Vol.2014-DPS-158 No.20
Vol.2014-CSEC-64 No.20
2014/3/6

プロセス特定困難化のためのプロセス関連情報の置換手法

佐藤 将也[1,a)]　山内 利宏[1]

概要：ウィルス対策ソフトウェアや管理ツールなど，システムの防御や管理のために用いられるソフトウェアは，マルウェアの攻撃の対象となり，無力化される可能性がある．そこで，攻撃者から重要なソフトウェアの特定を困難にし，攻撃の対象から除外する手法として，プロセス関連情報の不可視化によりプロセスの特定を困難にする攻撃回避手法を提案した．提案手法は，特定困難化の対象となるプロセスのプロセス関連情報を偽の情報に置換することにより，プロセス関連情報をもとにしたプロセスの特定を困難にする．提案手法は，仮想計算機モニタを用いることで，ゲスト OS のソースコードの改変や機能の追加なしに実現する．本稿では，提案手法のうち，プロセス関連情報の置換手法について，Linux を対象とした実現方式を述べる．この手法では，ゲスト OS におけるプロセス切り替えを仮想計算機モニタにより検知し，特定困難化の対象となるプロセスの走行中のみ本来のプロセス関連情報を用い，走行中以外は偽の情報に置換する．これにより，対象のプロセスの走行を妨害することなく，攻撃者によるプロセス特定を困難化する．

1. はじめに

プログラムの脆弱性を利用し，計算機を不正に操作する攻撃が増加している．このような攻撃への対策として，ウィルス対策ソフトウェアのように，攻撃の防止や攻撃による被害の抑制を目的としたソフトウェアが研究開発されている．しかし，このようなソフトウェアは，攻撃者によって不都合である場合が多いことから，攻撃の対象となり，無力化される可能性がある．例えば，Agobot [1] は，ウィルス対策ソフトウェアを停止する機能を持つ．また，t0rnkit [2] やその亜種である dica [3] は，マルウェアのインストールをシステム管理者から隠ぺいするために，ログ収集プログラムを停止する．これらのソフトウェアを停止または無効化されると，システムへのマルウェア感染や被害の拡大を招く．このため，ウィルス対策ソフトウェアやログ収集プログラムのように，システムの防御や管理のために用いられるソフトウェア（以降，重要サービス）への攻撃の検知と防止，被害の抑制，および攻撃の回避が重要な課題である．

重要サービスへの攻撃を防止するために仮想計算機モニタ（以降, VMM）を利用する手法が提案されている [4], [5], [6]．これらの手法は，仮想化技術を利用し，攻撃対象の計算機とは異なる計算機に隔離して重要サービスを動作させることで，攻撃による重要サービスへの影響を防止している．しかし，既存の重要サービスを改変なしには利用できず，既存のソフトウェアを有効に活用できない．

そこで，文献 [7] において，攻撃者から攻撃対象のプロセスの特定を困難にする攻撃回避手法を提案した．この手法では，重要サービスを提供するプロセスに関する情報を VMM により不可視化することで，当該プロセスの特定を困難にし，攻撃を回避する．また，この機能を提供する機構自体を攻撃者から検知されないようにし，機能の停止や無効化を困難にする．

本稿では，文献 [7] で提案した手法のうち，プロセス特定困難化のために重要サービスを提供するプロセスに関する情報を置換する手法について述べる．この手法では，ゲスト OS におけるプロセス切り替えを VMM により検知し，特定困難化の対象となるプロセスの走行中のみ本来のプロセスに関する情報を用い，走行中以外は偽の情報に置換する．これにより，対象のプロセスの走行を妨害することなく，攻撃者による攻撃対象プロセスの特定を困難にする．また，提案手法を導入した環境において，プログラム名をもとにしてプロセスを停止する攻撃を回避できるか否かを評価した結果を報告する．

2. 研究背景

2.1 ウィルス対策ソフトウェアへの攻撃

ウィルス対策ソフトウェアを攻撃するマルウェアとして，Agobot [1] がある．Agobot は，Windows を対象としてバックドアを設置するマルウェアである．また，プロセスの一覧からプロセスの実行ファイル名を検査し，ウィルス対策ソフトウェアに利用されているプログラム名が含まれ

[1] 岡山大学 大学院自然科学研究科
[a)] m-sato@swlab.cs.okayama-u.ac.jp

ている場合は，そのプロセスを停止する機能を持つ．2013年8月8日時点で，Agobot には，停止対象として 579 種類のプログラム名が定義されている．ウィルス対策ソフトウェアが停止された場合，システムへの被害が拡大する．

ログ収集プログラムを停止するマルウェアとして，t0rnkit [2] やその亜種である dica [3] がある．t0rnkit は，Linux を対象としたルートキットであり，バックドアの設置や自身の隠ぺいを行うツール群を含む．t0rnkit は，関連するプログラムのインストール時に，インストールの過程をシステム管理者から隠ぺいするために，Linux のログ収集プログラムである syslog デーモンを停止し，インストール後に再起動する．このため，システム管理者は，ログからは t0rnkit のインストール操作や存在を検知できない．

このように，マルウェアには，侵入後の活動の妨げとなるソフトウェアの停止や無効化を行う機能を持つものがある．重要サービスが攻撃を受けて停止または無効化された場合，システムへの被害が拡大する．このため，重要サービスへの攻撃の検知と防止，被害の抑制，および攻撃の回避により，システムへの被害の抑制が求められる．

2.2 既存手法

重要サービスへの攻撃を防止する研究として，ホスト型侵入検知システム（以降，IDS）を VMM により実現した VMwatcher [4] がある．VMwatcher は，OS よりも攻撃が困難な VMM により IDS を実現することで，IDS への攻撃を困難にしている．同様に，VMM を用いてカーネルレベルルートキットの実行を防止する手法として，NICKLE [5] が提案されている．NICKLE は，カーネルコードの実行を VMM により監視し，認証したカーネルコード以外の実行を防止する手法である．これらの手法は，既存手法では検知や防止が困難な攻撃に対し，VMM を用いることで対処している．

一方，これらの手法には，既存のマルウェア対策ソフトウェアを再利用できない問題がある．このような問題を解決する手法として，仮想計算機（以降，VM ）を用いる環境において，IDS を改変なしに VM 外部へオフロードする手法 [6] が提案されている．この手法は，システムコールエミュレータをオフロード先の VM に作成し，既存 IDS からのシステムコールをシステムコールエミュレータにより処理し，監視対象の VM の情報を IDS に返却することで，既存 IDS を改変なしに利用できる．

2.3 既存手法の問題点

既存手法には，重要サービスを提供する既存ソフトウェアを修正なしに利用できない問題がある．これまでにマルウェア対策やシステム管理のために多くのソフトウェアが開発されている．これらのソフトウェアは，そのソフトウェア自身が安全な環境においては有用である．しかし，これらのソフトウェアが攻撃された場合には，システム管理者はその機能を利用できない．2.2 節で述べた手法のうち，既存のソフトウェアと同様の機能を VMM により実現する手法は，VMM の特徴により，攻撃による被害を受けにくい利点がある．しかし，既存ソフトウェアを VMM により再実装する工数は大きい．また，文献 [6] の手法は，既存 IDS を改変なしに利用するためにシステムコールエミュレータを実現しているものの，uname システムコールしかエミュレートされていない．IDS が利用するシステムコールは IDS ごとに異なるため，IDS などの既存ソフトウェアにおいて利用されるシステムコールすべてについてシステムコールエミュレータを実現する工数は大きい．

このように，VMM を用いた有効な手法は提案されているものの，既存研究の多くは，既存ソフトウェアを修正なしには利用できず，その有用性を活用できていない．また，既存ソフトウェアの機能を VMM に実現するのは工数が大きい．さらに，応用プログラム（以降，AP）やカーネルとして動作する既存ソフトウェアの取得できる情報と VMM から取得できる情報にはセマンティックギャップがあるため，完全に既存ソフトウェアと同等の機能を実現するのは難しい．

3. プロセスの特定を困難にする攻撃回避手法

3.1 目的

本研究の目的は以下の 2 つである．

(目的 1) 重要サービスへの攻撃の回避

(目的 2) 既存ソフトウェアの改変なしの利用

攻撃防止のために多くの手法が提案されているものの，攻撃の多様化への対処が困難である．このため，本研究では，攻撃の防止ではなく，重要サービスへの攻撃を回避することを目的とする．また，既存ソフトウェアと同等の機能を VMM により実現する工数は大きい．このため，改変なしの既存ソフトウェアの利用を目的とする．

3.2 基本方式

3.1 節の目的を達成する手法（以降，提案手法）を文献 [7] において提案した．ここでは，提案手法の基本方式を述べる．提案手法は，攻撃者が攻撃対象のサービスを攻撃する際にサービスの存在を特定することに着目し，重要サービスを提供するプロセスに関する情報（以降，プロセス関連情報）を不可視化することにより，当該プロセスの特定を困難にし，攻撃を回避する．また，この機能を提供する機構自体を VMM 内に実現し不可視化することにより，機能の存在自体を攻撃者から検知されないようにし，機能への攻撃を困難にする．

VMM は，VM を提供するための機能に限定して開発されており，ゲスト OS からアクセスするインタフェースも少なく，ソースコードの規模も OS と比べて小さい．この

ため，VMM への攻撃は OS への攻撃よりも困難であるといえる．また，提案手法の実現において，ゲスト OS や重要サービスのソースコードの変更や機能の追加は行わない．VMM のみの改変により提案手法を実現することにより，既存のソフトウェアを改変なしに活用できる．これらの理由から，提案手法を VMM の改変により実現する．

3.3 重要サービスを提供するプロセスに関する情報の不可視化

重要サービスを不可視化する手法として，重要サービスを提供するプロセス（以降，重要プロセス）のプロセス関連情報を不可視化することで，このプロセスの特定を困難にする手法を提案した [7]．プロセスの特定を困難にする手法は，以下の 2 つの機能からなる．

(1) プロセス関連情報へのアクセス制御
(2) プロセス関連情報の置換

プロセス関連情報へのアクセス制御について，**図 1** に概略図を示す．この方式では，事前にプロセス関連情報へのアクセスを許可するカーネルテキスト部の領域を設定しておき，また，プロセス関連情報が存在するページの読み込みを禁止しておく．読み込みを禁止したページへのアクセス違反が発生した際に，アクセス元の命令が配置されているアドレスに応じて返却する値を変更し，アクセスを許可するカーネルテキスト部以外からのアクセスに対しては，偽の情報を返却する．これにより，本来のプロセス関連情報を攻撃者から不可視化できる．

プロセス関連情報の置換では，重要プロセスのプロセス関連情報を事前に偽の情報に置換しておき，重要プロセスの走行中は，本来のプロセス関連情報を利用する．重要プロセスから通常プロセスへのプロセス切り替えが発生すると，**図 2** に示すように，重要プロセスのプロセス関連情報を退避し，偽の情報に置換する．また，通常プロセスから重要プロセスへのプロセス切り替えが発生すると，**図 3** に示すように，重要プロセスのプロセス関連情報を復元する．これにより，他のプロセスからは，重要プロセスの本来のプロセス関連情報を参照できない．また，重要プロセスの走行を妨げない．プロセス関連情報の置換については，4 章で詳述する．

3.4 不可視化対象プロセスの特定方法と対処

3.4.1 不可視化対象プロセスの特定方法

攻撃者は，提案手法の存在を検知し，不可視化対象のプロセスを特定できた場合には，そのプロセスを停止または無力化できる．このため，あるプロセスについて，そのプロセスが不可視化対象か否かの攻撃者による判定を困難にする必要がある．

VM 上の AP やカーネルから不可視化対象プロセスを特定する方法として以下がある．

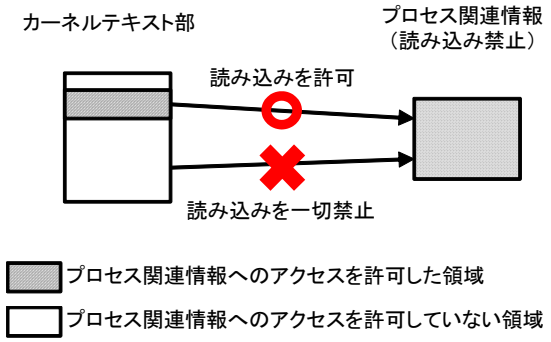

図 1 プロセス関連情報へのアクセス制御

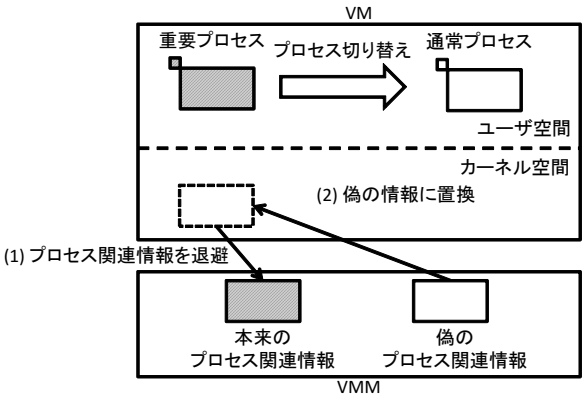

図 2 重要プロセスから通常プロセスへの切り替えにおけるプロセス関連情報の置換

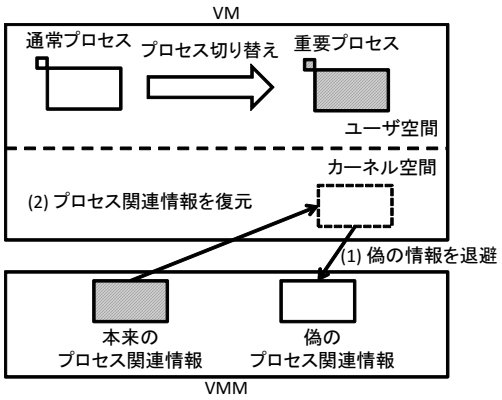

図 3 通常プロセスから重要プロセスへの切り替えにおけるプロセス関連情報の置換

(1) プロセス切り替えの処理時間の差異の検知
(2) プロセス関連情報の継続的な監視

不可視化対象プロセスは，プロセス切り替え発生時に，提案手法によりプロセス関連情報の不可視化処理が実施される．このため，プロセス切り替えの処理時間が他のプロセスよりも長い．

また，攻撃者がカーネルレベルで動作するソフトウェアを VM 上で動作させていた場合，攻撃者は，あるプロセスのプロセス関連情報を継続的に監視し，プロセス切り替えの度に検査することで，不可視化対象プロセスか否かを

検出できる．不可視化対象プロセスはプロセス切り替えの度にプロセス関連情報が置換される一方，他のプロセスは走行状態などを示す変数以外は変化しない．このため，プロセス切り替え時に本来変化しない変数が変化していた場合，攻撃者は，そのプロセスを不可視化対象プロセスと特定できる．

3.4.2 不可視化対象プロセスの特定方法への対処

3.4.1 項の (1) の方法による検知には，プロセス切り替え時に不可視化対象プロセスに対して行っている処理と同等の処理を VMM において実施することで対処できる．ただし，すべてのプロセスのプロセス切り替えにおいて性能低下が生じるため，システム全体の性能は低下する．

または，文献 [8] でマルウェア解析機能の一部として提案されているタイムコントロール機能の利用が考えられる．この機能は，特定の関数の実行にかかる時間を予め計測しておき，その関数実行時に，予め計測した時間との実実行時間の差異によりデバッガの存在を検知し，動作を停止するマルウェアを対象としている．タイムコントロール機能は，解析対象の実行を一時停止させる場合に，ゲスト OS 内の CPU クロックを完全に停止し，すべての仮想ハードウェアの動作を一時停止させる．この機能を応用し，提案方式においてゲスト OS から VMM に処理が移行した際に，ゲスト OS 内の CPU クロックを停止させることで，提案手法によるプロセス関連情報の置換処理にかかるオーバヘッドを隠ぺいできる．これにより，提案手法の導入によるオーバヘッドをもとにした不可視化対象プロセスの検知を回避できる．

3.4.1 項の (2) の方法による検知には，プロセス関連情報へのアクセス制御とプロセス関連情報の置換を連携させることで対処できる．ここでは，ゲスト OS として Linux を想定し，攻撃者が Loadable Kenrel Module (以降，LKM) を用いてプロセス関連情報の継続的な監視を行なっているものとする．まず，準備として，重要プロセスのプロセス関連情報のカーネルからの読み込みを禁止し，カーネルモジュールを含まないカーネルコードの仮想アドレス範囲を決定しておく．その後，重要プロセスのプロセス関連情報へのアクセス違反により VMM へ処理が移行した際に，**図 4** に示す手順に従い，メモリアクセスを許可するか否かを決定する．まず，VM 上の命令ポインタが事前に決定した範囲か否かを判定する．範囲外であった場合は，偽のプロセス関連情報を返却する．範囲内であった場合は，カーネルスタックを遡り，カーネル内関数の呼び出し元の関数すべての仮想アドレスを取得し，そのすべてが決定した範囲内であった場合のみ，メモリアクセスをエミュレートする．そうでない場合は，プロセス関連情報を置換する．以上の手順により, 事前に決定した範囲外からのプロセス関連情報の読み込み時には偽のプロセス関連情報を返却することで，継続的なプロセス関連情報の監視による不可視化対象プロセスの特定を回避できる．

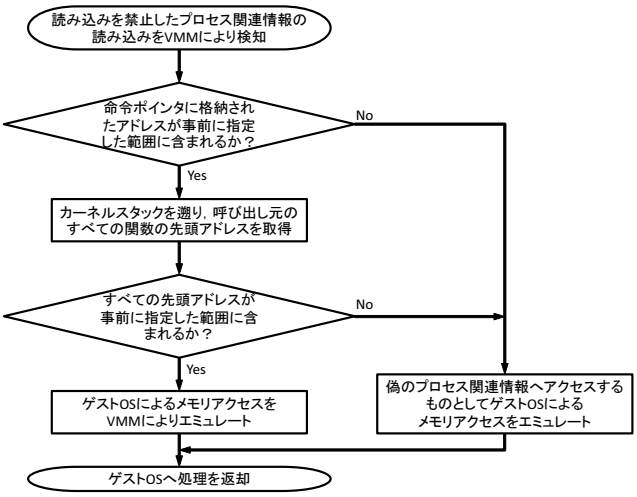

図 4 重要プロセスのプロセス関連情報へのアクセスを許可するか否かの判定

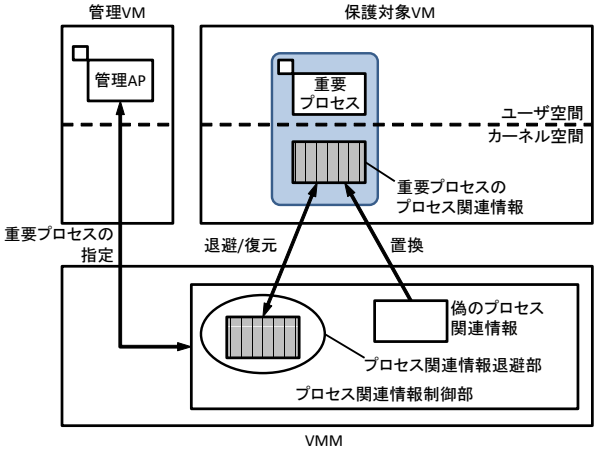

図 5 提案手法の全体像

3.5 提案手法の構成

以上の対処を実現した場合の提案手法の全体像を**図 5** に示す．提案手法は，プロセス関連情報の置換とプロセス関連情報へのアクセス制御を行うプロセス関連情報制御部を VMM 内に追加することで実現する．プロセス関連情報制御部は，保護対象 VM 上で発生するプロセス切り替えを監視し，プロセス切り替えが発生した際には，重要プロセスのプロセス関連情報の退避，偽の情報への置換，および本来の情報への復元を行う．プロセス関連情報は，VMM 内に確保したプロセス関連情報退避部に退避する．プロセス関連情報退避部は，保護対象 VM ごとに VMM 内に確保し，管理する．偽のプロセス関連情報は，予め複数種類を用意しておき，プロセス関連情報の置換の際にどの情報を用いるかを決定する．

提案手法において，保護対象 VM 上のどのプロセスを重要プロセスとするかは，管理 VM で動作する制御 AP によ

り指定する．このため，提案手法を利用する場合には，保護対象 VM の管理者は，事前に，どのプロセスを重要プロセスとするかを VMM の管理者に連絡し，VMM の管理者が管理 AP を用いて保護対象プロセスを VMM に通知し，指定する．

3.6 提案手法の限界

提案手法は，重要プロセスの攻撃者による特定を困難にするのみであり，プロセス自体を不可視にはしない．このため，偽のプロセス関連情報に置換した状態のプロセスは，攻撃者から認識できる．このことから，攻撃者が無作為にプロセスを停止させた場合には，重要プロセスを停止される可能性がある．

重要プロセスへの攻撃を防止するには，3.3 節で述べたプロセス関連情報へのアクセス制御が有用である．ただし，この攻撃への対処は，重要サービスへの攻撃の回避という本研究の目的とは異なる．このため，本稿では，詳細な検討はしない．

4. プロセス関連情報の置換手法

4.1 置換対象のプロセス関連情報

4.1.1 プロセス関連情報の定義

ゲスト OS として x86 または x64 アーキテクチャを対象とした Linux を用いた場合，プロセス関連情報として以下がある．

(1) プロセス制御ブロック
(2) カーネルスタック
(3) ハードウェアコンテキスト
(4) ページテーブル
(5) プロセスの利用しているメモリ

プロセスを完全に不可視化するには，上記のプロセス関連情報をすべて不可視化する必要がある．しかし，(3)，(4)，および (5) の情報をもとにプロセスが何かを特定するのは困難である．一方，(1) と (2) の情報は，プロセスを特定するために有用な情報を多く含む．このため，本稿における実現方式では，上記のプロセス関連情報のうち，(1) と (2) の 2 つの情報を扱う．

以下にそれぞれのプロセス関連情報の詳細を示す．

(1) プロセス制御ブロック (`task_struct` 構造体)
プロセス制御ブロックには，PID (Process ID)，TGID (Thread Group ID)，プロセスの実行ファイル名，および親プロセスの PID など，プロセスの特定に利用できる情報が多く含まれている．Linux においては，プロセス制御ブロックは，`task_struct` 構造体として定義されており，プロセスやスレッドごとに作成される．

(2) カーネルスタックと `thread_info` 構造体
カーネルスタックと `thread_info` 構造体は，共用体としてカーネルにより確保されている．カーネルスタックは `task_struct` 構造体ごとに確保される．カーネルスタックには，カーネル内で呼び出したカーネル内関数のアドレス，引数，および戻り値などが格納されている．`thread_info` 構造体は，`task_struct` 構造体と相互にリンクされている．これらの情報はプロセスの特定に利用できる．

4.1.2 置換対象

プロセス関連情報として 4.1.1 項で定義した情報は，プロセススケジューリングやシグナルの配送など，プロセスが走行中でない場合に利用される情報を含む．このため，プロセス関連情報のすべてを偽の情報に置換した場合，ゲスト OS におけるプロセススケジューリングやシグナルの配送を妨害する問題がある．これらのことから，置換対象とするプロセス関連情報を選択する基準として，プロセスの特定が可能か否か，また，プロセスの走行を妨害しないかの 2 つがある．

そこで，プロセス関連情報を置換する方針として，以下の 2 方針がある．

(置換方針 1) プロセスの動作に最低限必要な情報のみ本来の情報を設定し，その他のできるだけ多くの情報を置換

(置換方針 2) 攻撃者によるプロセスの特定を困難にできる情報のみを置換

(置換方針 1) は，(置換方針 2) よりもプロセスの特定が困難になる．しかし，多くの情報を置換する必要があるため，メモリコピーによる性能低下が大きい．一方，(置換方針 2) は，プロセスの特定に利用される少数の情報のみを置換するため，性能低下は小さい．ただし，(置換方針 2) は，実在するマルウェアがどのような情報から攻撃対象のプロセスを特定しているのかを調査する必要がある．

攻撃への対策の観点からは，(置換方針 1) が望ましい．しかし，実運用を考慮すると，できるだけ性能低下を抑えながら攻撃対象プロセスの特定をある程度困難にできる (置換方針 2) の方が実用的であると考えられる．このため，本稿における実現では，(置換方針 2) を採用する．

4.1.3 攻撃対象プロセスの特定に利用される情報

マルウェアが攻撃対象プロセスの特定に利用する情報について考察する．Agobot は，Windows を対象としており，走行中プロセスの一覧からプログラムの実行ファイル名を探索し，予め作成しておいたプログラム名の一覧に一致するエントリがあった場合，TerminateProcess 関数により，そのプロセスとプロセスが所属するすべてのスレッドを終了させる．t0rnkit の亜種である dica は，Linux を対象としており，killall コマンドにより syslogd を停止させる．killall コマンドは，proc ファイルシステムを利用し，走行中のプロセスのうちプログラム名に引数で渡された文字列を含むプロセスの PID を取得し，kill システムコールにより当該プロセスを終了させる．

この他にも，プロセスを停止させるマルウェアは多く存在するものの，その多くは，プログラム名をもとにプロセスを終了させる．このため，プロセス関連情報の置換において，プロセスのプログラム名の置換が効果的である．

4.1.4 偽のプロセス関連情報

本稿における実現方式では Linux を対象としている．このため，4.1.3 項で述べた情報より，Linux におけるプロセス関連情報のうち，プロセスのプログラム名が格納されている情報を置換する．

killall コマンドがプログラム名を取得する際には，`/proc/[PID]/stat` を読み込む．ここで，`[PID]` には，走行中のプロセスの PID が割り当てられる．`/proc/[PID]/stat` をプロセスが参照すると，カーネルは，プロセスが用意したバッファに，プロセスに関する情報をスペース区切りで格納する．この情報には，`task_struct` 構造体のうち，`pid`，`comm`，および `state` などが含まれる．`comm` メンバには，プロセスのプログラム名が格納されている．このため，`comm` メンバを置換することで，プログラム名をもとにプロセスを停止する攻撃を回避できる．

プログラム名を置換する場合に，固定の文字列に置換すると，攻撃者は，攻撃対象とするプログラム名に文字列を追加するだけで対処できる．このため，プログラム名を置換する場合は，他のプロセスから実行されることの多いプログラムのプログラム名や複数走行することの多いプロセスのプログラム名に置換する．他のプログラムから実行されることの多いプログラム名に置換した場合，攻撃者がそのプロセスを停止すると，他のプロセスの動作に問題が生じ，システム管理者が異変に気付きやすくなる．また，複数走行することの多いプロセスのプログラム名に置換した場合，同時に走行するプロセス数が増加しても，そのプロセスが本来の動作として複数走行しているのか提案手法により複数走行しているように見えるのかを区別できない．このようなプログラムの例として，udevd や apache2 がある．

4.2 プロセス関連情報を置換する契機

プロセス関連情報の置換では，プロセス切り替えの発生時に，切り替え元プロセスと切り替え先プロセスのそれぞれが不可視化対象のプロセスか否かを判断し，プロセス関連情報を置換する．このためには，VMM からゲスト OS におけるプロセス切り替えを検知する必要がある．

完全仮想化環境では，ゲスト OS は VMX non-root モードで，VMM は VMX root モードで動作する．VMX non-root モードで許可されていない操作を行うと，VM exit が発生し，VMX root モードで動作するソフトウェアに処理が移る．許可されていない操作の中には，CR3 レジスタへの書き込みがある．多重仮想記憶を利用している OS では，プロセス切り替え時にメモリ空間を切り替えるために，

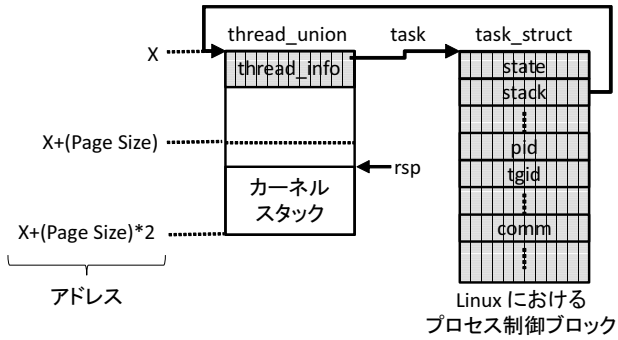

図 6 カーネルスタックとプロセス関連情報の関係

ページディレクトリの先頭アドレスが格納されている CR3 レジスタを書き換える．このため，CR3 レジスタへの書き込みによる VM exit を契機とすることで，プロセス切り替えを VMM により検知できる．

4.3 ゲスト OS のプロセス関連情報の取得

4.3.1 切り替え元プロセスのプロセス関連情報の取得

VM exit 発生時に VMM から参照できるのは，主に，VM の利用しているレジスタである．ゲスト OS のプロセス関連情報である `thread_info` 構造体と `task_struct` 構造体は，**図 6** に示すように，カーネルスタックのスタックポインタである rsp レジスタより計算できる．カーネルスタックの最大サイズはカーネルコンパイル時に確定するため，カーネルスタックの最大サイズをもとに，`thread_info` 構造体の先頭アドレスを rsp レジスタより取得できる．また，`thread_info` 構造体の `task` メンバは，そのプロセスの `task_struct` 構造体の先頭アドレスを保持している．これにより，`task_struct` 構造体を取得できる．以上のことより，VMM は，VM の rsp レジスタの値をもとに，ゲスト OS のプロセス関連情報を取得できる．ただし，VMM は，事前にゲスト OS の `thread_info` 構造体と `task_struct` 構造体の定義を保持している必要がある．

4.3.2 切り替え先プロセスのプロセス関連情報の取得

4.3.1 項で述べた方法は，rsp レジスタを用いているため，走行中プロセスのプロセス関連情報の取得にしか利用できない．このため，切り替え先プロセスのプロセス関連情報の取得方法を検討する必要がある．

取得方法

プロセス切り替え発生時に VMM から得られる情報のうち，切り替え先プロセスの特定に利用できる情報として，CR3 レジスタへ書き込まれようとしている値がある．この情報をもとに切り替え先プロセスのプロセス関連情報を取得する方法として，以下の方式が考えられる．

(走査方式) ゲスト OS 上のプロセスリストを走査する方式

(リスト管理方式) 可視化対象のプロセスの CR3 レジス

タの値と task_struct 構造体の先頭アドレスの組を VMM 内に保持する方式

(契機追加方式) 切り替え先プロセスを特定できるような箇所に VM exit を発生させる契機を追加する方法

(走査方式) は，プロセス切り替え時の CR3 レジスタへのアクセスによる VM exit を検知し，ゲスト OS のプロセスリストを走査する方式である．Linux において，すべての task_struct 構造体は，双方向環状のプロセスリストで連結されている．プロセスリストの先頭は，カーネルスレッドである init_task であり，カーネルの終了まで開放されない．また，プロセスとページディレクトリの対応は，task_struct の mm メンバを参照することで確認できる．このため，init_task から順にプロセスリストを走査し，VM exit 発生時に CR3 レジスタに書き込まれようとしている値をゲスト OS のプロセスリストから探索することで，切り替え先プロセスを特定できる．

(リスト管理方式) は，プロセス切り替え時に切り替え元プロセスが不可視化対象プロセスか否かを常に確認しておき，不可視化対象プロセスであった場合には，CR3 レジスタの値を VMM 内に確保した領域に保存する．また，CR3 レジスタの値だけでなく，task_struct 構造体の先頭アドレスも保存する．これにより，切り替え先プロセスの CR3 の値を取得した後に，VMM 内に保存されているものと比較することで，切り替え先プロセスが不可視化対象プロセスか否かを識別できる．また，task_struct 構造体の先頭アドレスと組にして保存しておくことで，切り替え先プロセスのプロセス名や PID をゲスト OS が利用しているメモリから即座に取得できる．不可視化対象プロセスが複数存在する場合には，これらの要素をメンバに持つ構造体のリストを作成する．これにより，CR3 へ書き込まれようとしている値を参照することで切り替え先プロセスを特定できる．

(契機追加方式) は，切り替え先プロセスを特定できるような契機を新たに追加する．例えば，プロセス切り替えを行う際に切り替え元プロセスと切り替え先プロセスの両方の task_struct 構造体の先頭アドレスを引数にとる関数にブレークポイントを設定して VM exit を発生させる方法が考えられる．この場合は，引数の情報をもとに切り替え先プロセスを特定できる．ブレークポイントの設定には，ゲスト OS のメモリを書き換えて INT3 命令を埋め込む方法やハードウェアブレークポイントを設定する方法を利用できる．既存の VM exit を用いるのではなく，追加の VM exit を発生させるという点で上記の 2 方式と異なる．

取得方法の比較

それぞれの方式の利点と欠点を**表 1** に示す．(走査方式) は，VM exit 発生時にゲスト OS のメモリを解析するだけで実現できる．このため，(リスト管理方式) のようにゲスト OS 上のプロセスの状態を VMM 内で把握する必要や (契機追加方式) のようにゲスト OS にブレークポイントを追加する必要はない．しかし，プロセス切り替えによる VM exit が発生する度にゲスト OS のプロセスリストを走査する，このため，プロセスリストの長さを N とした場合，VMM は平均で $N/2$ 回プロセス関連情報を取得し，切り替え先プロセスか否かを判定する必要があり，性能低下が大きいと予測できる．これに比べ，(リスト管理方式) と (契機追加方式) は，十分に小さい性能低下で実現できる．(リスト管理方式) は，完全仮想化環境において必ず生じる VM exit のみを用いて切り替え先プロセスを特定できるため，追加の VM exit は発生せず，最も性能低下が小さい．(契機追加方式) は，切り替え先プロセスを特定するために追加で VM exit を発生させるため，(リスト管理方式) よりも性能低下が大きいと予測できる．(リスト管理方式) の欠点として VMM 内に重要プロセスのリストを作成するため，メモリ使用量が増加する問題がある．しかし，重要プロセスとして想定するプロセスは，ウィルス対策ソフトウェアやシステム管理ツールであることから，不可視化対象となるプロセスは少数である．このことから，増加するメモリ使用量は，VMM 内のリストの 1 エントリのサイズが数 10 バイトだとしても，100 バイト程度であると予測できる．代表的な VMM の一つである Xen のメモリ使用量は，Xen 4.2.0 において 182 メガバイトであることから，100 バイトの増加は十分小さい．以上のことから，切り替え先プロセスの特定には，(リスト管理方式) を用いる．

表 1 切り替え先プロセスの特定方法の比較

方式	利点	欠点
(走査方式)	ゲスト OS のメモリの解析のみで実現可能	ゲスト OS のプロセスリスト探索の所要時間大
(リスト管理方式)	VMM 内のリスト探索の所要時間小	VMM のメモリ使用量増加
(契機追加方式)	切り替え先プロセスを即座に特定可能であることから，性能低下小	設定できるブレークポイント数に制限有

4.4 マルチコアプロセッサへの対応

マルチコア CPU を用いる環境を想定した場合，VM に複数の物理 CPU コアが割り当てられているとすると，あるコアで重要プロセスの走行中に，別のコアで他のプロセスが同時に走行する状況が考えられる．この場合，重要プロセスのプロセス関連情報は復元されており，本来の情報を利用している．このため，別のコアのプロセスやカーネルから重要プロセスのプロセス関連情報を参照できてしまう．

この問題へ対処するために，重要プロセスの走行中は，

表 2 評価環境

VMM	Xen 4.2.0
OS（管理 VM）	Debian 7.3 (Linux 3.2.0 64-bit)
OS（保護対象 VM）	Debian 7.3 (Linux 3.2.0 64-bit)

通常プロセスが並列に走行することを禁止する．重要プロセスの走行中は，重要プロセスが走行している CPU コア以外のコアを停止する．これは，VM に提供している仮想 CPU を VMM により一時的に停止することで実現する．これにより，通常プロセスによる保護対象プロセスのプロセス関連情報の参照を制限する．

5. 評価

5.1 評価環境

評価環境を**表 2** に示す．CPU には，Intel Core i7-2600 を用いた．保護対象 VM は，Intel VT-x の機能を用いて Xen により完全仮想化した．

5.2 評価の目的と評価方法

評価の目的は，攻撃によるプロセスの停止を提案手法により回避できることの確認である．あるプログラムについてプロセス関連情報を置換した際に，プログラム名をもとにしたそのプロセスの停止を回避できるか否かを確認する．攻撃によるプロセスの停止として，4.1.3 項で述べた killall コマンドの利用を想定する．killall コマンドは，プロセスのプログラム名を proc ファイルシステムから取得する．このため，本評価では，提案手法により不可視化対象プロセスのプログラム名を置換した場合に，本来のプログラム名が ps コマンドにより表示されないことを確認する．ps コマンドは，`task_struct` 構造体の `comm` メンバを参照する．proc ファイルシステムにおいてプログラム名を取得する際にも，同様に `comm` メンバを参照するため，ps コマンドにより，proc ファイルシステムで取得できるプログラム名と同じ名前を取得できる．

5.3 評価結果

提案手法により syslogd のプログラム名を変更した．ここでは，実験として，syslogd の名前を apache2 に変更した．

保護対象 VM において ps コマンドの利用によりすべてのプロセスのプログラム名を確認した．その結果，プロセスの一覧に syslogd は表示されず，apache2 という文字列が表示された．このことから，プロセスのプログラム名を置換できていることを確認できた．また，これにより，プログラム名をもとにしたプロセスの停止を回避できることが確認できた．

6. おわりに

プロセスの不可視化による攻撃回避手法について，プロセス特定困難化のためのプロセス関連情報の置換手法の実現方式を述べた．プロセス関連情報の置換では，プロセス切り替えを VMM により検知し，重要プロセスの走行中以外は偽の情報に置換する方法を述べた．置換する情報としてプロセス制御ブロックを想定し，マルウェアが重要プロセスを停止する際にプログラム名を利用することから，プログラム名を置換することにより重要プロセスを停止する攻撃を回避できることを述べた．また，プロセス切り替え時に切り替え元プロセスと切り替え先プロセスを特定し，プロセス関連情報を置換する方法について述べた．さらに，評価により，プロセスのプログラム名の変更を確認した．

残された課題として，プロセス関連情報へのアクセス制御の実現，多種のマルウェアを用いた評価，および提案手法の導入による性能への影響の評価がある．

謝辞 本研究の一部は，科学研究費補助金特別研究員奨励費による．

参考文献

[1] F-Secure: Agobot, available from 〈`http://www.f-secure.com/v-descs/agobot.shtml`〉(accessed 2014-02-09).
[2] F-Secure: t0rnkit, available from 〈`http://www.f-secure.com/v-descs/torn.shtml`〉(accessed 2014-02-09).
[3] packetstorm: dica.tgz, available from 〈`http://packetstormsecurity.com/files/26243/dica.tgz.html`〉(accessed 2014-02-10).
[4] Jiang, X., Wang, X. and Xu, D.: Stealthy Malware Detection Through VMM-Based “Out-of-the-Box” Semantic View Reconstruction, Proc. 14th ACM Conference on Computer and Communications Security (CCS ’07), pp.128–138 (2007).
[5] Riley, R., Jiang, X. and Xu, D.: Guest-Transparent Prevention of Kernel Rootkits with VMM-Based Memory Shadowing, Lecture Notes in Computer Science, Vol.5230, pp.1–20 (2008).
[6] 飯田貴大，光来健一：VM Shadow: 既存 IDS をオフロードするための実行環境，情報処理学会研究報告，Vol.2011-OS-119, No.6, pp.1–8 (2011).
[7] 佐藤将也，山内利宏：プロセス関連情報の不可視化によりプロセスの識別を困難にする攻撃回避手法，コンピュータセキュリティシンポジウム 2013 (CSS2013) 論文集，pp.1042–1049 (2013).
[8] 川古谷裕平，岩村 誠，伊藤光恭：ステルスデバッガを利用したマルウェア解析手法の提案，マルウェア対策人材育成ワークショップ 2008 (MWS2008), (2008).