情報処理学会論文誌 Vol.54 No.12 2402–2412 (Dec. 2013)

軽量ハイパバイザによるシステムイメージの完全性保護

忠鉢 洋輔[1,a)] 表 祐志[1] 品川 高廣[2] 加藤 和彦[1]

受付日 2013年3月14日, 採録日 2013年9月13日

概要：近年，オペレーティングシステム（OS）のカーネル権限を不正に取得する攻撃が増加している．攻撃者による永続的な不正アクセスを防ぐためにシステムイメージを OS の外部から保護する研究が行われているが，保護のための Trusted Computing Base（TCB）が大きくなりがちなこと，正確な保護が行えないことが問題となっている．本研究では，OS から透過的にバイト粒度での保護を実現するハイパバイザの設計と，このハイパバイザに最適化された完全性保護のための情報を生成する手法を示す．さらに，一般的なクライアント OS である Windows XP と FAT32 ファイルシステムを対象に，BitVisor をベースした実装と評価を行った．この結果，全体で 31KLOC と小さい TCB のハイパバイザで比較的低オーバヘッドの完全性の保護が実現できることを確認した．

キーワード：オペレーティングシステム，仮想化技術，セキュリティ

Protecting System Image Integrity with Lightweight Hypervisors

Yosuke Chubachi[1,a)] Yushi Omote[1] Takahiro Shinagawa[2] Kazuhiko Kato[1]

Received: March 14, 2013, Accepted: September 13, 2013

Abstract: In recent years, sophisticated attacks to gain unauthorized access to kernel privileges are prevalent. Protecting the system image integrity of operating systems (OSs) from outside of OSs is an effective approach to preventing attackers from persistently gaining unauthorized access. However, protecting system files from outside of OSs is not easy due to the problem of semantic gap between files and storage, making the overhead and the size of trusted computing base (TCB) larger. This paper presents a protection scheme using a lightweight hypervisor for protecting system image integrity. This scheme achieves strict file-level protection by using byte-granularity storage location information based on the specifications of file systems and achieves lightweight protection by using a tiny hypervisor that only passively performs byte-granularity inspection. We have built a prototype implementation supporting Windows XP on the FAT filesystem and confirmed that the size of the TCB is 31KLOC and the hypervisor incurs only small overhead.

Keywords: operating system, hypervisor, security

1. はじめに

近年，オペレーティングシステム（OS）のカーネル権限を不正に取得する攻撃が増加している [1]．攻撃者は，OS の脆弱性を突いたり不正にデジタル署名したりするなどの手法により，不正アクセスするプログラムをカーネル権限で動作させることができる．また，カーネルルートキットと呼ばれる不正アクセスを支援するプログラムを用いることで，セキュリティ対策ソフトウェアから隠蔽した状態でカーネル権限での不正アクセスを継続的に行うことができる．近年の複雑化した OS から脆弱性をなくしたり，証明書の不正利用を完全に防止したりすることは困難であるため，仮に攻撃者にカーネル権限を取得されたとしても，その被害を最小限に抑えられる仕組みが必要である．

攻撃者は，OS が再起動されても不正アクセスを継続するためには，カーネルルートキットをハードディスクなど

[1] 筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻
Department of Computer Science, Graduate School of Systems and Information Engineering, University of Tsukuba, Tsukuba, Ibaraki 305–8573, Japan
[2] 東京大学情報基盤センター
Information Technology Center, The University of Tokyo, Bunkyo, Tokyo 113–8658, Japan
[a)] bachi@osss.cs.tsukuba.ac.jp

の永続的なストレージにインストールする必要がある．本論文では，これをカーネルルートキットの永続化と呼ぶ．特に永続化がカーネルやデバイスドライバなどOSの基幹部分のシステムイメージ改ざんにより行われた場合，ファイルが隠蔽されたりセキュリティ対策ソフトウェアが無効にされたりするため，検出や除去が非常に困難になる．したがって，カーネルルートキットの永続化による被害を抑えるためには，OSの基幹部分のシステムイメージの完全性を保護することが必要である．

OSに脆弱性が存在しても完全性を保つ手法としては，仮想マシンモニタなどを用いてOSの外部から監視する手法がある [2], [3], [4]．これらの手法をシステムイメージの完全性保護に応用する場合は，ファイルとストレージとの間のセマンティックギャップ [5] を解決する必要がある．Penningtonらの研究 [6] では，仮想マシンモニタでファイル単位のアクセス情報を取得するためにNFSを用いる．しかし，この手法では既存のローカルディスク上で動作するOSに対して保護を適用することは難しい．ストレージデバイスに保護を組み込む研究 [7], [8] では，データの保護がセクタ単位であるため，アクセス日時の更新などファイルシステムのメタデータを適切に保護することが難しい [9], [10]．Zhangらの研究 [11] では，仮想マシンモニタで部分的にファイルシステムを解釈することにより，ファイル単位での侵入検知を実現している．しかしファイルシステム全体を正確に解釈はしていないため，OSのファイルシステムを経由せずに直接ディスクのイメージを書き換えた場合には検知できない可能性がある．また，Type IIの仮想マシンモニタを用いているため，TCB（Trusted Computing Base）のサイズや仮想化のオーバヘッドが大きくなる．

本論文では，軽量ハイパバイザを用いてシステムイメージの完全性を保護する手法を提案する．既存のローカルディスク上で動作するOSに適用できるようにするために，準パススルー型 [12] のハイパバイザを用いて，ストレージデバイスへのアクセスのみをOSから透過的に捕捉して保護を行う．また，メタデータへのアクセスを適切に保護するために，ストレージに書き込まれるセクタの中身のバイナリデータも取得し，セクタ単位よりも細かいバイト粒度での保護を実現する．さらに，システムイメージの完全性を正確に保護するために，ファイルシステムの仕様に基づいて，ファイルのデータとメタデータの位置をバイト単位ですべて正確に特定し，不正な書き込みを確実に防止する．TCBのサイズやオーバヘッドを抑えるために，ハイパバイザは能動的にはストレージにアクセスせず，OSからの書き込みを受動的に検査するだけでシステムイメージの完全性保護を実現する．

本論文では，OSから透過的にバイト粒度での保護を実現するための軽量ハイパバイザの設計およびファイルシステムを解釈して正確な完全性保護を実現するための情報を生成する手法を示す．また，BitVisor [12] をベースとした軽量ハイパバイザの実装や，FAT32ファイルシステム [13] において完全性保護のための情報を生成する手法を示す．また，FAT32で動作するWindowsXPを対象にして，実際にシステムイメージの完全性が保たれていることを示す．

本論文の構成を次に示す．まず2章では，本研究で想定する脅威モデルについて述べる．3章では，正確な完全性保護を実現するための情報を生成する手法とバイト粒度での保護を実現する軽量ハイパバイザの設計を示す．4章では，FAT32ファイルシステムを対象とした実装と，BitVisorを用いた軽量ハイパバイザの実装について詳しく説明する．5章では実装したハイパバイザのTCBや完全性保護のための情報のサイズの分析，ハイパバイザのオーバヘッドの測定結果を示す．6章で関連研究について述べ，7章で本論文をまとめる．

2. 脅威モデル

本論文では，OSが再起動されても不正アクセスを継続できるようにするために，攻撃者がハードディスクなどの永続的なストレージを改ざんして，OS起動時にカーネルルートキットが自動的にメモリに読み込まれて実行されるようにすることをカーネルルートキットの永続化と定義する．カーネルルートキットが永続化されていない場合は，OSを再起動することで容易に除去できるが，永続化された場合は検出や除去が困難になり，場合によっては再インストールなどが必要となる．

カーネルルートキットを永続化するためには，ストレージに実行イメージを格納し，さらにOS起動時に自動的に読み込まれて実行されるプログラムやライブラリなどのファイルの一部を改ざんして，その実行イメージが自動的に実行されるように設定する必要がある．たとえば，Rustock [14] と呼ばれるカーネルルートキットは，Windowsのデバイスドライバであるbeep.sysを改ざんして，起動時に必ずRustockが実行されるようにする．

本論文では，カーネルルートキットを永続化する攻撃のうち，OSの基幹部分のシステムファイルに対応するシステムイメージを改ざんするものを想定する．OSの基幹部分のシステムファイルとは，カーネルやデバイスドライバ，システムプログラムやライブラリ，管理プログラム，セキュリティ対策ソフトウェアなど，OSの完全性を保つうえで最低限必要なファイルである．また，ブート関係のデータ（MBRやパーティション情報）なども含まれることとする．基幹部分以外のファイルに対する改ざんは，OS上で動作するセキュリティ対策ソフトウェアなどで検出することを想定しており，本論文では対象としない．また，永続化を行わずメモリに常駐するだけのカーネルルートキットに関しては，他の手法と組み合わせて検出すること

を想定しており，本論文では対象としない．

本論文では，ハイパバイザ自身は信頼できることを仮定している．ハイパバイザ自身の正当性は，TPM [15] や Intel TXT [16] などにより検証可能であり，メモリ上のハイパバイザに対する攻撃は，HyperSafe [17] の手法を用いることで軽減可能である．また，BIOS などのファームウェアも信頼できることを仮定している．なお，本論文ではソフトウェアによる攻撃のみを想定しており，物理的にディスクを改ざんするなどの攻撃は対象としない．

本論文では，主に企業や教育機関などのように，専門の管理者がクライアント PC を管理している環境を想定している．したがって，提案システムを導入する時点でのシステムイメージの完全性は保たれていると仮定しているほか，一般ユーザが OS のアップデートやデフラグなどの管理作業は行わないことを想定している．

3. システムイメージの完全性保護

本章では，まず提案システムの概要を示す．次に，システムイメージの完全性を保護するために必要な情報である完全性保護リストについて述べ，最後に完全性保護を実現するためのハイパバイザの設計について述べる．

3.1 システム概要

提案システムでは，**図 1** に示すように管理モードと保護モードの 2 つのモードがある．管理モードでは，(1) 完全性保護リストの生成と，(2) メンテナンス作業を行う．(1) 完全性保護リストの生成では，プロテクションマネージャと呼ぶプログラムにより，保護するシステムファイルのリストから，対応するシステムイメージ領域を表す情報である完全性保護リストを生成する．生成した完全性保護リストはストレージ上に保存し，保護モードにおいてハイパバイザが起動時に読み込む．

(2) メンテナンス作業では，OS のアップデートやデフラグメンテーションなど，システムイメージを変更する可能性がある作業を実施する．メンテナンス作業は管理モードでのみ実施し，保護モードでは実施されないように OS を正しく設定する．管理モードで起動する際には管理用パスワードの入力を要求し，一般ユーザは使用できないようにして安全性を担保する．そのため，管理モードではハイパバイザは使用しない．

一般ユーザが利用するときは，保護モードに切り替えて起動する．保護モードでは，保護の対象となる OS をハイパバイザ上で実行する．ハイパバイザは，カーネルルートキットの永続化を防止するために，完全性保護リストに基づいてストレージ上のシステムイメージが格納されている領域への書き込みを禁止する．また，完全性保護リスト自身が格納された領域への書き込みも禁止する．いったん保護モードで起動した後に再度管理モードに切り替える際は，正しいシステムイメージから OS が起動するようにするために，必ず再起動を要求するようにする．

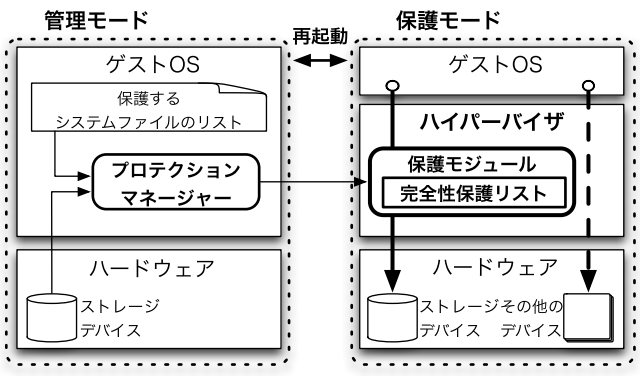


図 1 システム概要
Fig. 1 Design overview.

3.2 完全性保護リスト

完全性保護リストとは，保護の対象となるシステムファイルをストレージ上の領域にマッピングした情報である．提案システムでは，ハイパバイザの TCB サイズを抑えるために，ハイパバイザではファイルシステムを解釈せず，ハイパバイザで容易に監視できる低水準 I/O，すなわちセクタ単位での読み書きを監視するだけで完全性保護を実現する，このため，あらかじめ管理モードでファイルシステムを解釈して完全性保護リストを生成しておき，ハイパバイザでは低水準 I/O と完全性保護リストを照らし合わせるだけで保護を実現できるようにする．**図 2** では，「保護するシステムファイルのリスト」に対して，対応するストレージの領域（「保護対象領域」と示された斜線部分）を表す完全性保護リストを生成している様子を示している．完全性保護リストの生成にあたっては，保護するシステムファイルに対応するストレージ上の領域を特定する必要

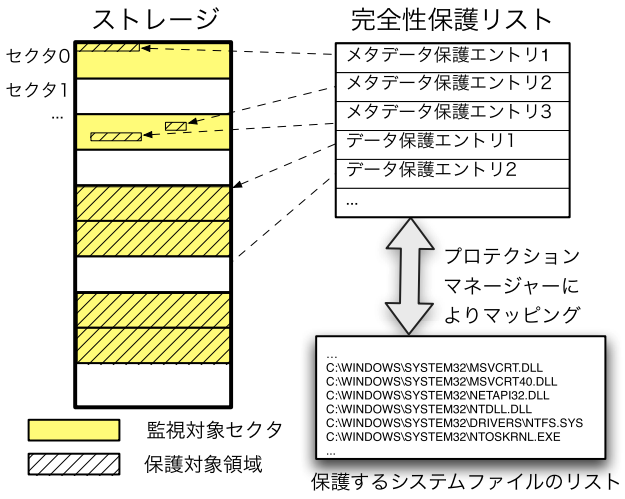


図 2 完全性保護リストの作成
Fig. 2 Generating a integrity protection list.

がある．多くのファイルシステムでは，ファイルはデータ本体とメタデータから構成され，このうちデータ本体にはブロックと呼ばれる単位で領域が割り当てられる．ブロックは複数の連続するセクタで構成されており，たとえばブロックサイズが4,096バイトでセクタサイズが512バイトの場合，1ブロックは連続する8セクタで構成される．このとき，異なるファイルのデータが1つのブロック内に格納されることは通常はないため，データ本体については対応するブロックをすべて書き込み禁止とすれば完全性を保護することができる．ブロックはセクタを単位とした領域で表現することができるため，ハイパバイザでセクタレベルの低水準I/Oを監視することで書き込み禁止を実現することができる．図2では，「完全性保護リスト」の後半部分において，書き込み禁止にするセクタ領域を「データ保護エントリ」として表現している．データ保護エントリは，「セクタ番号」と「セクタ数」の組で構成されている．

一方，メタデータに関しては，1つのセクタ内に書き込み禁止にすべき領域と書き込み禁止にすべきでない領域が混在することがある．たとえば，あるファイルのデータ本体の場所に関する情報とアクセス日時に関する情報が同じセクタに格納されている場合，データ本体の場所に関する情報を保護するためにセクタ全体を書き込み禁止にすると，アクセス日時が更新できなくなってしまう．また，ディレクトリ内に存在するファイルの情報を格納しているディレクトリエントリについても，1つのセクタ内に複数のファイルに関する情報が格納されている可能性があるため，セクタ全体を書き込み禁止にすると，書き込み保護が必要ないファイルに対しても，当該ディレクトリでの書き込みや移動などができなくなってしまう可能性がある．

そこで提案システムでは，セクタ内において書き込み禁止する領域と書き込み許可する領域が混在している場合には，セクタ内において保護すべき領域をバイト単位で表した情報を完全性保護リストに持たせる．これにより，セクタ内においてバイト粒度でのアクセス制御を行って，同一セクタ内でも部分的な領域のみを保護できるようにする．図2では，「完全性保護リスト」の前半部分において「メタデータ保護エントリ」として表現している．なお，ハイパバイザにおいてバイト粒度でのアクセス保護を実現する手法については，次節で説明する．また，ファイルとは別にブートローダやブートセクタなど起動に必要なセクタ領域に関する情報も完全性保護リストに追加しておく．

3.3 ハイパバイザ

提案システムでは，既存のOSのシステム構成を変更せずにシステムファイルの保護を実現するために，準パススルー型のハイパバイザを用いて書き込み保護を実現する．準パススルー型とは，ゲストOSからハードウェアへのアクセスを可能な限りパススルーとしてOSがハードウェアを直接制御できるようにしつつ，必要最小限のI/Oのみをハイパバイザで捕捉することで必要な機能を実現するアーキテクチャである．提案システムでは，ストレージ以外へのI/Oデバイスはすべてパススルーとするほか，ストレージに関しても基本的には仮想化を行わず，もともとOSがインストールされているディスクイメージをそのまま利用できるようにする．また，ハイパバイザで行う処理を少なくしてオーバヘッドを低減させるほか，ハイパバイザ自身のサイズを小さくすることでTCBを削減し，システムの安全性を高めることができる．

提案システムでは，保護が必要なのはストレージへの書き込みに対してのみであるため，ストレージからの読み込みについてはアクセス制御も行わず，自由にアクセスできるようにする．一方，ストレージへの書き込みに対しては，セクタ単位でのアクセスを行う低水準I/Oのレベルで監視を行い，完全性保護リストに基づいて，保護されている領域への書き込みを検出する．3.1節で述べたように，保護モードにおいてはOSのアップデートやでフラグメンテーションなどシステムファイルを変更する動作は行われないことを想定しているため，保護されている領域への書き込みを行おうとした場合には，カーネルルートキットが永続化を試みているなど，すでにカーネル権限が不正に取得されていると考えられる．したがって，それ以上OSの動作を継続することは被害の拡大につながるため，ハイパバイザは該当する書き込みを行わず，ただちにOSの動作を停止し，管理者への問合せを行うようなメッセージを表示する．

ハイパバイザによる書き込み監視は，すでに述べたように完全性保護リストに基づいて行う．完全性保護リストのうち，セクタ単位での保護を行う領域については，I/Oの内容を監視してセクタ番号とセクタ数を取得することで，容易に書き込み禁止の領域かどうか判定することができる．一方，バイト粒度での保護を行う領域については，書き込み自体はセクタ単位で行われるため，書き込まれようとしているセクタの内容を調べて，保護されている領域のバイト列が書き換わっていないかどうか調べる必要がある．このとき，元の正しいバイト列を調べるために，元のセクタをディスクからそのつど読み込むようにすると，ゲストOSからのI/Oを一時的に停止させる必要があるほか，ハイパバイザから別途ストレージへのI/Oを行う必要が生じるため，デバイスのアクセスやI/Oのスケジューリングなどでハイパバイザの処理が複雑になる．

そこで，提案システムでは，ハイパバイザでディスクのセクタを読み込む代わりに，あらかじめ完全性保護リスト内に元の正しいバイト列のバイナリデータを埋め込んでおく．これにより，セクタ内の保護すべき領域の内容を完全性保護リスト内のバイナリと照合するだけで，保護すべき領域が書き換わっていないかどうか判断できるようになり，

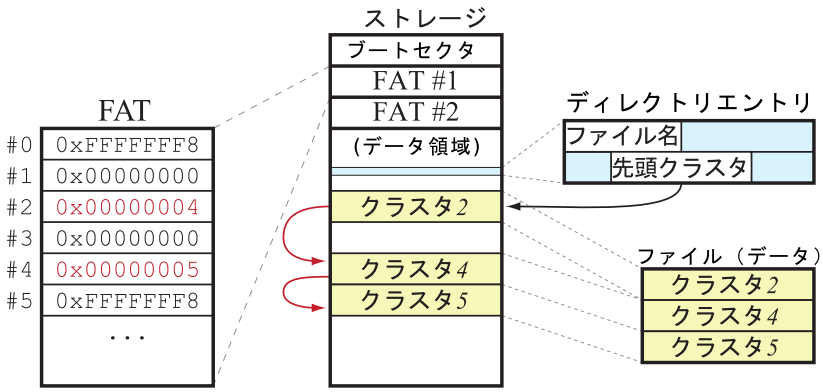


図 3 FAT32 ファイルシステムの概要
Fig. 3 The structure of the FAT32 file system.

即座にかつ容易にアクセスの可否を判断することができる．このように，提案手法はバイト列の比較のみで判断できるため，ディスクをファイルシステムの仕様に沿って「解釈」するファイルシステムのような機構を必要としない．ファイルシステムはバグを比較的多く作り込んでしまうことが知られているが [18]，提案システムではそのような複雑なサブシステムを必要としないため，ハイパバイザにバグや不具合を作り込んでしまう可能性が低くなる．このことから，ハイパバイザの安全性も向上すると考えられる．

4. 実装

提案システムの手法に基づき，完全性保護リストを生成するプロテクションマネージャと，ハイパバイザによる保護機構の実装を行った．完全性保護リストを生成する対象は，FAT32 ファイルシステム [13] 上で動作する Windows XP とした．また，ハイパバイザによる保護機構の実装は，BitVisor [12] をベースとして使用し，完全性保護リストに基づく保護を行う保護モジュールを組み込むことで行った．本章では，まず FAT32 の概要と完全性保護リストの生成について説明し，次に BitVisor の概要と保護モジュールの実装を述べる．

4.1 FAT32 ファイルシステムの概要

図 3 に FAT32 ファイルシステムの概要を示す．FAT32 では，ストレージの記憶領域をクラスタという 512 バイト～32 K バイトの固定長の領域に分割して管理する．ファイルのデータ本体の格納場所はクラスタの連鎖を用いて表現されており，この連鎖の情報は File Allocation Table（FAT）というストレージ先頭付近に確保された 1 エントリ 4 バイトのテーブル領域に記録される．メタデータもデータ本体と同様にクラスタに保存されており，ディレクトリエントリと呼ばれる 1 エントリ 32 バイトの配列によって管理されている．ディレクトリエントリにはファイル名（ディレクトリ名），ファイルタイプやデータの先頭クラスタ番号を示すフィールドがある．ファイルタイプがディレクトリのときは，データの先頭クラスタ番号はディレクトリエントリが保存されたクラスタを示している．

図 3 を用いて，File Allocation Table を用いたクラスタの連鎖の表現について説明する．図 3 では，ファイルのデータ本体はクラスタ番号 2，4，5 の 3 つのクラスタに格納されている．このデータ本体にアクセスする際は，まずディレクトリエントリに格納されているデータの先頭クラスタ番号を読み取る．例では先頭クラスタ番号は 2 であるため，データ本体の先頭部分はクラスタ 2 に格納されていることが分かる．次に，クラスタ 2 から連鎖しているクラスタの番号を取得するために，File Allocation Table の 2 番目のエントリを参照する．図 3 では，2 番目のエントリの値は 0x0000004 となっており，クラスタ 4 に連鎖していることが分かる．同様に，4 番目のエントリの値を参照することで，クラスタ 4 がクラスタ 5 に連鎖していることが分かる．5 番目のエントリは値が 0xFFFFFFF8 となっており，現在のクラスタが最終のクラスタであることを示している．このように，File Allocation Table を参照することでファイルのデータ本体が格納されているクラスタの場所を知ることができる．

4.2 完全性保護リスト生成

3.1 節で述べたように，完全性保護リストは管理モードにおいてファイルシステムを解釈して生成する．この完全性保護リストを生成するプログラムをプロテクションマネージャと呼ぶ．今回実装したプロテクションマネージャは Windows 上で動作するユーザレベルのプログラムである．なお，本論文では FAT32 ファイルシステムを対象としたが，ファイルシステムの仕様が分かれば他のファイルシステムに対しても同様の手法を適用することは可能である．

完全性保護リストの生成にあたっては，3.2 節で述べたように，保護の対象となるファイルのデータ本体とメタデータに分けて考える．FAT32 のデータ本体は，前節で述べたようにディレクトリエントリに格納されている先頭クラスタ番号から File Allocation Table を参照してクラスタ

の連鎖をたどることで，すべてのクラスタ番号を取得することができる．クラスタは 3.2 節で述べたブロックに相当し，そのサイズ（512 バイト～32 K バイト）はパーティションの先頭セクタであるブートセクタ内の BIOS Parameter Block（BPB）に格納されている．また，1 つのクラスタがセクタ境界をまたぐことはないため，クラスタの連鎖の情報から「セクタ番号」と「セクタ数」の組を生成することが可能である．たとえば図 3 では，クラスタ 2 とクラスタ 4，5 に対応するセクタ番号とセクタ数の組を生成すればよい．

一方，FAT32 のメタデータで書き込み禁止にするべき領域は，保護するファイルに対応する 32 バイトのディレクトリエントリのうちアクセス日付を除いた部分と，データ本体が格納されたクラスタの連鎖を示す File Allocation Table の各 4 バイトのエントリ群となる．これらはセクタサイズよりも小さいため，バイト粒度での保護が必要になる．そこで，これらのエントリが格納されたセクタ番号を取得したうえで，セクタ内での当該エントリのオフセット，サイズおよび正しいバイナリ列を読み取って，完全性保護リストに格納する．図 3 では，まず「ディレクトリエントリ」が格納されたセクタ番号およびセクタ内のオフセットとサイズ（32 バイト），さらに実際の値を表すバイナリ列の情報を取得し，オフセット 18 バイトからの 2 バイトに格納されているアクセス日付の部分を除いた領域に関する情報を作成する．また，FAT のエントリ#2 の 4 バイトおよびエントリ#4，#5 の 8 バイトの各領域に関しても，セクタ番号，セクタ内のオフセット，サイズ，正しいバイナリ列の情報を作成する．なお，ロングファイルネームに対応するための VFAT 領域の保護に関する説明は省略するが，バイト粒度の保護を行うことで同様に対応できる．

4.3 BitVisor の概要

本論文では実装のベースとなるハイパバイザに BitVisor を使用した．BitVisor は，準パススルー型アーキテクチャのハイパバイザで，基本的にはデバイスを仮想化せずゲスト OS から I/O デバイスへのアクセスをパススルーする一方，必要最小限の I/O は捕捉・変換することでセキュリティなどの付加機能を実現する方式である．準パススルー型アーキテクチャは，デバイスをパススルーするため複数のゲスト OS は同時には動作しないが，仮想化によるオーバヘッドを大幅に削減できるほか，ハイパバイザに必要な機能を大幅に削減して TCB のサイズを小さくできるといった利点がある．そのため，今回のように OS に依存しないセキュリティを実現する目的には適している．

本論文の実装では，保護を行うストレージとして，ATA デバイスを対象とした．BitVisor を用いて ATA デバイスへの ATA コマンドの発行を捕捉し，保護モジュールによりコマンドの内容をチェックして，発行の可否を判断する．ATA コマンドを捕捉することにより，仮にゲスト OS のカーネル権限を乗っ取られていたとしても，ストレージへのアクセスを確実に捕捉することが可能になる．また，ATA デバイス以外の I/O デバイスへのアクセスは監視する必要がないため，すべてパススルーに設定する．

ATA デバイスへのアクセスは，主に Logical Block Address（LBA），Sector Count，Command の 3 種類の内容をレジスタに書き込むことで行われる．LBA はストレージの先頭から各セクタに振られた番号（本論文の「セクタ番号」に相当する）であり，読み書きするセクタの場所を指定する．Sector Count は読み書きするセクタ数を指定し，Command で読み書きなどアクセスの内容を指定する．セクタ・データの転送は，Programmable I/O（PIO）と Direct MemoryAccess（DMA）の 2 種類の方法がある．PIO ではレジスタへの連続した I/O アクセスで，DMA では専用ハードウェアでデータ転送を行う．

BitVisor は，ATA デバイスのレジスタへの読み書きを監視できるほか，PIO によるデータ転送をバッファリングして内容を確認する機能を持っている．また，DMA によるデータ転送も捕捉して，内容を確認してから実際のデバイスへの書き込みを行うシャドウ DMA ディスクリプタという機能を持っている [12]．

4.4 保護モジュールによるストレージの保護

4.2 節の手法で生成した完全性保護リストに基づいて，ATA コマンドの発行の可否を判断する保護モジュールを実装して BitVisor に組み込んだ．保護モジュールでは，Command レジスタの内容を取得して，読み込みアクセスか書き込みアクセスかを判断する．提案手法では，書き込みアクセス以外は保護を行う必要がないため，書き込みアクセスでない Command はすべて許可する．一方，書き込みアクセスを行う Command であった場合には，システムイメージの改ざんを防止するために，完全性保護リストで保護対象としている領域を書き換える内容かどうかを判断する．対象領域を書き換える内容であった場合は，その書き込みアクセスをブロックして，実際には ATA デバイスには書き込まれないようにするとともに，強制的に OS の再起動を行う．

4.2 節で述べたように，完全性保護リストで保護すべき対象はファイルのデータ本体とメタデータがあり，データ本体はセクタ単位，メタデータはバイト単位での保護が必要となる．セクタ単位での保護については，保護モジュールで ATA コマンドの LBA と Sector Count の内容を取得し，完全性保護リストの「セクタ番号」と「セクタ数」の組で表される領域と重なる場合には，その書き込みをブロックする．一方，バイト単位での保護については，保護モジュールで取得した LBA と Sector Count で指定される領域に一致するセクタ番号が完全性保護リストにある場合

は，PIO や DMA で書き込もうとしているセクタ・データの内容を BitVisor の機能を利用して取得し，完全性保護リストに格納されたオフセットおよびサイズで示された場所のバイナリ列を，完全性保護リスト内に格納されている正しいバイナリ列と比較して，保護すべき内容が変更される場合は書き込みをブロックする．

このように，完全性保護リストをセクタ番号とセクタ数の組，およびセクタ番号，オフセット，サイズ，正しいバイナリ列の組で表現することにより，保護モジュールではゲスト OS から発行される ATA コマンドの内容を受動的に監視するだけで，システムイメージの保護を実現することが可能になる．これにより，ハイパバイザの TCB のサイズやオーバヘッドが削減される．

5. 評価

本章では，提案手法を実際の OS に適用して評価した結果を示す．OS は Windows XP SP3 を対象とし，ハイパバイザは BitVisor 0.7 をベースに提案手法を実装したものを用いた．ハードウェアは，CPU が Intel Core2 Duo E6850（3.0 GHz），メモリが 4 GB，SSD が MTRON MSP-SATA7035 で構成された PC を用いた．

5.1 システムイメージの保護

提案手法によるシステムイメージの保護が有効に動作することを評価するため，システムファイルへの改ざんを行うプログラムを作成し，保護モードで実行中の提案システム上で動作させた結果を分析した．システムファイルへの改ざんを行うプログラムは，Kernel Rootkit の一種である Rustock [14] の永続化を参考にした．Rustock は永続化の過程で，OS システムイメージの 1 つである C:\WINDOWS\system32\Drivers\beep.sys を悪意のあるドライバで上書きする．この動作を模倣したプログラムを作成し，beep.sys の改ざんを試みた．

まず，管理モードでプロテクションマネージャを起動し，C:\WINDOWS\system32\Drivers\beep.sys（以下 beep.sys）を直接指定して完全性保護リストを生成した．このときの完全性保護リストのうち，データ本体についてのエントリは，ファイルの先頭セクタは 110135 セクタ，セクタ数 12 となっていた．beep.sys のファイルのハッシュ値（SHA-1）を File Checksum Integrity Verifier で取得した結果，ハッシュ値は e3d2dc5eb273fa701de8af13b60d6baac7629260 であった．この完全性保護リストをハイパバイザに含め保護モードで起動した．そして，beep.sys を先頭から 4096 バイト分，0 で上書きするプログラムを管理者権限で実行した．この結果，110135 セクタへの書き込みを検知し，ハイパバイザが OS を停止させ，管理者に報告するように促すメッセージが表示された．その後，管理モードで OS を再度起動したのちに取得した beep.sys のハッシュ値は e3d2dc5eb273fa701de8af13b60d6baac7629260 であった．したがって，提案システムによってシステムファイルが保護されることが確認された．

5.2 TCB サイズの評価

一般に TCB のサイズは安全性を保つために小さい方が良いとされている．そこで提案システムの安全性を評価するために，TCB を構成するハイパバイザのサイズを測定した．測定には sloccount [19] を用い，ソースコードのコメントや空白行などを除いた有効な行数をカウントした．**表 1** に測定結果を示す．「ハイパバイザ・コア」はハイパバイザとして最低限動作するために必要な部分のコード，「ATA ドライバ」は 4.3 節で説明した ATA デバイスへのアクセスを監視するコード，「保護モジュール」は ATA ドライバと連携してバイト粒度の保護を実現するコードである．測定結果から，全体でも 3 万 1 千行程度に抑えられているほか，バイト粒度での完全性保護を実現するために必要な ATA ドライバと保護モジュールは合計でも 1,867 行で実現できることが分かった．これは，Xen [20] をベースに用いた関連研究 [11] で TCB が 98KLC（Xen Hypervisor）+ 56MLOC（Domain0）以上になることと比較して [21]，非常に小さなコードで保護を実現できていると考えられる．

5.3 完全性保護リストの分析

完全性保護リストが現実的であることを示すために，Windows XP SP3 においてシステムファイル属性が設定されている 2,350 個のファイルを保護する完全性保護リストを生成して，監視する総セクタ数およびエントリ数を測定した．**表 2** に測定結果を示す．データ本体が格納されたセクタ数は 876,365 個であったが，セクタ番号とセクタ数の組として表されるエントリの数は 2,357 個であった．これは，多くのファイルが連続したセクタに格納されているためである．メタデータが格納されたセクタは合計で 695

表 1 TCB のサイズ
Table 1 TCB size.

モジュール名	行数
ハイパバイザ・コア	29,163
ATA ドライバ	1,790
保護モジュール	77
合計	31,030

表 2 Windows XP の完全性保護リストの分析
Table 2 The analysis of the integrity-protection list.

	セクタ数	エントリ数
データ本体	876,365	2,357
ディレクトリエントリ	284	2,350
File Allocation Table	411	2,129
合計	877,060	6,836

セクタであり，エントリ数の総数は 4,479 個であった．エントリ数全体でも 6,836 個であり，ハイパバイザ内のメモリに格納するうえでも問題のないサイズであるほか，保護モジュールにより検査を行う際の比較作業もさほど大きくないデータ量であることが分かった．

5.4 オーバヘッドの測定

5.4.1 ディスクベンチマーク

ハイパバイザによる保護のオーバヘッドを確認するために，ファイルに対する読み書きのスループットを測定した．評価項目は，ファイルを先頭から順番にアクセスする Sequential Read/Write，ファイルのランダム領域に 512 KB ごとのアクセスを行う Random Read/Write 512 K，ファイルのランダム領域に 64 K ごとにアクセスする Random Read/Write 64 K の 6 種類である．各評価項目について 5 回ずつ評価を行い，この転送速度の平均値を測定値とした．ベンチマークでは上限 100 MB のファイルを，完全性保護リストの対象外の領域に新規作成する．ただし，ファイルの書き込み先は，保護の対象となるストレージデバイス上のディレクトリを設定している．このため，ハイパバイザによる保護を実行している場合，書き込み動作は完全性保護リストのすべてのエントリと照合されたのちに行われる．また，測定は常駐ソフトをすべて停止した状態で行った．

測定は 3 つの実行環境で行った．「Windows」はハードウェア上で OS を直接実行した結果，「Hypervisor」はハイパバイザ・コアのみでストレージ保護を行わない BitVisor 上で OS を実行した結果，「Protected」は提案システムの保護モードで OS を実行したときの結果を表す．なお，「Protected」では 5.3 節の完全性保護リストを用いた．測定結果を図 4 に示す．

提案システム（「Protected」）は Sequential Write において，保護を行わないハイパバイザ（「Hypervisor」）と比較して 25%ほどのオーバヘッドがあった．一方，Random Write 64 K においては，提案システムは保護を行わないハイパバイザと比べてオーバヘッドはほとんど見られなかった．また，読み込みアクセスにおいても，提案方式によるオーバヘッドはほとんど見られなかった．

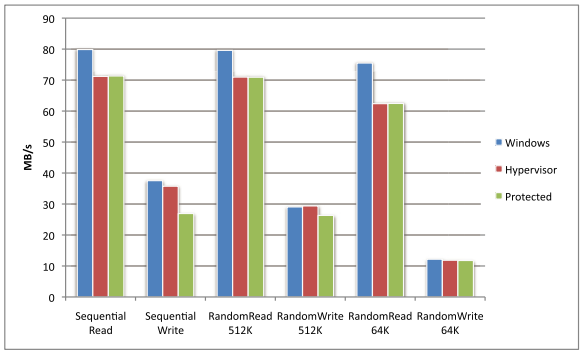


図 4 ディスクの性能評価
Fig. 4 Disk performance.

提案システムにおける Sequential Write で比較的オーバヘッドが大きい要因は，ベンチマークの特性と完全性保護リストを検索する CPU 計算オーバヘッドにあると考えられる．ベンチマークによる Sequential Write では DMA 転送命令が連続的に発行されるが，提案手法により 1 命令ごとに CPU 計算オーバヘッドが発生するため，DMA 転送命令の開始が順次遅れていくことを確認している．ハードウェア上で直接 Windows を動作させた場合と比較すると，この転送開始の遅れが積算されていくことが原因で相対的に時間あたりの転送量が減じていると考えられる．現在の保護モジュールの実装では線形探索の計算コストが支配的であり，この実装を改善することで Sequential Write におけるオーバヘッドを軽減できると考えられる．

また，Random Write 64 K，Random Write 4 K ではほとんどオーバヘッドが発生していない．これは，ベンチマークによる Random Write では OS によるファイルのシークが書き込み命令の間に行われるため，DMA 転送命令が遅れることによるオーバヘッドが少ないと考えられる．このように，一般的な OS の利用時に発生するワークロードであるランダムアクセスにおいては，提案手法によるオーバヘッドがおさえられていることを確認した．

5.4.2 OS 起動時間の測定

ハイパバイザによる保護が OS に与える影響を確認するため，OS の起動からユーザが利用可能になるまでの経過時間を測定する実験を行った．測定は，CPU のタイムスタンプカウンタ（TSC）の値を取得するプログラムを作成し，このプログラムを OS のスタートアップメニューに登録することで行った．TSC は，OS が初期化したタイミングを 0 とし，その後クロックごとにカウントアップされる CPU の機能である．Windows XP においては，起動ロゴ出現とほぼ同時に初期化されるため，本実験では，ユーザが体感する OS の起動時間として用いている．

測定項目は前項と同じく 3 つの実行環境であり，「Windows」はハードウェア上で OS を直接実行した結果，「Hypervisor」はハイパバイザ・コアのみでストレージ保護を行わない BitVisor 上で OS を実行した結果，そして「Protected」は提案システムの保護モードで OS を実行したときの結果を表す．各測定項目において 5 回起動を行い，そのつど TSC の値を取得し，その平均値を平均起動時間とした．結果を図 4 に示す．

表 3 Windows XP の平均起動時間
Table 3 Windows XP boot time.

測定対象	平均起動時間（秒）
Windows	11.4
Hypervisor	17.0
Protected	17.2

提案システム（「Protected」）と保護を行わないハイパバイザ（「Hypervisor」）を比較すると，保護によるオーバヘッドはほとんどないといえる．しかし，ベースとしたハイパバイザのオーバヘッドが比較的大きい．また，提案システムと保護を行わないハイパバイザでは，OS の起動前に平均 3 秒程度 BitVisor の起動時間を必要とする．このため，Windows を直接実行した場合と提案システムを比較すると，ユーザが体感する起動時間は実質的に 2 倍程度となる．ただし，ハイパバイザのオーバヘッドおよび起動時間は，最新の BitVisor を用いることで改善されることが予想される．

6. 関連研究

6.1 ストレージ保護

ファイルサーバにおいて，OS から独立した形で保護や侵入検知を行う研究が多数行われている．Self-Securing Storage [22] は，ログ構造のストレージを用いて，ファイルシステムに対する変更を一定期間記録したり元に戻したりできる．また，Pennington らの研究 [6] では，ファイルシステムに対するアクセスの挙動を監視することで，ルートキットなどを用いた侵入を検知できる．しかしこれらの研究では，ファイルレベルでの操作を監視するために，OS とファイルサーバのインタフェースが NFS である必要があり，OS 独自のファイルシステムで管理されているローカルディスクに適用することは難しい．

Rootkit-resistant Disks（RRD）[8] は，ネットワーク・ブロックデバイスを用いて，OS のシステムファイルが格納された領域をブロック単位で書き込み禁止とし，セキュリティトークンをデバイスに挿入したときだけ更新できるようにする．しかし RRD は保護の単位がブロックであり，メタデータを正確に保護するために必要なバイト粒度での保護はしていない．Zhang らの研究 [11] では，ハイパバイザを用いてセクタ単位の操作からファイル単位の操作を再構成し，あらかじめ記述したルールと照合して侵入検知を行う．しかし，ファイルシステムを厳密には解釈していないため，メタデータの直接改ざんや既存ファイルへの追記など，OS のファイルシステムを経由しない書き込みを検知できない可能性がある [9], [10]．また，侵入検知を行う際にハイパバイザから能動的にストレージにアクセスする必要があり，ハイパバイザの構造が複雑になる．

提案システムでは，ファイルシステムの仕様に基づいて，システムファイルの格納場所をバイト粒度で正確に特定するため，攻撃者がファイルシステムを経由せずにストレージを直接改ざんしようとした場合でも，システムファイルを確実に保護することができる．また，ストレージへのアクセスを受動的に検査するだけで保護を実現できるため，ハイパバイザがシンプルになり TCB も小さくできる．

6.2 ハイパバイザによる OS 保護・検証

SecVisor [4] は，ハイパバイザによりメモリ上の OS カーネルの完全性保護を行う．また，完全性保護に特化した軽量ハイパバイザを用いることで，TCB のサイズを非常に小さくできる．しかし，SecVisor はメモリ保護を対象としており，ストレージ保護は行っていない．

ハイパバイザを用いて，OS 起動時にカーネルを検証する手法も数多く提案されている [23], [24], [25]．しかし，これらの手法ではシステムイメージの一部しか保護の対象としていないほか，改ざん自体は防止できないことから，改ざん検知後には OS の再インストールなどが必要となる．

6.3 OS 自身による保護

OS 自身の機構によってシステムイメージの改ざんを検知する手法が提案されている [26], [27]．Windows2000 以降の Windows では，Windows FileProtection（WFP）と呼ばれる保護機構が搭載されており，システムファイルの破損や改ざんを起動時にチェックおよび修復することが可能である．しかし，カーネルルートキットはカーネル権限を取得するため，OS による保護機構を無効にすることが可能である．実際に，WFP を回避する手法も広く知られている [28]．本研究では，ハイパバイザにより OS の外部から保護を行うことで，カーネル権限を持つ不正なプログラムからの攻撃に対しても，システムイメージの完全性を保護することが可能である．

7. まとめ

本論文では，軽量ハイパバイザによりシステムイメージの完全性を保護する手法を提案した．既存のローカルディスク上で動作する OS に適用できるようにするために，準パススルー型アーキテクチャを用いてストレージデバイスへのアクセスのみを捕捉して保護する手法を実現した．また，TCB のサイズや仮想化のオーバヘッドを削減しつつもメタデータの完全性を確実に保護するために，ファイルシステムをあらかじめ解釈して保護すべきバイナリ列を含む完全性保護リストを生成することにより，ハイパバイザではセクタ単位での低水準 I/O を受動的に監視するだけでバイト粒度での保護を実現できる仕組みを提案した．また，実際に FAT32 ファイルシステムと ATA デバイスを対象とした実装を行い，Windows XP SP3 のシステムファイルを保護する完全性保護リストを生成した．評価の結果，ハイパバイザのサイズは全体でも 3 万 1 千行程度であること，完全性保護リストのエントリ数は 7 千個弱であること，一般的な OS の利用時に発生するワークロードではハイパバイザによる保護のオーバヘッドもそれほど大きくないことを確認した．

今後の課題として，ハイパバイザをさらに軽量にするための改良があげられる．現在の実装では，保護の対象と

なっていないセクタへのアクセス時にもすべての完全性保護リストを走査しているため，ハイパバイザの実装と完全性保護リストのデータ構造を改善することで，オーバヘッドをさらに減らすことができると考えられる．また，現在の実装では，BitVisor にもともと存在するコードの多くをそのまま流用しているため，提案手法には必要のない機能も一部含まれている．これらの機能を削減することで，より軽量なハイパバイザを実現できると考えられる．

また，ジャーナル機能を備えたファイルシステムへの適用についても，今後検討する必要がある．現在の実装では，保護モードで防止したシステムファイルへの書き込みが，管理モードで再起動した際にジャーナル機能により再度書き込まれてしまう可能性がある．したがって，保護モードから管理モードに再起動した際にはジャーナル機能を無効にする仕組みを導入することなどが考えられる．

謝辞 本研究の一部は総務省・戦略的情報通信研究開発推進制度（SCOPE）の支援により行われたものである．

参考文献

[1] McAfee: Signed Malware: You Can Run, But You Can't Hide (2012), available from ⟨http://blogs.mcafee.com/mcafee-labs/signed-malware-you-can-runbut-you-cant-hide⟩.

[2] Garfinkel, T. and Rosenblum, M.: A Virtual Machine Introspection Based Architecture for Intrusion Detection, *NDSS*, The Internet Society (2003).

[3] Wang, Z., Jiang, X., Cui, W. and Ning, P.: Countering kernel rootkits with lightweight hook protection, *CCS '09: Proc. 16th ACM Conference on Computer and Communications Security*, pp.545–554, ACM (online), DOI: http://doi.acm.org/10.1145/1653662.1653728 (2009).

[4] Seshadri, A., Luk, M., Qu, N. and Perrig, A.: SecVisor: a tiny hypervisor to provide lifetime kernel code integrity for commodity OSes, *SOSP '07: Proc. 21st ACM SIGOPS Symposium on Operating Systems Principles*, pp.335–350, ACM (online), DOI: http://doi.acm.org/10.1145/1294261.1294294 (2007).

[5] Jiang, X., Wang, X. and Xu, D.: Stealthy malware detection through vmm-based "out-of-the-box" semantic view reconstruction, *CCS '07: Proc. 14th ACM Conference on Computer and Communications Security*, pp.128–138, ACM (online), DOI: http://doi.acm.org/10.1145/1315245.1315262 (2007).

[6] Pennington, A.G., Strunk, J.D., Griffin, J.L., Soules, C.A.N., Goodson, G.R. and Ganger, G.R.: Storage-based intrusion detection: Watching storage activity for suspicious behavior, *SSYM'03: Proc. 12th Conference on USENIX Security Symposium*, p.10, USENIX Association (2003).

[7] Sivathanu, M., Prabhakaran, V., Popovici, F.I., Denehy, T.E., Arpaci-Dusseau, A.C. and Arpaci-Dusseau, R.H.: Semantically-Smart Disk Systems, *Proc. 2nd USENIX Conference on File and Storage Technologies*, pp.73–88, USENIX Association (online) (2003), available from ⟨http://portal.acm.org/citation.cfm?id=1090694.1090702⟩.

[8] Butler, K.R.B., Mclaughlin, S. and Mcdaniel, P.D.: Rootkit-resistant disks, *ACM Conference on Computer and Communications Security*, Ning, P., Syverson, P.F., Jha, S., Ning, P., Syverson, P.F. and Jha, S. (Eds.), pp.403–416, ACM (online), DOI: 10.1145/1455770.1455821 (2008).

[9] Griffin, J.L., Pennington, A., Bucy, J.S., Choundappan, D., Muralidharan, N. and Ganger, G.R.: On the Feasibility of Intrusion Detection inside Workstation Disks (2003).

[10] Grugg, T.: Defeating forensic analysis on Unix, *Phrack Magazine*, Vol.11, No.59, p.6 (2002).

[11] Zhang, Y., Gu, Y., Wang, H. and Wang, D.: Virtual-Machine-based Intrusion Detection on File-aware Block Level Storage, *Symposium on Computer Architecture and High Performance Computing*, pp.185–192 (online), DOI: http://doi.ieeecomputersociety.org/10.1109/SBAC-PAD.2006.32 (2006).

[12] Shinagawa, T., Eiraku, H., Tanimoto, K., Omote, K., Hasegawa, S., Horie, T., Hirano, M., Kourai, K., Oyama, Y., Kawai, E., Kono, K., Chiba, S., Shinjo, Y. and Kato, K.: BitVisor: A thin hypervisor for enforcing i/o device security, *Proc. 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, *VEE '09*, pp.121–130, ACM (online), DOI: 10.1145/1508293.1508311 (2009).

[13] Microsoft Corporation: *Microsoft Extensible Firmware Initiative FAT32 File System Specification* (2000).

[14] Chandra, P.: Rootkit Installation and Obfuscation in Rustock. available from ⟨http://www.sunbeltsecurity.com/dl/RootkitInstallationandObfuscationinRustock.pdf⟩.

[15] Trusted Computing Group: *TPM main specification, version 1.2 rev. 103* (2007), available from ⟨https://www.trustedcomputinggroup.org⟩.

[16] Intel Corporation: *Intel trusted execution technology software development guide* (2008).

[17] Wang, Z. and Jiang, X.: HyperSafe: A Lightweight Approach to Provide Lifetime Hypervisor Control-Flow Integrity, *IEEE Symposium on Security and Privacy*, pp.380–395, IEEE Computer Society (2010).

[18] Chen, H., Mao, Y., Wang, X., Zhou, D., Zeldovich, N. and Kaashoek, M.F.: Linux kernel vulnerabilities: state-of-the-art defenses and open problems, *Proc. 2nd Asia-Pacific Workshop on Systems*, *APSys '11*, pp.5:1–5:5, ACM (online), DOI: 10.1145/2103799.2103805 (2011).

[19] Wheeler, D.A.: *SLOCCount*, available from ⟨http://www.dwheeler.com/sloccount/⟩.

[20] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A.: Xen and the art of virtualization, *SOSP '03: Proc. 19th ACM Symposium on Operating Systems Principles*, pp.164–177, ACM (online), DOI: http://doi.acm.org/10.1145/945445.945462 (2003).

[21] Murray, D.G., Milos, G. and Hand, S.: Improving Xen security through disaggregation, *VEE '08: Proc. 4th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pp.151–160, ACM (online), DOI: http://doi.acm.org/10.1145/1346256.1346278 (2008).

[22] Strunk, J.D., Goodson, G.R., Scheinholtz, M.L., Soules, C.A.N. and Ganger, G.R.: Self-securing storage: protecting data in compromised system, *OSDI'00: Proc. 4th Conference on Symposium on Operating System Design & Implementation*, p.12, USENIX Association (2000).

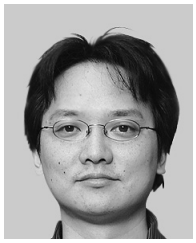
[23] Azab, A.M., Ning, P., Sezer, E.C. and Zhang, X.: HIMA: A Hypervisor-Based Integrity Measurement Agent, *ACSAC '09: Proc. 2009 Annual Computer Security Applications Conference*, pp.461–470, IEEE Computer Society (online), DOI: http://dx.doi.org/10.1109/ACSAC.2009.50 (2009).
[24] McCune, J.M., Li, Y., Qu, N., Zhou, Z., Datta, A., Gligor, V.D. and Perrig, A.: TrustVisor: Efficient TCB Reduction and Attestation, *Proc. IEEE Symposium on Security and Privacy* (Oakland 2010) (online) (2010), available from ⟨http://www.truststc.org/pubs/750.html⟩.
[25] Garfinkel, T., Pfaff, B., Chow, J., Rosenblum, M. and Boneh, D.: Terra: A virtual machine-based platform for trusted computing, *SOSP '03: Proc. 19th ACM Symposium on Operating Systems Principles*, pp.193–206, ACM (online), DOI: http://doi.acm.org/10.1145/945445.945464 (2003).
[26] Kim, G.H. and Spafford, E.H.: The design and implementation of tripwire: A file system integrity checker, *CCS '94: Proc. 2nd ACM Conference on Computer and Communications Security*, pp.18–29, ACM (online), DOI: http://doi.acm.org/10.1145/191177.191183 (1994).
[27] Patil, S., Kashyap, A., Sivathanu, G. and Zadok, E.: I3FS: An In-Kernel Integrity Checker and Intrusion Detection File System, *LISA '04: Proc. 18th USENIX Conference on System Administration*, pp.67–78, USENIX Association (2004).
[28] Collake, J.: Windows File Protection - How to replace a system DLL, available from ⟨http://www.bitsum.com/aboutwfp.asp⟩.

忠鉢 洋輔 （学生会員）

1986 年生まれ．筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻博士後期課程所属．システムソフトウェア，情報セキュリティに興味を持つ．人工知能学会，ACM 各会員．

表 祐志 （学生会員）

1987 年生まれ．筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻博士後期課程所属．オペレーティングシステムや仮想マシンモニタ等のシステムソフトウェアに興味を持つ．2011 年度コンピュータサイエンス領域奨励賞受賞．



品川 高廣 （正会員）

1974 年生まれ．2003 年東京大学大学院理学系研究科情報科学専攻博士課程修了，博士（理学）取得．同年東京農工大学工学部情報コミュニケーション工学科助手，2007 年筑波大学大学院システム情報工学研究科講師，2011 年より東京大学情報基盤センター情報メディア教育研究部門准教授．オペレーティングシステムや仮想マシンモニタ等のシステムソフトウェアに興味を持つ．平成 11 年度情報処理学会論文賞，平成 14 年度山下記念研究賞受賞．

加藤 和彦 （正会員）

1962 年生まれ．1985 年筑波大学第三学群情報学類卒業．1989 年東京大学大学院理学系研究科情報科学専攻中退．1992 年博士（理学）（東京大学大学院理学系研究科）．1989 年東京大学理学部情報科学科助手，1993 年筑波大学電子・情報工学系講師，1996 年同助教授，2004 年筑波大学大学院システム情報工学研究科教授，現在に至る．オペレーティングシステム，分散システム，仮想計算環境，セキュリティに興味を持つ．1990 年情報処理学会学術奨励賞，1992 年同研究賞，2005 年同論文賞，2004 年日本ソフトウェア科学会論文賞各受賞．