

# 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

赤尾 洋平<sup>1</sup> 山内 利宏<sup>1,a)</sup>

**概要:** オペレーティングシステムの脆弱性を悪用する攻撃が増加している。中でも、権限昇格攻撃により権限を奪取された場合、攻撃者は高い権限でシステムを操作できるようになり、攻撃の被害は甚大なものとなる。本稿では、オペレーティングシステムの脆弱性を悪用する権限昇格攻撃を防止する手法を提案する。提案手法は、プロセスの権限が特定のシステムコール処理でのみ変化し得ることに着目し、システムコール処理による権限情報の変化を監視することにより、権限昇格攻撃を防止する。提案手法を導入することにより、脆弱性の種類に関係なく、ゼロデイの脆弱性を悪用する権限昇格攻撃を防止できる。また、カーネルに脆弱性の修正パッチを適用することなく、権限昇格攻撃を防止できる。本稿では、提案手法の内容と実現方式、評価した結果、および提案手法の有効性について述べる。

**キーワード:** オペレーティングシステム, 権限昇格攻撃, システムコール

## 1. はじめに

近年、オペレーティングシステム（以降、OS）の脆弱性を悪用する攻撃が増加している [1][2]。OS は、計算機が動作するための基盤としての役割を担っており、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おり [2][3][4]、現状では、OS を完全に信頼できるとはいえない。また、OS のコード量は膨大であり、すべての脆弱性を取り除くことは困難である。

OS の脆弱性を悪用する攻撃の一つに権限昇格攻撃 (privilege escalation) がある。権限昇格攻撃では、OS カーネルの脆弱性を悪用するコードを実行することにより、プロセスの権限をより高い権限へ昇格させる。攻撃者による権限昇格攻撃が成功した場合、攻撃者は通常よりも高い権限でシステム全体を操作できるようになる。例えば、権限昇格攻撃により root 権限を奪取された場合、攻撃者はアクセス制御を回避し、システム上にあるすべての情報を読み書きできるようになる。このように、一つの権限昇格攻撃がシステム全体のセキュリティを脅かすことにつながり、攻撃者にとっては恰好の的である。また、Android OS において、権限昇格攻撃による root 権限の奪取は root 化と呼ばれ、多くの端末が root 化されている [5]。上記のような理由から、権限昇格攻撃は非常に大きな脅威であり、権限昇

格攻撃への対策は重要である。

既存の対策として、Control-Flow-Integrity (CFI) [6][7][8] がある。CFI を適用することにより、プログラムの control-flow を改ざんする攻撃を防止できる。しかし、CFI は、control-flow に直接は関係しないデータを改ざんする攻撃 (non-control-data 攻撃 [9]) を防止できない問題がある。Trusted Boot[10] や IMA[11] を用いることにより、プログラムの起動時に、プログラムが改ざんされていないことをチェックできる。しかし、ランタイムに脆弱性を悪用する攻撃を防止することはできない。セキュア OS[12][13][14][15] は、管理者権限を分割することにより、攻撃者に管理者権限を奪取された場合であっても、被害をポリシで制限した範囲内に抑制できる。しかし、ポリシの範囲内での被害は発生する。また、セキュア OS は、導入のためのポリシの設定や運用が難しい課題があり、広く利用されているとはいえない状況である。

そこで、本稿では、Linux カーネルの脆弱性を悪用する権限昇格攻撃を防止する手法を提案する。提案手法は、プロセスの権限に関するデータがカーネル空間に保存されていること、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、および各システムコールの役割は細分化されていることに着目し、システムコール処理の前後での権限の変化を監視する。提案手法により、特定のシステムコール処理の前後において、そのシステムコールが変化させることのない権限が変化しているこ

<sup>1</sup> 岡山大学 大学院自然科学研究科

<sup>a)</sup> yamauchi@cs.okayama-u.ac.jp

とを検知した際は、権限昇格攻撃が行われたと判断し、攻撃を防止する。提案手法は、プロセスの権限の変化に着目することから、脆弱性の種類に関係なく、ゼロデイの脆弱性を悪用する権限昇格攻撃を防止できる。また、提案手法をあらかじめシステムに導入しておくことにより、カーネルに脆弱性の修正パッチを適用することなく、権限昇格攻撃を防止できる。

2章では、OSの脆弱性を悪用する権限昇格攻撃について述べる。3章では、我々が提案する権限昇格攻撃の防止手法の設計について述べる。4章では、提案手法を実現し、評価した結果を述べる。5章では、関連研究と提案手法を比較した内容について述べる。最後に、6章で本稿の内容をまとめる。

## 2. OSの脆弱性を悪用する権限昇格攻撃

### 2.1 OSの脆弱性

多くのOSカーネルは、メモリ管理に関するバグが混入しやすいC言語やアセンブリ言語を用いて実装されており、OSカーネルの脆弱性の多くはメモリ管理の不備に起因する脆弱性である[8][16]。また、OSカーネルのコードは膨大であり、最新のLinuxカーネルのコード行数は1,500万行を超えている[17]。このため、OSの脆弱性を全て無くすることは困難である。

OSカーネルの脆弱性が発見された場合、脆弱性のある部分を修正するために、カーネルに対してパッチを適用する必要がある。しかし、OSカーネルへパッチを適用するには、多くの場合、OSの再起動を必要とする。このため、常時稼働し続ける必要のあるシステムに対してカーネルのパッチを適用することは困難である。また、パッチを適用するには適切なパッチをダウンロードする必要があることから、組み込み機器への適用も困難である。

上記の理由から、OSに実装の不備による脆弱性があることを前提に、OSに脆弱性がある場合であっても、脆弱性を悪用する攻撃を防ぐ機構を事前に組み込んでおくことが必要である。

### 2.2 権限昇格攻撃

権限昇格攻撃とは、ユーザやアプリケーション（以降、AP）に権限が設定されているシステムにおいて、ユーザやAPが本来ならば与えられていない権限を不正に奪取する攻撃であり、root権限を奪取するものが多い。攻撃者の権限昇格攻撃により権限を奪取された場合、攻撃者はより高い権限でシステムを操作することができ、結果として情報漏えいやサービス妨害（Denial of Service）につながる。

モバイル端末においても権限昇格攻撃は脅威とされている。Android OSにおいて、権限昇格攻撃によるroot権限の奪取はroot化と呼ばれ、多くの端末がroot化されている[5]。Android OSは内部でLinuxカーネルを使用してお

り、root化の多くはLinuxカーネルの脆弱性を悪用することにより行われる。端末がroot化された場合、端末メーカーが独自に開発したAPやライブラリ等の知的財産が漏えいする可能性がある[5]。

### 2.3 権限昇格攻撃の例（CVE-2014-3153）

本節では、権限昇格攻撃の例としてCVE-2014-3153を悪用する攻撃を取り上げ、文献[5]に記載されている内容を基に、以下で説明する。

CVE-2014-3153はfutexシステムコールの脆弱性であり、Android OSのroot化ツールにおいて多く悪用されている。futexシステムコールは、特定のメモリアドレスを利用したロックを実現するシステムコールである。CVE-2014-3153を悪用する権限昇格攻撃では、パラメータを巧妙に細工されたfutexシステムコールを呼び出し、CVE-2014-3153の脆弱性を悪用することにより、スレッドのaddr.limitの値を本来よりも高い値へ改ざんする。addr.limitは、ユーザ空間とカーネル空間の境界アドレスを格納するカーネル空間の変数である。addr.limitの値が本来よりも高い値へ改ざんされることにより、本来ならばカーネル空間に存在するはずのデータがユーザ空間に存在するとみなされる。攻撃者はこの状態を利用して、writeシステムコール等により、ユーザIDやケーパビリティ等の権限に関するデータを書き換える。上記の処理によりプロセスの権限が昇格し、攻撃者は本来与えられている権限よりも高い権限でシステムを操作できるようになる。

## 3. 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の設計

### 3.1 考え方

提案手法は、Linuxカーネルの脆弱性を悪用する権限昇格攻撃を防止することを目的とする。権限昇格攻撃を防止する手法について検討するために、我々はまず、Linuxにおける権限の管理方法について調査を行った。調査により、Linuxにおける権限管理について、以下の特徴があることが明らかとなった。

- (1) プロセスの権限に関するデータは、プロセス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化するのは、プロセスの権限を変化させる役割を持ったシステムコールが実行される際に限られると考えられる。しかし、Linuxカーネルの脆弱性を悪用する権限昇格攻撃が実行された際はこの限りでない。例えば、2.3節で述べたCVE-2014-3153の脆弱性を悪用する権限昇格攻撃では、futexシステムコールによりaddr.limitの値が変化し、writeシステムコール

表 1 提案手法が監視する権限

監視するデータ	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
cap_permitted	許可ケーパビリティ
cap_effective	実効ケーパビリティ
cap_bset	ケーパビリティバウンディングセット
addr_limit	ユーザ空間とカーネル空間の境界アドレス

によりユーザ ID やケーパビリティが変化する。しかし、futex システムコールや write システムコールは、本来ならばこれらのデータを変化させるシステムコールではない。つまり、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が変化しないシステムコールであるにもかかわらず、システムコール処理の前後でプロセスの権限が変化している。

そこで、我々は、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を提案する。提案手法は、特定のシステムコール処理の前後において、そのシステムコールが変化させることのない権限が変化していることを検知することにより、権限昇格攻撃を防止する。なお以降では、Linux 3.10.0 (64bit) を例に、提案手法の設計について述べる。ただし、提案手法は特定のバージョンに依存せず、その他のバージョンにおいても適用可能である。

### 3.2 監視対象の権限

提案手法は、システムコール処理の前後でプロセスの権限をチェックすることにより、システムコール処理における権限の変化を監視する。本節では、提案手法が監視するプロセスの権限について述べる。

提案手法が監視する権限（以降、権限情報と呼ぶ）を表 1 に記す。表 1 の権限情報は、すべてプロセスのカーネル空間に保存されている。

表 1 のうち、uid 群 (uid, euid, fsuid, suid) と gid 群 (gid, egid, fsgid, sgid) には、それぞれユーザ識別子とグループ識別子が保存され、ファイルやディレクトリへのアクセス権のチェックや、特権操作の可否のチェックに用いられる。

ケーパビリティ群 (cap\_inheritable, cap\_permitted, cap\_effective, cap\_bset) には、特定の処理の実行をプロセスに許可するか否かを示す複数のフラグが保存されている [18]。ケーパビリティの例としては「種々のネットワー

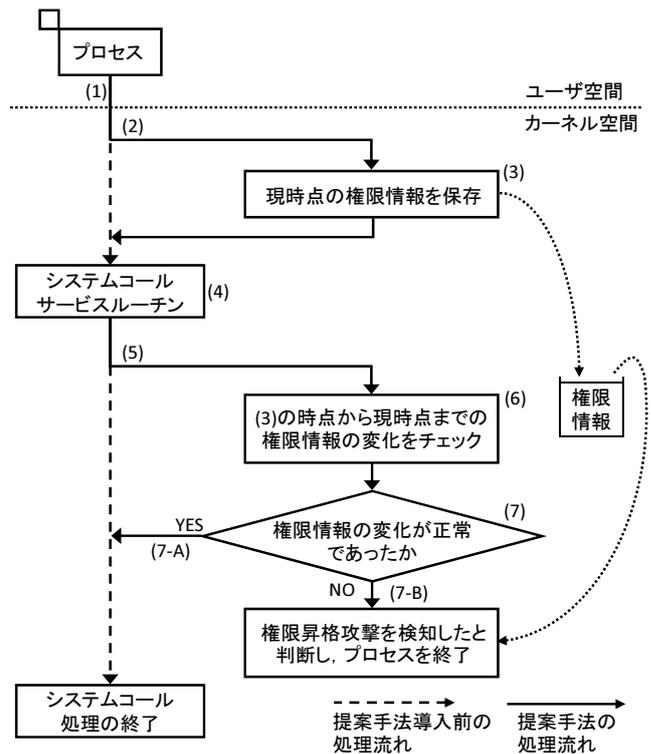


図 1 提案手法の処理の流れ

ク処理を許可する」や「chroot() の実行を許可する」などがある。

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。正常な状態では、uid 群, gid 群, およびケーパビリティ群はカーネル空間に保存されており、ユーザ空間から自由に書き換えることはできない。しかし、addr\_limit の値を攻撃者により改ざんされた場合、uid 群, gid 群, およびケーパビリティ群が保存されている領域がカーネル空間ではなくユーザ空間であると認識され、ユーザ空間から自由に書き換えられる状態となる。このため、addr\_limit の値も提案手法の監視対象とする。

### 3.3 基本方式

提案手法の処理の流れを図 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、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）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、提案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化（システムコール処理に

表 2 権限情報を変化させ得るシステムコール

システムコール	番号	処理内容
execve	59	実行可能プログラムを実行する
setuid	105	uid を設定する
setgid	106	gid を設定する
setreuid	113	uid, euid を設定する
setregid	114	gid, egid を設定する
setresuid	117	uid, euid, suid を設定する
setresgid	119	gid, egid, sgid を設定する
setfsuid	122	fsuid を設定する
setfsgid	123	fsgid を設定する
capset	126	カーナビリティ群を設定する
prctl	157	引数に応じてプロセスを操作する

よる権限情報の変化) 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化が正常なものであったかを確認する。

(A) 権限情報の変化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) 権限情報の変化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。

表 2 に、表 1 で示した権限情報を変化させ得るシステムコールの一覧を記す。調査の結果、Linux 3.10.0 (64bit) では全 313 個のシステムコールのうち、表 2 に記す 11 個のシステムコールが権限情報を変化させ得るシステムコールであった。

(7) における不正な権限情報の変化とは、それぞれのシステムコール処理において、変化しないはずの権限情報が変化している状態を指す。例えば、capset システムコールはカーナビリティ群を設定するシステムコールであるため、capset システムコールの前後においてカーナビリティ群以外の権限情報が変化していた場合は、不正な権限情報の変化であると判断する。

上記の処理により、システムコール処理による不正な権限情報の変化を検知することで、Linux カーネルの脆弱性を悪用する権限昇格攻撃を防止する。

### 3.4 提案手法の詳細

#### 3.4.1 権限情報を保存する領域

提案手法は、システムコール処理前の自スレッドの権限情報を取得し、保存する。保存する領域は、各スレッド毎に割り当てられるカーネル空間内の領域である。各スレッド毎に割り当てられる領域を用いることで、各スレッド毎の別々の領域に、それぞれの権限情報を保存できる。これにより、システムコール処理の最中にコンテキストスイッチが発生した場合であっても、各スレッドの権限情報をそ

れぞれ保存しておくことができる。また、マルチコア環境にも対応できる。

#### 3.4.2 権限昇格攻撃を検知した際の操作

3.3 節で述べた図 1 の (7-B) について、権限昇格攻撃を検知した際に提案手法はプロセスを終了させるが、権限昇格攻撃を検知した際に行う操作は他にも考えられる。例えば、権限昇格攻撃を検知した際、権限情報を正常なものに戻して再開させることにより、権限昇格攻撃を防止しつつプロセスの実行を継続できる。これは、可用性が重要視される状況では、選択肢の一つに成り得ると考えられる。他の例として、プロセスを終了させるのではなく停止させることにより、権限昇格攻撃を検知した時点でのプロセスの解析を行えるようになり、原因の究明に役立てることができる。権限昇格攻撃を検知した際の操作については、システムの状況に応じて検討する必要があると考えられる。

#### 3.4.3 拡張性

提案手法は、表 1 に示した権限情報の変化を監視する。ただし、提案手法の考え方は、権限情報以外のカーネル空間のデータにも有効である。また、提案手法において、監視するデータを拡張することは容易である。このため、カーネル空間のデータ構造とシステムコールの調査を進めることにより、権限情報以外のデータを改ざんする攻撃を防止するように提案手法を拡張することは可能である。

### 3.5 期待される効果

(効果 1) ゼロデイの脆弱性を悪用する権限昇格攻撃を防止可能

OS カーネルのコード量は膨大であり、脆弱性を未然に全て取り除くことは困難である。そこで提案手法を適用することにより、ゼロデイの脆弱性を悪用する攻撃を受けた場合であっても、脆弱性の内容に関係なく、権限情報の改ざんを防止できる。攻撃者による権限昇格攻撃が達成された場合、攻撃者は高い権限でシステムを操作できるようになり、システムに対する攻撃の被害は甚大なものとなる。このため、権限情報の改ざんを防止することは重要である。

(効果 2) カーネルに脆弱性の修正パッチを適用することなく権限昇格攻撃を防止可能

提案手法をあらかじめシステムに導入しておくことにより、カーネルに脆弱性の修正パッチを適用することなく、権限昇格攻撃を防止できる。OS カーネルの脆弱性が発見された場合は、脆弱性を修正するために、脆弱性の内容に応じたカーネルのパッチを適用する必要がある。しかし、サーバや組み込み機器などのように、カーネルへのパッチ適用が困難である状況もある。このため、security by design[19] の考え方のように、脆弱性があることを前提に、防御機構をあらかじめ組み込んでおくことが重要である。

表 3 権限昇格攻撃の検知実験の結果

脆弱性の CVE 番号	脆弱性の概要	カーネルのバージョン	検知の可否
CVE-2014-3153	futex() におけるリキュー操作を行うアドレスのチェックの不備	Linux 3.10.0	○
CVE-2014-0038	recvmsg() におけるパラメータのチェックの不備によるメモリ破壊	Linux 3.8.0	○
CVE-2013-1763	socket() におけるパラメータのチェックの不備による配列のインデックスエラー	Linux 3.5.0	○

## 4. 実現方式と評価

### 4.1 実現方式

3章で述べた権限昇格攻撃の防止手法を Linux 3.10.0 (64bit) に実現した。なお、今回の実現では、表 2 に示した権限情報を変化させ得る 11 個のシステムコールを除いた 302 個のシステムコールによる権限情報の変化を監視している。権限情報を変化させ得るシステムコールの処理内容ごとに監視する権限情報を変更する機構の実現は今後の課題とする。この機構は、表 2 のシステムコールの処理内容を調査することにより実現できる。

提案手法は、システムコールサービスルーチンの処理前 (図 1 の (2)) と処理後 (図 1 の (5)) をフックし、提案手法の関数へ処理を移行させる必要がある。提案手法におけるシステムコールサービスルーチンの処理前と処理後のフックは、システムコールの発行直後に実行されるシステムコールハンドラにおいて、システムコールサービスルーチンが呼ばれる前後で提案手法の関数を呼び出すことで実現する。

また、提案手法のフック関数内で表 1 に示した権限情報を取得する必要がある。表 1 に示した権限情報のうち uid 群, gid 群, およびケーパビリティ群は、プロセス制御ブロックの task\_struct 構造体内の cred 構造体から取得する。addr\_limit は、プロセス制御ブロックの thread\_info 構造体から取得する。

権限情報を変化させ得るシステムコールか否かの判断は、システムコール発行の直後に rax レジスタに格納されているシステムコール番号を参照することで実現する。

### 4.2 評価内容

提案手法を実現した環境を用いて評価を行う。評価項目と評価の目的を以下に記す。

#### (評価 1) 権限昇格攻撃の検知実験

- 提案手法を適用した環境において、複数の種類の権限昇格攻撃を実施し、提案手法が権限昇格攻撃を検知できるか否かを評価する。

#### (評価 2) 性能測定

- 提案手法の導入により、性能への影響があると予測される。そこで、提案手法の導入前と導入後の性能を測定し、比較することにより、提案手法の導入による性能への影響を明らかにする。

表 4 性能測定における評価環境

CPU	Intel Core i5-3470 3.2GHz (4 コア)
メモリ	4.0GB
OS	Linux 3.10.0 (64bit)

### 4.3 権限昇格攻撃の検知実験

提案手法を導入した Linux 環境において、Linux カーネルの脆弱性を悪用する権限昇格攻撃を実施し、提案手法が権限昇格攻撃を検知できるか否かを評価する。攻撃には、Web 上から入手できる 3 つの 익스プロイトコード [20][21][22] を用いた。これらはいずれも、Linux カーネルの脆弱性を悪用し、権限昇格攻撃を達成する 익스プロイトコードである。なお、Linux カーネルのバージョン毎にそれぞれの 익스プロイトコードが悪用する脆弱性の有無は異なる。このため、それぞれの 익스プロイトコードが悪用する脆弱性が存在するバージョンの Linux カーネルに対して、それぞれ提案手法を導入し、検知実験を行った。

検知実験の結果を表 3 に記す。表 3 には、検知実験に用いた 익스プロイトコードが悪用する脆弱性の CVE 番号と脆弱性の概要を記している。表 3 から、提案手法は、Linux カーネルの異なる 3 つの脆弱性を悪用する権限昇格攻撃を検知できたことが分かる。具体的には、CVE-2014-3153[20] では、futex システムコールの前後で addr\_limit が変化していることを検知した。CVE-2014-0038[21] では、open システムコールの前後で uid 群, gid 群, cap\_inheritable, cap\_permitted, および cap\_effective が変化していることを検知した。CVE-2013-1763[22] では、sendto システムコールの前後で uid 群, gid 群, cap\_inheritable, cap\_permitted, および cap\_effective が変化していることを検知した。なお、今回の検知実験に用いた 익스プロイトコード以外による攻撃であっても、システムコールを介して不正に権限を奪取する攻撃であれば、提案手法により検知できると推察できる。以上により、提案手法は、システムへの被害が生じる前に権限昇格攻撃を防止できることを示した。

### 4.4 性能測定

#### 4.4.1 システムコールのオーバーヘッド

はじめに、提案手法の導入によるシステムコールのオーバーヘッドを明らかにするために、OS のマイクロベンチマークスイートである Lmbench 3.0[23] の lat\_syscall を用いてシステムコールのオーバーヘッドを測定した。本節における評価環境を表 4 に記す。

表 5 システムコールのオーバーヘッド (単位:  $\mu\text{s}$ )

システムコール	提案手法の導入前	提案手法の導入後	オーバーヘッド
stat()	0.365	0.386	0.021 (5.8%)
fstat()	0.093	0.119	0.026 (28.0%)
write()	0.105	0.154	0.049 (46.7%)
read()	0.078	0.123	0.045 (57.7%)
getppid()	0.039	0.061	0.022 (56.4%)
open()+close()	1.133	1.162	0.015 (2.6%)

測定結果を表 5 に記す。表 5 から、システムコール 1 回当たりのオーバーヘッドは、 $0.015\mu\text{s}\sim 0.049\mu\text{s}$  であり、小さいことがわかる。なお、open() + close() の測定結果について、オーバーヘッドを 2 で割った値をシステムコール 1 回当たりのオーバーヘッドとして記している。提案手法の導入により追加される処理は、システムコールのフック、権限情報の取得、および権限情報の変化のチェックのみであり、処理負荷の高い処理は追加されないため、この結果は妥当であると推察できる。

#### 4.4.2 提案手法の導入による AP の性能への影響

提案手法の導入による AP の性能への影響を評価するために、提案手法の導入前と導入後における Web サーバの性能を測定し、比較した。評価に用いた Web サーバは Apache 2.4.6 である。評価では、ApacheBench 2.3 を用いて、1Gbps の通信路において、1KB、10KB、および 100KB のファイルに対し、それぞれ 10 万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。サーバ側の環境は、表 4 に記した環境である。クライアント側の環境は、CPU は Intel Pentium4 3.60GHz、メモリは 1.0GB、カーネルは Linux 2.6.32 である。なお、リクエストを送信する際の同時接続数は 10 としている。測定結果を表 6 に記す。表 6 より、提案手法の導入による 1 リクエスト当たりのオーバーヘッドは、 $0.4\mu\text{s}$  以下であることが分かる。なお、100KB の場合のオーバーヘッドが 0.0% となっているのは、ネットワーク帯域がボトルネックになっており、Web サーバ計算機上の提案手法によるオーバーヘッドが隠蔽されているためであると推察できる。

次に、提案手法の導入前と導入後におけるカーネル (Linux 3.10.0) のビルド時間を測定し、比較した。測定結果を表 7 に記す。表 7 より、提案手法の導入によるカーネルのビルド時間のオーバーヘッドは、3.5s であることが分かる。また、提案手法の導入前の処理時間に対するオーバーヘッドの割合は、0.1% である。

以上の結果より、提案手法の導入による実 AP の性能への影響は小さいことが分かる。

## 5. 関連研究

### 5.1 Control-Flow-Integrity

Control-Flow-Integrity (CFI) は、プログラムの control-

表 6 Apache の 1 リクエスト当たりの処理時間 (単位:  $\mu\text{s}$ )

ファイルサイズ	提案手法の導入前	提案手法の導入後	オーバーヘッド
1KB	70.21	70.38	0.17 (0.2%)
10KB	113.13	113.53	0.40 (0.4%)
100KB	941.73	941.72	-0.01 (0.0%)

表 7 カーネルのビルド時間 (単位: s)

導入前	導入後	オーバーヘッド
2649.1	2652.6	3.5 (0.1%)

表 8 CFI と提案手法が防止できる攻撃の種類

	control-flow の改ざん	non-control-data 攻撃
CFI	防止可能	防止不可能
提案手法	結果的に権限情報を改ざんする攻撃を防止可能	権限情報を対象とする攻撃を防止可能

flow を改ざんする攻撃を防止するセキュリティ機構である [6][7]。CFI の適用により、スタック領域のコード実行、return アドレスの改ざん、およびカーネル権限によるユーザ空間のコード実行 (return-to-user 攻撃 [8]) などを防止できる。CFI の多くはユーザ空間を対象としているが、カーネル空間の CFI も KCoFI [8] として実現されている。しかし、近年の研究において、細粒度の CFI に対しても攻撃が成功する可能性があることが示されており、セキュアなシステムを実現するには、CFI の適用のみでは不十分である [24][25]。

CFI で防止できない攻撃として、non-control-data 攻撃 [9] がある。non-control-data 攻撃は、control-flow に直接は関係しないデータを改ざんする攻撃であり、攻撃対象として、ユーザ入力、ユーザ識別子、およびフラグ (例えば is\_admin のような変数) などがあがる。表 8 に、CFI と提案手法が防止できる攻撃の種類を記す。提案手法は、CFI と比較して、権限情報を対象とする non-control-data 攻撃を防止できる利点がある。しかし、提案手法は、control-flow を改ざんする攻撃について、結果として権限情報を改ざんする攻撃は防止できるが、それ以外の攻撃は防止できない。そこで、CFI 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ている。

### 5.2 セキュア OS, Linux Security Module

セキュア OS は、強制アクセス制御 (Mandatory Access Control: MAC) と最小特権 (Least Privilege) を実現するセキュリティ機構である。代表的なセキュア OS として SELinux [12], TOMOYO Linux [13], および AppArmor [14] がある。また、Android OS においては、SELinux を Android OS 用に拡張した SEAndroid [15] が Android 4.3 から導入されている。セキュア OS を適切なポリシーで適用することにより、管理者権限を奪取された場合であっても、被害をポリシーで許可された範囲内に制限できる。

セキュア OS は、Linux Security Module (LSM) [26] を利用して実現されている。LSM は、システムコールの処理前に処理をフックし、カーネル空間のフック関数に移行させる。セキュア OS は、LSM のフック関数内でアクセス制御を行っている。提案手法と LSM の大きな違いとして、LSM はシステムコールの処理前のみをフックするのに対し、提案手法はシステムコールの処理前と処理後をフックする。システムコールの処理前のフックのみでは、システムコールの処理内容を監視することはできず、システムコール処理前の実行状態に基づいたアクセス制御しか行えない。このため、LSM では、システムコールの処理中に脆弱性を悪用され、不正な処理を実行されたとしても、それを検知することはできない。また、文献 [27] では、近年の攻撃において、攻撃者がカーネルの脆弱性を悪用し、LSM を無効化する問題が挙げられている。一方、提案手法では、システムコールの処理前と処理後をフックすることにより、システムコール処理による権限情報の変化をチェックするため、脆弱性を悪用する権限昇格攻撃を防止できる。

なお、現在の提案手法では、すべてカーネルを書きかえることにより実現している。一方、LSM はフック関数群をカーネルモジュールとして導入できる利点がある。今後の目標として、提案手法のフック関数をカーネルモジュールとして導入できるようにすることがある。

### 5.3 カーネルの完全性保護

Trusted Boot[10] や Integrity Measurement Architecture (IMA) [11] は、プログラムの起動時に、起動するプログラムが改ざんされていないことをチェックする。これらを用いることで、システム起動時のカーネルの完全性を保護できる。しかし、これらの手法では、ランタイムに脆弱性を悪用する攻撃を防止することはできない。

ランタイムにおけるカーネルの完全性保護手法 [28][29][30][31] は、プログラムの実行中に、カーネル空間に不正なコードが挿入されていないことをチェックすることにより、攻撃者によるカーネル空間へのコード挿入 (code injection) を防止する。これらの手法は、カーネル空間のコード領域のように、システムの実行中に変化しないデータを保護できる。しかし、権限情報のように、実行状態に応じて変化し得るデータの改ざんを防止できない。例えば、権限情報は、正常な状態であっても変化し得るデータであり、変化したとしても、それを不正な変化であると単純に断定することはできない。このため、システムの実行中に変化し得るデータの不正な変化を検知するためには、実行状態を考慮する必要がある。そこで、提案手法は、権限情報が特定のシステムコール処理でのみ変化し得ることに着目し、実行されたシステムコールの種類を考慮することにより、権限情報の正常な変化と不正な変化を判別する。これにより、プロセスの実行中における不正な権

限情報の変化を検知し、権限昇格攻撃を防止する。

## 6. おわりに

Linux カーネルの脆弱性を悪用する権限昇格攻撃を防止する手法を提案し、その方式と評価結果について述べた。提案手法は、プロセスの権限に関するデータがカーネル空間に保存されていること、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、および各システムコールの役割は細分化されていることに着目し、システムコール処理の前後での権限の変化を監視する。提案手法を適用することにより、脆弱性の内容に関係なく、ゼロデイの脆弱性を悪用する権限昇格攻撃を防止できる。また、提案手法をあらかじめシステムに導入しておくことにより、カーネルに脆弱性の修正パッチを適用することなく、権限昇格攻撃を防止できる。

また、提案手法を実現したシステムにおいて、Web 上から入手できるエクスプロイトコードを用いて権限昇格攻撃の検知実験を行った。評価結果から、提案手法を用いることにより、複数の種類の権限昇格攻撃を検知できることを示した。これにより、システムへの被害が生じる前に権限昇格攻撃を防止できることを示した。

性能評価では、LMbench を用いた性能測定により、提案手法の導入によるシステムコール 1 回当たりのオーバーヘッドは、 $0.015\mu\text{s}$ ~ $0.049\mu\text{s}$  と小さいことを示した。また、Apache の 1 リクエスト当たりの処理時間とカーネルのビルド時間を測定した結果、性能低下は 0.4% 以下であり、提案手法の導入による性能への影響は小さいことを示した。

残された課題として、さらなる権限昇格攻撃の脆弱性による評価、および提案手法の拡張がある。

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] Kemerlis, V. P., Portokalidis, G. and Keromytis, A.: D.: kGuard: lightweight kernel protection against return-to-user attacks, Proc. 21st USENIX conference on Security symposium (USENIX Security '12), pp. 459-474 (2012).
- [2] Kemerlis, V. P., Polychronakis, M. and Keromytis, A. D.: ret2dir: Rethinking Kernel Isolation, Proc. 23rd USENIX conference on Security Symposium (USENIX Security '14), pp.957-972 (2014).
- [3] Niu, S., Mo, J., Zhang, Z. and Lv, Z.: Overview of Linux Vulnerabilities, Proc. 2nd International Conference on Soft Computing in Information Communication Technology (SCICT 2014), pp.225-228 (2014).
- [4] Chen, H., Mao, Y., Wang, X., Zhou, D., Zeldovich, N. and Kaashoek, M. F.: Linux kernel vulnerabilities: state-of-the-art defenses and open problems, Proc. 2nd ACM SIGOPS Asia-Pacific Workshop on Systems (APSys '11), No.5 (2011).
- [5] 小久保博崇, 古川 和快, 兒島 尚, 武仲 正彦: Android/Linux 脆弱性についての一考察, 2015 年暗号と情報セキュリティシンポジウム (SCIS 2015) 論文集, 電子媒体 (2015).

- [6] Abadi, M., Budi, M., Erlingsson, U. and Ligatti, J.: Control-flow integrity, Proc. 12th ACM conference on Computer and communications security (CCS '05), pp.340-353 (2005).
- [7] Petroni, Jr., N.L and Hicks, M.: Automated Detection of Persistent Kernel Control-Flow Attacks, Proc. 14th ACM Conference on Computer and Communications Security (CCS '07), pp.103-115 (2007).
- [8] Criswell, J., Dautenhahn, N. and Adve, V.: KCoFI: Complete Control-Flow Integrity for Commodity Operating System Kernels, 2014 IEEE Symposium on Security and Privacy (IEEE S&P '14), pp.292-307 (2014).
- [9] Chen, S., Xu, J., Sezer, E. C., Gauriar, P. and Iyer R.K.: Non-control-data attacks are realistic threats, Proc. 14th conference on USENIX Security Symposium (USENIX Security '05), pp.177-192 (2005).
- [10] Trusted Computing Group: Trusted Boot, available from <http://www.trustedcomputinggroup.org/trusted-boot/> (accessed 2016-08-06).
- [11] Integrity Measurement Architecture (IMA), available from <https://sourceforge.net/projects/linux-ima/> (accessed 2016-08-06).
- [12] NSA/CSS: Security-Enhanced Linux, available from <https://www.nsa.gov/what-we-do/research/selinux/> (accessed 2016-7-28).
- [13] TOMOYO Linux, available from <https://tomoyo.osdn.jp/index.html> (accessed 2016-07-28).
- [14] AppArmor security project wiki, available from [http://wiki.apparmor.net/index.php/Main\\_Page](http://wiki.apparmor.net/index.php/Main_Page) (accessed 2016-07-28).
- [15] Security Enhancements (SE) for Android, available from <http://seandroid.bitbucket.org/> (accessed 2016-07-28).
- [16] Szekeres, L., Payer, M., Wei, T., Song, D.: SoK: Eternal War in Memory, Proc. 2013 IEEE Symposium on Security and Privacy (IEEE S&P '13), pp.48-62 (2013).
- [17] Linux Counter, available from <https://www.linuxcounter.net/statistics/kernel> (accessed 2016-07-23).
- [18] Hallyn, S. E. and Morgan A. G.: Linux Capabilities: making them work, Proc. Linux Symposium, pp.163-172 (2008).
- [19] WhatIs.com: security by design, available from <http://whatis.techtarget.com/definition/security-by-design> (accessed 2016-08-03).
- [20] Exploit Database: Linux Kernel 3.14.5 (RHEL / CentOS 7) - 'libfutex' Local Root Exploit, available from <https://www.exploit-db.com/exploits/35370/> (accessed 2016-07-22).
- [21] Exploit Database: Linux Kernel 3.4 < 3.13.2 (Ubuntu 13.04/13.10) - 'CONFIG\_X86\_X32=y' Local Root Exploit (3), available from <https://www.exploit-db.com/exploits/31347/> (accessed 2016-07-22).
- [22] Exploit Database: Linux Kernel 3.7.10 (Ubuntu 12.10 x64) - 'sock\_diag\_handlers' Local Root Exploit (2), available from <https://www.exploit-db.com/exploits/24746/> (accessed 2016-08-02).
- [23] LMBench - Tools for Performance Analysis, available from <http://www.bitmover.com/lmbench/> (accessed 2016-07-26).
- [24] Carlini, N., Barresi, A., Payer, M. and Wagner, D.: Control-Flow Bending: On the Effectiveness of Control-Flow Integrity, Proc. 24th USENIX conference on Security Symposium (USENIX Security '15), pp.161-176, (2015).
- [25] Evans, J., Long, F., Otgonbaatar, U., Shrobe, H., Rinaud, M., Okhravi, H. and Sidiroglou-Douskos, S.: Control Jujutsu: On the Weaknesses of Fine-Grained Control Flow Integrity, Proc. 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15), pp.901-913 (2015).
- [26] Wright, C., Cowan, C., Smalley, S., Morris, J. and Kroah-Hartman, G.: Linux Security Modules: General Security Support for the Linux Kernel, Proc. 11th USENIX conference on Security Symposium (USENIX Security '02), pp.17-31 (2002).
- [27] 古川和快, 兒島尚, 小久保博崇, 武仲正彦: Android Security に関する一考察 3, 2016 年暗号と情報セキュリティシンポジウム (SCIS 2016) 論文集, 電子媒体 (2016).
- [28] Azab, A. M., Ning, P., Shah, J., Chen, Q., Bhutkar, R., Ganesh, G., Ma, J. and Shen, W.: Hypervision Across Worlds: Real-time Kernel Protection from the ARM TrustZone Secure World, Proc. 21st ACM SIGSAC Conference on Computer and Communications Security (CCS '14), pp.90-102 (2014).
- [29] Riley, R., Jiang, X. and Xu, D.: Guest-Transparent Prevention of Kernel Rootkits with VMM-Based Memory Shadowing, Proc. 11th international symposium on Recent Advances in Intrusion Detection (RAID '08), pp.1-20 (2008).
- [30] Seshadri, A., Juk, M., Qu, N. and Perrig, A.: SecVisor: a tiny hypervisor to provide lifetime kernel code integrity for commodity OSes, Proc. 21st ACM SIGOPS symposium on Operating systems principles (SOSP '07), pp.335-350 (2007).
- [31] 小倉寛之, 大山恵弘, 岩崎英哉: カーネルレベルルートキット検知システムの構築, 情報処理学会研究報告, Vol.2008-OS-108, No.35, pp.51-58 (2008).