動的解析ログを活用した静的解析補助手法の提案

中島 将太[1,a)]　明田 修平[1]　瀧本 栄二[1]　齋藤 彰一[2]　毛利 公一[1]

概要：マルウェア対策では，マルウェア解析が重要である．一般的にマルウェア解析は，動的解析，静的解析の手順で行う．しかし，現状では動的解析の結果が，静的解析作業と十分に連携できているとは言えない．特に，動的解析時に記録した API 呼び出し情報と逆アセンブルコードを対応付けていないため，静的解析時に実行時の API 呼び出し情報を活用できていない．また，静的解析を行うためには，実行時にのみ展開されるコードを取得する必要がある．そこで，動的解析時の API 呼び出し情報と，メモリ上のマルウェアのコードを取得し，静的解析を積極的に補助する手法を提案する．本論文では，システムコールトレーサ Alkanet と逆アセンブラ IDA を連携させた静的解析補助手法について述べる．

キーワード：MWS，マルウェア，動的解析，静的解析，API トレース

Proposal of static analysis assistance method utilizing the dynamic analysis log

Shota Nakajima[1,a)]　Shuhei Aketa[1]　Eiji Takimoto[1]　Shoichi Saito[2]　Koichi Mouri[1]

Abstract: Malware analysis is important for anti-malware. General malware analysis is carried out in the order of dynamic analysis and static analysis. However, in the present circumstances, the results of dynamic analysis has not cooperate static analysis. We propose static analysis assistance method utilizing the dynamic analysis log. In the proposed method provide assistance information of static analysis. It includes the API call information and the code of the malware on the memory acquired by dynamic analysis. In this paper, we describe static analysis assistance method that cooperates the system call tracer Alkanet and disassembler IDA.

Keywords: MWS，malware，dynamic analysis，static analysis，api trace

1. はじめに

近年，マルウェアによる情報漏洩などのインシデントが多発している [1] [2]．インシデントへの対応と防止のためには，マルウェアの解析を行い，マルウェアの持つ機能や挙動の理解が必要である．マルウェアの解析手法は，表層解析，動的解析，静的解析の 3 つのプロセスに分類される [3]．表層解析は，マルウェアを実行せずに得られる情報を用いたファイルレベルの解析を行なう．アンチウイルスソフトを用いたスキャンなどのシグネチャによるパターンマッチングが該当する．表層解析は，短時間でマルウェアの情報を取得できるが，得られる情報が限定的である．動的解析は，マルウェアを実行させ，その挙動を解析する．実行時のトレース対象は，API，システムコールなどがある．トレース対象によって解析結果の粒度は異なるが，静的解析に比べて荒い．しかし，実行するだけで自動的にマルウェアの挙動を解析することができるため，短時間での解析が可能である．また，コードの暗号化・難読化の影響を受けないといった利点がある．以上より，マルウェアの挙動の概要を把握することに向いている．静的解析は，逆アセンブルなどのリバースエンジニアリングを行い，マルウェアを手動で解析する．静的解析は手動で解析するため，解析者に高度な技術が必要であることに加えて，難読化等の解析妨害技術の影響を受けやすく，解析時間も動的解析に比べて長くなる．しかし，動的解析では解析するこ

[1] 立命館大学
Ritsumeikan University
[2] 名古屋工業大学
Nagoya Institute of Technology
[a)] snakajima@asl.cs.ritsumei.ac.jp

とができない，マルウェアの暗号化アルゴリズムや独自プロトコルなどの情報を解析することができる．このため静的解析は，マルウェアの特定の機能を詳細に解析することに向いている．静的解析では，マルウェアの持つ解析妨害機能に対して対処できない問題が多くあることが知られている [4] [5]．特に，マルウェアが静的解析妨害のために，API の呼び出しを難読化する場合や，実行時にのみ存在するコードを実行した場合は，解析作業にマルウェア解析に関する高度な知識・技術が必要となり，解析の難易度が上昇する．

前述のとおり，各解析プロセスには，利点と欠点がある．動的解析は，短時間，低コストで解析ができるが，詳細な解析を行うことが出来ない．静的解析は，解析妨害機能の影響を受けやすく，解析の難易度が高い．よって動的解析でマルウェアの挙動の概要を把握し，静的解析が必要な特定の場合のみ静的解析することで，効率的なマルウェア解析ができる．しかし，現状のマルウェア解析では，解析技術を持った解析者が，動的解析の結果を元に判断して静的解析を行っているため，動的解析ログを活用できるかは，解析者のスキルに依存する部分が大きい．そのため，一定のスキルを持つ解析者のみしかマルウェアを静的解析することができず，日々増加する多様なマルウェアへの対処が迅速に行えていない．多くの解析者が効率的にマルウェア解析を行うには，動的解析を静的解析と連携させる必要がある．

そこで本論文では，効率的なマルウェア解析のために，動的解析ログを活用した静的解析補助手法を提案する．提案手法では，システムコールトレーサ Alkanet [6] を用いて，API 呼び出し情報とマルウェアの実行に関する全てのコードを取得する．マルウェアの実行に関する全てのコードを取得することで，静的解析に必要なコードを取得する作業を省略することが可能となる．逆アセンブラ IDA [7] のプラグインを用いて，API 呼び出し情報を逆アセンブルコードと対応付ける．これにより，実行時の API 呼び出し情報を静的解析時に確認することが可能となる．これらにより，動的解析ログを活用した静的解析補助を実現する．

提案手法のうち，Alkanet のシステムコールログを元に，逆アセンブルコードに API 呼び出し情報を結びつける機能の実装を行った．実際のマルウェアを解析することで，提案手法を評価し，従来の解析手法で困難であった，動的解析によって取得したマルウェアの挙動情報と逆アセンブルしたコードの対応付けることによる，動的解析ログを活用した静的解析補助手法の有効性を示した．本論文の貢献を以下に示す．

- 動的解析ログを静的解析に利用し，静的解析を補助する手法を提案したこと
- 動的解析ログに API 呼び出し情報と API 呼び出し元を記録することで，静的解析する範囲を絞り込むことが可能となったこと
- 逆アセンブルコードに動的解析ログを対応付けることで，実行時の API 呼び出し情報を参照した静的解析を実現したこと

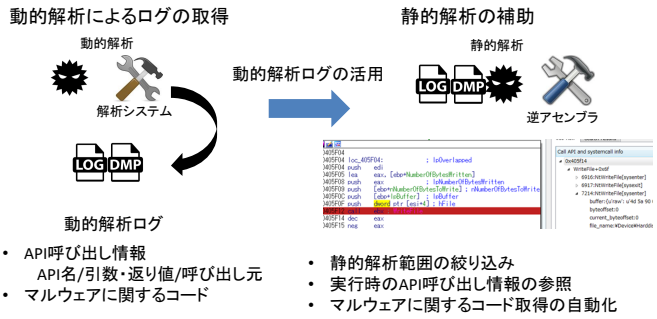

図 1　提案手法

本論文では，2 章で提案手法について述べ，3 章で提案手法の設計・実装について述べる．4 章で，提案手法の評価について述べ，5 章で関連研究について述べる．最後に，6 章で本論文をまとめる．

2. 提案手法

2.1 静的解析補助に必要な要件

静的解析で課題となっている点を以下に示す．これらが静的解析補助に必要な要件となる．

(1) 静的解析の必要な範囲の絞り込み
(2) 実行時の API 呼び出し情報の参照
(3) マルウェアに関するコードの取得の自動化

静的解析は，マルウェアの動作しない範囲や，動的解析の解析結果だけでは不十分な詳細情報を解析するために行う．そのため，API 呼び出しを元に静的解析の必要な範囲を絞り込む．ファイル作成，プロセス操作，レジストリの設定，外部への通信などのマルウェアの動作には API の呼び出しが必要となるため，API 呼び出しはマルウェアの挙動を解析する上での一つの指標となる．しかし，マルウェアの API 呼び出しは難読化されている場合があり，静的解析では容易に解析範囲を絞ることができない．また静的解析では，実行時に代入されるレジスタの値などを逆アセンブルコードから読み解かなければ，呼びだされた API 名や引数などの挙動に関する情報が把握できない．この作業は要求されるスキルが高く，静的解析の解析時間の増加，難易度の上昇の原因となっている．他にも，実行時にのみ存在するマルウェアに関するコードがある場合は，静的解析前にこれらのコードを取得する作業が必要となる．この作業は本来の解析作業ではないが，マルウェアが動的にメモリ上にコードを展開する場合や，子プロセスを生成する場合には，静的解析に必要なコードを取得するために必要な作業である．この作業に時間がかかれば，本来の解析作業が遅れる原因となる．

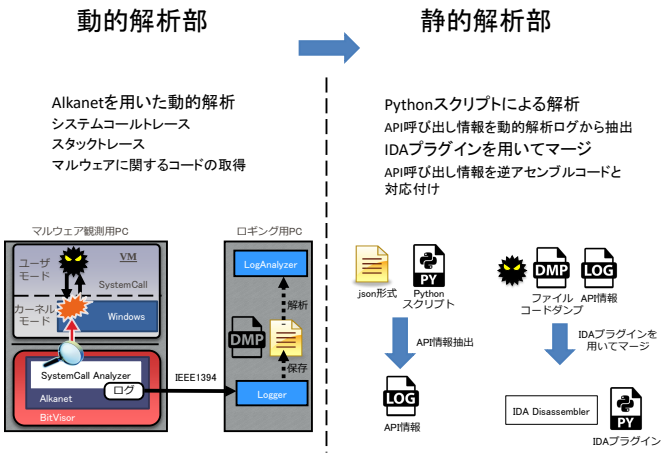

図 2　提案手法の設計

2.2　提案手法の概要

2.1 節で述べた静的解析の課題点は動的解析時の情報を用いることで解決できる．しかし，現状では動的解析の結果と，静的解析作業に十分に連携できているとは言えない．そのため，動的解析時に解析した情報を再び静的解析で解析しなければならない．

提案手法を図 1 に示す．動的解析では，API 呼び出し情報を記録し，マルウェアの実行に関する全てのコードを保存する．API 呼び出し情報として呼びだされた API 名，引数・返り値，呼び出し元を記録する．静的解析では，動的解析ログを元に 2.1 節で述べた要件を満たす静的解析補助を実現する．(1)(2) は，動的解析時に API 呼び出し情報と API 呼び出し元を記録することで実現する．呼び出し元を記録することで，マルウェア解析者が動的解析ログを確認し，必要に応じてその挙動に関する箇所を静的解析することが可能となる．また静的解析時に，逆アセンブルコードの CALL 命令に対応する API の呼び出し情報を対応付け表示することで，実行時の API 呼び出し情報を参照することが可能となる．(3) は，動的解析時にマルウェアが動的に展開したコードや作成したファイルなど，マルウェアの実行に関する全てのコードを取得することで実現する．これにより，静的解析に必要なコードを取得する作業を自動化し，本来の解析作業を進めることが可能となる．

3.　設計・実装

3.1　設計

図 2 に提案手法の設計を示す．提案手法は，静的解析部と動的解析部で構成される．動的解析部は，我々が開発しているマルウェア解析のためのシステムコールトレーサ Alkanet を用いる．動的解析部では，システムコールトレース，スタックトレース，マルウェアの実行に関する全てのコードの取得を行う．システムコールログは API 呼び出し単位にまとめ，呼び出し元とともに確認できるようにする．静的解析部は，ログ解析を行う python スクリプトと逆アセンブラ IDA で構成される．静的解析部では，動的解析部で記録した Alkanet のログを加工し，静的解析を補助する．まず，python スクリプトで Alkanet のログから API 呼び出し情報として，呼び出した API 名と API 呼び出しに対応するシステムコール情報を抽出する．そして，IDA プラグインを用いて，抽出した API 呼び出し情報を逆アセンブルコードと対応付ける．

表 1　Alkanet のログの各項目

項目	意味
No.	ログ番号
Time	システムコール記録時の CPU 時間
Cid	システムコールを発行したスレッドの PID と TID
Name	プロセス名
Type	sysenter か sysexit かを示す
Ret	返り値 (sysexit 時のみ)
SNo.	システムコール番号
Note	システムコール引数解析情報
StackTrace	スタックトレース情報

3.2　動的解析部

動的解析部では，Alkanet を用いた動的解析を行う．Alkanet で取得可能な情報は以下の 3 つである．

- システムコールトレース
- スタックトレース
- マルウェアに関するコード

図 3 は，Windows API の WriteFile 呼び出し時の Alanet のログである．図の各項目の意味について表 1 にまとめる．

Alkanet では，主に呼び出したシステムコールの詳細，システムコールを呼び出したプロセスの情報を記録する．図のスタックトレース情報について詳細に述べる．Alkanet は，システムコールトレース時のスタックトレースの記録に対応している [8]．スタックトレースから，関数呼び出し階層を取得する．この関数呼び出し階層から，システムコールトの呼び出元となる API を特定することができる．また，API の呼び出し元も同様にスタックトレースから取得することができる．スタックトレースの 1 行目では，システムコール記録時のスタックポインタ SP，スタックの上限と下限を示す StackBase と StackLimit を出力している．2 行目以降は，取得したリターンアドレスとそれを含むスタックフレームのベースポインタに関する情報である．戻り先については以下の情報を取得している．

API　マッピングされていれているファイルのエクスポートテーブルおよびシンボル情報から得た API 名とその API の先頭アドレスからのオフセットを示す．シンボル情報が存在しない場合，”-” を出力する．

Writable　リターンアドレスを含むページが書き込み可を出力する．この値は PTE(ページテーブルエントリ) か

```
No.:3951  Time:156673305  Type:sysenter  SNo.:112 (NtWriteFile)  Cid :70c.2d8  Name:sample.exe
Note:
  file_name: \Device\HarddiskVolume1\Documents and Settings\snakajima\Desktop\test.txt
  current_byteoffset:0  p_buffer:0x12fb48  length:0x4  byteoffset:0
  buffer:
         raw:41 41 41 41    utf16:\u4141\u4141
StackTrace:
  SP: 12f5d0, StackBase: 130000, StackLimit: 126000
  [00] <- 7c94df6c (API: NtWriteFile+0xc, Writable: 0, Dirty: 0, VAD: {7c940000--7c9dc000, ImageMap: 1,
            File: "\WINDOWS\system32\ntdll.dll"}), SP: 12f5d0
...
  [05] <- 7c817067 (API: BaseProcessStart+0x23, Writable: 0, Dirty: 0, VAD: {7c800000--7c933000, ImageMap: 1,
            File: "\WINDOWS\system32\kernel32.dll"}), BP: 12ffc0
```

図 3　WriteFile 呼び出し時の Alanet のログ

ら取得する．x86 アーキテクチャの PTE では，1 ビット目がこの値である．

Dirty　リターンアドレスを含むページが書き換えられたかを出力する．この値も Writable と同様に PTE から取得する．x86 アーキテクチャの PTE では，6 ビット目がこの値である．

VAD　Windows は，VAD(Virtual Address Descriptor) と呼ばれるデータ構造で，プロセスのメモリ領域を管理している [9]．VAD は，プロセスがメモリを確保するときや，ファイルをマッピングするときに作成される．VAD が保持する情報は，管理するアドレスの範囲や，その範囲のデフォルトのページ保護属性，ファイルのマッピングの有無，マッピングされている場合はそのファイルの情報などがある．各 VAD は，同じプロセスのアドレス空間を管理する他の VAD と連結され，平衡二分探索木を構成する．プロセスオブジェクトが持つ EPROCESS 構造体は，VAD ツリーのルートノードへのポインタを持つ．したがって VAD ツリーを辿ることで，リターンアドレスを含むメモリ領域にマップされているファイルの情報を得ることができる．さらに，マッピングされた実行ファイルのエクスポートテーブルやシンボル情報を用いて，リターンアドレスがどの API のものであるかも取得できる．

Alkanet は，システムコールトレースを目的に開発されているため，解析対象のマルウェアが作成するファイルや動的に確保したメモリ領域を取得する機能はない．そのため，マルウェアに関するコードの取得は，Alkanet に追加実装を行うことで可能となる．この機能については，実装方法についての検討はできているが，現段階では未実装である．以下に実装方法の検討について述べる．マルウェアが作成するファイルの取得は，NtWriteFile のトレース時にその引数から対象となるファイルを特定し，ダンプすることで実現する．動的に展開されるコードの取得は，スタックトレース記録時に行う．動的展開かつ実行された領域はスタックトレースのエントリの Dirty ビットが 1 になるため，その領域をダンプすることで実現する．また，Alkanet はマルウェア解析妨害技術の他プロセスへのコードインジェクションにも対応しているため，コードインジェクションしたコードの取得も可能である．

3.3　静的解析部

静的解析部では，python スクリプトで Alkanet のログから API 呼び出し情報を抽出し，IDA プラグインを用いて逆アセンブラコードと対応付ける．Alknaet はシステムコールトレーサであるため，API 呼び出しに関する情報を直接取得することができない．そこでシステムコールトレースの引数解析情報と返り値をマルウェアが呼び出した API と対応付けることで，API 呼び出し時の挙動の参考情報とする．システムコールトレースの利点としては，全ての API は最終的にシステムコールを呼び出すため，網羅性が高い点が挙げられる．しかし，API トレースとシステムコールトレースでは，記録される情報の粒度が異なるため，今後の検討課題とする．

MWS Datasets 2016 [10] に含まれる BOS2014 に活動が記録されているマルウェアを Alkanet で解析したログを用いて，提案手法の静的解析部の実装を説明する．当該検体は，子プロセスの作成，DLL の作成及び読み込み，動的コード展開を行う検体を選択した．前述の検体を c13.exe として実行し，Alkanet で解析したところ，starter.exe と splash_screen.dll を作成し，子プロセスとして starter.exe を実行した．さらに starter.exe は作成した splash_screen.dll のロードと動的コード展開を行う．

3.3.1　API 呼び出し範囲の取得

Alkanet を用いて提案手法を実現するには，Alkanet のシステムコールログから API 呼び出しに関する情報を取得する必要がある．そのため，python スクリプトを用いて，Alkanet のログから API 呼び出し範囲を取得する．Alkanet のスタックトレース中の VAD のファイルマッピング情報から，どのエントリがどのファイルに由来するものかを特定できる．マップされているファイルが解析対象のバイナ

```
"580": {"proc_image_name": "c13.exe",  "vad_end": "0x435000",  "vad_start": "0x400000"},    (a)
...
"812": {
       ...
       "proc_image_name": "starter.exe",
       "vad_file_info": [   (b)
           {"vad_filename": "\\DOCUME~1\\ADMINI~1\\LOCALS~1\\Temp\\RarSFX0\\starter.exe",
            "vad_file_end": "0x407000",     "vad_file_start": "0x400000"},
           {"vad_filename": "\\DOCUME~1\\ADMINI~1\\LOCALS~1\\Temp\\RarSFX0\\splash_screen.dll",
            "vad_file_end": "0x10005000",    "vad_file_start": "0x10000000"}],
       "dirty_memory": [    (c)
           {"dirty_memory_end": "0xa1b000",     "dirty_memory_start": "0x9f0000"},
           {"dirty_memory_end": "0x9f0000",     "dirty_memory_start": "0x8f0000"}]
...
```

図 4　API 呼び出し元となる範囲の抽出

リであれば，そのエントリのリターンアドレスは，解析対象のバイナリ中のアドレスとなる．スタックトレースをシステムコールスタブ側から辿り，以下の 3 つをリターンアドレスとして持つ最初に現れたエントリが API のリターンアドレスのエントリであり，その直前のエントリが，解析対象のバイナリが呼び出した API に関するエントリになる．

(1) 解析対象のバイナリがマップされている領域
(2) 解析中に作成されたファイルがマップされている領域
(3) 解析対象のプロセスが実行中に確保した領域かつ実行された領域

解析対象のバイナリがマップされている領域は，解析時に解析対象のプロセスがマップされている領域である．解析対象のプロセス名を元にスタックトレースの VAD から取得する．解析中に作成されたファイルがマップされている領域が必要になるのは，解析対象のプロセスが子プロセスの作成や，DLL のロードをした場合である．WriteFile で作成されたファイルがスタックトレースの VAD に現れた場合，解析対象のバイナリが作成したファイルを実行したと判定し，その領域を取得する．解析対象のプロセスが実行中に確保した領域かつ実行された領域が必要になるのは，動的コード展開が行われた場合である．動的に確保された領域でコードが実行されると，その領域がリターンアドレスになるため，取得する必要がある．スタックトレースの Dirty ビットが立っているエントリの領域を取得することで，解析対象のプロセスが実行中に確保した領域かつ実行された領域を取得できる．

(1)〜(3) の領域の範囲を Alkanet のログから抽出した結果を図 4 に示す．図中の (a) は解析時に解析対象のプロセスがマップされている領域となる．解析対象のプロセス名とスタックトレースの VAD のファイル名が一致するエントリを検索し，VAD のマップされている領域を抽出した．c13.exe のバイナリがマップされている領域が抽出できた．(b) は，システムコールログから，解析対象のプロセスが作成したファイルを抽出し，そのファイル名とスタックトレースの VAD のファイル名が一致するエントリを検索し，VAD のマップ領域を抽出した．解析対象のプロセスが作成した子プロセスである starter.exe がマップされている領域と starter.exe がロードした DLL がマップされている領域が抽出できた．(c) は，解析対象のプロセスが発行したシステムコールログから，スタックトレースの VAD の Dirty ビットが 1 のエントリの VAD のマップ領域を抽出した．stater.exe が動的に確保し，実行した領域が抽出できた．

3.3.2　API 呼び出しとシステムコールログの結び付け

抽出した (a)〜(c) の領域をリターンアドレスとして，その直前のエントリを解析対象のプロセスの API 呼び出しとして抽出した，また，同じリターンアドレスを持つ AP 呼び出しごとに，システムコールログを結びつける．結びつけた結果を図 5 に示す．(i) は，リターンアドレスを示す．(a)〜(c) の範囲を持つリターンアドレス，それぞれに対応する CALL で呼び出される API の範囲のリターンアドレスが (ii) である．(ii) で示した API 内で呼び出されるシステムコールログ群が (iii) である．基本的にシステムコールログは sysenter と sysexit の対で一組となる．API 内で複数のシステムコールが呼ばれる場合，一つの API 呼び出しに対して，複数のシステムコールログが結び付けられる．

3.3.3　API 呼び出し情報と逆アセンブルコードの対応付け

3 章で述べた (1)〜(2) の領域に対して，リターンアドレスごとに，API 呼び出しとシステムコールログの結び付けた結果を逆アセンブルコードへ対応付ける．python スクリプトが Alkanet のログから抽出した API 呼び出し情報とマルウェアに関する逆アセンブルコードを対応付ける IDA プラグインを実装した．図 6 に API 呼び出し情報と逆アセンブルコードの対応付けた結果を示す．ログに含まれるリターンアドレスの一つ前の命令の CALL 命令を色付けし，API 呼び出し情報があることを示す．その CALL 命令で呼び出される API 名と，その API によって呼び出されるシステムコールのログをプラグイン側に表示する．また，プ

```
"0x4059b3": {    (i)
     "CreateFileW+0x1b6":    (ii)
               [ {"name": "NtCreateFile",  "no": 3648, "type": "sysenter",  (iii)
                "optional": {
                     "file_name": "\\??\\C:\\c13.exe"}},
                  ...
```

図 5　API 呼び出しとシステムコールログの結び付け

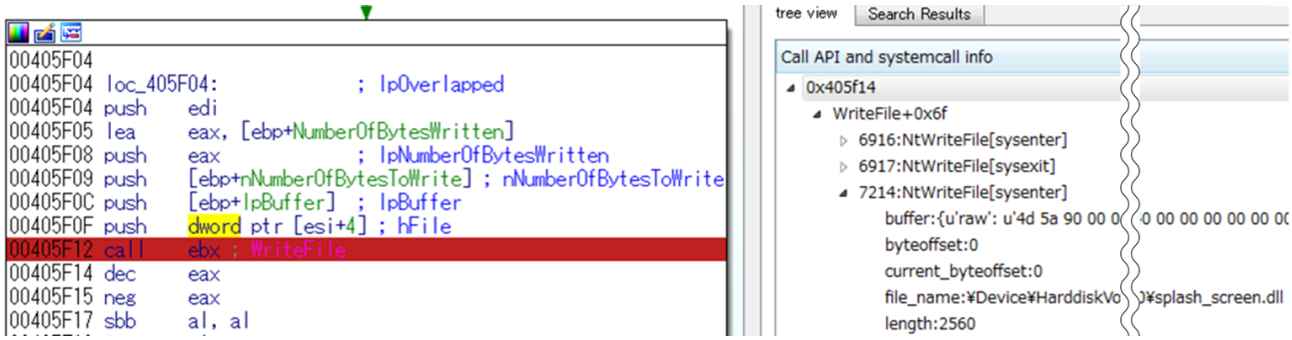

図 6　API 呼び出し情報と逆アセンブルコードの対応付け

ラグイン側のリターンアドレスを選択して，API の呼び出し元のアドレスへとジャンプすることもできる．リターンアドレスに対応する逆アセンブルコードの Call 命令に対して，上述のとおり正しく結び付けることができた．

4. 評価

4.1 評価目的と検体

提案手法の有効性を検証するため，マルウェアを用いた評価を行った．性能評価対象のマルウェアには，MWS Datasets 2016 [10] に含まれる BOS2014，BOS2016 に活動が記録されているマルウェアを用いた．当該検体は，3 章で述べた検体 c13.exe と API 呼び出しの難読化を持つ検体を選択した．以下の場合に対して，動的解析ログを活用し，静的解析を補助することで，提案手法の有効性を示す．

- API 呼び出し箇所の絞り込み
- API 呼び出し情報の参照

4.2 評価結果

4.2.1 API 呼び出し箇所の絞り込み

提案手法によって，動的解析時にプロセスの実行フローと，それぞれのプロセス空間において，マルウェアのコードがマップされている範囲を取得することができる．c13.exe の実行フローとそれぞれのマップアドレスを図 7 に示す．またマルウェアの挙動を抽出した動的解析ログを図 8 に示す．これらの情報を元に API 呼び出し箇所の絞り込みを行う．例えば，c13.exe がプロセスを作成する挙動を詳細に静的解析する場合を想定する．

まず，図 7 から c13.exe の解析対象範囲は 0x400000〜0x435000 だとわかる．そして，図 8 の動的解析ログから ShellExecuteExW 関数を呼び出し，starter.exe を子プロセスとして実行していることがわかる．この ShellExecuteExW 関数呼び出し時のリターンアドレスが 0x40c673 となっていることから，このアドレス前後の逆アセンブルコードを解析することで，c13.exe がプロセスを作成する挙動を解析することができる．このように，動的解析ログに API の呼び出し元を含めることで，その挙動に関する箇所を静的解析することが可能となった．

4.2.2 API 呼び出し情報の参照

インポートテーブルを用いた API 呼び出し時に，動的解析時の API 呼び出し情報を参照するケースについて述べる．通常，PE 形式の実行ファイルであれば，インポートテーブルを用いた API 呼び出しを行う．これはマルウェアの場合も同様である．インポートテーブルを用いた API 呼び出しの場合，IDA で逆アセンブルを行えば，インポート時に解決される API 名・引数名が逆アセンブルコードにコメント付けされ，参照することができる．しかし，引数として渡された数値，文字列等は逆アセンブルコードを読み，解析する必要がある．

提案手法では，API によって呼び出されたシステムコールと引数解析の情報を参照可能になる．API の引数を取得することができないが，API が呼び出したシステムコールとその引数から，API 呼び出しによって行われた挙動を確認することができる．図 6 はマルウェアのインポートテーブルを用いた API の呼び出しに対して提案手法を適用している．このコードでは，マルウェアが WriteFile を呼び出しファイルを作成する箇所である．提案手法を適用することで，WriteFile が呼び出した NtWriteFile が splash_screen.dll を作成していることが確認できる．本来であれば，API 呼び出しによるマルウェアの挙動を API 呼び出し前後の逆アセンブルコードを解析して把握する必要がある．しかし提

```
c13.exe[0x400000-0x435000]
└── starter.exe(Create)
  ├── File(\DOCUME~1\ADMINI~1\LOCALS~1\Temp\RarSFX0\splash_screen.dll)[0x10000000-0x10005000]
  ├── File(\DOCUME~1\ADMINI~1\LOCALS~1\Temp\RarSFX0\starter.exe)[0x400000-0x407000]
  ├── dirty_memory [0x8f0000-0x9f0000]
  ├── dirty_memory [0x9f0000-0xa1b000]
  └── svchost.exe(Create)
      ├── dirty_memory [0x8d0000-0x8fb000]
      └── dirty_memory [0x90000-0xad000]
```

図 7 プロセスの実行フローとマルウェアのコードマップ情報

```
------------------------------------------------------------------------
Created process
------------------------------------------------------------------------
{"ShellExecuteExW+0x67":
    { "0x40c673": {
         "systemcall": [
            {"status": 0,  "proc name": "starter.exe",  "name": "NtCreateProcessEx",  "proc_pid": 812
             "file name": "\\Device\\HarddiskVolume1\\DOCUME~1\\ADMINI~1\\LOCALS~1\\Temp\\RarSFX0\\starter.exe",
...
```

図 8 マルウェアの挙動を抽出した動的解析ログ

案手法によって，API 呼び出し前後の逆アセンブルコードを解析することなく，実行時の API 呼び出しによって行われた挙動を把握することができる．

難読化された API 呼び出し時に，動的解析時の API 呼び出し情報を参照するケースについて述べる．API 呼び出しを難読化する手法として，DLL の動的ロードを用いた API 呼び出しがある．この手法では，実行時に LoadDll 関数を使用して呼び出したい API を含む dll を読み込む．そして GetProcAddress 関数を使用して，DLL から API 名を元に API がマップされているアドレスを取得する．取得したアドレスを直接呼び出すことで，API を呼び出すことができる．マルウェアは，この処理を難読化することで，呼び出した API を特定できなくし，静的解析を妨害する．マルウェアが API 呼び出しの難読化を持つ場合，逆アセンブル後に難読化解除を行う処理を解析し，呼びだされた API に関する情報を特定する必要がある．図 9 に提案手法適用後の逆アセンブルコードを示す．逆アセンブルコードから，実行時に解決したアドレスを Call して API 呼び出しを行っていることが確認できる．本来であれば，IDA で逆アセンブルしただけでは，呼びだされた API に関する情報は参照できない．提案手法を適用すると，呼びだされた API 及び，API によって呼び出されたシステムコールと引数の情報が参考できていることが確認できる．提案手法によって，逆アセンブルしただけでは把握できない API 呼び出し情報が参照可能となる．

5. 関連研究

動的解析のログを静的解析に活用するツールとして，funcap [11] と IDA splode [12] がある．funcap は，IDA Debugger で実行ファイルをデバッグし、デバッグ時の情報を逆アセンブル結果に出力する．IDA splode は，Intel PIN と IDA を利用し，メモリアクセスと実行時のメタデータを記録する．どちらも実行時の情報を静的解析で活用しているが，マルウェアと同じ環境内で動作するデバッガ・ツールを用いて実行時の情報の記録を行っている．マルウェア解析を考慮した場合，アンチデバッグの観点から，マルウェアの動作環境外から実行時の情報の記録を行うことが望ましい．また，どちらもプログラム単体の解析を目的としたツールを使用して動的解析を行っている．このため，システム全体を監視することができず，プロセスを越えた挙動は解析することができない．提案手法では，マルウェア解析を目的とした Alkanet を用いている．このため，これらの問題に対して対応ができている．

マルウェア解析コストの軽減を目指したシステムに egg [13] がある．動的解析で内部構造などの詳細な記録を取得することで，静的解析の軽減を実現している．egg は，マルウェアの実行を 1 命令単位で解析し、かつプロセスを越えて拡散するマルウェアを解析できる動的解析システムである．解析ログとして，API 引数の取得・コールグラフ・分岐情報を取得する．API 引数の解析機能を持ち，作成したファイルを取得する機能がある．しかし，動的解析時のログを使って静的解析を補助する機能は分岐情報のみに留まっている．また，マルウェアが持つアンチデバッグ機能については，考慮されているが，マルウェアと同じ環境内にドライバと補助プログラムを動作させ，解析を行うためマルウェアからの検知のおそれがある．egg では，API の呼び出し元や引数など，静的解析で活用できる情報を取得している．しかし，逆アセンブルコードに対応付けている

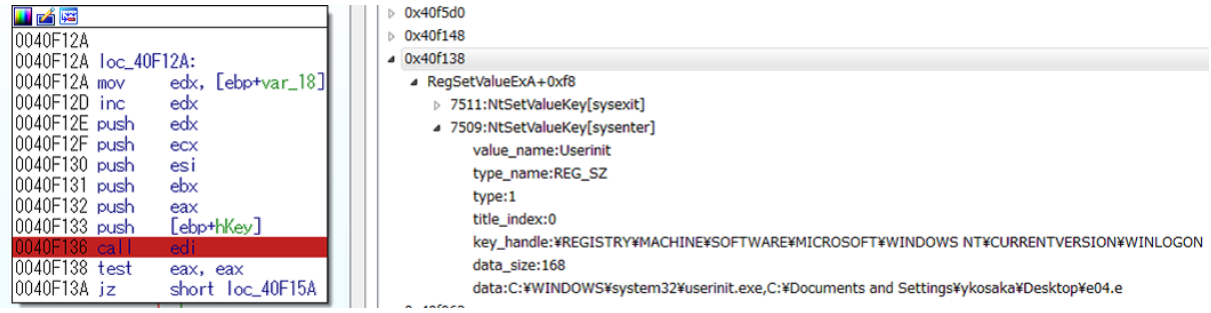

図 9 提案手法適用後の DLL の動的ロードを用いた API 呼び出し

情報は分岐情報のみである．提案手法では，静的解析補助のため，動的解析時のログを活用し，API 呼び出し情報を逆アセンブルコードと対応付けしている．また，Alkanet はマルウェア動作環境外からシステムコールトレースを取得するため，マルウェア特有のプロセスを越える挙動やアンチデバッグ機能に対応できている．

動的解析と静的解析を組み合わせたマルウェア解析手法の研究として，泉田らは展開型静的解析と動的解析を連携させたマルウェア解析手法 [14] を提案している．泉田らの手法では，静的解析と動的解析を連携させ，静的解析では得られない制御フローを動的解析で補うことを目的としている．動的解析では，メモリ書き込みを監視しておき，書き換えがあった場合に，静的解析時に解析不能な領域として再度，静的解析している．提案手法と同様に動的解析時にのみ展開されるコードを静的解析で活用している．この手法では，制御フローに注力して静的解析と動的解析を連携させているため，API 呼び出しに関する補助は行わない．API 呼び出しに着目した本研究とは明確に異なる．

6. おわりに

本論文では，動的解析時の API 呼び出し情報と，メモリ上のマルウェアのコードを取得し，静的解析を積極的に補助する手法について述べた．実装が完了した動的解析ログと逆アセンブルコードの対応付けが，正しく行えていることを確認した．提案手法によって，動的解析ログに API 呼び出し元を含めることで，静的解析する箇所の絞り込みが可能となった．また，マルウェアの API 呼び出しに対して，実行時の API 呼び出し情報を対応付けることで，静的解析の補助を実現した．今後は，実装方法を検討済みの，動的解析時にメモリ上のマルウェアのコードを取得する機能を実装する．また，マルウェア解析環境として，Windows10 を対象とした Alkanet [15] を用いた提案手法の実現を目指す．

参考文献

[1] トレンドマイクロ株式会社：標的型サイバー攻撃分析レポート 2015 年版 ～「気付けない攻撃」の高度化が進む ～(online)，入手先 ⟨https://app.trendmicro.co.jp/doc_dl/select.asp?type=1&cid=161⟩ (2016.07.05)

[2] 株式会社カスペルスキー：BLUE TERMITE. ブルーターマイト -日本を標的にする APT 攻撃 (online)，入手先 ⟨http://media.kaspersky.com/jp/Kaspersky_BlueTermite-PR-1013.pdf⟩ (2016.07.05)

[3] 八木 毅，青木 一史，秋山 満昭，幾世 知範，高田 雄太，千葉 大紀：実践サイバーセキュリティモニタリング，コロナ社 (2016)

[4] Andreas Moser, Christopher Kruegel, and Engin Kirda.: Limits of Static Analysis for Malware Detection, *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, pp. 421-430 (2007)

[5] Ilsun You, Kangbin Yim.: Malware Obfuscation Techniques: A Brief Survey, *Broadband, Wireless Computing, Communication and Applications (BWCCA), 2010 International Conference*, pp. 297-300 (2010)

[6] 大月 勇人，瀧本 栄二，齋藤 彰一，毛利 公一: マルウェア観測のための仮想計算機モニタを用いたシステムコールトレース手法，情報処理学会論文誌，Vol. 55，No. 9，pp. 2034-2046，(2014)

[7] Hex-Rays: IDA，入手先 ⟨https://www.hex-rays.com/products/ida/⟩ (2016.07.19)

[8] 大月勇人，瀧本栄二，齋藤彰一，毛利公一：Alkanet におけるシステムコールの呼出し元動的リンクライブラリの特定手法，コンピュータセキュリティシンポジウム 2013 論文集，Vol. 2013, No. 4, pp. 753-760 (2013)

[9] B. Dolan-Gavitt: The VAD tree: A process-eye view of physical memory, Digital Investigation, Vol. 4, pp. 62-64 (2007)

[10] 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（CSEC），Vol.2016-CSEC-74, No.17, pp. 1-8, (2016)

[11] ANDRZEJ DERESZOWSKI: funcap, GitHub, GitHub Inc., 入手先 ⟨https://github.com/deresz/funcap⟩ (2016.07.05)

[12] ENDGAME: IDA-splode, GitHub, GitHub Inc., 入手先 ⟨https://github.com/zachriggle/ida-splode⟩ (2016.07.05)

[13] Satoshi TANDA: “egg” - A Stealth fine grained code analyzer, *Recon2011*, (2011)

[14] 泉田 大宗，森 彰，二木 厚吉：展開型静的解析と動的解析を連携させたマルウェア解析手法，コンピュータ ソフトウェア，Vol. 29，No. 4，pp. 199-218，(2012)

[15] 大月勇人，中野進，明田修平，瀧本栄二，齋藤彰一，毛利公一：Windows10 x64 環境を対象とするシステムコールトレーサの実現手法，コンピュータセキュリティシンポジウム 2015 論文集，Vol. 2015, No. 3, pp. 839-846 (2015)