動的な関数アドレス解決APIの呼び出しログを用いたマルウェア分類

前田 優人[1]　大山 恵弘[1]

概要：本研究ではマルウェアの動的解析の結果に含まれる LdrGetProcedureAddress の呼び出しログを用いてマルウェアの分類を行う手法を提案する．LdrGetProcedureAddress は動的に関数アドレスを解決するための API である．この API の引数として渡された関数名を収集することで，動的解析時にフックされなかった API の情報を収集することができる．収集した情報を元に，機械学習を用いることでマルウェアの分類を行う．実際に FFRI Dataset に含まれる動的解析の結果に本手法を適用し，静的解析による分類および通常の動的解析による分類と性能を比較した．

キーワード：マルウェア分類，動的解析，アドレス解決，機械学習

Malware Classification Using the Call Log of an API for Dynamic Function Address Resolution

Yuto Maeda[1]　Yoshihiro Oyama[1]

Abstract: In this paper, we propose a method to classify malware using the log of LdrGetProcedureAddress calls included in the result of malware dynamic analysis. LdrGetProcedureAddress is an API for resolving function addresses dynamically. Collecting the function names given as the argument of the API, the method can collect the information on the API calls that were not intercepted in dynamic analysis. It then classifies malware using machine learning based on the collected information. We applied the method to the dynamic analysis result included in the FFRI Dataset, and compared the classification ability with a classification method using static analysis and an ordinary classification method using dynamic analysis.

Keywords: Malware classification, dynamic analysis, address resolution, machine learning

1. はじめに

マルウェアの種類数は増加し続けており，マカフィーによる脅威レポート [5] によれば，2018 年第 1 四半期には 4,000 万件以上のマルウェアが新たに発見されている．このペースで増加するマルウェアすべてを解析することは困難であるため，既知のマルウェアと同様の挙動を示すマルウェアを亜種として自動で分類することはマルウェアの解析者にとって非常に重要であると言える．しかし，現状ではマルウェアの分類の自動化の精度は十分に高いとは言えない．十分な精度が出ない理由の一つとして，マルウェアから得られた情報を十分に活用できていないという可能性が挙げられる．本研究では，動的解析で見落とされる可能性のある情報を使用する方法及びそのときの分類精度を測定することを目標とする．

通常，マルウェアは静的解析や動的解析によって得られる情報を用いて分類される．静的解析によって得られる情報として，マルウェアのインポートテーブルに載っている API 情報が挙げられる．静的解析による情報を用いた分類手法には，マルウェアを実行する必要がないという利点があるが，動的に実行しなければ得られない情報を活用することができない．動的解析によって得られる情報には，マ

[1] 筑波大学
University of Tsukuba

ルウェアが実際に実行した API の情報が挙げられる．マルウェアが実行した API の情報を用いてマルウェアを分類する手法はこれまでにも多く提案されている．記録できる API は，以下の 3 通りに分類できる．

(1) インポートテーブルに載っていて，プログラムの起動時にアドレスが解決される関数
(2) プログラムの実行時に動的にアドレスが解決される関数
(3) (1) と (2) の関数の内部で呼び出される関数

このうち，どの種類の API を用いればよいかは自明ではない．そこで本研究では，(2) の実行時に動的にアドレスが解決される関数に着目し，これらの関数の呼び出し情報を用いたマルウェアの分類方法を提案する．

マルウェアの動的解析に用いられるツールの一つに Cuckoo Sandbox がある．Cuckoo Sandbox は検体を仮想環境内で実行し，呼び出した Windows API を記録する．Cuckoo Sandbox では，独自に記録する API の一覧[*1]を持っており，それに載っている API の呼び出し情報を動的解析の結果として出力する．一覧に載っている API は (1)〜(3) の特に区別なく採用されているが，全体で 360 種類程度の API の呼び出ししか記録しないという問題がある．その API の中の 1 つに LdrGetProcedureAddress という API がある．この API は，動的に関数のアドレスを解決するために用いる GetProcAddress という API の内部で呼び出される．GetProcAddress の引数として解決したい関数の関数名を指定して呼び出すと，LdrGetProcedureAddress の引数にもその関数名が渡されて呼び出される．したがって，LdrGetProcedureAddress が呼び出されたときの引数を記録することで，(2) の種類の API を呼び出しうるという情報を得ることができる．本研究ではこの動的なアドレス解決に関する情報をマルウェア分類に用いる手法を提案する．

提案手法に加え，動的解析によって得られる通常の API ログと静的解析によって得られる API リストを用いてマルウェアの分類性能を評価する実験を行った．動的解析のデータには MWS Dataset 2018 内の FFRI Dataset 2017 を使用した．また，マルウェアから得られた情報を活用するという観点から，提案手法と他のデータを組み合わせた場合の実験も行った．実験はそれぞれのデータのみを用いた場合の 3 通りと，提案手法と他のデータを組み合わせた場合の 2 通りの合計 5 通り用意した．それらを実験の入力データとして 4 種類の機械学習アルゴリズムに対して適用し，マルウェア分類性能の比較を行った．その結果，提案手法では通常の API ログを用いた場合と同等程度の分類精度が得られた．また，データを組み合わせることでマルウェア分類の性能が向上することが示された．

[*1] https://github.com/cuckoosandbox/monitor/tree/master/sigs

2. 関連研究

API 呼び出しの情報を用いてマルウェアの分類や検知を行う方法は今までにも多くの研究で提案されている．Fujino らの研究 [1] では，本研究と同様に，FFRI Dataset の API 呼び出しログを用いてマルウェアを分類している．彼らの方法は，LdrGetProcedureAddress を含む API の引数の情報を利用するが，すべての API を等価に扱い，API の種類を考慮しない．よって，関数アドレス解決に関する挙動をマルウェア分類に用いることの有効性は明らかにされていなかったが，本研究ではそれを明らかにしている．

Kolosnjaji らの研究 [4] では，動的な API 呼び出しの列をニューラルネットワークでモデル化してマルウェア分類を行い，その精度が HMM や SVM を用いたときの精度を上回ることを示している．本研究とは異なり，彼らの研究では，API 呼び出しの引数や回数の情報を利用していない．また，すべての API を等価に扱い，API の種類を考慮していない．

中村らの研究 [10] では，本研究と同じく，FFRI Dataset に含まれる API 呼び出しの列を機械学習アルゴリズムによって学習する方法を比較評価している．彼らが評価している方法は，本研究とは異なり，マルウェア名を推定するものではなく，プログラムがマルウェアであるかどうかを判定するものである．また，彼らが評価している方法では，API の種類も引数も考慮していない．

Gupta らの研究 [2] では，マルウェアを実行して収集した API 呼び出しの列を Fuzzy Hashing によってシグネチャに変換する手法を提案している．シグネチャ間の距離を利用することにより，マルウェアの類似性をうまく捉えてマルウェアを分類できることを示している．本研究とは異なり，彼らの研究では，API の種類も引数も考慮していない．さらに，彼らの研究では，マルウェア名の推定は行っておらず，ワーム型かバックドア型かなどのマルウェアの大まかな種別を推定するにとどまっている．

Uppal らの研究 [7] では，プログラムの逆アセンブル結果から API 呼び出しの情報を取り出して特徴化している．さらに，それを SVM や RF を含む機械学習アルゴリズムで処理することにより，プログラムがマルウェアであるかどうかを判定している．Sami らの研究 [6] でも，プログラムのインポートテーブルから API の集合を抽出し，それを RF などのアルゴリズムで学習することにより，プログラムがマルウェアであるかどうかを判定している．どちらの研究も，本研究と異なり，マルウェアの動的な挙動に関する情報や API 呼び出しの引数の情報を用いていない．

3. 提案手法

本研究では，マルウェアの動的解析ログに含まれる LdrGetProcedureAddress の呼び出し情報を用いてマルウェア

を分類する手法を提案する．

3.1 LdrGetProcedureAddress

LdrGetProcedureAddress は Windows が関数のアドレスを解決する際に OS 内部で呼び出す API である．この API は DLL などからエクスポート済みのアドレスをシンボル名から入手するための API である GetProcAddress の内部で呼び出されている．プログラムが GetProcAddress を呼び出すと，LdrGetProcedureAddress を内部で呼び出し，その結果を元に目的の関数のアドレスを得る．

このような動的なアドレスの解決はプログラムの起動時に OS が行うほか，マルウェアがプログラムの耐解析処理を目的として使用することがある．通常であれば実行ファイルは使用する関数のテーブルを持っており，実行ファイルの起動時に OS がその関数のアドレスを解決しテーブル内に記録する．この時，解決する関数名をテーブル内に定義しておく必要があるため，静的解析の際に容易にその情報を取得できてしまう．そのため，一部のマルウェアは GetProcAddress を使用して動的に関数のアドレスを解決することで，実際に呼び出す関数を静的解析で知られないようにしている．実際に，FFRI Dataset 2017 に含まれる 6251 検体のうち，4782 検体がインポートアドレステーブルに GetProcAddress を含んでおり，非常に多くのマルウェアが GetProcAddress を用いた動的なアドレス解決を行っていることがわかる．したがって，この API の呼び出しを追跡することはマルウェアの分類をする上で非常に有用であると考えられる．

実際に図 1 に示すコードを用いて GetProcAddress から LdrGetProcedureAddress が呼び出されるまでの流れを説明する．このコードは user32.dll という DLL をロードし，その中の MessageBoxA 関数のアドレスを取得した後，そのアドレスを用いてメッセージボックスを表示するプログラムになっている．これをコンパイルし Cuckoo Sandbox 上で実行すると，API 呼び出しのログの一部として図 2 に示す結果が得られる．図 2 の `api` の部分は呼び出された関数名を表しており，図では LdrGetProcedureAddress が呼び出されている．`arguments` の部分は呼び出された関数に与えられた引数を表しており，`function_name` としてアドレスを解決したい関数名である MessageBoxA が書かれている．この関数名は GetProcAddress の引数として与えたものと同じであり，GetProcAddress から LdrGetProcedureAddress が呼び出されたことがわかる．

3.2 学習アルゴリズムとデータ形式

本研究では，決定木学習，ランダムフォレスト（RF），SVM，ナイーブベイズの 4 種類のアルゴリズムを用いてどの機械学習アルゴリズムが本手法において性能が高いかについても検証する．機械学習の手法を適用するために，

```
#include <windows.h>
typedef int (WINAPI *MSGA)(HWND, LPCSTR,
    LPCSTR, UINT);
int main() {
  HMODULE mod = LoadLibraryA("user32.dll");
  MSGA func = (MSGA)GetProcAddress(mod, "
      MessageBoxA");
  if (func == NULL) {
    return 1;
  }
  func(NULL, "title", "msg", MB_OK);
  return 0;
}
```

図 1 GetProcAddress によって動的に関数のアドレスを解決するプログラム

```
{
  "category": "system",
  "status": 1,
  "stacktrace": [],
  "api": "LdrGetProcedureAddress",
  "return_value": 0,
  "arguments": {
    "ordinal": 0,
    "module": "user32",
    "module_address": "0x76aa0000",
    "function_address": "0x76b0fd1e",
    "function_name": "MessageBoxA"
  },
  "time": 1533040341.109875,
  "tid": 2060,
  "flags": {}
}
```

図 2 LdrGetProcedureAddress の呼び出しログ

関数のアドレスが解決された回数を入力データとして使用する．

まず，Cuckoo Sandbox の API 呼び出しログから LdrGetProcedureAddress を呼び出している部分のみを抽出し，解決している関数名ごとに呼び出し回数を記録する．関数名には，LdrGetProcedureAddress の引数として与えられたものをそのまま利用した．回数を記録する理由としては，同じ関数のアドレス解決を複数回行う検体が存在したためである．また，関数アドレスの解決が成功したかどうかは区別しない．これをマルウェア検体数分だけ繰り返す．

この状態の集計結果では検体ごとに解決する関数の集合が異なり学習に向かないため，次の方法で集計結果を拡張する．全検体を通して解決された関数名のリストを作成する．次に各検体ごとにリストの順に沿うように関数名の呼び出し回数を配置する．呼び出されなかった関数の呼び出し回数は 0 とする．

以上の手順により，図 3 のような形式の特徴ベクトルを得る．この形式に変換したデータを機械学習の入力データ

検体	ラベル	MessageBoxW	PostMessageW	LoadIconW	...	Sleep
1	FamilyA	5	0	2		1
2	FamilyB	0	2	4	...	0
3	FamilyC	4	1	0		1
4	FamilyB	0	2	3		0
⋮	⋮		⋮		⋱	⋮
4457	FamilyZ	1	1	2	...	2

図 3 各マルウェア検体による API のアドレス解決に関する特徴ベクトル

として使用した．

4. 実験と評価

4.1 実験に用いたデータ

提案手法の性能を評価するために MWS Dataset 2018 [9] に含まれる FFRI Dataset 2017 [8] を使用して実験を行った．FFRI Dataset 2017 を使用した理由は，Cuckoo Sandbox による動的解析ログが提供されているデータセットで最新のものであるからである．

FFRI Dataset 2017 は Cuckoo Sandbox 上でマルウェアを実行して得られた動的解析結果のデータセットである．このデータセットには 2017 年 3 月から 4 月に収集されたマルウェア 6251 検体の解析結果が収録されている．結果には Windows 7 x86 上で検体を 90 秒間実行した動的解析ログのほか，VirusTotal による各アンチウイルスソフトによる解析結果や検体の静的解析結果などが含まれる．動的解析ログには，Cuckoo Sandbox がフックした API の名前及び引数や，フックした API ごとの呼び出し回数（図 4）が記録されている．フックした API ごとの呼び出し回数は Cuckoo Sandbox 内では APIStats と呼んでおり，本稿でも以降では APIStats と呼ぶ．静的解析の結果には，検体内のインポートアドレステーブルに含まれる関数名（図 5）や検体の各種ハッシュ値などが含まれる．比較のために，これらの動的解析ログや静的解析結果を用いた実験も行った．

分類するマルウェアの正解のファミリー名には，Kaspersky により命名されたマルウェア名を採用した．Kaspersky によるマルウェアの命名 [3] は以下に示す形式になっている．

`[Prefix:]Behaviour.Platform.Name[.Variant]`

Prefix にはマルウェアを検知した環境がヒューリスティック検知や振る舞い検知のような特殊な状況であった場合にその情報が入る．Variant には同種のマルウェアでも動作が異なるものが存在するときに，識別用のアルファベットが追加される．Behaviour には，Trojan や Virus など，マルウェアの主な挙動を表す文字列が入る．Platform には Win32 や MSIL などのマルウェアが動作する環境が入る．Name はファミリー名を表す文字列が入る．本実験ではマルウェアの亜種を分類することが目的であるため，正確でない可能性がある Prefix 付きのマルウェアは除外し，

```
"apistats": {
  "3644": {
    "CreateToolhelp32Snapshot": 1,
    "RegCreateKeyExW": 6,
    "DeviceIoControl": 2,
    "CoUninitialize": 1,
    "RegCloseKey": 8,
    "CopyFileW": 5,
    "NtDuplicateObject": 3,
    "GetSystemInfo": 2,
    "RegQueryValueExA": 1,
    "MoveFileWithProgressW": 7,
    "LdrGetProcedureAddress": 260,
    "WSAStartup": 1,
    ...
    "NtCreateFile": 12,
    "UuidCreate": 2,
    "CreateProcessInternalW": 1,
    "NtQueryValueKey": 10
  }
}
```

図 4 Cuckoo Sandbox のログに含まれる APIStats 部分

ファミリー名として `Behavior.Platform.Name` の部分を用いた．

データセットに含まれる全 6,251 検体のうち，実験には以下の条件を満たす 4,457 検体 17 ファミリーを使用した．

- Kaspersky によりマルウェアと判定され名称がついているもの．
- マルウェア名において Prefix がついていないもの．
- 同一ファミリーで 5 検体以上存在するもの．

4.2 実験方法

実験は表 1 に示す 5 通りの方法で行い，結果を比較した．表には実験に用いたデータの特徴ベクトルの長さも併せて表記した．機械学習アルゴリズムには，決定木学習 (CART)・ランダムフォレスト・SVM・ナイーブベイズの 4 種類を用いた．実験には，Python 3 (3.6.5) と scikit-learn (0.19.1) を使用した．

実験 2 で用いる関数の呼び出し回数は，APIStats に記録されていた値をそのまま使用した．実験 3 で用いる静的解析による API の情報は，関数がテーブルに載っている・載っていないの 2 値の情報しか得られないため，載っていれば 1 を，載っていなければ 0 を関数の呼び出し回数とし

表 1 各実験の内容とそのときの特徴ベクトルの長さ

	実験内容	特徴ベクトルの長さ
実験 1	LdrGetProcedureAddress の呼び出し記録から得られた，関数ごとの呼び出し回数を用いて学習する．	8,584
実験 2	Cuckoo Sandbox が出力する APIStats の結果を元に学習する．	289
実験 3	検体の静的解析により得られた API を元に学習する．	5,160
実験 4	実験 1 と実験 2 で用いたデータを合わせたものを用いて学習する．	8,873
実験 5	実験 1 と実験 3 で用いたデータを合わせたものを用いて学習する．	13,744

```
"pe_imports": [
  {
    "imports": [
      {
        "name": "EndPaint",
        "address": "0x423138"
      },
      {
        "name": "DestroyWindow",
        "address": "0x42313c"
      },
      {
        "name": "GetMessageA",
        "address": "0x423140"
      },
      ...
      {
        "name": "HeapReAlloc",
        "address": "0x42312c"
      },
      {
        "name": "LCMapStringW",
        "address": "0x423130"
      }
    ],
    "dll": "KERNEL32.dll"
  }
],
```

図 5 Cuckoo Sandbox のログに含まれるインポート関数情報

$$\text{精度} = \frac{\text{正しく分類できた検体数}}{\text{全 4,457 検体}}$$

$$\text{適合率} = \frac{\text{分類が的中した検体数}}{\text{分類した検体数}}$$

$$\text{再現率} = \frac{\text{正しく分類できた検体数}}{\text{あるファミリーの検体数}}$$

図 6 精度・適合率・再現率の計算式

表 2 各学習方法における分類結果 [%]

		決定木	RF	SVM	ベイズ
精度	実験 1	86.36	87.64	86.99	61.68
	実験 2	84.07	87.91	75.88	51.56
	実験 3	83.17	84.00	83.69	59.82
	実験 4	85.35	88.22	75.93	53.04
	実験 5	85.78	87.70	87.73	77.23
適合率	実験 1	76.00	78.83	74.25	52.25
	実験 2	70.79	82.04	73.21	49.20
	実験 3	65.85	69.36	68.10	49.07
	実験 4	74.38	78.54	73.19	51.11
	実験 5	69.44	78.09	75.70	64.36
再現率	実験 1	74.22	74.59	69.31	67.67
	実験 2	70.86	72.25	53.48	63.76
	実験 3	64.01	63.43	63.78	62.05
	実験 4	73.24	72.89	53.33	63.02
	実験 5	70.51	73.27	70.49	71.76

て用いた．実験 4 及び実験 5 は，提案するマルウェア分類手法を他の分類手法と組み合わせたときの分類性能を確かめるものである．各検体の特徴ベクトルをそのまま結合したものを新たな特徴ベクトルとして用いた．結合時に同じ関数名のものがあった場合でも，別のものとして扱った．

実験は 5 分割交差検証で行い，各回ごとの精度・適合率・再現率を求め，その平均値を求めた．精度はマルウェアのファミリーに関係なく全体を対象に求めた値である．適合率と再現率はそれぞれのマルウェアのファミリーごとの値を求め，その平均値を用いた．なお，精度，適合率，再現率は図 6 に示す式で算出される値である．

4.3 実験結果

実験結果を表 2 に示す．

4 種類のアルゴリズムで機械学習を行った結果を比較すると，実験 1〜5 のいずれにおいてもランダムフォレストを用いた場合の分類性能が最も高かった．

まず，それぞれのデータを単体で使用した実験（実験 1〜3）について述べる．本研究の目的である LdrGetProcedureAddress の呼び出し記録から得られる情報を用いた実験 1 の精度は 87.64 %であった．この結果は APIStats のみを用いてマルウェアを分類した実験 2 の精度 87.91 %より若干低い値である．しかし，ランダムフォレスト以外の 3 種類のアルゴリズムにおいては実験 1〜3 において最も高い精度を記録した．したがって LdrGetProcedureAddress の呼び出し記録から得られる情報を用いたマルウェア分類方法はある程度有効であると言える．学習アルゴリズムごとに適合率・再現率の結果を見ると，ランダムフォレストと用いたときの適合率を除いたすべての場合において，実験 1〜3 の中で提案手法を用いた実験 1 が最も高い値だった．ランダムフォレストと SVM でのみ適合率が低下してしまった理由は現在わかっていないが，ファミリーごとに値を計算し平均を取っているため，ファミリーによって分類の得意不得意がある可能性が考えられる．

次に，2 つのデータを組み合わせた実験について述べる．分類精度では，LdrGetProcedureAddress の呼び出し記録から得られる情報と APIStats の情報を組み合わせた実験 4 の精度が 88.22 %を記録した．これはすべての実験の中で最も高い精度である．また，LdrGetProcedureAddress の呼び出し記録から得られる情報と静的解析から得られる情報を組み合わせた実験 5 でも，それぞれを単体で使用した場合よりも高い精度を記録した．したがって，提案手法を他の手法と組み合わせることで，マルウェア分類の性能を向上できることが示された．一方で，適合率・再現率に関してはアルゴリズムと分類方法の組み合わせの一部の場合において，組み合わせたどちらの手法よりも数値が低かった．数値が低かった組み合わせは，実験 4 でランダムフォレストを用いた場合の適合率，実験 4 で SVM を用いた場合の適合率，実験 4 で SVM を用いた場合の再現率の 3 つである．数値が低かった理由はまだわかっていないが，おそらく実験 1〜3 と同様にファミリーによる得意不得意があると考えられるため，今後の検証が必要である．

5. まとめと今後の課題

本研究では，マルウェアの動的解析結果に含まれる LdrGetProcedureAddress の呼び出し記録から得られる情報を用いてマルウェアの分類を行う方法を提案した．提案手法は LdrGetProcedureAddress の引数に含まれる関数名を元に，どの関数が何回アドレス解決されたかをカウントし学習用のデータとするものである．さらに，FFRI Dataset 2017 を用いてマルウェア分類の実験を行い，分類性能を測定した．4 種類の機械学習アルゴリズムに対し，LdrGetProcedureAddress を用いた分類方法のほか，動的解析結果に含まれる単純なデータや静的解析結果を用いた分類を行い，性能を比較した．加えて LdrGetProcedureAddress を用いた分類方法とそれ以外の方法を組み合わせた場合についても実験を行った．実験の結果，LdrGetProcedureAddress の呼び出し記録から得られる情報を用いたマルウェアの分類性能は，動的解析から得られる情報を用いた分類性能に若干劣る結果となった．しかし，提案手法と他のデータを組み合わせて実験を行った場合に最も高い精度が得られたため，LdrGetProcedureAddress の呼び出し記録から得られる情報はマルウェアの分類を行うときに用いる情報源の 1 つとして有効であると言える．

今後の課題を次に述べる．精度については概ね期待通りの結果が得られたものの，適合率・再現率についてはファミリーごとの値を平均したため，ファミリーごとの得意不得意を見ることができず，結果の考察をすることができなかった．より深い考察を行うためにファミリーごとの適合率・再現率を調査することが必要であると考える．組み合わせるために用いたマルウェアの分類方法は，動的解析の API ログと静的解析による API リストであった．それ以外のファイル操作やレジストリ操作の情報と組み合わせた場合にどの程度まで分類精度が向上するかについても検証を行う必要があると考える．

謝辞 FFRI Datasets を提供して下さった (株) FFRI および MWS 組織委員会に感謝する．本研究の一部は JSPS 科研費 17K00179 の助成を受けている．

参考文献

[1] 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).

[2] Gupta, S., Sharma, H. and Kaur, S.: Malware Characterization Using Windows API Call Sequences, *Proceedings of the 6th International Conference on Security, Privacy, and Applied Cryptography Engineering*, pp. 271–280 (2016).

[3] Kaspersky: Rules for naming — Kaspersky Lab Encyclopedia, https://encyclopedia.kaspersky.com/knowledge/rules-for-naming/.

[4] Kolosnjaji, B., Zarras, A., Webster, G. D. and Eckert, C.: Deep Learning for Classification of Malware System Call Sequences, *Proceedings of the 29th Australasian Joint Conference on Artificial Intelligence*, pp. 137–149 (2016).

[5] McAfee: McAfee Labs Threats Report - June 2018, https://secure.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-jun-2018.pdf.

[6] Sami, A., Yadegari, B., Rahimi, H., Peiravian, N., Hashemi, S. and Hamze, A.: Malware Detection Based on Mining API Calls, *Proceedings of the 2010 ACM Symposium on Applied Computing*, pp. 1020–1025 (2010).

[7] Uppal, D., Sinha, R., Mehra, V. and Jain, V.: Exploring Behavioral Aspects of API calls for Malware Identification and Categorization, *Proceedings of the 6th International Conference on Computational Intelligence, Communication Systems and Networks*, pp. 824–828 (2014).

[8] 株式会社 FFRI：FFRI Dataset 2017 のご紹介, MWS2017 意見交換会 (2017).

[9] 高田雄太，寺田真敏，松木隆宏，笠間貴弘，荒木粧子，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 2018〜，情報処理学会研究報告コンピュータセキュリティ，Vol. 2018-CSEC-82, No. 38 (2018).

[10] 中村燎太，大山恵弘：ビヘイビアベースマルウェア検知におけるオンライン機械学習アルゴリズムの比較評価，コンピュータソフトウェア，Vol. 34, No. 4, pp. 156–177 (2017).