Vol.2019-DPS-178 No.17
Vol.2019-CSEC-84 No.17
2019/3/4

関数アドレス取得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：本研究では動的解析ログに含まれるLdrGetProcedureAddressの呼び出し記録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の引数には関数名が渡されるため，マルウェアが呼び出す可能性のある関数を収集することができる．収集した関数名を元に，機械学習（ランダムフォレスト，XGBoost，LightGBM）を用いてマルウェアの分類を行う．実際にマルウェアの動的解析ログのデータセットであるFFRI Dataset 2017に対して提案手法を適用し，その結果を静的解析および動的解析で得られるAPI名を用いた分類結果と比較した．

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 Logs of an API for Getting Function Addresses

Yuto Maeda[1]　Yoshihiro Oyama[1]

Abstract: In this paper, we propose a method to classify malware using the logs 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 an argument of the API, the method can collect the information on the function calls that were not intercepted in dynamic analysis. It then classifies malware using machine learning (Random Forest, XGBoost and LightGBM) based on the collected information. We applied the method to the dynamic analysis result included in the FFRI Dataset 2017, and compared the classification ability with a classification method using the APIs from static analysis and another method using the APIs from 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．この速度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1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可能性が挙げられる．本研究では，動的解析のログにおいて新たに分類に用いることができるデータを発見し，そのときの分類性能を測定することを目標とする．

通常，マルウェアの分類にはプログラムを動作させずに解析する静的解析や，動作させて解析する動的解析によって得られる情報を用いる．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

[1] 筑波大学
University of Tsukuba

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 [1,2,4,10]．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．すべての API を用いることは容易だが，API の呼び出し回数が膨大になるため，データ量が増加したり分類に時間がかかったりするという欠点がある．一方で（1）のみを用いた場合は，記録すべき API 数は少なくなるがデータの取りこぼしが多くなってしまう．また（3）を用いると呼び出し元の関数の情報が失われているため，大きな粒度での呼び出し情報を取りこぼしてしまう．（2）は（1）と（3）の中間に属し，API 呼び出し回数を絞りつつ大きな粒度での情報を記録できる．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 1 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（3）に属するものが多く，全体で 360 種類程度の API の呼び出ししか記録しない．これらは Windows に存在するの大量の API の中から低レベルな呼び出しを集めたものであるため，粒度の小さい情報しか記録できないという問題がある．Cuckoo Sandbox が記録する 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．マルウェアが呼び出す関数の名前はマルウェアの挙動を特徴づけると考えられるため，本研究ではこの関数名を用いてマルウェア分類を行う手法を提案する．

提案手法の評価のために，動的解析から得られる通常の API ログと，静的解析から得られるインポートしている API のリストを用いてマルウェアの分類性能を比較する実験を行った．また LdrGetProcedureAddress から得られた関数名に動的解析から得られる API ログや，静的解析によって得られる API リストを付け加えたデータ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．マルウェアのファミリー名には Kaspersky によって命名されたマルウェア名と AVClass [7] から得られたマルウェア名を使用した．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 3 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，LdrGetProcedureAddress から得られる関数名と他のデータを組み合わせることで，それぞれを単独で使用するよりもマルウェア分類の性能が向上することが示された．

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．

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

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 [8] 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 やランダムフォレスト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，それをランダムフォレスト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 [12]．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，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 6,251 検体のうち，4,782 検体がインポートアドレステーブルに GetProcAddress を含んでおり，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．またマルウェアが難読化を目的として使用する UPX などのパッカーを適用したマルウェアでは，アンパック後に呼び出す API はインポートアドレステーブルに含まれない．そのため，GetProcAddress を用いたアドレスの解決が行われる．

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 呼び出しログの一部として図 2 に示す結果が得られる．図 2 の `api` に格納されている文字列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` 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` という文字列が格納されている．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．もし静的にこのプログラムを解析し，インポートアドレステーブルから呼び出す関数を取得した場合，`MessageBoxA` はテーブルに含まれていない．さらに静的解析で文字列を収集し `MessageBoxA` という文字列を発見できたとしても，`title` や `msg` といった他の文字列ノイズがあるため，どれが関数名であるかを判定することは難しい．このような関数の呼び出しは呼び出した API が動的解析の記録対象になっていない限りは見落としてしまう．

呼び出される関数名を収集するということは，単純な API 呼び出しよりも粒度の大きい処理単位を分類のデータとして扱えるという利点もある．通常関数の内部では複数の API を呼び出すため，それらが API 呼び出しログとして 1 次元に羅列されてしまうと呼び出し元の関数の情報を復元することは難しい．そこで LdrGetProcedureAddress から得られる関数名を用いることで呼び出し元の関数の情報をマルウェア分類時の特徴として使うことができる．

以上の理由から，LdrGetProcedureAddress 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有用であると考えられる．

3.2 学習アルゴリズムとデータ形式

本研究では，ランダムフォレスト（RF），XGBoost，LightGBM の 3 種類の機械学習アルゴリズムを用いて実験を行い，どの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，関数のアドレスが解決された回数を特徴量として使用する．

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

```
#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

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多数存在したためである．また，関数アドレスの解決自体を試みたという事実を分類に用いるため，アドレスの解決に成功したかどうかは区別していない．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ベクトル化する．まず全検体を通して解決された関数名のリストを作成し，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．このとき呼び出されなかった関数の呼び出し回数は 0 として扱う．

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 [11] に含まれる FFRI Dataset 2017 [9] を使用して実験を行った．FFRI Dataset 2017 を使用した理由は，FFRI Dataset 内で 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 6,251 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ファミリー名として，Kaspersky により命名されたマルウェア名と，VirusTotal の結果から多数決を取る AVClass [7] によって命名されたマルウェア名の 2 通りを使用して実験を行った．Kaspersky によるマルウェア名を用いた理由は名前付けが明確であり，Generic といった汎用的な名前が用いられているものが少なかったからである．また AVClass を用いた理由は，1 つのベンダーに頼らないことでマルウェア名のミスを減らしつつ実験に使用できる検体数を増やすためである．

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，ファミリー名として `Behaviour.Platform.Name` の部分を用いた．Kaspersky によるマルウェア名を使用した実験では，

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

```
"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

```
"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

データセットに含まれる全 6,251 検体のうち，以下の条件を満たす 4,255 検体 73 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．

AVClass を用いた実験では，同一ファミリー内で 5 検体以上存在する検体を抽出し 5,703 検体 94 ファミリーを使用した．

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，ランダムフォレスト (RF)・XGBoost・LightGBM の 3 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．

DynStats 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．Static 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て用いた．DynStats+及び Static+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 (Accuracy)・適合率 (Precision)・再現率 (Recall) 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．

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
GetProc	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
DynStats	Cuckoo Sandbox が出力する APIStats の結果を元に学習する．	289
Static	検体の静的解析により得られた API を元に学習する．	5,160
DynStats+	GetProc と DynStats で用いたデータを合わせたものを用いて学習する．	8,873
Static+	GetProc と Static で用いたデータを合わせたものを用いて学習する．	13,744

$$\text{精度} = \frac{\text{正しく分類できた検体数}}{\text{全検体}}$$
$$\text{適合率} = \frac{\text{分類が的中した検体数}}{\text{分類した検体数}}$$
$$\text{再現率} = \frac{\text{正しく分類できた検体数}}{\text{あるファミリーの検体数}}$$

図 6 精度・適合率・再現率の計算式

4.3 実験結果と考察

Kaspersky によるラベルを用いた際の実験結果を表 3 に示す．

まずデータを単体で使用した実験（GetProc, DynStats, Static）について述べる．提案手法である LdrGetProcedureAddress の呼び出しから得られる情報を用いた GetProc の精度で最も良かったのは LightGBM の 88.65%であった．これは，動的解析の API ログを用いた DynStats でランダムフォレストを用いたときの精度 88.91%に迫る値である．また，適合率と再現率の観点からも GetProc は DynStats とほぼ同等の結果が得られた．一方で，検体がインポートしている API をもとに学習を行った Static ではランダムフォレストの精度 81.48%に留まった．これらの結果から，LdrGetProcedureAddress の呼び出しログから得られる情報はマルウェアの分類を行う上で有用であると言える．

次に，提案手法と他の種類のデータを組み合わせた実験（DynStats+, Static+）について述べる．DynStats+, Static+のどちらとも組み合わせる前のどちらよりも精度が高く，DynStats+のランダムフォレストでは 89.12%という結果であった．適合率と再現率では組み合わせる前よりも下がっている場合もあるが，提案手法を他の手法と組み合わせることで分類性能が向上したと言える．

次に，AVClass によるファミリー名を用いた際の実験結果を表 3 に示す．GetProc の精度で最も良かったのは LightGBM を用いたときの精度で 83.97%であった．これは DynStats でランダムフォレストを用いたときの精度と同じである．全体では AVClass によるファミリー名を用いた実験結果は，Kaspersky のものを用いた実験と傾向は同じで僅かに劣る結果となった．Kaspersky の実験よりも検体数ファミリー数ともに増えているため，分類がより困難になったことが原因だと考えられる．

表 2 各学習方法における分類結果（Kaspersky）[%]

		RF	XGBoost	LightGBM
精度	GetProc	87.83	88.25	88.65
	DynStats	88.91	88.44	88.27
	Static	81.48	81.34	80.61
	DynStats+	89.12	88.86	88.77
	Static+	88.46	88.65	88.88
適合率	GetProc	70.40	69.66	70.64
	DynStats	72.42	70.49	70.24
	Static	55.46	54.64	53.47
	DynStats+	71.78	70.30	71.10
	Static+	72.01	71.62	71.71
再現率	GetProc	68.14	68.29	67.72
	DynStats	69.59	68.39	67.04
	Static	54.49	54.12	52.69
	DynStats+	69.21	68.76	67.98
	Static+	69.64	69.86	68.12

表 3 各学習方法における分類結果（AVClass）[%]

		RF	XGBoost	LightGBM
精度	GetProc	82.48	83.55	83.97
	DynStats	83.97	83.53	83.45
	Static	78.75	79.41	78.55
	DynStats+	83.99	84.39	83.94
	Static+	84.53	84.73	85.01
適合率	GetProc	66.19	67.63	65.23
	DynStats	68.51	65.53	66.14
	Static	55.02	54.05	53.71
	DynStats+	69.00	69.15	68.14
	Static+	67.38	67.40	66.57
再現率	GetProc	66.82	68.14	65.88
	DynStats	67.18	65.53	64.96
	Static	53.47	54.85	54.22
	DynStats+	68.38	69.15	68.14
	Static+	68.16	68.94	67.08

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．分類に用いるファミリー名として，Kaspersky によって命名されたマルウェア名を用いるものと，AVClass を用いて多数決を取ったマルウェア名の 2 種類を用いて実験を行った．3 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は，Cuckoo Sandbox が出力する動的解析ログをそのまま用いたときの分類性能に近い性能であった．また，提案手法と他のデータを組み合わせた実験では，それぞれのデータを単体で用いた場合よりも分類精度が向上した．この結果から，LdrGetProcedureAddress の呼び出し記録はマルウェアの分類に有用であることを示した．

今後の課題を次に述べる．分類に用いた関数名の種類は検体数に応じて増加していくため，大量のマルウェアを分類しようとすると分類に用いる特徴ベクトルが長くなってしまうという問題がある．事前に分類する上で有用となる関数名の一覧を事前に作成しておき，特徴量を削減することが効果的であると考える．また，本研究では提案手法がどのファミリーの分類に効果的であるかの考察を行えていない．今後は少ないファミリー数で実験を行い，分類に向いているファミリーを明らかにすることも有意義だと考える．

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] Sebastián, M., Rivera, R., Kotzias, P. and Caballero, J.: AVclass: A Tool for Massive Malware Labeling, *Research in Attacks, Intrusions, and Defenses*, Cham, Springer International Publishing, pp. 230–253 (2016).

[8] 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).

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

[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).

[11] 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).

[12] 松木隆宏，新井　悠，寺田真敏，土居範久：マルウェアの耐解析機能を逆用した活動抑止手法の提案，情報処理学会論文誌，Vol. 50, No. 9, pp. 2118–2126 (2009).