Computer Security Symposium 2018
22 - 25 October 2018

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

森本 康太[1,a)]　鄭 俊俊[1]　齋藤 彰一[2]　瀧本 栄二[1]　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においては，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．このような場合にマルウェアのプロセスが早々に終了したり，終了はしていないが有効なログが取得できないなど，「うまく動作しない」マルウェアの存在が確認されている．それらは，マルウェアとしての動作をしたのか否かを判断するのが困難であり，動的解析を適切に終了できているとは言えない．この実態を明らかにすべく，マルウェア検体を対象に全システムコールの発行を観測した．その結果，プロセスの終了やスレッドの待ち状態が動的解析ログに影響を与えることが明らかとなった．

キーワード：MWS，マルウェア，動的解析，システムコールトレース

A Survey on Malware that Log Can not be Acquired in Dynamic Analysis

Kota Morimoto[1,a)]　Junjun Zheng[1]　Shoichi Saito[2]　Eiji Takimoto[1]　Kota Morimoto[1]

Abstract: Dynamic analysis of malwares can collect runtime execution information such as system calls and API calls by tracing the process execution and logging the system calls and API calls made by malware specimens. Using the log of dynamic analysis, we can analyze the malware behavior well and thereby developing efficient detection method. However, the log cannot always be acquired successfully in the cases where the execution process of malware finishes in a short time or the execution process does not finish but there only exists invalid log file. In such cases, it is difficult to capture the accurate malware behavior, since it is unknown that whether the dynamic analysis is done properly or not. Therefore, to clarify the reason why valid log cannot be acquired is necessary and important. This paper reveals that a rapid termination of process or a wait state of thread affects the log acquisition process of dynamic analysis of malwares, through observing the issuance of all the system calls made by various malware specimens.

Keywords: MWS, Malware, Dynamic Analysis, System Call Trace

1. はじめに

マルウェア対策では，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．マルウェアの技術は日々進歩しており, 次々と新しいマルウェアが出現している．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．マルウェアの動的解析では，解析にかかるオーバヘッドと取得情報を考慮して適切に解析を終了することが求められる．動的解析における課題として，マルウェアの動作判定，観測の終了条件が挙げられる．

マルウェアの動作判定

マルウェアの動的解析では，マルウェアが持つ機能がすべて実行された上で観測を終了することが望ましい．しかし，未知のマルウェアに対して動的解析を行う場合，マルウェアが持つ機能は不明である．また，マルウェアは，解析環境で本来の動作を行わない場合がある．動的解析は，実行されたマルウェアの挙動を観測

[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

するため，解析環境で実行されなかった機能は解析できない．したがって，動的解析において，マルウェアの機能がすべて動作したか否かを判断するのは困難である．

観測の終了条件

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．1 つのマルウェアを動的解析する場合において，マルウェアの実行時間は数分程度である場合が多く，静的解析と比較すると短時間でマルウェアの挙動を把握することが可能であるとされている．しかし，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が存在し，正確な解析を行うために長時間の観測を必要とする場合がある [1]．動的解析に必要な時間はマルウェアによって異なるはずであるため，マルウェアに対して適切な観測時間を設定できているとは言えない．一方，マルウェアのプロセスが終了する場合やスレッドが中断する場合，マルウェアのコードが実行されないため，マルウェアの観測を継続しても有効なログを得られない可能性がある．そのため，実際の動的解析では，マルウェアが動作を終了した場合に加え，マルウェアの観測を継続しても有効な動的解析ログを得られない可能性が高い場合も，観測を終了する基準として扱いたいケースが考えられる．

マルウェアの動的解析には，呼び出される Windows API を記録する API トレースや，発行されたシステムコールを記録するシステムコールトレースがある．これらは，API やシステムコール呼出し時にのみ処理をフックするため，機械語命令単位やメモリアクセス単位で記録するものと比べてオーバヘッドが小さい．また，実行された処理内容を API やシステムコールという意味のある単位で捉えるため，挙動の理解が容易である．したがって，高速さが求められる場合には API トレースやシステムコールトレースが有効である．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，解析のオーバヘッドを削減するため，マルウェアの解析に重要とされる API やシステムコールのみをフックして記録することが多い．このような解析環境において，マルウェアのプロセスが早々に終了したり，終了はしていないが有効なログが取得できないなど，「うまく動作しない」マルウェアの存在が確認されている．しかし，それらの具体的な動作内容については，十分に知見が得られておらず実態は定かではない．当該マルウェアの具体的な動作内容を明らかにすることで，挙動の検出や適切な観測の終了が可能となる．

本論文では，動的解析で「うまく動作しない」マルウェアの具体的な動作内容を調査するため，MWS データセット [2][3][4][5] で報告されているマルウェア検体を対象に全システムコールの発行を観測した．マルウェアの解析環境には，システムコールトレーサ Alkanet[6] を用いた．実態調査では，「うまく動作しない」マルウェアの具体的な動作内容と動的解析ログの関係性について調査を行った．本論文の実態調査により，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が明らかとなった．

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

本論文の調査では，動的解析で「うまく動作しない」マルウェアの具体的な動作内容を把握するため，マルウェア検体に対してシステムコールトレーサ Alkanet[6] を利用して動的解析を行った．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，VMM である BitVisor をベースとし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．システムコールのエントリポイントは，カーネル空間に存在するため，ユーザモードで動作するプロセスからはアクセスできず，エントリポイントを経由せずに呼び出すこともできない．マルウェアがユーザモードで動作している限り, システムに影響を与えるためにはシステムコールが必要であるため, システムコールをトレースすることで, マルウェアの挙動を把握することができる．

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．

Alkanet は，観測のオーバヘッド削減やログの可読性向上を目的として，マルウェアの挙動解析に有用なシステムコールに絞って引数や戻り値の情報取得とログ出力をすることができる．ここでは，トレース対象のシステムコールを絞る観測方法を限定モードと呼ぶ．限定モードの Alkanet がログに記録するシステムコールの例を表 2 に示

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

表 2　限定モードの Alkanet がログに記録するシステムコール
Table 2　System call logged by Alkanet in limited mode.

カテゴリ	システムコール
File	NtCreateFile, NtOpenFile, NtWriteFile, NtReadFile, NtSetInformationFile,NtDeletFile
Registry	NtCreateKey, NtOpenKey, NtOpenKeyEx, NtQueryKey, NtSetInformationKey, NtDeleteKey, NtSetValueKey, NtDeleteValueKey
Virtual Memory	NtWriteVirtualMemory, NtReadVirtualMemory, NtAllcateVirtualMemory,NtProtectVirtualMemory
File Mapping	NtCreateSection, NtOpenSection, NtMapViewOfSection,
Network	NtDeviceIoControlFile
Process	NtCreateProcess, NtCreateProcessEx, NtOpenProcess, NtTerminateProcess,NtCreateUserProcess
Thread	NtCreateThread, NtCreateThreadEx, NtOpenThread, NtSuspendThread, NtResumeThread, NtTerminateThread, NtGetContextThread,NtQueueApcThread
Driver	NtLoadDriver, NtUnloadDriver
Time	NtQueryPerformanceCounter
Mutant	NtCreateMutant, NtOpenMutant
Sleep	NtDelayExecution

す．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，限定モードではなく，全システムコールを観測対象としてログ出力を行う全出力モードでマルウェア検体の動的解析を行う．

2.3　観測ログの解析

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．

本論文の調査では，Alkanet から出力された観測ログを解析し，システムコールトレースを用いた動的解析において「うまく動作しない」マルウェアのログを抽出する．観測ログの解析は，動作プロセスの確認とマルウェアの最後に観測されたシステムコールの抽出によって行う．

動作プロセスの確認

本調査では，マルウェアが最後に発行したシステムコールに注目するため，マルウェアの挙動を正確に追跡する必要がある．プロセスを生成する挙動やコードインジェクションの挙動は，特定のシステムコールの発行を観測することで追跡できる．しかし，プロセスの生成やコードインジェクションを行う検体は，これらを行わない検体と比較して，マルウェアの挙動を正確に追跡するのに多くの時間を必要とする．そのため，本調査では，プロセスの生成やコードインジェクションを行わない検体のログを調査対象とする．そこで，Alkanet の解析結果を用いて，検体をプロセス生成やコードインジェクションを行う検体と行わない検体に分類する．

マルウェアの最後に観測されたシステムコールの抽出

プロセス生成やコードインジェクションを行わない検体の観測ログにおいて，マルウェアプロセスに関する最後に観測されたシステムコールを抽出する．また，マルウェアプロセスが発行したシステムコール発行列についても調査を行う．

3.　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

本章では，Alkanet を利用して行った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と結果について述べる．

3.1　調査目的

本調査の目的は，API トレースやシステムコールトレースにおいて「うまく動作しない」マルウェアの動作内容と動的解析ログの関係性を明らかにすることである．本調査では，マルウェアプロセスによるシステムコールの発行ログが観測されない原因を明らかにするため，観測時間のタイムアウトまでにマルウェアが発行したシステムコールのログを調査した．

3.2　調査対象と調査方法

本調査は以下の手順で行う．

手順 (1)　マルウェア 487 検体を 90 秒間観測する．

手順 (2)　解析結果を用いて，検体集合を (a) プロセス生成やコードインジェクションを行う検体集合と (b) 行わない検体集合に分類する．

手順 (3)　Alkanet の解析結果を用いて，検体集合（b）を (b1) プロセス終了やスレッド停止に関するシステムコールを発行した検体集合と（b2）発行していない検体集合に分類する．

手順 (4) 検体集合 (b1) と検体集合 (b2) についてそれぞれ考察する．

本調査では，マルウェアが観測時間のタイマアウトまでに行った挙動を追跡する必要がある．プロセスの生成やコードインジェクションを行う検体は，これらを行わない検体と比較して，マルウェアが動作停止するまでの挙動を解析するのに多くの時間を必要とするため，調査対象から除外する．そこで，手順 (2) では，Alkanet の解析結果を用いて，検体を (a) プロセス生成やコードインジェクションを行う検体と (b) 行わない検体に分類する．

API トレースやシステムコールトレースにおいて「うまく動作しない」マルウェアの動作として，プロセスやスレッドの動作に影響を与えるシステムコールが発行されている可能性がある．そこで，これらのシステムコールに注目し，マルウェアが発行したシステムコールの調査を行う．手順 (3) では，Alkanet の解析結果を用いて，検体集合（b）を (b1) プロセス終了やスレッドの中断に関するシステムコールを発行した検体と（b2）発行していない検体に分類し，手順 (4) では，検体集合 (b1) と検体集合 (b2) についてそれぞれ考察する．具体的には，プロセス終了やスレッドの動作中断に関するシステムコールの発行と動的解析ログの関係性について考察する．

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 487 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)

プロセスを生成する挙動は，NtCreateProcess システムコールや NtCreateProcessEX システムコールの発行を観測することで検出できる．また，コードインジェクションの挙動は，NtWriteVirtualMemory システムコールや NtCreateThread システムコールなどの発行を観測することで検出できる．

調査対象の 487 検体中 252 検体は，プロセス生成やコードインジェクションを行わず最初に起動されたプロセスのみで動作し，残りの 235 検体は，プロセス生成やコードインジェクションを行い，複数のプロセスで動作することを確認した．

3.3.2 手順 (3)

プロセス生成やコードインジェクションを行わない 252 検体において，60 検体が観測中にプロセスが終了することを確認した．プロセスが終了するマルウェアプロセスが最後に発行したシステムコールのログは，NtTerminateProcess システムコールまたは NtTerminateThread システムコールの sysenter 時のログであった．各システムコールの機能を以下に示す．

```
No.:    59989
Time:   135691748
Type:   sysenter
SNo.:   101 (NtTerminateProcess)
Cid :   b04.b08
Name:   002c8b63fa2d15d
Note:
 target:
  proc_pid: 0xb04
  proc_image_name: 002c8b63fa2d15d
```

図 1 NtTerminateProcess の発行ログ
Fig. 1 Log of NtTerminateProcess.

```
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 2 NtTerminateThread の発行ログ
Fig. 2 Log of NtTerminateThread.

NtTerminateProcess

指定されたプロセスとプロセスに含まれるスレッドを終了する．

NtTerminateThread

指定されたスレッドを終了する．

マルウェアプロセスが NtTerminateProcess システムコールまたは NtTerminateThread システムコールを発行したログを図 1 および図 2 に示す．図 1 が示すとおり，自身のプロセスを指定して，NtTerminateProcess を発行したことが分かる．NtTerminateProcess を発行したすべてのマルウェア検体は，NtTerminateProcess を二度発行したことを確認した．これは，ExitProcess API による NtTerminateProcess の発行パターンと一致する．図 1 では，終了するスレッドとして発行元のスレッドを指定して NtTerminateThread を発行したことが分かる．

プロセス生成やコードインジェクションを行わない 252 検体のうち残りの 192 検体は，観測時間のタイムアウトまでプロセスが存在したことを確認した．また，192 検体中 140 検体が GUI オブジェクトを表示した．マルウェア検体が表示する GUI オブジェクトは，ユーザによる GUI 操作を要求するものやエラーメッセージなどであった．

観測時間のタイムアウトまでプロセスが存在した 192 検体のマルウェアプロセスにおいて最後に観測されたシステムコールを表 3 に示す．多く確認されたシステムコールは，NtUserGetMessage，NtDelayExecution，NtUserWaitMessage などのログであった．これらは，最後に発行したシステムコールのログが sysenter 時のものであり，以後，

表 3　マルウェアプロセスにおいて最後に観測されたシステムコール
Table 3　Last observed system call in the malware process.

最後に発行されたシステムコール	検体数
NtUserGetMessage	112
NtDelayExecution	18
NtUserWaitMessage	17
NtWaitForSingleObject	5
NtProtectVirtualMemory	4
NtWaitForMultipleObjects	3
NtRequestWaitReplyPort	3
NtAllocateVirtualMemory	3
NtClose	3
NtUserQueryWindow	2
NtReadVirtualMemory	2
NtReplyWaitReceivePortEx	1
NtUserCallOneParam	1
NtQueryInformationProcess	1
NtOpenThreadToken	1
NtUserPeekMessage	1
NtUserCallMsgFilter	1
NtTerminateThread	1
NtUserShowWindow	1
NtContinue	1
NtunlockFile	1
NtQueryValueKey	1
NtFsControlFile	1
NtOpenKey	1
NtUserGetThreadState	1
NtOpenEvent	1
NtReleaseSemaphore	1
NtopenFile	1

システムコール発行が観測されなくなることを確認した．同様の挙動が得られたものを表 3 から抽出した結果を表 4 に示す．また，各システムコールの機能を以下に示す．

NtDelayExecution
　指定された時間が経過するまで，スレッドの動作を中断する．

NtUserGetMessage
　メッセージキューにメッセージが存在しない場合，スレッドの動作を中断する．

NtUserWaitMessage
　メッセージキューにメッセージが追加されるまで，スレッドの動作を中断する．

NtWaitForSingleObject
　指定されたオブジェクトがシグナル状態になるまでスレッドの動作を中断する．

NtWaitForMultipleObjects
　指定された複数のオブジェクトがシグナル状態になるまでスレッドの動作を中断する．

NtRequestWaitReplyPort
　プロセス間通信において，送信したメッセージに対する返信を受信するまで，スレッドの動作を中断する．

表 4　スレッドの動作が中断するプロセスの最終ログ
Table 4　Last log of process where thread is interrupted.

システムコール	検体数
NtUserGetMessage	112
NtDelayExecution	18
NtUserWaitMessage	17
NtWaitForSingleObject	5
NtWaitForMultipleObjects	3
NtRequestWaitReplyPort	3
NtReplyWaitReceivePortEx	1

NtReplyWaitReceivePortEx
　プロセス間通信において，メッセージを受信するまでスレッドの動作を中断する．

これらのシステムコールは，内部で特定の条件を満たすまでシステムコール発行元のスレッドを何らかの待ち状態とする．マルウェア検体が発行した NtDelayExecution システムコールのログエントリを図 3 に示す．Note の value エントリが引数に指定された待機時間の値を表す．このように，システムコールの引数を解析することで待ち状態の条件を取得することができる．マルウェア検体が発行した NtDelayExecution に指定された待機時間は，最大が 180 秒，最小が 0.5 秒であった．

スレッドの動作を中断するシステムコールを含むシステムコール発行列を図 4 および図 5 に示す．ログは，左からログ番号，プロセス名，発行されたシステムコール名を表す．GUI オブジェクトを表示した 140 検体中 107 検体の最終ログは，sysenter 時の NtUserGetMessage システムコールであった．また，これらのログでは，図 4 のように NtUserGetMessage を複数回発行する挙動が確認された．図 5 は，マルウェアプロセスのログとして，NtRequestWaitReplyPort システムコールが最後に観測されたことを表す．前述した Alkanet の限定モードでシステムコールトレースを行った場合，観測対象外の NtRequestWaitReplyPort の発行ログは記録されず，観測対象の NtOpenFile のログが最後に記録される．

最初に起動されたプロセスのみで動作する 252 検体の分類結果を表 5 に示す．プロセス終了に関するシステムコールを発行したのは 60 検体であった．スレッドの中断に関するシステムコールを発行したのは 158 検体であった．残りの 34 検体は，プロセス終了やスレッド停止に関するシステムコールを発行しない検体であった．これらは，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せず，観測時間のタイムアウトまで継続してシステムコールを発行することを確認した．

```
No. :  109539
Time:  146952746
Type:  sysenter
SNo.:  3b (NtDelayExecution)
Cid :  cf0.cf4
Name:  7c2e0dd2.exe
Note:
 alertable: false
 value: 0xffffffff94b62e00
```

図 3 NtDelayExecution の発行ログ
Fig. 3 Log of NtDelayExecution.

```
[340835] 6bd313f2b3a1dbd  KiUserCallbackDispatcher
[340836] 6bd313f2b3a1dbd  KiCallbackReturn
[340837] 6bd313f2b3a1dbd  NtUserGetMessage
[340838] 6bd313f2b3a1dbd  NtUserGetMessage
[341022] 6bd313f2b3a1dbd  KiUserCallbackDispatcher
[341023] 6bd313f2b3a1dbd  KiCallbackReturn
[341024] 6bd313f2b3a1dbd  NtUserGetMessage
[341025] 6bd313f2b3a1dbd  NtUserGetMessage
[341287] 6bd313f2b3a1dbd  KiUserCallbackDispatcher
[341288] 6bd313f2b3a1dbd  KiCallbackReturnre
[341289] 6bd313f2b3a1dbd  NtUserGetMessage
[341290] 6bd313f2b3a1dbd  NtUserGetMessage
```

図 4 NtUserGetMessage を含むシステムコール発行列
Fig. 4 System call log including NtUserGetMessage.

```
[1508229] 1bde10255cad621 NtMapViewOfSection
[1508230] 1bde10255cad621 NtMapViewOfSection
[1508231] 1bde10255cad621 NtClose
[1508232] 1bde10255cad621 NtClose
[1508233] 1bde10255cad621 NtQueryDefaultUILanguage
[1508234] 1bde10255cad621 NtQueryDefaultUILanguage
[1508235] 1bde10255cad621 NtOpenFile
[1508236] 1bde10255cad621 NtOpenFile
[1508237] 1bde10255cad621 NtQueryDefaultLocale
[1508238] 1bde10255cad621 NtQueryDefaultLocale
[1508239] 1bde10255cad621 NtQueryDefaultLocale
[1508240] 1bde10255cad621 NtQueryDefaultLocale
[1508241] 1bde10255cad621 NtQueryDefaultLocale
[1508242] 1bde10255cad621 NtQueryDefaultLocale
[1508243] 1bde10255cad621 NtOpenFile
[1508244] 1bde10255cad621 NtOpenFile
[1508245] 1bde10255cad621 NtRequestWaitReplyPort
```

図 5 NtRequestWaitReplyPort を含むシステムコール発行列
Fig. 5 System call log including NtRequestWaitReplyPort.

表 5 最初に起動されたプロセスのみで動作する 252 検体の分類結果
Table 5 Classification of 252 samples operated by only one process.

挙動	検体数
プロセス終了に関するシステムコールを発行する	60
スレッド中断に関するシステムコールを発行する	158
プロセス終了やスレッド中断に関するシステムコールを発行しない	34

4. 考察

4.1 プロセスやスレッドの終了

本論文の実態調査では，調査対象である 252 検体のうち，60 検体でプロセスが終了し，158 検体でスレッドが待ち状態となることがわかった．また，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どのように影響を与えるのか明らかとなった．

マルウェアのプロセスが終了する場合，マルウェアの観測を継続しても有効なログを得られない．ただし，その場合であっても，OS の自動実行機能を用いたプロセスの起動やインジェクションされたファイルやメモリの実行が行われる可能性がある．その場合は，解析を継続する必要がある．

解析環境においてマルウェアのプロセスが終了したとき，マルウェアの機能が実行された上でプロセスが終了したかが重要である．特に，マルウェアの機能が実行されないまま早々にプロセスが終了するものについては，マルウェアがもつ本来の機能を観測することができない．これらのマルウェアに対して動的解析を行うためには，プロセスが終了する要因を明らかにし，要因に対して適切に対応する必要がある．マルウェアのプロセスが終了する要因として，マルウェアのもつ耐解析性や環境依存性が挙げられる．マルウェアは，自身に対する解析の回避を試みる場合がある．動的解析に対する耐解析性として，挙動を複雑にすることで追跡を逃れるものや，解析環境を検知して動作を停止するものが存在する．また，OS，アプリケーションなど特定の条件を満たす環境でのみ動作するものも存在する．マルウェアがこのような動作を行った場合，関係する API やシステムコールが発行される可能性が高い．したがって，マルウェアのプロセスが終了するまでに観測されたログからプロセス終了までの一連の動作を把握することができると考える．

本調査では，マルウェアの最後に観測されたシステムコールに注目したため，マルウェアがプロセスを終了するまでに行った一連の動作については調査を行っていない．今後，プロセスが終了する挙動が観測されたものについては，プロセスを終了するまでに行った挙動を明らかにし，動的解析における対応を検討する必要がある．

4.2 待ち状態となる挙動

システムコール発行後に制御が戻らずシステムコール発行が観測されなくなる検体が存在した．それらのシステムコールは，内部で特定の条件を満たすまでシステムコール発行元のスレッドを待ち状態にする．したがって，システムコールによるスレッドの待ち状態が以後のシステムコール発行が観測されなくなる要因として考えられる．

GUI オブジェクトを表示した検体の多くは，NtUserGetMessage システムコールを発行して動作を中断することが確認された．Windows における GUI アプリケーションは，ユーザの操作に対応するメッセージを待機する処理が実装される．したがって，当該挙動は，何らかのユーザ操作を待機している状態と考えられる．そのため，GUI に

対して何らかの操作を行うことで動作を再開する可能性が高い．ただし，エラーメッセージを出力しているものに関しては，操作後にプロセスが終了する可能性がある．自動で動的解析を行う環境では，マルウェアによって GUI オブジェクトが表示されたとしてもユーザ操作を行わない場合がある．このような解析環境では，ユーザ操作をトリガとするマルウェアの挙動を観測できない．動的解析における対応として，GUI オブジェクトの解析やユーザ操作のエミュレートを行う方法が考えられる．

NtDelayExecution システムコールは，指定された時間が経過するまでスレッドを待ち状態にする．引数に指定される待機時間は様々であり，単体のログエントリのみから挙動の意図を掴むことは困難である．一方，調査結果では，60 秒を超えるような比較的大きな値が待機時間として指定されるケースが存在した．マルウェアは，スレッドの動作を中断することで，動作を遅延させるものが存在する．そのため，60 秒や 180 秒など明らかに大きな時間が指定されたものについては，解析の回避を意図している可能性が高い．マルウェアが動作の遅延を行った場合，動的解析では有効なログが取得できないため，解析環境において何らかの対応を行う必要がある．特に，長時間動作を中断するものについては，待機処理のスキップや待機時間の短縮などを検討する必要がある．

他のシステムコールについては，同期処理やプロセス間通信に関するものが観測された．これらは，システムコールの引数から同期対象のオブジェクトや通信対象のプロセスを特定することができる．ただし，システムコールから制御が戻る条件は，同期対象のオブジェクトや通信対象のプロセスに依存するため，単純に制御が戻るまでの時間を把握することはできない．また，これらのシステムコールの用途は多岐にわたるため，挙動の意図を把握するためには，詳細な解析が必要となる．

4.3 継続的に動作する場合

マルウェアのプロセスが観測の終了まで継続してシステムコール発行を行う場合，動的解析では，観測ログを遡ることでマルウェアが行った挙動を把握することが可能である．当該挙動を示すマルウェアについては，システムコールトレースを行う動的解析環境において継続して挙動を観測できると言える．ただし，マルウェアが動作を継続していたとしても有効なログが取得できるとは限らない．マルウェアが意味のない動作を繰り返すような場合，ログから得られた挙動に基づいて解析の終了を検討する必要がある．

本調査では，マルウェアが発行した特定のシステムコールにより，プロセスが終了したり，スレッドの動作が中断する挙動が明らかになった．マルウェアが当該挙動を行った場合，次の API やシステムコールの発行が行われないため，API トレースやシステムコールトレースを採用した動的解析環境では，有効なログが取得できないことがある．当該挙動を検出するためには，全システムコールのトレースや命令トレースを用いた手法が考えられる．しかし，これらの解析方法は，解析のオーバヘッドが大きくなるため，解析の高速さが求められる場合には適切ではない．そこで，一部の API 呼出しやシステムコールの発行を観測する解析環境において，関係するシステムコールの発行を監視する方法が有効であると考える．ただし，これらのマルウェアに対して適切に動的解析を行うためには，より詳細に挙動を明らかにする必要がある．具体的には，スレッドが待機状態となる条件やマルウェアのプロセスが終了するまでに行った挙動を明らかにすることが挙げられる．

5. 関連研究

青木らの研究 [7] 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，マルウェアがうまく動作しない場合の原因を明らかにするものではない．

大山の研究 [8] 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，さらにその原因の解明を広く試みるものである．

仲小路らの研究 [9] 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うまく動作しないマルウェアの具体的な動作内容を調査するものであるため，全システムコール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．

6. 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「うまく動作しない」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，有効なログが取得できない場合，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．
- GUI を表示する検体の多くでは，メッセージ待ちを行う NtUserGetMessage 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．

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が発行されることを確認した．また，プロセスの終了やスレッドの待ち状態が API トレースやシステムコールトレースを用いた動的解析のログに影響を与えることが明らかとなった．特定の API やシステムコールに限ってログを取得する環境では，観測のオーバヘッド削減やログの可読性を損なわずに当該挙動を適切に検出することが求められる．

参考文献

[1] 二木真明，佐藤元彦，山崎文明，内田勝也：標的型サイバー攻撃と APT に関する考察，情報処理学会研究報告，Vol. 2012, No. 20, pp. 1–8 (2012).

[2] 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).

[3] 秋山満昭，神薗雅紀，松木隆宏，畑田光弘：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，Vol. 2014-CSEC-66, No. 19, pp. 1–7(2014).

[4] 神薗雅紀，畑田充弘，寺田真敏，秋山満昭，笠間貴弘，村上純一：マルウェア対策のための研究用データセット〜MWS Datasets 2013〜，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. 1–8 (2013).

[5] 畑田充弘，中津留勇，秋山満昭：マルウェア対策のための研究用データセット 〜 MWS 2011 Datasets 〜，コンピュータセキュリティシンポジウム 2011 論文集，Vol. 2011, No. 3, pp. 1–5 (2011).

[6] 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).

[7] 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).

[8] 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，Vol. 2017,No. 2,pp. 487–494 (2017).

[9] 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).