プロセス情報不可視化のための仮想計算機モニタよるメモリアクセス制御

佐藤 将也　　山内 利宏　　谷口 秀夫

岡山大学 大学院自然科学研究科
700-8530 岡山県岡山市北区津島中 3-1-1
{sato, yamauchi, tani}@cs.okayama-u.ac.jp

あらまし 重要なソフトウェアが攻撃の対象となり無力化されると，攻撃による被害が拡大する可能性がある．そこで，攻撃者から重要なソフトウェアの識別を困難にする攻撃回避手法を提案した．また，攻撃回避を実現するために，プロセス情報を置換することでプロセス情報を偽装する手法を提案した．しかし，この方法は，プロセス情報の継続的な監視により，攻撃対象のプロセスを特定される可能性がある．そこで，本稿では，仮想計算機モニタを用いたアクセス制御により，プロセス情報の参照を制限する手法を提案する．これにより，本来のプロセス情報を攻撃者から参照を困難にし，プロセスの特定をより困難にする．

Memory Access Control using Virtual Machine Monitor for Process Information Hiding

Masaya Sato　　Toshihiro Yamauchi　　Hideo Taniguchi

Graduate School of Natural Science and Technology, Okayama University
3-1-1 Tsushima-naka, Kita-ku, Okayama 700-8530, JAPAN

Abstract Attacks for essential software may enlarge damages. To prevent this type of attack and to mitigate damages, we proposed an attack avoiding method that complicates process identification from attackers. To complicate process identification, we employed a replacement method for process information. However, this method have a problem that enable adversaries to identify an attack target process by monitoring process information continuously. To address this problems, this paper proposes a memory access control mechanism using virtual machine monitor. This mechanism hides original process information and provides much more difficulty for process identification.

1　はじめに

計算機への攻撃の防止は重要な研究課題となっており，被害を防ぐために，攻撃防止のためのソフトウェアが研究開発されている．また，プログラムの動作を記録し，解析するプログラムが開発されている．以降，攻撃による被害の防止や解析を行うプログラム，およびシステムの管理において重要なプログラムを重要サービスと呼ぶ．重要サービスは，攻撃者にと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．攻撃を行うソフトウェアのなかには，重要サービスを停止または無効化する機能を持つものがある．重要サービス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，重要サービスへの攻撃による被害の拡

大を防止することが重要である．

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 (Virtual Machine Monitor, VMM) を利用する方法 [1, 2] が提案されている．しかし，重要サービスを提供する既存ソフトウェアを改変なしには利用できない問題がある．また，プログラムを停止させる API 呼び出しを監視し，重要サービスの停止を防止する手法 [3] が提案されているものの，カーネルレベルでの攻撃により無効化される可能性がある．

そこで，文献 [4, 5] において，これらの問題に対処するために，プロセス特定困難化のためのプロセス情報の置換による攻撃回避手法を提案した．本手法は，重要サービスを提供するプロセス（以降，重要プロセス）を攻撃者から隠ぺいするために，重要プロセスの情報を偽の情報に置換し，攻撃者による重要プロセスの特定を困難化することで攻撃を回避する．重要プロセスの情報の置換は，VMM により行う．これにより，重要プロセスの情報を置換する機構自体の安全性を向上している．しかし，重要プロセスの情報を置換するだけでは，カーネル空間におけるプロセスの情報の継続的な監視により，重要プロセスを特定される可能性がある．

本稿では，この問題に対処するために，プロセス情報へのアクセス制御による攻撃回避手法について述べる．提案手法は，重要プロセスの情報への参照を制限することで，カーネル空間におけるプロセス情報の継続的な監視による重要プロセスの特定を困難にする．重要プロセスの情報へのアクセス制御は，VMM により実現する．これにより，攻撃者がカーネルモードで動作する攻撃コードを実行可能な場合においても，重要プロセスの情報の参照を制限し，重要プロセスの特定を困難にする．

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 がある．Agobot は，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ている場合は，そのプロセスを停止する機能を持つ．

ログ収集プログラムを停止するマルウェアとして，t0rnkit や dica がある．これらは，自身の隠ぺいを行うツール群を含み，自身や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，ログ収集プログラムである syslog デーモンを停止することで，インストール過程におけるログ生成を防止する．

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，攻撃の検知や対処が遅れ，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によるシステムへの被害を抑制することが求められる．

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 [1] がある．VMwatcher は，オペレーティングシステム（以降，OS）よりも攻撃が困難な VMM により IDS を実現することで，IDS への攻撃を困難にする．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 [2] 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に VMM を用いて対処している．

攻撃者からセキュリティソフトウェアの停止を防止する手法として ANSS[3] が提案されている．ANSS は，プロセスを停止させるシステムコール呼び出しを検知し，セキュリティソフトウェアの停止を防止する．ANSS は，Windows においてカーネルモードで動作するドライバとして実現されている．

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．

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，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 プロセス情報の不可視化による攻撃回避手法

3.1 目的

2.3 節の問題に対処するために，本研究は，以下を目的とする．

(1) 重要サービスへの攻撃の回避
(2) 既存ソフトウェアの改変なしの利用

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，既存の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，既存ソフトウェアの改変なしの利用を目的とする．

3.2 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

プロセスに関する情報（以降，プロセス情報）の不可視化を実現する手法として，プロセス情報の置換手法を提案した．図 1 にプロセス情報の置換手法を示す．プロセス情報の置換では，重要プロセスの特定に利用されるプロセス情報を偽の情報に置換することで，プロセス情報をもとにした重要プロセスの特定を困難にし，重要プロセスへの攻撃を回避する．重要プロセス以外のプロセスを通常プロセスとした場合，図

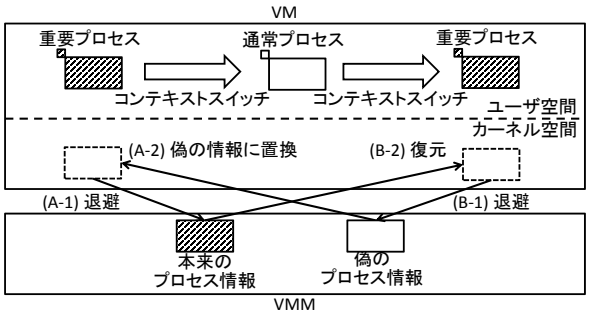

図 1 コンテキストスイッチ発生時におけるプロセス情報の置換

1 に示すとおり，提案手法は，コンテキストスイッチを契機とし，VMM によりプロセス情報を置換する．提案手法は，重要プロセスの走行中は本来のプロセス情報を利用させ，重要プロセスが走行中でない場合は，重要プロセスのプロセス情報を偽の情報に置換する．このため，重要プロセスの走行中は本来のプロセス情報を参照可能であり，プロセスの動作を妨げない．

3.3 プロセス情報の置換手法の問題

攻撃者は，カーネル空間でコードを実行可能な場合，プロセス情報の継続的な監視によりプロセス情報の置換を検出できる．これは，プロセス情報の置換手法は重要サービスを動作させるために，重要サービスの走行中は本来のプロセス情報を復元するためである．このため，プロセス情報の置換による重要サービスの不可視化は，プロセス情報の継続的な監視により，攻撃者から重要サービスを特定される可能性がある．そこで，本稿では，プロセス情報へのアクセス制御により，継続的な監視による重要サービスの特定を困難にする手法を提案する．

4 プロセス情報へのアクセス制御

4.1 基本方式

提案方式は，重要プロセスのプロセス情報へのアクセスを監視し，アクセス元に応じて本来の情報を返却するか偽の情報を返却するかを制御する．提案手法は，VMM を用いて実現する．これは，カーネル空間でコードを実行する攻撃への対処である．VMM はその性質からカーネルよりも攻撃を受けにくく，また，仮想計算機（Virtual Machine，以降，VM）のメモリを管

理していることから，メモリへのアクセスを制御するのに適している．また，本研究の目的は既存ソフトウェアの改変なしの利用であることから，完全仮想化を用いる．完全仮想化環境では，VM 上で動作する OS は，仮想化に対応したカーネルを用いる必要はない．このため，既存のソフトウェアを改変なしに利用できる可能性が高い．

4.2 プロセスの識別に利用される情報

本研究では，VM 上で動作する OS（以降，ゲスト OS）として Linux を想定した場合，プロセス制御ブロック，カーネルスタック，ハードウェアコンテキスト，ページテーブル，および，プロセスの利用するメモリ領域をプロセス情報とする．これらすべての情報を不可視化することでプロセス情報をもとにしたプロセスの特定を困難にできる．しかし，ハードウェアコンテキスト，ページテーブル，およびプロセスの利用するメモリ領域は，情報の参照や参照した情報からプロセスを特定することが困難である．そこで，本研究では，主に，プロセス制御ブロックとカーネルスタックをプロセス情報として扱う．

これらの情報には，プロセスの利用するソケットやファイルディスクリプタを含む．ネットワーク通信やファイルアクセスを監視することで，攻撃者は，プロセスの処理内容を容易に推測できる．しかし，ネットワーク通信やファイルアクセスに関する情報をプロセスと対応付けるには，プロセス制御ブロックやカーネルスタックを参照する必要がある．このため，これらをプロセス情報として扱い，アクセスを制御する．偽のプロセス情報は，文献 [5] で述べた情報を用いる．

4.3 メモリアクセスの制御

4.3.1 メモリアクセスの検知

プロセス情報が配置されたページへのアクセスを制御する方法として，以下の 2 通りの手法が考えられる．

(1) プロセス情報を参照する関数のフック
(2) ページテーブルの属性の変更

プロセス情報を参照する関数のフックでは，カーネル内において，プロセス情報にアクセスするための関数やマクロの処理を改変し，呼び出し元のコードに応じて処理を変更する．呼び出し元のコードは，カーネルスタックに積まれた戻りアドレスを参照することで取得できる．例えば，Linux においては，全プロセスを走査する for_each_process マクロや，構造体の先頭アドレスを算出する container_of マクロを改変することで，プロセス情報へのアクセスのうち一部を検知できる．

ページテーブルの属性の指定は，ページテーブルエントリの属性を変更し，特定のページの読み込みを制限する．重要プロセスの配置されたページの読み込みを制限した場合，そのページの読み込みにより例外が発生するため，この例外を検知し，ページの内容の読み込み可否を判断する．ただし，プロセス情報のサイズがページサイズと異なる，またはページ境界で整列されていない場合は，読み込みを禁止したページには，重要プロセスのプロセス情報だけでなく，他のプロセス情報も配置されている可能性がある．この場合，重要プロセスのプロセス情報以外を読み込む場合でも例外が発生するため，不要な例外が発生し，必要以上に性能が低下する可能性がある．

プロセス情報を参照する関数のフックは，実現が容易であり，かつアクセス制御により生じる性能低下を抑えられる．しかし，プロセス情報が配置されたメモリ領域のアドレスが攻撃者にとって既知の場合は，直接そのアドレスを参照することで，プロセス情報を攻撃者に参照される可能性がある．一方，ページテーブルの属性の変更では，読み込みを禁止したページへのアクセスごとに必ず例外が発生する．このため，プロセス情報の読み込みをもれなく検知できる．このことから，本稿における提案手法では，ページテーブルの属性の変更により，重要プロセスのプロセス情報が配置されたメモリの読み込みを制御する．

4.3.2 プロセス情報の読み込みの制御

重要プロセスのプロセス情報へのアクセスを検知した後に，アクセスを許可したコードからの読み込みに対しては，図 2 に示すとおり，本来のプロセス情報を返し，アクセスを許可して

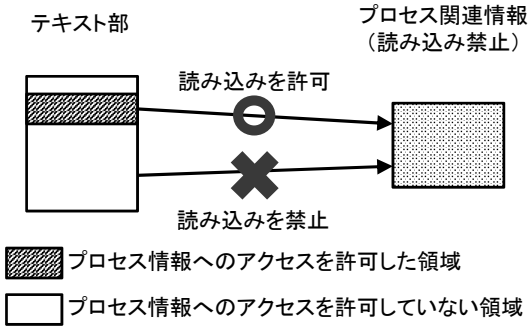

図 2 プロセス情報の読み込みの制御

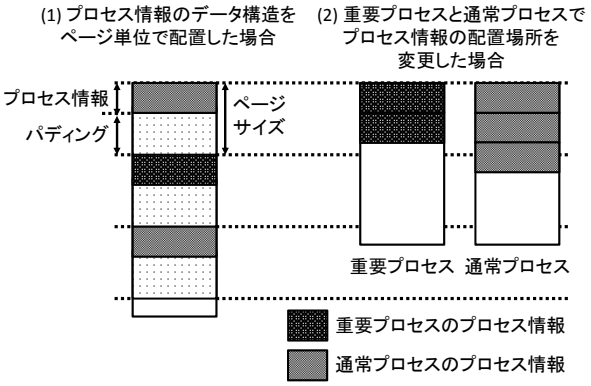

図 3 プロセス情報の配置方法

いないコードからの読み込みに対しては，偽のプロセス情報を返す．これにより，本来のプロセス情報を攻撃者から隠ぺいする．アクセス元のコードがアクセスを許可したコードか否かの判定には，カーネルスタックを用いる．カーネルスタックには，カーネル内の関数の戻りアドレスが格納されている．このため，重要プロセスのプロセス情報の読み込みを検知した際に，カーネルスタックからすべての戻りアドレスを取得し，取得したすべてのアドレスがアクセスを許可した領域に含まれていた場合には，本来のプロセス情報の読み込みをエミュレートする．そうでない場合には，偽のプロセス情報の読み込みをエミュレートする．

4.4 プロセス情報のメモリ配置の変更

ページ単位でメモリアクセスを制御する場合，プロセス情報がメモリ上にどのように配置されているかが重要となる．ページ単位でアクセス制御をする場合，不要な性能低下を抑制するためには，重要プロセスのプロセス情報が配置されるページにその他のデータが配置されないことが重要である．また，このためには，重要プロセスのプロセス情報をページ境界に整列して配置する必要がある．

重要プロセスのプロセス情報が配置されるページに他のデータを配置しないようにするには，図 3 に示す 2 通りの方法が考えられる．

(1) プロセス情報のデータ構造をページ単位で配置
(2) 重要プロセスと通常プロセスでプロセス情報の配置場所を変更

プロセス情報のデータ構造をページ単位で定義し，ページサイズの N 倍になるようにパディングすることで，プロセス情報がページ単位で確保できる．また，重要プロセスと通常プロセスの確保する領域を変更し，重要プロセスのプロセス情報を特定の領域に配置することで，重要プロセスのプロセス情報とその他のデータが同じページに配置されることを防止できる．重要プロセスのプロセス情報とその他のデータが同じページに配置されることを防止するには，重要プロセスのプロセス情報をページ境界で整列するように配置する必要がある．

上記 2 通りの方法について，プロセス情報のデータ構造をページ単位で配置する方法は，プロセス情報とページが 1 対 1 対応となるため，制御が容易になる．すなわち，特定のページにおいて例外が発生した場合にどのプロセスのプロセス情報の読み込みによる例外なのかの識別が容易になる．一方で，データ構造がページ単位になるようにメモリを確保するため，本来は不要な領域まで確保することになり，メモリの利用効率が低下する．重要プロセスと通常プロセスでプロセス情報の配置場所を変更する方法は，メモリの利用効率は高い．しかし，プロセス情報を配置する領域を重要プロセスの領域か通常プロセスの領域かをプロセス情報を確保する際に決める必要があるため，重要プロセスか通常プロセスかをプロセス情報の確保時に攻撃者から識別される可能性がある．また，プロセスの走行中に重要プロセスか通常プロセスかの切り替えはできない．このため，プロセスを走行中に重要プロセスとして指定することはでき

ず，プロセスの起動時に重要プロセスを指定する必要がある．ページ単位でプロセス情報が配置されている場合には，この問題は発生しない．以上より，制御の容易さと走行中プロセスを重要プロセスとして指定可能であるという利便性から，プロセス情報をページ単位で配置する手法を用いる．

プロセス情報をページ単位で配置するには，カーネルを改変する必要がある．プロセス情報としてプロセス制御ブロックを用いる場合を考えると，Linux においてプロセス制御ブロックは，kmem_cache_create 関数により確保された領域に配置される．そこで，この領域をはじめに確保する際に，ページ単位で整列するように領域を確保する．また，プロセス制御ブロックのサイズをページサイズにするために，本来のプロセス制御ブロックの後ろをパディングするよう，プロセス制御ブロックのデータ定義を変更する．これらにより，ページ単位でプロセス情報を確保し，かつページ境界にプロセス情報が配置されるようになる．

4.5 ページ単位のアクセス権限の設定

ページ単位でアクセス権限を設定するために，提案手法では，Extended Page Table （以降，EPT）を用いる．EPT を用いることで，ページ単位で読み込み，書き込み，および実行の権限を制御できる．そこで，ゲスト OS のプロセス情報が配置されたメモリの仮想アドレスをもとに，対応する EPT のエントリを探索し，読み込みビットを無効にする．これにより，ゲスト OS において当該ページの読み込みが発生した際に，処理が VMM に遷移する．

EPT のエントリの操作には，LibVMI[6] を用いる．LibVMI は，Virtual Machine Introspection のためのライブラリであり，管理 VM 上の応用プログラムから他の VM の情報取得やレジスタの操作が可能である．提案手法は，LibVMI を用い，VM 上の重要プロセスのプロセス情報に対応する EPT エントリの読み込みビットを無効にし，プロセス情報の読み込みを制限する．

4.6 重要プロセスの指定

4.5 節と同様，LibVMI を用いて重要プロセスを指定する．具体的には，管理 VM 上の応用プログラムから，VM 上のプロセス一覧を取得し，一覧の中から重要プロセスを指定する．

5 おわりに

プロセス情報へのアクセス制御による攻撃回避手法について述べた．提案手法は，重要プロセスのプロセス情報が配置されているページへのアクセスを検知し，許可したコードからのみ読み込みを可能にし，許可していないコードからのアクセスに対して偽のプロセス情報を返す．これにより，カーネル空間におけるプロセス情報の継続的な監視に対しても，重要プロセスの識別を困難にし，攻撃対象の特定を困難にすることで，攻撃を回避する．残された課題として，提案手法の実現と評価がある．

参考文献

[1] 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*, pp. 128–138 (2007).

[2] Riley, R., Jiang, X. and Xu, D.: Guest-Transparent Prevention of Kernel Rootkits with VMM-Based Memory Shadowing, *Recent Advances in Intrusion Detection*, Lecture Notes in Computer Science, Vol. 5230, pp. 1–20 (2008).

[3] Hsu, F.-H., Wu, M.-H., Tso, C.-K., Hsu, C.-H. and Chen, C.-W.: Antivirus Software Shield Against Antivirus Terminators, *IEEE Transactions on Information Forensics and Security*, Vol. 7, No. 5, pp. 1439–1447 (2012).

[4] 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).

[5] Sato, M. and Yamauchi, T.: Complicating Process Identification by Replacing Process Information for Attack Avoidance, *The 9th International Workshop on Security (IWSEC2014)*, Lecture Notes in Computer Science, Vol. 8389, pp. 33–47 (2014).

[6] LibVMI Project: LibVMI, `http://libvmi.com/` (2015).