウェアが活動を停止したことが考えられる．

提案方式は正規ソフトウェアとマルウェアで共通する振る舞いを検出対象としているため，検出率は約 59%に留まったがまる，マルウェアが正常に動作しなかったことなどにより，ある程度のマルウェアを検出することが可能であることが確認できた．特に，これらのマルウェアは従来のビヘイビア法では検出できない可能性があるため，本方式は他の検出手法を併用することで十分に有用性があると言える．

表 4　マルウェアの検出結果 (全 49 体)

危険な処理	検体数
実行ファイルの作成	26
自動実行への登録	9
他プロセスの強制終了	0
メール送信	1
スレッドの注入	3
実行ファイルの改ざん	未実装
全体	29

表 5　実験環境

OS	Windows7 32bit
CPU	Intel(R) Core(TM) I5 4210U 1.70GHz
メモリ	4GB
SSD	128GB

5.2　オーバーヘッドの評価

提案方式のプロトタイプのフック処理によるオーバーヘッドを調査した．実験環境を表 5 に示す．プロトタイプの導入の有無により，危険な処理を行うユーザモードの API の実行にかかる時間を計測した．これら Windows API は内部でプロトタイプがフックしているシステムコールを呼び出しており，計測したフック処理のパスは図 5 においてユーザへの問い合わせが発生しないパスの中で，最もオーバーヘッドがかかるパスを選択した（図中の上から順に，No → Yes →許可の順）．計測には QueryPerformanceCounter API を利用し，計測プログラムで危険な処理を行う API を 1000 回呼び出し，その平均時間を測定した．

表 6 に API の測定時間を示す．表 6 においてフックの有無の差がフック処理によるオーバーヘッドとみなすことができる．この結果より，フック処理のオーバーヘッドは最大で (4) の 19μs である．API の実行時間を考えると，オーバーヘッドは十分小さいと言える．メール送信については，connect 関数内で TCP の 3 ウェイシェイクハンドが行われるため，その処理時間はネットワークの応答時間に依存する．従って，CPU の処理時間はほとんど無視できるため，測定は行わなかった．

表 6　API の測定時間（1000 回平均）

	危険な処理とシステムコール	測定した API	フック	時間 [μs]
(1)	実行ファイルの作成	CreateFile	無	189.893
	ZwCreateFile		有	206.670
(2)	実行ファイルの作成	SetInformationByHandle	無	205.099
	ZwSetInformationFile		有	219.466
(3)	自動実行への登録	RegSetValueEx	無	29.163
	ZwSetValueKey		有	33.649
(4)	他プロセスの強制終了	TerminateProcess	無	86.009
	ZwTerminateProcess		有	104.806
(5)	他プロセスへのスレッド注入	CreateRemoteThread	無	56.696
	ZwCreateThreadEx		有	57.022

5.3　既存技術との比較

表 7 に提案方式と従来のビヘイビア法の比較を示す．従来のビヘイビア法として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を取り上げた．

- 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狭い範囲のマルウェアを検出する．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．特に，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も検出対象にできる．このように，従来のビヘイビア法では，マルウェア固有の振る舞いを検出する．これに対し，提案方式はユーザの判断を借りることで，マルウェアと正規ソフトウェアの共通する振る舞いを検出する．提案方式は従来のビヘイビア法とは異なる範囲の振る舞いを検出するため，従来では検出できなかったマルウェアを検出できる可能性がある．

- 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，研究にもよるがマルウェア固有の振る舞いを検出するため，比較的に誤検知が少ない．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という課題がある．提案方式における誤検知は，ユーザのセキュリティ意識・知識に依存するため，判定しなかった．

- 検出時のユーザへの説明
  この評価項目は，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る．マルウェアの特徴的な振る舞

いを検出する手法では，検出する振る舞いが特徴的であるために，PC に詳しくないユーザではその理由を理解することが難しい場合がある．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く，提案方式の利点と判断した．

- ユーザビリティ
  従来のビヘイビア法はともに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ため，危険な処理が行われる度にユーザへ承認要求が行われる．ただし，ホワイトリスト／ブラックリストの導入により，著しくユーザビリティが損なわれることはない．

表 7　提案方式と従来のビヘイビア法の比較

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
振る舞いの検出範囲	マルウェア固有	マルウェア固有	共通部分
誤検知	○	△	—
検出時のユーザへの説明	△	×	○
ユーザビリティ	○	○	△

第6章　今後の課題

提案方式には以下の課題を抱えている．

- 承認要求におけるユーザの対応
  危険な処理として検出したシステムコールの利用実態が想定したものに合わない場合，あるいは，ソフトウェアに関してユーザの知識が不足していた場合に，意図しないタイミングで承認要求が行われてしまうことが考えられる．このとき，ユーザが誤って正規ソフトウェアを誤検知してしまう恐れがある．また，ユーザへ承認要求が頻発すると，面倒に感じたユーザが常に許可にしてしまう．

- Windows サービスのホワイトリストへの登録
  Windows サービスはその性質から他のソフトウェアから危険な処理を請け負う可能性がある．このため，これらのサービスをホワイトリストへ登録してしまうと，マルウェアがサービスを利用することでユーザへの承認要求を回避できてしまう．ホワイトリストに登録しなかった場合でも，承認ダイアログ内のプロセスの情報は，危険な処理を依頼した本来のプロセスではないため，ユーザは承認のタイミングといった観点のみからマルウェアかどうかを判断しなければならない．

- 64 ビット版 Windows への本方式の対応
  本研究ではカーネルモードで動作するシステムコールフックを用いることで，危険な処理を検出した．しかしながら，64 ビット版の Windows ではカーネルパッチプロテクションにより，カーネルにパッチを当てる行為が防止されており，システムコールフックが困難である．このため，ユーザモードの API フックに置き換える方法や，仮想マシンモニタの技術 [18] を利用するといった方法を検討する必要がある．

第7章　まとめ

本研究では，Windows を対象として危険な処理のシステムコールをフックすることで，動的にユーザへ承認要求を行う方式を提案した．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ないものとして拒否することが可能になる．特に，ホワイトリスト／ブラックリストの導入や許可／拒否される危険な処理を子のプロセスへと継承することにより，ユーザビリティの向上や承認要求を回避する手段の一つを防止する事を検討した．本提案のプロトタイプシステムを実装し，マルウェア検知実験を行い，58%のマルウェアを検出することに成功した．また，OS による危険な処理を許可することで，ユーザへの承認要求が大幅に減ることを示し，さらに，フック処理のオーバーヘッドの測定を行い，ほとんど影響がない事を示した．

今後は，本提案のプロトタイプの実装を進め，実際にユーザが使用した際の利便性の検証を進める予定である．また，Linux など他のデスクトップ OS への本提案の適応を検討する予定である．

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．

本論文を作成するにあたり，快く副査を引き受けて頂きました，名城大学大学院理工学研究科 吉川雅弥教授に心より厚く御礼申し上げます．

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，鈴木秀和准教授，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] : McAfee 脅威レポート 2015 年第 2 四半期, http://www.mcafee.com/jp/resources/reports/rp-quarterly-threat-q2-2015.pdf.

[2] 酒井崇裕，長谷　巧，竹森敬祐，西垣正勝：自己ファイル READ/DELETE の検出によるボット検知の可能性に関する一検討，コンピュータセキュリティシンポジウム 2008 論文集， Vol. 8, pp. 109–114 (2008).

[3] 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).

[4] 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).

[5] 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).

[6] 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).

[7] 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).

[8] 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).

[9] 川端秀明，磯原隆将，竹森敬祐，窪田　歩，可児潤也，上松晴信，西垣正勝：Android における細粒度アクセス制御機構，情報処理学会論文誌，Vol. 54, No. 8, pp. 2090–2102 (2013).

[10] 渡邊華奈子，大月勇人，瀧本栄二，齋藤彰一，毛利公一：Android におけるユーザの意図しない情報の漏洩を防止するパーミッション動的制御，研究報告コンピュータセキュリティ（CSEC），Vol. 62, No. 23, pp. 1–8 (2013).

[11] 矢儀真也，山内利宏：SEAndroid の拡張による AP の動的制御手法の実現，情報処理学会論文誌，Vol. 54, No. 9, pp. 2220–2231 (2013).

[12] 加藤　真，松浦佐江子：ユーザの承認によるアプリケーション実行時の Android マルウェア対策，研究報告コンピュータセキュリティ（CSEC），Vol. 61, No. 6, pp. 1–6 (2013).

[13] : Requesting Permissions at Run Time, http://developer.android.com/intl/ja/training/ permissions/requesting.html.

[14] 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).

[15] 笠間貴弘，吉岡克成，井上大介，松本隆明勉：実行毎の挙動の差異に基づくマルウェア検知手法の提案，コンピュータセキュリティシンポジウム 2011 論文集， Vol. 3, pp. 726–731 (2011).

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

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

[18] 大月勇人，中野　進，明田修平，瀧本栄二，齋藤彰一，毛利公一：Windows 10 x64 環境を対象とするシステムコールトレーサの実現手法，コンピュータセキュリティシンポジウム 2015 論文集，Vol. 2015, No. 3, pp. 839–846 (2015).

研究業績

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.

3. 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”, マルチメディア，分散，協調とモバイル（DICOMO2014）シンポジウム，Jul.2014.

4. 神谷早紀，早川顕太，旭健作，鈴木秀和，渡邊晃，“自己複製挙動に着目したワーム検知手法の提案” 平成 26 年度電気・電子・情報関係学会東海支部連合大会論文集，Sep.2014.

5. 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と実装”, 情報学ワークショップ 2014（WiNF2014）論文集，Nov.2014.

付 録A　OSによる危険な処理の回数

PC の放置時における OS による危険な処理の発行頻度を調査した．実機 PC に，提案方式のプロトタイプシステムを導入し，これを約 3 日間放置し，危険な処理のログを採取した．その後，このログから危険な処理を実行したプログラムのパスを取得し，signtool を用いてそのプログラムに Microsoft 社による電子署名があるかどうかを検証した．

表 8 に危険な処理の呼び出し回数と，内訳を示す．表 8 のエラーは，プログラムがすでに終了しておりプロトタイプシステムがプログラム名の取得に失敗した，あるいは，署名検証時にプログラムのファイルが存在しなかったことを示す．この結果より，3 日間で OS による危険な処理は 118 回行われてたことが分かる．これらの危険な処理について，ユーザへ承認要求を行ってしまうと承認要求の多発により，利便性が著しく低下すると考えられる．また，ユーザがこれらの承認要求を適切に判断できるとも考えにくい．Windows 標準のプログラムの危険な処理を信頼できるものとして許可することにより，承認要求の回数を 118 回から 4 回まで大幅に削減することが可能であり，この方式の有用性が確認できた．本提案の必須な機能であると言える．

なお，非 Windows プログラムによる 4 回の実行ファイルの作成は，AdobeARM.exe というプログラムによって行われおり，Adobe 関連の更新プログラムが実行ファイルをダウンロードしていたものと考えられる．

表 8　危険な処理の呼び出し回数とその内訳

危険な処理	呼び出し回数	プログラムの内訳		
		Windows 標準	Windows 非標準	エラー
スレッドの注入	77	64	0	13
実行ファイルの作成	13	9	4	0
他プロセスの強制終了	28	28	0	0
自動実行への登録	0	0	0	0
メールの送信	0	0	0	0
計	118	101	4	13