# マルウェア解析のためのシステムコールトレースログと通信の対応付け手法

大倉 有喜 † 大月 勇人 † 田中 恭之 ‡ 明田 修平 †
瀧本 栄二 † 毛利 公一 †

† 立命館大学
525-8577 滋賀県草津市野路東 1-1-1
{yookura, yotuki, saketa, takimoto, mouri}@asl.cs.ritsumei.ac.jp

‡NTT コミュニケーション株式会社
108-8118 東京都港区芝浦 3-4-1 グランパークタワー 16F
yasuyuki.tanaka@ntt.com

**あらまし** ボットに代表されるマルウェアは，設定ファイルの取得やコマンドの受信のために外部と通信する．そのため，動的解析では，外部と通信できる環境でマルウェアを実行し，感染端末内部での挙動と外部への通信の両方を取得することが重要である．そして，それらのデータを互いに補い合うことで，詳細なマルウェア解析が行える．本論文では，システムコールトレーサ Alkanet を用いて取得したシステムコールトレースログと別の端末で取得した通信ログを対応付ける手法と，その基礎評価について報告する．

# Method of Connecting System Call Trace Log and Packet Capture Data to Analyze Malware

Yuki Okura† Yuto Otsuki † Yasuyuki Tanaka ‡ Shuhei Aketa †
Eiji Takimoto † Koichi Mouri †

†Ritsumeikan University
1-1-1 Nojihigashi, Kusatsu, Shiga, 525-8577, Japan
{yookura, yotuki, saketa, takimoto, mouri}@asl.cs.ritsumei.ac.jp

‡NTT Communications Corporation
Gran Park Tower 16F, 3-4-1 Shibaura, Minato-ku, Tokyo, 108-8118, Japan
yasuyuki.tanaka@ntt.com

**Abstract** Malware represented by bots communicates with remote servers to get configuration files or commands. Therefore, in the dynamic analysis, malware should be executed in the environment that malware can communicate with remote servers to observe both of behavior of an infected computer and communication. Then those data complement each other, they will achieve malware analysis successfully. In this paper, we describe a method to connecting system call trace log and packet capture data. And we also report a result of its fundamental evaluation.

# 1　はじめに

ボットに代表されるマルウェアは，設定ファイルの取得やマルウェア本体のダウンロード，コマンドの受信など，実行端末の外部と様々な通信を行う．通信に成功したマルウェアは，接続した通信先からの応答によって挙動を変化させる．しかし，外部と接続できない閉鎖環境では外部からの応答が得られないため，マルウェアは動作を停止したりするなど，本来の挙動を行わない．そのため，マルウェア本来の挙動を動的解析で解析するには，外部と通信可能な環境下でマルウェアを実行することが必要である．

既存の研究では，動的解析システムによるマルウェア実行端末内部での挙動解析 [1] と観測された通信の解析 [2] とがそれぞれ独立して行われている．しかし，片方の解析だけでは，実行端末内で発行された通信がマルウェアに起因するものであると特定することや，通信している内容が悪性なものであるのか判定することが困難である．そのため，マルウェアの挙動を詳細に解析するには，両方の解析結果を結びつけ互いに情報を補い合うことが必要である．

また，マルウェアには，コードインジェクションをすることで別のプロセスに感染を広げるといった動作をするものも存在する．このような動作をマルウェアが行ったとき，当該プロセス元来の通信とコードインジェクションに起因する通信を明確に区別することが困難である．そのため，マルウェアの通信を特定するためには，コードインジェクションされたスレッドを識別し，スレッドごとに通信の対応付けを行う必要がある．

本論文では，システムコールトレーサ Alkanet[3] を用いて取得したシステムコールトレースログと別の端末で取得した通信ログを対応付ける手法を提案する．具体的には，システムコールの引数に含まれる IP アドレスとポート番号を基に，感染端末から行われた通信をスレッド単位で分割する．また，システムコールの引数から送受信されるデータを取得することで，通信ログをパケット単位でシステムコールトレースログと対応付ける．

以下，本論文では，2 章で動的解析環境の構成と通信情報の取得方法について述べ，3 章でシステムコールトレースログと通信ログの対応付け手法について述べる．4 章で提案手法の評価について述べ，5 章で関連研究について述べる．最後に，6 章で本論文をまとめる．

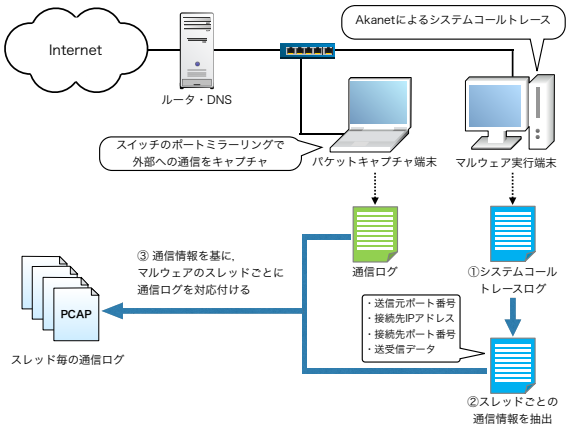


図 1: 提案手法の構成と概要

# 2　マルウェア解析情報の取得方法

## 2.1　動的解析環境の構成

提案手法では，マルウェア実行端末での挙動解析と通信データの 2 種類のログデータを取得する必要がある．そこで，提案手法では，図 1 に示す構成によって各ログデータを取得する．マルウェア実行端末 (以下，実行端末) 内でマルウェアが発行したシステムコールのログは，システムコールトレーサ Alkanet を用いて取得する (図 1 の①)．Alkanet は，スレッド単位でマルウェアを区別可能であるため正規プロセスにインジェクションされたコードによる挙動も観測可能である [3]。Alkanet が対応しているマルウェアの実行環境は，32bit 版 Windows XP SP 3 である．今回このバージョンの Windows を利用した．また，実行端末によって行われる通信は，実行端末が接続されたスイッチのポートに対してポートミラーリングを行い，パケットキャプチャ端末でキャプチャする．今回，ネットワークパケットアナライザ Wireshark[4] を利用して，パケットキャプチャを行った．

ルータとして動作する PC では，Linux 端末上で iptables と DNS サーバが動作している．iptables では，パケットフィルタリングと IP マスカレードを行っている．実行したマルウェアが

グローバルな環境へのスパムメールの送信やDDoS攻撃などを行う可能性があるため，送信先のポート番号がWell-Known Portである0番から1023番ポートについては，DNS，HTTP，HTTPS以外のポートをすべて閉じている．1024番ポート以降はすべて開放する．DNSサーバはBINDを用いており，ローカルネットワーク内からの名前解決要求のみ行う．

## 2.2 Alkanetの通信解析機能の拡張

Windows Socket API(Winsock)は，WinInetやWinHTTPなどの通信APIの下位APIとなっているため，これら通信APIを使用した場合でもWinsockの処理を必ず経由する．以上から，Winsockが発行するシステムコールを観測対象とした．Alkanetは，Winsockによって発行されるシステムコールから，スレッドが通信するときに利用したIPアドレスやポート番号を取得する機能を持つ．提案手法では，上記の機能をマルウェアが送受信したデータを取得できるように拡張した．拡張対象とするWinsock APIは，bind，connect，send，recv，sendtoの5つとした．さらに取得したIPアドレスやポート番号，ペイロードを用いてシステムコールトレースログと通信ログの対応付けを可能とした．

Winsockを用いた通信の流れを図2に示す[5]．Winsockは，AFD (Ancillary Function Driver)ドライバを利用して通信を行う．Winsockは通信時に，AFDのデバイスファイル\Device\Afd\Endpointに制御コードを送信する．制御コードを受け取ったAFDは，TCP/IP Protocol DriverにTDI IRPを送信することでソケット処理を実行し，Ethernetを通して外部へと送信を行う．

観測対象としている5つのWinsock関数は，NtDeviceIoControlFileシステムコールを用いて通信処理を実現する．NtDeviceIoControlFileシステムコールは，デバイスドライバに対して制御コードを送信することで，IPアドレスやポート番号の設定，データの送受信などを行う．詳細は次節で述べる．

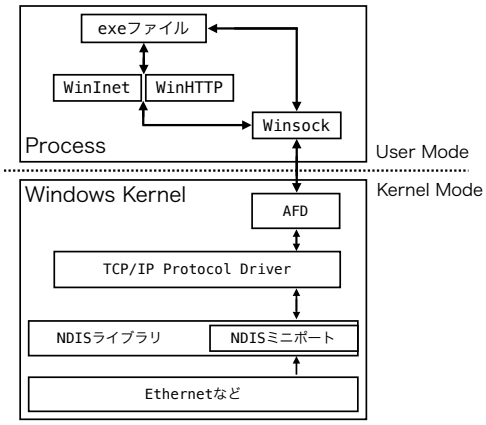


図2: Windowsの送受信処理の流れ

```
NTSTATUS NtDeviceIoControlFile(
 _In_  HANDLE            FileHandle,
 _In_  HANDLE            Event,
 _In_  PIO_APC_ROUTINE   ApcRoutine,
 _In_  PVOID             ApcContext,
 _Out_ PIO_STATUS_BLOCK  IoStatusBlock,
 _In_  ULONG             IoControlCode,
 _In_  PVOID             InputBuffer,
 _In_  ULONG             InputBufferLength,
 _Out_ PVOID             OutputBuffer,
 _In_  ULONG             OutputBufferLength
);
```

図3: NtDeviceIoControlFileの定義

## 2.3 NtDeviceIoControlFile

### 2.3.1 概要

NtDeviceIoControlFileシステムコールは，ファイルハンドルに指定されているデバイスファイルを介して，特定のデバイスドライバに制御コードを送信することで，指定したI/O制御オペレーションを実行させるシステムコールである．NtDeviceIoControlFileシステムコールのインタフェース定義を図3に示す． 第1引数には，デバイスファイルを指定する．第5引数のIoStatusBlockは，最終的な完了状態と制御に関する情報を格納している．第6引数は，第1引数で指定したデバイスファイルに対して，送信するI/O制御コード(IoControlCode)を指定する．指定したIoControlCodeの値によって，デバイスドライバの行う処理が変化する．第7引数のInputBufferは，デバイスドライバへの入力データが格納されているバッファのアドレスを示す．InputBufferが指す構造体は，IoControlCodeの値によって入力データが異なり変化する．第8引数のInputBufferLengthは，InputBufferのサイ

表 1: 提案手法で対象とする IoControlCode

| 値 | IoControlCode | 処理内容 |
|---|---|---|
| 0x12003 | IOCTL_AFD_BIND | 接続元ポート番号を設定する |
| 0x12007 | IOCTL_AFD_CONNECT | TCP の接続先 IP アドレスと接続先ポート番号を設定する |
| 0x12017 | IOCTL_AFD_RECV | TCP の受信したデータを取得する |
| 0x1201f | IOCTL_AFD_SEND | TCP の送信するデータを格納する |
| 0x12023 | IOCTL_AFD_SEND_DATAGRAM | UDP の接続先 IP アドレスと接続先ポート番号を設定する |

ズを示す．

今回，AFD に対して Winsock で発行された IoControlCode のうち，提案手法で対象とするものを表 1 で示す．表 1 の IoControlCode は，Winsock API における bind， connect，send などが呼ばれたときに引数で設定される．

### 2.3.2 IP アドレス，ポート番号の取得

IP アドレス，ポート番号は，IoControlCode の値を取得し，その値によって InputBuffer を解釈することで取得している．具体的には，下記に示す IoControlCode のとき，InputBuffer が示すアドレスが指す領域に，IP アドレスまたはポート番号が含まれている [6].

- IOCTL_AFD_BIND
- IOCTL_AFD_CONNECT
- IOCTL_AFD_SEND_DATAGRAM

IOCTL_AFD_BIND が設定されているときは，InputBuffer に接続元ポート番号が格納されている．また， IOCTL_AFD_CONNECT ，IOCTL_AFD_SEND_DATAGRAM が設定されているときは，接続先 IP アドレス，接続先ポート番号が InputBuffer に格納されている．

### 2.3.3 送受信されるデータの取得

観測対象の Winsock API では，NtDeviceIoControlFile システムコールを使って送受信するデータの受け渡しを行う．Alkanet は，IoControlCode の値を取得することで InputBuffer の構造を解釈し，送受信されるデータを取得する．表 1 のうち，送受信を行うときに使用

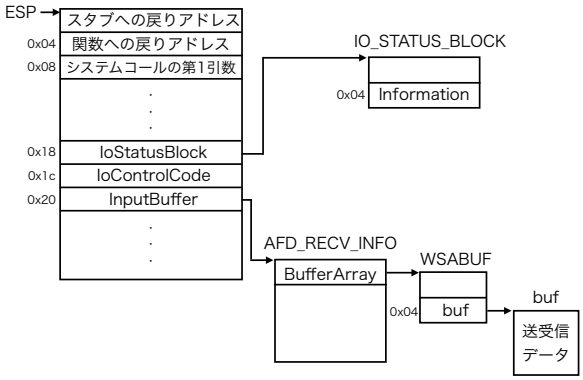


図 4: IOCTL_AFD_RECV 時の引数の構造

される IoControlCode は，IOCTL_AFD_SEND と IOCTL_AFD_RECV である．図 4 に， NtDeviceIoControlFile が IOCTL_AFD_RECV を設定していたときの引数の構造を示す．IOCTL_AFD_SEND が設定されている場合も，InputBuffer が指すアドレスの構造体名が異なるだけで構造体の要素は同じである．送受信されるデータは，InputBuffer に格納されているアドレスを図 4 で示されるようにたどることで，領域 buf から取得できる．このとき，送受信されるデータの長さは，第 5 引数の IoStatusBlock が指す領域のメンバ Information に含まれている．

## 3 提案手法

提案手法では，ログ解析ツールを拡張することで，システムコールトレースログからマルウェアのスレッドごとに以下の情報を抽出する (図 1 の②).

- 接続元ポート番号
- 接続先 IP アドレス
- 接続先ポート番号

- 送受信されたデータ

抽出したIPアドレス，ポート番号を用いて通信接続情報を作成し，通信接続情報を基に通信ログをスレッドごとに分割する．また，システムコールトレースで取得した送受信されたデータを通信ログと結びつけることで，システムコールとパケットを対応付ける (図1の③).

### 3.1 通信接続情報の作成

システムコールトレースログからマルウェアのスレッドが利用した以下の要素で構成される通信接続情報を抽出する．

- 接続元ポート番号
- 接続先IPアドレス
- 接続先ポート番号

これら要素で構成される通信接続情報は，NtDeviceIoControlFile システムコールの IoControlCode ごとに接続元・接続先のIPアドレス，ポート番号が設定され，プロトコルによりそれぞれ取得方法が異なる．

TCP 通信では，IOCTL_AFD_BIND が設定されたシステムコールトレースログと IOCTL_AFD_CONNECT が設定されたシステムコールトレースログを組み合わせることで通信接続情報を取得する．UDP による通信も同様に，IOCTL_AFD_BIND が設定されたシステムコールトレースログと IOCTL_AFD_SEND_DATAGRAM が設定されたシステムコールトレースログを組み合わせることでUDPの通信接続情報を取得する．

### 3.2 通信ログとの対応付け

マルウェアのスレッドごとに取得した通信接続情報を基に通信ログの分割を行うツールを作成した．通信ログは，1パケットごとに通信接続情報と比較することで分割する．スレッドごとの通信接続情報をパケットに含まれる接続元ポート番号と接続先IPアドレス，接続先ポート番号と比較する．比較により通信接続情報と一致したパケットをそのスレッドによって送信されたパケットと判定する．また，TCPの場合は通信先からの応答があるため，送信方向だけでなく受信方向の通信もそのスレッドに関係するパケットである．受信方向の通信として，通信接続情報に含まれる接続元ポート番号とパケットの接続先ポート番号が一致し，通信接続情報に含まれる接続先IPアドレス，接続先ポート番号とパケットに含まれる接続元IPアドレス，接続元ポート番号が一致したパケットを，そのスレッドが受信したパケットと判定する．

マルウェアのスレッドごとに分割した通信ログは，それぞれファイル名を“スレッド名 (cid: プロセスID. スレッドID).pcap”として出力する．CID(Client ID) は，プロセスIDとスレッドIDを組合わせたものである．出力された通信ログファイルは，そのCIDを持つマルウェアのスレッドによる通信のみとなる．そのため，そのマルウェアのスレッドがどのような通信をしていたのかが明らかになる．

さらに，発行されたシステムコールと通信ログのペイロードを結びつけることで，システムコールがどのパケットを送受信しているのか対応付けを行う．対応付けによって，1回のコネクションで複数のファイルが送受信された場合でも，システムコールがどの通信を行っているのか識別できる．2.3.3項で述べた方法で取得したTCPのペイロードを用いて，発行されたシステムコールと通信ログを対応付ける．取得したペイロードは，通信ログの1パケットずつと比較を行う．パケットに含まれるペイロードの中に，システムコールに含まれるペイロードが含まれていたとき対応するパケットと判定する．

## 4 評価

本章では，2.1節で述べた解析環境を用いてマルウェアの動的解析を行い，提案手法の評価を行った．本評価では，実行端末から外部へ行われる通信について，マルウェアによって行われた通信をスレッド単位で対応付けることが可能であるかを評価する．さらに，Alkanetで取得したペイロードをパケットと対応付け可能であるか評価する．マルウェアの解析時間は，300

```
[P] Dropper.exe    (pid:4dc)
  [T] Dropper.exe    (cid:4dc.51c)
    [P] leassnp.exe    (pid:668)
      [T] leassnp.exe    (cid:668.670)
    [P] wordpad.exe    (pid:680)
      [T] wordpad.exe    (cid:680.684)
```

図 5: Dropper.exe タスクツリーの一部

秒間とした．動的解析に用いたマルウェアは，MWS Datasets 2015[7] で報告されているマルウェア (Dropper.exe) と独自に収集したマルウェア (Downloader.exe) である．

## 4.1 Dropper.exe

1 つ目のマルウェアは，アンチウイルスソフトの検出名から Dropper.exe とした．解析時間中にパケットキャプチャ端末が観測した通信は，DNS(53/UDP) と HTTP(80/TCP) である．HTTP の通信内容は，POST メソッドによる文字列の送信と HTML ファイルの受信であった．

Alkanet で解析した Dropper.exe のシステムコールトレースのログから，ログ解析ツールを用いて抽出したレポートを図 5 と図 6 に示す．図 5 は，Alkanet で取得した Dropper.exe 由来のプロセス・スレッドの親子関係が表されている．先頭の大括弧内は，プロセス (P) またはスレッド (T) を表し，次にプロセス名を表す．小括弧内では，そのプロセス ID またはクライアント ID を表す．図 6 は，接続を行ったスレッドの接続元ポート番号，接続先 IP アドレス，接続先ポート番号を表す．

図 5 より，Dropper.exe によって wordpad.exe, leassnp.exe の 2 つのプロセスが生成されたことが確認できた．Dropper.exe の実行後は，wordpad.exe によって文書の表示が行われ，leassnp.exe がバックグラウンドで動作していた．図 5 に含まれる Dropper.exe 由来のスレッドのうち，通信を行ったスレッドの通信接続情報を図 6 に示す．図 6 では，leassnp.exe によって送信元ポート番号 1039 番，接続先 IP アドレス 125.*.*.*，接続先ポート番号 80 番で外部と通信を行っていることを確認できた．

図 6 の通信接続情報を用いて，通信ログとの

```
* Network
** leassnp.exe (pid:668)
*** leassnp.exe (tid:670)
    TCP Src = :1039, Dst = 125.*.*.*:80
```

図 6: Dropper.exe の通信接続情報

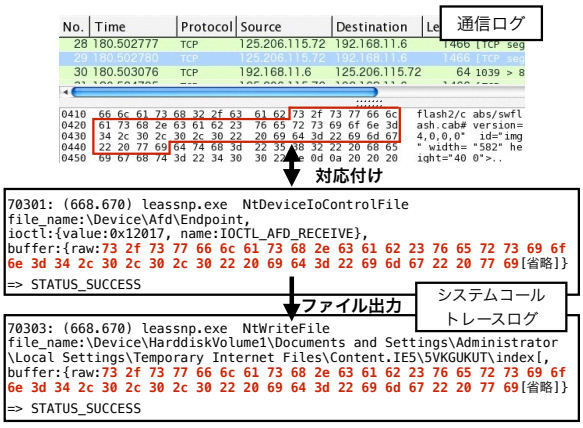


図 7: システムコールトレースログに含まれる受信データと通信ログの対応付け

対応付けを行った．その結果，DNS 以外の通信ログをすべて leassnp.exe による通信に対応付けることができた． 提案手法を用いることで，マルウェアによって生成されたプロセスが通信を行ったことが確認できた．また，受信したデータがファイルに書き出されていることをシステムコールトレースログから確認した (図 7)．分割した leassnp.exe による通信は，HTTP のみであり，通信を確立し，POST メソッドで文字列の送信，HTML ファイルの受信と接続の終了を示す HTTP の一連の通信手順である．以上から，提案手法を用いることで，マルウェアによる一連の通信手順を取得できた．

## 4.2 Downloader.exe

2 つ目のマルウェアは，アンチウイルスソフトの検出名から Downloader.exe とした．解析時間中に実行端末から行われた通信は，DNS や HTTP，SSDP(1900/UDP) が主に観測された．それ以外に観測した通信として，接続先ポート番号が 8003 番のものが存在している．しかし，ルータの iptables でフィルタリングをしていた

ため，外部と接続できなかった．

Alkanet で解析した Downloader.exe のシステムコールトレースのログから，ログ解析ツールを用いて抽出したレポートを図 8 と図 9 に示す． 図 8 では，DinoFileService や DinoFileInstall の名前を持つプロセスが生成されていることが確認できた．さらに，正規のプロセスである services.exe によって DinoFileService が起動していることを確認した． 図 8 に含まれる Downloader.exe 由来のスレッドのうち，通信を行ったスレッドの通信接続情報を図 9 に示す．図 9 では，Downloader.exe と DinoFileService によって通信が行われていることが確認できた． 図 9 の通信接続情報を基に，通信ログとの対応付けを行った．その結果，DNS 以外の通信ログをすべてマルウェア由来のスレッドに対応付けることができた．

提案手法により，実行された Downloader.exe による通信とダウンロードされた DinoFileService による通信を区別することができた．分割された Downloader.exe による通信ログを解析すると，DinoFileService.exe，DinoFileInstall.exe など 3 つのファイルがダウンロードされていることを確認した．また，システムコールトレースで受信データを取得することにより，Downloader.exe の受信データがファイルに書き出され，かつ，書き出されたファイルが DinoFileService と DinoFileInstall として実行されていることを確認した．実行された DinoFileService は，ゲートウェイ (192.168.11.1) に対して SSDP パケットを送信する挙動や，TCP のポート番号 8003 番で外部と通信を試みる挙動をそれぞれ分割された通信ログで確認できた．したがって，ダウンローダ型マルウェアによる他のマルウェアをダウンロードする通信とダウンロードされたマルウェアによる通信を分割できた．また，スレッドごとに通信を分割するため，DinoFileService のスレッドごとに通信をそれぞれ分割できた．

```
[P] Downloader.exe   (pid:288)
  [T] Downloader.exe   (cid:288.5a8)
    [T] Downloader.exe   (cid:288.fc)
    [P] DinoFileInstall  (pid:128)
      [T] DinoFileInstall  (cid:128.e0)
-----------------------------------------
[T] services.exe (cid:324.3ec) *** BENIGN
  [P] DinoFileService (pid:1cc)
    [T] DinoFileService (cid:1cc.1c8)
      [T] DinoFileService (cid:1cc.548)
        [T] DinoFileService (cid:1cc.540)
          [T] DinoFileService (cid:1cc.100)
```

図 8: Downloader.exe タスクツリーの一部

```
* Network
** Downloader.exe (pid:288)
*** Downloader.exe (tid:2dc)
    TCP Src = :1046, Dst = 127.0.0.1:1046
*** Downloader.exe (tid:fc)
    TCP Src = :1047, Dst = 211.*.*.*:80
** DinoFileService (pid:1cc)
*** DinoFileService (tid:540)
    TCP Src = :1048, Dst = 211.*.*.*:80
*** DinoFileService (tid:224)
    TCP Src = :1068, Dst = 27.*.*.*:8003
*** DinoFileService (tid:100)
    UDP Src = :1049, Dst = 192.168.11.1:1900
    UDP Src = :1050, Dst = 192.168.11.1:1900
    UDP Src = :1051, Dst = 192.168.11.1:1900
    [省略]
```

図 9: Downloader.exe の通信接続情報

### 4.3 考察

Alkanet は，Winsock を利用する通信を対象としているため，Winsock を用いない通信では通信接続情報を取得することができない．また，通信ログに含まれるパケットで対応付けができなかったプロトコルとして DNS が存在する．現在の実装では，DNS のパケットをそれぞれのスレッドに対応付けできない．これは，Windows のプロセスが名前解決する際に，LPC (Local Procedure Call) を利用してサーバプロセスに名前解決を依頼するためである [6]．サーバプロセスが DNS パケットの送信をするため，名前解決を要求したスレッドによる通信と判定できない．名前解決を要求したスレッドに DNS パケットを対応付けるためには，サーバプロセスとの通信を解析する必要がある．

## 5 関連研究

三村 [8] らは，Windows のカーネルモードドライバによって，通信を行うプロセスを特定する手法を提案している．ドライバでは，観測したすべてのプロセスについて，起動・終了・モジュールの読み込み・通信試行の 4 項目を記録

する．記録したログを不正な通信を検知した際に利用することで，不正な通信を行ったプロセスを特定・解析することができる．

神薗 [9] らは，標的型攻撃対策として，Windows API フックを用いることで，プロセスの一連の通信手続きを保全するフォレンジック手法を提案している．神薗らのシステムで出力されたログは，プロセス情報と通信内容の 2 つの情報を持っている．ホストベース監視製品でアラートが発生したときに，ログのプロセス情報を突合することでどのような通信が行われていたかを判別することができる．また，ネットワーク監視製品でアラートが発生したとき，ログの通信内容と突合することでどのプロセスによって通信が行われていたかを特定することができる．

我々の提案手法は，マルウェア解析システムを目的としているため，スレッド単位でマルウェアの通信を対応付けるなど粒度の細かいデータを取得できる．そのため，コードインジェクションされた正規プロセスが通信を行ったとき，マルウェアに起因する通信であるかを特定できる．

## 6 おわりに

本論文では，システムコールトレースによって取得した引数に含まれる IP アドレスやポート番号などにより，システムコールトレースログと通信ログを対応付ける手法について述べた．提案手法により，感染端末内部での挙動ログと通信ログを対応付けることが可能であることを確認した．今後の課題として，文献 [10, 11] で述べられているテイント解析技術による感染端末内部での挙動と通信の結びつけを，スタックトレースや Branch Trace Store を用いることで軽量かつ同等の情報を得ることができないか検討，調査する．また，我々は現在 Alkanet の Windows 7，10 への対応を進めている．提案手法を適用する場合，Windows Vista 以降では TCP/IP スタックのアーキテクチャが変更されているため調査する必要がある．

## 参考文献


[1] Ulrich Bayer, Christopher Kruegel, and Engin Kirda. TTAnalyze: A Tool for Analyzing Malware. In *15th European Institute for Computer Antivirus Research (EICAR 2006) Annual Conference*, 4 2006.

[2] Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. Botminer: Clustering Analysis of Network Traffic for Protocol- and Structure-independent Botnet Detection. In *Proceedings of the 17th Conference on Security Symposium*, SS'08, pp. 139–154. USENIX Association, 2008.

[3] 大月勇人, 瀧本栄二, 齋藤彰一, 毛利公一. 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. 情報処理学会論文誌, Vol. 55, No. 9, pp. 2034–2046, sep 2014.

[4] Wireshark. Wireshark. https://www.wireshark.org/.

[5] David Solomon and Mark Russinovich. インサイド Microsoft Windows 第 4 版 下, pp. 365–428. 日経 BP ソフトプレス, 2008.

[6] Helmut Petritshch. *Network Virtualisation*, pp. 39–51. VDM Verlag Dr. Mueller e.K, 2008.

[7] 神薗雅紀, 秋山満昭, 笠間貴弘, 村上純一, 畑田充弘, 寺田真敏. マルウェア対策のための研究用データセット〜mws datasets 2015〜. 研究報告コンピュータセキュリティ(CSEC), Vol. 2015-CSEC-70, pp. 1–8, jun 2015.

[8] 三村聡志, 佐々木良一. プロセス情報と関連づけたパケットを利用した不正通信原因推定手法の提案. マルチメディア、分散協調とモバイルシンポジウム 2014 論文集, 第 2014 巻, pp. 1973–1980, jul 2014.

[9] 神薗雅紀, 遠峰隆史, 衛藤侑, 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, October 2014.

[10] G. Jacob, R. Hund, C. Kruegel, and T. Holz. JACKSTRAWS: Picking Command and Control Connections from Bot Traffic. In *USENIX Security Symposium*, SEC'11, pp. 29–29. USENIX Association, 2011.

[11] 幾世知範, 青木一史, 八木毅, 針生剛男. 通信先と端末内の挙動との依存関係に基づくマルウェアダウンロードサイト特定手法. Vol. 2014, No. 2, pp. 1134–1141, oct 2014.