権限昇格攻撃防止手法における ARM TrustZoneを利用した権限の保護

吉谷 亮汰[1]　山内 利宏[1]

概要：オペレーティングシステムの脆弱性を悪用し，システムコール処理中にプロセスの権限を改ざんすることで，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．このような権限昇格攻撃に対し，我々は，システムコール処理による権限の変更内容を監視し，攻撃を防止する手法（以降，従来手法）を提案した．従来手法はシステムコール処理前にプロセスの権限をカーネルスタック内に格納して保存することで，権限の変更内容の監視を実現する．しかし，攻撃者がカーネルスタック内の権限の格納位置を特定し，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，従来手法は回避されてしまう．本稿では，この課題に対処するために，ARM TrustZoneを利用し，プロセスの権限を保護する手法を提案する．提案手法は，TrustZone が提供するセキュア領域にプロセスの権限に関する情報を保存することで，非セキュア領域の OS カーネル空間で攻撃コードが実行されたとしても，改ざんを防止できる．本稿では，提案手法の設計と実現方式を述べ，攻撃耐性や性能を評価した結果を報告する．

キーワード：オペレーティングシステム，権限昇格攻撃，TrustZone，Trusted Execution Environment

Protection of Privileges Using ARM TrustZone in Privilege Escalation Attack Prevention Method

Ryota Yoshitani[1]　Toshihiro Yamauchi[1]

Abstract: An attacker can control the system with higher privileges than originally granted by exploiting operating system vulnerabilities and altering process privileges during system call processing. To prevent such attacks, we have proposed a method by monitoring the change of privileges by system call processing. The previous method implements monitoring of the change in the process privileges by storing the privileges in the kernel stack before the system call processing. However, if an attacker identifies the storage location of privileges in the kernel stack and alters both the process privileges and the privileges in the kernel stack during system call processing, the previous method is bypassed. In this paper, in order to deal with this problem, we propose a method to protect the process privileges by using ARM TrustZone. The proposed method can prevent tampering even if an attack code is executed in the OS kernel space of the non-secure region by storing the information related to process privileges in the secure region provided by TrustZone. In this paper, we describe the design and implementation of the proposed method, and report the evaluation results of attack resistance and performance.

1. はじめに

多くのオペレーティングシステム（以降，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/C++などの低水準言語で実装されている [1,2]．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．さらに，システムの運用形態や機器の特性によって，脆弱性の修正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 の信頼性を確保す

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

るためには，脆弱性が報告されるたびに修正を行う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．

攻撃者は，OS カーネルの脆弱性を悪用するコードを実行することで，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を実施する可能性がある．この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，OS 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（以降，従来手法）を提案した [4]．従来手法は，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することで，攻撃を防止できる．従来手法は，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かかわらず，権限昇格攻撃を防止することが可能である．したがって，従来手法をあらかじめシステムに導入しておくことにより，OS カーネルの脆弱性を修正することなく，権限昇格攻撃を防止することができる．

一方で，従来手法では，権限情報の変更内容のチェックのために，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に格納して保存する．攻撃者がカーネルスタック内の権限情報の位置を特定し，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，権限情報の変更内容のチェックは正しく行われず，従来手法は回避されてしまう．

そこで，本稿では，ARM プロセッサのセキュリティ機能である TrustZone を利用し，従来手法で保存する権限情報を保護する手法（以降，提案手法）を提案する．TrustZone はメモリ領域を分離することで，安全なアプリケーション（以降，AP）実行環境のためのセキュアな領域をシステムに提供することが可能である．このセキュアな領域に権限情報を保存することで，提案手法が保存する権限情報を攻撃者による改ざんから保護することができる．本稿では，従来手法の課題について述べ，提案手法の設計，実現方式について述べ，攻撃耐性や性能を評価した結果について報告する．

2. OS の脆弱性を悪用する権限昇格攻撃の防止手法 [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．一方で，多くの OS カーネルは C/C++言語で実装されている．これらのような低水準言語は，メモリ管理に関するバグが混入しやすい．また，OS カーネルのコードは膨大である．Linux カーネルの場合，コード行数は増え続けており，2018 年 9 月の時点で 2,500 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．たとえば，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，アプリケーションなど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，修正パッチを適用できない場合でも，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

攻撃者は，OS の脆弱性を悪用することで，システムに対して権限昇格攻撃を実施する可能性がある．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も高い権限を不正に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持つユーザとしてシステムを操作することが可能になる．この結果，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，攻撃者はシステム全体を操作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攻撃は脅威とされている．Android 端末においては，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 [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，対策の必要がある．

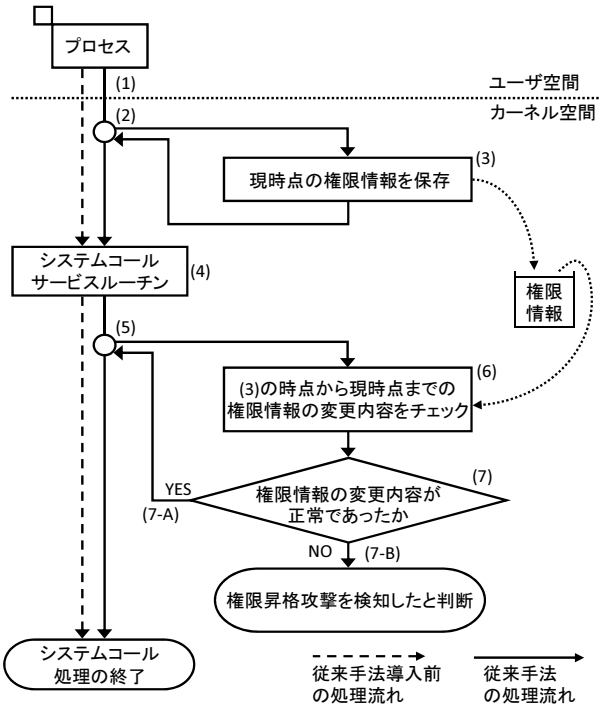

図 1　従来手法の処理の流れ

2.3　権限の変更に着目した権限昇格攻撃防止手法

従来手法であ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 [4] の考え方，基本方式，および課題について述べる．

2.3.1　考え方

従来手法は，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．一方で，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．たとえば，keyctl システムコールにおける脆弱性 CVE-2016-0728 を悪用して権限昇格攻撃を達成するエクスプロイトコード [6] 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，従来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　基本方式

従来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，正常でない変更を検知することで権限昇格攻撃を防止する．正常な変更とは，それぞれのシステムコール処理において，そのシステムコールが変更し得る権限情報のみが変更されることである．

従来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（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
(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，従来手法の処理へ移行
(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差分（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であるかを確認
  (A) 権限情報の変更内容が正常である場合，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正常でない場合，権限昇格攻撃が行われたと判断．また，攻撃を防止したことを示すログを出力

2.3.3　従来手法の課題

従来手法は，システムコール処理前に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する．カーネルスタックは，各プロセスに割り当てられる領域である．このため，システムコール処理後において，カーネルスタックから取得した権限情報がどのプロセスのものかを識別する必要はない．一方で，攻撃者がカーネルスタック内の権限情報の格納位置を特定し，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，権限情報の変更内容のチェックが正しく行われず，従来手法は回避されてしまう．

文献 [7] では，この問題への対処として，権限情報の格納位置をランダム化し，攻撃者による格納位置の推測を困難にする手法を提案している．しかし，カーネルスタックのサイズは，ランダム化によって情報を隠すには不十分である．また，ランダム化に利用する乱数は，従来手法によるシステムコールの処理の前後で保持されている必要がある．この値がシステムコール処理中に攻撃者に参照または改ざんされることで，権限情報の格納位置を特定または操作され，従来手法は回避されてしまう．

上記の理由から，従来手法を回避されないようにするには，攻撃者がカーネル空間内のデータの読み書きが可能な

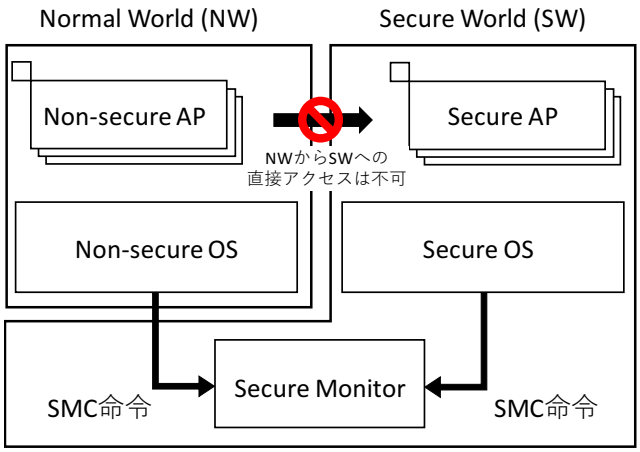

図 2　TrustZone の全体像

場合でも，従来手法が保存する権限情報を改ざんできないようにする必要がある．

3. TrustZone

TrustZone は，ARM プロセッサが提供するセキュリティ機能であり，システムに安全な AP 実行環境（TEE: Trusted Execution Environment）を提供することができる．本稿における TrustZone は，スマートフォン・タブレット向けのアーキテクチャである Cortex-A 系アーキテクチャの ARM プロセッサにおける TrustZone を指す．

TrustZone の全体像を図 2 に示す．TrustZone により，メモリ領域は非セキュアな領域（NW: Normal World）とセキュアな領域（SW: Secure World）に分離される．非セキュア領域では，Linux や Android などの汎用的な OS (Non-secure OS) と AP (Non-secure AP) が動作する．これに対し，セキュア領域では，GlobalPlatform [8] により標準化された TEE 仕様の OS（Secure OS）と AP（Secure AP）が動作する．

セキュア領域のデータやコードには，非セキュア領域から直接アクセスすることはできない．このため，攻撃者が非セキュア領域の AP や OS に対する攻撃に成功した場合でも，セキュア領域のコードやデータを攻撃者による漏えいや改ざんから守ることができる．この仕組みにより，セキュア AP はデジタル著作権の管理（DRM: Digital Rights Management）や鍵管理，認証処理など，高い秘匿性が求められる処理を行うことができる．

非セキュア領域からセキュア領域のデータやコードにアクセスするには，セキュアモニタ（Secure Monitor）による領域の切り替えを行う必要がある．セキュアモニタは，セキュア領域の OS や非セキュア領域の OS よりも高い特権レベルで動作しており，OS から SMC（Secure Monitor Call）命令を使用することで，セキュアモニタへの遷移が可能である．セキュアモニタは，セキュア構成レジスタ（SCR: Secure Configuration Register）が持つ NS ビット（Non-Secure bit）を操作し，切り替え先の領域を決定する．

4. 提案手法の設計

4.1 考え方

2.3.3 項で述べた従来手法の課題に対処するために，本稿では，TrustZone が提供するセキュア領域を利用して権限情報を保護する手法を提案する．攻撃者が非セキュア領域の Linux カーネルの脆弱性を悪用することで，カーネル空間内の情報の改ざんが可能な場合でも，セキュア領域に直接アクセスすることはできない．

そこで，提案手法は，非セキュア領域のプロセスが発行するシステムコール処理前において，プロセスの権限情報をセキュア領域に保存する．また，システムコール処理後において，権限情報の変更内容のチェックをセキュア領域で行う．これにより，提案手法が保存する権限情報を攻撃者による改ざんから保護することができる．

4.2 課題

提案手法がセキュア領域においてプロセスの権限情報を保護するには，以下の課題に対処する必要がある．

（課題 1）　セキュア領域の処理への移行

非セキュア領域で動作する Linux カーネルからは，セキュア領域のコードやデータに直接アクセスすることはできない．このため，権限情報の保存・チェック処理をセキュア領域で行う場合，Linux カーネルからセキュア領域へ処理を移行する方法について検討する必要がある．

（課題 2）　保存される権限情報の識別

2.3.3 項で述べたとおり，従来手法は，各プロセスに割り当てられる領域であるカーネルスタックに権限情報を保存するため，カーネルスタック内の権限情報がどのプロセスのものかを識別する必要はない．しかし，提案手法では，カーネルスタックを利用しないため，セキュア領域に保存された権限情報がそれぞれ非セキュア領域のどのプロセスのものかを識別し，権限情報の変更内容のチェックの際に，対応する権限情報を取り出す必要がある．

4.3 対処

4.3.1 セキュア領域の処理への移行

TEE が実現された環境において，非セキュア領域の AP は，セキュア領域の AP に対する接続（セッション）を確立することで，その AP との通信が可能になり，処理を要求することができる．

そこで，提案手法では，権限情報の保存・チェック処理のための AP（以降，提案手法の AP）をセキュア領域で起動しておき，Linux カーネルと提案手法の AP との間でセッションを確立する．これにより，システムコール処理

の前後において，それぞれ権限情報の保存・チェック処理を提案手法の AP に対して要求し，セキュア領域へ処理を移行することができる．

4.3.2 保存される権限情報の識別

提案手法では，プロセスを識別できる値を利用し，保存される権限情報を識別する．具体的には，提案手法は，Linux カーネルのプロセスのシステムコール処理前において，プロセスを識別できる値として PID を取得し，権限情報とともにセキュア領域に保存する．システムコール処理後に行う権限情報の変更内容のチェックの際は，その時点での非セキュア領域のプロセスから PID を取得し，セキュア領域に保存された権限情報のうち，取得した PID に対応する権限情報を取り出す．これにより，システムコール処理の前後における権限情報を正しく比較し，変更内容をチェックできる．

4.4 基本方式

提案手法の処理の流れを**図 3** に示し，以下で説明する．

(1) 非セキュア領域のプロセス A 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，セキュア領域の提案手法の処理へ移行
(3) 現時点（システムコール処理前）のプロセス A の権限情報とプロセス A を識別できる値を保存
(4) 非セキュア領域のカーネル空間に処理を移行し，システムコールサービスルーチンを実行
(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，セキュア領域の提案手法の処理へ移行
(6) プロセス A を識別できる値を用いて，対応する権限情報（(3) で保存した権限情報）を取得
(7) (3) の時点から現時点までのプロセス A における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
(8) システムコール処理による権限情報の変更内容が正常であるかを確認
  (A) 権限の変更内容が正常なものである場合，権限昇格攻撃は行われていないと判断し，非セキュア領域の元々の処理流れに戻り，システムコール処理を終了
  (B) 権限の変更内容が正常なものでない場合，権限昇格攻撃が行われたと判断し，(3) で取得した権限情報を利用してプロセス A の権限情報を復元．また，攻撃を検知したことを示すログを出力

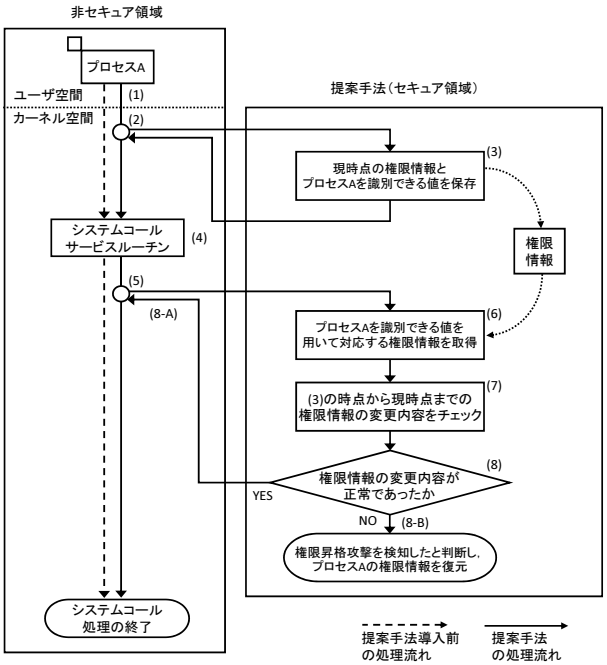

図 3 提案手法の処理の流れ

5. 実現方式

5.1 メモリ領域の分離の実現

64-bit ARM 仮想計算機上の環境において動作する Linux 5.1.0（64-bit）において，4 章で述べた提案手法を実現した．この環境では，TrustZone による TEE の実装を支援するオープンソースプロジェクトである OP-TEE [9] を利用して，メモリ領域の分離を実現している．分離されたメモリ領域のうち，非セキュア領域では Linux，セキュア領域では TEE 仕様の OS（OP-TEE OS）が動作する．

また，OP-TEE は，セキュア領域の AP と非セキュア領域の AP の間で通信を行うための API として，それぞれ TEE Client API と TEE Internal Core API を提供している．TEE Client API は，非セキュア領域の AP がセキュア領域の AP に対して処理を要求するための API である．一方で，TEE Internal Core API は，セキュア領域の AP のみが利用可能である．TEE Internal Core API には，非セキュア領域の AP による要求に応答して処理を行う API や，暗号化などのセキュアな処理を行う API がある．

5.2 実現における課題

提案手法を実現するために，以下の課題に対処する．

（実現課題 1） 提案手法の AP との通信

Linux カーネルから提案手法の AP に対し，セッションを確立する方法を検討する必要がある．また，システムコールサービスルーチン呼び出し前後において，Linux カーネルから提案手法の AP に権限情報の保存・チェック処理を要求する方法を検討する必要がある．

（実現課題 2） 提案手法を有効化する契機

Linux カーネル起動時においては，セキュア・非セキュア領域間で通信ができない．このため，提案手法を有効化する契機について検討する必要がある．

表 1　提案手法が利用する TEE サブシステムの API

	API の種類	API の概要
(1)	tee_client_open_context()	セキュア領域とのコンテキストを生成
(2)	tee_client_open_session()	セキュア領域 AP とのセッションを生成
(3)	tee_shm_alloc()	共有メモリを確保
(4)	tee_shm_free()	共有メモリを解放
(5)	tee_client_invoke_func()	セキュア領域 AP の関数を呼び出し

5.3 提案手法の AP との通信

Linux カーネルのシステムコールサービスルーチン呼び出し前後（図 3 の（2）と（5））における処理のフックは，システムコールハンドラに提案手法の AP との通信のための処理を追加することで実現する．セキュア領域で動作する提案手法の AP と通信するには，TEE Client API を利用することが考えられる．しかし，TEE Client API は非セキュア領域の AP のみが利用できる API であり，Linux カーネルからは利用できない．そこで，提案手法は，Linux カーネルの TEE サブシステムが提供する API を利用し，提案手法の AP と通信する．

提案手法が利用する TEE サブシステムの API を**表 1** に示す．（1）と（2）は，Linux カーネルと提案手法の AP の間での通信を可能にするための API であり，提案手法の有効化の際に使用する．（1）tee_client_open_context() は，Linux カーネルとセキュア領域の間における接続（コンテキスト）を生成する．セキュア領域の AP とのセッションを生成するには，コンテキストが生成されている必要がある．（2）tee_client_open_session() は，Linux カーネルと提案手法の AP の間におけるセッションを生成する．

（3）と（4）はそれぞれ共有メモリブロックを確保・解放するための API である．共有メモリはセキュア・非セキュア領域の両方から読み書き可能な領域であり，データの転送に利用される．提案手法は，（3）tee_shm_alloc() を使用して確保した共有メモリブロックに権限情報を一時的に格納することで，Linux カーネルと提案手法の AP の間における権限情報の受け渡しを行う．確保された共有メモリブロックは，保存処理またはチェック処理が完了後に（4）tee_shm_free() を使用して解放する．

（5）tee_client_invoke_func() は，提案手法の AP 内で定義されている TEE Internal Core API である TA_InvokeCommandEntryPoint() を呼び出す．(5) に引数として，保存・チェック処理関数のいずれかを呼び出すためのフラグ，権限情報を格納した共有メモリブロックの先頭アドレス，および，プロセスを識別できる値として PID を渡す．また，処理されるシステムコールについて，どの権限情報を変更し得るかを示す値についても引数として渡し，チェック処理に利用する．

5.4 提案手法を有効化する契機

セキュア・非セキュア領域間の通信には，通信を制御するためのデバイスドライバ（TEE ドライバ）が利用される．TEE ドライバは，Linux カーネル起動時に初期化される．このため，TEE ドライバの初期化前に Linux カーネルから提案手法の AP に対して通信しようとした場合，エラーが発生する．したがって，TEE ドライバの初期化の完了後に提案手法を有効化し，Linux カーネルと提案手法の AP の間の通信を開始する必要がある．

この課題への対処として，Linux カーネルに提案機能を有効化するためのシステムコールを追加する．また，追加したシステムコールを発行して，Linux カーネル内のパラメータを変更することで，提案手法を有効化するコマンドを作成する．このコマンドを/etc/inittab に登録することで，Linux カーネル起動後に自動的にコマンドが実行され，提案手法を有効化することができる．

6. 評価

6.1 評価項目

提案手法の有用性と提案手法の導入による性能への影響を明確にするために，以下の評価を実施した．

（評価 1）　攻撃耐性の比較
従来手法と提案手法において，手法が保存する権限情報を改ざんする攻撃への耐性を比較し，提案手法の有用性を示す．

（評価 2）　性能測定
提案手法の導入前の環境および導入後の環境におけるシステムコールの処理時間を測定し，結果を比較することで，提案手法の導入による性能への影響を評価する．

6.2 攻撃耐性の比較

権限情報をカーネルスタックに保存する従来手法では，カーネルスタック内の権限情報はプロセスの権限情報と同様に改ざんを受ける可能性がある．一方で，提案手法では，セキュア領域で権限情報を保護している．非セキュア領域の OS カーネルからはセキュア領域のアドレスのマッピングはできないため，セキュア領域へ直接アクセスすることはできない．このため，攻撃者は非セキュア領域において OS カーネルの脆弱性を悪用することで任意のアドレスの読み書きが可能であっても，セキュア領域における権限情報の格納位置を特定および改ざんすることは困難である．

したがって，提案手法は従来手法と比較し，手法に対する攻撃への高い耐性をもつことが推察できる．

6.3 性能測定

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，提案手法導入前の環境および導入後の環境におけるシ

表 2　システムコールの処理時間（単位: ms）

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
stat	0.566	2.094	1.527
fstat	0.143	1.500	1.357
write	0.004	1.150	1.147
read	0.004	1.140	1.136
getppid	0.003	1.141	1.138
open + close	0.399	1.845	1.446

ステムコールの処理時間を測定し，結果を比較した．評価は，非セキュア領域で Linux 5.1.0 (64-bit)，セキュア領域で OP-TEE OS が動作する 64-bit ARM 仮想計算機上で実施した．仮想化ソフトウェアには QEMU 3.0.93 を使用し，CPU アーキテクチャは ARM Cortex-A57，コア数は 2，メモリは 1024MB を指定している．また，処理時間の測定には，OS のマイクロベンチマークスイートである LMbench 3.0 [10] の lat_syscall を使用した．

システムコール処理時間の測定結果を**表 2** 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 2 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，1.136ms〜1.527ms であるとわかる．このうち，stat と open + close は，I/O 処理による影響で他のシステムコールと比べて処理時間が大きくなっていることが推察できる．

提案手法は，非セキュア領域の Linux カーネルにおいて，システムコール処理の前後をフックし，権限情報の保存・チェック処理を追加する．これらの処理は，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．このため，システムコール 1 回あ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 2 の結果は妥当であると推察できる．また，上記のことから，実アプリケーションにおいて生じるオーバヘッドは，システムコール発行回数に比例するといえる．

7. 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 [4] は，システムコール処理中にプロセスの権限情報と手法がカーネルスタック内に保存する権限情報の両方を改ざんする攻撃を検知できない．この課題への対処として，ARM プロセッサのセキュリティ機能である TrustZone を利用して保護する手法を提案し，設計，実現方式，および評価結果について述べた．

提案手法は，TrustZone によって提供される安全な AP 実行環境のためのセキュアなメモリ領域に権限情報を保存する．攻撃者が非セキュアなメモリ領域で動作する OS の脆弱性を悪用することで，カーネル空間内の情報の改ざんが可能な場合であっても，セキュアなメモリ領域に保存された権限情報を改ざんすることは困難である．

実現した提案手法の評価として，はじめに，手法に対する攻撃への耐性について提案手法と従来手法を比較した．比較より，非セキュア領域の OS から，セキュア領域のメモリをマッピングできないため，セキュア領域の権限を改ざんすることは難しいことを示した．

次に，提案手法の導入による性能への影響を評価するために，提案手法の導入前の環境と導入後の環境において，システムコールの処理時間を測定し，比較した結果，提案手法の導入によるシステムコール 1 回あたりのオーバヘッドは 1.136ms〜1.527ms であった．システムコール処理 1 回当たりのオーバヘッドは，1ms 強である．実アプリケーションでのオーバヘッドは，システムコール処理 1 回当たりのオーバヘッドをシステムコール発行回数で乗じた時間になると推察できる．このことから，システムコールを多く発行するアプリケーションでなければ，オーバヘッドは大きくならないと推察できる．

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] Criswell, J., Dautenhahn, N. and Adve, V.: KCoFI: Complete Control-Flow Integrity for Commodity Operating System Kernels, in Proceedings of IEEE S&P ’14, pp.292–307 (2014).

[2] Szekeres, L., Payer, M., Wei, T., Song, D.: SoK: Eternal War in Memory, in Proceedings of IEEE S&P ’13, pp.48–62 (2013).

[3] Phoronix: The Linux Kernel Has Grown By 225k Lines of Code So Far This Year From 3.3k Developers，入手先 ⟨https://www.phoronix.com/scan.php?page=news_item&px=Linux-September-2018-Stats⟩（参照 2019-08-08）.

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

[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] Exploit DB: Linux Kernel 4.4.1 - REFCOUNT Overflow/Use-After-Free in Keyrings Privilege Escalation (1)，入手先 ⟨https://www.exploit-db.com/exploits/39277/⟩ (参照 2019–08–08) .

[7] 吉谷亮汰，山内利宏：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，コンピュータセキュリティシンポジウム 2018 (CSS2018) 論文集，vol.2018，no.2，pp.964-970 (2018).

[8] GlobalPlatform，入手先 ⟨https://globalplatform.org/⟩（参照 2019-08-07）.

[9] Linaro: Open Portable Trusted Execution Environment - OP-TEE，入手先 ⟨https://www.op-tee.org/⟩（参照 2019-08-18）.

[10] LMbench - Tools for Performance Analysis，入手先

⟨http://www.bitmover.com/lmbench/⟩ (参照 2019-08-10).