Computer Security Symposium 2020
26 - 29 October 2020

マルウェアの動的解析回避処理の傾向についての Soliton Dataset 2020の分析

玉林 亜喬[1,a)]　大山 恵弘[1]

概要：現代のマルウェアの多くには，自身を解析されないように解析を回避する処理（解析回避処理）が組み込まれていることが知られている．中でもデバッガでの解析を回避するもの（アンチデバッグ）や，VM 環境での解析を回避するもの（アンチ VM）が主要である．これらについてはこれまでに多くの研究がなされ様々な手法が発見され対策されているが，これらの手法が，最新のマルウェアにおいてどの程度の割合で利用されているのかといった具体的な知見は不足している．本研究では，マルウェアの動的解析ログである Soliton Dataset 2020 を分析し，2019 年 1 月から 2020 年 4 月に収集されたマルウェアの解析回避処理の傾向を明らかにする．

キーワード：マルウェア，解析回避，ログ分析

Analysis of Soliton Dataset 2020 on Trends of Evasive Operations by Malware

Akyo Tamabayashi[1,a)]　Yoshihiro Oyama[1]

Abstract: It is known that most modern malware incorporates operations that evade analysis (analysis evasion operations). Among them, the ones that evade analysis by the debugger (anti-debug) and those that evade analysis in virtual machine environments (anti-VM) are the main ones. Many studies have been conducted on these methods and various methods have been discovered and countermeasures have been taken so far, but specific knowledge such as the proportion of these methods used is insufficient. In this research, we analyze Soliton Dataset 2020 which is a dynamic analysis log of malware, and clarify the tendency of analysis evasion operations of malware collected from January 2019 to April 2020.

Keywords: malware, analysis evasion, log analysis

1. はじめに

現代のマルウェアの多くには，自身の解析を妨害するような処理（解析回避処理）が施されていることが知られている．この解析回避処理には，デバッガによる解析を妨害する目的の処理（アンチデバッグ）や，VM・サンドボックス環境での手動・自動の，動的解析を回避する目的の処理（アンチ VM）が主要である [1–7]．本研究では，マルウェアが利用するこれらの解析回避処理について扱う．

[1] 筑波大学
University of Tsukuba
[a)] akyo@syssec.cs.tsukuba.ac.jp

解析回避処理についてはこれまでに多くの研究がなされている [3]．アンチデバッグ手法では，マルウェアがデバッガによる解析を阻止するために，デバッガを検知してマルウェアの本来の処理を終了するような手法が有名である．他にも，マルウェアは動いているがデバッガからは見えないようにする，もしくはデバッガの操作を撹乱するといった手法がある．アンチ VM 手法では，実機と VM 環境のリソースの違いなどを基にした痕跡を検知し，自身の挙動を変更する手法が主である．他には，Cuckoo Sandbox [8] のような自動動的解析システムのタイムアウトを狙った，長時間のスリープを利用した手法もある．

他にも多くの手法が開発・発見されており，それらの手法を解説した論文なども多く出されている．しかし近年のマルウェアにおいて，それらの手法がどれくらい利用されているのかという部分については具体的な調査は少なく，知見が不足していた．

これらの知見は，これまでマルウェア解析者の経験によるところが大きかったため，世間に共有されにくい傾向があった．特に，以下についての情報の共有はこれから高度化するマルウェアの解析において重要な知見になると考えている．

- ある解析回避手法をどれくらい数の検体が実装しているのか
- 1 検体にどれくらい数の手法を実装しているのか
- それらの手法を示す証拠としてマルウェア内部にどのような痕跡が残されているのか

本研究では，Soliton Dataset 2020 [9,10] の Cuckoo Sandbox の動的解析ログの API コール列を対象にこれらの調査を行う．また，解析回避処理の中でも，静的解析を回避する手法については扱わず，動的解析の回避を目的とした手法の調査のみを行う．

2. 対象とするデータセット

Soliton Dataset 2020 では，Windows 7 Professional をゲスト OS とした Cuckoo Sandbox 上に，InfoTrace Mark II（以下 Mark II）を構築し，Mark II のログと Cuckoo Sandbox のログの 2 つを提供している．InfoTrace Mark II は，端末上の操作・挙動を記録し，サイバー攻撃や内部不正の調査を支援する製品である．このデータセットの作成には，2019 年 1 月から 2020 年 4 月までに話題になったマルウェア 581 検体を利用している．作成に利用された検体は，マルウェアファイル形式として一般的だった PE 形式のマルウェアだけではなく，スクリプト型や，Word や Excel のマクロ型の形式のマルウェアなどが含まれている．内訳を**図 1** に示す．各項目は，データセット作成に用いられたマルウェアの拡張子の数である．

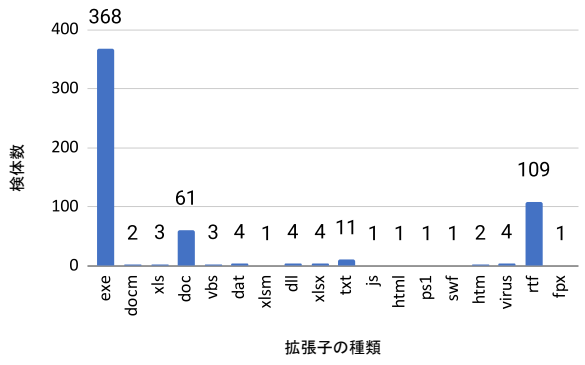

図 1 ファイル形式の内訳
Fig. 1 File format breakdown

581 検体のうち 372 検体 (64%) が exe ファイル,dll ファイルを含む PE 形式のマルウェアであり，そのうち exe ファイルが 368 検体，dll ファイルが 4 検体であった．PE 形式以外の 213 検体が rtf ファイルや doc ファイルなどのマルウェアであった．これらの PE 形式以外の Cuckoo 解析ログはマルウェア本体の挙動のログではなく，そのファイルを扱うソフトウェアのログになっている（doc ファイルだと解析対象のプロセスの名前が Word.exe のログが記録されている）ため今回の調査の対象からは除外した．dll のマルウェアは，当該ファイルを rundll32.exe を用いて DllMain 関数を実行し，解析を行っている．本研究では，Soliton Dataset 2020 の Cuckoo 解析ログのうち PE 形式のマルウェアである 372 検体を対象に調査を行った．

3. 対象とする手法

動的解析回避の手法は大きくアンチデバッグとアンチ VM の 2 つに分けられる．本研究ではさらに，マルウェアのアンチデバッグ手法をアンチデバッグに利用する要素を基に分類する．また，アンチ VM 手法についても同様に分類する．分類した手法のうち，どの手法を対象とするかを説明する．

まずアンチデバッグ手法については以下のように分類する．

- API を利用する手法：
  IsDebuggerPresent などの特定の API を利用しデバッガの有無を判定する方法や，API の機能によりデバッガからプロセスを隠すなどデバッガの操作を撹乱する方法が含まれる
- 例外を意図的に発生させる手法：
  例外を発生させて，その後の挙動の違いによりデバッガの有無を判別する方法が含まれる
- CPU 命令を直接利用する手法：
  特定の CPU 命令を利用，またはメモリアクセスや計算によりデバッガの有無を判定する方法が含まれる

Cuckoo Sandbox の動的解析ログには CPU 命令列は記録されないため，「CPU 命令を直接利用する手法」については今回の調査の対象外とする．

アンチ VM 手法については，以下のように分類する．

- API を利用する手法：
  ファイルアクセス API を用いて，VM 環境特有のファイルやレジストリを読み出して検知する方法や，特定の API を利用し，挙動の違いにより VM 環境を検知する方法が含まれる
- CPU 命令を直接利用する手法：特定のメモリ領域の読み出しを行って実機と VM 環境の違いを検知する方法や，実機と VM 環境で挙動が変わる CPU 命令を用いる方法が含まれる

アンチ VM 手法でもアンチデバッグ手法と同様に，「CPU

命令を直接利用する手法」については今回の調査の対象外とする．

またアンチ VM 手法の調査では，Cuckoo Sandbox の signature 機能を利用し分析を行う．signature 機能は実行対象のバイナリの内容や動作に見られる特徴である．例えば，他プロセスへのコードインジェクション，UPX によるパックなどがある．Cuckoo Sandbox は，対象のプログラムがどのファイルにアクセスしたのか，どのような引数でどのような API を呼び出したのか，といったことを監視しており，事前に定義したルールに基づいてマッチした signature があれば，それを解析ログに記録する．Cuckoo Sandbox ではアンチ VM に関する signature が豊富に定義されているため，本研究でも signature を利用する．

4. データセットの統計情報

今回解析する Soliton Dataset 2020 の Cuckoo Sandbox の動的解析結果のうち PE 形式のものの統計情報を**表 1** に示す．

各検体は，その中で生成された全てのプロセス・スレッド内で呼び出されたものも合わせて，平均で約 50000 回 API を呼び出しており，最大では約 760000 回もの呼び出しを行っている．また，Cuckoo の動的解析ログでは，API コール列に__exception__という名前で例外が記録され，それに付随する情報も記録されるため，発生した例外がどのような例外であったかを判定することができる．今回は__exception__() も数に含んでいる．最小値は 0 となっており，最小値が 0 となる検体は 10 検体存在した．これは全て，プロセス名が lsass.exe で pid が 492 のプロセスのみの検体であった．各検体が呼び出す API の種類は平均で 60 種類であり，データセット全体では 249 種類の API が呼び出されている．各検体が生成するプロセス数は平均で 4.5 個であり，最大で 1200 の検体もあった．圧倒的にその 1 検体が明らかな外れ値となっていたため，平均の計算ではこれを除外して計算した．また，1200 ものプロセスを生成した検体は MegaCortex [11] というファミリであった．スレッドに関しても同様に，平均は外れ値を除外して 20.6 個のスレッドが生成されている．最大値の 1800 個のスレッドを生成したのは上記の検体である．

5. アンチデバッグ手法の分析

前節で分類した手法ごとに，分析を行う．

5.1 API を利用する手法

API を利用してアンチデバッグを行う手法の数の調査については，**表 2** に示す検知方法を用いて集計した．また，それらを利用していると検出された検体数も表 2 に併記した．なお，手法の名前の内，「IsDebuggerPresent」と「FindWindow」以外のものは論文 [3] を参考にしている．この 2 つの手法も論文 [3] に記載されているが，「Reading-PEB」や「SystemArtifact」といった意味が広い名称が与えられているため，今回は利用する API 名を手法の名前とした．

検出されたそれぞれの手法について解説と考察を行う．検出されなかったものについては本論文では解説を省くため，詳細については文献 [3, 12, 13] を参照されたい．

5.1.1 IsDebuggerPresent

IsDebuggerPresent を利用したアンチデバッグ手法が一番多い結果となり，対象としたデータセットの約半数がこれを実装していた．IsDebuggerPresent は，デバッガによってプロセスがデバッグされているか判定する API である．プロセスがデバッグされているとこの API は真を返し，そうでなければ偽を返す．このように IsDebuggerPresent は非常に単純で使いやすいため多用されると思われる．しかし，IsDebuggerPresent はライブラリ関数内で勝手に呼び出されることもあり，マルウェア開発者がアンチデバッグの意図で組み込んでいないことも多いと考えられる．また DLL 型のマルウェアは，4 検体全てがこの IsDebuggerPresent を呼び出す手法のみを実行していた．

5.1.2 ParentCheck

自身の親プロセスを確認することでデバッガの存在を判定する手法である．CreateToolhelp32Snapshot, Process32First, Process32Next という API を利用することで親プロセスを確認する．CreateToolhelp32Snapshot は，実行すると引数の一部である Flag で指定した種類の情報が作成され，そのハンドルが返される．これをスナップショットと呼ぶ．親プロセスを検索する場合は TH32CS_SNAPPROCESS などのプロセス一覧を取得するための Flag を指定する．Process32First, Process32Next は，このスナップショットのハンドルを受け取り，取得したプロセス一覧の各プロセスの詳細な情報を構造体の形で取得する．まず Process32First を実行して，その後 Process32Next を繰り返し実行することで，全プロセスの情報を確認することができる．一般的にプログラムはダブルクリックで起動されることが多く，親プロセスが explorer.exe であることが多い．これを利用してこの手法では，上記の方法で確認した親プロセスが explorer.exe ではない，もしくは特定のデバッガであると推測される場合，デバッグされていると判定する．

5.1.3 Timing-Based Detection

時刻情報を 2 回取得し，その差を計算して事前に設定した閾値よりも長かった場合，この 2 回の間にデバッガでの操作が介入したと判定する手法である．時刻情報を取得する API は様々な用途で利用されるため，アンチデバッグの用途以外に呼ばれることのほうが多い．アンチデバッグの目的であると，時刻情報を 2 回取得する間に挟まる API が少ないと予想される．理由としては，そのほうがデバッガ

表 1 Soliton Dataset 2020 の統計情報
Table 1 Statistics for Soliton Dataset 2020

検体数	368
各検体の API 呼び出し数	最小：0　平均：50446　最大：768136
各検体が呼び出す API の種類数	最小：0　平均：60　最大：149
全検体を通して呼び出される API の種類数	249
各検体のプロセス数	最小：1　平均：4.5　最大：1200
各検体のスレッド数	最小：1　平均：20.6　最大：1800

表 2 API を用いた手法の検知方法
Table 2 Method of detecting method using API

名前	検知方法	検体数
IsDebuggerPresent	IsDebuggerPresent が呼び出されている	151
ParentCheck	CreateToolhelp32Snapshot, process32First, precess32Next が順番に呼び出されている	95
Timing-Based Detection	timeGettime で API コール列が挟まれている	83
DebugString	OutputDebugString が呼び出されている	70
FindWindow	FindWindow が呼び出されている	67
SuspendingThread	SuspendThread が呼び出されている	1
Search for Breakpoints	GetThreadContext が呼び出されている	0
SelfDebugging	DebugActiveProcess が呼び出されている	0
ThreadHiding	SetInformationThread が呼び出されている	0

表 3 時刻取得関数に挟まれた API
Table 3 sandwiched API between time acquisition functions

API	呼び出し回数	検体数
None	529845	69
NtDelayExecution	57703	82
__exception__	155	57
NtClose	2	1

での操作が介入していない状態のときとの時間の誤差が小さくなるためである．

時刻を取得する API を短いスパンで 2 回呼び出しているものを検出した．その際，間に挟まる API 数は 5 を限度にしてある．

2 回の時刻取得 API 関数の呼び出しに挟まれた API を**表 3** に示す．None は間に API が何も挟まれていなかったことを表している．__exception__は間に例外が発生したことを表している．圧倒的に最も多かったものが，間に API が何も挟まっていないものであった．これは間に CPU 命令が挟まれており，何らかの処理が行われているのではないかと考えられるが，今回の調査では調べることができなかった．次に多かったのは NtDelayExecution であった．これは引数に指定した数値 × ミリ秒スリープする API である．これが最も多い理由は，NtDelayExecution を挟むことにより，デバッガの操作が介入していない状態での実行時間が一定になりやすくなるためであると考えられる．

5.1.4 DebugString

OutputDebugString という API を利用して，実行後の挙動の違いによりデバッガの存在を判定する手法である．この API は引数として与えた文字列をデバッガに出力するものである．もし，デバッガにアタッチされていると問題なくデバッガに引数の文字列が送られるが，デバッガからアタッチされていなかった場合はエラーコードを出力する．このエラーコードを確認することでデバッガの存在を判定することができる．

表 4 OutputDebugString に与えられた文字列の一部
Table 4 Part of the string given to OutputDebugString

```
''
%s----------------------------------------------\n'
\r'
--- WinLicense Professional ---\n'
\r'
--- (c)2012 Oreans Technologies ---\n'
\r'
----------------------------------------------\r\n'
\n'
\n',
2812',
Click on :  x=400,y=0\n',
MainRun Exit',
first run'
```

OutputDebugString がアンチデバッグに利用されているかを確認するための情報として，デバッガに送られようとした文字列の一部を**表 4** に示す．

表 4 に示した文字列の他に，空の文字列を出力するものと，“C:/” で始まるファイルパスを出力するものが多数存在した．“MainRun Exit’,” や，“first run’” は本当にデバッグ用の文字列であると考えられる．さらには，ファイルパスや “--- WinLicense Professional ---\n’” と

表 5 マルウェアが検索したウィンドウ名
Table 5 Window name searched by malware

OLLYDBG
GBDYLLO
pediy06
WinDbgFrameClass
File Monitor - Sysinternals: www.sysinternals.com
Process Monitor - Sysinternals: www.sysinternals.com
Registry Monitor - Sysinternals: www.sysinternals.com
SafeNet Borderless Single Sign On.....
_AcroAppTimer'

表 6 例外の発生数
Table 6 Number of exceptions

例外シンボル名	検体数	累計数
STATUS_PRIVILEGED_INSTRUCTION	68	7798
STATUS_ILLEGAL_INSTRUCTION	67	80
STATUS_INVALID_PARAMETER	51	248
STATUS_ACCESS_VIOLATION	16	96
0x80040155	4	12
STATUS_BREAKPOINT	1	177
STATUS_INTEGER_DIVIDE_BY_ZERO	1	19
0x0eedfade	1	4

いった意図が不明なものもあるが，空の文字列を出力する挙動は，アンチデバッグである可能性が高いと推測できる．

5.1.5 FindWindow

FindWindow を利用して，立ち上がっているウィンドウからデバッガと思しきものを探す手法である．FindWindow に検索対象としてマルウェアが与えた文字列を**表 5** に示す．

表 5 を参照すると，OLLYDBG や Process Monitor など明らかな解析用の製品を検索していることが多いことが伺える．

また，GBDYLLO（OLLYDBG の反転）を検索している．これは，マルウェア解析者が “OLLYDBG” を検索する手法に対抗してウィンドウ名を反転させていることにさらに対抗していると考えられる．このことから FindWindow はアンチデバッグの用途で利用されている可能性が高い．

5.2 例外を意図的に発生させる手法

例外を発生させた検体数と発生した例外の累計数を**表 6** に示す．例外シンボル名には，Windows API が定める例外コードに対応したシンボル名を記載している．例外コードに対応したシンボル名が不明な場合は，例外コードを記載した．

それぞれの例外についての議論は本題ではないため，詳しい解説を省略する．

ここで注目したのは，STATUS_BREAKPOINT と STATUS_INTEGER_DIVIDE_BY_ZERO である．これら 2 つはそれぞれ 1 検体のみが多数回発生させており，デバッガの存在を逐一確認しているようにも見える．そのため，これら 2 つは意図的に発生させたアンチデバッグ目的の例外であると考えている．これら 2 つの詳細を解説する．

STATUS_BREAKPOINT

STATUS_BREAKPOINT は，int 3 命令を実行することにより発生する例外である．本来 int 3 命令は，デバッガによりソフトウェアブレークポイントが設定される際に，該当箇所の命令を置き換える形で利用される．そのため，デバッガがアタッチ中でない限りこの命令がプログラム中で利用されることは基本的にありえない．よってこの例外は，意図的に int3 命令を実行して発生させている例外であると考えられる．

STATUS_INTERGER_DIVIDE_BY_ZERO

STATUS_INTERGER_DIVIDE_BY_ZERO は，ゼロ除算を行った際に発生する例外である．ある数値をゼロで割れば発生するという状況は非常に意図的に発生させやすい．また，この例外を発生させている 1 検体を調べると，例外を発生させている命令は “div eax” という命令であった．div 命令は edx を上位 32 ビットに，eax を下位 32 ビットにした数値を被除数として，オペランドに与えられた数値で除算する命令である．“div eax” は一般的にあまり使われない命令であり，命令実行時のレジスタは edx も eax も 0 であった．この例外は非常に不自然な除算により発生しているため，意図的なものであると考えられる．

その他

その他の例外でもアンチデバッグの目的で意図的に発生させている可能性は考えられる．特に，STATUS_ACCESS_VIOLATION は意図的に発生させやすく簡単に実装できるため，今回検出したものの一部にもアンチデバッグの目的のものが含まれていると予想される．しかし，手軽な反面，意図せずとも環境の違いなどで自然に発生することも多いため，どれくらいの割合で含まれているのかは判断できない．

6. アンチ VM 手法の分析

6.1 本研究での調査結果

アンチ VM 手法の分析には，Cuckoo Sandbox の動的解析結果に記録された signature を利用する．アンチ VM 関連の signature には，signature の名前に “antivm_”, “antiemu_”, “antisandbox_” といった接頭辞が付加されているため，これを抽出することにより分析を行う．各 signature としては，“antivm_” が 9 種類，“antiemu_” が 1 種類，“antisandbox_” が 4 種類検出された．検出した結果を**表 7** に示す．それぞれの signature の説明を**表 8** に示す．これは，Cuckoo Sandbox の動的解析結果に記録された各 signature に付随して出力された文章である．

解析のタイムアウトを狙う用途でも利用されるスリー

表 7　シグネチャの検出件数
Table 7　Number of detected signatures

signature 名	検体数
antisandbox_sleep	181
antivm_memory_available	162
antivm_queries_computername	133
antivm_network_adapters	97
antisandbox_foregroundwindows	68
antivm_vmware_in_instruction	67
antivm_generic_bios	66
antivm_disk_size	16
antivm_vbox_keys	4
antisandbox_idletime	4
antisandbox_cuckoo_files	4
antiemu_wine	4
antivm_generic_cpu	3
antivm_generic_services	1

プ処理を実行する “antisandbox_sleep” が最も多い結果となった．続いて，利用可能メモリのサイズを調べる “antivm_memory_available”，コンピュータ名を確認する “antivm_queries_computername”，ネットワークアダプターを確認する “antivm_network_adapters” という結果となっている．しかし，これらは解析回避の目的以外にも利用されることが多いため上位に入っていると予想される．検出された signature には，他の目的を想定しにくく，本当に解析回避の目的で利用されたと思われる signature も含まれている．例えば，“antivm_vmware_in_instruction” と “antivm_generic_bios” と “antisandbox_foregroundwindows” である．“antivm_vmware_in_instruction” は，in 命令を利用して VMware 環境を検出する手法の signature である．“antivm_generic_bios” は，BIOS 情報を確認して VM 環境を検知する手法の signature である．“antisandbox_foregroundwindows” は，ウィンドウの変更を定期的に確認して，自動解析環境であるか推測する手法の signature である．

次に，1 検体に記録されているアンチ VM 処理の signature の数を調査した．signature が記録された数ごとの検体数を**図 2** に示す．

アンチ VM の処理を 1 回も実行していないとされる検体が最も多く，85 検体 (23%) であった．またこれにより，全体の 77%の検体は何らかのアンチ VM と思われる処理を組み込んでいることがわかった．続いて記録数が 3 つであるものが 75 検体，1 つのものが 70 検体という結果であった．最も多く signature が記録されていたのは 2 検体存在し，6 つの signature を記録した．

アンチ VM 処理の signature 数と検体数に関して，これといった特徴的な傾向は見受けられなかったが，6 つの signature を記録したものから急に検体数が少なくなっており，7 つの signature を記録した検体は存在しなかった．

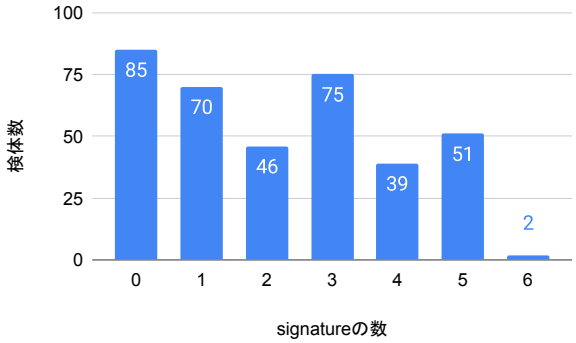

図 2　signature の数ごとの検体数
Fig. 2　Number of samples per number of signatures

全体として検出されたアンチ VM 処理の signature が 14 種類であるのに対し，1 つの検体には多くてもその半数も実装されていないことがわかる．これは，マルウェアの目的，またはファミリの違いにより実装する手法にばらつきがあるためだと考える．

6.2　過去の研究との比較

データセットを用いた解析回避処理の傾向調査の研究として，FFRI Dataset 2017 [14] を用いた研究 [2] がある．この研究では，解析回避に関する各 signature ごとの検体数の割合，1 検体に記録された解析回避に関する signature の分布についての結果が示されている．本研究で行う比較については，検出された解析回避に関する signature ごとの検体数の割合の比較のみ行い，1 検体に記録された signature の数についての比較は行わないことにしている．理由としては，研究 [2] では対象の解析回避に関する signature を，アンチデバッグなどを含むアンチ VM 以外の signature も対象にしているため，こちらの結果に差が出るのは当然であるためである．

記録された signature について，FFRI Dataset 2017 では，1 位，2 位のシグネチャは全体の 50 %を超えて記録されていたが，Soliton Dataset 2020 では 1 位のものでも全体の約 50 %である．また，今回 1 位で 50 %ほど記録されたスリープ挙動のシグネチャは，FFRI Dataset 2017 では 5 位となっており，検体の数の割合も 13.9 %と大きく異なっている．他にも，記録された上位の中での差ではあるのだが，異なっている部分が多い．

これらは，FFRI Dataset 2017 を用いた研究から 3 年が経ち，解析環境が変化したことも一つの原因であると考えるが，データセット作成のためのマルウェアの収集方法の違いが大きく起因していると考える．FFRI Dataset 2017 では，VirusTotal [15] を用いた収集，独自の Web Crawling による収集，他ベンダとの検体の交換，とできるだけランダムになっているが，Soliton Dataset 2020 では，話題になったマルウェアを中心に，そのファミリをいくつかまと

表 8　各 signature の説明
Table 8　Description of each signature

antivm_disk_size	Queries the disk size which could be used to detect virtual machine with small fixed size or dynamic allocation
antivm_generic_bios	Checks the version of Bios, possibly for anti-virtualization
antivm_generic_cpu	Checks the CPU name from registry, possibly for anti-virtualization
antivm_generic_services	Enumerates services, possibly for anti-virtualization
antivm_memory_available	Checks amount of memory in system, this can be used to detect virtual machines that have a low amount of memory available
antivm_network_adapters	Checks adapter addresses which can be used to detect virtual network interfaces
antivm_queries_computername	Queries for the computername
antivm_vbox_keys	Detects VirtualBox through the presence of a registry key
antivm_vmware_in_instruction	Detects VMWare through the in instruction feature
antisandbox_cuckoo_files	Attempts to detect Cuckoo Sandbox through the presence of a file
antisandbox_foregroundwindows	Checks whether any human activity is being performed by constantly checking whether the foreground window changed
antisandbox_idletime	Looks for the Windows Idle Time to determine the uptime
antisandbox_sleep	A process attempted to delay the analysis task.
antiemu_wine	Detects the presence of Wine emulator

めて収集している．傾向分析にはどちらのほうが良いというのは一概に判断できないが，データセットを用いた研究は，データセットによる違いなどを把握した上で行ったほうが良いと考える．

7. 関連研究

解析回避処理の傾向調査の研究はいくつかされており，同じようにデータセットを用いた調査では，研究 [1,2] が挙げられる．これらは FFRI Dataset の 2016 と 2017 のものを用いて，記録されている signature を基に主にアンチ VM に関する解析回避処理の定量的調査を行っている．この研究では，それぞれ 2016 年のマルウェアと 2017 年のマルウェアのデータセットが用いられており，現在からすると古い情報となっている．他にも傾向調査の研究 [4–6] は行われているが本研究では 2019 年内に収集された新しいデータを用いている，また，Black らの研究 [7] ではバンキングマルウェアを対象に，マルウェアファミリごとにどのような解析回避処理が利用されているかを調査している．しかし，それぞれのファミリごとの調査のため，バンキングマルウェア全体を通した定量的な調査は行われていない．

Afianian らの研究 [3] では，静的解析回避手法を含めた様々な手法の解説を行っている．マルウェアが利用する解析回避不法について網羅的に解説されており，実際にどのマルウェアに実装されているかという調査は行われているが，どれくらいの割合で利用されているのかという調査は行われていない．

Kirat らの研究 [16] では，解析回避処理の signature を，システムコールシーケンスから自動的に特定するシステムである MalGene を開発している．本研究で行ったような傾向調査では，signature は非常に有用な情報であるため，この signature の定義の自動化は，これからの傾向調査の研究のためにも重要である．

解析回避処理を無効化する技術やシステムについての研究は，これまでに多くなされている [17–20]．これらの研究がどのような手法を利用したとしても，それぞれの解析回避処理に関する知見は必要不可欠である．そのため本研究で行ったような傾向調査により，これらの研究のさらなる発展が期待できる．

8. まとめと今後の課題

Soliton Dataset 2020 を分析調査することにより，2019 年 1 月から 2020 年 4 月に収集されたマルウェアの解析回避処理の傾向を示した．

アンチデバッグ手法としては，IsDebuggerPresent を用いる手法が最も多く利用されており，続いて親プロセスをチェックする手法，時刻を取得し差を測る手法が多く利用されていた．また，時刻を用いる手法とデバッグ用文字列を用いる手法とウィンドウを検索する手法の分析では，手法に用いられる文字列などの要素も示し，痕跡を確認した．例外を利用する手法の分析では，発生した例外を全て洗い出すことでアンチデバッグの痕跡を探した．結果として，ブレークポイントを発生させる例外や，ゼロ除算例外といったアンチデバッグに利用されることが多い痕跡を発見した．

アンチ VM 手法としては，解析時間のタイムアウトを狙う手法でも用いられるスリープ処理が最も多く検出され，続いて，利用可能なメモリ量を調査する手法，コンピュータ名を調べる手法が多用されていた．1 つの検体から検出された signature は 0 のものが最も多く，続いて 3 つのもの，1 つのものとなっていた．また，最大で 6 つの signature が

1 つの検体に記録されていた．

今後の課題として，調査のためのデータの検体数を増やすことが挙げられる．過去の FFRI Dataset の動的解析では数千検体が用いられていたことに比べると，今回の 368 検体は，傾向分析を行うには少ない．他には，今回利用した解析回避処理の手法の検出方法には，利用される API の呼び出しを検知するだけといった false positive が検出されやすいものも多かったため，この false positive を下げる手法の開発を行いたい．さらには，未知の手法に対応できるように解析回避の傾向を掴むような研究が望ましい．

謝辞 Soliton Dataset 2020 を提供していただいた株式会社ソリトンシステムズと MWS 組織委員会に感謝する．本研究の一部は JSPS 科研費 17K00179，20K11741 の助成を受けている．

参考文献

[1] Yoshihiro Oyama. Trends of anti-analysis operations of malwares observed in API call logs. *Journal of Computer Virology and Hacking Techniques*, Vol. 14, No. 1, pp. 69–85, 2018.

[2] 大山恵弘．マルウェアが実行する耐解析処理の定量的傾向．日本ソフトウェア科学会大会論文集, Vol. 34, pp. 179–186, 2017.

[3] Amir Afianian, Salman Niksefat, Babak Sadeghiyan, and David Baptiste. Malware dynamic analysis evasion techniques: A survey. *ACM Comput. Surv.*, Vol. 52, No. 6, November 2019.

[4] Xu Chen, Jon Andersen, Z Morley Mao, Michael Bailey, and Jose Nazario. Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware. In *2008 IEEE international conference on dependable systems and networks with FTCS and DCC (DSN)*, pp. 177–186, 2008.

[5] Ping Chen, Christophe Huygens, Lieven Desmet, and Wouter Joosen. Advanced or not? a comparative study of the use of anti-debugging and anti-VM techniques in generic and targeted malware. In *IFIP International Conference on ICT Systems Security and Privacy Protection*, pp. 323–336. Springer, 2016.

[6] Rodrigo Rubira Branco, Gabriel Negreira Barbosa, and Pedro Drimel Neto. Scientific but not academical overview of malware anti-debugging, anti-disassembly and anti-vm technologies. *Black Hat*, 2012.

[7] P. Black and J. Opacki. Anti-analysis trends in banking malware. In *2016 11th International Conference on Malicious and Unwanted Software (MALWARE)*, pp. 1–7, 2016.

[8] Cuckoo Sandbox - Automated Malware Analysis. `https://cuckoosandbox.org/`.

[9] Soliton Dataset 2020. `https://www.iwsec.org/mws/2020/files/Soliton_Dataset_2020.pdf`.

[10] 寺田真敏, 秋山満昭, 松木隆宏, 畑田充弘, 篠田陽一. マルウェア対策のための研究用データセット mws datasets〜コミュニティへの貢献とその課題〜. Technical Report 8, 東京電機大学／株式会社日立製作所, NTT セキュアプラットフォーム研究所, 株式会社エヌ・エフ・ラボラトリーズ, 日本電信電話株式会社, 北陸先端科学技術大学院大学, jul 2020.

[11] 標的型攻撃ランサムウェア「MegaCortex」の内部構造を紐解く. `https://www.mbsd.jp/blog/20191113.html`.

[12] Anti Debugging Protection Techniques with Examples. `https://www.apriorit.com/dev-blog/367-anti-reverse-engineering-protection-techniques-to-use-before-releasing-software`.

[13] An Anti-Reverse Engineering Guide - CodeProject. `https://www.codeproject.com/Articles/30815/An-Anti-Reverse-Engineering-Guide`.

[14] FFRI Dataset 2017 のご紹介. `https://www.iwsec.org/mws/2017/20170606/FFRI_Dataset_2017.pdf`.

[15] VirusTotal. `https://www.virustotal.com/`.

[16] Dhilung Kirat and Giovanni Vigna. Malgene: Automatic extraction of malware analysis evasion signature. In *the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 769–780, 2015.

[17] Seokwoo Choi, Taejoo Chang, Sung-woo Yoon, and Yongsu Park. Hybrid emulation for bypassing anti-reversing techniques and analyzing malware. *The Journal of Supercomputing*, pp. 1–27, 2020.

[18] Min Gyung Kang, Heng Yin, Steve Hanna, Stephen McCamant, and Dawn Song. Emulating emulation-resistant malware. In *the 1st ACM Workshop on Virtual Machine Security*, VMSec ' 09, p. 11–22, New York, NY, USA, 2009.

[19] Hao Shi and Jelena Mirkovic. Hiding debuggers from malware with apate. In *the Symposium on Applied Computing*, pp. 1703–1710, 2017.

[20] Daniele Cono D'Elia, Emilio Coppa, Federico Palmaro, Lorenzo Cavallaro. On the dissection of evasive malware. *IEEE Transactions on Information Forensics and Security*, Vol. 15, pp. 2750–2765, 2020.