Vol.2017-DPS-170 No.15
Vol.2017-CSEC-76 No.15
2017/3/2

マルウェアのスリープ挙動の多様性に関する予備調査

大山 恵弘[1,a)]

概要：マルウェアは様々な目的でスリープを実行する．その目的には，資源消費速度の制御，攻撃開始までの潜伏，動的解析の妨害が含まれる．著者は，スリープ挙動がマルウェア間でどの程度異なるかを理解し，その知見をマルウェアの検知や分類に応用することを目指している．本稿では，長時間のスリープを実行するマルウェアの動的解析ログからスリープ挙動に関する特徴を抽出し，その特徴に基いてマルウェアを分類した予備調査について報告する．

Preliminary Investigation on Diversity of Sleep Behavior of Malware

Yoshihiro Oyama[1,a)]

Abstract: Malware sleeps for various purposes, which include controlling resource consumption speeds, staying dormant until the time of attacks, and thwarting dynamic analysis. The author aims to understand the diversity of sleep behavior between malware programs and then apply the finding to detection and classification of malware. This paper reports preliminary investigation in which the author extracted features of sleep behavior from a dynamic analysis log of malware programs that execute long sleeps, and classified the programs based on the features.

1. はじめに

マルウェアは様々な目的でスリープを実行する．マルウェアは，イベント待ちなどの通常のプログラムにも見られる目的に加えて，セキュリティシステムに対抗するためにスリープを実行することもある．例えば，長時間のスリープを実行することにより，早期の検知を回避したり，動的解析をタイムアウトさせたりすることが可能である．また，スリープと現在時刻の取得を組み合わせることにより，サンドボックスなどの解析システムの有無を推定することが可能である．最近では多くのマルウェアが自身の解析を回避するための処理（耐解析処理）を実行することが知られている [1,2,6,10,13]．

マルウェアの中には，スリープに関して特徴的な挙動を示すものも多いと予想されるが，実際のマルウェアがどのようなスリープ挙動を示すのかについては研究が乏しく，十分に理解されていない．スリープ挙動に関してマルウェア間で多様性があるのかどうかについても，明らかではない．また，個々のスリープの背後にあるマルウェアの意図を正確に推定する技術も，未だ途上段階にある．

マルウェアのスリープ挙動をより深く理解できれば，耐解析処理などの最近のマルウェアによる洗練された処理をより深く理解できる可能性がある．さらに，スリープ挙動をマルウェアの特徴として検知や分類に活用することに道が開かれる可能性もある．実際，現状の技術では，耐解析処理を実行するような洗練されたマルウェアの検知や分類の精度は十分に高くはなく，精度を高める方法が求められている．検知や分類のための手がかりを多くもたらすのは，ファイル I/O やネットワーク通信などの従来から豊富な知見が蓄積されてきた挙動ではあると思われるが，それはスリープ挙動を理解しなくて良いということを意味せず，マルウェアの様々な挙動に関する知見を組み合わせて用いるのが有効だと考える．

本稿では，最近収集されたマルウェアのスリープ挙動に関する傾向を示すとともに，スリープ挙動の特徴に基いてマルウェアを分類した調査の結果について報告する．本研究ではまず，Windows OS 向けマルウェアの動的解析ログ

[1] 筑波大学
University of Tsukuba, Tsukuba, Ibaraki 305–8577, Japan
[a)] oyama@cc.tsukuba.ac.jp

のデータセットである FFRI Dataset 2016 [14] に含まれる API コールログからスリープに関する部分を抽出した．その後，特に重要であると予想される特徴の集合を作成し，各マルウェアについてそれらの特徴の値を求めた．さらに，それらの特徴値を決定木生成アルゴリズム C5.0 [12] に与えてマルウェアを分類するための決定木を作成し，その決定木が正しく分類できるマルウェアの割合および優先的に用いる特徴を調査した．

マルウェアが実行する耐解析処理の傾向を定量的に調査した研究 [1, 2, 10, 13] は過去に存在するが，スリープに焦点を絞ってその挙動の傾向を明らかにした研究は存在しない．さらに，スリープ挙動を利用してマルウェアを分類した研究も存在しない．本研究の新規性および貢献はそれらを最初に行った点にある．

2. FFRI Dataset

FFRI Dataset [14] は FFRI 社が Cuckoo Sandbox [4] を用いて取得した，マルウェアの動的解析ログのデータセットである．FFRI Dataset として FFRI Dataset 2013 から 2016 までの 4 セットが提供されているが，本研究では FFRI Dataset 2016 を使用した．このデータセットには，マルウェアを Windows 10 (x64) 上で実行して取得したものと，Windows 8.1 (x64) 上で実行して取得したものが含まれるが，本研究では Windows 10 のものを使用した．このデータセットにはマルウェア 8243 検体を解析したログが含まれている．これらのマルウェアは 2016 年 1 月から 3 月にかけて FFRI 社が収集したものであり，どれも，10 社以上のアンチマルウェア製品によってマルウェアと判定されている．各検体は最大 90 秒間実行され，90 秒経過時に強制終了されている．

FFRI Dataset にはマルウェアによる API 呼び出し列や検出された *signature* の情報が含まれている．Signature とは Cuckoo Sandbox が検出を試みる典型的なマルウェアの特徴である．Signature の例には，HTTP リクエストの送信，コードインジェクション，実行ファイルの生成がある．

3. スリープ

3.1 スリープに用いられる API 関数

スリープに用いられる API 関数のうち，FFRI Dataset 2016 に記録されているものは `NtDelayExecution` である．より高レベルな API 関数として `Sleep` があるが，`Sleep` を呼び出しても，最終的には `NtDelayExecution` が実行される．Cuckoo Sandbox は `NtDelayExecution` の引数をログに記録する．記録される引数の 1 つはスリープ時間を表し，その単位はミリ秒である．`NtDelayExecution` の引数には 0 ミリ秒を与えることもでき，そのような呼び出しはスレッドの自発的なコンテキストスイッチに利用される．

3.2 スリープの目的

マルウェアがスリープを実行する主な目的には以下のものがある．後半の 3 つがマルウェア特有である．

資源消費速度の制御 定期的にスリープを実行することによって CPU 負荷やネットワーク負荷の増加を小さく抑える．この処理により，負荷上昇による検知やサービス拒否を回避したり，別のプロセスやスレッドに多くの資源を渡したりすることができる．データセットには，DNS での複数の名前解決を 1 秒ずつ間隔を空けて実行する API 呼び出し列が存在する．また，10 ミリ秒の間隔をおいて他スレッドに対する `NtGetContextThread` を繰り返し呼び出す列も存在する．この API 関数で取得した他スレッドの状態は，他スレッドの実行が望みの状態に入ったかどうかを知る目的に使っている可能性がある．

スレッドの実行制御，同期 間隔をおいてスレッドを休止，再開させることにより，スレッドの実行制御やスレッド間の同期を実現する．データセットにも，スリープを挟みながら他スレッドに対する `NtSuspendThread` と `NtResumeThread` を繰り返し呼び出す API 呼び出し列が存在する．このスリープにより，他スレッドが走れる時間を制御することができる．

潜伏 長時間のスリープを実行し，活動開始の条件が満たされるまで潜伏する．単純に長時間のスリープを実行して早期検知を回避することもあれば，他のプログラムからの活動開始の通知を待つためにスリープを実行することもある．

動的解析のタイムアウト 長時間のスリープを実行することにより動的解析をタイムアウトさせる．動的解析には数分から数十分の制限時間が付けられることが多いため，それを超える時間のスリープを実行すれば，重要な部分の解析を回避できる可能性がある．データセットにも，潜伏もしくはタイムアウトを狙ったと思われる API 呼び出し列がある．例えば，約 31 日間のスリープを実行しようとしたマルウェアが存在する．なお，Cuckoo Sandbox の anti-sleep 機構により，このスリープの実行はスキップされている．

解析システムの検出 ある時間のスリープを実行し，その時間を実際の経過時間と比較することにより，解析システムによるスリープのスキップなどの処理を検出する．

4. 調査方法

4.1 検体の選択

まず，全 8243 検体の中から，解析処理を遅らせる試みの signature が検出された 509 検体を選んだ．これらの検体は他の検体と比べて，スリープを活発に利用している可能性が高い．Cuckoo Sandbox は，マルウェアが試みたス

表 1　調査した特徴

識別番号	識別名	定義
(1)	maxdelay	スリープ時間の最大値
(2)	mindelay	スリープ時間の最小値
(3)	modedelay	スリープ時間の最頻値
(4)	avedelay	スリープ時間の平均値
(5)	ndelaykinds	スリープ時間の種類数
(6)	ncalls	`NtDelayExecution` を呼び出した回数
(7)	maxseqlen	シーケンスの長さの最大値
(8)	nseqs	シーケンスの数
(9)	napikinds	シーケンスの直前と直後に呼び出された API 関数の組の種類数
(10)	unrounded	スリープ時間の集合に半端な数（1 の位が 0 ではなく，かつ，1 ではない数）が含まれるか
(11)	progression	スリープ時間の集合に長さ 10 以上の等差数列が含まれるか
(12)	comm	シーケンスの直前か直後に通信関連 API 関数（`gethostbyname` など）が 1 回でも呼び出されたか
(13)	window	シーケンスの直前か直後にウィンドウ関連 API 関数（`GetForegroundWindow` など）が 1 回でも呼び出されたか
(14)	thread	シーケンスの直前か直後にスレッド関連 API 関数（`NtSuspendThread` など）が 1 回でも呼び出されたか
(15)	fileattr	シーケンスの直前か直後にファイル属性関連 API 関数（`GetFileAttributesW` など）が 1 回でも呼び出されたか
(16)	earlysleep	最初の 5 回の API 呼び出しに `NtDelayExecution` の呼び出しが含まれるスレッドがあるか
(17)	sleeper	全 API 呼び出し中の 95%以上が `NtDelayExecution` の呼び出しであるスレッドがあるか
(18)	sleepingend	最後の API 呼び出しが `NtDelayExecution` の呼び出しであるスレッドがあるか

リープの累積時間が 120 秒以上であるときにこの signature を検出し，ログに “`A process attempted to delay the analysis task.`” と出力する．スリープの累積時間の計算には `NtDelayExecution` の引数が用いられる．

4.2　特徴

マルウェア検体の**特徴**として**表 1** の値を求めた．これらの特徴は，著者が多くの検体の API 呼び出し列を解析する作業を通じて，マルウェア間で多様性がある可能性が高いと判断して選択した．

いくつかの特徴に対して説明を補足する．**シーケンス**とは，`NtDelayExecution` の連続する呼び出し列であるとする．シーケンスの長さとは，シーケンスに含まれる `NtDelayExecution` の呼び出しの数であるとする．単発の `NtDelayExecution` の呼び出しも，長さ 1 のシーケンスとみなす．長いシーケンスの出現には様々な原因が考えられる．例えば，解析システムによるスリープのスキップを避けるために短い時間のスリープを繰り返すことが挙げられる．Progression は 0, 3, 6, ..., 42, 45 と 3 ずつ引数を増やしながら連続して `NtDelayExecution` を呼び出すという珍しい処理を実行するマルウェアが存在したため，導入した．Earlysleep は，早い実行段階でのスリープは，そうでないスリープに比べて，耐解析や潜伏といった注意すべき目的のものである可能性がより高いと推測し，導入した．Sleeper は，ほとんどスリープしか実行しないスレッドの存在はマルウェアによる何らかの目的を暗示すると推測し，導入した．Sleepingend は，解析のタイムアウト時にスリープを実行していたスレッドの存在は，タイムアウトを狙う耐解析処理をマルウェアが実行していた可能性を高めると推測し，導入した．Napikinds，comm，window，thread，fileattr は，厳密な意味でスリープ挙動の特徴であるかどうかは微妙であるが，マルウェアがスリープを実行する目的を抽出できる可能性があると判断し，導入した．

4.3　分類

複数の特徴を組み合わせてマルウェアを分類する調査を行った．その調査では，アンチマルウェア製品により判定されたマルウェア名（ラベリングの結果）を教師データとして利用した．データセットには多くの製品によるマルウェア名が含まれるが，本研究では Microsoft のそれを用いた．以降では，Microsoft の製品により判定されたマルウェア名の同一性による分類を **Microsoft 分類**と呼ぶ．

調査した検体集合の中には，Microsoft のマルウェア命名規則 [9] で同じ family を意味する名前が付いている検体の組も多数存在する（例えば Trojan:Win32/Matsnu.M と Trojan:Win32/Matsnu.O）．この命名規則では，マルウェア名は type，platform，family，variant，information から構成される．本研究では，variant と information を除いた部分をマルウェア名として扱い，同じマルウェア名の検体を同種とみなした．

上で述べた 509 検体のうち Microsoft 分類でマルウェアと判定されなかった検体が 61 あったため，それらを除いた 448 検体を調査対象とした．検体数の分布を**表 2** に示す．

調査対象の検体を決定木生成アルゴリズム C5.0 に与えて，分類のための決定木を生成した．C5.0 は入力としてデータベクトルの集合および各データベクトルが属するクラスの情報（教師データ）を受け取ると，データベクトルをクラスに分類する決定木を情報エントロピーの概念を利

表 2　検体数の分布

マルウェア名	検体数
TrojanSpy:Win32/Ursnif	255
Trojan:Win32/Dynamer	98
Trojan:Win32/Tinba	30
Trojan:Win32/Skeeyah	16
Trojan:Win32/Matsnu	14
TrojanDownloader:Win32/Dofoil	5
Trojan:Win32/Bagsu, Trojan:Win32/Bulta	4
DDoS:Win32/Nitol, TrojanSpy:Win32/Skeeyah, TrojanDownloader:Win32/Banload, TrojanDownloader:Win32/Silcon	2
Backdoor:Win32/Dodiw, Backdoor:Win32/Fynloski, Backdoor:Win32/PcClient, PWS:Win32/Fareit, Ransom:Win32/Nymaim, Trojan:Win32/Malex, Trojan:Win32/MultiInjector, Trojan:Win32/Neurevt, Trojan:Win32/Toga, TrojanDownloader:Win32/Nymaim, TrojanSpy:Win32/Banker, VirTool:Win32/Obfuscator, Worm:Win32/Gamarue, Worm:Win32/Kasidet	1

用して生成し，出力する．連続値であるデータに対しては C5.0 が閾値を決定して離散化する．

本調査では，448 の検体の 18 の特徴値をデータベクトルの集合としてそのまま C5.0 に与えた．各データベクトルには教師データとして Microsoft 分類によるマルウェア名を付した．C5.0 アルゴリズムの実装としては，[12] で公開されている C5.0 Release 2.07 GPL Edition を用いた．

分類の評価においては，分類の構築に用いた検体とは別のテスト用の検体を使うことが望ましい．しかし，検体数が少ないため，本研究では予備調査として，分類の構築とテストの両方に全検体を与えて精度と再現率を求めた．すなわち，生成した決定木が，決定木の生成に用いた検体のうちどの程度の割合の検体を正しく（すなわち教師データと同じマルウェア名に）分類できるかを求めた．上記の C5.0 の実装では，特に指示しなくても，決定木の生成後にこの評価の結果を表示する．もし同種のマルウェアの間でスリープ挙動に関する多様性が小さく，異種のマルウェアの間でスリープ挙動に関する多様性が大きければ，この決定木で正しく分類できるマルウェアの割合は高くなると予想される．そうでないならば，その割合は低くなると予想される．ここで，精度とは，評価対象の分類で作られた検体集合において，正しい（Microsoft 分類と同じ）マルウェア名が与えられた検体の割合であるとする．再現率とは，Microsoft 分類で作られた検体集合において，評価対象の分類が正しいマルウェア名を与えた検体の割合であるとする．各検体集合ごとの検体数を合算して計算される精度と再現率を，精度と再現率のマイクロ平均と呼ぶ．各検体集合ごとの精度と再現率を検体集合の数で割って計算される値を，精度と再現率のマクロ平均と呼ぶ．本調査では精度のマイクロ平均と再現率のマイクロ平均は常に一致する．

表 3　全検体を通しての特徴値

識別名	特徴値
maxdelay	2728163227
mindelay	0
modedelay	50
avedelay	172.6
ndelaykinds	2270
ncalls	436066
maxseqlen	5632
nseqs	22189
napikinds	131

表 4　スリープ時間の出現数の上位 10 件

スリープ時間	出現数	使用する検体の数	使用する種の数
50 ミリ秒	364883	302	8
1 ミリ秒	14210	24	3
1 秒	11259	358	14
12 ミリ秒	9368	54	5
10 ミリ秒	6932	303	9
5 秒	4911	356	13
0 秒	3568	24	8
100 ミリ秒	3486	15	5
4 ミリ秒	2241	34	4
7 ミリ秒	2239	32	3

5. 調査結果

5.1 特徴値

まず，表 1 の特徴のうち，調査対象の 448 検体全てを通しての値が計算できるものの値を求めた．結果を**表 3** に示す．スリープ時間の最頻値は 50 ミリ秒，平均値は 172.6 ミリ秒である．平均値については，最大値 1 つが他の値から著しく離れていたため，それを外れ値として除外して計算した．スリープ時間の幅は 0 秒から約 31 日と幅広い．スリープの累積時間が長いマルウェアのみを対象としたにもかかわらず，数秒に渡る長時間の `NtDelayExecution` の呼び出しは支配的ではない．ただし，シーケンスを作って実質的に長時間のスリープを実現している可能性は残る．スリープ時間の種類数は 2270 と多い．シーケンスの長さの最大値は 5632 であり，非常に多くの `NtDelayExecution` を連続して呼び出しているマルウェアが存在する．

スリープ時間の分布についても調査した．調査対象の検体におけるスリープ時間の出現数の上位 10 件を**表 4** に示す．50 ミリ秒のスリープが圧倒的に多く使用されている．ただし使用する検体や種の数が最も多いのは 1 秒と 5 秒である．1 ミリ秒は比較的少数の検体や種で非常に多く使用されている．全体的には 1 ミリ秒，10 ミリ秒，100 ミリ秒などの半端ではない数の使用が多く，0 秒の使用も多い．

表 5　代表検体の特徴値

	マルウェア名	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
1	Worm:Win32/Gamarue	60000	3	3	818.3	6	236	72	164	4
2	DDoS:Win32/Nitol	30000	100	100	658.5	4	213	100	14	2
3	TrojanDownloader:Win32/Dofoil	30000	100	100	658.5	4	213	100	14	2
4	Backdoor:Win32/Dodiw	20000	300	300	619.2	3	291	1	291	5
5	Trojan:Win32/Matsnu	20000	4	12	213.2	10	703	172	22	12
6	TrojanDownloader:Win32/Silcon	11913	0	0	1616.6	85	98	41	17	3
7	Ransom:Win32/Nymaim	11781	6	216	1823.2	88	90	43	6	2
8	TrojanDownloader:Win32/Nymaim	11017	4	214	1882.2	82	87	39	9	2
9	Trojan:Win32/Malex	10000	10000	10000	10000.0	1	16	1	16	3
10	Trojan:Win32/Bagsu	10000	1	1	366.6	8	451	60	58	7
11	Trojan:Win32/Tinba	10000	1	1	155.6	8	1076	60	85	7
12	VirTool:Win32/Obfuscator	7000	500	1000	963.9	6	180	8	167	5
13	TrojanDownloader:Win32/Banload	5000	1000	1000	1440.0	2	100	63	14	4
14	Trojan:Win32/Skeeyah	5000	10	50	118.1	5	1301	1267	35	5
15	TrojanSpy:Win32/Ursnif	5000	10	50	118.0	5	1306	1268	39	5
16	Trojan:Win32/MultiInjector	5000	10	50	114.3	6	1388	1356	33	5
17	Trojan:Win32/Bulta	5000	10	50	114.3	6	1385	1354	32	5
18	Worm:Win32/Kasidet	5000	0	5000	3207.6	18	50	16	35	5
19	Trojan:Win32/Neurevt	5000	0	5000	3051.7	18	46	16	31	5
20	Trojan:Win32/Dynamer	5000	0	50	130.5	7	1172	1132	41	6
21	TrojanSpy:Win32/Banker	4000	50	2001	1932.8	4	399	43	105	5
22	TrojanSpy:Win32/Skeeyah	2001	0	2001	1082.0	4	651	45	81	18
23	PWS:Win32/Fareit	1000	494	600	748.7	3	236	148	2	1
24	Backdoor:Win32/PcClient	1000	100	100	138.7	3	1199	815	3	2
25	Trojan:Win32/Toga	1000	0	300	313.3	3	618	3	441	4
26	Backdoor:Win32/Fynloski	500	10	200	330.1	3	694	1	694	10
中央値		5000	10	150	658.5	5.5	345	60	34	5

	マルウェア名	(10)	(11)	(12)	(13)	(14)	(15)	(16)	(17)	(18)
1	Worm:Win32/Gamarue	T	F	F	F	F	F	T	T	T
2	DDoS:Win32/Nitol	F	F	F	F	F	F	T	F	F
3	TrojanDownloader:Win32/Dofoil	F	F	F	F	F	F	F	F	F
4	Backdoor:Win32/Dodiw	F	F	T	T	F	F	T	F	F
5	Trojan:Win32/Matsnu	T	F	T	F	T	T	T	T	F
6	TrojanDownloader:Win32/Silcon	T	F	F	F	T	F	T	T	T
7	Ransom:Win32/Nymaim	T	F	F	F	T	F	T	F	F
8	TrojanDownloader:Win32/Nymaim	T	F	F	F	T	F	T	T	T
9	Trojan:Win32/Malex	F	F	F	F	T	F	F	F	F
10	Trojan:Win32/Bagsu	F	F	T	F	F	F	T	F	F
11	Trojan:Win32/Tinba	F	F	T	F	F	F	T	F	F
12	VirTool:Win32/Obfuscator	F	F	T	F	F	F	T	F	F
13	TrojanDownloader:Win32/Banload	F	F	F	F	F	F	T	F	F
14	Trojan:Win32/Skeeyah	F	F	T	F	T	T	T	T	T
15	TrojanSpy:Win32/Ursnif	F	F	T	F	T	T	T	T	T
16	Trojan:Win32/MultiInjector	F	F	T	F	T	T	T	T	T
17	Trojan:Win32/Bulta	F	F	T	F	T	T	T	T	T
18	Worm:Win32/Kasidet	T	T	F	F	T	T	T	F	F
19	Trojan:Win32/Neurevt	T	T	F	F	T	T	T	F	F
20	Trojan:Win32/Dynamer	F	F	T	F	T	T	T	T	T
21	TrojanSpy:Win32/Banker	T	F	F	T	F	F	T	T	F
22	TrojanSpy:Win32/Skeeyah	T	F	T	F	T	T	T	T	T
23	PWS:Win32/Fareit	T	F	F	F	F	F	T	T	F
24	Backdoor:Win32/PcClient	F	F	F	F	T	F	T	T	F
25	Trojan:Win32/Toga	F	F	T	F	T	F	T	F	F
26	Backdoor:Win32/Fynloski	F	F	T	T	F	F	T	F	F
T と F の数		10/16	2/24	13/13	3/23	15/11	9/17	24/2	13/13	9/17

次に，Microsoft 分類で定まる種の集合の各々から，その種を代表する 1 検体（**代表検体**）をランダムに選び，代表検体の特徴値を求めた．検体は 26 の種に分類されたため，代表検体の数も 26 となる．結果を**表 5** に示す．一番上の

行は表 1 の特徴の識別番号を表し，一番左の列は検体の番号を表す．表中の T は真，F は偽を表す．スリープ挙動はマルウェア間で極めて多様であることがわかる．数値の特徴値についても真偽値の特徴値についても値が大きくばらついている．同時に，スリープ挙動だけからは区別が難しいと予想される検体の組があることもわかる．例えば，検体 14 から検体 17 までの 4 検体や，検体 18 と検体 19 の 2 検体では，全ての特徴値が近いか同じである．

5.2 C5.0 による分類

検体を分類する決定木を C5.0 を用いて生成した．**図 1** にその決定木を示す．不等号や等号の式は分岐の条件である．この条件に従って，根から葉へと枝を辿って検体を分類する．葉には，そこに分類された集合に付されるマルウェア名が書かれている．マルウェア名の後ろの括弧内の左の数は，その葉（検体集合）に分類される検体の数であり，右の数は，そのうちでその葉への分類（その葉のマルウェア名のラベリング）が誤りである検体の数である．

この決定木は，26 種 448 検体のデータのうち 9 種 380 検体に対して正しいマルウェア名を与えるため，精度のマイクロ平均は 84.8%となる．この決定木は対応する葉を持たない $26 - 9 = 17$ 種のマルウェアに検体を分類することはない．検体数が最多である TrojanSpy:Win32/Ursnif は 448 検体中 293 検体を占め，このマルウェア名が付いた葉における精度は 87.0%と高い．この精度が全体の精度を押し上げている．

種の間での検体数の偏りが大きいため，全検体を通しての正しい分類の割合だけではなく，各検体集合における正しい分類の割合にも注目する必要がある．精度と再現率のマクロ平均がその目安を与える．それらの値はそれぞれ 70.0% と 30.1%であり，マイクロ平均に比べて低い．特に再現率が非常に低いのは，この決定木は検体数が少ない種の多くに対して葉を提供しておらず，それらの検体に正しいマルウェア名を与えられていないことが原因である．

この決定木による分類では 6 種類の特徴が用いられる．高頻度で用いられる特徴は，検体数の順に thread，napikinds，mindelay であり，それぞれ，100%，91%，87%の検体で利用される．C5.0 ではこれらの特徴が分類に特に有効であるとみなされている．以降は ndelaykinds が 13%，sleeper が 6%，earlysleep が 2%と，上位からは隔たりがある．

分類の誤りには様々な原因が考えられ，現時点では主要な原因を断定することはできない．ただ，推測される原因の一つは，Microsoft 分類において同種（同 family）とされる検体集合に，スリープ挙動が互いに大きく異なる検体が含まれていることだと考えている．その例として，Trojan:Win32/Dynamer の 98 検体中の 10 検体の特徴値を**表 6** に示す．識別番号が 11 と 13 の特徴値は全検体で F だったので表からは省略した．なお，この 98 検体は全て，

```
thread = true:
:...mindelay <= 7:
:   :...napikinds <= 8: TrojanWin32Dynamer (71/10)
:   :   napikinds > 8:
:   :   :...mindelay <= 1:
:   :         TrojanSpyWin32Skeeyah (3/1)
:   :       mindelay > 1: TrojanWin32Matsnu (17/3)
:   mindelay > 7:
:   :...napikinds <= 3:
:         BackdoorWin32PcClient (2/1)
:       napikinds > 3:
:       :...napikinds <= 6:
:             TrojanSpyWin32Ursnif (293/38)
:           napikinds > 6: TrojanWin32Dynamer (2)
thread = false:
:...ndelaykinds > 6: TrojanWin32Tinba (34/4)
    ndelaykinds <= 6:
    :...sleeper = true: TrojanWin32Skeeyah (5/3)
        sleeper = false:
        :...napikinds > 2: TrojanWin32Dynamer (11/5)
            napikinds <= 2:
            :...earlysleep = true:
                  DDoSWin32Nitol (4/2)
                earlysleep = false:
                  TrojanDownloaderWin32Dofoil (6/1)
```

図 1 C5.0 が出力した決定木（一部フォーマットを変更）

元々のマルウェア名が命名規則の information の部分まで一致している．表に示した検体のスリープ挙動は大きく異なっている．また，これらの検体は，アクセスするファイル，レジストリ，IP アドレスについても大きく異なっており，例えばこれらの検体中の 2 検体以上が共通に読むレジストリは 1 つも存在しない．このような検体を同種とみなして良いかどうかについては議論の余地が残る．同様に，検体数の最も多い TrojanSpy:Win32/Ursnif についても，スリープ挙動が大きく異なる検体が多く含まれており，それが精度の低下をもたらしている．

5.3 特徴の全組み合わせによる分類

調査対象の検体数や特徴の数は少ないため，特徴の最適な組み合わせを全数探索によって求めることは現実的である．そこで，全数探索で決定した組み合わせを用いた分類も行い，C5.0 による分類と比較した．数値の特徴値については，著者の判断により，以下のように離散化した．

(1) スリープ時間に関する特徴値：(a) 60 秒以上，(b) 60 秒未満，5 秒より大きい，(c) 5 秒，(d) 5 秒未満，1 秒以上，(e) 1 秒未満，0 秒より大きい，(f) 0 秒
(2) 呼び出した回数や種類数などの個数に関する特徴値：(a) 1000 以上，(b) 1000 未満，100 以上，(c) 100 未満，10 以上，(d) 10 未満，2 以上，(e) 1

1 秒や 5 秒という閾値を導入した理由は，スリープ時間としてそれらの値が頻繁に用いられるからである．

N を変化させながら，N 種類の特徴の全組み合わせを

表 6　Trojan:Win32/Dynamer と判定された一部のマルウェアの一部の特徴値

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(12)	(14)	(15)	(16)	(17)	(18)
3000	0	0	119.3	3	1433	4	590	5	F	T	T	F	T	F	F
5000	0	50	130.5	7	1172	1132	41	6	F	T	T	T	T	T	T
5000	50	50	128.5	4	1485	1	1485	5	F	T	F	F	F	F	F
5000	1000	1000	1740.7	2	108	1	108	8	F	T	T	F	T	F	F
14835	6	12	1011.7	91	217	124	12	3	T	T	T	F	T	F	F
20000	4	100	267.3	11	601	135	21	13	T	T	T	T	T	T	F
30000	0	0	1334.6	88	147	41	62	4	T	F	T	F	T	T	T
30000	100	100	670.9	5	215	100	16	3	F	F	F	F	F	F	F
30000	2000	30000	24400.0	2	5	1	5	3	F	F	F	T	F	F	F
2728163227	0	20	209858879.0	4	13	9	4	3	T	F	T	F	T	F	F

表 7　所定の種類数の特徴を組み合わせたときに得られる精度のマイクロ平均の最大値および
その値を与えるときの精度と再現率のマクロ平均

組み合わせる特徴の種類数 (N)	1	2	3	4	5	6	7	8	9	10	11	12
精度のマイクロ平均 (%)	72.8	79.9	83.3	85.5	86.6	87.1	87.5	87.7	87.9	87.9	87.9	87.9
精度のマクロ平均 (%)	58.6	70.2	78.8	80.2	88.4	91.3	91.2	92.8	92.9	92.9	92.9	92.9
再現率のマクロ平均 (%)	14.4	18.2	33.5	64.3	66.0	66.1	69.9	70.0	70.0	70.0	70.0	70.0

作成した．各組み合わせに対して 448 検体の離散化された特徴値の組を求め，その組の同一性によって検体を分類した．特徴値全てが一致する検体を同種に分類し，1 つでも異なれば異種に分類した．同種と判定された各検体集合に対して，その集合が最も多く含む Microsoft 分類でのマルウェア名を，その集合の全検体のマルウェア名とした．異なる検体集合に同じマルウェア名が付いても良いとした．C5.0 を用いた分類とは異なり，こちらの分類では N 種類の特徴は対等な関係にあり，それらの適用が順序付けされることはない．また，こちらの分類では N を大きくすればするほど必ず精度のマイクロ平均は上がっていくが，代わりに汎化能力が低下して被覆率が下がり，初見の検体にマルウェア名を与えられなくなることが増える．

N 種類の特徴を組み合わせたときに得られる精度のマイクロ平均の最大値を**表 7** に示す．その値を与える組み合わせの精度と再現率のマクロ平均も表中に示している．合致する組み合わせが複数ある場合には，精度のマクロ平均，再現率のマクロ平均の優先順で大きな数を与える組み合わせを選んだ．最適な組み合わせの採用と特徴値の閾値の変更の 2 点の影響により，N がある程度大きいときには精度や再現率のマクロ平均は C5.0 でのそれらに比べてかなり高くなる．一方，精度のマイクロ平均での C5.0 との差は小さい．$N = 6$ 付近から，N を増やすことによる精度や再現率の向上は非常に小さくなる．6 種類程度の特徴でも最高に近い精度が得られることがわかる．$N = 6$ のときに表の値を与える特徴の組み合わせは maxdelay，modedelay，maxseqlen，nseqs，thread，fileattr である．

5.4　個々の API 呼び出し列から得られた知見

マルウェアが実行するスリープには，負荷軽減が目的と思われるものが多数存在した．多くの検体に見られた挙動は，繰り返される処理と処理の間に，数十ミリ秒から数秒，スリープを実行するというものである．たとえば，同じ DNS サーバへの連続するクエリの間にスリープが実行されることが多い．また，他のスレッドの再開と停止の間，プロセスリストの走査と走査の間，ファイルの属性の検査と検査の間にスリープが実行されることも多い．

`NtDelayExecution` を繰り返し呼び出すことだけしかしないスレッドも多く存在する．実行の初期に長時間のスリープを実行するスレッドも多く存在する．このようなスレッドの目的は負荷軽減ではなく解析の妨害や潜伏である可能性が高いと考えられる．経過時間の測定と組み合わせて解析システムを検出しようとしている可能性も残る．

スリープ時間については同じ数を繰り返し用いるマルウェアが多いが，少しずつ数を増やすマルウェアもある．ランダムに見えるスリープ時間を用いるマルウェアもある．実行のフェーズごとに，そのフェーズ固有のスリープ時間が用いられることも多い．1 の位が非零であるスリープ時間は少なく，そのような呼び出しを実行したマルウェアはそれだけで「目立つ」ことになる．

6.　関連研究

スリープによる耐解析処理への対策が組み込まれたサンドボックスシステムが多く存在する [4, 8, 11]．これらのサンドボックスは，マルウェアによるスリープの実行をスキップするなどの対策を実行する．これらのシステムではスリープによる耐解析処理の効果を減じることを目指しているが，本研究では，スリープによる耐解析処理の多様性を明らかにし，ひいてはその多様性を用いたマルウェアの検知や分類の技術を開拓することを目指している．

Fujino ら [5] は FFRI Dataset の API コール列を対象にしてマルウェアを分類している．彼らは引数情報を含む API コール情報のうち，マルウェアの各部分集合で頻出するものをその部分集合の API call topic と定め，API 呼び出し列と各 API call topic との間の距離を用いてマルウェアを分類している．彼らの研究は本研究とは異なり，個々の API 関数の仕様についての知識を利用していない．

中村ら [15] は FFRI Dataset の API 呼び出し列を対象にしてマルウェアを分類している．彼らはマルウェアの類似度を Kullback-Leibler 情報量によって表現している．この研究は各 API 関数を単なる記号として扱っており，個々の各 API 関数の仕様についての知識を利用していない．加えて，関数引数の情報も利用していない．一方，本研究では，スリープに焦点を絞り，その挙動に関する様々な特徴を明らかにすることを試みている．

大山 [10,13] は FFRI Dataset 2016 を対象に，耐解析処理を実行するマルウェアの割合や多くのマルウェアが実行する耐解析処理の種類を明らかにしている．他のいくつかの研究 [1,2] も，どの程度の割合のマルウェアがどのような耐解析処理を実行するかを明らかにしている．これらの研究は本研究とは対照的に，広範な耐解析処理を対象として鳥瞰的な傾向を示すにとどまっており，マルウェアが実行する個々の挙動の詳細を明らかにするものではない．

Crandall ら [3] は，仮想マシン内で観測される時刻情報の操作により，マルウェアの動作の「時刻表」および時刻に依存した動作を行うコード部分を特定するシステムを提案している．彼らの研究ではマルウェアによる時間に関連した処理を解析するための手法を提示しており，本研究では時間に関連した処理の傾向を示している．

Hasten [7] は実行の進行を遅らせる耐解析処理による影響を軽減する動的解析システムである．このシステムの研究と本研究は，実行の進行を遅らせる耐解析処理を扱っている点で共通しているが，Hasten は API 呼び出しなどの一定の処理を繰り返して時間を経過させる処理を主な対象としており，スリープは対象としていない．

7. まとめと今後の課題

スリープ挙動に関してマルウェア間で大きな多様性が存在することと，それを利用してマルウェアを分類する調査について報告した．本調査で C5.0 により生成した決定木は 6 種類の特徴値を組み合わせて用い，決定木の生成に用いた 448 中の 380（84.8%）のマルウェア検体に対して，教師データと同じマルウェア名を与える．

今後の課題を述べる．第一に，本研究での分類と Microsoft 分類の結果が違っていた検体について調査し，違いの原因を解明することがある．第二に，目的を推定できていないスリープがデータセットにはまだ多くあるため，その目的を明らかにすることがある．第三に，`NtDelayExecution` 以外の API 呼び出しの情報も組み合わせて検知や分類を実現する方法を構築することがある．

謝辞　本研究において，情報セキュリティ大学院大学の忠鉢洋輔氏，株式会社富士通研究所の小久保博崇氏，電気通信大学の高橋研介氏より有用な助言をいただいた．本研究の一部は JSPS 科研費 26330080 の助成を受けている．

参考文献

[1] Barbosa, G. N. and Branco, R. R.: Prevalent Characteristics in Modern Malware, Black Hat USA 2014 (2014).
[2] 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).
[3] Crandall, J. R., Wassermann, G., de Oliveira, D. A. S., Su, Z., Wu, S. F. and Chong, F. T.: Temporal Search: Detecting Hidden Malware Timebombs with Virtual Machines, *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 25–36 (2006).
[4] Cuckoo Sandbox, `https://cuckoosandbox.org/`.
[5] Fujino, A., Murakami, J. and Mori, T.: Discovering Similar Malware Samples Using API Call Topics, *Proceedings of the 12th Annual IEEE Consumer Communications and Networking Conference*, pp. 140–147 (2015).
[6] Kirat, D., Vigna, G. and Kruegel, C.: BareCloud: Bare-metal Analysis-based Evasive Malware Detection, *Proceedings of the 23rd USENIX Security Symposium*, pp. 287–301 (2014).
[7] Kolbitsch, C., Kirda, E. and Kruegel, C.: The Power of Procrastination: Detection and Mitigation of Execution-Stalling Malicious Code, *Proceedings of the 18th ACM Conference on Computer and Communications Security*, pp. 285–296 (2011).
[8] Lastline Labs: Not so fast my friend - Using Inverted Timing Attacks to Bypass Dynamic Analysis, `http://labs.lastline.com/not-so-fast-my-friend-using-inverted-timing-attacks-to-bypass-dynamic-analysis`.
[9] Microsoft: Naming malware, `https://www.microsoft.com/security/portal/mmpc/shared/malwarenaming.aspx`.
[10] Oyama, Y.: Trends of Anti-Analysis Operations of Malwares Observed in API Call Logs, *Journal of Computer Virology and Hacking Techniques* (2017).
[11] Payload Security: VxStream Sandbox, `https://www.payload-security.com/products/vxstream-sandbox`.
[12] RuleQuest Research: Data Mining Tools See5 and C5.0, `http://rulequest.com/see5-info.html`.
[13] 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).
[14] 高田雄太，寺田真敏，村上純一，笠間貴弘，吉岡克成，畑田充弘：マルウェア対策のための研究用データセット～MWS Datasets 2016～，情報処理学会研究報告コンピュータセキュリティ，Vol. 2016-CSEC-74 (2016).
[15] 中村燎太, 松宮　遼, 高橋一志, 大山恵弘：Kullback-Leibler 情報量を用いた亜種マルウェアの同定，コンピュータセキュリティシンポジウム 2013 論文集, pp. 877–884 (2013).