暗号関数の位置特定手法の評価

古川 凌也[1]　伊沢 亮一[2]　森井 昌克[1]　井上 大介[2]　中尾 康二[2]

概要：本稿では，暗号関数の位置特定の支援としてその候補を絞り込む手法を提案する．提案手法はプログラムに入力される暗号化データがはじめに復号処理を通過することに着目し，暗号化データが入力された付近で入出力をやりとりした関数を復号関数の候補として抽出する．著者らは 2015 年に，一般の暗号ライブラリを用いて，提案手法を評価する実験を行ったが，新たに実際のマルウェア検体 4 種を実験対象に加え，複数の異なるコミュニティ検出手法を用いて提案手法を適用した結果，実験対象が実行したすべての関数のうち，最高で 0.1%未満，平均しておよそ 2.0%程度にまで復号関数の候補数を絞り込むことができた．

キーワード：
マルウェア解析, バイナリ解析, ネットワークセキュリティ, 暗号技術

Locating Cryptographic Functions: an Evaluation

Ryoya Furukawa[1]　Ryoichi Isawa[2]　Masakatu Morii[1]　Daisuke Inoue[2]　Koji Nakao[2]

Abstract: In this paper, we propose a method to narrow down the candidate of cryptographic functions for the support of malware analysis. We focuses on the fact that the encrypted data to be input to the program passes through the decryption process at first. And we extract functions that exchange input-output at around the function read encrypted data as the candidate for the decryption function. With experiments using eight cryptographic programs and four real world malwares, we we extracted 2.0% of about 1000–5000 functions as candidates on average, and extracted less than 0.1% of them at best.

Keywords: Malware Analysis, Binary Analysis, Network Security, Cryptography

1. はじめに

近年，企業や政府のような特定の組織を狙った標的型攻撃が世界的な問題となっている．標的型攻撃は組織内の機密情報の奪取や計算機環境の破壊などを目的としており，目的の達成まで長期間にわたり段階的な攻撃が行われる．標的型攻撃において，標的内への侵入や侵入後の諜報活動を行うためのツールとしてマルウェアが用いられる．攻撃者はマルウェアを標的内の計算機に感染させることで，通信を介した不正な遠隔操作を可能にし，新たな攻撃プログラムのダウンロードや機密情報の送信などを行う．標的型攻撃の発覚後，被害状況を把握し事態を迅速に収束させるためには，攻撃者とマルウェアの間で送受信されたデータを知ることが重要である．ところが多くの場合，マルウェアは暗号技術を用いて通信を秘匿するため，その内容を調査することが困難になっている．このような時，感染 PC から捕獲したマルウェアのバイナリを解析し，それによって得られたマルウェアの暗号ロジックや鍵の情報を利用することで，暗号化された通信を復号する方法が考えられる．

しかしながらマルウェアのバイナリの解析には膨大な時間と労力が必要になる．たとえば，マルウェアが動作中に実行する関数の中に暗号ロジックが含まれているものとして，それらの関数すべてを解析者が手動によって調べるとする．解析者はそれぞれの関数がどういったデータを入出力とし，どのような処理を行っているのかをもとに，暗号ロジックを含む関数であるかどうかを判定することになる．

[1] 神戸大学大学院工学研究科
Graduate School of Engineering, Kobe University
[2] 国立研究開発法人情報通信研究機構
National Institute of Information and Communications Technology

このときマルウェアが実行する数百から数千の関数のすべてを暗号ロジックを含む暗号関数の候補とすると，仮に解析者が一つの関数を数分で調べることができたとしても，それらの候補のすべてを調べることは現実的ではない．効率的な解析のためには，暗号関数の候補を相当数絞り込まなければならない．

マルウェアの実行命令系列を取得し，そこから暗号アルゴリズムを含む関数の位置や暗号アルゴリズムの種類の特定，鍵や平文などのパラメータを取得するための研究がなされている [1][3][4][2][5]．ReFormat[2] や CipherXRay[5] は，暗号化や復号の対象となるデータの伝搬をメモリ上で追跡し，そのデータに対して実行された機械語命令の特徴やデータフローの特徴から暗号アルゴリズムらしい一連の処理や関数を推定する．これらの手法は高い粒度でデータの伝搬を追跡する必要があり，そのための手法としてテイント解析を用いている．一般的に，テイント解析には under-tainting と呼ばれるデータ伝搬の検知漏れに関する課題と，over-tainting と呼ばれるデータ伝搬の誤検知に関する課題が知られており，これらはトレードオフの関係になっている．暗号関数を候補に含むためには，over-tainting による誤検出は許容したとしても，under-tainting による検知漏れは防止しなければならない．

本稿では，マルウェア解析における暗号関数の位置特定の支援として暗号関数の候補を絞り込む手法を提案する．提案手法は暗号化通信などにおいてプログラムに入力された暗号化データは，はじめに復号処理を通過するという点に着目し，暗号化パケットを入力として読み込んだ関数の付近で入出力をやりとりする関数を復号関数の候補として抽出する．そのための方法として，既存手法とは異なる，各関数の入出力データによる依存関係を追跡するテイント解析を行う．あえて粒度の荒い，over-tainting の起きやすいテイント解析を用いることによって, under-tainting による検知漏れを防止することができる．追跡した依存関係をもとに，各関数をノード，入出力をエッジとしたグラフを生成し，グラフに含まれる関数を復号関数の候補とする．このとき，over-tainting によって増加した候補をいかに絞り込むかが重要になるが，提案手法では生成したグラフにコミュニティ検出の手法を適用し，暗号化データが入力された関数と互いに入出力の依存関係の強い一連の関数のみを復号関数の候補として抽出することで候補数を削減する．さらに，コミュニティ検出によって抽出した各候補の入出力のサイズを比較し，同サイズの入力と出力のペアをもつ候補を，より復号関数である可能性の高い，解析の優先順位の高い候補として抽出する．これは，多くの場合暗号化通信などでは共通鍵暗号方式が用いられることから，復号関数の入出力となる平文と暗号文のサイズが一致するという特徴に基づく．

評価実験では既存の暗号ライブラリを用いて作成した 8 個のテスト用プログラムと，4 種のマルウェア検体に対して提案手法を適用し，実行されたすべての関数のうち，最高でおよそ 0.1%未満，平均して 2.0%程度にまで復号関数の候補数を削減することができた．提案手法は解析者が調べるべき暗号関数の候補を相当数少なくすることで，マルウェアが用いる暗号関数の位置特定の支援となる．

2. 基礎知識

2.1 マルウェアの動作

例としてボットが暗号化通信を行う場合を考える．PC に感染したボットが C&C サーバに対してコマンドを要求するパケットを送信したとする．その後，C&C サーバから暗号化されたメッセージを受信したときのボットの動作は以下のプロセスに従うと考えることができる．

(1) 受信した暗号化パケットを入力として復号処理を行い，平文メッセージを出力する．

(2) 復号した平文メッセージを入力とし，実行すべきコマンドを抽出，出力する．

(3) 抽出したコマンドを入力とし，そのコマンドにしたがった処理を行う．

このように，処理 (1) による出力は処理 (2) の入力となり，処理 (2) による出力は処理 (3) の入力となることから，処理 (1) の復号処理が行われなければ，その後のプロセスに遷移することはできない．したがって，暗号化パケットを入力とする関数の付近で入出力をやり取りする関数のうちいずれかに，暗号ロジックを含む復号関数が存在すると考えることができる．

2.2 テイント解析

テイント解析はデータフロー解析技術の手法の一つであり，プログラムのデバッグやマルウェア解析，攻撃の検知など，さまざまな目的に応用されている [6][7]. テイント解析では，追跡対象のデータにテイントタグとよばれる目印を付与し，テイントタグが付与されたデータから新たに生成されたデータへと逐次テイントタグを付与することで，追跡対象のデータの伝搬を知ることができる．このときデータに対してテイントタグを付与する条件の取り決めを伝搬ルールと呼ぶ．例えば一般的なテイント解析では，CPU が演算命令やデータ遷移命令を実行する際にテイントタグが付与されたデータが参照されていれば，その演算結果に対してテイントタグを付与するというような伝搬ルールが与えられる．しかし，このような直接的な代入関係に注目した伝搬ルールのみでは，本来テイントタグが付与されることが期待されるデータに対してテイントタグが付与されないような，いわゆる under-tainting と呼ばれる伝搬漏れが発生することがある [8]．この under-tainting は以下のようなコードによって容易に引き起こすことが可能であると指摘されている [9].

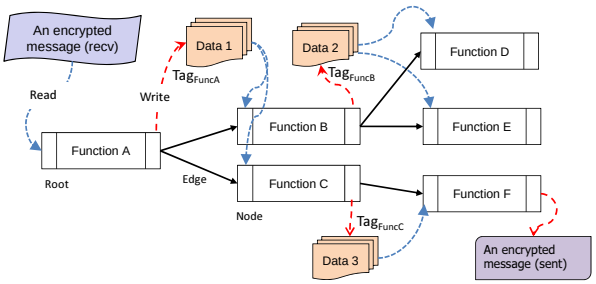

図 1　各関数の入出力データによる依存関係

```
for(i = 0; i < 256 ;i++)
    array[i]=i;
...
out = array[in];
```

上記のコードにおいて変数 in にテイントタグが付与されていたとき，変数 out の値は変数 in の値に依存していることから，変数 out に対してもテイントタグが付与されることが期待される．しかし実際には，変数 in と変数 out の間には直接的な代入関係がないため，under-tainting が発生する．このような伝搬漏れを防止するためには，個別に伝搬ルールを追加するなどの対応が必要になる．しかしそれによって，テイントタグが付与されることが望ましくないデータに対して過剰にテイントタグが付与されるような，over-tainting と呼ばれる誤伝搬が発生することも考えられる．under-tainting と over-tainting はトレードオフの関係になっており，目的に応じて適切な伝搬ルールを用いることが重要になる．

3. 提案手法

3.1 概要

提案手法は暗号関数の検知漏れを防ぎつつ，可能な限り候補を削減することで，暗号関数の位置特定を支援する．そのための方法として，プログラムに入力された暗号化データは，かならずはじめに復号処理を通過するという点に着目し，暗号化データを入力として読み込んだ関数と入出力をやりとりする付近の関数を復号関数の候補として抽出する．

提案手法では，各関数の入出力データによる依存関係を追跡するテイント解析を行う．ここでは，ある関数が出力したデータを別のある関数が 1 バイトでも入力として読み込んだとき，これらの関数は入出力データによる依存関係を持つものとする．実行された関数をノード，関数間でやり取りされた入出力をエッジとして図 1 のように入出力データによる依存関係を表すグラフを生成する．図 1 ではまず，プログラムが受信した暗号化データが Function A の入力としてメモリから読み込まれている．この Function A がグラフ生成の起点となる．そして Function A は，Data1

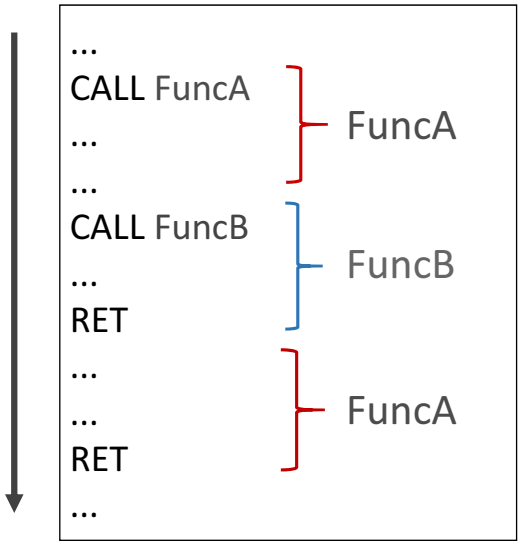

図 2　関数の定義

を出力としてメモリに書き出す．このとき Data1 が書き込まれたメモリは Function A によってタグ付けされたものとする．その後，Function A によってタグ付けされたメモリを Function B，Function C がそれぞれ読みだしたとき，Function B と Function C の入力は Function A の出力に依存するものとし，Function A から Function B，Function C へとエッジを結ぶ．このテイント解析は over-tainting が発生しやすく，多くの関数がエッジで結ばれることになるが，under-tainting による検知漏れを防止することができる．

提案手法は生成したグラフにコミュニティ検出の手法を用い，復号関数である可能性が高い候補のみを抽出することで，over-tainting による誤検出を抑え込む．生成したグラフに対してコミュニティ検出を行うことで，入出力の依存関係が比較的強い関数からなるグループにグラフを分割することができる．そして，暗号化データを入力とする関数と同一のコミュニティに含まれる関数を，暗号化データを入力として読み込んだ関数と入出力をやりとりする付近の関数として復号関数の候補として抽出する．

さらに提案手法では，候補として抽出した各関数の入力と出力のサイズを比較し，同サイズの入出力を持つ関数をより復号関数である可能性の高い関数として選び出すことができる．これは，多くの場合暗号化通信などには共通鍵暗号方式が用いられることから，暗号関数の入力や出力となる平文のサイズと暗号文のサイズが一致するという点に基づく．解析者はこのとき選び出された候補から優先して暗号関数であるかどうかを調べることにより，効率的に解析を行うことができる．

3.2 グラフの生成

提案手法におけるグラフの生成手順を説明する．提案手

表 1　Algorithm1 の変数

Var	Description
G	Algorithm1 によって生成されるグラフ
N_n	n 番目に呼び出された関数インスタンスを表すノード
$E_{m,n}$	N_m から N_n へと接続されるエッジ
I_n	実行トレース内で n 番目に実行された機械語命令
id	I_n が含まれる関数インスタンスの識別番号
M_{addr}	メモリアドレス $addr$ に書き込んだ関数インスタンスの識別番号
T	TAINT を実行した関数の識別番号を記録したリスト

Algorithm 1 グラフの生成

```
1: id ← 0
2: for i ←START to END do
3:    if I_i = CALL then
4:       id ← id + 1
5:       add N_id to G
6:    else if I_i = RET then
7:       id ← id − 1
8:    else if I_i = WRITE addr val then
9:       M_addr ← id
10:   else if I_i = READ addr val then
11:      add E_{m,n} to G
12:   else if I_i = TAINT val addr then
13:      add id to T
14:      M_addr ← id
15:   else
16:      pass
17:   end if
18: end for
```

法では，まず対象のマルウェアを解析環境内で動作させ，マルウェアが動作中に実行した一連の機械語命令を記録し，実行トレースとして取得する．この実行トレースには，マルウェアが実行した機械語命令に加え，実際にアクセスしたメモリアドレスと読み書きした値が含まれている必要がある．

提案手法における関数の定義を図 2 に示す．実行トレースのうち，関数の呼び出しを行う CALL 命令が実行されてから，次に関数からの復帰を行う RET 命令が実行されるまでの領域を一つの関数であるとみなす．ただし，図 2 のように，ある関数 A に含まれる領域の中で別の関数 B が呼ばれた場合には，その領域はもとの関数 A から除外され，関数 B のみに属する領域となる．

暗号化データを入力とする関数を特定するために，特定の目印を実行トレース中に挿入することで，外部からプログラムへの暗号化データの入力となりえるパケット及びファイルのメモリへの読み込みを記録する．例えば，外部からのデータがメモリアドレス $addr$ に書き込まれた時には，TAINT $addr$ val を実行トレースに挿入する．外部からのデータの入力は機械語命令 IN や recv() や read() といった API コールをもとに監視することができる．

次に取得した実行トレースをもとにしてグラフを生成する．提案手法では，実行トレースから取得できる情報のうち，関数の呼び出しと復帰，メモリへの読み書きに着目する．グラフの生成を行うアルゴリズムを Algorithm**??**に示す．提案手法ではグラフの生成時，関数の呼び出しごとに実行される関数を区別し，関数インスタンスと呼ぶ．生成されるグラフは関数インスタンスをノード，入出力をエッジとした有向グラフである．表 1 に Algorithm1 で用いる変数をまとめる．まず G は生成されるグラフを表す．そして N_n は n 番目に呼び出された関数インスタンスを表すノードであり，$E_{m,n}$ は N_m から N_n へと接続されるエッジを表す．I_n は実行トレース内で n 番目に実行された機械語命令を表す．また id の値は，I_i が何番目に呼び出された関数に含まれているものであるかを表す．そして M_{addr} には I_i が実行された時点でメモリアドレス $addr$ に格納されている値が何番目に呼び出された関数インスタンスによって書き込まれたものであるかが保存される．T は復号関数候補抽出の起点となる，暗号化データの読み込みが記録された関数が保存される．Algorithm1 では実行トレースのうち，START と END で指定した範囲に含まれる I_n がどのような機械語命令であるかに従って，以下の 5 種類の処理を行う．

- I_n が CALL 命令であったとき，id の値を 1 増加させ，N_{id} を G に追加する．
- I_n が RET 命令であったとき，id の値を 1 減少させる．
- I_n がメモリアドレス $addr$ への書き込みを行う機械語命令であったとき，M_{addr} に id の値を格納する．
- I_n がメモリアドレス $addr$ からの読み込みを行う機械語命令であったとき，$E_{M_{addr},id}$ を G に追加する．
- I_n がメモリアドレス $addr$ への TAINT であったとき，id を T に追加し，M_{addr} に id の値を格納する．

3.3　入出力バッファのサイズ比較

ボットの暗号化通信やランサムウェアによるファイルの暗号化の際には，多くの場合 AES や RC4 といった共通鍵暗号方式が用いられる．共通鍵暗号方式では，平文と暗号文のサイズが等しくなることから，同サイズの入力バッファと出力バッファを持つ関数は，暗号関数である可能性が高い．

提案手法では，ある関数インスタンスによって読み書きされた, 連続したメモリ領域をそれぞれ入力バッファ，出力バッファとする，同一のメモリアドレスに対して複数読み出しが行われたときには，関数インスタンス内ではじめて読み出しが行われたときに格納されていた値を入力データと定義する．また同一のメモリアドレスに対して複数書き込みが行われたときには，関数インスタンス内で最後に書き込みが行われたときに格納された値を出力データと定義する．各関数の実行が終了した時点での入出力のサイズを比較し，同サイズの入力バッファと出力バッファのペアを持つ関数を，復号関数の可能性が高いものとして，解析

者にとって優先的に調べるべき候補とする．

提案手法の手順をまとめると以下のようになる．

(1) 解析対象のマルウェアを解析環境内で動作させ，実行トレースを取得する．

(2) 実行トレースをもとにテイント解析を行い，グラフを生成する．

(3) グラフに対してコミュニティ検出を行い，復号関数の候補を抽出する．

(4) (3) で抽出した候補の入出力を調べ，同サイズの入力バッファと出力バッファのペアを持つものを解析の優先度の高い候補とする．

4. 評価実験

4.1 実験内容

提案手法を用いることで，実行トレースに含まれる関数の中から，暗号関数候補の数をどれだけ絞り込むことができるか，また候補の中に正解の暗号関数が含まれているかを確認し，提案手法の有効性を評価した．ただし，ここでは動作中に実験対象が実行した関数のうち，実際に暗号文および平文を入出力する関数一つを正解の暗号関数とした．

一般の暗号ライブラリを用いて作成した 8 個のテスト用プログラムと web 上から収集したマルウェア検体 4 種を実験対象とした．テスト用プログラムは，暗号化したメッセージを送受信し暗号化と復号を繰り返すといった簡単な暗号化通信を行う．テスト用プログラムの作成には Beecryfpt[11]，Brian Gladman による AES 暗号ライブラリ [12]，Crypto++[13]，OpenSSL[14] の 4 種類の暗号ライブラリを用い，それぞれ AES，Blowfish，DES，RC4 の 4 種類に暗号アルゴリズムを変えたものを用意した．マルウェア検体 4 種は，C&C サーバと暗号化通信を行うボットとして，Alina，Grum，Pony，Zeus を選んだ．これらの検体は現在ソースコードが公開されており，ソースコードから実験用の検体をコンパイルすることで，それぞれ正解の関数を正しく確かめることができた．なお本実験においては，隔離したネットワーク内に構築した解析環境に C&C サーバを構築し，実行トレースを取得した．

実行トレースの取得には qemu-2.1.2[15] のソースコードを改変したものを用いた．感染 PC には 32 ビット版 Windows7 をインストールし，解析対象のプログラムを実行した後 C&C サーバとの間でパケットの送受信が確認できるまで動作させた．

取得した実行トレースのうち，暗号化パケットの受信後 20MB 分の機械語命令のログに対して提案手法を適用した．提案手法の実装には Python-2.7 を用い，コミュニティ検出には igraph パッケージ [16] を利用した．なおコミュニティ検出の手法には，igraph パッケージにてサポートされているものの中から，infomap 法に基づく手法 [17]，Random Walk に基づく手法 [18]，貪欲法に基づく手法 [19]，多段階最適化に基づく手法 [20]，スピングラス法に基づく手法 [21] を用い，結果を比較した．

4.2 実験結果

コミュニティ検出によって復号関数の候補を抽出した結果を表 2 に示す．表の項目について説明する．まず，上端の行に表示している項目は実験対象の名称と実験対象が利用した暗号アルゴリズムの種類である．ただし Alina と Grum における XOR は鍵とする文字列と平文を規則的に，またはランダムに一度だけ XOR し, 暗号文として出力するような単純なアルゴリズムであった．また，average の項目の数値は各行の実験結果の平均値である．次に左端の列に表示している項目は，コミュニティ検出に用いた手法である．ただし，All Functions の項目はグラフ生成時に読みだした実行トレースに含まれたユニークな関数の個数を表しており，In Graph の項目は生成したグラフに含まれた関数の個数を All Functions に含まれる関数の個数に対する割合で表した数値である．それぞれのマスに表示された数値が実験結果を表しており，各列の実験対象にたいして各行のコミュニティ検出手法を適用した時に，抽出された候補の個数が各実験対象における All Functtions の個数に対してどれだけの割合になったかということを表している．この値が低いほど，全体に対して少ない個数に候補を絞り込むことができたということになる．ただし，表中の網掛けのマスの候補には正解の復号関数が含まれておらず，検知漏れが発生したということを表している．

表 2 の結果を見ると，利用したコミュニティ検出の手法や実験対象にによって絞り込みの粒度にばらつきがみられたが，平均にして 0.5%から 7.5%程度にまで復号関数の候補を絞り込むことができた．これは，個数にするとおよそ 10 個から 128 個ということになる．infomap 法に基づく手法とランダムウォークに基づく手法を Pony に適用した結果と，spinglass 法に基づく手法を Alina に適用した結果に検知漏れが発生している．Pony に対して infomap 法に基づく手法とランダムウォークに基づく手法を適用した結果，グラフはそれぞれおよそ 3500 個と 1500 個の細かいコミュニティに分割されており，どちらも個数にしておよそ 3 個というわずかな数が候補として抽出されていた．しかし spinglass 法の Alina の結果については，個数にして 96 個の関数が候補として抽出されており，グラフは 80 個程度のコミュニティに分割されていたが検知漏れが発生した．以上のことから，適切なコミュニティ検出の手法を選択するにあたってその粒度とアプローチについては検討の余地はあるが，これは今後の課題とする．

次に表 2 の結果に抽出された候補を，同サイズの入出力バッファのペアを持つかという条件に基づいて，さらに絞り込んだ結果を表 3 に示す．表の見方は表 2 と同様である．平均にして 0.2%から 3.1%程度，さらに個数にしてお

表 2　コミュニティ検出によって復号関数の候補を抽出した結果

method	beeCrypt		Gladman	OpenSSL				Crypto++	Alina	Grum	Pony	Zeus	Average
	AES	Blowfish	AES	AES	Blowfish	DES	RC4	RC4	XOR	XOR	RC4	RC4	
infomap	0.79%	0.53%	0.70%	0.26%	0.35%	0.23%	0.39%	0.27%	0.66%	0.74%	0.26%	0.87%	0.50%
walktrap	3.81%	6.24%	0.25%	0.43%	0.28%	0.23%	0.32%	0.27%	8.75%	2.55%	0.26%	1.80%	2.10%
fastgreedy	6.75%	8.53%	4.00%	0.52%	7.82%	0.46%	0.39%	0.27%	6.64%	3.43%	3.25%	6.77%	4.07%
multilevel	7.62%	8.53%	2.54%	5.45%	4.54%	0.46%	0.65%	0.27%	7.03%	2.70%	5.89%	6.67%	4.36%
spinglass	8.81%	8.27%	3.24%	8.74%	3.07%	12.54%	0.65%	0.27%	4.44%	2.99%	26.39%	10.31%	7.48%
In Graph	19.37%	8.71%	20.95%	23.81%	8.37%	22.55%	22.72%	29.49%	18.33%	8.78%	68.06%	31.76%	23.58%
All Functions	1260	1137	1575	1155	1433	1308	1549	1102	2275	2038	1171	4830	1736

表 3　同サイズの入出力を持つか否かで優先順位の高い候補を絞り込んだ結果

method	beeCrypt		Gladman	OpenSSL				Crypto++	Alina	Grum	Pony	Zeus	Average
	AES	Blowfish	AES	Blowfish	DES	RC4	RC4	RC4	XOR	XOR	RC4	RC4	
infomap	0.48%	0.18%	0.06%	0.09%	0.28%	0.23%	0.19%	0.27%	0.26%	0.34%	0.00%	0.27%	0.22%
walktrap	2.86%	4.40%	0.06%	0.09%	0.21%	0.15%	0.19%	0.27%	2.99%	0.98%	0.00%	0.46%	1.05%
fastgreedy	4.92%	6.24%	0.83%	0.09%	2.16%	0.23%	0.26%	0.27%	2.51%	1.32%	1.02%	2.36%	1.85%
multilevel	5.56%	6.24%	0.19%	1.90%	1.26%	0.23%	0.32%	0.27%	2.73%	1.08%	1.79%	2.28%	1.99%
spinglass	6.75%	5.98%	1.08%	2.77%	1.12%	4.43%	0.32%	0.27%	1.76%	1.13%	8.54%	3.29%	3.12%
In Graph	15.71%	6.33%	8.57%	8.83%	2.44%	7.72%	7.23%	19.69%	5.27%	3.48%	19.13%	9.59%	9.50%
All Functions	1260	1137	1575	1155	1433	1308	1549	1102	2275	2038	1171	4830	1736

よそ 4 個から 50 個程度にまで，復号関数の候補を絞り込むことができた．ここで，Grum におけるすべての結果について検知漏れが発生しているが，これは Grum の暗号方式が平文の入力と暗号文の出力が異なるアルゴリズムを用いていたことによるものである．

コミュニティ検出手法の種類によって絞り込みの粒度や検知漏れの結果に違いが現れたが，適切な手法を選択することができれば，数パーセント以下にまで復号関数の候補を絞り込むことが可能であるとわかった．実験対象のうち，Pony，Alina，および Grum における実験結果について復号関数の検知漏れが発生した．しかしながら，実際に解析者が提案手法を利用する場面を考えると，表 2，表 3 の結果の内，候補数の少ない検出結果に含まれる候補から順に調べていくことで，Alina については全体に対して最悪でも 0.26%，Grum については全体に対して最悪でも 0.74%，また Pony については最悪でも全体に対して 1.02%，の関数を調べれば，正解の関数にたどり着くことができる．その他の実験対象についても同様のことがいえる．図 3 は，貪欲法に基づく手法を用いた場合を例に，すべての関数の個数を 100%としたときに，入出力の依存関係グラフに含まれる関数の数とコミュニティ検出によって抽出される候補の数，入出力サイズの一致によってさらに絞りこんだ候補の数の全体に対する割合を，各実験対象について表したグラフである．提案手法を用いることで，解析者は各実験対象におけるグラフのうち，それぞれ面積の少ない色分けの箇所にふくまれる候補から優先して暗号関数であるか否かを調べることができるようになる．それにより，表 2，表 3 に示すように，全体に対して復号関数の候補を相当数削減することができ，解析を効率化することができるといえる．

5. 既存研究

本章では，マルウェアのバイナリから暗号ロジックを特定する手法に関する既存研究について，特に実行トレースを用いる手法に着目し概説する．

5.1 既知のアルゴリズムを利用する手法

Gröbert らの手法 [3] は暗号アルゴリズムのコードに多く見られる機械語命令の並びや定数の組み合わせからシグネチャを作成し、それをもとに暗号ロジックの位置を特定する．しかしながら Gröbert らの手法は、機械語命令の置き換えなどの難読化が行われていた場合や、マルウェア独自の未知のアルゴリズムが用いられていた場合には，シグネチャが作用せず，暗号ロジックの位置を検出できない．

また Calvet らの手法 [4] は暗号処理に多くループが含まれることに着目し，まずプログラム内でループとなっているコード部を推定し，暗号ロジックの候補として抽出する．次に抽出したコード部の中から暗号ロジックの特定を行う．そして抽出したコード部に対する入力を別に実装した既知の暗号アルゴリズムに入力し，得られた出力を抽出したコード部からの出力と比較，一致すればそのコード部を暗号ロジックとして検出する．機械語命令の置き換えのような難読化が行われているような場合であっても，ある入力に対する出力は変化しないことから，Calvet らの手法は有効である．しかし Gröbert らの手法と同様に，既知のアルゴリズムを利用する手法であるため，未知のアルゴリズムによる暗号ロジックは検出することができない．

提案手法は既知のアルゴリズムから得られる情報を利用

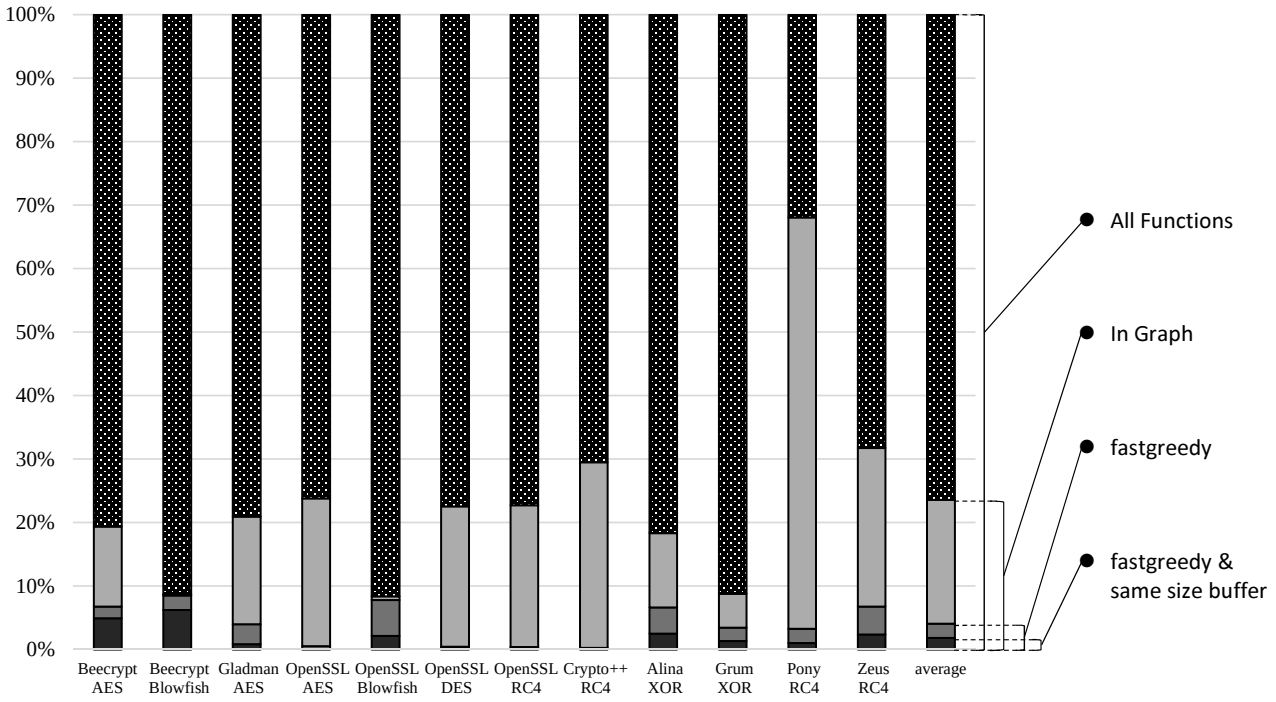

図 3 fastgreedy 法を用いた場合における復号関数候補の絞り込みの結果

しないため，未知の暗号アルゴリズムに対しても有効である．

5.2 テイント解析を用いる手法

Wang らの手法や [2] や Xin らの手法は [5] は，テイント解析を用いて，暗号化や復号の対象となるデータの伝搬をメモリ上で追跡し，そのデータに対して実行された機械語命令の特徴やデータフローの特徴から暗号ロジックの位置を推定する．

Wang らの手法は暗号処理においては，ビット演算命令や算術命令が実行される割合が多くなるが，暗号処理が終了すると，それらが実行される割合は減少するという特徴を用い，暗号処理が行われた箇所を推定する．

Xin らの手法は、マルウェアがあつかうデータの伝搬を解析し、暗号処理にあらわれる，雪崩効果とよばれる特徴をさがすことで、暗号ロジックの位置を特定する．なだれ効果とは，入力のわずかな変化が出力全体に変化を与える現象のことを言い，Xin らはこれをテイント解析によって検出している．コードに難読化が施されていたとしても，未知，既知を問わずセキュアな暗号アルゴリズムが実装されていれば入出力に雪崩効果があらわれ，雪崩効果が観測できる限り Xin らの手法は暗号ロジックを検出することができる．これらの手法はテイント解析の精度に依存し，またマルウェアがテイント解析を妨害する難読化コード [9] を用いていた場合には，単純にはデータの伝搬を追跡することができなくなるため，Xin ら手法を適用することは困難になる．

提案手法は，あえて over-tainting の発生しやすいテイント解析を用いた上で，候補を絞り込む手法であるため，正確で粒度の高いテイント解析を行う必要がなく，テイント解析への妨害するコードが用いられたとしても有効である．

6. 結論

本稿では，解析環境内でマルウェアを実行し，暗号化データを入力として読み込んだ関数と入出力をやりとりする付近の関数から，復号関数の候補を抽出する手法を提案した．

提案手法は，独自の伝搬ルールに基づいたテイント解析を行うことによって，under-tainting による検知漏れが発生しづらいといえる．そして over-tainting により増加した復号関数の候補は，コミュニティ検出の手法と共通鍵暗号方式の入出力バッファのサイズの特徴を利用して絞り込む．

提案手法は暗号アルゴリズムごとのシグネチャを必要とせず，未知の暗号アルゴリズムにも対応することが可能である．

評価実験では，実行されたすべての関数のうち，最高で 0.1%未満，平均して 2%程度にまで復号関数の候補数を絞り込むことができた．

提案手法は解析者が調べるべき暗号関数の候補を相当数削減し，マルウェアが用いる暗号関数の位置の特定を効率化，支援する．

参考文献

[1] N. Lutz, "Towards revealing attacker's intent by automatically decrypting network traffic," *Memoire de maitrise, ETH Zurich, Switzerland*(2008).

[2] Z. Wang, X. Jiang, W. Cui, X. Wang, and M. Grace, "ReFormat: Automatic Reverse Engineering of Encrypted Messages," *In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS*, vol. 5789, pp.200–215. Springer, Heidelberg (2009).

[3] F. Gröbert, C. Willems, and T. Holz, "Automated identification of cryptographic primitives in binary programs,"*In: Proc. Recent Advances in Intrusion Detection (RAID)*, pp. 41–60, Springer, (2011).

[4] J. Calvet, J. M. Fernandez and J. Marion, "Aligot: Cryptographic Function Identification in Obfuscated Binary Programs," *In:CCS' 12, Raleigh, North Carolina, USA*,pp. 169–182,October, (2012).

[5] L. Xin, W. Xinyuan, and C. Wentao , "CipherXRay: Exposing Cryptographic Operations and Transient Secrets from Monitored Binary Execution,"*In:IEEE transactions on dependable and secure computing*, vol. 11, no. 2,pp. 101–114, march/april (2014).

[6] J. Newsome and D. Song, "Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software." *In: Proceedings of the 14th Annual Network and Distributed System Security Symposium* ,(NDSS 2005), San Diego, CA (February 2005).

[7] G. Portokalidis, A. Slowinska and H. Bos, "Argos: an emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation," *In: ACM SIGOPS Operating Systems Review*, Vol. 40, No. 4, pp. 15–27, ACM(2006).

[8] L. Cavallaro, P.S axena, and R. Sekar, "On the limits of information flow techniques for—malware analysis and containment," *In: Detection of Intrusions and Malware, and Vulnerability Assessment*, pp. 143–163, Springer—Berlin Heidelberg (2008).

[9] L. Cavallaro, P. Saxena, and R. Sekar, "Anti-Taint-Analysis: PracticalEvasion Techniques Against Information Flow Based Malware Defense," *In: Secure Systems Lab at Stony Brook University*, Tech. Rep., (2007).

[10] E. J. Schwartz, T. Avgerinos, and D. Brumley. "All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask)," *In: IEEE Symposium on Security and Privacy*, Oakland, CA,USA, (May 2010).

[11] Beecrypt http://beecrypt.sourceforge.net/

[12] Brian Gladman's Home Page http://www.gladman.me.uk/

[13] Crypto++ Library 5.6.2 http://www.cryptopp.com/

[14] OpenSSL: The Open Source toolkit for SSL/TLS, https://www.openssl.org/

[15] F. Bellard. , "QEMU," http://www.qemu.org/

[16] G. Csardi and T. Nepusz, "The igraph software package for complex network research" *In: InterJournal Complex Systems 1695.5* pp. 1–9, (2006).

[17] M. Rosvall, Martin, and C. T. Bergstrom. "Maps of random walks on complex networks reveal community structure," *In:Proceedings of the National Academy of Sciences 105.4*, pp. 1118–1123, (2008).

[18] P. Pons and M. Latapy, "Computing communities in large networks using random walks," *In: International Symposium on Computer and Information Sciences*, pp. 284–293, Springer Berlin Heidelberg, (2005).

[19] A. Clauset, M. E. J. Newman and C. Moore, "Finding community structure in very large networks," *In: Phys. Rev. E 70*, pp. 066111,(December 2004).

[20] V.D. Blondel, J-L Guillaume, R. Lambiotte and E. Lefebvre, "Fast unfolding of community hierarchies in large networks," *In: J Stat Mech* pp. 10008 (2008).

[21] J.Reichardt and S. Bornholdt "Statistical mechanics of community detection," *In: Phys Rev E 74*, pp. 016110 (2006).