CL-005

権限情報の動的な再配置による特権昇格攻撃防止手法の提案と評価

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

概要

オペレーティングシステムへの攻撃として，カーネル脆弱性を利用したメモリ破壊による特権昇格攻撃が指摘されている．攻撃対策として，kernel address layout randomization（KASLR）では，カーネル起動時，カーネルの仮想記憶区間上のカーネルコードやデータ配置をランダム化し，攻撃困難化を行う．しかし，動作中のカーネルではカーネルコードやデータ配置は固定化されるため，権限情報に関するカーネルデータの仮想アドレスが特定された場合，特権昇格攻撃は成功する．本稿では，動作中カーネルの仮想記憶空間上のカーネルデータを動的に再配置し，メモリ破壊攻撃の困難化を図るセキュリティ機構を提案する．提案するセキュリティ機構では，複数の再配置専用ページをカーネルに備え，ユーザプロセスからのシステムコール呼出し時に複数の再配置専用ページから一つを選択し，保護対象のカーネルデータを複製する．これにより，保護対象のカーネルデータの格納先をカーネルの仮想記憶空間上で動的に再配置し，仮想アドレスを変化可能とする．動作中のカーネルに適用可能なことから，KASLR と組合わせ，攻撃成功にかかる攻撃コストを増加させ，カーネルの攻撃耐性を向上可能である．提案するセキュリティ機構の実現方式として，ユーザプロセスの権限情報に関するカーネルデータを保護対象とし，カーネルの仮想記憶空間上で動的に再配置可能とした．これにより，メモリ破壊を利用した特権昇格攻撃による権限情報の改ざんを困難化した．提案するセキュリティ機構の評価として，実現方式を備えた Linux にて，ユーザプロセスによる特権昇格攻撃を防止可能なことを確認した．性能評価として，システムコール発行時のオーバヘッドは最大 149.67%，カーネルの性能スコアへの影響は 2.50%となり，動作中のカーネルに与える影響は軽微であることを示した．また，カーネルデータの再配置による攻撃試行回数の評価から，攻撃コストを増加させることを示した．

1　はじめに

オペレーティングシステム（OS）カーネルにおけるメモリ破壊対策が課題とされている．メモリ破壊を利用した攻撃例として，権限情報に関するカーネルデータの改ざんによる特権昇格攻撃やアクセス制御に関するカーネルデータの改ざんによるセキュリティ機能の無効化攻撃が指摘されている [1, 2].

メモリ破壊攻撃への対策として，KASLR (kernel address space layout randomization) では，カーネル起動時にカーネルの仮想記憶空間上へカーネルコードやデータをランダムに配置する．KASLR により，攻撃対象となるカーネルデータの仮想アドレスの特定は困難化され，メモリ破壊によるカーネルデータの改ざん可能性を低減可能となる [3]．しかし，動作中カーネルにて，カーネルの仮想記憶空間上に配置されるカーネルデータの仮想アドレスは固定化されるため，次の課題がある．

課題：カーネルデータの改ざん

動作中のカーネルにおいて，攻撃を行うユーザプロセスが攻撃対象のカーネルデータの仮想アドレスを特定した場合 [4]，カーネル脆弱性を利用した攻撃により，カーネルデータは改ざん可能である．そのため，攻撃者は特権昇格攻撃やセキュリティ機能の無効化攻撃を行うことができる．

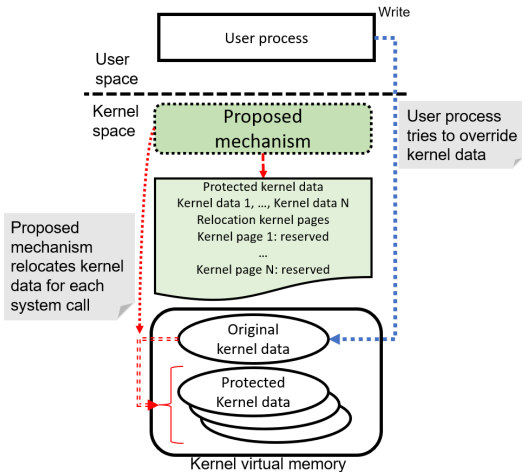

図 1　提案するセキュリティ機構の概要

本稿では，カーネルデータの改ざんに対する攻撃耐性をカーネルに備えるため，カーネルの仮想記憶空間上のカーネルデータを動的に再配置可能とするセキュリティ機構を提案する．提案するセキュリティ機構は，動作中のカーネルに適用でき，KASLR と組合わせ，カーネルへの攻撃耐性を向上可能である．図 1 に提案するセキュリティ機構の概要を示す．提案するセキュリティ機構では，ユーザプロセスに対し，システムコール発行時に特定のカーネルデータを再配置させる．カーネルデータの再配置により，仮想アドレスの値を変化させることで攻撃対象のカーネルデータの仮想アドレスの特定を困難化し，メモリ破壊攻撃に必要な攻撃コストを増加させる．

提案するセキュリティ機構において，権限情報に関するカーネルデータを保護対象とした場合，特権昇格攻撃を試みるユーザプロセスに対し，権限情報の仮想アドレスはシステムコール発行毎に変化する．権限情報の改ざんには予め権限情報の仮想アドレスの特定が必要であり，特権昇格攻撃は困難となる．

提案するセキュリティ機構の実現方式では，カーネルの仮想記憶空間において，ユーザプロセス毎に再配置専用ページ（4KB）を複数確保し，保護対象のカーネルデータの格納に利用する．システムコール発行時，ランダムに選択した再配置専用ページに保護対象のカーネルデータを複製し，保護対象のカーネルデータへの参照を複製先に変更する．保護対象のカーネルデータの格納先のページをシステムコール発効毎に変更することで，動作中のカーネルにおける保護対象のカーネルデータの仮

1）　神戸大学 大学院工学研究科
2）　岡山大学 学術研究院自然科学学域

表 1　カーネル脆弱性の種別 [5] (✓：提案手法による緩和)

Type	Description	Mitigation
Missing pointer check	ポインタ内容の確認漏れ	✓
Missing permission check	権限管理の確認漏れ	
Buffer overflow	スタックまたはヒープ領域の上書き	✓
Uninitialized data	変数の初期化漏れ	
Null deference	ヌル領域の参照	
Divide by zero	Zero 値での割り算処理	
Infinite loop	無限ループの発生	
Data race/Deadlock	競合条件の発生	
Memory mismanagement	メモリ管理の整合性	✓
Miscellaneous	その他	

表 2　カーネル脆弱性を利用した攻撃による影響 [5] (✓：提案手法が有効)

Effect	Description	Target
Memory corruption	カーネルコードやカーネルデータの改ざん	✓
Policy violation	カーネルにおける権限判定処理部分の実装不備	
Denial of Service	カーネルの強制停止	
OS information leakage	カーネルデータの初期化漏れによるデータ漏洩	

想アドレスを動的に変化させる．

提案するセキュリティ機構において，カーネルデータの仮想アドレスの特定困難化は，再配置対象のカーネルデータサイズと再配置専用ページ数に依存する．再配置対象を 256 bytes（8 bits），再配置専用ページ（4KB, 12bits）を 1 ページとした場合，4bits の範囲にて総当たり攻撃からカーネルデータを保護可能となる（詳細は 6.6 節を参照）．

提案するセキュリティ機構の攻撃防止例として，特権昇格攻撃に対し，動作中のカーネルにおいて，権限情報を格納するカーネルデータを保護対象に指定する．特権昇格攻撃を行うユーザプロセスが任意のメモリ破壊攻撃に利用可能な脆弱なカーネルコードを実行した場合，権限情報の改ざんは試みられる．しかし，提案するセキュリティ機構により，権限情報の正確な仮想アドレスの位置特定は困難化される．そのため，権限情報の改ざんは失敗し，特権昇格攻撃は防止される．

本稿での研究貢献は以下の通りである：

1. メモリ破壊攻撃防止のため，システムコール発行時にカーネルデータの動的な再配置を可能とするセキュリティ機構の設計と実装を行った．提案手法を Linux に実現し，ユーザプロセスの権限情報を保護対象とし，特権昇格攻撃への攻撃耐性を備えた．
2. 提案するセキュリティ機構の評価として，特権昇格攻撃を試みるユーザプロセスに対し，特権昇格を未然防止可能なことを確認した．また，ユーザプロセスおよびカーネル動作への影響として，性能評価を行い，システムコール発行時のカーネルへのオーバヘッドは 102.88%から 149.67%，カーネルの性能スコアへの影響は 2.50%であることを示した．カーネルデータの再配置による攻撃困難性の評価からメモリ破壊への攻撃耐性を向上可能なことを示した．

2　カーネル脆弱性を利用したメモリ破壊攻撃

カーネル脆弱性はカーネルの攻撃に利用し，動作に影響を与える実装不備であり，表 1 に不備内容からカーネル脆弱性の 10 種類の分類を示す．また，カーネル脆弱性を利用した攻撃による 4 種別の影響を表 2 に示す [5]．

提案するセキュリティ機構は，ポインタやスタック，ならびにメモリ操作に関するカーネル脆弱性を利用したメモリ破壊攻撃の対策手法である．メモリ破壊攻撃は，

```
1   // From Linux kernel v5.18.2
2   // include/linux/sched.h
3   struct task_struct {
4     ...
5     const struct cred __rcu   *cred;
6     ...
7   }
8   // include/linux/cred.h
9   struct cred {
10    ...
11    /* real UID of the task */
12    kuid_t      uid;
13    /* real GID of the task */
14    kgid_t      gid;
15    ...
16  }
17  // include/linux/uidgid.h
18  typedef struct {
19    // typedef __kernel_uid32_t uid_t;
20    // typedef unsigned int __kernel_uid32_t;
21    uid_t val;
22  } kuid_t;
23  typedef struct {
24    // typedef __kernel_gid32_t gid_t;
25    // typedef unsigned int __kernel_gid32_t;
26    gid_t val;
27  } kgid_t;
```

図 2　Linux におけるユーザ ID に関する構造体 [6]

カーネル脆弱性を利用してカーネルの仮想記憶空間上の任意の仮想アドレスを指定し，不正な読書きを試みる攻撃である．攻撃対象のメモリ領域が書換可能な場合，メモリ破壊により攻撃対象は上書きされる．

2.1　特権昇格攻撃

特権昇格攻撃では，カーネル脆弱性のうち，不正なポインタ操作や権限管理の確認漏れを利用し，権限操作の変更処理を行うカーネルコードを強制的に呼出す脆弱性 [7, 8, 9]，ならびにメモリ破壊攻撃を利用した脆弱性が報告されている [10]．

ユーザプロセスの権限情報に関するカーネルデータはカーネルの仮想記憶空間に配置され，カーネルにより管理される．特権昇格攻撃において，メモリ破壊攻撃を利用する場合，攻撃者はカーネル仮想記憶空間上の権限情報を格納するカーネルデータの改ざんを試みる．メモリ破壊攻撃を行える場合，攻撃者はユーザプロセスのユーザ ID を管理者ユーザへ変更可能である [10]．

図 2 に Linux におけるユーザ ID の構造体定義を示す． 5 行目にて，Linux では，ユーザプロセスを管理する `task_struct` 構造体にて，権限情報を `cred` 構造体に保持することを示している．ユーザ ID は，9 行目から 16 行目の `cred` 構造体に含まれる 12 行目の `kuid_t` 構造体の変数 `uid` に格納され， 18 行目から 22 行目の構造体 `kuid_t` の値 `val` である．特権昇格攻撃では，`uid` 構造体の値 `val` を `root` のユーザ ID（0）に書き換える．

攻撃者はメモリ破壊攻撃の起点となる脆弱なカーネルコードを介してカーネルデータは仮想アドレスを指定することで読書きできる．特権昇格攻撃を成功させるための前提として，権限情報を格納するカーネルデータの仮想アドレスを攻撃対象として正確に指定する必要がある．そのため，攻撃前に権限情報に関するカーネルデータの仮想アドレスをカーネルの仮想記憶空間上の位置として特定しなければならない．

3　脅威モデル

3.1　攻撃対象環境

本稿における脅威モデルとして，攻撃者はカーネル脆弱性を介してメモリ破壊攻撃を試みる．攻撃者によるメモリ破壊攻撃時の改ざん対象は権限情報に関するカーネルデータとし，特権昇格攻撃を試みる．脅威モデルとして想定する攻撃対象環境を以下にまとめる．

- 攻撃者： 一般ユーザ権限にてユーザプロセスを実

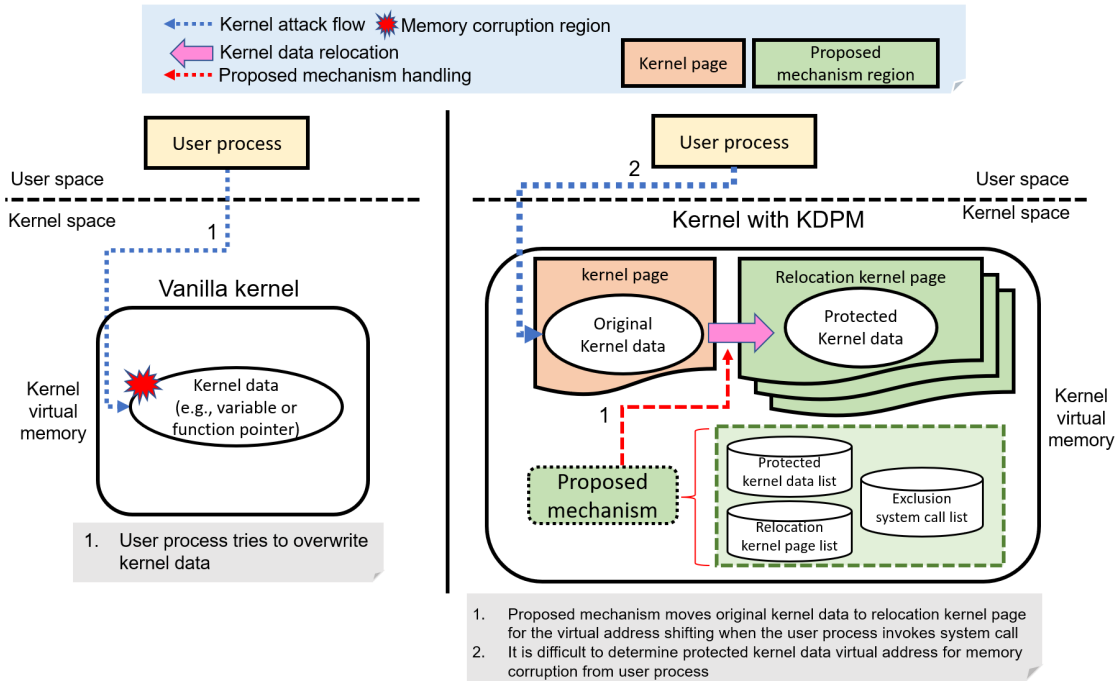

図 3　提案するセキュリティ機構の設計概要

行する．ユーザプロセスにて，攻撃コードを実行し，脆弱なカーネルコードを呼出し，メモリ破壊攻撃を試みる．

- カーネル：メモリ破壊攻撃に利用可能なカーネル脆弱性を含み，ユーザプロセスから脆弱なカーネルコードを呼出し可能とする．ユーザプロセスに対し，アクセス制御機能以外のセキュリティ機構は適用しない．
- カーネル脆弱性：任意の仮想アドレスを指定し，メモリ破壊攻撃に利用可能な脆弱なカーネルコードとする．ユーザプロセスより，攻撃対象の仮想アドレスおよび上書きデータを受取り，攻撃対象のカーネルデータの改ざんを行う．
- 攻撃対象：攻撃対象は，カーネルの仮想記憶空間上に配置されるカーネルデータとする．メモリ破壊による特権昇格攻撃では，ユーザプロセスの権限情報を格納するカーネルデータが攻撃対象となる．

3.2　攻撃シナリオ

想定する攻撃シナリオとして，攻撃者は，カーネルに対してメモリ破壊攻撃を試みる．攻撃者は，一般ユーザとして任意のユーザプロセスを実行する．ユーザプロセスはメモリ破壊攻撃可能な脆弱なカーネルコードを呼出す．脆弱なカーネルコードの呼出時，攻撃対象とするカーネルデータの仮想アドレスと上書きデータを脆弱なカーネルコードの引数に指定し受け渡し，攻撃対象のカーネルデータを任意のデータで上書きする．

攻撃例として，特権昇格攻撃では，攻撃者は，事前にユーザプロセスの権限情報が配置されるカーネルデータの仮想アドレスを特定する．続いて，メモリ破壊可能な脆弱なカーネルコードを利用し，特定したユーザプロセスの権限情報の仮想アドレスに対し，管理者権限のユーザ ID を上書きする．これにより，攻撃者のユーザプロセスは管理者権限を取得し，特権昇格攻撃が行われる．

4　提案手法

4.1　提案手法の要件

提案するセキュリティ機構は，カーネルの動作中において，保護対象としたカーネルデータをカーネルの仮想記憶空間上で動的に再配置する．設計として，次の要件を満たすことを目指した．

要件 1：想定する攻撃は，カーネル脆弱性を利用したメモリ破壊攻撃によるカーネルデータの改ざんとし，カーネルデータの改ざんはユーザプロセスから呼出されたシステムコールの実行中に行われることとする．

要件 2：ユーザプロセスに対し，保護対象のカーネルデータのカーネルの仮想記憶空間上での再配置制御は透過的に行う．

要件 3：カーネルデータの再配置先の特定を困難とするため，仮想アドレスの変化には規則性を持たせないようにする．

4.2　提案手法の設計

4.2.1　設計方針

設計において，提案するセキュリティ機構の設計方針を次のように定めた．

- 設計方針 1：ユーザプロセスからの攻撃緩和，ならびに攻撃対策検出の困難化を備えた設計とするため，カーネル内でカーネルデータの再配置処理を行う．
- 設計方針 2：提案するセキュリティ機構の備えるカーネルデータの再配置処理での攻撃防止効果により，ユーザプロセス，ならびにカーネルの動作に影響を与えないよう設計を行う．

4.2.2　設計概要

提案するセキュリティ機構の設計概要を図 3 に示す．設計方針に基づき，要件を満たすため，カーネル内にて保護対象とするカーネルデータ（例，ユーザプロセスの権限情報）のカーネルの仮想記憶空間上での再配置先として，複数の再配置専用ページを設ける．また，保護対象のカーネルデータの一覧，再配置先専用ページの一

覧，ならびに提案するセキュリティ機構にて再配置処理を除外するシステムコールの一覧をリストとして管理する．提案するセキュリティ機構は全てのユーザプロセスに適用し，システムコールの発行前に保護対象のカーネルデータの再配置制御を行う．

4.2.3 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，カーネルの仮想記憶空間上に再配置を行う保護対象カーネルデータは次の通りとする．

- 保護対象カーネルデータ：ユーザプロセス生成時に作成されるカーネルデータ（例．権限情報）

保護対象カーネルデータはユーザプロセス毎に指定し，再配置制御の対象とする．

4.2.4 再配置専用ページ

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の再配置先の再配置専用ページは次の通りとする．

- 再配置専用ページ：保護対象カーネルデータの再配置先とするカーネルページ

提案するセキュリティ機構では，複数の再配置専用ページをカーネルの仮想記憶空間上に設ける．

4.2.5 保護対象カーネルデータの再配置制御

提案するセキュリティ機構における保護対象カーネルデータの再配置制御はシステムコール実行前後に行う．

- システムコール実行前：保護対象カーネルデータを再配置専用ページに複製
- システムコール実行後：再配置専用ページの保護対象カーネルデータを元のカーネルデータの格納先に複製

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の再配置先の選定はシステムコール実行前に再配置専用ページのリストからランダムに選択することで行う．これにより，保護対象カーネルデータの再配置後の仮想アドレスは一定範囲で変化し，仮想アドレス特定の困難化を実現する．

4.2.6 保護対象カーネルデータの再配置適用除外

保護対象カーネルデータの種類により，カーネルの仮想記憶空間上で再配置を行うことでカーネルデータへの参照失敗や書込み失敗等により，カーネルならびにユーザプロセスの動作に影響を与える．

提案するセキュリティ機構では，カーネルおよびユーザプロセスの正常動作への影響を避けるため，保護対象カーネルデータ毎に再配置適用除外システムコールを予め指定可能とし，再配置制御の適用可否に用いる．指定された適用除外システムコールの実行時においては，保護対象カーネルデータの再配置処理を行わない．

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 とした．実現方式では，メモリ破壊を利用した特権昇格攻撃の防止を目的とし，保護対象のカーネルデータとして権限情報を指定し，カーネルの仮想記憶空間上で再配置可能とした．

5.1 実現方式の概要

実現方式の概要を図 4 に示す．実現方式では，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，新たなページを作成し，配置する．また，カーネルの起動時に一定数の再配置専用ページを作成し，再配置専用ページの一覧としてリスト管理する．実現方式における適用除外システムコールの一覧はカーネル起動時にリストとして管理する．また，実現方式の処理にて，システムコール実行前，ランダムに選択した再配置専用ページに対し，権限情報を格納したページを複製し，権限情報の仮想アドレスを変化させる．

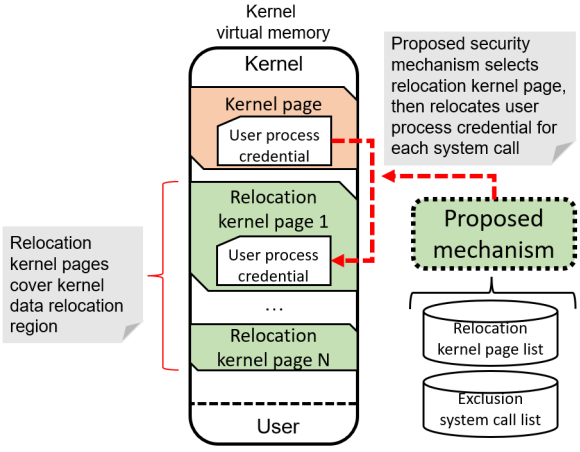

図 4 提案するセキュリティ機構の実現方式の概要図

表 3 実現方式における保護対象カーネルデータ（参考文献 [11] 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)

5.2 保護対象カーネルデータ

実現方式における，保護対象カーネルデータをユーザプロセスの権限情報とするため，カーネルページ（4KB）を作成し，権限情報のカーネルデータを格納する．保護対象の権限情報を表 3 に示す．ユーザプロセスの動作期間中，カーネルページ単位（4KB）にて，実現方式の再配置制御対象とする．

5.3 再配置専用ページ

実現方式では，ユーザプロセス生成時の負荷低減ノため，一定数の再配置専用ページをカーネル起動時に確保する．再配置専用ページ確保に，`alloc_pages` 関数を用いる．ユーザプロセス生成時，予め確保した再配置専用ページ（4KB）を複数個（例，10 ページ）割当てる．

複数の再配置専用ページをユーザプロセス毎に割当てることで，カーネルの仮想記憶空間上において特定の仮想アドレスの範囲を権限情報の再配置先とする．また，再配置専用ページを複数個のページからランダムに選択可能とすることで，再配置先の仮想アドレスの特定を困難化している．

5.4 保護対象カーネルデータの再配置制御

実現方式におけるに再配置制御フローを図 5 に示す．権限情報を格納するカーネルデータの再配置制御は，再配置専用ページのリスト，および適用除外システムコールのリストを用い，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. 適用除外システムコールのリストに含まれるシステムコール番号か判定
   (a) 適用除外システムコールの場合： 権限情報の再配置は行わない
   (b) 適用除外システムコール以外の場合：権限情報

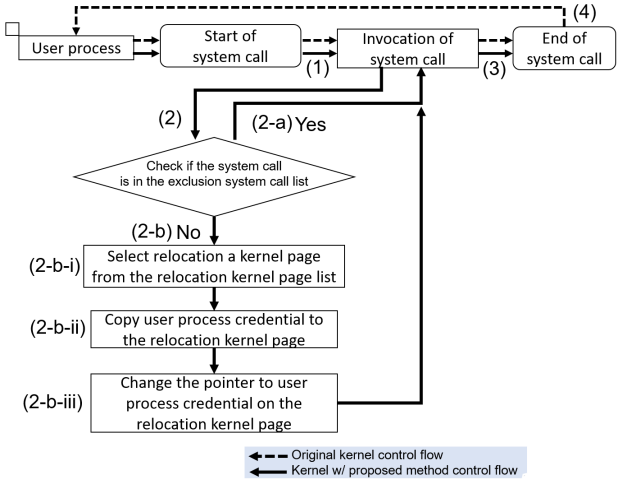

図 5　保護対象カーネルデータの再配置制御フロー

表 4　実現方式を適用除外とする権限操作を行うシステムコール（参考文献 [11] より抜粋）

Item	Description
Exclusion system call list	execve, setuid, setgid, setreuid, setregid setresuid, setresgid, setfsuid, setfsgid

の再配置を行う

i. 権限情報の再配置先とするカーネルページを再配置専用ページのリストからランダムに選択
ii. 権限情報を格納するカーネルデータをページ単位で再配置専用ページに複製
iii. カーネル内の権限情報への参照を複製先に変更

3. システムコールの実行を継続
4. システムコールを終了

権限情報への書込みはページフォルトを発生させる可能性があることから，権限情報への書込みを明示的に行うシステムコールは実現方式の適用除外リストとして管理する．表 4 に適用除外リストに含む権限情報の操作を行うシステムコールを示す．

5.4.1　ページフォルトハンドリング

特権昇格攻撃として権限情報を格納したカーネルデータへの不正な上書きの試みは，ページフォルトハンドラの `handle_page_fault` 関数にて補足する．Linux カーネルでは，ページフォルトの際に，参照先の仮想アドレスを把握できる．実現方式は，再配置前の権限情報の仮想アドレスとページフォルト発生時の仮想アドレスを比較し，不正な書込みと見なす場合，`force_sig_info` 関数にて対象ユーザプロセスに対して SIGKILL を送信する．

6　評価

6.1　評価項目と目的

提案するセキュリティ機構を備えたカーネルに対し，特権昇格攻撃に対するセキュリティ機能の調査，カーネル処理へのオーバヘッドの調査，ならびにカーネルデータの再配置による攻撃困難性の評価を目的として評価した．評価項目と内容を以下に示す．

1. 特権昇格攻撃に対するセキュリティ評価
   提案するセキュリティ機構を備えたカーネルにおいて，特権昇格攻撃のメモリ破壊に利用可能なカーネル脆弱性をカーネルに導入し， PoC コードによる特権昇格攻撃の試みを防止可能か評価した．
2. システムコール発行における性能評価
   提案するセキュリティ機構を備えたカーネルにおいて，システムコール発行前後でのカーネルデータの再配置処理に伴うオーバヘッドをベンチマークソフトウェアを利用して測定した．
3. カーネル動作における性能評価
   提案するセキュリティ機構を備えたカーネルにおいて，ベンチマークソフトウェアによるカーネル性能に関してベンチマークソフトウェアを利用してスコアを算定した．
4. カーネルデータ再配置による攻撃困難性評価
   提案するセキュリティ機構によるカーネルデータの再配置による仮想アドレスのランダム化粒度について， KASLR との比較を行い，攻撃試行回数に基づく攻撃困難性を評価した．

6.2　評価環境

セキュリティ評価，ならびに性能評価に評価用計算機を用いた．評価用計算機は Intel(R) Xeon(R) W-2295（3.00GHz，18 コア，メモリ 32GB），OS は Debian 11.3, Linux kernel 5.18.2 とした．提案するセキュリティ機構の実現方式の実装を Linux kernel 5.18.2 に行い，9 個のファイルに対して 248 行にて実現した．また，セキュリティ評価に用いるメモリ破壊に利用可能なカーネル脆弱性を 3 個のファイルに対して 32 行の追加， PoC コードは 134 行にて実現した．

6.2.1　特権昇格攻撃に利用可能なカーネル脆弱性

提案するセキュリティ機構のセキュリティ機能評価のため，次の通り，権限情報の仮想アドレスの特定に利用するシステムコール，ならびにメモリ破壊可能なカーネル脆弱性を利用するシステムコールを導入し特権昇格攻撃を可能とした．

- 独自システムコール 1：独自システムコール 1 では，ユーザプロセスの権限情報のカーネルデータの仮想アドレスを特定し，ユーザプロセスに返す．
- 独自システムコール 2：独自システムコール 2 では，第 1 引数に仮想アドレス，第 2 引数に上書きデータを指定する．独自システムコール 2 の実行では，指定した仮想アドレスに対して上書きデータを書込み，メモリ破壊を試みる．ユーザプロセスの権限情報のカーネルデータの仮想アドレスを指定した場合，特権昇格攻撃を可能とする．

6.3　特権昇格攻撃に対するセキュリティ評価

セキュリティ評価として，メモリ破壊を行うカーネル脆弱性を利用した特権昇格攻撃の実行履歴を出力する．攻撃を行うユーザプロセスは，独自システムコール 1 を使用し，権限情報の仮想アドレスを特定後，独自システムコール 2 を利用して特権昇格攻撃を試みる．

図 6 に，特権昇格攻撃を行うユーザプロセス実行時における提案するセキュリティ機構の攻撃防止結果を示す．攻撃を行うユーザプロセスにて，2 行目に，ユーザプロセスの権限情報を表示し，`uid` の値は 1,000 であり，一般ユーザと確認できる．4 行目にて，独自システムコール 1 を呼出し，権限情報を格納したカーネルデータの仮想アドレスを特定している．5 行目にて，特権昇格攻撃を行う独自システムコール 2 を実行している．

カーネルにおいては，8 行目において，権限情報を格

```
// PoC code running, process id is 1676
1.  user $ ./a.out
2.  uid=1000(user) gid=1000(user) groups=1000(user)
3.  [*] sys_kvuln01 system call invocation
4.  uid virtual address: ffff888007af9784
5.  [*] sys_kvuln02 system call invocation
6.  Killed user process

// Kernel log information
7.  // set kernel page of privilege at the user process creation
8.  [  363.704204] uid virtual address: ffff888007af9784
9.  // start system call invocation
10. [  363.702116] sys_kvuln02 system call invocation
11. [  363.702179] sysnum: 0x6a  (352)
12. [  363.702204] PID: user process 1676
13. // relocation kernel pages' region
14. // ffff888005d82000, …, ffff8880063e5000
15. // relocate kernel page of privilege
16. [  363.704204] uid virtual address: ffff8880063e2000
17. // Kernel memory corruption
18. [  363.704204] attack target virtual address: ffff888007af9784
19. [  364.216821] #PF: error code (0x0002), virtual address: ffff888007af9784
20. Page fault error code 2 (0b010)
21. Page fault error code bits: from Linux v5.18.2 :
    arch/x86/include/asm/trap_pf.h
      a.  bit 0 == 0: no page found
      b.  bit 1 == 1: write access, X86_PF_WRITE
      c.  bit 0 == 0: kernel-mode access
22. // finish system call invocation
23. // finish user process
```

Red text is the points of kernel memory corruption information

図 6　提案手法による特権昇格攻撃の防止結果

納したカーネルデータの仮想アドレスを表示している．13 行目，14 行目にて，再配置専用ページの仮想アドレスの範囲を示している．15 行目，16 行目にて，提案するセキュリティ機構は独自システムコール 2 の実行前に権限情報を格納するカーネルデータを専用ページへ移動し，仮想アドレスを変更させている．18 行目にて，独自システムコール 2 にて指定された仮想アドレスへの上書きが試みられている．19 行目にて，エラー番号 2 のページフォルトを捕捉している．権限情報を格納するカーネルデータの変更前の仮想アドレスに対するページへの書込み違反を示している．

6.4　カーネル処理における性能評価

提案するセキュリティ機構の実現方式では，システムコールの呼出時に保護対象のカーネルデータの複製処理を行う．評価においては，提案するセキュリティ機構を適用前の Linux kernel，ならびに提案するセキュリティ機構の適用時の Linux kernel にて，ベンチマークソフトウェア LMbench を 10 回実行し，システムコール呼出しにかかる平均値からオーバヘッドを算出した．

表 5 に性能評価結果を示す． LMbench では，評価項目毎にシステムコールの呼出し回数は異なり，fork+/bin/sh は 54 回，fork+execve は 4 回，fork+exit は 2 回，open/close は 2 回，その他は 1 回である．表 5 から，提案するセキュリティ機構の適用時，最もオーバヘッドの発生したシステムコール処理は write であり，149.67% の影響を示した．また，最も少ないオーバヘッドは fork+/bin/sh であり，102.88%の影響を示した．

6.5　カーネル動作における性能評価

カーネル動作時の性能評価として，提案するセキュリティ機構の適用前の Linux kernel，ならびに提案するセキュリティ機構の適用後の Linux kernel にて UnixBench を 5 回実行し，平均値から性能スコアを算出した．

評価結果を表 6 に示す． UnixBench は数値計算，ファイルコピー，プロセス処理，ならびにシステムコールに関する各種別の動作中カーネルの性能スコアを測定し，

表 5　lmbench を用いた提案手法のオーバヘッド (μs)

System call	Vanilla kernel	Implementation	Overhead
fork+/bin/sh	434.2899	446.8079	12.5180 (102.88%)
fork+execve	101.2726	129.0260	27.7534 (127.40%)
fork+exit	89.9990	94.8672	4.8682 (105.41%)
open/close	1.1642	1.4920	0.3278 (128.16%)
read	0.1177	0.1599	0.0422 (135.85%)
write	0.0908	0.1359	0.0451 (149.67%)
fstat	0.1484	0.1953	0.0468 (131.60%)
stat	0.5265	0.6979	0.1714 (132.55%)

表 6　UnixBench を用いた提案手法の性能スコア

	Vanilla kernel	Implementation
Dhrystone 2	4450.50	4440.50 (0.22%)
Double-Precision Whetstone	1557.54	1552.92 (0.30%)
Execl Throughput	1193.23	1187.14 (0.52%)
File Copy 1024 bufsize	4122.08	3997.08 (3.03%)
File Copy 256 bufsize	2790.40	2698.60 (3.29%)
File Copy 4096 bufsize	7401.80	7192.62 (2.82%)
Pipe Throughput	2109.68	2041.04 (3.25%)
Pipe-based Context Switching	806.02	785.34 (2.57%)
Process Creation	1019.10	1017.92 (0.12%)
Shell Scripts (1 concurrent)	2485.20	2456.13 (1.17%)
Shell Scripts (8 concurrent)	2298.00	2294.36 (0.16%)
System Call Overhead	1771.08	1620.68 (8.49%)
System Benchmarks Index Score	2195.16	2140.24 (2.50%)

性能が高いほど大きなスコア値を示す．表 6 から，提案するセキュリティ機構の適用前後の Linux kernel を比較し，提案するセキュリティ機構により，最もスコアへの影響が少ないのは Process Creation の 0.12%であり，最もスコアへの影響が大きいのは System Call Overhead の 8.49%であることを示した．また，全体的な性能スコアへの影響は 2.50%であることを示した．

6.6　カーネルデータ再配置による攻撃困難性評価

提案するセキュリティ機構，ならびに Linux KASLR [12] の攻撃困難性の比較を表 7 に示す．仮想アドレスのランダム化粒度はエントロピーで表される [3]．Linux KASLR 32 bits では，2 MB（21 bits）単位でランダム化され，512 MB（29 bits）で 8 bits エントロピー，ならびに 1 GB（30 bits）では 9 bits のエントロピーである [13]．提案するセキュリティ機構では，保護対象のカーネルデータサイズと再配置専用ページにより攻撃困難性は変化する．表 7 では，再配置対象を 256 byte，8 bits，再配置専用ページ（4 KB，12 bits）を 1，64，ならびに 4096 ページの場合，4, 10，ならびに 16 bits エントロピーとした．

仮想アドレスのランダム化粒度に対する攻撃として，仮想アドレスの総当たり攻撃によるメモリ破壊成功に必要な攻撃試行回数は， n bits エントロピーに対し，攻撃試行中に仮想アドレスを変化させない場合，$\frac{1}{2^n}$ 回であり，攻撃試行毎に仮想アドレスをランダム化可能な場合，2^n 回とされる [3]．Linux KASLR は起動時のみに仮想アドレスのランダム化を行うため，攻撃試行回数は $\frac{1}{2^n}$ 回要する．一方，提案するセキュリティ機構はユーザプロセスのシステムコール呼出し毎にカーネルデータの仮想アドレスをランダム化可能な事から，攻撃試行回数は n bits エントロピーに対し， 2^n 回要する．

7　考察

7.1　評価に対する考察

メモリ破壊攻撃への攻撃耐性の評価にて，提案するセキュリティ機構を備えたカーネルは，特権昇格攻撃による権限情報へのメモリ破壊を防止可能なことを確認し

表 7　仮想アドレスのランダム化の比較

Type	Entropy	Range	Align Size
Linux KASLR 32 bits	8 bits	512 MB (29 bits)	2 MB (21 bits)
Linux KASLR 64 bits	9 bits	1 GB (30 bits)	2 MB (21 bits)
Proposed method	4 bits	4 KB (12 bits)	256 byte (8 bits)
Proposed method	10 bits	256 KB (18 bits)	256 byte (8 bits)
Proposed method	16 bits	16 MB (24 bits)	256 byte (8 bits)

た．提案するセキュリティ機構の実現方式では，権限情報に関するカーネルデータを保護対象に指定し，カーネルの仮想記憶空間上で権限情報の再配置を行う．保護対象のカーネルデータの仮想アドレス特定は困難化されており，メモリ破壊攻撃からの保護を実現している．

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は，数値計算やプロセス操作への影響は少なく，ファイルコピー等のシステムコールを必要とする処理に対し，高いオーバヘッドを示した．オーバヘッドの要因として，システムコール発行後，カーネル内部処理として，保護対象カーネルデータの複製に要する処理時間が大きいと考えている．また，性能評価を通じ，カーネル動作の安定性に影響を与えないことを確認した．

7.2　提案手法の考察

7.2.1　提案手法の設計ならびに実現方式

ユーザプロセスに対してメモリ破壊攻撃への対策を透過的に行うため，提案するセキュリティ機構の設計として，カーネル内にてシステムコール発行毎に保護対象のカーネルデータをカーネルの仮想記憶空間上に再配置処理可能としている．権限情報は，メモリ破壊による特権昇格攻撃の際，攻撃対象となることが明らかであるため，実現方式において，ユーザプロセスの生成時にカーネルデータに格納されるユーザプロセスの権限情報を保護対象として指定し，保護可能とした．また，提案するセキュリティ機構によるカーネル動作への影響を最小化するため，権限情報の変更を伴うシステムコールは提案するセキュリティ機構の適用対象外とした．

7.2.2　カーネルデータの再配置処理

提案するセキュリティ機構においては，保護対象のカーネルデータ毎にメモリ位置の再配置処理の適切なタイミングは異なると考えている．権限情報以外を保護対象とする場合，カーネルデータ毎に書込み箇所を調査し，関連するシステムコールの検討が必要となる．また，保護対象のカーネルデータがページサイズ（4KB）を超える場合，ならびにカーネル内にて参照箇所が多岐にわたる場合，性能への影響を考慮して，提案するセキュリティ機構を適用可能か検討しなければならない．

7.2.3　再配置による攻撃困難化

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に対する攻撃試行回数は n bits エントロピーに対し，2^n 回である．メモリ破壊攻撃に必要な攻撃コストを増加させ，攻撃困難化を実現している．しかし，提案するセキュリティ機構では，保護対象のカーネルデータサイズならびに格納先の再配置ページ数により，n bits エントロピーは増減する．保護対象のカーネルデータの種別に合わせ，仮想アドレスの特定困難性の検討，メモリ破壊攻撃の攻撃コスト算出が必要である．

7.3　限界

提案するセキュリティ機構は脆弱なカーネルコードの呼出しや不正なメモリ書込みの防止は行わない． CFI は，コード呼出し順を検証し，不正なコード呼出しを防止する．また，Memory Protection Key（MPK）は CPU にて，ページ単位で書込み制限を可能としている．そのため，CFI および MPK を提案するセキュリティ機構と組合せ，カーネルの攻撃耐性を向上可能と考えている．

7.4　移植可能性

提案するセキュリティ機構の実現方式では，カーネルデータの保護にページ単位での仮想記憶空間の管理，ならびに特権昇格攻撃の防止においては，ユーザプロセス毎の権限情報管理を前提としている．移植可能性として，FreeBSD はページテーブルによりカーネルの仮想記憶空間を構築，管理している [14]．また，ユーザプロセス毎に権限情報を割当てることから提案するセキュリティ機構を実現可能であると考えている．

8　関連研究

カーネルの仮想記憶空間の保護

カーネルのメモリ破壊攻撃を緩和するため， KASLR は，カーネルデータ，およびカーネルコードの仮想アドレスを変化させ，攻撃を困難化させた [3]．Adelie は，KASLR の拡張手法として 64 ビット化とデバイスドライバへの適用手法を提案している [15]．また，仮想マシンモニタからゲスト OS に KASLR を適用する手法も提案されている [16]．kR^X はカーネルコード，ならびにカーネルデータの読込みと書込み権限を排他的に管理する手法を提案している [17].

不正コード実行防止

カーネルにおける攻撃防止手法として，Exclusive page frame ownership では，カーネルとユーザプロセスに対して排他的にページ割当てを可能とした [18]．KCoFI はカーネルにおいて，非同期処理を例外的に扱い，コード呼出し順検査による不正なコードの実行を防ぐ Control Flow Integrity（CFI）を適用可能とした [19].

カーネルの攻撃面最小化

カーネルの攻撃可能領域を削除する攻撃面最小化の手法として， kRazor は，ユーザプロセス実行時のカーネルコード単位で利用可否判断を行う [20]．KASR は，ユーザプロセス実行時に必要なカーネルコードとカーネルデータのみをメモリ空間に配置する [21].

8.1　既存研究との比較

先行研究 [3, 19, 20, 21] との比較を表 8 に示す．KASLR は，カーネル起動毎に攻撃対象となるカーネルデータ，およびカーネルコード呼出しに利用する仮想アドレスを変化させ，攻撃を困難化させる [3]．一方，カーネルの起動中，カーネルコードやカーネルデータの仮想アドレス位置は変化しない．サイドチャネル攻撃によりカーネルコードやカーネルデータの仮想アドレスを入手された場合，ランダム化の仮想アドレス範囲が特定され，攻撃に利用可能な点が指摘されている [4]．提案するセキュリティ機構は，再配置専用ページ（4KB）を複数用いるのみでカーネルデータ再配置を行う．再配置のために確保する仮想アドレスの範囲は少なく，カーネルデータの再配置される仮想アドレス情報の入手は困難であるといえる．また，動作中のカーネルに適用でき，KASLR と組合わせ，カーネルへの攻撃耐性を向上可能である．

KCoFI は，独自アーキテクチャにてカーネルを動作させ，非同期処理の呼出し順を検証可能としている [19]．不正なコード呼出しの防止に対し，CFI は有効である．

表 8　カーネルデータ保護手法の比較

機能	KASLR [3]	KCoFI [19]	kRazor [20], KASR [21]	提案方式
保護対象	カーネルコード・データ	カーネルデータ	カーネルデータ	カーネルデータ
実現方式	メモリ配置のランダム化	カーネルコード呼出し検証	カーネルコード削減	動的なメモリ再配置
限界	カーネル起動時のみ適用	非同期処理への対応	カーネル更新の対応	再配置先数の制限

一方，全てのカーネルコードの呼出し順への CFI 適用はオーバヘッドが増加する．提案するセキュリティ機構は，カーネルデータの再配置を特徴としており，攻撃の抑制は行わない．CFI と組合わせ，CFI を回避された場合における攻撃防止を可能と考えている．

カーネルの攻撃面最小化手法である kRazor や　KASR は，アプリケーションの動作に必要なカーネルコードとカーネルデータのリストを予め必要とする．アプリケーションが多数のカーネルコードとカーネルデータを必要とする場合，カーネルの最小化は困難となる．提案するセキュリティ機構は，権限情報などの保護対象を特定のカーネルデータのみに限定可能である．攻撃面最小化を適用できないカーネルデータに対し，メモリ破壊攻撃の防止に利用可能であると考えている．

9　おわりに

本稿では，ユーザプロセスによるシステムコールの呼出毎にカーネルデータをカーネルの仮想記憶空間上で再配置可能とし，メモリ破壊攻撃耐性を備えたセキュリティ機構を提案した．提案するセキュリティ機構では，再配置専用ページを複数備え，カーネルデータを無作為に選んだ再配置専用ページの 1 つに複製することで，動的にカーネルデータの仮想アドレスの変更を可能とする．実現方式では，権限情報を格納するカーネルデータを保護対象とし，権限情報の格納先仮想アドレスを動的に変化させ，メモリ破壊攻撃を困難化し，特権昇格攻撃を防止した．KASLR はカーネル起動時にメモリ空間上のカーネルコード・カーネルデータのメモリ配置をランダム化する．提案するセキュリティ機構は，動作中カーネルのカーネルデータのメモリ配置を変更するため，KASLR と併用可能であり，カーネルデータの特定困難性を高め，カーネル攻撃耐性向上に寄与する．

評価結果より，ユーザプロセスによる特権昇格攻撃を防止可能性を示した．また，オーバヘッド評価にて，システムコール発行時のカーネル負荷は 102.88%から 149.67%であり，カーネルの性能スコアへの影響は 2.50%である．また，提案するセキュリティ機構におけるカーネルデータの再配置は，攻撃困難性の評価より，KASLR と比較し，攻撃試行回数を多く要することから，メモリ破壊への攻撃耐性を向上可能なことを示した．

謝辞

本研究の一部は，JSPS 科研費 JP22H03592 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/. (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. (accessed 2018-08-10).

[3] Shacham, H., et al.: On the effectiveness of address-space randomization. *In: Proceedings of the 11th ACM Conference on Computer and Communications Security*, pp. 298-307, (2004).

[4] Gruss, D., et al.: KASLR is Dead: Long Live KASLR. *In: Proceedings of 2017 International Symposium on Engineering Secure Software and Systems*, vol. 10379, no. 3, pp. 161-176, (2017).

[5] Chen, H., et al.: Linux kernel vulnerabilities - state-of-the-art defenses and open problems. *In: Proceedings of the Second Asia-Pacific Workshop on Systems*, pp. 1-5, (2011).

[6] The Linux Kernel Archives, available from https://www.kernel.org/. (accessed 2022-06-10).

[7] CVE-2016-4997, available from https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-4997. (accessed 2019-05-12).

[8] CVE-2016-9793, available from https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-9793. (accessed 2019-05-12).

[9] CVE-2017-1000112, available from https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-1000112. (accessed 2019-05-12).

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

[11] 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).

[12] Kernel address space layout randomization, available from https://lwn.net/Articles/569635/. (accessed 2022-05-12).

[13] Randomize the address of the kernel image (KASLR), available from https://www.kernelconfig.io/config_randomize_base. . (accessed 2022-06-12).

[14] The FreeBSD documentation project, FreeBSD architecture handbook, https://www.freebsd.org/doc/en_US.ISO8859-1/books/arch-handbook/. (accessed 2019-08-08).

[15] Nikolaev, R., et al.: Adelie: continuous address space layout re-randomization for Linux drivers. *In: Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 483–498 (2022).

[16] Holmes, B., et al.: KASLR in the age of MicroVMs. *In: Proceedings of the Seventeenth European Conference on Computer Systems*, pp. 149-165 (2022).

[17] Pomonis, M., et al.: kR^X: comprehensive kernel protection against just-in-time code reuse. *In: Proceedings of the twelfth European Conference on Computer Systems*, pp. 420-436 (2017).

[18] Kemerlis, P, V., et al.: ret2dir: rethinking kernel isolation. *In: Proceedings of the 23rd USENIX Conference on Security Symposium*, pp. 957-972, (2014).

[19] Criswell, J., et al.: KCoFI: Complete Control-Flow Integrity for Commodity Operating System Kernels. *In: Proceedings of IEEE Security and Privacy*, pp. 292–307 (2014).

[20] Kurmus, A., et al.: Quantifiable Run-Time Kernel Attack Surface Reduction. *In: Proceedings of the 11th International Conference on Detection of Intrusions and Malware & Vulnerability Assessment*, vol. 8550, pp. 212-234 (2017).

[21] Zhang, Z., et al.: KASR: a reliable and practical approach to attack surface reduction of commodity os kernels. *In: Proceedings of the 21st International Symposium on Research in Attacks, Intrusions and Defenses*, vol. 11050, pp. 691-710 (2018).