Vol.2020-DPS-182 No.14
Vol.2020-CSEC-88 No.14
2020/3/12

準パススルー型ハイパーバイザーを用いた時系列メモリデータ取得機能の試作と評価

大森貴通[1]　稲垣怜[1]　平野学[1]　小林 良太郎[2]

概要：インターネットの急速な発展とともにマルウェアによる被害が増加している．マルウェアによる攻撃は企業活動や公共サービスに影響を与えることがあるため早期に検知する必要がある．これまでに我々の研究室ではストレージ装置のアクセスパターンを用いて，ランサムウェアの振る舞いを検知するシステムを開発してきた．しかし，ストレージアクセスパターンのみでは，似た振る舞いをする無害なプログラムを，ランサムウェアとして誤検知してしまう可能性があった．このため，ストレージ装置のアクセスパターンに加えて，RAM 上のメモリデータから得られる特徴量を用いることでランサムウェアの検知性能を向上できるのではないかと考えた．そこで本研究では準パススルー型ハイパーバイザーを用いて，ゲスト OS のユーザプロセスのメモリデータを取得する機能を提案する．メモリデータを取得する方法として以下の方法を実装した．まず，ゲスト OS 上でプロセスの仮想アドレスを物理アドレスに変換したのち，その物理アドレスをゲスト OS から VMCALL を用いてハイパーバイザーへ渡す．ハイパーバイザーではこの物理アドレスをもとに物理メモリを参照し，ページ単位でメモリデータを取得する．取得したメモリデータは，ゲスト OS から渡された仮想アドレスと時刻情報とともに解析サーバへ転送する．本稿では特にハイパーバイザーで RAM のメモリデータを取得する機能を検討するとともに，実装したメモリデータ取得機能の性能評価を報告する．

キーワード：マルウェア，ハイパーバイザー，仮想化，メモリフォレンジック

1. はじめに

コンピュータやインターネットが社会生活において不可欠なものになるにつれ，個人はもとより企業や国家を標的とするマルウェアの脅威が増加している．図 1 にドイツのインターネットセキュリティ研究機関である AVTEST が公開したデータ[1]を示す．マルウェアがここ 10 年で急増していることが分かる．AVTEST は 1 日あたり 35 万個を超える新しいマルウェアが出現していると報告している．マルウェアの作成者は既知のマルウェアに対し，暗号化や難読化を施すパッカーを用いることで容易に亜種マルウェアを作成できる．亜種マルウェアとは，元々存在するマルウェアに似た挙動を示すマルウェアであり，近年のマルウェア急増の原因のひとつになっている．

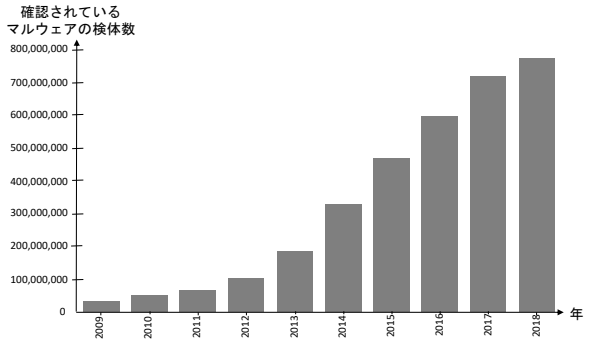

図 1　AVTEST が確認したマルウェアの検体数の推移

マルウェアの検知方法はその特徴から，シグネチャ型と振る舞い型の二つに大別できる．シグネチャ型で用いられるシグネチャとは特定のマルウェア検体に共通する一続きのバイト列[2]である．実際はバイト列を照合するのに時間がかかるため，シグネチャ型ではファイル単位のハッシュ値が用いられる．シグネチャ型で検知を行う場合，既知のマルウェアを確実に検知できる利点がある．しかし，マルウェアからシグネチャを生成する手法上，未知や亜種のマルウェアにすぐに対応できない欠点がある．このような特徴を持つシグネチャ型に対して，振る舞い型ではマルウェアのコンピュータ上での特徴的な挙動から検知を試みる．既知のマルウェアと同じ，または似た挙動を持つマルウェアを検知できるため，シグネチャ型よりも一見良い方法にみえるが，振る舞い型には誤検知の多さが問題点として挙げられる．シグネチャ型では誤検出がほとんどなく正確にマルウェアを検知できる．実際のマルウェア検知はどちらか一方の方式だけを使うのではなく，既存のマルウェアはシグネチャ型で確実に検知し，新種や亜種のマルウェアは振る舞い型でシグネチャが登録される前に警告を発するような両者を組み合わせたシステムが現実的と考えられる．

我々の研究室では，上述の亜種マルウェアの増加に対応するために準パススルー型ハイパーバイザーである BitVisor[3]を利用してストレージ装置のアクセスパターンを収集する研究[4][5]と，得られたストレージ装置のアクセスパターンを機械学習ならびに可視化することによってランサムウェアを検知する研究[6][7]を行ってきた．ここでランサムウェアとは被害者のコンピュータ上のファイルを暗号化して身代金を要求するタイプのマルウェアである．これらの先行研究ではストレージ装置のアクセスパターンに顕著な特徴が現れるランサムウェアを対象とした振

1 国立高等専門学校機構 豊田工業高等専門学校
National Institute of Technology, Toyota College
2 工学院大学
Kogakuin University

Vol.2020-DPS-182 No.14
Vol.2020-CSEC-88 No.14
2020/3/12

る舞い型の検知システムを評価してきた．しかし，ここ数年で Windows や MacOS などの主要なオペレーティングシステムが標準で暗号化ファイルシステムを採用し，ストレージ装置の入出力データを特徴量として利用しにくくなってきていることに加え，ストレージ装置に実体がないファイルレスマルウェアの増加といった変化が起きている．さらに，先行研究[6][7]のランサムウェア検知システムで採用しているストレージ装置のアクセスパターンだけでは，似た振る舞いを持つ無害なソフトウェアをランサムウェアと誤検知してしまう問題も生じている．

以上のような理由から，本研究では振る舞い検知のためには，ストレージ装置のアクセスパターンだけではなく，新たに RAM 上のメモリデータを用いることが不可欠であると考えた．通常のマルウェア解析では逆アセンブラと動的解析の機能を組み合わせた IDA Pro のようなツールが利用されている．IDA Pro のようなインタラクティブデバッガを用いれば実行中のメモリ内容を逐次確認していくことができる．攻撃コードや脆弱性の解析には IDA Pro のようなデバッガが適しているが，振る舞いの時系列データを効率よく収集するためには何か新しい枠組みが必要である．

そこで本研究では準パススルー型ハイパーバイザーである BitVisor[3]を基にした先行研究[4][5]を更に拡張し，従来のストレージ装置のアクセスパターンに加えて，新たに RAM 上のメモリデータを取得する機能を実装し，その性能評価をおこなった．

2. メモリ管理と仮想化

本稿ではまず監視対象となるゲスト OS でのメモリ管理について説明し，ハイパーバイザーによるゲスト OS のメモリ仮想化について説明する．これによってハイパーバイザーからメモリ領域を取得する際の課題を明らかにする．

2.1 OS の管理する仮想メモリ空間

メモリ管理は CPU に内蔵されているメモリ管理ユニット（MMU）とオペレーティングシステム（OS）が連携することで実現されている．現在の主要な OS が採用している仮想記憶方式はカーネルやプロセスが使用するメモリ領域に仮想アドレスを割り当て，CPU の MMU を用いて仮想アドレスから物理アドレスへ変換を行っている．

図 2 に Linux のプロセスのメモリマップを示す．アドレスは仮想アドレスである．Linux の 64 bit バージョンの場合，`0x0000000000000000` から `0x00007fffffffffff` が各プロセスのメモリ領域としてマッピングされる．メモリマップのアドレス上位（`0xffff800000000000` から `0xffffffffffffffff`）にはカーネルが利用するメモリ領域が割り当てられる．なお，従来は仮想メモリ空間のアドレス上位にカーネル空間全体がマッピングされていたが，最近の Linux カーネルでは Meltdown 等の対策として KPTI（Kernel Page Table Isolation）が採用されており，ユーザランドのプロセスからはカーネル空間の一部のメモリ領域しか見えなくなっている．とりあえず本稿ではユーザランドのプロセスを監視対象としたシステムを検討する．

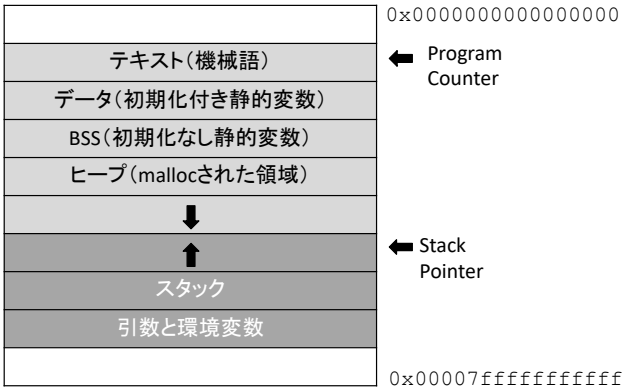

図 2　プロセスのメモリマップ

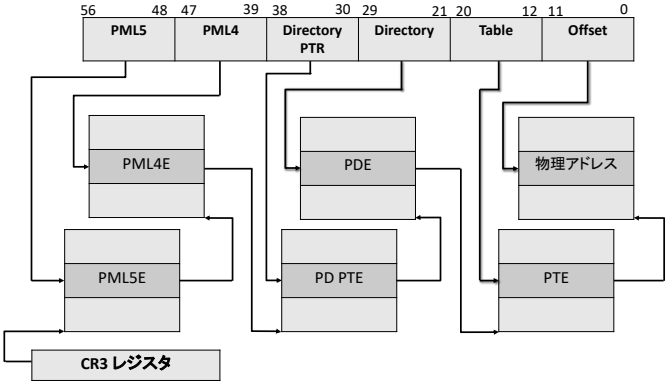

図 3　IA-32e paging による仮想-物理アドレス変換

図 2 に示すプロセスのメモリマップでは，アドレス下位にテキスト領域があり機械語がロードされる．以降でデータを保持する領域が続く．初期化付き静的変数を格納するデータ領域，初期化されていない変数を格納する BSS 領域，そしては malloc で確保された変数が配置されるヒープ領域である．アドレス上位にはスタック領域があり関数の局所変数や引数が格納される．ヒープは高い番地へ伸びるのに対し，スタックは低い番地へ伸びる．

本研究ではマルウェアの実行中のメモリ領域，あるいは攻撃コードが挿入される脆弱性のあるプログラムが実行中のメモリ領域を観測することを想定している．そのため図 2 のメモリマップから指定した仮想アドレスのメモリデータを時系列で取得できる機能を検討する．メモリマップの実体は OS が管理している構造体であるため，その下層で動作するハイパーバイザーからメモリ領域が何に使われているかを認識することは困難である．よってハイパーバイザーでアドレス空間とその用途を関連付けるための何らかの工夫が必要である．また，次に説明するように物理アドレスから仮想アドレスへの変換も考慮する必要がある．

Vol.2020-DPS-182 No.14
Vol.2020-CSEC-88 No.14
2020/3/12

2.2 仮想アドレスから物理アドレスへの変換

プロセスの仮想アドレスは最終的に RAM の物理アドレスに変換されてアクセスされる．Intel 社の CPU の 64 bit 拡張 IA-32e では図 3 に示す仕組みで仮想アドレスを物理アドレスに変換する[8][9]．4 階層のページング構造（4-level paging）を使ってアドレス変換をする場合はプロセスが使える仮想アドレス空間は 47 bit までである．この場合，CR3 レジスタは PML4 テーブルの物理アドレスを保持し PML5 テーブルは使われない．5 階層のページング構造（5-level paging）によってアドレス変換が行われる場合はプロセスが使える仮想アドレス空間は 56 bit となる．その場合，CR3 レジスタは PML5 テーブルの物理アドレスを保持する．

実際の仮想アドレスから物理アドレスへの変換は以下のように行われる．図 3 に示すように，仮想アドレスは 56 ビットで構成されており，先頭から PML5 のオフセット，PML4 のオフセット，ページディレクトリポインタテーブルのオフセット，ページディレクトリのオフセット，ページテーブルのオフセット，物理アドレスのオフセットである．始めに，CPU の CR3 レジスタの値（PML5 テーブルの先頭の物理アドレス）に PML5 のオフセットを足して，PML5E（エントリ）を求める．次に PML5E（エントリ）に PML4 のオフセットを足して PML4E（エントリ）を得る．さらに PML4E にページディレクトリポインタのオフセットを足して PD PTE（エントリ）の物理アドレスを求め，これにページディレクトリのオフセットを足すことで，PDE（エントリ）の物理アドレスが求められる．最後に PDE（エントリ）にページテーブルのオフセットを足して PTE（エントリ）を求め，最後に物理アドレスのオフセットを足すことで，仮想アドレスから物理アドレスへの変換が完了する．以上で述べた 5 階層のページング構造によるアドレス変換は高コストである．よって頻繁にアクセスされる仮想アドレスの変換結果は TLB（Translation Lookaside Buffer）にキャッシュされて高速にアクセスできるように工夫されている．本研究ではハイパーバイザーでメモリ空間を監視することを目指しているため，上述のアドレス変換の仕組みも必要となる．

2.3 ハイパーバイザーによるメモリの仮想化

オペレーティングシステムを仮想化するための CPU 拡張は Intel 社の CPU では VT-x と呼ばれ，AMD 社の CPU では AMD-V と呼ばれている．本稿では特に断りがない限り，Intel 社の VT-x をもとに説明する[10]．仮想マシンモニタまたはハイパーバイザーは VMX root-mode で実行され，ゲスト OS は VMX non-root mode で実行される．VT-x をサポートする Intel 社の CPU は Extended Page Table (EPT) と呼ばれるメモリ仮想化の拡張を備えている．図 4 に示すように EPT はゲスト OS の物理アドレス（ゲスト物理アドレス）を本物のホスト物理アドレスへ変換するテーブルである．EPT テーブルは VMCS (Virtual Machine Control Structure)

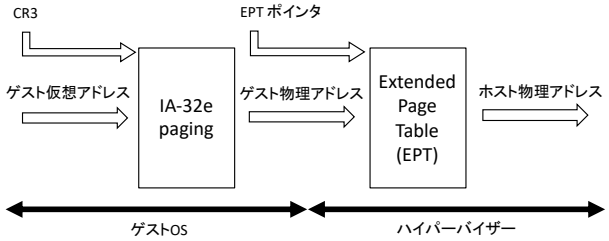

図 4　アドレス種別と変換のしくみ

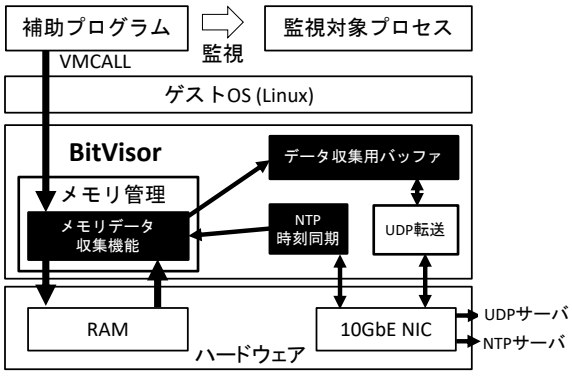

図 5　メモリデータ取得機能の構成

と呼ばれるハイパーバイザーが管理する構造体の EPT ポインタから参照される．EPT を用いてゲスト OS のメモリ空間を監視する手法のひとつに EPT violation を利用したものがある．これは EPT にあるゲスト OS の物理ページエントリのアクセス権限（Read access, Write access, Execute access）をわざと 0 に設定しておき，ゲスト OS がそのページにアクセスしたタイミングで VMM Exit を発生させてハイパーバイザーで該当ページを監視する手法である．本研究では将来的に EPT violation の利用も想定しているが，本稿では EPT を利用しないシンプルな手法を提案する．

2.4 準パススルー型ハイパーバイザーによるメモリ取得

本研究で利用している BitVisor [3] はゲスト OS を同時に 1 つしか実行しないハイパーバイザーであるため，基本的に本物の物理アドレス空間をそのままゲスト OS に見せている．つまり EPT によってゲスト物理アドレスを本物の物理アドレスに変換しても値は変化しない．ただし，ゲスト OS が BitVisor のメモリ空間を書き換えてしまうと困るため，BitVisor が利用しているメモリ空間だけは ROM に偽装して書き込みできないようにしている．BitVisor は当初は Shadow Page Table（SPT）と呼ばれるソフトウェアによるメモリ仮想化を提供していたが，現在は VT-x の EPT を用いたハードウェアによるメモリ仮想化を提供している．BitVisor では SPT と EPT のどちらを使うかを選択できるようになっている．本研究のシステムは EPT を有効にして動作させているが，現状では前述のとおり EPT を利用した監視はおこなっていない．とくに本研究では負荷の大き

Vol.2020-DPS-182 No.14
Vol.2020-CSEC-88 No.14
2020/3/12

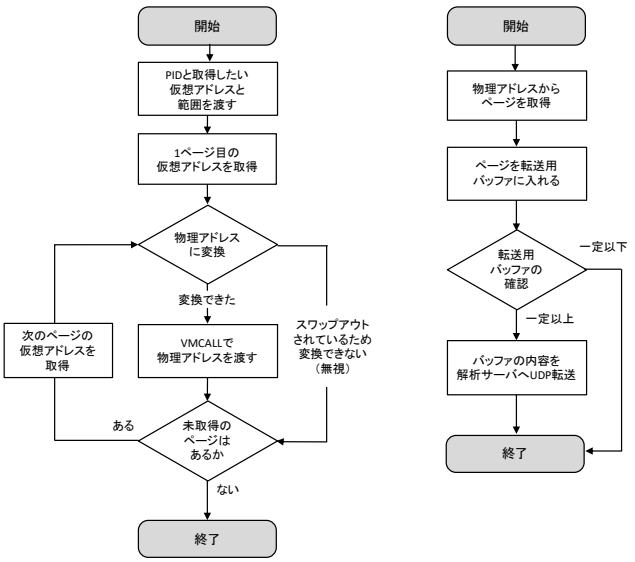

図 6　処理の流れ（左：ユーザランドの補助プログラム，右：ハイパーバイザーのメモリデータ収集機能）

い EPT violation ではなく，振る舞いを時系列で収集するために定期的にメモリ領域を取得する，より単純な方法を採用するものとした．

3. メモリデータ取得機能の試作

本研究で試作したメモリデータ取得機能の構成を図 5 に示す．本研究では，先行研究[4][5]で開発したストレージ装置のアクセスパターンを収集するためのバッファや転送部を流用した．本研究では新たにメモリデータ収集機能を実装した（図 5 の黒箱の部分）．先行研究では収集データは TCP によって Hadoop 分散ファイルシステムへ転送していたが，本研究では出来る限り大量のメモリデータを処理するため UDP 転送に変更した. 転送部は BitVisor が対応している Intel 社の 10 GbE NIC ドライバを用いて実装した．

3.1 補助プログラムとメモリデータ収集機能

試作したシステムは Linux のユーザランドで動作する補助プログラムと準パススルー型ハイパーバイザー BitVisor のメモリデータ収集機能から構成される．補助プログラムは Intel VT-x（仮想化技術拡張）の VMCALL [10] により BitVisor 側のメモリデータ収集機能を呼び出す．VMCALL 命令は VMX non-root mode で動作するゲスト OS のソフトウェアにハイパーバイザーの処理を利用可能にさせる命令である．VMCALL 実行後は VM Exit が発生してゲスト OS からハイパーバイザーに処理が遷移する．本研究で実装したメモリデータ取得機能の処理の流れを図 6 に示す．左のフローチャートはユーザランドで実行する補助プログラムの処理の流れである．監視対象プロセスの PID（プロセス ID）と，監視するメモリ領域を先頭の仮想アドレスと長さで指定する．補助プログラムは 4,096 byte のページ単位で仮想アドレスを物理アドレスに変換する．

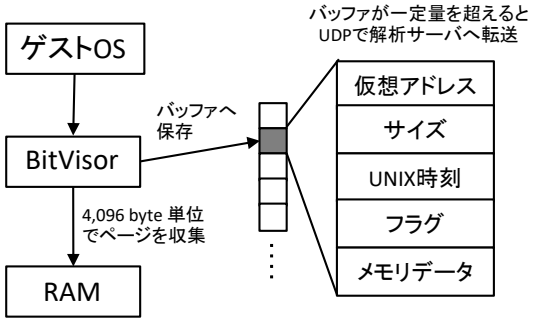

図 7　ハイパーバイザの転送用バッファ

表 1　開発環境

ハードウェア	規格
CPU	Intel Celeron G3920 2.9GHz
RAM	DDR4-2133，8GB
ネットワーク	Intel X550-T2 (10GbE)

ここで変換できない場合には RAM からスワップアウトされているため（利用されていないものとして）今回は監視対象外として無視する．物理アドレスに変換できたら VMCALL でハイパーバイザー側のメモリデータ収集機能（図 6 の右側）を呼び出す．4,096 byte のページ単位で VMCALL を実行して，指定された全ての領域のメモリデータを収集する. メモリデータ収集機能は BitVisor のメモリ管理機能を利用して物理アドレスにある 4,096 byte のページ内容をバッファにコピーし，バッファが一定量に達したら UDP でサーバに転送する．UDP サーバでは監視データを保管し，保管されたデータはその後に時系列解析や機械学習にかけられる[6][7].

3.2 収集する情報

図 7 に BitVisor 内部で収集したメモリデータを一時保管する転送用バッファの詳細を示す．BitVisor は準パススルー型ハイパーバイザーでありゲスト OS への影響を最小限にできる利点がある．本稿の提案システムでもこの方針を踏襲し，取得したメモリデータは解析対象のホストには一切保管せずに，NIC を介して解析サーバへ転送する．図 7 に示すように収集する情報は，物理アドレスからアクセスして取得した 4,096 byte のページ単位のメモリデータのほかに，プロセス空間での仮想アドレスの値（これは VMCALL で物理アドレスと一緒に BitVisor 側へ渡されたもの），サイズ（通常はページ単位なので 4,096 byte），収集したタイミングでの UNIX 時刻（ナノ秒単位），フラグも保存し，サーバへ転送された後で時系列データ解析に利用できるようにしている. フラグは先行研究[4][5]で実装済みのストレージ装置のセクタ単位での読み書きのデータと，今回提案しているメモリデータを区別するための情報である．

3.3 開発環境

これまでの設計を試作した環境を表 1 に示す．ゲスト OS は CentOS 7 (Linux) 64 bit を用いた．BitVisor は 2018 年 12 月 4 日時点のソースコードを改造した．補助プログラムは BitVisor に含まれている VMCALL 呼び出しライブラリを利用して C 言語を用いて実装した．

4. メモリ収集機能の動作検証

試作したメモリデータ収集機能が正しく動作しているかを確認するために，以下のようにプロセスのテキスト領域，スタック領域，ヒープ領域の 3 通りのメモリデータを取得する実験をおこなった．補助プログラムにはプロセス ID（PID）と仮想アドレスの先頭番地と長さを与えた．補助プログラムに与えた仮想アドレスの先頭番地は Linux の /proc/PID/maps にあるメモリマップをもとに決定した．

- Google Chrome を実行させ，テキスト（機械語）領域を取得し，収集したメモリデータが Chrome のバイナリの機械語と一致することを確認した．
- 自作 C プログラムでローカル変数を宣言し，スタック領域を取得した．ローカル変数の値が収集したメモリデータに存在することを確認した．
- 自作 C プログラムで malloc でメモリ領域を動的に確保し，ヒープ領域を 1 秒毎に取得した．最初は Hello world! という文字列を格納しておき，途中から Goodbye world! に書き換えた．収集したメモリデータ側でも途中から値が変わったことを確認した．

5. 性能評価

試作したメモリデータ収集機能のスループットを実験で確かめた．図 6 の左側の処理を実装した補助プログラムを用いて，自作 C プログラム（監視対象プロセス）で malloc するサイズを変化させ，malloc で確保したヒープ領域を収集し，補助プログラムの実行時間を Linux の time コマンドで計測した．スループットは仮想アドレスから物理アドレスに変換できた（つまりスワップアウトしていなかった）メモリデータの累計サイズ [MB] を，実際に補助プログラムの実行時間 [s] で割って求めた．計測したメモリデータ収集機能のスループットを図 8 に示す．スワップアウトされるページが 1,000 MB を確保した時点で増えているのは実験に用いた表 1 のマシンの物理メモリが 8 GB だったからと考えられる．最高スループットは 500 MB を malloc したときの 190.7 MB/s であった．

さらに，上記と同様の実験を VMCALL の発行部分をオン，オフして（つまりメモリデータ収集をオン，オフして）時刻時間を計測した．これによって，VMM コールの呼び出しから戻ってくるまでの時間が，補助プログラム全体の時間のどれだけを占めているかを計測した．VMCALL 実

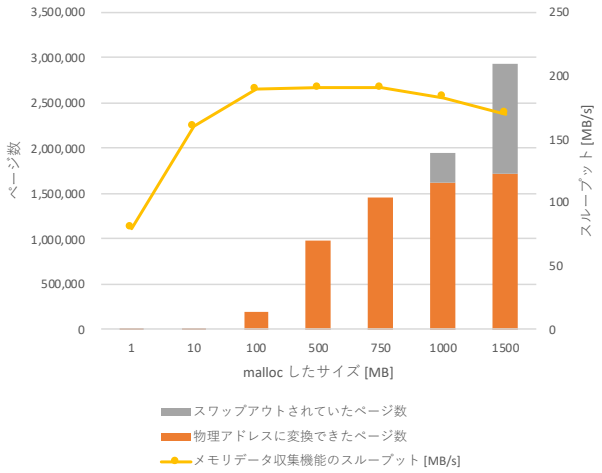

図 8　メモリデータ収集機能のスループット

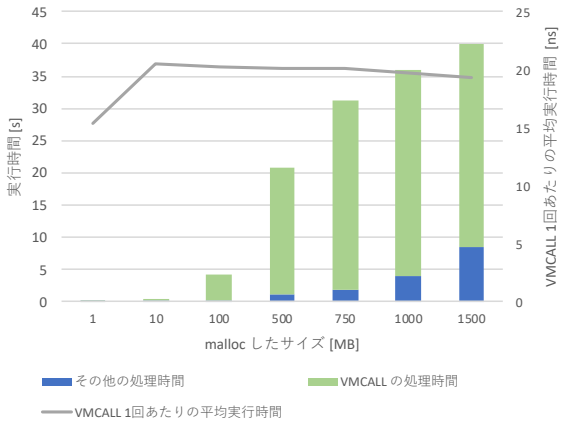

図 9　VMCALL 実行時間

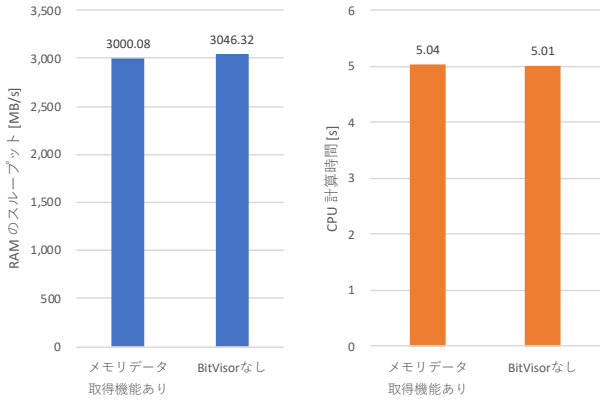

図 10　ベンチマークの実行結果

行時間のグラフを図 9 に示す．VMCALL の処理時間（青色）の割合が最も高くなったのは malloc で 500MB を確保したときであり，補助プログラム全体の実行時間の 94.3%が VMCALL の処理時間に費やされていた．VMCALL の 1 回の平均実行時間（灰色のグラフ）は 19.4 ns であった．

最後にメモリデータ収集機能を実行中と実行していないときのベンチマークプログラムの結果を比較した．ベンチ

マークには sysbench [11]を利用し，memory と cpu をそれぞれ５回実行して平均を求めた．実行結果を図 10 に示す．メモリ取得機能の有無がゲスト OS に与える影響は，メモリ帯域幅には 1.5%, CPU 計算速度には 0.5% の性能低下であることがわかった．

6. 考察

まずは性能評価の結果について考察する．メモリデータ収集機能の最大スループットについては最大で 190.7 MB/s となった．今回の実験で用いた Intel Celeron G3920 はメモリチャネルが 2 であり, DDR4-2133 が 2,133MHz で動作し，バス幅は 8 byte であるため, これらの積をとって理論上の最大メモリ帯域は 34.1 GB/s となる．今回実装したメモリデータ収集機能のスループットは理論上の最大メモリ帯域幅の 0.5% しか利用していないことがわかった．このことから最後のベンチマークで CPU ならびにメモリに影響をほとんど与えていない理由が説明できる．つまり，試作したメモリデータ収集機能の性能がそれほど高くないためシステムに影響を与えるほどの負荷がかかっていないことがわかった．

VMCALL 実行時間の計測結果では VMCALL をページ単位で発行するときの負荷の大きさが明らかになった．具体的には最大スループット時で補助プログラム全体の 94.3%が VMCALL の実行に費やされていた．VMCALL 以外の処理は仮想アドレスから物理アドレスへの変換処理で全体の処理時間の 6%未満ではあるが，メモリ帯域幅を考慮すると性能改善の余地が大きいため，この部分についても更なる処理時間の削減が求められる. VMCALL の発行回数を減らす方法としてはメモリデータ取得を複数ページにまとめて一度の VMCALL で発行する改善案を検討している．

7. おわりに

本稿では準パススルー型ハイパーバイザーである BitVisor にメモリデータ収集機能を実装し，評価結果を示した．試作した機能の最大スループットは 190.7 MB/s であった．現状ではスループットに大いに改善の余地があるものの，取得するメモリサイズを小さく抑えることで，ある程度は実用に耐えうる時系列データのメモリ取得機能を試作できた．今後はスループットの改善と負荷試験を継続して実施し，実際のマルウェアやランサムウェアの時系列解析に活用していきたい．

謝辞 本研究は JSPS 科研費 17K00198 の助成を受けたものです．

参考文献

[1] AVTEST: The independent IT-Security Institute Malware Statistics, https://www.av-test.org/en/statistics/malware/, (参照 2020-01-26).
[2] Kaspersky, Antivirus fundamentals: Viruses, signatures, disinfection, https://www.kaspersky.com/blog/signature-virus-disinfection/13233/, (参照 2020-02-14).
[3] Shinagawa, T. et al. Bitvisor: a thin hypervisor for enforcing i/o device security. In Proceedings of the 2009 ACM SIGPLAN/ SIGOPS international conference on Virtual execution environments, pp. 121-130, 2009.
[4] Hirano, M., Tsuzuki, T., Ikeda, S., Taka, N., Fujiwara, K., and Kobayashi, R. WaybackVisor: Hypervisor-based scalable live forensic architecture for timeline analysis. In International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage, pp. 219-230, Springer, Cham, 2017.
[5] 高直我, 池田征士朗, 平野学, 小林良太郎. 準パススルー型ハイパーバイザによるストレージアクセスパターンの収集システムの提案. コンピュータセキュリティシンポジウム 2018 論文集, 2018(2), pp. 678-685.
[6] Hirano, M. and Kobayashi, R.. Machine Learning Based Ransomware Detection Using Storage Access Patterns Obtained from Live-forensic Hypervisor. 2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS). IEEE, pp. 1-6, 2019.
[7] 池田征士朗, 髙直我, 平野学, 小林良太郎. ストレージアクセス履歴の時系列解析システムの実装とランサムウェア解析への応用. 研究報告コンピュータセキュリティ (CSEC), 2018(10), 1-7.
[8] Intel Corporation, Chapter 4 Paging, Intel® 64 and IA-32 Architectures Developer's Manual, Vol. 3A, 2016.
[9] Intel Corporation, 5-Level Paging and 5-Level EPT (white paper), https://software.intel.com/sites/default/files/managed/2b/80/5-level_paging_white_paper.pdf, (参照 2020-02-16).
[10] Intel Corporation, Chapter 23 Introduction to Virtual Machine Extensions and Chapter 28 VMX Support for Address Translation, Intel® 64 and IA-32 Architectures Developer's Manual, Vol. 3C, 2016.
[11] Scriptable database and system performance benchmark, https://github.com/akopytov/sysbench, (参照 2020-02-16).