CL-001

カーネルデータ監視による特権昇格攻撃検出手法の提案と評価

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において，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の検出は課題とされている．Linux カーネルでは，extended Berkley Packet Filter（eBPF）を用いてカーネルコードの呼出し前後に割込み処理を挿入し，カーネルコードの実行を追跡可能としている．しかし，eBPF は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，追跡は困難である．本稿では，動作中のカーネルにおいて，指定したカーネルデータを監視対象とし，カーネルデータの改ざん有無を遠隔から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，監視対象カーネルデータの一定間隔毎の出力による定期的な監視，ならびに一時的な書込み不可によるカーネルデータの動的な監視により，カーネルデータの変更検出を行う．提案するセキュリティ機構では，監視対象のカーネルデータの値等を遠隔に出力し，監視対象の計算機とは異なる計算機にて解析可能とする．提案するセキュリティ機構を用いることで，カーネルへの特権昇格攻撃によるユーザプロセスの権限情報改ざんを特定可能となる．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，定期的な監視，ならびに動的な監視の両手法を組合わせ，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定期的な監視は指定時間間隔を要すること，動的な監視において 0.83 秒，およびオーバヘッドは 0.726%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けるメモリ破壊攻撃として，権限情報を保存したカーネルデータを改ざんする特権昇格攻撃がある [1].

動作中のカーネルにおいて，メモリ破壊攻撃によるカーネルデータの改ざんを把握するためには，カーネルにおけるカーネルコードの呼出し，およびカーネルデータの操作を追跡する必要がある．

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 [2]，ならびにカーネルデータの変更有無をカーネルのメモリ全体から特定するセキュリティ機構を提案している [3]．しかし，依然として，カーネルデータの値変化有無の特定を軽量かつ迅速に行うには次の課題がある．

- 課題：カーネルデータの値変化の軽量かつ動的追跡
  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において，カーネルメモリ全体の定期的な調査により，値の変化は特定可能であるが，メモリ取得の負荷は高い，また改ざん発生時の迅速な検出は課題である．

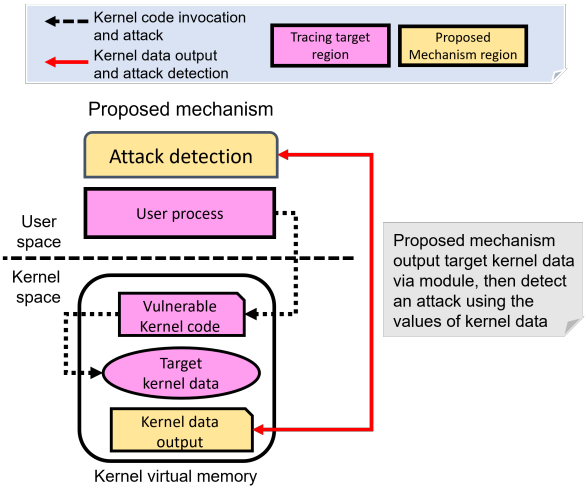

図 1　提案するセキュリティ機構の概要

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監視対象とし，値変化を軽量かつ迅速に特定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，カーネルデータの監視に 2 つの手法を備える．

1. 定期的な監視：専用カーネルページに格納した指定したカーネルデータの値等を一定間隔毎に外部出力
2. 動的な監視：指定したカーネルデータを格納するカーネルページを書込み不可に設定し，書込みが発生した場合，ページフォルトにて捕捉，カーネルデータの値等を外部出力

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，カーネルにおいて，監視対象のカーネルデータを専用カーネルページに格納する．定期的な監視はカーネルデータの変更有無の確認を行う．動的な監視はカーネルデータの変更をリアルタイムに捕捉するために行う．また，攻撃前後にてカーネルメモリ上から最小限の監視対象カーネルデータを外部出力し，カーネルデータの時系列的な変更を把握可能とする．外部出力は，監視対象カーネルデータ，ユーザプロセス ID，ならびに出力処理時刻とする．

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特権昇格攻撃とみなし，攻撃を検出する．外部出力は，ネットワークを経由した遠隔の計算機にテキストとして出力することで，仮想計算機環境等を想定した複数の計算機の監視，ならびに解析処理の負荷分散を実現する．

本稿での研究貢献は以下の通りである：

1. カーネルデータを監視し，値の変化を軽量かつ動的に検出可能とするセキュリティ機構の設計と実装を

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　カーネル脆弱性の種別 [4] (✓：提案手法による検出)

Type	Description	Target
Missing pointer check	ポインタ内容の確認漏れ	
Missing permission check	権限処理の確認漏れ	
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 [4] (✓：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	

行った．提案手法を Linux に適用し，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監視，改ざんを特権昇格攻撃として検出可能とした．

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，カーネルメモリ上のカーネルデータの値等の出力から特権昇格攻撃の検出にかかる時間は XX.XX 秒，オーバヘッドは YY.YY%あることを示した．

2　特権昇格攻撃

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 [4]．表 1 にカーネル脆弱性に関する 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により，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特権昇格攻撃が可能となる．

特権昇格攻撃として，Proof of Concept （PoC）が利用可能な Linux カーネルの脆弱性を表 3 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 [5, 6, 7]，ならびにメモリ破壊攻撃を利用した脆弱性がある [8]．

2.1　特権情報の管理

図 2 に Linux におけるユーザ ID の構造体定義を示す． 4 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）に改ざんとして，`uid` 構造体の値 `val` を `root` のユーザ ID（0）に書き換える．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．

表 3　PoC の利用可能な Linux カーネル脆弱性リスト

CVE ID	Types	Description
CVE-2016-4997[5]	DoS, Mem. Corr.	Boundary check error
CVE-2016-9793[6]	DoS, Mem. Corr.	Boundary check error
CVE-2017-1000112[7]	Mem. Corr.	Race condition
CVE-2017-16995[8]	DoS, Mem. Corr.	Boundary check error

```
1  // From Linux kernel v5.18.2 include/linux/sched.h
2  struct task_struct {
3    ...
4    const struct cred __rcu    *cred;
5    ...
6  }
7  // include/linux/cred.h
8  struct cred {
9    ...
10   kuid_t   uid; /* real UID of the task */
11   ...
12 }
13 // include/linux/uidgid.h
14 typedef struct {
15   // typedef __kernel_uid32_t  uid_t;
16   // typedef unsigned int  __kernel_uid32_t;
17   uid_t val;
18 } kuid_t;
```

図 2　Structures related to user ID in Linux[9]

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．攻撃者の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：攻撃者のユーザプロセスから特権昇格攻撃に利用可能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は，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，ユーザプロセスの権限情報を管理者権限に改ざん可能とする．
- 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

攻撃シナリオにおいて，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．最初に，攻撃者は一般ユーザ権限において攻撃用のユーザプロセスを実行する．攻撃者のユーザプロセスは権限情報を強制的に改ざんする脆弱な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）に変化し，特権昇格攻撃は成功する．

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，指定した監視対象カーネルデータの値を監視し，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．

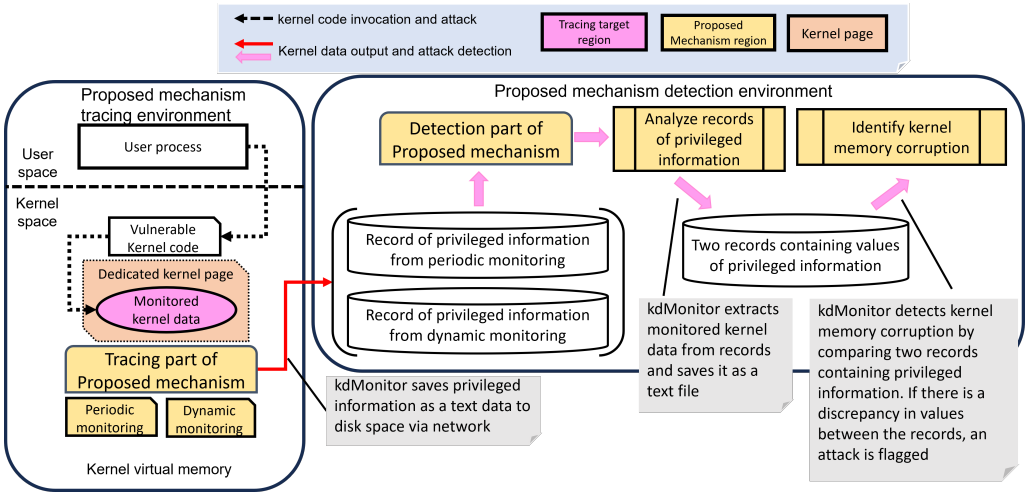

図 3　提案するセキュリティ機構の設計概要

- 要件 1：指定する監視対象カーネルデータは，権限情報等のユーザプロセスに関するカーネルデータとする．
- 要件 2：動作中のカーネルに対して，指定する監視対象カーネルデータの値の取得，外部出力を透過的に行う．
- 要件 3：カーネルデータの解析，および改ざん検出を異なる計算機において行う．

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，監視対象としたカーネルデータを格納したカーネルページを定期的に監視，出力可能とする．
- 設計方針 2：権限情報を格納したカーネルデータの改ざん時に動的に検出するため，監視対象としたカーネルデータを格納するカーネルページを導入，一時的に読込不可とし，書込み発生時にページフォルトを発生させ捕捉する．捕捉後，出力可能とする．
- 設計方針 3：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，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 3 に示す．要件を満たすために，設計方針に基づき，ユーザプロセスを管理するカーネルにおいて，指定したカーネルデータを監視対象とするセキュリティ機構を設ける．

提案するセキュリティ機構は，カーネルにて動作するため，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が可能である．提案するセキュリティ機構は，2 つの監視処理として，定期的な監視，および動的な監視を備える．両監視処理では，監視対象カーネルデータの値等を外部出力する．カーネルデータの値は，ネットワークを経由し，動作中のカーネルとは異なる計算機に対して保存する．

攻撃前後におけるカーネルメモリ上のカーネルデータの値等を確実に取得し，解析を行うことで，ユーザプロセスの権限情報の値変化を時系列に捉え，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．

4.2.3　専用カーネルページと監視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，監視における特権昇格攻撃の検出に利用するため専用カーネルページを導入する．専用カーネルページ，ならびに監視対象カーネルデータは次の通りとする．

- 専用カーネルページ：監視対象カーネルデータのみを格納するカーネルページ
- 監視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．ユーザプロセス ID，ユーザ ID）

4.2.4　監視方式

提案するセキュリティ機構における定期的な監視，および動的な監視において，全てのユーザプロセスに関して監視対象カーネルデータの値，ID，および出力処理時刻を外部出力し，特権昇格攻撃の検出に用いる．カーネルデータの値等の出力先は異なる計算機も対象とする．

- 定期的な監視：攻撃用ユーザプロセスによる特権昇格攻撃を確実に検出するため，監視対象カーネルデータの値等を定期的に外部出力
- 動的な監視：攻撃用ユーザプロセスによる特権昇格攻撃を攻撃時に検出するため，専用カーネルページの書込み権限を制御し，書込みが発生した場合，カーネルデータの値等を外部出力

4.2.5　特権昇格攻撃の検出

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．提案するセキュリティ機構では，監視対象カーネルデータに格納された権限情報の値変化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
  定期的な監視，ならびに動的な監視により出力された監視対象カーネルデータの値等から攻撃用のユーザプロセスの権限情報の値が取得される．提案するセキュリティ機構による外部出力毎に権限情報の値を前回の値と比較し，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た場合，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．

5.1 実現方式の概要

実現方式において，定期的な監視は，動作中のカーネルに対して，専用カーネルページに格納した監視対象カーネルデータを外部出力する処理をカーネルモジュールとして実現する．動的な監視では，専用カーネルページを書込み不可に設定，ページフォルト捕捉時に外部出力する処理として実現する．

特権昇格攻撃の検出は，監視対象カーネルデータの値の解析として，権限情報の値，ユーザプロセス ID の取得し，ユーザプロセスからの特権昇格攻撃が発生したか判断することで行う．実現方式において，監視対象計算機から，解析用の計算機に監視対象カーネルデータの値等をネットワーク経由にて送信し，遠隔での特権昇格攻撃の検出を行う．

5.2 権限情報管理

ユーザプロセスの生成時，専用カーネルページ（4KB）を生成し，4 に示す監視対象カーネルデータを格納する．

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．実現方式では，専用カーネルページ `cred_kernel_page` 構造体を新たに定義，ユーザプロセスのプロセス ID を示す変数 `pid`，および権限情報を示す `cred` 構造体を格納し，ユーザプロセスの権限管理を行う．

5.3 監視方式

実現方式における定期的な監視，ならびに動的な監視は次の通りである．

- 定期的な監視：`timer_list` に指定時間，ならびにカーネルデータの外部出力処理を追加し，カーネルデータの値等を定期的に外部出力する．
- 動的な監視：専用カーネルページ `cred_kernel_page` の page table entry （PTE）における flag に対し，`_PAGE_RW` をクリアすることで，書込み不可とする．書込みが発生した場合，ページフォルトを発生させ，ページフォルト処理を呼出し，カーネルデータの値等を外部出力する．

5.4 カーネルデータの外部出力

実現方式における定期的な監視，ならびに動的な監視でのカーネルデータの外部出力は，ユーザプロセスの関連情報リスト変数 `init_task` を用い，`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，全てのユーザプロセスの専用カーネルページ `cred_kernel_page` に格納したカーネルデータの値として，ユーザプロセスの権限情報，ID，および処理時刻を外部出力する．

実現方式におけるネットワークを経由したカーネルデータの値等の送受信は，実現方式のカーネルデータの値等の外部出力送信側が予め指定した TCP ポート番号にて待ち受ける．カーネルデータの値等の受信側は，ネットワークを介して指定した TCP ポート番号に接続し，カーネルデータの値等をテキストにて受信，保存する．定期的な監視の場合，送信側にて一定間隔毎にカーネルデータの値等を送信，受信側にて保存する．動的な監視の場合，送信側からカーネルデータの値等の強制的に送信し，受信側にて保存する．

表 4 実現方式における監視対象カーネルデータ

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

5.5 ページフォルト処理

動的な監視では，権限情報への書き込みを動的検出のため，ユーザプロセスの起動時に，権限情報を格納後，専用カーネルページは書込み不可として設定する．

動的な監視では，解析対象カーネルページ `cred_kernel_page` への書込みが発生した場合，ページフォルトが起こる．ページフォルト関数 `do_page_fault` にて権限情報への書き込みを捕捉し，専用カーネルページへの書込みは許可する．同時にカーネルメモリの出力フラグを設定し，権限情報の変更が行われた後，カーネルメモリを出力する．

5.6 特権昇格攻撃の検出処理

実現方式における特権昇格攻撃の検出処理のフローを図 4 に示す．特権昇格攻撃の検出処理は，カーネルデータの出力毎にカーネルデータの値等を解析，権限情報を格納するカーネルデータの値の変化に着目し，次の手順にて行う．

1. 攻撃用のユーザプロセスの実行前，定期的な監視による全てのユーザプロセスのカーネルデータの値等の出力処理を開始
2. 攻撃用のユーザプロセスによる特権昇格攻撃を実行
3. 動的な監視によりページフォルトを捕捉
4. 動的な監視によりカーネルデータの値等を出力
5. カーネルデータの値等の解析処理を開始，特権昇格攻撃の検出を行う
   (a) 特権昇格攻撃後のカーネルデータの値として，攻撃用ユーザプロセスを含む全てのユーザプロセスに対して監視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
6. 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るユーザプロセスが存在しない場合：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ユーザプロセスが存在する場合：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，定期的な監視，および動的な監視により，攻撃用のユーザプロセスによる特権昇格

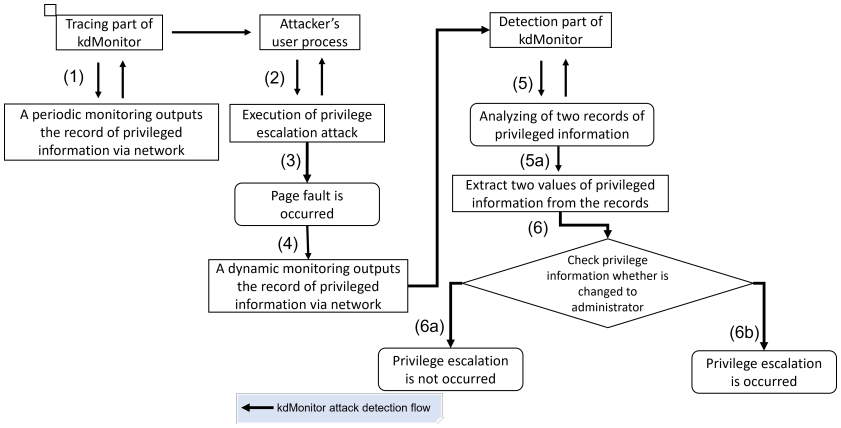

図 4　実現方式における特権昇格攻撃検出処理フロー

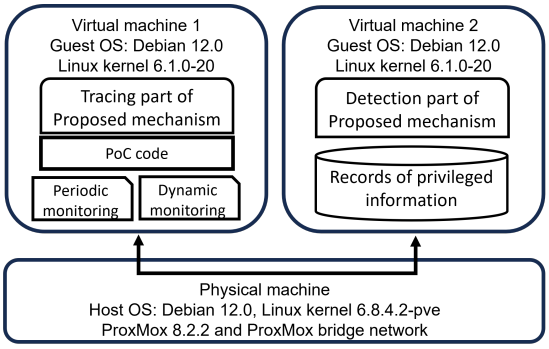

図 5　提案するセキュリティ機構の評価環境

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，定期的な監視，および動的な監視によるカーネルデータの外部出力，ならびにカーネルデータの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．
3. 特権昇格攻撃の検出にかかる性能負荷評価
   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，カーネル性能に対する負荷をベンチマークソフトウェアを用いて測定した．

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（3.00 GHz，18 コア，メモリ 32 GB）とし，仮想環境として ProxMox 8.1 を利用する．評価環境を図 5 に示す．攻撃対象の仮想マシン 1，特権昇格攻撃検出を行う仮想マシン 2（CPU 2 コア，メモリ 4 GB）を用意した．仮想マシンの OS はいずれも Debian 12，Linux kernel 6.1.0-20-amd64 とした．2 台の仮想マシンは，ProxMox の仮想ネットワークに接続し，相互に通信可能とした．

提案するセキュリティ機構の実現方式の実装を Linux kernel に行い，3 個のファイルに対して XX 行の追加にて実現した．監視対象カーネルデータの値等のネットワーク経由での出力は，カーネルモジュールで行い，特権昇格攻撃の検出は Python で実装し，68 行にて実現した．また，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 6 に，特権昇格攻撃を行う攻撃用ユーザプロセス実行結果を示す．攻撃用ユーザプロセスにて，2 行目に，攻撃用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．120 秒後，4 行目にて，独自システムコール 1 を呼出し，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．

6 行目から 8 行目にかけて，提案するセキュリティ機構における定期的な監視による結果を示す．攻撃用ユーザプロセスによる攻撃前であり，特権昇格攻撃の検出は行われない．特権昇格攻撃後，動的な監視は 9 行目から 11 行目にかけて，攻撃用ユーザプロセスの権限情報である `uid` の値が 1,000 から 0 に変更したことを確認でき，特権昇格攻撃の検出に成功している．

12 行目から 14 行目にかけての定期的な監視では，攻撃用ユーザプロセスの権限情報である `uid` は動的な監視で取得した際の 0 であることが確認できる．権限情報は変化していないため，攻撃検出は行わない．

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測定を 10 回行い，平均値を算出した．測定環境は，特権昇格攻撃検出を行う仮想計算機上の OS である．測定は，監視対象カーネルデータの出力時間，定期的な監視，および動的な監視における特権昇格攻撃の検出処理時間を対象とした．なお，定期的な監視と動的な監視では同様の監視対象カーネルデータの出力処理を用いる．定期的な監視は 120 秒間隔とし，監視直後に攻撃が行われた場合を想定し，前回からの監視経過時間を検出にかかる時間とした．

監視対象カーネルデータの出力処理時間においては，攻撃対象の仮想計算機上の OS のカーネルメモリにおける監視対象カーネルデータの出力処理時間，ならびに特

```
// PoC code running, process id is 645
1.  User $ ./a.out
2.  uid=1000(user) gid=1000(user) groups=1000(user)
3.  // wait 150 seconds
4.  [*] sys_kvuln01 system call invocation
5.  uid=0(root) gid=0(root) groups=0(root)

// Periodic monitoring for records of privileged information
// User process information (previous)
6.                              pid  uid   gid
7.  2024-06-10T15:01:26  645 1000  1000
// User process information (latest), no change
8.  2024-06-10T15:03:27  645 1000  1000

// Dynamic monitoring for records of privileged information
// User process information (previous)
9.                              pid  uid   gid
10. 2024-06-10T15:03:27 645 1000  1000
// User process information (latest), changed
11. 2024-06-10T15:03:56 645   0    1000

// Periodic monitoring for records of privileged information
// User process information before attack
12.                             pid   uid  gid
13. 2024-06-10T15:03:56 645    0    1000
// User process information (latest), no change
14. 2024-06-10T15:05:26 645    0    1000
```

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検出結果

権昇格攻撃検出を行う仮想計算機上の OS のディスクに対し， ProxMox の仮想ネットワークを介したカーネルデータのネットワーク転送の処理時間が含まれる．

表 5 から，提案するセキュリティ機構による監視対象カーネルデータの値等の平均出力時間は 0.59 秒である．また，定期的な監視における特権昇格攻撃の検出にかかる平均時間は 120.456 秒，動的な監視における特権昇格攻撃の検出にかかる平均時間は 0.83 秒である．

6.5　特権昇格攻撃の検出にかかる性能負荷評価

提案するセキュリティ機構の実現方式を備えたカーネルの性能評価をカーネルのベンチマークソフトウェア UnixBench を用いて行った．提案するセキュリティ機構の適用前の Linux カーネル，および提案するセキュリティ機構の適用後の Linux カーネルにて 5 回実行し，平均値から性能スコアを算出した．提案するセキュリティ機構において，動的な監視による定常的な性能負荷は発生しないため，定期的な監視のみ動作させる．評価における，定期的な監視での 1 回あたりの送受信データは約 800 個のユーザプロセスの情報，37.6 KB であった．

UnixBench は動作させるカーネルの性能が高いほど大きなスコア値を示し，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性能スコアを測定可能である．

UnixBench によるカーネル性能の評価結果を表 6 に示す．表 6 から，提案するセキュリティ機構の適用前後の Linux カーネルを比較し，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大きいのは Process Creation の 1.733 %であり，最もスコアへの影響が小さいのは数値計算の Double-Precision Whetstone の 0.000 % であることを確認した．提案するセキュリティ機構の実現方式はユーザプロセスの生成に関して影響があり，Process Creation に加え，Execl Throughput は 1.141%，Shell Scripts（8 concurrent）は 1.250%， Shell Scripts （1 concurrent）は 1.006%のオーバヘッドを示した．また，カーネルの性能

表 5　提案するセキュリティ機構における処理時間 (sec)

Item	AVG	Min	Max
監視対象カーネルデータの出力時間	0.59	0.32	0.86
定期的な監視	120.456	120.03	120.87
動的な監視	0.83	0.89	0.803

表 6　UnixBench を用いた提案手法の性能スコア

	Vanilla kernel	kdMonitor (Periodic)
Dhrystone 2	19584.18	19574.74 (0.048%)
Double-Precision Whetstone	6204.50	6204.48 (0.000%)
Execl Throughput	4535.06	4483.32 (1.141%)
File Copy 1024 bufsize	7954.26	7893.90 (0.759%)
File Copy 256 bufsize	5095.26	5089.02 (0.122%)
File Copy 4096 bufsize	15843.10	15740.06 (0.650%)
Pipe Throughput	3035.68	3014.50 (0.698%)
Pipe-based Context Switching	2279.26	2261.54 (0.777%)
Process Creation	3228.50	3172.56 (1.733%)
Shell Scripts (1 concurrent)	11017.90	10907.06 (1.006%)
Shell Scripts (8 concurrent)	10981.10	10843.86 (1.250%)
System Call Overhead	1637.66	1629.80 (0.480%)
System Benchmarks Index Score	5837.38	5795.00 (0.726%)

スコア全体への影響は 0.726%であることを確認した．

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，定期的な監視，および動的な監視によるカーネルメモリ上の出力から，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．また，提案するセキュリティ機構では，ネットワークを経由したカーネルデータの値等を外部出力を可能とし，監視対象の計算機とは異なる計算機においてユーザプロセス毎の権限情報の解析ならび特権昇格攻撃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，専用カーネルページの確保，ならびに監視対象カーネルデータの配置に負荷があり，ユーザプロセス生成時にオーバヘッドが発生していることを確認した．また，監視対象カーネルデータの外部出力は動作中のカーネルにおいて一定のオーバヘッドを発生させていることを示した．外部出力に関するオーバヘッドの要因は，カーネルメモリに配置されるカーネルデータの値等の参照処理，ならびにネットワークを介しての送信処理である．必要となる処理時間はカーネルメモリのサイズ，監視対象ユーザプロセス数，ならびに監視対象カーネルデータのサイズに依存する．

一定間隔毎にカーネルデータの値等を出力する場合，動作中カーネルおよびネットワークへの負荷は監視処理毎に発生している．一方，動的な監視においては，ページフォルト処理は発生するものの，送信するカーネルデータはページフォルトを引き起したユーザプロセスに限定可能であり，カーネル，およびネットワークへの負荷は限定的と考えている．また，外部出力されたカーネルデータの解析においては，前後のカーネルデータの値を全て読込む必要があり，特権昇格攻撃の検出処理時間の増減は実行中のユーザプロセス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設計では，専用カーネルページを導入した．専用カーネルページに格納した監視対象カーネルデータ，ユーザプロセス ID，出力処理時刻のみを外部出力することで監視対象の最小化，および定期的な監視，動的な監視の軽量化に対応可能とした．また，ネットワークを経由して他の計算機にカーネルデータの値等を送信，監視対象カーネルデータの継続的な変化を監視可能としており，仮想計算機環境等における複数の計算機の監視，IoT 機器の遠隔監視を行うなど，解析処理の集約化が可能であると考えている．

提案するセキュリティ機構の実装においては，ユーザプロセスの権限情報を専用カーネルページに格納した．専用カーネルページに対して，定期的な監視，動的な監視を適用し，特権昇格攻撃による権限情報の変化をカーネル動作に影響なく，対応可能なことを示した．

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，影響のより詳細な把握には多くのカーネルデータの値の変化を適宜，把握する必要がある．提案するセキュリティ機構の実現方式により，脆弱性を利用した攻撃による改ざん対象毎にカーネルデータの値をネットワーク経由にて出力し，解析可能となる．監視対象を広げ，複数のカーネルデータの値変化を解析することで，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解析できると考えている．

7.2.2 限界

提案するセキュリティ機構では，監視対象としたカーネルデータの改ざん検出を目的としている．実現方式では，監視対象カーネルデータを権限情報に限定しており，定期的な監視による監視対象カーネルデータの取得と外部出力による負荷は限定的である．また，動的な監視のためには専用カーネルページの書込み制限が必要である．

任意のカーネルデータを監視対象とし，各カーネルデータの書込み処理を監視する場合，性能負荷は監視対象数毎に増加する．そのため，提案するセキュリティ機構において，より低負荷かつカーネル変更を最小限にとどめる方式を検討しなければならないと考えている．

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，専用カーネルページに格納されたカーネルデータのみを外部出力し，解析を行っている．FreeBSD のカーネルメモリ出力 [10]，および Windows のカーネルメモリ出力 [11] は利用可能だが，カーネルメモリの全てを出力する．ユーザプロセスの権限情報等の特定のカーネルデータのみを監視対象とするためには，専用カーネルページの導入が必要であると考えている．

また，カーネルメモリ全体を解析対象とする場合，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ことからカーネルデータの改ざん検出は可能と考えている．

8 関連研究

カーネルトレーシング

Linux カーネルにおいては，eBPF を用いて動作中カーネルへのトレーシング機能を提供している． eBPF は，カーネルコードの実行前後に割込み処理を挿入し，実行追跡可能である [12]．LockDoc では，仮想化技術を利用し，仮想マシン上で動作する Linux カーネルに対するロック処理を抽出し，文書化する手法を提案している [13]．また，ProbeBuilder では，カーネルデータに対して，適切な箇所に調査用のプローブを挿入し，カーネルデータを追跡する手法を提案している [14]．

カーネルメモリ解析

カーネルメモリ解析では，カーネルのページテーブルを取得し，不正コード挿入有無を解析する手法が提案されている [15]．また，動的にカーネルメモリを解析する手法が提案されている [16]．カーネルメモリの解析を補助するため， Autoprofile では，カーネルのメモリ構造をプロファイルとして自動生成する手法を提案している [17]．一方，メモリの断片的な情報から推定し，カーネルデータの構造をプロファイルに依存せずにメモリ解析する手法が提案されている [18]．

特権昇格攻撃対策

PrivWatcher は，特権昇格攻撃対策のため，ユーザプロセスの権限情報を読込み可能とするため，権限情報を書込み不可のメモリ領域に配置する権限情報保護手法を提案している [19]．PrivGuard は，システムコール呼出し前後において，カーネルスタックに権限情報の複製を一時的に格納し，特権昇格攻撃による権限情報の改ざんを検出する手法を提案している [20]．また，AKO では，改ざんされた権限情報をシステムコール呼出し後に書き戻す手法を提案している [21]．

8.1 関連研究との比較

提案するセキュリティ機構と既存研究 [19, 20] について，保護対象，実現方式，ならびに実現方式の限界の観点からの比較を表 7 に示す．

表 7 特権昇格攻撃検出手法の比較

機能	PrivWatcher [19]	PrivGuard [20]	提案方式
監視対象	権限情報	権限情報	権限情報
実現方式	カーネル監視	カーネルデータ複製	カーネル外部監視
監視方式	カーネル内部	カーネル内部	カーネル内部・遠隔
限界	カーネル監視負荷	カーネルデータ種別	カーネルページ管理

PrivWatcher は，同一ハードウェアの隔離領域からカーネルデータの書込み権限を管理し，カーネルデータの改ざんを検出するセキュリティ機構である [19]．カーネルデータの書込みをカーネル外部から監視し，メモリ破壊を介した特権昇格攻撃を検出する．一方，監視対象カーネルデータ毎に読書き処理の適切な監視が必要であり，カーネルの外部監視に負荷が発生する．

PrivGuard は，ユーザプロセスの権限情報を予めカーネルスタックに複製，カーネルコードの実行前後で権限情報を複製と照合し，改ざん有無の検出を試みる [20]．カーネルデータ改ざんは防止せず，複製された権限情報との比較を行うため，低負荷で実現可能である．

PrivWatcher，ならびに PrivGuard は監視対象のユーザプロセスが動作するカーネルの権限情報の監視，改ざん検出を目的としている．提案するセキュリティ機構は，定期的な監視，および動的な監視による権限情報の改ざん検出をネットワークを介して可能としており，複数の計算機を遠隔から監視可能である．一方，PrivWatcher による直接的な権限情報の監視，ならびに PrivGuard によるカーネル内部における権限情報の監視と比較すると，ネットワークを介したデータの送受信や検出処理に

一定の時間を有する．

提案するセキュリティ機構における，権限情報の監視は，カーネル内部で実現しており，セキュリティ機構の攻撃耐性の向上，攻撃検出の迅速化が求められる．PrivWatcher や PrivGuard における権限情報監視手法を組み合わせ，より安全なセキュリティ機構の設計，および迅速な検出の実現が可能であると考えている．

9 まとめ

本研究では，動作中のカーネルに対して，カーネルデータの値変化を検出するセキュリティ機構を提案した．2 つの監視方式を備え，定期的な監視では，カーネルデータを指定し，一定間隔毎に外部出力する．動的な監視では，カーネルページを書込み不可に指定，カーネルデータの書込みをページフォルトにて検出し，外部出力する．外部出力はネットワークを経由して異なる計算機にテキスト形式で出力可能とし，遠隔でのカーネルデータの監視を可能とする．

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監視対象とした．評価結果において，特権昇格攻撃を試みるユーザプロセスに対し，権限情報を格納するカーネルデータの値等を定期的な監視，ならびに動的な監視においてネットワーク経由にて外部出力，解析可能なことを検証した．また，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．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として，定期的な監視は指定時間間隔を要すること，動的な監視において 0.83 秒であること，オーバヘッドは 0.726%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] Exploit Database, Nexus 5 Android 5.0 - Privilege Escalation, https://www.exploit-db.com/exploits/35711/. (accessed 2018-08-10).

[2] Kuzuno, H., et al.: Identification of Vulnerable Kernel Code Using Kernel Tracing Mechanism, *Journal of Information Processing*, vol.31, pp.788–801, (2023).

[3] 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).

[4] 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).

[5] CVE-2016-4997, available from https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-4997. (accessed 2024-05-12).

[6] CVE-2016-9793, available from https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-9793. (accessed 2024-05-12).

[7] CVE-2017-1000112, available from https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-1000112. (accessed 2024-05-12).

[8] CVE-2017-16995, available from https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-16995. (accessed 2024-06-10).

[9] The Linux Kernel Archives, available from https://www.kernel.org/. (accessed 2024-06-10).

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

[11] Winpmem, available from https://winpmem.velocidex.com/docs/memory/. (accessed 2024-06-13).

[12] eBPF, https://ebpf.io/. (accessed 2024-06-10).

[13] 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).

[14] 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).

[15] 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).

[16] Case, A., et al.: Treasure and tragedy in kmem_cache mining for live forensics investigation kmem cache mining for live forensics investigation, *Digital Investigation*, vol. 7, pp. S41-–S47, (2010).

[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).