単一フックポイントのゲストOS監視による検知可能な権限昇格攻撃の拡大とオーバヘッド削減の実現

福本 淳文[1]　山内 利宏[1]

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，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が提案された．この手法は，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，保存した権限情報が攻撃者に改ざんされる可能性がある．これらの課題に対して，我々はKVM内に同様のセキュリティ機構（以降，従来手法）を実現することで対処した．しかし，従来手法は，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できない．また，システムコールを発行するたびにVMexitが2回発生し，システムコール処理のオーバヘッドが大きい．そこで，本稿では，従来手法からシステムコール処理後のフックポイントを削除し，システムコール処理前の権限の監視のみで権限昇格攻撃を防止する手法を提案する．提案手法は，従来手法では対応できなかったシステムコール処理外で発生する権限の変更を検知できる．また，システムコールあたりに発生するVMexitが1回となり，システムコール処理のオーバヘッドが抑えられる．本稿では，提案手法の設計と実現方式について述べ，従来手法では検知できない攻撃を検知できることとオーバヘッドを半分に削減できることを述べる．

Expanding Detectable Privilege Escalation Attacks and Reducing Overhead by Guest OS Monitoring Using Single Hook Point

Akifumi Fukumoto[1]　Toshihiro Yamauchi[1]

Abstract: To address privilege escalation attacks, a prevention method focusing on the change of credentials by system calls has been proposed. Because this method is implemented in OS, it requires modification of kernel source code. In addition, the stored credentials may be forged by an attacker. We addressed these problems by implementing the same mechanism (referred to as previous method) in KVM. But the previous method only can detect the change of credentials that occurs during system call and two VMexit added by previous method per system call may lead to large overhead. To address these problems, we set a hook only before system call. The proposal method can detect the change of credentials that occurs outside system call processing. In addition, the overhead can be reduced because additional VMexit is reduced to once. In this paper, we describe the design and implementation of the proposed method and report the evaluation result.

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を提案した[1,2]．赤尾らの手法は，システムコールサービスルーチンの前後において，

[1] 岡山大学 大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University

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．赤尾ら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，赤尾ら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，赤尾らの手法を導入するためには，Linux カーネルのソースコードに対して，手法を導入するためのパッチをあらかじめ適用する必要がある．また，赤尾らの手法では，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，攻撃者は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．

これらの課題に対処するために，我々は KVM 内に赤尾らの手法と同様のセキュリティ機構を実現した [3]．従来手法の実現により，カーネルのソースコードを変更する必要がなくなった．また，権限情報をホストのメモリ領域に保存するため，ゲストのメモリ空間とホストのメモリ空間の分離により，保存した権限情報の改ざんが困難となった．

しかし，従来手法には以下の 2 つの課題が存在する．

（課題 1） システムコール処理外の改ざんは検知不可

従来手法はシステムコール処理の前後にフックポイントを設定しているため，これらのフックポイントに挟まれた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できない．

（課題 2） システムコール処理のオーバヘッド大

従来手法はシステムコールフックのために，システムコールあたり 2 回の VMexit が発生する．VMexit が発生すると，多くの処理が実行されるため，従来手法では，システムコール処理に大きなオーバヘッドが生じる．

これらの課題に対して，我々はシステムコール処理の前にのみフックポイントを設定し，権限を監視し，権限昇格攻撃を防止する手法を提案する．提案手法は，あるプロセスがシステムコールを発行したときから，次にそのプロセスがシステムコールを発行するまでの間に発生する当該プロセスの権限の改ざんを検知する．このため，従来手法とは異なり，システムコール処理外で発生する権限の改ざんを検知防止できる．また，提案手法はシステムコールあたり 1 回の VMexit しか発生しない．このため，従来手法に比べ，システムコール処理のオーバヘッドが抑えられる．

提案手法において，権限の改ざんを検知できるのは，当該プロセスが次にシステムコールを発行したときである．したがって，提案手法を導入したシステムでは，攻撃者は権限を改ざんした状態で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改ざんが検知される．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．システムコールの発行なしに，攻撃目的を達成することは多くの場合困難である．したがって，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際に行っても問題はない．

本稿では，従来手法の課題，提案手法の設計，実現課題，実現方式，および評価結果にについて述べる．

2. OS の脆弱性を悪用する権限昇格攻撃の防止手法

2.1 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．

2.2 権限の変更に着目した権限昇格攻撃防止手法

2.2.1 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 [4] 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，赤尾ら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．

2.2.2 赤尾らの手法の課題

赤尾らは権限昇格攻撃に対処するため，システムコール処理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提案した．しかし，この手法は OS 内で実現されており，OS と同じ権限で動作するため，以下の課題が存在する．

（課題 1） 導入にカーネルソースコードの変更が必要

赤尾らの手法を導入するためにはカーネルソースコードの変更が必要である．このため，カーネルの入れ替えが困難なシステムやカーネルソースコードが入手できない環境では従来手法を適用することが難しい．

（課題 2） 保存した権限情報の改ざんが可能

赤尾らの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．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赤尾らの手法をバイパスできる可能性がある．

2.3 KVM 上のゲスト OS における権限昇格攻撃防止手法（従来手法）

2.3.1 考え方

従来手法では，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，赤尾らの手法の**（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領域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，赤尾らの手法の**（課題 2）**に対処できる．

2.3.2 基本方式

従来手法の処理流れを**図 1** 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，従来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．

(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，この値を用いて，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．

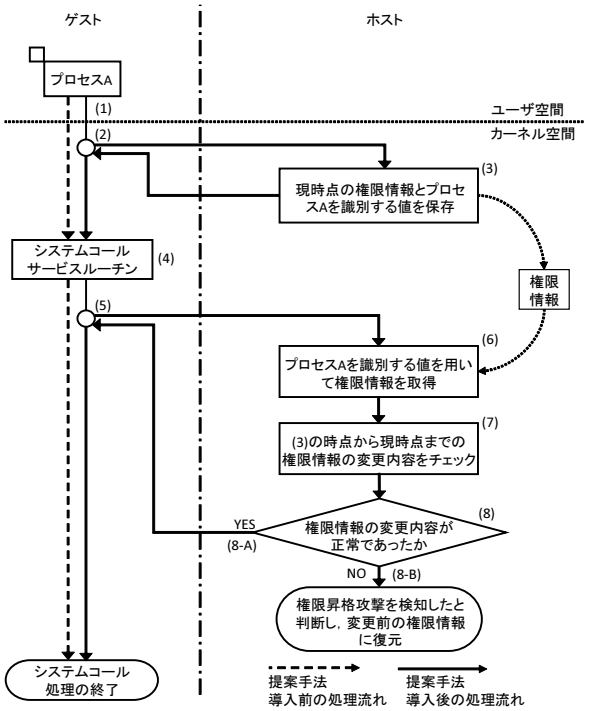

図 1 従来手法の処理流れ

2.3.3 従来手法の課題

従来手法はシステムコール処理の前後にフックポイントを設定しているため，以下の課題が存在する．

（課題 1） システムコール処理外の改ざんは検知不可

従来手法はシステムコール処理の前後にフックポイントを設定している．このため，これらのフックポイントに挟まれた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できず，システムコール処理外で発生する権限の改ざんは検知できない．

実際に Web 上から入手できる権限昇格攻撃の PoC コードを調査したところ，多くの PoC 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，従来手法をバイパスできることがわかった．

（課題 2） システムコール処理のオーバヘッドが大

VMexit が発生すると，VMX Non-root モードの状態保存処理や VMX root モードの状態復元処理が実行される．また，その後に従来手法による処理が実行される．このため，VMexit が発生すると，大きなオーバヘッドが追加される．従来手法はシステムコールあたり 2 回の VMexit が発生す

るため，システムコール処理のオーバヘッドが大きい．

3. 提案手法

3.1 考え方

提案手法は，あるプロセスがシステムコールを発行したときに，そのプロセスの権限情報を保存する．当該プロセスが次にシステムコールを発行したときに，その時点での権限情報と保存した権限情報を比較する．もし，前回のシステムコール処理では変更し得ない権限が変更された場合，権限昇格攻撃があったと判断する．提案手法はある時点でシステムコールを発行してから次のシステムコールが発行されるまでの間に発生する権限の改ざんを検知できる．このため，システムコール処理外で発生する権限の改ざんを検知することができ，**（課題 1）**に対処できる．また，提案手法はシステムコールあたり 1 回の VMexit しか発生しないため，従来手法に比べ，システムコール処理のオーバヘッドが小さくなり，**（課題 2）**に対処できる．

提案手法において，権限の改ざんを検知できるのは，当該プロセスが次にシステムコールを発行したときである．したがって，提案手法を導入したシステムでは，攻撃者は権限を改ざんした状態で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あと，攻撃者は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あと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．

3.2 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 3** 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で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得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，ゲストに処理を移行する．

(6) (3) で取得した権限情報をホストのメモリ領域に保存し，ゲストに処理を移行する．

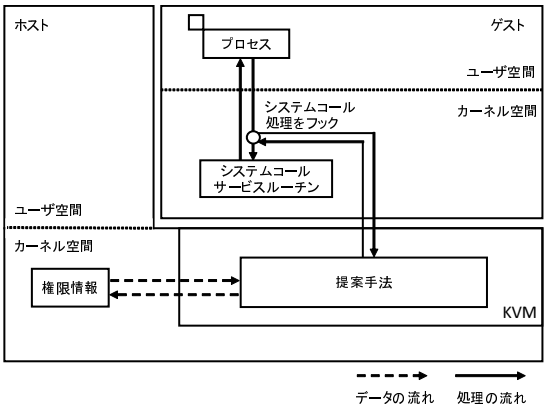

図 2 提案手法の全体像

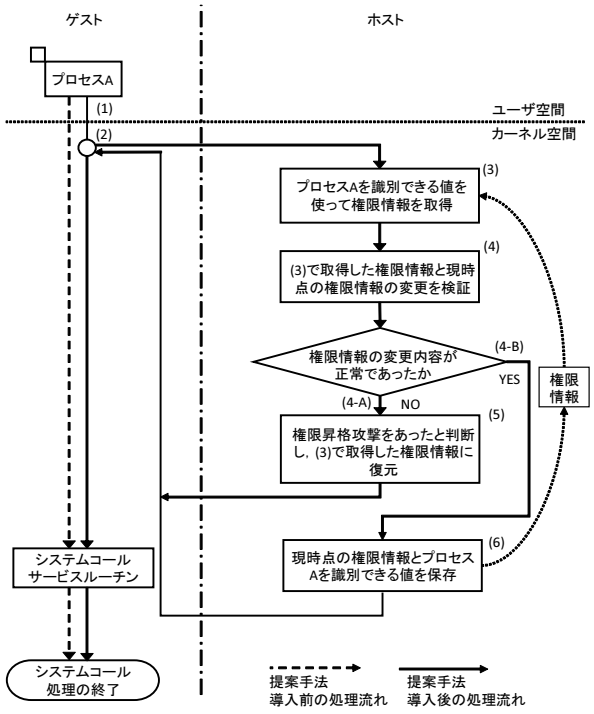

図 3 提案手法の処理流れ

3.2.1 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

提案手法で監視する権限情報を**表 1** に示す．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

表 1 提案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

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() の実行を許可する」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．

権限情報を変更し得るシステムコールを**表 2** に示す．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．

3.3 実現における課題

従来手法とは違い，提案手法は単一のフックポイントのみで権限昇格攻撃を検知する．提案手法を実現するためには以下の実現課題に対処する必要がある．

（実現課題 1） 前回発行したシステムコールの判別
ゲスト OS 上のプロセスがシステムコールを発行した場合，提案手法はシステムコール処理をフックし，同じプロセスが一つ前に発行したシステムコール処理による権限の変更を検証する必要がある．このため，提案手法は前回発行したシステムコールの種類を判別できる必要がある．

（実現課題 2） プロセスの権限情報の削除
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，提案手法は常に，プロセス毎に一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．

表 2 提案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
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

3.4 実現方式

3.4.1 前回発行したシステムコールの判別

提案手法はプロセス毎の格納領域を確保し，プロセスの権限情報とともに，発行したシステムコールのシステムコール番号をホストのメモリ領域に保存することで**（実現課題 1）**に対処した．プロセスがシステムコールを発行した際に，システムコール番号は rax レジスタに格納される．このため，提案手法はゲストの rax レジスタの値を取得し，この値をシステムコール番号として，プロセスの権限情報とともにホストのメモリ領域に保存する．

3.4.2 プロセスの権限情報の削除

提案手法では，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. 評価

4.1 評価内容と評価環境

従来手法および提案手法の適用によって生じるオーバ

表 3 評価環境

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

表 4 クライアントの環境

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

表 5 権限昇格攻撃の検知防止実験結果

CVE 番号	脆弱性の概要	攻撃タイプ	検知防止可能	
			従来手法	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		✓

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．

（評価 1） 攻撃検知防止実験
システムコール処理中に権限が改ざんされる場合とシステムコール処理中でないときに権限が改ざんされる場合のそれぞれにおいて，従来手法および提案手法が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示す．

（評価 2） システムコールのオーバヘッド測定
従来手法および提案手法を導入した環境において，手法の導入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，従来手法と提案手法のオーバヘッドを比較評価する．

（評価 3） AP 性能の評価
従来手法および提案手法を導入した環境において，手法の導入による AP 性能への影響を測定評価する．

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

4.2 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，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．

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を [5,6] 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，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，従来手法は**（タイプ 1）**の攻撃のみ検知防止できたことが分かる．提案手法は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，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）**攻撃であっても，検知防止することができた．従来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

表 6 システムコールのオーバヘッド（単位: 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	
		従来手法 (T2)	提案手法 (T3)	従来手法 (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

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

手法導入前 (T1)	手法導入後		オーバヘッド	
	従来手法 (T2)	提案手法 (T3)	従来手法 (T2 - T1)	提案手法 (T3 - T1)
1.072	1.481	1.271	0.409 (38.2%)	0.199 (18.6%)

いる権限情報を改ざんすることで，提案手法をバイパスすることができる．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かったり，攻撃に root 権限が必要な場合がある [7]．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現実的ではない．

4.3 システムコールのオーバヘッド測定

従来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従来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．従来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従来手法のオーバヘッドの 50.0%〜50.6%になった．このため，従来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．

オーバヘッドが削減された要因は 2 つあると推察できる．1 つ目は，システムコール処理前にのみフックポイントを設定することにより，システムコールあたりに発生する VMexit の回数が 2 回から 1 回に減ったことである．2 つ目は，従来手法で複数回実行されていた処理が，提案手法では 1 回しか実行されないことである．たとえば，従来手法では，ゲスト OS のメモリ領域からプロセスの権限情報を取得する処理をシステムコール処理前と処理後で 2 回実行していた．これに対し，提案手法では，ゲスト OS のメモリ領域からプロセスの権限情報を取得する処理はシステムコール処理前に 1 回実行するのみである．

4.4 AP 性能の評価

従来手法および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，手法導入前，従来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，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．測定結果より，従来手法を導入した場合は 38.2%の性能低下，提案手法を導入した場合は 18.6%の性能低下が見られた．このため，提案手法は従来手法に比べ，AP 性能への影響が小さいことがいえる．また，従来手法と提案手法はほぼすべてのシステムコールに同様の処理を追加するため，オーバヘッドは AP のシステムコール発行回数に比例して大きくなる．このため，従来手法と提案手法のオーバヘッドの差は，AP のシステムコール発行回数に比例して大きくなると推察できる．

5. 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 [8] がある．文献 [8] 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 [8] 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 [8] 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困難な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 [9] がある．文献 [9] 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 [9] 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 [10] では，ゲストのメモリ空間内に int3 命令を埋め込み，フックポイントを設定することで，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 [10] 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．

6. おわりに

従来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防止する．また，従来手法はシステムコール処理のオーバヘッドが大きい．これらの課題に対して，我々は単一フックポイントによるゲスト OS 監視手法を提案し，実現方式を示した．提案手法は従来手法と違い，システムコール処理前のフックのみにおいて，前回発行したシステムコール処理による権限の変更を検証する．提案手法により，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，従来手法に比べてより多くの攻撃を検知できる．

提案手法の攻撃検知範囲を評価するために，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を実施し，従来手法および提案手法がこれらの攻撃を検知防止できるか否かを評価した．システムコール処理中の権限改ざんについては，従来手法と提案手法の両方が権限昇格攻撃を検知防止できた．一方で，システムコール処理外で権限を改ざんする攻撃については，提案手法のみが権限昇格攻撃を検知防止でき，提案手法の有用性を示した．

また，LMbench を用いてシステムコールのオーバヘッドを測定した結果，提案手法のオーバヘッドは従来手法のオーバヘッドの 50.0%〜50.6%になることを示した．また，Apache の 1 リクエストあたりの処理時間を測定した．その結果，従来手法を導入した場合は 38.2%の性能低下，提案手法を導入した場合は 18.6%の性能低下が見られることを示した．以上のことから，提案手法は従来手法のオーバヘッドを半減できることを示した．

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, Proceedings of the 2018 IEEE Conference on Dependable and Secure Computing (IEEE DSC 2018), pp.172–179 (2018).

[3] 福本淳文，山内利宏：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評価，情報処理学会報告，vol.2019-CSEC-84，no.7，pp.1–7 (2019).

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

[5] PerceptionPointTeam: cve_2016_0728 exploit, GitHub Gist, available from ⟨`https://gist.github.com/PerceptionPointTeam/18b1e86d1c0f8531ff8f`⟩ (accessed 2019-06-10).

[6] 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).

[7] CROWDSTRIKE: VENOM Vulnerability, available from ⟨`https://venom.crowdstrike.com/`⟩ (accessed 2019-06-10).

[8] 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).

[9] 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 (2014).

[10] Estrada, Z., Pham, C., Deng, F., et al.: Dynamic VM Dependability Monitoring UsingHypervisor Probes, Proceedings of the 2015 11th European Dependable Computing Conference, pp.61–72 (2015).