CL-001

カーネル仮想記憶空間における排他的ページ参照機構によるデータ保護能力と性能評価
Evaluation of Exclusive Page Reference Mechanism Capability for Kernel Data

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

1　はじめに

オペレーティングシステム（OS）の脅威のうち，カーネル脆弱性を利用した攻撃では，特定のカーネルコードの不正な利用やデータの改竄を行うことで計算機資源の不正使用や他プロセスへの攻撃を可能としている．

従来，ユーザモード用の仮想記憶空間とカーネルモード用の仮想記憶空間は高速化のために 1 つのページテーブルを用いて構築されていた．ページテーブルにおけるカーネルコードやデータの保護として，仮想アドレスのランダマイズ化 [1]，および CPU によるアクセス制御が適用されてきた [2]．しかし，動作中プロセスからカーネルの仮想記憶空間を参照するサイドチャネル攻撃が考案され，対策にプロセスとカーネルの仮想記憶空間を異なるページテーブルに分割する手法が提案された [3]．

ユーザモードとカーネルモードにてページテーブルは分割されたが，依然としてカーネルの仮想記憶空間は 1 つのページテーブルから構築され利用されるため，カーネル脆弱性を利用した攻撃により，特権奪取やセキュリティ機構を無効化される可能性は存在している [4, 5]．カーネルモードにおける攻撃対策として，先行研究では，カーネルの仮想記憶空間を複数のページテーブルに細分化する手法 [6]，およびカーネルのタスク処理をグループ化する手法などが提案されている [7]．

コンテナや仮想マシン機能により，単一のカーネル上にて細かな計算資源環境の提供が行われるが，複数のコンテナや仮想マシンなどの安全動作を維持するためには，1 つのページテーブルから構成されるカーネルの仮想記憶空間に配置されるカーネルコードやデータを保護対象とし，攻撃緩和を図ることは重要である．

我々は，図 1 に示すように，攻撃時のカーネルコード・データへの不正参照や改ざん防止を可能とすることを目的とし，実行中のプロセスに対して特定のカーネルコード・データを利用制限する機能を備えた排他的ページ参照機構を提案している [8]．排他的ページ参照機構では，カーネルコード・データの利用制限を各プロセスの利用するカーネルコードの実行時にカーネルの仮想記憶空間を構成するページテーブルの走査として，ページテーブルにて保護対象カーネルコードの格納されたページの登録有無の探索を行い，ページ単位で参照可否を制御することで実現している．

先行研究においては，実際の攻撃プロセスに対する排他的ページ参照機構のデータ保護能力の有効性，ならびに動作への影響や負荷に関する性能評価は明らかではなかったことから，本稿では，以下の検討および検証を行なった：

- 排他的ページ参照機構を備えたカーネルにて，動作中プロセスからの攻撃に対して保護対象とするカーネルデータの参照制御タイミングの検討

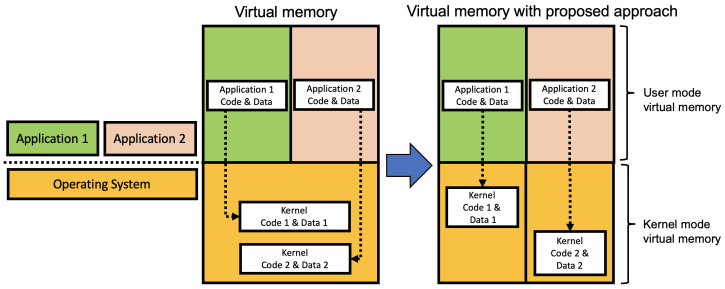

図 1　排他的ページ参照機構の適用例

- 排他的ページ参照機構にて保護対象とするカーネルデータについて，カーネル仮想記憶空間上の領域による制御対象プロセスやカーネル動作性能への影響の検証

我々の提案する排他的ページ参照機構を実現した Linux にて，保護対象としたカーネルデータへの保護機能の有効性評価，および性能評価を行った．

本稿での研究貢献は以下の通りである：

1. プロセス毎に利用可能なカーネルコード・データの制御可能な排他的ページ参照機構における制御タイミングの検討，ならびに保護対象とするカーネルデータを絞込みを行った．
2. 排他的ページ参照機構を組込んだ Linux にて，実際のカーネル脆弱性 CVE-2017-16995 [9] を介した攻撃に対し，特定のカーネルデータ保護の有効性評価した．また，性能評価を行い，システムコール処理に対しては，0.112 μs から 276.029 μs のオーバヘッド，ならびに Web クライアントプログラムに対しては，602.55 μs から 5,913.5 μs のオーバヘッドであることを示した．

2　背景知識

2.1　仮想記憶空間

仮想記憶は物理記憶の領域を超えた記憶空間の提供を可能とする機能である．Linux x86_64 における単一のページテーブルを用いて構築された仮想記憶空間を図 2 に示す．プロセスの利用するユーザモード用の仮想記憶空間（ユーザの仮想記憶空間）およびカーネルの利用する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，2 つの仮想記憶空間を仮想アドレスの領域と指定している．Kernel page table isolation（KPTI）を適用した Linux の場合，ユーザの仮想記憶空間とカーネルの仮想記憶空間は互いに異なるページテーブルから構築される [3]．

Linux x86_64 において，プログラムコードやデータを仮想記憶空間に配置する際の割当て処理は，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と参照は，ページテーブルを辿

1）　セコム株式会社 IS 研究所
2）　岡山大学 大学院自然科学研究科

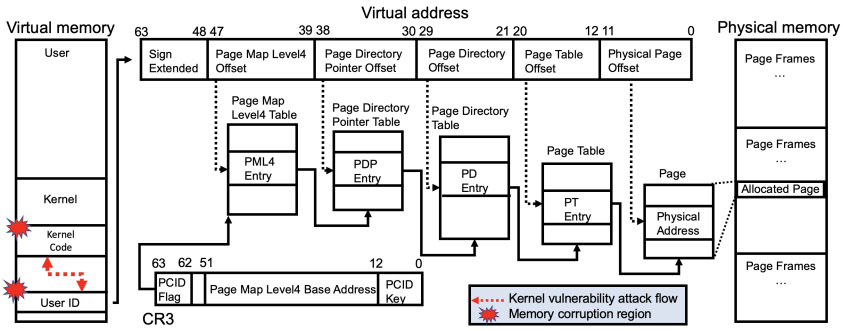

図 2　Linux における仮想記憶空間管理，および想定する脅威モデルにおける攻撃発生箇所

り，物理アドレスを特定することで実現している．

2.2　想定する脅威モデル

本稿での脅威モデルとして，図 2 に示すように，攻撃者は，カーネルの仮想記憶空間上に配置された脆弱性を含むカーネルコードを実行し，攻撃の起点となるカーネルコードを管理するページテーブルから参照可能な仮想記憶空間の書換えを可能なこととする．攻撃時は，特定の仮想アドレスを指定してカーネルデータの参照あるいは改ざんを行い，同一ページテーブル上に存在するカーネル機能を提供するカーネルデータを攻撃対象とする．

3　提案手法と実現方式

3.1　排他的ページ参照機構

我々の提案している排他的ページ参照機構では，カーネルの仮想記憶空間を構成するページテーブルにて，プロセス毎に特定のカーネルコード・データに割当てられたページを保護対象として指定することでカーネルコード・データの参照制御を可能としている．

図 1 に示すように，排他的ページ参照機構を適用することで，プロセス間でカーネルの仮想記憶空間の分割利用を可能にしている．これにより，攻撃プロセスによるカーネル脆弱性を介した攻撃から，他のプロセスの利用するカーネルコード・データの保護を実現する．

攻撃からの保護例として，カーネルの仮想記憶空間のページテーブルを排他的ページ参照機構により管理することで，カーネルモジュールからの特定のカーネルデータの参照，ならびにカーネル脆弱性を介した攻撃に対し，脆弱性を含むカーネルコードの実行の未然防止を可能であることを示した [8].

本稿で提案する項目は次の通りである．

（提案 1）　排他的ページ参照機構の制御タイミング
　排他的ページ参照機構の適用として，動作中のプロセス，ならびにカーネルに影響を及ぼさないように，ページテーブル走査，ページ参照処理を行う制御タイミングの考案

（提案 2）　保護対象とするカーネルデータ
　攻撃対象となり得る権限情報，ならびにセキュリティ機構のカーネルデータを保護対象に含めるかを明らかにする

3.2　制御タイミング

排他的ページ参照機構の適用対象となるプロセスに対し，カーネルで行われる 2 つの制御タイミングを挙げる．

制御タイミング 1　：システムコール処理前に排他的ページ制御処理を行い，保護対象のカーネルデータへの参照を制限

制御タイミング 2　：システムコール処理中に排他的ページ制御処理を行い，保護対象のカーネルデータへの参照を制限

いずれの制御タイミングにおいても，排他的ページ参照機構の処理は，カーネルの仮想記憶空間を構成するページテーブルの走査として，ページテーブルに保護対象のカーネルデータを格納するページが含まれているか探索を行い，適用対象プロセスの参照可能なページから保護対象のカーネルデータを格納するページを除外することで参照を制限する．

排他的ページ参照機構として，制御タイミング 1 はカーネルにおけるシステムコールの処理前のみ実行可能である．制御タイミング 2 はカーネルにおいてシステムコール処理中の任意の時点において実行可能とする．

排他的ページ参照機構の適用にあたり，保護対象とするカーネルデータの違いから，複数の制御タイミングを組合わせることは可能である．しかし，既にページテーブルから除外されたページは処理対象としない．

3.3　保護対象カーネルデータ

従来の排他的ページ参照機構では，保護対象をカーネルのグローバルな記憶領域として確保した仮想アドレスのページとしていた．本提案では，厳密にカーネルの仮想記憶空間上にある特定のカーネルデータを保護対象とする．

権限情報　：実行中のプロセスにおける権限に関する情報

セキュリティ関連　：セキュリティ機構の動作に必要となるアクセス制御ポリシや資源制御データに関する情報

排他的ページ参照機構の制御単位としては，カーネルの仮想記憶空間上において予め確保された仮想アドレスから参照可能なカーネルデータの格納されるページとする．

3.4　実現方式

排他的ページ参照機構の実現環境として，OS は KPTI を備えた Linux とし，CPU アーキテクチャは x86_64 を想定する．図 3 に排他的ページ参照機構を Linux に実現した際の構成を示す．

図 3 では，通常のプロセス X, Y, および攻撃元のプロセス Z の 3 プロセスが同一のカーネルの仮想記憶空間を利用している．排他的ページ参照機構が適用されるこ

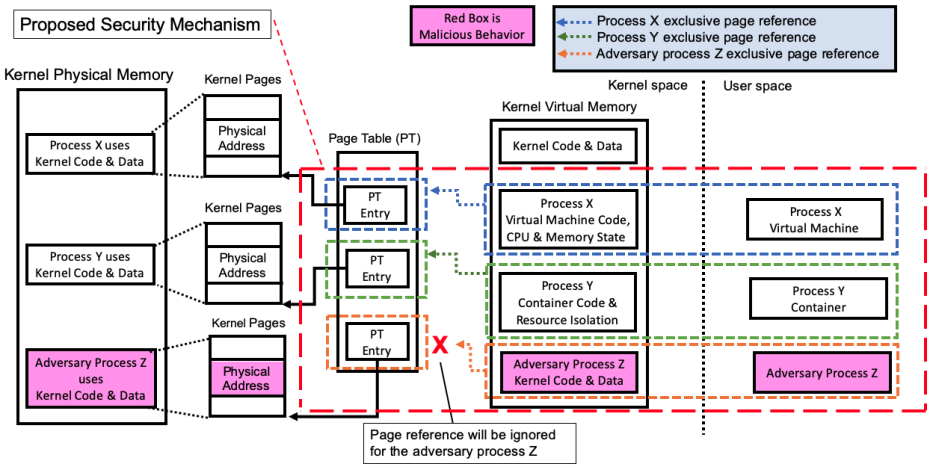

図 3　排他的ページ参照機構の実現図

とで，ページテーブル上にて参照可能なページが制御されることを示している．例として，プロセス X は仮想化機能，プロセス Y はコンテナ機能，プロセス Z は脆弱性を含むカーネルコードを利用することを想定している．排他的ページ参照機構による保護対象として，各プロセスのカーネルデータが指定され，攻撃元のプロセス Z が排他的ページ参照機構の制御対象となった場合，プロセス Z からのカーネルデータの参照の際，排他的ページ参照機構により保護対象とされたプロセス X と Y のページに対するアクセスは禁止される．

3.4.1　実現方式での制御タイミング

Linux において，カーネルの仮想記憶空間はページテーブルを示す `init_mm` 構造体の `pgd` 変数である．KPTI を適用した Linux において，プロセス生成時には，カーネルおよびユーザの仮想記憶空間が生成されるが，カーネルの仮想記憶空間はページテーブル `pgd` 変数を参照する．

実行中のプロセスからの要求を受け，カーネルでのシステムコール実行時には，`task_struct` 構造体である変数 `current` に含まれる `pgd` 変数を CR3 レジスタに書込み，参照可能なカーネルの仮想記憶空間とする．排他的ページ参照機構では，CR3 レジスタに書込まれている `pgd` 変数を制御対象のページテーブルとすることでカーネルデータの保護を実現する．実現方式における各制御タイミングでの処理を説明する．

制御タイミング 1　：`entry_SYSCALL_64` 関数におけるシステムコール関数呼出し前にページ参照制御を行う

制御タイミング 2　：`entry_SYSCALL_64` 関数におけるシステムコール関数を呼出し，関連する一連のカーネルコードの実行中にページ参照制御を行う

保護対象とするカーネルデータの仮想アドレスより，ページ番号およびページの仮想アドレスを特定する．続いて，アンマップ処理を行い，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の参照も削除することで，カーネルの仮想記憶空間からの参照を行えなくする．

表 1　排他的ページ参照機構の評価環境

評価用計算機 1	
OS	Debian 9.0 (Linux Kernel 4.4.114, x86_64)
CPU	Intel(R) Core(TM) i7 7700HQ（2.80GHz，4 コア）
Memory	16 GB
評価用計算機 2	
OS	Windows 10 Home
CPU	Intel(R) Core(TM) i5 4200U（1.6GHz，2 コア）
Memory	8 GB

3.4.2　実現方式での保護対象カーネルデータ

排他的ページ参照機構では，保護対象カーネルデータの仮想アドレス，ならびにページ制御対象か判別する識別子の管理を行う．仮想アドレスは，カーネルデータを示し，識別子は，制御対象のプロセス番号やカーネルスレッドを示す．

権限情報　：実行中のプロセス `current` 変数において，権限に関する `cred` 構造体に含まれるユーザ ID などの権限情報

セキュリティ関連　：SELinux などのセキュリティ機構の動作に必要となるカーネルコードへの参照を格納した `selinux_hooks` 変数やアクセス制御ポリシ，ならびに cgroup などの資源制御データに関する `cgroup_subsys` 構造体に関する情報

プロセスからカーネル実行中にマッピングされていない保護対象カーネルデータを参照された場合，ページフォルトが発生する．排他的ページ参照機構において，特定の仮想アドレスへのアクセスとして捕捉可能とすることでアクセス可否の判断に利用する．

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）　保護対象カーネルデータの参照制御実験

排他的ページ参照機構を適用したカーネルにおいて，保護対象のカーネルデータとして権限情報を指定し，排他的ページ参照機構により制御可能か評価した．

（評価 2）　システムコール呼出しのオーバヘッド測定

排他的ページ参照機構を適用したカーネルに

```
1. www-data$ ./cve-2017-16995
2. [*] creating bpf map
3. [*] sneaking evil bpf past the verifier
4. [*] creating socketpair()
5. [*] attaching bpf backdoor to socket
6. [*] skbuff =>ffff88001d1be400
7. [*] Leaking sock struct from ffff88001d2eab40
8. [*] Sock->sk_rcvtimeo at offset 472
9. [*] Cred structure at ffff88001d3806c0
10.[*] UID from cred structure: 33, matches the current: 33
11.[*] hammering cred structure at ffff88001d3806c0
12.[*] credentials patched, launching shell...
13.# id
14.uid=0(root) gid=0(root) groups=0(root),33(www-data)
```

図 4　排他的ページ参照機構適用前のログ

```
1. www-data$ ./cve-2017-16995
2. [*] creating bpf map
3. [*] sneaking evil bpf past the verifier
4. [*] creating socketpair()
5. [*] attaching bpf backdoor to socket
6. [*] skbuff => ffff88001d3c8b00
7. [*] Leaking sock struct from ffff88001d1b6f00
8. [*] Sock->sk_rcvtimeo at offset 472
9. [*] Cred structure at ffff88001c46df00
10.[*] UID from cred structure: -33686019,
       matches the current: -33686019
11.[*] hammering cred structure at ffff88001c46df00
12.[*] credentials patched, launching shell...
13.$ id
14.uid=4261281277 gid=4261281277 groups=4261281277,
   33(www-data)
```

図 5　排他的ページ参照機構適用時のログ

おいて，ベンチマークソフトウェアを動作させ，システムコールの実行にかかるオーバヘッドを測定した．

（評価 3）　Web クライアントのオーバヘッド測定

排他的ページ参照機構を適用したカーネルにおいて，Web サーバを動作させ，Web クライアントソフトウェア実行にかかるオーバヘッドを測定した．

評価に利用した環境を表 1 に示す．保護機能の評価，システムコール，ならびに Web クライアントのオーバヘッド測定に評価用計算機 1 を，Web クライアントの実行に評価用計算機 2 を用いた．カーネルデータ保護の評価として，CVE-2017-16995 [9] の Proof of concept（PoC）コードを用いた．排他的ページ参照機構の実装は，Linux kernel 4.4.114 に行い，40 個のファイルに対してページテーブル走査と制御タイミング 1 の機能を追加し，1,832 行で実現した．

4.2　保護対象カーネルデータの参照制御実験

CVE-2017-16995 [9] の PoC コードは権限情報のカーネルデータを改竄し，特権奪取攻撃を試みる．評価においては，攻撃対象となる権限情報として，実行中プロセスのユーザ ID を保持する `cred` 構造体を保護対象のカーネルデータとして指定した状況において，PoC コードを実行し，権限情報であるカーネルデータの仮想アドレスにアクセスした際のデータ出力を行なった．

保護対象カーネルデータの評価として，排他的ページ参照機構の適用前の攻撃成功時のログを図 4 に示す．1 行目にてユーザ ID が 33 である `www-data` ユーザにて PoC コードを実行し，9 行目から 11 行目にてユーザ ID

表 2　排他的ページ参照機構を適用した Linux におけるオーバヘッド (μs)

System call	Vanilla kernel	Our kernel	Overhead
fork+/bin/sh	958.349	2672.333	1713.984
fork+execve	284.808	881.561	596.753
fork+exit	264.473	816.531	552.058
open/close	7.079	25.238	18.159
fstat	0.359	0.559	0.200
read	0.355	0.538	0.183
write	0.313	0.425	0.112
stat	2.283	8.919	6.636

表 3　ApacheBench を用いた排他的ページ参照機構を適用した Linux におけるオーバヘッド (μs).

File size (KB)	Vanilla kernel	Our kernel	Overhead
1	1,190.5	1,792.75	602.25
10	1,950.75	3,226.25	1,275.5
100	10,332.0	16,245.5	5,913.5

をルート権限である 0 に上書きし，14 行目にて特権奪取後にシェルの起動に成功していることを確認できる．

排他的ページ参照機構の適用後の攻撃防止時のログを図 5 に示す．攻撃成功時と同様に，PoC コードの実行により，9 行目から 11 行目にてユーザ ID をルート権限への上書き試みるが，10 行目にて保護された `cred` 構造体への参照は排他的ページ参照機構により防止され，`www-data` ユーザ権限にてシェルを起動し，14 行目にて特権奪取に失敗していることを確認できる．

評価結果より，排他的ページ参照機構を有効にした場合，カーネル脆弱性を用いた攻撃の防止を可能としたうえ，カーネルの停止や対象となるプロセスの強制終了などの影響を及ぼしていないことを確認した．

4.3　システムコール呼出しのオーバヘッド測定

オーバヘッドの測定にはベンチマークソフトウェア lmbench を用い，排他的ページ参照機構の適用前の Linux kernel と適用後の Linux kernel にてそれぞれ 10 回実行し，平均値から実際のオーバヘッドを算出した．排他的ページ参照機構の制御タイミングによる影響を測定するため，システムコール呼出前に制御タイミング 1 を実行し，ページ参照制御を行う処理を測定した．

評価結果を表 2 に示す．lmbench でのシステムコール呼出回数は，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 回である．表 2 から，最もオーバヘッドの発生したシステムコール処理は fork+exit（276.029 μs），また，最も少ないオーバヘッドは write（0.112 μs）である．排他的ページ参照機構の制御タイミング 1 による，システムコール 1 回あたりのオーバヘッドは 0.112 μs から 276.029 μs となった．

評価に用いた lmbench はシステムコール発行のみの時間計測を行うため，オーバヘッドは排他的ページ参照機構の制御タイミング 1 による，ページ参照制御でのページテーブルの走査処理，ならびに保護対象のカーネルデータに対するページ除外可否の判定にかかる処理時間を示している．

4.4 Web クライアントのオーバヘッド測定

Web クライアントを実行した際のオーバヘッド測定では，Web サーバとして排他的ページ参照機構の適用前の Linux カーネルと適用後の Linux カーネルにて Apache 2.4.25 を動作させ，Web クライアントとして ApacheBench 2.4 を用いて評価した．評価条件は，通信経路 100Mbps，同時接続数 1 とし，1KB，10KB，100KB のファイルに 10 万回アクセスを 4 回実行し，1 リクエストあたりの平均値を算出した．

評価結果について表 3 に示す．排他的ページ参照機構の適用後においてのオーバヘッドは 602.25 μs から 5,913.5 μs となった．ApacheBench より発行された HTTP リクエストを処理する際の Web サーバのシステムコール発行回数は 22 回であり，システムコール毎のオーバヘッドは 27.375 μs から 268.795 μs である．

評価において，排他的ページ参照機構では制御タイミング 1 としてシステムコール発行毎に処理を行っている．ApacheBench のオーバヘッドは対象となる Web サーバプロセスで読書きするファイルサイズやメモリアクセスによるシステムコール処理時間に比例して変化することから，評価結果におけるオーバヘッドは排他的ページ参照機構に関する関数呼び出し，ページテーブル走査，ならびにページ制御にかかる負荷を示している．

5 考察

5.1 評価に対する考察

評価結果より，排他的ページ参照機構はカーネル脆弱性を利用した PoC コードにおける，権限情報の改竄処理に対して，動作中のプロセスならびにカーネルに影響なく，カーネルの仮想記憶空間を構成するページテーブルを直接走査し，ページ制御により，特定のカーネルデータを保護対象とすることが可能であることを示した．

性能評価では，排他的ページ参照機構の lmbench におけるシステムコール毎のオーバヘッドは 0.112 μs から 276.029 μs 示し，ApacheBench における Web アプリケーションのオーバヘッドは 669.0 μs から 5,945.0 μs を示した. 排他的ページ参照機構の制御タイミング 1 では，システムコールの呼出しに対し処理時間が増加すると考えられるが，負荷は一定ではないことから，走査対象のページテーブルのサイズやページ参照の処理に関する負荷があると考えられるため，カーネル処理における関数毎の実行時間などからより詳細に調査する必要がある．

今後，制御タイミング 2 の有効性についての検討，汎用アプリケーションを実行した際の性能評価を行う予定である．

5.2 カーネルデータ保護の考察

排他的ページ参照機構において，保護対象カーネルデータへのアクセスは，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可能である．これにより，カーネル脆弱性を利用した攻撃プロセスからのカーネルへの攻撃自体の事前防止が可能であると考えている．また，攻撃プロセスへも捕捉後にプロセスの停止や一時的な処理の中断など，柔軟な対処が取れる可能性がある．

動作中のカーネルにおいて，カーネルデータはグローバルに参照可能であり，アクセス制限対象とした場合，プロセスやカーネル動作への影響が懸念される．ベンチマークソフトウェアでは，カーネル全体の機能を網羅することは困難であるため，保護対象カーネルデータと関連するカーネル機能の検証が必要といえる．今後，クラウド環境で広く利用されている仮想環境機能やコンテナ機能に関するセキュリティ機構のカーネルデータに対し，排他的ページ参照機構を適用した場合の動作検証を進めていきたいと考えている．

5.3 排他的ページ参照機構の制限

実現方式では，排他的ページ参照機構の制御タイミングにおいて，ページテーブルの走査を行い，保護対象カーネルデータに関連するページを処理しているため，保護対象カーネルデータの増加，ページテーブルサイズの増大，ならびに制御タイミングの呼出し回数毎にオーバヘッドが増加する．

オーバヘッドの低減策として，プロセスにおいて，あらかじめ保護対象カーネルデータにアクセスしないことが判明している場合，プロセス生成時にページ制御処理を行い，システムコール処理時にページ参照制御行わないことで処理にかかるオーバヘッドの削減が有効であるといえる．

5.4 移植可能性

排他的ページ参照機構の他 OS への適用可能性として，カーネルにおいて，仮想記憶空間のページテーブルを採用し，ページ単位で管理している場合は適用可能といえる．一方，カーネルにてページテーブルを利用した仮想記憶空間を提供していない場合，あるいは MMU の存在しないアーキテクチャの場合，一定サイズの物理記憶領域毎にアクセス制御の細粒度管理を行うことで，排他的ページ参照機構の制御タイミングやカーネルデータへの参照制御可否を適用できる可能性はあると考えている．

6 関連研究

ハードウェアによる記憶空間の分離と保護

CPU によるハードウェア機能を利用し，記憶空間を複数に分割した Trusted Execution Environment（TEE）上において，OS を実行させ，攻撃された場合においても影響を緩和する環境を利用した手法が提案されている [10, 11, 12]．また，CPU のメモリ保護機能である Memory Protection Key を利用し，カーネルの仮想記憶空間へのアクセス制御を行う手法を提案している [13]. CPU の仮想化支援機能を利用し，カーネルの仮想記憶空間を分離する手法も提案されている [14].

カーネルの仮想記憶空間の分離

KPTI はカーネルレイヤにてページテーブル単位にてユーザとカーネルの仮想記憶空間を分離するための手法を確立した [3]．依然としてカーネルの仮想記憶空間は単一のページテーブルで管理されており，カーネルのダイレクトマッピング領域のページにユーザプロセスで確保したページ経由で攻撃を行う手法と対策が提案されている [15]．さらには，ユーザプロセス毎にカーネルの仮想記憶空間から特定のページを割当てる手法 [16]，カーネルにおいてシステムコール実行時に専用の仮想記憶空間を設ける手法 [17]，ならびにカーネルの仮想記憶空間を分割して管理するための機構 [6] が Linux カーネル開発者より提案されている．

カーネルの仮想記憶空間の保護

カーネルの仮想記憶空間に配置されるカーネルコードやデータの保護へのアクセス制御を行う手法として，

KHide は仮想化機能を利用することで読込み処理の制御し，kRˆX は読込みと実行権限の排他制御を可能としている [18, 19]．さらに，カーネルの仮想記憶空間自体を保護するため，ページテーブル位置のランダム化手法が提案されている [20]．xMP は Xen の仮想化機能を拡張し，ゲスト OS に対してユーザモード，およびカーネルモードで利用する仮想記憶空間上のデータを選択的に保護する機構を提案している [21]．

カーネルの攻撃領域削減

ユーザプロセスに対して利用可能なカーネルコードを削減することで攻撃領域を狭め，攻撃困難化を図る手法が提案されている．KRAZOR，ならびに KASR はあらかじめユーザプログラムの実行に関係するカーネルコードを調査しておき，プロセスとして実行中の際のみカーネルコードやページ単位で利用可能とすることで攻撃困難化を実現している [22, 23]，また，Multik はプロセス生成時に実行に必要となるカーネルコードのみマッピングした状態のカーネルの仮想記憶空間を構築し利用させる手法を提案している [24]．

7 おわりに

本稿では，排他的ページ参照機構におけるページ参照の制御タイミング，ならびに保護可能なカーネルデータの種別について検討し，評価を行った．排他的ページ参照機構においては，ページ参照制御により保護対象となるカーネルデータをカーネルの仮想記憶空間を構成する単一のページテーブル走査，ならびにページ制御処理により，特定のプロセスにのみ参照不可能とする．

保護能力について，カーネルデータを確実に保護可能とするための制御タイミングについて，システムコール実行前，ならびに実行中でのページ制御が有効であることを検討し，実現方式を提案した．

評価として，動作中のプロセス，ならびにカーネルの動作継続性への影響有無を調査するために排他的ページ参照機構を実現した Linux にて，実際のカーネル脆弱性を利用した攻撃の改竄対象である権限情報のカーネルデータを保護可能なことを確認した．また，オーバヘッド評価にて，システムコール毎の負荷は最大 276.029 μs であること，ならびに Web クライアントプログラムの負荷は最大 5,913.5 μs であることを示した．

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] 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, ACM (2004).

[2] Mulnix D.: Intel® Xeon® Processor D Product Family Technical Overview, https://software.intel.com/en-us/articles/intel-xeon-processor-d-product-family-technical-overview. (accessed 2018-08-10).

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

[4] Exploit Database, Nexus 5 Android 5.0 - Privilege Escalation, https://www.exploit-db.com/exploits/35711/. (accessed 2018-08-10).

[5] grsecurity: super fun 2.6.30+/RHEL5 2.6.18 local kernel exploit, https://grsecurity.net/~spender/exploits/exploit2.txt. (accessed 2018-08-10).

[6] Chartre, A.: Kernel address space isolation, https://lwn.net/Articles/813393/. (accessed 2020-06-12).

[7] Zijlstra, P.: Core scheduling, https://lwn.net/Articles/780703/. (accessed 2020-06-12).

[8] 葛野弘樹, 山内利宏:“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”, コンピュータセキュリティシンポジウム 2019 論文集, pp. 660-667, 情報処理学会 (2019).

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

[10] Ge, X., et al.: Sprobes: enforcing kernel code integrity on the trustzone architecture. In: Proceedings of the third Workshop on Mobile Security Technologies, ACM (2014).

[11] Lee, D., et al: Keystone: an open framework for architecting trusted execution environments. In: Proceedings of the Fifteenth European Conference on Computer Systems, pp. 1-16, ACM (2020).

[12] Melara, S, M., et al.: EnclaveDom: privilege separation for large-TCB applications in trusted execution environments, https://arxiv.org/abs/1907.13245. (accessed 2019-08-08).

[13] Gravani, S., et al.: IskiOS: lightweight defense against kernel-level code-reuse attacks, https://arxiv.org/abs/1903.04654. (accessed 2019-05-11).

[14] Hua. Z., et al.: EPTI: efficient defence against meltdown attack for unpatched VMs. In: Proceedings of the 2018 USENIX Annual Technical Conference, pp. 255–266, USENIX (2018).

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

[16] Hillenbrand, M., et al.: Process-local memory allocations for hiding KVM secrets, https://lwn.net/Articles/791069/. (accessed 2019-08-08).

[17] Rapoport, M.: x86: introduce system calls address space isolation. https://lwn.net/Articles/786894/. (accessed 2019-08-08).

[18] Gionta, J., et al.: Preventing kernel code-reuse attacks through disclosure resistant code diversification. In: Proceedings of the 2016 IEEE Conference on Communications and Network Security, IEEE (2016).

[19] Pomonis, M., et al.: kRˆX: comprehensive kernel protection against just-in-time code reuse. In: Protection of the twelfth European Conference on Computer Systems, pp. 420-436 (2017).

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

[21] Proskurin, S., et al.: xMP: Selective Memory Protection for Kernel and User Space. In: Protection of the 41st IEEE Symposium on Security and Privacy, IEEE (2020).

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

[23] 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), vol. 11050, pp. 691-710 (2018).

[24] Kuo, H, C., et al.: MultiK: a framework for orchestrating multiple specialized kernels. https://arxiv.org/abs/1903.06889v1. (accessed 2019-05-16).