RL-001

カーネルスタックの比較によるカーネルレベルルートキット検知システム
Kernel Level Rootkits Detection System by Comparing Kernel Stack

池上祐太 † 山内利宏 †
Yuta Ikegami Toshihiro Yamauchi

1. はじめに

近年，標的型攻撃の検知を困難にするため，ルートキットを使用する事例が増加している [1]．ルートキットとは，計算機へ侵入した痕跡，攻撃プログラムの存在，および自身の存在を隠蔽する機能などを持つマルウェアである．上記の機能により，ルートキットに感染した場合でも，計算機は，正常な場合と同様の振舞いをしているように見える．このため，利用者は，正常な場合の振舞いとルートキットに感染した場合の振舞いに気づかず，標的型攻撃の検知までに要する時間が長引く可能性が高くなる．攻撃の検知が遅れると計算機への被害が拡大する恐れがあるため，被害の抑制にはルートキットの早期検知が重要である．

ルートキットには，カーネルレベルで動作するカーネルレベルルートキット（以降，ルートキット）が存在する．ルートキットを作成するには，OS の仕組みについて深い知識と理解が必要である．しかし，2007 年以降，ルートキットの数（カーネルレベルルートキット以外も含む）は，急激に増加し，一日に約 2,000 個のルートキットが作成されている [2]．これは，Spy Eye[3] や Zeus[4] のようなマルウェア作成ツールの発展のためである．これらの現状から，ルートキットを検知する様々な方式が提案されている [5],[6],[7],[8],[9],[10],[11]．これらの方式のうち，ルートキットを早期に検知できるものは，[7],[8],[9],[10] である．しかし，早期に検知できたとしても，カーネルの拡張性を制限する問題がある．

ルートキットを早期に検知する手法として，カーネルメモリへの書き込みを禁止することで，不正なコードの挿入を検知可能な手法 [7],[8],[9] やカーネルモジュールの導入時に，カーネルモジュールのバイナリを検査する方式 [10] がある．しかし，カーネルメモリへの書き込みを禁止した場合，正規のデバイスドライバやカーネルモジュールを追加できない．このため，カーネルの拡張性が制限される．文献 [10] の手法では，正規のカーネルモジュールに，ルートキットと類似したバイナリが存在する場合，誤検知が発生し，正規のカーネルモジュールを追加できない可能性がある．また，未知のルートキットを検知できない可能性がある．

そこで，本研究では，カーネルスタックの比較により，ルートキットを早期に検知し，カーネルの拡張性を制限しないシステムを提案する．提案システムでは，感染時に，ルートキットに改ざんされる可能性の高いシステムコール（以降，監視対象システムコール）の発行後に呼び出されるカーネル関数（以降，オリジナル関数）の呼び出し前に，処理をフックし，その時点のカーネルスタックの情報を取得する．取得したカーネルスタックの情報とルートキットに感染前のカーネルスタックの情報を比較する．カーネルスタックの比較に用いる情報は，カーネルスタックのサイズとオリジナル関数の戻りアドレスである．ルートキットに感染前のカーネルスタックの情報は，事前にホワイトリストとして登録する．ルートキット感染後は，ルートキットが用意した関数（以降，ルートキット関数）がカーネルスタックに積まれる．この情報は，ルートキットに感染前には積まれないため，取得したカーネルスタックのサイズをホワイトリストと比較することで，ルートキットを検知できる．また，戻りアドレスをホワイトリストと比較することで，システムコール処理を改ざんしたプログラムがルートキットなのか正規のプログラムなのか判別できる．これにより，正規のプログラムの誤検知を防止でき，カーネルの拡張性を制限しない．

提案システムは，ルートキットに感染後，最初の監視対象システムコール発行時に，ルートキットを検知できる．このように，ルートキットを早期に検知できるため，計算機への被害を最小限に抑制できる．本論文では，提案システムの Linux を対象とした実現方式と評価結果について述べる．

2. ルートキットの感染手法と既存の検知手法の問題点

2.1. ルートキットの感染手法

ルートキットの主な感染手法を以下に示す．

(1) システムコールの改ざん
システムコールテーブルに格納されているシステムコールサービスルーチンのアドレスをルートキット関数のアドレスに改ざんする．

(2) システムコールテーブルのアドレスの改ざん
システムコールテーブルのアドレスをルートキットが用意したシステムコールテーブルのアドレスに改ざんする．

(3) IDT（Interrupt Descriptor Table）の改ざん
IDT の 0x80 番目のエントリをルートキット関数のアドレスに改ざんする．これにより，int 0x80 命令によりシステムコールを発行すると，ルートキット関数が呼び出される．

(4) SYSENTER_EIP_MSR の改ざん
SYSENTER_EIP_MSR に格納されているアドレスをルートキットが用意したシステムコールハンドラのアドレスに改ざんする．これにより，SYSENTER 命令によりシステムコールを発行した場合，ルートキットが用意したシステムコールハンドラが呼び出される．

(5) IDTR（Interrupt Descriptor Table Register）の改ざん

†岡山大学大学院自然科学研究科，Graduate School of Natural Science and Technology, Okayama University

IDTR をルートキットが用意した IDT のアドレスへ改ざんする．

(6) カーネル関数の改ざん

カーネル関数の先頭コードをルートキット関数へのジャンプ命令に改ざんする．これにより，カーネル関数が呼び出されると，ルートキット関数へ処理が移る．

(7) 動的なデータ領域の改ざん

Linux は，カーネルモジュールを連結循環のリンクリストで管理している．ルートキットは，リンクリストのポインタを改ざんすることで，特定のカーネルモジュールを切り離す．これにより，特定のカーネルモジュールを隠蔽できる．

(8) オリジナル関数の呼び出しの横取り

多くのルートキットは，ルートキット関数からオリジナル関数を呼び出し，正しいシステムコールの処理へ移行させる．しかし，ルートキット関数からオリジナル関数を呼び出さず，システムコールを改ざんすることもできる．

2.2. 既存のルートキット検知手法

2.2.1. メモリ完全性検査

文献 [5],[6] の手法は，ルートキットに感染前と感染後のカーネルメモリを比較し，差異がないか検査する手法である．カーネルメモリを比較するため，未知のルートキットや亜種を検知できる．文献 [5] は，PCI カードを用いて，異なる物理計算機から監視対象の計算機のメモリを周期的に監視する．しかし，この手法では，検査の周期によっては，ルートキットの検知が遅れる可能性がある．

文献 [6] は，ルートキットに感染前のカーネルメモリを固定長に区切り，保存する．カーネルメモリの保存後，システムコール発行のたびに，保存したカーネルメモリとその時点のカーネルメモリを比較する．しかし，一回のシステムコール発行につき，区切ったサイズ一つ分しかカーネルメモリを比較できない．このため，頻繁にシステムコールを発行しない環境では，検知が遅れる問題がある．

また，文献 [5],[6] は，カーネルメモリの完全性を検査するため，正規のカーネルモジュールを追加できない問題がある．

2.2.2. カーネルメモリへの書き込みを禁止することによる検知

文献 [7],[8],[9] は，カーネルメモリへの書き込みを禁止する検知手法である．保護対象 OS の仮想アドレスに対応するページテーブルエントリの WR ビットをクリアする．WR ビットがクリアされたメモリ領域へ書き込みがされた場合，ページ例外が発生する．このページ例外を検知することで，ルートキットの挿入を検知できる．しかし，メモリ領域の書き込みを禁止することで，正規のカーネルモジュールが追加できない問題がある．また，OS によりメモリ構造が異なるため，特定の OS やバージョンにしか対応できない問題がある．

2.2.3. バイナリ検査

文献 [10] は，カーネルモジュールのロード時に，カーネルモジュール のバイナリを検査し，ルートキットを検知する．カーネルモジュール のバイナリを検査するため，未知のルートキットや亜種を検知できる．しかし，正規のカーネルモジュールにルートキットと酷似したコードが存在する場合，誤検知が発生する．また，事前にルートキットが使用する可能性の高いコードのバイナリを登録する必要があるため，登録したバイナリ以外のコードを使用するルートキットは，検知できない．

2.2.4. クロスビュー検知

文献 [11] は，ルートキットに感染した OS で取得したファイル一覧と CD ブートした OS で取得したファイル一覧を比較する．クロスビュー検知とは，ユーザレベルで取得したファイル一覧とファイルシステムやレジストリをスキャンし，取得したファイル一覧を比較する手法である．ルートキットに感染していない OS で取得したファイル一覧を比較に用いるため，ルートキットによるファイル隠蔽を検知できる．しかし，文献 [11] の手法では，利用者の任意のタイミングで一度計算機を停止し，検査する必要がある．このため，検知が遅れる問題がある．

2.3. 既存の検知手法の問題点

これまでに述べた既存の検知手法には，以下のいずれかの問題が存在する．

(問題 1) ルートキットを早期に検知不可能

ルートキットの検知には，(問題 1) への対処が最も重要である．文献 [7],[8],[9] 以外の検知手法では，利用者が設定した周期やタイミングで検知システムを動作させる必要がある．このため，ルートキットの検知が遅れる可能性がある．

(問題 2) カーネルの拡張性の制限

文献 [10],[11] 以外の検知手法では，カーネルメモリへの書き込みを禁止するため，動的にカーネルモジュールやデバイスドライバを追加できない．

(問題 3) 特定の OS のバージョンのみ対応可能

文献 [5],[6],[11] 以外の検知手法は，OS の種類やバージョンに依存するため，多種の OS への適応性が低い．

3. カーネルスタックの比較によるカーネルレベルルートキット検知システムの設計

3.1. 考え方

本論文では，ルートキット感染時のシステムコール処理において，オリジナル関数の呼び出しがルートキットにフックされること，またはオリジナル関数が呼ばれず，その代わりにルートキット関数が呼ばれることを検知し，2.3 節で述べた問題を解決するシステムを提案する．

具体的には，監視対象システムコールの発行後に呼び出されるオリジナル関数の呼び出し前に，処理をフックし，その時点のカーネルスタックの情報を取得する．カーネルスタックの情報とは，オリジナル関数の呼び出し前のカーネルスタックのサイズと戻りアドレスである．取得したカーネルスタックの情報とルートキッ

トに感染前のカーネルスタックの情報を比較することで，ルートキットに感染後，最初に監視対象システムコールが発行された際に，ルートキットを検知できる．

また，システムコール発行の直後に，前回のシステムコール発行が監視対象システムコールであった場合，カーネルスタックの情報を取得できているか検査する．前回のシステムコール発行時に，オリジナル関数が呼び出されていなければ，ルートキットにより処理をフックされていたことが分かる．これにより，オリジナル関数を呼び出さないルートキットを次のシステムコールの呼び出し時に検知できる．上記の処理により，(問題 1) を解決できる．

(問題 2) を解決するために，監視対象システムコールをフックする処理を行う正規のカーネルモジュールを利用する場合は，事前に動作させ，カーネルスタックの情報を登録する．登録したカーネルスタックの情報とオリジナル関数呼び出し前のカーネルスタックの情報を比較することにより，正規のカーネルモジュールをルートキットと誤検知せず，カーネルの拡張性を制限しない．

(問題 3) を解決するために，提案システムを LKM で実現する．提案システムは，カーネルのシンボルテーブル（System.map）を参照し，システムコールをフックする．このため，System.map を取得可能であれば，OS のバージョンへの依存性は低い．

提案システムは，2 つの利用フェーズ（ホワイトリスト作成期間と実運用期間）を想定している．ホワイトリスト作成期間は，提案システムの導入後，ルートキットに感染前のカーネルスタックの情報と正規のカーネルモジュールの戻りアドレスを取得し，ホワイトリストを作成する期間である．実運用期間は，その時点のカーネルスタックの情報をホワイトリストと比較し，ルートキットを検知する期間である．

各問題への対処から，提案システムへの要求は以下のようになる．

(要件 1) システムコール発行とオリジナル関数呼び出しの監視

(要件 2) カーネルスタックの情報の取得

(要件 3) ホワイトリストの作成

3.2. 提案システムの構成

提案システムの全体像を図 1 に示す．提案システムは，以下の 4 つの機構により実現する．

- システムコールフック機構
- システムコール呼び出し確認機構
- カーネルスタック情報取得機構
- カーネルスタック比較機構

提案システムの処理の流れを以下に示す．

(1) システムコール発行をフック

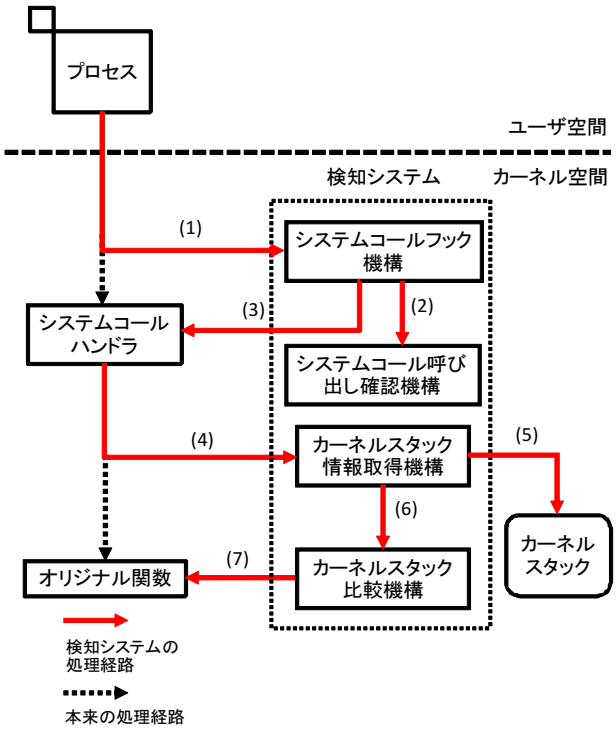

図 1 提案システムの全体像

(2) システムコール呼び出し確認機構を呼び出し

(3) 本来のシステムコールハンドラを呼び出し

(4) オリジナル関数をフック

(5) カーネルスタックの情報を取得

(6) カーネルスタックを比較

(7) オリジナル関数を呼び出し

なお，本論文では，提案システムへの攻撃やシステムの管理者による不正は想定しない．(要件 1) には，システムコールフック機構，システムコール呼び出し機構，およびカーネルスタック比較機構により対処する．(要件 2) と (要件 3) には，カーネルスタック情報取得機構により対処する．

3.3. システムコールフック機構

システムコールフック機構は，システムコール発行後に呼び出されるシステムコールハンドラの呼び出し前に，処理をフックし，システムコール番号を取得する．取得したシステムコール番号は，システムコール呼び出し確認機構で使用する．

Linux 2.6 以降では，システムコールの発行に，SYSENTER 命令を使用する．このため，システムコールフック機構は，SYSENTER_EIP_MSR をシステムコールフック機構のアドレスに書き換えることで，システムコール処理をフックする．SYSENTER_EIP_MSR の書き換えは，提案システムの導入時に行う．これにより，システムコール発行のたびに提案システムへ処理が移行する．

3.4. システムコール呼び出し確認機構

システムコール呼び出し確認機構は，まず，前回に発行されたシステムコールのシステムコール番号を参照し，監視対象システムコールか否かを検査する．監視対象システムコールの場合，前回に発行されたシステムコールにおいて，カーネルスタックの情報を取得できているか検査する．カーネルスタックの情報を取得できていない場合，オリジナル関数が呼び出されていないことを検知できる．これにより，オリジナル関数を呼び出さないルートキットを次のシステムコール呼び出し時に検知できる．

監視対象システムコールは，文献 [12] を参考に，exit()，fork()，read()，write()，open()，close()，execve()，ioctl()，readlink()，stat64()，lstat64()，getuid32()，および getdents64() の 13 個に決定した．文献 [12] は，10 種類のルートキットを解析し，ルートキットがフックするシステムコールを分析している．

3.5. カーネルスタック情報取得機構

カーネルスタック情報取得機構は，オリジナル関数呼び出し前に，処理をフックし，カーネルスタックの情報を取得する．カーネルスタックの取得する情報は，カーネルスタックのサイズと戻りアドレスである．また，ホワイトリスト作成期間では，カーネルスタックの情報と監視対象システムコールをフックする正規のカーネルモジュールの戻りアドレスを取得し，保存する．カーネルスタック情報取得機構の処理の流れを以下に示す．

(1) オリジナル関数をフック

(2) 現在の EBP レジスタの値を取得

(3) オリジナル関数の戻りアドレスを取得

(4) thread_info 構造体のアドレスを取得

(5) カーネルスタックの底のアドレスを取得

(6) カーネルスタックのサイズを取得

提案システムは，オリジナル関数をフック後に，EBP レジスタの値を取得する．EBP レジスタとは，スタックフレームの底のアドレスを格納するレジスタである．戻りアドレスは，フレームポインタの 4 バイト上位のアドレスへ格納されている．このため，EBP レジスタの値に，4 バイト加算することで，提案システムの戻りアドレスを取得できる．続いて，カーネルスタックのサイズを取得するため，カーネルスタックと共有してメモリを割り当てられている thread_info 構造体のアドレスを取得する．カーネルスタックと thread_info 構造体に割り当てられているメモリ領域の大きさは，8,192 バイトである．このため，thread_info 構造体のアドレスに，8,192 バイトを加算することで，カーネルスタックの底のアドレスを取得できる．カーネルスタックの底のアドレスから，現在の ESP レジスタを減算することで，カーネルスタックのサイズを取得できる．ESP レジスタとは，スタックフレームの先頭のアドレスを格納するレジスタである．

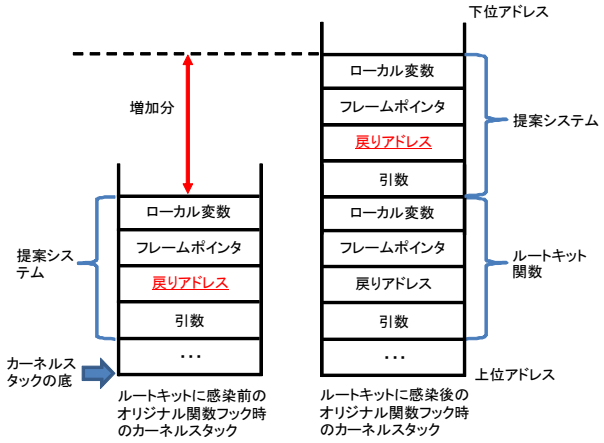

図 2 カーネルスタックの比較

3.6. カーネルスタック比較機構

カーネルスタック比較機構は，ホワイトリストとその時点のカーネルスタックの情報を比較する．ルートキットに感染前と感染後のカーネルスタックの比較を図 2 に示す．図 2 のように，ルートキット感染後のカーネルスタックは，ルートキットに感染前のカーネルスタックと比べ，ルートキット関数の分だけサイズが大きくなる．また，カーネルスタックに積まれている提案システムからの戻りアドレスがルートキット関数のアドレスを指すように変化する．比較結果が異なる場合，ルートキットに感染していると判断し，利用者への警告としてログを出力する．

3.7. 利用形態

提案システムの利用形態は，ホワイトリスト作成期間と実運用期間の 2 つに分類される．ホワイトリスト作成期間は，提案システムを導入し，ホワイトリストの作成が終わるまでの期間である．ホワイトリスト作成期間は，監視対象システムコールがすべて呼ばれた時に，終了する．

実運用期間は，ホワイトリスト作成期間の終了から，カーネルスタックを比較し，ルートキットを検知する期間である．

3.8. 期待される効果

提案システムの導入により期待される効果を以下に示す．

(1) ルートキットを早期に検知可能

提案システムは，システムコール処理を監視するため，ルートキットを早期に検知できる．提案システムは，ルートキットに感染後，カーネルスタックにルートキットの情報が初めて積まれた段階でルートキットを検知できる．ルートキットによるシステムコールの改ざんを早期に検知できるため，ルートキットによるシステムへの被害を最小限に抑制できる．

(2) カーネルの拡張性を制限しない

正規のカーネルモジュールの戻りアドレスを提案システムのホワイトリストへ登録するため，正規のカー

```
[   47.464952] Rootkit detector: Start rootkit detection.
[   64.060113] Rootkit: Inserted.
[   64.060360] Rootkit detector: Detected a rootkit.
```

図 3 KBeast を検知した際のログ

```
[ 1692.992344] Rootkit detector: Start rootkit detection.
[ 2034.868356] Rootkit: Inserted.
[ 2034.868728] Rootkit detector: Detected a rootkit.
```

図 4 Hook_getuid を検知した際のログ

ネルモジュールをルートキットと誤検知しない．このため，動的にカーネルを拡張するカーネルモジュールやドライバの導入を制限しない．

(3) OS のバージョンによる依存性が低い

提案システムが OS に依存するのは，カーネルのシンボルテーブルのみである．Linux では，カーネル導入時にシンボルテーブルを生成し，特定のディレクトリに保存する．提案システムは，シンボルテーブルのみ取得できればよいため，OS のバージョンへの依存性は低い．

(4) 既存の検知手法との併用による検知対象の拡張

提案システムは，カーネルスタックの情報を比較し，ルートキットを検知する．既存の検知システムには，カーネルスタックを比較するものはない．このため，提案システムを既存の検知システムと併用することにより，検知対象を拡張できる．

4. 評価

4.1. 評価の目的と評価環境

本評価の目的と評価の観点を以下に示す．

(1) ルートキットの検知実験

提案システムによりルートキットを検知できることを示し，ルートキットに感染前と感染後のカーネルスタックの変化を明らかにする．また，ルートキットの検知にかかる時間を明らかにする．

(2) 監視対象システムコールにおけるオーバヘッドの測定

提案システムの導入による監視対象システムコールのオーバヘッドを測定し，その影響を示す．

評価環境は，CPU は，Intel Pentium 4 3.60 GHz，メモリは 2.0 GB，およびカーネルは Linux 2.6.32-5 である．

4.2. ルートキットの検知実験

検知実験で使用するルートキットを以下に示す．

- KBeast
- Hook_getuid

KBeast は，Linux 2.6.32 上で動作し，システムコールテーブルを改ざんするルートキットである．Hook_getuid は，自身で作成したルートキットであり，getuid32() を改ざんし，オリジナル関数を呼び出さないルートキットである．

表 1 KBeast に感染前と感染後のカーネルスタックのサイズの変化

変化したシステムコール	KBeast に感染前 (Bytes)	KBeast に感染後 (Bytes)
open()	116	148
read()	116	176
getdents64()	116	156

表 2 KBeast に感染前と感染後の提案システムの戻りアドレスの変化

変化したシステムコール	KBeast に感染前	KBeast に感染後
open()	0xc0408e7c	0xf7e530c3
read()	0xc0408e7c	0xf7e527f3
getdents64()	0xc0408e7c	0xf7e52c10

評価のため，2 つのルートキット のプログラムを改変し，ルートキットの感染時に「Rootkit: Inserted.」という文字列をログとして出力させる．また，提案システムがルートキットを検知した場合，「Rootkit detector: Detected a rootkit.」という文字列をログとして出力させる．

図 3 に提案システムが KBeast を検知した際のログの出力を示す．ログに表示された時間から，提案システムは，KBeast に感染してから，0.25 ms で KBeast を検知できた．図 4 に提案システムが Hook_getuid を検知した際のログの出力を示す．ログに表示された時間から，提案システムは，KBeast に感染してから，0.37 ms で Hook_getuid を検知できた．これらの結果から，提案システムは，ルートキットの感染を早期に検知できたことが分かる．Hook_getuid の検知速度が KBeast よりも 0.12 ms 遅くなったのは，オリジナル関数を呼び出さないルートキットは，次の監視対象システムコールの発行時に検知するためである．

次に，KBeast に感染前と感染後のカーネルスタックのサイズの変化を表 1 に示す．KBeast に感染前と感染後のカーネルスタックのサイズは，open() の場合は 32 Bytes，read() の場合は 60 Bytes，getdents() の場合は 40 Bytes 増加した．また，KBeast に感染前と感染後の戻りアドレスの変化を表 2 に示す．KBeast に感染前と感染後で，オリジナル関数呼び出し前の戻りアドレスが変化している．KBeast に感染後のオリジナル関数呼び出し前の戻りアドレスは，KBeast の関数のアドレスである．この結果から，ルートキットの挿入により，オリジナル関数呼び出し前の戻りアドレスが変化することが分かる．

4.3. 監視対象システムコールにおけるオーバヘッドの測定

提案システムが監視対象システムコールのオーバヘッドを表 3 に示す．また，表 4 に，read() と write() のシステムコールにおいて，1 KB 分を入出力する場合

表 3 監視対象システムコールのオーバヘッド（単位: μs）

システムコール	提案システム導入前	提案システム導入後	オーバヘッド
fork() +exit()	142.18	142.49	0.31（0.22%）
fork() +execve()	487.64	490	2.36（0.48%）
open()	1.61	1.62	0.01（0.62%）
close()	0.27	0.29	0.02（7.4%）
ioctl()	0.90	0.92	0.02（2.2%）
readlink()	2.32	2.42	0.10（4.3%）
stat64()	0.98	0.99	0.01（1.02%）
lstat64()	1.01	1.02	0.01（0.99%）
getuid32()	0.25	0.26	0.01（4.0%）
getdents64()	0.28	0.29	0.01（3.6%）

表 4 read() と write() のオーバヘッド（単位: μs）

システムコール	ファイルサイズ	提案システム導入前	提案システム導入後	オーバヘッド
read	1 KB	1.80	1.86	0.06（3.3%）
	100 KB	21.12	21.49	0.37（1.8%）
write	1 KB	0.90	0.95	0.05（0.56%）
	100 KB	12.77	13.01	0.24（1.9%）

と 100 KB 分を入出力した場合のオーバヘッドを示す．表 3 と表 4 のスコアは，各システムコールを 1,000 回実行し，1 回当たりの平均値を算出した．表 3 と表 4 から，1 回のシステムコール当たり，0.01 μs から 2.36 μs のオーバヘッドであり，十分に小さいことがわかった．ほとんどの場合で，0.4 μs 未満のオーバヘッドである．readlink()，read()，および write() は，ディレクトリの探索やファイルの入出力のサイズが増加することにより，オーバヘッドが他のシステムコールよりも大きくなったと考えられる．

5. おわりに

既存のルートキット検知手法の問題点を 3 つ述べ，これらの問題を解決するために，カーネルスタックの比較によるルートキット検知システムを提案した．提案システムは，監視対象システムコールの発行後に呼び出されるオリジナル関数の呼び出し前に，処理をフックし，その時点のカーネルスタックの情報を取得する．その後，取得したカーネルスタックの情報とホワイトリストを比較する．このため，ルートキットに感染後，最初の監視対象システムコール発行時，または最初の監視対象システムコールのシステムコール発行時に，ルートキットを検知できる．これにより，ルートキットを早期に検知できる．また，カーネルスタックを比較する情報として，戻りアドレスを比較する．戻りアドレスを比較することで，呼び出し元の関数が正常なプログラムかルートキットなのか判断できる．これにより，正規のプログラムの誤検知を防止でき，カーネルの拡張性を制限しない．

評価では，提案システムが実在するルートキットを検知できることを示した．提案システムは，システムコールを改ざんするルートキットを感染してから 0.25 ms で検知し，オリジナル関数を呼び出さないルートキットを感染してから 0.37 ms で検知した．また，提案システムの導入におけるオーバヘッドを測定した．監視するシステムコールのオーバヘッドの評価から，1 回のシステムコール当たり，0.01 μs から 2.36 μs のオーバヘッドであり，十分に小さいことを示した．

残された課題として，ベンチマークソフトによる実 AP への性能への影響の評価がある．

謝辞 本研究の一部は，平成 25 年度公益財団法人ウエスコ学術振興財団学術研究費助成による．

参考文献

[1] “国内で最も使われた標的型攻撃ツールは何だ!?— 2012 年上半期の標的型攻撃分析，” http://wp.techtarget.itmedia.co.jp/contents/?cid=12026

[2] “OS を超えたルートキット対策で「未知」の脅威を防ぐ，” http://wp.techtarget.itmedia.co.jp/contents/?cid=11745

[3] “SpyEye(スパイアイ) とは，”http://securityblog.jp/words/2776.html

[4] “Zeus(ゼウス) とは，”http://securityblog.jp/words/812.html

[5] Nick L. Petroni, Jr., Timothy Fraser, Jesus Molina, William A. Arbaughi, “Copilot - a Coprocessor-based Kernel Runtime Integrity Monitor, ”Proc. 13th USENIX Security Symposium, pp.179-194, 2004.

[6] 小倉寛之，大山恵弘，岩崎英哉，“カーネルレベルルートキットの検知システムの構築，” 情処学研報，Vol. 2008-OS-108，No.35，pp.51-58，2008.

[7] Junichi Murakami, “A Hypervisor IPS based on Hardware Assisted Virtualization Technology, ”Black Hat USA，2008.

[8] 秋月 康志，今泉 貴史，“ベアメタルハイパーバイザを用いたカーネルレベルルートキット検知システムの実現，” 情処学研報，Vol.2010-IOT-9，No.7，pp.1-6，2010.

[9] “McAfee Deep Defender，”http://www.mcafee.com/japan/products/deep-defender.asp

[10] Kruegel, C., Robertson, W., Vigna, G., “Detecting Kernel-Level Rootkits Through Binary Analysis, ”Proc. 20th Annual Computer Security Applications Conference（ACSAC’04），pp.91-100，2004.

[11] Yi-Min Wang, Doug Beck, Binh Vo, Roussi Roussev, and Chad Verbowski, “Detecting Stealth Software with Strider GhostBuster，”http://research.microsoft.com/pubs/70147/tr-2005-25.pdf

[12] Ryan Riley, Xuxian Jiang, Dongyan Xu, “Multi-Aspect Profiling of Kernel Rootkit Behavior，”Proc. the 4th ACM European conference on Computer systems (EuroSys’09)，pp.47-60，2009.

[13] “KBeast，” http://packetstormsecurity.com/files/108286/KBeast-Kernel-Beast-Linux-Rootkit-2012.html