Vol.2018-CSEC-82 No.29
Vol.2018-SPT-29 No.29
2018/7/25

# 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

吉谷 亮汰[1]　山内 利宏[1,a)]

**概要**：オペレーティングシステムの脆弱性を悪用した権限昇格攻撃によって，攻撃者に管理者権限を奪取された場合，攻撃者はシステム全体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．そこで，我々は，システムコールによるプロセスの権限の変更に着目し，プロセスの権限の変更内容を監視する権限昇格攻撃防止手法を提案した [1]．この手法は x86–64 アーキテクチャのみで実現していた．本稿では，手法を拡張し，ARM 環境に対する権限昇格攻撃を防止する手法について述べ，評価した結果を報告する．提案手法は，ARM アーキテクチャ用のシステムコールハンドラ内で行われるシステムコールサービスルーチン呼び出しの前後においてプロセスの権限の変更内容を監視することで，権限昇格攻撃による不正な権限の変更を検知することができる．また，提案手法は，監視対象とするプロセスの権限および権限を変更し得るシステムコールを新たに追加する．さらに，提案手法は，システムコールハンドラ内の高速パスおよび低速パス両方のシステムコールサービスルーチン呼び出しにおいてプロセスの権限の変更内容を監視する．

Ryota Yoshitani[1]　Toshihiro Yamauchi[1,a)]

## 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]，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 [3]，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．権限昇格攻撃が成功した場合，攻撃者は，本来与えられている権限より高い権限でシステムを操作できるようになる．特に，攻撃者に管理者権限が奪取された場合，システム全体のセキュリティが脅かされることになる．

我々は，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，システムコールによるプロセスの権限の変更に着目し，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権限昇格攻撃防止手法を提案した [1]．この手法は，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更し得ないプロセスの権限が変更されていることを検知した場合，権限昇格攻撃が行われたと判断し，攻撃を防止する．

文献 [1] の手法は x86–64 アーキテクチャのみで実現している．しかし，モバイル端末や IoT (Internet of Things) 機器では，消費電力を抑える目的から ARM アーキテクチャが多く採用されている．モバイル端末のうち 8 割のシェアを獲得している Android 端末 [4] においては，権限昇格攻撃による管理者権限の奪取は root 化と呼ばれ，多くの端末が root 化されている [5]．これらの理由から，ARM アーキテクチャにおける権限昇格攻撃への対策は重要であり，文献 [1] の手法を ARM アーキテクチャへ対応させることが課題となっている．

Android における既存のセキュリティ技術としては，SE for Android [6] がある．SE for Android は SELinux [7] を Android に応用したものであり，管理者権限を分割することで，攻撃者に管理者権限を奪取された場合でも，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，導入のためのポリシの設定や運用が難しいという問題がある．組込みシステム向けの既存のセキュリティ技術としては，ARM プロセッサのセキュリティ機能である TrustZone [8] を利用している

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

© 2018 Information Processing Society of Japan

LiSTEE$^{TM}$ [9] がある．LiSTEE$^{TM}$ は暗号処理のような保護対象の処理について，更新可能なようにソフトウェアとして実装しつつ，不正な解析や改変に対する強度を保つ仕組みを実現し，組み込みシステムの長期安全性を確保する．一方で，保護対象の処理の実行環境である OS やミドルウェアの脆弱性を悪用された場合，ソフトウェアの更新の仕組みが改変されたり，暗号処理における鍵などの機密情報が不正に取得されたりする可能性がある．

文献 [1] の手法の ARM アーキテクチャへの対応以外の課題としては，プロセスの権限が文献 [1] で調査した Linux 3.10 以降で新たに追加されたことや権限を変更し得るシステムコールに漏れがあったことについて対処することが挙げられる．また，文献 [1] の手法の権限の変更内容の監視に漏れがあり，監視できていない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）呼び出しを用いてプロセスの権限を改ざんされた場合，権限昇格攻撃を防止できないため，対策を取る必要がある．

これらの課題に対処するために，本稿では，文献 [1] の手法を拡張し，ARM 環境に対する権限昇格攻撃を防止する手法（以降，提案手法）について述べ，評価した結果を報告する．提案手法は，ARM アーキテクチャ用のシステムコールハンドラ内で行われるシステムコールサービスルーチン呼び出しの前後においてプロセスの権限の変更内容を監視することで，権限昇格攻撃による不正な権限の変更を検知する．提案手法は新たに，監視対象とするプロセスの権限に周辺ケーパビリティセット，権限を変更し得るシステムコールに setns システムコールと unshare システムコールを追加する．また，提案手法は，システムコールハンドラ内の高速パスおよび低速パス両方のシステムコールサービスルーチン呼び出しにおいてプロセスの権限の変更内容を監視する．

提案手法はプロセスの権限の変更内容に着目した手法であるため，システムコール処理中において OS の脆弱性を悪用する権限昇格攻撃であれば，脆弱性の種類に関係なく，攻撃を防止することができる．また，提案手法をあらかじめシステムに導入することにより，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. OS の脆弱性を悪用する権限昇格攻撃の防止手法 [1]

### 2.1 OS の脆弱性

OS は計算機が動作するための基盤の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 の脆弱性は毎年数多く報告されている．2017 年には Linux Kernel で 453 件，Mac OS X で 299 件，Windows 10 で 268 件の脆弱性が報告された [2]．また，OS カーネルのコードは膨大である．2017 年に正式リリースが発表された Linux 4.14 [10] では，カーネルのコード行数が 2,000 万行を超えている [3]．このため，OS の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，脆弱性のある部分を修正するための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パッチの適用が困難な場合がある．例えば，多くの場合，カーネルへ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のあるシステムに対し，新たに脆弱性が見つかるたびに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もつことを前提に，事前にシステムに組み込むことで，OS の脆弱性を悪用する攻撃を防ぐことが可能な機構が必要である．

### 2.2 権限昇格攻撃

OS の脆弱性を悪用する攻撃の一つとして，権限昇格攻撃がある．権限昇格攻撃はユーザが本来与えられていない権限を奪取する攻撃である．この攻撃が成功した場合，攻撃者はより上位の権限をもつユーザとして，システムを操作することができるようになり，結果として，システムは機密情報の漏えいやサービス妨害 (Denial of Service: DoS) を受けることになる．特に，管理者権限の奪取に成功した場合，攻撃者はシステム上にある全ての情報を読み書きできるようになるため，システムは甚大な被害を受ける可能性がある．

権限昇格の脅威の対象にはモバイル端末や IoT 機器も含まれている．モバイル端末のうち 8 割のシェアを獲得している Android [4] においては，管理者権限の奪取は root 化と呼ばれる．多くの場合，Android では端末メーカが独自に開発したアプリケーション（以降，AP）やライブラリなどの知的財産が漏えいする可能性から，ユーザの管理者権限による操作を許可していない．一方で，利便性の確保を目的に，ユーザによって多くの端末が root 化されている [5]．

上記の理由から，権限昇格攻撃は非常に大きな脅威であり，対策を取る必要がある．

### 2.3 権限の変更に着目した権限昇格攻撃防止手法

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 [1] について説明する．

#### 2.3.1 考え方

文献 [1] の手法は，システムコール処理中において，Linux カーネルの脆弱性を悪用する権限昇格攻撃を対象としている．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

© 2018 Information Processing Society of Japan

**表 1** 監視する権限情報（文献 [1] の手法）

| 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 |
| addr_limit | ユーザ空間とカーネル空間の境界アドレス |

(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る際に限られると考えることができる．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．例えば，CVE-2016-0728 [11] を悪用する権限昇格攻撃の例では，keyctl 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．しかし，keyctl システムコールは，本来ならばプロセスの権限を変更するシステムコールではない．そこで，文献 [1] の手法は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更することのない権限が変更され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．

### 2.3.2 監視対象の権限情報

プロセスの権限に関する情報（以降，権限情報）のうち，文献 [1] の手法が監視するものを**表 1** に示す．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．文献 [1] の手法はシステムコール処理の前後においてこれらの情報の変更内容を監視する．

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，ファイルおよび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．

ケーパビリティセットは特定の操作や操作クラスの実行をプロセスに許可するか否かを示すフラグ（ケーパビリティ）の集合であり，ケーパビリティセット群 (cap_inheritable, cap_permitted, cap_effective) に保存される．ケーパビリティの例としては「種々のネットワーク処理を許可する」や「chroot システムコールの実行を許可する」などがある．

**表 2** 権限情報を変更し得るシステムコール（文献 [1] の手法）

| 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, addr_limit |
| setuid | uid, euid, fsuid, suid, cap_inheritable, cap_permitted, cap_effective |
| setreuid | uid, euid, fsuid, suid, cap_inheritable, cap_permitted, cap_effective |
| setresuid | uid, euid, fsuid, suid, cap_inheritable, cap_permitted, cap_effective |
| setfsuid | fsuid, cap_inheritable, cap_permitted, cap_effective |
| setgid | gid, egid, fsgid, sgid |
| setregid | gid, egid, fsgid, sgid |
| setresgid | gid, egid, fsgid, sgid |
| setfsgid | fsgid |
| capset | cap_inheritable, cap_permitted, cap_effective |
| prctl | cap_inheritable, cap_permitted, cap_effective |

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セット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 の値を攻撃者に改ざんされた場合，uid 群，gid 群およびケーパビリティセット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 の値も権限情報として監視する．

権限情報を変更し得るシステムコールを**表 2** に示す．文献 [1] の手法は，それぞれの権限情報が表 2 に含まれない内容で変更された場合，不正な権限情報の変更であると判断する．例えば，表 2 から capset システムコールは cap_inheritable, cap_permitted および cap_effective を変更し得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 cap_inheritable, cap_permitted, および cap_effective 以外の権限情報が変更されていた場合は，権限情報が不正に変更されたと判断する．

### 2.3.3 課題

文献 [1] の手法には，以下の課題が存在する．

**(課題 1) ARM アーキテクチャへの対応**

文献 [1] の手法は，x86–64 アーキテクチャのみで実現しており，ARM アーキテクチャにおける実現はできていない．ARM アーキテクチャは消費電力を抑える特徴を持った CPU アーキテクチャであり，モバイル端末や IoT 機器で広く用いられている．多くのモバイル端末や IoT 機器について，文献 [1] の手法を導入して攻撃を防止する場合，手法を拡張し，ARM アーキ

© 2018 Information Processing Society of Japan

テクチャへ対応させることが必要である．

**(課題 2) 監視対象とする権限情報および権限情報を変更し得るシステムコールの追加**

文献 [1] で調査された Linux 3.10 から表 1 に含まれない権限情報として，周辺ケーパビリティセットが新たに追加されているため，監視対象に追加する必要がある．また，表 2 に含まれない権限情報を変更し得るシステムコールとして，setns システムコールと unshare システムコールの漏れがあったため，これらのシステムコールによる権限情報の変更内容を正常なものとするよう設計に加える必要がある．

**(課題 3) システムコールサービスルーチン呼び出しを漏れなく監視**

文献 [1] の手法は，システムコールサービスルーチン呼び出しの前後で権限情報の変更内容を監視する．しかし，文献 [1] の手法には監視に漏れがあり，監視できていないシステムコールサービスルーチン呼び出しを利用してプロセスの権限が改ざんされた場合，攻撃を防止できない．このため，システムコールサービスルーチン呼び出しを漏れなく監視できるように対策する必要がある．

上記の課題について，本稿の提案手法で対処する．なお，（課題 2）と（課題 3）は x86–64 と ARM の両アーキテクチャで共通する課題である．

## 3. 提案手法の設計

### 3.1 考え方

2.3.3 項で述べた課題の対処法として，文献 [1] の手法を拡張し，ARM 環境に対する権限昇格攻撃を防止する手法を提案する．提案手法はシステムコールの処理中に権限情報を改ざんする権限昇格攻撃を防止することを目的とする．

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はアーキテクチャに依存している．このため，2.3.3 項の（課題 1）の対処として，ARM アーキテクチャ用のシステムコールハンドラである SVC (Supervisor Call) ハンドラ内で行われるシステムコールサービスルーチン呼び出しを**図 1** の処理流れで監視することで，権限昇格攻撃による不正な権限情報の変更を検知することができる．以降では，Linux 4.4 (64bit) を例に，提案手法の設計について述べる．

### 3.2 基本方式

提案手法はシステムコールにおける権限情報の変更内容を監視し，不正な変更を検知することで権限昇格攻撃を防止する．不正な変更とは，それぞれのシステムコール処理において，変更されないはずの権限情報が変更されることである．

提案手法の処理の流れを図 1 に示し，以下で説明する．

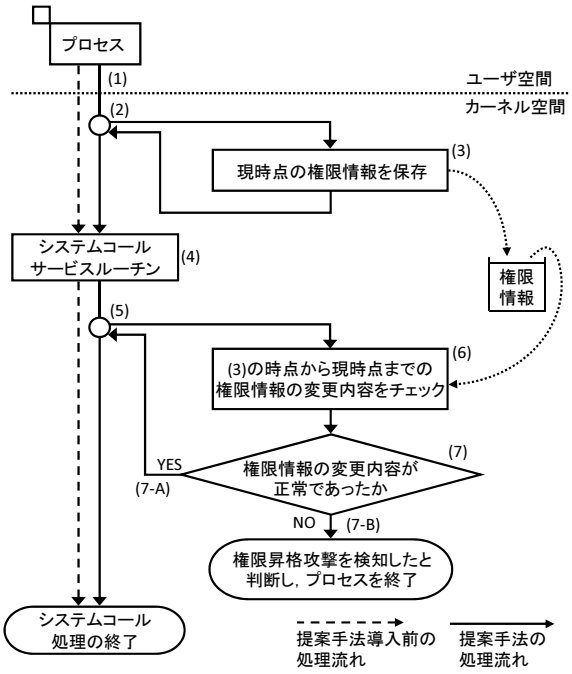


**図 1** 提案手法の処理の流れ

(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

(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

(3) 現時点（システムコール処理前）の権限情報を保存

(4) システムコールサービスルーチンの実行

(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

(6) (3) で保存したシステムコール処理前の権限から現時点までの権限の変更内容（システムコール処理による権限の変更内容）をチェック

(7) システムコール処理による権限の変更内容が正常なものであったかを確認

- (A) 権限の変更内容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
- (B) 権限の変更内容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．また，攻撃を防止したことを示すログを出力

ARM アーキテクチャの場合，プロセスは SVC 命令を使用してカーネル空間へ処理を移行し，ARM アセンブリ言語で記述された SVC ハンドラを呼び出す．SVC ハンドラでは，発行されたシステムコールに対応したシステムコールサービスルーチンが実行される．そこで，(2) と (5) では，提案手法は SVC ハンドラに変更を加えることで，システムコールサービスルーチン呼び出しの前後において処理をフックする．

(6) では，発行されたシステムコールがそれぞれの権限情報を変更し得るか否かをフラグとして管理し，変更し得

© 2018 Information Processing Society of Japan

**表 3** 監視する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）

| 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** |

ない権限情報についてのみ変更内容をチェックする．例えば，ケーパビリティセット群のみを変更し得る capset システムコールが発行された場合，ケーパビリティセット群の権限情報のフラグを無効に，それ以外の権限情報のフラグを有効にした後，フラグが有効の権限情報についてのみ変更内容をチェックする．

### 3.3 監視対象の権限情報

2.3.3 項の（課題 2）の対処として，新たに追加する監視対象の権限情報および権限情報を変更し得るシステムコールについて述べる．この対処は x86–64 と ARM の両アーキテクチャにおいて実施する必要がある．

提案手法は，文献 [1] の手法が監視する権限情報（表 1）に加え，周辺ケーパビリティセット (cap_ambient) を監視する．周辺ケーパビリティセットは Linux 4.3 以降に追加されたケーパビリティセットであり，特権を持たない親プロセスから子プロセスへのケーパビリティの継承を実現する [12]．

提案手法が監視する権限情報と権限情報を変更し得るシステムコールを**表 3** に示す．本研究で追加したものを太字にしている．提案手法は，表 2 のシステムコールに加え，setns システムコールと unshare システムコールを権限情報を変更し得るシステムコールとする．Linux 3.8 以降において，setns システムコールと unshare システムコールは CLONE_NEWUSER フラグを指定して呼び出すことで，setns システムコールは既存のユーザ名前空間，unshare システムコールは新しく作成されたユーザ名前空間のメンバとなり，そのユーザ名前空間のすべてのケーパビリティを獲得する．文献 [1] の手法では，CLONE_NEWUSER フラグが指定された unshare システムコールまたは setns システムコールの処理によるケーパビリティセット群の変更を誤って不正な変更として検知してしまう．このため，提案手法では，これらのシステムコールの処理において，ケーパビリティセット群の変更を正常な内容とし，それ以外の権限情報が変更されていた場合，変更内容が不正であると判断する．

### 3.4 監視するシステムコールサービスルーチン呼び出し

2.3.3 項の（課題 3）の対処として，新たに追加する監視対象の権限情報および権限情報を変更し得るシステムコールについて述べる．この対処についても，3.3 節の対処と同様に，x86–64 と ARM の両アーキテクチャにおいて実施する必要がある．

調査の結果，Linux 4.4（64bit）の場合，システムコールハンドラ内では処理流れが高速パスと低速パスに分岐し，いずれかの流れにおいてシステムコールサービスルーチンが呼び出されることが分かった．そこで，提案手法は，高速パスと低速パス両方のシステムコールサービスルーチン呼び出しにおいて権限情報の変更内容を監視する．

分岐の条件は，プロセスの `thread_info` 構造体の `flags` メンバに特定のフラグが設定されているか否かである．特定のフラグの例としては，システムコールが追跡されているか否かを示す TIF_SYSCALL_TRACE フラグやシステムコールが監査されているか否かを示す TIF_SYSCALL_AUDIT フラグなどがある．特定のフラグが全く設定されていない場合，処理は高速パスに移行し，フラグが 1 つでも設定されている場合，処理は低速パスに移行する．

## 4. 実現方式と評価

### 4.1 実現方式

3 章で述べた提案手法を Raspberry Pi 3 Model B 上で動作する Linux 4.9.80-v8+ (64bit) に実現した．Raspberry Pi 3 Model B は 64 ビット ARM プロセッサを搭載したシングルボードコンピュータである．

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)）のフックは，SVC ハンドラ内の高速パスと低速パスの両方において，システムコールサービスルーチン呼び出しの前後で提案手法の関数を呼び出す処理を含めたマクロを実行することで実現する．マクロには提案手法の関数を呼び出す処理の他に，カーネル空間内に権

© 2018 Information Processing Society of Japan

表 4　Raspberry Pi 3 Model B の構成

| | |
|---|---|
| CPU | Broadcom BCM2837 (4 コア), 1.2GHz |
| メモリ | 1.0 GB |
| OS | Gentoo Linux |
| カーネル | Linux 4.9.80−v8+ (64bit) |
| NIC | オンボード NIC (100BASE−T) |

表 5　システムコールのオーバヘッド（単位: $\mu$s）

| システムコール | 提案手法の導入前 | 提案手法の導入後 | オーバヘッド |
|---|---|---|---|
| stat | 2.994 | 3.542 | 0.548 |
| fstat | 0.537 | 0.952 | 0.415 |
| write | 0.398 | 0.824 | 0.426 |
| read | 0.503 | 0.938 | 0.435 |
| getppid | 0.255 | 0.688 | 0.433 |
| open + close | 4.092 | 4.662 | 0.570 |

限情報を保存する領域を確保する処理や保存した権限情報を取り出す処理を ARM アセンブリ言語で記述している．

権限情報の変更内容のチェック（図 1 の (6)）は，システムコール発行の直後に x8 レジスタに格納されているシステムコール番号を取得し，表 3 の内容と照らし合わせてフラグを設定することで実現する．なお，execve システムコールについては全ての権限情報を変更し得るため，execve システムコールのシステムコール番号を取得した時点でフック関数を終了し，権限の変更内容を監視しない．

### 4.2　評価項目と評価環境

提案手法の導入により，計算機の性能への影響があると予測される．そこで，提案手法の導入前および導入後の環境において，以下の 2 項目を測定し，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評価する．

(1) システムコールのオーバヘッド
(2) AP 性能のオーバヘッド

評価環境の Raspberry Pi 3 Model B の構成を**表 4** に示す．評価は，提案手法の導入前と導入後の Linux カーネルをそれぞれ Raspberry Pi 3 Model B 上で動作させて行った．提案手法を導入した環境において権限昇格攻撃が検知できるか否かの評価については，今後の課題とする．

### 4.3　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，OS のマイクロベンチマークスイートである LMbench 3.0 [13] 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 5** に示す．なお，open + close の測定結果について，オーバヘッドを 2 で割った値をシステムコール 1 回当たりのオーバヘッドとして示している．表 5 より，システムコール 1 回当たりのオーバヘッドは，0.415$\mu$s〜0.570$\mu$s であり，システムコールの違いによるオーバヘッドの差は最大でも 0.155$\mu$s と小さいことが分かる．

表 6　クライアント側の環境

| | |
|---|---|
| CPU | Intel Core i7-6700, 3.40GHz（4 コア） |
| メモリ | 8.0 GB |
| OS | Ubuntu 16.04.3 LTS |
| カーネル | Linux 4.4 (64bit) |
| NIC | Intel(R) Ethernet Connection (2) I219−V (1000BASE−T) |

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

| ファイルサイズ | 提案手法の導入前 | 提案手法の導入後 | オーバヘッド |
|---|---|---|---|
| 1KB | 1.753 | 1.777 | 0.024 (1.4%) |
| 10KB | 2.428 | 2.456 | 0.028 (1.2%) |

提案手法は，全ての権限情報を変更し得る execve システムコールを除く全てのシステムコールに対し，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更内容のチェックの処理を同様に追加する．このため，execve システムコール以外のシステムコールでは，1 回当たりに発生するオーバヘッドはほぼ一定になると考えることができる．したがって，表 5 の結果は妥当であると推察できる．

### 4.4　AP 性能のオーバヘッド

AP 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 2.4.33 である．評価では，ApacheBench 2.3 [14] を用いて，通信速度 100Mbps，帯域幅 100MHz の通信路において，1KB と 10KB 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 で示した環境である．クライアント側の環境を**表 6** に示す．なお，リクエストを送信する際の同時接続数は 1 とする．

測定結果を**表 7** に示す．表 7 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ヘッドの割合は，最大 1.4% であり，比較的小さいことがわかる．また，サーバ側の環境で 1 リクエスト当たりに発行されるシステムコール回数を測定した結果，27 回であった．システムコール 1 回当たりのオーバヘッドがほぼ一定である場合，1 リクエスト当たりのオーバヘッドは，1 リクエスト当たりのシステムコール発行回数に比例して大きくなる．したがって，表 5 の結果より，1 リクエスト（27 回）当たりのオーバヘッドは 11.21$\mu$s〜15.39$\mu$s と算出できる．この値は表 7 のオーバヘッドと比較して半分程度である．これは，lat_syscall が各システムコールを連続して発行していることが原因で，通常の AP が発行するシステムコールと比較して，システムコール処理時間が短くなることにより，表 5 のオーバヘッドも小さくなっているからであると推察できる．

以上の結果より，提案手法の導入による AP 性能の影響

© 2018 Information Processing Society of Japan

は小さいことが分かる．

## 5. おわりに

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 [1] を拡張し，ARM 環境に対する権限昇格攻撃を防止する手法について述べた．提案手法は ARM アーキテクチャ用システムコールハンドラ内のシステムコールサービスルーチン呼び出しの前後においてプロセスの権限の変更内容を監視することで，システムコール処理中に行われる権限昇格攻撃を防止する．また，提案手法は，監視対象とする権限情報に周辺ケーパビリティセット，権限情報を変更し得るシステムコールに setns システムコールと unshare システムコールを追加する．さらに，提案手法はシステムコールハンドラ内の高速パスおよび低速パス両方のシステムコールサービスルーチン呼び出しにおいて，プロセスの権限の変更内容を監視する．

提案手法の導入による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の環境において性能測定を実施し，測定結果を比較した．LMbench で測定した結果，システムコール 1 回当たりのオーバヘッドは，0.415$\mu$s〜0.570$\mu$s 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，性能低下は最大 1.4%であり，提案手法の導入による AP への性能の影響は小さいことを示した．

### 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，情報処理学会シンポジウムシリーズ コンピュータセキュリティシンポジウム 2016 (CSS2016) 論文集，vol.2016，no.2，pp.542-549 (2016)．
[2] CVE Details: Top 50 products having highest number of cve security vulnerabilities in 2017, available from 〈https://www.cvedetails.com/top-50-products.php?year=2017〉 (accessed 2018–01–22).
[3] Linux Counter, available from 〈https://www.linuxcounter.net/statistics/kernel〉 (accessed 2018–01–19).
[4] Statista: Mobile OS market share 2017, available from 〈https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/〉 (accessed 2018–01–25).
[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] SELinux Wiki: SEforAndroid, available from 〈http://selinuxproject.org/page/SEAndroid〉（accessed 2018–06–25）．
[7] NSA/CSS: Security-Enhanced Linux, available from 〈https://www.nsa.gov/what-we-do/research/selinux/〉 (accessed 2018–06–20).
[8] Arm: TrustZone, available from, 〈https://www.arm.com/products/security-on-arm/trustzone〉 (accessed 2018–01–25).
[9] 磯崎 宏，金井 遵：ハードウェアセキュリティ機能を利用した長期安全性の確保が可能な組込みシステム，情報処理学会論文誌，Vol.56，No.8，pp.1604–1620(2015)．
[10] LKML: Linus Torvalds: Linux 4.14, available from 〈https://lkml.org/lkml/2017/11/12/123〉 (accessed 2018–01–19).
[11] Exploit DB: Linux Kernel 4.4.1 - REFCOUNT Overflow/Use-After-Free in Keyrings Privilege Escalation (1), available from 〈https://www.exploit-db.com/exploits/39277/〉 (accessed 2018–01–23).
[12] LWN.net: capabilities: Ambient capabilities, available from 〈https://lwn.net/Articles/636533/〉 (accessed 2018–01–17).
[13] LMbench - Tools for Performance Analysis, available from 〈http://www.bitmover.com/lmbench/〉 (accessed 2018–02–05).
[14] ab - Apache HTTP server benchmarking tool - Apache HTTP Server Version 2.4, available from 〈https://httpd.apache.org/docs/2.4/programs/ab.html〉 (accessed 2018–02–06).

© 2018 Information Processing Society of Japan