# 動的なユーザ承認による Windows 向けマルウェア対策手法

143430021　早川 顕太
渡邊研究室

## 1. はじめに

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 を対象に危険な処理のユーザへの承認機構を提案する．危険な処理の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，動的にユーザへ承認要求を行う．ユーザはマルウェアがバックグラウンドで行う危険な処理を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．

## 2. 承認機構の既存技術

承認機構の既存技術としては，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．しかしながら，UAC はプログラムの起動時に行われる承認機構であって，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う課題がある．

一方，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[1]．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．しかし，インストール時に一度パーミッションが承認されてしまうと，アプリケーションが実行時にパーミッションを利用するタイミングをユーザは把握することができない．従って，与えられた権限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．Android における承認機構の研究では，情報漏えいなどの危険性を含むパーミッションを対象として，その利用時に動的にユーザへ承認を問い合わせることにより，この課題を解決する．しかし，これらの研究は，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 に流用することができない．

## 3. 提案方式

### 3.1 概要

提案方式は，Windows を対象とした危険な処理のユーザへの動的な承認機構である．図 1 に提案システムの動作を示す．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．ユーザは行われようとしている危険な処理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て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．

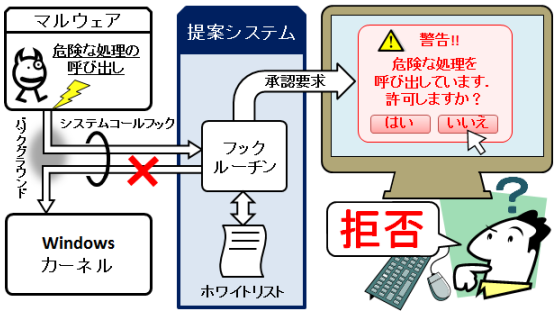



図 1. 提案システムの動作

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対象にするが，本提案はユーザの判断を借りることで，正規ソフトウェアとマルウェアで共通する振る舞いが検出可能となる．従って，従来の手法とは検出する振る舞いが異なるために，本手法により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という利点がある．

### 3.2 危険な処理の定義

本研究における危険な処理とは，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．例えば，メール送信の多くの場合，ユーザがメーラーを操作して意図的に送信する．これに対し，マルウェアの一部はバックグランドでスパムメールを送信する．従って，メール送信時に，ユーザへ承認要求を行うことにより，それが正規なものかをユーザにより判断することが可能である．既存研究を参考に危険な処理として表 1 を独自に定義した．

表 1. 危険な処理の定義

| 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 |
| 自動実行への登録 | 不正インストール |
| メールの送信 | スパムメール |
| 他プロセスの強制終了 | セキュリティの無効化 |
| スレッドの注入 | 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 ホワイトリスト・ブラックリスト機能

提案方式はホワイトリスト・ブラックリストを導入し，ユーザビリティの向上を図る．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，ユーザはその応答を任意にホワイトリスト・ブラックリストに記憶することがでる．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／拒否の決定を自動化することが出来る．

しかし，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起動することで危険な処理を回避することができる可能性がある．そこで，プロセス単位で許可／拒否される危険な処理を管理し，それを子のプロセスへ引き継がせることによりこれを防止する．また，OS が行う正常な危険な処理の誤検知により，利便性が低下してしまうこと，あるいは OS の動作に支障をきたしてしまうことを防止するため，Windows 標準のプログラム，すなわち，Microsoft 社の電子署名を持つプログラムについては信頼済みプロセスとして危険な処理を全面的に許可するものとする．

## 4. プロトタイプシステムの実装

本提案の有用性を評価するため，32 ビット版 Windows7 を対象としてプロトタイプシステムを一部実装した．システムコールフックには SSDT（System Service Descriptor Table）フックを採用した．プロトタイプはシステムコールフックを行うためのデバイスドライバと，検出した危険な処理についてユーザへ承認を行う駐在プロセスの 2 つから構成される．フック処理のフローチャートを図 2 に示す．

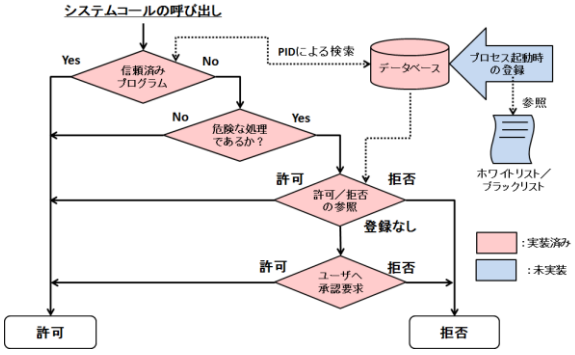



図 2. フック処理のフローチャート

## 5. 評価

### 5.1 マルウェア検知実験

プロトタイプを用いてマルウェアの検知実験を行った．仮想マシン上に本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から，独自に入手した実行可能なマルウェア 49 体である．

実験の結果，29 体のマルウェアを検出できることが確認できた．本提案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 59%に留まったが，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．従って，本方式と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．

### 5.2 オーバーヘッドの評価

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．実験に使用した実機 PC は Windows7 32bit を搭載し，CPU は Intel Core I5 1.70Hz，メモリは 4GB，SSD は 128GB である．危険な処理を行うユーザモードの API を 1000 回呼び出し，その平均時間を測定した．計測したフック処理のパスは，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．

表 2 に API の測定結果を示す．表 2 においてフック有無の差が本提案によるオーバーヘッドとみなすことができる．オーバーヘッドは最大で，18 マイクロ秒であった．API の動作時間を考えると，十分小さいといえる．メール送信については，TCP の 3 ウェイシェイクハンドのため，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 2. API の測定結果（1000 回平均）

| 危険な処理 | フック | 時間[ $\mu$ s] |
|---|---|---|
| 実行ファイルの作成 | 有 | 188.893 |
| CreateFile | 無 | 206.670 |
| 実行ファイルの作成 | 有 | 205.099 |
| SetInformationByHandle | 無 | 219.466 |
| 自動実行への登録 | 有 | 29.163 |
| RegSetValueEx | 無 | 33.649 |
| 他プロセスの強制終了 | 有 | 86.009 |
| TerminateProcess | 無 | 104.809 |
| スレッドの注入 | 有 | 56.696 |
| CreateRemoteThread | 無 | 57.022 |

## 6. 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．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った結果，58%のマルウェアを検出することに成功した．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，実際の利用ケースにおける利便性の検証を行う予定である．また， Linux などの他 OS への本提案の適用を検討する予定である．

### 参考文献

[1] 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.

# 動的なユーザ承認による Windows向けマルウェア対策手法

理工学研究科　情報工学専攻
渡邊研究室
143430021　早川 顕太

Watanabe Lab.

# 研究背景

- 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

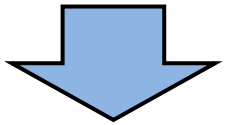
ユーザは気づくことができない

- マルウェア検出技術

| | 検出方法 | 未知・亜種マルウェア | 暗号化マルウェア | 誤検知 |
|---|---|---|---|---|
| パターンマッチング法 | 静的なシグネチャ検出 | × | × | ○ |
| ビヘイビア法 | 動的な振る舞い検出 | ○ | ○ | × |

# ビヘイビア法の課題

- **マルウェアと正規ソフトとの分離**
  - マルウェア固有の振る舞いを定義するのが困難
  - 正規ソフトウェアとの**誤検知**が問題に



**本研究**

- **ユーザの判断を借りる**ことで正規ソフトウェアとマルウェアを分離可能な場合があることに着目

**ユーザへの動的な承認機構を提案**

※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，アプリごとに与えられる権限
    - 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認

**課題**

◆インストール時の承認機構
- 実行時の不正なパーミッションの利用を把握できない

◆アプリのインストールには全てのパーミッション承認が必要

# 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問い合わせを行う研究**
  - パーミッションの不正な利用をユーザが気づくことが出来る
  - 最新のAndroid6.0ではRuntime Permissionとして標準搭載

- 川端秀明，他：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).
- 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).

**（問題点） Androidの仕組みに特化 ⇒ 他のOSに流用できない**

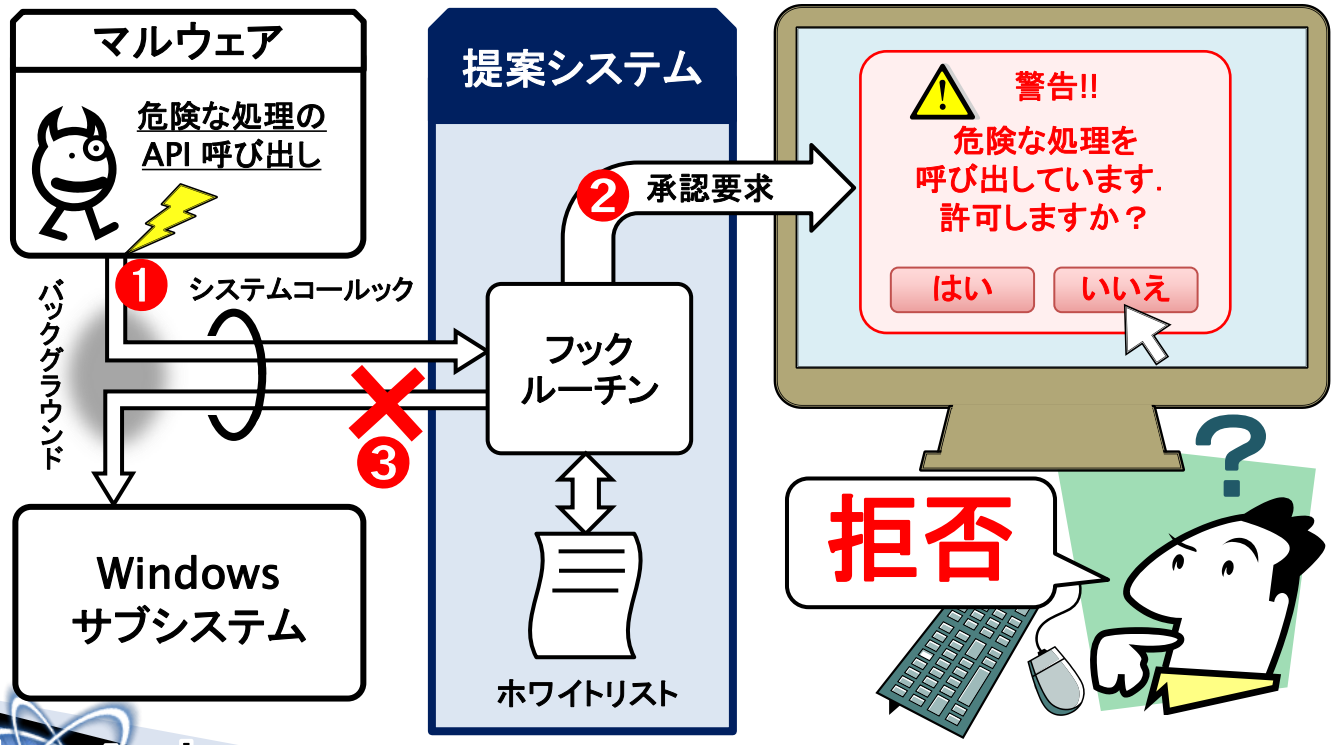
**本研究**

デスクトップOSの主流である**Windowsを対象**に

**危険な処理のユーザへの動的な承認機構を提案**

# 提案方式

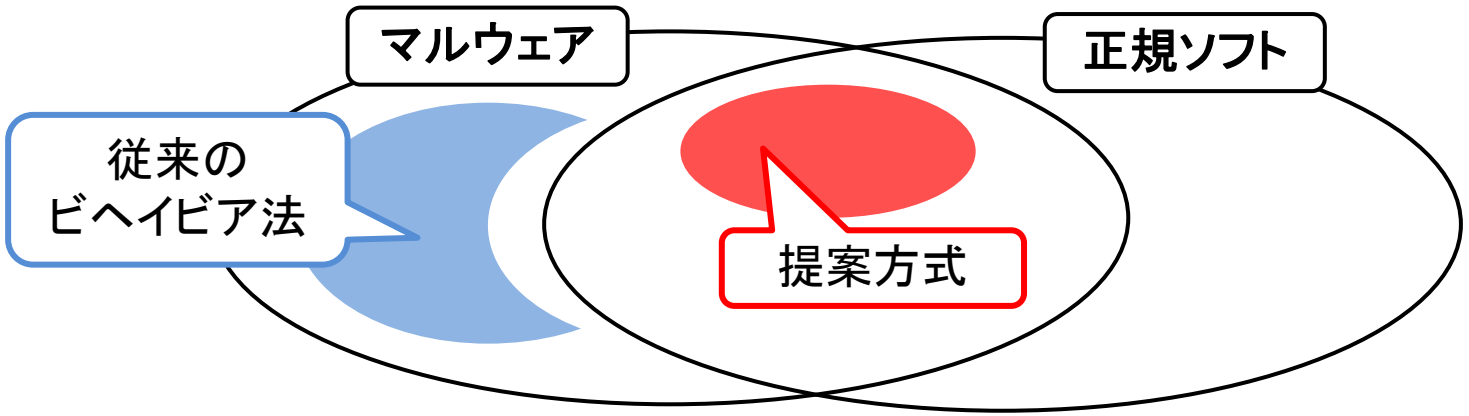
- **プログラムの実行中に行われる危険な処理の承認機構**
  - ❶ プログラムが危険な処理を行うために発行するシステムコールをフック
  - ❷ 承認ダイアログによる，ユーザへの承認要求
  - ❸ ユーザの応答により，処理を続行／中断



# 危険な処理の定義(1/2)

## (提案方式が定義する)危険な処理

- マルウェアと正規ソフトの双方が行う振る舞い
  - ⇒ 従来のビヘイビア法とは検出範囲が異なる
  - ⇒ 併用によりセキュリティが向上
- 正規ソフトにおいてはユーザが予測可能な振る舞い
  - ⇒ 承認の際ユーザが適切な判断を下せるように



# 危険な処理の定義(2/2)

- 現在以下の処理を検討

| 危険な処理 | 正常な利用 | 悪用されるケース |
|---|---|---|
| 実行ファイルの作成 | インストール | 不正インストール |
| 自動実行への登録 | インストール | 不正インストール |
| メール送信 | メールの送信 | スパムメール,<br>脅迫メール |
| 他プロセスの強制終了 | フリーズした<br>プロセスの終了 | セキュリティ無効化 |
| スレッドの注入 | 特になし | 他プロセスへの感染 |
| プログラムの改ざん | プログラムの<br>パッチ適用 | プログラムへの感染 |

# 承認ダイアログのイメージ

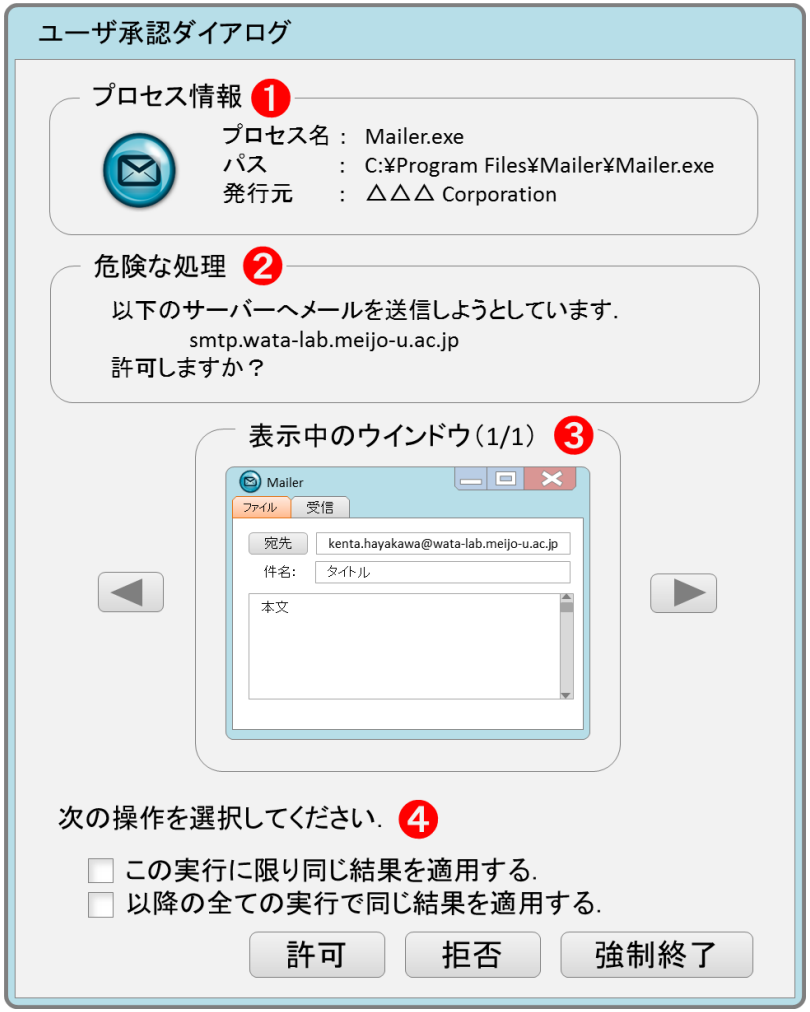
- **提示する情報**

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・ブラックリスト機能

**ホワイトリスト・ブラックリスト**
- ユーザによりプログラム単位で登録し承認要求を省略

ユーザビリティを向上

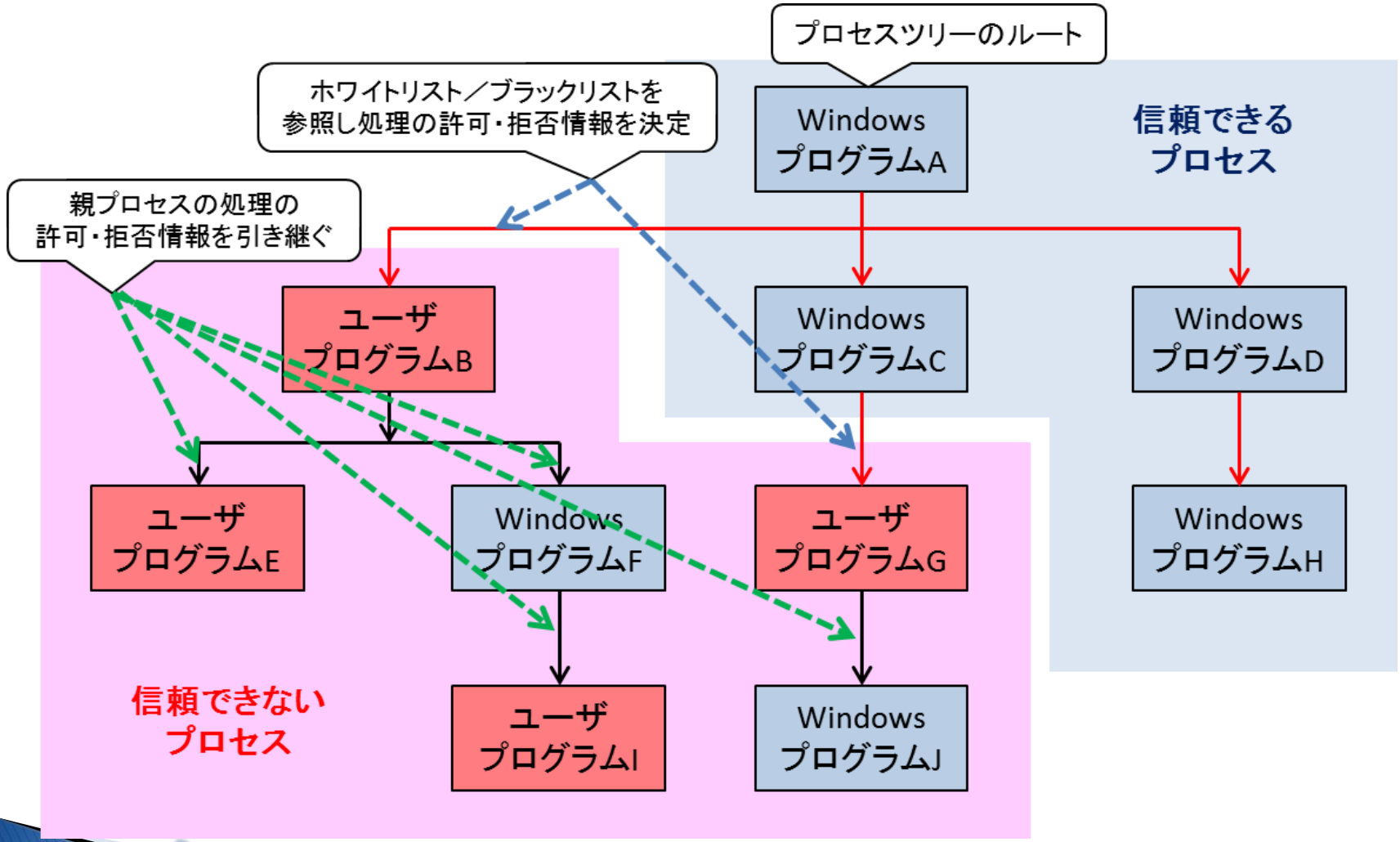
**Windows標準ソフトの処理は常に許可**
- プログラムの電子証明がMicrosoft社かどうかで判定

OSの正常な動作を止めないようにする

**処理の許可／拒否情報をプロセス単位で管理し子プロセスへ引き継ぐ**

ホワイトリスト登録済みプログラムの悪用を防止

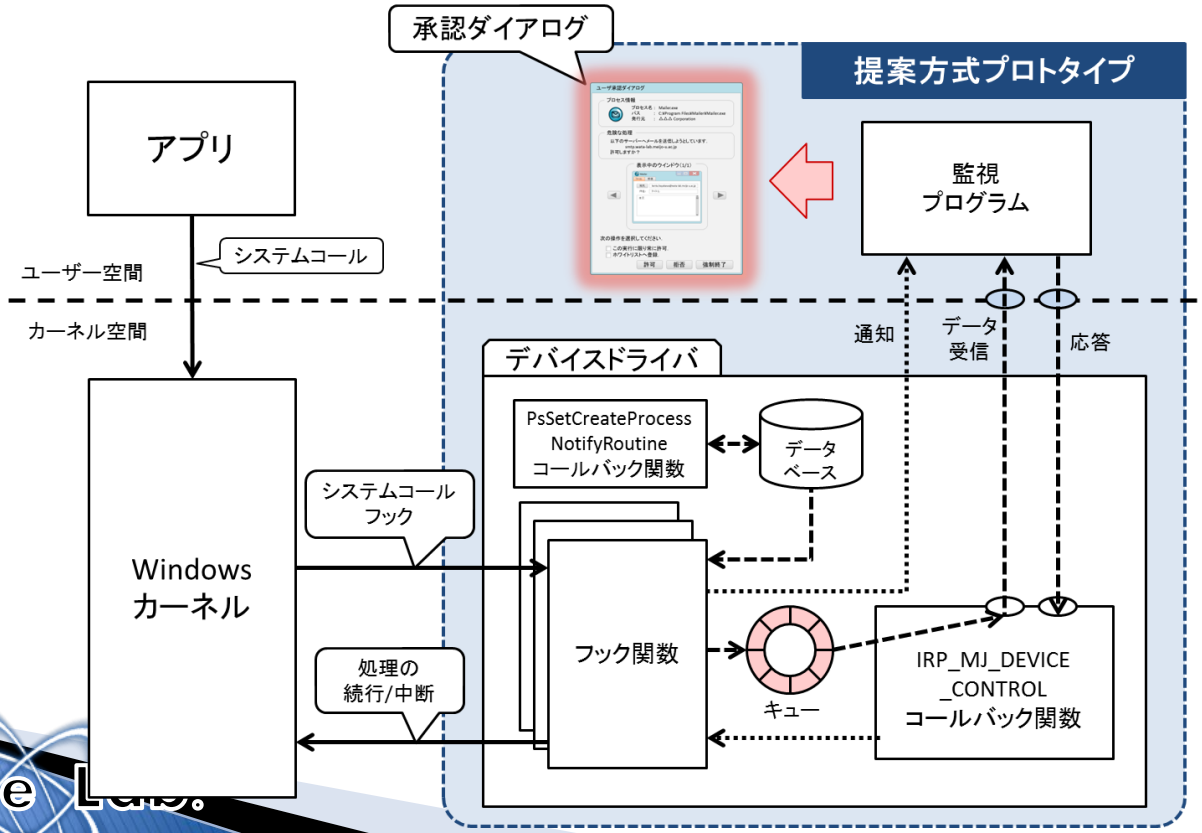
# ホワイトリスト・ブラックリスト機能の運用



# 提案方式のプロトタイプシステム実装(1/2)

- **デバイスドライバ**
  - SSDTフックを用いてシステムコールをフックし危険な処理を検出
  - プログラムの起動を検出し処理の許可・拒否情報をデータベースへ登録
- **監視プログラム**
  - 承認ダイアログの表示やログを出力する駐在プロセス

※SSDT：System Service Descriptor Table



# 提案方式のプロトタイプシステム実装(1/2)

- **実装済みの危険な処理**
  - 以下のシステムコールが条件を満たした場合に検出

| 危険な処理 | フックするシステムコール | 引数の条件 |
|---|---|---|
| 実行ファイルの作成 | ZwCreateFile | 実行ファイルの拡張子を持つ府ファイルを新たに作成した場合 |
| | ZwSetInformationFile | 実行ファイルの拡張子へとリネームした場合 |
| OS起動時の自動実行への登録 | ZwSetValueKey | 自動実行に関するレジストリのへ書き込む場合 |
| 他プロセスの強制終了 | ZwTerminateProcess | 自身と子プロセス以外のプロセスを強制終了する場合 |
| メール送信 | ZwDeviceIoControlFile | メール送信に関するポートへアクセスする場合 |
| スレッドの注入 | ZwCreateThreadEx | 自身以外のプロセスに対してスレッドを作成する場合 |

# 実験と評価

- **マルウェア検知実験**
  - プロトタイプ上でマルウェアを実行
  - マルウェアによる危険な処理を検知
- **オーバーヘッドの評価**
  - プロトタイプのフック処理のオーバーヘッドを測定

# マルウェア検知実験

- **実験環境**
  - 仮想マシンにプロトタイプシステムを導入
  - 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49体

- **実験結果**
  - 検出率は59%（29/49体）
  - 未検出のマルウェア
    - 仮想マシンを察知した
    - 動作条件に合わない

**従来のビヘイビア法とは検出範囲が異なる**
**⇒　既存手法と併用が可能**

| 危険な処理 | 検体数（全49体） |
|---|---|
| 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 |
| 全体 | 29 |

# オーバーヘッドの測定

- **実験環境**
  - 次の実機PCを使用

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

- **測定方法**
  - プロトタイプの有無で各ユーザモードのAPIを1000回計測
  - 計測したフック処理のパスは，ユーザへ問い合わせが発生しない最長のパス

# オーバーヘッドの評価

- フック処理のオーバーヘッドは最大で19 μs
  - オーバーヘッドが十分小さいことを確認

| 測定したAPI | フック | 平均時間[μs] | オーバヘッド |
|---|---|---|---|
| 実行ファイルの作成 CreateFile | 無 | 189.893 | |
| | 有 | 206.670 | +16.781 |
| 実行ファイルの作成 SetInformationByHandle | 無 | 205.099 | |
| | 有 | 219.466 | +14.367 |
| 自動実行への登録 RegSetValueEx | 無 | 29.163 | |
| | 有 | 33.649 | +4.486 |
| 他プロセスの強制終了 TerminateProcess | 無 | 86.009 | |
| | 有 | 104.806 | +18.797 |
| スレッド注入 CreateRemoteThread | 無 | 56.696 | |
| | 有 | 57.022 | +0.326 |

# 提案方式の課題

- **Windowsサービスのホワイトリスト登録**
  - マルウェアにより悪用される恐れがあるため
    ホワイトリストに登録できない
    ⇒　利便性の低下
    ⇒　発行元のプログラムの特定が困難
- **64ビット版Windowsへの対応**
  - カーネルパッチプロテクションの導入により
    システムコールフックが困難
  - ユーザモードのAPIフックへ置き換える

# まとめと今後の予定

- Windowsを対象に危険な処理のユーザへの動的な承認機構を提案
- プロトタイプシステムを実装
- マルウェア検知実験とオーバーヘッドを評価

- 今後の予定
  - 実際の利用ケースにおける利便性の検証
  - 他のデスクトップOSへの本提案の適応を検討

# 付録

# OSによる危険な処理の回数

- プロトタイプを導入した実機PCを3日間放置しログを採取
  - OSの処理を常に許可することで承認の回数を101回→4回に
    - 4回はAdobeの更新プログラムによる実行ファイルの作成
  - ユーザがアプリケーションを実行しなければ基本的に危険な処理が行われないことを確認

| 危険な処理 | プログラムの内訳 | | |
|---|---|---|---|
| | Windows | 非Windows | 不明 |
| 実行ファイルの作成 | 64 | 0 | 13 |
| 自動実行への登録 | 9 | 4 | 0 |
| メール送信 | 28 | 0 | 0 |
| 他プロセスの強制終了 | 0 | 0 | 0 |
| スレッドの注入 | 0 | 0 | 0 |
| 全体 | 101 | 4 | 13 |

# フック処理のフローチャート