Vol.2015-DPS-162 No.26
Vol.2015-CSEC-68 No.26
2015/3/5

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]　山本　匠[†1]　河内　清人[†1]　桜井　鐘治[†1]

昨今のマルウェアの中には，暗号技術を用いて通信を秘匿するものがある．暗号化された通信ログからでは，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ロジックと鍵を特定し，暗号化された通信ログを復号することで，通信の内容を明らかにする必要がある．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，暗号ロジックが有する特徴を用いて，マルウェアが利用している暗号ロジックを特定する手法が提案されている．しかし，これらの技術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．そこで本稿では，暗号処理が算術・論理・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求めることで，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暗号ブロックを軽量に特定する手法を提案する．

Light Weight Identification Cryptographic Block Using Features of Cryptographic Functions

Hiroki Nishikawa[†1] Takumi Yamamoto[†1] Kiyoto Kawauchi[†1] Shoji Sakurai[†1]

Recently, some malwares encrypt communication to conceal what transmitted is. When the communication is encrypted, it becomes difficult to identify what malware attack is, what information is stolen from communication logs. Therefore, identifying encryption logic and key is needed to reveal the communication. Some methods to identify encryption logic from execution traces using features of encryption logic are proposed. However, those methods analyze a lot of execution traces, those take a lot of time. Therefore, based on the feature that encryption uses arithmetic, logic and bit operation frequently, we propose light weight analyzing method to identify encryption block, which can decrease analysis targets of execution traces. The proposed method can decrease the amount of execution trace to be analyzed by calculating the code blocks on which arithmetic, logic or bit operations are concentrated.

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件は記憶に新しいが，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メール攻撃は，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]．米セキュリティ企業 FireEye 社によると，2013 年度に標的型攻撃のターゲットとされた企業は，金融(15%)，メディア及びエンタテイメント(13%)，製造(10%)，航空宇宙・防衛(6%)となっており [2]，国防，社会インフラだけでなく様々な業種へと標的型攻撃のターゲットの裾野が拡大している事が伺える．

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，ユーザ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，アップロードすることで目的を達成する．これらの情報漏洩被害の深刻化を背景に，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，暗号技術を用いて通信を秘匿するものがある．このようなマルウェアの通信は，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，マルウェアが通信で用いている暗号ロジックと，暗号化に用いる鍵を特定して，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り，膨大な手間と時間を必要とするため，自動的にマルウェアの暗号ロジックを特定できるツールが求められ，[3]-[8]において研究されている．

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，マルウェアの暗号ロジックを特定している．しかしこれらの手法は，処理に時間がかかる，誤検知が多いといった課題がある．そこで本稿では，暗号処理が算術・論理・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求めることで，解析対象とする実行トレースを特定した暗号処理の部分に絞り込み，暗号ロジックを構成する命令群である暗号ブロックを軽量

[†1] 三菱電機株式会社
Mitsubishi Electric Corporation

Vol.2015-DPS-162 No.26
Vol.2015-CSEC-68 No.26
2015/3/5

```
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 実行トレースの例

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

に特定する手法を提案する．

本論文の構成は，2.で関連研究について示し，3.で提案手法の説明を行い，4.で本提案手法の考察を行う．

2. 関連研究

マルウェアの検体から，マルウェアの暗号ロジックを特定する技術として，文献[3]-[8]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，暗号ロジックを構成する暗号ブロック候補を抽出し，抽出された候補を更に解析することでマルウェアが利用している暗号ロジックを特定する．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．実行トレースを利用した暗号ロジックの特定は，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に強い特徴があるため，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段といえる．実行トレースの抽出には Pin[9]等のツールを用いる．

以下に，関連研究[3]-[8]について概要と課題を説明する．

文献[3]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 文などの制御命令を伴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できる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，暗号ロジックの特定では，マルウェアが用いる暗号ロジックが既知の暗号ロジックであることが多いという仮定で，暗号ブロック候補の入出力と一致するような既知暗号ロジックの入出力を探し出し，一致した既知暗号ロジックを暗号ロジック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造で表現されたループではないため，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．ループを特定するためには，この増加するループを構成する可能性がある命令列を全て記憶し，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課題があった．また，マルウェアが未知の暗号ロジックを利用している場合，検知することができない課題がある．

文献[4]の手法である kerckhoffr は，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，暗号ロジックに特有の命令列や定数の組み合わせをシグネチャとして暗号ロジックの特定を行う．シグネチャを用いた解析であるため，特定できる暗号ロジックは，Aligot と同様に既知のものに限定されてしまう課題がある．

文献[5]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課題がある．

文献[6]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ロジックを特定している．この方式では，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．この方式では，関数として呼ばれない暗号ブロックを特定できず，また，関数中から更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，いくつも関数を呼んで暗号ロジックを構成している場合は検知できない課題がある．

著者らは, [7][8]においてマルウェアが暗号ロジックを利用する際に特徴があることを利用して暗号ロジック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ロジック候補と入出力との関係を，暗号ロジックの入出力特徴や，マルウェアの暗号ロジック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

今回提案する手法は，[7][8]の手法で解決されていなかった課題である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ロジック特定処理を図 3 を用いながら簡単に説明する．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．今回提案する手法は，高精度な暗号ブロック特定手法を実行する前に，実行トレースから軽量に暗号ブロックを特定することを目的としている．

軽量に暗号ブロックを特定するために，暗号ブロックらしさを出力する評価関数を考案した．暗号ブロック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 といった算術演算，xor といった論理演算，shl 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，非暗号ブロックでは低い評価値を，暗号ブロックでは高い評価値を出力するため，この関数を用いることで，軽量に暗号ブロック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の説明を行い，次にここで提案した評価関数を用いた暗号ブロックを特定する手順，動作例を説明する．

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 つは暗号ブロックとして抽出したい領域を抽出できなくなるもの，もう 1 つは暗号ブロックではない部分を暗号ブロックとして抽出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．

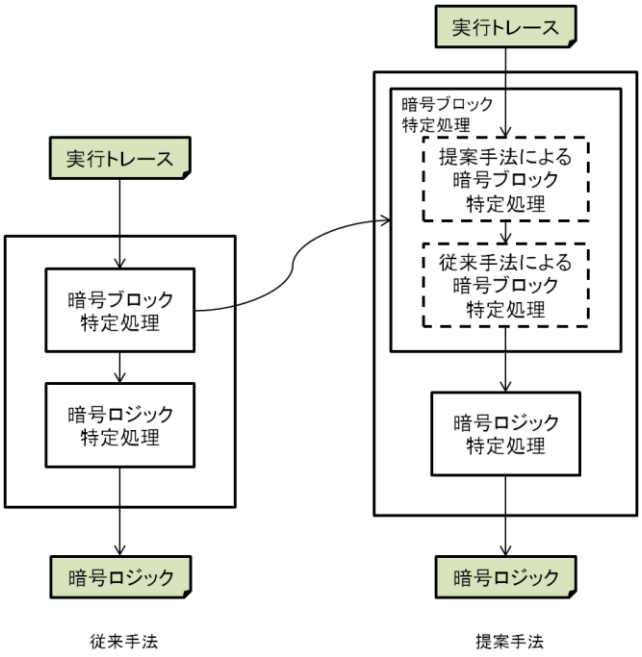

図 3 実行トレースからの暗号ロジック特定処理の従来手法と提案手法との比較

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

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

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

ただし，$F(t)$ は t における暗号らしさの値，$f_i(t)$ は i 番目に励起される減衰関数であり，t は経過時間(ここでは実行トレースが何番目かを指す), 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・論理・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 は暗号ブロックである確からしさとして定義される値であり，δ は暗号ブロックである確からしさの減衰量である．$f_i(t)$ は算術・論理・ビット演算が処理されたタイミングで励起される．$f_i(t)$ は正の値しか取らないため，減衰関数 $g_i(t)$ の値が負となる時に $f_i(t)$ は 0 となる．

以下では，評価関数の妥当性について説明する．

(1)処理の軽量・高速化

式(1)と後述する手続きより，評価関数の計算量は $O(m)$ で表すことができる．一方，Aligot[3]ではループ検知に $O(m^2)$ のオーダー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．オーダーを比較すると評価関数の方が計算量が低く，処理が高速になっていることが

分かる．処理の軽量化については考察で述べる．
(2)暗号ブロック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．式(3)のように減衰関数を定義することにより，算術・論理・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，暗号ブロックの始点と終点を特定できる．
(3)雑音耐性
暗号ロジックを含む部分であっても，処理の途中に算術・論理・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，暗号ブロックとして検出されることが期待される部分であるにも関わらず，上記演算が連続しない部分でも，連続部分に近ければ暗号ブロックとして読み取ることができるため，多少の雑音を無視して検知することができる．一方で，暗号ロジックを含む部分以外でも算術・論理・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，連続する個数に閾値を設けることで，暗号ブロックではない部分を取り除くことができる．

3.2 暗号ブロックの特定

評価関数を用いた提案手法の手続きについて説明する．また，次節で動作例を示す．
提案手法は大きく分けて，評価関数を用いたブロック候補抽出処理と，抽出されたブロック候補から暗号ブロックを特定する暗号ブロック特定処理の 2 つからなる．ここでは，この 2 つの処理について説明する．
まず，ブロック候補抽出処理について説明する．図 4 に動作のフローチャートを示し，以下で詳細に説明する．
ここで, 実行トレースの行番号を実行トレース ID と呼ぶ．

(1)実行トレースを順方向で読み込む．暗号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一行（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・論理・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)キューに暗号ブロックである確からしさとして定義される値αを追加する．
(5)キューの中身全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全てから各々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．

上記の処理と同様に，逆方向からもブロック候補抽出処理を適用する．
次に暗号ブロック特定処理について説明する．

(1)まず，暗号ブロック候補テーブルには，ブロック候補抽出処理によって，暗号ブロック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．
(2)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．
(3)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として特定する．

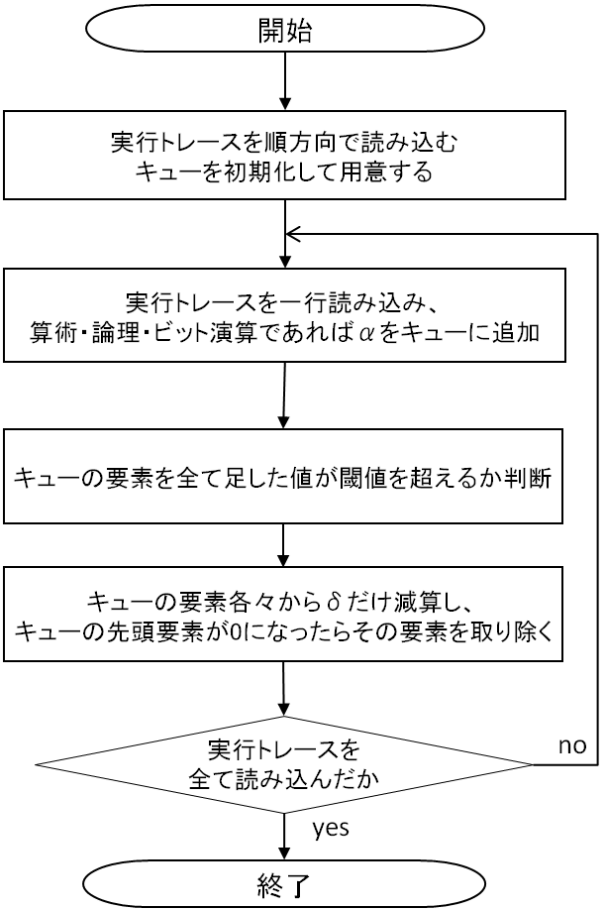

図 4 ブロック候補抽出処理のフローチャート

表 1 動作の例示（順方向）

実行トレース ID	命令ありなし	キュー	評価値
0		0	0
1	○	5	5
2		4	4
3	○	3, 5	8
4	○	**2, 4, 5**	***11***
5	○	**1, 3, 4, 5**	***13***
6	○	**2, 3, 4, 5**	***14***
7	○	**1, 2, 3, 4, 5**	***15***
8	○	**1, 2, 3, 4, 5**	***15***
9		**1, 2, 3, 4**	***10***
10		1, 2, 3	6
11		1, 2	3
12	○	1, 5	6
13	○	**4, 5**	***9***
14		3, 4	7
15	○	**2, 3, 5**	***10***
16		1, 2, 4	7
17		1, 3	4
18		2	2
19		1	1
20		0	0
21		0	0
22	○	5	5
23		4	4
24		3	3
25	○	2, 5	7
26		1, 4	5
27		3	3
28		2	2
29	○	1, 5	6
30		4	4

表 2 動作の例示（逆方向）

実行トレース ID	命令ありなし	キュー	評価値
0		1, 2, 4	7
1	○	**1, 2, 3, 5**	***11***
2		**1, 2, 3, 4**	***10***
3	○	**1, 2, 3, 4, 5**	***15***
4	○	**1, 2, 3, 4, 5**	***15***
5	○	**2, 3, 4, 5**	***14***
6	○	**3, 4, 5**	***12***
7	○	**4, 5**	***9***
8	○	1, 5	6
9		1, 2	3
10		2, 3	5
11		1, 3, 4	8
12	○	**2, 4, 5**	***11***
13	○	3, 5	8
14		4	4
15	○	5	5
16		0	0
17		0	0
18		1	1
19		2	2
20		3	3
21		1, 4	5
22	○	2, 5	7
23		3	3
24		4	4
25	○	1, 5	6
26		2	2
27		3	3
28		4	4
29	○	5	5
30		0	0

3.1 動作例

提案手法の動作例を示す.

表 1 は，31 行からなる実行トレースに対して，順方向でブロック候補抽出処理を実行した際のキューとその合計値の推移を示した表である．実行トレース ID は，この命令が何行目の実行トレースであるかを示しており，暗号ブロック特定処理で有効に働く．この例ではこの実行トレースは, 算術・論理・ビット演算命令を何か所かに含んでおり，ここでは簡略のために，その箇所を図中の「命令ありなし」で丸印が付いている箇所とする．ここで $\alpha = 5$，$\delta = 1$ としている．キューには現在励起された減衰関数の値が一つ一つ格納されており，実行トレース ID が一つ進むたびに δ だけ減少していく．評価値はキューに格納された値を全て足したものである．

閾値 M が 8 の時, 暗号ブロックである確からしさであるキューの合計値を出力とすれば，出力が 8 を超える時の実行トレース ID が暗号ブロック候補テーブルに登録される．これを逆方向からも同様に行う．表 2 はその様子を示しており，実行トレース ID が高い方向から低い方向へ向かって計算が行われる．

順方向，逆方向の両方の処理で，それぞれの処理において出力が 8 を超える時の実行トレース ID が登録される．ただし，逆方向の処理時に既に登録されているものは登録しない．ここまでの登録処理によって得られる暗号ブロック候補テーブルの状態が図 5 である．

閾値 L が 5 の時，実行トレース ID が 1～9 のものは連続する個数が 9 個であるため，この部分を暗号ブロックとみなす．一方，12～13 は，連続しているが，連続する個数が

2 個であるため暗号ブロックとみなさない．この時，評価値を縦軸に，実行トレース ID を横軸にして，順方向と逆方向それぞれに対して描画したグラフが図 6 である．

4. 考察

4.1 評価

評価関数の処理の軽量・高速化についてと，提案手法のプロトタイプを TEA[10]に適用した際の結果を示す．

評価関数の処理が Aligot と比較して高速であることは 3.1 で述べたので，ここでは処理が軽量となることを考察する．Aligot[3]ではループを検知する際に，一行実行トレースを読み込むたびにループの候補を増やす必要がある．このループ候補は，ループ検知のアルゴリズムで必要となるため，実行トレース一行全ての情報を記録する必要がある．これらのループ候補を保持する必要があるため，メモリを大量に必要とする．一方で提案手法で保持しておく必要があるものは，ブロック候補抽出処理においてのキューと，ブロック候補テーブルの二つで，記録する情報も評価値や ID なので軽量である．

提案手法を TEA に適用した結果, 図 7 に示すアドレスが 401043 から 40109F までのコード領域を特定した．この領域は TEA の暗号化処理を行っている部分であり, 提案手法を用いることで暗号ブロックを特定することができた．動作時間は 11msec であった．

4.2 課題

本手法は算術・論理・ビット演算が暗号ロジックが有する特徴である点に着目して暗号ブロックを特定する手法である．そのため，マルウェア作成者が意味のない XOR や ADD 命令等を大量にコードへ加えた場合, 本手法が有効に働かない．対策としては，ある範囲のスコープにおいて意味がないと判断される XOR 等の命令列を取り除くといった前処理を施すことが考えられる．

5. おわりに

本稿では，マルウェアの実行トレースから，暗号ブロックを軽量に特定する手法を示した. 既に提案した[7][8]の手法と組み合わせることで, 暗号ロジック特定処理を素早く，メモリ消費を抑えて実現できる．今後は提案手法に対して様々な検体を試し，提案手法の有効性を検証していく．

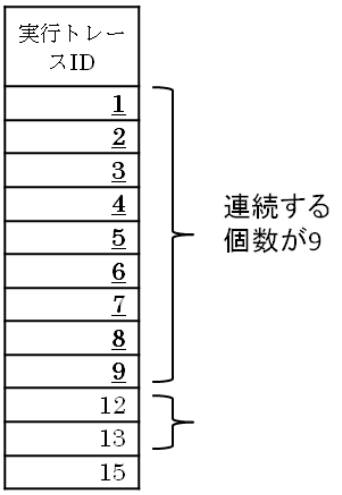

図 5 ブロック候補テーブルの様子

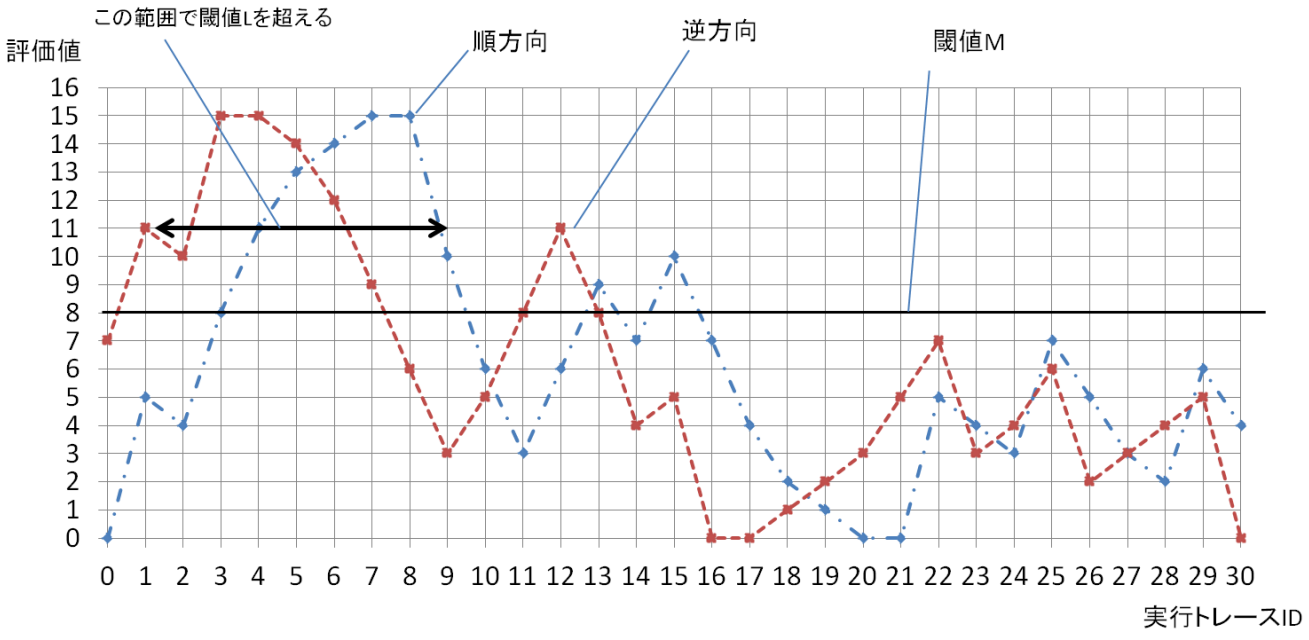

図 6 動作例における評価値のグラフ

```
.text:00401043                  mov     edx, 0C6EF3720h
.text:00401048                  mov     [ebp+var_C], edi
.text:0040104B                  mov     [ebp+var_8], esi
.text:0040104E                  mov     [ebp+var_4], 20h
.text:00401055
.text:00401055 loc_401055:
.text:00401055                  mov     esi, ecx
.text:00401057                  shr     esi, 5
.text:0040105A                  add     esi, [ebp+var_8]
.text:0040105D                  mov     edi, ecx
.text:0040105F                  shl     edi, 4
.text:00401062                  add     edi, [ebp+var_C]
.text:00401065                  xor     esi, edi
.text:00401067                  lea     edi, [edx+ecx]
.text:0040106A                  xor     esi, edi
.text:0040106C                  sub     eax, esi
.text:0040106E                  mov     esi, eax
.text:00401070                  shr     esi, 5
.text:00401073                  add     esi, [ebp+var_10]
.text:00401076                  mov     edi, eax
.text:00401078                  shl     edi, 4
.text:0040107B                  add     edi, [ebp+var_14]
.text:0040107E                  xor     esi, edi
.text:00401080                  lea     edi, [edx+eax]
.text:00401083                  xor     esi, edi
.text:00401085                  sub     ecx, esi
.text:00401087                  add     edx, 61C88647h
.text:0040108D                  dec     [ebp+var_4]
.text:00401090                  jnz     short loc_401055
.text:00401092                  mov     edx, [ebp+arg_0]
.text:00401095                  pop     edi
.text:00401096                  mov     [edx], ecx
.text:00401098                  mov     [edx+4], eax
.text:0040109B                  pop     esi
.text:0040109C                  mov     esp, ebp
.text:0040109E                  pop     ebp
.text:0040109F                  retn
.text:0040109F sub_401020       endp
```

図 7 提案方式で特定した TEA の暗号ブロック

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,
http://www.npa.go.jp/kanbou/cybersecurity/H26_kami_jousei.pdf
2) FireEye: M-Trends 2014 Annual Threat Report: Beyond the Breach
https://www2.fireeye.com/fireeye-mandiant-m-trends-report.html
3) Joan Calvet, Jose M. Fernandez and Jean-Yves Marion: Aligot: Cryptographic Function Identification in Obfuscated Binary Programs, Proceedings of the ACM Conference on Computer and Communications Security (2012).
4) Felix Grobert, Carsten Willems and Thorsten Holz: Automated Identification of Cryptographic Primitives in Binary Programs, Proceedings of the 14th International Conference on Recent Advances in Intrusion Detection (2011).
5) Wang, Z., Jiang, X., Cui, W., Wang, X., Grace, M.: ReFormat: Automatic ReverseEngineering of Encrypted Messages, Proceedings of the European Symposium on Research in Computer Security (2009).
6) Caballero, J., Poosankam, P., Kreibich, C., Song, D.: Dispatcher: Enabling Active Botnet Infiltration using Automatic Protocol Reverse-Engineering, Proceedings of the ACM Conference on Computer and Communications Security (2009).
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, コンピュータセキュリティシンポジウム 2014, pp. 835- 842 (2014).
8) 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).
9) Intel: Pin - A Dynamic Binary Instrumentation Tool,
http://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool
10) Aligot Project: aligot - Cryptographic function identification in obfuscated programs, r120,
http://code.google.com/p/aligot/source/browse/trunk/vanilla/testBinaries/TEA/TEA.exe