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

高 直我[†1]　池田 征士朗[†1]　平野 学[†1]　小林 良太郎[†2]

概要：マルウェア対策のひとつとして、そのマルウェアの振る舞いを分析する手法がある。近年、WannaCry のようなストレージアクセスパターンに特徴を持つランサムウェアが流行している。そこで、ストレージアクセスパターンを時系列に着目して収集できるシステムを提案する。本システムを準パススルー型ハイパーバイザである BitVisor を用いて実装することで、収集システムによって発生するオーバヘッドを大幅に抑制しつつ、マルウェアの振る舞いに応じて発生するストレージへのアクセス頻度やそのパターンを調べることが可能となる。本稿では、提案システムの実装を示し、収集システムを動作させた時と動作させていない時の転送速度を比較する。さらに、WannaCry を実行したときと定常状態のアクセスパターンを収集した結果を報告する。

キーワード：マルウェア解析、動的解析、仮想マシンモニタ、Hadoop 分散ファイルシステム

1. はじめに

インターネットの発展とともに、マルウェアの被害も増加している。ドイツのセキュリティソフトウェア評価機関である AV-TEST が開示しているマルウェアに関する調査 [1] から分かるように、活動が把握できているマルウェアの数は増加傾向にあり、ここ 10 年間で約 30 倍となっている。AV-TEST は Sunshine や VTEST などの解析ツールを用いて毎日新たに 35 万件のマルウェアを検出している。インターネットから直接アクセスできないダークウェブ上の不正サイトにおいて、サイバー犯罪の実行に必要なマルウェア作成ツールやサービスを購入することができる。さらにマルウェア作成者は、既知のマルウェアに対して暗号化や難読化を施すパッカーを用いて簡単に亜種マルウェアを作成することができる。亜種マルウェアとは元々存在するマルウェアに類似した挙動を示すマルウェアを指し、亜種マルウェアは近年急速に増加している。

これらマルウェアの脅威への対策は大きく 2 つに大別できる。パターンマッチング型が代表として挙げられるリアクティブな対策と、アノマリ型が代表として挙げられるプロアクティブな対策である。マルウェアによって発生したインシデント後で対策をするのか、その前で対策をするのかという観点でリアクティブ（事後）とプロアクティブ（事前）という言葉を用いている。パターンマッチング型の対策とは、検査対象のプログラムのシグネチャとマルウェアのデータベース上にあるシグネチャを比較することでマルウェアを検知しコンピュータを防御する仕組みを指している。既知のマルウェアを確実に発見できるという利点があるが、新たなマルウェアが発生する速度にシグネチャの生成が追い付かないことが問題点として挙げられる。

アノマリ型の対策アプローチとして、コンピュータの振る舞いによってマルウェアを検知することでコンピュータを保護する仕組みがある。ヒューリスティック方式とビヘイビア方式の 2 つが存在する。ヒューリスティック方式とはマルウェアのコードを静的に解析することで挙動を把握し、その振る舞いをコンピュータに登録することでマルウェアを検知する対策手法である。

一方、ビヘイビア方式ではサンドボックス環境等において実際にマルウェアを動作させた上で動的に振る舞いを解析しており、動的ヒューリスティック方式とも呼ばれる。パターンマッチング方式では検出できない亜種のマルウェアを検出できることがアノマリ型対策の優位性である。しかしマルウェアがパッキングされることによって、ヒューリスティック方式でも解析が困難となる。これに対してビヘイビア方式では、マルウェアがパッキングされていたとしても、振る舞いは同じであるので検知することができる。

しかしビヘイビア方式にも問題があり、マルウェアに解析環境下であることを特定され、感染動作を停止されてしまうという問題点がある。解析環境下であることを特定する方法として、解析プロセスの存在を確認する方法や、プログラムの実行時間から仮想環境下を特定する方法など [2] がある。本研究ではこれらの問題を解決するために、仮想マシンモニタ BitVisor [3] を用いて動的にマルウェアの振る舞いを収集するシステムを提案する。提案システムは先行研究 [4] で示したストレージに対する入出力を監視するシステムをマルウェア解析に応用したものである。本稿では、先行研究で示していなかったシステムの詳細設計とマルウェアの挙動を解析した結果を報告する。

仮想マシンモニタは OS が動作する実際のコンピュータを、ソフトウェアによって実現するために必要な制御を行っている。ハイパーバイザ型の仮想マシンモニタを用いることで、仮想化によるオーバヘッドを減らすことができ

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

© 2018 Information Processing Society of Japan

る。それらの中でも特に BitVisor はオーバヘッドが低いものとして知られている。

WannaCry のようなストレージアクセスに特徴を持つランサムウェアが流行していることを受け、本システムではコンピュータの振る舞いの 1 つであるストレージアクセスパターンを時系列に収集する。さらに BitVisor を用いて実装することで、仮想化するハードウェアを限定することができるため、マルウェアの振る舞いを観測する際に発生するオーバヘッドを大幅に抑制できる。これらによってマルウェアに解析環境下であることを検知されず、動的に解析を行うことが可能となる。

本稿では、2 章で仮想マシンモニタを用いた監視機能を持つシステムをいくつか挙げたのち、3 章で仮想マシンモニタおよび BitVisor の説明をする。4 章では収集システムの実装方法を述べ、5 章でストレージ転送速度とアクセスパターンの可視化結果から本システムの有用性を述べる。最後に 6 章でまとめを行う。

2. 関連研究

Qi らはコードサイズの小さい軽量な、ハイパーバイザ型の仮想マシンモニタである ForenVisor [5] を、最小限のデバイスドライバのみで実装した。ForenVisor はプロセスやネットワークトラフィック、メモリなどのデータをローカルのストレージ領域へ保存するシステムである。

品川らによって開発された BitVisor [3] は、クライアントコンピュータへセキュリティ機能を提供する目的で作られた仮想マシンモニタである。仮想マシンモニタレベルでストレージデータやネットワークの通信データを強制的に暗号化することによって、セキュリティ機能を確保している。

大山らはマルウェアの動的解析のために、OS のチェックポイントを生成するシステム BVCP [6] を、BitVisor を拡張する形で実現した。ある時点でのメモリデータ、CPU のレジスタデータ、ストレージデータを取得し、別のパーティションへ保存する。マルウェアの実行後には、パーティションに保存したスナップショットをロールバックすることで、インシデント前の状態へコンピュータを復元することができる。

大月らは Alkanet [7] というシステムコールをトレースすることができる仮想マシンモニタを提案した。山下らは Alkanet を Windows10 に拡張させた [8]。Alkanet も BitVisor を基盤としている。CPU の EAX レジスタと EDX レジスタを参照することでシステムコールと引数のアドレスを特定し、フックした CR3 レジスタからシステムコールの発行元プロセスを特定している。監視データは IEEE 1394 を介してロギングコンピュータへと転送される。Alkanet により仮想マシンモニタのレイヤで、マルウェアの挙動をシステムコールから捉えることが可能となった。

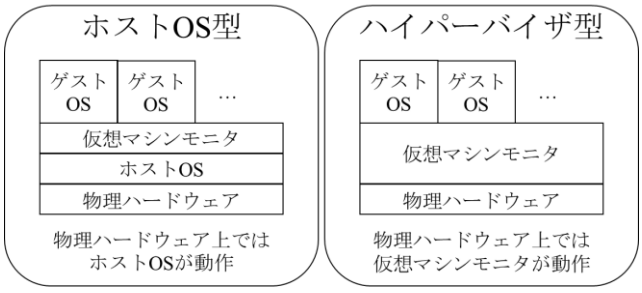


図 1　ホスト OS 型とハイパーバイザ型による仮想マシンモニタの仕組みの違い

ForenVisor や BVCP はコンピュータ内に監視、収集したデータを蓄える。一方、我々が提案する収集システムでは外部のサーバへとデータを集約させるため、より多くのデータを集めることができる。ForenVisor はフォレンジック目的でのデータ収集を主な目的としており、解析機能は有していない。対して、提案システムは収集したデータを解析、可視化する機能を有している。Alkanet はシステムコールからマルウェアを解析するが、提案システムではストレージアクセスパターンに着目している点が異なっている。本稿では、ストレージアクセスの記録のみを外部のサーバへ転送する機構を提案するが、同じ機構を利用することで、将来的にメモリや他のデバイスの入出力を収集することも検討している。

3. 仮想マシンモニタ

仮想マシンモニタは OS よりも低いレイヤで動作し、仮想的にハードウェアリソースを提供することで 1 台のコンピュータ上で複数の OS を動作させることができる。仮想マシンモニタ上で動作する OS はゲスト OS と呼ばれ、動作させることができるゲスト OS の数は仮想マシンモニタによってさまざまである。仮想マシンモニタは仮想化の方法の違いから 2 つに大別され、それぞれが異なった特性を有している。1 つはホスト OS 型で、他方はハイパーバイザ型である。

ホスト OS とは仮想化されていない OS を指しており、そのホスト OS 上のアプリケーションとして動作する仮想マシンモニタをホスト OS 型の仮想マシンモニタという。よって物理ハードウェア上ではホスト OS が動作しており、デバイスドライバやメモリ管理、プロセス管理といったホスト OS が提供する OS としての機能を、仮想マシンモニタ上で動作するゲスト OS がそのまま利用することができる。

図 1 にホスト OS 型とハイパーバイザ型の仮想マシンモニタの違いを示す。ハイパーバイザ型ではホスト OS 型とは異なり、物理ハードウェア上で仮想マシンモニタが動作し、その上にゲスト OS が存在するアーキテクチャになっ

© 2018 Information Processing Society of Japan

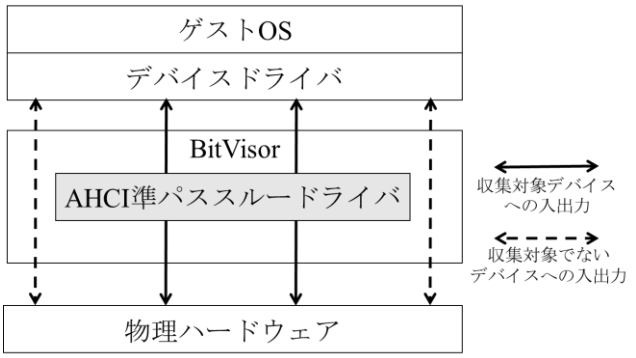


図 2　AHCI 準パススルードライバの仕組み

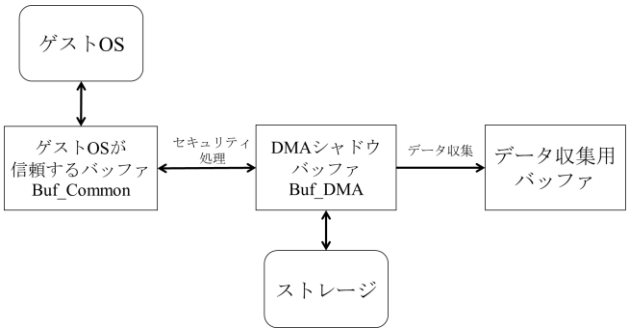


図 3　ストレージデータに対する暗号化の仕組み

ている。デバイスドライバやメモリ管理、プロセス管理といった OS の機能は仮想マシンモニタが提供する。ホスト OS 型ではすでに利用しているコンピュータに対しても手軽に導入できるが、ハードウェアリソースへのアクセスにホスト OS を介するため、余計なオーバヘッドが生じてしまう。仮想化によるオーバヘッドを減らすことによって、システムの安定性が向上し、マルウェアに解析環境下であることを隠ぺいできる。以上のことから、本システムではハイパーバイザ型の仮想マシンモニタを採用した。さらにハイパーバイザ型は仮想化の方法から、完全仮想化と準仮想化に分かれる。完全仮想化方式ではハードウェア自体を仮想化するため、カーネルを改変することなく Windows をはじめとする様々な OS を動かすことができる。

一方、準仮想化方式ではハードウェアを完全に再現せず、仮想マシン上で動作するゲスト OS 側の機能を一部利用する。仮想化する領域が縮小するため軽量に動作するが、ゲスト OS に対してその仮想化環境に対応した改変をしなければならない。したがって Linux のようなオープンな OS とは異なり、Windows などプロプライエタリな OS は仮想化することができない。ストレージアクセスパターンの収集を目的とする本システムにおいて OS が制約されてしまうのは適当でないため、完全仮想化方式のハイパーバイザが望ましい。

BitVisor はコンピュータへセキュリティ機能を提供するために作られた仮想マシンモニタであり、完全仮想化方式のハイパーバイザである。それに加え本システムを実装するにあたり BitVisor を利用する利点は、準パススルードライバというデバイスドライバを採用している点である。

図 2 に準パススルードライバの仕組みを示す。準パススルードライバは、仮想化対象のハードウェアに対してそれぞれ用意され、デバイスドライバと対象ハードウェア間の入出力は準パススルードライバを介して行われる。仮想化対象外のハードウェアについては仮想化を行わずパススルーする。これにより仮想化領域を局所化させることができるため、仮想化によって発生するオーバヘッドを削減することができる。現在のストレージの規格の大半は SATA (Serial Advanced Technology Attachment) である。SATA 規格のハードウェアに対するデバイスドライバは AHCI (Advanced Host Controller Interface) と呼ばれており、BitVisor は AHCI に対応した準パススルードライバである AHCI 準パススルードライバを持っている。

図 3 に BitVisor によるストレージデータに対する暗号化の仕組みを示す。通常のコンピュータでは、ストレージと OS の両方が信頼する共通なバッファ（以後 Buf_Common とする）を介している。一方 BitVisor を導入したコンピュータでは、ゲスト OS が Buf_Common を信頼することは同じだが、ストレージは BitVisor が提供する DMA シャドウバッファ（以後 Buf_DMA とする）をバッファ領域として認識する。そのため BitVisor が動作しているコンピュータ上でストレージからデータを呼び出す際には、ストレージ、Buf_DMA、Buf_Common、ゲスト OS の順にデータが遷移する。そして BitVisor は Buf_DMA から Buf_Common へと転送される際に、ストレージデータの暗号化を行っている。

本研究では、紹介した Buf_DMA に加え、もうひとつバッファを BitVisor 内で用意し、そこに解析に必要なデータを収集し格納する。このデータは Buf_DMA へアクセスを行うことで取得している。

4. 実装

4.1 全体構成

図 4 にシステム全体の構成を示す。本システムを実現するにあたり、BitVisor を用いて新たに実装した部分をグレーでマスキングしている。AHCI 準パススルードライバへ、ストレージに関するデータを収集する仕組みを実装する。収集したデータは一度 BitVisor 内にあるデータ収集用バッファへ保存される。保存される際にインターネットを介して取得した時刻情報も合わせて保存する。一定量バッファリングされることをトリガとして、HDFS (Hadoop Distributed File System) サーバへの転送が開始される。

収集したデータは Hadoop を用いて解析を行うため、その Hadoop で利用されている HDFS というファイルシステムを採用したサーバ（HDFS サーバ）へ転送される。HDFS は GFS (Google File System) をベースとした、高スループッ

© 2018 Information Processing Society of Japan

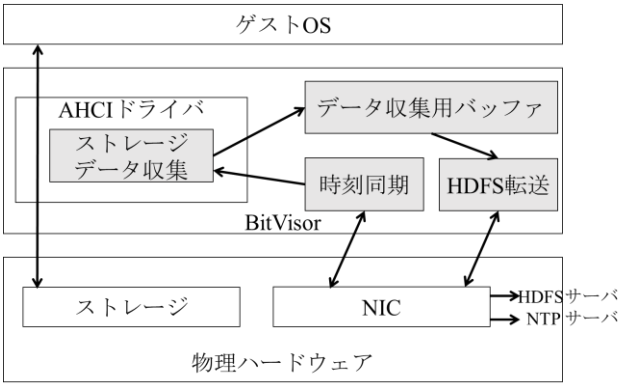


図 4　システム全体の構成

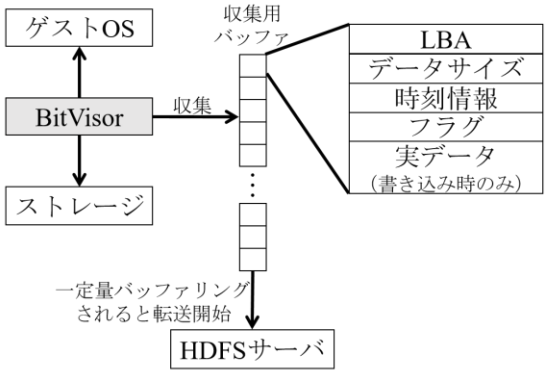


図 5　HDFS サーバへ転送されるデータ

トかつデータの安全性を担保することができるファイルシステムである [9]。HDFS ではデータの分散保存位置やファイル名、パーミッションといったメタデータを保持する NameNode と、データそのものを格納する DataNode という 2 つのノードから構成されている。HDFS サーバへのデータ書き込みは以下のようになる。クライアントはまず HDFS の構造を管理する NameNode へ、どの DataNode へ書き込めばよいか問い合わせる。次にその返答に沿った DataNode へデータを書き込む。本研究でも同様の仕組みを使って HDFS サーバへデータを集約し、Hadoop でデータの処理を行っている。

HDFS サーバへ転送されるデータについて図 5 に示す。収集データは、ストレージと OS との間でやり取りされた実際のデータ（以降、実データと呼ぶ）と、その実データのメタデータとなるヘッダの 2 つで構成されている。ヘッダにはストレージへの読み書きが行われたデータのアドレスを示す LBA (Logical Block Address) 、データサイズ、読み書き発生時の時刻情報、read/write フラグの 4 つを取得し、バッファへ格納する。read/write フラグは、読み込みの収集データか書き込みの収集データかを判別する。また、ストレージへの書き込みに対するデータを収集する際には、実データを 4KiB ごとに分割して格納する。書き込みデータの収集時に、実データと LBA との対応付けがなされているため、読み込み時は LBA の値から実データを特定することができる。以上のことから、読み込みのデータを収集するときは実データを取得せず、ヘッダのみを収集するようにした。

各収集パラメータの取得方法については次項から順に述べる。

4.2 LBA の取得

LBA とはストレージの各セクタに一意に値を割り当て、その値を用いてアドレッシングを行うアドレス表現の 1 つである。1 つの LBA の単位であるストレージのセクタサイズには 512 バイトと 4,096 バイトの 2 つの規格がある。従来のストレージは 512 バイトのセクタサイズだが、フォーマット効率の良さや記録容量を拡大できることから 4,096 バイトセクタへの移行が推進されている。

ストレージ内のデータをアドレッシングする方法は LBA 方式の他に CHS 方式がある。CHS 方式ではシリンダ、ヘッド、セクタの 3 つの値を用いてアドレス表現を行う。しかしストレージの大容量化しており、またセクタ数がシリンダ毎で異なることから、CHS 方式でアドレス表現をすることが難しくなった。そこで CHS 方式に変わって提案されたアドレス表現が LBA 方式である。SATA 規格では LBA 方式が採用されている。

BitVisor は SATA デバイスのデバイスドライバである AHCI に対応し、SATA デバイスの入出力データを収集している。AHCI では FPDMA (First Party Direct Memory Access) と呼ばれるメモリとストレージとの間で直接転送を行う方式が採用され、その転送時にいくつかのレジスタが使用される。Command List はそのレジスタの 1 つで、ストレージへの複数のコマンドを格納しているリストである。図 6 へ AHCI における LBA の取得方法と、次の項で述べる読み書きされたデータサイズの取得方法を示す。Command List は要素数 32 の Command Header の配列で、Command Header は 1 つの SATA 転送コマンドに対応している。つまり Command List は最大 32 個の転送コマンドを保持し、それらを実行している。

このように複数の転送コマンドを一括で管理している理由は、NCQ (Native Command Queuing) と呼ばれる技術を利用するためである。NCQ は AHCI が複数のコマンドを効率よく実行するための技術であり、シーク時間が最短となるようにストレージへの読み書きの順番を並び替えてから実行する。順番に実行するよりもシーク時間が削減されるため、データへのアクセス速度を向上させることができる。Command Header は 1 つの Command Table を指している。Command Table は、デバイスへの転送コマンドが格納される Command FIS (Frame Information Structure) と、実際に転送を行うデータが存在するメモリ領域を参照している PRDT (Physical Region Descriptor Table) を保持している。

LBA の取得には Command FIS を参照する。Command FIS は fis_0x27 レジスタというデータ領域を保持しており、こ

© 2018 Information Processing Society of Japan

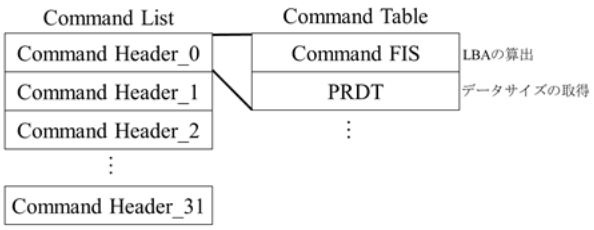


図 6　AHCI における LBA とデータサイズの取得方法

のデータ領域に転送元、あるいは転送先のシリンダ、ヘッド、セクタの値が格納されている。これらの情報を取得し、BitVisor の AHCI 準パススルードライバで実装されている CHS 方式から LBA 方式への変換方法を踏襲して、読み書きが発生したデータの LBA を取得している。

4.3 データサイズの取得

図 6 に Command List、Command Header、Command Table 等の関係性および LBA とデータサイズの取得方法を示す。読み書きが行われたデータのサイズは、LBA を取得する方法と同様に Command List、Command Header、Command Table と数珠繋ぎにメモリ空間を参照することで取得できる。LBA は Command Table が指し示す Command FIS からストレージのシリンダ、ヘッド、セクタの値を取得し、算出している。データサイズは実際に転送されるデータが存在するメモリ領域を指定している、PRDT から取得することができる。PRDT は Command Table によって示されているメモリ領域のひとつで、65,536 個の要素を持つ配列となっている。個々の要素は転送データが存在するメモリアドレスの先頭アドレスと、転送データのデータサイズを保持しているため、この領域にアクセスすることで転送されるデータのサイズを取得することができる。なお、実データの先頭アドレスについても同時に取得している。

4.4 時刻情報の取得

コンピュータはマザーボード上に実装されている RTC (Real Time Clock) から時刻情報を取得している。時刻の取得は OS の機能として与えられるものであるが、OS よりも低いレイヤで動作する BitVisor においてこの機能を利用することはできない。さらに BitVisor に RTC にアクセスする機能は実装されていないため、時刻を取得する機構を独自に作る必要がある。そこで BitVisor が持つ軽量の TCP/IP スタックである lwIP (lightweight Internet Protocol) を利用し、NTP (Network Time Protocol) サーバから時刻情報を取得する。本研究では、時刻情報を取得するために NTP の簡易版である SNTP (Simple NTP) を用いている。

サーバへの時刻の取得はインターネットを介するため、データをバッファへと格納する際に毎回 NTP サーバへアクセスを行うと、その処理がボトルネックになり、システムのパフォーマンスに影響を与える可能性がある。そこで時刻サーバへの問い合わせは 1 分間隔とし、問い合わせを行わない間については、BitVisor が起動してからの時間を加算することで時刻を特定している。起動からの経過時間は BitVisor の準パススルードライバが提供している。インターネットアクセスよりもメモリアクセスのほうが高速であるため、時刻の取得が与えるシステムへの影響はこちらの手段のほうが小さい。

表 1　転送速度の計測環境

CPU	AMD Athlon Ⅱ X2 245 2.9GHz
RAM	DDR3 SDRAM 8GiB
NIC	Intel PRO1000 (BitVisor)
	Broadcom Gigabit Ethernet (Guest OS)
ストレージ	TOSHIBA MQ03ABB300 3TB HDD
ゲスト OS	Windows 7 Professional SP1 64bit
	Debian GNU / Linux 8 (amd64)
仮想マシンモニタ	BitVisor BitBucket で 2016 年 2 月 17 日に取得

4.5 実データの取得

実データの取得は、先頭アドレスとデータサイズに基づいて行われる。Command Header は DMA 転送用のバッファを参照するためのアドレスを保持しているため、はじめにそのアドレスを取得する。DMA 転送用のバッファの中には実データのアドレスが示されていることから、その領域と PRDT から取得したデータサイズがわかることで実データを順々に取り出すことが可能となる。

4.6 収集データの格納

前項までに必要とされるデータの収集方法を示した。これら読み込み書き込みのデータは同一の配列で保持されているため、データを配列に格納する際に read/write フラグをヘッダに付加している。

収集データは 4 つのメンバを持つ構造体配列として定義されている。4 つのメンバはメタデータであるヘッダ、実際に転送が行われた実データ、各要素の状態を表すパラメータ、各要素のデータサイズである。ヘッダには前述したように、読み書きが行われたデータの LBA、時刻、データサイズ、read/write フラグから構成されている。各要素の状態を表すパラメータは、EMPTY、BUFFERD、TRANSFERRD という 3 つの状態から定義され、各々の要素が空の場合は EMPTY、バッファリングされている場合は BUFFERED、HDFS サーバへ転送中の場合は TRANSFERRED が設定される。これらの状態は配列内の収集データの上書きを防ぐ役割を担っている。各要素のデータサイズは、HDFS サーバへ転送する際のデータ量を示している。配列の各要素から、書き込みの際はヘッダと実データ、読み込みの際はヘッダのみを収集したデータとして HDFS サーバへ転送する。

4.7 HDFS サーバへの送信

配列に格納済みの収集データを HDFS サーバへと集約し解析を行うため、BitVisor から取得したデータを HDFS サー

© 2018 Information Processing Society of Japan

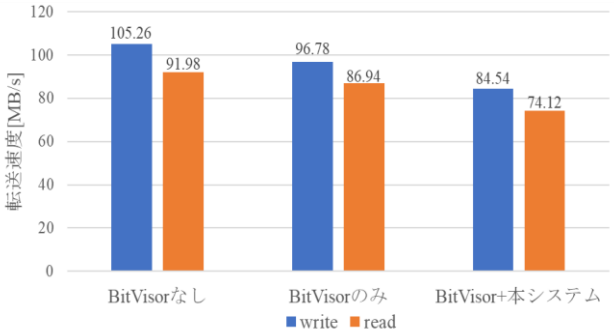


図 7　転送速度の計測結果

バへ転送する機構を実装する。HDFS サーバへの送信も、時刻情報の取得と同様に lwIP を使用し、HTTP によって HDFS サーバと通信ができる WebHDFS [10] というプロトコルを用いる。

転送するデータのポインタ、転送を行うデータの配列のインデックス、転送するデータサイズの 3 つの情報を取り出す。その後、転送用のスレッドを確立し、収集したデータの転送先を指定する。NameNode へ問い合わせることで、転送すべき DataNode が得られるので、そこへデータを書き込む。HTTP の正常終了を示すステータスコードを受け取ると、最後に収集データを格納している配列から転送した分だけクリアし、再び収集を始める。

4.8 監視機能のスイッチ

監視機能をゲスト OS からオンオフする機能を実現した。BitVisor ではゲスト OS から BitVisor に関する情報の取得や、BitVisor へ命令を与えることができる dbgsh という機能が実現されている。ゲスト OS から仮想マシンモニタへ機能を呼び出す機構は一般的に VMCALL と呼ばれ、dbgsh も VMCALL を用いて実現されている。dbgsh には、BitVisor から出力されたログを出力する log コマンドや、メモリダンプを出力する debug コマンドなど、いくつかのコマンドが用意されている。本システムはこの dbgsh へ収集開始と収集終了を実行するコマンドを追加した。ストレージアクセスパターンの収集開始と終了を、ゲスト OS から操作することができる。

5. 評価

5.1 転送速度

仮想化によるオーバヘッドが低いという点が、BitVisor を採用した 1 つの理由であった。収集によるストレージと OS 間の速度低下が顕著な場合、マルウェアによって動的解析環境下であることが特定される可能性がある。そこで、ストレージアクセスパターンを収集している場合と、通常時の転送速度を計測し比較した。表 1 へ計測環境を、図 7 へ転送速度の計測結果を示す。Hadoop のバージョンは 2.9.0 を利用した。収集システムそのものが与える転送速度への

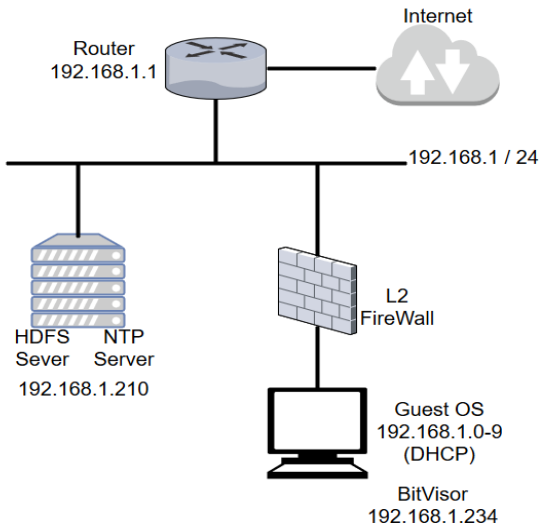


図 8　解析環境のネットワーク構成

影響を知るために、収集システムを有していない BitVisor についても同様に転送速度を測った。収集システムのバッファサイズが 64MiB であるため、dd コマンドを用いて 512MiB のファイルの読み書きを行い、転送速度を測定した。ゲスト OS には Debian GNU / Linux 8 を用いた。

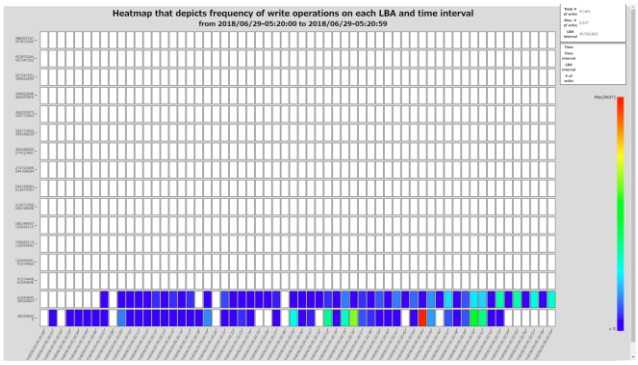
結果として OS からストレージへのデータの書き込み速度は 84.54 [MB/s] を記録し、ストレージから OS へのデータ読み込み速度は 74.12 [MB/s] となった。一般的なハードディスクによる転送速度と変わりない転送速度を実現することができた。まず、BitVisor がない通常のシステム（図 7 の左のグラフ）と転送速度を比較すると、書き込み時は約 20%、読み込み時は約 15% のオーバヘッドが発生した。次に、標準の BitVisor が動作している環境（図 7 の中央のグラフ）と比較すると、書き込み速度と読み込み速度が共に約 10% 低下した。BitVisor を用いたことや、収集システムにおける読み込み時に実データを保存しないこと、時刻同期の回数を減らす工夫をしたことによって、オーバヘッドが低い収集システムが実装できた。

5.2 アクセスパターンの可視化

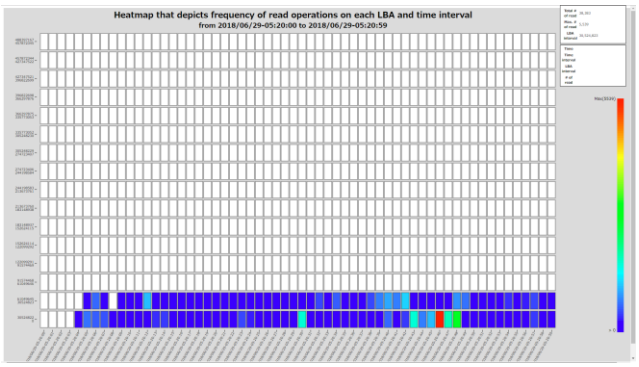
我々は WannaCry のようなストレージアクセスパターンに特徴を持つランサムウェアの流行を受けて、ストレージアクセスパターンを収集するシステムを実装した。実装したシステムを用いて WannaCry を動作させ、ストレージアクセスパターンを収集し、解析を行った。図 8 に解析環境のネットワーク構成を示す。本システムによって収集されるデータは HDFS サーバへと転送される。ゲスト OS はルータを介してインターネットへ接続する。解析環境は L2 ファイアウォールで隔離できるようになっており、ワーム活動を開始すると一定時間でネットワークを遮断する。

WannaCry のストレージアクセスパターンを収集することに加えて、定常時におけるストレージアクセスパターンを収集する。定常時の振る舞いとして今回はウェブブラウ

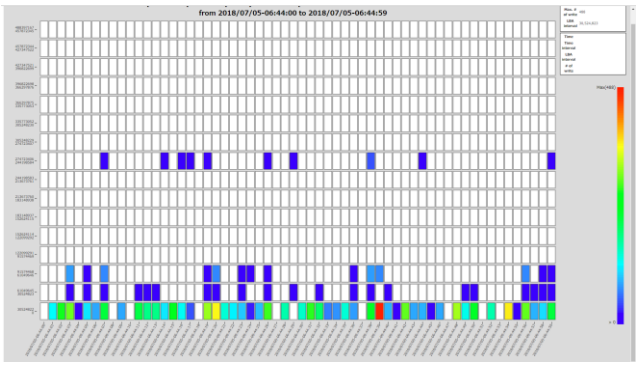
© 2018 Information Processing Society of Japan



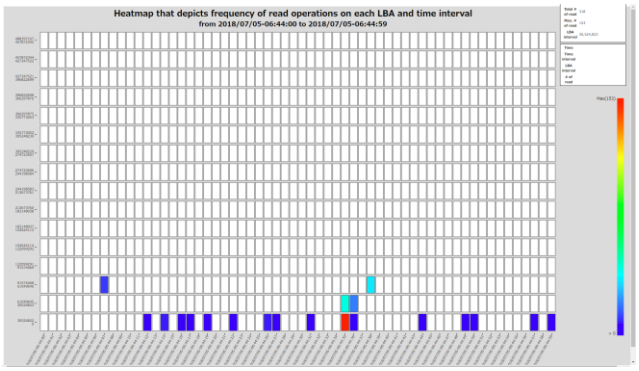
(a) WannaCry 動作時の書き込みの
ストレージアクセスパターン



(b) WannaCry 動作時の読み込みの
ストレージアクセスパターン



(c) ウェブブラウジング時の書き込みの
ストレージアクセスパターン



(d) ウェブブラウジング時の読み込みの
ストレージアクセスパターン

図 9　ストレージアクセスパターンの可視化結果
横軸は時刻、縦軸は Logical Block Address (LBA)、色はアクセス回数を示している（赤が書き込み回数が多い）

ジングを取り上げる。収集したアクセスパターンを比較することで WannaCry 特有の動作を考察する。自動ウェブブラウジングツールの Kantu Browser Automation [11] を用いてデータを収集する。Kantu Browser Automation は Chrome および Firefox 上のアドオンとして公開されている。使用したブラウザは Firefox である。

図 9 (a) に WannaCry 動作時の書き込みのアクセスパターンを、図 9 (b) に WannaCry 動作時の読み込みのアクセスパターンを、図 9 (c) に定常時の書き込みのアクセスパターンを、図 9 (d) に定常時の読み込みのアクセスパターンをそれぞれ示す。各グラフはヒートマップになっており、横軸に時刻、縦軸は LBA の範囲を示している。同じ LBA の範囲へのアクセス回数が増えると、マスの色が青から赤へと変化する。図 9 のグラフはすべて、1 分間の監視記録を可視化したものである。監視記録を WannaCry が実行された直後の 1 分間としたのは、できるだけ早く検知をするための特徴を発見するためである。

図 9 に示した結果は LBA の低い領域にアクセスが集中している。ストレージには Windows7 Professional SP1 と、BiVisor の監視切り替えに必要なプログラムのみをインストールした。ストレージに保存されているデータ量が少ないことと、監視時間が 1 分間と短いことから、一定の LBA 領域のみにアクセスが集中したと考えられる。

WannaCry はストレージ内のファイルに対して順々に暗号化を行う。暗号化前のファイルを読み込み、暗号化後のファイルを新たに書き込むため、特徴的なアクセスパターンが発生している様子が図 9 (a) と図 9 (b) から分かる。対して、ウェブブラウジング時のストレージアクセスパターン（図 9 (c)と図 9 (d)）を見ると、WannaCry 動作時ほどの一様なストレージアクセスではなく、離散的なアクセスが発生していることがわかる。ウェブブラウジング時の書き込みは、WannaCry 動作時と比較して LBA の低い部分へアクセスが多く発生しており、読み込みは WannaCry 動作時と比較してアクセスに連続性がない。

以上で述べたように、WannaCry 動作時とウェブブラウジング動作時ではストレージアクセスパターンが異なっていることが分かった。この違いを機械学習させることにより、将来的にマルウェアの自動検知ができる可能性がある。今

© 2018 Information Processing Society of Japan

後は収集システムを、機械学習データを収集するプラットフォームとして拡張する予定である。

重田ら [12] はランサムウェアを検知するための特徴量として、ファイルの読み書きが行われた回数や読み書きが発生したファイルの拡張子を取り上げている。一方、本稿で提案した収集システムは、ファイルシステムの情報を扱えない反面、ゲスト OS にかかわらずストレージのアクセスパターンからマルウェアを検出できる可能性がある。

6. おわりに

近年、亜種マルウェアの急速な増加が問題になっている。亜種マルウェアに対しては従来のパターンマッチング型の対策でなく、コンピュータの振る舞いからマルウェアが動作していることを推測しコンピュータを保護する、アノマリ型の対策が有効である。本稿では、WannaCry のようなストレージアクセスに特徴を持つマルウェアの流行を受けて、コンピュータのストレージアクセスパターンを収集するシステムを提案し、BitVisor を用いて実装した。

開発したシステムを評価した結果、OS からストレージへのデータの書き込み速度は 84.54 [MB/s] を、ストレージから OS へのデータ読み込み速度は 74.12 [MB/s] を記録し、実用的な速度を実現できた。開発したシステムで WannaCry を動作させ、定常時と比較した際のストレージアクセスパターンの違いを示した。さらに、WannaCry のようなストレージへの大量のアクセスが発生するランサムウェアに対して、収集システムが機能することを確認した。

今後は複数のマルウェアについてストレージアクセスパターンを収集し、WannaCry とその他のマルウェアを比較する。WannaCry と他のマルウェアにおけるストレージアクセスパターンに違いが表れれば、WannaCry を検知するためのデータセットとしての有用性を示すことができると考えている。さらに、ゲスト OS の種類によるストレージアクセスパターンの違いを調査するために、Windows 10 や MacOS でも収集システムを動作させ、データの収集と比較を行う予定である。

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 2018-06-20).

[2] 岩本一樹, 高田一樹, 津田侑, 遠峰隆史, 井上大介. マルウェアに実装されている仮想マシン検知機能の調査分析. コンピュータセキュリティシンポジウム 2017 論文集. 2017.

[3] Shinagawa, T., Eiraku, H., Tanimoto, K., Omote, K., Hasegawa, S., Horie, T., Hirano, M., Kourai, K., Oyama, Y., Kawai, E. et al.. BitVisor: a thin hypervisor for enforcing I/O device security. Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. 2009, pp. 121–130.

[4] Hirano, M. et al.. WaybackVisor: Hypervisor-Based Scalable Live Forensic Architecture for Timeline Analysis. International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage. 2017, pp. 219-230.

[5] Qi, Z., Xiang, C., Ma, R., Li, J., Guan, H., and Wei, D. S.. ForenVisor: A tool for acquiring and preserving reliable data in cloud live forensics. IEEE Transactions on Cloud Computing. 2017, pp. 443-456.

[6] Oyama, Y., Kawasaki, Y., and Takahashi, K.. Checkpointing an operating system using a parapass-through hypervisor. Journal of Information Processing. 2015, pp. 132-141.

[7] 大月勇人, 瀧本栄二, 齋藤彰一, 毛利公一. マルウェア観測のための仮想計算機モニタを用いたシステムコールトレース手法. 情報処理学会論文誌. 2014, 55(9), pp. 2034-2046.

[8] 山下雄也, 明田修平, 瀧本栄二, 齋藤彰一, 毛利公一. 仮想計算機モニタを用いた Windows10 64bit 環境におけるスタックトレースの実現. コンピュータセキュリティシンポジウム 2017 論文集. 2017.

[9] Shvachko, K., Kuang, H., Radia, S., and Chansler, R. The hadoop distributed file system. Proceedings of Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th symposium on. 2010, pp. 1-10.

[10] “The Apache Software Foundation WebHDFS REST API”. http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/WebHDFS.html, (参照 2018-08-02)

[11] “Kantu: Fresh Browser Automation plus Selenium IDE”. https://addons.mozilla.org/en-US/firefox/addon/kantu/, (参照 2018-07-04)

[12] 重田貴成, 伊沢亮一, 森井昌克, 井上大介, 中尾康二. ランサムウェア検知のための特徴解析. コンピュータセキュリティシンポジウム 2017 論文集. 2017.

© 2018 Information Processing Society of Japan