情報処理学会論文誌 Vol.55 No.9 2034–2046 (Sep. 2014)

# 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

大月 勇人[1,a)] 瀧本 栄二[1] 齋藤 彰一[2] 毛利 公一[1]

受付日 2013年12月2日, 採録日 2014年6月17日

**概要**：マルウェア対策には，マルウェアの挙動を調査する必要がある．しかし，次々に新種のマルウェアが出現しているため，個々のマルウェアの調査に時間を費やすことはできない．このような場合，実際にマルウェアを動作させて挙動を調査する動的解析が有効である．従来の解析システムは，処理速度が十分でなかったり，アンチデバッグ機能により解析していることをマルウェアに検出されるなどの課題を有していた．そこで，我々は，それらの課題を解決すべく動的解析システム Alkanet を開発した．Alkanet は，仮想計算機モニタ BitVisor をベースとしている．対象とするオペレーティングシステムは Windows XP とし，Windows の最小の実行単位であるスレッドを単位としてシステムコールをトレースする機能を実現している．また，トレースログを IEEE1394 を通じてリアルタイムに転送する機能を有し，トレースログを即座に別のコンピュータ上で処理することが可能である．本論文では，Alkanet の設計と実装について述べた後，Alkanet が，他の動的解析システムと比べて十分な処理速度で解析可能であること，典型的なアンチデバッグ機能に対して高い耐性を持つことなどを示す．

**キーワード**：マルウェア，動的解析，仮想計算機モニタ，システムコールトレース

# System Call Tracer based on Virtual Machine Monitor for Malware Analysis

Yuto Otsuki[1,a)] Eiji Takimoto[1] Shoichi Saito[2] Koichi Mouri[1]

Received: December 2, 2013, Accepted: June 17, 2014

***Abstract:*** Recently, malware has become a major security threat to computers. Responding to threats from malware requires malware analysis and understanding malware behavior. However, malware analysts cannot spend the time to analyze each instance of malware. The method is to execute the malware and observe its behavior. The major dynamic analysis systems often cannot analyze malware quickly. In addition, recent malware detects the systems by anti-debugging techniques. To solve these problems, We are developing Alkanet, a malware analyzer that uses a virtual machine monitor based on BitVisor. Alkanet executes malware on Windows XP, and traces system calls invoked by threads. Therefore, the system can analyze malware that infects other running processes. Also, the system call trace logs are obtained in real time via a IEEE 1394 interface. In this paper, we describe the design and implementation of Alkanet. We confirmed that Alkanet run faster than other dynamic analysis systems. We also confirmed that most of anti-debugging techniques cannot detect Alkanet.

***Keywords:*** malware, dynamic analysis, virtual machine monitor, system call tracing

## 1. はじめに

近年，日本国内でも特定企業・政府機関に対する標的型攻撃や，話題となった遠隔操作ウイルスなど，マルウェアの脅威が問題となっている．Symantec 社によれば，2011

[1] 立命館大学
Ritsumeikan University, Kusatsu, Shiga 525–8577, Japan
[2] 名古屋工業大学
Nagoya Institute of Technology, Nagoya, Aichi 466–8555, Japan
[a)] yotuki@asl.cs.ritsumei.ac.jp

© 2014 Information Processing Society of Japan

年だけで 4 億 300 万種を超えるマルウェアが新たに出現し，前年比 41%増となっている [1].

アンチウイルスソフトでマルウェアの駆除を行うためには，マルウェアの挙動を解析した結果に基づいて対処方法を決める必要がある．しかし，次々に新種のマルウェアが出現しているため，個々のマルウェアの調査に時間を費やすことはできない．マルウェアを解析する手法には，人手でマルウェアのコードを読み解く静的解析と，マルウェアを実際に実行して挙動を観察する動的解析がある．動的解析は静的解析と異なり，難読化やパッキングなどの影響を受けず，マルウェアの挙動を解析することができる．そこで，我々は，動的解析を用いた，短時間でマルウェアの解析が可能な技術の確立を目指している．

動的解析をするうえで課題となるのが，マルウェア自身が動的解析の対象となっていることを検知し，実行の停止や解析の妨害などを行うアンチデバッグ機能である [2], [3], [4]. そのため，仮想計算機モニタ（VMM）やエミュレータを用いて，マルウェア動作環境の外部に解析機能を実現する手法が用いられている [5], [6], [7], [8]．しかし，エミュレータは，システム全体をソフトウェアでエミュレートするため，オーバヘッドが大きく高速な解析には不向きである．また，一般的な VMM は，特定のハードウェアのエミュレーションや VMM とゲスト OS との通信用のインタフェースなど，実機にない特徴を有していることが多いため，マルウェアに検出されやすい．

さらに，マルウェアとして追跡する単位も課題である．2007 年から今なお猛威を奮う Zeus [9], [10] などは，活動の隠蔽や解析の困難化のために他のプロセスに対してスレッドの挿入を行う．プロセス単位で解析対象を区別している場合，通常のプロセスの中に作成された悪意あるスレッドとその他のスレッドを明確に区別できない．そのため，マルウェアの挙動のみを特定することが困難である．

上記の背景から，以下に示すような動的解析システムが必要である．

(1) オーバヘッドが少なく，解析速度の低下を軽減できる
(2) 解析に耐性を持つマルウェアでも解析できる
(3) 他のプロセスへ感染するマルウェアでも解析できる

以上の 3 点を満たすために，VMM を用いた動的解析システム Alkanet [11] を開発している．我々の既存論文 [11] と比較して，本論文の貢献は以下の点である．

- ネットワーク API に対応するシステムコールのトレースの実装を強化し，一般的な Windows Sockets API に対応したこと．
- VMM とゲスト OS 間のインタフェースの隠蔽機能を実装し，よりアンチデバッグに対する耐性を強化したこと．
- パフォーマンス評価を行い，Alkanet が従来の動的解析システムと比較して十分に高速であることを確認したこと．
- 典型的なアンチデバッグ手法に対する耐性評価を行い，実際に Alkanet が高い耐性を持つことを確認したこと．

以下，本論文では，2 章で Alkanet の概要と構成について，3 章で Alkanet によるシステムコールトレース手法について述べる．4 章で，PCMark05 を用いて Alkanet のベンチマーク評価を行った結果について述べる．5 章で，典型的なアンチデバッグ手法に対する耐性を評価した結果を述べる．6 章では，実際に Alkanet 上で動作させたマルウェアの解析結果と考察について述べる．さらに 7 章で関連研究について述べ，8 章で異なるバージョンの Windows に対して，Alkanet の適応性に関する考察を述べる．

## 2. Alkanet

### 2.1 基本アプローチ

#### 2.1.1 仮想計算機モニタ基盤

解析に耐性を持つマルウェアも解析でき，かつ，オーバヘッドを少なく抑えた動的解析システムを構築するために，準パススルー型の VMM である BitVisor [12] をベースとする．BitVisor では，ゲスト OS が実マシンに搭載されているハードウェアを認識するため，マルウェアに対して実環境とほぼ同等の環境を提供することができる．したがって，ハードウェア的特徴からマルウェアに検出されることはない．また，ハードウェアによる仮想化支援機構を利用するため，ソフトウェアのみで実現されたエミュレータや VMM に比べ，高速に動作することが可能である．これにより，解析時間の短縮，時間計測を用いたアンチデバッグの回避などが見込める．

ただし，BitVisor もゲスト OS との通信用のインタフェースを提供している．当該インタフェースの呼び出しには，Intel VT（Intel Virtualization Technology）[13] を用いる場合，vmcall 命令が用いられる．vmcall 命令は，VMM が動作していない状態で使用すると，#UD（未定義オプコード）例外が発生する．したがって，Alkanet では，vmcall 実行時に仮想計算機（VM）へ #UD 例外を発生させることで，BitVisor が動作していることを隠蔽する．

また，Windows 上で動作するマルウェアを解析するには，Windows 内部の情報を取得する必要がある．しかし，VMM と OS は互いに独立したソフトウェアであるため，VMM が OS 内部の情報を取得することは容易ではない [14], [15]. 既存の動的解析システムの多くは，OS 内部に情報提供用のプロセスやドライバを導入している．しかし，プロセスやファイルの存在を検査するマルウェアが存在するため，検出・攻撃されることがある．そこで，Alkanet では，Windows が使用するメモリ領域を直接参照し，独自に Windows のデータ構造の解釈を行う．

© 2014 Information Processing Society of Japan

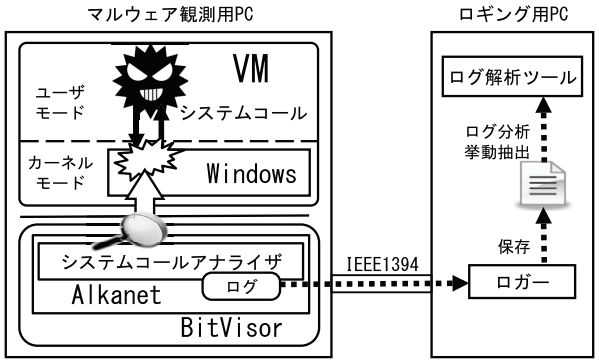


図 1 Alkanet の全体構成
**Fig. 1** Overview of Alkanet.

#### 2.1.2 解析の粒度

解析の粒度として細粒度のものには，機械語命令単位やメモリアクセス単位での解析が考えられる．細粒度の解析は，詳細な情報の取得が可能であるが，オーバヘッドが大きく，また，処理内容の理解に手間がかかる．粗粒度のものには，Windows API を記録する API トレースがある．API トレースは，API 呼び出し時にのみ処理をフックするため，細粒度のものと比べてオーバヘッドが小さい．また，実行された処理内容を API という意味のある単位でとらえるため，挙動の理解が容易である．しかし，マルウェアがより下位の API やシステムコールを直接呼び出した場合，フックを回避されてしまう．また，Software Breakpoint Detection や Stolen Bytes などのアンチデバッグ手法 [2], [3], [4] も存在する．これらは，API を提供する DLL がユーザ空間にマップされていることを利用し，フックの検出・回避を行う．

以上から，Alkanet は，マルウェアがシステムに影響を与えるためにシステムコールを必要とすることを利用して，中粒度のシステムコールをトレースする．具体的には，システムコールをフックし，その種類や引数などを記録する．システムコールのエントリポイントは，カーネルモードのメモリ領域に存在するため，ユーザモードのプロセスからはアクセスできず，また，エントリポイント以外から呼び出すこともできない．すなわち，マルウェアによる意図的なフックの回避は困難である．システムコールトレースのログは，API トレースのログと比較して抽象度が低いものの，機械語命令単位やメモリアクセス単位と比べれば抽象度が高く挙動の理解が容易である．以上から，ユーザモードのマルウェアを解析するうえで，システムコールをトレースすることは有効であると考えられる．これによって，オーバヘッドの軽減と処理内容の理解の容易さを両立可能とする．

### 2.2 構成

Alkanet の全体構成を**図 1** に示す．Alkanet は，2 台の PC を用いてマルウェアの観測と解析を行う．マルウェア観測用 PC 上では，マルウェアを動作させ，システムコールトレースを行う．ロギング用 PC では，ログを収集し，分析を行う．

Alkanet は，BitVisor 内部に拡張機能として実装されている．マルウェアの実行環境であるゲスト OS には，32 bit 版 Windows XP SP3 を用いている．Alkanet は，この環境で発行されるシステムコールをシステムコールハンドラの入口と出口でフックし，Alkanet 内部のシステムコールアナライザに制御を移し，その種類や引数などを取得・記録する．

Alkanet のログは，ロギング用 PC のロガーによって IEEE 1394 の Direct Memory Access を用いてロギング用 PC に転送される．このログ取得手法は，Alkanet による送信処理が不要であるため，Alkanet に負荷をかけずにログを取得できる．ログ取得用マシンのログ解析ツールは，Alkanet から取得したログから，システムコールのグループ化や，各種 ID やファイルパスなどを用いたログどうしの関連付け，別プロセスに挿入されたスレッドの追跡などを行う．

### 2.3 検査対象とするシステムコール

効率的にマルウェアの挙動を観測するためには，マルウェアが使用するシステムコールに絞って情報を取得する必要がある．たとえば，ファイルやレジストリの改竄には，ファイルやレジストリを読み書きするシステムコールが必要である．また，DoS 攻撃やスパムメールの送信を行う場合，ネットワークの操作を行うシステムコールの発行が必要である．典型的なマルウェアの機能は，以下のような挙動で構成される．

- ファイルのオープン，作成，読み出し，書き込み
- レジストリの参照，設定
- 仮想メモリの読み出し，書き込み，確保，権限の変更
- ファイルマッピングの作成，オープン，ビューの作成
- ネットワークの送信，受信
- プロセスの作成，終了
- スレッドの作成，終了，停止，再開，コンテキスト変更
- ドライバのロード，アンロード
- 処理時間の計測
- スリープ

Alkanet では，上記の挙動に対応するシステムコールに限定して，情報を取得し，ログを取得する．これにより，より高速な動的解析を実現する．Windows XP における具体的なシステムコールを**表 1** に示す．

我々の既存論文 [11] と比較して，ネットワークの挙動解析機能を強化している．具体的には，文献 [16] を参考に，NtDeviceIoControlFile システムコールの引数に与えられる Windows Sockets（Winsock）API 固有の制御コードやデータ構造の解釈に対応した．これにより，Winsock

© 2014 Information Processing Society of Japan

表 1 Alkanet が監視するシステムコール
Table 1 System calls observed by Alkanet.

| 挙動 | フックするシステムコールの例 |
|---|---|
| ファイル | NtCreateFile, NtOpenFile, NtWriteFile, NtReadFile, NtSetInformationFile, NtDeleteFile |
| レジストリ | NtCreateKey, NtOpenKey, NtQueryKey, NtSetInformationKey, NtDeleteKey, NtSetValueKey, NtQueryValueKey, NtDeleteValueKey |
| 仮想メモリ | NtWriteVirtualMemory, NtReadVirtualMemory, NtAllocateVirtualMemory, NtProtectVirtualMemory |
| ファイルマッピング | NtCreateSection, NtOpenSection, NtMapViewOfSection |
| ネットワーク | NtDeviceIoControlFile, NtCreateFile, NtReadFile, NtWriteFile |
| プロセス | NtCreateProcess, NtCreateProcessEx, NtTerminateProcess |
| スレッド | NtCreateThread, NtSuspendThread, NtResumeThread, NtTerminateThread, NtGetContextThread, NtSetContextThread |
| ドライバ | NtLoadDriver, NtUnloadDriver |
| 時間 | NtQueryPerformanceCounter |
| スリープ | NtDelayExecution |

API における bind，connect，accept，send，recv，sendto，recvfrom について IP アドレスやポート番号などの挙動の観測が可能である．

### 2.4 取得する情報

Alkanet は，システムコールのフック時に以下の情報を取得する．

- システムコール発行元の Cid と実行ファイル名
- システムコール番号
- システムコールの引数と戻り値

Cid は，プロセス ID とスレッド ID の組であり，ログ取得時におけるシステムコール発行元の区別に用いられる．また，その実行ファイルの名前も取得する．さらに，システムコールが示す挙動を調査するために，システムコールの引数と戻り値も取得する．

## 3. システムコールトレースの実装

システムコールトレースの実装には，以下の機能が必要である．以下，本章では，各機能の実装について述べる．

- システムコールのフック
- システムコールの特定
- プロセスとスレッドの特定
- 引数と戻り値の取得

### 3.1 システムコールのフック

32 bit 版 Windows XP SP3 におけるシステムコールは，通常 sysenter 命令によってカーネルモードへ遷移し，sysexit 命令でユーザモードへ遷移する．Alkanet は，sysenter 命令のジャンプ先である `KiFastCallEntry` と，sysexit 命令を実行する `KiSystemCallExit2` に，ハードウェアブレイクポイントを設定し，フックする．これらのアドレスは，公開されているシンボル情報を用いる [17]．なお，ブレイクポイントの設定に使用するデバッグレジスタは，デバッグレジスタに対する mov 命令をフックすることで，ゲスト OS から隠蔽する．ハードウェアブレイクポイントの使用の有無を確認するアンチデバッグ機能を持つマルウェアについては 5 章で述べる．設定したブレイクポイントによって，sysenter 命令実行時と sysexit 命令実行時にデバッグ例外が発生し，Alkanet に制御が移る．以下，本論文では，sysenter，sysexit 命令実行時のフックをそれぞれ sysenter フック，sysexit フックと呼称する．

### 3.2 システムコールの特定

Windows では，システムコール発行時にシステムコール番号が EAX レジスタに格納される．したがって，sysenter フックでは，EAX レジスタの値からシステムコールを特定する．sysexit フック時は，レジスタなどにシステムコール番号が格納されていないため，同様の方法が使えない．そこで，Alkanet では，システムコールが ntdll.dll に実装されているスタブを用いて発行されることを利用してシステムコールを特定する．スタブは，発行するシステムコールと同名のシンボルになっている．そのため，スタックに格納された戻りアドレスから実行されたスタブを求めることで，発行されたシステムコールが特定できる．

Windows では，システムコール発行時に ESP レジスタの値を EDX レジスタに格納する．一方，sysexit 命令は，ECX レジスタの値を ESP レジスタにロードする仕様である．したがって，Alkanet は，システムコールを発行したプロセスのスタックの位置を，sysenter フック時は EDX レジスタ，sysexit フック時は ECX レジスタからそれぞれ特定する．その後，スタックに格納された戻りアドレスを調べ，発行されたシステムコールを特定する．

### 3.3 プロセスとスレッドの特定

Alkanet は，システムコール発行元の Cid と実行ファイル名を収集するために，Windows の PCR（Processor Control Region）と呼ばれるデータ構造を用いる．PCR は，Windows のカーネルモードおいて各プロセッサの FS

© 2014 Information Processing Society of Japan

レジスタが指すセグメントの先頭に存在し，現在動作しているスレッドの情報を示すスレッドオブジェクトへのポインタを保持している．スレッドオブジェクトは，Cid と自身が属するプロセスオブジェクトへのポインタを保持している．実行ファイル名は，そのプロセスオブジェクトから取得できる．

### 3.4 引数と戻り値の取得

マルウェアの挙動を調査するためには，システムコールの種類に加えて，マルウェアがアクセスしたファイルやレジストリなどの操作対象の情報も必要である．これらの情報は，システムコールの引数から取得できる．また，Alkanet が監視対象とするシステムコールの戻り値は，NTSTATUS と呼ばれる状態を示す値であり，この値からシステムコールの成否および失敗した場合はその理由が分かる．

Windows の API では，stdcall 呼出規約 [18] が用いられる．stdcall では，引数はスタックに，戻り値は EAX に格納される．したがって，それらの値を sysenter フックと sysexit フックで取得する．

ただし，引数には，ポインタや OS 固有のデータ構造が用いられることが多く，その値だけでは不十分な場合がある．したがって，OS 領域やプロセス領域内にあるそれらのデータ構造を解釈し，必要な情報を取得する．

## 4. オーバヘッド評価

Alkanet が，小さなオーバヘッドで解析できることを確認するために，PCMark05 [19] の System Test Suite を用いて評価を行った．また，比較対象として，TTAnalyze [5] や Panorama [6] などのベースとなっている QEMU でも同様の評価を行った．

### 4.1 計測環境

**表 2** に示す評価用マシンで，Native，BitVisor，QEMU，Alkanet の 5 種類の環境において，PCMark05 による評価を行った．各環境の説明を以下に示す．

(1) **Native**
評価用マシンに直接インストールした Windows 上で評価する．

(2) **BitVisor**
BitVisor 1.2 のゲスト OS の Windows 上で評価する．Alkanet と同様に，CPU とメモリのみの仮想化を行い，その他のハードウェアへのアクセスはパススルー（直接アクセス可）する．BitVisor が使用するメモリ量を Alkanet に合わせて 512 MB に設定した．すなわち，VM は 3.5 GB のメモリを使用できる．

(3) **QEMU**
ホスト OS の 64 bit 版 Ubuntu 12.04.2 LTS（Linux kernel 3.5.0-26-generic）に QEMU 1.4.0 をインストールし，ゲスト OS の Windows 上で評価する．ただし，VM の CPU コアの数は 1 とした．これは，QEMU では VM の CPU コアの数を増加させても，ホスト OS で実行される QEMU プロセスのスレッドの数が増えず，実質使用できる物理 CPU が 1 つであるためである．また，VM に大量にメモリを割り当てるとホスト OS の動作を阻害する場合があるため，VM に割り当てるメモリ量を 1 GB とした．

(4) **Alkanet**
Alkanet のゲスト OS として動く Windows 上で評価した．なお，本評価では 5 章で述べるオーバヘッドの隠蔽手法は適用しない．また，Alkanet は 512 MB のメモリを必要とするため，VM は 3.5 GB のメモリを使用できる．

**表 2** 評価用マシンの構成
**Table 2** Specification of PC for evaluation.

| | |
|---|---|
| CPU | Intel Core 2 Quad Q6600 2.4 GHz |
| ゲスト OS | Windows XP SP3（32-bit） |
| Memory | 4 GB |

### 4.2 結果と考察

4.1 節で示した各環境において，PCMark05 の System Test Suite を実行した結果，および，System Test Suite に含まれるテスト項目について個別の比較を行うためにそれぞれ単体でテストを行った結果を**表 3** に示す．括弧内の数字は，Native の結果を 100 として正規化した値である．

BitVisor では，CPU とメモリのみを仮想化した．したがって，Native–BitVisor1.2 間での性能低下の要因の大半は，ページフォルトが発生したことによる VM Exit と VM Entry によるものである．

QEMU では，Physics and 3D，3D - Pixel Shader，Web Page Rendering において計測に失敗しており，これにともない，総合評価である System Test Suite の数値も N/A となっている．Physics and 3D，3D - Pixel Shader については，この他のグラフィックス関連の結果が Native の 10%以下となっていることと合わせて，QEMU では Native や BitVisor と異なり Graphics Card に直接アクセスできないため計測に失敗したと考えられる．Web Page Rendering は，複数の環境で検証したが計測に失敗するケースがあるため，QEMU 特有の問題と考えられる．一方，HDD - XP Startup や HDD - General Usage では，Native 環境の 3 倍～4 倍の性能になっている．これは，ホスト OS 側で HDD の内容がメモリ上にキャッシュされるためであると考えられる．本評価では QEMU に CPU を 1 つしか割り当てられないことも QEMU の性能が低下した要因であるが，一部の結果を除いて 90%前後性能が低下する QEMU

© 2014 Information Processing Society of Japan

表 3 PCMark05 によるベンチマーク結果
Table 3 Benchmark result by PCMark05.

| Benchmark | Native | BitVisor | | QEMU | | Alkanet | |
|---|---|---|---|---|---|---|---|
| System Test Suite (PCMarks) | 5,516 | 5,265 | (95) | N/A | (-) | 4,317 | (78) |
| HDD - XP Startup (MB/s) | 7.6 | 7.6 | (99) | 31.7 | (414) | 7.5 | (99) |
| Physics and 3D (FPS) | 172.4 | 138.7 | (80) | failed | (-) | 136.7 | (79) |
| Transparent Windows (windows/s) | 397.3 | 387.0 | (97) | 17.9 | (5) | 347.3 | (87) |
| 3D - Pixel Shader (FPS) | 70.8 | 71.3 | (101) | failed | (-) | 71.0 | (100) |
| Web Page Rendering (pages/s) | 3.8 | 2.6 | (68) | failed | (-) | 0.9 | (25) |
| File Decryption (MB/s) | 66.2 | 66.0 | (100) | 6.1 | (9) | 65.9 | (100) |
| Graphics Memory - 64 lines (FPS) | 631.4 | 630.3 | (100) | 7.2 | (1) | 639.8 | (101) |
| HDD - General Usage (MB/s) | 4.8 | 4.9 | (101) | 16.1 | (333) | 4.8 | (99) |
| Audio Compression (KB/s) | 3,097.2 | 3,087.9 | (100) | 58.7 | (2) | 3,023.3 | (98) |
| Video Encoding (KB/s) | 428.6 | 372.0 | (87) | 18.1 | (4) | 344.0 | (80) |
| Text edit (pages/s) | 146.5 | 142.4 | (97) | 9.4 | (6) | 78.8 | (54) |
| Image Decompression (Mpixels/s) | 34.3 | 34.3 | (100) | 3.7 | (11) | 34.3 | (100) |
| File Compression (MB/s) | 10.7 | 10.5 | (98) | 1.2 | (11) | 10.5 | (98) |
| File Encryption (MB/s) | 66.1 | 65.8 | (100) | 6.5 | (10) | 65.8 | (99) |
| HDD - Virus Scan (MB/s) | 71.2 | 69.3 | (97) | 232.5 | (327) | 65.0 | (91) |
| Memory Latency - Random 16 MB (Maccesses/s) | 9.8 | 9.7 | (100) | 2.8 | (28) | 2.8 | (28) |

は，高速な解析に不向きであるといえる．

Alkanet の評価結果では，Web Page Rendering，Text Edit，Memory Latency がそれぞれ Native から 75%，46%，72%の性能低下となった．Web Page Rendering および Text Edit では，GUI 関連のシステムコールの発行回数が多いという傾向を確認している．特に，GetKeyboardLayout や GetQueueStatus などの API で使用される NtUserCallOneParam システムコールの発行回数が多く，最も多い Text Edit は他のテスト項目の 10 倍を超えていた．当該システムコールは 1 回あたりの処理時間が短いため，フックのオーバヘッドによる処理時間の増加率が高くなり，スコアが低下したと考えられる．一方，Web Page Rendering では，NtUserCallOneParam システムコールだけでなく，レジストリアクセス用のシステムコールが他の項目に対して約 7 倍と突出していた．これは，当該テスト項目が Internet Explorer を実際に起動して，ベンチマークを計測するためである．レジストリはデータ構造が複雑であるため，レジストリアクセス用のシステムコールのログを取得する Alkanet では大きなオーバヘッドとなる．

Memory Latency のスコア低下の原因は，パフォーマンスカウンタの取得によるものと考えられる．パフォーマンスカウンタを取得する NtQueryPerformanceCounter システムコールの発行回数は，他のメモリテストと比較して約 150 倍であった．また，前述のように，処理時間が短いシステムコールほどフックによるオーバヘッドの増加率が高くなるため，大幅にスコアが低下したと考えられる．このことは，処理時間が長い I/O を実行するシステムコールが多数発行される HDD 関連のテスト項目では Alkanet によるフックとログ取得のオーバヘッドがほとんど影響していないことからも確認できる．

Alkanet は，全体のスコアでは Native と比較して 22%の低下であり，Alkanet のベースである BitVisor とでは 17%の低下であった．QEMU はほとんどのテストでスコアが 90%前後低下していることを考慮すると，Alkanet は QEMU をベースとする既存の動的解析システムと比較して十分に高速であると結論づけられる．

## 5. 耐解析手法に対する評価

### 5.1 評価方法

本章では，典型的なアンチデバッグ手法に対する耐性評価について述べる．対象とするアンチデバッグ手法は，Branco らによる分類 [4] と PoC（Proof of Concept）集 [20] に基づいて選択した．評価方法は，Alkanet 上で上記アンチデバッグ手法を実行し，Alkanet が検知されるか否かを評価する．

### 5.2 結果と考察

評価結果を**表 4** に示す．この結果より，Alkanet は，34 種類のアンチデバッグ手法に対して，32 種類で検知されずに解析可能であることを確認した．評価に用いたアンチデバッグ手法は，大別して以下のとおりに分類することができる．

(1) Windows が提供するデバッグ機能の使用の検出
(2) Windows 上の解析ツールの妨害
(3) 特定ツール（OllyDdg，VMWare など）の検出
(4) 時間計測による解析ツールの検出

Alkanet は VMM であり，Windows の提供するデバッグ機能などをいっさい用いないため，上記 (1)，(2) に該当

© 2014 Information Processing Society of Japan

表 4 耐解析手法に対する評価結果
**Table 4** Evaluation results against anti-debugging techniques.

| アンチデバッグの名称 | 回避可能か |
|---|---|
| (1) Windows が提供するデバッグ機能の使用の検出 | |
| PEB NtGlobalFlag | ○ |
| CheckRemoteDebuggerPresent | ○ |
| Debug Objects - ProcessDebugFlags Class | ○ |
| Debug Objects - ProcessDebugObjectHandle Class | ○ |
| Guard Pages | ○ |
| Hardware Breakpoint | ○ |
| Heap Flags | ○ |
| Instruction Counting | × |
| IsDebuggerPresent | ○ |
| NtQueryInformationProcess - ProcessDebugPort | ○ |
| NtQuerySystemInformation - SystemKernelDebuggerInformation | ○ |
| NtSetDebugFilterState | ○ |
| OpenProcess - SeDebugPrivilege | ○ |
| OutputDebugString | ○ |
| RtlQueryProcessDebugInformation | ○ |
| SS register | ○ |
| Software Breakpoint Detection | ○ |
| UnhandledExceptionFilter | ○ |
| (2) Windows 上の解析ツールの妨害 | |
| Alternative Desktop | ○ |
| BlockInput | ○ |
| DbgBreakpoint Overwrite | ○ |
| Self-Debugging | ○ |
| Self-Execution | ○ |
| SuspendThread | ○ |
| Thread Hiding | ○ |
| (3) 特定ツール（OllyDdg，VMWare など）の検出 | |
| CPU Instructions Results Comparison | ○ |
| Device Names | ○ |
| FindWindow | ○ |
| OllyDbg - OutputDebugString | ○ |
| Parent Process | ○ |
| SoftICE - Interrupt 1 | ○ |
| VMWare - IN Instruction | ○ |
| VirtualPC - Invalid Instruction | ○ |
| (4) 時間計測による解析ツールの検出 | |
| Execution Timing | △ |

○：回避可能，△：一部回避可能，×：回避不可

するアンチデバッグ手法によって検知されることはない．(3) に該当する手法についても，ハードウェアをパススルーするため，Instruction Counting を除く既存の手法で Alkanet を検知することはできない．本評価には含まれていないが，VMM の存在を検知する手法として vmcall 命令を発行する手法がある．この手法に対しても，Alkanet は 2.1.1 項で述べたとおり，vmcall 命令をフックして #UD 例外を発行することで回避可能である．

Instruction Counting は，デバッグ例外を利用する．Alkanet は，3.1 節で述べたようにデバッグ例外を使用するため現時点では検出されるが，vmcall 命令と同様に例外をエミュレートすることで回避可能である．または，システムコールフックの実装に，ソフトウェアブレイクポイントやステルスブレイクポイント [21] を利用することでも回避可能である．なお，Hardware Breakpoint では，デバッグレジスタを使用されていることを確認するが，3.1 節で述べたように，Alkanet が使用しているデバッグレジスタについては隠蔽しているため，この手法で検出されることはない．

(4) に関する手法（Execution Timing）では，時間計測の方法によって異なる結果となった．具体的には，直接 TSC（Time Stamp Counter）を参照する RDTSC 命令と，GetTickCount などの API を用いた時間計測では検知されなかった．これは，CPU の仮想化支援機構を用いていることと，Alkanet がシステムコール発行時にのみフックすることにより，それらの手法では Native 環境と同様の計測結果となるためである．一方，Alkanet を検知した手法では，QueryPerformanceCounter API が用いられていた．この API は，4.2 節でも述べた NtQueryPerformanceCounter システムコールを利用している．したがって，Alkanet による当該システムコールのフックにともなうオーバヘッドよって Alkanet が検知された．このようなアンチデバッグ手法に対しては，フックによって発生するオーバヘッド分だけ減算したカウンタ値に改竄することで対処できると考えている．

以上から，Alkanet は，PoC 集に掲載された代表的なアンチデバッグ手法のうち，34 項目中 32 項目，すなわち 94%に対して耐性があり，さらに未対応の例外のエミュレートおよびカウンタ値の改竄を施すことですべての手法に対する耐性を実現することが可能である．したがって，Alkanet は，アンチデバッグ手法に対して高い耐性を持つといえる．

## 6. システムコールトレース機能の評価

これまで Alkanet の開発を通して，Alkanet のマルウェア解析性能を確認するため，MWS2013 Datasets [22] で活動が記録されているマルウェアや独自で収集したマルウェアなどの解析を行ってきた．解析は Alkanet 上で実際にマルウェアを約 2 分間実行し [23]，ログ解析ツールによって得られた結果を解析結果とした．ただし，マルウェアが外部へ攻撃してしまうことを防ぐために，本章におけるマルウェアの解析はすべてネットワークに接続せずに行ったものである．解析結果の妥当性は，セキュリティベンダが提供するマルウェアレポートに基づいている[*1]．

以下，本章では，SdBot.exe のシステムコールトレースのログを用いて，システムコールトレースによりマルウェ

[*1] マルウェアには亜種が多く，セキュリティベンダごとに呼称も異なるため，複数のレポートで共通する挙動に基づいて判断した．

© 2014 Information Processing Society of Japan

```
No. :  1356      Time:  1697432035
Cid :  158.544   Name:  SdBot.exe
Type:  sysenter
SNo.:  b7 (NtReadFile)
Note:  \...\My Documents\SdBot.exe

No. :  1357      Time:  1697432105
Cid :  158.544   Name:  SdBot.exe
Type:  sysexit   Ret :  0 (STATUS_SUCCESS)
SNo.:  b7 (NtReadFile)
Note:  \...\My Documents\SdBot.exe                    (1)

No. :  4796      Time:  1697997485
Cid :  158.544   Name:  SdBot.exe
Type:  sysenter
SNo.:  25 (NtCreateFile)
Note:  \??\C:\WINDOWS\system32\ssms.exe

No. :  4800      Time:  1697998049
Cid :  158.544   Name:  SdBot.exe
Type:  sysexit   Ret :  0 (STATUS_SUCCESS)
SNo.:  25 (NtCreateFile)
Note:  \WINDOWS\system32\ssms.exe                     (2)

No. :  4806      Time:  1697999212
Cid :  158.544   Name:  SdBot.exe
Type:  sysenter
SNo.:  112 (NtWriteFile)
Note:  \WINDOWS\system32\ssms.exe

No. :  4807      Time:  1697999802
Cid :  158.544   Name:  SdBot.exe
Type:  sysexit   Ret :  0 (STATUS_SUCCESS)
SNo.:  112 (NtWriteFile)
Note:  \WINDOWS\system32\ssms.exe                     (3)
```

図 2 SdBot.exe：システムフォルダへのコピー
**Fig. 2** SdBot.exe: copying file of itself to system32 folder.

```
No. :  8652      Time:  1700450001
Cid :  65c.2c0   Name:  ssms.exe
Type:  sysenter
SNo.:  e0 (NtSetInformationFile)
Note:  DELETE: \...\My Documents\SdBot.exe

No. :  8653      Time:  1700450168
Cid :  65c.2c0   Name:  ssms.exe
Type:  sysexit   Ret :  0 (STATUS_SUCCESS)
SNo.:  e0 (NtSetInformationFile)
Note:  DELETE: \...\My Documents\SdBot.exe
```

図 3 SdBot.exe：自身のファイルの削除
**Fig. 3** SdBot.exe: deleting file of itself.

アの典型的な挙動を解析するに足る情報が取得できることを確認する．さらに，プロセスに感染するマルウェアの解析が可能であることを確認するために，Polipos.exe の解析結果を用いて検証する[*2]．最後に，これまでに Alkanet を用いて解析してきたマルウェアの解析結果の要約を述べ，さまざなマルウェアの解析に Alkanet が有効であることを示す．

### 6.1 SdBot.exe のシステムコールログ取得結果

SdBot.exe 実行時の Alkanet のログの一部を**図 2**，**図 3** に示す．各項目は以下のとおりである．

**No.** ログの通し番号
**Time** 記録を取った時点での CPU 時間
**Cid** システムコールを発行したスレッドの Cid
**Name** プロセスの実行ファイル名
**Type** sysenter 時のログか sysexit 時のログかを示す．
**Ret** 戻り値（sysexit のログのみ）
**SNo.** システムコールの番号とその名前
**Note** 引数を解析した結果などの付加情報

図 2 は，Sdbot.exe が自身の実行ファイルをシステムフォルダへ ssms.exe という名称でコピーを行った際のログである．ログより，SdBot.exe は Cid として 158.544 が割り当てられていることが分かる．図 2 中 (1) に示すログ No.1356，1357 より，SdBot.exe は，NtReadFile システムコールを発行して自身の実行ファイルを読み込んだことが分かる．その後，SdBot.exe は，システムフォルダである `C:\WINDOWS\System32` に ssms.exe という名前で自身のコピーを NtCreateFile システムコール（図 2 中 (2)）と NtWriteFile システムコール（図 2 中 (3)）によって作成した．これらのシステムコールは，sysexit 時の返り値が 0 (STATUS_SUCCESS) であることから，正常に行われたことが確認できる．コピーを作成した SdBot.exe は，その後コピーである ssms.exe を起動し，終了する．コピーである ssms.exe は，図 3 に示したとおり NtSetInformationFile システムコールによって本体である SdBot.exe を削除した．図 2，図 3 で示されたマルウェアのコピー作成と本体の削除という一連の動作は，発見されにくくするためにマルウェアが行う典型的動作の 1 つである．

他にも，Alkanet によって収集したログから，次のような挙動を確認することができた．これらは，セキュリティベンダのレポートと合致する [24], [25].

- アンチデバッグ
- レジストリ操作
- 他アプリケーションの起動

ここで，SdBot.exe は，アンチデバッグとして表 4 の Device Names に相当する方法で動的解析ツールの有無を確認していた．ただし，5 章でも述べたとおり，Alkanet は当該手法に対する耐性があるため，検知されずに解析を継続することができた．

以上から，Alkanet のログから，システムコールトレースによってマルウェアの典型的動作を解析するに十分たる情報が得られることが確認できた．

### 6.2 Polipos.exe の解析結果

Polipos あるいは Polip という名前で検出されるマルウェアは，動作しているほとんどのプロセスに感染し，自身の存在を隠蔽する特徴を持つ [26], [27]．6.1 節で述べた SdBot は，自身の複製を別プロセスとして起動することで，挙動の隠蔽を図っていたが，起動されたプロセスの挙動を追跡することで，SdBot の挙動の全容を把握可能であった．一方，Polipos が行うスレッド挿入は，正常なプロセスに潜り込むため，プロセス単位の追跡では挙動を把握できない．

[*2] マルウェアの名称は，あるアンチウイルスソフトにおける検出名を用いている．

© 2014 Information Processing Society of Japan

```
No. : 6340      Time: 689820959
Cid : bc.304    Name: Polipos.exe
Type: sysexit   SNo.: 35 (NtCreateThread)          (1)
Ret : 0 (STATUS_SUCCESS)
Note: Cid: b0.1e8, ProcessName: explorer.exe
-------------------------------------------------------
No. : 6384      Time: 691816285
Cid : bc.304    Name: Polipos.exe
Type: sysexit   SNo.: 35 (NtCreateThread)          (2)
Ret : c0000008 (STATUS_INVALID_HANDLE)
Note: PID: 4, ProcessName: System
```

図 4 Polipos：スレッド挿入
**Fig. 4** Polipos: thread injection.

```
Polipos.exe (54c.18c) -> svchost.exe (480.2c4)
  svchost.exe (480.2c4) -> svchost.exe (480.22c)
    svchost.exe (480.22c) -> svchost.exe (480.38c)
    svchost.exe (480.22c) -> svchost.exe (480.360)
    svchost.exe (480.22c) -> svchost.exe (480.720)
      svchost.exe (480.720) -> rundll32.exe (220.7f8)
```

図 5 Polipos：svchost.exe に挿入されたスレッドの追跡
**Fig. 5** Polipos: tracing a injected thread in svchost.exe.

そこで，Alkanet のログを解析し，別のプロセス内部に挿入されたスレッドの挙動を追跡できるか確認した．

**図 4** は，Polipos.exe が，NtCreateThread システムコールを発行し，別のプロセスである explorer.exe と System に対してスレッドを作成する挙動の sysexit のログである．ログの各項目は 6.1 節で述べたものと同様である．No.6340 のログ（図 4 中 (1)）では，NtCreateThread システムコールが成功し，explorer.exe 内にスレッド ID 1e8 のスレッドが作成されたことが分かる．その後，このスレッドは，Polipos.exe を存在しないユーザのゴミ箱に複製する挙動やネットワークへの接続を試みる挙動などを示した．一方，No.6384 のログ（図 4 中 (2)）では，戻り値が STATUS_INVALID_HANDLE となっており，このシステムコールは失敗していることが分かる．この場合，実際にスレッドは作成されていないため，この後の System プロセスの挙動を分析する必要はない．

上記のようなスレッド挿入で作成されたスレッドを Cid を基にログを関連づけていくことで，別のプロセスに作成されたスレッドの追跡が可能である．ログを解析することによって得られた追跡結果の一部を**図 5** に示す．図 5 中の 1 行目は，Polipos.exe の Cid 54c.18c スレッドによって，プロセス ID 480 の svchost.exe に Cid 480.2c4 のスレッドが作成されたことを示す．この Cid 480.2c4 のスレッドは，さらに Cid 480.22c のスレッドを作成する（図 5 中の 2 行目）．同様に，Cid 480.22c のスレッドは，Cid 480.38c，480.360，480.720 などの複数のスレッドを作成している（図 5 中の 3〜5 行目）．さらに，図 5 中の 6 行目において，Cid 480.720 のスレッドは，rundll32.exe に Cid 220.7f8 のスレッドを挿入していることが確認できる．同様に，Cid 220.7f8 のスレッドからさらに派生するスレッドも追跡可能である．したがって，Alkanet のログから，別のプロセスに挿入されたスレッドを正確に追跡可能であると結論づけられる．

### 6.3 多様なマルウェアに対する有効性の評価

本節では，まず，マルウェア 33 体について，各セキュリティベンダのレポートに記述されたシステムに影響を与え得る挙動を Alkanet で観測できたかについて述べ，Alkanet が多くのマルウェアに対して有効であることを示す．ここで対象とする挙動は，ファイル，レジストリ，プロセス，スレッド，ドライバ，他プロセスへ感染の 6 項目である．なお，本評価では古いバージョンの Alkanet による解析結果を含んでいるため，ネットワークの挙動については除外している．

解析したマルウェアの検出名，検体数および各項目における解析結果の成否を**表 5** に示す．検体 33 体は，MWS 2013 Datasets の CCC DATAset 2012 内に活動が記録されている実行可能なマルウェアの中から，カスペルスキー社のアンチウィルスによる検出名で網羅するように選別した．ここで，Net-Worm.Win32.Allaple.a，Net-Worm.Win32.Allaple.b，Net-Worm.Win32.Allaple.e の数が多くなっているが，CCC DATAset 2012 の大半をこれらが占めていたためである．また，33 検体の内，Backdoor.Win32.IRCBot.kzr などの 5 検体は，カーネルモードへ侵入する挙動を持つ．これらについては，カーネルへ侵入するまでを解析対象とし，それまでに観測された挙動がレポートの内容を鑑みて妥当であるかを確認した．

表 5 に示す解析結果において，Alkanet は，対象の検体すべてで各レポートに記述された挙動と同等の挙動が観測できた．特に，33 検体の内 13 体（39%），検出名別では 16 種類中 10 種類（63%）のマルウェアが，他プロセスに感染する挙動を持つが，Alkanet はそのすべてで当該挙動を観測することが可能であった．この結果から，Alkanet が多くのマルウェアについて挙動を解析するに足る情報を得ることが可能であると結論づけられる．

さらに，ネットワークに対する挙動を解析する機能の評価として，Palevo.exe，Waledac.exe の解析結果について述べる．本評価においても，前述の理由により，ネットワークには接続していない環境で解析を行った．両者とも，ファイルやレジトリ，プロセスなどの挙動について，レポートと合致する観測結果が得られた．Palevo.exe の解析では，8800 番ポートへ複数回 connect を試みる挙動が観測された．Waledac.exe の解析では，53 番および 80 番ポートに bind する挙動や，複数の IP アドレスの 80 番ポートに対して connect を試みる挙動が観測された．これらの観測結果は，セキュリティベンダのレポートと合致する．以上から，Alkanet がマルウェアのネットワークに対する挙動を観測が可能であることが確認できた．33 体の解析結果とあわせて，Alkanet はマルウェアの挙動を網羅できているといえる．

© 2014 Information Processing Society of Japan

表 5 マルウェア 33 体の解析結果
Table 5 Analysis for 33 malware samples.

| 検出名（カスペルスキー） | 検体数 | ファイル | レジストリ | プロセス | スレッド | ドライバ | 他プロセスへの感染 |
|---|---|---|---|---|---|---|---|
| (1) ユーザモードのみで動作するもの | | | | | | | |
| Backdoor.Win32.EggDrop.cpc | 1 | ○ | ○ | - | - | - | ○ |
| Backdoor.Win32.EggDrop.cpe | 1 | ○ | ○ | - | - | - | ○ |
| Backdoor.Win32.IRCBot.jwy | 2 | ○ | ○ | ○ | - | - | ○ |
| Backdoor.Win32.IRCBot.xu | 1 | ○ | ○ | ○ | - | - | - |
| Backdoor.Win32.Rbot.bni | 2 | - | ○ | - | ○ | - | ○ |
| Net-Worm.Win32.Allaple.a | 5 | ○ | ○ | - | ○ | - | - |
| Net-Worm.Win32.Allaple.b | 8 | ○ | ○ | ○ | ○ | - | - |
| Net-Worm.Win32.Allaple.e | 4 | ○ | ○ | ○ | ○ | - | - |
| Trojan-Dropper.Win32.Injector.bslj | 1 | ○ | ○ | - | - | - | ○ |
| Trojan.Win32.Genome.rioo | 1 | - | ○ | - | ○ | - | - |
| Trojan.Win32.VBKrypt.imgt | 1 | ○ | ○ | - | - | - | ○ |
| Worm.Win32.Ngrbot.jit | 1 | ○ | ○ | - | - | - | ○ |
| (2) カーネルへの侵入を試みるもの | | | | | | | |
| Backdoor.Win32.IRCBot.kzr | 1 | ○ | ○ | - | - | ○ | - |
| Backdoor.Win32.Rbot.adqd | 2 | ○ | ○ | - | - | - | ○ |
| Trojan.Win32.Inject.achx | 1 | ○ | ○ | ○ | - | ○ | ○ |
| Virus.Win32.Virut.av | 1 | ○ | ○ | - | - | - | ○ |

○：観測できた，-：レポートに記載なし

## 7. 関連研究

TTAnalyze [5] は，エミュレータである QEMU [28] をベースとし，マルウェアの呼び出すシステムコールや API を挙動を要約したレポートを出力するマルウェア解析システムである．しかし，QEMU をベースとしているため，高速な解析に不向きである．また，QEMU はこれまで解析環境に広く使われてきた背景から，マルウェアに検出され，正しく解析できない場合がある [29]．Alkanet は，特定のハードウェアをエミュレートしないため，同様の手法で検出されることはない．また，TTAnalyze では，CR3 の値をベースにプロセスレベルでマルウェアを区別するため，別のプロセスに挿入されたスレッドを追跡できない．Alkanet は，スレッドレベルでマルウェアを区別するため，別のプロセス内に潜む悪意あるスレッドの挙動も追跡できる．

Panorama [6] は，プライバシ情報を漏洩するようなマルウェアを自動的に分類および検出するシステムである．個人情報などへのアクセスやその検体の動作をモニタ・記録するために，システム全体に対してテイント解析を行う．Alkanet は，マルウェアの挙動全般やスレッド挿入により他のプロセスへ感染するような挙動を解析することを目的としている．また，Panorama には，QEMU ベースであることや内部に情報取得用のモジュールが動作している点など，マルウェアに検出されやすい要素が存在している．

Ether [7] は，VMM である Xen [30] をベースとしたマルウェア解析のためのフレームワークである．Ether は，実行命令，メモリ書込み，システムコールのフックを実装している．これにより，命令レベルのトレースとシステムコールトレースが実現できる．しかし，Ether はあくまでフレームワークであるため，実際にマルウェア解析を行うためには，Ether の提供する機能を基にマルウェア解析のための機能を実現する必要がある．Ether フレームワークを用いたシステムコールトレースを行う実装として EtherTrace がある．EtherTrace については，既存の QEMU などの VM を検知するマルウェアに対する有効性などが確認されている．しかし，他のプロセスへ感染するマルウェアに対する有効性は示されていない．Alkanet は，このようなマルウェアの解析を可能とする．

Nitro [8] は，KVM [31] をベースとしたゲスト OS に依存しないシステムコールトレーサである．ゲスト OS に依存しないために，仮想ハードウェアから取得できる情報のみを用いて解析を行う．OS 内部の情報を得たい場合は，他のメモリ解析ツールと連携することを想定している．すなわち，Nitro は，OS 固有のデータ構造を解釈しないため，取得できる情報が限定されてしまい，そのままマルウェア解析に使用するのは困難である．特にプロセスを実行単位とするため，スレッド挿入の挙動を解析するのは困難である．Alkanet は，実際にマルウェアの解析に使用できるように詳細な内部の情報まで取得する．スレッドレベルでマルウェアを追跡できるため，スレッド挿入の挙動も解析できる．

## 8. Windows のバージョンに関する考察

本論文で述べてきた Alkanet は，32 bit 版 Windows XP

© 2014 Information Processing Society of Japan

SP3 をマルウェア解析環境としている．本章では，異なるバージョンの Windows への適応性に関する考察を述べる．2.1.1 項で述べたように，Alkanet は Windows の使用するメモリ領域から情報を得る．この手法は，情報提供用プロセスなどを用いる手法と比べて，Windows の提供する API を用いないため，別のバージョンの Windows に対応する場合に実装コストが大きくなる．しかし，Windows はソースコードが公開されていない OS であるが，各種公式文書やシンボル情報などから内部のデータ構造や処理の流れなどを詳細に把握することが可能である．

バージョンの異なる Windows に対応する際に追加や変更が必要な点について，3 章で述べた以下の 4 つの項目ごとに述べる．

(1) システムコールのフック
(2) システムコールの特定
(3) プロセスとスレッドの特定
(4) 引数と戻り値の取得

上記 (1)，(3)，(4) については，対応する Windows が 32 bit 版である場合には，後述するデータ構造解釈機能を除き，本論文で述べた実装が使用できる．一方，64 bit 版の場合には，新規の実装が必要である．これについては，現在移植を進めている．

上記 (2) については，Windows のバージョンごとにシステムコールの番号やスタブのアドレスなどが必要である．これらはシンボル情報から調査できる．また，(3) について，PCR やスレッドオブジェクトなどについては，Windows のバージョンごとに各種データ構造の解釈機能の実装が必要である．(4) についても同様に，監視対象のシステムコールに対してそれぞれ個別の処理と，引数に応じたデータ構造解釈機能が必要となる．2.3 節で述べたとおり，Alkanet は，マルウェアの挙動 10 種類を観測するために，36 個のシステムコールおよび 7 個の Winsock API を監視対象としている．Windows ではサービスパック単位でシステムコールの追加や変更などが行われる．したがって，別のバージョンの Windows に対応するためには，そのバージョンにおける Alkanet が対象とする 10 種類の挙動に使用されるシステムコールを調査する必要がある．

システムコール個別の処理およびデータ構造解釈機能について，実際に実装コストが発生するケースは，既存のシステムコールやデータ構造に変更がある場合と，新しいものが追加された場合の 2 つである．前者においては，変更前の実装から流用できる部分が多いと考えられるため，少ない実装量で抑えられる．また，Alkanet のデータ構造解釈機能の一部はシンボル情報を基に生成しているため，データ構造のバージョン間の差が軽微であれば，この部分で吸収できる．後者については，基本的に新規に実装する必要があるが，実際には既存の実装から流用できる部分が多く，実装量を軽減できると考えられる．これは，Alkanet の対象とする挙動の範囲であれば，すでに対応済みのデータ構造を使うことが多いと考えられるためである．

## 9. おわりに

本論文では，アンチデバッグ機能を持つマルウェアでも解析を可能とするために，仮想計算機モニタを用いたマルウェアの動的解析を実現する Alkanet の設計と実装について述べた．マルウェアによく用いられるアンチデバッグ手法について評価し，デバッグ例外を用いるものと NtQueryPerformanceCounter システムコールを用いて時間計測を行うものを除く，すべての手法を回避できることを確認した．上記の 2 つについても対策が可能である．また，PCMark05 によるベンチマークテストを行い，Alkanet は実環境の 78%の性能であり，QEMU をベースとする動的解析システムより高速であることを示した．さらに，実際に Alkanet を用いてマルウェアを解析できることおよびマルウェアが別のプロセスに挿入したスレッドについても正確に追跡が可能であることを示した．

現在，Alkanet が対象とするマルウェア動作環境を，Windows XP から，64 bit 版 Windows 7 SP1 へ変更するための検討・調査を進めている．

### 参考文献


[1] Wood, P., Egan, G., Haley, K., et al.: Internet Security Threat Report Vol.17, Technical Report, Symantec Corporation (2012).
[2] Falliere, N.: Windows Anti-Debug Reference, Symantec Corporation (online), available from ⟨http://www.symantec.com/connect/articles/windows-anti-debug-reference⟩ (accessed 2013-11-29).
[3] Yason, M.V.: The Art of Unpacking, *Black Hat USA 2007* (2007).
[4] Branco, R.R., Barbosa, G.N. and Neto, P.D.: Scientific but Not Academical Overview of Malware Anti-Debugging, Anti-Disassembly and Anti-VM Technologies, *Black Hat USA 2012* (2012).
[5] Bayer, U., Kruegel, C. and Kirda, E.: TTAnalyze: A Tool for Analyzing Malware, *5th European Institute for Computer Antivirus Research* (*EICAR 2006*) *Annual Conference* (2006).
[6] Yin, H., Song, D., Egele, M., Kruegel, C. and Kirda, E.: Panorama: Capturing system-wide information flow for malware detection and analysis, *Proc. 14th ACM Conference on Computer and Communications Security*, *CCS '07*, pp.116–127, ACM (2007).
[7] Dinaburg, A., Royal, P., Sharif, M. and Lee, W.: Ether: Malware Analysis via Hardware Virtualization Extensions, *Proc. 15th ACM Conference on Computer and Communications Security*, pp.51–62, ACM (2008).
[8] Pfoh, J., Schneider, C. and Eckert, C.: Nitro: Hardware-based System Call Tracing for Virtual Machines, *The 6th International Workshop on Security* (*IWSEC 2011*) (2011).
[9] Falliere, N. and Chien, E.: Zeus: King of the Bots, White Paper, Symantec Corporation (2009).
[10] Internet Initiative Japan Inc.: Internet Infrastructure


© 2014 Information Processing Society of Japan

Review Vol.16, Technical Report 16, Internet Initiative Japan Inc. (2012).
[11] Otsuki, Y., Takimoto, E., Kashiyama, T., et al.: Alkanet: A Dynamic Malware Analyzer based on Virtual Machine Monitor, *World Congress on Engineering and Computer Science 2012* (*WCECS 2012*), Vol.1, pp.36–44 (2012).
[12] Shinagawa, T., Eiraku, H., Tanimoto, K., et al.: BitVisor: A thin hypervisor for enforcing i/o device security, *Proc. 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pp.121–130, ACM (2009).
[13] Intel Corporation: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3B: System Programming Guide, Part 2 (2009).
[14] Garfinkel, T. and Rosenblum, M.: A Virtual Machine Introspection Based Architecture for Intrusion Detection, *Proc. Network and Distributed Systems Security Symposium* (2003).
[15] Nance, K., Bishop, M. and Hay, B.: Virtual Machine Introspection: Observation or Interference?, *Security & Privacy*, Vol.6, No.5, pp.32–37, IEEE (2008).
[16] Petritsch, H.: *Network Virtualisation*, VDM Verlag Dr. Mueller e.K. (2008).
[17] Microsoft：デバッグツールとシンボル：はじめに，入手先 ⟨http://msdn.microsoft.com/ja-jp/windows/hardware/gg462988⟩（参照 2013-11-29）.
[18] Microsoft: __stdcall, available from ⟨http://msdn.microsoft.com/en-us/library/zxk0tw93.aspx⟩ (accessed 2013-11-29).
[19] Futuremark Corporation: PCMark05 - PC benchmark test for Windows XP from Futuremark, Futuremark Corporation (online), available from ⟨http://www.futuremark.com/benchmarks/pcmark05/⟩ (accessed 2012-12-10).
[20] Branco, R.R.: rrbranco/blackhat2012 - GitHub, GitHub, Inc. (online), available from ⟨https://github.com/rrbranco/blackhat2012⟩ (accessed 2013-11-29).
[21] Vasudevan, A. and Yerraballi, R.: Stealth Breakpoints, *Computer Security Applications Conference, 21st Annual*, pp.10–392 (2005).
[22] 神薗雅紀，畑田充弘，寺田真敏ほか：マルウェア対策のための研究用データセット—MWS Datasets 2013，マルウェア対策研究人材育成ワークショップ 2013（MWS2013）(2013).
[23] 青木一史，川古谷裕平，岩村　誠，伊藤光恭：動的解析における検体動作時間に関する検討，コンピュータセキュリティシンポジウム 2010（CSS2010）論文集，pp.543–548 (2010).
[24] McAfee, Inc.: W32/Sdbot.worm - Malware - McAfee Labs Threat Center, available from ⟨http://www.mcafee.com/threat-intelligence/malware/default.aspx?id=100454⟩ (accessed 2013-11-29).
[25] Symantec Corporation: Backdoor.Sdbot Technical Details — Symantec, available from ⟨http://www.symantec.com/en/us/security_response/writeup.jsp?docid=2002-051312-3628-99&tabid=2⟩ (accessed 2013-11-29).
[26] Microsoft: Virus:Win32/Polip.A, available from ⟨http://www.microsoft.com/security/portal/threat/encyclopedia/entry.aspx?Name=Virus:Win32/Polip.A#tab=2⟩ (accessed 2013-11-29).
[27] Symantec Corporation: W32.Polip Technical Details — Symantec, available from ⟨http://www.symantec.com/security_response/writeup.jsp?docid=2006-042309-1842-99&tabid=2⟩ (accessed 2013-11-29).
[28] Bellard, F.: QEMU, a Fast and Portable Dynamic Translator, *Proc. Annual Conference on USENIX Annual Technical Conference*, pp.41–46, USENIX Association (2005).
[29] Bayer, U., Habibi, I., Balzarotti, D., Kirda, E. and Kruegel, C.: A View on Current Malware Behaviors, *Proc. 2nd USENIX Conference on Large-scale Exploits and Emergent threats: Botnets, Spyware, Worms, and More*, pp.8–19, USENIX Association (2009).
[30] Barham, P., Dragovic, B., Fraser, K., et al.: Xen and the Art of Virtualization, *Proc. 19th ACM Symposium on Operating Systems Principles*, pp.164–177, ACM (2003).
[31] Red Hat, Inc.: Main Page - KVM, available from ⟨http://www.linux-kvm.org⟩ (accessed 2013-11-29).

**大月 勇人**　（学生会員）

1988 年生．2011 年立命館大学情報理工学部情報システム学科卒業，2013 年同大学大学院理工学研究科博士前期課程情報理工学専攻修了．同年同大学大学院情報理工学研究科博士後期課程情報理工学専攻に入学，現在に至る．オペレーティングシステム，仮想化技術，コンピュータセキュリティ等の興味を持つ．

**瀧本 栄二**　（正会員）

1976 年生．1999 年立命館大学理工学部情報学科卒業，2001 年同大学大学院理工学研究科博士前期課程修了，2005 年同大学大学院研究科博士後期課程単位取得退学，同年（株）ATR 適応コミュニケーション研究所専任研究員，2010 年立命館大学情報理工学部情報システム学科助手，現在に至る．主にシステムソフトウェア，無線通信に関する研究に従事．

© 2014 Information Processing Society of Japan

**齋藤 彰一** （正会員）

1993 年立命館大学理工学部情報工学科卒業．1995 年同大学大学院博士前期課程修了．1998 年同大学大学院博士後期課程単位習得中退．同年和歌山大学システム工学部情報通信システム学科助手．2003 年同講師，2005 年同助教授．2006 年名古屋工業大学大学院助教授，2007 年同准教授．現在に至る．オペレーティングシステム，インターネット，セキュリティ等の研究に従事．博士（工学），ACM，IEEE-CS 各会員．

**毛利 公一** （正会員）

1994 年立命館大学理工学部情報工学科卒業，1996 年同大学大学院理工学研究科修士課程情報システム学専攻修了，1999 年同大学大学院研究科博士課程後期課程総合理工学専攻修了．同年東京農工大学工学部情報コミュニケーション工学科助手，2002 年立命館大学理工学部情報学科講師，2004 年同大学情報理工学部情報システム学科講師，2008 年同准教授，2014 年同教授となり，現在に至る．博士（工学）．オペレーティングシステム，仮想化技術，コンピュータセキュリティ等の研究に従事．電子情報通信学会，ACM，IEEE-CS，USENIX 各会員．

© 2014 Information Processing Society of Japan