情報処理学会論文誌 Vol.61 No.9 1531–1541 (Sep. 2020)

推薦論文

64-bit ARM環境における権限の変更に着目した権限昇格攻撃防止手法

吉谷 亮汰[1] 山内 利宏[1,a)]

受付日 2019年12月9日, 採録日 2020年6月1日

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は，システムに甚大な被害を与える可能性がある．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た権限昇格攻撃防止手法（AKO：Additional Kernel Observer）を提案した．AKOは，プロセスの権限情報の変更内容をチェックし，システムコール処理前に権限情報をカーネルスタック内に格納して保存することで，権限情報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，AKOによる攻撃防止は回避されてしまう．本論文では，モバイル端末やIoT機器などの保護を目的とし，64-bit ARM環境においてAKOと同様に権限昇格攻撃を防止する手法を提案する．また，AKOの問題への対処として，ARMプロセッサの機能であるTrustZoneを利用し，提案手法が保存する権限情報を保護する．本論文では，提案手法の設計，実現方式，および評価結果について述べる．評価では，エクスプロイトコードによる権限昇格攻撃の検知実験，およびシステムコールとアプリケーションの実測による性能測定を実施した．この結果，攻撃検知による防御が成功する一方，本手法による性能低下は限定的であり，本手法の有効性が示された．

キーワード：権限昇格攻撃，システムコール，ARM，TrustZone

Privilege Escalation Attack Prevention Method by Focusing on Privilege Changes on 64-bit ARM

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

Received: December 9, 2019, Accepted: June 1, 2020

Abstract: Privilege escalation attacks that exploit operating system vulnerabilities can cause significant damage to the associated systems. We previously proposed an additional kernel observer (AKO), a prevention method that focuses on modifying process privileges by system calls. AKO verifies the modification in process privilege data and monitors the modification in privilege data by storing them in the kernel stack before processing the system call. However, if an attacker identifies the storage location of privilege data in the kernel stack and alters both process privilege data and privilege data in the kernel stack while the system call is being processed, AKO can be bypassed. Hence, in this paper, we propose a new method for preventing privilege escalation attacks in the 64-bit ARM environment as well as AKO for protecting mobile devices and IoT devices. To address the issues of AKO, the new method protects the stored privilege data employing the ARM TrustZone technology. In this paper, the new method's design, implementation, and evaluation results are described. In the evaluation, we performed a privilege escalation attack detection experiment using an exploit code and measured performance of system calls and applications. The evaluation results showed that protection by attack detection was successful and the performance degradation due to this method was limited.

Keywords: privilege escalation attack, system call, ARM, TrustZone

[1] 岡山大学大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University, Okayama 700–8530, Japan
[a)] yamauchi@cs.okayama-u.ac.jp

本論文の内容は2018年7月の第82回CSEC研究発表会で報告され，同研究会主査により情報処理学会論文誌ジャーナルへの掲載が推薦された論文である．

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．また，OS のコード量は膨大である．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 [1]．しかし，これらの特徴を持つにもかかわらず，多くの OS カーネルは C/C++ 言語などの低水準言語で実装されている．低水準言語を用いたソフトウェア開発では，メモリ管理に関するバグが混入しやすい [2], [3]．このため，すべての脆弱性を取り除くことは困難であり，毎年数多くの脆弱性が報告されている [4]．

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，AP 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，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．

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．これは，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の妨害を受けることになる．

モバイル端末や IoT 機器においても，権限昇格攻撃への対策は重要である．モバイル端末の場合，第三者による端末への攻撃のほか，利便性の確保を目的に端末ユーザが管理者権限を取得する行為が問題とされている．多くの場合，端末メーカは端末のセキュリティの低下や個人情報および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．Android 端末では，管理者権限の奪取は root 化と呼ばれ，OS の脆弱性を利用したツールにより，多くの端末が root 化されている [5]．

また，IoT 機器では，2016 年以降，IoT 機器を対象とした Mirai やその亜種などのマルウェアによる DDoS 攻撃の被害が発生している．発見当初の Mirai は，製品出荷時点の認証情報を利用していた．しかし，2017 年以降には，既知の脆弱性を悪用する亜種が報告されている [6], [7]．このことから，今後マルウェアが IoT 機器の管理者権限を奪取するために，OS の脆弱性を悪用した権限昇格攻撃を実施する可能性が考えられる．

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た権限昇格攻撃防止手法（AKO：Additional Kernel Observer）を提案した [8]．AKO は，システムコール処理の前後をフックし，権限に関する情報（以降，権限情報）の変更内容を監視する．AKO 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，AKO 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．

一方で，AKO 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，AKO による攻撃防止は回避されてしまう．

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．そこで，本論文では，モバイル端末や IoT 機器などの保護を目的とし，64-bit ARM 環境において AKO と同様に権限昇格攻撃を防止する手法を提案する．提案手法は，カーネルスタックに保存した権限情報の改ざん可能性への対処として，ARM プロセッサのセキュリティ機能である TrustZone を利用し，提案手法が保存する権限情報を保護する．TrustZone は，メモリ領域を分離し，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．

本論文では，提案手法の設計，実現方式について述べ，提案手法の有効性と性能に与える影響を評価した結果を報告する．

2. AKO [8]

2.1 AKO の設計

本章では，文献 [8] で提案した AKO について説明する．

AKO 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の実行に限られる．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 [9] 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，AKO 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．

AKO 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
(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
(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，元々の処理の流れに戻り，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

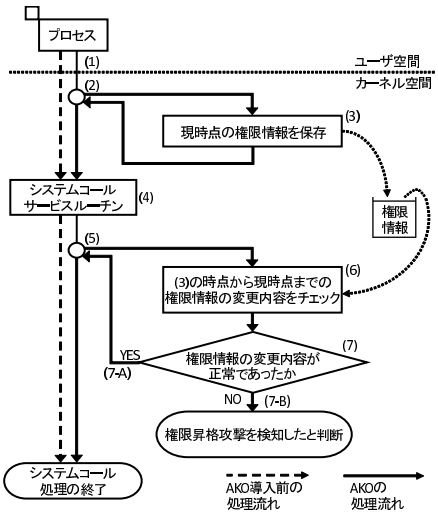

図 1 AKO の処理の流れ
Fig. 1 Process flow of the AKO.

権限情報の正常な変更とは，各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うる権限情報のみが変更されることである．

2.2 AKO の問題

AKO 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，AKO による攻撃防止は回避されてしまう．

文献 [10] 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，AKO 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，AKO による攻撃防止は回避されてしまう．

上記の理由から，AKO による攻撃防止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場合でも，AKO 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．本論文では，この問題への対処として，ARM プロセッサが提供するセキュリティ機能である TrustZone を利用して権限情報を保護する手法（以降，AKO-ARM+TZ）を提案する．

3. AKO-ARM+TZ の設計

3.1 考え方

ARM 環境においては，ARM プロセッサが提供するセキュリティ機能である TrustZone が利用可能である．TrustZone は，システムに安全な AP 実行環境（TEE：Trusted Execution Environment）を提供する．この機能により，メモリ領域は非セキュアな領域とセキュアな領域に分離される．非セキュア領域では，Linux や Android などの汎用的な OS と AP が動作する．これに対し，セキュア領域では，GlobalPlatform [11] により標準化された TEE 仕様の OS と AP が動作する．

セキュア領域のコードやデータには，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，セキュア領域の 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．

そこで，AKO-ARM+TZ は，2.2 節で述べた問題の対処として，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．攻撃者が非セキュア領域の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，セキュア領域のメモリをマッピングすることはできない．このため，AKO-ARM+TZ 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．

3.2 課題

TrustZone を利用し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．これらの対処については，3.3 節で述べる．

（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

AKO-ARM+TZ では，カーネルスタックを利用しない．このため，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．

3.3 対処

3.3.1 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, AKO-ARM+TZ では, 権限情報の保存･チェック処理のための AP（以降, AKO-ARM+TZ の AP）をセキュア領域で起動しておき, Linux カーネルと AKO-ARM+TZ 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 AKO-ARM+TZ 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．

3.3.2 保存される権限情報の識別

AKO-ARM+TZ 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，AKO-ARM+TZ 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．

3.4 基本方式

AKO-ARM+TZ の処理の流れを**図 2** 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，セキュア領域へ処理を移行
(3) (2) のフック時点（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) 非セキュア領域へ処理を移行し，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，セキュア領域へ処理を移行
(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

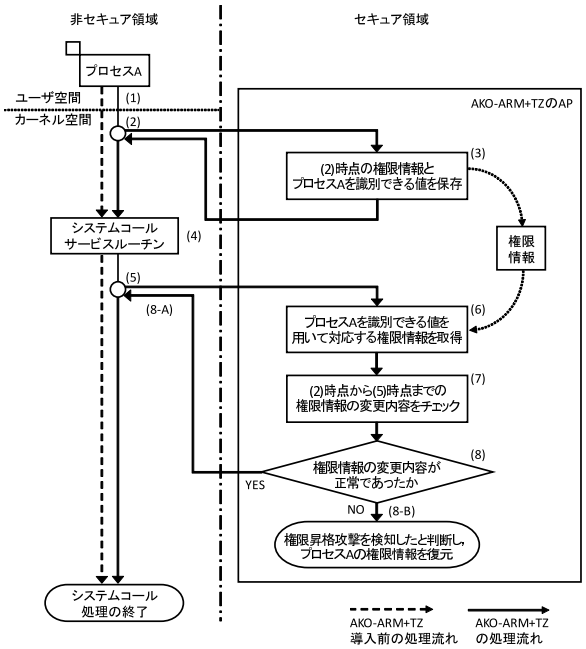

図 2 AKO-ARM+TZ の処理の流れ
Fig. 2 Process flow of the AKO-ARM+TZ.

(7) (2) のフック時点から (5) のフック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．非セキュア領域へ処理を移行して元々の処理の流れに戻り，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，(2) 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

なお，AKO-ARM+TZ の AP の処理は，システムコールに連動して行われるため，割り込みは非セキュア領域側から行われる．また，AKO-ARM+TZ の処理はシステムコールに関係するため，スケジューリングは，通常プロセス以上の優先度で動作することを想定している．

3.5 監視対象の権限情報

Linux 4.4.0（64-bit）を例に，AKO-ARM+TZ が監視対象とする権限情報および監視対象の権限情報を変更しうるシステムコールについて述べる．

AKO-ARM+TZ が監視対象とする権限情報を**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．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る．これらの値は，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．

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）の集合であるケーパビリティセットは，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る．addr_limit は，プロセスの権限に直接関係する情報ではない．しかし，この値が改ざんされた場合，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，AKO-ARM+TZ では，addr_limit を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うるシステムコールを示す．AKO-ARM+TZ 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，正常でない権限情報の変更であると判断する．たとえば，表 2 の capset システムコールはケーパビリティセット群のみを変更しうるシステムコールである．このため，capset システムコールの前後においてケーパビリティセット群以外の権限情報が変更されていた場合は，正常でない権限情報の変更であると判断する．

表 1 監視対象の権限情報
Table 1 Privilege data monitored by the AKO-ARM+TZ.

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

表 2 権限情報を変更しうるシステムコール
Table 2 System calls that may change privilege data.

システムコール	変更しうる権限情報
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

4. 実現方式

4.1 実現における課題

64-bit ARM 環境上で動作する Linux を対象に，提案手法の実現方式を述べる．提案手法を実現するためには，以下の課題に対処する必要がある．それぞれの対処については，以降の節で述べる．

（実現課題 1） システムコール処理のフック
AKO において，システムコール処理の前後のフックは，システムコールハンドラに変更を加えることで実現している．一方で，システムコールハンドラはアーキテクチャに依存しており，AKO は x86-64 環境でのみ実現している．このため，64-bit ARM 環境におけるシステムコール処理のフックの方法を検討する必要がある．

（実現課題 2） AKO-ARM+TZ の AP との通信
Linux カーネルから AKO-ARM+TZ 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 AKO-ARM+TZ 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 3） AKO-ARM+TZ 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，AKO-ARM+TZ を有効化する契機について検討する必要がある．

上記の課題のうち，(実現課題 1) のみに対処し，TrustZone を利用しない提案手法（以降，AKO-ARM）を Raspberry Pi 3 Model B 上で動作する Linux 4.9.80-v8+（64-bit）に実現した．また，上記の課題すべてに対処し，AKO-ARM+TZ を 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 [12] を利用し，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．

4.2 システムコール処理のフック

64-bit ARM 環境では，システムコール発行直後に実行される SVC（Supervisor Call）ハンドラ内において，発行されたシステムコール番号に対応したシステムコールサービスルーチンが呼び出される．このため，提案手法は，SVC ハンドラに変更を加え，サービスルーチン呼び出しの前後のフック（図 2 の (2) と (5)）を実現する．

Linux のバージョンによっては，ハンドラは ARM アセンブリ言語で記述されている場合がある．この場合，提案手法の関数を呼び出す処理を含めたマクロを追加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が含まれている．

4.3 AKO-ARM+TZ の AP との通信

TrustZone を利用して権限情報を保護する場合，システムコールサービスルーチン呼び出し前後で AKO-ARM+TZ の関数を呼び出す代わりに，AKO-ARM+TZ 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 AKO-ARM+TZ の AP と通信するには，OP-TEE が提供する 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，AKO-ARM+TZ は，Linux カーネルの TEE サブシステム [13] が提供する API を利用し，AKO-ARM+TZ の AP と通信する．

AKO-ARM+TZ が利用する TEE サブシステムの API を**表 3** に示す．(1) と (2) は，Linux カーネルと AKO-ARM+TZ の AP の間での通信を可能にするための API であり，AKO-ARM+TZ 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 AKO-ARM+TZ 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．AKO-ARM+TZ 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 AKO-ARM+TZ 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の完了後に (4) `tee_shm_free()` を使用して解放する．

表 3 AKO-ARM+TZ が利用する TEE サブシステムの API
Table 3 TEE subsystem APIs used by the AKO-ARM+TZ.

	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) tee_client_invoke_func() は，AKO-ARM+TZ 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，保存処理またはチェック処理の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う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．

4.4 AKO-ARM+TZ 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 AKO-ARM+TZ 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 AKO-ARM+TZ を有効化し，Linux カーネルと AKO-ARM+TZ 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 AKO-ARM+TZ 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，AKO-ARM+TZ 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，AKO-ARM+TZ を有効化することができる．

5. 評価

5.1 評価項目

提案手法の有用性と提案手法の導入によって生じ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） 権限昇格攻撃の検知実験

提案手法を導入した環境において，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．

（評価 2） 性能測定

提案手法の導入前および導入後の環境において，性能測定を実施し，結果を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．

5.2 権限昇格攻撃の検知実験

提案手法の有用性を示すために，AKO-ARM を導入した Linux カーネルに，権限昇格攻撃を実施し，攻撃を検知できるか否かを評価した．攻撃には，Web 上で入手できるエクスプロイトコード [9] の一部を改変したものを用いた．このコードは，Linux カーネルの脆弱性 CVE-2016-0728（keyctl システムコール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）を悪用し，権限昇格攻撃を達成するエクスプロイトコードである．なお，Linux カーネルのバージョンごとにそれぞれのエクス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，今回の検知実験では，CVE-2016-0728 の脆弱性を持つ Linux 3.13.0（64-bit ARM）に AKO-ARM を導入し，攻撃を実施した．

実験の結果，AKO-ARM は，エクスプロイトコード [9] による権限昇格攻撃を検知した．具体的には，keyctl のシステムコールサービスルーチンの前後で uid 群，gid 群，およびケーパビリティセット群の権限情報が変更されていることを検知した．

AKO-ARM は，悪用される脆弱性の種類にかかわらず，システムコール処理中におけるプロセスの権限の改ざんを検知できる．したがって，今回の検知実験に用いたエクス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，AKO-ARM は攻撃を検知可能であると推察できる．また，AKO-ARM と AKO-ARM+TZ では，権限情報の保存・チェック処理の内容のみが異なり，システムコール処理のフックポイントや監視する権限情報の種類は同様である．このため，AKO-ARM+TZ も同様に，システムコールを介して不正に権限を奪取する攻撃を検知可能であると推察できる．

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可能であることを示した．

5.3 性能測定

5.3.1 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 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．

まず，Raspberry Pi 3 Model B 上で動作する Linux カーネルにおいて，AKO-ARM の導入前後のシステムコールのオーバヘッドを測定し，結果を比較した．評価環境である Raspberry Pi 3 Model B の構成を**表 4** に示す．

システムコールのオーバヘッド（AKO-ARM）の測定結果を**表 5** 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 回あたりの処理時間として示している．表 5 より，システムコール 1 回あたりのオーバヘッドは，0.415 μs〜0.570 μs 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 μs

表 4 評価環境（AKO-ARM）
Table 4 Environment for evaluation (AKO-ARM).

CPU	Broadcom BCM2837（4 コア），1.2 GHz
メモリ	1 GB
OS	Linux 4.9.80-v8+（64-bit）

表 5 システムコールのオーバヘッド（AKO-ARM）（単位：μs）
Table 5 System call overhead (AKO-ARM) (μs).

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

表 6 評価環境（AKO-ARM+TZ）
Table 6 Environment for evaluation (AKO-ARM+TZ).

CPU	ゲスト	ARM Cortex-A57 CPU（2 コア）
	ホスト	Intel Core i7-6700（4 コア），3.40 GHz
OS	ゲスト	Linux 5.1.0（64-bit）
	ホスト	Linux 4.4.0（64-bit）
メモリ	ゲスト	1 GB
	ホスト	8 GB

表 7 システムコールのオーバヘッド（AKO-ARM+TZ）（単位：ms）
Table 7 System call overhead (AKO-ARM+TZ) (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

である．AKO-ARM の処理は，システムコールの種類に関係なく同じであるため，特定のシステムコールでオーバヘッドが増大することはないと推察できるが，何らかの要因でオーバヘッドの差が生じたと考えられる．

次に，64-bit ARM 仮想計算機上で動作する Linux カーネルにおいて，AKO-ARM+TZ の導入前後のシステムコールのオーバヘッドを測定し，結果を比較した．評価環境を**表 6** に示す．なお，ゲスト側の Linux 5.1.0 は非セキュア領域で動作する OS であり，セキュア領域では OP-TEE OS が動作している．また，仮想化ソフトウェアには QEMU 3.0.93 を使用している．

システムコールのオーバヘッド（AKO-ARM+TZ）の測定結果を**表 7** に示す．なお，表 5 と同様，`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 回あたりの処理時間として示している．表 7 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，1.136 ms〜1.527 ms であると分かる．AKO-ARM+TZ の処理は，システムコールの種類に関係なく同じ処理であるため，特定のシステムコールでオーバヘッドが増大することはないと推察できるが，何らかの要因でオーバヘッドの差が生じたと考えられる．

表 8 クライアント側の環境
Table 8 Environment of client.

CPU	Intel Core i7-6700（4 コア），3.40 GHz
メモリ	8 GB
OS	Linux 4.4.0（64-bit）

提案手法は，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，システムコールの種類によらず同程度であると推察できる．したがって，表 5 と表 7 の結果は妥当であると推察できる．なお，測定に利用した 6 種類のシステムコール以外のものについても，AKO-ARM や AKO-ARM+TZ の導入で追加される処理は，システムコールの種類によらず同じである．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も種類によらず同程度であると推察できる．さらに，上記のことから，実 AP 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．

5.3.2 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，AKO-ARM の導入前と導入後における実 AP の性能を測定し，比較した．

エッジコンピューティングにおいて利用されるルータやゲートウェイ，組み込みサーバなどの IoT 機器では，性能向上のために，汎用 OS 上で複数 AP を同時実行する ARM 環境が構築される傾向がある [14]．また，IoT 機器には，機器の設定や制御，状態確認などを行うための Web ユーザインタフェース（以降，Web UI）を提供するものが多く存在する [15]．これらの理由から，本評価では，ARM プロセッサを搭載したルータやゲートウェイなどの IoT 機器において，Web UI を提供するための Web サーバが動作している状況を想定し，Web サーバの性能を測定した．利用した Web サーバは Apache 2.4.33 である．

評価では，ApacheBench 2.3 を用いて，100 Mbps の通信路において，1 KB と 10 KB のファイルに対し，それぞれ 10 万回アクセスした際の 1 リクエストあ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は**表 8** 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 9** に示す．表 9 より，AKO-ARM の導入による 1 リクエストあ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分かる．また，サーバ側の環境で 1 リクエストあ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

表 9 Apache の 1 リクエストあたりのオーバヘッド（単位：ms）
Table 9 Processing time per request in Apache (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
1 KB	1.753	1.777	0.024（1.4%）
10 KB	2.428	2.456	0.028（1.2%）

あたりのオーバヘッドがほぼ一定である場合，1 リクエストあたりのオーバヘッドは，1 リクエストあ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）あたりのオーバヘッドは 11.21 μs〜15.39 μs と算出できる．この値は表 9 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．

また，AKO-ARM+TZ が実 AP に与えるオーバヘッドについても，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．これらの結果から，システムコールを多く発行する AP でなければ，提案手法が実 AP に与える性能への影響は大きくならないと推察できる．

6. TEE に対する攻撃の可能性の考察

AKO-ARM+TZ が実現した環境では，TEE であるセキュア領域とその外部にある OS は密接に関係する．この結果，攻撃者が AKO-ARM+TZ の機構を悪用することで，TEE の独立性が損なわれる可能性がある．

攻撃の一例として，AKO-ARM+TZ の AP のリソースを枯渇させる攻撃が考えられる．AKO-ARM+TZ の AP は，起動時に権限情報を保存するための領域を構造体配列として確保する．この領域は，システムコール処理中のプロセスのみに使用され，配列の各エントリに 1 プロセス分の権限情報（84 バイト）が保存される．また，権限情報のチェック処理が終わり次第，使用された権限情報を持つエントリは解放される．このため，攻撃者がスレッドを起動し，待ち状態に入るシステムコールを発行した後，そのスレッドを強制終了することを繰り返した場合，権限情報の保存に必要な領域が増大してしまう．このような攻撃を受けた場合，セキュア領域の保存領域が枯渇し，権限情報の監視が妨害されてしまう可能性がある．

AKO-ARM+TZ の安全性の検証や攻撃への対処については，残された課題とする．

7. 関連研究

TrustZone を利用して機密データやシステムを保護する研究として，文献 [16], [17], [18] がある．

文献 [16] では，モバイル端末や IoT 機器の AP が持つ機密データを保護するための手法として，Ginseng が提案されている．Ginseng は，OS が脆弱性を持ち，攻撃者に制御されてしまうことを前提に，OS より高い権限で動作するソフトウェアを利用し，機密データの保護や機密データを扱う処理の整合性の検証を実現する．しかし，Ginseng を利用して AP が持つ機密データを保護するには，開発者が AP のソースコードを修正し，機密データを保持する変数について，Ginseng の保護対象であるという情報を付与する必要がある．一方，AKO-ARM+TZ は，導入時において，非セキュア領域の OS のソースコードを一部修正し，AKO-ARM+TZ の AP のソースコードをセキュア領域の AP として OP-TEE のコンパイル対象に含めるのみであり，監視する権限情報を指定するために，非セキュア領域の AP のソースコードを修正する必要はない．

文献 [17] では，組み込みシステム向けのセキュリティ技術として，LiSTEE が提案されている．LiSTEE は，暗号鍵などの機密性の高いデータやそれらを取り扱う処理などを保護するために，TrustZone を利用して保護対象の処理をその他の汎用的な処理と分離する．これにより，機密性の高いデータや処理を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ている．一方，AKO-ARM+TZ は，同様に TrustZone を利用するものの，セキュア領域の AP により，非セキュア領域の OS に対する権限昇格攻撃を防止し，非セキュア領域の処理を監視する点で異なる．

文献 [18] では，汎用 OS とリアルタイム OS のハイブリッド OS システムにおいて，OS 間の独立性を高め，システムの高信頼化を実現するためのソフトウェアモジュールである SafeG が提案されている．SafeG を用いたハイブリッド OS 方式により，リアルタイム OS が実行する制御系機能は，汎用 OS が実行する情報系機能から保護され，結果として，システム全体の信頼性とリアルタイム性を確保できる．一方，AKO-ARM+TZ は，セキュア領域の AP と非セキュア領域の OS の連携によってシステムの保護を実現している．このため，セキュア領域と非セキュア領域の間の独立性は低下するものの，AKO-ARM と比べ，システム全体のセキュリティを高めることができる．

OS やブートローダなどの不正な改変により，TEE 上の機密データが漏えいする可能性がある．このような攻撃の対策として，システム起動時において OS やブートローダの完全性を検証するセキュアブート技術の利用がある．文献 [19] では，ARM 環境で動作する Android で利用可能なセキュアブートが提案されている．Android 端末においては，ユーザが AP をインストールしても，システム領域は変更されない．このため，システム領域が本来あるべき状態から変更されている場合，root 化を行うマルウェアの感染やユーザ自らによる不正な改造であると判断できる．しかし，セキュアブートによる検証はシステム起動時のみで

ある．一方で，AKO-ARM+TZ は，システム起動後に，マルウェアやユーザ自らによる不正な権限昇格を防止できる．

OS のセキュリティを強化する研究として，Security-Enhanced Linux（SELinux）[20] がある．SELinux は，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する．また，SELinux を Android 用に拡張した Security-Enhanced Android（SEAndroid）[21] が Android 4.3 から導入されている．一方で，ポリシの範囲内での被害は発生することや，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．

文献 [8] では，AKO は拡張の例として，SELinux に対する攻撃を防止するための設計と実現について述べている．提案手法についても，同様の設計を ARM 環境上で実現することで，SELinux に対する攻撃を防止できる．

8. おわりに

モバイル端末や IoT 機器などの保護を目的とし，64-bit ARM 環境において，システムコールによるプロセスの権限の変更に着目し，権限昇格攻撃を防止する手法を提案した．提案手法は，TrustZone によって提供されるセキュアなメモリ領域を利用し，保存したプロセスの権限情報を保護できる．

エクスプロイトコードを利用した権限昇格攻撃の検知実験を実施した結果から，提案手法を導入することにより，攻撃を検知できることを示した．また，提案手法の導入による性能への影響の評価結果から，実 AP でのオーバヘッドは，システムコール処理 1 回あ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ことを示した．このことから，システムコールを多く発行する AP でなければ，提案手法が性能に与える影響は大きくならないと推察できる．

残された課題として，セキュア領域において，負荷が大きくなった場合や，提案手法の処理中にハードウェア割り込みやコンテキストスイッチが発生した場合の挙動についての検証，およびシステムコールの種類ごとのオーバヘッドに差が生じる要因の調査がある．

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

[2] Criswell, J., Dautenhahn, N. and Adve, V.: KCoFI: Complete Control-Flow Integrity for Commodity Operating System Kernels, *Proc. IEEE S&P '14*, pp.292–307 (2014).

[3] Szekeres, L., Payer, M., Wei, T. and Song, D.: SoK: Eternal War in Memory, *Proc. IEEE S&P '13*, pp.48–62 (2013).

[4] CVE Details: Top 50 products having highest number of CVE security vulnerabilities, available from ⟨https://www.cvedetails.com/top-50-products.php⟩ (accessed 2019-11-13).

[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] Krebs on Security: Reaper: Calm Before the IoT Security Storm?, available from ⟨https://krebsonsecurity.com/2017/10/reaper-calm-before-the-iot-security-storm/⟩ (accessed 2019-11-08).

[7] New Mirai Variant Adds 8 New Exploits, Targets Additional IoT Devices, available from ⟨https://unit42.paloaltonetworks.com/new-mirai-variant-adds-8-new-exploits-targets-additional-iot-devices/⟩ (accessed 2019-11-08).

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

[9] 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 2019-08-08).

[10] 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).

[11] GlobalPlatform, available from ⟨https://globalplatform.org/⟩ available from ⟨2019-08-07⟩.

[12] Linaro: Open Portable Trusted Execution Environment - OP-TEE, available from ⟨https://www.op-tee.org/⟩ (accessed 2019-08-18).

[13] TEE subsystem, available from ⟨https://www.kernel.org/doc/Documentation/tee.txt⟩ (accessed 2019-11-25).

[14] Guan, L., Liu, P., Xing, X., et al.: TrustShadow: Secure Execution of Unmodified Applications with ARM TrustZone, *Proc. 15th Annual International Conference on Mobile Systems, Applications, and Services*, pp.488–501 (2017).

[15] 藤田 彬，江澤優太，田宮和樹，中山 颯，鉄 顕，吉岡克成，松本 勉：特定の IoT 機器の WebUI を狙ったサイバー攻撃の分析，情報処理学会論文誌，Vol.61, No.3, pp.695–706 (2020).

[16] Yun, M.H. and Zhong, L.: Ginseng: Keeping Secrets in Registers When you Distrust the Operating System, *Proc. Network and Distributed System Security Symposium* (*NDSS 2019*), pp.1–15 (2019).

[17] 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).

[18] 中嶋健一郎，本田晋也，手嶋茂晴，高田広章：セキュリティ支援ハードウェアによるハイブリッド OS システムの高信頼化，電子情報通信学会論文誌 D，Vol.J93-D, No.2, pp.75–85 (2010).

[19] 竹森敬祐，川端秀明，磯原隆将，窪田 歩：Android(ARM)+TMP によるセキュアブート，2013 年暗号と情報セキュリティシンポジウム（SCIS 2013）論文集，電子媒体 (2013).

[20] NSA/CSS: Security-Enhanced Linux, available from

⟨https://www.nsa.gov/what-we-do/research/selinux/⟩ (accessed 2019-11-14).
[21] Smalley, S. and Craig, R.: Security Enhanced (SE) Android: Bringing Flexible MAC to Android, *Proc. Network and Distributed System Security Symposium* (*NDSS 2013*), pp.1–18 (2013).

推薦文

IoT セキュリティが社会的な問題として議論されるなか，モバイル端末や IoT 機器などに広く使われている ARM プロセッサの TrustZone 機能を利用した権限昇格攻撃に着目し，その検知・阻止手法を提案している．システムコール性能およびアプリケーションの実測による検証の結果，攻撃検知による防御が成立する一方，本機能による性能低下は極めて限定的なことが確認されているなど，有効性および実用性の観点でも優れている．

（コンピュータセキュリティ研究会主査　寺田 雅之）

吉谷 亮汰　（正会員）

2018 年岡山大学工学部情報系学科卒業．2020 年同大大学院自然科学研究科博士前期課程修了．コンピュータセキュリティに興味を持つ．

山内 利宏　（正会員）

1998 年九州大学工学部情報工学科卒業．2000 年同大学大学院システム情報科学研究科修士課程修了．2002 年同大学院システム情報科学府博士後期課程修了．2001 年日本学術振興会特別研究員（DC2）．2002 年九州大学大学院システム情報科学研究院助手．2005 年岡山大学大学院自然科学研究科助教授．現在，同准教授．博士（工学）．オペレーティングシステム，コンピュータセキュリティに興味を持つ．2010 年度 JIP Outstanding Paper Award，2012 年度情報処理学会論文賞等受賞．電子情報通信学会，ACM，USENIX，IEEE 各会員．本会シニア会員．