Computer Security Symposium 2019
21 - 24 October 2019

カーネル仮想記憶空間における排他的ページ参照による カーネルの攻撃耐性の実現と評価

葛野 弘樹[1,2]　山内 利宏[1]

概要：オペレーティングシステムカーネルの仮想記憶空間は，全てのプロセスで共有して管理されることが多い．仮想化やコンテナ等のカーネル機能を提供するプロセスでは，CPU 状態やセキュリティポリシをカーネルの仮想記憶空間に保存する．一部のプロセスからカーネルの脆弱性を利用した攻撃を想定した場合，他のプロセスの利用するカーネル仮想記憶空間領域を侵害される可能性はある．カーネルの仮想記憶空間の保護として，カーネルモードとユーザモード，システムコール単位にて仮想記憶空間を分離する手法が提案されている．既存手法では，プロセス毎にカーネルの仮想記憶空間において参照可能な記憶領域は明確に分離されず，依然としてカーネルへの攻撃により，全てのプロセスで共有するカーネルの仮想記憶空間は攻撃の影響を受ける．本稿では，プロセス単位やカーネルの特定機能に対し，カーネル仮想記憶空間を構成する特定ページを排他的に参照可能とする排他的ページ管理機構を提案する．Linux にて提案を実現し，提案手法によるカーネルの攻撃耐性と有効性を評価し，考察を行う．

Design and Implementation of Exclusive Page Reference Mechanism Mitigates Kernel Vulnerability Attack

Hiroki Kuzuno[1,2]　Toshihiro Yamauchi[1]

Abstract: Operating System kernel has the sharing mechanism of kernel virtual memory for each user process. Some kernel features and processes store virtual CPU status or security policy on the kernel virtual memory (e.g., virtualization or container). An adversary's process compromised OS kernel via kernel vulnerability. It overwrites other process's data on the kernel virtual memory. Kernel virtual memory isolation methods separate the one kernel virtual memory to user mode, kernel mode, and system call invocation timing. Although these methods mitigate that an adversary's process occurs suspicious activity from user mode to kernel mode interaction, user processes have shared reference available pages on kernel virtual memory. In this paper, we propose a novel mechanism that provides an exclusive page reference feature. It enables that user process keeps domestic pages on the kernel virtual memory. It is implemented and evaluated on the latest Linux kernel, then discussion for kernel attack mitigation capability.

1. はじめに

オペレーティングシステム（OS）カーネルにて複数のアプリケーションを実行する際，カーネルはユーザモードで動作するプロセス毎に仮想記憶空間を生成し，相互にアクセス制御を行うことで排他的な記憶空間として利用させる．

複数のプロセス間では，カーネルモードで利用するカーネルの仮想記憶空間は共通化されており，攻撃者のプロセスによりカーネル脆弱性を利用した攻撃を受け，カーネルの仮想記憶空間を侵害された場合，カーネルの仮想記憶空間上にある他のプロセスの利用している情報を改ざんされる可能性はある．改ざんによる影響として，カーネルの仮想化機能を利用するプロセスでは，カーネルの仮想記憶空間に配置した CPU やメモリ状態の改ざん，ならびにコンテナ機能を利用したプロセスにおいては，資源の実行制限の不正操作がある．

[1] 岡山大学 大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University
[2] セコム株式会社 IS 研究所
Intelligent Systems Laboratory, SECOM Co., Ltd., Japan

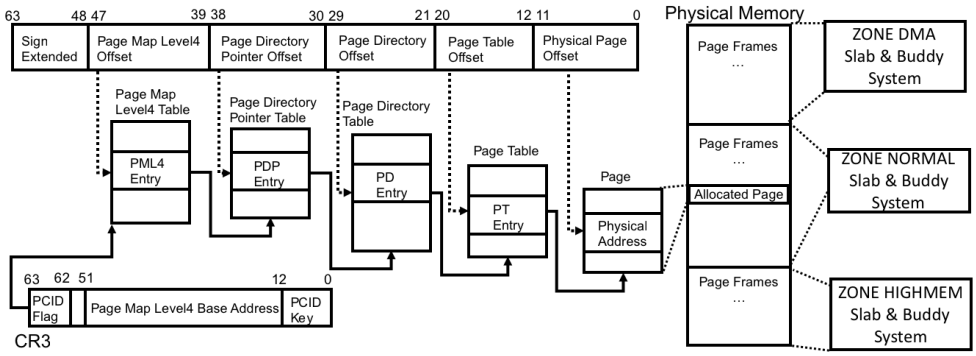

図 1　Linux x86_64 におけるページテーブルおよびページ管理
Fig. 1　Page allocation and page table manage virtual address and physical address

仮想記憶空間は，Meltdown 攻撃の対策として KPTI（Kernel Page Table Isolation）により，ユーザモード用とカーネルモード用に分離された [1]．カーネルの仮想記憶空間の分離として，ProcLocal （Process Local Memory）では，プロセスに対してカーネルの仮想記憶空間の一部領域を占有的に利用可能としている [2]．SCI（System Call Isolation）では，プロセス毎にシステムコール処理時にのみ利用するカーネル仮想記憶空間を作成している [3]．ProcLocal や SCI により，特定のカーネル機能のみ占有可能な仮想記憶領域や独立したカーネルの仮想記憶空間を構築することは可能である．しかし，カーネルの仮想記憶空間において脆弱性の存在するカーネルコードや物理記憶空間と紐づくダイレクトマッピング領域を管理するページは明確に分離されない．依然としてカーネル脆弱性を利用した攻撃により．カーネル仮想記憶空間上にて共有されるカーネルコードやデータは侵害される可能性はある．

本稿では，カーネルのセキュリティ機能，仮想計算機，ならびにコンテナプロセスの動作に必要な情報に対する攻撃の影響軽減を目的とし，カーネルへの攻撃を介してカーネルの仮想記憶空間における他のプロセスの利用するカーネルコード・データ改ざんを困難とする新たなセキュリティ機構を提案する．具体的な提案手法は以下の通りである：

- 提案手法は，カーネルの仮想記憶空間においてプロセス単位やカーネル機能に対してページ単位での排他的な参照を可能にする．プロセス単位やカーネル機能毎にカーネルの仮想記憶空間に排他的な記憶領域を確保させることで，カーネルの攻撃耐性を実現する．
- 提案手法の効果として，攻撃元プロセスに対するカーネル脆弱性の存在するカーネルコードと紐づくページ参照の困難化，ならびに攻撃によりカーネルの仮想記憶空間の改ざんを可能であったとしても，攻撃元プロセス以外で確保した一部のカーネルの仮想記憶空間のページに対する書換えは困難化される．

我々は，提案手法を最新の Linux において実現し，実際のカーネル脆弱性を利用した攻撃に対する提案手法の有効性について評価した．本稿での我々の研究による貢献ならびに得られた結果は以下の通りである：

- カーネルの仮想記憶空間においてプロセス単位にてカーネルの仮想記憶空間上のカーネルコード・データをページ単位で排他的に参照管理する方式を提案した．この方式は，ハードウェアによる物理記憶空間の分離技術を利用することなく，セキュリティ機能，仮想化，ならびにコンテナなどカーネルにおける特定のカーネル機能の安全性の向上を実現する．また，ベアメタルのみならず仮想計算機上のカーネルでも利用可能なため，クラウド上に構築された仮想環境においても適用可能である．
- KPTI および SCI を適用した Linux にて提案を組合せた実現方式，およびカーネル脆弱性 CVE-2017-16995[4] を利用した攻撃プロセスに対し，提案手法により攻撃対象カーネルデータならびに脆弱性を含むカーネルコードの参照を管理することでカーネルの攻撃耐性を評価した．

2. 背景知識

2.1 仮想記憶空間の管理

仮想記憶はプロセス毎に物理記憶の領域を超えた記憶空間を提供する機能である．カーネルにより，ユーザモード用の仮想記憶空間（ユーザの仮想記憶空間）およびカーネルモード用の仮想記憶空間（カーネルの仮想記憶空間）として仮想アドレスの領域は分割管理される．

図 1 において，Linux x86_64 では仮想記憶空間は多層ページテーブルにより構築されており，仮想記憶空間の割当てバディシステムとスラブアロケータを介し，ページ（x86_64 では 4KB）を確保後，ページテーブルにページを登録し行う．仮想アドレス（x86_64 では 48 ビット）から物理アドレスの変換は多層ページテーブルを辿り，該当ページと仮想アドレスのオフセットから物理アドレスを特定しアクセスする．

2.2 想定する脅威モデル

本稿での脅威モデルは，攻撃者によるカーネル脆弱性を含むカーネルコードを利用した攻撃を介したカーネルの仮想記憶空間の改ざんとする．攻撃では，カーネル脆弱性を

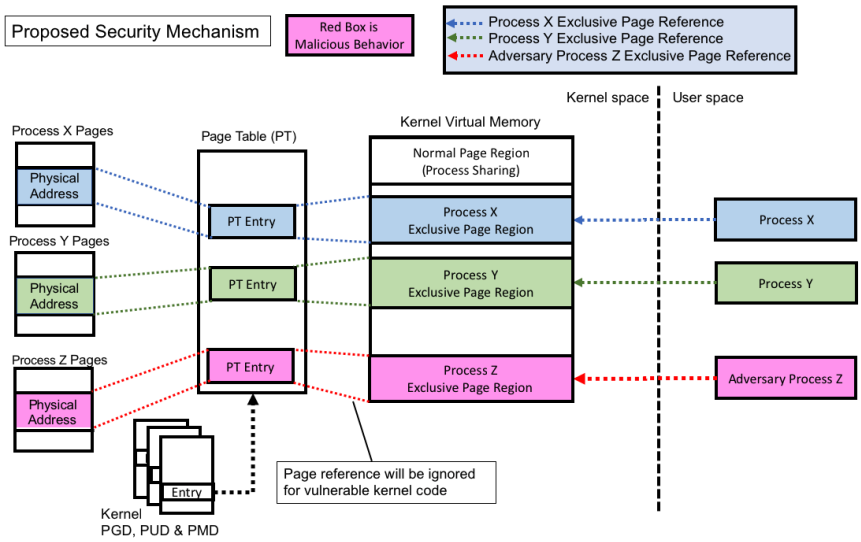

図 2　排他的ページ参照管理の概要図

Fig. 2　The overview of exclusive page reference mechanism

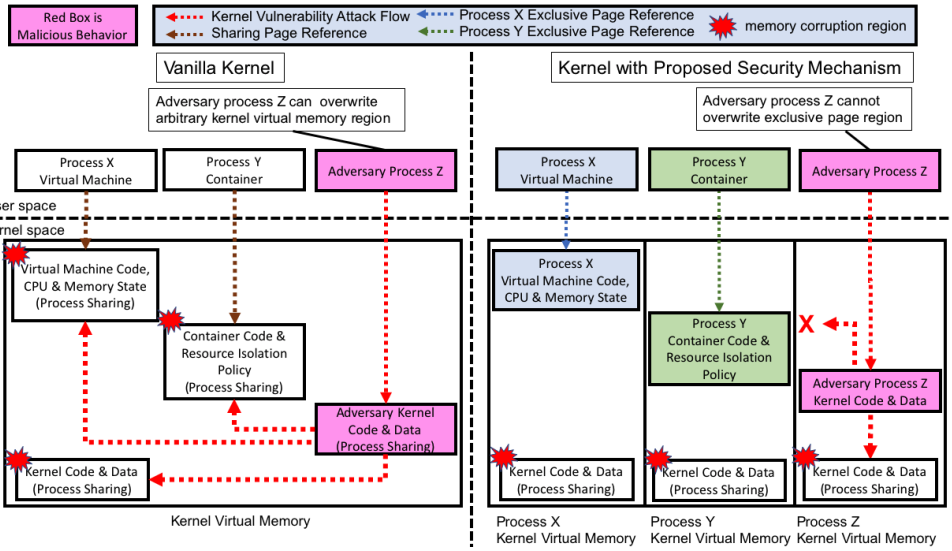

図 3　排他的ページ参照管理適用有無の概要図

Fig. 3　The overview of exclusive page reference mechanism

含むカーネルコードを格納したページを管理するページテーブルから構成される仮想記憶空間のみ書換え可能とする．攻撃において指定される仮想アドレスは，攻撃時に参照される仮想記憶空間の範囲のうち既存のセキュリティ機構，ならびに特定のカーネル機能を提供するカーネルコード・データとする．

3.　提案手法と実現方式

3.1　提案手法における仮想記憶空間の排他的ページ管理

攻撃者によるカーネルの脆弱性を介した攻撃により，プロセスの利用しているカーネルの仮想記憶空間に配置されたカーネルコードやデータは侵害される可能性はある．その課題は次の通りである．

課題：カーネルの仮想記憶空間はユーザモードで動作する全てのプロセスで共有している．セキュリティ機能，仮想化，ならびにコンテナなどの特定のカーネル機能を利用するプロセスは実行中に必要となる仮想環境情報や資源制限ポリシをカーネルの仮想記憶空間に格納する．カーネルの脆弱性を介した攻撃により，カーネルの仮想記憶空間に配置されるカーネルの機能に関するカーネルコードならびにデータを改ざんすることで特定のカーネル機能を利用するプロセスに対して影響を与える．

本稿では，カーネルの仮想記憶空間において通常ページと排他ページを設けることでプロセス毎に排他的なページ参照と管理を行うための新たなセキュリティ機構を提案する．**図 2** のセキュリティ機構では，プロセス毎に利用するカーネル機能を備えるカーネルコードならびにデータを分離可能とする．

提案手法を適用することで，特定のカーネルの機構を利用しているプロセスのカーネルコードやデータを排他ページとして指定し，他のプロセスからのカーネルの脆弱性を介した攻撃からの保護を実現する．**図 3** においては，複数のプロセス X, Y, Z 用のカーネルの仮想記憶空間を個別に生成し，これらのプロセスの排他ページは，当該プロセス

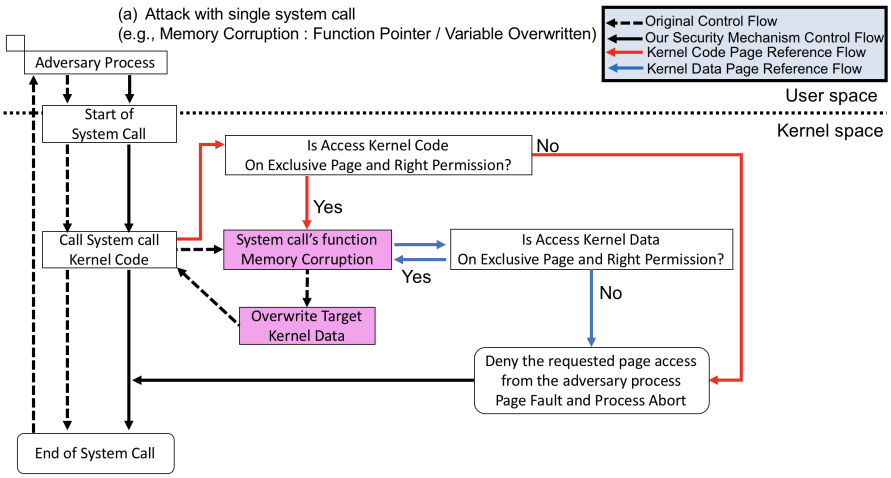

図 4　攻撃プロセスに対する排他的ページ参照管理の処理
Fig. 4　Exclusive page reference mechanism for adversary process

の仮想記憶空間にのみマッピングされている．

3.2　排他ページの管理と処理

提案するセキュリティ機構におけるカーネルの仮想記憶空間は通常ページおよび排他ページを参照する．

通常ページ： カーネルの仮想記憶空間において，複数のプロセスならびにカーネルスレッドにおいて共通して利用され，グルーバルに参照されるカーネルコードおよびカーネルデータを格納するページ

排他ページ： カーネルの仮想記憶空間において，特定のプロセス，ならびにカーネルスレッドの処理にのみ参照可能とし，利用を許可するカーネルコードおよびデータを格納するページ

仮想アドレスを利用して排他ページに指定するカーネルコード，およびデータは，排他ページ管理リストに登録する際に，通常ページの起点となる仮想アドレスを算出し，排他ページとして参照可否を制御する．

物理記憶空間の過度の断片化および速度低下を避けるため，排他ページは一定の指定されたページ数のみ利用可能とする．また，同一ページではあるが，排他ページ管理リストに記載されていない仮想アドレスを利用する場合は排他ページを参照することを可能とする．

3.3　排他ページの格納オブジェクトの対象

セキュリティ機構において，カーネルの提供する機能にて排他ページとして指定可能なカーネルコード，データの種別を挙げる．

カーネルコード： カーネルを構成するカーネルコードのうち，特定の機能としてプロセスやカーネルスレッドから利用されるカーネルコード．

カーネルデータ： カーネルを構成するデータのうち，特定の機能を利用するプロセスやカーネルスレッドにおいて利用されるデータ．

脆弱性を含むカーネルコード： カーネルを構成するカーネルコードのうち，カーネル脆弱性として利用可能なカーネルコード．

プロセスの生成時に作成される仮想記憶空間のうち，カーネルの仮想記憶空間において，他のプロセスやカーネルスレッドにて指定された排他ページについてはページテーブルの配置より排除する．

3.4　排他ページの利用と参照

カーネルにおいて，カーネルコード・データに対する排他ページへの指定は，排他ページ参照を管理するカーネルコードを介し，仮想アドレスを登録することで行う．プロセス識別子，カーネルスレッド，ならびに権限情報は排他ページ参照許可の識別子として利用する．

カーネルコード・データ，および脆弱性を含むカーネルコードの格納は通常ページに行い，排他ページとして指定される際の仮想アドレスならびに識別子により，参照制限を行うことで該当するカーネルコード・データに対する排他ページとして機能することを可能とする．他のカーネルコードと通常ページの共有を考慮し，排他ページとしたカーネルコードの仮想アドレス以外へのアクセスはページフォルトハンドラやトラップにて参照許可として扱う．

3.5　攻撃発生時の処理

提案手法を適用したカーネルにおけるカーネルへの脆弱性を利用した攻撃に対する処理を挙げる（図 4 を参照）．

脆弱性を含むカーネルコードを利用可能な場合： 脆弱性を含むカーネルコードは通常ページに存在するため，攻撃者の利用するカーネルの仮想記憶空間上において参照可能である．攻撃によりカーネルの仮想記憶空間上に配置された他のカーネルコード・データの改ざんを行える．排他ページとして指定されたカーネルコード・データはカーネルの仮想記憶空間に存在しないため，改ざんは困難である．

脆弱性を含むカーネルコードを利用不可能な場合： 脆弱性を含むカーネルコードは排他ページに指定されるた

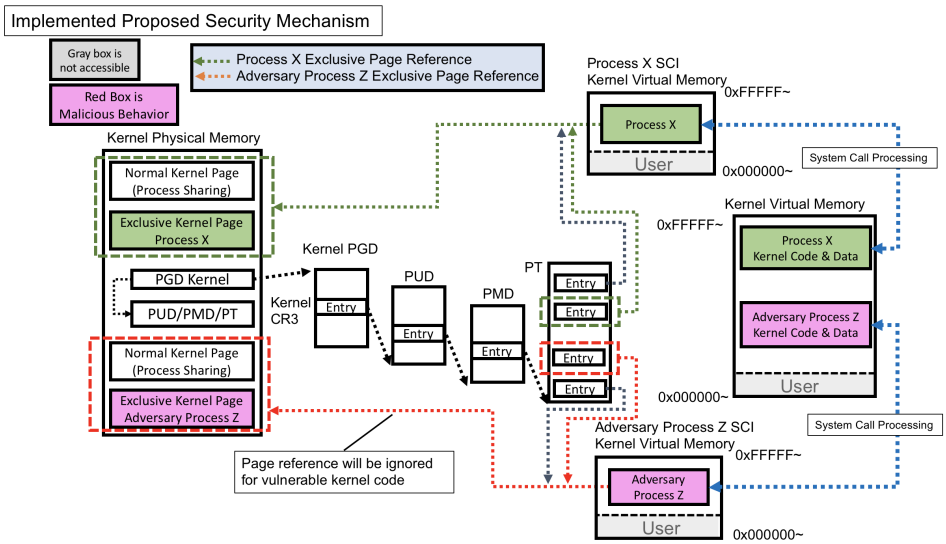

図 5　排他的ページ参照管理の実現図
Fig. 5　The implementation of exclusive page reference mechanism

め，攻撃者の利用するカーネルの仮想記憶空間上においては参照不可能であり，攻撃自体は困難となる．

排他ページへの参照に対しては，ページフォルトを意図的に発生させる．カーネルにおいて，ページフォルトハンドラやトラップにて検出し，排他ページに対応する仮想アドレスへのアクセスを破棄し，該当プロセスへは終了シグナルを発行し終了させる．

3.6　実現方式

提案するセキュリティ機構の実現環境として， OS は KPTI を備え SCI を部分的に導入した Linux とし，CPU アーキテクチャは x86_64 を想定する．

3.6.1　排他的ページ管理の構築

提案手法である排他的ページ管理を Linux に実現した際の概要を図 5 に示す．Linux におけるカーネルの仮想記憶空間は通常ページおよび排他ページから構成される．

実現方式では，カーネルの仮想記憶空間である `init_mm` 構造体の `pgd` 変数をカーネルの仮想記憶空間の初期値とする．排他ページ管理リストを `exclusive_page_list` として，排他ページに登録される仮想アドレスと排他ページ参照可否を判断する識別子の管理を行う．仮想アドレスは，脆弱性を含む，あるいは含まないカーネルコードおよびカーネルデータを示す．識別子は，参照を許可するプロセス番号やカーネルスレッドを示す．

プロセス生成時に作成される仮想記憶空間はカーネル，ユーザ，ならびに SCI 用である．SCI 用の仮想記憶空間は，`mm_struct` 構造体として `pgd` 変数を含む `sci` を `task_struct` 構造体に設置する．実行中のプロセスを示す変数 `current` のシステムコール実行時に SCI 用の仮想記憶空間である `sci` 変数を含む `pgd` 変数を CR3 レジスタに書込み利用する．

3.6.2　排他ページの参照処理

プロセス生成時における排他ページの管理として，カーネルの仮想記憶空間の初期値を SCI 用のカーネルの仮想記憶空間に複製した後，ページテーブルを走査し，排他ページ管理リストを元に排他ページのアンマップ処理を `remove_pagetable` 関数により行い，排他ページへの参照を削除する．同時に，ダイレクトマッピング領域から排他ページへの参照も同時に削除する．

プロセス動作中，他のプロセスにて新たに通常ページから排他ページへと変更された場合，システムコール実行前のカーネル処理を行う前段階にて排他ページ管理リストを元にカーネルの仮想記憶空間を走査し，該当する排他ページのアンマップ処理を行い排他ページへの参照を除外する．

プロセスからのカーネルの仮想記憶空間領域においてマッピングされていない排他ページへの参照は，ページフォルトハンドラ `do_page_fault` または `do_double_fault` にて，特定の仮想アドレスへのアクセスとして捕捉する．排他ページへのアクセスであればアクセス拒否として扱い，必要に応じて `force_sig_info` 関数にて該当プロセスに対して SIGKILL の送信を行う．

3.6.3　排他的ページ管理適用時の割込み処理

Linux では，割込みによりカーネルの処理は非同期的に行われる．この際，実行中のプロセスである `current` 変数のカーネルの仮想記憶空間を利用する．排他ページへの参照を除外した SCI 用のカーネルの仮想記憶空間では，カーネル処理を正しく実行できない可能性がある．割込処理の開始前において，カーネルの仮想記憶空間である `current` 変数の `mm_struct` 構造体である `active_mm` の `pgd` 変数を CR3 レジスタに書込み，仮想記憶空間を切替える．割込み処理後，SCI の仮想記憶空間である `current` 変数の `mm_struct` 構造体である `sci` の `pgd` 変数を CR3 レジスタに書込み，仮想記憶空間を切替える．

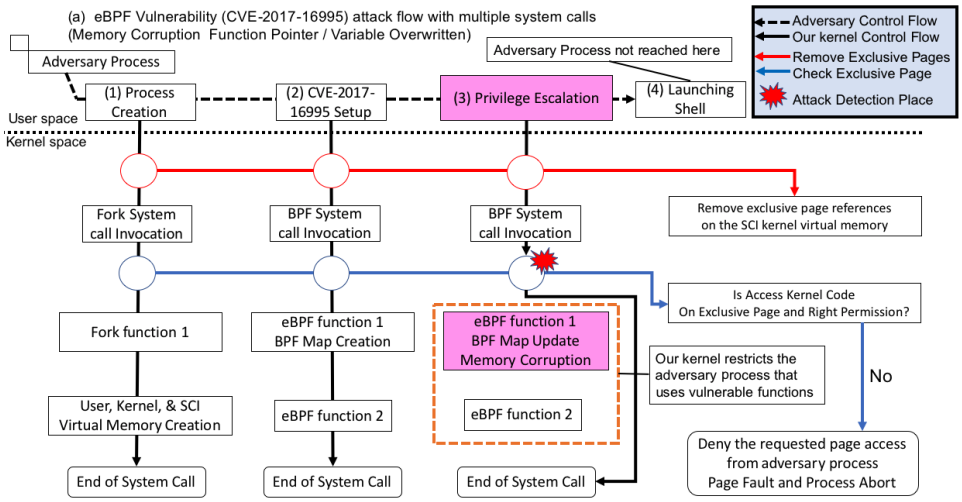

図 6　実際のカーネル脆弱性を利用した攻撃プロセスに対する排他的ページ参照管理の処理
Fig. 6　The handling of exclusive page reference for the actual kernel vulnerability

3.6.4　排他的ページ管理適用時の攻撃例

実現方式において，eBPF に関するカーネル脆弱性 CVE-2017-16995[4] を利用した攻撃を想定する（**図 6** を参照）．CVE-2017-16995 を利用した Proof-of-Concepts (PoC) では，ユーザモードのプロセスから攻撃の起点となるカーネルコード”`kernel/bpf/syscall.c`” に含まれる `map_update_elem` を介して，カーネルの仮想記憶空間の任意の仮想アドレスに対する読書きを可能とし，特権奪取および管理者権限でのシェル起動を行う．

実現方式において，PoC を実行し，あらかじめ排他ページに設定した一部のカーネルデータに対して攻撃を行った場合，ページフォルトが発生し，攻撃対象としたカーネルデータの参照と書換えは困難である．また，PoC の実行時に，脆弱性を含むカーネルコードの関数 `map_update_elem` を排他ページ相当として登録した場合，システムコール `sys_bpf` の実行時にページフォルトが発生する．ページフォルトハンドラにおいて攻撃プロセスに対して SIGKILL を送信するため，攻撃自体を成立させることは困難となる．

4.　評価

4.1　評価の目的と評価環境

提案手法に対する評価目的と内容を以下に示す．

（評価 1）　特定のカーネルデータに対する排他的ページ適用実験
提案手法を適用したカーネルにおいて，特定のカーネルデータを排他ページとして指定しカーネルの仮装記憶空間に配置したカーネルデータへの改ざんへのアクセス可否を評価した．

（評価 2）　脆弱性を含むカーネルコードに対する排他的ページ適用実験
提案手法を適用したカーネルにおいて，実際のカーネル脆弱性を利用した攻撃に対し，カーネルの仮想記憶空間に配置される脆弱性を含むカーネルコードに対して排他的ページ参照を適用し，カーネルの攻撃耐性を評価した．

```
1.  [  414.866573] exclusive_pages_module: module license 'unspecified'
    taints kernel.
2.  [  414.868065] Disabling lock debugging due to kernel taint
3.  [  414.922945] LKM Exclusive Pages Init
4.  [  414.923330] exclusive 1st virtual address (uint): ffffffff81e2a780
5.  [  414.923386] exclusive 1st virtual address (data): tkpt_dummy
6.  [  414.923403] exclusive 2nd virtual address (uint): ffff880001e2a780
7.  [  414.923417] exclusive 2nd virtual address (data): tkpt_dummy

// Add Virtual Address ffffffff81e2a780 to exclusive page management
8.  [  414.923575] calling add_exclusive_page()
9.  [  414.923819] calling exclusive_address_list_add()
10. [  475.104271] 30 secs passed.
11. [  475.104543] exclusive 1st virtual address (uint): ffffffff81e2a780
12. [  475.104585] exclusive 1st virtual address (data): ¥xffffffd¥xff
    ffffd¥xffffffd¥xffffffd¥xffffffd¥xffffffd¥xffffffd¥xffffffd¥xffffffd¥xff ffffd
13. [  475.104612] exclusive 2nd virtual address (uint): ffff880001e2a780
14. [  475.104636] exclusive 2nd virtual address (data): ¥xffffffd¥xff
    ffffd¥xffffffd¥xffffffd¥xffffffd¥xffffffd¥xffffffd¥xffffffd¥xffffffd¥xffffffd
(a) Exclusive page management hides the specific kernel
```

図 7　特定のカーネルデータへの排他的ページ適用後の参照ログ
Fig. 7　Preventing of kernel data access with exclusive page

評価環境は，QEMU（メモリ 512 MB），OS は Debian 9.0 (Linux Kernel 4.4.114, x86_64) とし，SCI は Linux Kernel 4.4.114 向けに移植した．

4.2　特定のカーネルデータに対する排他的ページ適用実験

評価では，特定のカーネルデータを備えた状況において，自作したカーネルモジュールにより，特定のカーネルデータの仮想アドレスを排他ページとして登録する前後においてのデータ出力を行なった．モジュールからのページ参照となるため，排他ページへのアクセスには無効なデータを返す．カーネルの仮想記憶において，“exclusive 1st virtual address (uint)” はカーネルテキスト領域，および “exclusive 2nd virtual address (uint)” はダイレクトマッピング領域を示す．また，“(data)” は仮想アドレスを介して格納された物理記憶空間上のデータを示す．

実験における提案手法を適用したカーネルデータへのアクセス結果出力を**図 7** に示す．ログ出力結果より，図 7 の 4 行目および 7 行目にて特定のカーネルデータ `tkpt_dummy` へのアクセスは正しいデータを得られている．一方，8 行目にモジュールから排他ページ登録が行われたことで，12

```
// PoC code running, process id is 1642
1.  # ./cve-2017-16995
2.  [.]
3.  [.] t(-_-t) exploit for counterfeit grsec kernels such as KSPP  and linux-hardened t(-_-t)
4.  [.]
5.  [.]   ** This vulnerability cannot be exploited at all on authentic  grsecurity kernel **
6.  [.]
7.  [*] creating bpf map
8.  Killed

// Kernel log message
9.  [   91.425545] target system call, pid : 1642
10. [   91.425884] sysnum: 0000000000000141
11. [   91.426426] exclusive_page_code_enable :  1
// ffffffff81131e00 is the virtual address of map_update_elem
12. [   91.426586] remove_exclusive_pages() : ffffffff81131e00
//   0x30 is page fault error code
13. [   91.426925] error_handling() pid: 1642, 0x30
14. [   91.426966] killing target pid: 1642, 0x30
```

(a) Exclusive page management disallow specific kernel code

図 8 特定のカーネルコードへの排他的ページ適用時のログ
Fig. 8 Preventing of kernel code calling with exclusive page

行目および 14 行目においてデータへのアクセスにて得られるデータは隠蔽され，提案手法の適用は正しく行われている．排他ページとされたカーネルデータに対しては，カーネルテキスト領域および，ダイレクトマッピング領域の仮想アドレスを介したページ参照は共に行えない．

4.3 脆弱性を含むカーネルコードに対する排他的ページ適用実験

評価では，予めカーネルコード `map_update_elem` の仮想アドレスを排他ページとして登録した環境において，eBPF のカーネル脆弱性 CVE-2017-16995[4] の PoC による攻撃を実行した．攻撃に対し，カーネルにおいて，システムコール番号，排他ページ処理状況，ページフォルトの捕捉，ならびに攻撃プロセスへの SIGKILL 送信をログ出力した．

実験における攻撃結果ならびにログ出力を**図 8** に示す．ログ出力結果より，攻撃プロセスによる `sys_bpf` システムコールの呼び出しに対応し，図 8 の 12 行目にてカーネルにより排他ページの処理が行われている．その後，ページフォルトの発生を捕捉し，攻撃プロセスに SIGKILL の送信を行なうことで該当プロセスは終了している．提案手法を適用することで，脆弱性を含むカーネルコードを利用させることなく，攻撃は未然に防止されている．

5. 考察

5.1 評価に対する考察

評価結果より，モジュールからの処理にて動作中のカーネルにおいて動的な排他ページの追加および削除を可能であり，実行中のプロセスに対してはシステムコール実行前に SCI を用いて専用のカーネルの仮想記憶空間を設けることにより，SCI 用のカーネルの仮想記憶空間に対して排他ページリストを適時反映できる．

カーネル脆弱性を利用した攻撃プロセスに対しては，システムコール実行前にカーネルの仮想記憶空間から排他ページに登録された脆弱性を含むカーネルコードへの参照を削除している．該当カーネルコードを利用する際に，カーネルにてページフォルトをハンドラおよびトラップにて捕捉することで，カーネルによる攻撃自体の事前防止，ならびに攻撃プロセスを終了可能なことを示した．

今後，システムコール単位でのベンチマークソフトウェアや汎用アプリーションを実行した際のオーバヘッド，ならびに仮想環境およびコンテナへの攻撃に対する提案手法を適用したカーネルの攻撃耐性の評価を進める予定である．

5.2 排他的ページ管理によるカーネルの攻撃耐性の考察

提案手法では，脆弱性を含むカーネルコードを排他ページに登録することでカーネルの攻撃耐性を実現している．攻撃プロセスによる既知のカーネルの脆弱性を利用は未然防止可能であり，パッチ適用前のカーネルにおける脆弱性を含むカーネルコードの制限や，コンテナに対する部分的なカーネルコードの制限にも利用可能と考えている．一方，プロセス生成やネットワーク処理などのカーネルにおいて汎用性の高い機能を提供するカーネルコード，ならびに発見されていない脆弱性を含むカーネルコードについては，事前に排他ページへの登録を行うことは困難である．

特権奪取を行う攻撃においては，脆弱性を含むカーネルコードを起点として `commit_creds`，ならびに `prepare_kernel_cred` 関数などの呼出しを試行される場合がある．予め，権限上昇と関連する関数を排他ページに登録することで，PoC において権限上昇を行う関数を利用した一部の攻撃を軽減することは可能になるといえる．

5.3 排他的ページ管理の制限

実現方式では，KPTI および SCI を備えた Linux に対し，動作中プロセスのシステムコール実行前に，カーネルの仮想記憶空間の排他ページリストの適用処理，ならびにカーネルの仮想記憶空間の切替えに伴うオーバヘッドは存在する．XPFO では，ページ割当時にユーザモードおよびカーネルモードの排他指定を可能としている．性能への影響を低減するためにはカーネルモードを細分化した排他ページ割当を行うことは有効であると考えている．

提案手法の他 OS への適用可能性として，FreeBSD においては仮想記憶空間をページ単位で管理していることから実現可能である [5]．今後，Windows 等での実現性についても検討したい．

5.4 サイバーセキュリティ研究における倫理的配慮

本稿の研究について，コンピュータセキュリティシンポジウム 2019 サイバーセキュリティ研究における倫理的配慮のためのチェックリスト [6] に基づき，情報処理学会 倫理綱領の準拠 [7]，ならびにチェックリストのいずれの項目にも該当しないことを確認した．

6. 関連研究

Linux におけるカーネルの仮想記憶空間での攻撃対策と

して，KPTI[1] はユーザとカーネルの仮想記憶空間を分離した．eXclusive Page Frame Ownership (XPFO) は，ユーザモード用のページをダイレクトマッピングから分離した [8]．ProcLocal[2] は，一部の仮想記憶領域をカーネル機能に排他的に割当て，SCI[3] はシステムコール実行時，Address Space Isolation (ASI) [9] はカーネル機能の実行時に専用の仮想記憶空間を利用可能としている．

物理記憶空間の分離は Trusted Execution Environment（TEE）として CPU 機能を利用してカーネルの管理する物理記憶空間とは異なる物理記憶領域を確保し，カーネルの攻撃による影響を受けない分離した記憶空間での OS やアプリケーションの実行手法が提案されている [10], [11], [12]．

カーネルの仮想記憶空間の保護手法として，ページテーブル位置のランダム化 [13]，ならびに CPU の仮想化機能を利用したカーネル仮想記憶空間の複数分離手法が提案されている [14]．仮想記憶空間上のカーネルコードやデータへの保護として，KHide は仮想化機能を利用した読込み制御，kR^X は読込みと実行の排他制御，および IskiOS は Memory Protection Key を用いたカーネルの仮想記憶空間領域の細粒管理手法を提案している [15], [16], [17]．

カーネルの仮想記憶空間に配置するカーネルコードを削減し攻撃困難化する手法として，kRazor は利用可能なカーネルコードをセキュリティコンテキスト毎にリスト化し管理する [18]，また，KASR はページ単位でカーネルコードの実行属性を制御し，プロセス実行に必要なページのみ有効とする [19]．Multik はアプリケーションに最低限必要なカーネルコードのみ事前にマッピングした仮想記憶空間を構築しプロセスに利用させる [20]．

kMVX は，攻撃検出にカーネル処理に対し，異なる仮想記憶空間領域およびスタックの振舞い用意し，同期的に実行させた際の影響範囲の差異を利用しており [21]，カーネル脆弱性による攻撃の軽減にも有効と言える．

7. おわりに

本稿では，カーネルの仮想記憶空間に対してページ単位の排他的な記憶領域の参照を行う分離手法を提案し，プロセス単位やカーネル機能においてカーネル脆弱性への攻撃耐性を高めた．提案するセキュリティ機構においては，カーネルの仮想記憶空間に通常ページと排他ページを設け，プロセス毎に排他的なページ参照と管理を行うことで，プロセスの利用する特定のカーネル機構のカーネルコードやデータを排他ページとして指定し，他のプロセスによるカーネルの脆弱性を介した攻撃からの保護を可能とした．

提案手法を SCI を備えた Linux にて実現し，評価において，排他ページとして指定したカーネルデータへのアクセスを制御することで特定のカーネルデータへの参照の拒否，ならびにカーネル脆弱性を利用した攻撃に対し，排他ページとした脆弱性を含むカーネルコードの呼出しを制限し，攻撃を事前防止可能なことを示した．

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] Lipp, M., et al.: KASLR is Dead - Long Live KASLR, ESSoS 2017.
[2] Hillenbrand, M., et al.: Process-local memory allocations for hiding KVM secrets, https://lwn.net/Articles/791069/, (2019). (accessed 2019-08-08).
[3] Rapport, M., et al.: [RFC PATCH 0/7] x86: introduce system calls address space isolation, https://lwn.net/ml/linux-kernel/1556228754-12996-1-git-send-email-rppt@linux.ibm.com/, (2019). (accessed 2019-08-08).
[4] CVE-2017-16995, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-16995, (accessed 2019-08-08).
[5] The FreeBSD Documentation Project, : FreeBSD Architecture Handbook, https://www.freebsd.org/doc/en_US.ISO8859-1/books/arch-handbook/, (accessed 2019-08-08).
[6] CSS 2019 サイバーセキュリティ研究における倫理的配慮のためのチェックリスト, https://www.iwsec.org/css/2019/ethics_list.html, (accessed 2019-08-08).
[7] 情報処理学会倫理綱領, https://www.ipsj.or.jp/ipsjcode.html, (accessed 2019-08-08).
[8] Kemerlis, P, V., et al.: ret2dir - Rethinking Kernel Isolation, USENIX Security 2014.
[9] Marsden, G.: Improve Security with Address Space Isolation, https://blogs.oracle.com/linux/improve-security-with-address-space-isolation-asi, (2019). (accessed 2019-08-08).
[10] X. Ge., et al.: Sprobes: Enforcing Kernel Code Integrity on the TrustZone Architecture, arXiv:1410.7747v1, 28-Oct-2014.
[11] Lee, D., et al.: Keystone: A Framework for Architecting TEEs, arXiv:1907.10119v1, 23-Jul-2019.
[12] Melara, S, M., et al.: EnclaveDom: Privilege Separation for Large-TCB Applications in Trusted Execution Environments, arXiv:1907.13245v1, 30-Jul-2019.
[13] Davi, L., et al.: PT-Rand: Practical Mitigation of Data-only Attacks against Page Tables, NDSS 2016.
[14] Hua. Z., et al.: EPTI - Efficient Defence against Meltdown Attack for Unpatched VMs, USENIX ATC 2018.
[15] Gionta, J., et al.: Preventing kernel code-reuse attacks through disclosure resistant code diversification, CNS 2016.
[16] Pomonis, M., et al.: kR^X: Comprehensive Kernel Protection against Just-In-Time Code Reuse, EuroSys 2017.
[17] Gravani, S., et al.: IskiOS: Lightweight Defense Against Kernel-Level Code-Reuse Attacks, arXiv:1903.04654v1, 11-Mar-2019.
[18] K. Anil., et al.: Quantifiable Run-Time Kernel Attack Surface Reduction, DIMVA 2014.
[19] Zhang, Z., et al.: KASR: A Reliable and Practical Approach to Attack Surface Reduction of Commodity OS Kernels, RAID 2018.
[20] Kuo, H, C., et al.: MultiK: A Framework for Orchestrating Multiple Specialized Kernels, arXiv:1903.06889v1, 16-Mar-2019.
[21] Österlund, S., et al.: kMVX: Detecting Kernel Information Leaks with Multi-variant Execution, ASPLOS 2019.