情報処理学会論文誌 Vol.59 No.2 800–811 (Feb. 2018)

推薦論文

動的解析ログを活用した静的解析補助手法

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

受付日 2017年3月22日, 採録日 2017年11月7日

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，静的解析は動的解析で得られた情報を活かせていない．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マルウェアの実行に関するすべてのコードを取得する必要がある．以上の背景から，本論文では動的解析時のAPI呼び出し情報とマルウェアの実行に関するすべてのコードを取得し，静的解析を補助する手法を提案する．さらに提案手法をシステムコールトレーサAlkanetと逆アセンブラIDAへ適用し，マルウェア25検体を解析することで提案手法の有効性を示す．

キーワード：マルウェア，静的解析，動的解析，API呼び出し

Static Analysis Assistance Method Utilizing the Dynamic Analysis Log

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

Received: March 22, 2017, Accepted: November 7, 2017

Abstract: Malware analysis is important for anti-malware. Generally, static analysis is performed after dynamic analysis. However, static analysis has not utilized the information acquired by dynamic analysis. In this paper, we propose a static analysis assistance method utilizing the dynamic analysis log. The dynamic analysis log includes the API Call information and all codes involved in malware. We apply the proposed method to the system Call tracer "Alkanet" and disassembler "IDA". Furthermore, we show the effectiveness of the proposed method by analyzing 25 specimens of malware.

Keywords: malware, staticanalysis, dynamicanalysis, API call

1. はじめに

近年，日本を狙った標的型サイバー攻撃が増加している [1], [2]．また，2016年のランサムウェアによる被害報告件数は，過去最大であった [3]．こうしたインシデントの対応と防止のためにはマルウェア解析が重要な役割を果たす．マルウェア解析によってマルウェアの機能や挙動を明らかにすることで，流出した情報などの被害の特定やウイルスの駆除，暗号化されたファイルの復元ツールの作成，マルウェアの検出ソフトの作成などを行うことができる．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つに分類される [4]．

表層解析は，既知のマルウェアの場合に有効である．アンチウイルスソフトによるスキャンやシグネチャを用いたパターンマッチングなど，マルウェアを実行せずに得られる情報を使用した解析を行う．ただし，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，その挙動をトレースして解析する．トレースの粒度はAPI，システムコール，機

[1] 立命館大学
Ritsumeikan University, Kusatsu, Shiga 525–8577, Japan
[2] 名古屋工業大学
Nagoya Institute of Technology, Nagoya, Aichi 466–8555, Japan
[†1] 現在，NTTセキュアプラットフォーム研究所
Presently with NTT Secure Platform Laboratories
[a)] snakajima@asl.cs.ritsumei.ac.jp

本論文の内容は2016年10月のコンピュータセキュリティシンポジウム2016/マルウェア対策研究人材育成ワークショップ2016にて報告され，同プログラム委員長により情報処理学会論文誌ジャーナルへの掲載が推薦された論文である．

械語命令などがある．動的解析のログは，既知のマルウェアと同様の挙動を行うマルウェアを亜種として検知することや解析者がマルウェアの動作概要を把握することなどに利用される．動的解析の粒度は，基本的に後述の静的解析に比べて粗い．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．静的解析は，動的解析では解析することができないマルウェアの暗号化方法やマルウェア独自の通信プロトコルなどを詳細に解析することができる．しかし，マルウェアを逆アセンブルして手動で解析するため，解析に高度な知識・スキルが必要となる．また，動的解析に比べてマルウェアの解析妨害の影響を受けやすい．API 呼び出しが難読化されている場合は，逆アセンブルコードからマルウェアが呼び出す API の特定が困難になる．さらに，マルウェアの実行によってファイルやメモリへ展開されるコード（以下，感染コード）が存在する場合は，感染コードを取得をしなければ静的解析を行うことができない．以上より，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．

前述のとおり，それぞれの解析手法には利点と欠点がある．このため一般的なマルウェア解析では，解析の目的に応じて解析手法を選択したり組み合わせたりすることで解析する．静的解析を行う場合であっても，動的解析でマルウェアの挙動の概要を把握してから，静的解析で詳細に解析する．これは，動的解析と静的解析を組み合わせることでそれぞれの欠点を補い，より効率的なマルウェア解析が可能になるからである．現状のマルウェア解析では，高い解析技術を持った解析者が動的解析のログをもとにマルウェアの概要を理解し，静的解析の範囲を絞り込んだり，動的解析時の挙動を逆アセンブルしたコードと対応付けたりして解析する．すなわち，静的解析の解析精度は解析者のスキルに依存しており，一定以上の技術がなければ静的解析で有用な情報を得られない．

そこで，本論文では，動的解析ログを活用した静的解析補助手法を提案する．動的解析で静的解析を補助可能な解析ログを取得し静的解析の補助を行うことで，解析精度を解析者のスキルに依存せずに高めることができる．提案手法では動的解析時にログとして感染コード，API の呼び出し情報，および API の呼び出し元情報を記録する．動的解析時に感染コードを取得することで，静的解析時の感染コードの取得作業を省略できる．さらに，呼び出し元情報をもとに逆アセンブルコードと API 呼び出し情報を対応付けることで，動的解析時の API 呼び出しの情報を参照した静的解析が可能となり静的解析の補助ができる．この対応付けを行うツールを静的解析で用いられる逆アセンブラ IDA [5] のプラグインとして実装した．また，動的解析システムにシステムコールトレーサ Alkanet [6] を拡張して提案手法の実現に必要な動的解析ログを取得するように実装した．

本論文の貢献を以下に示す．

- 動的解析ログを利用し，静的解析を補助する手法を提案したこと
- 動的解析時に API 呼び出し情報と API の呼び出し元を記録することで，静的解析を行うべき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，動的解析時の API 呼び出し情報を参照した静的解析を実現したこと
- 動的解析時に感染コードを取得することで，静的解析時の感染コードの取得作業を省略できるようになったこと
- 提案手法をシステムコールトレーサ Alkanet と逆アセンブラ IDA へ適用し，マルウェア 25 検体を解析することで提案手法の有効性を示したこと

以下，本論文では 2 章でマルウェアの静的解析における課題について述べる．3 章で提案手法について，4 章で提案手法の実装について，5 章で提案手法の評価について述べ，6 章で関連研究について述べる．7 章で本論文をまとめる．

2. マルウェアの静的解析における課題

静的解析では，マルウェアを逆アセンブルし，実行時に代入されるレジスタの値やメモリの値を読み解くことでマルウェアの挙動を明らかにする．静的解析にはアセンブリ言語を正確に効率良く読むスキルが必要となる．しかし，一般的にアセンブリ言語は抽象度が低いため，習得は難しいとされる．また，マルウェアは静的解析妨害のために難読化されることが知られている [7], [8], [9]．

静的解析は，動的解析時にマルウェアが動作しない範囲や，動的解析範囲の解析結果だけでは不十分な詳細情報を得るために行われる．すなわち，静的解析ではマルウェア全体を解析する必要はなく，対象となる範囲のみを解析できればよい．しかし，解析範囲を絞ること自体が容易ではない．解析範囲を絞る方法として，マルウェアの API 呼び出しを指標とする方法がある．この場合，API 呼び出し時の引数をアセンブリ言語から解読する必要があり，これは前述のとおり解析者に高いスキルが要求される．また，API 呼び出しが難読化された場合は，逆アセンブルコードから API 呼び出し時の挙動を把握することが困難になり，解析時間が増加し，難易度も上昇する．さらに，マルウェアが感染コードを生成する場合は，マルウェアの解析対象となるファイルやメモリ領域を明らかにし，それらを取得しなければマルウェアの挙動の一部分しか解析することができない．しかし，解析妨害を意図して感染コードの生成を行っているため，感染コードの取得作業の難易度は高い．この取得作業に時間がかかってしまうと，本来の解析作業が遅れる原因となる．また，感染コードの生成が行われる

と複数のプロセス，ファイルが解析の対象となる．静的解析でこれらを特定する場合，すべての感染コードを順に解析する必要がある．マルウェアが生成するプロセス名や親子関係，生成されるファイルの関係性（以下，実行フロー）とプロセスのメモリ領域の中の感染コードのアドレス範囲（以下，コードマップ情報）が把握しにくい．

以上をふまえてマルウェアの静的解析における課題を以下にまとめる．

(1) 静的解析を行うべき範囲の絞り込みが困難であること
(2) API 呼び出し時の挙動を把握するための情報（API 名，引数など）が容易に参照できないこと
(3) 感染コードの取得の難易度が高く，時間がかかること

3. 提案手法

2 章で述べた課題をふまえて，静的解析の補助に必要な要件を検討した．以下に 3 つの要件を示す．

(1) 静的解析対象範囲の絞り込み
(2) 動的解析時の API 呼び出し情報の参照
(3) 感染コードの取得の自動化

本論文では，動的解析ログを活用することで，上記の静的解析の補助に必要な 3 つの要件を満たす静的解析補助手法を提案する．提案手法では，動的解析時に API 呼び出し情報（API 名や引数などの API 呼び出し時の挙動に関する情報），API の呼び出し元情報（API が呼び出された Call 命令のアドレス），および感染コードを記録する．感染コードとして，動的解析時にマルウェアが作成したファイルとメモリ上に展開したコードを取得する．

静的解析の補助に必要な要件 (1) は，静的解析対象範囲を絞るためのログを提供することで実現する．具体的には，実行フローとコードマップ情報を提供する．また，マルウェアの挙動を API 呼び出し情報と API の呼び出し元情報と関連付けたログを提供し，API がどの命令で呼び出されたかを把握できるようにする．マルウェア解析者がこれらのログを確認することで，静的解析時に必要に応じて解析したい挙動に関連する範囲を解析することが可能となる．また，実行されていない領域には API 呼び出し情報を対応付けることはできないが，動的解析で解析した領域とまだ解析できていない領域を区別することで解析者の負担を軽減することができる．

静的解析の補助に必要な要件 (2) は，逆アセンブルコードの Call 命令と API の呼び出し情報を対応付けることで実現する．逆アセンブルコードと対応付けて動的解析ログを提示することで，動的解析時に呼び出された API 呼び出し情報を参照しながら静的解析を行うことを可能とする．

静的解析の補助に必要な要件 (3) は，動的解析時に感染コードを自動で取得することで実現する．感染コードの取得作業を自動化することで，静的解析時に感染コードの取得作業を行わずに，本来の解析作業を進めることが可能となる．動的解析時に感染コードを取得することで，API の呼び出し元が実行ごとに変化する場合でも，動的解析ログに記録した API の呼び出し元情報と対応付いた感染コードを取得することができる．

静的解析補助の実現のためには，以下の要件を満たす動的解析システムが必要となる．

- マルウェアの API 呼び出し情報が記録できること
- API の呼び出し元を特定するための情報が記録できること
- 動的解析時にマルウェアの感染コードを保存できること

これらの要件を満たす動的解析システムであれば，提案手法を適用することができる．提案手法の設計は Windows のバージョンに依存しない．

4. 実装

4.1 概要

提案手法を実現する解析システムのソフトウェア構成を**図 1** に示す．解析システムは，動的解析部と静的解析部で構成される．

動的解析部では，マルウェア解析のためのシステムコールトレーサ Alkanet を用いて動的解析を行う（図 1 の (1)）．本実装では，Windows XP 32 bit を対象とした Alkanet を用いた．現状の実装では Windows XP 32 bit にのみ対応している．Alkanet は，マルウェアが発行したシステムコールとその引数を記録するシステムコールログとコードダンプログの取得を行う．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の感染コードを取得する機能はない．そこで，感染コードの元となるコードダンプログを取得する機能を Alkanet に追加実装した．システムコールログには，システムコール呼び出し時の関数呼び出し階層を記録したスタックトレースログ [10] が含まれている．また，システムコールログを解析し，実行フローとコードマップ情報を提供する．コードダンプログには，マルウェアによって作成されたファイルと

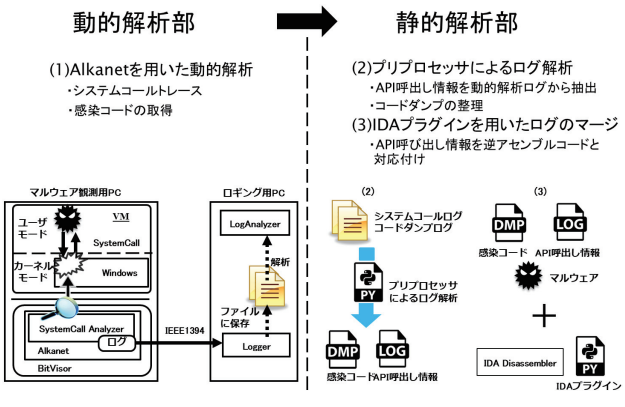

図 1 提案手法のソフトウェア構成
Fig. 1 Software configuration of the proposed method.

動的にメモリ上に展開され実行されたコードが含まれる．

静的解析部は，プリプロセッサ群と逆アセンブラ IDA，IDA のプラグインで構成される．プリプロセッサでシステムコールトレースとコードダンプログから，静的解析補助に必要な情報を抽出する（図 1 の (2)）．Alkanet はマルウェアが直接呼び出す API の引数や返り値を取得していないため，システムコールログの引数解析情報と返り値を API を呼び出す Call 命令と結び付けることで API 呼び出し情報とする．API プリプロセッサは，この結び付けを行う．また，動的解析で記録した API 呼び出し情報と静的解析時の逆アセンブルコードを対応付けるためには，API の呼び出し元を特定する必要がある．API プリプロセッサはスタックトレースログを解析し，API の呼び出し元の特定も行う．コードプリプロセッサは，取得したコードダンプログから感染コードの抽出を行う．最後に，抽出した API 呼び出し情報を IDA のプラグインを用いて逆アセンブルコードと対応付ける（図 1 の (3)）．このプラグインが Call 命令に対応する動的解析時の API 呼び出し情報を表示することで，IDA を使った静的解析を補助する．

本論文では，上記のうち動的解析部の感染コードの取得と静的解析部について新たに実装した．

4.2 動的解析部

動的解析部では，マルウェア解析のためのシステムコールトレーサ Alkanet を用いて動的解析を行う．Alkanet で記録可能な情報と Alkanet に追加した感染コードの取得について述べる．

4.2.1 システムコールログ

Alkanet がシステムコールログとして取得している情報を**表 1** に示す．Alkanet は，システムコールの発行に関する情報とシステムコールの引数，システムコールを呼び出したプロセスの情報，スタックトレース情報をシステムコールログとして記録する．

スタックトレース情報はシステムコールに到達するまでの関数呼び出し階層を記録する．この関数呼び出し階層から記録したシステムコールを呼び出した API を特定することができる．さらに，その API の戻り先も同様に特定することができる．なお，スタックトレース情報には戻り先のアドレスだけでなく追加情報として，リターンアドレスで示される戻り先のメモリ領域の PTE（Page Table Entry）から取得した書き込み可能か否かを示す Writable ビットと書き込まれたか否かを示す Dirty ビットの情報，Windows のプロセスのメモリ領域を管理している VAD（Virtual Address Descriptor）と呼ばれるデータ構造 [11] から取得した当該メモリ領域にマップされているファイルの情報を記録する．

動的解析が終わると，システムコールログを LogAnalyzer で解析し，システムコールの引数の値の意味や戻り先の API 名などの情報の付加や，マルウェアの実行フローとコードマップ情報を作成を行う．

表 1 Alkanet のログの各項目
Table 1 Each item of Alkanet's log.

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

4.2.2 コードダンプログ

マルウェアが感染コードを生成する場合，少なくとも以下のいずれか一方を行う．

- ファイルへのコード展開
- メモリ上へのコード展開

ファイルへのコード展開をする場合，マルウェアは NtWriteFile システムコールを用いる．NtWriteFile は，引数にファイルに書き込むバッファへのポインタと書き込む長さをとる．Alkanet は NtWriteFile をトレース対象にしているためシステムコールログ取得時に，これらの引数が示す領域のデータをコードダンプログとして取得する．

メモリ上へのコードを展開する場合，マルウェアはシステムコールを呼び出して動的なメモリ領域の確保や書き換えを行う．Alkanet は NtAllocateVirtualMemory や NtWriteProcessMemory を記録しているため，この挙動を記録することができる．しかし，マルウェアが動的に確保したメモリ領域にコードを展開する処理はシステムコールを介さずに行うことができる．このため，感染コードの取得はスタックトレース時に記録したリターンアドレスの示すメモリ領域の Dirty ビットを確認しながら，関数呼び出し階層をたどることで実現する．Dirty ビットが立っている領域は，メモリに変更が行われた領域である．この領域がコールスタック上へ現れた場合，その領域は実行時に変更された領域かつ実行された領域と判断することができる．この領域をシステムコール呼び出しごとに取得することで，マルウェアが展開後に実行したコードが取得できる．また，一度にシステム全体のダンプを取得するのではなくシステムコール呼び出しごとに取得することで，マルウェアが同じメモリ領域のコードを複数回を書き換える場合であっても，システムコール呼び出し時に展開されているコードを取得することができる．なお，Alkanet では Windows 上のすべてのプロセスのシステムコールログを記録しているため，展開領域が他プロセスの場合であって

```
[00] <- 7c94cf5c (API: NtAllocateVirtualMemory+0xc, Writable: 0, Dirty: 0,
        VAD: {7c940000--7c9dc000, ImageMap: 1, File: "\WINDOWS\system32\ntdll.dll"}), SP: 12fe10
[01] <- 7c809b32 (API: VirtualAllocEx+0x30, Writable: 0, Dirty: 0,
        VAD: {7c800000--7c933000, ImageMap: 1, File: "\WINDOWS\system32\kernel32.dll"}), SP: 12fe14
[02] <- 7c809af9 (API: VirtualAlloc+0x18, Writable: 0, Dirty: 0,
        VAD: {7c800000--7c933000, ImageMap: 1, File: "\WINDOWS\system32\kernel32.dll"}), BP: 12fe5c
[03] <- 8f03ed (API: -, Writable: 1, Dirty: 1, VAD: {8f0000--9f0000, ImageMap: 0}), BP: 12fe78
[04] <- 8f01f1 (API: -, Writable: 1, Dirty: 1, VAD: {8f0000--9f0000, ImageMap: 0}), BP: 12ff6c
```

図 2 Alkanet のスタックトレースログ
Fig. 2 Alkanet's stack trace log.

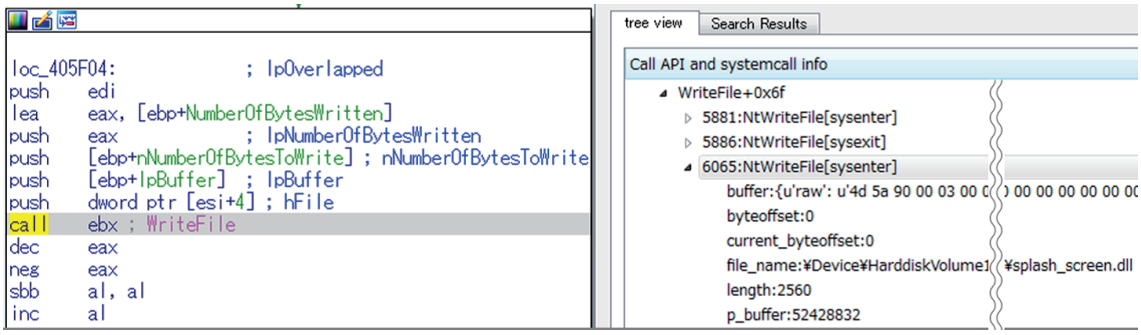

図 3 API 呼び出し情報と逆アセンブルコードの対応付け
Fig. 3 The association of the API Call information and disassemble code.

も取得できる．

4.3 静的解析部

4.3.1 API プリプロセッサによる API 呼び出し元領域の特定

マルウェアが API を呼び出す Call 命令とシステムコールログを結び付けるために，まずスタックトレース内に現れるリターンアドレスのうちどれがマルウェアの Call 命令によるものであるかを判定する必要がある．具体的には，スタックトレースのリターンアドレスのうち，次の 3 つの領域のいずれかにあてはまるリターンアドレスのうち最初に現れたものがマルウェアの Call 命令によるものである．

- **解析対象のマルウェアの実行ファイルがマップされている領域**
  マルウェアが感染コードを生成しない場合は，この領域がリターンアドレスとなる．
- **解析中に作成されたファイルがマップされている領域**
  感染コードがファイルへ展開された場合は，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リターンアドレスになる．
- **解析対象のプロセスが実行時に変更した領域かつ実行された領域**
  感染コードがメモリ上に展開された場合は，解析対象のプロセスが実行時に変更された領域かつ実行された領域がリターンアドレスになる．

図 2 に Alkanet のスタックトレースログを示す．スタックトレースをシステムコールスタブ（[00]）側からたどり，前述の 3 種類のいずれかの領域内にリターンアドレスを持つ最初のエントリ（[03]）が API のリターンアドレスのエントリとなる．また，その直前のエントリ（[02]）が解析対象のマルウェアが呼び出した API のエントリになる．このログでは，[00] の NtAllocateVirtualMemory を呼び出した API は [02] で，そのリターンアドレスが [03] の 0x8f03ed となる．

4.3.2 API 呼び出しとシステムコールログの結び付け

次に，解析対象のマルウェアが呼び出した API のエントリをスタックトレースから抽出する．さらに，同じリターンアドレスを持つ API 呼び出し，すなわち Call 命令ごとにそこから呼び出されたシステムコールログを集約する．なお，API 内で複数のシステムコールが呼び出される場合は 1 つの API 呼び出しに対して複数のシステムコールログを結び付ける．この結果は API 呼び出し情報として，次項の逆アセンブルコードの対応付けに利用する．また，この API 呼び出し情報を使って，マルウェアの挙動と呼び出し元を関連付けた動的解析ログを作成する．

4.3.3 API 呼び出し情報と逆アセンブルコードの対応付け

実装した IDA プラグインを用いて，逆アセンブルコードの Call 命令とその Call 命令と対応する API 呼び出し情報（API 名および，API によって呼び出されるシステムコールの情報）を対応付ける．API 呼び出し情報と逆アセンブルコードを対応付けた結果を**図 3** に示す．API 呼び出し情報に含まれるリターンアドレスの 1 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（図 3

の左側）．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たシステムコールのログをプラグイン側に表示する（図 3 の右側）．これにより，解析者は Call 命令で呼び出される API の挙動を容易に把握することができる．また，解析時に逆アセンブルコードを読むだけでなく，API 呼び出しの一覧からも解析する領域を選びたい場合がある．こういった場合のため，解析の補助機能として，プ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機能と選択した逆アセンブルコードのアドレスに対応する API 呼び出し情報を表示する機能もある．

5. 評価

5.1 評価目的と検体

提案手法の有効性を検証するため，マルウェアの実検体を用いた評価を行った．評価対象のマルウェアには，MWS Datasets 2016 [12] に含まれる BOS2014，BOS2015，BOS2016 に活動が記録されているマルウェアと独自に入手したマルウェアを用いた．

静的解析が必要なケースを想定し，RAT（Remote Access Tool）とランサムウェアに分類される検体を 25 検体選択した．以下，静的解析が必要と想定されるケースについて述べる．RAT は，外部の C&C サーバからコマンドを受信し動作する．このため C&C サーバからのコマンドが受信しない状態では，動的解析で十分に解析することができない．ランサムウェアの解析では，復号ツール [13], [14], [15] などを作成するケースが考えられる．このため，静的解析しランサムウェアの暗号化方法を把握する必要がある．

本章では，以下の場合に対し，提案手法の有効性について述べる．

- 静的解析対象範囲の絞り込み
- 動的解析時の API 呼び出し情報の参照
- 感染コードの自動取得

25 検体のすべてにおいて静的解析対象範囲の絞り込み，動的解析時の API 呼び出し情報の参照が可能であった．また，感染コードを生成する場合は，そのすべてにおいてコードの取得が可能であった．いくつかの典型的な例を抜粋して提案手法の有効性を述べる．

5.2 静的解析対象範囲の絞り込み

5.2.1 RAT の解析

RAT は攻撃者が遠隔操作によって情報窃取などを行うために用いるマルウェアである．インシデントレスポンスでは，RAT が外部と行った暗号化された通信のログが記録されていれば，そのログから窃取された情報を明らかにする場合が想定できる．またアンチウイルスベンダなど，マルウェアの持つ機能を把握するための解析では，RAT のコマンド一覧の解析が想定される．通信の暗号化処理の把握やコマンド一覧の解析のためには静的解析が必要となるが，RAT は外部との通信機能を持つコードが主要な動作を行うため，通信に関する API をもとに静的解析対象の範囲の絞り込みを行うことができる．

評価用の検体は，アンチウイルスソフトに PlugX として検出されたため，ファイル名を PlugX.exe とした．PlugX.exe の実行フローとコードマップ情報を**図 4** に，感染コード（svchost.exe の 0x90000-0xad000 の範囲）の挙動を抽出した動的解析ログを**図 5** に示す．図 5 から通信に関する挙動を行う API を呼び出しているアドレスが 0x96xxxx 付近であるということが分かる．よって図 4 から svchost.exe の 0x90000-0xad000 の領域が通信を行う感染コードだということが把握できる．これにより，通信に関連する挙動の解析のために，PlugX.exe 本体や starter.exe などの感染コー

```
...
------------------------------------
Network
------------------------------------
{
  "socket+0x50": "9648c4",
  "gethostbyname+0x92": "96417d",
  "gethostbyname+0x5d": "96417d",
  "gethostbyname+0xdf": "96417d"
}
```

図 5 PlugX.exe の感染コードの挙動を抽出した動的解析ログの抜粋
Fig. 5 Excerpt of dynamic analysis log extracting behavior of infected code of PlugX.exe.

```
PlugX.exe[0x400000-0x435000]
└── starter.exe
    ├── 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
        ├── dirty_memory [0x8d0000-0x8fb000]
        └── dirty_memory [0x90000-0xad000]
```

図 4 PlugX.exe の実行フローとコードマップ情報
Fig. 4 PlugX.exe execution flow and code map information.

```
1264:CERBER.exe[0x400000-0x45d000]
├── 1752:explorer.exe
│    └── dirty_memory [0x90000-0x9d000]
├── dirty_memory [0x400000-0x45d000]
└── dirty_memory [0xe80000-0xed0000]
```

図 6 CERBER.exe の実行フローとコードマップ情報
Fig. 6 CERBER.exe execution flow and code map information.

```
...
-----------------------------------------
Network
-----------------------------------------
{
  "HttpOpenRequestA+0x179": "943d2",
  "HttpSendRequestA+0x1d": "9443c",
  "InternetOpenA+0x61": "940ea",
  "HttpOpenRequestA+0x238": "943d2",
  "InternetCrackUrlA+0x1a": "93c9f",
  "InternetCloseHandle+0xa7": "940bc"
}
-----------------------------------------
Crypt
-----------------------------------------
{
  "CryptAcquireContextW+0x84": "95e04"
...
```

図 7 CERBER.exe の感染コードの挙動を抽出した動的解析ログの抜粋
Fig. 7 Excerpt of dynamic analysis log extracting behavior of infected code of CERBER.exe.

ドを解析する必要がなくなる．提案手法によって，静的解析対象範囲を絞り込むことが可能となった．

5.2.2 ランサムウェアの解析

評価用の検体は，アンチウイルスソフトに CERBER として検出されたため，ファイル名を CERBER.exe とした．CERBER.exe の実行フローとコードマップ情報を**図 6** に，感染コード（explorer.exe の 0x90000-0x9d000 の範囲）の挙動を抽出した動的解析ログを**図 7** に示す．ランサムウェアは，コンピュータ内のファイルを暗号化し，復号に必要な鍵と引き換えに身代金を要求するマルウェアである．ランサムウェア中の暗号化処理を絞り込んで静的解析を行うことができれば，解析の効率が上がる．ランサムウェアは，暗号化に用いる鍵を外部のサーバからダウンロードすることが多い．また，暗号化には CryptoAPI などの既存のライブラリがよく用いられる．これらの API を呼び出しているアドレスをもとに静的解析対象の範囲の絞り込みを行う．図 7 から通信および暗号化に関する挙動を行う API が 0x90000-0x9d000 の範囲で呼び出されていることが分かる．よって図 6 から explorer.exe 内の 0x90000-0x9d000 の領域がランサムウェアの主な挙動を行っていることが把握できる．提案手法によって，静的解析対象範囲を絞り込むことが可能となった．

5.3 動的解析時の API 呼び出し情報の参照

API の呼び出し方法には，Windows で標準として用意されているインポートテーブルを用いた呼び出し方法とそれ以外の方法でプログラムが動的にアドレスを解決することで呼び出す方法がある．本節ではその両者に対してそれぞれ評価を行った．

5.3.1 インポートテーブルを用いた API 呼び出し

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は，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 は逆アセンブル時に解決可能な API 名・引数名が逆アセンブルコードにコメント付けし，解析時に参照することができる．しかし，引数として渡される数値，文字列など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 によって実際に呼び出されたシステムコールログを参照可能になる．API の引数を直接取得することはできないが，API が呼び出したシステムコールログから，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．**図 8** 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 API 呼び出し情報として API 名とシステムコール名，システムコール引数情報を表示する（図 8 の右側）．このログから，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 呼び出し前後の逆アセンブルコードを解析することで API 呼び出しによるマルウェアの挙動を把握する必要がある．提案手法を適用することで，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，API 呼び出しによって行われる挙動を把握することができる．

5.3.2 DLL の動的ロードを用いた API 呼び出し

DLL の動的ロードを用いた 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．マルウェアの DLL の動的ロードを用いた API 呼び出しは，API 呼び出しの難読化のためや，コードを動的に展開した感染コードで行われる．具体的には，まず LoadLibrary 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 名をもとに 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．

DLL の動的ロードを用いた API 呼び出しがあるマルウェアを解析する場合，逆アセンブル後に API のアドレス

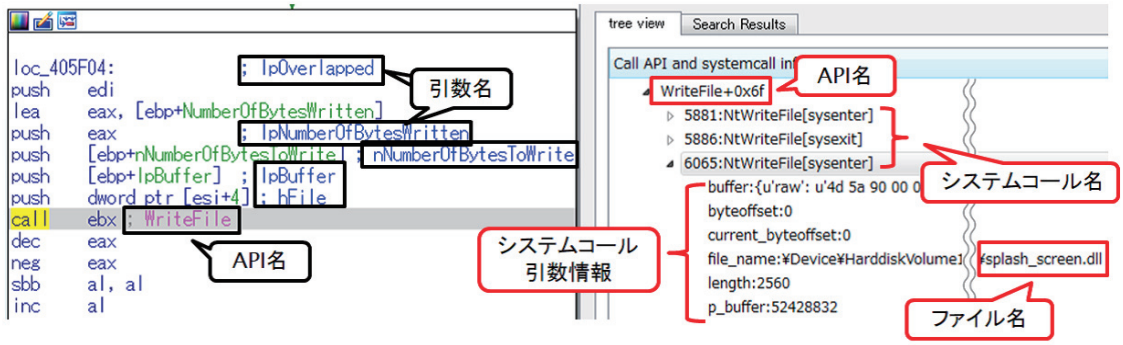

図 8 提案手法適用後のインポートテーブルを用いた API の呼び出し
Fig. 8 API call using the import table after applying the proposed method.

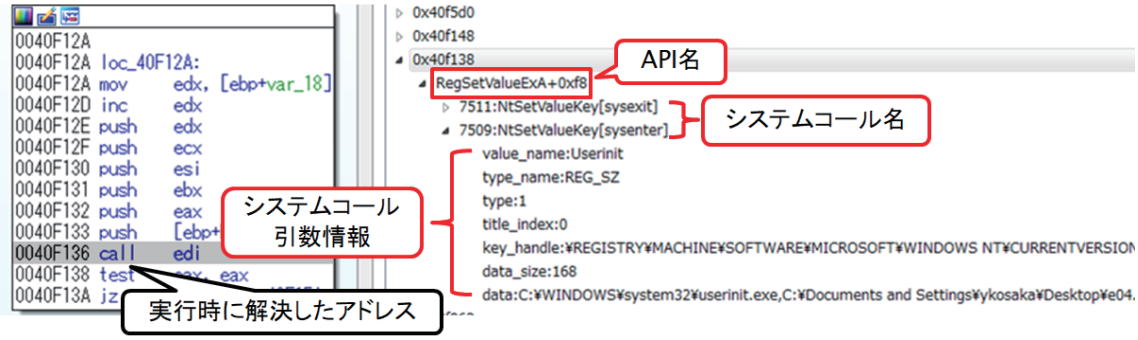

図 9 提案手法適用後の DLL の動的ロードを用いた API 呼び出し
Fig. 9 API call using dynamic load of DLL after applying proposed method.

解決を行う処理を解析し，呼び出された API を特定する必要がある．**図 9** は，マルウェアの DLL の動的ロードを用いた API 呼び出しに対して提案手法を適用している．逆アセンブルコード（図 9 の左側）から，実行時に解決したアドレスをレジスタに格納し，そのアドレスを呼び出すことで API を呼び出していることが確認できる．プラグイン側の表示（図 9 の右側）を見ると RegSetValueExA が呼び出され，RegSetValueExA が呼び出した NtSetValueKey がレジストリの値を変更し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，呼び出された API に関する情報は参照できない．提案手法を適用することで呼び出された API 名および，API によって呼び出されたシステムコールログが参照できた．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り，静的解析を補助することができる．

5.4 感染コードの自動取得

動的解析時に，静的解析を行うために必要な感染コードを取得することで，感染コードの取得作業を省略することが可能となる．以下の挙動を行うマルウェアにおいて，提案手法の有用性を評価する．

- コード展開型マルウェア
- 自己書き換え型マルウェア
- ファイルを作成するマルウェア

```
1084:PoisonIvy.exe[0x400000-0x40c000]
└── dirty_memory [0x380000-0x390000]
```

図 10 PoisonIvy.exe の実行フローとコードマップ情報
Fig. 10 PoisonIvy.exe execution flow and code map information.

5.4.1 コード展開型マルウェア

コード展開型マルウェアの実行ファイルは，展開ルーチンと暗号化されたコードから構成され，実行時に動的にメモリ領域を確保してコードを展開する．評価用の検体は，アンチウイルスソフトに PoisonIvy として検出されたため，ファイル名を PoisonIvy.exe とした．実行フローとコードマップ情報を**図 10** に示す．これからマルウェアの実行ファイルのバイナリがマップされているメモリ領域以外に，動的に確保されたメモリ領域（[0x380000-0x390000]）が感染コードとして現れていることが確認できる．動的解析時に取得したこの感染コードを逆アセンブルすると，展開後のコードが確認できた．提案手法は，コード展開型マルウェアであっても，展開後のコードを取得することができるといえる．

5.4.2 自己書き換え型マルウェア

自己書き換え型のマルウェアは，コード展開型マルウェアの亜種である．コード展開型マルウェアと違いは，マルウェアの実行ファイルのバイナリがマップされているメモ

```
1328:Emdivi.exe[0x400000-0x44f000]
└── dirty_memory [0x400000-0x44f000]
```

図 11 Emdivi.exe の実行フローとコードマップ情報
Fig. 11 Emdivi.exe execution flow and code map information.

```
2796:NfLog.exe[0x400000-0x43e000]
├── 2804:leassnp.exe
│    └── File(\DOCUME~1\ADMINI~1\LOCALS~1\Temp\leassnp.exe)[0x400000-0x439000]
└── 2816:wordpad.exe
```

図 12 NfLog.exe の実行フローとコードマップ情報
Fig. 12 NfLog.exe execution flow and code map information.

リ領域を書き換える点である．評価用の検体は，アンチウイルスソフトに PoisonIvy として検出されたため，ファイル名を PoisonIvy.exe とした．実行フローとコードマップ情報を**図 11** に示す．これからマルウェアの実行ファイルのバイナリがマップされているメモリ領域と同じメモリ領域（[0x400000-0x44f000]）が感染コードとして現れていることが確認できる．実行前の Emdivi.exe を逆アセンブルしたが，ほとんどの領域がデータとして認識された．動的解析時に取得した感染コードを逆アセンブルすると，データとして認識されていた領域にコードが確認できた．このことから，提案手法は自己書き換え型マルウェアの場合であっても感染コードを取得できるといえる．

5.4.3 ファイルを作成するマルウェア

ドロッパやダウンローダなどのマルウェアはファイルを作成し，子プロセスとして実行する．評価用の検体は，アンチウイルスソフトに NfLog として検出されたため，ファイル名を NfLog.exe とした．実行フローとコードマップ情報を**図 12** に示す．NfLog.exe は leassnp.exe を作成し，子プロセスとして実行した後に自身のプロセスを終了する．またシステム上に存在する wordpad.exe を実行する．コードダンプログから leassnp.exe を抽出し，別途手動で取得した leassnp.exe のハッシュ値を比較したところ同一の値であった．なお，wordpad.exe は元々システム上に存在するファイルのため提案手法では取得しない．以上より提案手法は，動的解析時にマルウェアが作成するファイルを取得できるといえる．

6. 関連研究

提案手法で実現した感染コードの取得と同様に，マルウェアのコードを抽出する研究としてアンパックの研究がある．アンパックの研究として，PolyUnpack [16] や OmniUnpack [17]，Renovo [18]，RAMBO [19]，PinDemonium [20] がある．PolyUnpack は，ゲスト OS 内からメモリを監視し静的解析時のコードと比較することで，静的解析時に存在しないコードを記録する．OmniUnpack はカーネルドライバとして実装されており，プログラムの実行を監視し，プログラムが対象となるシステムコールを呼び出したときに，マルウェア検出器を用いてメモリ上をスキャンする．マルウェアが検出されたページのみを記録することで低オーバヘッドを実現している．Renovo はテイント解析機能を持ったエミュレータ TEMU [21] を拡張して実装されており，実行時にプログラムのメモリ書き込みを監視し，実行中のコードが新しく生成されたかどうかを判断することで，実行可能なコードを抽出する．RAMBO は TEMU をベースに開発されており，動的解析で対象のバイナリをトレースすることでアンパックを行う．トレース時にシンボリック実行とシステムレベルのスナップショット，テイント解析を用いることでマルチパス探索実現している．PinDemonium は，Intel Pin と Scylla を用いて実装されている．メモリの書き込み可能と実行可能のフラグをもとに Scylla でメモリをダンプする．ヒープ領域のダンプと IAT の難読化，プロセスインジェクションにも対応している．提案手法は，システムコールの挙動とシステムコール呼び出し元のメモリ領域に着目してコードを取得している．そのため，システムコールが呼び出された実行可能なメモリ領域をダンプ可能である．

マルウェアの動的解析時にメモリダンプを取得するシステムも存在する．Xen [22] ベースのマルウェア解析フレームワーク Ether [23] を用いた実装に EtherUnpack と EtherTrace がある．EtherUnpack は，メモリ書き込みおよび実行された命令を監視することによって動的に生成されたコードを抽出する．EtherTrace は，システムコールトレースを行う．また，Cuckoo [24] はゲスト OS 内のプログラムが API トレースと同時に解析対象のプロセスのメモリダンプを取得する．提案手法は，動的解析時にシステムコールログとコードダンプログを取得するだけでなく，システムコールログをもとにした API 呼び出し情報と感染コードを対応付けるなど，それらを活用することで静的解析の補助を行っている．

動的解析のログを静的解析に活用するツールとして，funcap [25] と IDA splode [26] 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 API 呼

び出し情報を逆アセンブル結果にコメント付けするものである．IDA splode は，Intel PIN と IDA を利用し，メモリアクセスとプログラム実行時のメタデータ，API 名などを記録するものである．どちらもプログラム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実行時の情報を記録している．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の解析や感染コードの取得ができない．提案手法は，Alkanet をベースにしているためこれらの問題に影響を受けることなく静的解析の補助を実現している．マルウェア解析コストの軽減を目指したシステムに egg [27] 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．解析ログとして，API 引数の取得・コールグラフ・分岐情報を取得できる．また，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もある．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が，逆アセンブルコードに対応付けている情報は分岐情報のみである．提案手法は，静的解析補助のために動的解析時のログを活用し，API 呼び出し情報と逆アセンブルコードの対応付けを実現している．また T.A.C.O [28] は，Cuckoo のログを IDA に読み込ませ，表示することで静的解析補助を行っている．T.A.C.O は Cuckoo の実行時の API 呼び出し情報を活用した静的解析を実現しているが，Cuckoo 自体が静的解析の利用を前提としていない．このため，Cuckoo のログに含まれるマルウェアに関連するプロセスやスレッドのツリーを提示できるが，それらのコードマップ情報はログに含まれない．よって，どの領域を逆アセンブルすべきか判断することができず，静的解析範囲の絞り込みを行うことができない．提案手法では，静的解析の利用を前提とした動的解析システムの必要性とその要件を提案しており，Alkanet を拡張した動的解析システムとそのログを静的解析で活用するためのツール群を実装した．

動的解析と静的解析を組み合わせた解析手法の研究として，Yee らはマルウェアのリバースエンジニアリングを簡易化するために，静的解析と動的解析を用いて可視化グラフを作成する手法 [29] を提案している．コントロールフローの可視化によって，実行可能ファイルのコントロールフローを短時間で把握することができる．しかし，この手法では API 呼び出しに関する情報を提供しておらず，マルウェアの挙動に着目した提案手法と異なる．ほかにも，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 [30] 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て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．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点は提案手法と同様であるが，この手法では制御フローに注力しており API 呼び出しに関する補助は行わない．

7. おわりに

本論文では，静的解析時の解析作業を軽減するための手法として，API 呼び出し情報と API の呼び出し元を記録した動的解析ログによる静的解析の範囲の絞り込み，動的解析ログと逆アセンブルコードの対応付けによる静的解析時の API 呼び出し情報の提示，動的解析時の感染コードの取得の 3 つについて提案した．また，システムコールトレーサ Alkanet の拡張と逆アセンブラ IDA のプラグインの実装によってそれらが実現可能であることを示した．さらに，実際のマルウェア検体を用いてその有効性を示した．提案手法によって，2 章で述べた静的解析における課題を解決し，静的解析の負担を軽減することができた．

今後の課題として，静的解析のために動的解析時に取得する情報をさらに充実させることがあげられる．具体的には，実行パスの取得とセマンティックギャップへの対応がある．現状では，Call 命令しか記録しておらず，マルウェアの分岐情報を記録できていない．Alkanet にはブランチトレース機能 [31] があるため，それを用いて実行パスを取得することで，さらなる補助が可能となる．また，システムコールトレースを用いて実装しているため，動的解析ログと逆アセンブルコードの API 呼び出しにセマンティックギャップがあった．マルウェアが直接呼び出した API の情報を取得可能な手法を用いることで，これを軽減できる．

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（オンライン），入手先 ⟨https://app.trendmicro.co.jp/doc_dl/select.asp?type=1&cid=161⟩ (2015).

[2] Kaspersky：BLUE TERMITE. ブルーターマイト—日本を標的にする APT 攻撃（オンライン），入手先 ⟨http://media.kaspersky.com/jp/pdf/pr/Kaspersky_BlueTermiteDaily-PR-1016.pdf⟩ (2015).

[3] トレンドマイクロ株式会社：2016 年個人と法人の三大脅威：日本におけるサイバー脅迫元年，入手先 ⟨http://blog.trendmicro.co.jp/archives/14229⟩（参照 2017-01-19）.

[4] 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).

[5] Hex-Rays: IDA: About – Hex-Rays (online), available from ⟨http://www.hex-rays.com/products/ida/⟩.

[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] Moser, A., Kruegel, C. and Kirda, E.: Limits of Static Analysis for Malware Detection, *Computer Security Applications Conference, 2007* (*Proc. ACSAC 2007*) *23rd Annual*, pp.421–430 (2007).
[8] You, I. and Yim, K.: Malware Obfuscation Techniques: A Brief Survey, *Broadband, Wireless Computing, Communication and Applications* (*Proc. BWCCA*), *2010 International Conference*, pp.297–300 (2010).
[9] Yason, M.V.: The Art of Unpacking, *Black Hat USA 2007* (2007).
[10] 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).
[11] Dolan-Gavitt, B.: The VAD tree: A process-eye view of physical memory, *Digital Investigation*, Vol.4, pp.62–64 (2007).
[12] 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).
[13] カスペルスキー：暗号化型マルウェア「Polyglot（別名 MarsJoke）」と復号ツール（オンライン），入手先 ⟨https://blog.kaspersky.co.jp/polyglot-decryptor/12751/⟩（参照 2017-01-22）.
[14] マカフィー株式会社：McAfee Labs が LeChiffre ランサムウェアの暗号化解除に成功（オンライン），入手先 ⟨http://blogs.mcafee.jp/mcafeeblog/2016/06/mcafee-labslech-64bc.html⟩（参照 2017-01-22）.
[15] トレンドマイクロ株式会社：ランサムウェアにより暗号化されたファイルの復号ツールを公開（オンライン），入手先 ⟨http://blog.trendmicro.co.jp/archives/13408⟩（参照 2017-01-22）.
[16] Royal, P., Halpin, M., Dagon, D., Edmonds, R. and Lee, W.: PolyUnpack: Automating the Hidden-Code Extraction of Unpack-Executing Malware, *Proc. Annual Computer Security Applications Conference*, pp.289–300 (2006).
[17] Martignoni, L., Christodorescu, M. and Jha, S.: Omniunpack: Fast, generic, and safe unpacking of malware, *Proc. Annual Computer Security Applications Conference* (2007).
[18] Kang, M.G., Poosankam, P. and Yin, H.: Renovo: A Hidden Code Extractor for Packed Executables, *Proc. 2007 ACM Workshop on Recurring Malcode*, *WORM '07*, pp.46–53 (2007).
[19] Ugarte-Pedrero, X., Balzarotti, D., Santos, I. and Bringas, P.G.: RAMBO: Run-time packer analysis with multiple branch observation, *DIMVA 2016*, *13th Conference on Detection of Intrusions and Malware & Vulnerability Assessment* (2016)
[20] Mariani, S., Fontana, L., Gritti, F. and D'Alessio, S.: PinDemonium: A DBI-based generic unpacker for Windows executables, *Black Hat USA 2016* (2016).
[21] Song, D., Brumley, D., Yin, H., Caballero, J., Jager, I., Kang, M.G., Liang, Z., Newsome, J., Poosankam, P. and Saxena, P.: BitBlaze: A New Approach to Computer Security via Binary Analysis, *Proc. 4th International Conference on Information Systems Security, Keynote Invited Paper* (2008).
[22] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A.: Xen and the Art of Virtualization, *Proc. 19th ACM Symposium on Operating Systems Principles*, pp.164–177, ACM (2003).
[23] Dinaburg, A., Royal, P., Sharif, M. and Lee, W.: Ether: Malware Analysis via Hardware Virtualization Extensions, *Proc. 15th ACM Conference on Computer and Communications Security*, *CCS '08*, ACM (2008).
[24] Guarnieri, C., Tanasi, A., J.B.M.S.: Cuckoo Sandbox: Automated Malware Analysis, available from ⟨https://cuckoosandbox.org⟩ (accessed 2017-03-14).
[25] DERESZOWSKI, A.: funcap (online), available from ⟨https://github.com/deresz/funcap⟩ (accessed 2017-01-22).
[26] ENDGAME: IDA-splode (online), available from ⟨https://github.com/zachriggle/ida-splode⟩ (accessed 2017-01-22).
[27] Tanda, S.: "egg" – A Stealth fine grained code analyzer *Recon 2011* (2011).
[28] Jones, J.: IDATACO IDA Pro Plugin (online), available from ⟨https://github.com/arbor-jjones/idataco⟩ (accessed 2017-08-12).
[29] Yee, C.L., Chuan, L.L., Ismail, M. and Zainal, N.: A Static and Dynamic Visual Debugger for Malware Analysis, *18th Asia-Pacific Conference on Communications* (*Proc. APCC*), pp.765–769 (2012).
[30] 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).
[31] 大月勇人，瀧本栄二，齋藤彰一，毛利公一：ブランチトレース機能を用いたシステムコール呼出し元識別手法，情報処理学会論文誌，Vol.57, No.2, pp.756–768 (2016).

推薦文

動的解析によって得られる API コール情報に注目し，静的解析の補助を行うことが本研究の目的である．その目的を達成するために必要となる実装，評価をしっかりと行っており，得られた結果は静的解析の時間短縮に大きく貢献するという実用的な成果を得ている．動的解析ログを静的解析の結果にマッピングする手法には既存研究が存在するが，本研究では BitVisor ベースのマルウェア動的解析システム Alkanet と連携することで，これらの既存研究と比較しても解析に有用なより多くの情報を取得できている点で利点がある．よって推薦論文として推薦する．

（マルウェア対策研究人材育成ワークショップ 2016
プログラム委員長 森 達哉）

中島 将太 （正会員）

2015 年立命館大学情報理工学部情報システム学科卒業，2017 年同大学大学院情報理工学研究科博士前期課程情報理工学専攻修了．

大月 勇人 （正会員）

2011 年立命館大学情報理工学部情報システム学科卒業，2013 年同大学大学院理工学研究科博士前期課程情報理工学専攻修了．2016 年同大学院情報理工学研究科博士後期課程情報理工学専攻修了，同年日本電信電話株式会社入社，現在，NTT セキュアプラットフォーム研究所勤務．博士（工学）．マルウェア解析技術に関する研究に従事．

明田 修平 （学生会員）

2013 年立命館大学情報理工学部情報システム学科卒業，2015 年同大学大学院情報理工学研究科博士前期課程情報理工学専攻修了．同年同大学院情報理工学研究科博士後期課程情報理工学専攻に入学，現在に至る．コンピュータネットワークに関する研究に従事．

瀧本 栄二 （正会員）

1999 年立命館大学理工学部情報学科卒業，2001 年同大学大学院理工学研究科博士前期課程修了，2005 年同研究科博士後期課程単位取得退学，同年（株）ATR 適応コミュニケーション研究所専任研究員，2010 年立命館大学情報理工学部情報システム学科助手，2017 年立命館大学情報理工学部情報理工学科助教，現在に至る．主にシステムソフトウェア，無線通信に関する研究に従事．博士（工学）．電子情報通信学会会員．

齋藤 彰一 （正会員）

1993 年立命館大学理工学部情報工学科卒業．1995 年同大学大学院博士前期課程修了．1998 年同大学院博士後期課程中退．同年和歌山大学システム工学部情報通信システム学科助手．2003 年同講師，2005 年同助教授．2006 年名古屋工業大学大学院助教授，2007 年同准教授，2016 年同教授，現在に至る．オペレーティングシステム，インターネット，セキュリティ等の研究に従事．博士（工学），ACM，IEEE-CS 各会員．

毛利 公一 （正会員）

1994 年立命館大学理工学部情報工学科卒業，1996 年同大学大学院理工学研究科修士課程情報システム学専攻修了，1999 年同研究科博士課程後期課程総合理工学専攻修了．同年東京農工大学工学部情報コミュニケーション工学科助手，2002 年立命館大学理工学部情報学科講師，2004 年同大学情報理工学部情報システム学科講師，2008 年同准教授，2014 年同教授となり，現在に至る．博士（工学）．オペレーティングシステム，仮想化技術，コンピュータセキュリティ等の研究に従事．電子情報通信学会，ACM，IEEE-CS，USENIX 各会員．