異なる解析環境を組み合わせたステルス性の高い動的バイナリ計装によるマルウェアの通信解析

黒米 祐馬 †　　武田 圭史 †

† 慶應義塾大学 環境情報学部
252-0882 神奈川県藤沢市遠藤 5322
gomachan@sfc.wide.ad.jp, keiji@sfc.keio.ac.jp

あらまし マルウェアの通信と制御フローの関連性を解析する手法として動的バイナリ計装が知られている．動的バイナリ計装では，実行中のプログラムにコードを挿入することで，マルウェアの詳細な挙動を把握することができる．しかし，現状の手法はマルウェアから検知されやすく，検知されてしまうと期待される解析結果が得られない．そこで本研究では，動的バイナリ計装を適用したい処理のみを専用の解析環境上で実行し，それ以外の処理を検知されにくい解析環境上で実行することにより，マルウェアによる解析環境の検知を回避する．さらに，これを用いて動的バイナリ計装による通信の解析を実現する．

Malware Communication Analysis using Dynamic Binary Instrumentation in Heterogeneous Analysis Environments for Stealthiness

Yuma Kurogome†　　Keiji Takeda†

†Faculty of Environment and Information Studies, Keio University
5322 Endo, Fujisawa-shi, Kanagawa 252-0882 Japan
gomachan@sfc.wide.ad.jp, keiji@sfc.keio.ac.jp

Abstract DBI(Dynamic Binary Instrumentation) is a known method for analyzing the relevance between network communication and execution flow of malware. We can reveal details on malware activities by dynamically inserting code with DBI. However, the existing methods are easily detectable from malware, and when the analysis environment is detected, we cannot get expected analysis result. In this paper, we present our design and implementation of a dynamic malware analysis system using heterogeneous environments. We use DBI environment for communication analysis, and use stealth environment for avoiding sandbox detection.

1　背景

マルウェアの解析者は様々な技術やツールを用いてマルウェアを分析しようとするが，とりわけ，実際にマルウェアを実行し，その詳細な挙動を分析する動的解析と呼ばれる手法が広く用いられている．動的解析は一般的にサンドボックスと呼ばれる隔離された解析環境を用いて行われる．この解析環境には感染拡大の防止と環境の復旧が容易であることから仮想マシンモニタが用いられることが多い．一方で，マルウェアの開発者は動的解析を妨害すべく，解析環境の痕跡をもとに自身が解析下にあることを検知して動作を停止したり，無意味な動作を継続するといった機能をマルウェアに実装している．そのた

め，期待される解析結果を得るためには，マルウェアの解析環境はマルウェアから検知されないことが望ましい．

また，多くのマルウェアはC&Cサーバから受信したコマンドをもとに制御フローを変更し，コマンドに対応した処理を実行することが知られている．ここで，ネットワーク通信と制御フローの関連性を詳細に分析することのできる動的解析の手法に，動的バイナリ計装（Dynamic Binary Instrumentation）がある．動的バイナリ計装では，デバッガやエミュレータを用いてプログラムを実行し，任意のコードをプログラムに挿入することによりその挙動を分析する．これを用いたものに，解析対象にテイントタグと呼ばれる属性情報と伝搬ルールを付与し，メモリやレジスタにおけるテイントタグの伝搬を通してデータ間の依存関係を分析するテイント解析という手法がある．テイント解析はマルウェアがC&Cサーバから受信するコマンドのセマンティクスを復元する研究 [1] や，マルウェアの複数の通信先からC&Cサーバのみを抽出する研究 [2] に応用されている．

これまで，仮想マシンモニタ特有の情報に基づく解析環境検知と，その対策手法について広く検討がなされてきた．しかしながら，動的バイナリ計装に関しては複数の検知手法が提案されている一方でさしたる対策が講じられないまま現在に至っている．

2　研究目的

本研究では，多くのマルウェアにおいて解析環境検知が実行の早い段階で行われることに着目する．マルウェア開発者の主目的は感染拡大や情報窃取であり，これらはネットワーク通信を伴う．そして，それらの処理を分析させないために解析環境検知は行われる．

そこで，本研究では以下の仮説を提示する．

仮説 A　マルウェアによる解析環境検知はネットワーク通信処理よりも先んじて行われる．

仮説 B　マルウェアによる解析環境検知はネットワーク通信処理とは独立に行われる．

この仮説のもと，本研究では，解析環境検知を回避するための環境と動的バイナリ計装を用いてネットワーク通信処理を解析するための環境との間でマルウェアの実行状態を共有することで，マルウェアから検知されずに動的バイナリ計装を実現する．すなわち，解析環境検知とネットワーク通信処理というマルウェアの異なる処理に対して異なる解析環境を提供することにより，ネットワーク通信処理に対してのみ動的バイナリ計装を行う．

提案手法を用いた実験では，D3M データセットと独自に収集した検体について動的解析を実施し，ネットワーク通信処理の解析を行った．実験の結果，仮説 B の有効性が確認された．

3　関連研究

解析環境検知の回避を目的として開発された解析環境に Xen の上に実装された DRAKVUF[3] がある．DRAKVUF はゲスト OS の FS レジスタが保持する KPCR 構造体からのオフセットをもとに Windows のプロセスリストを構成する EPROCESS 構造体の情報を取得するため，ゲスト OS に挿入したプログラムを経由してプロセスリストを得る必要がない．DRAKVUF は CPU ネイティブの仮想化支援機能である Intel VT を用いるため検知されにくい．また，ゲスト OS の仮想物理アドレスとホスト OS の物理アドレスを紐付ける EPT の書き込み権限を操作することにより API をフックするが，これはゲスト OS から不可視である．DRAKVUF はゲスト OS のメモリとディスクをコピー・オン・ライト形式で扱うことができ，仮想マシンを容易に複製することができる．

DECAF[4] は DRAKVUF と同様にゲスト OS にプログラムを挿入することなくプログラムの実行を監視することができる．DECAF がゲスト OS のプロセスリストを取得する手法は DRAKVUF と同様である．しかし，DECAF は Intel VT を利用する Xen ではなくエミュレーションを利用する QEMU を拡張することにより動的バイナリ計装を実現している点で DRAKVUF と異なる．DECAF の利用者は共有ライブラリとして DECAF に読み込まれるプラグインを作成することにより動的バイナリ計装を行うことができる．QEMU におけるゲスト OS の実行は次の手順にしたがって行われる．

1. 実行対象のバイナリ [1] を分岐命令ごとに区切った基本ブロック単位で逐次的に逆アセンブルする．

[1] ゲスト OS とマルウェア検体

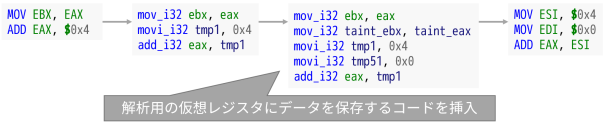

図 1: DECAF による動的バイナリ計装

2. 逆アセンブルした基本ブロックを中間表現に変換する．

3. 中間表現に変換した基本ブロックをホスト OS のアーキテクチャの命令に変換し，キャッシュする．

4. キャッシュされた基本ブロックの鎖を構築して実行する．

DECAF はこの中間表現からの変換過程にコールバック関数を挿入し，図 1 のように命令単位ないし基本ブロック単位の動的バイナリ計装を行う．しかし，QEMU の問題点として，低速であること，実機とは実行スケジューリングが異なること [5]，rep 命令や FPU に関する命令を適切にエミュレーションできないこと [6] や，実機とは異なり EFLAGS の更新が遅延評価されること [7] が指摘されている．これらの問題は DECAF においても同様であり，マルウェアから検知される要因となる．また，動的バイナリ計装の方式によっては，DECAF は QEMU よりも平均 605.07%低速化する [4]．

本研究のように，異なる動的解析環境を組み合わせることにより，マルウェアによる解析環境検知を回避しようとする試みに，DSM[6] と V2E[7] がある．

DSM は解析環境検知を回避するための環境として Ether[8] を，動的バイナリ計装を用いる解析環境として DECAF の前身である TEMU[9] を用いる．TEMU はプロセスリストを取得するためゲスト OS にカーネルモジュールの挿入を必要とする．よって，そのファイルの存在がマルウェアから検知される要因となる．また，TEMU のカーネルモジュールは MmLoadSystemImage および ZwQuerySystemInformation を用いてゲスト OS にロードされたモジュールの情報を取得するが，マルウェアはカーネルモジュールを用いてこれらの API をフックすることにより TEMU の解析結果を偽装することが可能である．DSM ではそれぞれの環境でマルウェアを並列に実行した際の挙動の差異から QEMU 上で実行されるコードを動的に変更し，マルウェアからの解析環境検知を回避する．しかし，この手法は複数の解析環境を同時に稼働させなければならないためハードウェア資源を圧迫する．

V2E は解析環境検知を回避するための環境として KVM を，動的バイナリ計装を用いる解析環境として TEMU を用いる．KVM は Intel VT を用いるため検知されにくい．V2E では KVM 上でマルウェアの実行ログを取得したのち，TEMU 上でそのログに対して擬似的に動的バイナリ計装を行い，マルウェアからの解析環境検知を回避した上で，動的バイナリ計装を用いてマルウェアをアンパックする．この手法では TEMU の検知については考慮しなくてもよいが，マルウェアを事実上二度にわたって実行しなければならない．

4 提案手法

本研究では，異なる解析環境の間でゲスト OS ごとマルウェアの実行状態を共有することで，動的バイナリ計装の欠点を補う．

ここで，DRAKVUF と DECAF の特徴を総合すると以下のようになる．

解析環境名	DRAKVUF	DECAF
仮想マシンモニタ	Xen 4.4.1	QEMU 1.0
仮想化方式	Intel VT	Emulation
速度	高速	低速
検知耐性	○	-
動的バイナリ計装	-	○
API フック	○	○

両者の利点と欠点を鑑み，本研究では，解析環境検知を回避するための環境として DRAKVUF を，動的バイナリ計装を用いてネットワーク通信処理を解析する環境として DECAF を用いる．DRAKVUF は Xen の非特権ドメインとしてゲスト OS を実行するが，DECAF はそれとは異なる非特権ドメインの Linux[2] 上で動作させる．ゲスト OS のイメージファイルは論理ボリュームマネージャを用いて両者から読み出せるようにする．また，ゲスト OS には Windows 7 SP1 32bit を用いる．

提案手法の概要を図 2 に示す．

提案手法は DRAKVUF と DECAF に加え，両者の間でゲスト OS の状態を共有するマネージャから

[2] ここでは Ubuntu 14.04 を用いる

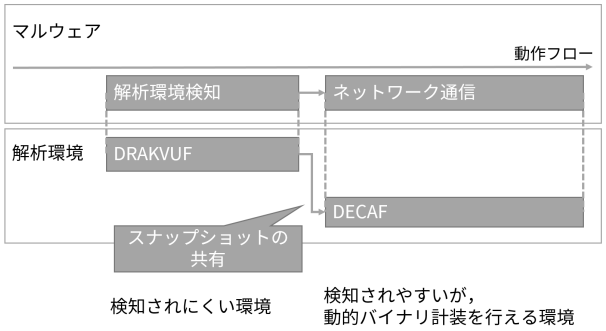

図 2: 提案手法概要

成る．マネージャは次のアルゴリズムにしたがって動的解析を実施する．

Input: 実行可能形式のマルウェア検体
Output: ログファイル
1: DRAKVUF 上でゲスト OS を起動
2: **while** true **do**
3: マルウェアを実行，ログ保存
4: **if** ws2_32.dll, wininet.dll の API 呼び出し直前 **then**
5: break;
6: **end if**
7: **end while**
8: ゲスト OS のスナップショットを取得
9: DECAF 上でゲスト OS を起動
10: スナップショットを適用
11: マルウェアを実行，ログ保存

提案手法では，マルウェアの状態に応じて解析環境を起動して切り替えるため，複数の解析環境を同時に稼働させる必要がなく，ハードウェア資源を節約することができる．

いずれの環境においても，マネージャは以下の DLL がエクスポートしている API の呼び出し履歴をログファイルに保存する．呼び出し履歴には API の名前と引数，その結果が含まれる．

- kernel32.dll, user32.dll, psapi.dll, advapi32.dll, ws2_32.dll, wininet.dll, ntdll.dll

これは，DRAKVUF においては EPT を用いた API フックの適用範囲を拡張することにより，DECAF においては hookapi_hook_function_byname API を用いることにより行う．スナップショットの取得および適用は，Xen と QEMU それぞれから読み書きすることのできる qcow2 イメージの形式に則り，メモリとレジスタ，実行クロックの状態を転送することにより行う．スナップショットの取得のトリガとなるのはマルウェアによって呼び出されようとしている API がネットワーク通信処理に関する ws2_32.dll および wininet.dll のエクスポート関数であるか否かである．

マルウェアが DECAF 上で実行されている場合，マネージャは API の呼び出し履歴に加え，次のアルゴリズムにしたがって動的バイナリ計装を行い，結果をログファイルに保存する．

1: **if** 基本ブロック単位のコールバック **then**
2: 基本ブロックの命令数を取得，ログ保存
3: **if** 命令単位のコールバック **then**
4: **if** 仮想 NIC の受信データを参照 **then**
5: 参照している命令を取得，ログ保存
6: **end if**
7: **end if**
8: **end if**

これは，マルウェアがネットワークからデータを受信する際，仮想 NIC で通信先を一意に特定できるタグを設定し，テイント解析を適用することにより行う．タグは図 1 で示した DECAF の仮想レジスタに記録され，データのコピーおよび参照に応じて他のデータに伝搬する．

ここで，API の呼び出し履歴と，動的バイナリ計装によって得られた受信データの参照履歴を照合することにより，どの API がデータの受信に用いられたか，どの API がそのデータを利用したか判別することができる．したがって，マルウェアが C&C サーバから受信したコマンドにどのようなねらいがあるか分析する端緒となる．

5 実験

5.1 実験 A

実験 A では，予備実験として仮説 A の検証を行った．

本実験では，2013 年 1 月から 2015 年 6 月までの間にセキュリティベンダのブログ等で解析環境検知を行うとの報告があったマルウェア 15 個を DECAF 上で実行した．実行にあたっては，既知の解析環境検知処理を検出し，コールバックを発行するよう DECAF のプラグインを作成し，wc2_32.dll および wininet.dll のエクスポート関数の呼び出しとの順序を確認した．また，wc2_32.dll のエクスポート関数が呼び出される前に DECAF 上でのマルウェアの実行が終了すれば報告にない解析環境検知が行われたと考えられる．

実験の結果，次の検体[3]によって提案手法の課題が明らかになった．

Trojan-Downloader.Win32.Banload.cwpb 解析環境検知よりも先んじてInternetOpenAを用いて異なるマルウェアのダウンロードを行った．解析環境検知はダウンロードされたマルウェアによって行われた．

Backdoor.Win32.Agent.dkbp 解析環境検知よりも先にWSAStartupによるソケットの初期化を行ったが，ネットワーク通信よりも先んじてRegOpenKeyExAを用いた解析環境検知を行った．

Backdoor.Win32.Dyreza.cu 初回実行の際は自身をスタートアップに登録するのみで，解析環境検知を行わなかった．

Trojan.Win32.Vilsel.plx GetCursorPosを用いてマウスカーソルの座標を確認し，座標に変動があるまでループを続ける

Backdoor.Win32.ZAccess.aqj NtDelayExecutionを用いて18時間待機したのちネットワーク通信を行った．

明らかになった課題は次の通りである．

- マルウェアによる解析環境検知はネットワーク通信処理よりも先んじて行われるわけではない．
- 初回実行の際は自身をスタートアップに登録するのみの検体について個別の対策をとらなければならない．
- マウスカーソルの変動の有無により解析を遅らせる検体について個別の対策をとらなければならない．
- Sleep相当のAPIにより解析を遅らせる検体について個別の対策をとらなければならない．

5.2 提案手法の改善

まず，マルウェアによる解析環境検知がネットワーク通信処理よりも後に行われる場合を考慮する．すなわち，ダウンローダー型の検体である場合と，ネットワーク通信処理の前後に解析環境検知を行うような高度な検体である場合である．これらを分析するためには，DECAFでネットワーク通信処理を行った後のマルウェアを再度検知されにくいDRAKVUF環境上に移して実行する必要が生じる．

ここで問題となるのは，何をもってネットワーク通信処理が終了したとみなすかである．ネットワーク通信に関するAPIの終了をもって関連する処理が終了したとみなしてしまっては，動的バイナリ計装による受信コマンドを分析することができない．

この問題を解決するため，本研究ではChange Finder[10]と呼ばれる変化点検出アルゴリズムを用いる．変化点検出は，時系列データを入力として将来の値を予測し，その予測値と観測値との間に隔たりが生じた点を検出する手法である．Change FinderはSDAR(Sequentially Discounting Auto Regressive)モデルという忘却型学習アルゴリズムを用いており，忘却により過去のデータの影響を廃することによって非定常なモデルの学習を行う．Change Finderは次の手順にしたがって時系列モデルの2段階学習を行う．

1. SDARアルゴリズムにより時系列データの確率密度関数を学習する．
2. 学習した確率密度関数の各時点ごとの外れ値スコアを平滑化する．
3. 移動平均スコアの時系列を構成する．
4. SDARアルゴリズムにより移動平均スコアの時系列の確率密度関数を学習する．
5. 学習した確率密度関数の各時点ごとの外れ値スコアを平滑化する．

Change Finderは第1段階学習によって時系列の外れ値を検出し，それらを平滑化することによりノイズに反応した外れ値をフィルタリングし，第2段階学習によって変化点のみを検出する．ここで，提案手法の改善方法として，Change FinderをDECAF上でのマルウェアによる受信データの利用状況に対して適用することで，ネットワーク通信とそれに付随した処理の終了を検出する．すなわち，基本ブロック中に仮想NICの受信データを参照する命令が含まれる状態を正常系とし，ネットワーク通信後に受信データを参照しなかった基本ブロックの連なりから変化点を検出する．これをもってDRAKVUF上にゲストOSとマルウェアの実行状態を切り替えるトリガとする．

[3]以下，マルウェアの表記にはカスペルスキーによる識別名を用いる

続いて，初回実行の際は自身をスタートアップに登録するのみの検体への対策であるが，これらは検体がネットワーク通信処理を一切行わなかった場合再度 DRAKVUF 上から実行するようマネージャにルールを設定することで対処する．

マウスカーソルの変動を確認するまではループを脱しない検体については，GetCursorPos の戻り値を毎度ランダマイズすることにより解析の遅延を回避する．

Sleep 相当の API により自身の実行を遅らせる検体については，API の引数を 0 ミリ秒に設定することにより，解析の遅延を回避する．しかし，次のようなコードを用いて，Sleep にパッチが当てられていないか確認する手法 [11] が知られている．

```
static inline int IsSleepPatched()
{
  DWORD time = GetTickCount();
  Sleep(500);
  if ((GetTickCount() - time) > 450)
    return 0;
  else
    return 1;
}
```

これは，CPU のクロックごとに加算されるタイムスタンプカウンタを GetTickCount を用いて参照し，Sleep の実行に期待されるクロック数が費やされていなかった場合に自身が解析下にあることを検知する手法である．また，rdtsc 命令を用いれば GetTickCount によらずタイムスタンプカウンタを参照することができる．

この手法を用いるマルウェアを分析するため，本研究では記号的実行を用いる．記号的実行は，解析対象となる処理の変数に具体値ではなく記号値を代入し，任意の実行パスに到達する条件を静的解析する手法である．ここで，マネージャを拡張し，次の手順にしたがって Sleep 相当の API とタイムスタンプカウンタを用いた解析環境検知を回避する．

直後の条件分岐に影響する値の算出には制約充足ソルバ Z3[12] を用いる．Z3 により，実行を継続するようなタイムスタンプカウンタの値と，解析環境であると検知されてしまうようなタイムスタンプカウンタの値を求めることができるが，それぞれがどちらに該当するか判別することはできない．そこで，スナップショットからそれぞれの値を設定した状態

1: DRAKVUF 上でゲスト OS を起動
2: **while** true **do**
3: 　マルウェアを実行，ログ保存
4: 　**if** GetTickCount, rdtsc 命令の呼び出し直前 **then**
5: 　　break;
6: 　**end if**
7: **end while**
8: ゲスト OS のスナップショットを取得
9: DECAF 上でゲスト OS を起動
10: スナップショットを適用
11: マルウェアの EIP 前後 20 命令を逆アセンブル
12: 直後の条件分岐に影響する値の算出
13: タイムスタンプカウンタの値の書き換え
14: 実行再開

でゲスト OS を実行し，即座 [4] にマルウェアの実行が終了するかどうかをもとに期待通りマルウェアを解析できているか判別する．なお，値の書き換えには DECAF の動的バイナリ計装を利用する．

これらの改善により，実験 A によって明らかになった問題点を解決した．

5.3 実験 B

実験 B では，仮説 B の検証を行った．

本実験では，D3M データセット（2012 2015）に含まれる 69 検体のうち，2015 年 8 月現在で正常に外部と通信を行った 13 個 [5] について，改善を加えた提案手法による動的解析を実施した．動的解析の実施時間はひとつの検体につき 30 分とした．

実験の結果，いずれの検体についてもネットワーク通信処理を適切に解析することができた．

Trojan-PSW.Win32.Tepfer.gen

例として D3M データセット 2015 に含まれる検体を解析したログファイルの一部を示す．なお，本検体はネットワーク通信よりも後に NtOpenFile を用いて Virtual Box の検知を試みる．

```
[ws2_32.dll]gethostbyname
+ EIP = 004036c5, EAX = 1564680
+ name = ftp.ami****.com
[ws2_32.dll]connect
+ EIP = 00403757, EAX = 0
[ws2_32.dll]setsockopt
+ EIP = 71ad2e5b, EAX = 0
[ws2_32.dll]send
```

[4] ここでは EIP から 50 命令以内
[5] 以下，D3M 検体

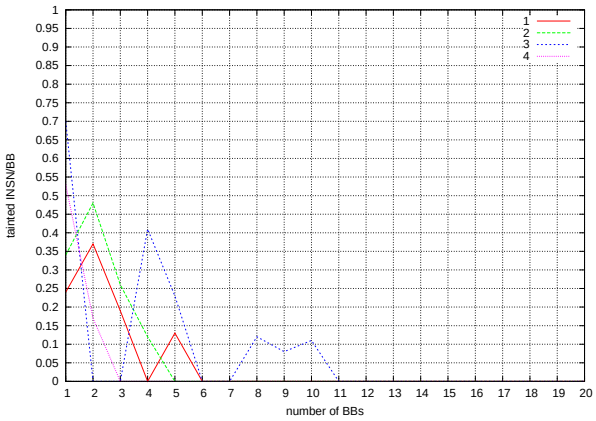

図 3: Trojan-PSW.Win32.Tepfer.gen

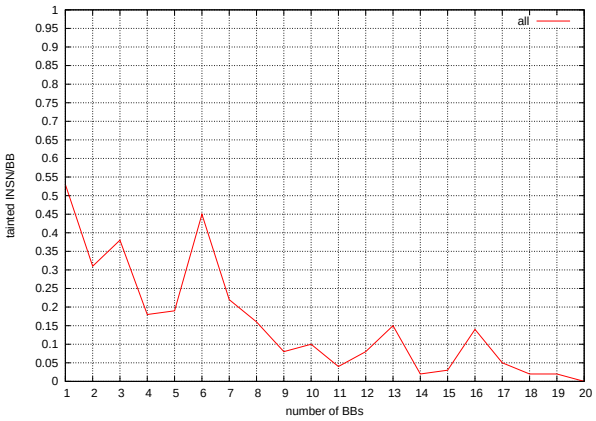

図 4: 全検体

```
+ EIP = 00403797, EAX = 266
+ buf =
POST /po/gate.php
HTTP/1.0 Host: ftp.ami****.com
Accept:*/*
Accept-Encoding:identity, *;q=0
Content-Length:507
...
[ws2_32.dll]WSARecv
+ EIP = 71ad2ea3, EAX = 0
+ tainted = true
...
[advapi32.dll]RegOpenKeyExA
+ EIP = 77f68c26, EAX = 0
+ tainted =true
+ hKey =
HKEY_LOCAL_MACHINE
+ lpSubKey =
Software\Microsoft\Windows\...
[advapi32.dll]RegCloseKey
+ EIP = 77f68b19, EAX = 0
...
```

本検体の解析中，4 回にわたって解析環境の切り替えが発生した．動的バイナリ計装を用いて，解析環境が DECAF に切り替わった各回について本検体による受信データの参照を分析した結果を図 3 に示す．横軸は初回の参照から起算した基本ブロックの個数であり，時系列に対応している．縦軸は受信したデータを参照した命令数をその命令が含まれる基本ブロックの命令数で割ったものである．

実験 B に用いた全検体に対して同様に受信データの参照を分析した結果を図 4 に示す．解析環境の切り替えは，最も少なかったもので 1 回，最も多かったもので 5 回行われ，平均 3.2 回であった．

6 考察

実験 A より，マルウェアによる解析環境検知は必ずネットワーク通信処理の先に行われるわけではない．

実験 B より，D3M 検体による解析環境検知はネットワーク通信処理とは独立に行われる．

また実験 B より，D3M 検体は受信したデータをその後最大 19 個目の基本ブロックにわたって参照する．提案手法に組み込んだ Change Finder は検体それぞれの受信データの参照状況に応じて解析環境の切り替えを行うが，変化点として通知された基本ブロックは初回の参照から起算して最大 21 個目であった．

したがって，D3M 検体において，受信データと制御フローの関連性を分析するためには，受信データの初回の参照から基本ブロック 20 個にわたって動的バイナリ計装を適用すればよいと言える．これまで述べてきた通り，動的バイナリ計装は検知されやすく，できるだけ狭い範囲での適用が望ましい．

また，提案手法は，ネットワーク通信処理と解析環境検知を同時に行うマルウェアに対して適用することができないが，D3M 検体にはそのようなマルウェアは含まれていなかった．

7 まとめ

本研究の貢献は次の通りである．

- ゲスト OS の実行状態を共有することにより，一度きりのマルウェア実行[6] で解析環境検知を回避し，動的バイナリ計装を用いてネットワーク通信処理を分析する手法を提案した．
- D3M 検体による解析環境検知はネットワーク通信処理とは独立に行われていることを示した．
- D3M 検体において，受信データと制御フローの関連性を分析するためには，受信データの初回の参照から基本ブロック 20 個にわたって動的バイナリ計装を適用すればよいことを示した．

8 謝辞

This material is partially based upon work supported by Asian Office of Aerospace Research and Development, U.S. Air Force Ofice of Scientic Research under Award No. FA2386-15-1-4068.

参考文献

[1] Younghee Park, D.S. Reeves, “Identification of Bot Commands by Run-time Execution Monitoring,” Annual Computer Security Applications Conference, pp. 321-330, Honolulu, HI, 2009.

[2] Zhaoyan Xu, Antonio Nappa, Robert Baykov, Guangliang Yang, Juan Caballero and Guofei Gu, “AUTOPROBE: Towards Automatic Active Malicious Server Probing Using Dynamic Binary Analysis,” Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pp. 179-190, Arizona, USA, 2014.

[3] Tamas K. Lengyel, Steve Maresca, Bryan D. Payne, George D. Webster, Sebastian Vogl, and Aggelos Kiayias, “Scalability, Fidelity and Stealth in the DRAKVUF Dynamic Malware Analysis System,” Proceedings of the 30th Annual Computer Security Applications Conference, pp. 386-395, NY, USA, 2014.

[4] Andrew Henderson, Aravind Prakash, Lok Kwong Yan, Xunchao Hu, Xujiewen Wang, Rundong Zhou, and Heng Yin, ”Make it work, make it right, make it fast: building a platform-neutral whole-system dynamic binary analysis platform,” Proceedings of the 2014 International Symposium on Software Testing and Analysis, ISSTA’14, pp.248-258, 2014.

[5] Thanasis Petsas, Giannis Voyatzis, Elias Athanasopoulos, Michalis Polychronakis and Sotiris Ioannidis, “Rage Against the Virtual Machine: Hindering Dynamic Analysis of Android Malware,” Proceeding EuroSec’14 Proceedings of the Seventh European Workshop on System Security, no. 5, NY, USA, 2014.

[6] Min Gyung Kang, Heng Yin, Steve Hanna, Stephen McCamant and Dawn Song, “Emulating Emulation-Resistant Malware,” Proceeding VMSec’09 Proceedings of the 1st ACM workshop on Virtual machine security, pp. 11-22, NY, USA, 2009.

[7] Lok-Kwong Yan, Manjukumar Jayachandra, Mu Zhang and Heng Yin, “V2E: Combining Hardware Virtualization and Software Emulation for Transparent and Extensible Malware Analysis,” Proceedings of the 8th ACM SIGPLAN/SIGOPS conference on Virtual Execution Environments, pp. 227-238, NY, USA, 2012.

[8] Artem Dinaburg, Paul Royal, Monirul Sharif and Wenke Lee, “Ether: Malware Analysis via Hardware Virtualization Extensions,” Proceedings of the 15th ACM Conference on Computer and Communications Security, pp. 51-62, NY, USA, 2008.

[9] Dawn Song, David Brumley, Heng Yin, Juan Caballero, Ivan Jager, Min Gyung Kang, Zhenkai Liang, James Newsome, Pongsin Poosankam, and Prateek Saxena, “BitBlaze: A New Approach to Computer Security via Binary Analysis,” Proceedings of the 4th International Conference on Information Systems Security, ICISS’08, pp.1-25, 2008.

[10] Kenji Yamanishi, Junichi Takeuchi, “A Unifying Framework for Detecting Outliers and Change Points from Non-Stationary Time Series Data,” Proceedings of the 8th International Conference on Knowledge Discovery and Data Mining, ACM SIGKDD’02, pp. 676-681, NY, USA, 2002.

[11] “a0rtega/pafish,” `https://github.com/a0rtega/pafish/`

[12] “Z3Prover/z3,” `https://github.com/Z3Prover/z3`

[13] 秋山満昭, 神薗雅紀, 松木隆宏, 畑田充弘, “マルウェア対策のための研究用データセット～MWS Datasets 2014～,” 情報処理学会研究報告コンピュータセキュリティ（CSEC） Vol. 2014-CSEC-66, no. 19, pp. 1-7, 2014.

[6] 初回実行の際は自身やダウンロードした他のマルウェアをスタートアップに登録するのみの検体を除く