CL-002

カーネル脆弱性を利用した攻撃に対する仮想記憶空間の切替え処理の保護と改ざん検出

Kernel Virtual Memory Switching Protection for Illegal Memory Corruption via Kernel Vulnerability

葛野 弘樹[1)2)]　山内 利宏[1)]
Hiroki Kuzuno　Toshihiro Yamauchi

1　はじめに

オペレーティングシステム（OS）カーネルへの攻撃として，カーネル脆弱性を起点とした任意のプログラムコードの実行により，攻撃者は OS の特権を奪取可能となるため，攻撃への対策が求められている．

カーネルにおける攻撃対策として，カーネルコード実行時のスタック領域の監視 [1]，カーネルの仮想記憶空間に配置される特権情報の監視 [2]，ROP（Return Oriented Programming）対策としてカーネルコードの呼び出し制御フロー検証 [3]，システムコール処理用にカーネルの仮想記憶空間を生成する SCI（System Call Isolation）[4]，および，カーネルコードとデータのカーネルの仮想記憶空間における配置をランダム化する KASLR (Kernel Address Space Layout Randomization) がある [5]．また，Meltdown によるカーネルへのサイドチャネル攻撃に対し，ユーザモードとカーネルモードの仮想記憶空間分離手法が提案され，Linux KPTI （Kernel Page Table Isolation）[6] など各種 OS に導入された．

カーネル脆弱性を利用した攻撃による特権奪取ではカーネルの仮想記憶空間上の権限情報を改ざんする．さらに，管理者権限への制限回避のため強制アクセス制御などのカーネルの提供するセキュリティ機構の無効化を試みる [7, 8]．管理者権限の最小化や権限制限による被害低減の維持には，カーネルの仮想記憶空間上に配置されセキュリティ機構を提供するカーネルコード・データの攻撃からの保護と早期の改ざん検出は重要である．

本稿では，カーネル脆弱性を利用した攻撃によるカーネルの提供するセキュリティ機構の無効化への対策として，カーネルにおいて複数の仮想記憶空間を用いた攻撃困難化手法を提案する．具体的な提案手法は以下の通りである：

- 提案手法によるカーネルのセキュリティ機構の保護として，カーネルに中継用，セキュリティ用，ならびにカーネルの仮想記憶空間を設ける．セキュリティ機構への攻撃を困難化するため，カーネル機能の多くはカーネルの仮想記憶空間上に配置し，カーネルの提供するセキュリティ機構は中継用，ならびにセキュリティ用の仮想記憶空間に配置可能とする．セキュリティ機構の無効化を目的としたカーネルへの攻撃に対して，カーネルの仮想記憶空間上にあるセキュリティ機構のみ改ざん可能にすることで，中継用，ならびにセキュリティ用の仮想記憶空間に配置されたカーネルコード・データへの攻撃を困難化し，セキュリティ機構への影響を低減する．

提案手法を KPTI，SCI，ならびに我々の開発するカーネルの仮想記憶空間を監視するセキュリティ機構 [9, 10] を導入した Linux に実現し，実際の脆弱性および我々のセキュリティ機構によるカーネル監視にて，既存のセキュリティ機構の改ざん検出の評価，ならびにカーネルの仮想記憶空間の切替え処理への攻撃検出の評価を行った．本稿での研究貢献ならびに得られた結果は以下の通りである：

1. カーネルに新たに中継用，セキュリティ用，ならびにカーネルの仮想記憶空間を設けた場合においての仮想記憶空間の切替え方式を示した．セキュリティ機構を提供するカーネルコードを中継用，セキュリティ用の仮想記憶空間に配置し，改ざん対象のカーネルの仮想記憶空間から分離することで攻撃の困難化を実現した．また，提案手法を適用した環境にてカーネル監視のセキュリティ機構を動作させ，カーネルの仮想記憶空間への攻撃を検出できることを示した．
2. Linux にて KPTI，SCI，ならびに提案を組合せた実現方式，および eBPF のカーネル脆弱性 CVE-2017-16995[11] を利用し，SELinux[12] への攻撃に対する我々のセキュリティ機構による攻撃検出能力ならびにカーネルの仮想記憶空間の切替え処理の攻撃耐性を評価した．

2　背景知識

2.1　カーネル脆弱性を利用した攻撃

カーネルへの攻撃の主な目的は OS のみ読書き可能なカーネルの仮想記憶空間においてユーザから管理者権限へ変更する特権奪取がある．また，特権を制限するセキュリティ機能の無効化もあげられる．これらの実現のため，カーネルに対して任意のプログラムコードを挿入し，実行する際にカーネル脆弱性を利用する．

Linux での特権奪取攻撃では，管理者への権限変更を行うためにカーネルコードのうち `commit_creds`，ならびに `prepare_kernel_cred` 関数の強制的な呼出し，あるいは権限情報を格納した `cred` 構造体の `uid` 変数を直接改ざんすることで行う．Linux のセキュリティ機構を回避するための無効化攻撃では，SELinux 等の強制アクセス制御を提供するための LSM（Linux Security Module）フックのカーネルコードを管理する `security_hook_list` 変数を直接書換え，呼出すカーネルコードを改ざん，あるいはセキュリティコンテキストの変数を書換えることで行う．

2.2　仮想記憶空間の分離

特権奪取ならびにセキュリティ機構の無効化においては，改ざん対象とな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・データの仮想アドレスを特定する必要がある．ユーザプロセスからカーネルの仮想記憶空間への参照は CPU およびカーネルにて特権レベルの保護により禁止されており，さらに KASLR を利用した

1）　岡山大学 大学院 自然科学研究科
2）　セコム株式会社 IS 研究所

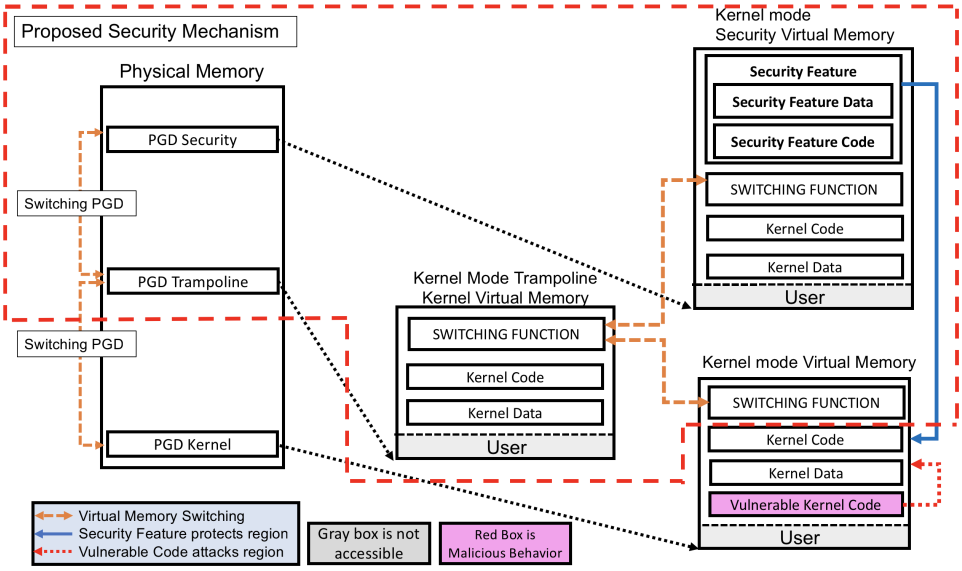

図 1　複数の仮想記憶空間の切替えとカーネル処理

カーネルでは攻撃に利用するためのカーネルコード・データの仮想アドレスの特定は困難とされていた．

カーネルへの Meltdown サイドチャネル攻撃は，本来は許可されていないユーザプロセスからカーネルの仮想記憶空間を参照し，カーネルコード・データの仮想アドレスを特定し KASLR を回避できることを示した．これにより，カーネル脆弱性を利用した攻撃時に ROP によるカーネルコードの断片的組合わせから任意のプログラムを構築可能となる．Meltdown 対策として，ユーザモードで動作するプロセス向けの仮想記憶空間（ユーザの仮想記憶空間）とカーネルモードで動作するカーネル向けの仮想記憶空間（カーネルの仮想記憶空間）の分離手法が提案され，Linux では KPTI として導入された [6]．また，カーネルコードを利用した ROP の対応手法として，プロセス毎にシステムコール処理用のカーネルの仮想記憶空間を生成し最小限のカーネルコードのみ利用可能とする実装が SCI として提案されている [4]．

2.3　想定する脅威モデル

本稿における脅威モデルは，攻撃者によるカーネル脆弱性を利用した攻撃によるカーネルの仮想記憶空間の改ざんとする．攻撃においては，カーネル脆弱性を含むカーネルコードの配置される同一の仮想記憶空間のみ書換え可能とする．攻撃時に指定できる仮想アドレスの範囲は，既存のセキュリティ機構，ならびに提案手法における仮想記憶空間の切替え処理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　提案手法における複数の仮想記憶空間

カーネル脆弱性を利用してカーネルのセキュリティ機構を無効化することで管理者制限などセキュリティ機能を回避できる可能性はある．その課題は次の通りである．

課題：　カーネルのセキュリティ機構においては，セキュリティ機能の提供を行うカーネルコード・データはカーネルの仮想記憶空間に配置される．カーネル脆弱性を利用し，セキュリティ機能を提供するカーネルコード・データを無効化することでセキュリティ機能に関するカーネルコードの処理，あるいはデータを改ざんすることによりセキュリティ機能を回避される可能性がある．

本稿では，カーネル脆弱性を利用した攻撃からカーネルのセキュリティ機構への攻撃を困難化する手法として，カーネルに対して，カーネル，中継用，ならびにセキュリティ用の 3 つの仮想記憶空間を設ける新たなセキュリティ機構を提案する（図 1 を参照）．カーネルモードで行われる処理はカーネルにおいて適切な仮想記憶空間に切替えた後に行われる．それぞれの仮想記憶空間において処理される機能は次の通りである．

カーネル：　システムコールからの要求，割込み要求，ならびにカーネル脆弱性を利用した攻撃の対象となるカーネル機能を処理する仮想記憶空間

中継用：　ユーザモードとカーネルモード間の遷移時，ならびにカーネル・セキュリティ用の仮想記憶空間の切替え先となる仮想記憶空間

セキュリティ用：　カーネルのセキュリティ機能を処理する仮想記憶空間

3.2　仮想記憶空間の切替え処理

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カーネル，中継用，ならびにセキュリティ用の間で切替えを行う．ユーザモードからカーネルモードへの遷移とカーネル提供機能の処理との関連により，切替えパターンを挙げる．

パターン 1：　システムコール発行や割込み要求時のユーザモードからカーネルモードへの遷移時，ユーザの仮想記憶空間から中継用の仮想記憶空間を経て，カーネルの仮想記憶空間に切替え，対応するカーネルコードを実行．

パターン 2：　カーネルモードからユーザモードへの遷移時，カーネルの仮想記憶空間から中継用の仮想記憶空間を経て，ユーザの仮想記憶空間に切替える．

パターン 3：　セキュリティ機能の処理開始時，システムコール処理前は中継用の仮想記憶空間からセキュリティ用の仮想記憶空間に切替え，

カーネル処理中またはシステムコール処理後はカーネルの仮想記憶空間から中継用の仮想記憶空間を経て，セキュリティ用の仮想記憶空間に切替え対応するカーネルコードを実行．

仮想記憶空間の切替え処理は各仮想記憶空間に配置した専用の Switching Function にて管理する．中継用の仮想記憶空間からはカーネルならびにセキュリティ用の仮想記憶空間に切替え可能であるが，カーネルならびにセキュリティ用の仮想記憶空間は中継用の仮想記憶空間にのみ切替え可能とする．また，カーネルモードとユーザモード間の遷移時は，中継用の仮想記憶空間とユーザの仮想記憶空間の間でのみ切替えを行う．

3.3 仮想記憶空間の切替え処理への攻撃

カーネルモードでの攻撃は脆弱性を含むカーネルコードと同一の仮想記憶空間上にのみ影響を及ぼす．カーネル機能に脆弱性の存在を想定した場合，カーネル脆弱性を利用した攻撃はカーネルの仮想記憶空間で処理され，カーネルの仮想記憶空間に配置されたカーネルコード・データのみ攻撃可能となる．そのため，攻撃によるセキュリティ機能の回避には，セキュリティ用の仮想記憶空間への切替えを全て阻止する必要がある．カーネル脆弱性を利用した攻撃により，カーネルの仮想記憶空間上の仮想記憶空間の切替え処理を無効化された場合，カーネル処理中およびシステムコール処理後の監視処理は回避可能である．

中継用およびセキュリティ用の仮想記憶空間はカーネルの仮想記憶空間上の攻撃の影響は受けない．システムコールの処理前など，中継用，セキュリティ用のみでカーネルの仮想記憶空間を介さないタイミングでセキュリティ機能を実行した場合，攻撃によるセキュリティ機能の回避は困難である．

攻撃により仮想記憶空間への切替え処理が不安定となった場合，ユーザプロセスの処理やカーネル動作に影響し，攻撃後のプロセスまたはカーネル処理の継続は困難となる可能性がある．

3.4 実現方式

提案するセキュリティ機構を実現する環境として，OS は KPTI を備え SCI を導入した Linux，CPU アーキテクチャは x86_64 を想定する．

3.4.1 複数の仮想記憶空間の構築

提案手法である複数の仮想記憶空間を Linux にて実現した際の概要を図 2 に示す．提案手法において，プロセスはユーザ，カーネル，中継用，セキュリティ用の 4 つの仮想記憶空間を利用する．Linux KPTI は，プロセス毎にユーザおよびカーネルの仮想記憶空間を作成し，SCI は，プロセス毎にシステムコール処理用の仮想記憶空間を作成する．KPTI でのユーザの仮想記憶空間（KPTI User mode Virtual Memory）は従来のままとし，カーネルの仮想記憶空間は中継用（KPTI Kernel mode Virtual Memory）とする．SCI により追加される仮想記憶空間（Kernel mode Virtual Memory）はカーネル機能の処理を行う．セキュリティ用の仮想記憶空間（Kernel mode Monitoring Virtual Memory）はセキュリティ機能として我々のカーネル監視を行うセキュリティ機構 [9, 10] を処理し，他プロセスと共有する．

実現方式では，KPTI でのカーネルの仮想記憶空間

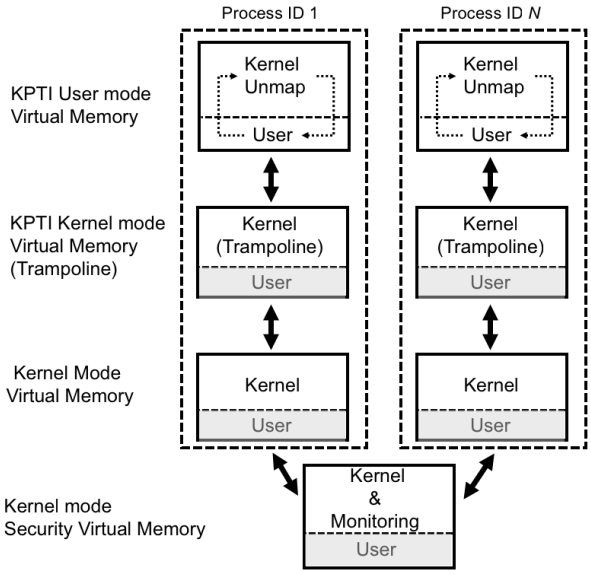

図 2　複数の仮想記憶空間の切替えの概要

である init_mm 構造体の pgd 変数を中継用の仮想記憶空間の初期値とする．ユーザの仮想記憶空間の初期値を init_mm 構造体の pgd 変数から 1 ページサイズ（x86_64 では 4Kbytes）論理積した物理アドレスに配置する．セキュリティ用の仮想記憶空間は，init_mm 構造体の pgd 変数から 4 ページサイズ（x86_64 では 16Kbytes）論理積した物理アドレスに配置する．カーネルの仮想記憶空間は，SCI にて task_struct 構造体に追加される mm_struct 構造体の kernel_pgd 変数とする．

プロセス生成時に作成される仮想記憶空間は中継用，カーネル，ユーザ用である．current 変数の task_struct 構造体に含まれる mm_struct 構造体の pgd 変数を中継用の仮想記憶空間，kernel_pgd 変数をカーネルの仮想記憶空間とし，pgd 変数の物理アドレスから 1 ページサイズ論理積した物理アドレスをユーザの仮想記憶空間とし，それぞれ初期値より複製する．

3.4.2 仮想記憶空間の切替え処理

実現方式における，カーネルの仮想記憶空間の切替えはそれぞれの仮想記憶空間に配置されたカーネルコードにより行う（図 3 を参照）．仮想記憶空間の切替え処理と各パターンについて説明する．

パターン 1：ユーザモードからカーネルモードへの遷移後，ユーザの仮想記憶空間の SWITCH_KPTI_CR3 にて，ユーザの仮想記憶空間の物理アドレスから 1 ページサイズ排他的論理和した値を CR3 レジスタに書込み，中継用の仮想記憶空間に切替える．中継用の仮想記憶空間の SWITCH_KERNEL_CR3 にて，current 変数の kernel_pgd 物理アドレスを CR3 レジスタに書込み，カーネルの仮想記憶空間に切替える．

パターン 2：カーネルモードからユーザモードへの遷移の前に，カーネルの仮想記憶空間の SWITCH_KERNEL_CR3 にて，current 変数に含まれる pgd 変数の物理アドレスを CR3 レジスタに書込み，中継用の仮想記憶

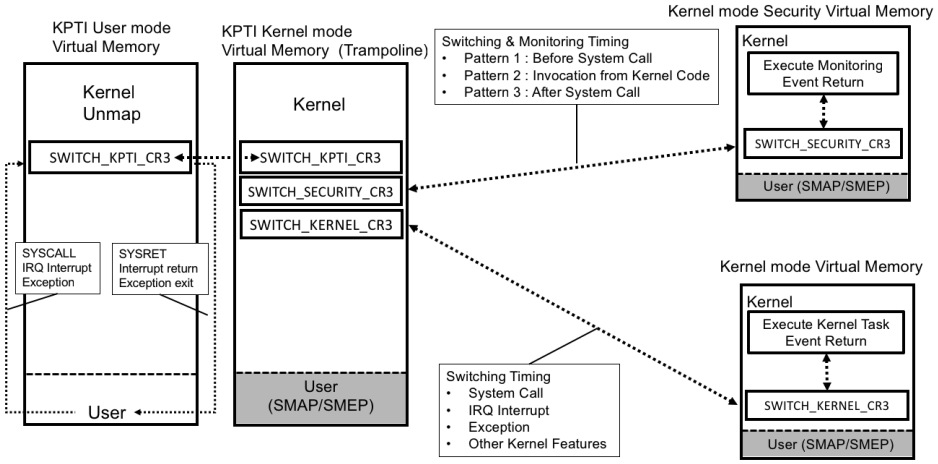

図 3　複数の仮想記憶空間の切替えタイミング

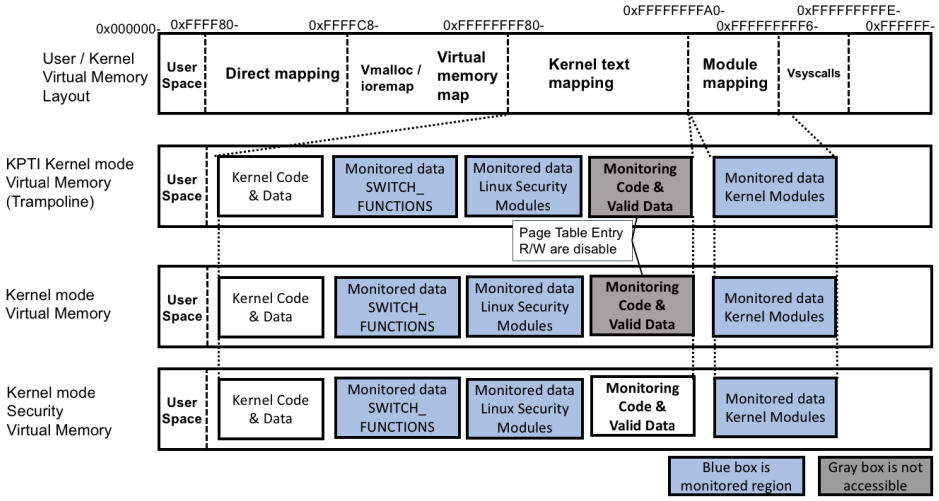

図 4　実現方式において監視対象とした仮想記憶空間のカーネルコード

空間を切替える．中継用の仮想記憶空間の SWITCH_KPTI_CR3 にて，中継用の仮想記憶空間の物理アドレスから 1 ページサイズ論理積した値を CR3 レジスタに書込み，ユーザの仮想記憶空間に切替える．

パターン 3：　システムコール処理前のセキュリティ機能の実行は，中継用の仮想記憶空間の SWITCH_SECURITY_CR3 にて，init_mm の pgd 変数から 4 ページサイズ論理積した物理アドレスを CR3 アドレスに書込み，セキュリティ用の仮想記憶空間に切替え行う．カーネル処理中およびシステムコール処理後のセキュリティ機能の実行はカーネルの仮想記憶空間の SWITCH_KERNEL_CR3 にて，中継用の仮想記憶空間に切替え，その後，セキュリティ用の仮想記憶空間に切替え行う．

3.4.3　仮想記憶空間の切替え処理の監視

実現方式においては，従来のカーネルモジュール，Linux での強制アクセス制御を提供する LSM に加え，カーネルの仮想記憶空間における仮想記憶空間の切替え処理を監視し保護対象とした（図 4 を参照）．

実現方式における監視に利用する正当データ作成の手順では，まず，Linux カーネルの起動時，kaiser_init 関数の終了後において，KPTI のユーザ，カーネルの仮想記憶空間が作成され，カーネルコード・データの仮想アドレスの位置を定める．その後，中継用の仮想記憶空間，セキュリティ用の仮想記憶空間の領域を物理記憶空間に確保し，カーネルの仮想記憶空間上に配置される，カーネルモジュールのリスト，LSM および仮想記憶空間の切替え処理を行う SWITCH_KERNEL_CR3 の仮想アドレスをセキュリティ用の仮想記憶空間の領域に正当なデータとして複製する．

我々のセキュリティ機構におけるカーネル監視を行うタイミングは，システムコールの実行前と実行後，およびカーネルの処理中における任意のタイミングとしている．監視処理においては，セキュリティ用の仮想記憶空間に切替え後，カーネルモジュール，LSM を呼出すカーネルコードの仮想アドレス，ならびに仮想記憶空間の切替え処理の仮想アドレスに対して改ざんの有無を正当データと比較することで判断する．

3.4.4　仮想記憶空間の切替え処理への攻撃

実現方式のカーネルモードで動作させる仮想記憶空間は中継用，セキュリティ用，カーネルである．中継用の仮想記憶空間は SWITCH_KPTI_CR3，SWITCH_SECURITY_CR3，SWITCH_KERNEL_CR3，ならびに割込みや例外捕捉関連のカーネルコード

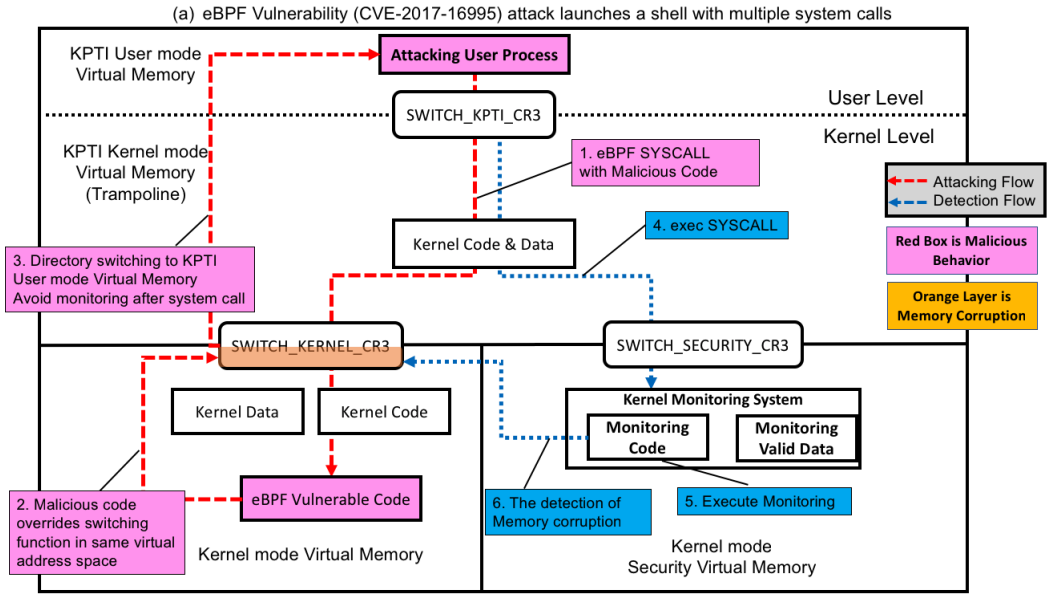

図 5　カーネル脆弱性を利用した仮想記憶空間の切替え処理への攻撃と検出例

を実行する．セキュリティ用の仮想記憶空間は SWITCH_SECURITY_CR3，ならびに監視処理用のカーネルコードを実行する．カーネルの仮想記憶空間は，SWITCH_KERNEL_CR3，ならびにその他のカーネル機能を提供するカーネルコードを実行する．

カーネルモードの処理毎に仮想記憶空間を分離して管理し，カーネルコード・データはそれぞれの配置された仮想記憶空間にて実行・読書きする．カーネルコードに脆弱性が存在する場合，該当する仮想記憶空間上にマッピングされているカーネルコード・データのみ攻撃対象となる．実現方式においては，殆どのカーネル機能はカーネルの仮想記憶空間にマッピングされるため，セキュリティ機能の回避のためにカーネル脆弱性を利用した攻撃による攻撃対象はカーネルの仮想記憶空間上にある仮想記憶空間の切替えを行う SWITCH_KERNEL_CR3 となる．

セキュリティ機構の監視回避として，カーネルの仮想記憶空間に配置される SWITCH_KERNEL_CR3 に対して攻撃を巧妙に行なった場合，カーネルの仮想記憶空間から直接ユーザの仮想記憶記憶空間に切替え，ユーザモードに遷移することは可能といえる．カーネルの仮想記憶空間の SWITCH_KERNEL_CR3 を改ざんした場合，セキュリティ機構の監視のうちカーネル処理中，ならびにシステムコール処理後の監視処理は回避される．しかしながら，中継用，セキュリティ用の仮想記憶空間は攻撃の影響を受けないため，システムコール処理前の監視処理は回避されない．

実現方式を備えたカーネルへの攻撃では，セキュリティ機構における全ての仮想記憶空間の切替え処理は改ざんを困難とし，部分的な保護を実現している．我々のカーネル監視機構の行う監視タイミングにおいても攻撃による全ての監視処理の回避は行えない．攻撃による SWITCH_KERNEL_CR3 の改ざんにより，カーネルの仮想記憶空間においてカーネルパニック等を発生した際は，カーネルモードからユーザモードに遷移できず攻撃は失敗する．

3.4.5　仮想記憶空間の切替え処理への攻撃例

eBPF のカーネル脆弱性 CVE-2017-16995[11] を利用した攻撃では，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/verifier.c の存在する仮想記憶空間の任意の仮想アドレスに対して，本来は許可されていない読書きを可能とする．

実現方式においてカーネル脆弱性 CVE-2017-16995 を利用して攻撃を試みる場合，カーネルコード kernel/bpf/verifier.c の実行はカーネルの仮想記憶空間となり，同一の仮想記憶空間にある仮想記憶空間の切替え処理を行う SWITCH_KERNEL_CR3 改ざん可能である（図 5 を参照）．

カーネルの仮想記憶空間における攻撃時には，中継用およびセキュリティ用の仮想記憶空間の仮想アドレスは指定できないため，それらの仮想記憶空間に配置されるカーネルコード・データの改ざんは困難である．このため，攻撃後に新たなシステムコールが発行される際にはシステムコール処理前のカーネル監視機構における監視処理は回避されない．中継用の仮想記憶空間からセキュリティ用の仮想記憶空間に切替え，監視処理を実行することでカーネルの仮想記憶空間に存在する SWITCH_KERNEL_CR3 を呼出すカーネルコードの改ざん有無は検出可能である．

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の目的と内容を以下に示す．

（評価 1）　既存のセキュリティ機能への攻撃監視実験
カーネル脆弱性を利用した攻撃により，カーネルの提供する強制アクセス制御を管理する LSM を改ざんし，検出可能か評価，また，攻撃検出にかかる時間を測定した．

（評価 2）　仮想記憶空間の切替え処理への攻撃監視実験
カーネル脆弱性を利用した攻撃により，カーネルの仮想記憶空間に配置された仮想記憶空間の切替え処理を行うカーネルコードを改ざんし，攻撃による影響ならびに検出可能かを評価

```
1.  // Install LKM
2.  [   78.654425] lkm_address_module: module license 'unspecified' taints kernel.
3.  [   78.654853] Disabling lock debugging due to kernel taint
4.  [   78.718459] dummy_hook_function Address Value ffffffffa0000000
5.  [   78.718427]  selinux_hooks[56].hook.file_permission pointer Address ffffffff81e77c18
6.  [   78.718444] selinux_hooks[56].hook.file_permission pointer Address Value ffffffff812f3f20

7.  // Switching to Secret virtual memory and monitoring
8.  [  100.767961] Address ffffffff812f3f20 (Valid), ffffffff812f3f20 (Invalid)
9.  // Switching to KPTI Kernel virtual memory (Trampoline)

10. // Switching to Kernel virtual memory
11. // CVE-2017-16995 attack overrides LSM function address via sys_bpf()
12. // Switching to KPTI Kernel virtual memory (Trampoline)

13. // Switching to Secret virtual memory and monitoring after sys_bpf()
14. [  100.772834] Invalid lsm function is detected
15. [  100.772854] Address ffffffff812f3f20 (Valid), ffffffffa0000000 (Invalid)
16. // Switching to KPTI Kernel virtual memory (Trampoline)

17. // LKM automatically outputs the target function virtual address
18. [  108.769291] selinux_hooks[56].hook.file_permission pointer Address Value ffffffffa0000000
```

(a) Monitoring of Linux Security Module's Function

図 6　CVE-2017-16995 を利用した既存のセキュリティ機構への攻撃検出時のログ

した．

評価用計算機は，Intel(R) Core(TM) i7 7700HQ (2.80GHz，4 コア，メモリ 16GB，OS は Debian 9.0 (Linux Kernel 4.4.114, x86_64) とした．SCI は Linux Kernel 5.X 系列へのパッチのため 4.4.114 にて動作するよう移植した．また，CVE-2017-16995 の PoC コードは特権奪取のみ行うため，任意の仮想アドレスへの書込みに対応した．

4.2　既存のセキュリティ機能への攻撃監視実験

評価では，eBPF のカーネル脆弱性 CVE-2017-16995[11] を利用し，ユーザプロセスから強制アクセス制御を提供する LSM のフック関数をあらかじめ用意したカーネルモジュールのカーネルコードのアドレスに上書きした．攻撃に対し，カーネル監視機構によりセキュリティ用の仮想記憶空間において正当なデータと比較し，差異の検出結果をログ出力することで行った．正規の関数アドレスはカーネル起動時にセキュリティ用の仮想記憶空間に保持させ，システムコール処理の前後に監視処理行う．

攻撃では，`sys_bpf` システムコール処理時に `selinux_hooks` 変数に格納される SELinux カーネルコードを呼出す仮想アドレスの一つをカーネルモジュールのカーネルコードの仮想アドレスにて上書きし改ざんする．差異検出時のログでは，正当なデータと異なるカーネルコードのアドレスの検出時に “Invalid lsm function is detected”，および “Virtual Address (Invalid)” と文字列表示する．

実験における提案手法による差異の検出時のログ出力を図 6 に示す．ログ出力結果より，カーネル脆弱性を利用した攻撃による既存のセキュリティ機構の改ざんに対し提案手法において動作させたカーネル監視機構による検出は正しく行われている．攻撃はカーネルの仮想記憶空間において発生し，その後，中継用を経てセキュリティ用の仮想記憶空間への切替えた後に攻撃を検出している．システムコール発行毎に監視を行うため，検出時の監視インターバルは 0.0049 ms であり，その間に発生した攻撃による改ざんとその検出を行なっている．

4.3　仮想記憶空間の切替え処理への攻撃監視実験

仮想記憶空間の切替え処理への攻撃監視の評価では，既存のセキュリティ機構への攻撃と同様にユーザプロセスからカーネル脆弱性 CVE-2017-16995[11] を利用した．攻撃対象は，カーネルの仮想記憶空間に配置

```
1.  // Install LKM and
2.  [  105.738923] lkm_address_module: module license 'unspecified' taints kernel.
3.  [  105.739633] Disabling lock debugging due to kernel taint
4.  [  105.802694] dummy_hook_function Address Value ffffffffa0000000
5.  [  105.802637] vmem_switching_function pointer Address ffffffff821257e0
6.  [  105.802657] vmem_switching_function pointer Address Value ffffffff810593e0

7.  // Switching to Secret virtual memory and monitoring
8.  [  159.480809] Address ffffffff810593e0 (Valid), ffffffff810593e0 (Invalid)
9.  // Switching to KPTI Kernel virtual memory (Trampoline)

10. // Switching to Kernel virtual memory
11. // CVE-2017-16995 attack overrides the virtual memory switching function address
    via sys_bpf
12. // Switching to KPTI Kernel virtual memory (Trampoline)

13. // Switching to Secret virtual memory and monitoring at next system call
14. [  159.484758] Invalid vmem switching function is detected
15. [  159.484776] Address ffffffff810593e0 (Valid), ffffffffa0000000 (Invalid)
16. // Switching to KPTI Kernel virtual memory (Trampoline)

17. // LKM automatically outputs the target virtual address
18. [  165.857354] vmem_switching_function pointer Address Value ffffffffa0000000
```

(b) Monitoring of Virtual Memory Switching Function

図 7　CVE-2017-16995 を利用した仮想記憶空間の切替え処理への攻撃検出時のログ

された仮想記憶空間の切替えを行うカーネルコード `SWITCH_KERNEL_CR3` の呼出しに利用する仮想アドレスとした．

攻撃においては，攻撃対象の仮想アドレスをあらかじめ用意したカーネルモジュールのカーネルコードの仮想アドレスに上書きする．これにより，カーネルの仮想記憶空間から中継用とセキュリティ用の仮想記憶空間を介さずに直接ユーザの仮想記憶空間へ切替え，提案手法におけるシステムコール終了後の監視回避を試みる．差異検出時のログとして，正当なデータと異なるカーネルコードのアドレスの検出時に “Invalid vmem switching function is detected”，および “Virtual Address (Invalid)” と文字列表示する．

提案手法を適用したカーネルにおけるログ出力を図 7 に示す．ログ出力結果より，`SWITCH_KERNEL_CR3` 呼出しに用いる仮想アドレスの改ざんに対し提案するセキュリティ機構の検出は正しく行われている．ログ出力において，攻撃対象であるカーネルの仮想記憶空間上にある `SWITCH_KERNEL_CR3` を呼出す際のカーネルコードの仮想アドレスはカーネルモジュールの仮想アドレスに改ざんされたことを確認した．

攻撃により，カーネルから中継用，セキュリティ用の仮想記憶空間へ切替えられることなくカーネルの処理は進み，カーネルの仮想記憶空間からユーザの仮想記憶空間に直接切替り，カーネルモードからユーザモードへと遷移している．提案手法を用いたカーネル監視機構において，システムコール終了後の監視処理は回避されたものの，中継用，セキュリティ用の仮想記憶空間は攻撃による影響を受けていないことから，攻撃後のシステムコール処理前に，ユーザ，中継用，セキュリティ用の仮想記憶空間への切替えと監視処理は正常に動作し，監視インターバル 0.0039 ms の間に発生した攻撃による改ざんと検出は正しく行われたことを確認した．

5　考察

5.1　評価に対する考察

提案手法を用いた環境にて，カーネル脆弱性を利用した攻撃による既存のセキュリティ機構呼出し部分の改ざん，ならびに実現方式の用いる仮想記憶空間の切替え関数の改ざんを適切に検出できることを確認した．カーネルの仮想記憶空間を複数設けた場合においても，セキュ

リティ用の仮想記憶空間にてカーネル監視のセキュリティ機構を動作させ監視処理を行えている．

既存のセキュリティ機構への攻撃監視では，改ざんに利用したシステムコールを終了した直後のタイミングにおいて，カーネルから中継用，セキュリティ用の仮想記憶空間へと切替え，セキュリティ機構のカーネル監視処理を実行することで改ざんを検出可能である．仮想記憶空間の切替え関数への攻撃監視では，改ざんの影響により，カーネルからユーザの仮想記憶空間へと直接切替えられ，改ざんに利用したシステムコールの終了後の監視処理は回避された．しかし，攻撃成功以降に発行されるシステムコールの処理前において，中継用からセキュリティ用の仮想記憶空間への切替え，カーネル監視処理は機能しているため改ざんを検出可能なことを示した．

今後，ベンチマークソフトウェアや Web サーバなどの汎用アプリーションを実行した際のオーバヘッドや攻撃による仮想記憶空間改ざん可能な範囲の限定化と検出可能なタイミングの評価を進める予定である．

5.2 カーネルコードの保護に対する考察

評価においては，CVE-2017-16995 を用いて，既存のセキュリティ機構である LSM ならびに提案手法における仮想記憶空間の切替え処理の呼出しを無効化し，カーネル監視を行うセキュリティ機構の回避を一部可能であることを示した．

カーネルのセキュリティ機構をカーネルモード上での攻撃から保護あるい攻撃を困難化することはカーネルにおけるセキュリティ確保に繋がる．提案手法では，カーネルの仮想記憶空間を中継用，セキュリティ用，ならびにカーネルに分離し，攻撃はカーネル脆弱性を含むカーネルコードの置かれるカーネルの仮想記憶空間においてのみ動作可能とした．攻撃対象はカーネルの仮想記憶空間の仮想アドレスの範囲に限定される．攻撃時に，中継用，ならびにセキュリティ用の仮想記憶空間の仮想アドレスを指定することは困難であり，それらの仮想記憶空間にカーネルコード・データを配置することで攻撃への耐性を備えられる．

提案手法においては，セキュリティ機能回避の攻撃が成功した場合でも，複数ある仮想記憶空間の切替え処理の一部に影響を受けるのみである．攻撃後に発行されるシステムコールの処理前にはセキュリティ機能の処理を実行可能なことから，カーネルモードにおける仮想記憶空間の切替え処理の保護は実現されたと言える．また，カーネル監視のセキュリティ機構を備えることで仮想記憶空間の切替え処理を改ざん検出可能である．

複数の仮想記憶空間を使い分け，カーネルコードの適切なマッピングを制御することでカーネル脆弱性を利用した攻撃の影響範囲を細かく制限し，より攻撃の困難化を実現できる可能性がある．そのため，カーネル脆弱性の被害を低減するためにはアプリケーションやカーネルの処理毎により適切なカーネルの仮想記憶空間の構築が必要になると考えている．

5.3 複数の仮想記憶空間管理に対する考察

提案手法では，プロセス毎にユーザ，中継用，ならびにカーネルの仮想記憶空間を生成し，セキュリティ用の仮想記憶空間と合わせて 4 種類の仮想記憶空間をカーネルモードにおけるカーネル処理に応じて切替える．カーネルにおけるプロセス間のコンテキストスイッチはプロセス毎に仮想記憶空間を 1 度切替えるのみだが，KPTI により，プロセスのコンテキストスイッチ後，ユーザモードとカーネルモードの遷移毎に仮想記憶空間は切替えられている．提案手法においては，カーネルモードにて中継用，セキュリティ用，カーネルの仮想記憶空間の切替え処理を追加したことからオーバヘッドはさらに増加した．また，各仮想記憶空間を構成するページテーブルの管理は複雑になり，物理記憶空間の新たな領域確保を必要とした．

提案手法の適用先である KPTI を備えた Linux においては，SCI を部分的に導入することで，プロセス単位で管理されているユーザとカーネルの仮想記憶空間に新たに中継用を設け，セキュリティ用のカーネルの仮想記憶空間は個別に管理するセキュリティ機構を実現した．Page Table Isolation を導入した FreeBSD に対しては Linux と同様のアプローチにて提案手法を適用可能と考えている [13]．さらには，Windows や他の OS への適用可能性の検討，ならびに複数の仮想記憶空間管理を x86_64 以外の CPU アーキテクチャでも実現可能か調査を進めたい．

6 関連研究

カーネル脆弱性は，オーバフローや記憶空間の管理漏れなどの実装の不具合を 13 種類，攻撃による影響として仮想記憶空間の改ざん，ポリシー違反，Denial of Service，および情報漏洩の 4 種類に分類される [14]．Linux においては攻撃対策として，スタック領域監視 [1]，カーネルでの制御フロー検証 [3]，カーネルの仮想記憶空間配置をランダム化する KASLR[5]，そして，ユーザとカーネルの仮想記憶空間を分離のための KPTI [6] は導入された．また，強制アクセス制御を行う SELinux[12] や権限細分化のためのケイパビリティ [15] により，プロセスへの特権最小化を行うことで特権奪取後の被害軽減を実現可能としている．さらに，カーネル脆弱性の要因となりうる仮想記憶空間利用の検査機構として kmemcheck [16] や KASAN [17] があり，syzbot と syzkaller による自動ファジングに利用されている [18]．

カーネル脆弱性を利用した攻撃の与える影響を利用したセキュリティ機構として，Kernel multi-variant execution (kMVX) は，カーネル処理に個別の仮想記憶空間割当てと異なるスタックの振舞い用意し，同期的に実行させた際の攻撃による実行環境への変化有無を基に攻撃検出を行う [19]．kMVX は検出対象とするカーネル脆弱性による攻撃を的確に補足できるとされており，提案手法を適用した環境での攻撃対応能力やオーバヘッドの比較を考えている．

攻撃対象となるカーネルコードの領域を減らし攻撃困難化を図る手法として，KRAZOR はセキュリティコンテキスト毎に利用可能なカーネルコードをリスト化してカーネル機能を制限する [20]，KASR はカーネルコードの実行属性をページ単位で制御し，アプリケーションの実行に必要なページのみ有効とする [21]．Multik は事前にアプリケーション毎に最低限必要となるカーネルコードのみマッピングした仮想記憶空間を構築する [22]．カーネルコードの低減は攻撃緩和に有効であり，提案手法におけるカーネルの仮想記憶空間の生成と組合わせることが可能であると考えている．

カーネルの完全性を検証し実行中においても外部か

ら保護するための手法として，Trusted Computing Base（TCB）では起動時のカーネルイメージの検証を行う [23]．SecVisor や TrustVisor では仮想マシンモニタからゲスト OS として動作中のカーネルコードやデータを検証する [24, 25]．Sprobes では，CPU による保護領域である Trusted Execution Environment を利用した検証機構 [26]，ならびに GRIM では GPU からカーネルを保護する機構を提案している [27]．

カーネルモードにて動作中のカーネルや仮想マシンモニタを監視するセキュリティ機構として，SIM は仮想マシンモニタを介したゲスト OS の仮想記憶空間の操作によりカーネル監視を実現している [28]．ED-Monitor はカーネルモジュールによりレジスタ操作の補足等による仮想マシンモニタの監視を実現している [29]．OS より低位レイヤやハードウェアによる支援によるカーネルの検証と監視は攻撃対策として有効である．提案手法はカーネルと一体化して動作するためベアメタルや既存のクラウド上でも適用可能なことから組合せることで多層的なカーネル保護を実現できる．

カーネルコードやデータの保護としては，W^X に加えて R^X の排他制御を行うことでより細かなカーネル構成を行う手法が提案されている [30]．カーネルの仮想記憶空間の保護としては，仮想記憶空間を構成するページテーブル位置のランダム化 [31]，Intel CPU の Extended Page Table 機能を利用したカーネル仮想記憶空間を複数に分離する手法も提案されている [32]．提案手法では仮想記憶空間の切替え処理の保護を目的とした，今後，既存手法におけるカーネルの保護対象と比較し，カーネルの保護能力の向上を進める予定である．

7 おわりに

カーネル脆弱性を利用した攻撃への対策として OS では，スタック監視や制御フロー検証，KASLR，および KPTI の導入が進んでいる．しかし，カーネルモードでのカーネル脆弱性を利用した攻撃による特権奪取やセキュリティ機構無効化の可能性は依然存在する．

本稿では，カーネルにて複数の仮想記憶空間を用いることでカーネルの提供するセキュリティ機能，ならびに仮想記憶空間の切替え処理の保護手法を提案し，セキュリティ機能の回避を目的とした攻撃の困難化を進めた．提案手法を KPTI，SCI，および我々の開発するカーネルの仮想記憶空間の監視を行うセキュリティ機構を備えた Linux にて実現し，実際のカーネル脆弱性を利用した既存のセキュリティ機構の改ざん攻撃検出，ならびに仮想記憶空間の切替え処理への攻撃の検出とカーネルへの影響を評価した．

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] Kemerlis, P, V., et al.: kGuard - Lightweight Kernel Protection against Return-to-User Attacks, the 21st USENIX Conference on Security symposium, (2012).

[2] Yamauchi, T., 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 (DSC), (2018).

[3] Abadi, M., et al.: Control-Flow Integrity Principles, Implementations, the 12th ACM Conference on Computer and Communications Security (CCS), (2005).

[4] Rapoport, M.: x86: introduce system calls addess space isolation, https://lwn.net/ml/linux-kernel/1556228754-12996-1-git-send-email-rppt@linux.ibm.com/

[5] Hund, R., et al.: Practical Timing Side Channel Attacks against Kernel Space ASLR, 2013 IEEE Symposium on Security and Privacy, (2013).

[6] Lipp, M., et al.: KASLR is Dead - Long Live KASLR, 2017 International Symposium on Engineering Secure Software and Systems (ESSoS), vol. 10379, no. 3, (2017).

[7] Exploit Database, Nexus 5 Android 5.0 - Privilege Escalation, https://www.exploit-db.com/exploits/35711/

[8] grsecurity: super fun 2.6.30+/RHEL5 2.6.18 local kernel exploit, https://grsecurity.net/~spender/exploits/exploit2.txt

[9] 葛野弘樹, 山内利宏.: 独自のカーネル用仮想記憶空間を用いたカーネルモジュール監視手法, コンピュータセキュリティシンポジウム 2018, pp.971–978 (2018).

[10] 葛野弘樹, 山内利宏.: カーネルに対する攻撃における独自の仮想記憶空間の切替え手法の検出能力と防御手法, 情報処理学会第 178 回 DPS・第 84 回 CSEC 合同研究発表会, March, (2019).

[11] CVE-2017-16995, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-16995, (accessed 2019-06-10).

[12] Security-enhanced Linux, available from http://www.nsa.gov/research/selinux/, (accessed 2019-05-22).

[13] Tetlow, G.: Response to Meltdown and Spectre, https://lists.freebsd.org/pipermail/freebsd-security/2018-January/009719.html, (accessed 2019-05-21).

[14] Chen, H., et al.: Linux kernel vulnerabilities - state-of-the-art defenses and open problems, the 2nd Asia-Pacific Workshop on Systems (APSys), (2011).

[15] Linden, A. T.: Operating System Structures to Support Security and Reliable Software, ACM Computing Surveys (CSUR), vol. 8, no. 4, pp. 409–445, (1976).

[16] Getting started with kmemcheck, https://www.kernel.org/doc/dev-tools/kmemcheck.html, (accessed 2019-05-21).

[17] The Kernel Address Sanitizer (KASAN), https://www.kernel.org/doc/dev-tools/kasan.html, (accessed 2019-05-21).

[18] syzkaller is an unsupervised, coverage-guided kernel fuzzer, https://github.com/google/syzkaller/, (accessed 2019-05-22).

[19] Österlund, S., et al.: kMVX: Detecting Kernel Information Leaks with Multi-variant Execution, the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), (2019).

[20] K. Anil., et al.: Quantifiable Run-Time Kernel Attack Surface Reduction, 11th International Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA), (2014).

[21] Zhang, Z., et al.: KASR: A Reliable and Practical Approach to Attack Surface Reduction of Commodity OS Kernels, The 21st International Symposium on Research in Attacks, Intrusions and Defenses (RAID), (2018).

[22] Kuo, H, C., et al.: MultiK: A Framework for Orchestrating Multiple Specialized Kernels, arXiv.org, 16-Mar-2019.

[23] Trusted computing group. tpm main specification. "http://www.trustedcomputinggroup.org/resources/tpm_main_specification, 2003, (accessed 2018-08-10).

[24] Seshadri, A., et al.: SecVisor - a tiny hypervisor to provide lifetime kernel code integrity for commodity OSes, the 21st ACM SIGOPS symposium on Operating systems principles (SOSP), (2007).

[25] McCune, M, J., et al.: TrustVisor - Efficient TCB Reduction and Attestation, 2010 IEEE Symposium on Security and Privacy, (2010).

[26] X. Ge, H. Vijayakumar, and T. Jaeger, “Sprobes: Enforcing Kernel Code Integrity on the TrustZone Architecture,” arXiv.org, vol. cs.CR. 29-Oct-2014.

[27] Koromilas, L., et al.: GRIM - Leveraging GPUs for Kernel Integrity Monitoring, the 19th International Symposium on Research in Attacks, Intrusions and Defenses, (2016).

[28] Sharif, I, M., et al.: Secure in-VM monitoring using hardware virtualization, the 16th ACM Conference on Computer and Communications Security (CCS), (2009).

[29] Deng, L., et al.: Dancing with Wolves: Towards Practical Event-driven VMM Monitoring, the 13th ACM SIGPLAN/SIGOPS International Conference, (2017).

[30] Pomonis, M., et al.: kR^X: Comprehensive Kernel Protection against Just-In-Time Code Reuse, the Twelfth European Conference on Computer Systems (EuroSys), (2017).

[31] Davi, L., et al.: PT-Rand: Practical Mitigation of Data-only Attacks against Page Tables, the 23th Network and Distributed System Security Symposium (NDSS), (2016).

[32] Hua. Z., et al.: EPTI - Efficient Defence against Meltdown Attack for Unpatched VMs, USENIX Annual Technical Conference (ATC), (2018).