Vol.2010-DPS-142 No.35
Vol.2010-CSEC-48 No.35
2010/3/5

準パススルー型VMMのマルウェア検出機能による拡張

Tran Truong Duc Giang[†1]　大山恵弘[†1]　忠鉢洋輔[†2]
品川高廣[†2]　加藤和彦[†2]

仮想マシンモニタ（VMM）を用いたセキュリティ向上は，最近数年の間に非常に良く研究された効果的なアプローチである．BitVisor はストレージデータの暗号化や VPN の構築を含む多様なセキュリティ機能を提供する VMM である．BitVisor は準パススルー型アーキテクチャを用いて構築されている．そのアーキテクチャでは，OS からの大半の I/O アクセスは VMM を通過し，セキュリティ機能を実装するための最小限のアクセスだけが VMM によって捕捉される．そのアーキテクチャは小さいオーバヘッドと Trusted Computing Base（TCB）をもたらす．現在の BitVisor はプライバシーの保護はできるが，マルウェアの検出はできない．そこで本研究では，マルウェア検出機能を準パススルー型 VMM に組み込むための方式を提案する．我々はその方式に基づいて BitVisor の拡張を実装した．その拡張は，データ I/O の中身を，VMM に保存されたマルウェアのシグネチャと比較する．我々は予備実験を行い，その拡張の実行時間オーバヘッドが極めて小さいことを確認した．

Extension of Parapass-through VMM for Malware Detection

Tran Truong Duc Giang,[†1] Yoshihiro Oyama,[†1]
Yosuke Chubachi,[†2] Takahiro Shinagawa[†2]
and Kazuhiko Kato[†2]

Security enhancement using a virtual machine monitor (VMM) is an effective approach studied deeply for the last decade. BitVisor is a VMM that provides various security functionality including encryption of storage data and creation of virtual private networks. BitVisor is implemented using a parapass-through architecture, in which most of the I/O accesses from the operating system are passed through the VMM, while the minimum accesses necessary to implement security functionality are mediated by the VMM. The architecture brings a small overhead and trusted computing base (TCB). Although the current BitVisor does good work for privacy protection, it lacks functionality for malware detection. In this paper, we propose a scheme for incorporating malware detection functionality into a parapass-through VMM. According to the scheme, we implemented an extension of BitVisor for malware detection. The extension compares the contents of data I/O with malware signatures stored in the VMM. We confirmed through preliminary experiments that the runtime overheads imposed by the extension was extremely small.

1. はじめに

コンピュータの利用におけるマルウェアによる被害が多発している．ユーザは，不注意や攻撃者の巧妙な誘導によって，メールに添付されていたり，USB メモリに含まれていたりするマルウェアを実行してしまうことがある．また，ユーザによる操作を経由せず，プログラムの脆弱性を突いてコンピュータに侵入するマルウェアも存在する．そのようなマルウェアの攻撃は，ユーザが十分に注意していても防げない場合がある．マルウェアを検出し被害を未然に防ぐには，アンチウィルスの利用が効果的である．実際，アンチウィルスは非常に広く普及している．

しかし，アンチウィルスの運用面に関してはいくつかの問題がある．第一の問題は，処理が重いなどの理由から，ユーザがアンチウィルスなしで OS を利用することがある点である．そもそも初めからインストールしない場合もあれば，インストールされているアンチウィルスをアンインストールすることもある．特に，処理能力の低いネットブックと呼ばれるコンピュータの近年における普及は，アンチウィルスなしに OS を利用する習慣を拡大させる可能性がある．第二の問題は，マルウェアが管理者ユーザや OS カーネルの権限を乗っ取った場合には，アンチウィルスを無力化できる点である．まず，アンチウィルスのアンインストールが可能となる．また，アンチウィルスが稼働し続けているように偽装しつつ，実際には検査処理を実行させないといった攻撃も可能となる．これは潜在的な問題ではあるが，実際にそのようなケースが発生したときの被害は甚大であるため，対策は検討しておく必要がある．

上記の問題を解決するための一つの有効な方法は，OS よりも下の層で動作するソフトウェ

†1 電気通信大学大学院電気通信学研究科情報工学専攻
†2 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

Vol.2010-DPS-142 No.35
Vol.2010-CSEC-48 No.35
2010/3/5

アである仮想マシンモニタ（VMM）においてセキュリティ処理を行うことである．VMM によるセキュリティ向上について，これまでに非常に多くの研究が行われている[4)–7),9)–12),16)]．これらの研究が提案するシステムでは，OS のメモリ上にあるデータや，OS とハードウェアとの相互作用を VMM が監視し，OS の動作を制御する．たとえば，プログラムの異常な挙動の検知，ネットワークを流れる攻撃データの検知，データの暗号化などを行う．セキュリティ処理は OS 上ではなく，VMM の中で実行される．よって，OS のユーザと VMM の管理者を別の人にするという運用を行えば，その OS からは，セキュリティ処理を無効にすることができない．加えて，その OS のユーザやカーネルの権限を乗っ取った攻撃者もセキュリティ処理を無効にすることができない．

上記の運用は，会社などの組織においては十分に現実的である．組織の各メンバーのコンピュータにインストールされた VMM を組織の管理者が一元的に管理するとする．そして，各メンバーはその VMM 上で動作する自分の OS を管理する権限だけを与えられるとする．セキュリティに明るくないメンバーが OS を危険な状態にしたとしても，VMM によってセキュリティが保たれる．

BitVisor[12)] は，上記の運用形態の実現に適している仮想マシンモニタの一つである．BitVisor はストレージデータの暗号化や，Virtual Private Network（VPN）の構築などを VMM 層で行う．BitVisor 上で OS を動作させることにより，OS 上で特別な処理を行うことなく，それらのセキュリティ処理を利用できるようになる．BitVisor は準パススルー型と呼ばれる方式に基づいて構築されている．この方式の採用により，ディスクの暗号化などのセキュリティ機能を実現しつつも，実行時間オーバヘッドをある程度小さく抑えている．さらに，実装が小さくなり，その結果 Trusted Computing Base（TCB）が小さく抑えられるという利点もある．ただし，残念ながら，現在の BitVisor はプライバシーを保護することはできるが，マルウェアを検出することはできない．

本研究では，準パススルー型仮想マシンモニタ BitVisor をマルウェア検出機能で拡張したシステムの構築を目的とする．具体的には，ストレージやネットワークのデータと，マルウェアを特徴づける文字列との間のマッチング処理を BitVisor に組み込み，VMM 層でアンチマルウェア処理を実現する．VMM 層での実現により，上述した無効化への耐性という利点に加え，特定の OS に依存しなくなるという利点も得られる．Windows，Linux，FreeBSD を含む幅広い OS を本システムで保護できるようになる．本システムの利用にあたり，VMM 上で動作する OS のカーネルを改変する必要はなく，OS 上で特別なプログラムを動作させる必要もない．

BitVisor はハードウェア上で直接動作し，BitVisor 上では，ユーザが利用する OS のみが動作する．そのため，OS の上で動作する VMM[8),14),15)] や，管理用の仮想マシンを必ず動作させる VMM[2)] に対してとは異なり，BitVisor にマルウェア検出機能を組み込む方法は自明ではなく，工夫を要する．

本研究の貢献を以下に要約する．

- 準パススルー型 VMM の中でマルウェア検出機能を実現するための一つの設計を示した．
- その設計に基づいてシステムを実装し，そのシステムが OS 上に生成されたマルウェアのバイト列を実際に検出したことを確認した．
- 標準的なベンチマークを用い，そのシステムのオーバヘッドを計測する予備実験を行った．

本論文は以下のように構成される．2 章では BitVisor の概要を述べる．3 章では提案システムの概要を説明し，4 章では実装の詳細を説明する．5 章では提案システムによるマルウェアの検出についての議論を提示する．6 章では予備実験の結果を示す．7 章では関連研究と本研究との比較について述べる．8 章ではまとめと今後の課題を述べる．

2. BitVisor

BitVisor は，筑波大学などによって開発された，ハードウェア上で直接動作するセキュリティ向上のための準パススルー型 VMM である．準パススルー型とは，OS からハードウェアに可能な限り透過的にアクセスさせ，セキュリティ機能の実現のために最低限必要なアクセスのみを VMM が捕捉する方式である．BitVisor は，一部のアクセスのみを捕捉することにより，ストレージやネットワークの暗号化などのセキュリティ機能を実現している．BitVisor は Intel VT-x や AMD Virtualization（AMD-V）等の CPU の仮想化支援機構を用いて実装されている．

BitVisor の構成を**図 1** に示す．図中の実線矢印は BitVisor に捕捉される I/O 処理を表し，点線矢印は捕捉されない I/O 処理を表している．BitVisor は準パススルードライバと呼ばれるデバイスドライバによってデバイスを制御する．準パススルードライバは OS が発行する制御 I/O とデータ I/O を捕捉する．制御 I/O は，デバイスによるデータ転送を制御するための I/O で，転送するデータの場所やアクセス方法，データ転送の開始，終了などを指定する．データ I/O は，実際にデータ転送を行う I/O である．BitVisor は制御 I/O を捕捉してアクセスの状態を把握し，データ I/O を捕捉してデータを取得，更新する．これにより，OS から透過的に暗号化などの処理を実現できる．たとえば，ハードディスクデバイスに関しては，BitVisor は，ディスクに書き込まれる前のデータ I/O の値を暗号化し，

Vol.2010-DPS-142 No.35
Vol.2010-CSEC-48 No.35
2010/3/5

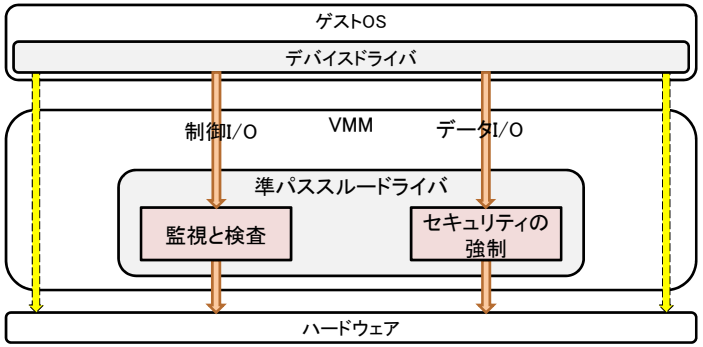

図 1 準パススルー型 VMM BitVisor の構成
Fig. 1 Structure of a parapass-through VMM BitVisor

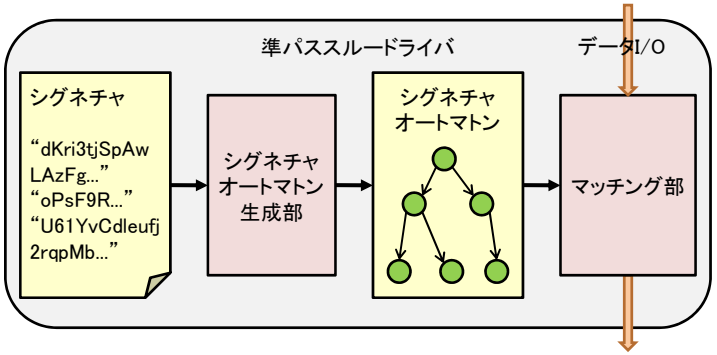

図 2 提案システムの構成
Fig. 2 Structure of the proposed system

ディスクから読み出されたデータ I/O の値を復号する．

3. 提案システムの概要

本研究のシステムの構成を**図 2** に示す．本システムはシグネチャオートマトン生成部とマッチング部から構成される．両者とも準パススルードライバの中に実装されている．シグネチャオートマトン生成部は，マルウェアを表現する文字列パターン（シグネチャ）の集合を受け取り，マッチング処理に適したメモリ上のデータ構造（シグネチャオートマトン）を構築する．マッチング部は，データ I/O の内容をシグネチャオートマトンと照合することによりマルウェアを検出する．具体的には，暗号化の前，および，復号の後に，BitVisor 内部のバッファが保持しているデータに対してマッチング処理を実行する．シグネチャにマッチするバイト列がある場合，対策処理（後述）を実行する．

BitVisor は，PIO（プログラム I/O），メモリマップト I/O，DMA（Direct Memory Access）の 3 つの転送方式によるデータ I/O を捕捉できる．本システムは，それらすべての方式のデータ I/O で転送されるデータに対してマッチング処理を行う．データ I/O が捕捉されるデバイスは，ATA（ハードディスク）デバイス，ネットワークデバイス，USB ストレージデバイスである．ただし現時点では，ATA デバイスのデータに対するマッチング処理のみが実装されている．シグネチャとしては，ClamAV[3] が提供しているものを本システム用に加工して再利用する．

本システムが有効にはたらくシナリオの 1 つを以下に述べる．ユーザは会社において，自分に割り当てられたコンピュータを用いて業務を行っている．そのコンピュータには本システムがインストールされており，そのユーザは本システムの上で Windows OS を稼働させて利用している．そのユーザはセキュリティや OS に関する基本的な知識を持っていない．ある日，そのユーザは，知人から借りた USB メモリの中に，中身が未知である ZIP ファイルを発見する．それはマルウェアのバイナリコードを ZIP 圧縮したファイルである．そのユーザは ZIP ファイルを展開し，生成されたファイルを興味本位でダブルクリックする．もし本システムが存在しなかったら，そのマルウェアは実行されてしまう．しかし実際には，本システムが ZIP ファイルの展開時に，生成されようとしているファイルの中身にシグネチャを検出し，警告を発する．

4. 実装の詳細

4.1 シグネチャオートマトン生成部

準パススルー型 VMM は，VMM の下で動作する OS（ホスト OS）を持たない．よって，シグネチャをホスト OS のファイルシステムから読み込むことはできない．また，本研究では，VMM の上で動作する OS（ゲスト OS）上のファイルは，OS のユーザや攻撃者によって変更される可能性があると仮定している．よって，シグネチャをゲスト OS に置くこともできない．そこで，現在の実装では，本システムの実行型バイナリにシグネチャを予め埋め込んでおくという方法を採用している．当然ながら，この方法ではシグネチャが古いままになるという問題がある．本システムが起動後にサーバに接続して，自身のメモリやバイナリ

Vol.2010-DPS-142 No.35
Vol.2010-CSEC-48 No.35
2010/3/5

に埋め込まれたシグネチャを更新する処理を組み込むことは，今後の課題である．

本システムのソースコード中には，シグネチャを表現する文字列の配列が埋め込まれている．シグネチャオートマトン生成部は，VMM の起動時にその配列のデータを読み込み，Aho-Corasick 法[1] によって，シグネチャオートマトンを作成する．シグネチャオートマトンにより，膨大な数のシグネチャと検査対象データとの間のマッチングを高速に行うことができる．

Aho-Corasick 法は，検査対象の文字列から特定の文字列パターンを探すためのアルゴリズムである．複数の文字列パターンに対して一斉にマッチングを行えるため，特にパターンの数が膨大である場合に性能面で効果を発揮する．Aho-Corasick 法では，まず，複数のパターンを表現する有限オートマトン（より具体的には，トライ木）を構築する．検査時には，検査対象の文字列から 1 文字ずつ取り出してその有限オートマトン上で遷移を行うことにより，マッチング処理を実現する．

4.2 マッチング部

4.2.1 マッチング関数

マッチング部は，VMM が捕捉したデータのバイト列と，シグネチャオートマトンとの間のマッチング処理を実行する．現在は，ATA デバイスにおける PIO 転送と DMA 転送のデータに対してのみマッチング処理が実装されている．本システムは，マッチング処理を実行する関数 `sigmatch` を定義している．`sigmatch` は，バッファへのポインタを引数に受け取り，バッファに含まれるバイト列をシグネチャオートマトンと照合する関数である．マッチした場合には，VMM のログに警告を書き込んだり，そのバイト列を別のバイト列に置換したり，デバイスへのアクセスを失敗させるなどの対策処理を実行する．現在は，ログへの警告の書き込み処理のみが実装されている．

以下では，BitVisor が，ATA デバイスとの間で PIO 転送および DMA 転送を行う部分の処理，および，その部分に対する本研究の拡張について述べる．

4.2.2 PIO データ転送

PIO データ転送では，OS が専用の I/O 命令を発行してデータの読み書きを行う．VMM は I/O 命令を捕捉して，OS が実行しようとしている処理を解釈し，必要に応じて制御する．ATA デバイスのデータ I/O では，複数の連続する I/O 命令によって転送されるデータを VMM がバッファリングする．I/O 命令の 1 回の実行で転送されるデータが 1〜4 バイトであるのに対し，物理デバイスの転送単位が非常に大きい（512 バイトなどである）ためである．BitVisor は，必要な量のデータが溜まった時点で暗号化および復号を行う．

```
struct ata_channel {
        ...
        int pio_buf_index;
        int pio_block_size;
        u8 pio_buf[2048];
        bool interrupt_disabled;
        int (*pio_buf_handler)(struct ata_channel *channel, int rw);
        ...
        u32 guest_prd_phys;
        void *shadow_prd;
        u32 shadow_prd_phys;
        void *shadow_buf;
        ...
};
```

図 3 ATA チャネルを表現する構造体
Fig. 3 Structure for ATA channels

上記のバッファリングで用いられるバッファは，BitVisor の ATA ドライバの部分のソースコードにおいて，ATA チャネルを表現する構造体の中に宣言されている．その構造体の宣言の抜粋を**図 3** に示す．`pio_buf` がそのバッファである．BitVisor は関数 `ata_handle_data` と関数 `ata_handle_pio_data` において，このバッファを用いて PIO 転送を行う．本システムでは，これらの関数の中で，`pio_buf` を引数として関数 `sigmatch` を呼び出すことにより，マッチング処理を実現している．

4.2.3 DMA データ転送

DMA は，CPU を介在させることなく，DMA コントローラと呼ばれるハードウェアがデバイスとメモリの間で直接データを転送する仕組みである．デバイスドライバが転送の指示を DMA コントローラに与えて DMA の動作を開始させると，デバイスとメモリの間で自動的にデータが転送される．

BitVisor は I/O データを暗号化するために，DMA で転送されるデータも取得する必要がある．BitVisor はそのためにシャドウ DMA ディスクリプタという仕組みを用いている．シャドウ DMA ディスクリプタとは，OS が作成する DMA ディスクリプタ（ゲスト DMA ディスクリプタ）とは別に，VMM が作成する DMA ディスクリプタである．DMA コントローラのハードウェアに実際に設定されるのは，シャドウ DMA ディスクリプタのほうである．シャドウ DMA ディスクリプタでは，VMM 内部のバッファ（シャドウバッファ）が転送用のバッファとして設定されている．その結果，DMA によるデータ転送は，デバイス

Vol.2010-DPS-142 No.35
Vol.2010-CSEC-48 No.35
2010/3/5

とシャドウバッファとの間で行われる．この仕組みにより，DMA 転送の制御の多くを OS に任せつつ，DMA で転送されるデータを VMM が取得，加工できる．

シャドウバッファは，図 3 に示した ATA チャネルを表現する構造体の中に宣言されている．この構造体に含まれる `shadow_buf` がシャドウバッファである．BitVisor は関数 `ata_dma_handle_rw_sectors` において，このバッファを用いて DMA 転送を行う．本システムでは，この関数の中で，`shadow_buf` を引数として関数 `sigmatch` を呼び出すことにより，マッチング処理を実現している．

5. 議　　論

5.1 検出漏れが発生する場合

本システムではデータ I/O で転送されるデータをブロック単位で検査するので，いくつかの場合で検出漏れが発生する．まず，マルウェアが複数のデータブロック（典型的にはディスクの複数のセクタ）にまたがる場合には，個々のブロック上にはマルウェアの一部だけしか存在しないため，検出漏れが発生する．また，シグネチャの文字列が非常に長い場合（たとえばセクタのサイズよりも長い場合）には，現在の実装では，その文字列とのマッチングは必ず失敗する．これらの問題を緩和するには，過去に転送されたいくつかのブロックを VMM に保存しておき，複数ブロックを結合したデータに対してマッチング処理を行う拡張が有効であると予想される．ただしその拡張はオーバヘッドを増加させるため，性能とのトレードオフを考慮する必要がある．

また，本システムは，OS 上で暗号化が用いられた場合に，マルウェアを検出できないことがある．たとえば，ファイルシステム層でデータを暗号化している場合には検出できない．さらに，自身を暗号化するマルウェアも検出できない．本システムはデータ I/O で転送されるバイト列は検査するが，それ以外のメモリ上のデータは検査しない．よって，自身の暗号化と復号により，シグネチャにマッチするデータをメモリ上だけにしか出現しないようにしているマルウェアは，本システムでは検出できない．

5.2 キャラクタデバイスとネットワークデバイス

キャラクタデバイスのデータ I/O は，ATA デバイスのようなブロックデバイスのデータ I/O と異なり，1 バイト単位でデータを転送する．また，ネットワークデバイスのデータ I/O では，パケット単位でデータが転送される．これらのデータ I/O に対するマッチング処理をどう実現すべきかは，自明ではない．今後，この問題についても取り組んでいく必要がある．

表 1　実験環境
Table 1　Platform of experiments

CPU	Intel Core 2 Duo P8600 2.4GHz
メモリ	1GB
HDD	Toshiba MK1252GS 120 GB
VMM	BitVisor 1.0
ベスト OS	Debian Lenny 5.0, 2.6.26-2-686
ベンチマーク	Unixbench 4.1.0

6. 予 備 実 験

本システムの有効性を評価するために，2 つの実験を行った．実験環境を**表 1** に示す．本実験では BitVisor の暗号化機能は用いていない．すなわち，BitVisor はデータ I/O の捕捉後に暗号化や復号を行わず，マッチング処理のみを行う．ATA デバイスのデータ転送は DMA によって行った．

1 つ目の実験は，本システムが実際に OS 上のマルウェアを検出できるかどうかを確認するものである．本システムの上で動作する OS 上でテキストエディタを実行し，シグネチャにマッチする文字列を入力した．その文字列をファイルに保存しようとしたところ，本システムがそれを検知したことを確認した．

2 つ目の実験は，本システムが与える実行時間オーバヘッドを測定するものである．Unixbench ベンチマーク[13] を実機，拡張前の BitVisor，本システムの上で動作させ，スコアを測定した．シグネチャの数は 1 個，50 個，100 個と変化させた．シグネチャの長さはすべて 30 バイトとした．なお，ベンチマークの実行中，シグネチャには一度もマッチしなかった．

結果を**表 2** に示す．まず，BitVisor の導入により，全体のスコア（Final Score）が約 55%低下した．この性能低下は I/O 処理や例外（ページフォルトなど）の捕捉に伴うものであると考えられる．一部のベンチマークの低いスコアに引かれる形で全体のスコアが低くなっているが，個々のベンチマークのスコアの多くでは，差は極めて小さい．元の BitVisor と本システムはほぼ同等のスコアであり，今回導入したマッチング処理によってほとんど性能は低下しなかった．また，本実験程度の規模のシグネチャ数の変化では，オーバヘッドはほとんど変化しないことも分かった．一般に，Aho-Corasick 法では，マッチングにかかる計算量はシグネチャの数に比例する．よって，シグネチャの数を例えば数万個や数十万個に増やすと，オーバヘッドが際立つ可能性もある．現在は実装上の問題によりその実験は行え

Vol.2010-DPS-142 No.35
Vol.2010-CSEC-48 No.35
2010/3/5

表 2 Unixbench のスコア
Table 2 Score of Unixbench

	実機	BitVisor	提案システム	提案システム	提案システム
シグネチャ数	—	—	1	50	100
Dhrystone	1088.3	1106.2	1092.6	1117.6	1102.7
Whetstone	410.8	411.3	422.0	349.0	429.2
Execl	357.1	28.9	27.8	28.9	28.9
File Copy (1024, 2000)	1150.4	1120.2	1122.1	1108.2	1052.4
File Copy (256, 500)	889.8	894.3	892.8	886.4	820.3
File Copy (4096, 8000)	1632.6	686.7	691.8	686.0	685.6
Pipe	829.1	832.0	834.4	835.5	836.5
Process Creation	408.3	36.7	28.4	29.7	29.3
Shell Scripts	2541.7	288.3	286.0	286.0	286.7
System Call	1282.6	1278.1	1281.5	1277.5	1278.5
Final Score	**884.9**	**398.8**	**387.9**	**383.5**	**385.6**

ていないが，今後取り組む予定である．

7. 関 連 研 究

Livewire[4] は，攻撃やマルウェアを VMM 層で検知する方式を示した先駆的な研究である．本研究のシステムと同じく，OS 上のデータの中から特定の文字列を検出する機能も有している．Livewire はホスト OS 上で動く VMM である VMware Workstation を改造して実装されている．よって，BitVisor のような準パススルー型 VMM の中でマルウェアを検知するための方式については，彼らの研究では明らかにされていない．

VMwatcher[5] は，VM 上で動作する OS のためのアンチマルウェア処理を，その VM の外で動くソフトウェアによって実行するシステムである．VMwatcher は VMware Server，QEMU，Xen，User-Mode Linux の 4 種類の VMM をサポートしている．これらの VMM はどれも，OS 上で動作するか，管理用の特別な OS を必ず動作させる．よって，検査対象である VM の外でアンチマルウェアソフトウェアを実行させることは難しくない．一方，準パススルー型 VMM では，VM の外でアンチマルウェアソフトウェアを実行させることは単純ではない．本研究はそれを実現するための一つの設計を示した．

SecVisor[11] は事前に許可していないコードがカーネル権限で実行されるのを防ぐ VMM である．SecVisor は BitVisor と同じくハードウェアの上で直接動作する．SecVisor を利用するためには，SecVisor の上で動く OS のソースコードを修正する必要がある．よって，Windows のようなクローズドソースの OS を SecVisor で保護することは難しい．一方，本研究のシステムは，OS のソースコードを必要としない．

忠鉢らは，ユーザがファイルレベルでアクセス制御のポリシーを与えつつも，VMM がディスクブロックレベルでそのポリシーを強制するシステムを提案している[16]．そのシステムは，本研究のシステムと同じく，BitVisor に含まれる ATA デバイスのドライバを改造することにより実装されている．彼らのシステムはアクセス制御機能を提供しているのに対し，本研究のシステムはマルウェア検出機能を提供している．

Viton[6] は，本研究のシステムと同じく BitVisor を改造して構築されたセキュリティ向上のための VMM である．Viton は，システムレジスタ，カーネルコード，カーネルデータなどの改ざんを検出することができる．Viton が提供する機能はあくまで改ざん検出であり，マルウェアのシグネチャを検出することはできない．

野村らは，BitVisor を拡張してマルウェアの挙動解析システムを構築した研究について報告している[17]．そのシステムは OS 上で動作するマルウェアのシステムコール情報を取得するものであり，本研究で提案したような，マルウェアを検出するシステムではない．

Intel により，LVMM と呼ばれる軽量な VMM を用いて攻撃を検知する方式が提案されている[9]．この方式では，LVMM 上で 2 つの仮想マシンを動作させる．片方の仮想マシン（user partition VM）は日常業務に利用する．他方の仮想マシン（services partition VM）は，user partition VM のネットワーク通信を監視し，攻撃などを検出する．この方式によるシステムでは，本研究のシステムと異なり，セキュリティ処理のための専用の仮想マシンを動作させる必要がある．また，ストレージに対する I/O 処理を監視しないため，ストレージを介して侵入するマルウェアを検出できない．

Lares に関する研究[7] では，ゲスト VM とセキュリティVM と呼ばれる 2 つの仮想マシンを VMM 上で動作させ，セキュリティVM がゲスト VM のセキュリティ検査を行うアーキテクチャが提案されている．ゲスト VM 上で動く OS のコードに予めフックを挿入し，その OS で発生したイベントの情報が，セキュリティVM 上で動く OS に送られるようにする．セキュリティVM 上では，セキュリティのためのアプリケーションが動作しており，送られたイベントの情報が正常なものであるかどうかを検査する．ゲスト VM のためのマルウェア検出処理をセキュリティVM が実行する着想も示されて．このアーキテクチャでは，本研究のシステムとは異なり，OS のコードを修正する必要がある．さらに，セキュリティのための特別な仮想マシンを動作させる必要がある．

NICKLE[10] はカーネルコードの integrity を保護するための，VMM を用いたセキュリティシステムである．NICKLE はあくまでルートキットなどのカーネルレベルのマルウェア

Vol.2010-DPS-142 No.35
Vol.2010-CSEC-48 No.35
2010/3/5

の実行を防止するものである．本研究のシステムとは異なり，ユーザレベルのマルウェアを検出することはできない．また，NICKLE は QEMU，VirtualBox，VMware Workstation の 3 種類の VMM 上に実装されているが，それらの VMM はいずれもホスト OS 上で動作するものであり，本研究が対象としている準パススルー型 VMM とは構造が大きく異なる．

8. まとめと今後の課題

本論文では，準パススルー型 VMM である BitVisor を拡張して，VMM 層でのマルウェア検出を可能にしたシステムについて述べた．実験では，マルウェアのシグネチャにマッチするデータをディスクに保存しようとしたときに，本システムがそれを検出したことを確認した．また，オーバヘッドの測定を行い，シグネチャの数が少なく長さが短い場合には，本システムによるオーバヘッドが極めて小さいことを確認した．

今後の課題を以下に述べる．第一に，シグネチャの数と長さを増加させたときのオーバヘッドを測定することが挙げられる．第二に，5.1 節で指摘した検出漏れの問題を解決することが挙げられる．第三に，ATA デバイス以外のデバイスでもマッチング処理ができるようにシステムを拡張することが挙げられる．

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）の支援を受けて行われた．

参 考 文 献

1) Aho, A.V. and Corasick, M.J.: Efficient String Matching: An Aid to Bibliographic Search, *Communications of the ACM*, Vol.18, No.6, pp.333–340 (1975).
2) 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, *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP 2003)*, pp.164–177 (2003).
3) Clam AntiVirus, `http://www.clamav.net/`.
4) Garfinkel, T. and Rosenblum, M.: A Virtual Machine Introspection Based Architecture for Intrusion Detection, *Proceedings of the 10th Annual Network and Distributed System Security Symposium (NDSS 2003)* (2003).
5) Jiang, X., Wang, X. and Xu, D.: Stealthy Malware Detection Through VMM-Based "Out-of-the-Box" Semantic View Reconstruction, *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS 2007)*, pp. 128–138 (2007).
6) Murakami, J.: A Hypervisor IPS based on Hardware Assisted Virtualization Technology, *Black Hat USA 2008* (2008).
7) Payne, B.D., Carbone, M., Sharif, M. and Lee, W.: Lares: An Architecture for Secure Active Monitoring Using Virtualization, *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, pp.233–247 (2008).
8) QEMU, `http://www.qemu.org/`.
9) Ramachandran, M., Smith, N., Wood, M., Garg, S., Stanley, J., Eduri, E., Rappoport, R., Chobotaro, A., Klotz, C. and Janz, L.: New Client Virtualization Usage Models Using Intel Virtualization Technology, *Intel Technology Journal*, Vol. 10, No.3, pp.205–216 (2006).
10) Riley, R., Jiang, X. and Xu, D.: Guest-Transparent Prevention of Kernel Rootkits with VMM-based Memory Shadowing, *Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection (RAID 2008)*, Lecture Notes in Computer Science, Vol.5230, pp.1–20 (2008).
11) Seshadri, A., Luk, M., Qu, N. and Perrig, A.: SecVisor: A Tiny Hypervisor to Provide Lifetime Kernel Code Integrity for Commodity OSes, *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP 2007)*, pp.335–350 (2007).
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, *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE 2009)*, pp.121–130 (2009).
13) Unixbench, `http://ftp.tux.org/pub/benchmarks/System/unixbench/`.
14) VirtualBox, `http://www.virtualbox.org/`.
15) VMware Workstation, `http://www.vmware.com/`.
16) 忠鉢洋輔，品川高廣，加藤和彦：仮想マシンモニタによるゲスト OS のファイル保護，情報処理学会研究報告，Vol.2009-OS-111, No.31 (2009).
17) 野村和裕，吉村拓也，毛利公一：仮想計算機モニタを使ったマルウェアの挙動解析，マルウェア対策研究人材ワークショップ 2009 (MWS 2009) (2009).