Vol.2020-SPT-36 No.25
2020/3/3

マルウェアの動的解析における ログ出力が停止する現象の実態調査

森本 康太[1,a)]　鄭 俊俊[1]　瀧本 栄二[1]　齋藤 彰一[2]　毛利 公一[1]

概要：システムコールや API 呼出し等を観測するマルウェアの動的解析システムでは，観測ログへの出力が観測途中で停止したように見える場合がある．これらの原因については十分な知見が得られておらず，観測を継続したとしても有効な観測ログを得られない場合，結果的に観測時間の浪費につながる．特にマルウェアが典型的に使用する API やシステムコールに限ってログを取得する動的解析システムでは，観測終了の判断がより困難となる．この実態を明らかにするため，マルウェア検体を対象に API の呼出しとシステムコールの発行を観測した．その結果，マルウェアによる特定のシステムコールの発行が観測ログの出力に影響を与えることが明らかとなった．さらに，システムコールや API の観測ログからこれらを把握する方法について検討した．

Understanding Malware's Actual Behaviors from Dynamic System Call Tracer's Log

1. はじめに

近年，マルウェアによる脅威が問題となっている．マルウェア対策のためには，マルウェアの解析を行い，挙動を明らかにする必要がある．マルウェアの挙動を短時間で把握することが求められる状況において，マルウェアを動作させて挙動を調査する動的解析手法が有効とされる．一方，システムコールや API 呼出し等を観測するような動的解析システムでは，観測ログへの出力が観測途中で停止したように見える場合がある．動的解析において観測を継続したとしても有効な観測ログを得られない場合，結果的に観測時間の浪費につながる．

動的解析におけるログ出力の停止とマルウェアの挙動の対応関係については十分に知見が得られておらず，明らかになっていない．特に，解析の高速性を考慮して一部のシステムコールや API 呼出しのみを観測する Cuckoo Sandbox[1] のような動的解析システムの場合，ログ出力の停止に対応するマルウェアの動作を把握することが困難となり，観測終了の判断を行うのが難しい場合がある．

上記のとおり，システムコールや API 呼出し等を観測するような既存の動的解析システムでは，ログ出力の停止に対して，適切な観測時間で観測を終了できているとはいえない．動的解析において，観測ログへの出力が停止するケースに対応するためには，これらの実態を明らかにし，動的解析システムにおいて検出を行う必要がある．

動的解析においてログ出力が停止する現象を検出するためには，ログ出力が停止する現象と得られた観測ログの関係を明らかにする必要がある．そこで，マルウェア検体を対象に動的解析を行い，API の呼出しとシステムコールの発行を観測した．その結果，マルウェアによる特定のシステムコールの発行が観測ログの出力に影響を与えることが明らかとなった．さらに，それらの挙動をシステムコールログから検出する方法を検討した．

本論文では，2 章で解析環境について述べ，3 章で動的解析においてログが取得できないマルウェアの実態調査について述べる．4 章で考察について述べ，5 章で関連研究について述べる．最後に 6 章でまとめる．

2. 解析環境

本論文では，API 呼出しやシステムコールの発行を観測

[1] 立命館大学
Ritsumeikan University
[2] 名古屋工業大学
Nagoya Institute of Technology
[a)] kmorimoto@asl.cs.ritsumei.ac.jp

する動的解析に注目するため，調査に利用する動的解析システムは，API 呼出しやシステムコール発行を観測できることが望ましい．また，動的解析では観測対象外の挙動は，把握できないため，観測対象外の挙動を見落とす可能性がある．したがって，マルウェアが利用する API やシステムコールを幅広く観測できる必要がある．

本論文の調査では，ログ出力が停止する現象と得られた観測ログの関係を明らかにするため，マルウェア検体に対してシステムコールトレーサ Alkanet[2] を利用して動的解析を行う．また，多数の検体に対して観測ログを取得するため，自動観測環境を用いた．

本章では，マルウェア検体が発行したシステムコールの発行を観測するシステムコールトレーサ Alkanet について述べる．

2.1 Alkanet

Alkanet は，マルウェア観測用端末とロギング用端末の 2 台の端末を用いてマルウェア解析を行う．Alkanet[2] は，VMM である BitVisor の拡張機能として実装されたシステムコールトレーサで，ゲスト OS である Windows 上で動作するプロセスが発行したシステムコールをトレースする．具体的には，システムコール処理の前後にハードウェアブレイクポイントを設置し，処理をフックすることでその種類や引数などを記録する．Alkanet は，ゲスト OS で発行された全てのシステムコールをトレース可能である．システムコールや API 呼出し等を観測するマルウェアの動的解析では，マルウェアが典型的に使用する API 呼出しに限ってログを取得する場合が多い．しかし，このような解析環境では，観測対象外の挙動を見落とす可能性がある．そこで，本論文の調査では，マルウェアの挙動をより正確に把握するため，すべてのシステムコールを記録する．

2.2 観測ログ

Alkanet のシステムコールトレースログの構成要素を表 1 に示す．Alkanet は，システムコール番号とシステムコール発行元プロセスやスレッドの情報に加えて，引数を解析した結果などの付加情報を取得する．ただし，引数と戻り値を正確に取得するため，Alkanet は，一つのシステムコールの発行に対して，システムコール処理の前後で二つのログが出力される．さらに，システムコール記録時にスタックトレースを行うことでシステムコールが発行されるまでのコントロールフローを関数単位で取得する [3]．スタックトレースでは，システムコールが発行されるまでにユーザスタックに格納された関数単位の戻りアドレスについて以下の情報を取得する．

API マップされていれているファイルのエクスポートテーブルとシンボル情報から得たリターンアドレスに対応する API 名とその API の先頭アドレスからのオフセットを示す．シンボル情報が存在しない場合，”-” を出力する．

Writeble リターンアドレスを含むページが書き込み可か否かを出力する．この値は PTE(ページテーブルエントリ) から取得する．

Dirty リターンアドレスを含むページが書き換えられた可か否かを出力する．この値は Writable と同様に PTE から取得する．

VAD プロセスのメモリ領域を管理するデータ構造の VAD(Virtual Address Descriptor) から取得した情報を示す ．VAD が保持する情報は，管理するアドレスの範囲や，その範囲のデフォルトのページ保護属性，ファイルマップの有無，マップされているファイルの情報などがある． VAD を参照することで，リターンアドレスを含むメモリ領域にマップされているファイルの情報を得ることができる．

表 1 システムコールトレースログの構成
Table 1 System call trace log configuration.

構成要素	概要
No.	ログの通し番号
Time	記録を取った時点での CPU 時間
Cid	システムコールを発行したスレッドの Cid
Name	プロセスの実行ファイル名
Type	sysenter 時のログか sysexit 時のログかを示す
Ret	戻り値 (sysexit のログのみ)
SNo.	システムコールの番号とその名前
Note	引数を解析した結果などの付加情報
StackTrace	スタックトレース情報

Alkanet による動的解析を行うことで，すべてのシステムコールおよびシステムコールが発行されるまでに呼び出された API を観測することができる．ただし，システムコールの発行は複数の API を経由して行われるため，マルウェアプロセスが利用した API をすべて調査するのはコストがかかる．そこで，観測ログの解析を行うことで，システムコールの発行までに呼び出した API 群からマルウェアのコードが直接呼び出した API のみを抽出する．これにより，システムコールレベルの挙動に加えて API レベルの挙動も効率的に把握することが可能となる．

2.3 観測ログの解析

ユーザプロセスがプロセス生成やメモリ操作などを行う場合，特定のシステムコールが発行される．そこで，表 2 に示すシステムコールの引数を解析することでマルウェアの挙動を追跡する．最初に起動されたマルウェアプロセスによるプロセスやスレッドの生成を分析することで，最初のプロセスをルートとしたプロセス，スレッドの派生を把握できる．マルウェアが作成や書き込みを行ったファイルがプロセスとして起動された場合や，実行権限付きで仮想アドレス空間にマップされた場合などにも追跡を行うこと

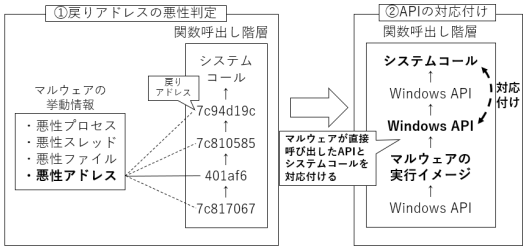

図 1 マルウェアコードが呼び出した API の特定
Fig. 1 Identify APIs called by malware code.

表 2 解析対象のシステムコール
Table 2 System call to be analyzed.

システムコール	機能
NtCreateProcessEx	プロセス生成
NtCreateProcess	プロセス生成
NtCreateThread	スレッド生成
NtWriteFile	ファイル書込み
NtWriteVirtualMemory	メモリ書込み
NtMapViewOfSection	ファイルマップ
NtAllocateVirtualMemory	メモリ割当て
NtProtectVirtualMemory	メモリ保護

で，他のプロセスの操作によって，マルウェアが作成したファイルを一般のプロセスが起動したり，DLL としてロードするなどの挙動の追跡が可能となる．

さらに，図 1 に示すとおり，スタックトレースで得られた戻りアドレスと悪性なメモリ領域を照合することで，マルウェアの実行コードおよびマルウェアが直接呼び出した API を特定することが可能となる．悪性なメモリ領域の判定には，システムコールによって行われたファイルマップとメモリ書込みに加えて，ページテーブルエントリから取得した Dirty フラグを用いる．Dirty ビットが立っている領域は，実行時に確保された領域かつ実行された領域であるため，システムコール発行を必要としないコード展開であっても悪性な戻りアドレスを判定することが可能である．

ログアナライザは，システムコール発行時の戻りアドレスからマルウェアの実行コードおよびマルウェアコードが直接呼び出した API を特定し，1 ログエントリに対して API 名とシステムコール名のログを出力する．ログの構成要素は表 3 に示すとおりである．

本論文の調査では，マルウェアのスレッド単位の挙動に注目し，API やシステムコールのログをマルウェアのスレッド単位で分割する．さらに，最後に観測された API とシステムコールのログを抽出して，タイムスタンプからログ出力が停止した時間を算出する．

2.4 自動観測システム

多数の検体に対して観測ログを取得するため，観測環境には，図 2 に示す自動観測環境を用いた．観測システムは，マルウェア検体の実行からロギング作業を自動化する．観測システムは，Alkanet を構成する 2 端末および外部との通信を制限するファイアウォール端末から構成される．ファイアウォール端末には，ハニーポットの通信を制限することを目的とした CentOS ベースのシステムである Honeywall[5] を利用した．自動観測システムは，検体の実行からロギング作業までを自動で行う．

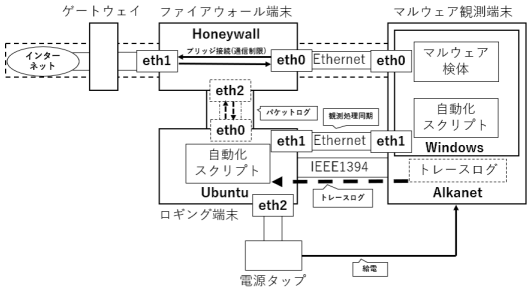

図 2 解析環境の構成
Fig. 2 Configuration of analysis environment.

本論文の調査では，自動観測システムで得られた Alkanet の観測ログを解析し，ログの出力が停止したスレッドを特定する．

3. 動的解析ログが記録されないマルウェアの実態調査

本調査では，Alkanet を用いて，マルウェア検体の動的解析を行い，動的解析ログの出力が停止する現象を調査する．本章では，調査内容と調査結果について述べる．

3.1 調査目的

本調査の目的は，マルウェアの動的解析におけるより効率的な観測終了方法を実現するため，動的解析ログの出力が停止する現象を明らかにすることである．具体的には，マルウェアの挙動情報を表す API やシステムコールのログと動的解析ログの出力が停止する現象の関係を明らかにする．本論文では，本調査の結果に基づいて，システムコールや API の観測ログからこれらを把握する方法について検討する．

3.2 調査対象と調査方法

動的解析ログの出力が停止する現象について以下の手順で調査を行った．

手順 (1) データセットに含まれる各検体を Alkanet で 5 分間検体のトレースを行い，スレッド単位でシステムコールログおよび API ログを取得する．

手順 (2) 観測ログの出力が停止したスレッドのログを抽出し，最終ログからログ出力の停止時間を算出する．

手順 (3) 最終ログで観測されたシステムコールの発行元 API についてログ出力の停止時間を算出する．

表 3　API とシステムコールのログ構成
Table 3　API and System call log configuration.

構成要素	概要
ログ番号	ログの通し番号
悪性フラグ	ログがマルウェアに起因するのものは 1，そうでないものは 0
プロセス名	システムコール発行元のプロセス名
CID	システムコールを発行したスレッドの PID と TID
実行イメージ	API 呼出し元の戻りアドレスに対応する実行ファイル名
API	マルウェアコードが直接呼び出した API
システムコール	システムコール名

```
No .: 71553
Time : 89371156
Type : sysenter
SNo .: 102 (NtTerminateThread)
Cid : aa4 . aa8
Name :0b45b257e701be9
Note :
 proc_image_name : 0b45b257e701be9
 proc_pid : 0xaa4
 proc_tid : 0xaa8
```

図 3　NtTerminateThread システムコールのログ
Fig. 3　System calls related to thread wait status.

```
No . : 649936
Time : 331931034
Type : sysenter
SNo .: 3b (NtDelayExecution)
Cid : f28.f2c
Name : 1012f67a2bb3574
Note :
 alertable : false
interval :
 meaning : wait for 180000ms.
```

図 4　NtDelayExecution システムコールのログ
Fig. 4　NtDelayExecution System call log

手順 (1) では，マルウェアの実行環境のゲスト OS は，32bit の Windows XP Service Pack 3 とし，MWS データセット [4] に記録された EXE 形式のマルウェア 343 検体に対して 5 分間観測を行う．観測環境には，2.4 節で述べた自動観測環境を用いる．

手順 (2) では，観測で取得したログの解析を行い，マルウェアのログをスレッド単位で抽出する．また，各スレッドにおける最終ログのタイムスタンプから観測終了までのログ出力停止時間を算出し，各システムコールの発行と観測ログの出力が停止する現象の関係について調査を行う．手順 (3) では，API の呼出しとログ出力の停止との関係を明らかにするため，最終ログで観測されたシステムコールの発行元 API についてもログ数の集計を行う．

3.3　調査結果

Alkanet を用いて，MWS データセットで報告されている EXE 形式の 343 検体に対して動的解析を行った．本節では，手順 (2)，(3) の調査結果について述べる．

3.3.1　手順 (2) の結果

マルウェアスレッドにおける最終ログからのログ出力停止の平均時間が 1 秒以上のシステムコールを表 4 に示す．最終ログとして観測された各システムコールのログ数から，特定のシステムコールがスレッドの最終ログとして観測され，ログの出力が 10 秒以上停止したことが分かる．特に，スレッドの終了やスレッドの待ち状態に関するシステムコールが多数観測された．

図 4 は，スレッドの動作を終了し，スレッドオブジェクトを解放する NtTerminateThread システムコールが最後に観測されたログである．図 4 は，指定した時間が経過するまでスレッドの動作を停止させる NtDelayExecution システムコールが最後に観測されたログである．これらは，ログタイプの項目が sysenter であることから，システムコール処理前に設置されたフックポイントの出力ログである．また，これらのシステムコール処理が完了したことを表すログが観測されていないことを表す．したがって，スレッドは，観測終了時にカーネルから制御が戻っていない状態であるといえる．上記の通り，これらのスレッドは，ユーザモードにおけるコード実行が停止し，以降のシステムコール発行が行われないため，システムコール発行に関するログが観測されないと考えられる．

同様の挙動が得られたシステムコールの機能を表 5，表 6 に示す．表 5 は，スレッドの終了に関するシステムコールを表す．表 6 は，スレッドの待ち状態に関するシステムコールを表す．

一方，スレッドの終了やスレッドの待ち状態に関係しないシステムコールがスレッドの最終ログであってもログの出力が停止するケースが存在する．マルウェアのタスクツリーから，マルチスレッドで動作するプロセスの情報を抽出したものを図 5 に示す．図 5 は，プロセス ID が 0xc40 のプロセスにスレッド ID が 0xc44 とスレッド ID が 0xc64 の二つのスレッドが動作していたことを表す．また，スレッド ID が

表 4　各システムコールのログ出力停止時間の集計結果
Table 4　Aggregated result of log output suspension time for each system call.

システムコール	スレッド数	ログ出力停止の平均時間 (秒)
NtTerminateThread	3701	127
NtDelayExecution	651	42
NtWaitForSingleObject	394	76
NtTerminateProcess	336	175
NtWaitForMultipleObjects	224	168
NtRemoveIoCompletion	118	100
NtReplyWaitReceivePortEx	87	91
NtUserGetMessage	78	130
NtUserWaitMessage	51	119
NtSignalAndWaitForSingleObject	27	265
NtProtectVirtualMemory	22	10
NtClose	13	15
NtRaiseException	10	111
NtReadFile	10	44
NtSetValueKey	7	4
NtFsControlFile	6	121
NtUnmapViewOfSection	5	42
NtCreateEvent	4	89
NtWaitHighEventPair	4	253
NtSetInformationThread	3	3
NtUserMessageCall	3	3
NtGetPlugPlayEvent	2	299
NtCreateSection	2	136
NtUserWaitForInputIdle	1	2
NtRaiseHardError	1	212
NtNotifyChangeKey	1	255
NtUserCallNoParam	1	157

```
[P]pid:c40    status:1 cmd.exe    [72858]Process Generated by malicious syscall of thread:35c.364 [Lastlog]:[88008]NtTerminateProcess
[T]cid:c40.c44 status:1           [75000]Thread Generated by malicious syscall of thread:35c.364 [Lastlog]:[88008]NtTerminateProcess
[T]cid:c40.c64 status:1           [76173]Thread Generated by malicious syscall of thread: 8c.aa0 [Lastlog]:[87168]NtProtectVirtualMemory
```

図 5　マルウェアのスレッド情報
Fig. 5　Malware thread information.

表 5　スレッドの終了に関係するシステムコール
Table 5　System calls related to thread termination.

システムコール	機能
NtTerminateThread	スレッドの終了
NtTerminateProcess	プロセスの終了

0xc44 のスレッドの最終ログは，NtProtectVirtualMemory システムコールのものである．ログエントリのログタイプは，sysenter であった．NtProtectVirtualMemory システムコールは，前述したとおり，メモリ保護を行うシステムコールであり，スレッドの動作が長時間ブロッキングされることはない．一方，スレッド ID が 0xc44 のスレッドの最終ログとして NtTerminateProcesss システムコールが観測されたことを表す．NtTerminateProcess システムコールは，プロセスオブジェクトを開放するシステムコールであり，引数に指定されたプロセスが終了したことを表す．したがって，スレッド ID が 0xc44 のスレッドは，NtProtectVirtualMemory システムコールの処理中にプロセスの終了が行われたために，システムコールの処理が完了しないまま，スレッドの動作が終了したと考えられる．このように，マルチスレッドで動作するプロセスでは，プロセス終了のシステムコールが発行された場合には，プロセス終了やスレッド終了に関係しないシステムコールがスレッドの最終ログとして観測されることがある．

3.3.2　手順 (3) の結果

表 7 は，NtDelayExecution システムコールの発行元 API についてマルウェアスレッドで観測されたすべてのログ総数と最終ログで観測されたログの総数を表す．最終ログで観測されなかった API については一部省略する．NtDelayExecution システムコールの発行元 API については，Sleep API の呼出しによるシステムコール発行が最も多く，

表 6　スレッドの待ち状態に関係するシステムコール
Table 6　System calls related to thread wait status.

システムコール	機能
NtDelayExecution	スリープ
NtWaitForSingleObject	同期
NtWaitForMultipleObjects	同期
NtRemoveIoCompletion	I/O 操作
NtReplyWaitReceivePortEx	Local Procedure Call
NtUserGetMessage	メッセージ受信
NtUserWaitMessage	メッセージ受信
NtSignalAndWaitForSingleObject	同期
NtRaiseException	例外通知
NtReadFile	名前付きパイプ
NtFsControlFile	名前付きパイプ
NtWaitHighEventPair	同期
NtGetPlugPlayEvent	プラグアンドプレイ
NtUserWaitForInputIdle	プロセスの初期化待ち
NtRaiseHardError	エラー通知
NtNotifyChangeKey	レジストリキーの変更待ち

最終ログにおける割合も高い．一方，rtcDoEvents API や TransformMD5 API などの API については，NtDelayExecution システムコールの発行を行っているにも関わらず最終ログとして観測されないものが存在した．スレッドの動作を停止させる他のシステムコールについても，最終ログとして観測される API と観測されない API が存在した．最終ログとして観測された API によるシステムコールの処理時間は比較的長いと考えられる．そのため，主に一部の API 呼出しによるシステムコール発行がログ出力の停止に影響しているといえる．

4. 考察

調査において，特定のシステムコールの発行によるプロセスの終了やスレッドの中断がログ出力が停止する主な原因であることが明らかとなった．本章では，調査結果の考察について述べる．

5. プロセス終了とスレッド終了

プロセス終了やスレッド終了については，それらに関する観測ログの出力が完全に停止するため，観測を継続しても観測ログが取得できない可能性が高い．ただし，一方，プロセスの終了やスレッドの中断が行われたとしても，OS の自動実行機能を用いたプロセスの起動やスレッドの再開が行われる可能性がある．プロセス終了やスレッド終了の原因については，命令レベルの挙動情報が必要であるため，システムコールや API のログのみでは判断が難しい．一方，プロセス終了に関するシステムコールの発行を行ったスレッドは，プロセス内におけるプロセスの終了を制御するスレッドであるといえる．したがって，当該スレッドの挙動に注目することでプロセス終了の原因を把握できる可能性がある．

6. スレッドの中断

スレッドの中断については，同期，メッセージ待ち，スリープなどのシステムコールが観測された．特に，同期やメッセージ受信などの悪性な挙動に直接関与しないものも観測された．このような API やシステムコールは，解析の高速性を考慮して一部のシステムコールや API 呼出しのみを観測する環境では観測の対象外となりやすい．

スリープや同期に関するシステムコールについては，スレッドを待ち状態にするシステムコールと同等の機能をもつ API をマルウェアが呼び出した場合，スレッドの待機処理が抽象化されている API よりも待ち時間が長い傾向が得られた．これは，スレッドの待機処理が抽象化されている場合，API 内部で最低限のタイムアウト時間が設定されており，マルウェアのコードは，スレッドの待機処理を直接制御できないためと考えられる．また，これらのシステムコールは，マルウェアのループ処理内で行われるものが観測された．このような場合，API ログとシステムコールログにおいて同様の記録が繰り返されるため，個別の処理における待機時間が短時間であっても，処理全体における待機時間が大きくなる場合がある．Alkanet のログエントリには実行命令のアドレスを表すリターンアドレスが付加されているため，同じリターンアドレスを含むログから，これらを検出することができる．このように，API ログやシステムコールログに実行命令のアドレス情報が付加されていれば，ループ処理を検出することが可能である．スレッドの待ち時間は，引数に待機時間が指定されるシステムコールを除いて，制御が戻る前に待ち時間を把握するのは困難である．

メッセージ受信に関するシステムコールでは，メッセージループの処理でログの出力が停止する検体が多数観測された．また，受信されたウィンドウメッセージについても，一部のメッセージしか観測されなかった．これは，マウス操作やキーボード操作を行わない観測環境で検体の実行を行ったためと考えられる．これらの検体について，イベントに対応する処理を把握するためには，GUI 操作をエミュレートする動的解析やメッセージを処理するメッセージハンドラの解析を行う必要がある．

7. ログ出力停止の検出

ログ出力が停止する挙動に動的解析システムで対応するためには，まず該当する挙動を検出することが求められる．特に，高速性を考慮して一部のシステムコールや API 呼出しのみを観測するような動的解析システムの場合，フック対象の API やシステムコールを検討する必要がある．調査結果から，これらのシステムコールの発行元の API は，複

表 7　NtDelayExecution システムコールの発行元 API
Table 7　Caller API of NtDelayExecution System call.

システムコール発行元 API	ログの総数	最終ログ数
Sleep	321433	442
SleepEx	10321	10
NtDelayExecution	1234	2
rtcDoEvents	13444	0
TransformMD5	2499	0
ThunRTMain	1768	0
UpdateDriverForPlugAndPlayDevicesA	18	0
CoUninitialize	14	0
SetupDiCallClassInstaller	5	0
connect	5	0
GetAdaptersAddresses	2	0

表 8　最終ログで観測された API
Table 8　API observed in the last log.

システムコール	システムコールの発行元 API
NtTerminateThread	ExitThread，RtlExitUserThread，TerminateThread
NtDelayExecution	Sleep，SleepEx，NtDelayExecution
NtWaitForSingleObject	HttpSendRequestA，WSAConnect， BasepMmHighMemoryConditionEventName， connect，_seh_longjmp_unwind， RtlEnterCriticalSection， recvfrom，select，WaitForSingleObject， LoadLibraryA，RtlEnterCriticalSection，recv
NtTerminateProcess	ExitProcess，TerminateProcess，exit
NtWaitForMultipleObjects	BasepMmHighMemoryConditionEventName， MsgWaitForMultipleObjects， CreateFileMappingA，WaitForMultipleObjects
NtRemoveIoCompletion	SockAsyncThread，GetQueuedCompletionStatus
NtReplyWaitReceivePortEx	BaseThreadStart
NtUserGetMessage	GetMessageA，GetMessageW
NtUserWaitMessage	MessageBoxW，DialogBoxIndirectParamA， BasepMmHighMemoryConditionEventName， MessageBoxA，DialogBoxParamW
NtSignalAndWaitForSingleObject	SignalObjectAndWait
NtRaiseException	RtlRaiseException，KiUserExceptionDispatcher
NtReadFile	ReadFile，StartServiceCtrlDispatcherA
NtFsControlFile	ConnectNamedPipe
NtWaitHighEventPair	NtWaitHighEventPair
NtGetPlugPlayEvent	PnpEventThread
NtUserWaitForInputIdle	WaitForInputIdle
NtRaiseHardError	UnhandledExceptionFilter
NtNotifyChangeKey	RegNotifyChangeKeyValue

数存在するため，すべてフックするのはコストがかかる．そのため，該当するマルウェアの挙動の検出は，システムコールレイヤにおける観測が適切であるといえる．API フックによる検出を行う場合は，ログが多数観測されたかつ待機時間が長い API をフックすることで，効率的にログ出力の停止を検出できる．一方，ログ出力が停止する挙動に対して観測終了の判断を行うためには，これらの挙動に対して適切に観測終了を行う条件を設定する必要がある．特に，スレッドの中断は，観測を継続することで動作が再開する可能性があるため，観測を終了する判断が難しい．今後の課題として，該当するシステムコールの引数やログ出力パターンなどの調査を行う必要がある．

8. 関連研究

青木らの研究 [6] では， マルウェアの解析中に実行されたコード量と解析時間の関係性を明らかにすることで，動的解析を行う際に設定すべき解析時間を検討している．当該研究では，マルウェアの動的解析を行い，実行されたコード量と解析時間の関係性を調査しており，2 分程度マルウェアを実行すれば, 30 分間で動作して得られた結果の約 90%が得られることが示されている． ただし，本論文で対象とするような，観測ログの出力が停止する原因を明らかにするものではない．

大山の研究 [7] では，マルウェアの挙動に与える影響をできるだけ小さくしながら，スリープのスキップやスリープ時間の短縮を実現する方式が提案されている．マルウェアがうまく動作しない原因の一つについてそれを回避しようとする試みであるが，本論文は，スレッドの動作の中断などの解明を広く試みるものである．

仲小路らの研究 [8] では，マルウェアを多種類の解析環境で同時並列的に実行させる多種環境マルウェア動的解析システムを提案している．仲小路らの手法では，既存の動的解析ツールを活用して複数の解析エンジン，異なる環境のサンドボックス群上でマルウェアをマルウェアを同時並列で実行してその挙動を観測，挙動解明，動作環境を分析する．複数のサンドボックスを用いることにより，プラットフォームや OS，アプリケーション環境を選ぶマルウェアであっても，用意されたいずれかの環境で本来の挙動を観測できる確率が高まる．また，多種類の解析環境から得られた挙動からマルウェアが動作する条件の推定が可能となる．本論文の調査は，API トレースやシステムコールトレースにおいて観測ログへの出力が停止する原因を調査するものであるため，システムコールおよび API を対象とした動的解析によってマルウェア検体の動作を観測した．本論文の実態調査では，プロセスの終了やスレッドの待ち状態がシステムコールや API 呼出し等を観測する動的解析のログに影響を与えることを明らかにした．

9. おわりに

本論文では，API トレースやシステムコールトレースにおいてログ出力が停止するマルウェアの具体的な動作内容を調査するため，MWS Dataset で報告されているマルウェア検体に対してシステムコールトレースログの調査を行った．その結果，以下の事実が明らかとなった．

- マルウェアのプロセスが終了する場合，NtTerminateProcess システムコールや NtTerminateThread システムコールが発行されていた．
- マルウェアのプロセスが終了していないが，ログの出力が停止する場合，NtDelayExecution システムコールや NtUserGetMessage システムコールなどのスレッドを何らかの待ち状態にするシステムコールが発行されていた．
- マルウェアがプロセスの終了やスレッドの待ち状態に関するシステムコールを発行した場合，以降のログが取得できない場合がある．
- システムコールの発行によるログ出力の停止時間は，発行元 API によって差異がみられた．

以上のことから，マルウェアのプロセスが終了する場合やスレッドの動作が中断する場合には，特定のシステムコールの発行が行われることを明らかにした．また，それらが動的解析における観測ログへの出力が停止する原因であることを明らかにした．さらに，システムコールの発行によるログ出力の停止時間は，発行元 API によって異なることを示した．

参考文献

[1] Foundation, C.: Automated Malware Analysis - Cuckoo Sandbox (2017). https://cuckoosandbox.org/.
[2] 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 (2014).
[3] 大月勇人，瀧本栄二，齋藤彰一，毛利公一：Alkanet におけるシステムコールの呼出し元動的リンクライブラリの特定手法，コンピュータセキュリティシンポジウム 2013 論文集，Vol. 2013,No. 4, pp. 753–760 (2013).
[4] 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〜, 情報処理学会研究報告, Vol. 2015-CSEC-70, No. 6, pp. 1–8(2015).
[5] Honeynet.org: Know Your Enemy:GenII Honeynets . http://old.honeynet.org/papers/gen2/(2018).
[6] 青木一史，川古谷裕平，岩村誠，伊藤光恭：動的解析における検体動作時間に関する検討，コンピュータセキュリティシンポジウム 2010 論文集, Vol. 2010, No. 9, pp. 543–548 (2010).
[7] 大山恵弘：動的マルウェア解析においてスリープ時間を短縮する方式，コンピュータセキュリティシンポジウム 2017 論文集，pp. 487–494 (2017).
[8] 仲小路博史，重本倫宏，鬼頭哲郎，林直樹，寺田真敏，菊池浩明：多種環境マルウェア動的解析システムの提案および評価，情報処理学会論文誌，Vol. 56, No. 9, pp. 1730–1744 (2015).