Windowsにおける危険な処理の承認機構の提案と実装

早川 顕太，鈴木 秀和，旭 健作，渡邊 晃

名城大学院理工学研究科

Proposal of Approval Mechanisms for Dangerous Behavior of Malware in Windows and its Implementation

Kenta Hayakawa, Hidekazu Suzuki, Kensaku Asahi and Akira Watanabe

Graduate School of Science and Technology, Meijo University

1 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常駐し，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．本稿の目的は，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，マルウェアらしい振る舞いの定義方法に研究者による実験と考察を用いた研究 [1, 2, 3, 4] や機械学習を用いる研究 [5, 6] 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．マルウェアの活動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．

本稿では，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．ユーザが意図したかどうかによって，マルウェアが行う危険な処理と正規ソフトウェアが行う危険な処理を区別することができる場合があると考え，マルウェア検出の 1 つのアプローチとして，ユーザに判断を借りる承認機構という方法を提案する．提案方式はユーザの利便性を損なわないようホワイトリスト／ブラックリストを導入し，ホワイトリストに登録後もユーザによる追跡調査が行えるようログ機能を備えるものとする．また，提案方式の実装方法について検討し，その一部を実装し動作検証を行った．危険な処理の検出にはカーネルモードにおけるより強力なシステムコールフックを使用することにより実現する．

以下，2 章で承認機構に関する既存技術を取り上げる．3 章では，提案方式について述べる．4 章では，カーネルモードのシステムコールフックを用いた提案方式のプロトタイプの実装について述べる．最後に 5 章で現状の課題を述べ 6 章でまとめを行う．

2 既存技術

ここでは，Windows 上の承認機構に関する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についても取り上げる．

2.1 Windowsにおける承認機構

現在，著者らが知る限り 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不正な変更や，スパム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 Android における承認機構

スマートフォン向けの OS である 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 [7, 8, 9, 10] 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．研究の方針によって制御対象とするパーミッションは多少異なるが，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．また，これらの研究は動的な承認機構を提案しているが，承認要求時にユーザがその許可／拒否の判断を正しく行えるかといった評価が十分になされていない．

3 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．これは 2.2 節で示した Android におけるパーミッション利用時に動的な承認要求を行うシステムを，Windows を対象として提案するものである．しかしながら，Windows は Android と異なりパーミッションに該当する概念がない．そこで，提案方式では Windows 上の危険な処理を定義し，アプリケーションが発行するシステムコールをフックすることでこれを検出する．

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，提案方式が検出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．

3.2 危険な処理の検討

危険な処理とは，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

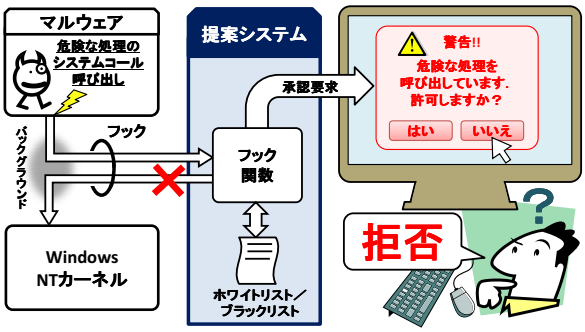


図 1: 提案方式の概要

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や解析妨害
メール送信	スパムメールの送信

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と考えている．ビヘイビア法に関する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ハングアップした場合など，意図的に強制終了を行う．

3.3 機能要件

提案方式が承認機構として，求められる機能を以下に整理した．

＜機能 1 ＞ 提案システムの必須機能

(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) ユーザの承認応答を得たら，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

図 2: 承認ダイアログのイメージ図

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．

＜機能 2 ＞ ホワイトリストとブラックリスト

アプリケーションが危険な処理を行う度に承認ダイアログを表示すると，ユーザの利便性が損なわれる場合がある．これを解決するために，ホワイトリストとブラックリストを導入する．承認要求の際，ユーザは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る．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／拒否を自動的に決定する．ホワイトリスト／ブラックリストへの登録は危険な処理毎に行え，登録のキーはプログラムの絶対パスを利用する．また，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，ホワイトリスト／ブラックリストに登録しなくても，承認ダイアログにおいてその実行に限って許可／拒否の決定を自動化できるようにする．この機能は，例えば，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，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．そこで，その実行に限って許可／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．

＜機能 3 ＞ 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，ログを残すようにする．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．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，ユーザに確認をうながす．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．

＜機能 4 ＞ 正当なプロセス／プログラムへの寄生防止機能

マルウェア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により，マルウェア自身が危険な処理を行わないで，承認を回避する方法が考えられる．各々の寄生方法とその対策を以下に述べる．

- 正当なプロセスへの寄生
  【寄生方法】ホワイトリストに登録された正当なプロセスに対して，仮想メモリ上に危険な処理を行うコードを注入し，そのコードを実行するスレッドを作成する．
  【対策】NtCreateThread(Ex) システムコールをフックすることにより，自身以外の他のプロセスに対してスレッドを注入する行為を制限する．

- 正当なプログラムへの寄生
  【寄生方法】ホワイトリストに登録された正当なプログラム（実行ファイル）にコードを書き込み，プログラムの実行中に危険な処理を行うように改竄する．
  【対策】ハッシュを用いることでプログラム改竄されていないことを保証する．ホワイトリス登録時に実行ファイルのハッシュ値を記憶し，登録済みのプログラムが起動した際にハッシュ値を比較する．

なお，Android では，サンドボックスと呼ばれるセキュリティ機構により，アプリケーションが他のアプリケーションに干渉することができない．このため，2.2 節で示した Andoroid を対象とした既存研究においては，このような対策は不要である．しかし，Windows ではマルウェアのホワイトリスト回避を防止するために必須の機能である．

4 実装

著者らは以前にユーザモードの API フックライブラリである Detours ライブラリを用いてプロトタイプを実装し，マルウェアに対して検証実験を行った [11]．このプロトタイプでは表 1 に示す危険な処理の内，実行ファイルの作成と自動実行への登録の 2 つの処理を検出していた．実験の結果，53 体中 31 体のマルウェアにおいて承認ダイアログを表示させることができ，提案方式の有用性を確認している．

しかしながら，このプロトタイプはユーザモードの API フックを使用したため，OS の一部として動作するプログラムが行う危険な処理を検出できなかった．このため，例えば Windows サービスを経由して危険な処理を行った際に検知漏れが発生し，正確な誤検知実験が行えなかった．また，監視対象となる全てのプロセスに対して，API フックを仕掛ける必要があるという手間があった．

そこで，今回はカーネルモードのシステムコールフックを使用したプロトタイプについての検討を行い，一部を実装し動作確認を行った．本提案システムが対象としている OS は現在，32 ビット（x86）版の Windows7 である．本章では実装方針やプロトタイプの実装の詳細を述べる．

4.1 実装方針

4.1.1 システムコールフックの方法

Windows におけるシステムコールフックの実現方法としては，主に sysenter 命令フック [12] と SSDT フック [13] の 2 つの方法があり，以下にその詳細を述べる．

- sysenter 命令フック
  32 ビット版 Windows は x86 系 CPU の sysenter 命令によってシステムコールが実現されている．sysenter 命令は，MSR と呼ばれる特殊なレジスタからカーネルコードのエントリポイントをプログラムカウンタにロードするなどし，特権レベルを高速にユーザモードからカーネルモードへ移行する命令である．sysenter 命令フックは，MSR のエントリポイントの値をフック関数のアドレスへと上書きすることでフックを行う方法である．

- SSDT フック
  sysenter 命令によってカーネルモードに移行した後，カーネルは eax レジスタに積まれたシステムコール番号に応じて，システムサービスルーチンを呼び出す．ここで，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は SSDT（SystemServiceDescriptorTable）と呼ばれる配列に記憶されている．SSDT フックは，フックする SSDT 内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を行う方法である．

提案方式では，全てのシステムコールをフックする必要がないことや，ドライバからのシステムサービスルーチンの呼び出しをフックできる可能性があるという観点から，SSDT フックを採用する．なお，これらのフックはいずれもカーネルモードでしか行えないため，カーネルモードで動作するデバイスドライバで実装を行う必要がある．

4.1.2 提案方式の各機能の実装箇所の検討

ここでは，3.3 節で述べた提案方式の各機能の実装箇所をどこで実装すべきかを検討する．機能 1 の (1) と (3) はカーネルモードのシステムコールフックを用いるため，ドライバで実装する必要がある．機能 1 の (2) の承認ダイアログの表示は，カーネルモードでは行えないため，ユーザモードのアプリケーションで実装する必要がる．機能 2 と機能 4 は，ユーザモードのアプリケーションで実装するとユーザへの承認要求を行わない場合に，処理が大幅に遅くなる可能性があるため，ドライバで実装する．また，機能 2 のホワイトリスト／ブラックリストと機能 4 のハッシュ値はレジストリで管理・記憶する．提案システムが動作している間は，これらの情報を機能 2 の実行に限った許可／拒否の情報とともに，カーネルメモリ上のデータベースとして管理する．機能 3 はアプリケーションでログファイルを一元管理した方が簡単なため，アプリケーションで実装する．

4.2 プロトタイプの実装

4.1 節の実装方針に従って，機能 1，3 の実装を行った．開発環境には Windows Driver Kit と Visual Studio C++を利用している．ここでは，プロトタイプの構成と危険な処理として検出するシステムコールの詳細について述べる．

4.2.1 プロトタイプの構成

プロトタイプはシステムコールをフックするための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．図 3 にプロトタイプの全体像を示す．以下にプロトタイプの各要素を説明する．

- ドライバ: ドライバは，SSDT フックによって危険な処理を行うシステムコール毎に対応するフック関数へとフックを行う．フック関数では，はじめに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，データベースにそのプログラム（実行ファイルのパス）が記憶されているかどうか，すなわち，ホワイトリスト／ブラックリストやその実行に限り許可／拒否が登録されているかどうかを検索する．プログラムがデータベースに登録されていなかった場合，許可用と拒否用の 2 つの通知イベントオブジェクトをひも付けた承認要求パケットを作成し，循環キューに格納する．承認要求パケットには，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．その後，スレッドは KeWaitForMultipleObjects ルーチンにより通知イベントがシグナル状態になるまで待機する．ユーザの承認によって，その応答に対する通知イベントがシグナル状態になり，待機していたスレッドが再開される．再開したスレッドは，その応答に応じて処理を振り分ける．データベースに登録済みであった場合は，承認要求パケットと同一フォーマットのログパケットを作成し，循環キューに格納する．その後，スレッドは処理を中断することなくデータベースに記憶された応答に応じて処理を振り分ける．また，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パケットの受け渡しや，通知イベントオブジェクトによるフック関数内のスレッドの再開を行う．なお，ドライバがシステムコールのフックを開始するタイミングは，ドライバがロードされる時点ではなく，監視アプリケーションによってドライバのデバイスをオープンした時点で開始される．

- 監視アプリケーション: このプログラムは，危険な処理の呼び出しをドライバからパケットとして受け取り，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，定期的（500ms 毎）に DeviceIoControl API（コントロールコード=受信）を使用してドライバにアクセスし，循環キューに格納された先頭のパケットを取得する．取得したパケットから，プロセス名（実行ファイルのパス）などの詳細情報を解決し，パケットのタイプがリクエストであれば，パケットの内容からユーザへ承認ダイアログを表示し，ユーザの応答を得る．次に，DeviceIoControl API（コントロールコード=送信）によって，応答に対した通知イベントをシグナル状態にし待機していたスレッドを再開する．最後に，パケットの詳細情報を CSV 形式でログファイルに出力する．また，パケットのタイプがログであった場合には，単にパケットから得た詳細情報をログファイルに出力する．なお，現在，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ている．

4.2.2 危険な処理として検出するシステムコール

本項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．危険な処理を検出するために，フックす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，承認ダイアログを表示する．

実行ファイルの作成は，Windows において実行ファイルの拡張子として使用される’.exe’ のファイルの作成を検出する．OS 起

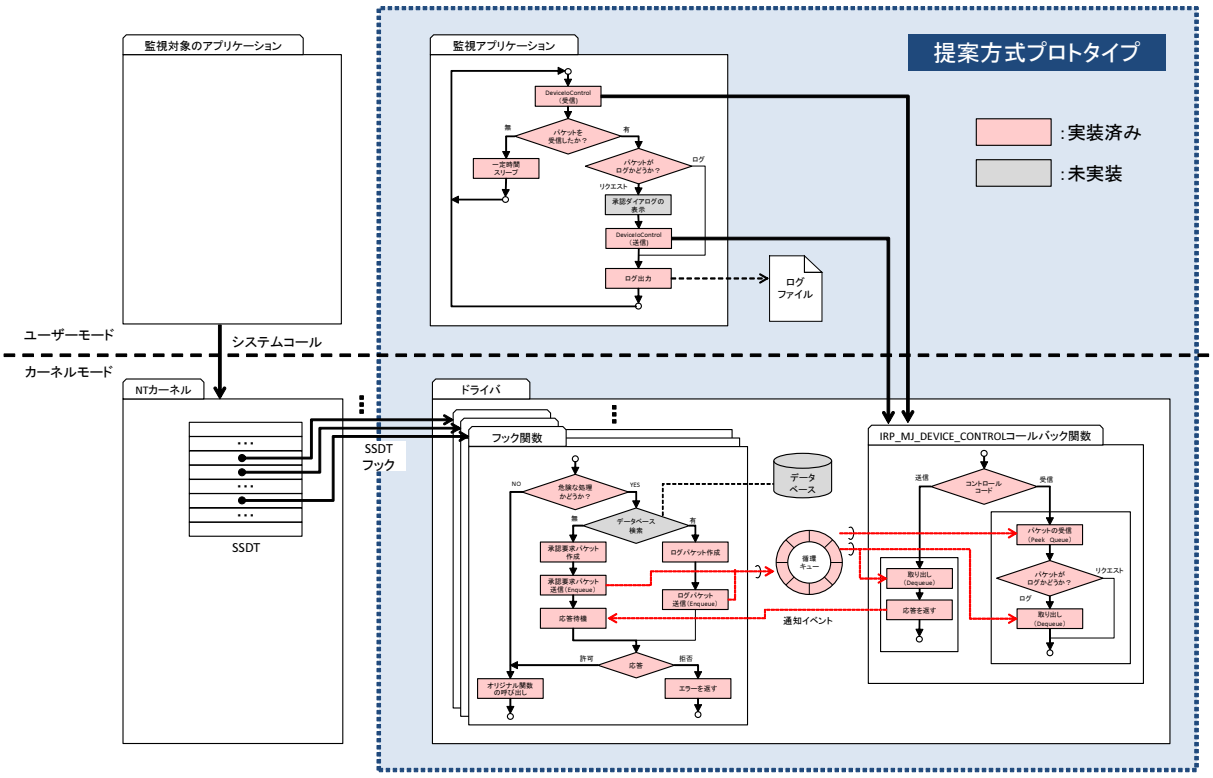


図 3: 提案方式のプロトタイプ構成

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

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，親子関係にあたるプロセスはプログラム間に関係がある可能性があるため，親プロセスが子プロセスを強制終了する場合を例外的に許可する．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）となる実装されていため，そのやり取りとなる NtDeviceIoControlFile をフックすることで検出する．

4.3 動作検証

提案方式のプロトタイプの動作検証を行った．動作環境は仮想マシン上の 32 ビット版の Windows7 である．ゲスト OS にプロトタイプのドライバをインストールし，監視アプリケーションを管理者権限で起動する．なお，このプロトタイプではホワイトリスト／ブラックリストが未実装であるため，ドライバのフック関数では危険な処理をログとして検出するようにビルドした．

ゲスト OS 上で，故意に各危険な処理を実行し，プロトタイプによってログが適切に出力されているかどうかを確認した．プロトタイプが出力した CSV 形式のログファイルの結果を表 4 に示す．表の 1 行目は，エクスプローラによって実行ファイル（test.exe）をコピーしたのを実行ファイルの作成として検出している．2 行目は，レジストリエディタで実行ファイル（test.exe）を OS 起動時に自動実行するように HKCU/SOFTWARE/Microsoft/Windows/CurrentVersion/Run へ登録したのを検出している．3 行目は，起動した Windows ペイントをタスクマネージャで強制終了させたのを検出している．ただし，プロセス名を監視アプリケーションで取得していたため，その時すでにプロセスが終了しておりプロセス名が取得できていない．このため，ドライバでプロセス名を取得するなどの改善を行う必要がある．4 行目は，Thunderbird（メールソフト）によってメールを送信したのを検出している．

動作検証から，プロトタイプシステムが危険な処理を正しく検出していることが確認できた．

5 提案方式の課題

提案方式は，以下に示す課題を抱えている．

- ホワイトリストに登録できないプログラム
  他のプロセスから危険な処理の依頼を受けるようなプログラムをホワイトリストに登録してはならない．例えば，Windows はレジストリを操作するための REG コマンドを標準搭載している．マルウェアは REG コマンドを実行して自身を自

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	フック対象となるシステムコール	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 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	NtTerminateProcess	自身あるいは親プロセス以外のプロセスから強制終了される場合
メール送信	NtDeviceIoControlFile	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

表 4: 動作検証

No	タイムスタンプ	危険な処理	プロセス名	ユーザ名	PID	TID	データベース	承認結果	詳細
1	2014/10/12/23:22:05.0455	実行ファイル作成	C:/Windows/explorer.exe	hayakawa	5980	552	登録	許可	ファイル名=/??/C:/Users/hayakawa/Desktop/test - コピー.exe
2	2014/10/12/23:22:19.0136	自動実行へ登録	C:/Windows/regedit.exe	hayakawa	1648	4196	登録	許可	値=C:/Users/hayakawa/Desktop/test.exe
3	2014/10/12/23:22:41.0460	他プロセスの強制終了	C:/Windows/system32/taskmgr.exe	hayakawa	3800	1652	登録	許可	対象プロセス名=不明なプロセス名 (PID:1664, ユーザ名:不明なユーザ名)
4	2014/10/12/23:24:18.0399	メール送信	C:/Program Files/Mozilla Thunderbird/thunderbird.exe	hayakawa	724	4264	登録	許可	ポート番号=465, 宛先 IP アドレス=114.111.99.228

動実行のレジストリに登録することが考えられる．従って，REG コマンドはホワイトリストに登録されるべきではない．同様な性質を持ったプログラムとして Windows サービスなどが挙げられる．また，この場合に表示される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．

- OS が行う危険な処理の誤検出
  今回の実装ではカーネルモードのシステムコールフックを採用しているため，OS がデフォルトで行う危険な処理についても検出する．承認によって OS が行う処理を止めてしまうと，動作に支障をきたしてしまう可能性がある．

- 承認におけるユーザによる誤検知
  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．

これらの課題は，今後プロトタイプを用いて基礎実験を行い，その実態を調査したうえで，承認ダイアログの表示方法や危険な処理を見直したり，事前にホワイトリスト／ブラックリストへ適切な登録を行うなどの検討が必要となる．

6 まとめ

本稿では，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．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．ユーザによる誤検知が少なくなるように承認対象とする危険な処理について検討した．また，提案方式はユーザの利便性を考慮しホワイトリスト／ブラックリストを導入し，ユーザの追跡調査が行えるようログ機能を備えるものとした．さらに，カーネルモードのシステムコールフックを用いて提案方式の一部を実装し・動作検証を行った．

今後は，プロトタイプの実装を進めるとともに，危険な処理に該当すると考えられるものを随時, 検討していく．また，基礎実験によって承認要求が行われるタイミングや回数の評価を行う予定である．

参考文献

[1] 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, September 2007.

[2] 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, September 2010.

[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, September 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, September 2009.

[5] 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, September 2006.

[6] 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, September 2009.

[7] 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, May 2013.

[8] 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. 情報処理学会論文誌, Vol. 62, No. 23, pp. 1–8, Jul 2013.

[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, Aug 2013.

[10] 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, Sep 2013.

[11] 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）シンポジウム論文集, Vol. 2014, pp. 1720–1727, Jul 2014.

[12] Hooking system calls through msrs. http://resources.infosecinstitute.com/hooking-system-calls-msrs/.

[13] Hooking the system service dispatch table (ssdt). http://resources.infosecinstitute.com/hooking-system-service-dispatch-table-ssdt/.

Windowsにおける危険な処理の承認機構の提案と実装

早川 顕太† 旭 健作† 鈴木 秀和† 渡邊 晃†
†名城大学大学院 理工学研究科

Watanabe Lab.

研究背景(1/2)

- マルウェアは多様化が進み，様々な活動を行う
  - 不正インストール，情報漏えい，スパムメール，etc
  - 特にWindowsマルウェアの種類・数は膨大
- これらの活動はバックグラウンドで行われる

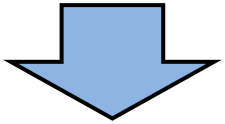
ユーザは気づくことができない

- マルウェア検出技術

	検出方法	未知 マルウェア	暗号化 マルウェア
パターンマッチング法	静的なシグネチャ検出	×	×
静的ヒューリスティック法	静的な振る舞い検出	○	×
ビヘイビア法	動的な振る舞い検出	○	○

研究背景(2/2)

- ビヘイビア法の課題
  - (課題1)振る舞い定義の難しさ
    - 多様化したマルウェアの活動を一概に定義できない
  - (課題2)正規ソフトウェアとの分離
    - マルウェアの活動の多くが正規ソフトウェアにも観測されるため，誤検知が発生しやすい



本研究

危険な処理のユーザへの承認機構を提案

- 振る舞い定義の難しさ　⇒　**複数の危険な処理を設定**
- 正規ソフトウェアの分離　⇒　**ユーザの判断を借りる**

※対象OSはWindows

Windows における承認機構に関連した既存技術

- **ユーザアカウント制御（UAC；User Account Control）**
  - Windows Vista以降，標準搭載
  - 管理者としてログインしても，標準ユーザの権限で動作
  - 昇格プロンプトにより，ユーザの承認を得れば，管理者権限で起動
    ⇒ UACはプログラム起動時の承認機構

課題

◆**プログラム実行中の承認機構ではない**
- 実際にどんな処理がいつ行われるのか分からない

◆**標準ユーザの権限で行える悪意のある活動を防げない**
- カレントユーザのみへの不正インストール
- スパムメールの送信など

Android における承認機構に関連した既存技術(1/2)

- **パーミッション機構**
  - 重要な情報や機能を利用するため，アプリに毎に与えられる権限
    - SMSの送信，ネットワークへの接続，アドレス帳や位置情報の取得など
  - アプリケーションをインストールする際にユーザがパーミッションの利用を承諾することで，アプリケーションにパーミッションが付与される

課題

- ◆ マルウェアがアプリケーションの説明文を偽り，ユーザにパーミッションの利用を承諾させ，悪用する
- ◆ アプリケーションをインストールするには，ユーザは全てのパーミッションに承諾しなければならない
- ◆ インストールで承諾されたパーミッションは，アプリケーションは自由に利用可能
  ⇒パーミッションの利用をユーザが把握できない

Android における承認機構に関連した既存技術(2/2)

- **パーミッションの利用時にユーザへ動的に承認要求を行う研究**
  - アプリケーションによるパーミッションの利用タイミングをユーザが確認
    ⇒マルウェアの不正なパーミッションの利用に気づくことが可能

    - 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, Aug 2013.
    - 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, Sep 2013.

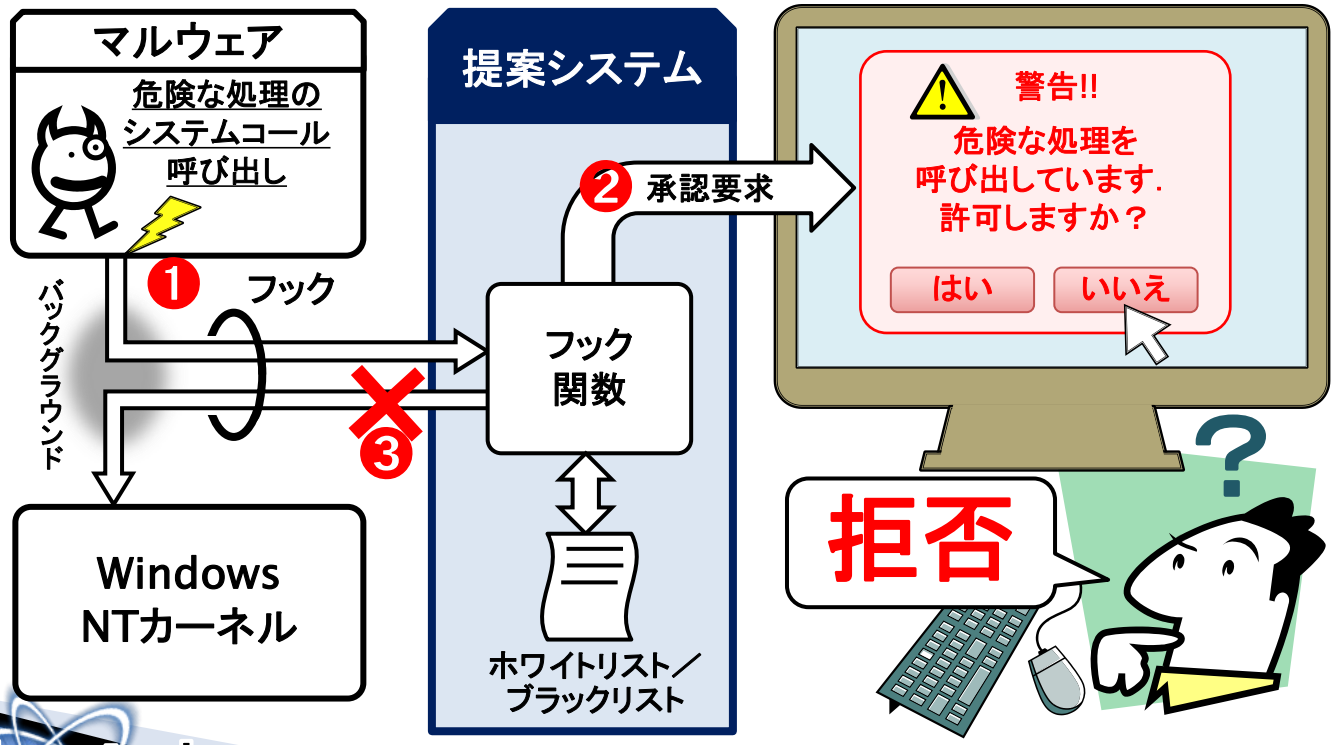
本研究

同様のシステムをWindows OSを対象に提案

- ただし, Windowsにはパーミッションと同様の機構がない
- 何を検出対象とするか検討が必要

提案方式

- **プログラムの実行中に行われる危険な処理の承認機構**
  - ❶ プログラムが危険な処理を行うために発行するシステムコールをフック
  - ❷ 承認ダイアログによる，ユーザへの承認要求
  - ❸ ユーザの応答により，処理を続行／中断



危険な処理の定義

危険な処理

- マルウェアによって悪用の恐れがある処理
- 正規ソフトウェアにおいてはユーザが推測可能

- 現在，以下の振る舞いを危険な処理として検討

危険な処理	想定される被害
実行ファイルの作成	不正インストール
OS起動時の自動実行へ登録	不正インストール
他プロセスの強制終了	セキュリティ無効化
メール送信	スパムメールの送信

承認ダイアログのイメージ

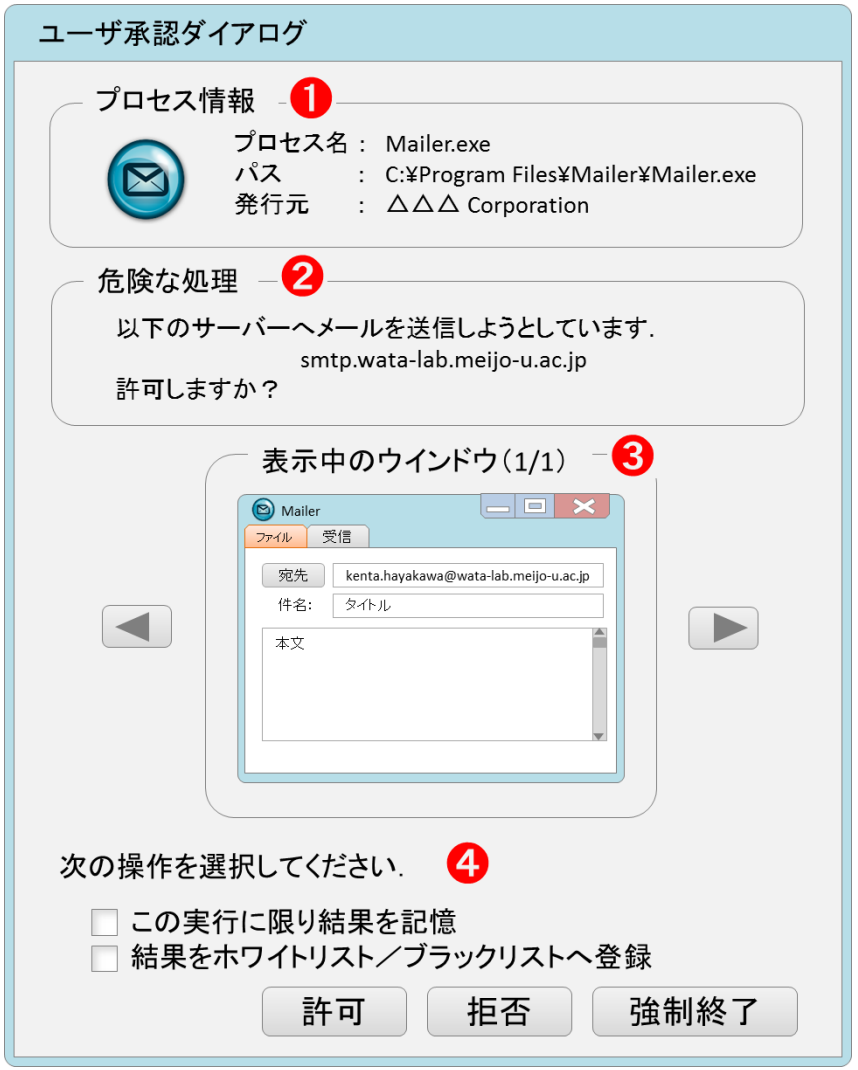
表示する情報

❶プロセスに関する情報
- プロセス名
- プロセスID
- イメージアイコン
- イメージの絶対パス
- 電子署名の発行元

❷ 行われる処理内容

❸ プロセスが表示中のウインドウ

❹ ユーザの選択肢



ホワイトリストを導入

- 登録されたプログラムは，承認ダイアログの表示を省略

ユーザビリティの向上

- ホワイトリストの登録内容
  - プログラムの絶対パスと許可される処理（複数可）

プログラム	許可される処理
Explorer	（実行ファイルの作成）
タスクマネージャ	（他プロセスの強制終了）
Internet Explorer	（実行ファイルの作成）

調査･検証実験

- **既存ウイルス対策ソフトによる承認機構の導入調査**
  - 承認機構が既存のウイルス対策ソフトにすでに導入されているかどうかを調査
- **提案方式によるマルウェア検知実験**
  - どの程度のマルウェアが危険な処理を行うか調査

調査･検証実験

- **既存ウイルス対策ソフトによる承認機構の導入調査**
  - 承認機構が既存のウイルス対策ソフトにすでに導入されているかどうかを調査
- **提案方式によるマルウェア検知実験**
  - どの程度のマルウェアが危険な処理を行うか調査

既存ウイルス対策ソフトにおける承認機構の導入調査

- **調査方法**
  - 危険な処理を行うサンプルプログラムを自作
  - ウイルス対策ソフト上でサンプルプログラムを実行し，その際ユーザへ承認要求が行われるかどうかを調査
- **調査したウイルス対策ソフト**
  - ESET Smart Security体験版(Version：7.0.302.31)
  - ノートン360体験版(Version：21.1.0.18)
  - ウイルスバスタークラウド体験版(Version：7.0.1240 - JPS7038.2551)
  - ウイルスセキュリティ体験版(Version：12.2.0224)
  - Avira Internet Security体験版(Version：13.0.0.4042)
  - カスペルスキーインターネットセキュリティ体験版(Version：4.0.0.4651(f))

既存ウイルス対策ソフトにおける承認機構の導入調査

- **実験結果**

	メールの送信	強制終了	実行ファイルの作成	自動実行への登録
A	×	×	×	×
B	×	×	×	×
C	×	×	×	×
D	×	×	×	○
E	×	×	×	×
F	×	×	×	×

既存のウイルス対策ソフトにおいて，承認機構はほとんど未導入

A：ESET Smart Security体験版
B：ノートン360体験版
C：ウイルスバスタークラウド体験版
D：ウイルスセキュリティ体験版
E：Avira Internet Security体験版
F：カスペルスキーインターネットセキュリティ体験版

調査・検証実験

- 既存ウイルス対策ソフトによる承認機構の導入調査
  - 承認機構が既存のウイルス対策ソフトにすでに導入されているかどうかを調査
- **提案方式によるマルウェア検知実験**
  - どの程度のマルウェアが危険な処理を行うか調査

マルウェア検知実験

- **実験目的**
  - どの程度のマルウェアがバックグラウンドで危険な処理を行っているかを調査
- **実験環境**
  - 仮想マシン上の32ビット版Windows 7 SP1
- **使用したマルウェア**
  - 以下のマルウェア収集サイトから独自に入手した実行可能なマルウェア　53体
    - Offensie Computing（http://www.offensivecomputing.net/）
    - VX Vault（http://vxvault.siri-urz.net/）
- **システムコールのフック方法**
  - Microsoft Researchが提供するユーザモードのAPIフックライブラリであるDetoursライブラリを使用

マルウェア検知実験

- **危険な処理を検出するためにフックするシステムコール**
  - 今回の実験では，実行ファイルの作成と自動実行への登録のみ検出

危険な処理	フック対象となるシステムコール	検出条件
実行ファイルの作成	NtCreateFile	ファイルを新たに生成し，ファイルの拡張子が”.exe”である場合
	NtSetInformationFile	拡張子が”.exe”以外のファイル”を.exe”へリネームした場合
OS起動時の自動実行への登録	NtSetValueKey	マルウェアがよく利用する自動実行に関するレジストリエントリへ書き込む場合
他プロセスの強制終了	NtTerminateProcess	自身あるいは親プロセス以外から強制終了される場合
メール送信	NtDeviceIoControlFile	コントロールコードがconnect関数の場合で，ポート番号がメール送信に関するポート番号(25,456，584)

実験結果

- 約半数のマルウェアで，承認ダイアログが表示された

これらのマルウェアは提案方式が有効

マルウェア検体数（計53体）	結果
26	危険な処理が検出された
24	危険な処理は検出されず，実行を続ける
3	エラーにより実行不可

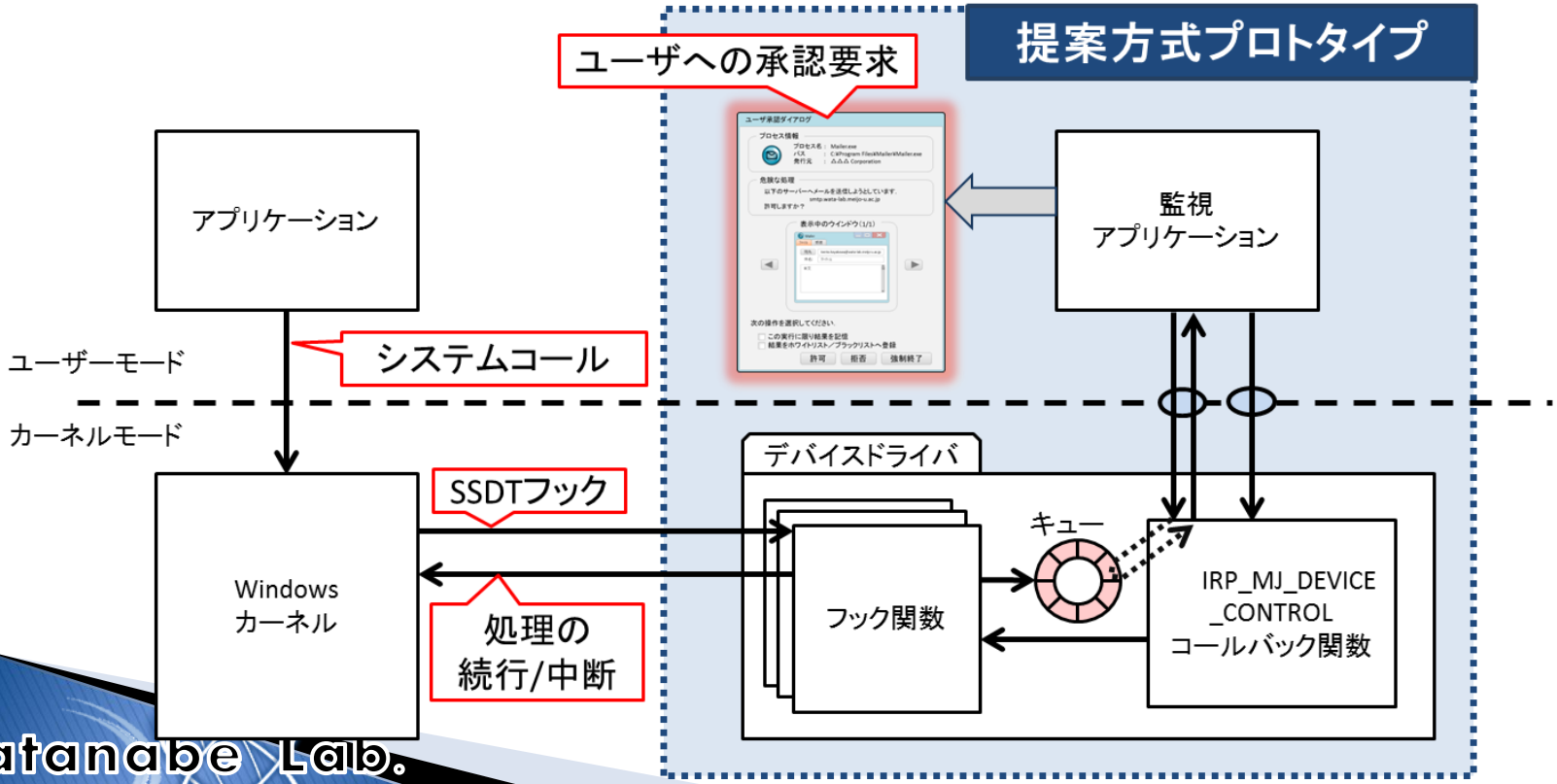
- ユーザモードのフックでの課題
  - 全てのユーザプロセスにAPIフックを仕掛ける手間が必要
  - ユーザモードのフックなので，理論的にフックの回避が可能
  - システムプロセスを経由することで検知漏れが発生
    ⇒ 正確な検知/誤検知実験が行えない

提案方式のプロトタイプ実装(1/2)

- システムコールフックの方法
  - 32ビット版のWindowsを対象とし，カーネルモードで動作するSSDTフックを採用
    - SSDTフック
      - SSDT※内のサービスルーチンのアドレスをフック関数のアドレスで上書きする方法
      - ※SSDT(SystemServiceDescriptorTable)：Windowsにおいて，システムコール番号とそのサービスルーチンアドレスの対応を記憶するカーネル内のテーブル
    - カーネル空間でのフックであるため，ユーザモードで動作するマルウェアからは回避不可能

提案方式のプロトタイプ実装(2/2)

- **システムコールフック用デバイスドライバ**
  - カーネル空間で動作し，SSDTフックによってシステムコールをフック
- **監視アプリケーション**
  - デバイスドライバに定期的にアクセスする常駐プロセス
  - 危険な処理の呼び出しを検知し，ユーザへ承認要求を行う



提案方式の今後の検討課題

- **ホワイトリストに登録できないプログラム**
  - WindowsサービスやWindows標準のコマンドなど，他のプロセスから危険な処理を依頼受けるプログラムをホワイトリストに登録できない
- **OSや正規アプリが行う危険な処理の誤検出**
  - OSや正規アプリケーションが危険な処理をユーザの意図しないところで行う可能性
    ⇒ユーザは誤って誤検知を起こしてしまう
    ⇒OSの動作を止めてしまい，動作に支障をきたす可能性
  - 例：アップデートのための実行ファイルのダウンロードなど

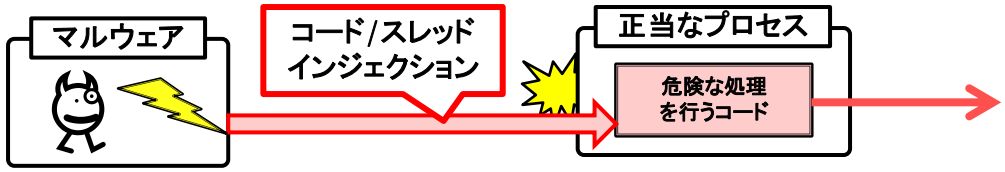
まとめ

- Windows上における危険な処理のユーザへの動的な承認機構を提案
- 実験により約半数のマルウェアで，提案方式が有効であることを確認した
- カーネルモードのシステムコールフックを用いたプロトタイプの検討・実装を行った
- 今後は正規のアプリケーションで誤検知実験を行う

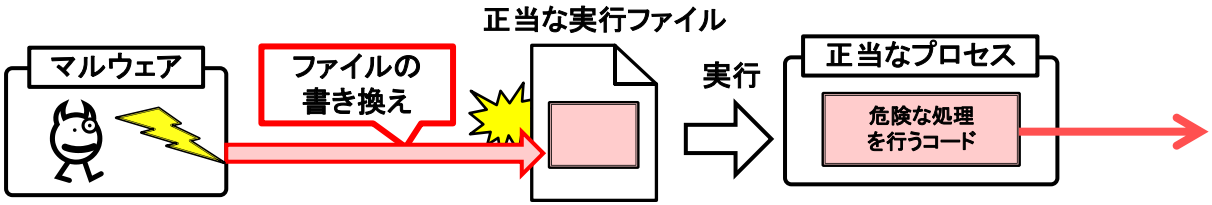
補足

正当なプロセス/プログラムへの寄生防止

- ホワイトリストに登録されたのプログラム/プロセスへマルウェアが寄生することにより，承認ダイアログを回避することが可能なため対策が必要
  - **ホワイトリストに登録済みの正当なプロセスへの寄生**



    - 【対策】ホワイトリストへ登録されたプログラムに対してスレッドの注入や仮想メモリの書き換えを禁止
  - **ホワイトリストに登録済みの正当なプログラムへの寄生**



    - 【対策】ホワイトリストへ登録されたプログラムに対してプログラムの書き換えを禁止やハッシュ値のチェックを行う

UACの昇格プロンプト