Computer Security Symposium 2017
23 - 25 October 2017

仮想マシン検知回避機能を持つ動的解析ツールの開発

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

概要：動的解析は，静的解析に比べて短時間で容易にマルウェアの挙動を把握することが可能であり，広く用いられている．一般的に，動的解析には仮想マシンを用いる．このため，マルウェアの中には仮想マシン環境を検知し動的解析を妨げる機能を有するものが存在している．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を行うためのシステムが必要である．我々は，マルウェアの保有する仮想マシン検知機能の調査解析を実施し，対応方法の提案を行った．本稿では，調査結果に基いて開発した仮想マシン検知機能を回避し動的解析を行うツールについて述べる．更に実マルウェアを用いて仮想マシン検知の回避機能の有効性の検証を実施した．

キーワード：マルウェア，アンチ VM，仮想マシン，動的解析

Development of Dynamic Analysis Tool that Avoid of Anti Virtual Machine Function on Malicious Software

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

Abstract: Dynamic analysis is useful to reveal behavior of the malware. Generally on dynamic analysis, a virtual machine(VM) is used. However, some of malware have antiVM techniques. These malware detect VM and change their behavior to avoid dynamic analysis. Therefore, We need hide characteristics of the VM against antiVM techniques. Thus far we have surveyed antiVM techniques and proposed the methods to neutraize them. In this paper, we present aaVMNetMonitor, which is a dynamic analysis tool. aaVMNetMonitor enables us to neutralize antiVM techniques.Further, we experimented on its effectiveness of the function to neutralize antiVM techniques of the our tool.

Keywords: Malware, AntiVM, Virtual Machine, Dynamic Analysis

1. はじめに

近年，特定の組織を狙ったサイバー攻撃，標的型攻撃が社会問題となっている．標的型攻撃は，攻撃用ツールやマルウェアを利用して，ネットワーク越しに対象組織の情報を窃取する手法が一般的である．

標的型攻撃は，標的となる組織用にカスタマイズされた攻撃手法が用いられるため，一般的なシグネチャやアノマリ検知などでは検知できないおそれがある．そこで，標的型攻撃では，どの様な攻撃が行われるのか把握するための攻撃手法を分析するための解析環境について提案が行われている [1], [2]．標的型攻撃などに用いられるマルウェアの解析においては，マルウェアプロセスの挙動および通信に着目して解析を行う．また，ネットワーク監視の視点からネットワークを流れるパケットを収集，分析することで攻撃や悪意のある通信を検出する解析が行われる．しかし，これらの各端末上のプロセスの挙動・通信の解析とネットワーク監視による通信解析を相互に結びつけるための情報は不足していると考えられる．

我々は，これらの不足している情報を補完するために端末上のプロセスの挙動および通信を記録するためのプロセ

[1] 株式会社セキュアブレイン
ScureBrain 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_takada@securebrain.co.jp

© 2017 Information Processing Society of Japan

スの通信手続き保全ツール（通称: NetMonitor）[3] を用いた動的解析システムの開発を行ってきた．NetMonitor は，子プロセスの生成や他プロセスへのインジェクション，監視対象プロセスの通信手続きなどを記録することを可能とした．

マルウェア解析環境は一般的に仮想マシンで構成される．このためマルウェアが，自身の挙動や機能を隠蔽することを目的とした仮想マシンを検知し解析を行えない様にする機能を有するケースが存在していることが知られている．そこで，多くの未知のマルウェアに対し動的解析を行うためには，仮想マシン検知を回避する必要がある．

本研究では，NetMonitor にマルウェアによる仮想マシン検知を回避する機能を実装することで，仮想マシンを検知するマルウェアの動的解析を仮想マシン上で行うことを可能とする仮想マシン検知回避機能を有するプロセスの通信手続き保全ツール anti-anti-VM-NetMonitor（以下，aaVMNetMonitor）の開発を行った．

はじめに，我々は，マルウェアの保有する仮想マシン検知機能について検知対象，検知手法を静的解析による調査を行い実態を把握した [4]．また，調査結果に基づいた仮想マシン検知機能を有する検証プログラム detector を作成した．これらの調査結果に基いて NetMonitor に対して仮想マシン検知回避機能の実装を行い aaVMNetMonitor を開発した．本稿では，aaVMNetMonitor に実装した仮想マシン検知回避機能について解説する．また，マルウェアを用いて仮想マシン検知回避機能の有効性の検証実験を実施した結果について述べる．

本論文の構成は以下の通りである．2 章で関連研究の調査結果について記述する．3 章で aaVMNetMonitor の概要および回避対象とする仮想マシン検知機能について記述する．4 章でマルウェアを用いて実施した有効性検証実験の方法と結果について記述する．最後に 5 章で考察と今後の課題について記述する．

2. 関連研究

仮想マシン検知機能を回避する動的解析の研究は数多くされている．その中でも広く知られているものに Cuckoo Sandbox[5] が挙げられる．Cuckoo Sandbox は，オープンソースのサンドボックスであり，マルウェアを投入することで自動で解析することを可能とする．また，仮想マシン検知機能を回避するためにレジストリ情報の隠蔽を行う機能を保有している．しかし，Cuckoo Sandbox では，解析は自動化されておりマルウェア実行時に任意の操作等を行うことなどは困難である．また，VM 検知回避機能に関しても柔軟性があるとは言えない．

大月らの開発した Alkanet システム [6] が存在する．Alkanet システムは，アンチデバッグ機能と他プロセスへのインジェクション機能を有したマルウェアを短時間で解析するための BitVisor ベースの動的観測システムである．Kirat らの提案する手法 [7] では，仮想化技術を使用せず論理ボリュームマネージャを用いて実行マシンのイメージを管理し，マルウェア実行中の通信ログの記録と実行完了後のイメージの内容を比較する手法である．Zhang らの提案する手法 [8] は，x86 CPU のシステムマネジメントモードを使用したベアメタルデバッグシステムである．これは高度な解析に特化した動的解析システムである．Lengyel らは，DRAKVUF と Xen 仮想マシンマネージャを用いた独自の動的解析システムを提案している [9]．

関連研究はいずれも独自のシステムを構築して動的解析を行うものであるが，aaVMNetMonitor は，特定のシステムや仮想マシン環境には依存せず一般的な Windows 環境であれば導入が可能なツールである．また，サンドボックスの様な短時間かつ自動的にマルウェア解析を行う用途や文献 [1], [2] の様なマルウェアの長期観測や解析者によるデバッガ等による解析等においても仮想マシン検知回避機能を提供することを目的とする点が異なる．

3. aaVMNetMonitor

3.1 ツール概要

NetMonitor は対象とするプロセスの Windows API をフックすることで通信手続きを保全することを可能とした．また，マルウェアがインジェクションして利用する既存のプロセスや新たに起動する子プロセスを追従して対象とすることが可能である．

aaVMNetMonitor のモジュール概要を**図 1** に示す．aaVMNetMonitor は，32bit および 64bit の Windows に対応している．aaVMNetMonitor では，従来のフック対象の Windows API に加えて，文献 [4] の調査結果を基にマルウェアが OS の情報を取得するために使用する Windows API もフック対象とし，呼び出し結果を変更することでマルウェアによる仮想マシン検知を回避する．また，Windows API 呼び出し結果の変更方法を指定するためのルール機構を持つ．

3.2 回避対象とする仮想マシン検知機能

文献 [4] の調査結果に基き回避対象とする仮想マシン検知機能を決定した．我々の調査により明らかになった仮想マシン検知機能は，33 種類存在する．aaVMNetMonitor では，このうち 20 種の仮想マシン検知機能に対応する．仮想マシン検知機能を**図 2** に示す．なお，各仮想マシン検知機能の詳細に関しては，文献 [4] を参照することとする．

回避対象とする仮想マシン検知機能の決定において，静的解析を実施した 12 種のマルウェアが保有する機能の内で他のマルウェアでも実装される可能性が高いと判断した機能に対応を行った．なお，回避非対応とする仮想マシン検知機能のうち rdtsc 命令を始めとする CPU 命令系検

© 2017 Information Processing Society of Japan

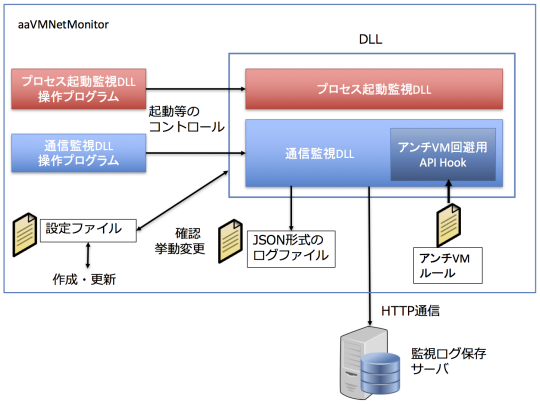

図 1 aaVMNetMonitor モジュール概要図
Fig. 1 aaVMNetMonitor Module Overview

回避対応機能

プロセス名，プロセスバージョン情報，ウィンドウ列挙，ウィンドウ存在確認，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，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，ファイル存在確認，デバイス存在確認，レジストリキー，レジストリ値，レジストリ Uninstall，デバイス情報，親プロセス名検証，フック検知

回避非対知機能

rdtsc 命令，sidt 命令，sldt 命令，cpuid 命令，VMware バックポート，実行時間，サービス列挙，モジュール列挙，モジュール存在確認，MAC アドレス，WMI Provider，カレントプロセス名検証，クラウド

図 2 仮想マシン検知機能
Fig. 2 Emvironment Detection Function

知機能，VMware バックポート，実行時間，クラウドの 7 種類の機能に関しては Windows API のフックによる対応が困難な項目である．他の 6 種類に関しては，実際のマルウェアで使用されている例が非常に少ないと思われるもの，我々が対象としない特定のサンドボックスソフトウェアなどを検知するものなどであったため非対応とした．

文献 [4] において解析調査を実施し，本研究において評価に使用した 12 種のマルウェアは，**表 4** の通りである．各マルウェアの保有する仮想マシン検知機能については，文献 [4] を参照することとする．

3.3 仮想マシン検知回避ルール

aaVMNetMonitor は，Windows API をフックして呼び出し状況を監視し，呼び出し結果を変更することで仮想マシン環境を隠蔽する．その際，呼び出し内容，もしくは呼び出し結果に仮想マシン環境の特徴を示す内容が含まれるかを判定する必要がある．また，API 毎に呼び出し結果変更時の挙動が異なる．

aaVMNetMonitor では，5 種類のルールタイプを定義し，監視対象となる API をこの 5 種類のルールタイプに割り当てることで API 呼び出し結果の変更を行っている．5 種類のルールタイプについて**表 1** に示す．各ルールにおける指定方法および一致した場合の動作の例を付録 A.1 に示す．

表 1 ルールタイプ
Table 1 Rule type

タイプ名	内容
A	チェック対象がルールに一致した場合，対象 API の戻り値をルールに指定された値に変更する．
B	チェック対象がルールに一致した場合，チェック対象の文字列をルールに指定された値に置換する．
C	対象 API が呼び出された場合，戻り値をルールに指定された値に変更する．
D	チェック対象がルールに一致した場合，不一致になるまで元（または別）API を呼び出しなおす．
E	対象 API が呼び出された場合，ルールに従ってアウト引数の値を置換する．

表 2 検証環境
Table 2 Verification Environment

検証環境 1	ホスト OS	Centos release 6.5 (Final)
	仮想環境	KVM + QUEM コンパイル時に使用したライブラリ: libvert 0.10.2 使用中のライブラリ: libvert 0.10.2 使用中の API: QEMU 0.10.2 実行中のハイパーバイザー: QEMU 0.12.1
	CPU	Intel Xeon E312xx (Sandy Bridge) 2.40GHz
検証環境 2	仮想環境	VMware ESXi5.1 検知回避有効時に，VMWare Backdoor Port 無効化，CPUID 編集
	CPU	Intel Xeon E312xx (Haswell) 3.10GHz
共通	ゲスト OS	Windows 7 Professional SP1 32bit/64bit Windows Update，Firewall 停止
	解析ツール	OllyDbg，Wireshark，Process Explorer，Fiddler2, VMware Tools

*VMware Tools は検証環境 2 のみ．

4. 仮想マシン検知回避機能の有効性検証実験

4.1 検証環境

仮想マシン検知回避機能の有効性実験を行った検証環境について**表 2** に示す．いずれの検証環境においても検証実験中は全ての解析ツールが起動した状態とする．各仮想マシンは，コンピュータ名，ユーザ名，ワークグループ名等は，調査の結果から検知に使用される Sandbox，Test，Sample といった単語は使用しない構成とする．

本環境を用いて 3 種類の検証実験を行った．各実験の詳細については，4.2，4.3，4.4 の各節で述べる．

4.2 検証用ソフトウェアを用いた検証

文献 [4] で作成した仮想マシン検知ソフト detector を用

© 2017 Information Processing Society of Japan

表 3　detector による検証結果
Table 3　Result of Using detector

	検証環境 1				検証環境 2			
	32bit		64bit		32bit		64bit	
回避機能	オフ	オン	オフ	オン	オフ	オン	オフ	オン
検知項目数	14	4	11	2	15	5	15	5
（回避対象）	(9)	(0)	(9)	(0)	(10)	(0)	(10)	(0)

いて aaVMNetMonitor の仮想マシン検知回避機能の検証を行った．検証環境は，表 2 の全ての Windows 環境で実施した．検証方法は，各検証環境上で aaVMNetMonitor を適用した状態で detector を実行し，仮想マシン検知回避機能のオン，オフで detector により検知される項目数の差を比較する．

実施結果を**表 3** に示す．この結果より，aaVMNetMonitor は，いずれの環境においても対象とする仮想マシン検知機能を無効化することが可能であることが分かる．

4.3　静的解析済マルウェアを用いた検証

4.3.1　実験方法

文献 [4] で静的解析済の 12 検体を用いて aaVMNetMonitor の仮想マシン検知回避機能の検証を行った．検証環境は，表 2 の全ての Windows 環境で実施した．

検証方法は，まず，実機上で仮想マシン検知回避機能をオフにした aaVMNetMonitor を適用した状態で各検体を実行後に 15 分程度放置し，NetMonitor ログの収集を行った．その後，各検証環境上で aaVMNetMonitor の仮想マシン検知回避機能オン，オフのみが異なる 2 つの仮想マシン上で同一の検体を並行して実行後に 15 分程度放置し，NetMonitor ログの収集を行った．検証環境で得られた NetMonitor ログを静的解析結果および実マシンで収集した NetMonitor ログと比較し，仮想マシン検知回避機能のオン，オフでどの様に振る舞いが変化したかを分析した．

4.3.2　実験結果の概要

各検証環境における検証結果を表 4 に示す．

この結果から検証環境 1 では，6 検体（うち 1 検体は 32bitOS のみ）で検知回避オフの状態では，仮想マシンである事を検知し動作を停止またはダミーの動作となった．また，仮想マシンであることを検知した 6 検体全てで，検知回避オンにすることで，仮想マシン検知を回避し実マシンと同様の動作となることを確認した．

検証環境 2 では，10 検体（うち 1 検体は 32bitOS のみ）で検知回避オフの状態では，動作を停止またはダミー動作となった．検知回避オンとすることで 9 検体において実マシンと同様の挙動となり，1 検体で挙動に変化が見られないことを確認した．このことから，aaVMNetMonitor の仮想マシン検知回避機能は，調査対象とした 12 検体に対しては十分に有効であると考えられる．

結果の詳細な分析については以下に述べる．

4.3.3　仮想マシン検知回避に成功した検体

検証環境 1 で検知回避に成功した 6 検体および検証環境 2 で検知回避に成功した 9 検体について述べる．

No.1, 5 の検体は，ディスクデバイス名に含まれる “qemu” および “vmware”，No.2 の検体では “vmware” の文字列で仮想マシンを検知する機能を有しており，ディスクデバイス名を隠蔽することで検知を回避している．

No.3 の検体では，ディスクデバイス名に含まれる “vmwa” の文字列による検知および解析ツール Wireshark のプロセス名を検知する機能を有しており，ディスクデバイス名および起動中の該当プロセス名を隠蔽することで検知を回避している．

No.4 の検体では，解析ツール VMware Tools のプロセス名を検知する機能を有している．なお，VMware Tools は，VMware Backdoor port を無効にするとプロセスが起動しなくなるため VMware Backdoor port を有効にした状態で回避機能オンにして確認を実施した結果，該当プロセスを隠蔽することで検知を回避している．

No.6 の検体は，起動中のプロセスのウィンドウを列挙し解析ツール OllyDbg，Process Explorer，Wireshark の 3 種類のウィンドウを検知する機能を有しており，該当プロセスのウィンドウを隠蔽することで検知を回避している．

No.8 の検体では，解析ツール OllyDbg，Wireshark のプロセス名を検知する機能を有しており，起動中の該当プロセス名を隠蔽することで検知を回避している．ただし，No.8 はその後，外部との通信による検知機能により動作を停止することが確認された．これについては，4.3.7 に分析結果を述べる．

No.11 の検体では，デバイス情報のフレンドリ名に含まれる “vmware”，No.12 の検体では “vmware” および “qemu” の文字列で仮想マシンを検知する機能を有しており，フレンドリ名を隠蔽することで検知を回避している．

これらの結果から，ディスクデバイス名，プロセス名検知，ウィンドウ列挙，デバイス情報による仮想マシン検知を行うマルウェアに対して検知回避機能が有効であることを確認した．

また，No.8，11 の検体は，API フックを検知するフック検知機能を有しており，特に No.8 の検体に関しては，検知対象の API Process32NextW を aaVMNetMonitor で API フックしているが検知されていないことから既知のフック検知に耐性があることを確認した．

4.3.4　有効な仮想マシン検知機能を持たない検体

検証環境 1 において No.2，4，7，10，11 の検体は，有効な仮想マシン検知機能を有していなかったため検知回避オフの状態でも本来の検体の挙動を示した．また，No.2，4，7，11 の検体は検知回避オンの状態でも同様の動作が確認されたことから検知回避機能により動作が阻害されないこ

© 2017 Information Processing Society of Japan

表 4　静的解析済マルウェアを用いた検証結果
Table 4　Result of Using Sample Malware

No	検体名	検証環境 1				検証環境 2			
		32		64		32		64	
		回避オフ	回避オン	回避オフ	回避オン	回避オフ	回避オン	回避オフ	回避オン
1	Dofoil	検知	回避	検知	回避	検知	回避	検知	回避
2	Shiotob	△				検知	回避	検知	回避
3	AndromedaBot	検知	回避	検知	回避	検知	回避	検知	回避
4	ZeuS	△				検知	回避	検知	回避
5	Goldun	検知	回避	検知	回避	検知	回避	検知	回避
6	EMDIVI	検知	回避	検知	回避	検知	回避	検知	回避
7	Dyreza	▲				▲			
8	Shiz	検知	回避	╳	╳	検知	回避	╳	╳
9	Recam	╳	╳	╳	╳	╳	╳	╳	╳
10	Rovnix	△	検知	△	検知	検知	検知	検知	検知
11	URLZone	△				検知	回避	検知	回避
12	Ursnif	検知	回避	検知	回避	検知	回避	検知	回避

△は，該当環境に対する有効な検知機能なし．▲は，判断不能．╳は，検体が正しく動作しない．

とが分かる．しかし，No.10 の Rovnix においては挙動が変化して仮想マシンが検知された際の動作となっている．これについては，4.3.6 に分析結果を述べる．

4.3.5　rdtsc 命令や処理の実行時間による検知を行う検体

No.2，3，11 の検体は，rdtsc 命令や処理の実行時間による検知を行う検体であるため，通常の検証に加えて確認を複数回繰り返し実施した．

この結果，検証環境 1 での No.2 の検体の検証実験において稀に仮想マシンを検知する動作が確認された．これは，検体の rdtsc 命令や処理の実行時間が検体の持つしきい値よりも大きい際に仮想マシンとして判断する機能が動作した結果と考えられる．しかし，CPU スペック等の性能の高い検証環境 2 では，仮想マシンを検知されることは無かった．また，No.3 や No.11 では，いずれの環境でも検知回避オンの際に仮想マシンを検知されることは無かった．

これらの結果は，文献 [4] の調査結果の通り，rdtsc 命令や処理の実行時間による検知は，十分な性能を持つ環境では検知されないことを表している．また，性能の低い検証環境 1 においても No.2 のみが稀に検知をする場合があると言う状況であり，これらの機能により検知される可能性は非常に低いと考えられる．

4.3.6　No.10 の検証結果

No.10 の検体は検証環境 1 では，環境に対して有効な検知機能を有していないため検知回避オフでは想定通りに動作することを確認した．しかし，検知回避オンの状態では仮想マシンが検知された際の挙動となった．また，検証環境 2 では，検知回避オン，オフのいずれの状態でも仮想マシンが検知された際の挙動となった．

検証環境 2 における検知回避オフの際には同時に有効としている VMware Backdoor port を検知しているものと考えられるため問題はない．しかし，検知回避オンの際には，いずれの環境でも仮想マシンを検知した際の挙動を示している．これは，検知回避ルールに誤りがあり検知されるケースと，aaVMNetMonitor の API フック自体が検体の挙動に影響を与えるケースの 2 つの原因が考えられる．これを確認するため，検証環境 1 の 32bitOS 上で，検知回避オンの状態で検知回避ルールが存在しない状態で検体の挙動を確認した．

その結果，検知回避ルールが存在しない状態においても，同様に仮想マシンを検知した際の挙動となった．このことから，No.10 の検体は，API フック自体により影響を受けていることがわかる．No.10 の検体は文献 [4] の結果では，API フックの検知機能は有していないことが分かっており，継続して原因を調査する必要がある．

4.3.7　正しく動作しないケースの分析

これまでに述べた以外で，正しく動作しないケースに関して分析した結果について述べる．

No.7 の検体は，収集した環境情報をサーバーに送信する機能のみを有している．このため検体が動作する環境側では，解析環境の判定条件が不明と言う結果になった．また，自プロセス起動の一部が NetMonitor のサポートしていない ZwQueueApcThread を用いた手法で実装されており検体へのインジェクションが行えないことため送信情報の改ざんが行えなかった．これには，ZwQueueApcThread を用いたプロセス起動の方法を追加実装することで対応が可能である．

No.8 の検体は 64bitOS においては仮想マシン検知機能が有効に動作しないことが確認された．また，起動後に 32bit の検体プロセスから 64bit の explorer.exe プロセスにインジェクションを行うために強制終了していることが確認された．このことから．No.8 は，32bitOS のみを考慮して作成されたマルウェアであることが分かる．

© 2017 Information Processing Society of Japan

No.8 の検体は前述の通り，Windows API による情報収集による仮想マシン検知の後に外部との通信による検知で動作を停止していた．これは，No.8 の検体は動作環境が外部と正しく通信可能であるかを確認するため複数のサイトに対し HTTPS で接続し証明書の検証を実施し 1 つでも一致しない場合に動作を停止するためである．検証環境からはいずれの外部サーバに対しても接続可能な状態としたが含まれるサイトのうち 1 つの証明書が変更されていたため，それ以降のサイトに対する通信は行われなかった．この様に，外部と通信を行う機能に対しては，解析環境内に擬似サーバを用意する方法などが考えられる．本検体では，証明書の検証を行っており擬似環境を用意することが難しく，動作をさせることが困難な検体であると言える．

No.9 の検体に関しては，いずれの環境においても検体のインジェクション対象プロセスが強制終了する結果となった．これは，NetMonitor の有する API フック機能と検体が他プロセスにインジェクションする機能がお互いに干渉することで，この様な結果となったと考えられる．API フックの実装には，十分な考慮を行っているが対象となるマルウェアの全ての挙動が明らかではないためマルウェアの挙動により，この様な結果は発生する可能性が常に存在する．

4.4 未知のマルウェア検体セットを用いた検証

4.4.1 実験方法

未知のマルウェア検体セットとして The WildList[10] に登録されている検証開始時点で最新の 20170503 に追加された 594 検体のうち PE ファイルである 563 検体を用いて検証を行った．検証環境は，動的解析システム ZHR[11] のフレームワークを利用し，検証環境 1 のうち 32bitOS のみを用いて実行した．

検証方法は，aaVMNetMonitor の仮想マシン検知回避機能のオン，オフのみが異なる 2 つの仮想マシン上で同一の検体を並行して実行後に 15 分間放置し，出力される NetMonitor ログを収集し，検知回避オン・オフの違いによる検体毎のログファイル数およびログファイルサイズの違いを分析した．

なお，ログファイル数は検体および検体により生成・インジェクションされたプロセス数に相当し，ログファイルのサイズはプロセスの生成・インジェクションまたは通信が行われたことを示す．

4.4.2 分析対象検体の抽出

結果の分析にあたり，動的解析を実施した 563 検体の結果から分析対象とする検体を抽出した．

はじめに，検知回避オン・オフ双方で，NetMonitor ログのファイル数が 1 以下の検体とその他の検体に分類した．この条件に一致する場合，検体の起動に失敗または起動に成功したが検体プロセスへのインジェクションに失敗したことが判別可能である．この条件に一致する検体は，21 検体存在した．なお，検知回避オン・オフのどちらか一方だけがこの条件に一致する検体は存在しなかった．

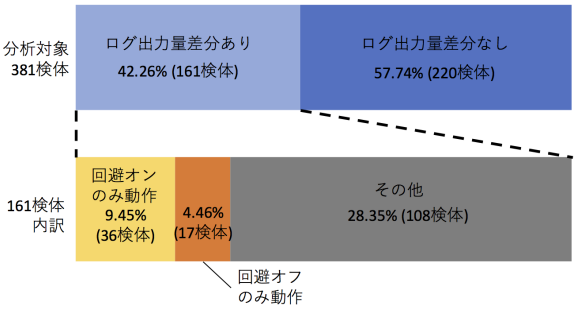

図 3 ログ分析結果
Fig. 3 Result of the Aanalysis Log

次に，検知回避オン・オフ双方で，NetMonitor ログのファイル数が 2 以下かつファイルサイズの合計が 1390bytes 以下の検体とその他に分類した．この条件に一致する場合，検体の起動および aaVMNetMonitor による検体プロセスへのインジェクション以外の挙動を示すログが一切出力されないことが判別可能である．この条件に一致する検体は，161 検体存在した．

これらの条件に一致する合計 182 検体を除外した 381 検体を対象に有効ログ出力量の分析を実施した．なお，これ以降，有効ログとは，検体の起動および aaVMNetMonitor による検体プロセスへのインジェクション以外の何らかのログが出力されたことを指す．

4.4.3 実験結果の概要

分析対象検体のログ分析結果を**図 3** に示す．分析対象検体のうち 220 検体では，検知回避オン・オフに関わらず検体の活動を示す有効なログを出力しており，その出力ファイル数が同一かつファイルサイズの差が 10%以内であったため同様の動作を行ったと考えられる．このことより，これらの検体は，仮想マシン検知機能を保有しないまたは，検証環境に対して有効な仮想マシン検知機能を保有しない検体であると考えられる．

分析対象検体のうち 161 検体では，検知回避オン・オフでログ出力量に差が見られた．ログ出力量に差がみられる検体については以下に分析結果を述べる．

4.4.4 仮想マシン検知回避の可否が判断可能な検体

ログの出力量に差がみられた検体のうち，36 検体では検知回避オンの状態でのみ有効ログが出力されたことを確認した．この結果から解析対象検体のうち 9.45%の検体が仮想マシン検知機能を有しており，検知回避機能によりそれらを回避することに成功したことが分かる．

これに対し，17 検体で，検知回避オフの状態でのみ有効なログが出力されたことを確認した．これは，解析対象検体のうち 4.46%の検体で検知回避機能を使用することで有

© 2017 Information Processing Society of Japan

効ログが得られないことを示している．これらの検体は，何らかの検知機能を有しており検知回避機能を使用することで仮想マシンであることが検知された場合と仮想マシン検知機能のバグ等で動作が停止した場合の双方が原因として考えられる．このため，これらの検体に対しては，個別に解析調査が必要であると考える．

4.4.5 仮想マシン検知回避の可否が判断困難な検体

ログの出力量に差がみられた検体のうち 108 検体に関しては，検知回避機能オン・オフの双方で，検体の活動を示す有効ログが検出されたが，その出力量に何らかの差が見られた検体である．これらの検体は，ログの出力量による分析では検知回避機能により検体の動作が変化したかを判断することは困難であった．そこで，ログの出力量に顕著な差（ログの出力ファイル数の差が 1 以下かつログファイルサイズの合計の差が 50%を超える）がみられる 10 検体を抽出して内容の比較を実施した．

その結果，10 検体全てで検知回避機能オン・オフのどちらか一方が起動後早期に動作を停止しており，仮想マシンを検知して動作を止めた可能性が高いことが判明した．なお，検知回避オンのみで検体の活動が継続したものが 9 検体，検知回避オフのみで検体の活動が継続したものが 1 検体であった．これにより，検知回避オン・オフ双方に有効なログがみられるケースにおいても，仮想マシン検知機能により検体の挙動が変化しているものが存在していることが確認された．また今回抽出した 10 検体においては，検知回避機能により仮想マシン検知を回避するケースが殆どであることがわかった．

その他の 98 検体についても検知回避機能により，動作が変化しているものが含まれると考えられる．しかし，現在の aaVMNetMonitor では仮想マシン検知機能により動作が変化したかの判断をする機能を有していないため，有効ログの有無以外での判断は困難である．このため，検知回避用 API フック対象の API 呼び出し履歴および検体プロセスの終了や長時間のスリープ等を判別する機能の追加が必要である．

5. 考察と今後の課題

検証用ソフトウェアを用いた検証結果より，aaVMNetMonitor では調査結果を基に想定した仮想マシン検出機能を回避可能であることを確認した．また，調査対象とした静的解析済みマルウェア（12 検体）を用いた検証実験では，検知回避機能が実際のマルウェアに対しても有効であることを確認した．これらの結果から，文献 [4] の提案手法が検証用ソフトウェアおよび調査対象の 12 検体に対して有効に機能することを実証した．

未知のマルウェア検体セット（文献 [10]）を用いて行った検証では，分析対象の検体のうち約 10%が何らかの仮想マシン検知回避機能を持っており，aaVMNetMonitor の検知回避機能により回避可能であることを確認した．しかし，動的解析を実施した 563 検体のうち 161 検体では，検知回避オン・オフに関わらず検体の起動のみしか確認出来ておらず，これらの中にも仮想マシン検知機能を有するマルウェアが含まれる可能性は十分に考えられる．このため，未知のマルウェア検体セットを用いた検証では，aaVMNetMonitor の検知回避機能は期待通りの効果は得られなかったと考える．しかし，前述の通り，対応済の仮想マシン検知機能に対しては回避可能であることが確認されている．また，現在 aaVMNetMonitor は，文献 [4] で調査済の全機能に対応していない．これらのことから，継続して検知回避機能の追加対応を行うことで，aaVMNetMonitor は，仮想マシン検知機能に対し有効なツールとなると考えられる．

今後の課題として，文献 [4] の調査結果のうち未実装の機能に関して対応を行う必要がある．Dyreza の検証実験で判明した ZwQueueApcThread を用いたプロセス起動も同様に対応が必要である．また，Rovnix や未知のマルウェア検体セットの一部で，仮想マシン検知回避機能を使用した際にのみ検体が動作を停止することが判明しているものに対して継続して調査を行い原因の究明および対応を実施する．

さらに，ツールの有効性を向上するために，仮想マシン検知機能の解析調査と検知回避機能の追加を継続的に行うことが必要である．加えてツールの有効性およびマルウェアの仮想マシン検知機能の有無を判別するため，仮想マシン検知回避機能の有効性可否を判断する機能の追加について検討を進める．

参考文献

[1] 津田　侑，神薗雅紀，遠峰隆史，安田真悟，三浦良介，宮地利幸，衛藤将史，井上大介，中尾康二：標的型攻撃再現のための攻撃シナリオ定義インタフェースの実装，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. 450–457 (2014).

[2] 国立研究開発法人情報通信研究機構：サイバー攻撃誘引基盤 “STARDUST”（スターダスト）を開発，（オンライン），入手先 ⟨https://www.nict.go.jp/press/2017/05/31-1.html⟩（参照 2017-06-14）.

[3] 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).

[4] 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，Vol. 2017 (2017).

[5] malware oompa loompas, T.: Automated Malware Analysis - Cuckoo Sandbox, (online), available from ⟨https://www.cuckoosandbox.org⟩ (accessed 2017-06-14).

[6] 大月勇人：仮想化技術に基づいたマルウェア解析のためのシステムコールトレース手法に関する研究，博士論文，立命館大学 (2016).

© 2017 Information Processing Society of Japan

[7] Kirat, D., Vigna, G. and Kruegel, C.: BareCloud: Bare-metal Analysis-based Evasive Malware Detection, *23rd USENIX Security Symposium (USENIX Security 14)*, San Diego, CA, USENIX Association, pp. 287–301 (online), available from ⟨https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/kirat⟩ (2014).
[8] Zhang, F., Leach, K., Stavrou, A., Wang, H. and Sun, K.: Using Hardware Features for Increased Debugging Transparency, *2015 IEEE Symposium on Security and Privacy*, IEEE, pp. 55–69 (2015).
[9] Lengyel, T. K., Maresca, S., Payne, B. D., Webster, G. D., Vogl, S. and Kiayias, A.: Scalability, Fidelity and Stealth in the DRAKVUF Dynamic Malware Analysis System, *Proceedings of the 30th Annual Computer Security Applications Conference*, ACSAC '14, New York, NY, USA, ACM, pp. 386–395 (online), DOI: 10.1145/2664243.2664252 (2014).
[10] International, W. O.: The WildList, WildList Organization International (online), available from ⟨http://www.wildlist.org/CurrentList.txt⟩ (accessed 2017-07-11).
[11] 株式会社セキュアブレイン：Zero-Houre Response System, 株式会社セキュアブレイン (online)，available from ⟨http://www.securebrain.co.jp/products/zhr/index.html⟩ (accessed 2017-07-11).

付　　録

A.1 仮想マシン検知回避ルール例

A.1.1 ルール A

適用対象 API: PathFileExistsA

```
[rule_A]
API = PathFileExistsA
ret_val = false
string = c:¥sampl¥epos.exe
string = *sandboxstarter.exe
```

ルール適用時の動作

PathFileExistsA の引数が string で指定されたいずれかの文字列に一致する場合は，ret_val = false の内容に従って戻り値を false で返却する．

A.1.2 ルール B

適用対象 API: DeviceIoContorol

```
[rule_B]
API = DeviceIoControl
rpl_str = ABCDEFG123456709
string = *array*
string = *virtual*
```

ルール適用時の動作

DeviceIoContorol でデバイスのプロダクト ID の取得が確認された場合，呼び出し結果のプロダクト ID が string で指定されたいずれかの文字列に一致する場合は，rpl_str = ABCDEFG123456709 の内容で一致した文字列を置換する．その際，rpl_str に指定された文字列が置換対象文字列より長い場合は，置換対象文字列長分だけ置換する．また，rpl_str に指定された文字列が置換対象文字列より短い場合は，置換対象文字列長分になるまで繰り返して置換を実施する．

A.1.3 ルール C

適用対象 API: 現在対応する API なし

```
[rule_C]
API = API 名
ret_val = 戻り値
```

ルール適用時の動作

本ルールタイプを適用された API が呼び出された場合は，常に ret_val = 戻り値の内容に従って戻り値を変更して返却する．

A.1.4 ルール D

適用対象 API:EmunWindows

```
[rule_D]
API = EmunWindows
string = ollydbg
srring = Wireshark
string = Process Explorer
```

ルール適用時の動作

EmunWindows の呼び出し結果のクラス名またはウィンドウ名が string で指定されたいずれかの文字列に一致する場合は，ルールに一致しない値が返却されるまで処理を継続し，ルールに一致しない結果のみを呼び出し元に返却する．

A.1.5 ルール E

適用対象 API:GetVolumeInformationA

```
[rule_E]
API = GetVolumeInformationA
pos = 1
val = OS
pos = 3
val = 0xE2A05C10
```

ルール適用時の動作

GetVolumeInformationA の呼び出し結果のアウト引数（API 定義で pos で指定された位置に存在する引数）を常に指定された値（pos の直後に存在する val に指定された値）に改ざんして返却する．

© 2017 Information Processing Society of Japan