Soliton Dataset 2021 における マルウェアによる解析回避処理の調査

大山 恵弘[1,a)]

概要：多くのマルウェアがサンドボックスやアンチウィルスによる解析を回避するための処理（解析回避処理）を実行することが知られている．解析回避処理にはたとえば仮想マシンの検知や長時間のスリープがある．それらの手法についてはすでに多くの知見が共有されているが，マルウェアによる解析回避処理の利用の実態については知見が少ない．また，マルウェアの傾向は時系列的に変化するので，実態に関する過去の知見が現在にもあてはまるとは限らない．そこで本研究では，最近のデータセットの 1 つである Soliton Dataset 2021 を用いて，マルウェアが実行する解析回避処理の傾向や手法を調査した．本論文ではその調査結果について，具体的には個々の解析回避処理を実行すると判定された検体の割合や，それらの処理のために実行された API コールなどについて述べる．

キーワード：マルウェア，解析回避，アンチ VM，アンチデバッグ，動的解析，Cuckoo Sandbox

Investigation on Evasive Operations of Malware in Soliton Dataset 2021

Yoshihiro Oyama[1,a)]

Abstract: It is known that many malware programs execute evasive operations, i.e., operations to avoid analysis by sandboxes and anti-viruses. Evasive operations include virtual machine detection and long sleep. Although many knowledge has been shared about the methods of these operations, there is little knowledge about the actual use of these operations by malware. In addition, since the trend of malware changes over time, past knowledge about the actual situation does not necessarily apply to the present. In this study, we used the Soliton Dataset 2021, one of the most recent datasets, to investigate the trends and methods of evasive operations performed by malware programs. In this paper, we describe the results of the study, specifically, the ratio of malware samples that were determined to execute each evasive operation, and the API calls executed for these operations.

Keywords: Malware, analysis evasion, anti-VM, anti-debug, dynamic analysis, Cuckoo Sandbox

1. はじめに

マルウェアが実行する処理は長い間に渡って継続的に洗練されてきており，セキュリティシステムによる解析や検知を回避するためにマルウェアが実行する処理（解析回避処理）も，極めて高度になっている．解析回避処理についてはこれまで多くの研究がなされており，多くの知見が過去の文献などで共有されている [1–5, 7, 9, 11, 13, 14]．しかし，マルウェアに組み込まれた解析回避処理は時とともに変化するため，その傾向については継続的に調査し理解しておく必要がある．なお，本研究では解析回避処理という言葉を広い意味で用い，仮想マシンの検出や長時間のスリープなどの解析や検出を避けるための受動的な処理に限らず，セキュリティシステムを攻撃するなどの能動的に解

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

析や検知を妨害する処理も含むものとする．

本論文では，最新のデータセットの 1 つである Soliton Dataset 2021 [15,16] を用いて，実際のマルウェアが実行する解析回避処理の実態について調査した結果を報告する．過去に同様の調査を行い分析結果を報告した研究は存在する [2,5,11,13,14] が，それらの結果が現在のマルウェアにどの程度あてはまるかは明らかではなく，できるだけ新しいデータを用いて継続的に分析を行う必要がある．

本論文では，データセットを分析して得られた，最近のマルウェアが実行する解析回避処理の全体的傾向や具体的な処理内容を報告する．それにより，そのような処理に対する注意を喚起するとともに，マルウェアをより深く理解し，対策を構築する助けとなる情報を提供することを目指す．さらに，データセットの作成に用いられたサンドボックスによる動的解析の制限についても述べる．

2. Soliton Dataset 2021

Soliton Dataset 2021 には，マルウェア 787 検体を動的解析のサンドボックスである Mark II と Cuckoo Sandbox の両方で解析した結果が記録されている．さらに，Windows 7 と Windows 10 のそれぞれをゲスト OS に用いて解析した結果がともに記録されている．なお，データセットが対象とする検体は無作為ではなく作成者の判断に基づいて選択されているため，それらは世界におけるマルウェアの分布や傾向を必ずしも忠実に反映していない．

データセットが対象としているマルウェアをファイル形式で分類した結果を**表 1** に示す．File format の列はマルウェアのファイルの形式，#files の列は各ファイル形式のファイル数を示している．解析結果は JSON ファイルに記録されているが，3 つの検体の解析において，解析が中断しており，JSON ファイルも途中で途切れている．表の最後の行の 3 検体がそれに相当する．なお，この 3 検体はいずれも PE DLL (32 bit) である．

本研究では Mark II と Cuckoo Sandbox の両方の解析結果を比較した結果，Cuckoo Sandbox 上の Windows 10 を用いた解析結果を分析することにした．その理由は，Mark II の解析結果はインシデント対応などで役立つと思われるものの，解析回避処理の理解という側面からは Cuckoo Sandbox の解析結果のほうが具体的な情報が多く使いやすいと判断したからである．

Soliton Dataset に関する過去の知見としては，玉林ら [13] が，Soliton Dataset 2020 のアンチデバッグ処理とアンチ VM 処理についての分析結果を提示している．本研究では彼らの研究とは異なり，2021 年版のデータセットを用いると共に，より広い範囲の解析回避処理を分析対象とする．具体的には，彼らの研究では，文献 [1] の手法で検出できるアンチデバッグ処理と，Cuckoo Sandbox が出す signature から把握できるアンチ VM 処理のみを対象にしていたが，

表 1 Soliton Dataset 2021 で対象とされているマルウェアのファイル形式

Table 1 File format of malware programs targeted in Soliton Dataset 2021.

File format	#files
PE DLL (32 bit)	281
PE non-DLL GUI (32 bit)	235
PE non-DLL console (32 bit)	36
PE DLL (64 bit)	75
PE non-DLL GUI (64 bit)	49
PE non-DLL console (64 bit)	27
PE native (64 bit)	1
MS-DOS	2
Document files (Word, PDF, ...)	78
(Broken analysis result)	3
Total	787

表 2 分析対象検体の統計情報

Table 2 Statistical information on analysis target malware.

分析対象検体数	348
各検体の最小/平均/最大プロセス数	1/4.9/452
各検体の最小/平均/最大スレッド数	1/26.7/4443
各検体の最小/平均/最大 API コール数	0/36560.6/5584934
各検体の最小/平均/最大検出 signature 種類数	0/6.4/26
全検体を通しての検出 signature 種類数	115

本研究では，Cuckoo Sandbox が出す signature から把握できる解析回避処理一般を対象とする．この変更に伴い，彼らの研究で行われていた，例外発生の調査や時刻情報を用いる検出手法に関する調査も，本研究では行わない．

本研究では DLL ファイルではない PE ファイル 348 検体 (=235+36+49+27+1 検体) を対象に，Cuckoo Sandbox に記録された API コール列と signature 情報を主な手がかりとして，マルウェアの解析回避処理を分析した．以降ではこれらの検体を**分析対象検体**と呼ぶ．なお，signature とは，マルウェアに見られる典型的な挙動やファイルの特徴の各々に割り当てられたラベルデータである．Cuckoo Sandbox における signature の例としては「他プロセスのメモリを操作する」「既知のパッカーでパックされている」「ファイルの存在を通じて VirtualBox の存在を検知しようとしている」などがある．分析対象検体での API コール列や signature などに関する統計情報を**表 2** に示す．Cuckoo Sandbox の解析結果では例外が擬似的な API コールとして表現されているため，本研究では，例外も API コールとみなしている．

3. 解析回避処理の全体的傾向

本研究では，Soliton Dataset 2021 で Cuckoo Sandbox が検出した signature から主に解析回避処理に関するものを選択し，解析結果中のそれらに関係する部分を精査した．

表 3　対象とする解析回避処理と検出検体数

Table 3　Target evasive operations and numbers of detected samples.

	Name	Category	Description	Severity	#samples
1	checks_debugger	anti-debug	Checks if process is being debugged by a debugger	1	145 (41.7%)
2	pe_features	packer	The executable contains unknown PE section names indicative of a packer (could be a false positive)	1	143 (41.1%)
3	packer_entropy	packer	The binary likely contains encrypted or compressed data indicative of a packer	2	120 (34.5%)
4	antivm_queries_computername	AntiVM	Queries for the computername	1	87 (25.0%)
5	peid_packer	packer	The executable uses a known packer	1	36 (10.3%)
6	antisandbox_sleep	anti-sandbox	A process attempted to delay the analysis task.	2	33 (9.5%)
7	antisandbox_cuckoo_files	anti-sandbox	Attempts to detect Cuckoo Sandbox through the presence of a file	3	24 (6.9%)
8	stealth_hiddenfile	stealth	Attempts to modify Explorer settings to prevent hidden files from being displayed.	3	10 (2.9%)
9	stealth_hidden_extension	stealth	Attempts to modify Explorer settings to prevent file extensions from being displayed.	3	10 (2.9%)
10	antivm_network_adapters	anti-vm	Checks adapter addresses which can be used to detect virtual network interfaces	2	9 (2.6%)
11	antivm_vmware_in_instruction	anti-vm	Detects VMWare through the in instruction feature	3	7 (2.0%)
12	antiav_servicestop	anti-av	Attempts to stop active services	3	6 (1.7%)
13	packer_upx	packer	The executable is compressed using UPX	2	5 (1.4%)
14	disables_security	anti-av	Disables Windows Security features	4	4 (1.1%)
15	modify_uac_prompt	stealth	Attempts to modify UAC prompt behavior	3	4 (1.1%)
16	terminates_remote_process	persistence, stealth	Terminates another process	2	3 (0.9%)
17	packer_vmprotect	packer	The executable is likely packed with VMProtect	2	3 (0.9%)
18	antidbg_windows	anti-debug	Checks for the presence of known windows from debuggers and forensic tools	3	2 (0.6%)
19	antidbg_devices	anti-debug	Checks for the presence of known devices from debuggers and forensic tools	3	2 (0.6%)
20	antiav_detectreg	anti-av	Attempts to identify installed AV products by registry key	3	2 (0.6%)
21	bypass_firewall	bypass	Operates on local firewall's policies and settings	3	1 (0.3%)
22	antisandbox_foregroundwindows	anti-sandbox	Checks whether any human activity is being performed by constantly checking whether the foreground window changed	2	1 (0.3%)
23	antisandbox_restart	anti-sandbox	Attempts to shutdown or restart the system, generally used for bypassing sandboxing	3	1 (0.3%)

選択した signature を**表 3** に示す．以降ではこれらを**精査対象 signature** と呼ぶ．この選択は，検出用コード，検出用コード内の文字列やコメント，検出用コードが書かれたファイルの名前などの情報をもとに，著者が各解析回避処理の重要度や興味深さを見積もった結果に従って行われた．精査対象 signature 以外にも解析回避処理に関する signature は多数検出されていたが，それらは false positive が著しく多いと予想されたり，検出数が非常に少ないことから，精査対象からは除外した．なお，最新版（2021 年 8 月 15 日時点）の Cuckoo Sandbox のソースコードでは総 signature 種類数は 563 であり，精査対象 signature はその中のほんの一部である．

表中の Name，Category，Description は各 signature の検出用コード内で文字列で与えられており，それぞれ，検出しようとしている処理や特徴の名前と，それらを分類した区分名，それらの簡潔な説明である．Severity は処理や特徴の深刻さを意味する値であり，やはり検出コード内で定義されている．この数が大きい signature はより深刻な兆候を示しており，ひいては，より重要である．#samples はその signature が検出された検体の数であり，括弧内の数値は全分析対象検体中での割合である．

Signature 検出用コードのファイルには名前が `anti` で始まるものが多数ある．それらのコードが検出する signature は通常，マルウェアがデバッガや仮想マシンに対抗するための，いわゆるアンチ処理についてのものである．たとえば `antidebug_` や `antidbg_` で始まる名前のファイルにはアンチデバッグ処理の signature の検出用コードが書かれている．`anti` で始まるファイルに書かれたコードで検出

される signature は，false positive が多いと予測した一部のものを除く大半を精査対象 signature に含めている．

精査対象 signature に関する全体的な指標や特徴について述べる．まず，1 検体あたりの平均検出 signature 数が 6.4 であるのに対して，精査対象 signature の検出数は 1.6 である．また，全体の 19%となる 66 の検体では精査対象 signature が 1 つも検出されていない．したがって，少なくとも検出検体数の面では，全 signature における精査対象 signature の割合は高くはない．過去の研究で，解析回避処理は一部の検体のみが実行するものであり各検体が実行する数も大きくはない，という知見は報告されていた [11] が，その傾向は続いていると著者は考えている．

しかし，一部の検体の実行では多数の精査対象 signature が検出されている．最も多くの signature が検出されたのはハッシュ値 3299f07b... の検体であり，その数は 26，精査対象 signature に限れば 5 である．この検体はデータセットではファミリ名は `CoronaVirus` と判定されており，アンチウィルスベンダにより詳しい解析情報が提供されている [12]．最も多くの精査対象 signature が検出されたのはハッシュ値 91f6fdf9... の検体であり，検出 signature 数は 15，精査対象 signature に限れば 8 である．この検体はデータセットではファミリ名は `Lazarus` ... と判定されており，やはり詳しい解析情報が提供されている [6]．なお，精査対象 signature 数を 6 以上まで拡げると，該当検体は 3 検体あり，5 以上までだと 18 検体となる．

表 3 に示した結果は，Soliton Dataset 2020 についての過去の報告 [13] におけるそれと傾向が大きく異なる．たとえばその報告では `antisandbox_sleep` の signature は 368 検体中の 49.2%である 181 検体で検出されていたが，今回の結果では 9.5%の 33 検体とかなり少なくなっている．また，`antivm_network_adapters`，`antisandbox_foregroundwindows`，`antivm_vmware_in_instruction` についても，その報告では 26.4%，18.5%，18.2%という多くの割合の検体で検出されていたが，今回の結果ではそれらの値はそれぞれ 2.6%，0.3%，2.0%でしかない．一方で，`antisandbox_cuckoo_files` については，4 検体で 1.1%だったのに対して今回は 24 検体で 6.9%と大きく増えている．これらの大きな変化は今回のデータセットでの検体選択方式に起因すると著者は考えている．Soliton Dataset では 2020 年版も 2021 年版も，解析対象検体は無作為ではなく作成者が独自の基準で選択している．よって，Soliton Dataset の各版における検体の傾向は作成者の基準に影響されるため，それが大きく変化するのは不自然ではないと考えている．

さらに，精査対象 signature の各々について，検出の原因となった具体的な API コールや挙動を著者が見て，それらの意図や実行の理由を推測するという調査を行った．その調査で得られた，マルウェアの挙動や特徴についての全体的な印象は以下の通りである．

- 検出された signature の少なくとも半分以上は，その原因となった処理の意図を把握しにくいものである．
- 検出された signature の少なくとも半分以上は，false positive であると思われる．すなわち，マルウェアにはその signature が表す処理を行う意図がないにもかかわらず，その意図が存在する可能性を示す signature が検出されていると思われる．

これらは著者の現時点における個人的な印象であり，数値や客観的な根拠に基づく判断ではないことを注意しておく．解析スキルや見落としなどの事情で意図を推定できなかった可能性もありうる．また，一見 false positive に見える処理が実は本当に解析回避のためのものである可能性もありうる．

False positive が多いと思われる signature をサンドボックスの開発者が提供することの是非は自明ではない．用途により，それらが役立つ場合もあれば，ノイズとして混乱を誘発する場合もある．サンドボックスは精度に関わらずできるだけ多くの情報を提供すべきであり，各解析者が自身の基準でそれらをフィルタリングすべきであるという考え方もある．上記の調査を行っての著者の提案は，signature の開発者はその signature が false positive である可能性の目安となる情報もあらかじめ付与しておくというものである．たとえば 3 段階や 5 段階のレベルからなる可能性の情報を付与しておけば，signature が検出された原因を解析者がより高い精度で推測できると考える．現在でも可能性の情報はマルウェアや OS について詳しい解析者が検出用コードを理解すればある程度得られるが，詳しくない解析者もいるとともに，各解析者がコードを理解するコストを払う必要がある．

4. 個々の解析回避処理

表 3 に列挙した各 signature を精査して得た情報を，それらの特徴に基づいて著者が分類したグループごとに説明する．具体的には，各 signature が検出される原因となった API コール列や，それらの API コール列を呼び出す意図に関する考察などについて述べる．以降では各 signature を単にその signature の名前で呼ぶ．

4.1 解析環境についての情報収集

解析環境についての情報を収集している可能性があることを示す signature として，`checks_debugger`，`antivm_queries_computername`，`antisandbox_cuckoo_files`，`stealth_hiddenfile`，`stealth_hidden_extension`，`antivm_network_adapters`，`antivm_vmware_in_instruction`，`antidbg_windows`，`antidbg_devices`，`antiav_detectreg`，`antisandbox_foregroundwindows` などが検出され

ている．以下でそれらの一部を説明する．

checks_debugger は，デバッガの存在を検査する IsDebuggerPresent 関数か CheckRemoteDebuggerPresent 関数を呼び出すだけで検出される．また，antivm_queries_computername は，善良プロセスリストにない名前のプロセスが，コンピュータ名を取得するための GetComputerName で始まる API 関数を呼び出すだけで検出される．antivm_network_adapters は，善良プロセスリストにない名前のプロセスが，ネットワークアダプタの情報を得るための GetAdaptersAddresses 関数を呼び出すだけで検出される．一般にそれらの情報は様々な目的で取得され，これらの signature には false positive が多いと予想されるため，詳しい分析はしていない．

antisandbox_cuckoo_files は通常の設定で Cuckoo Sandbox のゲスト OS に配置される agent.py や cuckoo.dll などのファイルがゲスト OS のルートディレクトリにあれば検出される．この signature についても false positive が多かったと著者は考えている．たとえばある検体ではデスクトップや Downloads フォルダにあるファイルを次々と暗号化しているが，Downloads に置かれていた agent.py も暗号化されて agent.py.encrypted という名前のファイルで保存されている．この検体では agent.py へのアクセスは Cuckoo Sandbox を検出するためではなく，単なる暗号化のためであると推測する．

antivm_vmware_in_instruction は，マルウェアが in 命令の実行で例外を発生させ，その際どれかのレジスタに 0x564d5868 という VMware ハイパバイザのマジックナンバが入っていれば検出される．この signature が検出されるには，当該マジックナンバを使用するという意図がほぼ不可欠であり，この signature には false positive は少ないと著者は考えている．in 命令は 1 バイト命令であるため，マルウェアがバグなどにより意図に反して in 命令を実行することは稀ではないが，その実行時に当該マジックナンバが偶然レジスタに入っている可能性は極めて低いと著者は考えている．結局，この signature が検出された検体ではマルウェアが意図的にハイパバイザ検出を行った可能性が高いと考えている．

antidbg_windows と antidbg_devices は，既知のデバッガやフォレンジックツールの存在を検査する処理を示す signature である．これらはそれぞれ，それらの解析ツールのものである可能性が高い文字列を含むウィンドウ名やクラス名を引数として API コールを実行した場合と，それらの解析ツールが用いる可能性が高いデバイスの文字列を含むファイルパスをアクセスした場合に検出される．ウィンドウの文字列としてはたとえば ProcessHacker，FilemonClass，wireshark.exe がある．デバイスの文字列としてはたとえば SIWDEBUG，NTICE，REGSYS がある．データセットでは 2 検体で両方の signature が検出されている．それらの実行では，ウィンドウ名に OLLYDBG や PROCMON_WINDOW_CLASS など，ファイルパスに \\??\\SIWVID や \\??\\SICE などの，検出の原因となる文字列が使用されている．

antiav_detectreg は，アンチウィルス製品が使用すると推測されるレジストリキーをアクセスすると検出される．検出用コードに列挙されたキーの文字列のパターンには，製品名がそのまま含まれているものが多くあり，たとえば ESET, Kaspersky, TrendMicro などの文字列を含んでいる．データセットではこの signature は 2 検体で検出されており，片方は AvpSecurity，もう片方は MpEngine というキー名のレジストリへのアクセスによるものである．前者の検体は，...\Windows\CurrentVersion\Run\AvpSecurity というキー名のレジストリをオープンしようとするが，存在しないという結果を受け取り，自身で同名のレジストリキーを作成している．後者の検体は，...\Windows Defender\MpEngine，および，Microsoft Antimalware\MpEngine というキー名のレジストリをオープンしようとするがどちらも存在しないという結果を受けとり，以降は何もせず実行を終えている．

この signature に該当するキーは，全レジストリのスキャンでアクセスされることもあるが，目当てのキー名やそのキーがある場所に意図的にピンポイントでアクセスされることも多いと著者は考えている．よって，この signature については，false positive による検出かどうか，当該キーへのアクセスの目的は何かなどについてよく調査する必要があると考えている．たとえば上記の前者の検体は，RegSetValueExA 関数によって，HKEY_LOCAL_MACHINE\\ ... \\CurrentVersion\\Run\\AvpSecurity というキーに自身の実行ファイルのパスをセットし，自身が自動的に起動されるようにしている．すなわち signature が元々意図しているアンチウィルス製品の特定ではなく自身の永続化の目的でこの API 関数を呼び出していると推測される．

antisandbox_foregroundwindows は，フォアグラウンドウィンドウのハンドルを取得する GetForegroundWindow 関数と，指定の時間だけスリープする NtDelayExecution 関数を両方とも 101 回以上呼び出すと検出され，人間による操作の存在を推定する処理を示唆する．今回のデータセットではこの signature は 1 検体のみで検出されている．この検体では GetForegroundWindow 関数を 7637 回，NtDelayExecution 関数を 22505 回呼び出しており，しきい値の 101 回をゆうに超えている．それらの多数回の実行が行われているのは以下のような API コール列においてである．NtDelayExecution 関数で短いスリープを入れつつ，GetForegroundWindow 関数で最前面のウィンドウのハンドルを取得する操作をマルウェアは繰り返している．

```
...
NtDelayExecution(skipped=0, milliseconds=10) = 0
```

```
GetForegroundWindow() = 131338
NtDelayExecution(skipped=0, milliseconds=10) = 0
NtDelayExecution(skipped=0, milliseconds=10) = 0
NtDelayExecution(skipped=0, milliseconds=10) = 0
GetForegroundWindow() = 131338
NtDelayExecution(skipped=0, milliseconds=10) = 0
NtDelayExecution(skipped=0, milliseconds=10) = 0
...
```

また，別の場所では以下のように FindWindowW，GetCursorPos，GetKeyState などの他の API 関数の呼び出しも混ぜている．例に示した FindWindowW 関数の呼び出しはタスクバーのハンドルを取得するものであり，GetCursorPos 関数と GetKeyState 関数の呼び出しはそれぞれマウスカーソルの位置とキー入力を取得するものである．なお，この検体はこれらのような API コールを繰り返したまま，他の処理に移ることなくタイムアウトで実行を終えている．これらの API コールは人間の操作の存在を検査する意図で呼び出されている可能性もあるが，そう特定することは難しく，別の意図で呼び出されている可能性もあると著者は考えている．

```
NtDelayExecution(skipped=0, milliseconds=10) = 0
FindWindowW(class_name="Shell_TrayWnd",
            window_name="") = 65686
...
GetForegroundWindow() = 131338
NtDelayExecution(skipped=0, milliseconds=10) = 0
GetCursorPos(y=261, x=1748) = 1
GetKeyState(key_code=1) = 1
GetKeyState(key_code=2) = 0
...
GetKeyState(key_code=18) = 0
NtDelayExecution(skipped=0, milliseconds=10) = 0
...
```

4.2 解析環境に対する攻撃や設定変更

解析環境に対して攻撃や設定変更を行って影響を与えることを試みるマルウェアも存在した．これらの処理に相当する signature は antiav_servicestop，disables_security，modify_uac_prompt，terminates_remote_process，bypass_firewall，antisandbox_restart である．

antiav_servicestop は，ControlService 系の API 関数を引数 SERVICE_CONTROL_STOP とともに呼び出してサービスを停止させようとすると検出される．

disables_security は，Windows のファイアウォール，UAC，Defender などのセキュリティ機構のレジストリを書き換えようとすると検出される．データセットではこの signature は 4 検体で検出されているが，いずれも EnableLUA という UAC のためのレジストリキーに 0 をセットしようとしている．

modify_uac_prompt は，UAC のプロンプトに関するレジストリを書き換えようとすると検出される．データセットではこの signature は 4 検体で検出されているが，いずれも ConsentPromptBehaviorAdmin という UAC のためのレジストリキーに 0 をセットしようとしている．

terminates_remote_process は NtTerminateProcess 関数を 0 と −1 以外の引数で呼び出してプロセスを終了させようとすると検出される．データセットではこの signature は 3 検体で検出されているが，そのうち 2 検体は自分自身や自身の検体中の他プロセスに対してのみ呼び出している．残りの 1 検体では，自分自身と自分の親プロセスに対して呼び出している．すなわち，今回のデータセットでのこの signature の検出はどれも，セキュリティシステムや善良なプログラムのプロセスを終了させるような，マルウェアの外への攻撃についてのものではない．

bypass_firewall は，FirewallPolicy という場所のレジストリキーにアクセスするか，または，netsh と advfirewall の両方の文字列を含むコマンドを実行すると，検出される．データセットではこの signature は 1 検体のみで検出されているが，実際この検体は cmd.exe を起動して netsh advfirewall で始まるコマンドなどを実行し，5650 番ポートを開けるなどの操作を試みている．この signature の検出が false positive だった可能性は低く，この検体は意図的にファイアウォールの操作を試みたと考えている．

antisandbox_restart は，シャットダウンや再起動に関する API 関数を呼び出すと検出される．データセットではこの signature は前述の 3299f07b... というハッシュ値の 1 検体のみで検出されている．この検体は ExitWindowsEx というシャットダウンや再起動を行うための API 関数を呼び出している．

4.3 長時間のスリープ

antisandbox_sleep は長時間のスリープを示唆する signature である．この signature は解析対象プログラムが累計 120 秒以上のスリープを試みると検出される．実際には Cuckoo Sandbox がスリープをスキップすることがあるので，実際のスリープ時間はそれより短くなる可能性がある．Soliton Dataset 2021 では，スリープを試みた累積時間は最短で 120 秒であり最長は 2728173 秒だった．その最長のスリープを試みた 1 検体を除く 33 検体では，累積スリープ試行時間はどれも 120 秒から 680 秒の間に収まっていた．すなわち，長時間のスリープを実行して解析を妨害するようなマルウェアはほぼなかった．

累積 2728173 秒のスリープを試みた検体について詳しく説明する．この検体は，6c7f4343... というハッシュ値を持っている．データセットではファミリー名は Buer と判断されており，アンチウィルスベンダにより詳しい解析情報も提供されている [8]．マルウェア本体のプロセスが 1 つ

の子プロセスを作り，その子プロセスが powershell.exe を実行する．このプロセスが長時間のスリープを試みている．その子プロセスは最初のスレッドも含めると合計 23 のスレッドを生成する．マルウェア本体のプロセスはスレッドは生成しない．マルウェア本体のプロセスは 5277 回の API コールを試み．powershell.exe のプロセスは全スレッド合わせて 26383 回の API コールを試みる．また，マルウェア本体のプロセスは 2444 回の NtDelayExecution の実行を試み，powershell.exe のプロセスは全スレッド合わせて 5084 回の NtDelayExecution の実行を試みる．

マルウェア本体のプロセスによるスリープ時間のほとんどは数十ミリ秒であるが，最後に 1 回だけ 60 秒のスリープを試みる．powershell.exe のプロセスによるスリープ時間は，1 回だけが 2728163227 ミリ秒であり，最後に 1 回だけ 10 秒のスリープを試みる．残りの 5082 回の API コールで与えている時間は 0 ミリ秒である．他にスリープは試行していない．2728163227 ミリ秒のスリープを試みた API コールはエラーコード 192 (STATUS_USER_APC) を返している．このエラーコードは所定の時間より前にユーザモードの APC（非同期呼び出し）が行われたことを意味し，すなわちこのスリープは割り込まれて中断している．

マルウェアが 2728163227 ミリ秒のスリープを試みる事象は著者の 2018 年の文献 [10] で報告されている．API コール列を見る限りでは，このスリープは解析回避のためではなく，スレッド間の同期のために用いられている可能性が高いと考えている．実際，このスリープを実行したスレッドは他スレッドにより 4.8 秒後に起こされており，この長時間のスリープは事実上，一時的なサスペンド処理になっている．

4.4 パッカー

pe_features，packer_entropy，peid_packer，packer_upx，packer_vmprotect はパッカーに関する signature である．以下でそれらの一部を説明する．

peid_packer は，Cuckoo Sandbox が，解析対象プログラムのファイルをパックしたパッカーを Cuckoo Sandbox が外部ツール PEiD によって推定できた場合に検出される．パッカー名は解析結果に含められる．packer_upx は，パッカー UPX の使用を示唆する signature であり，プログラムの PE ファイルのセクション名に UPX で始まるものがある場合に検出される．packer_vmprotect は，パッカー VMProtect の使用を示唆する signature であり，プログラムの PE ファイルのセクション名に.vmp で始まるものがある場合に検出される．

peid_packer に関して，推定されるパッカーとして Soliton Dataset 2021 に記録されているものを以下に列挙する．括弧内の数は検体数である．

BobSoft Mini Delphi (13)，Armadillo v1.71 (12)，Microsoft Visual C++ V8.0 (Debug) (11)，PureBasic 4.x (7)，PureBasic DLL (3)，UPX 2.90 [LZMA] (3)

UPX や Armadillo のような著名なパッカーの名前もあるが，言語処理系の名前と思われるものもパッカーとして記録されている．

packer_upx が検出された検体はどれも UPX0 と UPX1 という名前のセクションを含んでいた．いくつかの検体はさらに UPX2 という名前のセクションも含んでいた．packer_vmprotect が検出された検体はどれも.vmp0 と.vmp1 という名前のセクションを含んでいた．

これらの signature は検出技術の面では単純，もしくは外部のツール任せであり推定方法がブラックボックスであるため，これ以上深く調査はしていない．

5. 関連研究

著者の過去の研究 [11, 14] では，FFRI Dataset 2016 の Cuckoo Sandbox による動的解析結果を分析し，マルウェアの主に解析回避処理についての傾向を明らかにしている．それらの研究でも本研究と同様に主に API コール列と signature の情報を分析している．本研究では同様の分析を，より新しいマルウェアを対象に含む Soliton Dataset 2021 に対して行ったものである．

Branco らによる研究 [2] では，解析回避処理を含む様々なマルウェアの特徴的な処理の傾向を明らかにしている．たとえば，たとえば，個々の解析回避処理が観測されるマルウェアの割合を示している．この研究が発表された時期からは約 10 年が経過しているため，最新のマルウェアにおける傾向は不明であり，本研究のような最新のデータを用いた調査が必要である．

Chen らによる研究 [5] では，様々なファミリーのマルウェアについて，どの程度の割合の検体がアンチデバッグとアンチ VM 処理を実行するかを調査した結果が報告されている．本論文で扱った解析回避処理のいくつかも彼らの研究で調査されている．彼らの研究では本研究とは異なり，静的解析で得られた情報のみを利用している．また，本研究で示したような，マルウェアが具体的に実行する処理の詳細についての情報を，ほとんど提供していない．

Chailytko らによる論文 [4] では，マルウェアが Cuckoo Sandbox による解析を回避するための処理と，そのような処理に対抗するための解析者側の対策について詳しく説明されている．本研究で対象とした解析回避処理の多く（たとえばスリープによる解析の長時間化やゲスト OS 内の Cuckoo Sandbox 関連ファイルの存在検査）は，彼らの論文でも議論されている．彼らの論文ではマルウェアによる解析回避処理とその対策処理が説明されているだけであり，現実のマルウェアを用いた調査の結果は示されていない．よって，本研究で示したような解析回避処理の利用の

実態を知ることはできない．

文献 [7] でも，マルウェアが Cuckoo Sandbox の存在を検出するための方法と，それらへの対策処理が詳しく述べられている．しかしやはり，現実のマルウェアを用いた調査の結果は示されておらず，マルウェアによる Cuckoo Sandbox の検出に関する実態は明らかにされていない．

6. おわりに

Soliton Dataset 2021 から得られるマルウェアの解析回避処理についての情報を分析し，その全体的傾向や個別の処理についての実態を明らかにした．分析で得られた知見として，解析回避処理は頻繁に観測される処理ではないことや，サンドボックスにより検出された解析回避処理には false positive が多い可能性があることなどを述べた．一方，解析回避処理である可能性が高い処理が確かに実行されていることも実例とともに示し，注意を喚起するとともに，対策を検討する助けになる情報を提供した．

今後の課題としては，第一に，サンドボックスが検出した signature の原因となった API コール列や挙動のうち，意図が依然として不明であるものが多くあるので，それらの意図を推定することが必要である．さらに，その推定の妥当性を裏付ける証拠を効率的に集める方法や，妥当性を評価するための方法も開発する必要がある．第二に，今回使用した Cuckoo Sandbox による signature からは把握できなかった解析回避処理も多く存在すると思われるため，そのような解析回避処理を発見するとともに，それらを高い精度で効率的に検出する方法も構築する必要がある．

謝辞 Soliton Dataset 2021 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 20K11741 の助成を受けている．

参考文献

[1] Afianian, A., Niksefat, S., Sadeghiyan, B. and Baptiste, D.: Malware dynamic analysis evasion techniques: A survey, *ACM Computing Surveys*, Vol. 52, No. 6 (2019).

[2] Branco, R. R., Barbosa, G. N. and Neto, P. D.: Scientific but Not Academical Overview of Malware Anti-Debugging, Anti-Disassembly and Anti-VM Technologies, Black Hat USA 2012 (2012).

[3] Bulazel, A. and Yener, B.: A Survey On Automated Dynamic Malware Analysis Evasion and Counter-Evasion: PC, Mobile, and Web, *Proceedings of the 1st Reversing and Offensive-oriented Trends Symposium* (2017).

[4] Chailytko, A. and Skuratovich, S.: Defeating Sandbox Evasion: How to Increase Successful Emulation Rate in your Virtualized Environment, *Proceedings of the 26th Virus Bulletin Conference* (2016).

[5] Chen, P., Huygens, C., Desmet, L. and Joosen, W.: Advanced or Not? A Comparative Study of the Use of Anti-debugging and Anti-VM Techniques in Generic and Targeted Malware, *Proceedings of the 31st IFIP International Conference on ICT Systems Security and Privacy Protection*, pp. 323–336 (2016).

[6] Cherepanov, A. and Kálnai, P.: ESET: Lazarus supply - chain attack in South Korea, `https://www.welivesecurity.com/2020/11/16/lazarus-supply-chain-attack-south-korea/` (2020).

[7] Ferrand, O.: How to detect the Cuckoo Sandbox and to Strengthen it?, *Journal of Computer Virology and Hacking Techniques*, Vol. 11, No. 1, pp. 51–58 (2015).

[8] Gallagher, S.: Sophos Blog: Hacks for sale: inside the Buer Loader malware-as-a-service, `https://news.sophos.com/en-us/2020/10/28/hacks-for-sale-inside-the-buer-loader-malware-as-a-service/` (2020).

[9] Or-Meir, O., Nissim, N., Elovici, Y. and Rokach, L.: Dynamic Malware Analysis in the Modern Era – A State of the Art Survey, *ACM Computing Surveys*, Vol. 52, No. 5 (2019).

[10] Oyama, Y.: Investigation of the Diverse Sleep Behavior of Malware, *Journal of Information Processing*, Vol. 26, pp. 461–476 (2018).

[11] Oyama, Y.: 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).

[12] P, S., Karnik, A. and Grindstaff, L.: McAfee Labs Blog: COVID-19 – Malware Makes Hay During a Pandemic, `https://www.mcafee.com/blogs/other-blogs/mcafee-labs/covid-19-malware-makes-hay-during-a-pandemic/` (2020).

[13] 玉林亜喬, 大山恵弘: 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，コンピュータセキュリティシンポジウム 2020 論文集，pp. 912–919 (2020).

[14] 大山恵弘: マルウェアによる対仮想化処理の傾向についての分析，コンピュータセキュリティシンポジウム 2016 論文集，pp. 534–541 (2016).

[15] 尾曲晃忠: Soliton Dataset 2021, MWS 2021 プレミーティング (2021).

[16] 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〜，情報処理学会研究報告 情報基礎とアクセス技術, Vol. 2020-IFAT-139, No. 8 (2020).