L-028

多段パックされたマルウェアからのコード取得

Code Capture from Self-Modifying Malwares

中村 徳昭 †　森井 昌克 †　伊沢 亮一 ‡　井上 大介 ‡　中尾 康二 ‡
Noriaki Nakamura　Masakatu Morii　Ryoichi Isawa　Daisuke Inoue　Kouji Nakao

1 はじめに

マルウェアの持つ機能の全体像を把握するには逆アセンブラやデバッガなどを利用して静的解析するのが有効である．しかし，近年のマルウェアのほとんどは静的解析を妨害するためのパッカーと呼ばれるツールにより，難読化，解析妨害機能の具備が施されている．これらのマルウェアを解析する為には，解析妨害機能を回避しつつ難読化を解き，マルウェアの本来のプログラムコードであるオリジナルコードを取りだす作業，アンパックが必要となる．アンパックには複数の方法があるが，通常は既知のパッカーに対応した専用のアンパッカーを用いなければならない．数多くのパッカーに対応した RL!Depacker[1] といった強力なアンパッカーも存在するが，実行可能な状態までアンパックできないケースも多い．特に，インポートアドレステーブル（IAT）の値をヒープに作成したダミーコードのアドレスに書き換えてしまうような多段パッカーに対して，IAT のリビルドを正確に行うことは難しい．現在このような手法を用いたパッカーには tElock[2]，PESpin[3]，yoda's Crypter[4] が挙げられ，このような多段パッカーから完全なオリジナルコードが取得できないことが現在大きな課題となっている．

そこで本論文では，このような多段パッカーに対して，IAT をリビルドする以前のオリジナルコードを抽出し，それを用いてマルウェアの特徴を把握する手法を提案する．パック以前のオリジナルコードを取得できているかどうかはメモリアドレス中のオペコードを比較することで評価を行う．また，取得できたオリジナルコードの有用性を示すため，これらの部分的なコードを用いた類似度判定を行う．類似度判定では，コードを「ベーシックブロック」と呼ばれる一定の基準に基づいて分割し，各検体のベーシックブロックの一致率を調べることにより算出する．

本提案手法の評価のため，マルウェア 168 検体と 3 種類の多段パッカー (tElock,PESpin,yoda's Crypter) を用意し，2 つの実験を行った．実験 1 では，すべてのマルウェア検体とパッカーの組み合わせでパッキングを行い，提案手法でメモリダンプした結果，元検体のオリジナルコードのオペコード部分をほぼすべて取得することができた．実験 2 では，取得したコードをベーシックブロックに分割し，各検体同士のブロック一致率を調べることにより，マルウェア検体間の類似度を算出することに成功した．また，類似度結果の可視化を行うことで，マルウェアの特徴を視覚的に把握できるようにした．本提案手法を用いることで，多段パックされたマルウェア検体から部分的なオリジナルコードを取得することができ，オペコード部分をブロック単位で比較することによってマルウェアの特徴を把握することが可能となる．

† 神戸大学大学院工学研究科，Graduate School of Engineering, Kobe University
‡ 独立行政法人情報通信研究機構，National Institute of Information and Communications Technology

2 関連研究

近年ではコードに着目した類似度算出手法が数多く提案されており，コードから得られる制御フローをグラフ等に変換し構造の比較を行う手法 [5] や，バイナリコードを処理ブロックに分割し，ブロック単位で比較を行う手法 [6] などがある．これらの手法では，パッカーによる圧縮・暗号化処理に対しては既に何らかの方法でアンパックが成功しており，解析対象となるコード本来の動作を示すオリジナルコードが取得済みであることが前提となっている．

そこで，ステルスデバッガや DBI (Dynamic Binary Instrumentation) により，書き込みや実行といったメモリアクセスを監視することでオリジナルコードを推定する手法 [7, 8, 9] が数多く検討されている．また，DBI による実行命令トレースをシグネチャマッチングすることによってパッカーの種類を特定する手法 [10] も提案されている．しかし，DBI を用いた手法では実際にマルウェア検体自体を実行する必要があるため，tElock や PESpin といった，自己を書き変える機能を持つ強力なパッカーに対しては実行命令を全く取得できないという課題がある．加えて，マルウェアが解析時に必ず特徴的な挙動を示すという保障がない点や，マルウェアによる様々な解析環境検知機能により，重要な挙動が隠蔽される可能性がある点も問題として挙げられる．文献 [11] では，このような多段パッカーに対して，オリジナルエントリポイントを発見する手法が提案されているが，完全にアンパックすることはできない．

3 多段パックされたマルウェアに対するコード取得

この章では，多段パックマルウェアに対してどのようにオリジナルコードを取得する手法を述べる．まず 3.1 節では一般的なパックと多段パックではどのようにセクション構造が異なっているのかを説明し，3.2 節で多段パックに用いられている「Import Redirection」という手法について説明する．最後に 3.3 節で多段パックからコードセクションをメモリダンプする手順を述べる．

3.1 多段パックのセクション構造

パッカーはファイルの本体であるセクションを暗号化して圧縮を行う．特にコードセクションはサイズも大きく，ローダーによる初期化処理には不要な部分であり，またクラッカーの目から最も隠したい部分であるため，他のセクションよりも優先的に圧縮される．ここでは通常のパック (UPX) と多段パック (tElock) のセクションテーブルを比較し，動作や構造がどのように

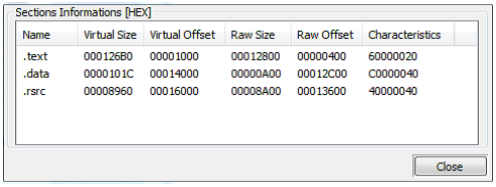

図 1　オリジナルの calc.exe のセクション

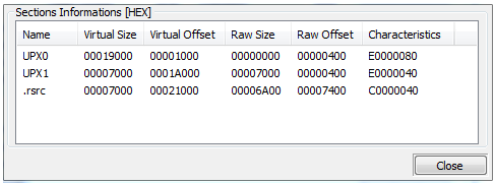

図 2　UPX でパックされた calc.exe のセクション

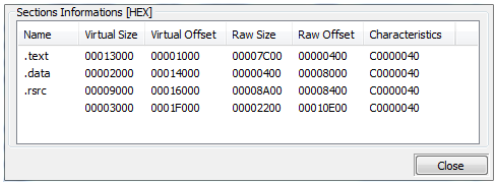

図 3　tElock でパックされた calc.exe のセクション

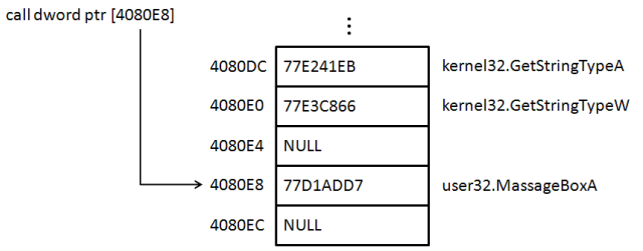

図 4　IAT を用いた通常の API 呼び出し

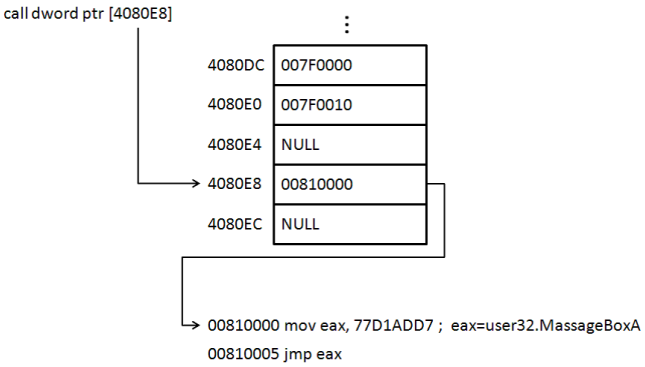

図 5　Import Redirection の例

異なるのかを示す．サンプルファイルとして calc.exe を用意し，それぞれ UPX，tElock でパックしてセクション構造の違いを確認する．図 1 にオリジナルの calc.exe のセクションを，図 2 に UPX でパックされた calc.exe(calc_UPX.exe) のセクションを，図 3 に tElock v0.98 でパックされた calc.exe(calc_tElock.exe) のセクションをそれぞれ PETools[12] で表示したものを示す．

図 1 と図 2 のセクションテーブルを比較すると，オリジナルに対して UPX でパックされたものはセクション数は 1 つ減っており，セクション名が UPX0 と UPX1 となっている．UPX0 セクションの RawsSize（ファイル中のサイズ）が 0 であるので，ファイル中では UPX 0 セクションは存在せず，メモリ上で初めて実体を持つということがわかる．また，エントリポイントはオリジナルが 00012475 であるのに対し，UPX でパックされたものは 00020CA0 であることから UPX1 セクションに復号ルーチンと圧縮データが存在し，メモリ上の UPX0 セクションに text セクションと data セクションがまとめて展開されることになる．

一方，図 1 と図 3 のセクションテーブルを比較すると，オリジナルに対して tElock でパックされたものは無名のセクションがひとつ増加している．また，エントリポイントはオリジナルが 00012475 であるのに対し，tElock でパックされたものは 00020BD6 であり，新たに加えられたセクションにエントリポイントが移されている．これらのことより，加えられた無名のセクションに圧縮データとそれを展開する為の復号ルーチンが入っているとわかる．

3.2　Import Redirection

IAT のリビルド時や逆アセンブラでの解析時に，インポートしている API をわかりにくくする手法が Import Redirection と呼ばれている手法である．通常，IAT には図 4 に示すようにインポートした API のアドレスが格納されている．したがって，IAT を参照すれば，どのような API をインポートしているかを把握することができる．しかし，Import Rediredtion が行われている場合，IAT は API のエントリポイントではなく，API を呼びだすための関数のアドレスに書き換えられており，そのアドレスを見ただけではどのような API が呼び出されるかわからないといった問題がある．

また，API を呼びだすための関数内では，単純に call 命令だけで API に制御を移すのではなく，いったん汎用レジスタに API のアドレスを格納してから jmp 命令で関数を呼び出す，などのやりかたをとる場合があり，書き換えの手法は一様ではない．

図 5 の例では mov 命令を使った後にジャンプ（jmp）する形で API 呼び出しを行っているが，push 命令で API のアドレスをスタックに積み，ret 命令で API のエントリポイントに飛ぶ方法など，多段パッカーの種類によってさまざまなやり方が考えられる．このため，多段パックされたマルウェアに対して IAT のリビルドを完全に行うことは非常に困難である．

3.3　多段パックからのオリジナルコード取得

ここでは，多段パックされた EXE ファイルから OEP を検出し，オリジナルコードをメモリダンプする手法を示す．以下の 3 つ Step を行うことにより，オリジナルコードを取得する．

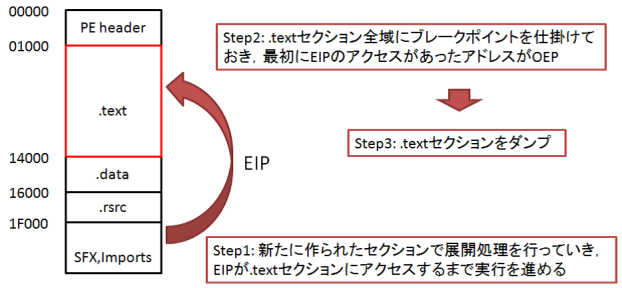

図 6　多段パックからのオリジナルコード取得

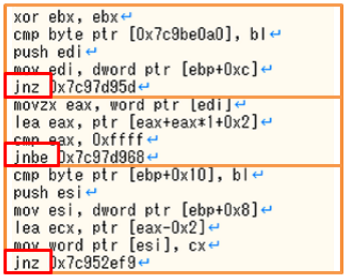

図 7　ベーシックブロック分割

オリジナルコード取得手順

Step1 展開ルーチン処理の実行

Step2 コードセクション上でのアクセス検出

Step3 コードセクションのメモリダンプ

3 つの Step の実行の様子を図 6 に示す．まず Step1 では，多段パックによって新たに作られたセクションにエントリポイントが移されているので，このセクション内で 1 ステップずつ展開処理を行う．展開ルーチンによる処理が終了後，コードセクションにオリジナルコードの書き込みが行われるので，書き込みが終了した時点でコードセクション全域にメモリアクセスブレークポイントを仕掛ける．次に Step2 では，インストラクションポインタ（EIP）の制御がコードセクションへと移る瞬間をブレークポイントによって捉える．ここで停止した地点がオリジナルエントリポイントとなる．最後に Step3 では，さきほど Step2 で停止した地点をエントリポイントとしてコードセクションのメモリダンプを行い，展開されたオリジナルコードを取得する．

なお今回の実験では，デバッガには OllyDbg v1.10[13] を，メモリダンプには OllyDbg のプラグインである OllyDump[14] を用いた．

4　オリジナルコードのオペコード部分を用いたマルウェア間の類似度判定

ここでは，3 章で提案した手法によって取得したオリジナルコードのオペコード部分を用いて，多段パックの影響を受けないようなマルウェア間の類似度を判定する手法を述べる．4.1 節でオリジナルコードを扱う単位となる「ベーシックブロック」の概念について説明し，4.2 節で類似度判定の手法について説明する．

4.1　ベーシックブロックの導入

本研究で取得できたオリジナルコードは 1 検体につき数万行に渡って構成される場合がほとんどである．自己書き換え型マルウェアから取得したコードは，IAT の呼び出し部分が多段パックによって書き換えられているため，本来のオリジナルコードとは異なる部分も存在する．また，このコードが特定の命令長で一致しても同じ機能を有しているとは言い難い．そこで，図 7 に示すように，このオペコード列をベーシックブロック単位で分割して比較を行う．ベーシックブロックとは，「分岐命令・分岐先命令で区切られた連続した命令列」と定義する．このブロック単位で比較を行う理由としては，分岐命令にまでの連続した命令が行われることで，マルウェアの一つの機能に関する一連の動作を行っていると考えられるからである．以後，本手法の類似度判定ではベーシックブロック単位でコードを分割して扱う．

4.2　ベーシックブロックを用いた類似度判定

オリジナルコードのオペコード部分をベーシックブロック単位で用いて，マルウェア間の類似度判定を行う手法を説明する．マルウェア間の類似度はベーシックブロックの一致率によって算出する．マルウェア A に対するマルウェア B の類似度を $sim(A, B)$ とし，$sim(A, B)$ を以下の式 (1) で定義する．

$$sim(A, B) = \frac{C_A \cap C_B}{C_A} \quad (1)$$

- C_A:マルウェア A の全コードブロック
- C_B:マルウェア B の全コードブロック

$sim(A, B)$ の値が大きいほど，マルウェア A はマルウェア B のコード部分を多く含むことになり，マルウェア B の機能を多く有していると考えられる．

5　類似度の可視化

本章では，複数のマルウェア間の類似度を可視化するための手法およびその結果を示す．まず，5.1 節で可視化システムの概要を述べ，5.2 節でグラフ可視化の意味および可視化モデルの種類について説明し，5.3 節で可視化システムに用いた手法を説明する．

5.1　概要

類似度を可視化することにより，複数の検体間の類似度を総当りで算出しても，複雑な類似度関係を容易に把握することができる．可視化はノード (点) とそれらを結ぶエッジ (辺) から成るグラフを構築することによって実現する．各マルウェア検体をノードとして，類似度の高いノード同士をエッジで結びつける．エッジのつながりが密になっているほど，検体同士の距離が短くなるように配置 (マッピング) する．マッピング結果からつながりが疎である辺を取り除くことにより，クラスタリングを行う．マッピング手法には Fruchterman-Reingold 法（FR 法）[15] を，クラスタリング手法には Newman 法 [16] を用いる．なお，本可視化システムは JUNG[17] を用いて

図 8　検体 A から検体 B へのつながり

図 9　双方向からのつながり

実装を行った．

5.2　グラフ可視化

可視化とは，人間が直接見ることのできない関係性などを，画像や映像，グラフ，図，表などにより見ることのできるものにすることをいう．本論文では，類似度関係の情報はグラフとして表現できるものであるので，グラフにより情報を可視化する．隣接行列表現などで表された要素とその要素間の接続関係を低次元空間へマッピングする問題のことをグラフ埋め込み問題という．これまで，グラフ埋め込み問題を定式化するための様々なモデルが考案されている．例えば以下のようなものがある，

- **多次元尺度法** 個体間の親近性データを，2 次元あるいは 3 次元空間に類似したものを近く，そうでないものを遠くに配置する手法
- **バネモデル** エッジを仮想的なバネとみなして，力学的に安定するノードの配置を求める手法
- **スペクトラルクラスタリング** ノードを球面上に配置する手法

本システムではバネモデルを使用する．バネモデルはノードやエッジにどのような力が働くかによって複数の種類がある．ここでは，Fruchterman-Reingold 法を用いる．

5.3　辺 (エッジ) をつなぐ条件

本可視化システムでは，各検体同士のつながりを類似度によって決定する．4 章で定義した類似度 $sim(A, B)$ を用いて，検体 A から検体 B へ辺をつなぐ条件を以下の式 (2) によって定義する．

$$sim(A, B) = \frac{C_A \cap C_B}{C_A} > \mathrm{Z} \tag{2}$$

- C_A:検体 A の全コードブロック
- C_B:検体 B の全コードブロック
- Z:しきい値

検体 A のコードブロックが検体 B 中に一定の割合以上含まれていれば，図 8 のように検体 A から検体 B へ辺をつなぐ．検体 A と検体 B を入れ替えてもこの式が成り立てば，図 9 のように双方向から辺をつなぐ．この割合はしきい値 Z としてシステム上でユーザが自由に調整できるようになっている．

5.4　可視化に用いた手法

5.4.1　Fruchterman-Reingold 法（FR 法）

グラフ中の検体間の距離は Fruchterman-Reingold 法 (FR 法) によって定義する．FR 法は，Fruchterman, Reingold らが考案したモデルであり，既存のバネモデルを拡張してノードとエッジに働く力を再定義したものである．具体的には，エッジをバネと仮定した上でエッジの両端に

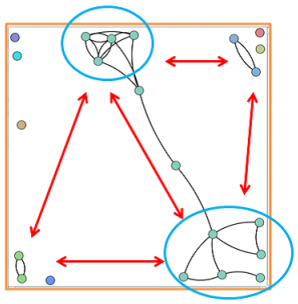

図 10　Fruchterman-Reingold 法（FR 法）

あるノードに対しては，ノード間距離 d の二乗に比例する引力 f_a が働き，全ての 2 頂点間にはノード間の距離に反比例する斥力 f_r が働くというモデルである．図 10 のように，ノード間のつながりが密になっているほど強い引力 f_a で結びつき，つながりのないノード同士は斥力 f_r のみが働いて引き離される．引力 f_a と斥力 f_r は以下の式 (3)，式 (4) で定義する．FR 法の特徴は，ノードやエッジに働く力が単純なので，比較的高速に計算を行うことができることである．

$$f_a = \frac{d^2}{k} \tag{3}$$

$$f_r = -\frac{k^2}{d} \tag{4}$$

5.4.2　Newman 法

Newman 法は，同一クラスタ (に含まれるノード) 間にはエッジが多く，異なるクラスタ間にはエッジが少なくなるようにノードをクラスタリングするアルゴリズムである．Newman 法では，クラスタリングの最適さを以下の関数 Q で定義し，この値が大きくなるようにクラスタリングを行う．

$$Q = \sum_i (e_{ii} - {a_i}^2) \tag{5}$$

ただし，

$$e_{ij} = \frac{i \text{ 番目と } j \text{ 番目のクラスタ間のエッジ数}}{\text{全エッジ数}} \tag{6}$$

$$a_i = \sum_k e_{ik} \tag{7}$$

である．Newman 法では，通常のボトムアップクラスタリングと同様に，はじめにノードと同じ数だけをクラスタを作成する．そして，Q の値が最も大きく増加するように，2 つのクラスタを統合する．i 番目と j 番目のクラスタを統合したときの Q の増分は以下の式で求めることができる．

$$\Delta Q = e_{ij} + e_{ji} - 2a_i a_j = 2(e_{ij} - a_i a_j) \tag{8}$$

上記の処理は Q の値が極大値に達するまで続ける．Newman 法の利点の一つは，クラスタ数をあらかじめ設定しておく必要がないことである．アルゴリズム終了後はノー

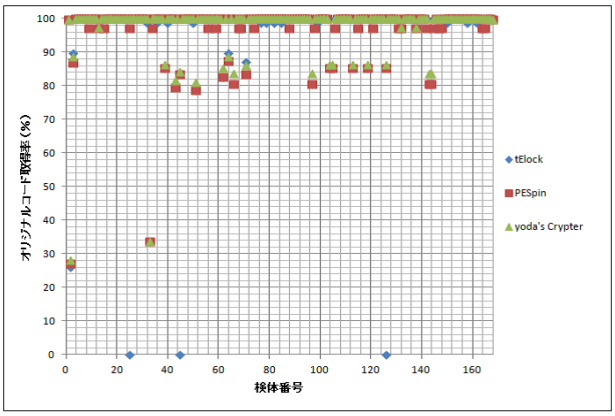

図 11　多段パックに対するオリジナルコード取得率

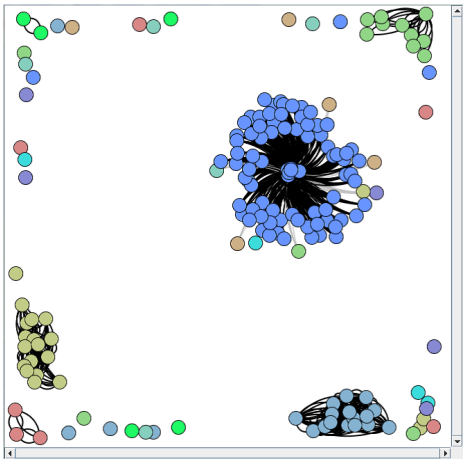

図 12　パックされていない 168 検体の可視化結果

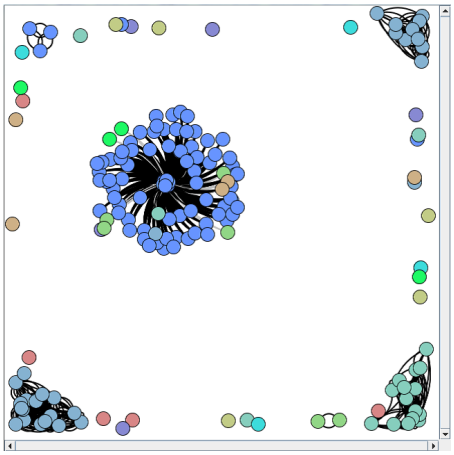

図 13　tElock でパックされた 168 検体の可視化結果

ドを任意のクラスタ数に分けることができ，ユーザ自身が解析を行いやすいようなクラスタリングを実現できる．

6　評価実験

本章では提案手法の評価のために行った 2 つの実験手法およびその結果について述べる．

6.1　実験 1：多段パックからのオリジナルコード取得

実験 1 では，多段パックされたマルウェア検体から提案手法によって元のオリジナルコードのオペコード部分を取得できるかどうかを評価する．そのため，168 検体のマルウェア検体に対して 3 種類の多段パッカーを用い，すべての組み合わせでパッキングを行い，パックする以前のオリジナルコードとオペコード部分で比較する．なお，コード比較は提案手法の 4 章で説明したベーシックブロック単位で行った．図 11 に各多段パックに対するオリジナルコード取得率を示す．横軸は検体番号，縦軸は元検体と比較したときのオリジナルコード取得率である．図 11 に示す通り，ほぼすべてのパッカーと検体の組み合わせでパック以前のオリジナルコードのベーシックブロックを取得できていることがわかる．しかしながら，数検体についてはパック以前のオリジナルコードを十分に取得できなかった．特に全くコードを取得できなかったパターンも存在したが，これは検体とパッカーの相性が悪かったことが原因だと考えられる．

6.2　実験 2：ベーシックブロックを利用したマルウェアの類似度判定

実験 2 では，各検体から取得したベーシックブロックを用いて，パックごとにマルウェアの類似度判定を行えるかどうかを評価する．検体は実験 1 で用いた 168 検体を用いて，総当たりで類似度判定を行う．その類似度関係を可視化システムによって表すことで，パッキングに影響されない類似度判定が行えることを示す．類似度判定は，(1) パック無し 168 検体，(2)tElock でパックされた 168 検体，(3)PESpin でパックされた 168 検体，(4)yoda's Crypter でパックされた 168 検体の 4 つのパターンに対して行った．今回，式 (2) で設定した可視化システムのしきい値はすべて $Z = 90(\%)$ である．

各パッカー別の類似度可視化結果をそれぞれ図 12，図 13, 図 14, 図 15 に示す．また，実験に使用した検体のアンチウイルスソフト（TrendMicro[18] または Symantec[19]）によって定義される科名を表 1 に示す．

これらの可視化結果より，3 種類の多段パッカーでパックされた検体に対して，パック前とほぼ同様の類似度判定が行えることが示せた．また，各クラスタリング結果はアンチウイルスソフトによって定義されるマルウェア科名と一定の整合性を確認することができたため，類似度判定の精度も確かめることができた．このようにマルウェア間のつながりを可視化することで，検体の起源や全体の関係を容易に把握することができる．

7　まとめ

本稿では，多段パックされたマルウェアから IAT をリビルドする以前のオリジナルコードを取得し，オペコード部分をマルウェア間で比較することで類似度判定を行う手法を提案した．提案手法の評価方法として，マルウェア 168 検体に対して多段パックを施してメモリダンプを行い，パックする以前のオリジナルコードが取得できるかを検証した．その結果，ほぼすべての検体と多段パッカーの組み合わせに対してオリジナルコードを取得する

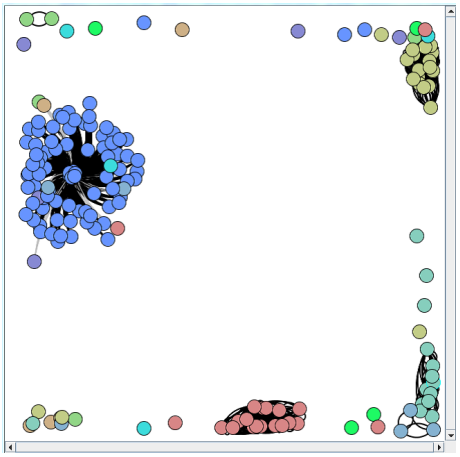

図 14　PESpin でパックされた 168 検体の可視化結果

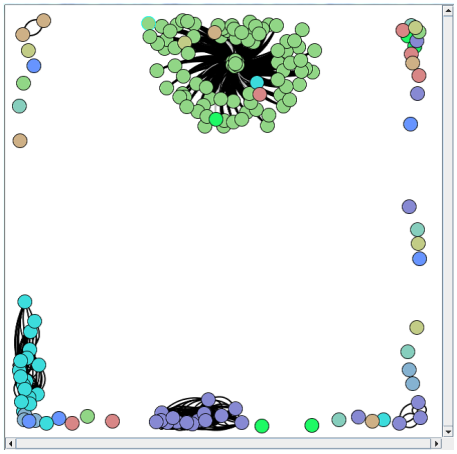

図 15　yoda's Crypter でパックされた 168 検体の可視化結果

ことができた．これらのコードブロックを用いることで多段パックの影響を受けない類似度判定を行うことができた．また，可視化システムによるクラスタリング結果より，アンチウイルスソフトによって定義されるマルウェア科名との一定の整合性を確認することができた．

謝辞

(独) 情報通信研究機構ネットワークセキュリティ研究所サイバーセキュリティ研究室各位のご助言，ご協力に感謝する．

参考文献

[1] RL!Depacker, http://www.reversinglabs.com/forum/index.php?topic=11.0
[2] tElock 0.98, http://www.telock.com-about.com/
[3] PESpin v1.33, http://pespin.w.interia.pl/
[4] Yoda's Crypter , http://www.yodas-crypter.com-about.com/
[5] 岩本一樹, 和田克己, "コンピュータウイルスのコード静的解析による特徴抽出と分類について"
[6] Marius Gheorghescu, "AN AUTOMATED VIRUS CLASSIFICATION SYSTEM"
[7] 織井達憲，吉岡克成，四方順司，松本 勉，金 亨燦，井上大介，中尾康二， "パッキングされたマルウェアの類似度算出手法とその評価， " 信学技報, ICSS, 109(285), pp. 7-12, 2009.
[8] H. C. Kim, D. Inoue, M. Eto, Y. Takagi, and K. Nakao, "Toward Generic Unpacking Techniques for Malware Analysis with Quantification of Code Revelation," JWIS 2009, 2009.
[9] 岩村誠, 伊藤光恭, 村岡洋一, "コンパイラ出力コードの尤度に基づくアンパッキング手法" MWS2008, pp.103 108, 2008.
[10] 川古谷裕平, 岩村誠, 針生剛男, "実行命令トレースに基づく動的パッカー特定手法" Computer Security Symposium 2011
[11] Piotr Bania "Generic Unpacking of Self-modifying, Aggressive, Packed Binary Programs"
[12] PETools,http://uinc.ru/files/neox/PE_Tools.shtml
[13] OllyDbg, http://www.ollydbg.de/
[14] OllyDump, http://www.openrce.org/downloads/details/108/OllyDump
[15] THOMAS M. J. FRUCHTERMAN and EDWARD M. REINGOLD, Graph drawing by forcedirected placement. Software Practice and Experience, vol. 21, pp. 1129-1164, November 1991.
[16] Newman, M. and Girvan, G., "Finding and Evaluating Community Structure in Networks," In Physical Revew,E, 2004.
[17] JUNG Java Universal Network/Graph Framework, http://jung.sourceforge.net/
[18] TrendMicro, http://jp.trendmicro.com/jp/home/
[19] Symantec, http://www.symantec.com/index.jsp

表 1　類似度可視化に用いた 22 検体と多段パッカー

TrendMicro または Symantec による科名	検体数
TROJ_PAKES.HJ	76
TROJ_DROPPER.CVN	4
TROJ_AGENT.TPA	3
TROJ_DELF.FUS	3
TROJ_DIAL.EL	1
WORM_ALLAPLE.IK	11
WORM_VANBOT.VS	6
WORM_MAINBOT.XO	2
WORM_RBOT.GD	2
BKDR_VANBOT.PM	6
PE_VIRUT.XO-4	1
PE_VIRUX.A-3	1
W32.IRCbot	2
W32.Virut.A	1
W32.Virut.B	3
W32.Rahack.W	2
w32virut.f	1
Unknown	45