Computer Security Symposium 2017
23 - 25 October 2017

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

岩本 一樹[1,a)]　高田 一樹[1]　津田 侑[2]　遠峰 隆史[2]　井上 大介[2]

概要：マルウェアの動的解析は静的解析に比べて短い時間でマルウェアの動作概要を知ることができる．そのため初動の対応として有効である．しかし，仮想マシンを検知することで動的解析を妨げる機能を有するマルウェアが存在する．ゆえにマルウェアの仮想マシン検知機能を無効化して動的解析するシステムが必要である．システム構築のために，まずマルウェアに実装されている仮想マシン検知機能についての情報を収集する．本研究では実際のマルウェアを静的解析した結果および文献から，仮想マシン検知機能を分類し，それらへの対応方法を提案する．

キーワード：マルウェア, アンチ VM, 仮想マシン, 動的解析

Survey Analysis of Anti Virtual Machine Functions in Malicious Software

Kazuki Iwamoto[1,a)]　Kazuki Takada[1]　Yu Tsuda[2]　Takashi Tomine[2]　Daisuke Inoue[2]

Abstract: Dynamic analysis gives an outline of malware shorter time than static analysis. It is effective as initial response. However, some of malware have a function to prevent analyst from dynamic analysis by detecting virtual machine. Therefore, we need a dynamic analysis system which disable the virtual machine detection. For building the system, we survey functions of the virtual machine detection implemented in malware. In this paper, we propose the anti detection methods from the results of static analysis of wild malware and related works.

Keywords: Malware, AntiVM, Virtual Machine, Dynamic Analysis

1. はじめに

標的型攻撃ではマルウェアや攻撃ツールなどを利用して，対象の機器を外部から制御可能にし，ネットワーク越しに情報を窃取する手法が一般的である．そのため，標的型攻撃に用いられるマルウェアなどの動的解析を行う際には，それらのプロセスの活動とプロセスが行った通信に着目して解析を行う．一方，ネットワークを利用した標的型攻撃の解析では，ネットワークを流れるパケットを収集し，その中に攻撃とみられる通信や悪意のあるプログラムが存在するかという点に着目して解析を行う．しかし，現在ではそれぞれの解析は別に行われており，相互の関連を示す情報が不足していた．

我々は，以上のような背景から，標的型攻撃の解析を行うにあたって，マルウェアのプロセスの動作およびそれが行う通信の双方を紐付けて記録するシステム「NetMonitor」[1] を研究開発してきた．しかし，一般的にマルウェア自身が仮想マシンで構築された解析環境であることを検知し，解析を妨げる機能を有すことが知られており，多くの解析環境で動作させるためにはマルウェアが持つ仮想マシン検知機能を可能な限り回避する必要がある．

そこで本研究ではマルウェアが持つ仮想マシン検知機能を可能な限り回避可能な NetMonitor の実現に向けて調査分析を実施する．本研究ではまずシステム構築のために，

[1] 株式会社セキュアブレイン
SecureBrain Corporation
[2] 国立研究開発法人 情報通信研究機構
National Institute of Information and Communications Technology
[a)] kazuki_iwamoto@securebrain.co.jp

表 1　調査分析に用いた環境

仮想マシン	CPU	Intel Xeon E312xx 2.40 GHz
	ホスト OS	Ubuntu 12.04 LTS
	仮想環境	KVM + QEMU
		Compiled against library: libvir 0.9.8
		Using library: libvir 0.9.8
		Using API: QEMU 0.9.8
		Running hypervisor: QEMU 1.0.0
	ゲスト OS	Windows 7 Professional SP1 32bit
		Windows 7 Professional SP1 64bit
		Windows Update 停止，Firewall 停止
実機	CPU	Intel Core 2 Duo E6600 2.40GHz
	OS	Windows 7 Professional SP1 32bit
	CPU	Intel Core i5 5200U 2.20GHz
	OS	Windows 7 Professional SP1 64bit
解析ツール		OllyDbg, WireShark,
		Process Explorer

マルウェアの仮想マシン検知機能について調査を行い，検知対象と検知の技術的方法に基づいて分類する．本研究のために静的解析した 12 種類のマルウェア，既に解析済みのマルウェア，および 6 つの文献で言及されている仮想マシン検知機能を**表 A·1** に示す．また調査した仮想マシン検知機能を保有する検証プログラム detector を作成する．それらの結果から，仮想マシン検知機能の回避方法とシステムに実装すべき機能について提案する．

本稿の 2 章では関連研究について調査した結果をまとめる．3 章ではマルウェアの解析した結果をまとめる．4 章では仮想マシン検知機能の検知対象と検知の技術的方法について分類し，detector の実装について記述する．5 章では detector の実行結果について考察し，仮想マシン検知を無効化する機能の実装方法を提案する．最後に，6 章では今後の課題について記述する．

1.1　動作環境とシステム構成

文献 [1] のシステムでは，解析ツール NetMonitor がマルウェアのプロセスの動作およびそれが行う通信の双方を紐付けて記録する．本研究では NetMonitor に仮想マシン検知回避機能を加えることを検討する．

本研究のシステムは**表 1** に示す仮想マシンで構築し，NetMonitor は仮想マシンのゲスト OS で動作する．本研究では，表 1 に示す解析ツールを起動した状態で仮想マシンの検知を検証する．また仮想マシンではない実機を仮想マシンと誤検知しないことを確認する．

1.2　検知対象と対応範囲

下記はマルウェアが検知する対象と考えられる．

デバッガ　OllyDbg，WinDbg，Immunity Debugger など
仮想マシン　VMware，Virtual PC，Virtual Box など
解析ツール　Wireshark，Fiddler，Sysinternals など
サンドボックス　Cuckoo，Sandboxie など

この中でデバッガとサンドボックスは表 1 に示す仮想マシンには関係ないので本研究では扱わない．また表 1 に示す仮想マシンだけを対象とする．ただし同一の方法で容易に実装できる場合には，対象外のデバッガとサンドボックスであっても可能な限り detector および NetMonitor に実装する．たとえば，4.3 節のプロセス名にデバッガやサンドボックス，その他の仮想マシンの名前を加えたとしても，実装の手間が増えたり実行時のパフォーマンスが低下することはないので，detector および NetMonitor に実装する．

本研究のシステムで解析ツールを動作させることもありうるので，解析ツールにも可能な限り対応する．本研究では仮想マシン検知に，解析ツールも含めて扱う．

2.　関連研究

2.1　仮想マシン検知機能の技術情報

アナライジング・マルウェア [2] の 3 章に仮想マシン検知の方法と対処法が書かれている．多くはデバッガを対象としているが，仮想マシンを対象とする方法も含まれる．しかし具体的な検知対象（プロセス名やウインドウ名など）が書かれていない方法がある．Issa の研究 [3] ではプロセス開始時のレジスタの値やスタックのアドレスを用いる検知方法についても述べている．これは解析したマルウェアや他の文献には見られない方法である．しかし表 1 に示す環境には影響がないので，本研究では対象とせず表 A·1 にも含めない．Branco らの研究 [4] は耐解析技術全般について扱っており，その中で仮想マシンの検知について言及している．Hoffman の研究 [5] はある 1 つの検体に実装されている仮想マシン検知機能について解説している．Assor らの研究 [6] は仮想マシンおよびサンドボックスの検知について解説している．Griffin の研究 [7] はランサムウェアの Locky に新たに実装された API の実行時間の違いに基づく仮想マシンの検知について解説している．

文献では具体的なコードが示されていない場合が多いので，文献の情報は主に静的解析を行った検体の情報を補うために使用する．ただし文献にしかない検知機能は，一般的と考えられる方法で detector に実装し，NetMonitor の検証に利用する．

2.2　仮想マシン検知機能の実装状況

大山の研究 [8] では FFRI Dataset 2016 に含まれる 8,243 のマルウェアの動的解析の結果を対象にして，マルウェアに実行されている仮想マシン検知機能の傾向を分析している．動的解析の API のログを分析しているため，API による検知のみを対象としている．

Chen らの研究 [9] では 16,246 の一般的なマルウェアと 1,037 の APT 攻撃のマルウェアに実装されているアンチデバッグ機能と仮想マシン検知機能の傾向を分析している．

Chen らの研究では利用されている API，逆アセンブル結果の命令，検体内部の文字列からアンチデバッグ機能と仮想マシン検知機能の有無を判定している．論文では静的解析と称しているが，この判定手法は表層解析と言うべき手法である．

2.3 仮想マシン検知プログラム

Pafish[10] は解析環境を検知するプログラムである．主な対象はサンドボックスと仮想マシン，デバッガであり，1.2 節に記述する本研究の対象に含まれる検知機能もある．

3. 検体の解析

これまでに解析したマルウェアの情報を基に，下記の条件を満たすと思われる検体の仮想マシンの検知機能について静的解析した．

- 表 1 に示す環境で動作する．
- 1.2 節に記述する本研究の対象となる仮想マシン検知機能がある．

その結果，表 A·1 に示す 12 種類の検体を選定した．次節以降では，解析した検体の特徴をまとめる．なお，すべての検体は 32 ビットの実行可能ファイルである．ただし，Dyreza が他のプロセス（ブラウザ）にインジェクションする DLL は 32 ビットと 64 ビットの両方がある．

3.1 マルウェア本体とアンパック処理

Dofoil および Goldun，Recam，ZeuS の一部の亜種はパックされており，アンパック処理の途中に仮想マシン検知機能がある．また Dofoil および Shiz はマルウェア本体にも仮想マシン検知機能がある．このような場合，マルウェア本体とパッカーの組み合わせにより有する仮想マシン検知機能が変化する．

たとえば，オリジナルの ZeuS には仮想マシン検知機能は無いが，表 A·1 の検体のうち 1 つはアンパック処理の途中で「4.3 プロセス名」と「4.5 モジュール列挙」が実装されている．一方，ZeuS の亜種の 1 つである Citadel ではマルウェア本体で「4.3 プロセスのバージョン情報」が実装されている．そのため表 A·1 の ZeuS は 3 つの機能が有効になっているが，同時に 3 つの機能を実装している検体は表 A·1 の中には存在しない．

3.2 仮想マシン検知機能の実行機会

仮想マシンの検知にはマルウェアとしての動作を隠す目的があるので，Dyreza を除きアンパック処理またはマルウェア本体の最初で仮想マシン検知機能が動作する．

一方，Dyreza は 4.15 項で説明するクラウド型検知であり，他の検体とは方法が異なる．そのため，Dyreza はマルウェア本体が DLL を他のプロセス（ブラウザ）にインジェクションし，インジェクション先のプロセスで仮想マシン検知機能が動作する．

3.3 仮想マシン検知時の動作

複数の仮想マシン検知機能を実装している検体は順番に検知機能を実行する．仮想マシンを検知したときには，その後の検知機能は実行されず，検知したときの動作を行う．

多くの検体では仮想マシンを検知したときにはプロセスを直ちに終了するが，下記の検体は異なる．(2) および (3) は動作を継続するが，検知後の動作は本来のマルウェアの動作よりも少なくなる．

(1) Dofoil および Goldun，Recam，ZeuS の一部の亜種のアンパック処理中の検知では，予期しない命令が実行されることで例外が発生して終了する．
(2) Dofoil のマルウェア本体および AndromedaBot の一部の亜種は Sleep を呼び出し続けて停止する．
(3) EMDIVI および AndromedaBot の一部の亜種はダミーの動作を行う．
(4) Rovnix はデバッガを検知したときには C&C サーバに検知情報を送信して終了する．
(5) Dyreza は 4.15 項で説明するクラウド型検知であり，その後に実行されるコードは変わらないが，C&C サーバが検体に送信する設定や命令を変える可能性がある．

3.4 その他

我々がこれまでに解析したマルウェアで発見した仮想マシン検知機能を表 A·1 の「その他」の列にまとめる．本研究のために静的解析した 12 種類のマルウェアとは異なり，その他にまとめた検体は仮想マシン検知機能についての知見を得ることが目的で解析を行っていない．そのため，検体が実装している仮想マシン検知機能のすべてを解析できているとは限らない．また 3.1 節の実装箇所，3.2 節の実行機会，3.3 節の検知後の動作も明確ではない．

しかし，同一の検知機能であっても 12 種類のマルウェアとは異なる方法で実装されている場合がある．たとえば，4.3 節に記述するプロセス名で検知する場合，3 種類の方法があり，プロセス名も検体毎に異なる．ゆえに，多くの実装方法や対象を構築するシステムに取り入れるために，その他にまとめた検体も必要に応じて参照する．

4. 検知方法および検証プログラムの実装

本章では 2 章の関連研究の調査結果と 3 章の解析結果を対象毎に分類し，検知方法および detector の実装について記述する．解析結果または関連研究の検知方法が複数あるときには，4.16 節で特に記述しない限り detector はすべてを包括する方法で実装する．

検知方法の detector への実装状況は表 A·1 の検証の列に示す．「—」の項目は detector に実装していない検知方法である．「✓」または空白の項目は detector に実装され

ている検知方法である．

4.1 CPU

仮想マシンでは命令の実行時間が長くなる可能性があるので，いくつかの命令の実行時間を測定し，その値に基づいて仮想マシンを検知できる．detector では Shiotob に実装されている rdtsc 命令で cpuid 命令の実行時間を 10 回測定し，その合計が 40,000 以上または 10 未満のときには仮想マシンとみなす方法と，文献 [7] の GetProcessHeap と CloseHandle の実行時間の比率を求める方法を実装する．

sidt 命令による仮想マシンの検知は，文献 [2] においてマルチプロセッサ化が進んでいるので難しいと指摘されている．

文献 [2] には，sldt 命令を実行してローカルディスクリプタテーブルレジスタ（LDTR）のアドレスを取得し，LDTR のセグメントレジスタの値が非 0 ならば仮想マシンとみなす方法が記述されている．

レジスタ EAX の値に応じて cpuid 命令は値を返す．下記のいずれかの条件を満たすときには仮想マシンとみなす．

- EAX=0 で仮想マシンに関する特定の値が返された
- EAX=1 でハイパーバイザが有効または MMX がサポートされていないことを示す値が返された

VMware のゲスト OS では VMware バックポートにコマンド番号 0Ah（VMware バージョン取得）でアクセスしたときには EBX が 564D5868h になる．値を取得できたときに VMware とみなす実装と，例外が発生しないときに VMware とみなす実装がある．

4.2 実行時間

4.1 節の rdtsc 命令と同様に実行時間を測定するが，時間の取得に rdtsc 命令ではなく API を用いる．仮想マシンで実行時間で長くなることから検知する方法以外に，API が返す時間の整合性を検証する方法がある．

4.3 プロセス

プロセスを列挙して特定の名前のプロセスがみつかったときには仮想マシンとみなす．対象となるプロセスはデバッガや仮想マシンのプログラム，解析のためのツールなど多岐にわたる．名前の比較は CRC32 値を比較する方法と文字列を比較する方法がある．文字列の場合，一致か含む，大小文字の区別など細かい差異がある．プロセスの列挙方法は主に下記の 3 種類である．

- CreateToolhelp32Snapshot，Process32First，Process32Next
- 第 1 引数を SystemProcessInformation（5）で ZwQuerySystemInformation を呼び出す
- WMI プロバイダ

プロセスを列挙してプロセスのファイルのバージョン情報の ProductName と CompanyName から仮想マシンを検知する方法もある．ZeuS の一部の亜種は特定の文字列が含まれているときには仮想マシンとみなすが，その他の検体はより厳密で ProductName と CompanyName の組み合わせで文字列が一致するときは仮想マシンとみなす．

プロセスの名前から検知する方法では，プロセスのファイル名を変えることで検知を回避できる．一方，プロセスのバージョン情報を変えるためには，ファイルの内容を書き換える必要があり，回避はより難しくなる．

4.4 サービス列挙

文献 [6] では VMware に関する名前のサービスが存在しているときには仮想マシンとみなす．しかし文献 [6] にはサービスの存在を確認する具体的な方法が書かれていない．

4.5 モジュール

プロセスのモジュールから仮想マシンを検知する方法は列挙と存在確認の 2 つある．

1 つは CreateToolhelp32Snapshot, Module32First, Module32Next でプロセス内のモジュールを列挙する方法である．特定の名前の関数がエクスポートされているか，デバッガやサンドボックスに存在するモジュールを示す文字列が列挙したモジュールのパスに含まれているときには仮想マシンとみなす．

もう 1 つは特定の名前のモジュールを示す文字列のハンドルを GetModuleHandle で取得できたときには仮想マシンとみなす方法である．

4.6 ウインドウ

ウインドウから仮想マシンを検知する方法も列挙と存在確認の 2 つある．

1 つは EnumWindows を呼び出してウインドウを列挙し，GetClassName でクラス名，GetWindowText でウインドウ名を取得する方法である．ウインドウ名にデバッガや解析ツールを示す文字列が含まれているとき，または取得したクラス名が OLLYDBG（大小文字を区別しない）のときには仮想マシンとみなす．

もう 1 つはデバッガや解析ツールのクラス名またはウインドウ名を引数にして FindWindow を呼び出し，ウインドウハンドルが取得できるならば仮想マシンとみなす方法である．

4.7 ディスクドライブ

GetVolumeInformation でドライブのボリューム名とシリアル番号を取得する．ボリューム名文字列の CRC32 値が 20C7DD84h または 53BEF79Bh のときには仮想マシンとみなす．またはシリアル番号が CD1A40h または 70144646h のときには仮想マシンとみなす．これらの値は

マルウェアで実装されていた値であり，これらの値が示す環境は不明である．

DeviceIoControl を使用する方法もある．IOCTL_STORAGE_QUERY_PROPERTY を実行することにより得られる STORAGE_DEVICE_DESCRIPTOR 情報から最初の物理ディスクである \\.\.PhysicalDrive0 のプロダクト ID 文字列を取得する．プロダクト ID 文字列に仮想マシンに関する文字列が含まれているときには仮想マシンとみなす．

4.10 節と重複するが，レジストリのディスクに関する情報から検知する方法もある．レジストリの HKLM\system\currentcontrolset\services\disk\enum\0 の値に仮想マシンに関する文字列が含まれているときには仮想マシンとみなす．

ディスクのサイズで検知する方法もある．CreateFile を介して PhysicalDrive0 へのハンドルを取得し，DeviceIOControl を用いて IOCTL_DISK_GET_LENGTH_INFO（7405Ch）を実行することでハードドライブのサイズを取得する．サイズが 10GB 以下のときには仮想マシンとみなす．この検知方法の実行には管理者権限が必要である．

4.8 名前

GetComputerName と GetUserName が返す名前が条件に一致していたときには仮想マシンとみなす．GetUserName ではなくプロセスのユーザ名を取得する方法もある．コンピュータ名またはユーザ名のいずれかを比較する検体と，両方を比較していずれも一致したときに仮想マシンとみなす検体がある．文献 [3] ではレジストリを参照してユーザ名を取得しているが，具体的なレジストリの名前については記述がない．

その他，ネットワークのワークグループ名から仮想マシンを検知する方法もある．

4.9 ファイルとデバイス

CreateFile または GetFileAttributes，PathFileExists を呼び出して特定のファイルが存在するか確認することで仮想マシンを検知する．または，デバッガや解析ツールに関するデバイスを CreateFile で開くことができたときには仮想マシンとみなす．

4.10 レジストリ

仮想マシンや解析ツールなどに関するレジストリのキーが存在するときには仮想マシンとみなす．または特定のレジストリの値に，特定の文字列が一致するか含まれるときには仮想マシンとみなす．

4.11 デバイス情報

SetupAPI でデバイスの情報を取得し，情報に特定の文字列が含まれているときには仮想マシンとみなす．

4.12 MAC アドレス

MAC アドレスのプレフィックス（ネットワークアダプタのベンダ）が VMware や Virtual Box に固有の値のときには仮想マシンとみなす．

4.13 WMI Provider

WMI Provider の Win32_ComputerSystem クラスを取得し，次の条件のいずれかを満たすときには仮想マシンとみなす．

- Manufacturer の値が「microsoft corporation」でありかつ Model に「VIRTUAL」を含む（大小文字を区別しない）
- Model に「vmware」を含む（大小文字を区別しない）
- Model が「Model」である

4.14 自己診断

GetModuleFileName で自身のモジュールのパスを取得する．パスの中に特定の文字列が含まれているか，自身のハッシュ値がファイル名になっているときには仮想マシンとみなす．

Rovnix は親プロセスのファイル名に perl や python，解析ツールに関する文字列が含まれているときには仮想マシンとみなす．文献 [4] では親プロセスの名前が Explorer.exe でないときには仮想マシンとみなす．また文献 [4] では親プロセスの取得方法について列挙している．

その他，API の先頭の命令が jmp 命令であったりソフトウェアブレークポイントが設定されている，または特定の命令になっているときには仮想マシンとみなす．

4.15 クラウド型検知

Dyreza はレジストリからインストールされているソフトウェアのリストを外部の C&C サーバに送信する．C&C サーバが検体から送られた情報に基づいてマルウェアの実行環境を判定し，検体に送信する設定や命令を変える可能性がある．しかし，送信された情報が仮想マシンの検知に使われるか否か，どのような条件で検知されるのかは，C&C サーバのコードを解析しない限りわからない．この方法を本研究では「クラウド型検知」と称する．

4.16 検証プログラムの実装

4.1 節の CPU 命令の rdtsc 命令による時間測定では，detector は Shiotob に実装されている rdtsc 命令で cpuid 命令の実行時間を 10 回測定し，その合計が 40,000 以上または 10 未満のときには仮想マシンとみなす方法と，文献 [7] の GetProcessHeap と CloseHandle の実行時間の比率を求める方法を実装する．sidt 命令による仮想マシンの検知は

detector では実装しない．また，detector は VMware バックポートで例外が発生しないときには VMware とみなす．

4.2 節の実行時間では 2 つの検知方法を実装する．1 つは Shiotob および文献 [5] の方法である．GetTickCount を呼び出し OS が起動してからの時間を取得する．その後 Sleep で 500 ミリ秒停止した後，もう 1 度 GetTickCount で時間を取得する．最初の GetTickCount と 2 回目の GetTickCount の返す値の差が 450 ミリ秒以下ならば仮想マシンとみなす．

もう 1 つは URLZone で実装されている方法である．Sleep で 1 分間停止して終了するスレッドを作成する．検体は 513 バイトのメモリを，500,000,000 回コピーする．作成したスレッドが終了している（コピーが 1 分以上かかっている）ときには仮想マシンとみなす．

4.3 節のプロセスの列挙の段階で NetMonitor が対象となるプロセスを隠蔽できれば十分である．そのためプロセスの列挙で対応するのでバージョン情報については detector では実装しない．

4.4 節のサービスでは，detctor は API を利用してサービスを列挙する方法と，サービスに関するレジストリを列挙してサービスを列挙する方法を実装する．いずれも文献 [6] が示すサービス名があるときには仮想マシンとみなす．

4.7 節のディスクドライブでは，detector はすべてのドライブを対象とする．また，detector は GetDiskFreeSpaceEx でシステムがインストールされているドライブのサイズを取得し，10GB 以下のときには仮想マシンとみなす．

4.8 節のネットワークのワークグループ名による検知では，名前が ANALYSERS のときには仮想マシンとみなす．大小文字区別せず英数字以外の文字は無視される．

4.9 節のファイルでは，detector は 4.9 節に挙げた 3 つの API と FindFirstFile と FindNextFile の組み合わせを用いて，いずれかの API で特定のファイルの存在を確認できるならば仮想マシンとみなす．

4.10 節のレジストリについて，detector は次の 3 つを実装する．

- 特定のレジストリのキーを RegOpenKey または RegOpenKeyEx で開くことができるならば仮想マシンとみなす．
- プロダクト ID のレジストリの値が特定の値であるとき，またはハードウェアの説明文のレジストリの値が特定の値に仮想マシンに関する文字列が含まれているときには仮想マシンとみなす．
- レジストリの HKLM\Software\Microsoft\Windows\CurrentVersion\Uninstall 以下のキーを列挙し，AutoItv3CCleanerWIC という文字列に含まれるキーが 3 つ以上あるときには仮想マシンとみなす．

4.11 節のデバイス情報では，detector はデバイスの説明やフレンドリーネームに仮想マシンに関する文字列が含まれているときには仮想マシンとみなす．

表 2 クラウド検知

対象	情報	対応
Dyreza	プログラム	4.10 節
	サービス	4.4 節
Rovnix	ディスクのボリュームシリアル番号	4.7 節
	アンチウイルス製品のバージョン情報	4.3 節
	カレントプロセス名	4.14 節
	親プロセス名	4.14 節
	プロセスのユーザ名	4.8 節

4.14 節の自身のモジュール名では，detector はサンドボックスで使われる文字列か，MD5 や SHA-1 などのハッシュ値の 16 進数表記の先頭 5 文字が含まれてでいる（大小文字を区別しない）ときには仮想マシンとみなす．親プロセスの名前では，detector は Rovnix のによる検知方法を実装する．また，detector は実装しないが NetMonitor は API のフックが検知されないような実装を行う必要がある．

detector は 4.15 節のクラウド型検知を実装しないが NetMonitor は仮想マシンの検知につながるような情報を検体が取得できないようにする必要がある．Dyreza および Rovnix で取得され送信される情報とその対応を**表 2** に示す．

5. 考察

4 章の結果から検証プログラム detector を作成した．これを表 1 に示す仮想マシンで解析ツールを起動した状態，および実機で実行した結果を表 A·1 に示す．実機では仮想マシンを誤検知することはなかった．

5.1 技術的方法

検知を行う方法は，4.1 節の CPU の命令を利用した検知と，それ以外の API 呼び出しの結果による検知に分けることができる．API の呼び出しで検知する場合には，API をフックしてその結果を改変することで検知を回避できる．一方，CPU の命令で検知を行う場合，その結果を改変することは困難である．4.1 節の cpuid 命令や VMware のバックポートの結果は，仮想マシンの設定で変更することができる場合がある．

よって，CPU の命令による検知は仮想マシンの設定で対応し，NetMonitor は API のフックにより仮想マシンの検知を回避する．

5.2 時間

4.2 節の実行時間は，表 1 に示す環境で実行しても検知しなかった．十分な性能の CPU を搭載していたため，仮想マシンであっても時間内に処理を終えることができたことが原因だと考えられる．

表 3　rdtsc 命令実験環境

仮想マシン	CPU	Intel Core i7 5557U 3.10GHz
	ホスト OS	OS X 10.11.6
	仮想環境	VMware Fusion 8.5.7
	ゲスト OS	Windows 7 Professional SP1 32bit
		Windows 7 Professional SP1 64bit

表 4　rdtsc 命令実行時間

環境		32 ビット		64 ビット	
		最小値	最大値	最小値	最大値
表 1	仮想	54,232	65,512	21,837	34,230
	実機	2,466	2,889	1,086	2,340
表 3	仮想	11,520	15,342	9,090	10,839

4.1 節の rdtsc 命令は，32 ビット OS の仮想マシンだけで検知した．rdtsc 命令による実行時間の判定についてより詳しく調べるために**表 3** に示す環境でも測定を行った．表 1 と表 3 で detector を 5 回実行したのときの実行時間の最小値と最大値を**表 4** に示す．

表 4 を見る限り，64 ビットの方が実行時間が短い．また，十分な性能の環境を準備すれば仮想マシンであっても，マルウェアが実装するしきい値の 40,000 を下回ることができる．一方，NetMonitor でソフトウェア的な方法で時間に関する機能を操作し，整合性を保たせることで検知を回避するのは困難である．よって，NetMonitor ではこの方法に対応する機能は実装せず，十分な性能の環境を準備することで対応する．

5.3　未検知の仮想マシン検知機能

4.5 節のモジュールはデバッガやサンドボックスを対象としているため，表 1 に示す環境で実行しても検知しなかった．しかしモジュールの列挙や存在確認で仮想マシンや解析ツールを検知することも可能であると考えられる．同様に表 A·1 で未検知となっている仮想マシン検知機能であっても，その機能を使って表 1 に示す環境を検知するマルウェアが存在する，もしくは今後作られる可能性はある．よって，優先順位は下がるものの，現状で未検知の仮想マシン検知機能にも対応する必要がある．

5.4　システム構築の仕様による対応

コンピュータ名，ユーザ名，ワークグループ，カレントプロセス名などによる検知は API のフックで対応できる．しかしシステムを構築する段階で仮想マシンとみなされるような名前を避けることでも容易に対応できる．API のフックによる対応とあわせて，システム構築での対応も検討できると考えられる．

5.5　NetMonitor の検知

現状では NetMonitor は公開されていないので，NetMonitor を検知の対象とするマルウェアは存在しない．しかし，実行環境（プロセスやウインドウなど）が C&C サーバに送信されることで，NetMonitor の存在が明らかになってしまう可能性がある．ゆえに，NetMonitor は自身を隠蔽しなければならない．マルウェアが NetMonitor に関するプロセス，モジュール，ファイル，レジストリなどの情報を取得できないようにする必要がある．

6.　おわりに

本研究では，仮想マシン検知機能を回避可能な NetMonitor の実現に向けた調査分析を実施した．マルウェア解析や文献調査から仮想マシン検知機能を分析した．また，その結果から検証プログラム detector を実装した．

今後の課題は，本研究で提案した解析ツール NetMonitor を改変し，NetMonitor で detector および実際のマルウェアによる検知が回避できることを確認することである．

本研究では NetMonitor の改変を目的に仮想マシン検知機能を調査し detector を実装した．しかし NetMonitor の改変に限定せず，他の仮想マシン，サンドボックス，デバッガなどに調査範囲を広げることも考えられる．それに伴い detector を汎用的な解析環境検知プログラムにすることができれば，同種の研究や解析環境の構築に役立てることができると思われる．

また，日々新しいマルウェアが公開される中，新しい仮想マシン検知機能も実装されることが考えられる．ゆえに，常に新しいマルウェアを解析し，新しい仮想マシン検知機能に対応していくことが必要である．

付　　録

表 A·1 の「静的解析を行ったマルウェアファミリ」は 4 章に記述する検知方法の各マルウェアにおける実装状況を示す．同様に表 A·1 の「文献」は 2 章の関連研究で挙げた文献における記述の有無を示す．表 A·1 の「検証」で「—」になっている項目は detector に実装していない検知方法である．「✓」は detector が検知した方法であり，空白は detector が検知しなかった方法である．

参考文献

[1] 神薗雅紀，遠峰隆史，津田　侑，衛藤将史，星澤裕二，井上大介：プロセスの通信手続きに基づくフォレンジック手法の提案，コンピュータセキュリティシンポジウム 2014 論文集，Vol. 2014, No. 2, pp. 167–174 (2014).

[2] 新井　悠，岩村　誠，川古谷裕平，青木一史，星澤裕二：アナライジング・マルウェア—フリーツールを使った感染事案対処，オライリージャパン (2010).

[3] Issa, A.: Anti-virtual machines and emulations, *Journal in Computer Virology*, Vol. 8, No. 4, pp. 141–149 (online), DOI: 10.1007/s11416-012-0165-0 (2012).

[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* (2012).

表 A·1 仮想マシン検知機能

検知機能		静的解析を行ったマルウェアファミリ													文献						検証			
																					実機		仮想	
		Dofoil	Shiotob	AndromedaBot	ZeuS	Goldun	EMDIVI	Dyreza	Shiz	Recam	Rovnix	URLZone	Ursnif	その他	[2]	[3]	[4]	[5]	[6]	[7]	32bit	64bit	32bit	64bit
4.1	rdtsc 命令		✓	✓																✓			✓	
	sidt 命令														✓						—	—	—	—
	sldt 命令														✓									
	cpuid 命令									✓									✓				✓	✓
	VMware バックポート										✓				✓	✓			✓					
4.2	実行時間		✓									✓			✓		✓	✓						
4.3	プロセス名			✓	✓				✓					✓	✓	✓			✓				✓	✓
	プロセスのバージョン情報													✓							—	—	—	—
4.4	サービス列挙																		✓				✓	
4.5	モジュール列挙				✓									✓										
	モジュール存在確認	✓	✓	✓		✓			✓					✓				✓						
4.6	ウインドウ列挙						✓							✓	✓								✓	✓
	ウインドウ存在確認													✓	✓		✓						✓	✓
4.7	ボリューム名			✓										✓										
	ボリュームシリアル番号	✓							✓															
	ディスクのプロダクト ID													✓									✓	✓
	ディスクドライブ情報	✓	✓	✓		✓								✓									✓	✓
	ディスクドライブサイズ																	✓						
4.8	コンピュータ名とユーザ名								✓					✓		✓		✓						
	ワークグループ								✓															
	プロセスのユーザ名										✓													
4.9	ファイル存在確認								✓					✓					✓					
	デバイス存在確認										✓			✓	✓	✓	✓							
4.10	レジストリキー													✓				✓	✓				✓	✓
	レジストリ値		✓						✓					✓		✓		✓						
	レジストリ Uninstall	✓																						
4.11	デバイス情報											✓	✓										✓	✓
4.12	MAC アドレス																		✓					
4.13	WMI Provider													✓										
4.14	カレントプロセス名検証	✓	✓						✓		✓			✓				✓						
	親プロセス名検証										✓						✓							
	フック検知								✓			✓		✓	✓						—	—	—	—
4.15	クラウド							✓			✓										—	—	—	—
検体数		2	2	7	10	1	1	2	1	1	1	1	1	29										

[5] Hoffman, N.: VM Checking and Detecting, (online), available from ⟨http://securitykitten.github.io/vm-checking-and-detecting/⟩ (accessed 2017-06-14).

[6] Assor, Y. and Slotky, A.: Anti-VM and Anti-Sandbox Explained, Cyberbit Advanced Malware Research Group (online), available from ⟨https://www.cyberbit.net/endpoint-security/anti-vm-and-anti-sandbox-explained/⟩ (accessed 2017-06-14).

[7] Griffin, N.: Locky Returned With A New Anti-VM Trick, Forcepoint (online), available from ⟨https://blogs.forcepoint.com/security-labs/locky-returned-new-anti-vm-trick⟩ (accessed 2017-06-14).

[8] 大山恵弘：マルウェアによる対仮想化処理の傾向についての分析，コンピュータセキュリティシンポジウム 2016 論文集，Vol. 2016, No. 2, pp. 534–541 (2016).

[9] Chen, P., Huygens, C., Desmet, L. and Joosen, W.: *Advanced or Not? A Comparative Study of the Use of Anti-debugging and Anti-VM Techniques in Generic and Targeted Malware*, pp. 323–336 (online), DOI: 10.1007/978-3-319-33630-5_22, Springer International Publishing (2016).

[10] Ortega, A.: Pafish (Paranoid Fish), (online), available from ⟨https://github.com/a0rtega/pafish⟩ (accessed 2017-08-15).