Vol.2022-DPS-190 No.14
Vol.2022-CSEC-96 No.14
2022/3/10

カーネルにおけるMemory Protection Keyを用いたカーネルデータ保護機構の拡張性検討と性能評価

葛野 弘樹[1]　山内 利宏[2]

概要：オペレーティングシステムカーネルへの攻撃として，カーネル脆弱性を利用したメモリ破壊によるカーネルデータの改ざんが知られている．メモリ破壊により，特権昇格攻撃やセキュリティ機能の無効化が可能とされている．カーネルへの攻撃困難化として，KCoFI によるカーネルコードの実行順序の検査，KASLR によるカーネルコード，およびカーネルデータの仮想アドレス配置のランダム化がある．しかし，これらのセキュリティ機構では，カーネルデータへの書込みは禁止されない．カーネル脆弱性の利用に成功した場合，カーネルデータは依然として改ざん可能であり，特権昇格やセキュリティ機能が無効化される可能性は存在する．我々はカーネルにおいて特定のカーネルデータを保護するセキュリティ機構を提案しており，Memory Protection Key （MPK）を利用し，カーネルデータへの書込み制限制御を可能としている．本稿では，先に提案したセキュリティ機構によるユーザプロセスの権限情報の保護機能の整理，ならびに，提案しているセキュリティ機構の拡張性を検討し，書き込み制限対象領域の細粒度化と負荷低減のための機能について考察を進めた．提案方式は，MPK を利用可能なエミュレータ上の Linux にて実現しており，権限情報保護機能の性能評価として，システムコール呼出しに対して 2.96% から 9.01% のオーバヘッドであること，ならびに MPK 操作にかかる性能負荷を計測した．

1. まえがき

オペレーティングシステム（OS）カーネルのに対する攻撃として，特権昇格攻撃とセキュリティ機能の無効化攻撃が課題とされている．これらの攻撃は，攻撃者のユーザプロセスが，脆弱性を含むカーネルコード（以降，カーネル脆弱性）を利用し，メモリ破壊を起こし，権限情報の改ざん，ならびにセキュリティ機能に関するカーネルデータを改ざんする攻撃である．特権昇格攻撃に加え，セキュリティ機能の無効化攻撃の事例では，アクセス制御に関するカーネルデータを改ざんの後，管理者権限に昇格し，制限の回避が試みられる [1,2].

カーネル脆弱性を介した攻撃の防止手法として，Kernel Address Space Layout Randomization（KASLR）は，カーネルコードやカーネルデータの仮想アドレスをランダム化し，攻撃時の仮想アドレスの特定を困難化した [3]．また，不正コード呼出し制限のため，KCoFI では，コード呼出し順を検査する Control Flow Integrity（CFI）[4] をカーネルに適用可能とした [5].

既存の手法により，カーネルにおける，カーネル脆弱性を介したカーネルデータ改ざん，ならびに不正コード呼出しは軽減される．しかし，攻撃が成功した場合，権限情報やセキュリティ機能に関するカーネルデータは改ざん可能であり，以下の課題があると考えている．

課題　：**メモリ破壊によるカーネルデータの改ざん**
カーネルモードにおいて，カーネルデータの書込み制限は行われない．カーネル脆弱性を利用し，メモリ破壊に成功する場合，攻撃者は権限情報やセキュリティ機能に関するカーネルデータを書替え可能である．従来手法では，セキュリティ機能に関するカーネルデータの書込み制限の制御は行われないことから，改ざん防止は困難である．

我々は， 動作中のカーネルにおいて，特定のカーネルデータを保護するためのセキュリティ機構を提案しており [6]，**図 1** に概要を示す．先行提案のセキュリティ機構においては，Intel Memory Protection Key（MPK）Protection Keys for Supervisor（PKS）[7,8] を用いて，カーネルデータへの書込み制限を行う．PKS はカーネルモードに対し，メモリへの読書き可否を制御する Intel CPU の機能である．2022 年 1 月時点ではハードウェアは提供されておらず，QEMU 環境でのみ PKS を利用できる [9].

[1] 神戸大学 大学院工学研究科
Department of Electrical and Electronic Engineering, Graduate School of Engineering, Kobe University
[2] 岡山大学 学術研究院自然科学学域
Graduate School of Natural Science and Technology, Okayama University

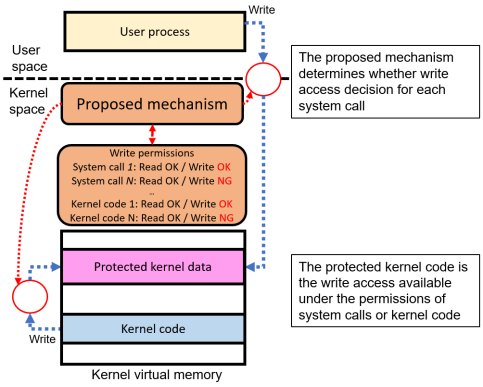

図 1 提案するセキュリティ機構の概要

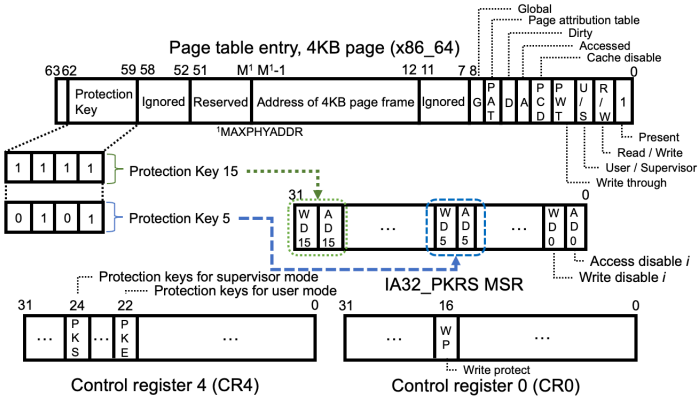

図 2 Intel Memory Protection Key の概要 [7]

提案しているセキュリティ機構では，実現方式 1 として，汎用性を重視し，特権昇格攻撃を防止する権限情報保護機能を実現している．本稿では，新たに，提案するセキュリティ機構の実現方式 2 として，カーネルデータの書き込み制限対象領域の細粒度化とオーバヘッド低減を可能とする，セキュリティ機能の無効化攻撃を防止のためのセキュリティ機能保護機能を検討する．

- **実現方式 1**：特権昇格攻撃の防止のため，ユーザプロセスの権限情報の保護を目的とし，システムコール単位にて権限情報の書込み制限を制御する手法

- **実現方式 2**：セキュリティ機能の無効化攻撃の防止のため，カーネルコード単位にてセキュリティ機能に関するカーネルデータの書込み制限を制御する手法

本稿では，2 つの実現方式の機能検討と考察を行い，MPK PKS 利用時における性能評価について検証を行なった：

(1) 提案するセキュリティ機構の設計を整理し，実現方式 1 および 2 の処理の共通化，ならびに保護対象とするカーネルデータの違いに起因する制御タイミングを比較した．

(2) 提案するセキュリティ機構の実現方式 1 におけるシステムコール呼出しにかかるオーバヘッドは 2.96% から 9.01% であること，ならびに MPK PKS の書込みに 22.1 ns，レジスタ操作の読込みに 30.5 ns，および書込みに 1347.9 ns の処理負荷がかかることを示した．

2. 背景

2.1 Memory Protection Key

MPK は Intel CPU の提供するセキュリティ機能であり，仮想記憶空間のページ単位（Page Table Entry, PTE）にて読書き制限を制御可能とする [7]．MPK では，ユーザモードで利用する Protection Keys for Userspace（PKU）と読書き制限操作レジスタ Protection Key Right for User-mode (PKRU)，ならびにカーネルモードで利用する Protection Keys for Supervisor（PKS）と読書き制限操作レジスタ `IA32_PKRS_MSR` レジスタ（以降，PKRS）がある．

図 2 に示す通り，PTE において，4 bit の Protection key (Pkey) が割当てられており，16 個の Pkey を指定可能である．また，32 bit の読書き制限操作レジスタ（Pkey あたり，Write disable（WD），Access disable（AD）の 2 bit）を介して，Pkey 単位で読書き制限の制御を行う．

Pkey i ($0 \leq i \leq 15$) に対する読書き制限制御は，読書き制限操作レジスタを介して行う．$i \times 2$ のビット AD i の値が 1 の場合，読込み不可．0 の場合，読込み可，$i \times 2$+1 のビット WD i+1 が 1 の場合，書込み不可，0 の場合，書込み可とされる．

MPK において，読書き制限操作レジスタでの読込み制限，書込み制限は個別に制御可能である．また，複数の PTE に対して，Pkey を指定することで，読書き制限操作レジスタを利用して一括に読書き制限制御を可能としている．

2.2 カーネル脆弱性を利用した攻撃

特権昇格攻撃，およびカーネルの提供するセキュリティ機能に対する無効化攻撃では，カーネルへの攻撃に利用可能とされる実装不備であるカーネル脆弱性のなかでも，カーネル脆弱性を介した任意のカーネルコードの挿入，実行によるメモリ破壊攻撃が利用される [10]

Linux カーネルにおいて，全てのユーザプロセスは同一のカーネルの仮想記憶空間を利用している．攻撃の起点となるカーネル脆弱性と共通の仮想記憶空間に存在するカーネルコード，およびカーネルデータは，仮想アドレスを指定し参照可能であり，メモリ破壊攻撃に成功した場合，任意のカーネルデータは改ざん可能となる．

特権昇格攻撃では，カーネルにおける権限操作を行うカーネル関数を強制的に呼出し，権限情報が変更される [11–13]．また，権限情報を格納するカーネルデータの変数 `cred` を改ざんし，ユーザプロセスのユーザ ID を管理者ユーザへ変更する [14]．

セキュリティ機能である強制アクセス制御の無効化攻撃では，カーネルにおける強制アクセス制御を管理する関数ポインタの一覧 `selinux_hooks` の一部を強制アクセス制

御を回避するカーネル関数へのポインタへ変更し，セキュリティ機能の無効化を行う [1,2]．強制アクセス制御が無効化された場合，管理者権限の制限は行われない．特権奪取攻撃と組み合わせることで，管理者権限を自由に利用可能となる．

3. 脅威モデル

本稿において，提案するセキュリティ機構の想定する脅威モデルとして，攻撃者の目標は特権奪取およびカーネルの提供するセキュリティ機能の無効化とする．

3.1 攻撃対象環境

想定する脅威モデルにおける攻撃者および攻撃対象環境は以下とする．

- 攻撃者： 一般ユーザ権限にて，カーネル脆弱性を利用する PoC コードを介して特権奪取およびセキュリティ機能の無効化を試行可能．
- カーネル：特権奪取とセキュリティ機能の無効化に利用可能なカーネル脆弱性を含む．セキュリティ機能（例，強制アクセス制御）は有効化して提供する．
- カーネル脆弱性：攻撃者の実行するユーザプロセスより呼出されるメモリ破壊可能なカーネル脆弱性．脆弱なカーネルコードから任意の仮想アドレスに対して，書込み可能．
- 攻撃対象：カーネルデータのうち，ユーザプロセスの権限情報，およびセキュリティ機能に関するカーネルデータ．

3.2 攻撃シナリオ

想定する攻撃者の攻撃シナリオは，攻撃対象となるカーネルに対して，カーネル脆弱性を利用する PoC コード利用するユーザプロセスを実行し，脆弱なカーネルコードを介することで攻撃対象の改ざんを行う．例として，強制アクセス制御の無効化攻撃では，アクセス制御箇所を関数ポインタの改ざんにより，アクセス制御判定を行わないカーネルコードへ置換える．カーネルの提供する強制アクセス制御が働かなくなることから，管理者権限への制限を排除する．その後，特権奪取攻撃により，ユーザ権限を管理者権限に書換え，計算機の完全な制御を可能とする．

提案するセキュリティ機構を用いたカーネルにより，脅威モデルとして想定する攻撃対象環境において，攻撃者による攻撃シナリオに沿ったセキュリティ機能の無効化攻撃を防止する．

4. 提案手法

提案するセキュリティ機構の設計においては，カーネルにおいて，指定したカーネルデータへの書込み制限を指定するため，次の要件を満たすことを目指した．

要件： カーネル脆弱性を利用したカーネルデータの改ざんによる特権昇格，ならびにセキュリティ機能の無効化攻撃を想定する．カーネルデータに対する書込み制限の管理として，保護対象に指定したカーネルデータ毎に書込み制限制御を特定のシステムコール単位，ならびにカーネルコード単位で処理可能とする．書込みを許可されたシステムコールに関するカーネルコード，ならびにカーネルコードの実行中においてのみ保護対象に指定したカーネルデータへの書込みを可能とする．

4.1 設計

提案するセキュリティ機構として，要件を満たす設計概要を図 3 に示す．提案するセキュリティ機構では，保護対象として指定するカーネルデータ（例，変数値，および関数ポインタ），ならびに識別子を設ける．また，識別子をシステムコール，およびカーネルコードと紐付け，保護対象カーネルデータの書込み制限の制御に用いる．

4.1.1 保護対象カーネルデータ

提案するセキュリティ機構において管理する保護対象カーネルデータおよび識別子は次の通りとする．

- 保護対象カーネルデータ：ユーザプロセスのカーネルデータ（例．権限情報），およびカーネルにおけるセキュリティ機能において利用されるカーネルデータ（例．関数ポインタ）
- 識別子：保護対象カーネルデータの読書き制限を設定するために用いる識別子．書込み制限制御のため，識別子に対し，保護対象カーネルデータ，および，書込み許可システムコール，書込み許可カーネルコードの紐付けを行う．

提案するセキュリティ機構では，カーネル内部において，保護対象カーネルデータ，識別子のリストを予め設ける．

4.1.2 読書き制限の管理と制御

提案するセキュリティ機構において，保護対象カーネルデータへの読書き制限制御は特定のシステムコール，およびカーネルコードにて行う．提案するセキュリティ機構では，次の 2 種類を定義し，管理する．書込み制限の制御は，システムコールの発行時，ならびにカーネルコード実行時において書込み制御機能を呼出すことで行う．

- 書込み許可システムコール：保護対象カーネルデータの書込み権限を有するシステムコール．
- 書込み許可カーネルコード：保護対象カーネルデータの書込み権限を有するカーネルコード．

提案するセキュリティ機構における書込み制限制御は，書込み許可システムコール発行時，および書込み許可カー

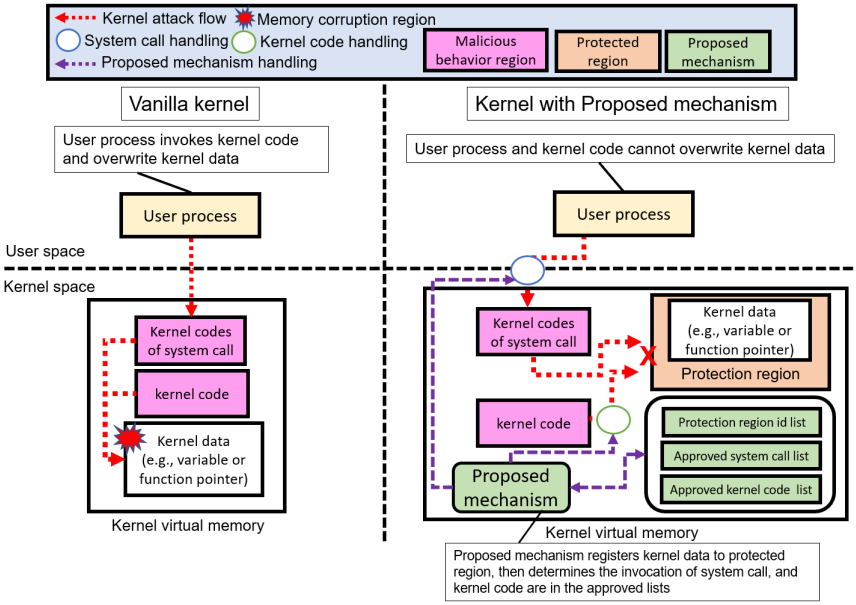

図 3 提案するセキュリティ機構の設計概要図

表 1 実現方式の比較

Item	Implementation 1	Implementation 2
Protected kernel data	Priviledge information	Kernel code pointer & Access policy
Handling granuality	System call	Kernel code
Mitigation	Priviledge escalation	Security feature defeating
Performance effect	High	Low

ネルコード実行前に，識別子に基づき，保護対象カーネルデータへの書込みを許可する．その後，システムコール終了時，およびカーネルコード実行後に保護対象カーネルデータへの書込みを制限する．

提案するセキュリティ機構では，識別子毎に書込み許可リストとして，書込み許可システムコール，および書込み許可カーネルコードのリストを管理し，書込み制限の管理制御に用いる．

5. 実現方式

実現方式の想定環境は，x86_64 CPU アーキテクチャの Linux とし，権限情報保護を目的とした方式，ならびにカーネルのセキュリティ機能の保護を目的とした方式の 2 つの実現方式を考案した．

- **実現方式 1**：書込み許可システムコールとして，権限情報の変更を行うシステムコールの実行時のみ，権限情報を格納したカーネルデータを書込み可能に制御
- **実現方式 2**：書込み許可カーネルコードとして，セキュリティ機能の設定を変更するカーネルコード実行時のみ，セキュリティ機能に関するカーネルデータを書込み可能に制御

5.1 実現方式による保護対象と書込み制御タイミングの差異

実現方式による保護対象カーネルデータ，書込み制御タイミングについて**表 1** に示す．実現方式 1 は，権限情報保護機能として，ユーザプロセスの権限情報を保護対象カーネルデータに指定し，書込み許可システムコールとして，権限情報の変更を行うシステムコールの実行時のみ書込み可能となるよう制御を行う．ユーザプロセスが特権昇格攻撃を目的とした攻撃を試みた場合でも，権限情報の書込みは行えず，攻撃抑止を実現する．

一方，実現方式 2 は，カーネルのセキュリティ機能の保護機能として，セキュリティ機能のうち， Linux Security Module (LSM) による強制アクセス制御を呼出すカーネルコードの関数ポインタ，およびアクセス制御ポリシを保護対象カーネルデータに指定し，書込み許可カーネルコードとして，セキュリティ機能の設定を変更するカーネルコード実行時のみ書込み可能となるよう制御を行う．ユーザプロセスがセキュリティ機能の無効化を試みた場合でも，セキュリティ機能の呼出し処理は無効化できず，攻撃抑止を実現する．また，カーネル内部における処理のため，ユーザプロセスの性能に与える影響は低い．

5.2 実現方式における共通処理

実現方式 1 および 2 において，保護対象カーネルデータの管理，およびページフォルトハンドリングは共通処理とする．

5.2.1 保護対象カーネルデータ

実現方式を適用した Linux カーネルでは，実現方式 1 および 2 にてページ単位（4KB）毎に配置された保護対象カーネルデータに対し，識別子を設定し，PKS による書込み制限制御に用いる．

- 識別子：保護対象カーネルデータの書込み制限制御のため，保護対象カーネルデータ毎に識別番号 i を設定

する．識別番号 i は PKS において，PTE に指定する Pkey i（4 bit）の値と同値とする．

- PKS による書込み制限制御：保護対象カーネルデータの識別番号 i を PTE に対する Pkey i として指定する．PKRS の WDi の値を 1 とした場合，書込み制限，0 とした場合書込み許可として書込み制限制御を行う．

実現方式 1 および 2 における PKRS による書込み制限制御の操作は，個別処理として，各実現方式の保護対象カーネルデータの書込み制御において行う（詳細は 5.3.2 節，5.4.2 節を参照）．

5.2.2 ページフォルトハンドリング

ページフォルトハンドラ `do_page_fault` 関数，ならびに `do_double_fault` 関数にて PKS で書込み制御された保護対象カーネルページへの不正な参照を捕捉する．Linux カーネルにおいては，ページフォルト（エラー番号 35）の場合， Pkey により保護されたページへの書込み保護違反となる．実現方式において，保護対象カーネルデータへの書込みを許可しない場合，`force_sig_info` 関数にて対象ユーザプロセスに対して SIGKILL の送信を行う．

5.3 実現方式 1 の個別処理

実現方式 1 の概要を図 4 に示す．実現方式 1 では，ユーザプロセス毎に権限情報を保護対象カーネルデータとするため，保護対象カーネルデータの一覧，書込み許可システムコールの一覧をリストとして管理する．

5.3.1 保護対象カーネルデータ

実現方式 1 における，保護対象カーネルデータとして，ユーザプロセスの生成時，専用ページ（4KB）を生成し，**表 2** に示すユーザプロセスの権限情報を格納し，ユーザプロセスの動作期間中，PKS による書込み制限制御対象とする．また，書込み許可システムコールの一覧も保護対象とし，カーネルの起動時に PKS による書込み制限制御を行う．

5.3.2 保護対象カーネルデータの書込み制御

実現方式 1 において，書込み許可システムコールのリストに含めるシステムコールは，権限情報の操作を行うシステムコールとし，表 2 に示す．実現方式 1 における， PKS Pkey を用いた保護対象カーネルデータの書込み制限の制御は，次の手順にて行う．

（1） ユーザプロセスによるシステムコール呼出しを捕捉．

（2） 書込み許可システムコールリストに含むシステムコール番号か判定．

（a） 書込み許可システムコールの場合：PKS による書込み制限制御を行い，保護対象カーネルデータを書込み許可に設定．

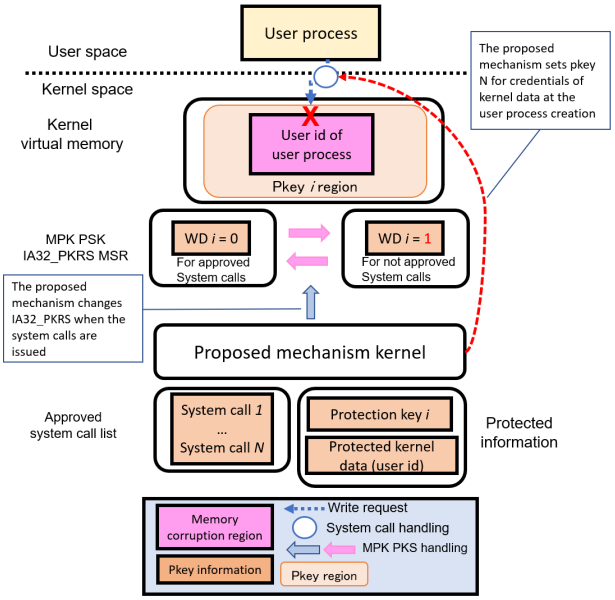

図 4 提案するセキュリティ機構の実現方式 1

表 2 実現方式 1 における保護対象カーネルデータと書込み許可システムコール（参考文献 [15] より抜粋）

Item	Description
Protected kernel data	User ID (e.g., uid, euid, fsuid, suid) Group ID (e.g., gid, egid, fsid, sgid)
Allowed system call	execve, setuid, setgid, setreuid, setregid setresuid, setresgid, setfsuid, setfsgid

（3） システムコールの実行を継続する．

（4） システムコールの終了後，PKS による書込み制限制御を行い，保護対象カーネルデータを書込み制限に設定．

5.4 実現方式 2 の個別処理

実現方式 2 の概要を図 5 に示す．実現方式 2 においては，書込み許可カーネルコードとして，セキュリティ機能の変更を伴うカーネルコード実行時のみセキュリティ機能に関するカーネルデータの書込み制限を制御するため， LSM に関するカーネルデータの一覧，および書込み許可カーネルコードの一覧をリストとして管理する．

5.4.1 保護対象カーネルデータ

実現方式 2 において，保護対象とするカーネルデータは，**表 3** に示す LSM に関連するカーネルデータの一部である関数ポインタを格納する変数，ならびにアクセス制御ポリシを格納する変数とるす．また，書込み許可カーネルコードのリストについても PKS を用いた書込み制限制御による保護対象とする．

5.4.2 保護対象カーネルデータの書込み制御

実現方式 2 における保護対象カーネルデータのうち，LSM に関連するカーネルデータの関数ポインタを格納する変数，および書込み許可カーネルコード名リストは Linux カーネルの起動中に値が設定される．実現方式 2 において

表 3 実現方式 2 における保護対象カーネルデータと書込み許可カーネルコード.

Item	Description
Protected kernel data	Function pointer (e.g., selinux_hooks) Security policy (e.g., selinux_state)
Allowed kernel code	Kernel functions in the selinux_hooks avc_init, avc_insert, avc_node_delete, avc_node_replace

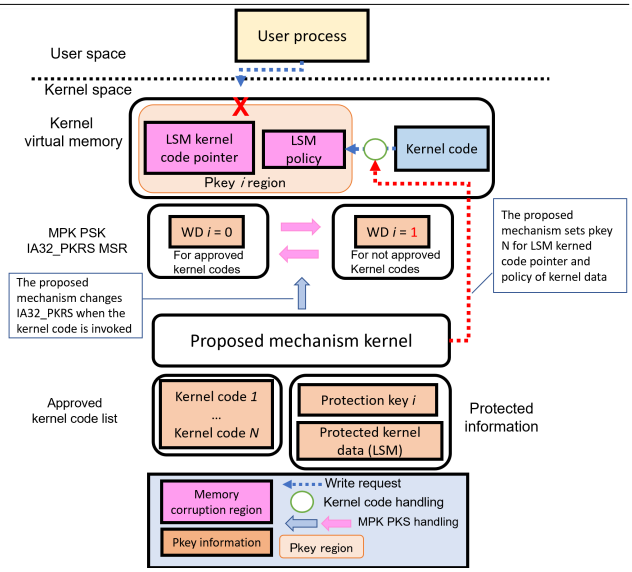

図 5 提案するセキュリティ機構の実現方式 2

は，個々の値がカーネルデータに格納された直後に PKS による書込み制限対象とし，管理する．

実現方式における書込み許可カーネルコードに対する PKS を用いた制限制御は次の手順にて行う．

(1) 書込み許可カーネルコードにより，実現方式 2 のカーネルコードを呼出す．

(2) 実現方式 2 のカーネルコードにて，書込み許可カーネルコードからの呼出しか判定．

(a) 書込み許可カーネルコードの場合：PKS による書込み制限制御を行い，保護対象カーネルデータを書込み許可に設定．

(b) 保護対象カーネルデータの書込み制限設定を検証するカーネルコードをタイマに登録，一定期間後に呼出すよう設定．

(3) 書込み許可カーネルコードの処理を継続．

(4) 書込み許可カーネルコードの終了前，実現方式 2 のカーネルコードを呼出し，PKS による書込み制限制御を行い，保護対象カーネルデータを書込み制限に設定．

(5) 書込み許可カーネルコードの処理を終了

書込み許可カーネルコードのリストに含めるカーネルコードは，表 3 の LSM の制御ポリシの操作を行うカーネルコードとする．

実現方式 2 において，タイマ登録される書込み制限設定を検証するカーネルコードでは，書込み許可が意図せずに継続されることを防止する．書込み許可と制限を設定するカーネルコードの呼出し回数が一致しない場合，ならびに書込み許可の継続時間が規定時間を超えた場合，強制的に書込み制限に設定する．

6. 評価

提案するセキュリティ機構を適用したカーネルのセキュリティ機能評価は先行研究において示している [6]．本稿では，性能評価として，実現方式 1 に対して，ベンチマーク測定によるカーネルとユーザプロセスへの動作影響有無の調査，および実現方式 2 で用いる PKS 操作による性能への影響を確認した．評価項目と内容を以下に示す．

(1) カーネル処理におけるオーバヘッド
実現方式 1 を適用した Linux カーネルにおいて，ベンチマークソフトウェアを用いたシステムコールのオーバヘッドの測定した．

(2) PKS 操作におけるオーバヘッド
提案するセキュリティ機構で用いる PKS の利用にかかるオーバヘッドとして，PKS 操作に関する処理時間を測定した．

PKS に対応した CPU は 2022 年 1 月時点では提供されていない．評価には CPU Intel(R) Core(TM) i7-7700HQ (2.80GHz，4 コア)，Memory 16 GBytes を備えた計算機を用い，性能評価を行う仮想計算機環境として PKS に対応した QEMU 6.0.91 を用いた [9]．QEMU 上のゲスト OS は Debian 10.2 とし，実現方式 1 を Linux kernel 5.3.18 を対象に，15 個のファイルに対して 431 行を追加し実装した．また，PKS 操作の性能負荷評価のための専用の測定プログラムとして 165 行を Linux カーネル に追加した．

6.1 カーネル処理におけるオーバヘッド

実現方式 1 のカーネル負荷として，システムコールのオーバヘッドの測定をベンチマークソフトウェア LMbench により評価した．LMbench を実現方式 1 の適用前の Linux kernel と適用した Linux kernel にてそれぞれ 10 回実行し，平均値からシステムコールのオーバヘッドを算出した．

評価結果を表 4 に示す．LMbench では， fork+/bin/sh は 54 回, fork+execve は 4 回, fork+exit は 2 回, open/close は 2 回，および，その他は 1 回のシステムコール呼出しを行う．表 4 から，提案するセキュリティ機構の適用時において，最もオーバヘッドの発生したシステムコール処理は fork+execve であり，9.01% を示した，また，最も少ないオーバヘッドは stat であり，2.96%を示した．

6.2 PKS 操作におけるオーバヘッド

PKS の利用にかかる性能評価として，PKS 操作に関するオーバヘッドを測定した．専用の測定プログラムにお

表 4　lmbench を用いたオーバヘッド (μs)

System call	Vanilla kernel	Implementation 1	Overhead
fork+/bin/sh	227111.28	236738.69	9627.41 (4.24%)
fork+execve	12780.0566	13931.6703	1151.6136 (9.01%)
fork+exit	10837.0729	11285.5603	448.4874 (4.14%)
open/close	1302.5639	1334.5312	41.9672 (2.95%)
read	168.8898	180.4594	11.5696 (6.85%)
write	164.2567	176.4273	12.1705 (7.41%)
fstat	195.0063	203.7508	8.7445 (4.48%)
stat	613.7426	631.9393	18.1966 (2.96%)

表 5　MPK PKS 操作にかかるオーバヘッド (ns)

Instruction	Our proposed mechanism
Pkey write	30.5
PKRS read	22.1
PKRS write	1347.9

いて，カーネルにおける PTE への Pkey の設定ならびに PKRS 読書きを 10,000 回繰返す処理を 10 回行い，平均値を測定した．

評価結果について表 5 に示す．Pkey の書込みには 30.5 ns ならびに PKRS の読込みに 22.1 ns，書込みに 1347.9 ns のオーバヘッドを必要とすることを示した．

7. 考察

7.1 評価に対する考察

性能評価結果より，提案するセキュリティ機構の実現方式において，カーネル処理，および PKS による読書き制御にオーバヘッドを要することを示した． PKS 操作にかかる時間は実現方式 1 および 2 において共通の処理であり，カーネル処理への影響となる．

実現方式ごとの負荷の差異として，実現方式 1 においては，ユーザプロセスによるシステムコール呼出し毎に書込み許可システムコール番号かの判定を行うため，システムコール実行時間に影響し，ユーザプロセスに対する負荷が発生する．一方，実現方式 2 においては，セキュリティ機能の呼出毎に書込み許可カーネルコードの判定を行うのみである．カーネル処理への影響は発生するが，ユーザプロセスへの直接的な負荷はアクセス制御の判定が必要となる場合に限定されると考えている．

7.2 提案手法の考察

提案するセキュリティ機構の実現方式においては，権限情報およびセキュリティ機能として強制アクセス制御に関するカーネルデータを保護可能とした．

ユーザプロセスの権限情報以外のカーネルデータや強制アクセス制御の SELinux 以外では，各実現方式においても適用対象とする保護対象カーネルデータや適切な制限操作レジスタの操作タイミングは異なると考えている．提案するセキュリティ機構のさらなる汎用化のため，保護対象とすべきカーネルデータの検討，ならびに保護対象を追加した場合，性能負荷への影響を考慮して検討する必要があると考えている．

7.3 限界

先行提案と合わせ，我々のセキュリティ機構では，書込み許可システムコール，ならびに書込み許可カーネルコードに対して，Pkey を利用したカーネルデータの書込み制限を管理している．Pkey 数に制限があることから，保護対象とするカーネルデータの識別子数に上限がある．保護対象カーネルデータの種別数に制約を受けるため，提案手法の適用においては，適切な保護対象カーネルデータの分類検討が必要である．また，保護対象カーネルデータの種別毎に制限操作レジスタの操作処理の追加が必要であり，性能負荷に加え，カーネルの変更が求められる．

提案手法において，PKS はレジスタ操作のみで Pkey を指定したページの読書き制限を制御可能であり，カーネルにおいて，軽量かつ実装容易性を備えたカーネルデータの保護機能として利用可能である．一方，カーネルにおける割込みや例外による非同期処理時の書込み制限の処理漏れ，複数の保護対象カーネルデータにて，Pkey を共有した場合における排他処理の影響について検討が必要であると考えている．

8. 関連研究

アプリケーションにおける MPK を利用したデータ保護
アプリケーションにおける MPK を利用したデータ保護として，libmpk ユーザプロセスにおいて，PSU を利用した保護対象データの操作を柔軟にするための抽象化ライブラリを提案している [16]．ERIM は PSU を利用してアプリケーションにおいて，複数のユーザプロセスに保護対象データを分離して管理する手法を提案している [17].

カーネルにおける MPK を利用したデータ保護
カーネルにおける MPK を利用したカーネルコードおよびカーネルデータの保護として，xMP は複数のドメインを用意し，カーネルの仮想記憶空間を構成する複数のページを各ドメインに割当て，Pkey により，仮想マシンモニタ（VMM）から管理する機構を提案している [18]. libhermitMPK はカーネルコードとデータを複数の Pkey に分割管理し，Pkey 単位で管理することで不正な読書きから保護する機構を提案している [19].

不正コード実行防止
カーネルにおける不正なカーネルコードの実行防止として，コード呼出し順を検査する CFI [4] の適用が進められている [20]．また，カーネルへの CFI 適用のため，KCoFI では，独自アーキテクチャとして非同期処理発生時におけるコード呼出順の整合性保存機構を提案している [5].

表 6　カーネルデータ保護手法の比較

Feature	libhermitMPK [19]	xMP [18]	Proposed mechanism
Protection	Entire kernel	Entire kernel	Kernel data
Granuality	Kernel code	VMM	System call & Kernel code
Implementation	Unikernel	VMM monitoring	Internel kernel
Limitation	Kernel code security	VMM overhead	Pkey number

8.1　関連研究との比較

既存研究 [18,19] との比較を表 6 に示す．libhermitMPK は，カーネルを 2 つの領域（Safe/Unsafe）に分割し，領域毎に Pkey を割当て管理する．カーネルコードと同一領域に属するカーネルデータへの読書きのみに制限することでカーネルデータ保護を実現している．xMP は，カーネルの仮想記憶空間を複数ドメイン領域に分割，ドメイン毎に Pkey を設定し，カーネルコード，カーネルデータを割当てることでカーネルの保護実現する． xMP においては，ドメインの適用を VMM からゲスト OS に行う．

libhermitMPK，および xMP において，脆弱なカーネルコードとカーネルデータが同一の領域ならびにドメインに配置された場合，Pkey は共有されることから，カーネルデータは改ざんされる可能性がある．提案するセキュリティ機構では，保護対象カーネルデータのみに Pkey を割当て，書込み制限を制御する．カーネルに脆弱性が発見され，攻撃に利用される場合，脆弱なカーネルコードが書込み許可システムコールや書込み許可カーネルコードでなければ保護対象カーネルデータの改ざんは防止される．

9.　まとめ

本稿では，メモリ破壊攻撃への対策として，我々の提案しているカーネルデータへの書込み制限を動的制御するセキュリティ機構の設計を整理し，権限情報を保護するための実現方式 1，ならびにカーネルのセキュリティ機能を保護する実現方式 2 について機能考察と既存研究との比較考察を進め，性能評価を行った．

提案したセキュリティ機構では，CPU の提供するメモリ保護機構である MPK PKS を用いることで動作中のカーネルにおいて指定したカーネルデータの書込み制限を制御可能としている．性能評価においては，実現方式 1 を実装した Linux にて，システムコール呼出しに対しては， 2.96% から 9.01% のオーバヘッドに留まること，ならびに MPK PKS 操作にかかる負荷は 22.1 ns から 1347.9 ns のオーバヘッドとなることを明らかにした．

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] Exploit Database, Nexus 5 Android 5.0 - Privilege Escalation, https://www.exploit-db.com/exploits/35711/, (2015). (accessed 2018-08-10).
[2] grsecurity, super fun 2.6.30+/RHEL5 2.6.18 local kernel exploit, https://grsecurity.net/~spender/exploits/exploit2.txt, (2009) (accessed 2018-08-10).
[3] Shacham, H., et al.: On the effectiveness of address-space randomization, *Proc. 11th ACM Conference on Computer and Communications Security*, pp. 298–307, (2004).
[4] Abadi, M., et al.: Control-Flow Integrity Principles, Implementations, *Proc. 12th ACM Conference on Computer and Communications Security*, pp. 340–353 (2005).
[5] Criswell, J., Dautenhahn, N. and Adve, V.: KCoFI: Complete Control-Flow Integrity for Commodity Operating System Kernels. *Proc. IEEE Security and Privacy*, pp. 292—307 (2014).
[6] 葛野弘樹, 山内利宏カーネルにおける Memory Protection Key を用いた権限情報保護機構の提案コンピュータセキュリティシンポジウム 2021 論文集, Vol.2021, pp.647-654, (2021).
[7] Intel Corporation.: Intel(R) 64 and IA-32 Architectures Software Developer’s Manual, available from https://www.intel.co.jp/content/www/jp/ja/architecture-and-technology/64-ia-32-architectures-software-developer-system-programming-manual-325384.html, (accessed 2021-08-18).
[8] Gleixner, T.: PKS: Add Protection Keys Supervisor (PKS) support, available from https://lwn.net/Articles/826091/. (accessed 2021-08-18).
[9] Bonzini, P.: [PATCH] target/i86: implement PKS, available from https://lore.kernel.org/qemu-devel/20210127093540.472624-1-pbonzini@redhat.com/. (accessed 2021-08-18).
[10] Chen, H., et al.: Linux kernel vulnerabilities - state-of-the-art defenses and open problems. *Proc. Second Asia-Pacific Workshop on Systems*, pp. 1–5 (2011).
[11] CVE-2016-4997, available from https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-4997. (accessed 2019-05-12).
[12] CVE-2016-9793, available from https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-9793. (accessed 2019-05-12).
[13] CVE-2017-1000112, available from https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-1000112. (accessed 2019-05-12).
[14] CVE-2017-16995, available from https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-16995, (accessed 2019-06-10).
[15] Yamauchi, T., et al.: Additional kernel observer: privilege escalation attack prevention mechanism focusing on system call privilege changes, *International Journal of Information Security*, vol. 20, pp. 461–473 (2021).
[16] Park, S., et al.: libmpk: Software Abstraction for Intel Memory Protection Keys (Intel MPK), *Proc. 2019 USENIX Annual Technical Conference*. pp. 241–254 (2019).
[17] Vahldiek-Oberwagner, A., et al.: ERIM: Secure, Efficient In-process Isolation with Protection Keys (MPK), *Proc. 28th USENIX Conference on Security Symposium*, pp. 1221–1238, (2019).
[18] Proskurin, S., et al.: xMP: Selective Memory Protection for Kernel and User Space, *Proc. 2020 IEEE Symposium on Security and Privacy*, pp. 563–577 (2020).
[19] Sung, M., et al.: Intra-Unikernel Isolation with Intel Memory Protection Keys, *Proc. 16th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pp. 143–156 (2020).
[20] Edge, J.: Control-flow integrity for the kernel, https://lwn.net/Articles/810077/. (accessed 2022-01-08).