情報処理学会論文誌 Vol.57 No.12 2827–2835 (Dec. 2016)

推薦論文

暗号アルゴリズムの特徴を利用した軽量な暗号ブロック特定手法

西川 弘毅[1,a)] 山本 匠[1] 河内 清人[1] 桜井 鐘治[2]

受付日 2015年12月4日, 採録日 2016年9月6日

概要：昨今のマルウェアの中には，暗号技術を用いて通信データを秘匿するものがある．暗号化された通信データのログからでは，マルウェアによってどのような攻撃が行われ，どのような情報が窃取されたかを特定することが困難となる．そのため，マルウェアが暗号処理に用いている暗号アルゴリズムと鍵を特定し，暗号化された通信データを復号することで，通信の内容を明らかにする必要がある．そのような背景の下，マルウェアを実行した際の動作ログである実行トレースから，暗号アルゴリズムが有する特徴を用いて解析し，マルウェアが利用している暗号アルゴリズムを特定する手法が提案されている．しかし，これらの手法は大量の実行トレースに対して解析処理を行っているため，解析に時間を要するという課題を有している．そこで本論文では，暗号アルゴリズムが算術・ビット演算を頻繁に用いるという特徴を利用し，これらの演算が集中している箇所を計算によって特定することで，解析対象とする実行トレースを減らすことが可能である軽量な暗号ブロック特定手法を提案する．さらに実際のマルウェアの検体に対しての評価実験を行い，提案手法によって，マルウェアから暗号ブロックを特定できることを示す．

キーワード：マルウェア解析，暗号，ネットワークセキュリティ

Light Weight Identification of Cryptographic Block Using Features of Cryptographic Algorithm

Hiroki Nishikawa[1,a)] Takumi Yamamoto[1] Kiyoto Kawauchi[1] Shoji Sakurai[2]

Received: December 4, 2015, Accepted: September 6, 2016

Abstract: Recently most of malwares encrypt their communication, which makes it almost impossible to identify their malicious activities and leaked information unless the communication can be decrypted. To decrypt the communication, the encryption algorithm and the key used for the malicious communication have to be identified. So far several techniques employing execution traces have been proposed to identify encryption functions and keys in malware binaries by focusing on characteristics commonly found in cryptographic operations. Those existing techniques, however, have serious drawbacks that they consume huge amount of time and computational resources in the analysis, which makes those techniques impractical. Thus we propose light weight cryptographic function identification method spotting areas where logic operations or arithmetic operations appear with high frequency, which results in elimination of useless execution traces and much faster analysis. Moreover we carried out evaluation experiments with a wild malware to see if the proposed method can identify encryption functions used in the malware in an effective manner.

Keywords: malware analysis, encryption, network security

1. はじめに

近年，機密情報の窃取を目的に，企業や官公庁に対して

[1] 三菱電機株式会社情報技術総合研究所
Mitsubishi Electric Corporation, Information Technology R&D Center, Kamakura, Kanagawa 247–8501, Japan
[2] 三菱電機株式会社電力システム製作所
Mitsubishi Electric Corporation, Energy Systems Center, Hyogo, Kobe 652–8555, Japan
[a)] Nishikawa.Hiroki@dn.MitsubishiElectric.co.jp

本論文の内容は 2015 年 3 月の第 68 回 CSEC 研究発表会にて報告され，同研究会主査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．

行われる標的型攻撃が多発し，セキュリティ上の重大な脅威となっている．2011 年の国内重工業メーカへの標的型攻撃事例は記憶に新しいが，それ以外にも国内では多くの企業が標的型攻撃の標的となっている．警察庁によると，サイバーインテリジェンス情報共有ネットワークを通じて把握された標的型攻撃は，2014 年度上半期には 216 件に上り，前年同期比で 15 件（7.5%）の増加となっている [1]．

一般的な標的型攻撃は，巧妙に文面を細工したメールを攻撃対象に送信するところから開始される．同メールにはマルウェアを含んだ文書ファイルが添付されており，メール受信者が同文書を開いた瞬間，その端末はマルウェアに感染してしまう．攻撃者は同マルウェアをインターネット上の指令サーバ（C&C サーバ：Command and Control サーバ）から制御し，標的組織内部のネットワークから機密情報を探索，C&C サーバへアップロードすることで目的を達成する．このような情報漏えい被害の深刻化を背景に，マルウェアに感染したパソコンやサーバ等が生成したログを解析することで，マルウェアの感染端末内での挙動を明らかにするネットワークフォレンジック技術が注目されている．

しかし昨今のマルウェアの中には，通信データを暗号化することで，通信データを秘匿するものがある．このようなマルウェアの通信データは，暗号化された状態で記録されるため，そのままでは解析することができない．そのため，マルウェア解析者は，マルウェアが通信データの暗号化で用いている暗号アルゴリズムと，暗号化に用いる暗号鍵を特定して，暗号化された通信を復号することが必要となる．この作業はマルウェアのリバースエンジニアリングが必要となるため，一般に膨大な手間と時間を必要となることから，自動的にマルウェアの暗号アルゴリズムを特定可能な手法が研究されている [2], [3], [4], [5], [6], [7]．これらの手法は，マルウェアの検体を動作させて得られる動作履歴である実行トレースを解析することで，マルウェアの暗号アルゴリズムを特定している．しかし，これらの手法は，解析処理の計算量が大きく，非常に時間がかかるという課題がある．たとえば，関連研究の 1 つである Aligot [2] は数時間動作させても解析処理が終了しないため，実用性に課題があった．

そこで，まず軽量に暗号ブロック候補を特定し，従来手法による精度が高い暗号ブロック特定処理へ入力する実行トレースを絞り込むことで計算量を抑制することができると考えた．ここで暗号ブロックとは，暗号処理を含んだアセンブラ命令列のまとまりのことを指す．

本論文では，暗号処理が算術・ビット演算を頻繁に用いるという特徴を利用し，これらの演算が集中している箇所を，評価関数を用いた計算によって特定することで，暗号ブロック候補を軽量に特定する手法を提案する．

さらに，提案手法を用いて，実際のマルウェア検体から暗号ブロック候補を特定する評価実験を行い，提案方式の有効性を検証する．評価対象のマルウェアとしては，容易に入手可能であり，通信の暗号アルゴリズムとして，多くの国際的な標準暗号として選定されている 128 ビットブロック暗号 Camellia [8] を利用している Poison Ivy [9] を選択した．評価実験の結果，Poison Ivy から暗号ブロック候補を約 1.6 秒で特定することができた．さらに，特定した暗号ブロック候補で絞り込んだ実行トレースのサイズをもとに従来手法 [2] による解析時間を試算したところ，オリジナルの実行トレースの解析には約 216 日間かかるのに対し，絞り込んだ実行トレースの解析には約 2 日間しかかからないことが判明した．そのため，本手法が暗号アルゴリズム特定の高速化に寄与しているといえる．

本論文の構成は，2 章で関連研究について示し，3 章で提案手法の手順を示す．4 章で提案手法に対する評価実験を行い，5 章で誤検知等の課題について考察する．

2. 関連研究

マルウェアの検体から，マルウェアの暗号アルゴリズムを特定する研究として，文献 [2], [3], [4], [5], [6], [7] が知られている．これらの手法はマルウェアを実行して得られたログである実行トレースを解析して，マルウェアの暗号処理を構成する暗号ブロックを抽出し，解析することでマルウェアが利用している暗号アルゴリズムを特定する．Aligot での実行トレースの例を**図 1** に示す．また，実行トレース一行分の構成は，命令のアドレス，命令，命令対象，メモリ・レジスタへのアクセス情報となっている．この構成図を**図 2** に示す．一般に，実行トレースを利用した暗号アルゴリズムの特定は，リバースエンジニアリングによる静的解析と比較して難読化されたマルウェアに対しても解析が容易であるという特徴があるため，難読化が施されたマルウェアへの対抗策として有効な手法といえる．実行トレースの抽出には Pin [10] 等のツールが用いられる．

以下に，関連研究 [2], [3], [4], [5], [6], [7] について概要と課題を示す．

文献 [2] の手法である Aligot は，マルウェアの実行トレースからループ構造に着目して暗号ブロックを抽出している．この手法におけるループ検知の特徴として，for 文等の制御命令をともなわずに，同一の命令を何度も書き込むことでループを表現する展開されたループに対しても，抽出することが可能なループ検知手法を用いることで，暗号ブロックの検知精度を高めている．また，暗号アルゴリズムの特定では，マルウェアが用いる暗号アルゴリズムが既知の暗号アルゴリズムであることが多いという仮定で，暗号ブロックの入出力と一致するような既知暗号アルゴリズムの入出力を探し出し，一致した既知暗号アルゴリズムを暗号アルゴリズムとして判定している．しかし，この手法ではループを検知する際に，ジャンプ命令といった制御構

```
401055!mov esi, ecx!RR_ecx_63c02188!WR_esi_63c02188
401057!shr esi, 0x5!RR_esi_63c02188!WR_esi_31e010c
40105a!add esi, dword ptr [ebp-0x8]!RM_12ff14_4_deadbee4!RR_ebp_12ff1c_esi_31e010c!WR_esi_e1cbbff0
40105d!mov edi, ecx!RR_ecx_63c02188!WR_edi_63c02188
40105f!shl edi, 0x4!RR_edi_63c02188!WR_edi_3c021880
401062!add edi, dword ptr [ebp-0xc]!RM_12ff10_4_deadbee3!RR_ebp_12ff1c_edi_3c021880!WR_edi_1aafd763
401065!xor esi, edi!RR_esi_e1cbbff0_edi_1aafd763!WR_esi_fb646893
401067!lea edi, ptr [edx+ecx*1]!RR_ecx_63c02188_edx_28b7bd67!WR_edi_8c77deef
40106a!xor esi, edi!RR_esi_fb646893_edi_8c77deef!WR_esi_7713b67c
40106c!sub eax, esi!RR_eax_a3651d14_esi_7713b67c!WR_eax_2c516698
40106e!mov esi, eax!RR_eax_2c516698!WR_esi_2c516698
```

図 1 Aligot での実行トレースの例
Fig. 1 Example of execution trace used by Aligot.

命令のアドレス	命令 (オペコード)	命令対象 (オペランド)	メモリ・レジスタ アクセス情報

図 2 実行トレース一行分の構成図
Fig. 2 Structure of record in execution trace.

造で表現されたループの抽出とは異なり，どの命令の連続がループとなるのかを事前に予測できず，読み込むたびにループを構成する可能性がある命令列が増加していく．そのため，ループを特定するには，ループを構成する可能性がある命令列をすべて記憶し，組合せを総当たりで探索する必要があるため，処理が遅く，大量のメモリ領域を必要とするという課題があった．

文献 [3] の手法である kerckhoffr は，ジャンプ命令に着目してプログラム内のループ構造を探し出し，ループを含むブロックに対して，暗号アルゴリズムに特有の命令列や定数の組合せをシグネチャとして暗号アルゴリズムの特定を行う．本手法はジャンプ命令に着目し，暗号ブロックを特定するため，ジャンプ命令を利用せずに暗号ブロックを形成する場合，暗号ブロックを特定することができないという課題があった．

文献 [4] の手法である ReFormat は，算術・ビット演算が暗号処理では頻繁に実行されることを利用して，実行トレースの初めからこれらの演算を数え上げていき，その時点での全体における算術・ビット演算の割合を計算し，その割合のピークによって暗号ブロックを特定する．しかしこの手法では，実行トレース中に複数の暗号ブロックが存在していると，ピークを正しく検知できず，暗号ブロックを検知することができないという課題があった．

文献 [5] の手法である Dispatcher は，算術・ビット演算が，関数中の命令で占める割合を計算することで暗号ブロックを特定している．この手法では，暗号ブロックの特定は，call 命令から ret 命令の関数部分を暗号ブロックとして判断している．そのため，関数として呼ばれない暗号ブロックを特定できず，また，関数中からさらに関数を呼び出した場合はその先の命令は追わないため，多重に関数を呼んで暗号ブロックを構成している場合は検知できないという課題があった．

著者らは，文献 [6], [7] においてマルウェアが暗号アルゴリズムを利用する際に特徴があることを利用して暗号アルゴリズムを特定する手法を提案している．この手法では，実行トレースより得られた暗号ブロックと入出力との関係を，暗号ブロックの入出力特徴や，マルウェアの暗号アルゴリズム利用方法に基づいて解析することで鍵や平文を抽出する．しかし，この手法では暗号ブロックの特定に，Aligot と同様のループ特定処理を行っており，計算量が多いという課題が解決できていなかった．

3. 提案手法

本論文で提案する手法は，文献 [2], [6], [7] の従来手法で解決されていなかった課題である，暗号ブロックの特定において計算量が多いという点を解決することを目的としている．

まず，従来手法と提案手法それぞれにおける実行トレースからの暗号アルゴリズム特定処理を，**図 3** を用いて示す．従来手法で計算量が多かった原因は，暗号ブロック特定処理において精度の高いループ特定を，解析対象の実行トレース全体に対して行っていたことである．そこで提案手法では，まず軽量に暗号ブロック候補を特定し，従来手法による精度が高い暗号ブロック特定処理へ入力する実行トレースを絞り込むことで計算量を抑制することができると考えた．ここで，提案手法による暗号ブロック候補特定処理は，暗号らしさの閾値を超えた実行トレースの行を暗号ブロック候補テーブルへ登録する処理と，暗号ブロック候補テーブルへ登録された実行トレースの行番号から暗号ブロック候補を特定する処理の 2 つに分かれる．詳細については 3.2 節で示す．

基本的なアプローチは次のとおりである．まず，軽量に暗号ブロック候補を特定するために，暗号処理である確からしさを出力する評価関数を考案した．暗号ブロック候補を特定するためには，その始点と終点を抽出する必要がある．そのため，実行トレース全体において，暗号ブロック候補である部分とそれ以外の部分について，評価関数を適用した際に，その結果が著しく異なるような評価関数が必要であった．暗号処理はその特性から，add といった算術

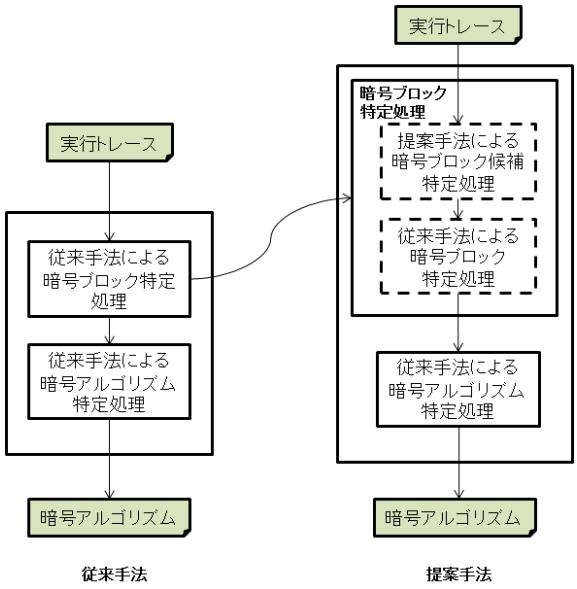

図 3 実行トレースからの暗号アルゴリズム特定処理の従来手法と提案手法との比較

Fig. 3 Comparison of the existing method with the proposed method.

演算，xor といったビット演算を多く含んでいると考えられる．そこで，これらの演算命令が集中している箇所とその範囲を特定できる評価関数を考案した．この評価関数は，非暗号ブロック候補では低い評価値を，暗号ブロック候補では高い評価値を出力するため，この評価関数を用いることで，軽量に暗号ブロック候補を特定することが可能となる．

本章では，まず評価関数について示し，次にその評価関数を用いた暗号ブロック候補を特定する手順を示す．

3.1 評価関数

本手法の評価関数に求められる要件は，(1) 処理が軽量で高速であること，(2) 暗号ブロック候補の始点と終点を特定することができること，(3) 雑音に耐性を有すること，の 3 点である．雑音は 2 種類あり，1 つは暗号ブロック候補として特定したい領域を抽出できなくなるもの，もう 1 つは暗号ブロック候補ではない部分を暗号ブロック候補として特定してしまうものである．これらの要件を考慮して，以下の式で表される評価関数を定義する．

$$F(t) = \sum_{i=0}^{N} f_i(t) \tag{1}$$

$$f_i(t) = \begin{cases} 0 & (g_i(t) < 0) \\ g_i(t) & (g_i(t) \geq 0) \end{cases} \tag{2}$$

$$g_i(t) = \begin{cases} 0 & (t < \tau_i) \\ \alpha - \delta(t - \tau_i) & (t \geq \tau_i) \end{cases} \tag{3}$$

ただし，t は経過時間（ここでは実行トレースが何行目かを指す），$F(t)$ は t における暗号処理である確からしさの値，$f_i(t)$ は i 番目に励起される減衰関数であり，N は計測対象において励起される関数 $f_i(t)$ の数（ここでは実行トレース中に含まれる算術・ビット演算の個数），τ_i は i 番目に励起される減衰関数が励起された時刻（ここでは励起されたタイミングにおいて実行トレースが何番目かを指す）である．α は暗号処理である確からしさとして定義される値であり，δ は暗号処理である確からしさの減衰量である．$f_i(t)$ は算術・ビット演算が処理されたタイミングで励起される．$f_i(t)$ は正の値しかとらないため，減衰関数 $g_i(t)$ の値が負となるときに $f_i(t)$ は 0 となる．

以下では，評価関数の妥当性について述べる．

(1) 処理の軽量・高速化

式 (1) と次節で示す手順より，評価関数の計算量は $O(m)$ で表すことができる．一方，Aligot [2] ではループ検知に $O(m^2)$ のオーダが必要と記述されている．ここで，m は実行トレースの行数である．オーダを比較すると，評価関数の方が Aligot と比べて計算量が低く，処理が高速になっていることが分かる．

(2) 暗号ブロック候補の始点と終点の特定

本評価関数の特徴は，励起した減衰関数がそれぞれ δ だけ減少していくことと，減衰関数の合計値を暗号らしさの指標としていることである．評価関数のように減衰関数を定義することにより，算術・ビット演算が連続した後では減衰関数が複数励起しており，それぞれの関数の値が減少していくため，関数が励起しなくなった段階で評価値は急激に減少する．そのため，暗号ブロック候補の終点を正確に特定することが可能となる．始点を特定するために，実行トレースを行番号が大きい方から小さい方へ順に評価関数を適用する．これらの処理によって，暗号ブロック候補の始点と終点を特定できる．

(3) 雑音耐性

暗号ブロック候補を含む部分であっても，処理の途中に算術・ビット演算以外の処理が含まれる場合がある．

このように，暗号ブロック候補として検出されることが期待される部分であるにもかかわらず，算術・ビット演算以外の処理で分断されてしまう部分であっても，評価関数の特性により，算術・ビット演算が連続する部分に近ければ，高い評価値を維持しているため，暗号ブロック候補として読み取ることができる．したがって，雑音を無視して検知することができる．一方で，暗号ブロック候補を含む部分以外でも算術・ビット演算を数命令行う場合がある．このような場合でも，算術・ビット演算命令が連続する個数に閾値（特に下限）を設けることで，暗号ブロック候補ではない部分を取り除くことができる．

3.2 暗号ブロック候補の特定手順

評価関数を用いた提案手法の手順について示す．

提案手法は大きく分けて，暗号処理である確からしさの

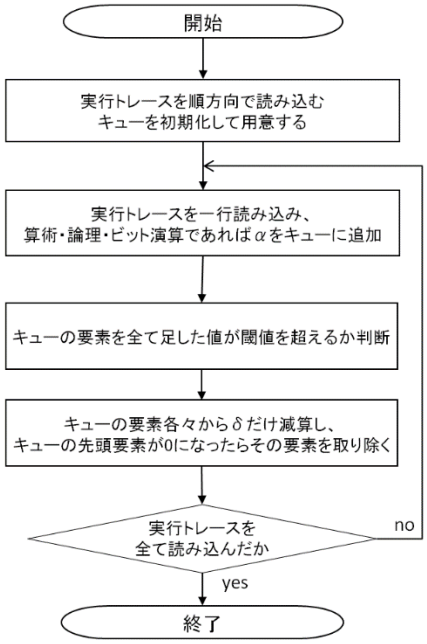

図 4 評価関数を用いた暗号ブロック候補テーブルへの登録処理フローチャート

Fig. 4 Flow diagram depicting encryption block candidate extraction.

閾値を超えた実行トレースの行を暗号ブロック候補テーブルへ登録する処理と，暗号ブロック候補テーブルへ登録された実行トレースの行番号から暗号ブロック候補を特定する処理の 2 つからなる．ここでは，この 2 つの処理について示す．

まず，暗号ブロック候補テーブルへ登録する処理について示す．**図 4** に動作のフローチャートを示し，以下で詳細に示す．

ここで，実行トレースの行番号を実行トレース ID と呼ぶ．また，暗号ブロック候補テーブルは，暗号ブロック候補となりうる実行トレース ID が登録されたものである．

(1) 実行トレースを順方向で読み込む．暗号処理である確からしさを示す値を記録するキューを初期化して用意する．
(2) 入力された実行トレースを 1 行（以下，実行トレース行）読み込む．
(3) 読み込んだ実行トレース行の命令部分を確認し，命令が算術・ビット演算である場合は (4) へ，そうでない場合は (5) を実行する．
(4) キューに暗号処理である確からしさとして定義される値 α を追加する．
(5) キューの中身すべてを加算した値を算出し，その値が，暗号ブロック候補として見なす閾値である M を超えている場合 (6) へ，そうでない場合は (7) へ移る．
(6) 現在読み込んでいる実行トレース ID を暗号ブロック候補テーブルへ登録する．
(7) キューに含まれている要素すべてから各々 δ だけ減算処理を行う．
(8) キューの先頭要素が 0 以下である場合 (9) へ，そうでない場合は (10) へ移る．
(9) デキュー処理を行い，キューの先頭要素を削除し，他の要素を前へ詰める．
(10) 現在読み込んでいる実行トレース行が，初めに入力した実行トレースの最後の行であるかどうかを確認する．最後である場合処理を終え，そうでなければ (2) へ戻る．

上記の実行トレース ID の順に処理するのと同様に，逆方向からも同様の処理を適用する．

次に暗号ブロック候補テーブルへ登録された実行トレース ID に対して行う処理について示す．

このとき，暗号ブロック候補テーブルには，暗号処理である確からしさである評価値が閾値 M を超えた実行トレース ID が登録されている．本処理は単純であるため，フローチャートは省略する．

(1) 登録された実行トレース ID が連続している場合，連続している個数を数える．
(2) 連続している個数が閾値 L を超えている部分を，数え上げを始めた実行トレース ID を始点，連続の最後となっている実行トレース ID を終点として持つ暗号ブロック候補として特定する．

4. 評価実験

本章では，提案手法の有効性を検証するために，評価実験と，評価実験を行うための準備として予備実験を行う．

4.1 予備実験

予備実験の目的は，本手法により，暗号ブロック候補を適切に特定できるパラメータを決定することである．予備実験では，暗号処理を行う 2 つのプログラム [11], [12] を用意した．

ここで，TEA と RC4 の 2 つのプログラムを選択した理由は，ソースコードが公開されていることと，ブロック暗号と，ストリーム暗号を比較できるためである．本論文では研究の最初のステップとして，パラメータを決定するための暗号を 2 種類に限定しているが，様々な暗号に対応できるよう今後種類を増やすことが必要であると考えている．

予備実験の手順は次のとおりである．

(1) 予備実験の対象プログラムである，TEA，RC4 を実行するプログラムから実行トレースを取得する．
(2) 取得した実行トレースに対して，本手法を実行し，出力される暗号ブロック候補数が，1 となるようにパラメータを調整する．

この予備実験により，条件に沿ったパラメータ値を，ヒューリスティックに求めた結果，次のようになった．

表 1 実行トレースからの暗号ブロック候補特定結果
Table 1 Results of extraction block candidates from execution trace.

プログラム名	実行トレースサイズ[MB]	処理時間[s]	暗号ブロック候補数
7zip [13]	19.8	1.2	2
PoisonIvy [9]	11.7	1.6	296

$$[\alpha,\quad \delta,\quad M,\quad L] = [15,\quad 1,\quad 10,\quad 50] \tag{4}$$

4.2 本実験

式 (4) に示した，予備実験の結果から得られたパラメータを用いて，通信で暗号処理を行う Poison Ivy [9] と，圧縮処理を行う 7zip [13] に対して評価実験を行う．ここで Poison Ivy を取り上げた理由は，公開されているため入手しやすいこと，標的型攻撃において現在も用いられていること，暗号アルゴリズムとして，多くの国際的な標準暗号として選定されている 128 ビットブロック暗号 Camellia [8] を用いており，評価する暗号アルゴリズムとして適切であると判断したためである．また，7zip を評価対象に含めた理由は，暗号処理に似た圧縮処理についても検証するためである．

本手法は解析する実行トレースの量を削減することが目的である．そのため，本手法によって特定した暗号ブロック候補が，暗号処理を行っている範囲を含んでいれば，本実験は成功と見なす．

評価実験の手順は次のとおり．

(1) 実験対象のプログラムから実行トレースを取得する．このとき，暗号処理や圧縮処理を実行させるように調整する．
(2) 実行トレースに対して，本手法を適用する．出力としては，始点と終点を持った暗号ブロックを得る．このとき，始点と終点は実行トレース ID により定義されている．

比較として，ソースが公開されており，実際に動かすことができる Aligot により Poison Ivy の実行トレースを解析したところ，数時間経過しても処理が終了しなかったのに対し，**表 1** に示す実験結果から分かるように，処理時間は長くても 2 秒弱である．

次に，それぞれの特定された暗号ブロック候補が，暗号処理を行う範囲を正しく含んでいるか調査した．

圧縮処理を行った 7zip は，2 個の暗号ブロック候補を得た．ここで，解析対象とした 7zip の実行トレースは，テキストファイルを zip に圧縮する際に得られたものである．Poison Ivy では，296 個の暗号ブロック候補が得られた．

特定した暗号ブロックの実行トレースと，Poison Ivy 実行時のメモリイメージ，Camellia の仕様を比較した結果，本手法で特定した暗号ブロック候補が，暗号処理部分を含んでいることが確認できた．以下に，その根拠を示す．

```
00400ECF    90              NOP
00400ED0    8B06            MOV EAX,DWORD PTR DS:[ESI]
00400ED2    3345 00         XOR EAX,DWORD PTR SS:[EBP]
00400ED5    D7              XLAT BYTE PTR DS:[EBX+AL]
00400ED6    C1C8 08         ROR EAX,8
00400ED9    D7              XLAT BYTE PTR DS:[EBX+AL]
00400EDA    D0C0            ROL AL,1
00400EDC    C1C8 08         ROR EAX,8
00400EDF    D7              XLAT BYTE PTR DS:[EBX+AL]
00400EE0    D0C8            ROR AL,1
00400EE2    D0C4            ROL AH,1
00400EE4    C1C8 08         ROR EAX,8
00400EE7    D7              XLAT BYTE PTR DS:[EBX+AL]
00400EE8    3107            XOR DWORD PTR DS:[EDI],EAX
00400EEA    C1C0 08         ROL EAX,8
00400EED    3107            XOR DWORD PTR DS:[EDI],EAX
00400EEF    C1C0 08         ROL EAX,8
00400EF2    3147 04         XOR DWORD PTR DS:[EDI+4],EAX
00400EF5    C1C0 08         ROL EAX,8
00400EF8    3107            XOR DWORD PTR DS:[EDI],EAX
00400EFA    3147 04         XOR DWORD PTR DS:[EDI+4],EAX
00400EFD    8B46 04         MOV EAX,DWORD PTR DS:[ESI+4]
00400F00    3345 04         XOR EAX,DWORD PTR SS:[EBP+4]
00400F03    D7              XLAT BYTE PTR DS:[EBX+AL]
00400F04    D0C0            ROL AL,1
00400F06    C1C8 08         ROR EAX,8
00400F09    D7              XLAT BYTE PTR DS:[EBX+AL]
00400F0A    D0C8            ROR AL,1
00400F0C    D0C4            ROL AH,1
00400F0E    C1C8 08         ROR EAX,8
00400F11    D7              XLAT BYTE PTR DS:[EBX+AL]
00400F12    C1C8 08         ROR EAX,8
00400F15    D7              XLAT BYTE PTR DS:[EBX+AL]
00400F16    C1C8 08         ROR EAX,8
00400F19    8AC8            MOV CL,AL
00400F1B    32CC            XOR CL,AH
00400F1D    C1C8 10         ROR EAX,10
00400F20    32C8            XOR CL,AL
00400F22    32CC            XOR CL,AH
00400F24    8AE9            MOV CH,CL
00400F26    66:33C1         XOR AX,CX
00400F29    C1C0 10         ROL EAX,10
00400F2C    66:33C1         XOR AX,CX
00400F2F    3107            XOR DWORD PTR DS:[EDI],EAX
00400F31    3147 04         XOR DWORD PTR DS:[EDI+4],EAX
00400F34    C3              RETN
```

図 5 暗号ブロック候補が含んでいた関数
Fig. 5 Function found in encryption function candidate block.

```
Registers (FPU)
EAX 0012F040
ECX 0011E93C
EDX A4F00303
EBX 00400FA0 test_ser.00400FA0
ESP 0011E94C
EBP 0012FAE3
ESI 001269B3
EDI 001269BB
```

図 6 関数の処理開始時のレジスタ
Fig. 6 Register values just before executing the entry point of the function.

ここでは，本手法により特定した暗号ブロック候補に含まれる関数について検証する．この関数が Camellia であることを示すことで，今回特定した暗号ブロック候補が，暗号処理をとらえていることを示す．この関数を**図 5** に示す．

この関数では，XLAT 命令により参照しているテーブルを追跡した．このときの EBX は**図 6** のように 00400FA0 (16 進数表現) であった．さらに，このアドレスから始まるメモリの値を**図 7** に示す．Camellia の仕様と見比べると，これは Camellia の S-Box であり，この処理が Camellia を用いており，暗号処理であることが確認できた．したがって，本手法により特定した暗号ブロック候補は，実際に暗号処理を行う部分を特定することに成功したといえる．

図 7 XLAT が EBX により参照しているメモリ
Fig. 7 Memory referred by XLAT via EBX.

5. 考察

5.1 計算量について

本節では，3.1 節で示した計算量の議論をふまえて，本手法を用いた暗号ブロック特定処理と Aligot による暗号ブロック特定処理の時間の違いを考察する．

今回の実験で明らかになったように，本手法によれば，11.7 MB の実行トレースに対して 1.6 秒で暗号ブロック候補を特定することができる．一方，Aligot による暗号ブロック特定は，数時間が経過しても，処理が終了しなかった．これは本手法の計算量が $O(m)$ であるのに対し，Aligot の計算量が $O(m^2)$ であるからと考えられる．ここで，m は実行トレースのサイズである．すなわち Aligot における処理時間を概算すると次のようになる．

$$11.7 \times 10^6 \times 1.6 = 18.72 \times 10^6 \text{ [s]}$$

これは，時間換算すると 5,200 時間であり，約 216 日かかる計算となる．そのため，Aligot による暗号ブロック特定処理は膨大な時間が必要となり，実用的な手法とはいえない．一方，5.2 節で詳細を説明するが，提案手法においては，1.6 秒で暗号ブロック候補を抽出した後に，抽出した暗号ブロック候補に対して従来手法による暗号ブロックの特定を実施したとしても，約 2 日で処理が完了する試算となる．そのため，提案手法は実用的な手法であるといえる．

5.2 検知精度について

本実験で起きた誤検知について考察する．原因としては，(1) 暗号処理とは関係ないが算術・ビット演算命令を頻度高く行う部分を検出していることと，(2) 本手法による暗号ブロック候補特定がメモリ上のアドレス単位ではなく実行トレース単位であること，の 2 つが考えられる．

原因の (1) として，xor や shr といった命令を繰り返す，暗号処理とは関係のない処理についても暗号ブロック候補として検出していた．この課題については，算術・ビット演算以外の暗号処理に特有なその他の情報も利用することで，切り分け精度を高めることで対応できると考えている．暗号処理に特有なその他の情報の検討については今後の課題とする．

原因の (2) として，本手法ではメモリ上のアドレスによる暗号ブロック候補特定ではなく，実行トレース単位での暗号ブロック候補特定となっているため，同一の命令列を繰り返し検出していることがある．**図 8** に，この例を示す．これは，Poison Ivy の実行トレースから，本手法により抽出した暗号ブロック候補のうちから 2 つ抽出したものである．図 8 から分かるように，それぞれ，まったく同一の命令を実行している．このように，別の暗号ブロック候補として特定されたものの中には，同一命令を実行しているため，メモリ上のアドレスにおいては同一の暗号ブロック候補と見なすことができるものが多数存在した．これらを取り除いた結果，本手法で特定した暗号ブロック候補数は 25 個となった．このような処理の最適化については，類似したアドレスを指す暗号ブロック候補に関しては，同一暗号ブロック候補と見なすようにして，暗号ブロック候補数を削減することが考えられる．

ここで，暗号ブロック候補が 296 個（最適化すると 25 個）と大量に得られた Poison Ivy に焦点を当てて検知精度の妥当性について議論する．まず，Poison Ivy の実行トレースにおける，False Positive Rate（FPR）と False Negative Rate（FNR）について述べる．FPR は，今回特定したブロックの中の，暗号処理を含まないブロックの割合を表す．今回特定したブロックのうち，暗号処理を含んでいたブロックの数（True Positive）は 25 個中 10 個であり，暗号処理を含んでいないブロックの数の割合，すなわち FPR は 60%となる．

また，Poison Ivy 実行中のメモリイメージを解析した結果，今回の検出対象である Poison Ivy の暗号処理部分がすべて検出できていた．そのため，今回の実験では暗号処理を含んでいるブロックを検出できない割合 False Negative Rate（FNR）は 0%である．

次に本手法の FPR が 60%であることを許容できることを示す．本手法で抽出した暗号ブロック候補に対応する実行トレースは，952 KB であった．

ここで，抽出した 25 個の暗号ブロック候補を解析すると，アドレスの範囲が他のブロックに包含されているものが複数見つかった．そこで，他の暗号ブロック候補に包含されている暗号ブロック候補を同一のものと見なすことで，暗号ブロック候補を 4 つに絞ることができた．これにより，実行トレースを 144 KB まで絞り込むことができた．つまり，オリジナルの実行トレース（ファイルサイズが 11.7 MB）の約 1%にまで絞り込むことができたことになる．

```
*==blockNo: 295 : 303661 -> 304995 :1334==*
400cff!mov ebp, dword ptr [ebp+0x10]!RM_d4fde4_4_d4fdfa!RR_ebp_d4fdd4!WR_ebp_d4fdfa
400d02!add ebp, 0x40!RR_ebp_d4fdfa!WR_ebp_d4fe3a
400d05!mov eax, dword ptr [edi]!RM_d57f5b_4_8!RR_edi_d57f5b!WR_eax_8
400d07!mov edx, dword ptr [edi+0x4]!RM_d57f5f_4_8!RR_edi_d57f5b!WR_edx_8
400d0a!xor eax, dword ptr [ebp]!RM_d4fe3a_4_3a260102!RR_eax_8_ebp_d4fe3a!WR_eax_3a26010a
400d0d!xor edx, dword ptr [ebp+0x4]!RM_d4fe3e_4_e5bd5392!RR_edx_8_ebp_d4fe3a!WR_edx_e5bd539a

*==blockNo: 296 : 305003 -> 306337 :1334==*
400cff!mov ebp, dword ptr [ebp+0x10]!RM_d4fde4_4_d4fdfa!RR_ebp_d4fdd4!WR_ebp_d4fdfa
400d02!add ebp, 0x40!RR_ebp_d4fdfa!WR_ebp_d4fe3a
400d05!mov eax, dword ptr [edi]!RM_d57f6b_4_3d4!RR_edi_d57f6b!WR_eax_3d4
400d07!mov edx, dword ptr [edi+0x4]!RM_d57f6f_4_253!RR_edi_d57f6b!WR_edx_253
400d0a!xor eax, dword ptr [ebp]!RM_d4fe3a_4_3a260102!RR_eax_3d4_ebp_d4fe3a!WR_eax_3a2602d6
400d0d!xor edx, dword ptr [ebp+0x4]!RM_d4fe3e_4_e5bd5392!RR_edx_253_ebp_d4fe3a!WR_edx_e5bd51c1
```

図 8 同一の命令が複数の暗号ブロックで実行される例

Fig. 8 Example of a sequence of instructions found in both 2 different execution trace blocks.

最終的に絞り込んだ実行トレースのサイズから試算すると，暗号ブロック候補を絞り込んだうえで従来手法による暗号ブロックの特定を行う場合，約 2 日で処理が完了することになり，現実的な処理時間になる．よって，今回の検知精度は妥当な結果といえる．

5.3 パラメータについて

α と δ は，利用目的は異なるが，どちらも実行トレースの t 行目の暗号処理である確からしさ $F(t)$ の度合いを決定するパラメータである．α は算術・ビット演算単体の暗号処理である確からしさを表現したものである．

δ は，実行トレース上での命令列の暗号処理である確からしさの文脈を考慮しない度合いを表したパラメータである．δ が大きくなると，実行トレースの t 行目より前に現れた実行トレースの暗号処理である確からしさを引き継がない傾向になる．実行トレースの前の行の暗号処理である確からしさを引き継ぐことで，算術・ビット演算が密集している領域の合間に現れるような算術・ビット演算以外の演算も暗号処理らしい領域としてとらえることが可能となる．

α を大きく δ を小さく設定すると，算術・ビット演算単体の暗号処理である確からしさの度合いが増し，さらに，実行トレースの t 行目より前に現れた実行トレースの暗号処理である確からしさも引き継ぐようになるため，実行トレースの t 行目の暗号処理である確からしさが増加しやすくなる傾向にある．

M は実行トレースの t 行目の暗号処理である確からしさが暗号ブロック候補に属するかどうかを判定するための閾値である．そのため，M を大きくすれば暗号処理である確からしさがより高い領域に絞った検出を行うことになる．

L は，暗号のブロックとして抽出する際の最低限のブロックのサイズを指定する値である．暗号処理を行う一連の処理の長さに適した値を設定するとよい．

どのパラメータも FPR や FNR に密接に関係しているため，事前準備で適切なパラメータ設定を実施する必要がある．

5.4 解析妨害機能の回避について

実行トレースを利用する手法に共通の課題として，マルウェアに解析対象の処理を実行させなければならないことがある．多くのマルウェアは，解析を妨害するために様々な工夫をしており，解析環境においては正常な挙動を観測することができない．今後，マルウェアから，全自動で暗号処理を特定するシステムを考える際には，これらの解析妨害機能を自動で回避して，マルウェアに解析対象の暗号処理を実行させるようにしなければならない．

6. むすび

暗号処理が算術・ビット演算を頻繁に用いるという特徴を利用し，これらの演算が集中している箇所を計算によって特定することで，暗号ブロックを軽量に特定する手法を提案した．さらに，提案手法を用いて，実際のマルウェアから暗号ブロックを特定する評価実験を行った．評価対象のマルウェアとしては，容易に入手可能であり，通信の暗号アルゴリズムとして，多くの国際的な標準暗号として選定されている Camellia を利用している Poison Ivy を選択した．

評価実験の結果，Poison Ivy から暗号ブロック候補を約 1.6 秒で特定することができた．さらに，特定した暗号ブロック候補で絞り込んだ実行トレースのサイズをもとに Aligot による解析時間を試算したところ，オリジナルの実行トレースの解析には約 216 日間かかるのに対し，絞り込んだ実行トレースの解析には約 2 日間しかかからないことが判明した．そのため，本手法が暗号アルゴリズム特定の高速化に寄与することを示すことができた．

今後は，複数のマルウェア検体に対しての評価や，本提案方式における誤検知の低減，解析妨害機能の回避について研究を進める．

謝辞 本論文の執筆および評価実験において多大なご支援をいただいた三菱電機の松井充氏，IPA の時田俊雄氏に感謝いたします．

参考文献

[1] 警察庁：平成 26 年上半期のサイバー空間をめぐる脅威の情勢について，入手先 ⟨http://www.npa.go.jp/kanbou/cybersecurity/H26_kami_jousei.pdf⟩.
[2] Calvet, J., Fernandez, J.M. and Marion, J-Y.: Aligot: Cryptographic Function Identification in Obfuscated Binary Programs, *Proc. ACM Conference on Computer and Communications Security* (2012).
[3] Grobert, F., Willems, C. and Holz, T.: Automated Identification of Cryptographic Primitives in Binary Programs, *Proc. 14th International Conference on Recent Advances in Intrusion Detection* (2011).
[4] Wang, Z., Jiang, X., Cui, W., Wang, X. and Grace, M.: ReFormat: Automatic ReverseEngineering of Encrypted Messages, *Proc. European Symposium on Research in Computer Security* (2009).
[5] Caballero, J., Poosankam, P., Kreibich, C. and Song, D.: Dispatcher: Enabling Active Botnet Infiltration using Automatic Protocol Reverse-Engineering, *Proc. ACM Conference on Computer and Communications Security* (2009).
[6] 山本 匠，西川弘毅，河内清人，中嶋純子，桜井鐘治：暗号ロジック特定手法の提案，コンピュータセキュリティシンポジウム 2014，pp.835–842 (2014).
[7] 山本 匠，西川弘毅，河内清人，中嶋純子，桜井鐘治：暗号ロジック特定手法の提案 その 2，2015 暗号と情報セキュリティシンポジウム (2015).
[8] 128 ビットブロック暗号 Camellia アルゴリズム仕様書，入手先 ⟨https://info.isl.ntt.co.jp/crypt/camellia/dl/01jspec.pdf⟩.
[9] FireEye, Poison Ivy Assessing Damage and Extracting Intelligence, available from ⟨https://www.fireeye.com/content/dam/fireeye-www/global/en/current-threats/pdfs/rpt-poison-ivy.pdf⟩.
[10] Luk, C.-K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V.J. and Hazelwood, K.: Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation, *Programming Language Design and Implementation* (*PLDI*), Chicago, IL, pp.190–200 (June 2005).
[11] Aligot Project: aligot – Cryptographic function identification in obfuscated programs, r120, available from ⟨http://code.google.com/p/aligot/source/browse/trunk/vanilla/testBinaries/TEA/TEA.exe⟩.
[12] Aligot Project: aligot – Cryptographic function identification in obfuscated programs, r120, available from ⟨https://code.google.com/p/aligot/source/browse/trunk/vanilla/testBinaries/RC4/RC4.exe⟩.
[13] 7zip, available from ⟨http://7-zip.org/⟩.

推薦文

プログラムの実行トレース情報をもとに，暗号化に使用されると考えられる演算ブロックを高速に抽出するという手法に新規性があり，提案手法の動作検証も行って効果も示している．

（コンピュータセキュリティ研究会主査 鳥居 悟）

西川 弘毅 （正会員）

2014 年東京理科大学大学院工学研究科修了．同年三菱電機株式会社情報技術総合研究所入社．情報セキュリティ技術に関する研究・開発に従事．

山本 匠 （正会員）

2006 年静岡大学情報学部情報科学科卒業．2007 年 9 月同大学大学院修士課程修了．2010 年 9 月同創造科学技術大学院博士課程修了．日本学術振興会特別研究員（DC1），同研究員（PD）を経て，2011 年 4 月三菱電機株式会社情報技術総合研究所入社．情報セキュリティに関する研究に従事．2014 年 10 月 University of California，Berkeley 客員研究員（～2016 年 3 月）．博士（情報学）．

河内 清人 （正会員）

1996 年慶應義塾大学大学院理工学研究科修了．同年三菱電機株式会社入社．現在，情報技術総合研究所にて，情報セキュリティ技術に関する研究・開発に従事．

桜井 鐘治 （正会員）

1989 年九州大学工学部電子工学科卒業．同年三菱電機株式会社入社．以来，OS/ネットワーク，セキュリティの研究･開発に従事．現在，電力システム製作所にて，制御システムセキュリティの開発に従事．ACM，IEEE-CS 各会員．