Vol.2012-OS-122 No.9
2012/8/1

VMMを用いたマルウェア検出システムのための シグネチャデータ更新機能とメモリデータ検査機能

河崎 雄大[1]　大山 恵弘[1]

概要：仮想マシンモニタ（VMM）を用いてセキュリティを向上するアプローチとして，ストレージ暗号化などのセキュリティ機能を提供する BitVisor や，BitVisor を拡張してマルウェアの検出機能を持たせた BVMD が提案されている．BVMD では，入出力データをマルウェアのシグネチャと照合することにより，マルウェアを検出する．しかし，BVMD は，動的なシグネチャデータの更新や，メモリ上のみに存在するマルウェアの検出はできない．そこで本研究では，BVMD のためのシグネチャデータ更新機能とメモリ上のマルウェアを検出する機能を提案する．シグネチャデータ更新機能に関しては，ゲスト OS 上にシグネチャの更新データを置き，それを VMM に読み込ませる．ただし，ゲスト OS 上に置いたデータは攻撃者によって改ざんされる心配があるので電子署名によってデータの整合性をとる．メモリ上のマルウェアを検出する機能に関しては，検査するメモリのアドレス範囲をゲスト OS のユーザが指定し，そのメモリの内容を VMM がシグネチャと照合する．我々はこれらの機能で BVMD を拡張し，シグネチャデータ更新とマルウェア検出の実験を行った．

Signature Data Update and On-Memory Data Inspection for a Hypervisor-Based Malware Detection System

Abstract: Much literature has been published on approaches for security enhancement that are based on a virtual machine monitor (VMM). Examples of them are BitVisor and BVMD; BitVisor provides many security facilities including storage encryption, and BVMD is an extension to BitVisor that provides a malware detection facility by matching I/O data with malware signatures. Unfortunately, BVMD does not support dynamic update of malware signatures or detection of malware that is stored only on memory. In this work, we propose two mechanisms for BVMD: a mechanism for signature update and a mechanism for detecting on-memory malware. The user of the first mechanism places new signature data on the guest OS and requests the VMM to read them. Since the signature data may be modified by an attacker, the mechanism guarantees its integrity by using electronic signatures. The user of the second mechanism specifies the address range of the memory that the user requests to inspect, and then the VMM matches the content of the memory with malware signatures. We extended BVMD with these mechanisms and conducted experiments of signature data update and malware detection.

1. はじめに

近年，コンピュータの利用者の増加に伴い多くのマルウェアが発見されている．マルウェアの多くはユーザの不注意をついたり，プログラムの脆弱性をついたりして攻撃を行い，マルウェアによる被害は多く確認されている．このようなマルウェアによる被害を防ぐ方法としてアンチウイルスの利用がある．

アンチウイルスの利用は効果的であり，現在広く普及している．しかし，OS の管理者権限をもったユーザやマルウェアによってアンチウイルスの無効化ができるといった問題もある．この問題点を防ぐ方法として，OS よりも下の層で動いている仮想マシンモニタ (VMM) でセキュリティ処理を行うというものがある．その方法に基づくシステムの 1 つとして BitVisor [1] がある．BitVisor はストレージの暗号化や Virtual Private Network (VPN) の構築などを行う機能を備えた VMM である．また，BitVisor は準パススルー型という方式で作られた VMM であり，実行時間のオーバーヘッドや Trusted Computing Base (TCB) を小さく抑えた設計となっている．しかし，ストレージの暗号

[1] 電気通信大学
The University of Electro-Communications

化などのセキュリティ機能は備えているもののマルウェアを検出する機能までは備えていない．

BitVisor にマルウェアを検出する機能を付け加えたシステムとしては BVMD [2] がある．BVMD は読み書きされるストレージデータに対してマルウェアの検出を行う．しかし，BVMD には，以下のような 2 つの問題が存在する．第 1 の問題は，マルウェアのシグネチャデータを動的に更新できないことである．シグネチャデータは BVMD のビルド時に与え，BVMD のバイナリに埋め込まなければならない．これは BVMD が OS 上ではなくハードウェア上で直接動いていることからくる制約である．近年，新種のマルウェアが次々と発見されており，新しいシグネチャデータをすぐに組み込めることは不可欠である．第 2 の問題は，現在の BVMD では読み書きされるストレージデータに対してのみマルウェアの検出を行うことができ，メモリのみに存在するマルウェアの検出はできないことである．攻撃のすべてを被害にあった PC のメモリ上で行い，ストレージにファイルを作成しないマルウェアも存在するため，メモリ上のマルウェア検出は不可欠である．

本研究では，BVMD をマルウェアのシグネチャデータをゲスト OS を動作させたまま更新する機能とメモリのみに存在するマルウェアを検出する機能で拡張したシステムを提案する．マルウェアのシグネチャデータを更新する機能に関しては，新しいマルウェアのシグネチャデータを一時的にゲスト OS 上に置き，管理者の指示によりそのデータを VMM に読み込ませる．しかし，この方法を単純に用いるとゲスト OS がマルウェアに乗っ取られている場合にシグネチャデータが改ざんされるといった問題が生じる．そこで，本研究では，シグネチャデータに RSA 暗号 [3] によって作成された電子署名を加え，データの整合性を保つといった手法を用いる．そして，シグネチャデータの更新の際は，VMM が電子署名をもとにシグネチャデータの整合性を確認し，データの内容が正しいときのみシグネチャデータの更新を行うようにする．また，メモリのみに存在するマルウェアを検出する機能に関しては，ゲスト OS の管理者がゲスト OS のメモリ上の範囲を指定し，VMM がその範囲のメモリに対してシグネチャマッチングを行うといった手法を用いる．

本論文の構成は以下のとおりである．2 章では既存システムの BitVisor と BVMD の概要を述べ，3 章では提案システムの設計を述べる．4 章では提案システムの実装を述べ，5 章では本システムで新たに生じる攻撃などの議論を述べる．6 章では本システムの評価を述べ，7 章では関連研究について述べる．そして，8 章では本研究のまとめと今後の課題を述べる．

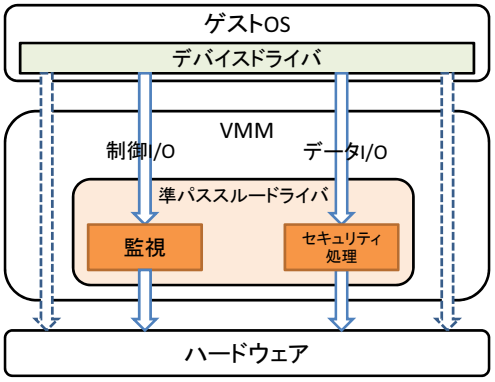

図 1 準パススルー型 VMM BitVisor の構成
Fig. 1 Structure of a parapass-through VMM BitVisor

2. 既存システム

2.1 BitVisor

BitVisor は，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，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 (通過) させつつ，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って，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．また，BitVisor は Intel Virtualization Technology (Intel VT) 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し，ゲスト OS が発行する制御 I/O とデータ I/O を捕捉する．ここで，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 I/O である．また，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，更新を行うことによって，ゲスト OS から転送されるデータに対して暗号化などのセキュリティ処理を実現している．

2.2 BVMD

BVMD は，BitVisor に対して機能拡張を行い，読み書きされるストレージデータに対してマルウェアの検出を行う機能を備えたシステムである．BVMD は BitVisor の準パススルードライバに対して拡張を行い、捕捉したデータ I/O に対してシグネチャマッチングを行ってマルウェア検

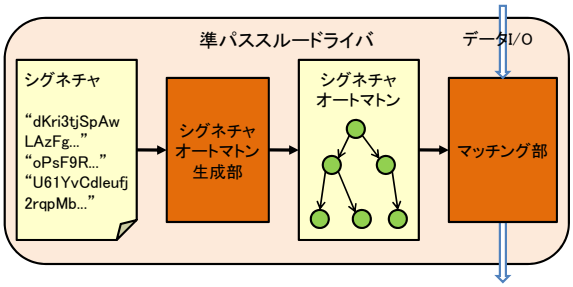

図 2 BVMD で拡張された準パススルードライバの構成
Fig. 2 Structure of an extended parapass-through driver in BVMD

```
virus1=0f3be373e986a78cddbd0f150f4764a273d78a70dd0
9bcdd0296cfb3102635562ab5c978e0a4ad9d2811830f60844
285ab4fcb8b9e9e9b07e
virus2=373e986a78cddbd0f150f4764a273d78a70dd09bcdd
0296cfb3102635562ab5c

40d6ce3e7d50953bb11234c3a73618d45eac77747c146cf550
e55ee62dc22bf8f899eef8790b92bd33fee00c330fdebdaf06
4f21668586d2b60a92c8b9d9773e
```

図 3 シグネチャファイルのフォーマット
Fig. 3 The format of a signature file

出を行っている．

BVMD で拡張された準パススルードライバの構成を**図 2** に示す．BVMD 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．シグネチャオートマトン生成部では，マルウェアのシグネチャデータの集合を受け取り，マッチング処理に適したメモリ上のデータ構造であるシグネチャオートマトンを生成する．シグネチャオートマトンを生成する際は Aho-Corasick 法と呼ばれるアルゴリズムを用いている．マッチング部では，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でマッチング処理を行う．

マルウェアのシグネチャデータは ClamAV[4] が提供したデータを加工して使用している．準パススルー型 VMM である BitVisor は VMM の下で動く OS (ホスト OS) を持たないので，BVMD では VMM の実行型バイナリにマルウェアのシグネチャデータを埋め込んでいる．

3. 提案システムの設計

3.1 前提条件

本研究では，CPU の仮想化支援機構である Intel VT [5] が備わった環境を想定している．Intel VT ではゲスト OS が処理を行う際の VMX non-root operation と VMM が処理を行う際の VMX root operation が用意されている．VMM 上でゲスト OS を動かす場合，基本的には VMX non-root operation でゲスト OS に命令を実行させるが，特別な命令を実行する際は VMX root operation に移行し VMM が命令を実行するように設計されている．Intel VT では，VMX root operation に移行する VM exit を発生させる特別な命令として VMCALL という命令が用意されている．本研究では VMCALL 命令を用いて実装を行う．

3.2 設計

本研究が提案するシステムは，BVMD を 2 つの機能で拡張したものである．第 1 の機能は，マルウェアのシグネチャデータを更新する機能である．第 2 の機能は，ゲスト OS のメモリに対してシグネチャマッチングを行う機能である．まず，マルウェアのシグネチャデータを更新する機能を説明する．本システムのユーザは，まず，新しいシグネチャデータのファイルをシグネチャデータ提供者のサーバからダウンロードする．次に，シグネチャデータ提供者によって作られたシグネチャデータ更新プログラムをゲスト OS の上で実行する．このプログラムは，シグネチャデータをファイルから読み込んでメモリ上に置き，VMM に更新要求をする．

ただし，ゲスト OS がマルウェアなどによって乗っ取られた場合，シグネチャデータが改ざんされるといった問題が考えられる．そのため，マルウェアのシグネチャと電子署名を組にしたファイルを入れたファイルをシグネチャデータの更新ファイルとする．更新ファイルのフォーマットを**図 3** に示す．更新ファイルの前半部分には，マルウェアの名前とそのシグネチャを 16 進表記したものを “=” で結んだもののリストが書かれている．空行で区切られた後半部分には，電子署名を 16 進表記したデータが入っている．電子署名は，公開鍵暗号方式である RSA 暗号で作成されたシグネチャデータ提供者の秘密鍵を用いて作成する．電子署名を用いて整合性を確認する際に使うシグネチャデータ提供者の公開鍵は，VMM の実行型バイナリに埋め込む．

次に，VMM による更新処理を説明する．ゲスト OS からシグネチャデータの更新要求を受け取った VMM はゲスト OS から新しいシグネチャデータと電子署名を受け取る．次に，VMM が受け取ったシグネチャデータが正しいかどうかの確認を，受け取った電子署名と VMM の中に埋め込んである公開鍵をもとに行う．そして，その結果，データの整合性が確認されなかった場合は警告メッセージを通知し，処理をゲスト OS へ戻す．データの整合性が確認された場合は，新しいシグネチャデータを BVMD のシグネチャオートマトンに加える．最後に，シグネチャデータの更新が成功したことをゲスト OS に通知し，処理をゲスト OS へと戻す．

次に，ゲスト OS のメモリに対してシグネチャのマッチングを行う機能を説明する．ユーザは，提案システムと共に配布されたメモリ上のシグネチャマッチングプログラムをゲスト OS 上で実行する．このプログラムは，ユーザが指定したアドレスの範囲の情報をメモリ上にのせ，シグネ

チャマッチングを VMM に要求する．本来は，この機能は VMM が自動的に行うのが望ましいが現在はゲスト OS のユーザの協力を得て行う．

次に，ゲスト OS のメモリに対してシグネチャのマッチングを行う部分を説明する．本システムでは，まず，シグネチャマッチングの要求が来た場合，VMM がゲスト OS からマッチングする範囲の情報を受け取る．次に，受け取った範囲の情報をもとにゲスト OS のメモリ上のデータのシグネチャマッチングを行う．シグネチャマッチングを行う際は，BVMD のマッチング部を用いる．最後に，メモリ上のシグネチャマッチングが成功したかどうかの情報をゲスト OS に通知し，処理をゲスト OS へと戻す．この機能についても，本来は VMM が自動的に行うのが望ましいが，現在は実装の途中段階であるため，ゲスト OS 上のプロセスが自身のシグネチャマッチングを要求する方式になっている．現在の方式では，他プロセスのメモリ領域を検出することはできないが，カーネル領域を検出することはできる．これにより，カーネルレベル rootkit を検出できる．

4. 提案システムの実装

4.1 VMCALL 命令

この章では上述の 2 つの機能の実装について述べる．この 2 つの機能はゲスト OS 上で VMCALL 命令を実行することによって呼び出される．VMCALL 命令の実行はいわゆるハイパーコール命令の呼び出しとして利用される．VMCALL 命令を呼び出す関数を**図 4** に示す．図 4 の関数は，RAX，RBX，RCX，RDX レジスタに a，b，c，d の値を入れ，オペコードが “0F 01 C1” である VMCALL 命令を実行する関数である．この関数は，VMCALL 命令の返り値が入っている RAX レジスタの値を返り値として返す．本システムでは，RAX レジスタに入れられた番号の値によって機能の種類を区別する．また，各機能に渡すデータが格納されているアドレスを汎用レジスタに入れることによりゲスト OS から VMM にデータを送る．しかし，VMM がそのアドレスの内容をそのまま読み込もうとしても，そのアドレスはゲスト OS の仮想アドレスなのでうまくいかない．そのため，本システムでは，ページごとにゲスト OS の仮想アドレスをゲスト OS の物理アドレスに変換し，その物理アドレスで指定したメモリから VMM の仮想メモリにコピーを行っている．一連のメモリアドレス変換とデータコピーの様子を**図 5** に示す．この例は，5 つのページにまたがって連続したメモリのデータを VMM にコピーする例である．各ページの先頭アドレスであるゲスト OS の仮想アドレスをゲスト OS の物理アドレスに変換し，その物理アドレスで指定したメモリから VMM の仮想メモリにコピーを行っている．本システムでは，このような方法を用いて，VMM がゲスト OS の仮想アドレス空間からデータをコピーしている．

```
static int vmcall(unsigned long a, unsigned long b,
                  unsigned long c, unsigned long d){
  int ret;
  __asm__ __volatile__
    (".byte 0xf,0x1,0xc1"
     : "=a" (ret) : "a" (a), "b" (b), "c" (c), "d" (d),
     );
  return ret;
}
```

図 4 VMCALL 命令を呼び出す関数
Fig. 4 A function calling VMCALL instruction

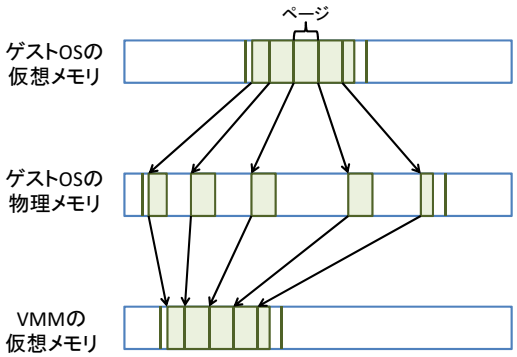

図 5 アドレス変換とデータのコピー
Fig. 5 Address translation and data copying

表 1 シグネチャデータ更新機能を呼び出す際のレジスタの値
Table 1 Values of the registers for updating signature data

レジスタ	レジスタの値
RAX	1
RBX	シグネチャデータが置かれたメモリ領域のアドレス
RCX	シグネチャデータの長さ
RDX	電子署名が置かれたメモリ領域のアドレス

4.2 マルウェアのシグネチャデータ更新機能

まず，マルウェアのシグネチャデータ更新機能を呼び出す際に実行する VMCALL 命令について述べる．本機能を呼び出す際にセットする各レジスタの値を**表 1** に示す．本機能に対しては 1 番の番号を割り振る．これらの適した値を適したレジスタに入れて VMCALL 命令を実行することでシグネチャデータの更新機能が呼び出される．

次に，VMM が受け取ったマルウェアのシグネチャデータの整合性を確認する処理についてその流れについて述べる．シグネチャデータの整合性を確認する方法を**図 6** に示す．本システムが扱っている電子署名は，マルウェアのシグネチャデータの MD5 ハッシュをシグネチャデータ提供者の秘密鍵で暗号化したものである．そのため，シグネチャデータの整合性の確認は，送られてきたシグネチャデータの MD5 ハッシュと電子署名をシグネチャデータ提供者の公開鍵で復号化したものを比較することにより実現

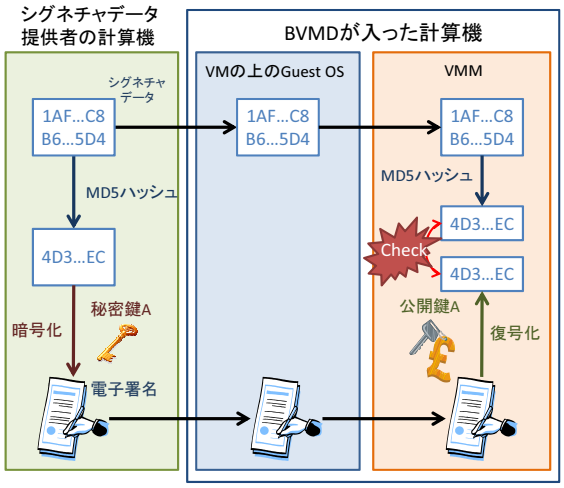

図 6 シグネチャデータの整合性を確認する方法
Fig. 6 How to verify the integrity of the signature data

表 2 メモリ上のデータに対するシグネチャマッチングの機能を呼び出す際のレジスタの値

Table 2 Values of the registers for matching signature data with on-memory data

レジスタ	レジスタの値
RAX	2
RBX	検査するメモリ領域の先頭アドレス
RCX	検査するメモリのサイズ

する．電子署名に関する処理を行うために，OpenSSL のライブラリを BVMD に組み込んでいる．

次に，新しいシグネチャデータを BVMD のシグネチャオートマトンに加える処理について述べる．この処理は上述の整合性の確認が成功した時のみ実行される．この処理では BVMD のシグネチャオートマトン生成部の機能を用い，Aho-Corasick 法によってシグネチャオートマトンに新たな端点や枝を追加していく．

4.3 ゲスト OS のメモリ上のシグネチャマッチング機能

ゲスト OS のメモリ上のシグネチャマッチング機能も VMCALL 命令によって呼び出される．本機能を呼び出す際にセットする各レジスタの値を**表 2** に示す．本機能に対しては 2 番の番号を割り振る．これらの適した値を適したレジスタに入れて VMCALL 命令を実行することでゲスト OS のメモリ上のデータに対するシグネチャマッチングの機能が呼び出される．

上記の機能が呼び出されると，VMM は，与えられた範囲のメモリの内容を VMM のメモリ領域にコピーする．そして，コピーしたものに対してシグネチャマッチングを行う．BVMD はシグネチャマッチングを行う関数を提供しているので，コピーした領域の情報をその関数に与えて実行することでシグネチャマッチングを実現する．

表 3 実験環境

Table 3 Platform of experiments

CPU	Intel Core 2 Quad Q9300 2.5GHz
メモリ	4GB
VMM	BitVisor 1.2
ゲスト OS	Ubuntu 12.04 32-bit, linux 3.2.0-25-generic-pae

5. 議論

5.1 新たな攻撃

本システムは更新するシグネチャデータをゲスト OS に置くので新たな攻撃が考えられる．まず，シグネチャデータを更新させない攻撃が考えられる．この攻撃は，攻撃者がゲスト OS に置いたシグネチャデータを改ざんし続けることによって更新をさせない攻撃である．この攻撃によってすでに VMM に存在しているシグネチャデータは被害にあわないが，新たにシグネチャデータを追加する作業はできなくなる．この攻撃を防ぐには，更新するシグネチャデータをゲスト OS に置かない手法が考えられ，今後，取り組んでいきたいと思う．

次に，シグネチャデータを昔のものに戻す攻撃が考えられる．この攻撃は，昔のシグネチャデータとその電子署名を攻撃者が記憶しておき，十分に時間がたった後、そのシグネチャデータをもとに更新を行う攻撃である．この攻撃で用いられるシグネチャデータと電子署名は正しいものなのでデータの整合性の確認だけでは防ぐことができない．そのため，この攻撃を防ぐにも，更新するシグネチャデータをゲスト OS に置かない手法が考えられ，今後，取り組んでいきたいと思う．

5.2 公開鍵の問題

本システムでは，シグネチャデータの整合性を確認する際に用いられるシグネチャデータ提供者の公開鍵が VMM のバイナリの中に埋め込まれており，動的に更新できないという新たな問題が考えられる．公開鍵を更新する機能を追加する場合，新たな認証が必要となる．現状では，公開鍵は VMM のバージョンアップなどで再コンパイルする場合に更新するという方法をとっている．

6. 評価

マルウェアのシグネチャデータ更新機能とメモリ上のシグネチャマッチング機能に対しての動作確認の実験と，シグネチャマッチング時間を計測する実験を行った．実験環境を**表 3** に示す．

6.1 提案機能の動作を確認する実験

まず，シグネチャデータ更新機能を用いて新たなシグネ

```
Adding signature
signature 1000 (len 32): new_virus1
Signature added!

Checking memory
address = 4007c1, len = 61530
*** Matched with signature 1000 ***
```

図 7 シグネチャデータの更新結果とメモリデータの検査の結果
Fig. 7 Results of updating signature data and checking memory data

表 4 メモリ上のシグネチャマッチングの測定結果 (秒)
Table 4 The amount of time taken for signature matching of on-memory data (second)

	1MB	5MB	10MB
500 個	0.360	1.690	3.345
1000 個	0.539	2.531	5.040

チャデータを追加し，追加したシグネチャにマッチするマルウェアも検出できるかどうかを確認する実験を行った．結果を**図 7** に示す．BVMD がシグネチャデータが追加されたことをシリアルポートのデバッグ用出力に出力している．マルウェア名 “new_virus1”，長さ 32B のシグネチャデータが加わり，それに対して 1000 番という番号が与えられていることがわかる．次に，新たに追加されたシグネチャにマッチするデータをゲスト OS 上のプロセスのメモリ上に書き込んだ上で，メモリ上のシグネチャマッチング機能を実行したところ，本システムがそのデータを検知した．アドレス 0x4007c1 から 61530B を検査するよう要求し，その結果，1000 番のシグネチャデータがマッチしたことがわかる．

6.2 メモリ上のシグネチャマッチング時間の計測実験

この実験は，メモリ上のデータのシグネチャマッチング機能にかかる実行時間を計測するものである．シグネチャの数は 500 個，1000 個と変化させて実験をした．シグネチャの長さは最大でも 187B である．また，シグネチャマッチングを行うメモリサイズは 1MB，5MB，10MB と変化させた．結果を**表 4** に示す．

シグネチャマッチングを行うメモリサイズに対して，実行時間がほぼ線形に増加することがわかった．また，マッチングに用いるシグネチャの数を 2 倍に増やしても，実行時間は 2 倍に達しないこともわかった．一般に Aho-Corasick 法では，パターンの増加に対する探索時間の増加は穏やかであり，この結果はそれを裏づけるものとなっている．

7. 関連研究

既存のアンチウイルスソフトウェアとして，Clam AntiVirus[4] がある．このソフトウェアでは，シグネチャデータが入ったパターンファイルを更新する際，公開鍵暗号によって作成された電子署名によってファイルの整合性を確認してから更新する．本システムでは，この更新方法を VMM によるマルウェア検出システムに対して適用した．

VMwatcher [6] は，VM 上で動作するゲスト OS のハードディスクやメモリ上の情報をホスト OS で動くアンチウイルスが検査することを可能にするシステムである．このシステムでは既存の VMM である VMware Server，QEMU，Xen，User-Mode Linux をサポートしている．このシステムではホスト OS 上で動くプログラムがメモリ上の情報を検査してアンチウイルスの機能を提供しているが，本システムでは VMM がメモリ上の情報を検査してアンチウイルスの機能を提供しているという点で異なる．

8. まとめと今後の課題

本研究では，BitVisor にマルウェアを検出する機能を備えたシステムである BVMD に対して，マルウェアのシグネチャデータを更新する機能やメモリのみに存在するマルウェアを検出する機能を提案した．提案機能に対する実行確認の実験を行った結果，提案機能によってマルウェアのシグネチャデータの追加とメモリ上のデータに対してのマルウェアの検出が確認できた．また，メモリ上のシグネチャマッチングにかかる時間を実験により明らかにした．

今後の課題としては，2 つが考えられる．まず，第 1 の課題は，VMM がゲスト OS のユーザを介在させずにシグネチャデータを更新することである．現在のシステムでは，ゲスト OS のユーザがゲスト OS に新たなマルウェアのシグネチャデータを置き，シグネチャデータの更新を VMM に要求するといった方法を用いている．しかし，この方法だとゲスト OS のユーザが処理を行わないといけないといった問題が存在する．そのため，今後は VMM が独自にサーバと通信を行ってシグネチャデータを更新するなどといった方法についても考えていきたい．第 2 の課題は，メモリ，ストレージ以外のデータに対してもマルウェアの検出を行うことである．具体的には，ネットワークパケットに対してもマルウェアの検出を行うといったことが考えられる．

謝辞 本研究は科研費（23700032）の助成を受けたものである．また，本研究の遂行にあたっては，東京大学の品川高廣氏，産業技術総合研究所の須崎有康氏の助言が非常に有益であった．

参考文献

[1] T. Shinagawa, H. Eiraku, K. Tanimoto, K. Omote, S. Hasegawa, T. Horie, M. Hirano, K. Kourai, Y. Oyama, E. Kawai, K. Kono, S. Chiba, Y. Shinjo and K. Kato: BitVisor: A Thin Hypervisor for Enforcing I/O Device Security, *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Envi-*

ronments (VEE 2009), pp. 121–130 (2009).

[2] Y. Oyama, T. T. D. Giang, Y. Chubachi, T. Shinagawa and K. Kato: Detecting Malware Signatures in a Thin Hypervisor, *Proceedings of the 27th Annual ACM Symposium on Applied Computing (SAC '12)*, pp. 1807–1814 (2012).

[3] R. L. Rivest, A. Shamir and L. Adleman: A method for obtaining digital signatures and public-key cryptosystems, *Communications of the ACM Volume 21, Issue 2*, pp. 120–126 (1978).

[4] Clam AntiVirus, 入手先 ⟨http://www.clamav.net/⟩.

[5] Intel 64 and IA-32 Architectures Software Developer Manuals 入手先 ⟨http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html⟩

[6] X. Jiang, X. Wang and D. Xu: 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).