Computer Security Symposium 2018
22 - 25 October 2018

権限昇格攻撃防止手法における権限の格納位置のランダム化

吉谷 亮汰[1]　山内 利宏[1,a)]

概要：我々は，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対し，システムコールによるプロセスの権限の変更に着目して攻撃を防止する手法を提案した．提案手法は，システムコール処理の前後においてプロセスの権限の変更内容を監視し，そのシステムコールが本来変更し得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できる．一方で，提案手法は，システムコール処理前において，プロセスの権限をカーネルスタック内の特定の位置に格納して保存するため，カーネルスタック内の権限の位置は攻撃者に推測されやすい．攻撃者がシステムコール処理中にプロセスの権限とカーネルスタック内の権限の両方を改ざんした場合，提案手法は攻撃を防止できない可能性がある．そこで，本稿では，格納位置をランダム化することで，攻撃者にカーネルスタック内の権限の改ざんを困難にする提案手法について述べる．また，性能評価した結果を報告する．

キーワード：OS，権限昇格攻撃，システムコール，ランダム化

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

1. はじめに

各種オペレーティングシステム (以降，OS) において，毎年数多くの脆弱性が報告されている [1]．OS のコード量は膨大であり [2]，すべての脆弱性を取り除くことは困難である．また，発見された脆弱性を取り除くには，脆弱性のある部分を修正するためのパッチを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．これらの理由から，計算機が動作するための基盤である OS の信頼を確保するためには，報告された脆弱性を修正するだけでは不十分であり，未修正の脆弱性を悪用する攻撃を防ぐことが可能な機構を事前にシステムに組み込む必要がある．

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．この攻撃では，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．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も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可能性があるため，権限昇格攻撃への対策は重要である．

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 [3]．提案手法は，システムコール処理の前後において，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得ないもの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．提案手法は，システムコール処理中に権限情報を改ざんする攻撃であれば，悪用される脆弱性の種類に関わらず，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，文献 [4] では，提案手法を拡張し，モバイル端末や IoT 機器で広く用いられる ARM アーキテクチャに対応させた．さらに，新たな権限情報の追加や権限情報を変更し得るシステムコールの漏れ，システムコールサービスルーチン呼び出しの監視の漏れの問題について対処した．

しかし，提案手法では，システムコール処理前において，プロセスの権限情報をカーネルスタック内の特定の位置に

[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

格納して保存する．このため，カーネルスタック内の権限情報の位置は攻撃者に推測されやすい．攻撃者がシステムコール処理中にプロセスの権限情報とカーネルスタック内の権限情報の両方を改ざんした場合，システムコール処理による権限情報の変更内容のチェックが正しく行われず，提案手法は攻撃を防止できない可能性がある．

そこで，本稿では，格納位置をランダム化することで攻撃者にカーネルスタック内の権限の改ざんを困難にする方式を提案する．本稿では，従来方式の問題点について述べ，提案する権限昇格攻撃防止手法における権限情報の格納位置をランダム化する手法の設計，実現方式，および評価結果について述べる．

2. OS の脆弱性を悪用する権限昇格攻撃の防止手法 [3] [4]

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 [1]．また，OS カーネルのコードは膨大である．例えば，Linux カーネルの場合，2017 年に正式リリースが発表された Linux 4.14 [5] では，カーネルのコード行数は 2,000 万行を超えている [2]．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やデバイスの特性により，パッチのダウンロードや適用が困難な場合がある．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．また，アプリケーション（以降，AP）等の動作検証が必要となり，パッチを即座に適用できない場合もある．

上記の理由から，OS が脆弱性を持つことを前提に，事前にシステムに組み込むことで，修正パッチを適用することなく，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，ユーザや AP が本来与えられている権限より上位の権限を不正に奪取する攻撃である．攻撃が成功した場合，攻撃者はより上位の権限を持つユーザとして，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．

モバイル端末においても権限昇格の脅威とされている．モバイル端末のうち 8 割のシェアを獲得している Android [6] 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 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 [7]．

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

文献 [3] [4] で提案し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説明する．

2.3.1 考え方

提案手法は，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
(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，keyctl システムコールにおける脆弱性 CVE-2016-0728 [8] 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，提案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．

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．

2.3.2 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

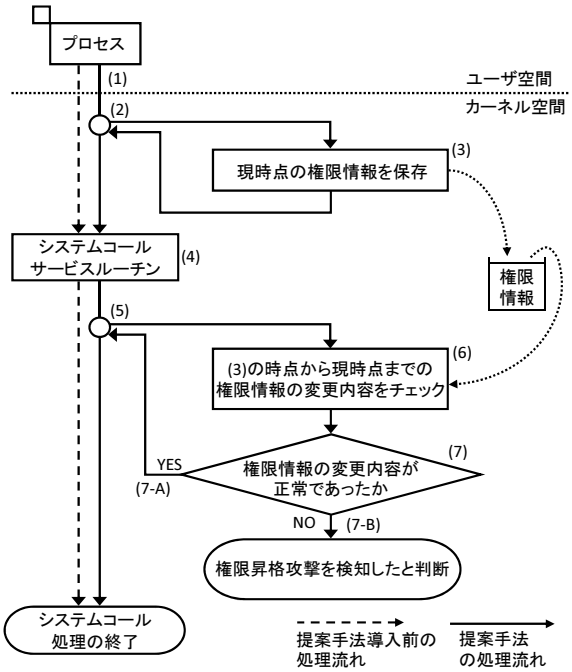

図 1 提案手法の処理の流れ

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．

(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情報から現時点までの権限情報の変更内容（システムコール処理による権限情報の変更内容）をチェック
(7) システムコール処理による権限情報の変更内容が正常なものであったかを確認
  (A) 権限情報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) 権限情報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

文献 [4] の ARM アーキテクチャにおける方式の場合，(2) と (5) は SVC (Supervisor Call) 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フックして提案手法の処理へ移行する．ARM アーキテクチャでは，プロセスはシステムコール発行時に SVC 命令を使用してカーネル空間へ処理を移行し，SVC ハンドラを実行する．

表 1 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

2.3.3 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

表 1 に提案手法が監視対象とする権限情報を示す．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．提案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, cap_ambient) 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．

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．

表 2 に，表 1 で示した権限情報を変更し得るシステムコールを示す．提案手法は，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

表 2　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

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

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．

2.3.4　監視対象のサービスルーチン呼び出し

文献 [4] で実施したシステムコールハンドラの調査の結果，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，いずれかのパス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．

2.3.5　問題点

提案手法は，システムコールサービスルーチン実行の直前においてプロセスの権限情報をカーネルスタック内に格納して保存し，サービスルーチン実行直後の権限情報の変更内容のチェックに使用する．権限情報は，サービスルーチンがカーネルスタックに格納するローカル変数や引数，戻りアドレスなどの値によって上書きされないように，サービスルーチン実行直前のスタックポインタの値から固定のオフセット値（以降，提案手法のオフセット）を減算して得られるアドレスに格納される．このため，攻撃者が提案手法のオフセットを知ることができる場合，権限情報の格納位置は容易に攻撃者に推測されてしまう．攻撃者が権限情報の格納位置を特定できた場合，OS の脆弱性を用いてシステムコール処理中にプロセスの権限情報とカーネルスタック内の権限情報の両方を改ざんするコードを実行することで，システムコール処理による権限情報の変更内容のチェックが正しく行われず，提案手法は権限昇格攻撃を防止できない可能性がある．このため，このような攻撃を防止するには，格納位置の推測を困難にすることが有効である．

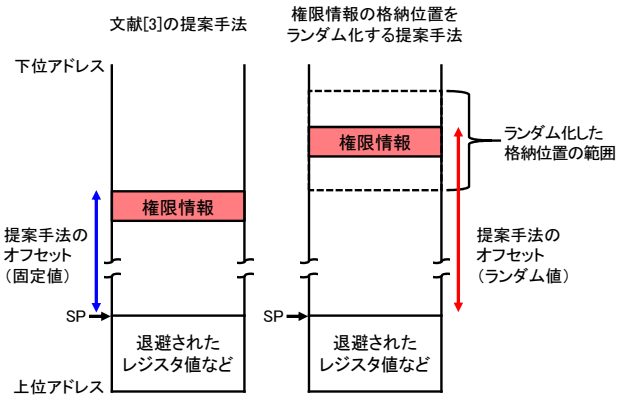

図 2　システムコールサービスルーチン実行直前のカーネルスタック

3.　権限情報の格納位置をランダム化する提案手法の設計

3.1　考え方

2.3.5 項で述べた問題に対処するために，本稿では，権限情報の格納位置をランダム化する手法を提案する．文献 [3] [4] の提案手法と権限情報の格納位置をランダム化する提案手法それぞれの場合におけるシステムコールサービスルーチン実行直前のカーネルスタックを図 2 に示す．2.3.5 項で述べたように，提案手法は，サービスルーチン実行直前のスタックポインタ（SP）の値から提案手法のオフセットを減算して得られるアドレスに権限情報を格納する．文献 [3] では提案手法のオフセットが固定値であるため，カーネルスタック内における権限情報の格納位置は固定される．

そこで，提案手法のオフセットをランダムな値にすることで，カーネルスタック内における権限情報の格納位置をランダム化し，攻撃者が格納位置を推測することを困難にできる．

3.2 課題

権限情報の格納位置をランダム化するには，攻撃者が推測困難であるランダムな値を取得する必要がある．また，格納された権限情報はシステムコールサービスルーチン実行直後に権限情報の変更内容のチェックに使用される．このため，サービスルーチン実行直後において，ランダム化に利用した値を再度利用して権限情報の格納位置を特定する必要がある．したがって，取得するランダムな値は，サービスルーチン実行前後で同じ値でなければならない．また，ランダムな値を攻撃者に簡単に改ざんされないことも必要である．

3.3 対処

提案手法の処理でランダムな値を生成して，利用することが考えらえる．しかし，サービスルーチン実行前後で参照するために，ランダムな値を格納する必要があり，その格納位置が決まっていれば，それを参照するコードを作成することは難しくない．このため，攻撃者が推測困難な別のランダム値を利用する必要がある．

そこで，提案手法は，OS カーネルのセキュリティ機能である KASLR (Kernel ASLR) から得られるランダムな値を利用する．KASLR は，プロセスのアドレス空間レイアウトをランダム化手法である ASLR(Address Space Layout Randomization) をカーネルに適用したものであり，ブート毎にカーネルがロードされるアドレスをランダム化する．KASLR が有効である場合，攻撃者は，カーネルの脆弱性を悪用してユーザ空間から任意のカーネル内の値や関数を使用する際に，それらのアドレスを推測しなければならない．

3.4 基本方式

権限情報の格納位置をランダム化する提案手法の処理の流れを**図 3** に示し，以下で説明する．[3] の提案手法の処理の流れからの変更点として，権限情報の格納位置を決定するために (3) と (4) の処理を追加している．また，システムコールサービスルーチン実行直後において，ランダム化された格納位置から権限情報を取り出すために (8) と (9) の処理を追加している．

(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
(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) カーネルから得られるランダムな値を取得
(4) ランダムな値を利用してカーネルスタック内における権限情報の格納位置を決定
(5) 決定した格納位置に現時点（システムコール処理前）の権限情報を格納して保存
(6) システムコールサービスルーチンの実行

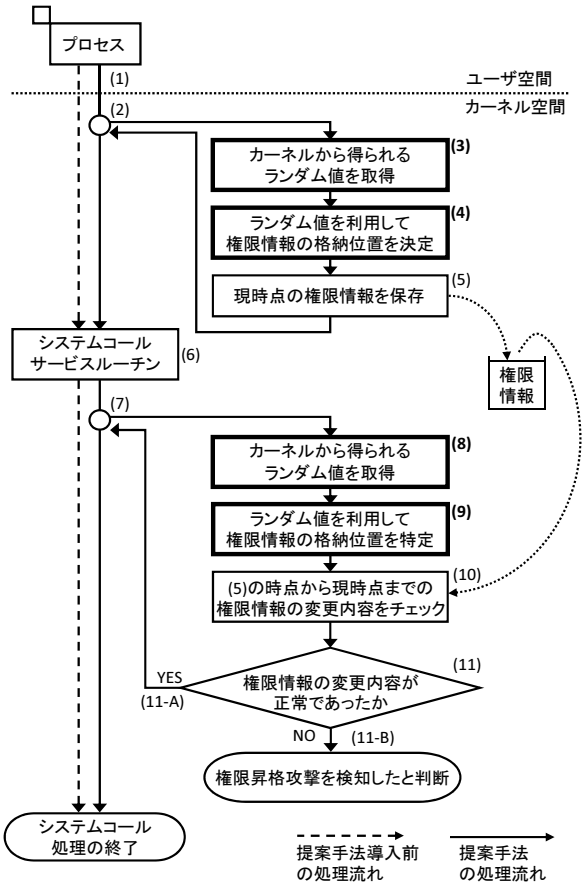

図 3 権限情報の格納位置をランダム化する提案手法の処理の流れ

(7) 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
(8) カーネルから得られるランダムな値（(3) で得られる値と同じ値）を取得
(9) ランダムな値を利用して (5) で保存した権限情報の格納位置を特定
(10) (5) の時点から現時点までの権限情報の変更内容（システムコール処理による権限情報の変更内容）をチェック
(11) 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

4. 実現方式と評価

4.1 実現方式

3 章で述べた権限情報の格納位置をランダム化した提案手法を Linux 4.4（x86–64）に実現した．また，文献 [4] で ARM アーキテクチャでのみ実現していた監視対象の権限情報への cap_ambient の追加，権限情報を変更し得るシス

表 3　システムコールの処理時間（単位: μs）

システムコール	導入前 (t1)	導入後		オーバヘッド	
		ランダム化無 (t2)	ランダム化有 (t3)	ランダム化無 (t2−t1)	ランダム化有 (t3−t1)
stat	0.446	0.489	0.488	0.043	0.042
fstat	0.136	0.172	0.171	0.036	0.035
write	0.107	0.143	0.143	0.036	0.036
read	0.139	0.177	0.177	0.038	0.038
getppid	0.073	0.109	0.109	0.036	0.036
open + close	0.467	0.505	0.501	0.038	0.034

表 4　Apache の 1 リクエスト当たりの処理時間（単位: ms）

ファイル サイズ (KB)	導入前 (T1)	導入後		オーバヘッド	
		ランダム化無 (T2)	ランダム化有 (T3)	ランダム化無 (T2−T1)	ランダム化有 (T3−T1)
1	1.105	1.111	1.122	0.006 (0.5%)	0.017 (1.5%)
10	1.405	1.377	1.401	-0.028 (-2.0%)	-0.004 (-0.3%)
100	3.273	3.386	3.346	0.113 (3.5%)	0.073 (2.2%)

テムコールへの setns システムコールと unshare システムコールの追加，および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における権限情報の変更内容の監視を新たに x86–64 アーキテクチャに実現した．

Linux 4.4 (x86–64) の場合，カーネル関数 kaslr_offset() の戻り値から KASLR のオフセット（ランダム化によって各カーネルシンボルのアドレスに加算される値）が得られる．オフセットをビット列として見た場合，実際にランダム化されているのはビット列の一部のみである．この部分ビット列が示す値は，オフセットをカーネルの物理アドレスのアライメントの値 PHYSICAL_ALIGN で割ることで得られる．Linux 4.4 (x86–64) では，デフォルトの設定で 6 ビットのエントロピーがある．

4.2　評価内容

提案手法に権限情報の格納位置のランダム化の処理を追加したことによる性能への影響を評価した．具体的には，提案手法の導入前，権限情報の格納位置をランダム化しない従来方式の提案手法の導入後，および権限情報の格納位置をランダム化する提案手法の導入後において以下の 2 項目を測定した．測定結果を比較することにより，権限情報の格納位置のランダム化による性能への影響を評価した．

(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

評価環境には，CPU が Intel Core i7-6700（3.40GHz，4 コア），メモリが 8.0GB，カーネルが Linux 4.4.0 (64bit) の計算機を使用した．

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 [9] の lat_syscall を用いてシステムコールの処理時間を測定した．

システムコール処理時間の測定結果を**表 3** に示す．なお，open + close の測定結果は，測定値を 2 で割った値をシステムコール 1 回当たりの処理時間として示している．

表 3 より，権限情報の格納位置をランダム化する提案手法の場合，ランダム化の有無によるシステムコール 1 回当たりのオーバヘッドの差は 0.001μs 未満であり，小さいことがわかる．権限情報の格納位置のランダム化の処理は，ランダム値の取得およびランダム値を用いた格納位置の決定と特定のみであり，負荷の高い処理はない．

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，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 2.4.18 である．評価では，ApacheBench 2.3 [10] 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．クライアント側の環境には，CPU が Intel Core i5-3470（3.20GHz，4 コア），メモリが 4.0GB，カーネルが Linux 3.10.0 (64bit) の計算機を使用した．また，リクエストを送信する際の同時接続数は 1 とした．

Apache の 1 リクエスト当たりの処理時間の測定結果を**表 4** に示す．

表 4 より，権限情報の格納位置をランダム化する提案手法の場合，1 リクエスト当たりのオーバヘッドの割合は，ランダム化しない場合と比較して 2%未満の差であることがわかる．権限情報の格納位置のランダム化の処理は，監視対象のサービスルーチン呼び出しを行う全てのシステムコールに対して追加される．このため，ランダム化による

AP 性能のオーバヘッドは，AP のシステムコール発行回数に比例して大きくなる．一方で，4.3 節の結果より，ランダム化によるシステムコールのオーバヘッドは 0.001μs 未満と小さいことから，AP 性能に与える影響は小さいと推察できる．

以上の結果より，権限情報の格納位置のランダム化による AP 性能への影響は小さいことがわかる．

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 [3] [4] について，手法がカーネルスタック内に格納する権限情報の格納位置をランダム化する手法を提案し，設計，実現方式，および性能評価の結果について述べた．提案手法は，KASLR を利用してブート毎に権限情報の格納位置をランダム化することで，攻撃者による格納位置の推測を困難にし，ランダム値の改ざんも困難にしている．これにより，OS の脆弱性を悪用した攻撃により，プロセスの権限と提案手法が格納する情報の両方の改ざんを困難にすることができる．

権限情報の格納位置をランダム化した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，権限情報の格納位置をランダム化しない提案手法の導入後，およびランダム化する提案手法の導入後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，権限情報の格納位置をランダム化する提案手法の場合，ランダム化の有無によるオーバヘッドの差は 0.001μs 未満と小さいことを示した．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．この結果，権限情報の格納位置をランダム化する提案手法の場合，1 リクエスト当たりのオーバヘッドの割合は，ランダム化しない場合と比較して 2%未満の差であることから，権限情報の格納位置のランダム化による AP 性能への影響は小さいことを示した．

参考文献

[1] 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 2018–08–07).

[2] Linux Counter, available from 〈https://www.linuxcounter.net/statistics/kernel〉 (accessed 2018–08–07).

[3] 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 (CSS2016) 論文集，vol.2016，no.2，pp.542-549 (2016).

[4] 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).

[5] LKML: Linus Torvalds: Linux 4.14, available from 〈https://lkml.org/lkml/2017/11/12/123〉 (accessed 2018–01–19).

[6] 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).

[7] 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).

[8] 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).

[9] LMbench - Tools for Performance Analysis, available from 〈http://www.bitmover.com/lmbench/〉 (accessed 2018–02–05).

[10] 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).