Vol.2010-IOT-9 No.7
2010/5/13

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

秋 月 康 志[†1]　今 泉 貴 史[†2]

現状のマルウェア対策では，アンチウイルスソフトウェアを各ホストにインストールする方法が標準的となっている．しかし，ルートキット技術を利用して自身の存在を隠すステルスマルウェアが蔓延し始め，既存のアンチウイルスソフトウェアによる検知や対策は困難になってきている．

本論文では，仮想化技術を用いて OS の外部から OS を監視することで，マルウェアを隠蔽しているルートキットを検知・無効化する手法を提案する．この手法を用いることで，既存のアンチウイルスソフトウェアでもステルスマルウェアを検知できるようになる．

Implementing Kernel-level Rootkits Detection System by Bare-Metal Hypervisor

Yasushi Akizuki[†1] and Takashi Imaizumi [†2]

System administrators usually install anti-virus software to each host as countermeasures against malware. As the stealth malware that hides itself using rootkit technologies spread, an anti-virus software becomes hard to detect it.

In this paper, we proposes the technique that detects and nullifies the rootkits by monitoring OS behavior from the outside of OS using the virtualization technique. By using this technique, we can detect the stealth malware even with existing anti-virus software.

†1 千葉大学大学院 融合科学研究科
Graduate School of Advanced Integrated Science, Chiba University
†2 千葉大学 総合メディア基盤センター
Institute of Media and Information Technology, Chiba University

1. はじめに

近年のインターネットの急速な普及により，マルウェアの脅威が無視できないものとなっており，機密情報の漏洩やコンピュータの乗っ取りなど大きな問題となっている．2009 年の日本国内のマルウェア感染被害の総報告数は 45,310 件で，2008 年の 56,880 件から約 20 %減少した[8]．しかし，依然としてマルウェア感染被害は存在するため，被害を防ぐためにマルウェア対策を講じる必要がある．現在では，インターネットセキュリティスイート，特にアンチウイルスソフトウェアを各ホストにインストールし，常駐させてマルウェア対策を行うことが標準的となっている．

しかし，近年，マルウェアの中でも自身の存在を隠すステルスマルウェアが蔓延しつつある．ステルスマルウェアは，ルートキット技術を利用している．ルートキットとは，典型的にはコンピュータシステムへ侵入した痕跡や自身の存在を，システム管理者から隠蔽する機能を持つツールセットである[6]．元々は Unix においてそのような機能を持つツールセットを指していたが，現在では Unix 系 OS だけでなく，Microsoft Windows などの非 Unix 系の OS において同様の機能を持つツールセットもルートキットと呼ばれている．2003 年から 2006 年にかけて，ルートキット技術が利用されたソフトウェアは 6 倍以上になっており[3]，ルートキット技術の利用率は年々急速に高まっている．また，2000 年から 2005 年にかけて，ルートキットの複雑さは 4 倍以上になっており，さらに，2005 年の第 1 四半期と 2006 年の同時期を比べた場合，その複雑さは 9 倍以上になっているという報告もある[3]．ルートキット技術を利用しているマルウェアの数が増加している上に，巧妙化の一途を辿っているため，既存のアンチウイルスソフトウェアを用いた検知や対策が非常に困難になっている．

本論文では，既存のアンチウイルスソフトウェアでもステルスマルウェアを検知できるように，マルウェアを隠蔽しているルートキットを検知・無効化する手法を提案する．システムの内部で動作するルートキットにはユーザレベルルートキットとカーネルレベルルートキットがあるが，ユーザレベルルートキットは既存のアンチウイルスソフトウェアで容易に検知できる[4] ため，本論文では取り扱わない．

本ルートキット検知手法では，ルートキットと疑わしきソフトウェアを検知すると，まずホワイトリストと比較する．ソフトウェアの情報がホワイトリストと一致すれば，安全なソフトウェアとみなす．一方，ホワイトリストと一致しない場合は，ユーザに問い合わせを行い，安全なソフトウェアかどうかの判断をユーザに委ねる．最終的に，疑わしいソフトウェアをルートキットと判定すると，そのソフトウェアを停止させて無効化する．この処理によ

り，マルウェアはルートキット技術を利用していない状態になるため，既存のアンチウイルスソフトウェアで検知・対策が可能になる．

2. 関 連 研 究

2.1 小倉らによるカーネルレベルルートキットの検知システム

ルートキット検知システムとしては，小倉らによって，カーネル領域の書き換えを異常と定義し，カーネル領域を正常状態と比較する手法が提案されている[6]．この手法では，まずカーネル領域を固定長で区切る．そして，全てのシステムコールの先頭に比較処理を行うコードを挿入し，システムコールが呼び出される毎に区切ったメモリ領域を比較する．

システムの振る舞いを監視しているため，既知のルートキットだけでなく未知のルートキットにも対応できる．しかし，検知システム自体を書き換えた場合や，システムコールの先頭に挿入したコードを変更された場合は，完全に無効化されてしまう．無効化を防ぐために別スレッドで検知システムやシステムコールを監視しているが，スレッド自身の監視システムを解除することで無効化が実行できてしまう．

2.2 Viton

村上によって Viton が提案されている[5]．この手法では，Viton が OS の外部で動作し，ルートキットが書き換える特定の領域を保護・監視する．保護した領域の書き換えが試行されると異常とみなし，書き換えを阻止する．

小倉寛之らによるカーネルレベルルートキットの検知システム同様，システムの振る舞いを監視しているため未知のルートキットにも対応できる．加えて，OS の外部で動作しているため書き換えられることもない．しかし，OS の内部に処理用のコードを書き込むため，既存のアンチウイルスソフトウェアとの協調が図られていない．また，カーネル領域の全域を保護しておらず，コード領域と関数ポインタ郡といった一部データ領域のみを保護するため，データ領域の書き換えに対応していない．

3. 提 案 手 法

3.1 提案手法の概要

本論文では，ルートキットを検知・無効化する手法として **GUARD**（**G**eneric **U**nnoticeable **A**rchitecture for **R**ootkit **D**etection）を提案する．ルートキットを無効化することで，ステルスマルウェアを既存のアンチウイルスソフトウェアで検知できるようになる．

GUARD はハイパーバイザとして動作し，実際のマシンの OS をゲスト OS として仮想化する．その後，OS の外部からカーネル領域や重要なレジスタなどの監視を続け，不正な書き換えから保護する．

GUARD の動作の流れは，以下のようになる．また，動作の流れを**図 1** に示す．

(1) ルートキットがカーネル領域または重要なレジスタの書き換えを試行する
(2) システムの制御がゲスト OS 上から GUARD 上に遷移する
(3) 書き換え先のシンボル，実行ファイルの名前やデジタル署名を，あらかじめ登録してあるホワイトリストと比較する
(4) 一致すれば，ゲスト OS に制御を戻す
(5) 一致しなければ，ログを出力して，正常な実行ファイルからの書き換えかどうかをユーザに問い合わせる
(6) 問い合わせた結果，正常な実行ファイルの場合，ホワイトリストに登録し，ゲスト OS に制御を戻す
(7) 正常な実行ファイルでない場合，実行ファイルを終了させるように実行状態を変更し，ゲスト OS に制御を戻す

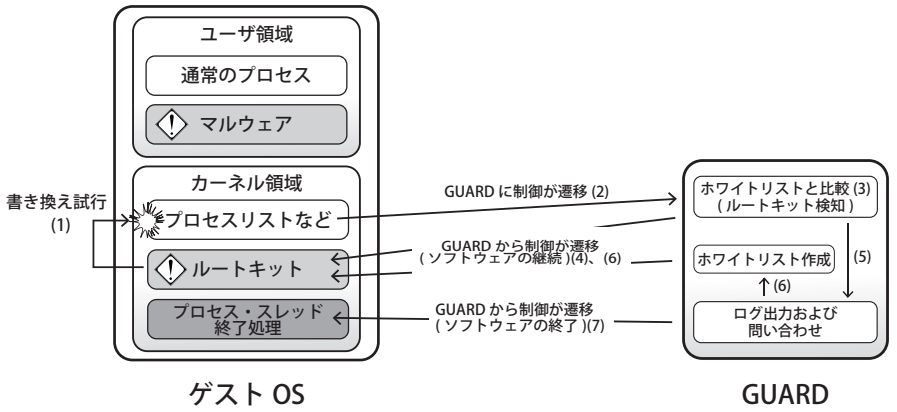

図 1 GUARD の動作の流れ
Fig. 1 The flow of GUARD

3.2 GUARD を導入したシステムの構成

GUARD を導入していないシステムは**図 2** のような構成となっており，カーネルからの入出力が直接ハードウェアへ届く．

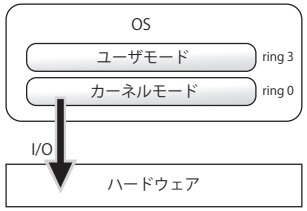

図 2 GUARD を導入していないシステムの構成
Fig. 2 The construction of the host system without GUARD

一方，GUARD をシステムに導入した場合，**図 3** のような構成になる．カーネル領域や重要なレジスタを保護する機能を実現するため，保護領域への入出力だけを監視・仮想化し，無関係な入出力は透過的にハードウェアに届く．必要なものだけを監視することで，オーバーヘッドを最小に保つ．

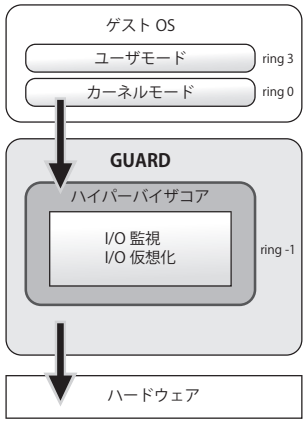

図 3 GUARD を導入したシステムの構成
Fig. 3 The construction of the host system with GUARD

3.3 機 能 構 成

GUARD の機能構成は大きく以下の 3 つに分けられる．

- ルートキット検知機能
- ログ出力および問い合わせ機能
- ホワイトリスト作成機能

以降，それぞれの機能について述べる．

ルートキット検知機能

ルートキットがマルウェアを隠蔽する際は，必ず以下のリソースを一つ以上書き換える．

- カーネル領域
- 特定のモデル固有レジスタ（SYSENTER_EIP_MSR）
- 特定のシステムレジスタ（IDTR）
- CR0 レジスタの読み書きフラグ

これらの領域に対して書き換えが試行されると異常とみなし，書き換えを試行したソフトウェアを疑わしきものとして検知する．次にホワイトリストと書き換えを試行したソフトウェアの情報を比較し，情報が一致しない場合はルートキットの可能性があるソフトウェアとして検知する．具体的には，カーネル領域全域を書き込み保護し，カーネル領域に対する書き込みをページフォルトとして検知する．また，重要なレジスタの場合は，書き込みが試行されると CPU でイベントが発生し，GUARD に制御が遷移するため，それを検知する．

ログ出力および問い合わせ機能

ルートキットの可能性があるソフトウェアを検知すると，まずログを補助記憶装置上にファイルとして出力する．このログには，以下のデータが含まれる．

- 書き換え先のシンボル
- 日時
- 実行ファイルのファイル名
- デジタル署名
- MD5 ハッシュ

また，悪意のないソフトウェアをルートキットとして検知して強制終了させる恐れがあるため，ユーザに安全なソフトウェアかどうか該当ログと共に問い合わせる．問い合わせた結果，安全なソフトウェアの場合は後述するホワイトリストに登録する．安全なソフトウェアでない場合はルートキットを書き換え，ルートキット自身を終了させるシステムコールを呼び出すようにする．最後に OS に制御を戻し，実際にルートキットを終了させる．

ホワイトリスト作成機能

アンチウイルスソフトウェアが行うシステム監視機能の提供や，OS 自身による管理機能などにより，カーネル領域の書き換えを行う場合がある．それらの悪意のないソフトウェアをルートキットとして検知して強制終了することを避けるため，ホワイトリストを作成し，補助記憶装置上に記録する．ホワイトリストには以下のデータを含む．

- 書き換え先のシンボル
- 実行ファイル名
- 実行ファイルのデジタル署名

このホワイトリストを元に，GUARD に状態が遷移した際に疑わしきソフトウェアか安全なソフトウェアかを判断する．

4. 評価実験

オーバーヘッドを計測するため，BitVisor[10] というハイパーバイザをベースにしてプロトタイプシステムを実装した．プロトタイプシステムではカーネル領域の初期化は許可し，二度目の書き込み試行をトラップする．オーバーヘッドの計測が目的のため，二度目の書き込み試行をトラップした後は単純に許可する．これにより，一度目または二度目の書き込みの際に必ずプロトタイプシステムを経由する仕組みが実現でき，実際に GUARD を導入した場合にページフォルトが発生する状況を再現できる．

プロトタイプシステムを用いてカーネルレベルのソフトウェア，ユーザレベルのソフトウェアの実行速度を計測した．また，プロトタイプシステムの物理メモリの使用量を計測した．実行環境を**表 1** に示す．ネットワークの影響を除くため，ネットワークには物理的に接続せずに実験を行った．

表 1 実行環境
Table 1 The test environment

OS	Fedora 12（kernel-PAE）
CPU	Intel Core Duo T2400（1.83GHz）
RAM	DDR2 PC2-5300 1GB × 2（Dual Channel）
チップセット	Intel 945GM

4.1 カーネルレベルのソフトウェア

カーネルレベルのソフトウェアの実験では，Fedora 12 の起動時間を計測した．Fedora 12 の起動時間の計測では，ブートローダで Fedora 12 を選択して起動させた瞬間を 0 秒とし，ユーザ名の一覧が出た時点で起動完了とみなした．プロトタイプシステムを利用せずに Fedora 12 を起動させた場合とプロトタイプシステムを利用して Fedora 12 を起動させた場合の起動時間をそれぞれ**表 2** に示す．

表 2 Fedora 12 の起動時間
Table 2 Fedora 12 startup times

試行回数	プロトタイプシステムを利用しない場合の起動時間（秒）	プロトタイプシステムを利用する場合の起動時間（秒）	増加率（%）
1 回目	36.01	74.08	206
2 回目	35.89	74.01	206
3 回目	35.95	74.33	207
4 回目	35.95	74.54	207
5 回目	36.36	74.27	204
平均	36.03	74.25	206

プロトタイプシステムを利用した場合，利用しない場合に比べて OS の起動時間は約 2 倍となった．

4.2 ユーザレベルのソフトウェア

ユーザレベルのソフトウェアの実験では，プロトタイプシステムを利用した場合とプロトタイプシステムを利用しない場合でそれぞれ Fedora 12 上で姫野ベンチ[7] を実行した．

姫野ベンチは，M サイズの C, static allocate version を利用した．プロトタイプシステムを実行していない状態の Fedora 12 でコンパイルし，それで得られた実行ファイルをそれぞれ実験に用いた．結果を**表 3** に示す．

プロトタイプシステムを利用した場合，利用しない場合に比べて，同じプログラムを実行した際に 1.5 倍時間がかかるという結果を得られた．

4.3 物理メモリの使用量

プロトタイプシステムの物理メモリの使用量の計測では，Fedora 12 を起動させてスーパーユーザとしてログインし，free コマンドで得られる物理メモリの認識量の変化を用いた．計測した結果，プロトタイプシステムを利用しない場合の物理メモリの認識量は 2,054,044KB となり，プロトタイプシステムを利用した場合の物理メモリの認識量は 1,920,860KB となった．つまり，プロトタイプシステムは 133,184KB，約 130MB 物理メモリを消費する．

表 3 姫野ベンチの実行結果
Table 3 The results of himeno benchmark

	プロトタイプシステムを利用しない場合の実行結果（MFLOPS）	プロトタイプシステムを利用する場合の実行結果（MFLOPS）
1 回目	792.349921	549.015371
2 回目	793.624253	544.815882
3 回目	790.449137	543.278939
4 回目	793.463121	543.449926
5 回目	791.741352	543.313730
平均	792.325557	544.774770

5. 考　　察

5.1 類似手法との比較

カーネルレベルルートキットの検知システム

小倉寛之らによるこの手法と同様，GUARD はシステムの振る舞いを監視しているため，既知のルートキットだけでなく未知のルートに対応できる．またこの手法には，書き換えられることによって検知不能に陥るという弱点があったが，GUARD では OS の外部で動作し，書き換えられないため検知回避を防ぐことができる．

Viton

Viton では既存のアンチウイルスソフトウェアとの協調がはかられていなかったが，GUARD では既存のアンチウイルスソフトウェアを利用することを前提としている．そのため，既存のアンチウイルスソフトウェアと共存できる．また，Viton はカーネル領域内のコード領域と一部のデータ領域の保護のみであったが，GUARD はカーネル領域全域を保護するためプロセスリストの改竄などに対応できる．

5.2 GUARD について

オーバーヘッドの評価

プロトタイプシステムを用いたオーバーヘッドの計測について第 4 節で述べた．OS の起動などのカーネルレベルのソフトウェアの場合は，主にカーネル領域にアクセスするためページフォルトが頻発して処理速度が低下したものと考えられる．ベンチマークソフトウェアなどのユーザレベルのソフトウェアの場合は，入出力処理やシステムコールを呼び出す場合以外にカーネル領域へのアクセスを行わないため，プロトタイプシステムによる影響があまり出なかったものと考えられる．カーネルレベル，ユーザレベル共に実行速度が低下したが，GUARD を導入するようなサーバマシンなどでは速度より安全性が重視されるため，この程度のオーバーヘッドは許容できると考える．

ベアメタルハイパーバイザについて

GUARD では，1 台のマシンで完結でき，なおかつ外部から OS を監視する機能を提供するという点でベアメタルハイパーバイザを利用している．そのため，ベアメタルハイパーバイザに限定せず，同様の機能を持つものであれば良い．

ハイパーバイザ検知の恐れ

仮想化を用いて解析されることを恐れ，ハイパーバイザを検知するマルウェアが存在する．ハイパーバイザを検知したマルウェアは，無害であるかのように振る舞いを変えたり，マルウェア自身を削除することがある．マルウェアを解析する場合では問題となるが，GUARD はマルウェアの被害を防ぐための手法であるので問題ない．

ハイパーバイザの衝突

GUARD を導入するシステムが既に仮想化されている場合を考える．多重仮想化を防ぐため，またはハイパーバイザを悪意のあるハイパーバイザに交換されるなどのセキュリティ上の問題を防ぐため，ハイパーバイザは仮想マシンに対して仮想化関連の命令を提供しない．そのため，GUARD を導入するシステムが既に仮想化されている場合は衝突が発生して導入できない．また，GUARD も仮想マシンに対して仮想化関連の命令を提供しないため，新たにハイパーバイザを導入する場合も同様に衝突が発生する．そのため，悪意のあるハイパーバイザのインストールを防ぐことができる．

検 知 ミ ス

アンチウイルスソフトウェアが行うシステム監視機能の提供や，OS 自身による管理機能などにより，カーネル領域または重要なレジスタの書き換えを行う場合がある．GUARD はカーネル領域や重要なレジスタの書き換えを異常とみなすため，それらを疑わしきソフトウェアとして検知してしまう，つまりフォールスポジティブは発生する．システムで検知したのち，最終的にルートキットと判定してしまうことへの対策としてユーザへの問い合わせがある．仮に，ユーザが安全なソフトウェアをルートキットと判定してしまったとしても，OS そのものでない限りドライバが動作しない程度で済む．

また，カーネル領域や重要なレジスタ，つまり GUARD が保護する領域を書き換えないルートキットは存在しないため，ホワイトリストに登録されていない限りフォールスネガティブは発生しない．そのため通常は安全性が保たれているが，もしユーザが誤ってルートキットを安全なソフトウェアと判定してしまうとホワイトリストに登録される．すると，そ

のルートキットに関して常にフォールスネガティブが発生して検知不能に陥り，ルートキットおよびルートキット技術で隠蔽されたマルウェアの動作に気付かないという事態となる．

検知不能なルートキット

GUARD を利用することでカーネルレベルのルートキットは検知できるが，ハイパーバイザルートキット，ファームウェアルートキットおよびユーザレベルのルートキットは検知できない[1)2)9)]．ハイパーバイザルートキットが起動している場合は，前述したハイパーバイザの衝突により検知以前に GUARD を導入できない．また，ファームウェアルートキットは完全にシステムの外で動作しているため GUARD では検知できない．ユーザレベルのルートキットは容易に検知できる[4)]ため，既存のアンチウイルスソフトウェアでも検知可能であると仮定し，本論文では取り扱っていない．

GUARD の移植性

GUARD やホワイトリストを別のマシンに移植する場合を考える．GUARD は仮想化支援技術に対応しているマシンであればマシンは問わない．しかし，OS ごとにカーネル領域の大きさ，システムコールやファイルシステムが違ってくるので OS ごとに仕様を合わせて GUARD を作る必要がある．また，同じ OS 間の場合，ホワイトリストは容易に移植できる．しかし，ハードウェア構成が違うとインストールされているドライバも違ってくるため，ほとんど問い合わせが来ない状態になったホワイトリストを移植しても問い合わせが発生する可能性がある．同一の種類の OS でバージョンが異なる場合，シンボルの変換を行う仕組みを実現すれば自動で変換でき，移植も可能である．

6. おわりに

6.1 まとめ

本論文では，既存の手法で検知が困難なルートキットの検知・無効化を行う手法，GUARD を提案した．また，オーバーヘッド評価を行うプロトタイプシステムを作成し，評価を行った．

GUARD を利用することでルートキットの無効化を行い，既存のアンチウイルスソフトウェアでもステルスマルウェアを検知できることを期待できる．

6.2 今後の課題

- システムの実装
  プロトタイプシステムはオーバーヘッド評価のための簡素な実装のみであった．設計通りにシステムを実装し，アンチウイルスソフトウェアを導入したマシン上で実際に競合が発生しないか試験を行う．また，正しくルートキットが無効化され，アンチウイルスソフトウェアで検知可能かどうか試験を行う．
- OS やドライバのバージョンアップへの対応
  OS やドライバをバージョンアップすると，補助記憶装置上の実行ファイルの情報が変更され，実行ファイルに関連するホワイトリストを更新する必要がある．既にホワイトリストに登録された実行ファイルを監視し，実行ファイルの更新を検知して次回起動時に自動でホワイトリストを更新することで対応できると考えられる．実行ファイルの更新を行うプロセスが正しいプロセスかどうかを確認する処理は，既存のアンチウイルスソフトウェアに任せても問題ないと考えている．
- 問い合わせ機能の廃止
  問い合わせ自体がユーザにとって煩わしい上，間違った回答を行う可能性があるため，問い合わせを行わないことが望ましい．対策として，問い合わせに頼ることなくホワイトリストのみでルートキットかどうかの判定を行う必要がある．そのホワイトリストの作成方法について検討していきたい．

参考文献

1) Chen, K.: Reversing and exploiting an Apple firmware update, *Black Hat USA 2009*, Black Hat (2009).
2) Embleton, S. and Sparks, S.: SMM Rootkits, Technical report, Clear Hat Consulting, Inc. (2008).
3) McAfee, Inc.: rootkit 第 1 回 拡大する脅威 (2006).
4) McAfee, Inc.: ルートキット (第 2 回):技術情報 (2007).
5) Murakami, J.: A Hypervisor IPS based on Hardware assisted Virtualization Technology, *Black Hat USA 2008*, Black Hat USA (2008).
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，情報処理学会研究報告， Vol.2008, No.35(OS-108), pp.51–58 (2008).
7) 理化学研究所情報基盤センター：姫野ベンチマーク. http://accc.riken.jp/HPC/HimenoBMT.html.
8) Trend Micro Incorporated: インターネット脅威年間レポート - 2009 年度〜2009 年 1 月 1 日〜12 月 31 日データ最終版〜 (2010). http://jp.trendmicro.com/jp/threat/security_news/monthlyreport/article/20100107014909.html.
9) Ukai, Y., Jack, B. and Kanai, R.: -Ring ”-1” Rootkit- システムの「外」で動作する新たな Rootkit や BOT の脅威と対策，技術報告，eEye Digital Security (2006).
10) University of Tsukuba: BitVisor 概要 - BitVisor - セキュア VM プロジェクト (2008). http://www.securevm.org/bitvisor-abs.html.