Computer Security Symposium 2017
23 - 25 October 2017

デバイスドライバを用いたプロセス挙動保全ツールの提案

竹久 達也[1,2]　牧田 大佑[1]　神宮 真人[1,3]　丑丸 逸人[1,4]　福森 大喜[1,4]　津田 侑[1]　遠峰 隆史[1]
井上 大介[1]

概要：マルウェアの動的解析を行う際，マルウェアに含まれる解析環境検知技術が動的解析の妨げになることが問題となっている．そのため，これら解析の妨げになる検知技術を回避し動的解析可能にする提案も多い．ユーザーモードで動作するマルウェアは，ユーザーモードで動作する解析環境を検知しやすい．そのため，本稿では Windows のカーネルモードで動作するデバイスドライバだけでプロセス情報を収集し外部への送信を行うツールを提案する．また，提案するデバイスドライバにて収集したマルウェア挙動の一例を紹介する．

キーワード：マルウェア，動的解析，カーネルモード，デバイスドライバ

Preserving Tool for Process Behavior using Kernel Mode Device Driver

Tatsuya Takehisa[1,2]　Daisuke Makita[1]　Masato Jingu[1,3]　Hayato Ushimaru[1,4]　Daiki Fukumori[1,4]
Yu Tsuda[1]　Takashi Tomine[1]　Daisuke Inoue[1]

Abstract: Evasion techniques (e.g., analysis environment detection) implemented in malware are problematic for conducting the dynamic analysis. To overcome the evasion techniques, many proposals have been made for preventing the detection. Malware running in the user mode are able to detect analysis environments operating in the user mode easily. In this paper, we present a tool to collect process information by using a device driver operated in the kernel mode on Windows. We provide some experimental results of malware behavior obtained with the proposed tool.

Keywords: Malware, Dynamic Analysis, Kernel Mode, Device Driver

1. はじめに

マルウェアによる被害が社会問題となっている．たとえば，ボットに感染させることで DDoS 攻撃のようなサイバー攻撃に利用されたり，標的組織の保有する機器に RAT(Remote Administration Tool) を仕込み，それに向かって攻撃者が C&C(Command and Control) サーバからコマンドを送ることで，遠隔操作をされたり情報を窃取されたりといった被害が出ている．このような攻撃に用いられるマルウェアは日々進化し，既存のマルウェアを改造した亜種や新たな攻撃手法を実装した新種が登場することも日常的になっている．マルウェア解析者は新しいマルウェアが登場するたびに各々が持つ技術を駆使して解析するが，それには大きな人的コストを要する．

加えて，最近のマルウェアは，マルウェア解析者に簡単に解析させないよう，さまざまな工夫が施されている．その中の一つに，マルウェアが取り扱うデータの暗号化が挙げられる．たとえば，マルウェアは C&C サーバなどとの通信内容を暗号化することで，IDS の回避やマルウェアが行う通信目的・内容の隠蔽を図っている．さらに，マル

[1] 国立研究開発法人情報通信研究機構
National Institute of Information and Communications Technology
[2] 株式会社ニッシン
Nissin inc.
[3] 株式会社日立システムズ
Hitachi Systems, Ltd.
[4] 株式会社サイバーディフェンス研究所
Cyber Defense Institute, Inc.

ウェアの中には，動的解析システムなどの解析システムに解析されないようアンチデバッグ機能が含まれていることがあり，多くのアンチデバッグテクニックが存在する [1].

マルウェアの動的解析を行う際，通信内容を暗号化された場合は，暗号方式と復号するための鍵が取得できなければ，通信内容からだけではマルウェアが行う挙動を解析することが難しい．また，アンチデバッグ機能によって動的解析システムを検知すると，活動の停止や痕跡の削除などにより解析されないように妨害するマルウェアも存在する．

本稿では，これらの問題を解決するために，Windows OS 上で起動しているプロセスの挙動情報として，ファイルアクセス，レジストリアクセス，プロセスの生成と終了，スレッドの生成と終了，ネットワーク通信などを，ユーザーモードを利用せずにカーネルモードのみで情報収集し，外部に挙動ログを送信する手法について提案および実装し，評価を与える．

以下，本論文では，2 章で関連研究を述べる．3 章で提案手法を説明し，4 章で簡単な評価を与え，5 章で考察を述べ，最後に 6 章でまとめる．

2. 関連研究

マルウェアを動的解析するシステムとして，これまで多くのシステムが提案されている．本稿では，代表的な例として，Cuckoo Sandbox，Sysmon，Alkanet を取り上げる．

Cuckoo Sandbox[2] は，Cuckoo Foundation が提供する Open Source Software(OSS) であり，マルウェアの動的解析システムのデファクトスタンダートともいえるシステムである．Cuckoo Sandbox では，マルウェアを動作させるための特別なサンドボックス環境内でマルウェアを動作させ，マルウェアの挙動情報を得る．Cuckoo Sandbox でマルウェアを動作させる OS には Python やユーザモードエージェントをインストールする必要がある．

Sysmon は，Microsoft 社が提供するフリーソフトウエアである．Sysmon は，プロセスの挙動を得るために Windows のイベントログ機能を用いる．Sysmon では，カーネルモードデバイスドライバと，システムサービスの 2 つを利用して機能する．Sysmon は，VM 環境は必要とせず，仮想化できない環境にても利用が可能である．

Alkanet[4] は，Bitvisor と呼ばれる Virtual Machine Monitor(VMM) を利用した動的解析システムである．Alkanet は，マルウェアのアンチデバッグ機能を回避するために，マルウェアを動作させる OS には何も手を加えずに，改造した VMM を利用し情報を収集する．収集した情報は，VMM から IEEE1394 を用いて，ロギング用 PC へと転送する．Alkanet も Cuckoo Sandbox と同様に特別な環境が必要であり，仮想化できない環境では動作させることができない．

これらの動的解析システムの中でも，ユーザーモードプロセス（エージェント）が必要な Cuckoo Sandbox や Sysmon では，ユーザーモードで動作するマルウェアから検知されやすい．また，Alkanet は仮想化技術を利用するため，仮想化環境を検知する機能をもったマルウェアを解析することができない．

これらのことから，我々は，カーネルモードデバイスドライバのみでマルウェアの挙動を収集し外部へ転送するデバイスドライバを提案する．提案手法では，仮想化できない環境においても動作し，ユーザモードエージェントが存在しないため検知されにくく，仮想化環境を検知するマルウェアも動作させることが可能である．

3. 提案手法

3.1 設計方針

以下の方針の元，提案手法を設計した．

長期的な解析

多くの動的解析環境は設定された時間が満了すると解析を終了させることが多い．標的型攻撃のように長期間をかけて攻撃を成功させるようなマルウェアに対しては解析時間を設定することができない．そのため，提案手法では長期的な解析を視野に入れる必要があり，マルウェアを動作させる環境上に挙動情報を蓄えず外部に転送し，外部の受け取り装置などに記録を委ねることで長期間解析時に発生する挙動情報の保存場所の容量に関する問題を解決する．

64bit 環境でも動作可能

動的解析環境によっては，32bit 環境でのみ解析がおこなえるシステムもあるが，近年，64bit 環境でのみ動作するマルウェアの数も増えており 32bit 環境でも 64bit 環境でも挙動情報を収集する必要がある．そのため，32bit ／ 64bit 環境それぞれで違う情報収集を行わず，標準的な API を利用する設計を行う．また，動作する OS として，Windows 7 以降の複数のバージョンで動作させるためにも，標準的な API を用いる設計を行う．

ユーザーモードプロセスや仮想マシン環境が不要

マルウェアが解析環境で動作させられていることをアンチデバッグやアンチ VM などの検知手法で検知されにくくするために，専用のユーザーモードプロセスを必要とせず，動作する環境を選ばない（実マシン環境，仮想マシン環境を選ばない）カーネルモードでのみ動作する必要があり，カーネルモードデバイスドライバとして実装する必要がある．そのためデバイスドライバから利用できるカーネルモード API のみを利用した設計とする．前述のカーネルモード API のみを利用する制限により，提案手法では挙動情報としてファイルのアクセス，レジストリへのアクセス，プロセスの生成と削除，スレッドの生成と削除，DLL の

ロード，ネットワーク通信に絞って収集する．

一般的なデバイスを利用した情報出力

収集した挙動情報を出力する際，特別なハードウェアやソフトウェアが必要となると利便性が下がるため，一般的なデバイスを利用して情報を出力する必要がある．提案手法では，ネットワークデバイスを利用する方針とし，デバイスドライバから直接ネットワーク通信を行うために，Transport Driver Interface(TDI) を用いて挙動情報を出力する．また，出力するプロトコルはコネクションレス型の UDP とする．これは，出力先 IP アドレスの指定を変えるだけで，ユニキャスト通信やブロードキャスト通信が利用できる．UDP での通信は，一般的な PC で簡単に受信可能である．

上記，設計方針に基づき，カーネルモードで動作するデバイスドライバとして，挙動情報を収集し，出力する際の方法について簡単に述べる．

提案手法では，カーネルモードにおいてプロセスが引き起こす挙動の 1）ファイルアクセス，2）レジストリアクセス，3）プロセスの生成と削除，4）スレッドの生成と削除，5）DLL のロード，6）ネットワーク通信を収集し，外部へ転送する．

提案手法の全体構成を図 1 に示す．提案手法では，収集した各種情報を通知キューで一時的に保管し，システムスレッドとして生成した作業スレッドにより，順次 UDP 通信として外部へ転送する．UDP 通信は TDI を利用して，\\Device \\Udp に対して，IoCallDriver() で送信要求を行う．

以下では，挙動情報の取得方法について簡単に述べる．

3.2 ファイルアクセス

ファイルアクセス情報収集の概略を図 2 に示す．プロセスが行うファイルアクセスの挙動情報をカーネルモード API で得るには，ファイルシステム・ミニフィルタードライバー（以下，ミニフィルター）[5] としてデバイスドライバを作成し，登録・実行することで可能である．

各プロセスからのファイルシステムへの操作は，Win32/Native API により，デバイスドライバ呼び出しとして変換される．変換された呼び出しの呼び出し先が，ファイルシステムドライバの場合，I/O マネージャは呼び出しをフィルターマネジャーへと引き渡す．フィルターマネジャーは，本来呼び出す先のファイルシステムドライバを呼び出す前に，事前操作（Pre-Operation）として，登録されているミニフィルターのコールバック関数を登録されている深度（Altitude）が大きい順に呼び出す．ミニフィルターをすべて呼び出した後，各ミニフィルターで呼び出しが破棄されていなければ，本来のファイルシステムドライバを呼び出す．ファイルシステムドライバが処理を行った後，フィルタマネージャに処理が戻ってくると，次は，事後操作（Post-Operation）として，登録されているミニフィルターのコールバック関数を Altitude が小さい順に呼び出す．PostOperation の呼び出しが終了すると，I/O マネージャへ制御を戻し，I/O マネージャが呼び出し元に結果を返す．

提案手法では，ミニフィルターとして事前操作，事後操作をインターセプトすることで，どのプロセスのどのスレッドが，どのファイル名のファイルに対して Create/Open，Read，Write，Delete を行ったか記録する．また，ファイルの Delete が行われた際は，そのファイルの SHA1 ハッシュ値を記録する．これは，削除されたファイルのハッシュ値から外部のデータベースなどを利用して照合をかけることで，マルウェアによって 2 次生成されたファイルの実体を得たりするために役立つ．取得した情報の一例を図 3 に記載する．

3.3 レジストリアクセス

プロセスによるレジストリへのアクセス情報をカーネルモード API を利用して得るには，Windows カーネルモード・コンフィグレーションマネージャルーチンの CmRegisterCallback() を呼び出し，コールバック関数を登録することで可能である [6].

提案手法では，この CmRegisterCallback() を利用し，登録したコールバック関数内で，どのプロセスのどのスレッドが，どのレジストリキーを Create/Write(Set)/Delete を行ったか記録する．取得した情報の一例を図 3 に記載する．

3.4 プロセスの生成と削除

プロセスの生成・削除情報をカーネルモード API を利用して得るには，Windows カーネルモード・プロセスマネージャルーチンの PsSetCreateProcessNotifyRoutine() を呼び出し，コールバック関数を登録することで可能である [7].

提案手法では，PsSetCreateProcessNotifyRoutine() を利用し，登録したコールバック関数内で，生成・削除されたプロセスの，プロセス ID(PID) や親プロセス ID(PPID)，プロセス名，ユーザ名，コマンドライン引数を記録する．取得した情報の一例を図 3 に記載する．

3.5 スレッドの生成と削除

プロセス内におけるスレッドの生成・削除情報をカーネルモード API を利用して得るには，Windows カーネルモード・プロセスマネージャルーチンの PsSetCreateThreadNotifyRoutine() を呼び出し，コールバック関数を登録することで可能である [7].

提案手法では，PsSetCreateThreadNotifyRoutine() を利用し，登録したコールバック関数内で，生成・削除され

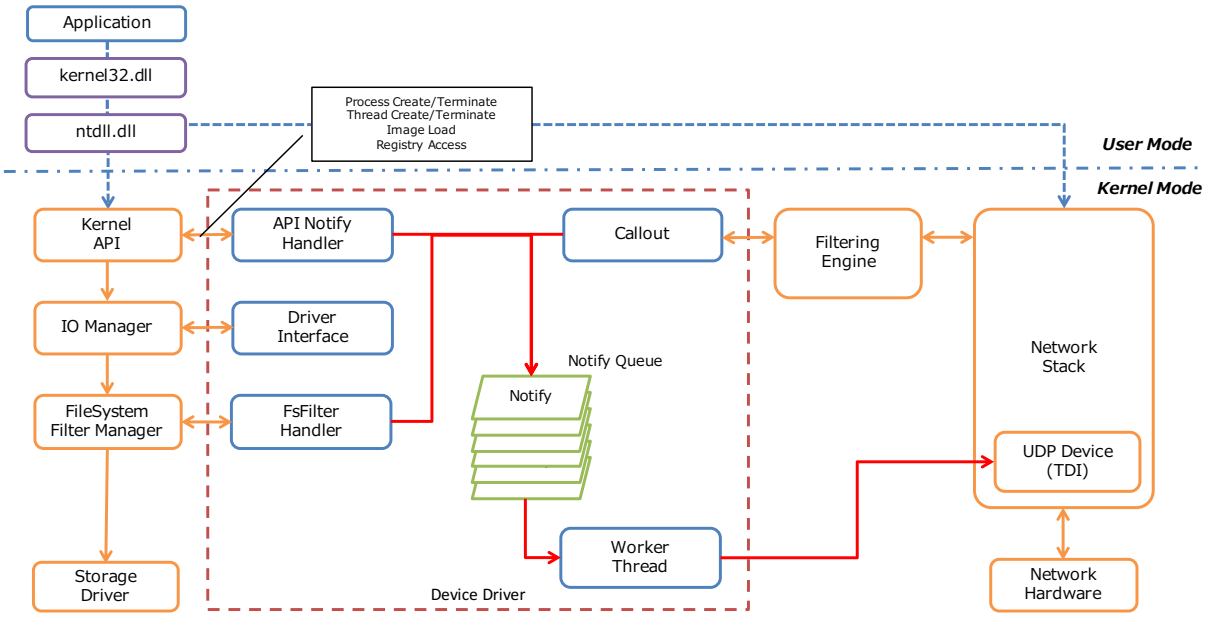

図 1　全体構成図

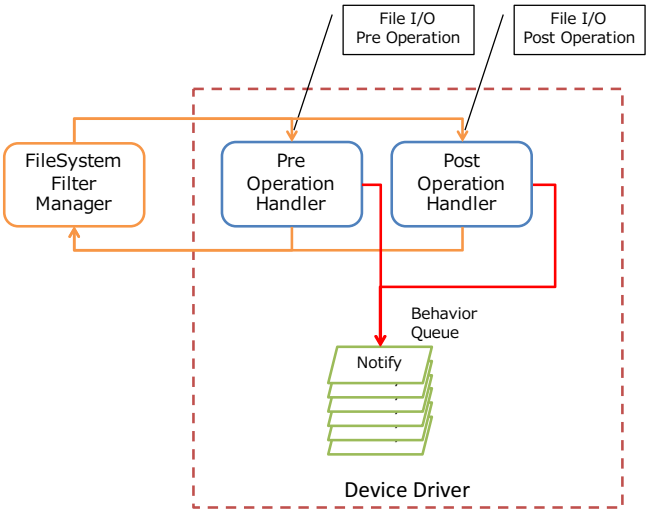

図 2　ファイルアクセス情報収集

たスレッドの，プロセス ID(PID) やスレッド ID(TID)，プロセス名を記録する．取得した情報の一例を図 3 に記載する．

3.6　DLL のロード

プロセスにおける DLL のロード情報をカーネルモード API を利用して得るには，Windows カーネルモード・プロセスマネージャルーチンの PsSetLoadImageNotifyRoutine() を呼び出し，コールバック関数を登録することで可能である [7].

提案手法では，PsSetLoadImageNotifyRoutine() を利用し，登録したコールバック関数内で，ロードしたプロセスの，プロセス ID(PID) や DLL ファイル名，DLL が読み込まれたアドレス，DLL が読み込まれたサイズを記録する．取得した情報の一例を図 3 に記載する．

3.7　ネットワーク通信

プロセスが行うネットワーク通信情報をカーネルモード API を利用して得るには，Windows フィルタリング プラットフォームを利用した Callout ドライバにより可能である [8].

提案手法では，コネクション型通信 (TCP) の情報を得るために，Callout として，Application Layer Enforcement(ALE) の，FWPM_LAYER_ALE_AUTH_CONNECT, FWPM_LAYER_ALE_AUTH_RECV_ACCEPT, FWPM_LAYER_ALE_FLOW_ESTABLISHED を利用し，通信元と通信先，ポート番号を記録する．また，コネクションレス型通信 (UDP) の情報を得るために，Callout として，Transport Layer の FWPM_LAYER_OUTBOUND_TRANSPORT, FWPM_LAYER_INBOUND_TRANSPORT を利用して，通信元と通信先，ポート番号，送受信データサイズを記録する．

また，提案手法では通信パケットのペイロードまでは取得しておらず，ペイロードの保全は外部のネットワークパケット収集装置などに任せ，収集装置との突合を行うために必要な情報を得ることだけを行う．取得した情報の一例を図 3 に記載する．

4.　評価と実験

本章では，提案手法の実装を用いることによるパフォーマンスへの影響を調べるための評価と，マルウェア検体を用いた実験結果を示す．

```
＜プロセス・スレッドの生成と削除＞
2017/08/28 02:54:10.417 PROCESS-CREATE 2016 160836 C:¥Users¥Xxx¥TestApp.exe Xxx-PC¥Xxx "C:¥Users¥Xxx¥TestApp.exe"
2017/08/28 02:54:10.417 THREAD-CREATE 160836 160840 C:¥Users¥Xxx¥TestApp.exe
2017/08/28 02:54:36.761 THREAD-TERMINATE 160836 160840 C:¥Users¥Xxx¥TestApp.exe
2017/08/28 02:54:37.807 PROCESS-TERMINATE 2016 160836 C:¥Users¥Xxx¥TestApp.exe Xxx-PC¥Xxx "C:¥Users¥Xxx¥TestApp.exe"

＜ファイルの生成，書き込み，読み込み，ハッシュ値，削除＞
2017/08/28 02:54:10.472 FILE-CREATE(OVERWRITE_IF) 2016 160836 ¥Device¥HarddiskVolume2¥Users¥Xxx¥TestApp.exe Xxx-PC¥Xxx ¥Device¥HarddiskVolume2¥Users¥Xxx¥TestFile.dat
2017/08/28 02:54:10.472 FILE-WRITE 2016 160836 ¥Device¥HarddiskVolume2¥Users¥Xxx¥TestApp.exe ¥Device¥HarddiskVolume2¥Users¥Xxx¥TestFile.dat OFFSET:0,REQ:4096,RES:4096
2017/08/28 02:54:10.472 FILE-READ 2016 160836 ¥Device¥HarddiskVolume2¥Users¥Xxx¥TestApp.exe ¥Device¥HarddiskVolume2¥Users¥Xxx¥TestFile.dat OFFSET:0,REQ:4096,RES:4096
2017/08/28 02:54:10.472 FILE-HASH 2016 160836 ¥Device¥HarddiskVolume2¥Users¥Xxx¥TestApp.exe ¥Device¥HarddiskVolume2¥Users¥Xxx¥TestFile.dat 1CEAF73DF40E531DF3BFB26B4FB7CD95FB7BFF1D
2017/08/28 02:54:10.472 FILE-DELETE 2016 160836 ¥Device¥HarddiskVolume2¥Users¥Xxx¥TestApp.exe ¥Device¥HarddiskVolume2¥Users¥Xxx¥TestFile.dat

＜レジストリへの書き込み＞
2017/08/28 02:59:48.671 REGISTRY-SETVALUE(REG_SZ) 169168 C:¥Users¥Xxx¥TestApp.exe ¥REGISTRY¥MACHINE¥SYSTEM¥ControlSet001¥services¥Beep¥DisplayName Beep

＜DLLのロード＞
2017/08/28 02:54:11.478 IMAGE-LOAD 160876 ¥Windows¥System32¥kernel32.dll BASE:0000000076BE0000,SIZE:1175552

＜ネットワーク通信　UDP送信受信，TCP接続＞
2017/08/28 02:54:11.175 NET-SENDTO 1452 LOCAL:10.0.2.15:54013 REMOTE:239.255.255.250:1900 PROTO:17 LEN:213 ¥device¥harddiskvolume2¥windows¥system32¥svchost.exe
2017/08/28 02:54:11.175 NET-PACKETOUT 1452 LOCAL:10.0.2.15:54013 REMOTE:239.255.255.250:1900 PROTO:17 LEN:213
2017/08/28 02:54:11.175 NET-PACKETIN 0 LOCAL:239.255.255.250:1900 REMOTE:127.0.0.1:54014 PROTO:17 LEN:161
2017/08/28 02:54:11.175 NET-RECVFROM 1452 LOCAL:239.255.255.250:1900 REMOTE:127.0.0.1:54014 PROTO:17 LEN:161 ¥device¥harddiskvolume2¥windows¥system32¥svchost.exe
2017/08/28 03:17:50.302 NET-CONN(ACTIVE) 2092 LOCAL:10.0.2.15:49157 REMOTE:23.51.210.106:443 PROTO:6 LEN:0 ¥device¥harddiskvolume2¥program files¥internet explorer¥iexplore.exe
```

図 3　収集情報の例

表 1　計測環境

OS	Windows 7 Professional Service Pack 1 64bit
CPU	Intel Core i7 920 2.66GHz
RAM	6GB(DDR3,1066MHz)
Storage	500GB(WD5003ABYX-18EWRA0)

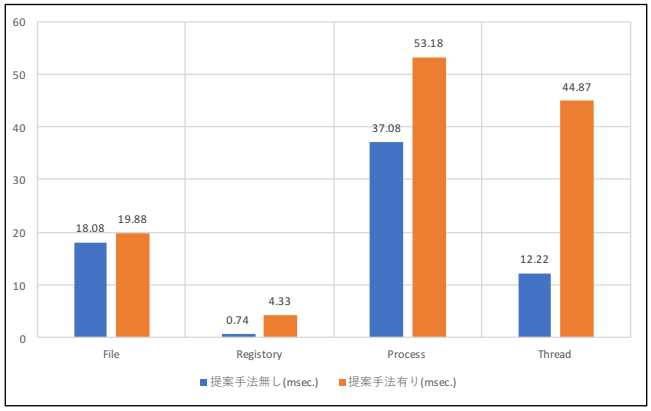

図 4　パフォーマンス計測結果

4.1　パフォーマンス

本節では，提案手法を用いることで，PC がどのくらい遅くなっているかオーバーヘッドを計測する．

計測した環境は表 1 の通りである．計測した項目は，ファイルアクセス，レジストリアクセス，プロセスの生成と終了，スレッドの生成と終了である．

ファイルアクセス項目として，計測したファイルへのアクセスは，空のデータファイルを生成し（CreateFile() API），そのファイルに対して，ファイルの先頭に 4KB のデータを書き込み（WriteFile() API），ファイルの先頭から 4KB のデータ読み込み（ReadFile() API），ファイルのクローズ（CloseHandle() API），ファイルの削除（DeleteFile() API）を行い．そのデータファイルの生成〜ファイルの削除までを 10 回繰り返した時の時間を計測し，それを 100 回計測した時の平均値を計測値とし，それを提案手法有り，無しの時間を計測した（図 4 の File）．

レジストリアクセス項目として，計測したレジストリへのアクセスは，レジストリ・キーのオープン（RegOpenKeyEx() API），文字列レジストリ値の読み出し（RegQueryValueEx() API），文字列レジストリ値の書き込み（RegSetValueEx() API），レジストリ・キーのクローズ（RegCloseKey() API）を行い，そのレジストリ・キーのオープン〜クローズまでを 100 回繰り返した時の時間を計測し，それを 100 回計測した時の平均値を計測値とし，それを提案手法有り，無しの時間を計測した（図 4 の Registory）．

プロセスの生成と終了項目として，計測したプロセス生成と終了は，cmd.exe を引数”/C exit”でプロセスの生成（CreateProcess() API），プロセスの終了待ち（WaitForSingleObject() API）を行い，そのプロセス生成〜終了待ち完了までを 10 回繰り返した時の時間を計測し，それを 100 回計測した時の平均値を計測値とし，それを提案手法有り，無しの時間を計測した（図 4 の Process）．

スレッドの生成と終了項目として，計測したスレッド生成と終了は，スレッドの生成（ CreateThread() API に，CREATE_SUSPENDED フラグ付加），スレッドの破棄（TerminateThread() API）を行い，そのプロセス生成〜終了待ち完了までを 10 回繰り返した時の時間を計測し，それを 100 回計測した時の平均値を計測値とし，それを提案手法有り，無しの時間を計測した（図 4 の Thread）．

図 4 を見ると，ファイルアクセス項目では，提案手法無しを 1 としたとき，提案手法有りは 1.10 倍遅く，レジストリアクセス項目では，提案手法有りでは 5.88 倍遅く，プロセスの生成と終了項目では，提案手法有りでは 1.43 倍遅く，スレッドの生成と終了項目では，提案手法有りでは 3.67 倍遅いことがわかる．

結果からみると，各項目で 1.10〜5.88 倍と提案手法有りと無しで遅さ（オーバーヘッド）にばらつきがみられ，特にレジストリアクセスに関しては 5.88 倍遅い結果となった．

表 2　実験で使用したマルウェア検体

ファイル名	2017_富士山会合プログラム.txt.lnk
ハッシュ値（MD5）	25dacaf77a7c4c0f5ea4ddafe83e1032
ファイルタイプ	Windows shortcut

図 5　実行検体のプロセスに関するプロセスツリー（一部）

4.2　ケーススタディ

本節では，提案手法の有効性を示すため，実際のマルウェア検体を我々の研究基盤 [9] で動作させた結果を述べる．

表 2 に本実験で使用したマルウェアの概要を示す．このマルウェアを実行すると，指定された URL から mshta.exe がファイルを取得し，その中に記載されている Powershell のスクリプトが実行される．この Powershell スクリプトは，Empire[12] と呼ばれる OSS の Post-Exploitation フレームワークの一部であり，実行後，この Empire が C&C サーバと通信を行った．Empire の通信は暗号化されており，通信からのみでは攻撃者の挙動を把握することは困難である．

提案手法で取得したしたプロセス情報について，Empire 以下のプロセスツリーの一部を図 5 に，攻撃者によって実行されたコマンド群の一部を図 6 に示す．この結果より，攻撃者がホスト内やネットワーク内を探索したり，OSS のパスワード取得ツール laZagne[13] を用いてパスワード等の機密情報を収集しようとしたりしていたことが，提案手法により取得できていることが確認できる．

5.　考察

本章では，提案手法に対する検知方法について簡単に述べ，それらへの対策についての考察と，評価結果に対する考察を与える．

5.1　検知手法

マルウェアが提案手法を検知する方法としては，以下の方法が考えられる．

コマンドによる検知

提案手法は，Windows カーネルモードデバイスドライバであるため，driverquery コマンドを実行することでインストールされていることが取得できる [10]．

Process Status API(PSAPI) による検知

デバイスドライバを列挙するための API である，EnumDeviceDrivers() と，GetDeviceDriverBaseName() を利用することで，システムにインストールされているデバイスドライバを取得することができる [11]．提案手法のデバイスドライバ名を攻撃者が把握している場合，デバイスドライバ名から検知することが可能である．

ファイルによる検知

一般に，Windows におけるデバイスドライバの格納位置は，%systemroot%\system32\drivers であるため，提案手法のデバイスドライバ名を攻撃者が把握している場合，そのフォルダの.sys ファイルを列挙することで検知することが可能である．

パケットキャプチャによる検知

WinPcap などのパケットキャプチャを利用することで，提案手法が出力するパケットを検知することができる．

上記，検知方法の対策方法として，コマンドと API による検知方法に対しては，デバイスドライバ側からの APC インジェクションなどを利用した API フックを行い，API からの戻り値を改変することで検知を回避できると考えられる．

ファイルによる検知方法に対しては，提案手法のデバイスドライバの実体である.sys ファイルを，特定のプロセス以外は列挙・オープンされても存在しないように隠すことで検知を回避できると考えられる．

パケットキャプチャによる検知に対しては，提案手法では TDI レイヤーでのパケット送信を行っているが，さらに低いレイヤーから送信できるように工夫するか，WinPcap などが利用する DLL ファイルやデバイスドライバを無効化することで検知を回避できると考えられる．

5.2　パフォーマンスの計測結果に対する考察

4.1 節で示したパフォーマンス計測結果のうち，レジストリアクセスとスレッド生成と削除項目の提案手法の実装によるオーバーヘッドが大きくなっている原因として，Windows 内におけるレジストリアクセスとスレッド生成と削除は非常に高速に処理が行われるため，提案手法の実装で費やされる時間の割合が大きくなることが原因と考えられる．

また，著者らが解析基盤において実装されたドライバを実行させた際，体感上ではあるが操作が遅くなったりすることはなかった．

6.　まとめと今後の課題

本稿では，マルウェアの動的解析を行う際に利用するツールとして，Windows のカーネルモードファイルフィルタドライバのみで，プロセスの挙動情報を外部へ転送することにより，マルウェア挙動の解析及び挙動情報の保全を行う手法を提案した．また，提案手法を利用する際に発

```
2017年07月25日 18:56:20.082 C:¥Windows¥system32¥cmd.exe /C net user /domain
2017年07月25日 18:57:25.680 C:¥Windows¥system32¥cmd.exe /C net user /domain
2017年07月25日 19:04:38.674 C:¥Windows¥system32¥cmd.exe /C net use Xxx/domain
2017年07月25日 19:05:42.119 C:¥Windows¥system32¥cmd.exe /C net user Xxx/domain
2017年07月25日 20:15:46.046 C:¥Windows¥system32¥cmd.exe /C dir c:¥windows¥tem¥w*
2017年07月25日 20:15:58.791 C:¥Windows¥system32¥cmd.exe /C dir c:¥windows¥temp¥w*
2017年07月25日 20:16:17.917 C:¥Windows¥system32¥cmd.exe /C dir
2017年07月25日 20:17:44.107 C:¥Windows¥system32¥cmd.exe /C dir *.zip
2017年07月25日 20:18:03.373 C:¥Windows¥system32¥cmd.exe /C move Windows.zip ../
2017年07月25日 20:18:11.469 C:¥Windows¥system32¥cmd.exe /C dir ..
2017年07月25日 20:20:03.946 C:¥Windows¥system32¥cmd.exe /C ..¥Windows¥laZagne.exe -?
2017年07月25日 20:20:18.656 C:¥Windows¥system32¥cmd.exe /C ..¥Windows¥laZagne.exe -all
2017年07月25日 20:20:37.345 C:¥Windows¥system32¥cmd.exe /C ..¥Windows¥laZagne.exe all
2017年07月25日 20:20:51.791 C:¥Windows¥system32¥cmd.exe /C ..¥Windows¥laZagne.exe all -v
2017年07月25日 20:22:02.911 C:¥Windows¥system32¥cmd.exe /C dir ..
2017年07月25日 20:22:12.615 C:¥Windows¥system32¥cmd.exe /C dir ../Windows
2017年07月25日 20:22:28.792 C:¥Windows¥system32¥cmd.exe /C dir ..
2017年07月25日 20:22:38.292 C:¥Windows¥system32¥cmd.exe /C net use
2017年07月25日 20:23:38.181 C:¥Windows¥system32¥cmd.exe /C netstat -afq
2017年07月25日 20:24:01.394 C:¥Windows¥system32¥cmd.exe /C netstat -af
2017年07月25日 20:31:36.992 C:¥Windows¥system32¥cmd.exe /C net group /domain
2017年07月25日 20:32:55.398 C:¥Windows¥system32¥cmd.exe /C net group “Domain Computers”
2017年07月25日 20:32:59.298 C:¥Windows¥system32¥cmd.exe /C net group “Domain Computers” /domain
2017年07月25日 20:39:54.462 C:¥Windows¥system32¥cmd.exe /C net user /domain
```

図 6 攻撃者によるコマンド実行のリスト（一部）

生する速度的な遅延に関する結果と，提案手法を利用してマルウェアの挙動を解析した結果を示すことで本提案手法の有効性を示した．

今後の課題としては，長期的にマルウェアを動作させることでマルウェア感染後の偵察活動の把握や本提案手法がマルウェアから検知されないような仕組みの検討と実装，APC などを利用したマルウェアが利用する API 呼び出し情報の収集方法の検討，カーネルモードからユーザーモードアプリケーション挙動のより詳しい情報収集法の検討などが挙げられる．

謝辞

本研究に関して多大な御協力戴いた，NTT セキュアプラットフォーム研究所の岩村 誠 氏，NICT の金谷 延幸 氏，NTT アドバンステクノロジ株式会社の清水 雄介 氏，岩崎 圭佑 氏に謝意を表する．

参考文献

[1] Symantec, Windows Anti-Debug Reference, 入手先 ⟨https://www.symantec.com/connect/articles/windows-anti-debug-reference⟩
[2] Cuckoo Foundation, Cuckoo Sandbox, 入手先 ⟨https://cuckoosandbox.org⟩
[3] Microsoft, Windows Sysinternals - Sysmon, 入手先 ⟨https://docs.microsoft.com/en-us/sysinternals/downloads/sysmon⟩
[4] Y. Otsuki at el.,“Alkanet: A Dynamic Mal- ware Analyzer based on Virtual Machine Mon- itor,”In World Congress on Engineering and Computer Science 2012 (WCECS 2012), Vol. 1, pp. 36–44 (2012).
[5] Microsoft, File System Minifilter Drivers, 入手先 ⟨https://docs.microsoft.com/ja-jp/windows-hardware/drivers/ifs/file-system-minifilter-drivers⟩
[6] Microsoft, Filtering Registry Calls, 入手先 ⟨https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/filtering-registry-calls⟩
[7] Microsoft, Windows Kernel-Mode Process and Thread Manager, 入手先 ⟨https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/windows-kernel-mode-process-and-thread-manager⟩
[8] Microsoft, Windows Filtering Platform Callout Drivers, 入手先 ⟨https://docs.microsoft.com/ja-jp/windows-hardware/drivers/network/windows-filtering-platform-callout-drivers2⟩
[9] NICT, サイバー攻撃誘引基盤“STARDUST”（スターダスト）を開発, 入手先 ⟨https://www.nict.go.jp/press/2017/05/31-1.html⟩
[10] Microsoft, Technet - Driverquery, 入手先 ⟨https://technet.microsoft.com/ja-jp/library/bb490896.aspx⟩
[11] Microsoft, Enumerating All Device Drivers in the System, 入手先 ⟨https://msdn.microsoft.com/en-us/library/windows/desktop/ms682619(v=vs.85).aspx⟩
[12] Powershell Empire, 入手先 ⟨http://www.powershellempire.com/⟩
[13] AlessandroZ/LaZagne: Credentials recovery project 入手先 ⟨https://github.com/AlessandroZ/LaZagne⟩