Vol.2019-DPS-178 No.7
Vol.2019-CSEC-84 No.7
2019/3/4

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の実現

福本 淳文[1]　山内 利宏[1,a)]

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，赤尾ら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，従来手法）を提案した．しかし，従来手法はOS内で実現されており，OS内のセキュリティ機構を導入するには，カーネルのソースコードを変更する必要がある．また，従来手法では保存した権限情報が攻撃者によって改ざんされる可能性がある．本稿では，これらの課題に対処するため，仮想マシンモニタであるKVM内に従来手法と同様のセキュリティ機構を実現する手法を提案する．この手法の実現により，カーネルの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側のメモリに保存することによって，ゲストのプロセスによる保存された権限情報の改ざんが困難となる．権限の変更を検証するために，提案手法は，KVMから取得できる情報のみを用いて，ゲストOS上で動作するプロセスの権限情報を取得する．本稿では，従来手法の課題，提案手法の設計，実現方式および性能評価の結果を報告する．

1. はじめに

オペレーティングシステム（以降，OS）は計算機が動作する基盤としての役割を担っているため，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おり，OSを完全に信頼することはできない．また，OSのソースコード量は膨大であり，脆弱性を完全に取り除くことは困難である．

報告されたOSの脆弱性を取り除くためには，脆弱性がある部分を修正する修正パッチをOSカーネルに適用する必要がある．しかし，システムの運用形態や機器の特性により，パッチのダウンロードや適用が困難な場合がある．このような原因から，OSを信頼するためには，脆弱性を修正する修正パッチをOSカーネルに適用するだけでは不十分であり，OSカーネルに未修正の脆弱性が存在することを前提に，これらの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．

OSの脆弱性を悪用した攻撃の1つに権限昇格攻撃が存在する．この攻撃では，OSの脆弱性を悪用するコードを実行することによって，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昇格されると，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，赤尾ら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 [1], [2], [3]（以降，従来手法）を提案した．従来手法は，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．従来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，悪用される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，従来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，従来手法をシステムに導入するためには，Linuxカーネルのソースコードに対して，従来手法を適用するためのパッチをあらかじめ適用する必要がある．したがって，すでに稼働しているシステムで，カーネルを入れ替えられない状況やOSのソースコードが入手できない状況では，従来手法をシステムに導入することができない．また，従来手法はプロセスの権限情報の変更を検証するために必要な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，攻撃者はカーネルスタックに保存

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

© 2019 Information Processing Society of Japan

された権限情報を改ざんすることで，従来手法の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．

これらの課題に対処するために，本稿では仮想マシンモニタであるKVM内に従来手法と同様のセキュリティ機構を実現する方法を提案する．提案手法の実現により，カーネルのソースコードを変更する必要がなくなる．また，権限情報をホストOS側のメモリに保存するため，ゲストメモリとホストメモリの分離により，保存した権限情報の改ざんが困難となる．

本稿では，従来手法の課題について述べ，提案するKVM内に従来手法と同様のセキュリティ機構を実現する手法の設計，実現課題，実現方式，および評価結果にについて述べる．また，従来手法と提案手法の間に存在するトレードオフについても述べる．

2. OSの脆弱性を悪用する権限昇格攻撃

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，年々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 [4]．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[5] 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2400万行を超えている．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．

OSカーネルの脆弱性が発見された場合，脆弱性がある箇所を修正する修正パッチをカーネルに適用する必要がある．しかし，システムの運用形態によっては修正パッチの適用が困難な場合がある．たとえば，多くの場合，カーネルにパッチを適用するには，OSの再起動が必要である．このため，常時稼働し続ける必要があるシステムに対して，修正パッチをシステムに適用することは困難である．また，修正パッチを適用するためには，まず，パッチをダウンロードする必要がある．このため，組み込み機器に対して修正パッチを適用することは困難である．

上記の理由から，OSが未修正の脆弱性を持つことを前提に，あらかじめシステムに組み込むことで，OSの脆弱性を悪用した攻撃を防ぐことができる機構が必要である．

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，読み書きを実行することが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．上記の理由から，権限昇格攻撃は大きな脅威であり，対策を取る必要がある．

提案手法は権限昇格攻撃の中でも，OSカーネルの脆弱性を悪用し，システムコール処理中にプロセスの権限情報を改ざんする攻撃を検知，防止の対象とする．

3. 権限の変更に着目した権限昇格攻撃防止手法

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わかる．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 [6] 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，従来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．

3.2 基本方式

従来手法であるOS内で実現されているシステムコールによるプロセスの権限の変更に着目した権限昇格攻撃防止手法について，その処理流れを**図 1**に示し，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，従来手法の処理へ移行する．

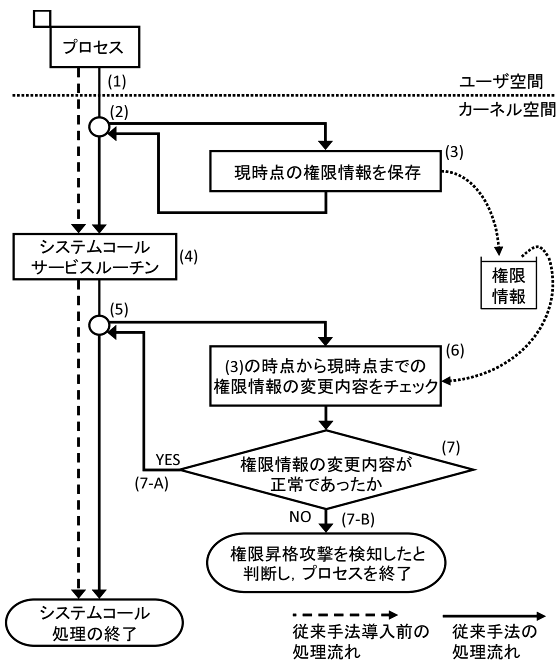

図 1 従来手法の処理流れ

(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，従来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．

3.3 従来手法における監視対象の権限情報と権限情報を変更し得るシステムコール

文献 [2] の手法で監視する権限情報を**表 1** に示す．表 1 のうち，uid 群 (uid, euid, fsuid, suid) はプロセスのユーザ識別子であり，gid 群 (gid, egid, fsgid, sgid) はプロセスのグループ識別子である．これらの値はプロセスがファイル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，「種々のネットワーク処理を許可する」や「chroot() の実行を許可する」などがある．これらの値はすべてカーネル空間に保存されている．

表 1 従来手法における監視対象の権限情報

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

addr_limit はプロセスの権限情報ではないが，従来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，従来手法ではプロセスの権限情報に加えて，addr_limit の値も監視する．

文献 [2] で調査した権限情報を変更し得るシステムコールを**表 2** に示す．従来手法ではシステムコール処理の前後において，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．たとえば，setuid システムコールでは，システムコール処理の前後において，変更し得る権限情報は uid 群とケーパビリティ群である．このため，もし，gid 群が変更された場合は権限昇格攻撃が実行されたと判断する．

3.4 OS 内の実現手法の課題

従来手法は OS 内で実現されており，OS と同じ権限で動作するため，以下の課題がある．

（課題 1） 導入にカーネルのソースコードの変更が必要
システムに従来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．このため，すでに稼働してるシステムで，カーネルの入れ替えができないシステムやカーネルのソースコードが入手できない状況では従来手法を適用することができない．

（課題 2） 保存した権限情報の改ざんが可能
従来手法では権限情報の変更を検証するために，システムコールの処理前に取得した権限情報をメモリ上にあるカーネルスタックに保存している．このため，権限情報が保存されているメモリ領域のアドレスが攻撃者に知られると，攻撃者は保存されている権限情報を書き換えることで従来

© 2019 Information Processing Society of Japan

表 2　従来手法で監視する権限情報を変更し得るシステムコール

システムコール	変更し得る権限情報
execve	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

手法の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することができる．

4. 提案手法の設計

4.1 考え方

本研究では，仮想マシンモニタである KVM 内に従来手法と同様のセキュリティ機構を実現することによって，3.4 節で述べた課題に対処する．仮想マシンモニタ内に権限昇格攻撃防止手法を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，（課題 1）に対処できる．また，ゲスト OS のプロセスはホストのメモリに対して読み書きを実行できないため，システムコール処理前のプロセスの権限情報をホスト OS 側のメモリに保存することによって，権限情報の改ざんが困難になり，（課題 2）に対処できる．なお，KVM が動作するためには，CPU が仮想化支援機能をサポートしている必要がある．提案手法は Intel VT が実装されている CPU 上での動作を想定している．

4.2 基本方式

提案手法の全体像を**図 2** 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．システムコールの入口におけるフックでは，フックした後にゲスト OS のプロセスの権限情報をホスト OS 側のメモリに保存する．システムコールの出口におけるフックでは，フックした後に保存した権限情報を取り出し，現在の権限情報と比較する．ゲスト OS のプロセスが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，権限昇格攻撃があったと判断する．

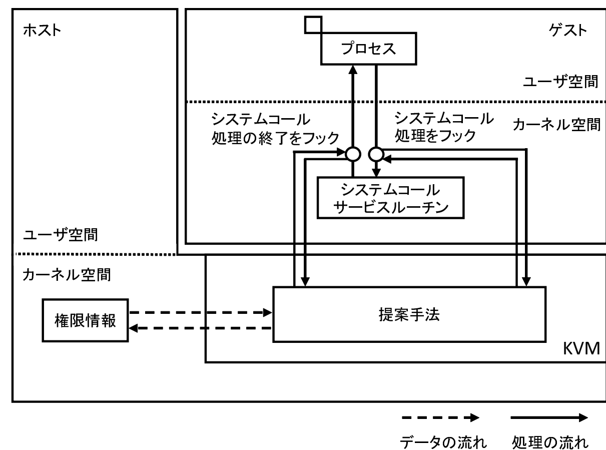

図 2　提案手法の全体像

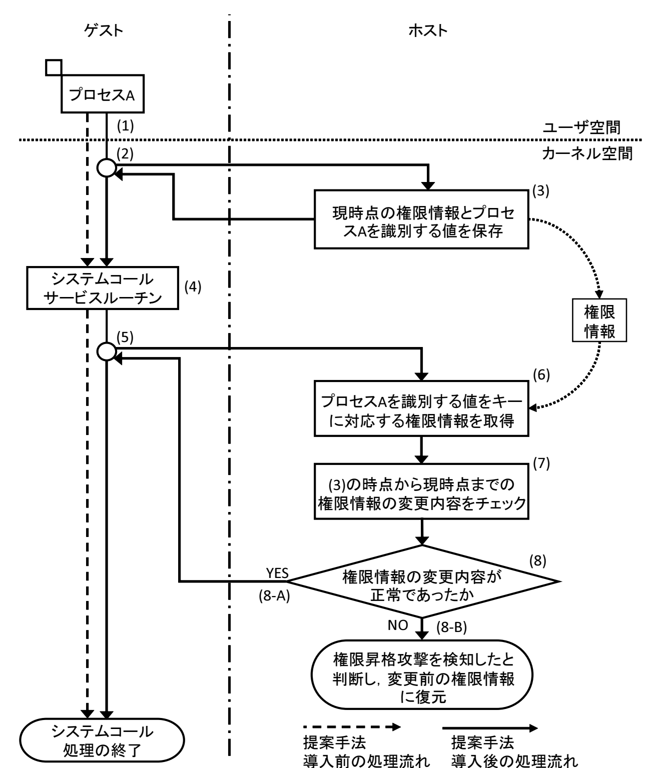

図 3　提案手法の処理流れ

図 2 の提案手法の部分の処理流れを**図 3** に示し，以下で説明する．

(1) ゲスト OS 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（システムコール本来の処理）への移行をフックし，提案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）とプロセス A を識別できる値を保存する．

表 3 新たに追加した監視する権限情報を変更し得るシステムコール

システムコール	変更し得る権限情報
execveat	uid, euid, fsuid, suid, gid, egid, fsgid, sgid, cap_inheritable, cap_permitted, cap_effective, cap_ambient, addr_limit

(4) ゲスト OS へ処理を移行し，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直後に処理をフックし，提案手法の処理へ移行する．
(6) プロセス A を識別できる値を取得し，この値をキーにして，対応する権限情報を取得する．
(7) (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．
(8) 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，ゲスト OS に処理を移行する．
  (B) 権限情報の変更が正常なものでなかった場合，権限昇格攻撃が行われたと判断し，プロセス A の権限情報を (3) の時点の権限情報に復元する．

4.3 提案手法における監視対象の権限情報と権限情報を変更し得るシステムコール

提案手法で監視対象となる権限情報は表 1 と同じである．本研究で新たに追加した監視する権限情報を変更し得るシステムコールを**表 3** に示す．提案手法では権限情報を変更し得るシステムコールとして，execveat システムコールを追加した．execveat システムコールと execve システムコール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．このため，execveat システムコールが変更し得る権限情報は execve システムコールと同じである．

4.4 トレードオフ

提案手法を導入することで従来手法が持つセキュリティ上の問題に対処することができる．一方で，提案手法は従来手法に比べて，システムの処理性能に与える影響が大きいことが推察できる．提案手法は仮想マシンモニタ内で実現されているため，提案手法の機構を実行するためには，ゲスト OS が動作する VMX non-root モードからホスト OS が動作する VMX root モードに移行する必要がある．この移行では，ゲスト OS の状態を保存する処理などが実行される．このため，OS 内で実現されている従来手法の機構を実行するより，仮想マシンモニタ内で実現されている提案手法の機構を実行する方がより多くの時間を要することが推察できる．このように，処理性能とセキュリティの間にはトレードオフが存在する．

5. 実現方式

5.1 実現における課題

本研究では，ゲスト OS とホスト OS がともに Linux 4.15.18 である環境で提案手法を実現する．提案手法を実現するために，以下の実現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
ゲスト OS のシステムコール処理前後におけるプロセスの権限情報の変更を検証するためには，ゲスト OS のシステムコール処理前と処理後のそれぞれにおいて，ホスト OS 側に処理を遷移させ，提案手法の機構を実行する必要がある．このため，提案手法はゲスト OS のシステムコール処理をフックできる必要がある．

（実現課題 2） KASLR への対処
Linux 3.14 から Kernel Address Space Layout Randomization（以降，KASLR）と呼ばれるセキュリティ機構が実装されている．KASLR が有効化されている場合，Linux カーネルがロードされるアドレスは，ある固定値からランダム値（以降，KASLR オフセット）だけずれることになる．このため，システムコールの入口と出口のアドレスはブートことに異なる値となる．このような環境で，システムコールフックを実現するためには，ブートごとにランダム化されるシステムコールの入口と出口のアドレスを取得できる必要がある．

（実現課題 3） 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 [7] 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．

5.2 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 [8] 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の入口と出口のアドレスをセットすることで，これらのアドレスに格納されている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ているアドレスに格納されている命令を実行す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 のシステムコール処理をフックできる．

5.3 KASLR への対処

KASLR によってランダム化されるカーネルの先頭アドレスはデフォルトでは 2MB にアラインメントされている．提案手法はこの特性を利用して，以下のように KASLR オフセットを計算する．

(1) カーネルの先頭 2MB 以内に存在する特権命令の実行をフックする．
(2) (1) の時点での rip レジスタの値を取得し，下位 21 ビットをゼロクリアする（この値がカーネルの先頭アドレスとなる）．
(3) (2) で計算した値と KASLR が無効時におけるカーネルの先頭アドレスの差を計算する（この値が KASLR オフセットとなる）．

提案手法では，上記の手順で計算した KASLR オフセットを用いて，KASLR 有効時におけるシステムコールフックを実現する．

5.4 セマンティックギャップへの対処

5.4.1 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，3.4 節で述べた従来手法の（課題 2）に対処することができなくなる．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からゲスト OS の情報を特定することで，セマンティックギャップに対処する．

5.4.2 権限情報の取得

システムコール処理の前後におけ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．以上で述べた変数や構造体の関係を**図 4** に示す．提案手法は図 4 のように構造体の各メンバを辿って，権限情報を取得する．

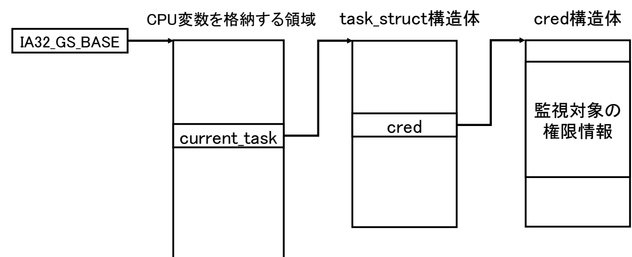

図 4 権限情報の場所

5.4.3 プロセスを識別できる情報の取得

従来手法ではシステムコール処理前に取得した権限情報をカーネルスタックに保存している．システムコール処理後に権限の変更を検証する際には，カーネルスタックに保存した権限情報を取り出す．このように，従来手法はシステムコールを呼び出したプロセスのカーネルスタックに権限情報を保存しているため，検証時に取り出した権限情報がどのプロセスのものか識別する必要がない．しかし，提案手法は権限情報をホスト OS 側のメモリ領域に保存するため，システムコール処理後に正しく対応する権限情報を取り出すためには，保存した権限情報がどのプロセスのものかを識別できる必要がある．

提案手法では取得した権限情報がゲストのどのプロセスのものか識別するために，cr3 レジスタの値を利用する．cr3 レジスタにはプロセスのページテーブルが格納されており，この値はプロセスごとに異なる．このため，cr3 レジスタの値はゲスト OS のプロセスを識別するための識別子として利用できる．提案手法は権限情報とともに cr3 レジスタの値をホスト OS 側のメモリ内に保存する．システムコール処理後に権限の変更を検証する際は，その時点でのゲストの cr3 レジスタの値を取得し，その値に対応する権限情報をホスト OS 側のメモリ内から取り出す．

5.5 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 [3] では以下の 3 つの処理を挙げている．

（処理 1） 権限昇格攻撃を無効化する
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する
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する
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 [3] 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

© 2019 Information Processing Society of Japan

表 4 評価環境

CPU		Intel(R) Core(TM) i5-6500 @ 3.20GHz
OS	ゲスト	Ubuntu 18.04 LTS (Linux 4.15.18, 64bit)
	ホスト	Ubuntu 18.04 LTS (Linux 4.15.18, 64bit)
メモリ	ゲスト	2048MB
	ホスト	4096MB

表 5 システムコールのオーバヘッド（単位: μs）

システムコール	提案手法導入前	提案手法導入後	オーバヘッド
getppid	0.461	23.064	22.603
read	0.640	23.547	22.907
write	0.605	23.322	22.717
stat	1.421	24.780	23.358
fstat	0.698	23.616	22.919
open+close	1.335	24.703	23.368

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，提案手法には攻撃を解析するための機構がない．

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．

5.6 期待される効果

（効果 1） カーネルソースコードの修正が不要
提案手法は KVM 内で実現されているため，導入に伴うゲスト OS のカーネルソースコードの修正が不要である．このため，すでにシステムが稼働しており，カーネルの入れ替えができない状況やソースコードが入手できない状況でも提案手法を導入することができる．

（効果 2） 保存された権限情報を改ざんすることが困難
提案手法は権限情報をホスト OS 側のメモリに保存している．ゲストメモリとホストメモリの分離により，ゲスト OS で動作する攻撃プログラムがホスト OS 側のメモリ内に存在する権限情報を改ざんすることは困難である．

6. 評価

6.1 評価項目と評価環境

本章では，提案手法導入前の環境と導入後の環境それぞれにおいて，以下の 2 項目を評価することによって，提案手法の導入による性能への影響を明らかにする．
(1) システムコールのオーバヘッド
(2) AP 性能への影響

評価に用いる環境を**表 4** に示す．なお，ゲストに割り当てる vCPU の数は 1 とする．

表 6 クライアントの環境

CPU	Intel(R) Core(TM) i7-6700 @ 3.40GHz
メモリ	8192 MB
OS	Ubuntu 16.04 LTS (Linux 4.4.0-141-generic, 64bit)

表 7 Apache の 1 リクエスト当たりのオーバヘッド（単位: ms）

提案手法導入前	提案手法導入後	オーバヘッド
2.126	2.630	0.504 (23.7%)

6.2 システムコールのオーバヘッド

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．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 5** では元の測定値を 2 で割った値を記載している．

測定した結果を表 5 に示す．表 5 より，システムコール 1 回当たりに追加されるのオーバヘッドは 22.603μs〜23.368μs であり，システムコールの種類によらず，オーバヘッドがほぼ一定であることがわかる．提案手法ではゲスト OS が発行したシステムコールをフックし，KVM 側で権限情報の取得，権限情報の保存，および権限情報の変更内容の検証を行う．これはシステムコールの種類によらず，すべてのシステムコールに対して追加される処理である．このため，システムコールのオーバヘッドはほぼ一定になると考えられる．したがって，表 5 の値は妥当である推察できる．

6.3 AP 性能への影響

提案手法の導入による AP 性能への影響を明らかにするために，ApacheBench 2.3 を用いて，提案手法導入前と導入後の Web サーバの性能を測定した．サーバ側の環境は表 4 に示した環境であり，クライアントの環境は**表 6** に示す環境である．なお，評価に用いた Web サーバは Apache 2.4.29 である．評価では，10KB のファイルに対し，10,000 回アクセスした際の 1 リクエスト当たりの処理時間を測定した．なお，リクエストを送信する際の同時接続数は 1 である．

測定した結果を**表 7** に示す．表 7 より，提案手法導入後では 0.504ms のオーバヘッドが追加されており，23.7%の性能低下が生じていることがわかる．また，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，22 回であった．提案手法はすべてのシステムコールに同じ処理を追加するため，処

© 2019 Information Processing Society of Japan

理に追加されるオーバヘッドは処理中に発行するシステムコールの数に比例する．このため，表 7 のオーバヘッドの値を 22 で割るとシステムコールのオーバヘッドが算出できる．表 7 より算出したシステムコールのオーバヘッドは 22.909μs であった．これは 6.2 節で測定したシステムコールのオーバヘッドである 22.603μs〜23.368μs に近い値である．このため，AP のオーバヘッドは，処理におけるシステムコール発行回数に比例すると推察できる．

7. 関連研究

権限情報を保護することによって権限昇格攻撃を防止する手法として文献 [9] がある．文献 [9] 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することができる．しかし，文献 [9] 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 [9] 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，カーネルの入れ替えができない状況やソースコードが入手できない状況でも提案手法を導入できる．

権限昇格検知する手法として，文献 [10] がある．文献 [10] 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 [10] 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．

8. おわりに

システムコールの前後における権限の変更に着目した権限昇格攻撃防止手法を KVM 内で実現する手法を述べた．提案手法は，ゲスト OS におけるシステムコール処理の開始時と終了時をフックし，ゲスト OS 上で動作するプロセスの権限情報の取得と権限昇格攻撃の検知を行う．また，権限昇格攻撃を検知した後，システムコール処理前に保存したプロセスの権限情報を復元することによって，権限昇格攻撃を防止する．

提案手法の実現により，導入の際に OS カーネルのソースコードを修正する必要がなくなる．これにより，提案手法をより多くの環境に適用することが可能になる．また，提案手法ではシステムコール処理前のプロセスの権限情報をホスト OS 側のメモリに保存する．このため，ゲストメモリとホストメモリの分離により，ゲスト OS 上で動作する攻撃プログラムがホスト OS のメモリ内に存在する権限情報を改ざんすることが困難になる．

提案手法の導入による性能への影響を明らかにするために，提案手法の導入前と導入後において性能評価を行った．LMbench を用いた評価では，システムコール 1 回当たりのオーバヘッドは，22.603μs〜23.368μs であった．また，AP 性能への影響の評価として，Apache の 1 リクエスト当たりの処理時間を測定した結果，オーバヘッドは 0.504ms であり，23.7%の性能低下が見られた．

謝辞 本研究の一部は,科学研究費補助金基盤研究 (B)(課題番号: 16H02829) による．

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] 吉谷亮汰，山内利宏：権限の変更に着目した権限昇格攻撃防止手法の ARM への拡張，情報処理学会報告，vol.2018-CSEC-82，no.29，pp.1–7 (2018).

[3] Yamauchi, T., Akao, Y., Yoshitani, R., et al.: Additional Kernel Observer to Prevent Privilege Escalation Attacks by Focusing on System Call Privilege Changes, Proceedings of the 2018 IEEE Conference on Dependable and Secure Computing (IEEE DSC 2018), pp.172–179 (2018).

[4] 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).

[5] AlDanial: cloc, GitHub, available from ⟨https://github.com/AlDanial/cloc⟩ (accessed 2019-01-10).

[6] CVE: CVE - CVE-2013-1763, CVE, available from ⟨https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-1763⟩ (accessed 2019-01-17).

[7] Chen, P.M., Noble, B.D.: When Virtual Is Better Than Real, Proceedings Eighth Workshop on Hot Topics in Operating Systems, pp.133–138 (2001).

[8] Fujii, S., Sato, M., Yamauchi, T., 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).

[9] Chen, Q., Azab, A.M., Ganesh, G., et al.: PrivWatcher: Non-bypassable Monitoring and Protection of Process Credentials from Memory Corruption Attacks, Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, pp.167–178 (2017).

[10] Pham, C., Estrada, Z., Cao, P., et al.: Reliability and Security Monitoring of Virtual Machines Using Hardware Architectural Invariants, Proceedings of the International Conference on Dependable Systems and Networks, pp.13–24 (2016).

© 2019 Information Processing Society of Japan