CL-005

カーネルメモリ解析を用いた特権昇格攻撃検出手法の提案と評価

葛野 弘樹[1)]　西村 拓也[1)]　白石 善明[1)]　山内 利宏[2)]
Hiroki Kuzuno　Takuya Nishimura　Yoshiaki Shiraishi　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．攻撃調査として，extended Berkley Packet Filter（eBPF）では，カーネルコードの呼出し前後において，割込み処理を挿入し，カーネルコードの実行を追跡可能である．しかし，カーネルデータの書込み前後の操作に対しては割込み処理を挿入できず，追跡が困難である．本稿では，動作中カーネルのカーネルメモリを特権昇格攻撃の前後に出力し，メモリ解析により権限情報を格納するカーネルデータの改ざん有無を検出するための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，カーネルへの特権昇格攻撃による影響として，カーネルデータの権限情報改ざんを特定可能とする．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検出可能なことを確認した．性能評価として，特権昇格攻撃の検出にかかる時間は平均で 18.5337 秒であることを示した．

1　はじめに

オペレーティングシステム（OS）カーネルの脆弱性においてメモリ破壊を伴う攻撃が知られている．メモリ破壊攻撃として，特権昇格を目的とした権限情報を保存したカーネルデータの改ざん，ならびにセキュリティ機能の無効化を目的としたアクセス制御に関する関数ポインタを保存したカーネルデータの改ざんがある [2, 3]．

動作中のカーネルにおいて，メモリ破壊攻撃による影響を把握するためには，攻撃試行時，ならび攻撃成功後におけるカーネルの振舞いとして，カーネルコードの呼出し履歴，およびカーネルデータの改ざん有無を正確に追跡する必要がある．

Linux カーネルにおいて，動作中のカーネルにおけるカーネルコードの追跡は extended Berkley Packet Filter（eBPF）を用いることで実現可能である．著者らは，eBPF と DWARF 情報を組合わせることで，カーネルへの攻撃時に利用されたカーネルコードを収集するセキュリティ機構を提案している [4]．しかし，eBPF ではカーネルメモリに配置されるカーネルデータを指定した書込み等の追跡は困難である．カーネルデータの動的参照には，GDB 等を利用したカーネルデバッグが必要となる．動作中のカーネルに対する攻撃解析においては，攻撃の影響調査のため，カーネルデータの値変化有無の特定が必要であり，次の課題がある．

課題：カーネルデータの値変化の追跡
ユーザプロセスによる脆弱性を利用したメモリ破壊攻撃では，カーネルデータの改ざんとして，カーネルデータの格納する値の上書きが試みられる．動作中のカーネルに対する攻撃解析におけるカーネルデータの追跡では，攻撃対象となるカーネルデータについて，メモリ破壊攻撃前後で格納された値の変化を特定することが課題となる．

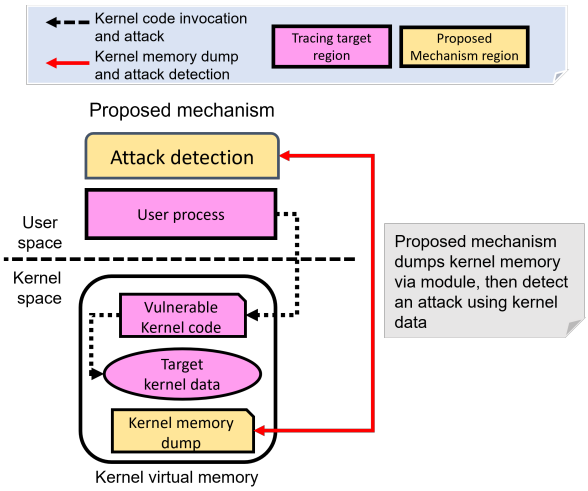

図 1　提案するセキュリティ機構の概要

本稿では，カーネルへの特権昇格攻撃の前後において，特定のカーネルデータを解析対象とし，値変化を特定するためのセキュリティ機構を提案する．

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，カーネルに対し，カーネルメモリ出力用のカーネルモジュールを挿入する．カーネルモジュールを介し，ネットワークを経由してカーネルメモリ全体をイメージファイルとして他の計算機に保存する．続いて，カーネルメモリ解析を行い，カーネルメモリに含まれるカーネルデータの値を取得する．

カーネルデータの改ざん発生有無を捕捉し，カーネルデータの値変化を特定可能とするため，提案するセキュリティ機構では，カーネルの仮想記憶空間全体をカーネルメモリのイメージファイルとする．カーネルの仮想記憶空間全体を出力対象とするため，カーネルの管理するカーネルデータはイメージファイルに含まれる．これにより，攻撃前後にてカーネルメモリのイメージファイルを解析し，解析対象のカーネルデータの値を必ず取得可能とし，カーネルデータの時系列的な変更を把握する．

提案するセキュリティ機構の実現方式では，特権昇格攻撃を試みるユーザプロセスによる権限情報を格納するカーネルデータの改ざんを検出する．実現方式では，特権昇格攻撃前後のカーネルメモリのイメージファイルを出力し，カーネルメモリ解析により，ユーザプロセスの権限情報を格納するカーネルデータに含まれる値を取得，値変化の有無を特定する．これにより，権限情報の改ざんによる特権昇格攻撃を検出可能とする．

本稿での研究貢献は以下の通りである：

1. メモリ破壊攻撃によるカーネルデータの改ざんを追跡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 に適用し，特権昇格攻撃時において，ユーザプロセスの権限情報を格納するカーネルデータの改ざんを検出可能とした．

1）　神戸大学 大学院工学研究科
2）　岡山大学 学術研究院環境生命自然科学学域

表 1　PoC の利用可能な Linux カーネル脆弱性リスト．
DoS: Denial of Service, Mem. Corr.: Memory Corruption

CVE ID	Types	Description
CVE-2016-4997[6]	DoS, Mem. Corr.	Boundary check error
CVE-2016-9793[7]	DoS, Mem. Corr.	Boundary check error
CVE-2017-1000112[8]	Mem. Corr.	Race condition
CVE-2017-16995[9]	DoS, Mem. Corr.	Boundary check error

2. 提案するセキュリティ機構の評価として，脆弱性を利用して特権昇格攻撃を行うユーザプロセスによるカーネルデータの改ざんを検出可能なことを確認した．また，カーネルメモリのイメージファイルの出力から特権昇格攻撃の検出にかかる時間が 18.5337 秒であることを示した．

2　カーネルに対する攻撃

カーネル脆弱性は，カーネルへの攻撃に利用可能な実装不備である [5]．特に，任意のカーネルコードを強制的に呼出す脆弱性，ならびにメモリ破壊に利用可能な脆弱性は，ユーザプロセスの権限情報の改ざん，およびカーネルのセキュリティ機能を無効化できることから脅威とされている [2, 3]．

2.1　特権昇格攻撃

特権昇格攻撃は，一般ユーザ権限のユーザプロセスがカーネル脆弱性を利用して管理者権限を奪取する攻撃である．Proof of Concept （PoC）が利用可能な Linux カーネルの脆弱性を表 1 に示す．特権昇格攻撃で利用されるカーネル脆弱性として，カーネルにおける権限操作処理を行うカーネルコードを強制的に呼出す脆弱性 [6, 7, 8]，ならびにメモリ破壊攻撃を利用した脆弱性がある [9]．

特権昇格攻撃では，カーネルメモリに配置されているユーザプロセスの権限情報を格納するカーネルデータの改ざんを行う．特権昇格攻撃が成功した場合，攻撃前後において，権限情報に関するカーネルデータの値は一般ユーザ権限（例，ユーザ ID 100）から管理者権限（例，ユーザ ID 0）に改ざんされる．これにより，特権昇格攻撃を行ったユーザプロセスは管理者として，計算機を操作することが可能となる．

3　脅威モデル

3.1　攻撃対象環境

本稿にて想定する脅威モデルでは，攻撃者の実行するユーザプロセスがカーネル脆弱性を利用した攻撃を行い，特権昇格を試みる．脅威モデルにおけ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行する．実行したユーザプロセスは攻撃コードを介して脆弱なカーネルコードを呼出し，特権昇格攻撃を行う．
- カーネル：特権昇格攻撃に利用可能なカーネル脆弱性を含む．攻撃者の実行する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て，権限情報に基づくアクセス制御機能のみを適用する．
- カーネル脆弱性：特権昇格攻撃に利用可能なカーネル脆弱性とし，権限操作可能なカーネルコードの強制的な呼出しを可能とする．攻撃者のユーザプロセスより，カーネル脆弱性を含むカーネルコードが実行された場合，ユーザプロセスの権限情報を管理者権限に改ざんする．
- 攻撃対象：攻撃対象は，ユーザプロセスの権限情報を格納するカーネルデータとする．

3.2　攻撃シナリオ

攻撃シナリオとして，攻撃者は，カーネル脆弱性を利用した特権昇格攻撃を試みる．まず，一般ユーザとして攻撃用のユーザプロセスを実行する．ユーザプロセスは権限情報を強制的に改ざんするカーネルコードの呼出しを行う．続いて，脆弱なカーネルコードの呼出時，攻撃対象とする権限情報を格納するカーネルデータの値の改ざんが行われる．改ざんにより，ユーザプロセスの権限情報は一般ユーザ権限（例，ユーザ ID 100）から管理者権限（例，ユーザ ID 0）に変化する．これにより，攻撃者の実行した攻撃用のユーザプロセスは管理者権限を取得し，特権昇格攻撃が成功する．

攻撃シナリオにおいて，カーネルで実行中のユーザプロセス一覧情報を取得するコマンド等から攻撃用のユーザプロセスの特権昇格攻撃前のユーザ ID は 100，特権昇格攻撃後のユーザ ID は 0 と把握可能とする．

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，動作中のカーネルに対して，指定した解析対象カーネルデータの値を取得し，値変化を特定可能とする．設計において，次の要件を満たすことを目指した．

- 要件 1: 指定する解析対象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．
- 要件 2：指定する解析対象カーネルデータは，権限情報を格納したカーネルデータとする．
- 要件 3：ユーザプロセス，ならびにカーネルに対して，指定する解析対象カーネルデータの値の取得，ならびに特権昇格攻撃の検出処理は透過的に行う．

4.2　提案手法の設計

4.2.1　設計方針

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：権限情報を格納したカーネルデータの改ざん有無を確実に特定するため，ユーザプロセスを管理するカーネルのメモリ全体を解析対象とする．
- 設計方針 2：権限情報を格納したカーネルデータの解析処理は，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，ネットワークで接続された異なる計算機で行う．

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 2 に示す．要件を満たすために，設計方針に基いて，ユーザプロセスを管理する動作中のカーネルに対して，カーネルメモリ全体の出力を行う機構を設ける．動作中のカーネルとは異なる計算機に対してカーネルメモリを出力し，イメージファイルとして保存する．これにより，ユーザプロセス，およびカーネルからは透過的にカーネルメモリの解析処理を可能とする．攻撃前後にてカーネルメモリ解析を行うことで，ユーザプロセスの権限情報の値を取得し，特権昇格攻撃による権限情報の改ざん有無の検出を可能とする．

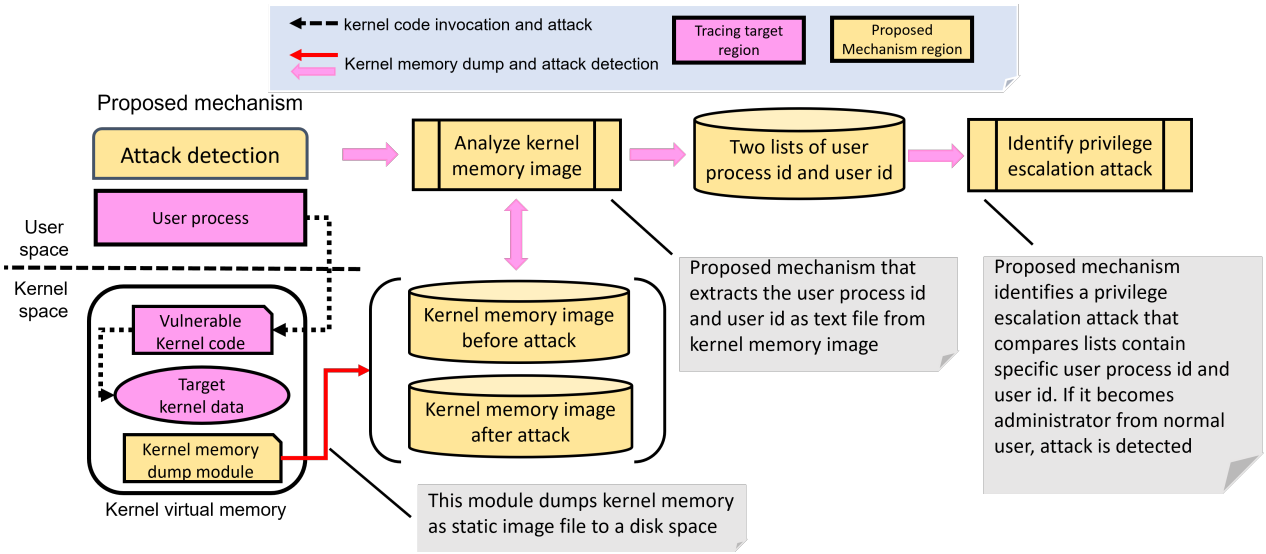

図 2　提案するセキュリティ機構の設計概要

4.2.3　解析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，特権昇格攻撃の検出に利用する解析対象カーネルデータは次の通りとする．

- 解析対象カーネルデータ：ユーザプロセスの権限情報を格納するカーネルデータ（例．ユーザ ID）

カーネルメモリのイメージファイル毎に解析対象カーネルデータをユーザプロセス単位で取得し，特権昇格攻撃の検出対象とする．

4.2.4　カーネルメモリの出力

提案するセキュリティ機構において，動作中のカーネルのメモリをイメージファイルとして出力する．

- 特権昇格攻撃前：攻撃用ユーザプロセスによる特権昇格攻撃の開始前，カーネルメモリの出力し，イメージファイルを作成
- 特権昇格攻撃後：攻撃用ユーザプロセスによる特権昇格攻撃の終了後，カーネルメモリの出力し，イメージファイルを作成

特権昇格攻撃前後において，提案するセキュリティ機構により，カーネルメモリのイメージファイルを作成する．カーネルメモリには，イメージファイルの作成時点におけるカーネルにて実行しているユーザプロセスに関する全ての情報が含まれる．

4.2.5　特権昇格攻撃の検出

解析対象カーネルデータに格納された値である権限情報の値が次の条件を満たした場合，特権昇格攻撃が発生したとみなす．

- 特権昇格攻撃とみなす条件
  特権昇格攻撃が成功した場合，攻撃用のユーザプロセスの権限情報は管理者権限に変化する．特権昇格攻撃前，特権昇格攻撃後で攻撃用のユーザプロセスの権限情報の値を比較し，通常ユーザ権限から管理者権限へ権限情報の値が変化している場合，特権昇格攻撃による改ざんとみなす．

5　実現方式

提案するセキュリティ機構の実現方式の環境は x86_64 CPU アーキテクチャの Linux とした．

表 2　実現方式における解析対象カーネルデータ

Item	Description
Target kernel data	User ID (e.g., uid, euid, fsuid, suid)

5.1　実現方式の概要

実現方式では，動作中のカーネルに対して，カーネルメモリの出力処理をカーネルモジュールとして配置する．カーネルモジュールは，カーネルの外部にネットワークを介してカーネルメモリをイメージファイルとして送信できる機能を備える．また，カーネルメモリのイメージファイル解析では，カーネルメモリ構造をプロファイルを用いて，カーネルデータの構造体や変数の型に従い，カーネルデータの格納する値を参照できる機能を備える．これにより，解析対象カーネルデータ（例．権限情報）の値取得が可能となる．

5.2　解析対象カーネルデータ

実現方式における，解析対象カーネルデータはユーザプロセスの権限情報とし，表 2 に示す． Linux カーネルにおけるユーザプロセスの権限情報は，ユーザプロセスの情報を定義した `task_struct` 構造体に含まれる `cred` 構造体の変数 `uid` に格納される． また，ユーザプロセスを識別するプロセス ID は，`task_struct` 構造体に含まれる変数 `pid` に格納される．

カーネルメモリのイメージファイルの解析は，カーネルメモリの構造に基づいて行う．解析処理においては，カーネルメモリにおいて，動作中のカーネルのユーザプロセス一覧を示す `task_struct` 構造体の変数 `init_task` の仮想アドレスを特定する．

変数 `init_task` は，ユーザプロセスの関連情報をリストとして管理している．`for_each_process` マクロにより，変数 `init_task` を起点とし，構造体 `task_struct` の各ユーザプロセス情報を参照可能である．ユーザプロセス一覧の探索中において，攻撃用ユーザプロセスかどうかをプロセス ID を利用して識別し，攻撃用ユーザプロセスであれば，`cred` 構造体の変数 `uid` より，権限情報の取得を行う．

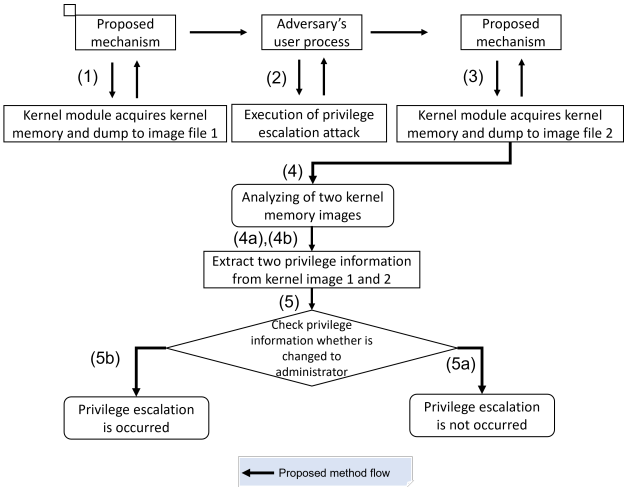

図 3　実現方式における特権昇格攻撃検出処理フロー

5.3　カーネルメモリの出力

実現方式では，Linux カーネルの仮想記憶空間をカーネルメモリ全体として出力する．Linux カーネルでは，`resource` 構造体の `iomem_resource` 変数から動作中のカーネルのメモリを参照可能である．

`resource` 構造体では，カーネルメモリを木構造として管理している．`iomem_resource` 変数に含まれる，`resource` 構造体の `iomem_resource.child` 変数は木構造の頂点ノードである．頂点ノードの配下ノードとして，`resource` 構造体の `iomem_resource.child.sibling` 変数を再帰的に辿ることで，Linux カーネルの仮想記憶空間をカーネルメモリ全体として読取り可能である．

`resource` 構造体の各ノードを示す変数の参照時，各ノードの変数に含まれるカーネルメモリの開始仮想アドレスを示す `resource_size_t` の `start` 変数，終了仮想アドレスを示す `end` 変数の範囲を把握可能である．これにより，Linux カーネルの仮想記憶空間に対して指定した仮想アドレスの範囲から順次，出力し，ネットワークを介して指定した計算機に転送し，カーネルメモリのイメージファイルとして保存する．

5.4　特権昇格攻撃の検出処理

実現方式におけるに特権昇格攻撃の検出処理のフローを図 3 に示す．特権昇格攻撃の検出処理は，カーネルメモリの出力毎に，権限情報を格納するカーネルデータの値を取得し，値の変化に着目し，次の手順にて行う．

1. 攻撃用のユーザプロセスの実行前，カーネルメモリの出力処理を開始
2. カーネルメモリを出力後，攻撃用のユーザプロセスによる特権昇格攻撃を実行
3. 攻撃用のユーザプロセスの実行後，カーネルメモリの出力処理を開始
4. カーネルメモリの解析処理を開始，解析対象カーネルデータの値を取得
   (a) 特権昇格攻撃前のカーネルメモリのイメージファイルより，解析対象カーネルデータである攻撃用ユーザプロセスの権限情報の値を取得
   (b) 特権昇格攻撃後のカーネルメモリのイメージファイルより，解析対象カーネルデータである攻撃用ユーザプロセスの権限情報の値を取得
5. 特権昇格攻撃の検出処理を開始
   (a) 権限情報の値が一般ユーザ権限から変化していない場合：特権昇格攻撃は行われていない，または攻撃失敗とみなす
   (b) 権限情報の値が異なり，管理者権限に変化している場合：特権昇格攻撃が行われており，攻撃成功とみなす

6　評価

6.1　評価項目と目的

提案するセキュリティ機構に対し，セキュリティ機能の評価として，特権昇格攻撃の検出能力，ならびに性能評価として特権昇格攻撃の検出にかかる時間を評価した．評価項目と内容を以下に示す．

1. 特権昇格攻撃の検出能力の評価
   提案するセキュリティ機構を適用したカーネルにおいて，特権昇格攻撃に利用可能なカーネル脆弱性をカーネルに導入し，攻撃用のユーザプロセスによる特権昇格攻撃成功時に攻撃を検出可能か評価した．
2. 特権昇格攻撃の検出にかかる性能評価
   提案するセキュリティ機構を適用したカーネルにおいて，カーネルメモリのイメージファイル出力からカーネルメモリ解析による特権昇格攻撃の検出にかかる時間を測定した．

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）とし，VirtualBox を利用し，攻撃対象のゲスト OS（CPU 1 コア，メモリ 1GB），および特権昇格攻撃検出を行うゲスト OS （CPU 1 コア，メモリ 3GB）を用意した．ホスト OS は Windows 11，ゲスト OS はいずれも Debian 9，Linux kernel 4.9.9-19-amd64 とした．両ゲスト OS はホスト OS 上の VirtualBox ブリッジアダプタの仮想ネットワークで接続可能とした．

提案するセキュリティ機構の実現方式の実装を Linux kernel に行うため，カーネルメモリ出力のために LiME 1.3 svn.r21[10] を用いた．また，カーネルメモリ解析のために Volatility 3 v2.4.0 [11] を用いた．特権昇格攻撃の検出は Python で実装し， 68 行にて実現した．特権昇格攻撃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特権昇格攻撃の検出能力の評価のため，次の通り，特権昇格可能な脆弱性を備えるシステムコールを導入した．

- 独自システムコール 1：独自システムコール 1 では， Linux カーネルにおける権限情報変更に利用するカーネルコードを独自システムコールを介して呼出し，ユーザプロセスの権限情報を管理者権限に強制的に上書きする．

6.3　特権昇格攻撃の検出能力の評価

特権昇格攻撃の検出能力の評価結果として，図 4 に，特権昇格攻撃を行うユーザプロセス実行時の結果を示す．攻撃用の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する．`uid` の値は 1,000 であり，一般ユーザ権限と確認できる．4 行目にて，独自システムコール 1 を呼出すことで特権昇格攻撃成功後，`uid` の値は 0 となり，管理者権限と確認できる．

また，図 4 に提案するセキュリティ機構において，特

```
// PoC code running, process id is 8065
1.  user $ ./a.out
2.  uid=1000(user) gid=1000(user) groups=1000(user)
3.  [*] sys_kvuln01 system call invocation
4.  uid=0(root) gid=0(root) groups=0(root)

// Analyzing two kernel memory images
// User process information before attack
5.  Name  Pid  PPid  Uid  Gid
6.  a.out  8065  8059  1000  1000
// User process information after attack
7.  a.out  8065  8059  0  1000
```

Red text is the points of kernel memory corruption information

図 4　提案手法による特権昇格攻撃の検出結果

表 3　提案するセキュリティ機構における処理速度 (sec)

Item	AVG	Slow	Fast
カーネルメモリ出力時間	6.5946	7.005	6.185
カーネルメモリ解析時間	11.1077	11.236	11.023
特権昇格攻撃の検出時間	0.8314	0.89	0.803
合計	18.5337	19.131	18.011

権昇格攻撃前後において，出力したカーネルメモリのイメージファイルから取得した解析対象カーネルデータとして，攻撃用ユーザプロセスの権限情報の値を示す．6 行目にて，特権昇格攻撃前の攻撃用ユーザプロセスの権限情報として uid の値は 1,000 である．特権昇格攻撃後，7 行目にて，uid の値は 0 であることが確認でき，特権昇格攻撃の検出に成功している．

6.4　特権昇格攻撃の検出にかかる性能評価

提案するセキュリティ機構の性能評価として，特権昇格攻撃の検出にかかる時間の測定を 10 回行い，平均値を算出した．測定環境として，カーネルメモリ出力処理時間，カーネルメモリ解析処理時間，ならびに特権昇格攻撃の検出処理時間の測定は，特権昇格攻撃検出を行うゲスト OS にて行った．

カーネルメモリの出力処理時間においては，攻撃対象のゲスト OS のカーネルメモリのイメージファイル出力の処理時間，ならびに特権昇格攻撃検出を行うゲスト OS のディスクに対し，ホスト OS 上の仮想ネットワークを介したカーネルメモリのイメージファイル転送の処理時間が含まれている．また，カーネルメモリのイメージファイルのサイズは 1GB とした．

表 3 から，提案するセキュリティ機構によるカーネルメモリのイメージファイル生成から特権昇格攻撃の検出にかかる平均時間は 18.5337 秒である．

7　考察

7.1　評価に対する考察

特権昇格攻撃の検出能力評価として，提案するセキュリティ機構を適用した Linux カーネルにて，特権昇格攻撃前後のカーネルメモリのイメージファイルから，攻撃者のユーザプロセスによる一般ユーザ権限から管理者権限への権限情報改ざんを検出可能なことを確認した．

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，カーネルメモリイメージの出力において，動作中のカーネルに対してオーバヘッドが必要となることを示した．オーバヘッドの要因として，カーネルメモリ全てを参照し，さらにネットワークを介してイメージファイルを出力することから，必要となる処理時間はメモリサイズに依存する．頻繁にカーネルメモリを出力する場合，動作中カーネルおよびネットワークへの負荷が大きいと考えている．一方，カーネル動作の安定性には影響を与えないことを確認した．また，カーネルメモリのイメージファイル解析においても，カーネルメモリ構造全体の参照が必要となる．そのため，処理時間の増減は解析対象となるカーネルのメモリサイズならびに解析対象のカーネルデータ数に依存すると考えている．

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

提案するセキュリティ機構の実現方式では，カーネルメモリイメージを出力，解析を行っている．カーネルメモリには，全てのユーザプロセスの情報が含まれており，カーネルメモリのイメージファイルから特定のユーザプロセスのユーザ ID を取得可能である．そのため，特権昇格攻撃前後のカーネルメモリのイメージファイルを用いることで，攻撃の影響を受けた権限情報の変化は検出可能と言える．

著者らは，カーネル脆弱性を利用した攻撃の際のカーネルコード呼出し履歴の動的な捕捉を可能としている [4]．一方，カーネル脆弱性を利用した攻撃によるカーネルデータの改ざん有無や，影響をより詳細に把握するためにはカーネルデータの値の変化を適宜，把握する必要がある．本実現方式では，ファイルとしてカーネルメモリイメージを出力するため，動作中のカーネルに影響を及ぼすことなく，カーネルへの攻撃発生前後のカーネルデータへの影響を必要に応じて解析できる可能性があると考えている．

7.2.2　限界

提案するセキュリティ機構では，特定のカーネルデータの改ざん検出を目的としている．特権昇格攻撃に対し，カーネルのメモリ改ざんを迅速に検出するためには，動作中のカーネルにおいて，任意のカーネルデータ書込みの動的捕捉が必要である．また，カーネルメモリイメージの出力は負荷が高いため，任意のカーネルデータの書込み処理に対し，低負荷かつ確実な割込みを実現可能なセキュリティ機構を検討しなければならない．

7.3　移植可能性

提案するセキュリティ機構の実現方式では，Linux のカーネルメモリをイメージファイルとして出力し，メモリ解析ツールにて解析を行っている．FreeBSD のカーネルメモリ出力 [12]，および Windows のカーネルメモリ出力 [13] は可能である．また，メモリ解析ツールにおいて，OS カーネル毎のメモリ構造をプロファイルとして生成されていれば任意のカーネルデータの値を取得できる．そのため，他の OS において，ユーザプロセスに関する情報がカーネルにて管理されており，出力したカーネルメモリイメージファイルに，ユーザプロセスの権限情報が含まれている場合，提案するセキュリティ機構は移植可能であると考えている．

8　関連研究

カーネルトレーシング

Linux カーネルのトレーシング機能として eBPF があり，動作中のカーネルに対して割込み処理を挿入し，各種カーネル機能のカーネルコードの実行を追跡可能としている [1]．また，仮想化技術を利用したトレーシング手法として，LockDoc では，Linux カーネルのメモリ操作や I/O 処理を追跡したロック処理の抽出，文書化の

試みを提案している [14]．ProbeBuilder では，動作中のカーネルのデータに対して，ポインタ配置のオフセットを解析し，適切な箇所にプローブを挿入することでカーネルデータの効率的な追跡手法を提案している [15]．

カーネルメモリ解析

カーネルメモリ解析のうち，メモリフォレンジックにおいて，不正コード挿入を検出するために，カーネルのページテーブルを取得し，解析する手法が提案されている [16]．また，Autoprofile では，カーネルメモリの解析に必要となるメモリ構造のプロファイルを自動生成する手法を提案している [17]．一方，カーネルデータの構造をメモリの断片的な情報から推定し，プロファイルを用いずにメモリ解析を行う手法が提案されている [18]．

特権昇格攻撃対策

特権昇格攻撃への対策として，PrivWatcher は，ユーザプロセスの権限情報を書込み不可のメモリ領域に配置し，ユーザプロセスにおいて参照時にのみ読込み可能とする権限情報保護手法を提案している [19]．また，PrivGuard では，カーネルスタックに権限情報の複製を格納し，システムコール呼出し後において，特権昇格攻撃による権限情報の改ざんを検出する手法の提案 [20]，AKO では，システムコール呼出し後の権限情報の改ざんを書き戻し処理を提案している [21]．

9 おわりに

本稿では，動作中のカーネルに対して，カーネルメモリをイメージファイルとして出力し，イメージファイルをメモリ解析することで，権限情報を格納するカーネルデータの改ざん有無を検出する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の実現方式では，カーネルモジュールによりカーネルメモリをイメージファイルとして攻撃前後で出力，メモリ解析により，特権昇格攻撃を試みたユーザプロセスの権限情報を格納するカーネルデータを取得し，権限情報のカーネルデータの値の変化を特権昇格攻撃として検出可能とした．

評価結果において，特権昇格攻撃前後にてカーネルメモリのイメージファイル出力可能なこと，権限情報を格納するカーネルデータを参照し，特権昇格攻撃の検出が可能であることを示した．また，性能評価として，カーネルメモリの出力から特権昇格攻撃の検出にかかる時間は 18.5337 秒であることを示した．

謝辞

本研究の一部は，JSPS 科研費 JP22H03592, JP23K16882 の助成，ならびに電気通信普及財団研究助成を受けたものです．

参考文献

[1] eBPF, https://ebpf.io/. (accessed 2023-06-10).

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

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

[4] Kuzuno, H., et al.: vkTracer: Vulnerable Kernel Code Tracing to Generate Profile of Kernel Vulnerability, *The 23rd World Conference on Information Security Applications*, LNCS, vol.13720, pp.222–234, (2023).

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

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

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

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

[10] LiME, available from https://directory.fsf.org/wiki/LiME. (accessed 2022-06-13).

[11] Volatility, available from https://www.volatilityfoundation.org/. (accessed 2022-06-13).

[12] FreeBSD memdump, available from https://www.freshports.org/sysutils/memdump/. (accessed 2022-06-13).

[13] Winpmem, available from https://winpmem.velocidex.com/docs/memory/. (accessed 2022-06-13).

[14] Lochman, A., et al.: LockDoc: Trace-Based Analysis of Locking in the Linux Kernel, *In: Proceedings of the Fourteenth EuroSys Conference 2019*, pp. 1-–15, (2019).

[15] Wang, C-W., et al.: ProbeBuilder: Uncovering Opaque Kernel Data Structures for Automatic Probe Construction, *IEEE Transactions on Dependable and Secure Computing*, vol. 13, pp. 568–581, (2016).

[16] Block, F., et al.: Windows memory forensics: Detecting (un)intentionally hidden injected code by examining page table entries, *Digital Investigation*, vol. 29, pp. S3–S12, (2019).

[17] Pagani, F., et al.: Autoprofile: Towards automated profile generation for memory analysis, *ACM Transactions on Privacy and Security*, vol. 25, issue. 1 no. 6, pp 1—26, (2021).

[18] Oliveri, A., et al.: An OS-agnostic Approach to Memory Forensics, *Network and Distributed System Security Symposium 2023*, (2023).

[19] Chen, Q., et al.: PrivWatcher: Non-bypassable Monitoring and Protection of Process Credentials from Memory Corruption Attacks, *In: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security* pp. 167-–178, (2017).

[20] Qiang, W., et al.: PrivGuard: Protecting Sensitive Kernel Data From Privilege Escalation Attacks, *IEEE Access*, vol. 6, pp. 46584–46594, (2018).

[21] 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).