情報処理学会論文誌 Vol.61 No.9 1507–1518 (Sep. 2020)

推薦論文

KVM上のゲストOSにおける権限の変更に着目した権限昇格攻撃防止手法

福本 淳文[1] 山内 利宏[1,a)]

受付日 2019年12月9日, 採録日 2020年6月1日

概要：権限昇格攻撃はシステムの改ざんや情報漏えいにつながる可能性がある．これに対処するために，我々はシステムコールによる権限の変更に着目した権限昇格攻撃防止手法（以降，先行研究の手法）を提案した．しかし，先行研究の手法はOS内で実現されており，導入するために，カーネルソースコードを変更する必要がある．また，先行研究の手法では，変更の検証のために保存したカーネル空間内の権限情報を，攻撃者に改ざんされる可能性がある．本論文では，これらの課題に対処するために，仮想マシンモニタであるKVMを用いて権限昇格攻撃を防止する手法を提案する．提案手法は，ゲストOS上のシステムコール発行をフックし，システムコール処理による権限の変更を検証する．提案手法の実現により，手法の導入にともなうカーネルソースコードの変更が不要となる．また，権限情報をホストOSのメモリ領域に保存することで，権限情報の改ざんが困難となる．本論文では，先行研究の手法の課題を示し，提案手法や評価の結果について述べる．

キーワード：権限昇格攻撃，仮想化，仮想マシンモニタ，KVM，セキュリティ

Privilege Escalation Attack Prevention Method Focusing on Privilege Changes in Guest OS on KVM

Akifumi Fukumoto[1] Toshihiro Yamauchi[1,a)]

Received: December 9, 2019, Accepted: June 1, 2020

Abstract: Privilege escalation attacks can lead to system tampering and information leakage. To address such attacks, we previously proposed a privilege escalation attack prevention method that focus on the modification of privileges by system calls. However, the said method needs to be implemented in the operating system (OS), and its application thus requires alteration of the kernel source code. Additionally, privilege data stored in the kernel space may be forged by attackers. To address these issues, we propose a new method in this paper for preventing privilege escalation attacks by employing KVM, (a virtual machine monitor). The new method hooks the system call invoked in the guest OS and verifies the modification of privileges through system call processing. Application of the new method does not require alteration of the kernel source code. Furthermore, forging of privilege data is deterred by storing privilege data in the memory of the host OS. In this paper, drawbacks of the previously proposed method are discussed, and the new proposed method and its evaluation results are described.

Keywords: Privilege Escalation Attack, Virtualization, Virtual Machine Monitor, KVM, Security

1. はじめに

オペレーティングシステム（以降，OS）の脆弱性を悪用した攻撃に権限昇格攻撃が存在する．この攻撃では，OSの脆弱性を悪用して，プロセスの権限をより高い権限へ昇格させる．権限昇格攻撃が成功すると，攻撃者は本来与えら

[1] 岡山大学大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University, Okayama 700–8530, Japan
[a)] yamauchi@cs.okayama-u.ac.jp

本論文の内容は2019年10月のコンピュータセキュリティシンポジウム2019(CSS2019)にて報告され，同プログラム委員長により情報処理学会論文誌ジャーナルへの掲載が推薦された論文である．

れる権限よりも高い権限でシステムを操作することが可能となる．特に，攻撃者の権限が管理者権限へと昇格した場合，システム全体のセキュリティが脅かされる可能性がある．このため，権限昇格攻撃に対処することは重要である．

Linux カーネルの脆弱性を悪用する権限昇格攻撃の対策として，我々はプロセスの権限の変更に着目して，システムコール処理の前後における権限の変更内容を監視することで権限昇格攻撃を防止する手法（以降，先行研究の手法）を提案した [1], [2]．先行研究の手法は，OS カーネル内で実現されており，システムコールサービスルーチンの前後において，プロセスの権限に関する情報（以降，権限情報）のうち，そのシステムコールが変更しない権限が変更された場合，権限昇格攻撃が実行されたと判断し，攻撃を防止する．先行研究の手法はシステムコール中に権限情報を改ざんする脆弱性を悪用した攻撃であれば，脆弱性の種類にかかわらず，権限昇格攻撃を防止できる．また，先行研究の手法をあらかじめシステムに導入することで，カーネルに未報告の脆弱性が存在した場合でも，権限昇格攻撃を防止できる．

しかし，先行研究の手法を導入するためには，Linux カーネルのソースコードを修正する必要がある．また，先行研究の手法は権限の変更を検証するために，システムコール処理前の権限情報をカーネルスタックに保存する．このため，攻撃者に保存したアドレスを知られると，システムコール処理中にカーネルスタックに保存された権限情報を改ざんすることで，システムコール処理前後における権限の変更の検証をバイパスできる可能性がある．さらに，先行研究の手法はシステムコール処理の前後にフックを仕掛けているため，システムコール処理中に発生する権限の改ざんしか検知防止できない．

本論文では，先行研究の手法に存在するこれらの課題に対処するために，仮想マシンモニタである KVM を用いて権限昇格攻撃を防止する手法（以降，提案手法）を提案する．提案手法は KVM 上のゲスト OS のシステムコール発行をフックし，システムコール処理による権限の変更を検証する．もし，発行したシステムコールでは変更し得ない権限情報が変更された場合，ゲスト OS 内で権限昇格攻撃が発生したと判断し，攻撃を防止する．

本研究の主な貢献は，以下のとおりである．

(1) 仮想マシンモニタである KVM を利用して，権限昇格攻撃を検知防止する手法を提案する．この手法は先行研究の手法とは異なり，適用対象のシステムのカーネルソースコードを変更する必要がない．このため，カーネルの再構築が困難な環境においても本手法を導入できる．
(2) ゲスト OS 上の攻撃者は保存した権限情報を改ざんし，検証をバイパスすることが困難である．これは，検証に用いる権限情報をホスト OS のメモリ領域に保存するため，アドレス空間の分離により，ゲスト OS 上のプロセスはホスト OS のメモリ領域に対して読み書きを実行できないためである．
(3) 提案手法はシステムコール処理中に発生しない権限の改ざんを検知防止できる．先行研究の手法はシステムコール処理の前後にフックを仕掛けていたため，検知できるのはシステムコール処理中に発生する改ざんのみであった．一方，提案手法はシステムコール処理の前にのみフックを仕掛けるため，システムコール処理中に発生しない権限の改ざんでも検知防止できる．

2. OS の脆弱性を悪用する権限昇格攻撃の防止手法

2.1 OS の脆弱性

OS は計算機が動作するための基盤となる役割を担うソフトウェアであり，高い信頼性が求められる．しかし，OS の脆弱性は，数多く報告されている．2018 年には Linux Kernel で 170 件，Mac OS X で 107 件，Windows 10 で 248 件の脆弱性が報告された [3]．また，OS カーネルのソースコードは膨大である．コード行数計測ツールである cloc [4] を使った我々の調査では，2018 年 12 月 23 日にリリースされた Linux 4.20 のカーネルソースコードは 2,400 万行を超えている．このように，OS カーネルのソースコード量は膨大であるため，脆弱性をすべて取り除くことは困難である．このため，OS には脆弱性が存在することを前提に，脆弱性を悪用した攻撃を防ぐための機構をシステムにあらかじめ適用する必要がある．

2.2 権限昇格攻撃

OS の脆弱性を悪用する攻撃の 1 つに権限昇格攻撃がある．権限昇格攻撃は攻撃者が本来与えられる権限よりも高い権限を奪取する攻撃である．権限昇格攻撃が成功すると，攻撃者はより高い権限でシステムを操作することが可能となる．もし，攻撃者によって管理者権限が奪取された場合，攻撃者はシステム上のすべてのファイルに対して，読み書きを実行できるようになるため，システムは深刻な被害を受ける可能性がある．また，権限昇格攻撃につながる OS の脆弱性は 2018 年に 14 件報告されており [3]，対策が必要であることが分かる．

2.3 権限の変更に着目した権限昇格攻撃防止手法

2.3.1 考え方

Linux カーネルにおける権限の管理方法には以下の特徴がある．

(1) プロセスの権限がメモリのカーネル空間に保存されていること
(2) カーネル空間のデータを操作するにはシステムコールを経由する必要があること

(3) 各システムコールの役割は細分化されていること

上記 3 つの特徴により，プロセスの権限が変更されるのは，プロセスの権限を変更する役割を持ったシステムコールが実行された際に限られることが分かる．しかし，Linux カーネルの脆弱性を悪用する権限昇格攻撃では，本来ならばプロセスの権限を変更し得ないシステムコールの処理中にプロセスの権限が変更される．たとえば，脆弱性 CVE-2013-1763 [5] を悪用した権限昇格攻撃の例では，ソケットのアドレスファミリを適切にチェックしていない不備がある．このため，`send` システムコールを使って，このようなソケットにデータを送信すると，プロセスの権限が変更される．しかし，`send` システムコールは本来権限情報を変更し得ないシステムコールである．

そこで，先行研究の手法はシステムコール処理の前後でプロセスの権限情報をチェックし，そのシステムコールが本来変更し得ない権限情報が変更されているかを監視する．もし，そのシステムコールが本来変更し得ない権限情報が変更されている場合は，権限昇格攻撃が実行されたと判断し，攻撃を防止する．

2.3.2 処理流れ

先行研究の手法の処理流れを以下で説明する．

(1) プロセスがユーザ空間からシステムコールを発行し，カーネル空間へ処理を移行する．
(2) システムコールサービスルーチン（システムコール本来の処理）への移行をフックし，先行研究の手法の処理へ移行する．
(3) 現時点の権限情報（システムコール処理前の権限情報）を保存する．
(4) システムコールサービスルーチンが実行される．
(5) システムコールサービスルーチンの実行の直後に処理をフックし，先行研究の手法の処理へ移行する．
(6) (3) で保存したシステムコール処理前の権限情報から現時点までの権限情報の変更（システムコール処理による権限情報の変更）をチェックする．
(7) システムコール処理による権限情報の変更が正常なものであったかを確認する．
    (A) 権限情報の変更が正常なものであった場合，権限昇格攻撃は行われていないと判断し，元々の処理流れに戻り，システムコール処理を終了する．
    (B) 権限情報の変更が不正なものであった場合，権限昇格攻撃が行われたと判断し，攻撃を防止するためにプロセスを終了させる．また，権限昇格攻撃を防止したことを示すログを出力する．

2.3.3 先行研究の手法の課題

本研究では，1 章で述べた先行研究の手法の以下の課題を解決する．

（課題 1） 導入にカーネルソースコードの変更が必要
（課題 2） 保存した権限情報の改ざんが可能
（課題 3） システムコール処理中に発生しない権限の改ざんを検知不可

先行研究の手法はシステムコール処理中に発生する権限の改ざんのみ検知できる．しかし，実際に Web 上から入手できる権限昇格攻撃のエクスプロイトコードを調査したところ，多くのエクスプロイトコードはシステムコール処理中ではないときに権限を改ざんするよう変更でき，先行研究の手法をバイパスできることが分かった．

3. 提案手法

3.1 考え方

提案手法はゲスト OS 上のシステムコール発行をフックし，システムコール処理による権限の変更を検証する．もし，発行したシステムコールが変更し得ない権限が変更された場合は権限昇格攻撃があったと判断し，攻撃を防止する．仮想マシンモニタ内で実現することによって，手法の導入にカーネルソースコードの変更が不要となり，先行研究の手法の（課題 1）に対処できる．また，ゲスト OS 上のプロセスはホスト OS のメモリ領域に対して読み書きを実行できない．このため，検証に用いる権限情報をホスト OS のメモリ領域に保存することで，権限情報の改ざんが困難となり，先行研究の手法の（課題 2）に対処できる．

3.2 基本方式

3.2.1 システムコールフックの方式

提案手法を設計するうえで，仕掛けるフックの数に関して，以下の 2 つの選択肢が存在する．

（方式 1） システムコールあたり 2 つのフック

この方式は先行研究の手法と同様，システムコール処理の前後にそれぞれ 1 つのフックを仕掛ける．この方式はシステムコール処理中に発生する権限の改ざんを検知できるが，システムコール処理中に発生しない権限の改ざんは検知できない．また，（方式 2）に比べて，フックの数が 1 つ多く，システムコール処理のオーバヘッドが大きい．

（方式 2） システムコールあたり 1 つのフック

この方式はシステムコール処理の前後いずれ一箇所にのみフックを仕掛ける．システムコール処理あたりに 1 回のフックしか発生しないため，システムコール処理のオーバヘッドが（方式 1）に比べて小さい．フックが 1 つであるため，前回のシステムコール発行から現在のシステムコール発行までに発生する権限の改ざんを検知できるようになり，システムコール処理中ではないときに発生する改ざんも検知できる．

この方式を選ぶ場合，システムコール処理の前にフックを仕掛けるかシステムコール処理の後にフックを仕掛けるかの選択肢が存在する．システムコール処理の前にフックを仕掛ける場合，システムコール処理中に権限が改ざんされると，その改ざんを検知できるのは次のシステムコー

ル発行時となる．つまり，攻撃者は権限を改ざんした後，ユーザ空間で処理を実行できる．しかし，攻撃者がシステムコールを発行し，OS に処理を依頼すると，システムコール処理前に提案手法によって攻撃が検知される．このため，攻撃者はシステムコールを発行せずに攻撃目的を達成する必要がある．

多くの場合，権限昇格攻撃で権限を改ざんした後，攻撃者は機密情報の窃取やデータの改ざんなどの攻撃目的を達成するためにシェルを起動する．しかし，シェルを起動するためには execve システムコールを発行する必要がある．このように，攻撃者が権限を改ざんした後にシステムコールを発行せずに攻撃目的を達成することは困難である．このため，権限の変更の検証をシステムコール処理の直後ではなく，次にシステムコールを発行したときに行っても問題はない．

システムコール処理の後にフックを仕掛ける場合，システムコール処理中に改ざんが発生しても，ユーザ空間に戻る前に改ざんを検知できる．このため，システムコール処理の前にフックを仕掛ける場合に比べ，改ざんをより早く検知できる．しかし，システムコール処理の後にフックを仕掛ける場合，システムコール番号の改ざんにより検知をバイパスされる可能性がある．提案手法は権限の変更を検証するために，発行されたシステムコールの種類を判別できる必要がある．システムコール処理前なら，システムコール番号が格納された rax レジスタの値を見ればよい．しかし，システムコール処理後では，rax レジスタがシステムコールサービスルーチンによって変更されている可能性があるため利用できない，この場合，メモリ上に保存されているシステムコール番号を取得する必要があるが，この値は攻撃者によって改ざんされる可能性があるため信用できない．

以上の理由から，提案手法はより多くの攻撃を検知でき，オーバヘッドが小さい（方式 2）を採用する．また，（方式 2）のなかでも，システムコール処理の前にフックを仕掛ける方式を採用する．

3.2.2 処理流れ

提案手法（方式 2）の全体像を**図 1** に示す．提案手法はホスト OS の KVM 内で実現され，ゲスト OS のシステムコール処理をフックする．システムコールサービスルーチンの実行前に，ゲスト OS の処理をフックし，提案手法の処理に移行する．提案手法の処理流れを**図 2** に示し，以下で説明する．

(1) ゲスト OS 上のプロセス（以降，プロセス A）がシステムコールを発行する．
(2) システムコールサービスルーチンへの移行をフックし，提案手法の処理へ移行する．
(3) プロセス A を識別できる値を取得し，この値を用いて，ホスト OS のメモリ領域に保存したプロセス A の権限情報を取得する．
(4) (3) で取得した権限情報と現時点の権限情報を比較し，権限情報の変更を検証する．
  (A) 前回のシステムコールでは変更し得ない権限情報が変更された場合，権限昇格攻撃が行われたと判断し，(5) に移行する．
  (B) 前回のシステムコールでは変更しうる権限情報が変更された場合，または権限情報が変更されなかった場合，権限昇格攻撃は行われていないと判断し，(6) に移行する．
(5) (3) で取得した権限情報を使って，プロセス A の権限を復元し，ゲスト OS に処理を移行する．
(6) 現時点の権限情報をホスト OS のメモリ領域に保存し，ゲスト OS に処理を移行する．

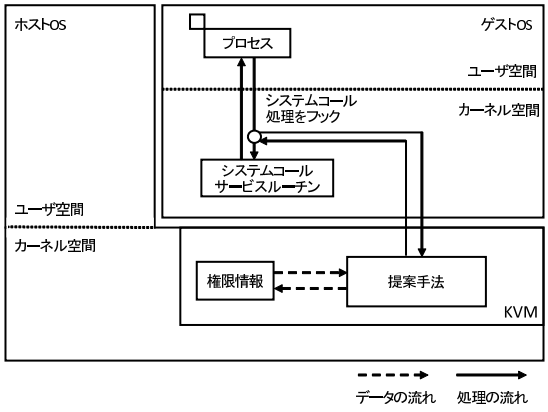

図 1 提案手法の全体像
Fig. 1 Overview of proposed method.

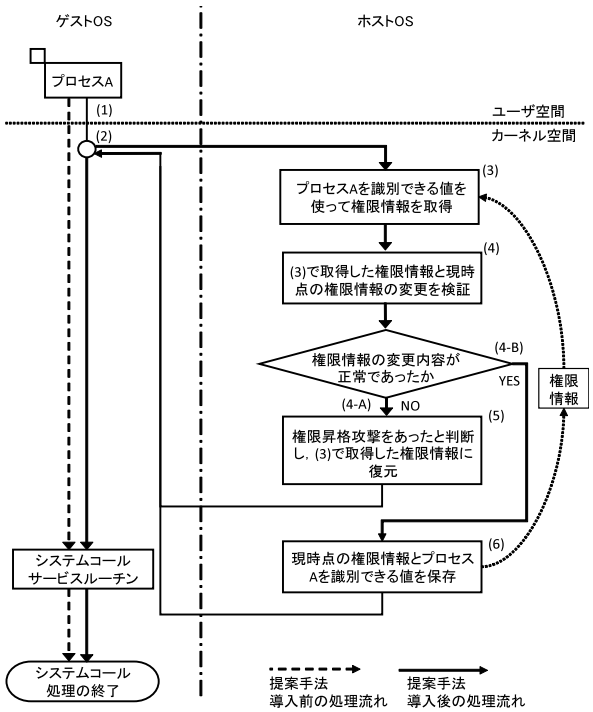

図 2 提案手法（方式 2）の処理流れ
Fig. 2 Process flow of proposed method (method 2).

表 1 提案手法における監視対象の権限情報
Table 1 Privileges monitored by proposed method.

権限情報	内容
uid	ユーザ ID
euid	実効ユーザ ID
fsuid	ファイルシステムユーザ ID
suid	保存ユーザ ID
gid	グループ ID
egid	実効グループ ID
fsgid	ファイルシステムグループ ID
sgid	保存グループ ID
cap_inheritable	継承ケーパビリティセット
cap_permitted	許可ケーパビリティセット
cap_effective	実効ケーパビリティセット
cap_ambient	周辺ケーパビリティセット
addr_limit	ユーザ空間とカーネル空間の境界アドレス

3.3 監視対象の権限情報と権限情報を変更しうるシステムコール

提案手法で監視する権限情報を**表 1** に示す．表 1 は先行研究の手法で監視する権限情報から cap_bset を削除し，cap_ambient を追加したものである．cap_bset は許可ケーパビリティセットの制限に利用されるため，攻撃者は cap_bset を改ざんしても，今保持している許可ケーパビリティセットに新しいケーパビリティを付与できない．このため，cap_bset は攻撃に利用されることはないと判断し，監視対象の権限情報から削除した．また，cap_ambient は Linux 4.3 で新たに追加されたケーパビリティセットである．表 1 のうち，uid 群（uid，euid，fsuid，suid）と gid 群（gid，egid，fsgid，sgid）はプロセスやディレクトリにアクセスするときのセキュリティチェックに用いられる．また，ケーパビリティ群（cap_inheritable，cap_permitted，cap_effective，cap_ambient）はプロセスが特定の操作を実行できるか否かを示したフラグの集合である．特定の操作には，たとえば，「chroot() を実行する」などがある．

addr_limit はプロセスの権限情報ではないが，提案手法ではこの値も監視対象としている．addr_limit にはユーザ空間とカーネル空間の境界アドレスが保存されているため，この値が不正に書き換えられると，カーネル空間のデータがユーザ空間から自由に書き換えられてしまう．このため，提案手法では addr_limit の値も監視する．

権限情報を変更しうるシステムコールを**表 2** に示す．表 2 は先行研究の手法が監視する権限情報を変更しうるシステムコールに，execveat システムコールを追加したものである．execveat システムコールは Linux 3.19 で新たに追加されたシステムコールであり，execve システムコールとは実行するプログラムのパスを指定する方法が異なるだけで，内部では同様に動作する．提案手法ではシステムコール処理による権限の変更において，表 2 で記されている権限情報以外の権限情報が変更された場合，権限昇格攻撃が実行されたと判断する．

表 2 提案手法で監視する権限情報を変更しうるシステムコール
Table 2 System calls that may change privileges.

システムコール	変更しうる権限情報
execve	uid, euid, fsuid, suid, gid, egid, fsgid, sgid, cap_inheritable, cap_permitted, cap_effective, cap_ambient, addr_limit
execveat	uid, euid, fsuid, suid, gid, egid, fsgid, sgid, cap_inheritable, cap_permitted, cap_effective, cap_ambient, addr_limit
setuid	uid, euid, fsuid, suid, cap_inheritable, cap_permitted, cap_effective, cap_ambient
setreuid	uid, euid, fsuid, suid, cap_inheritable, cap_permitted, cap_effective, cap_ambient
setresuid	uid, euid, fsuid, suid, cap_inheritable, cap_permitted, cap_effective, cap_ambient
setfsuid	fsuid, cap_inheritable, cap_permitted, cap_effective, cap_ambient
setgid	gid, egid, fsgid, sgid
setregid	gid, egid, fsgid, sgid
setresgid	gid, egid, fsgid, sgid
setfsgid	fsgid
capset	cap_inheritable, cap_permitted, cap_effective, cap_ambient
prctl	cap_inheritable, cap_permitted, cap_effective, cap_ambient
setns	cap_inheritable, cap_permitted, cap_effective, cap_ambient
unshare	cap_inheritable, cap_permitted, cap_effective, cap_ambient

4. 実現方式

4.1 実現における課題

以降では，KVM の x86-64 アーキテクチャの VM 上で動作する Linux を対象にした実現方式について述べる．提案手法を実現するために，以下の課題に対処する必要がある．

（**実現課題 1**） システムコールフックの実現

提案手法でシステムコール処理によるプロセスの権限情報の変更を検証するためには，システムコール処理前に，ホスト OS 側に処理を遷移させる必要がある．このため，提案手法はゲスト OS 上のシステムコール発行をフックできる必要がある．

（**実現課題 2**） セマンティックギャップへの対処

OS 内で取得できる情報と仮想マシンモニタから取得できる情報には，OS によって意味付けされているか否かの差がある．この差をセマンティックギャップ [6] と呼ぶ．セ

マンティックギャップが原因で，KVM からゲスト OS のメモリ上の情報を取得しても，取得した情報がゲスト OS 上でどのような意味を持っていたか KVM から解釈するのは難しい．提案手法を実現するためには，セマンティックギャップに対処し，KVM からゲスト OS 上で動作しているプロセスの権限情報を取得できる必要がある．

（**実現課題 3**） 保存した権限情報の削除

提案手法はシステムコールを発行してから，当該プロセスが次にシステムコールを発行するまでの間に起こる権限の変更を検証する．このため，提案手法はつねに，プロセスごとに 1 つ前にシステムコールを発行した時点での権限情報を保持する必要がある．しかし，権限情報を削除せずに保持し続けると，ホスト OS のメモリ領域を消費し続け，メモリリークを引き起こす．このため，提案手法はプロセス終了時にホスト OS のメモリ領域に保存した権限情報を削除する必要がある．

4.2 システムコールフックの実現

提案手法では，ゲスト OS におけるシステムコール処理をフックするために，文献 [7] で述べられているハードウェアデバッグレジスタを利用したシステムコールフック手法を採用する．ハードウェアデバッグレジスタにシステムコール処理を呼び出す前の命令のアドレスをセットすることで，その命令にハードウェアブレークポイントを設定できる．ハードウェアブレークポイントが設定されたアドレスに存在する命令が実行されるとデバッグ例外が発生する．このため，デバッグ例外を契機に VMexit を発生させると KVM 側でそれを捕捉し，ゲスト OS のシステムコール発行をフックできる．

システムコールフックにハードウェアデバッグレジスタを利用する場合，ハードウェアデバッグレジスタの値を攻撃者に改ざんされると提案手法が回避される可能性がある．このため，提案手法はハードウェアデバッグレジスタに対する書き込みがあった場合，KVM 側でそれを捕捉し，検証する．もし，設定したハードウェアブレークポイントを外すような書き込みだった場合，書き込みを無効化することで，提案手法が回避されることを防ぐ．

4.3 セマンティックギャップへの対処

4.3.1 対処の方針

セマンティックギャップに対処するための方法として，ゲスト OS に仮想マシンモニタと通信し，仮想マシンモニタが要求するデータを受け渡すモジュールを組み込む方法と仮想マシンモニタから得られる情報のみを使ってゲスト OS の情報を特定する方法がある．前者は，ゲスト OS の協力が得られるため，ゲスト OS の情報を容易に取得できる．しかし，そのようなモジュールを組み込むことで，モジュール自体が攻撃対象になる可能性がある．また，そのようなモジュールの導入にはゲスト OS の変更が必要であるため，2.3.3 項で述べた先行研究の手法の（課題 1）に対処できなくなる．これらの理由から，提案手法は KVM から取得できる情報のみを使って，ゲスト OS の情報を特定し，セマンティックギャップに対処する．

4.3.2 権限情報の取得

システムコール処理による権限の変更を検証するために，ゲスト OS で動作しているプロセスの権限情報を取得できる必要がある．プロセスの権限情報は `cred` 構造体に格納されており，プロセス制御ブロックである `task_struct` 構造体内の `cred` メンバから参照されている．したがって，プロセスの権限情報を取得するには，まず，`task_struct` 構造体の先頭アドレスを取得する必要がある．ゲスト OS で動作しているプロセスの `task_struct` 構造体の先頭アドレスは CPU 変数である `current_task` 変数に格納されている．Linux では MSR の `IA32_GS_BASE` レジスタに CPU 変数領域の先頭アドレスを格納している．提案手法は上記で述べた変数や構造体の各メンバを辿って，権限情報を取得する．

4.3.3 プロセスを識別できる情報の取得

ゲスト OS 上のプロセスがシステムコールを発行すると，提案手法は当該プロセスが前回システムコールを発行した際に保存した権限情報を取得する．取得した権限情報と現在の権限情報を比較することで，前回発行したシステムコールの処理による権限の変更を検証する．このため，提案手法は保存した権限情報がどのプロセスのものか判別できる必要がある．

この課題に対処するために，提案手法はプロセスの権限情報とともに，プロセスの PID をホスト OS のメモリ領域に保存する．PID はプロセスごとに異なる値を取るため，プロセスの判別に利用できる．

4.4 保存した権限情報の削除

Linux ではプロセスが終了する際に `exit` システムコールを発行する．`exit` システムコールを発行したプロセスは終了するため，それ以降システムコールを発行することはない．このため，保存した当該プロセスの権限情報を削除してもよい．提案手法はあるプロセスが `exit` システムコールを発行したときに，当該プロセスの権限情報をホスト OS のメモリ領域から削除することで，（実現課題 2）に対処した．

4.5 検出時の処理

権限昇格攻撃を検出した際の処理として，文献 [2] では以下の 3 つの処理をあげている．

（**処理 1**） 権限昇格攻撃を無効化

権限昇格攻撃を検知した際に，保存した権限情報を復元することで権限昇格攻撃を無効化する．

（処理 2） 実行中のプロセスを終了

権限昇格攻撃が成功した後に，攻撃者が目的を達成するために，新たな攻撃プログラムを実行できないよう，実行中のプロセスを終了する．

（処理 3） 解析のために実行中のプロセスを停止

攻撃の詳細を解析するために，実行中のプロセスを停止する．

（処理 2）を実現する場合，プロセスを安全に終了させる必要がある．文献 [2] では（処理 2）を実現する方法として，プロセスに SIGKILL シグナルを送信する方法があると述べている．OS 内でプロセスに SIGKILL シグナルを送信する場合，`kill` システムコールを発行すればよい．しかし，ホスト OS 側からゲスト OS のプロセスに SIGKILL シグナルを送信する場合，セマンティックギャップに対処する必要がある．また，攻撃対象となったプロセスを終了することでシステムの可用性が損なわれる可能性がある．（処理 3）を実現する場合，攻撃を解析する機構が存在することが前提である．しかし，提案手法は攻撃を解析するための手法ではないため，解析の機構が存在しない．

上記の理由から，提案手法は（処理 1）を実現する．（処理 1）を実現するために，提案手法は権限昇格攻撃を検知した際，ホスト OS 側に保存した権限情報をゲスト OS 内のプロセスの権限情報に復元する．

5. 評価

5.1 評価内容と評価環境

提案手法の適用によって生じるオーバヘッドと提案手法の有用性を明確にするために，以下の評価を行った．なお，評価では 3.2.1 項で述べた（方式 2）の利点を確かめるために，（方式 1）の手法を採用した場合の評価も合わせて行った．以降，（方式 1）を採用した手法を（方式 1）の手法と呼ぶ．

（評価 1） 攻撃検知防止実験

システムコール処理中に権限を改ざんする場合とシステムコール処理中でないときに権限を改ざんする場合のそれぞれにおいて，（方式 1）の手法および提案手法が権限の改ざんを検知防止できるかを検証し，提案手法の有用性を評価した．

（評価 2） システムコールのオーバヘッド測定

（方式 1）の手法および提案手法を導入することによって発生するシステムコールのオーバヘッドを測定し，比較評価した．

（評価 3） AP 性能の評価

（方式 1）の手法および提案手法を導入することによって発生する AP 性能への影響を測定し，比較評価した．

評価に用いる環境を**表 3** と**表 4** に示す．なお，ゲスト OS に割り当てる vCPU の数は 1 とした．

5.2 攻撃検知防止実験

提案手法の有用性を示すために，**表 5** に示す脆弱性を悪用した権限昇格攻撃を実施し，（方式 1）の手法および提案手法が攻撃を検知防止できるか否かを評価した．評価では以下に示す 2 つのタイプの攻撃を実行した．

（タイプ 1） システムコール処理中に権限を改ざん

あるプロセスがカーネルに存在する脆弱性を悪用して，システムコール処理中に自身の権限を改ざんする．

（タイプ 2） システムコール処理外に権限を改ざん

あるプロセスが子プロセスを作り，子プロセスがカーネルに存在する脆弱性を悪用して，システムコール処理中に親プロセスの権限を改ざんする．子プロセスが攻撃を実行している間，親プロセスはシステムコールを発行せず，ユーザ空間で処理を続ける．このため，親プロセスから見た場合，システムコール処理中ではないときに，権限を改ざんされることになる．

表 3 評価環境
Table 3 Environment for evaluation.

CPU		Intel(R) Core(TM) i5-6500 @ 3.20 GHz
OS	ゲスト	Ubuntu 14.04 LTS (Linux 3.13.0, 64 bit)
	ホスト	Ubuntu 18.04 LTS (Linux 4.15.18, 64 bit)
メモリ	ゲスト	4 GB
	ホスト	32 GB

表 4 クライアントの環境
Table 4 Environment of client.

CPU	Intel(R) Core(TM) i7-6700 @ 3.40 GHz
メモリ	8 GB
OS	Ubuntu 16.04 LTS (Linux 4.4.0-141-generic, 64 bit)

表 5 権限昇格攻撃の検知防止実験結果
Table 5 Results of the detection experiment of privilege escalation attacks.

CVE 番号	脆弱性の概要	攻撃タイプ	検知防止可能	
			（方式 1）の手法	提案手法
CVE-2016-0728	`keyctl()` における整数オーバフローならびに解放済みメモリの使用	タイプ 1	✓	✓
		タイプ 2		✓
CVE-2014-0038	`recvmmsg()` におけるパラメータのチェック不備によるメモリ破壊	タイプ 1	✓	✓
		タイプ 2		✓

表 6 システムコールのオーバヘッド（単位：μs）
Table 6 System call overhead (μs).

システムコール	手法導入前（T1）	手法導入後		オーバヘッド	
		（方式 1）の手法（T2）	提案手法（T3）	（方式 1）の手法（T2–T1）	提案手法（T3–T1）
`getpid`	0.045	16.659	8.468	16.614	8.423
`read`	0.084	16.723	8.475	16.639	8.391
`write`	0.112	16.765	8.529	16.653	8.417
`stat`	0.599	16.544	9.175	16.945	8.576
`fstat`	0.103	16.912	8.573	16.809	8.470
`open+close`	0.593	17.590	9.084	16.997	8.491

権限昇格攻撃には Web 上で入手できる 2 つのエクスプロイトコード [8], [9] を利用した．（タイプ 1）の攻撃では入手したエクスプロイトコードをそのまま利用し，（タイプ 2）の攻撃では入手したエクスプロイトコードを改変して利用した．

表 5 より，（方式 1）の手法は（タイプ 1）の攻撃のみ検知防止でき，提案手法は（タイプ 1）の攻撃と（タイプ 2）の攻撃をすべて検知防止できたことが分かる．（方式 1）の手法が（タイプ 2）の攻撃を検知防止できなかったのは，（方式 1）の手法は先行研究の手法と同様に，システムコール処理の前後にフックを仕掛けるためである．システムコール処理の前後にフックを仕掛ける場合，システムコール処理中に発生する改ざんしか検知できない．このため，システムコール処理中ではないときに権限を改ざんする（タイプ 2）の攻撃を検知防止できなかった．これに比べて，提案手法はシステムコール処理の前にのみフックを仕掛けるため，あるプロセスがシステムコールを発行してから，次にそのプロセスがシステムコールを発行するまでに発生する権限の改ざんを検知する．このため，（タイプ 2）の攻撃であっても検知防止できた．

（方式 1）の手法や提案手法は攻撃を検知した際，改ざんされた権限情報をログに出力する．このため，攻撃を検知できたか否かはログに出力があったか否かで判断した．CVE-2016-0728 を悪用した攻撃では，`uid` 群，`gid` 群，`cap_permitted`，および `cap_effective` が変化していることを検知した．CVE-2014-0038 を悪用した攻撃では，`uid` 群，`gid` 群，`cap_permitted`，および `cap_effective` が変化していることを検知した．また，利用したエクスプロイトコードでは攻撃が成功した場合，root 権限でシェルが起動する．このため，攻撃を防止できたか否かは root 権限でシェルが起動したか否かで判断した．

提案手法が検知防止できるのは，権限情報を改ざんする権限昇格攻撃のみである．たとえば，所有者が root であり，`setuid` ビットがセットされた実行ファイルに脆弱性があり，この脆弱性を悪用して，root 権限で動作するプログラムの実行を奪取する権限昇格攻撃に対して，提案手法は検知防止できない．

また，提案手法は仮想マシンモニタには脆弱性がなく，信頼できると仮定している．このため，仮想マシンモニタに脆弱性があり，ホスト OS 側のメモリ領域を改ざんできる場合，攻撃者はホスト OS 側のメモリ領域に保存されている権限情報を改ざんすることで，提案手法をバイパスできる．しかし，仮想マシンモニタの脆弱性を悪用した攻撃の多くは特定の環境下でしか動作しない場合や攻撃に root 権限が必要な場合がある [10]．このため，仮想マシンモニタの脆弱性を悪用した攻撃で提案手法をバイパスすることは難しい．

5.3 システムコールのオーバヘッド測定

（方式 1）の手法および提案手法の導入によるシステムコールのオーバヘッドを明らかにするために，マイクロベンチマークである LMbench 3.0 の `lat_syscall` を用いて，システムコールのオーバヘッドを測定した．測定結果を**表 6** に示す．なお，`open+close` の処理時間を測定するプログラムは `open` と `close` の 2 つのシステムコールにかかる処理時間を測定しているため，表 6 では元の測定値を 2 で割った値を記載している．

表 6 より，システムコール 1 回あたりに追加されるオーバヘッドは（方式 1）の手法と提案手法の両方において，ほぼ一定であることが分かる．（方式 1）の手法と提案手法の両方において，ほぼすべてのシステムコールに対して同様の処理を追加する．このため，システムコールのオーバヘッドがほぼ一定になることは妥当であると推察できる．

また，測定の結果，提案手法のオーバヘッドは（方式 1）の手法のオーバヘッドの 50.0%〜50.6%になった．このため，（方式 1）の手法よりも提案手法を導入した方が，システムコールのオーバヘッドを抑えられることが分かる．

（方式 1）の手法に比べ，提案手法のオーバヘッドが小さくなったのは，システムコールフックの回数が 2 回から 1 回に削減されたためである．これにより，VMexit の発生回数が減少し，VMexit の発生によるオーバヘッドと VMexit 発生後に実行される提案手法の処理によるオーバヘッドが削減された．

表 7 Apache の 1 リクエストあたりの処理時間（単位：ms）
Table 7 Processing time per request in Apache (ms).

ファイルサイズ	手法導入前（T1）	手法導入後		オーバヘッド	
		（方式 1）の手法（T2）	提案手法（T3）	（方式 1）の手法（T2–T1）	提案手法（T3–T1）
10 KB	1.072	1.481	1.271	0.409（38.15%）	0.199（18.56%）
100 KB	10.484	10.866	10.677	0.382（3.64%）	0.193（1.84%）

表 8 カーネルビルドの処理時間（単位：s）
Table 8 Processing time in building the kernel (s).

	手法導入前（T1）	手法導入後		オーバヘッド	
		（方式 1）の手法（T2）	提案手法（T3）	（方式 1）の手法（T2–T1）	提案手法（T3–T1）
user	72.02	73.26	73.06	1.24	1.04
sys	10.42	54.08	32.21	43.66	21.79
real	82.87	128.08	105.78	45.21（54.55%）	22.91（27.64%）

表 9 圧縮処理の処理時間（単位：s）
Table 9 Processing time in compression (s).

	手法導入前（T1）	手法導入後		オーバヘッド	
		（方式 1）の手法（T2）	提案手法（T3）	（方式 1）の手法（T2–T1）	提案手法（T3–T1）
user	45.79	46.15	45.97	0.36	0.19
sys	0.15	2.70	1.49	2.55	1.34
real	46.05	49.05	47.63	3.00（6.51%）	1.58（3.43%）

5.4 AP 性能の評価

5.4.1 Web サーバ

本評価では，ApacheBench 2.3 を用いて，手法導入前，（方式 1）の手法導入後および提案手法導入後の Web サーバの性能を測定した．サーバ側の環境は表 3 に示した環境であり，クライアントの環境は表 4 に示す環境である．なお，評価に用いた Web サーバは Apache 2.4.7 である．評価では，10 KB と 100 KB のファイルに対し，10,000 回アクセスした際の 1 リクエストあたりの処理時間を測定した．Apache が 1 リクエストを処理する際に発行するシステムコールの数をサーバ側の環境で測定した結果，ファイルサイズにかかわらず 22 回であった．なお，リクエストを送信する際の同時接続数は 1 である．

測定結果を**表 7** に示す．オーバヘッドは，ファイルサイズに関係なく，（方式 1）の手法で 400 μs，提案手法で約 200 μs であった．表 7 のオーバヘッドを 22 で割ってシステムコールあたりのオーバヘッドを算出したところ，ファイルサイズ 10 KB の場合，（方式 1）の手法と提案手法の場合で，それぞれ 18.59 μs と 9.05 μs であった．ファイルサイズ 100 KB の場合，（方式 1）の手法と提案手法の場合で，それぞれ 17.36 μs と 8.77 μs であった．これらは表 6 に示したシステムコールのオーバヘッドに近い値である．

5.4.2 カーネルビルド

本評価では，手法導入前，（方式 1）の手法導入後および提案手法導入後におけるカーネルビルドの処理時間を測定した．ビルド対象のカーネルは Linux 3.13 であり，コンフィグファイルは `make allnoconfig` コマンドで生成したものを利用した．なお，ビルド時のジョブ数は 1 であり，ビルド処理の時間は `time` コマンドを使って測定した．

カーネルビルドを 3 回行い，ビルドに要した時間の平均値を**表 8** に示す．ここで，表 8 における user と sys はそれぞれ，カーネルビルド処理がユーザ空間で動作した時間とカーネル空間で動作した時間である．また，real はカーネルビルド処理に要した全体の時間である．測定結果より，（方式 1）を導入した場合は 54.55%の性能低下，提案手法を導入した場合は 27.64%の性能低下が見られた．また，カーネルビルド時に発行したシステムコールの数を測定した結果 2,741,471 回であった．表 8 に示したシステム時間のオーバヘッドとシステムコール発行回数からシステムコールあたりのオーバヘッドを算出したところ，（方式 1）の手法を導入した場合は 15.92 μs，提案手法を導入した場合は 7.95 μs となった．これらは表 6 に示したシステムコールのオーバヘッドに近い値である．

5.4.3 圧縮処理

本評価では，手法導入前，（方式 1）の手法導入後および提案手法導入後における圧縮処理の処理時間を測定した．圧縮対象のファイルは Linux 3.13 のソースコードを tar ファイルにまとめたものであり，サイズは 525 MB である．なお，圧縮プログラムには bzip2 を利用し，処理時間は `time` コマンドを使って測定した．

圧縮処理を 3 回行い，圧縮に要した時間の平均値を**表 9** に示す．測定結果より，（方式 1）を導入した場合は 6.51%の性能低下，提案手法を導入した場合は 3.43%の性能低下が見られた．また，圧縮処理時に発行したシステムコールの

数を測定した結果 156,817 回であった．表 9 に示したシステム時間のオーバヘッドとシステムコール発行回数からシステムコールあたりのオーバヘッドを算出したところ，（方式 1）の手法を導入した場合は 16.26 μs，提案手法を導入した場合は 8.54 μs となった．これらは表 6 に示したシステムコールのオーバヘッドに近い値である．

5.4.4 考察

AP 性能の評価すべてにおいて，提案手法は（方式 1）の手法よりもオーバヘッドが小さい結果となった．これは，5.3 節で述べたように，提案手法は（方式 1）に比べて，システムコールフックの数が少ないためだと推察できる．また，AP の処理のオーバヘッドとシステムコール発行回数からシステムコールあたりのオーバヘッドを算出したところ，表 6 のオーバヘッドに近い値となった．このことから，AP 処理のオーバヘッドは，AP 処理におけるシステムコール発行回数に比例して大きくなるといえる．さらに，システムコールの発行回数が多いカーネルビルド処理では，手法導入後のオーバヘッドが大きく，システムコールの発行回数が少ない圧縮処理では，手法導入後のオーバヘッドが小さい結果となった．このことから，AP 処理におけるシステムコール発行回数が少ない方が，手法導入後のオーバヘッドが小さいといえる．

提案手法は仮想マシンモニタを利用しているため，OS 内で実現されている先行研究の手法に比べて，手法導入におけるオーバヘッドが大きい．先行研究の手法のように，OS 内で実現されている場合，手法の処理に遷移する際のコストが低い．しかし，提案手法のように仮想マシンモニタを利用する場合，手法の処理に遷移するためには VMexit を発生させる必要があり，これには，ゲスト OS の状態の保存やホスト OS の状態の復元がともなうためコストが高い．一方で，仮想マシンモニタを利用した場合，権限情報をホスト OS のメモリ領域に保存することで，権限情報の改ざんを困難にでき，システムのセキュリティを強化できる．このように，仮想マシンモニタを利用する方法と OS 内に手法を実現する方法には，安全性と処理性能のトレードオフが存在する．

6. 関連研究

仮想マシンモニタを用いて，権限昇格攻撃を検知または防止する手法として，文献 [11], [12], [13] がある．文献 [11] はカーネルから分離され，MMU を制御する実行ドメインを実装している．この実行ドメインは権限情報をカーネルによって書き込みが不可能な領域に再配置する．その結果，権限情報は改ざんから守られ，権限昇格攻撃を防止できる．しかし，文献 [11] の手法を導入するにはカーネルのソースコードを変更する必要がある．このため，文献 [11] の手法を導入できる環境は限られる．一方で，提案手法は導入のためにカーネルのソースコードを変更する必要がない．このため，すでにシステムが稼働しており，カーネルの再構築が困難な状況でも提案手法を導入できる．

文献 [12] は VM のイベントを監視し，イベントの発生を対応する監査モニタに通知する監視フレームワークである HyperTap を提案している．文献 [12] では HyperTap を用いて，権限昇格攻撃を検知する HT-Ninja と呼ばれる手法を実装している．HT-Ninja はプロセスのコンテキストスイッチや I/O 関連のシステムコールの実行を契機に，ゲスト OS 上の全プロセスのリストを走査し，プロセスの権限情報を検証する．管理者権限を持つプロセスの親プロセスが管理者権限ではないことを検知することで，権限昇格攻撃を検知できる．しかし，HT-Ninja は権限昇格攻撃を検知できるものの，防止できない．一方で，提案手法は，不正な権限情報の変更を検知した場合は，保存した権限情報を復元することで，権限昇格攻撃を防止する．

文献 [13] では，ゲスト OS のメモリ領域に `int3` 命令を埋め込み，フックを仕掛けることで，ゲスト OS の動作を監視するフレームワークである hprobe を提案している．文献 [13] は，hprobe の有用性を示すために，CVE-2008-0600 を悪用した権限昇格攻撃を検知するための手法を実装している．この手法は，脆弱性が存在する `vmsplice` システムコールの処理内にフックを仕掛け，システムコールの引数を検証することで攻撃を検知している．しかし，この検知手法は脆弱性に依存する．このため，新しい脆弱性を検知するためには，脆弱性がある箇所にフックを設定し直し，脆弱性に応じた検知処理を実装しなければいけない．また，この手法は攻撃を検知できるのみで，防止はできない．一方で，提案手法は脆弱性の種類によらず攻撃を検知できる．また，提案手法は攻撃を検知した場合，保存した権限情報を復元することで，権限昇格攻撃を防止する．

カーネルのデータが改ざんされることを防ぐ手法として，文献 [14] がある．文献 [14] の手法では，カーネル内のセキュリティチェックに関連するデータを自動で収集し，Data-Flow Integrity [15] を利用して，これらのデータの完全性を保証する．たとえば，文字列内のデータがリターンアドレスの領域に書き込まれないよう保証できる．提案手法と比較して，文献 [14] の手法は権限情報だけでなく，カーネル内のセキュリティチェックに関連するデータを保護対象とする．このため，提案手法に比べ，より多くのデータを改ざんから保護できる．一方で，文献 [14] 内において，この手法は Use-After-Free を悪用した攻撃に脆弱であると述べている．たとえば，あるプロセスの cred 構造体が不正に解放され，その後に，管理者権限のプロセスのために，その cred 構造体が利用された場合，元のプロセスは管理者権限で動作できるようになる．このような場合であっても，提案手法は，保存してある cred 構造体と改ざん後の cred 構造体を比較することで，権限昇格攻撃を検知防止できる．また，文献 [14] の手法を適用するためには，

カーネルのソースコードを変更する必要があるが，提案手法を適用する際はカーネルのソースコードを変更する必要がない．

仮想マシンモニタの動作を監視し，検証する手法として，文献 [16] がある．文献 [16] では，イベント駆動で仮想マシンモニタの動作を監視する ED-monitor を提案している．ED-monitor は仮想マシンモニタのメモリ領域の完全性，制御フローの完全性，および ED-monitor へのエントリポイントの一意性を保証でき，仮想マシンモニタが攻撃者によって侵害されることを防ぐ．ED-monitor は仮想マシンモニタによる特権命令の実行や MMU の設定などの操作に干渉し，その操作の妥当性を検証する．また，ED-monitor は自身が配置される仮想アドレスをランダム化することによって，検証処理がバイパスされることを防ぐ．提案手法は仮想マシンモニタには脆弱性がなく，信頼できると仮定している．このため，仮想マシンモニタに脆弱性があった場合，その脆弱性を悪用され，提案手法がバイパスされる可能性がある．そこで，文献 [16] の手法と提案手法を併用することにより，システムのセキュリティを強化できると考える．

7. おわりに

仮想マシンモニタである KVM を用いて，システムコール処理による権限の変更に着目した権限昇格攻撃を防止する手法を提案した．仮想マシンモニタ内で実現することにより，導入のためにカーネルソースコードを変更する必要がない．また，ゲスト OS 上の攻撃者がホスト OS に保存した権限情報を改ざんすることは困難になる．さらに，提案手法はシステムコール処理後の 1 回のフック処理を削減することで，オーバヘッドの削減を実現し，かつ先行研究の手法では検知できないシステムコール処理中に発生しない権限の改ざんを検知防止できる．

提案手法を Linux カーネルに存在する 2 つの脆弱性を悪用した権限昇格攻撃で評価した．この結果から，提案手法はシステムコール処理中だけでなく，システムコール処理外で発生する権限の改ざんも検知防止できることを示した．

LMBench を用いた評価から，システムコール 1 回あたりのオーバヘッドは，8.391 μs〜8.576 μs であることを示した．また，AP を用いた性能評価では，Web サーバの性能評価，カーネルビルドの性能評価，および圧縮処理の性能評価を行った．Web サーバの性能評価において，Apache の 1 リクエストあたりの処理時間を測定した結果，10 KB と 100 KB のファイルに対してアクセスした場合，提案手法導入後ではそれぞれ 18.56%と 1.84%の性能低下が見られた．また，カーネルビルドや圧縮処理の処理時間を測定した結果，提案手法導入後ではそれぞれ 27.64%と 3.43%の性能低下が見られた．これらの結果から，提案手法導入後において，オーバヘッドは，システムコール発行回数に比例することが分かる．

謝辞 本研究の一部は，JSPS 科研費 JP19H04109 の助成を受けたものです．

参考文献

[1] 赤尾洋平，山内利宏：システムコール処理による権限の変化に着目した権限昇格攻撃の防止手法，コンピュータセキュリティシンポジウム 2016 論文集，Vol.2016, No.2, pp.542–549 (2016).
[2] Yamauchi, T., Akao, Y., Yoshitani, R., et al.: Additional Kernel Observer to Prevent Privilege Escalation Attacks by Focusing on System Call Privilege Changes, *Proc. 2018 IEEE Conference on Dependable and Secure Computing* (*IEEE DSC 2018*), pp.172–179 (2018).
[3] CVE: Top 50 Products By Total Number Of Distinct Vulnerabilities in 2018, CVE Details, available from ⟨https://www.cvedACetails.com/top-50-products.php?year=2018⟩ (accessed 2019-01-10).
[4] AIDanial: cloc, GitHub, available from ⟨https://github.com/AlDanial/cloc⟩ (accessed 2019-01-10).
[5] CVE: CVE-2013-1763, available from ⟨https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-1763⟩ (accessed 2019-01-17).
[6] Chen, P.M. and Noble, B.D.: When Virtual Is Better Than Real, *Proc. 8th Workshop on Hot Topics in Operating Systems*, pp.133–138 (2001).
[7] Fujii, S., Sato, M., Yamauchi, T. and Taniguchi, H.: Evaluation and Design of Function for Tracing Diffusion of Classified Information for File Operations with KVM, *The Journal of Supercomputing*, Vol.72, No.5, pp.1841–1861 (2016).
[8] PerceptionPointTeam: cve_2016_0728 exploit, GitHub Gist, available from ⟨https://gist.github.com/PerceptionPointTeam/18b1e86d1c0f8531ff8f⟩ (accessed 2019-06-10).
[9] ExploitDatabase: Linux Kernel 3.4 < 3.13.2 (Ubuntu 13.04/13.10 x64) - 'CONFIG_X86_X32=y' Local Privilege Escalation (3), available from ⟨https://www.exploit-db.com/exploits/31347⟩ (accessed 2019-06-10).
[10] CROWDSTRIKE: VENOM Vulnerability, available from ⟨https://venom.crowdstrike.com/⟩ (accessed 2019-06-10).
[11] Chen, Q., Azab, A.M., Ganesh, G., et al.: PrivWatcher: Non-bypassable Monitoring and Protection of Process Credentials from Memory Corruption Attacks, *Proc. 2017 ACM on Asia Conference on Computer and Communications Security*, pp.167–178 (2017).
[12] Pham, C., Estrada, Z., Cao, P., et al.: Reliability and Security Monitoring of Virtual Machines Using Hardware Architectural Invariants, *Proc. International Conference on Dependable Systems and Networks*, pp.13–24 (2014).
[13] Estrada, Z., Pham, C., Deng, F., et al.: Dynamic VM Dependability Monitoring Using Hypervisor Probes, *Proc. 2015 11th European Dependable Computing Conference*, pp.61–72 (2015).
[14] Song, C., Lee, B., Lu, K., et al.: Enforcing Kernel Security Invariants with Data Flow Integrity, *23rd Annual Network and Distributed System Security Symposium*, pp.1–15 (2016).
[15] Castro, M., Coasta, M. and Harris, T.: Securing Software by Enforcing Data-Flow Integrity, *Proc. 7th Symposium on Operating Systems Design and Implementa-*

tion, pp.147–160, (2006).
[16] Deng, L., Liu, P., Xu, J., et al.: Dancing with Wolves: Towards Practical Event-Driven VMM Monitoring, *Proc. 13th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pp.83–96 (2017).

推薦文

従来手法よりもフック箇所を削減し，かつ検知方法を改良することで，攻撃検知能力の向上とオーバヘッドの低減を両立している．具体的には，従来手法では検知できない攻撃を検知でき，オーバヘッドを半減している．実装による評価もしっかりしており良い論文である．以上のことから，推薦論文に推薦する．

（コンピュータセキュリティシンポジウム 2019
プログラム委員長　國廣 昇）

福本 淳文　（正会員）

2018 年兵庫県立大学工学部電子情報電気工学科卒業．同年岡山大学大学院自然科学研究科博士前期課程入学．2020 年同大学院自然科学研究科博士前期課程修了．コンピュータセキュリティに興味を持つ．

山内 利宏　（正会員）

1998 年九州大学工学部情報工学科卒業．2000 年同大学大学院システム情報科学研究科修士課程修了．2002 年同大学院システム情報科学府博士後期課程修了．2001 年日本学術振興会特別研究員（DC2）．2002 年九州大学大学院システム情報科学研究院助手．2005 年岡山大学大学院自然科学研究科助教授．現在，同准教授．博士（工学）．オペレーティングシステム，コンピュータセキュリティに興味を持つ．2010 年度 JIP Outstanding Paper Award，2012 年度情報処理学会論文賞等受賞．電子情報通信学会，ACM，USENIX，IEEE 各会員．本会シニア会員．