マルウェア解析のための高速かつ安全なVMI機構

森 瑞穂[1,a)]　味曽野 雅史[2]　八巻 隼人[1]　三輪 忍[1]　本多 弘樹[1]　品川 高廣[2]

概要：マルウェアの挙動や攻撃手法を解析する手段として，仮想マシン上のプログラムの内部状態を観察する Virtual Machine Introspection (VMI) という手法が用いられている．VMI には、主に外部のハイパーバイザから行う Out-of-the-box 方式と仮想マシン内部から行う In-the-box 方式の 2 つがあるが，両者は解析時の動作速度の高速性と解析システムを保護・隠蔽する安全性の面でトレードオフの関係にある．そこで我々は，高速かつ安全な VMI 機構として FastVMIX を提案する．FastVMIX では，マルウェアを解析する解析エージェントを仮想マシン内部に挿入することによってハイパーバイザへのコンテキストスイッチを減らしつつ，Intel CPU がサポートする VMFUNC の EPTP Switching と Huge Page を用いた高速な動的メモリ保護変更機構により、マルウェアから解析エージェントのメモリ領域を保護・隠蔽する．また，準パススルー型ハイパーバイザを用いることで、仮想化のオーバーヘッド削減及び隠蔽度の向上を図る．本論文では，BitVisor をベースに FastVMIX を実装した結果を報告する．

キーワード：バイナリ解析, セキュリティ, VMI, HugePage, EPTPSwitching

A Fast and Secure VMI Mechanism for Malware Analysis

Abstract: As a means of quickly analyzing malware behavior and attack methods, a technique called Virtual Machine Introspection (VMI) is used to observe the internal state of programs on a virtual machine. A typical VMI system mainly takes either an out-of-the-box (i.e., with hypervisor) or in-the-box (i.e., within the virtual machine) approach; however, these two approaches involve a trade-off between the analysis speed and the security of protectiong and hiding the analysis system. In this paper, we propose FastVMIX that realizes fast and secure VMI. FastVMIX reduces the number of context switches to the hypervisor during malware analysis by inserting an analysis agent in the target virtual machine, while protecting and hiding the agent's memory area by switching memory protection with EPTP switching of VMFUNC and huge pages supported by Intel CPUs. In addition, we used a para-pass-through hypervisor to reduce the overhead of virtualization and improve the degree of hiding. This paper reports several experimental results of FastVMIX built on BitVisor.

Keywords: Binary Analysis, Security, VMI, HugePage, EPTPSwitching

1. はじめに

マルウェア解析では，その挙動や攻撃手法をすばやく解析することが必要であるが，一般にマルウェアのソースコードは入手できないため，バイナリ解析によるリバースエンジニアリングが行われている．リバースエンジニアリングによるマルウェア解析には，主にマルウェアを動作させずに行う静的解析と実際にマルウェアを動作させながら行う動的解析の 2 種類の手法が存在するが，動的解析は静的解析に比べて確実に動作を把握することができることから，近年様々な手法が研究されている [9].

動的解析手法の 1 つとして，ハイパーバイザを用いた解析システムがある [6]．このシステムでは，サンドボックス化された仮想マシン上でマルウェアを実行し，その振る舞いを観測することによりマルウェアの解析を行う．ホストマシンはハイパーバイザによって保護されるため，仮にマルウェアがゲスト OS に侵入した場合でも当該仮想マシンを破棄すればよく，マルウェアの解析を安全に行うことができる．また，マルウェアの中には動的解析耐性を持つ

[1] 電気通信大学
1-5-1, Chofugaoka, Chofu, Tokyo 182–8585, Japan
[2] 東京大学
7-3-1, Hongo, Bunkyo-ku, Tokyo 113–8654, Japan
[a)] m-mori@hpc.is.uec.ac.jp

ものがあり，単なるデバッガでは解析を妨害されてしまう恐れがあるが，ハイパーバイザを用いることでマルウェアから解析システムを隠蔽しやすくなる [3].

仮想マシン上で動作するマルウェアを解析する手法に Virtual Machine Introspection (VMI) [4] があり，仮想マシンのメモリ内容を解析してゲスト OS に頼らずに内部状態を把握できる．VMI の実現方式は，ハイパーバイザから解析を行う Out-of-the-box 方式 [7], [10], [12] と仮想マシン内部から解析を行う In-the-box 方式 [1], [2], [4], [5] の 2 つに分けられる．Out-of-the-box 方式では，解析システムがハイパーバイザ内で動作するため，仮想マシンから保護・隠蔽されてマルウェアに攻撃・検知されにくい反面，解析時にハイパーバイザへのコンテキストスイッチが多く発生して動作速度が低下する．一方 In-the-box 方式は，仮想マシン内部に解析エージェントを挿入するため，コンテキストスイッチが不要で解析を高速に行える反面，マルウェアによって解析エージェントのコードやデータが検知・攻撃される可能性がある．すなわち，両者は高速性と安全性の面でトレードオフ関係にある．

本研究では, In-the-box 方式で高速性を実現しつつ, Out-of-the-box 方式と同様に解析エージェントを保護・隠蔽できる高速かつ安全な VMI である FastVMIX を提案する．FastVMIX は，解析時のコンテキストスイッチを削減するために，解析エージェントを仮想マシン内部に挿入して同じアドレス空間内で動作させつつ，解析エージェントのメモリ領域を保護するために，解析エージェントのコード・データ領域へのアクセスを解析エージェント実行時にのみ許可することで、マルウェアからの検知・攻撃を防止する．

アクセス権の切り替えを高速に行うために、Intel 社製の CPU でサポートされている仮想化支援機構に含まれる拡張機能 VMFUNC の一つである EPTP Switching を利用して、ハイパーバイザへコンテキストスイッチすることなく 1 命令でネステッドページテーブルを切り替えることで，ページ単位で保護モードを切り替える。また、仮想マシン内の OS のページテーブル保護を低オーバーヘッドで実現するために、Huge Page を活用して保護対象のページを削減する．さらに、準パススルー型ハイパーバイザを採用することで、仮想化によるオーバーヘッドを削減して解析時の動作速度を向上させるとともに、ゲスト OS から仮想マシン環境であることを検知されにくくすることで隠蔽性も向上させる。

本稿の構成は以下のとおりである．まず 2 章で関連研究を説明する．3 章では FastVMIX の概要を説明し，4 章で実装を詳しく説明する．5 章で評価結果を示し，6 章で本論文をまとめる．

2. 関連研究

2.1 In-the-box 方式

Qin らはアプリケーションの解析基盤 NEM を提案している [10]．NEM は In-the-box 方式を採用しており，EPTP Switching と VE を用いることで VMEXIT を実行することなく高速にアプリケーションを解析する．NEM ではプロセスレベルのトレースのみを行うことができる一方で，仮想マシン全体を保護することはできない．また，安全性の観点から見ると，ページテーブル保護が行われておらず，仮想メモリのマッピングを操作し，攻撃することが可能である．

Shi らはゲスト OS の解析基盤 Shadow Monitor を提案している [12]．Shadow Monitor は，EPTP Switching と VE を用いることで VMEXIT を実行することなく，高速かつ自由にトラップを仕掛けてゲスト OS の情報を取得することを目的としたシステムである．Shadow Monitor では，自由なトラップのために INT20 などの命令を挿入し，In-the-box 方式で高速に解析用エージェントを動作させる．また，安全性を担保するために EPTP Switching による Memory Isolation を行っている．ただし，安全性の観点から見ると，保護対象のページテーブルが 4 エントリのみとなっており脆弱である．

Monirul らはゲスト OS の解析基盤 SIM を提案している [11]．SIM は Shadow Paging と CR3_TARGET_LIST を活用し，VMEXIT なしで高速に解析が行える KVM 上に実装された基盤である．CR3_TARGET_LIST は Intel CPU に実装された，ゲスト OS が CR3 レジスタ (ページテーブルのトップレベルアドレスが格納されている) を操作しても VMEXIT しないようにできるホワイトリスト機能である．Shadow Paging と呼ばれるページテーブルの分離により Isolation が達成されているが，CR3_TARGET_LIST には登録可能な個数の制限があり，ホワイトリストのメンテナンスコストが存在する．

Luţaş らは，ゲスト OS の解析基盤 VE-VMI を提案している [7]．VE-VMI は VE と EPTP Switching により VMEXIT なしで VMI を用いてゲスト OS を防御するためのシステムである．システムには Out-of-the-box な Main VMI Module が存在するが，システムの要となる Filtering Agent が In-the-box であるため本節に記載した．ゲスト OS のページテーブルエントリのビット部分を触ると VE にてゲスト内で処理できるようにし，不正なメモリ操作を防いでいる．実際に VMI をする部分は Out-of-the-box の Main VMI Module であるが，機能として防御のみに特化しているため，提案手法と動機が異なる．

2.2 Out-of-the-box 方式

Tamas らはゲスト OS の解析基盤 DRAKVUF を提案している [5]．DRAKVUF は Xen ハイパーバイザをベースとして開発されたバイナリ解析ツールである．Out-of-the-box 方式を採用しており，ステルスブレークポイントなどの高度な技術を有している．またシステムコールのトラップなど詳細な分析も行うことができる．しかし，DRAKVUF は Out-of-the-box 方式のため，システムコールトラップ時にはハイパーバイザと仮想マシン間のコンテキストスイッチが発生し，その時間的なオーバーヘッドが問題となる．

3. FastVMIX の概要

本章では FastVMIX で想定する脅威モデルと提案システムの概要を述べる，

3.1 脅威モデル

FastVMIX では，ゲスト OS 内部に侵入した攻撃者は、仮想マシンの操作を自在に行い，ゲスト OS のカーネル内の重要なデータを書き換えることが可能であると想定する．しかし，攻撃者はマシンに対する物理的アクセスはできないものとする．

想定される具体的な攻撃としては，仮想マシン内の OS のページテーブルへの攻撃がある．攻撃者はゲスト OS の仮想アドレスと物理アドレスのマッピングを操作し，ゲスト上で実行される任意のプログラムの動作を妨害する．具体的には，ページテーブルの内容を書き換えてゲストのシステム領域のメモリ空間を破壊することが考えられる．その他の攻撃としては，ゲストのシステム領域のメモリを読み書きして破壊する，ゲスト OS のシステムコールのエントリポイントを書き換えて不正なシステムコールを実行する，などが考えられる．また，攻撃時にはゲスト OS のカーネル内のコードが使用されること（コード再利用攻撃）も想定する．

3.2 システム概要

FastVMIX の概要を**図 1** 右側に示す．仮想マシン上で動作するバイナリの解析を高速に行うために，FastVMIX では In-the-box 方式を採用する．仮想マシン内部に挿入された解析エージェントは，図 1 に示すようにゲスト OS と同じ権限で動作する．そのため，図 1 左側に示した Out-of-the-box 方式とは異なり，解析エージェントを実行するために仮想マシンとハイパーバイザ間でコンテキストスイッチを行う必要はなく，FastVMIX は高速なバイナリ解析が行える．

前節で述べたように仮想マシン上で動作するマルウェアは仮想ページテーブルを攻撃できるため，従来の In-the-box 方式の解析システムではマルウェアが解析エージェントの存在を検知し，その動作を妨害してしまう恐れがある．そ

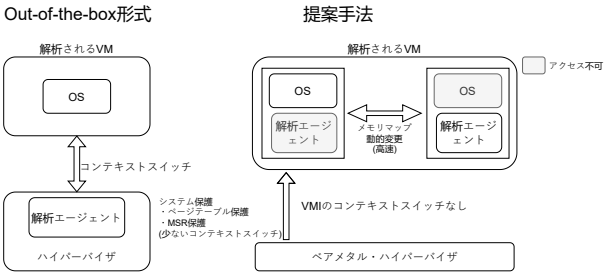

図 1 FastVMIX

こで FastVMIX では，通常（マルウェア）動作時と解析時の 2 種類の EPT を用意し，これらを切り替えることでマルウェアによる仮想ページテーブルへの攻撃を防ぐ．具体的には，通常動作時には解析用コードをゲストの物理メモリにマップしないことで，マルウェアが解析エージェントの存在を検知できないようにする．また解析時には，解析用コードをゲストの物理メモリにマップした上で解析用コード以外のコードの実行を禁止することで，マルウェアの解析を行いつつマルウェアによるコード再利用攻撃を防ぐ．FastVMIX は上述のメモリマップ変更を高速に行うことで，解析時のシステム性能低下を抑制する．

また，仮想ページテーブルを保護するために，ページテーブルの読み出し時にエントリの内容が変更されていないことを検証する機能をハイパーバイザに追加する．上述の検証作業はシステムの性能を低下させる要因となり得ることから，FastVMIX では保護対象のページを解析エージェントの実行に関係するページに限定した上で保護対象のページの数を最小化する．

さらに，システムコールエントリーポイントの書き換えを防ぐために，エントリーポイントに対応する MSR への読み書きを監視する機能もハイパーバイザに追加する．

4. 実装

我々は BitVisor [13] をベースに FastVMIX の実装を行った．FastVMIX は主に以下の 3 つのモジュールで構成されている．

(1) 解析エージェントとして仮想マシン内部に挿入される PIC（Position Independent Code）バイナリ
(2) システム全体を保護するハイパーバイザ
(3) ハイパーバイザの操作やカーネル空間での準備を行うカーネルモジュール

以下，それぞれについて詳しく述べる．

4.1 PIC バイナリ

PIC バイナリは主にシステムコールのトラップ機能を担当する解析エージェントである．解析エージェントを位置独立コードとしてビルドすることにより，任意のメモリ空間に配置可能なバイナリとなる．PIC バイナリは後述するカーネルモジュールによってロードされ，カーネル内へ

マッピングされる．

PIC バイナリはフック機能と VMI 機能を有する．以下，それぞれの機能を説明する．

4.1.1 フック機能

フック機能はアセンブリ言語で記述されており，システムコールをフックして次項で述べる VMI 機能を実行する．また，VMI 機能の実行の前後で EPT の切り替えも行う．フック機能の処理の流れを以下に示す．

(1) SWAPGS 命令で GS レジスタをカーネルモードにする．
(2) 現在のスタックポインタを PER_CPU_SECTION 内の領域に保存する．
(3) カーネルのスタックポインタを PER_CPU_SECTION 内の領域から復帰する（これは Linux カーネルが準備している変数の current_top_of_stack）．
(4) 現在のレジスタ情報をスタックに積む．
(5) VMFUNC 命令で解析用のメモリマップに変更する．
(6) VMI 機能を呼び出す．
(7) VMFUNC 命令で通常のメモリマップに変更する．
(8) レジスタ情報を復帰する．
(9) 元のシステムコールエントリポイントへ飛ぶ（PER_CPU_SECTION 内にアドレスが保存されている）．

フック機能では，まず最初に SWAPGS 命令を用いて GS レジスタをカーネルモードに変更する．これは，以降の処理で PER_CPU_SECTION 内の変数にアクセスする必要があり，上記の変数にアクセスする際はカーネルモードでなければならないためである．

次に，スタックポインタに対する操作を行う．後述するように PIC バイナリはユーザ空間でページの確保とマッピングを行うが，カーネル空間とユーザ空間ではスタックが異なるために，これらの復帰，保存操作を行う．この操作では PER_CPU_SECTION 内の変数にアクセスする必要がある．

次に，VMFUNC 命令を用いて EPT を切り替え，VMI 機能を実行する．VMIFUNC 命令は後述するハイパーバイザの機能に関連した命令であり，これを利用することでハイパーバイザへのコンテキストスイッチなしに EPT を切り替えることができる．前章で述べたように FastVMIX では 2 種類の EPT を動的に切り替えることでページテーブルを保護しており，特権的な操作の必要な VMI 機能の前後でメモリマップを変更する．

最後に，元のシステムコールエントリポイントにジャンプする．ただし，ジャンプ先のアドレスは，元のシステムコールエントリポイントにあるオフセット値を足した位置とする．これは，元のシステムコールエントリポイントには SWAPGS 命令を実行して GS レジスタをカーネルモードに変更する処理が含まれており，この冗長な処理の実行を避けるためである．上記の処理は PIC バイナリの実行開始時に既に行っていることから，PIC バイナリは元のシステムコールエントリポイントに SWAPGS 命令の長さ分のオフセット値を足したアドレスへジャンプすることにより，元のシステムコールエントリポイントにある SWAPGS 命令の実行をスキップする．

4.1.2 VMI 機能

今回は単純な VMI 機能のみを実装した．動作としては，システムコールの引数とシステムコール番号をログ用のバッファに書き込むだけである．今回実装した PIC バイナリは，glibc などの標準ライブラリとリンクしない小さなバイナリを志向した．そのため，PIC バイナリ内で C 言語の標準関数を利用できず，ログ出力に必要な sprintf 関数や memcpy 関数などは自前で実装した．

後述するように，PIC バイナリは簡単に入れ替えることが可能であり，機能の変更や拡張が容易に行える利点がある．例えば，今回の実装では PIC バイナリを標準ライブラリをリンクしないバイナリとしたが，将来的には標準ライブラリをリンクするようなバイナリに変更することも可能である．また，VMI 機能をより複雑にしたり，様々なバリエーションを持たせる場合でも，独立した PIC バイナリをロードするだけで機能が拡張できる．さらに，PIC バイナリはハイパーバイザとは独立しているため，ハイパーバイザに手を加える必要があった従来の VMI 機能の実装と比べてデバッグが容易である．

4.2 ハイパーバイザ

BitVisor ベースのハイパーバイザは主にシステムの保護を担当する．3.1 節で述べた攻撃を防ぐために，FastVMIX では BitVisor に以下の 3 つの保護機能を追加する．

(1) 1GB Huge Page を用いたページテーブル保護
(2) EPTP Switching を用いたメモリマップの動的変更による保護
(3) MSR 保護

以下，それぞれについて詳しく述べる．

4.2.1 1GB Huge Page を用いたページテーブル保護

ページテーブルを読み込む際は必ず CR3 レジスタにページテーブルのアドレスがセットされる．そこで FastVMIX では，ページテーブルを保護するために MOV-TO-CR3 命令をトラップし，ページテーブルエントリの書き換えの有無を検証する．ページテーブルの読み出し時に上記の検証作業を行うことで，ページテーブルエントリが書き換わっていないことを保証し，メモリの安全性を担保する．

ページテーブルの読み込みは頻繁に発生する．例えばプロセスの切り替わる際には必ずページテーブルが切り替わる．そのため，この保護の段階でコストがかかってしまうと新たなシステムのオーバーヘッドになりうる．

そこで FastVMIX では，保護する領域を 1GB Huge Page

に集約させることで，保護するエントリ数を大幅に削減する．ページテーブルは通常 4 レベルページングを採用しており，1GB Huge Page は PDPE（レベル 3），通常の 4KB ページは PTE（レベル 1）に該当する．4 レベルページングにおいてあるレベルのエントリを保護するためには，そのページよりも上位のレベルのエントリをすべて保護する必要がある．これは，上位のレベルのエントリが書き換えられた場合，下位のレベルのエントリを正しく参照できなくなるからである．そのため，FastVMIX では保護する領域を 1GB Huge Page に集約することで保護対象のエントリ数を減らし，ページテーブルの検証に必要なコストを削減する．

具体的には，FastVMIX では MOV-TO-CR3 の命令をトラップして 1GB Huge Page のエントリを 2 つ検証する．2 つの 1GB Huge Page は，それぞれ，前節で述べた PIC バイナリのコード領域とログ用のバッファ領域に相当する．FastVMIX は，これらが既定値以外を示した場合はシステムへの攻撃と判断する．

なお，1GB Huge Page の確保は，フロントエンドコマンドを通じて，Hugetlbfs を用いてユーザ空間で行う．確保された 1GB Huge Page は，後述するカーネルモジュールを用いて PIC バイナリのロードとカーネル空間へのマッピングを行う．

4.2.2 EPTP Switching を用いたメモリマップの動的変更による保護

図 2 に示すように，FastVMIX では通常用と解析用の 2 種類の EPT を用意する．通常用の EPT では，PIC バイナリのコードがマップされておらず，またログ用のバッファはリードオンリーになっている．一方，解析用の EPT では，解析用コードが実行可能，ログ用のバッファが読み書き可能になっている．また解析用の EPT では通常のコード領域がリードオンリーに設定されてある．これはコード再利用攻撃などを防ぐ意図がある．

これにより，通常時は解析用コードを読むことができず，バッファは読み込みしかできない．また，解析時は解析コードを実行可能になり，バッファは読み書き可能になる．このようにメモリ領域の権限を動的に切り替えることにより，マルウェアによる解析コードの検出とメモリに対する破壊攻撃を防ぎ，メモリの安全性を担保する．

メモリマップの動的変更には EPTP Switching を用いた．EPTP Switching は VMFUNC 命令の 0 番に CPU レベルで実装された機能であり，EPT をコンテキストスイッチなしで高速に切り替える命令である．VMFUNC 命令は，各レジスタに VMFUNC 番号と引数をセットすることで，ユーザ空間から発行できる．EPTP Switching を行う場合は，VMFUNC 命令の引数として EPT 配列のインデックスを設定する．

上述した機能は，本来 BitVisor がコアごとに 1 つのみ持つ EPT に解析用の EPT を追加することで実装した．システムが初期化された時にすべての EPT も初期化され，上述したマッピングを行うように EPT のエントリを作成する．

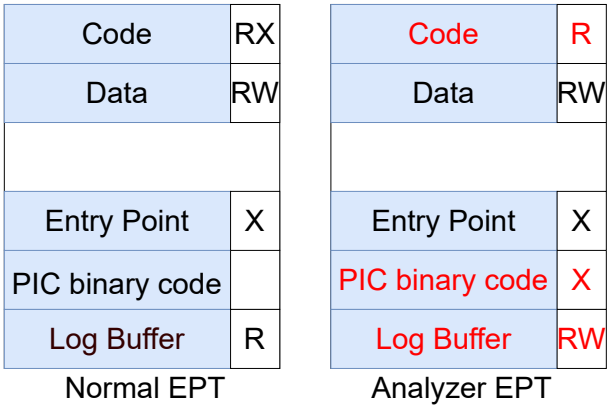

図 2 メモリマップの動的変更

4.2.3 MSR 保護

システムコールをトラップするために，FastVMIX ではシステムコールエントリポイントが PIC バイナリのフックコード領域の先頭アドレスとなっている．上記のシステムコールエントリポイントが攻撃者によって書き換えられるのを防ぐために，MSR の保護機能を追加する．MSR で保護する領域は，具体的には IA32_LSTAR（システムコールエントリポイント）である．

MSR 保護機能の追加により，上記の MSR の読み込み時にはデフォルトのシステムコールエントリポイントのアドレスが返される．これにより，マルウェアに対して FastVMIX がインストールされていることを不可視にしている．

また MSR 保護機能により，デフォルトのシステムコールエントリポイントアドレスと PIC バイナリのフックコード領域以外は上記の MSR への書き込みを受け付けないようにする．これにより，システムコールエントリポイントが任意の悪意のあるアドレスで上書きされることを防ぐ．

MSR のトラップは，VMCS（VM の設定項目を保持する構造体）の MSR トラップのフラグを有効化することで行う．

4.3 カーネルモジュール

カーネルモジュールはカーネル内の情報を入手したり，PIC バイナリをロードしたりなど特権的な振る舞いをするためのプログラムである．カーネルモジュールでは以下の機能を提供する．

- ユーザ空間からの ioctl を受け付けるデバイスの挿入
- 1GB Huge Page のカーネル内へのマッピング
- オリジナルのコンテキスト情報の保存
- カーネル内のオフセット情報の取得
- 1GB Huge Page への PIC バイナリのロード

- ゲストのシステムコールエントリポイントの変更
- ゲストのシステムコールエントリポイントの復元
- 1GB Huge Page にあるログ用バッファの読み出し
- ハイパーバイザの初期化

次にこれらの各機能を説明する．

ユーザ空間からの ioctl を受け付けるデバイスの挿入
カーネルモジュール機能の呼び出しは ioctl 経由で行う．カーネルモジュールはランダムな名前をつけてデバイスをカーネル内に挿入する．これは特定のデバイス名からシステムが発見されてしまうのを防ぐためである．デバイスはユーザ空間から ioctl 経由でコマンドを受け取り，サービスを提供する．提供するサービスは以下の通りである．

- カーネル内のオフセット情報を取得するサービス
- 1GB Huge Page に PIC バイナリをロードさせるサービス
- ゲストのシステムコールのエントリポイントを書き換えるサービス
- ゲストのシステムコールのエントリポイントを元に戻すサービス
- 1GB Huge Page にあるログ用のバッファを読み出すサービス
- ハイパーバイザを初期化するサービス

1GB Huge Page のカーネル内へのマッピング　本機能はユーザ空間で確保しマッピングした 1GB Huge Page を，カーネル内にマッピングする機能である．これによりカーネル空間の中に 1GB Huge Page が完全に固定されることになる．

オリジナルのコンテキスト情報の保存　オリジナルのシステムコールエントリポイントのアドレスと，ユーザ空間でのスタックポインタの値を保存する領域を PER_CPU_SECTION（各 CPU コアにマッピングされる特別な領域．GS レジスタベースでアクセスされる）に保存する機能である．保存した情報はカーネルモジュールだけでなく PIC バイナリも参照する．これらの情報はコアごとに異なる可能性があるため，PER_CPU_SECTION に保存する．

カーネル内のオフセット情報の取得　4.1 節で述べた PIC バイナリはカーネル内の変数や関数を使用する．そのために必要なアドレスのオフセット情報を取得する機能である．主に PER_CPU_SECTION にある，ユーザ空間のスタックアドレスの保存領域，カーネルのスタックアドレスの保存領域，オリジナルのシステムコールアドレスの保存領域の情報を取得する．

1GB Huge Page への PIC バイナリのロード　1GB Huge Page の中に PIC バイナリをロードする機能である．これは kernel_read_file_from_path 関数で与えられたパスからバイナリをカーネルの中にロードする．ロードしたバイナリは普通のページに保存されているので，memcpy 関数を用いてこれを直接 1GB Huge Page の中にコピーする．なお，後述するように，ハイパーバイザの初期化を行うまでは，カーネルモジュールは Huge Page の中身を直接読み書き実行する権限を有している．

ゲストのシステムコールエントリポイントの変更　この機能はシステムコールが発生したときに，PIC バイナリ内のフック関数に処理が飛ぶようにエントリポイントのアドレスを書き換える機能である．システムコールのエントリポイントとして，INT0x80（割り込み），sysenter 命令，syscall 命令の大きく 3 つが挙げられるが，今回は syscall 命令のみに対応した．syscall 命令のエントリポイントは MSR（Model-specific register，CPU コアにある特別な設定領域）の IA32_LSTAR（0xC0000082）に格納されている．今回はこれを wrmsr 命令で直接上書きしてエントリポイントを書き換えた．

ゲストのシステムコールエントリポイントの復元　この機能はシステムコールハンドラのアドレスを元のエントリポイントのアドレスに切り替える働きを行う．4.2.3 項で述べたように MSR への書き込みと読み込みはハイパーバイザによって保護されているが，特別に正規の値の書き込み（元のエントリポイントのアドレスと PIC バイナリ内のフック関数の値）は許可されているため問題なく上書きができる．

1GB Huge Page にあるログ用バッファの読み出し　ログ用のバッファは 1GB Huge Page に存在し，カーネル空間にマッピングされている．そのためユーザ空間からのリクエストを受付け，ログを読み出す機能を実装している．

ハイパーバイザの初期化　ハイパーバイザを初期化することで FastVMIX の完全な動作が始まることになる．初期化のためには，1GB Huge Page の情報，元のシステムコールのエントリポイントアドレス，PIC バイナリの情報を hypercall（ハイパーバイザの機能を呼び出す特殊な命令．ユーザ空間でカーネルの機能を呼び出すシステムコールと似た機能）経由で送信してから，もう一度初期化のための hypercall を呼び出す．初期化が行われると後述するセキュリティ機能が働き，システムは保護される．

5. 評価

5.1 評価方法

本章では**表 1** に示す 4 つのモデルを評価する．すなわち，仮想環境を使用せずに Linux を動作させた場合である **native**，オリジナルの BitVisor 上で上記の Linux を動作させた場合である **BitVisor**，Out-of-the-box 方式のバイナリ解析システム上で上記の Linux を動作させた場合であ

表 1 評価モデル

名前	説明
native	通常の Linux
BitVisor	オリジナルの BitVisor
DRAKVUF	先行研究 [5]
FastVMIX	提案手法

表 2 自作のベンチマークの実行時間

	ベンチマーク実行時間 [sec]			
	native	BitVisor	DRAKVUF	FastVMIX
1M 回	0.051	0.055	75.049	0.275
10M 回	0.487	0.503	757.362	2.789

る **DRAKVUF**，提案システム上で上記の Linux を動作させた場合である **FastVMIX** の 4 つである．上記 4 つのモデルのうち，**native** と **BitVisor** はバイナリ解析機能を有していない点に注意されたい．これらのモデルは提案手法の性能オーバーヘッドを明らかにする目的で評価した．

評価には自作のベンチマークと LMBench[8] を使用した．自作のベンチマークは，システムコールに要する時間を計測することを目的としており，getpid システムコールを 1M 回または 10M 回繰り返し実行する．LMBench は UNIX 向けのレイテンシ，バンド幅を計測するベンチマークである．

評価には Intel CPU Core i7-9700（8 コア，3GHz）を使用した．メモリは 12GB である．各評価モデルで使用する Linux カーネルのバージョンは 4.15 に統一した．

5.2 評価結果

5.2.1 自作のベンチマーク

自作のベンチマークによる評価結果を**表 2** に示した．表より，**BitVisor** と **native** の処理時間はほぼ等しい結果になった．一方，**FastVMIX** は **native** に比べておよそ 5.73 倍遅かったものの，**DRAKVUF** に比べて 271.55 倍速い結果となった．

5.2.2 LMBench

LMBench で得られた結果の内，特に変化の大きいものを**表 3**~**7** に示す．レイテンシとバンド幅の大きく 2 つに分かれる．

システムコールレイテンシ 表 3 から表 5 に示すように，おおよそ自作ベンチマークと同じような結果になった．表 3 では，システムコールレイテンシが示されている．**FastVMIX** は **DRAKVUF** に比べ，最大 337 倍速かった．

表 4 では，ファイルディスクリプタへの select システムコールのレイテンシが示されている．これも同じような傾向がみられる．**FastVMIX** は **DRAKVUF** に比べ，最大 237 倍速かった．

表 5 では，ネットワークファイルディスクリプタへの select システムコールのレイテンシが示されている．これも同じような傾向が示されている．**FastVMIX** は **DRAKVUF** に比べ，最大 203 倍速かった．

表 3 システムコールレイテンシ

	システムコールレイテンシ [ms]			
	native	BitVisor	DRAKVUF	FastVMIX
read	0.1179	0.1181	94.203	0.30674
write	0.089	0.0889	93.0164	0.276
stat	0.3319	0.3316	82.9851	0.5347
fstat	0.1178	0.1181	84.9844	0.294
open/close	0.1178	0.1181	84.9844	0.294

表 4 ファイルディスクリプタへの slelect システムコールレイテンシ

	slelect システムコールレイテンシ [ms]			
	native	BitVisor	DRAKVUF	FastVMIX
10 回	0.1822	0.1804	85.3676	0.3595
100 回	0.7656	0.7645	81.8286	0.9411
250 回	1.7192	1.7231	86.2222	1.8735
500 回	3.3696	3.3696	85.9839	3.525

表 5 ネットワークファイルディスクリプタへの slelect システムコールレイテンシ

	slelect システムコールレイテンシ [ms]			
	native	BitVisor	DRAKVUF	FastVMIX
10 回	0.2300	0.2319	84.5077	0.4157
100 回	1.9475	1.9393	86.2667	2.0886
250 回	4.7659	4.7767	94.4912	4.8868
500 回	9.6017	9.5448	104.7843	9.5393

表 6 ネットワークレイテンシ

	ネットワークレイテンシ [ms]			
	native	BitVisor	DRAKVUF	FastVMIX
TCP	6.5244	6.6740	337.2004	13.3315
UDP	5.3639	5.3582	267.4606	11.8572
TCP/IP	8.4030	8.4083	503.0000	10.3250

プロセス間通信 ローカルホストへのネットワークによるプロセス間通信のレイテンシを計測した．測定結果を表 6 に示す．プロセス間通信レイテンシについてもシステムコールレイテンシなどと同じ傾向がみられるものの，各評価モデル間の速度差はシステムコールレイテンシに比べて小さい結果となった．

ソケットによるプロセス間通信待ち時間とパイプによるプロセス間通信待ち時間に関してバンド幅測定を行った．測定結果を表 7 に示す．ソケットを用いた通信では，**native**, **FastVMIX**，**BitVisor** がほぼ同じバンド幅に対して，**DRAKVUF** はかなり小さく，**native** に比べ，45.56 倍，FastVMIX に比べ，42.66 倍遅かった．一方パイプに関しては，システムコールレイテンシに近い傾向がみられた．**native** が一番大きい値を示し，**BitVisor** がそれに近い値を示す．

表 7　ソケットとパイプによるプロセス間通信のバンド幅

プロセス間通信のバンド幅 [MB/sec]				
	native	BitVisor	DRAKVUF	FastVMIX
AF_UNIX	17631	16492	387	16505
パイプ	7132	6676	388	5695

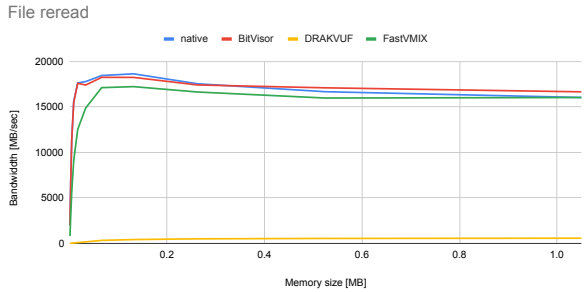

図 3　ファイル読み込みを繰り返した時の平均バンド幅

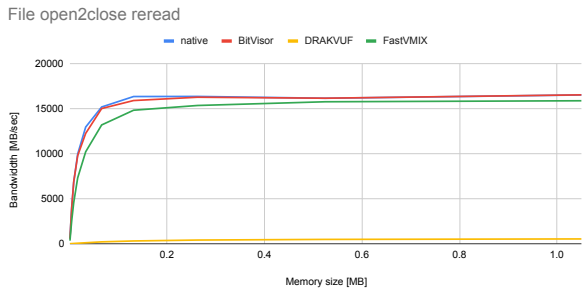

図 4　ファイルの読み込み，開く・閉じるを繰り返したときの平均バンド幅

FastVMIX はこれらにくらべ僅かに小さい値を示し (**native** に比べて 1.25 倍遅かった)，**DRAKVUF** は **native** に比べ 18.38 倍遅かった．

メモリ　さまざまな種類のメモリアクセスを行った際の平均メモリバンド幅を**図 3**〜**9** に示す．

ファイルの読み書き（図 3 と図 4）に関しては，**native** と **BitVisor** がほぼ同等で，**FastVMIX** がそれに続く．また，**DRAKVUF** に関してはこれら 3 つに比べて大きく劣っていた．特に，図 6 では，システムコールレイテンシの評価結果と同じ傾向がみられた．これは mmap，open，read，close，munmap を繰り返したときの結果であり，システムコールのレイテンシが大きく影響を与えていることが考えられる．

mmap したときの平均メモリバンド幅（図 5）は，全ての評価モデルがほぼ同じ値を示していた．これはシステムコールレイテンシの及ぼす影響が少ないことが考えられる．

bcopy を繰り返した場合（図 9）は，グラフの形が大きく異なる結果となった．**native**，**FastVMIX**，**BitVisor** は一度立ち上がるがすぐにバンド幅は下がる．一方 **DRAKVUF** は，立ち上がるとそのまま収束していく．しかし，最初に示す値は大きく異なるものの，収束する値はほぼ等しい．これはシステムコールレイテンシよりも bcopy の処理が支配的で，コピーサイズが大きくなるとこれが顕著になり，差が小さくなるのだと考えられる．

メモリの読み書きはすべてのシステムで似た傾向，数値が得られた．これはシステムコールが発生しないためである．

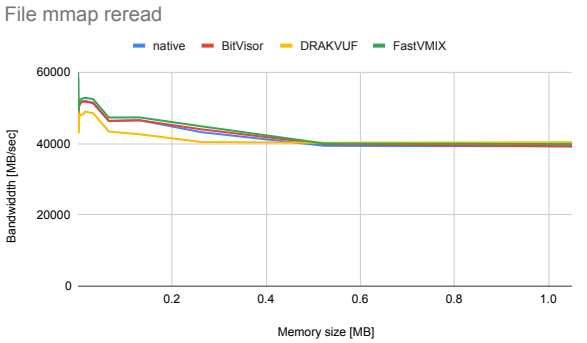

図 5　mmap してアクセスしたときの平均バンド幅

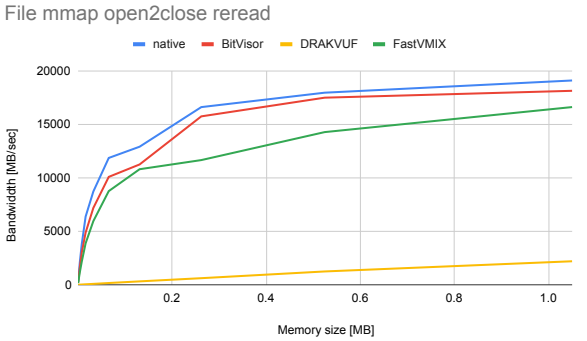

図 6　mmap して読み込み，開く・閉じるしたときの平均バンド幅

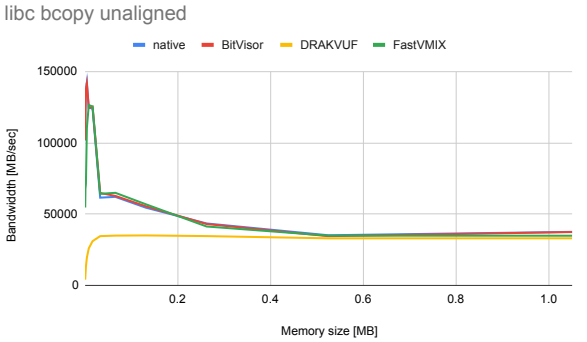

図 7　bcopy を繰り返ししたときの平均バンド幅

6. おわりに

本論文では高速かつ安全に VMI が行えるシステム FastVMIX を提案，評価した．FastVMIX は高速な VMI を行うために In-the-box 方式をとり，そこで生じるセキュリティリスクからシステムを EPTP Switching と HugePage

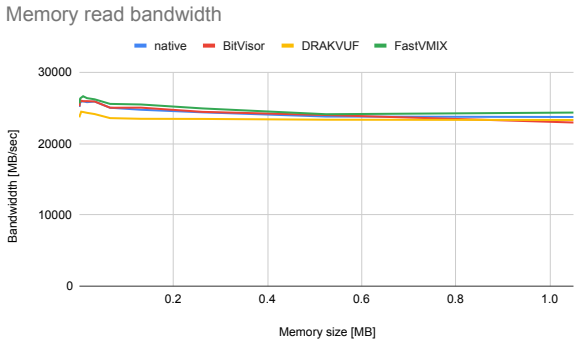

図 8 メモリ読み込みのバンド幅

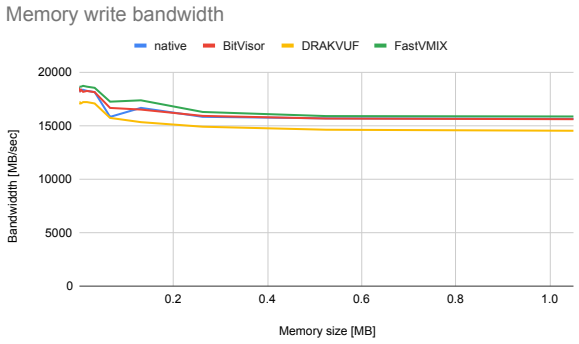

図 9 メモリ書き込みのバンド幅

を用いた高速なページテーブル保護にて防御した．評価では自作のベンチマークと LMBench を用いて提案システムが先行研究に比べて優位性を持つことを定量的に示した．

今後の展望として，Windows などの他の OS に対応させること，PIC バイナリの高機能化を考えている．特に PIC バイナリの高機能化については，早急に着手し，ファイルシステムやネットワークの解析が効果的に行えるようにする．

参考文献

[1] Abdelraoof, A., Taubmann, B., Dangl, T. and Reiser, H. P.: Introspect Virtual Machines Like It Is the Linux Kernel!, *Proceedings of the 18th Detection of Intrusions and Malware & Vulnerability Assessment*, DIMVA 2021, pp. 258–277 (2021).

[2] Dangl, T., Taubmann, B. and Reiser, H. P.: RapidVMI: Fast and Multi-Core Aware Active Virtual Machine Introspection, *Proceedings of the the 16th International Conference on Availability, Reliability and Security*, ARES 2021, (online), DOI: 10.1145/3465481.3465752 (2021).

[3] Dinaburg, A., Royal, P., Sharif, M. and Lee, W.: Ether: Malware Analysis via Hardware Virtualization Extensions, *Proceedings of the 15th ACM Conference on Computer and Communications Security*, CCS '08, pp. 51–62 (online), DOI: 10.1145/1455770.1455779 (2008).

[4] Garfinkel, T., Rosenblum, M. et al.: A Virtual Machine Introspection Based Architecture for Intrusion Detection, *Proceedings of the Network and Distributed System Security Symposium*, pp. 191–206 (2003).

[5] Lengyel, T. K., Maresca, S., Payne, B. D., Webster, G. D., Vogl, S. and Kiayias, A.: Scalability, Fidelity and Stealth in the DRAKVUF Dynamic Malware Analysis System, *Proceedings of the 30th Annual Computer Security Applications Conference*, ACSAC '14, pp. 386–395 (online), DOI: 10.1145/2664243.2664252 (2014).

[6] Leon, R. S., Kiperberg, M., Zabag, A. A. L. and Zaidenberg, N. J.: Hypervisor-assisted dynamic malware analysis, *Cybersecurity*, Vol. 4, No. 19 (online), DOI: 10.1186/s42400-021-00083-9 (2021).

[7] Luţaş, A., Sebestyen, G., Toşa, R. and Coleşa, A.: VE-VMI: High-Performance Virtual Machine Introspection Based on Virtualization Exception, *Proceedings of the 2021 20th International Symposium on Parallel and Distributed Computing (ISPDC)*, pp. 73–80 (online), DOI: 10.1109/ISPDC52870.2021.9521609 (2021).

[8] McVoy, L. and Staelin, C.: Lmbench: Portable Tools for Performance Analysis, *Proceedings of the 1996 Annual Conference on USENIX Annual Technical Conference*, pp. 279–294 (1996).

[9] Or-Meir, O., Nissim, N., Elovici, Y. and Rokach, L.: Dynamic Malware Analysis in the Modern Era—A State of the Art Survey, *ACM Computing Surveys*, Vol. 52, No. 5 (online), DOI: 10.1145/3329786 (2019).

[10] Qin, J., Shi, B. and Li, B.: NEM: A NEW In-VM Monitoring with High Efficiency and Strong Isolation, *Proceedings of the 2nd International Conference on Smart Computing and Communication*, pp. 396–405 (online), DOI: 10.1007/978-3-319-73830-7_39 (2018).

[11] Sharif, M. I., Lee, W., Cui, W. and Lanzi, A.: Secure In-VM Monitoring Using Hardware Virtualization, *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pp. 477–487 (online), DOI: 10.1145/1653662.1653720 (2009).

[12] Shi, B., Cui, L., Li, B., Liu, X., Hao, Z. and Shen, H.: ShadowMonitor: An Effective In-VM Monitoring Framework with Hardware-Enforced Isolation, *Proceedings of the 21st International Symposium on Research in Attacks, Intrusions, and Defenses*, pp. 670–690 (online), DOI: 10.1007/978-3-030-00470-5_31 (2018).

[13] 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 '09, pp. 121–130 (online), DOI: 10.1145/1508293.1508311 (2009).