

マルウェアの自己書き換え動作をメモリアクセスに着目して可視化する方法

塩谷 正治† 森 博志† 吉岡 克成† 松本 勉†

†横浜国立大学

240-8501 神奈川県横浜市保土ヶ谷区常盤台 79-7

{shiotani-masaharu-rg, mori-hiroshi-pg, yoshioka, tsutomu}@ynu.ac.jp

あらまし 近年のマルウェアはプログラム本体のコードを書き換えるパッカーソフトによって難読化(パッキング)されていることが多い。マルウェアの動作をコード解析により把握するには難読化の解除(アンパック)が不可欠だが、多段でパッキングされたマルウェアや、複雑な復号処理を行うマルウェアに対して難読化の解除を完全に自動化することは困難といえる。そこで、本研究ではマルウェアによる命令実行の様子と、命令により読み書きされるメモリ上のデータの関係を可視化し、動画として実行順に連続表示することで、マルウェア解析者を支援する方法を提案する。また、パッカーを適用したサンプルプログラムや実際のマルウェアの自己書き換え動作の可視化により、自己書き換え動作の様子が分かりやすく表現される例を示す。

Memory-Access-Based Visualization of Self-Modifying Malware Behavior

Masaharu Shiotani† Hiroshi Mori† Katsunari Yoshioka† Tsutomu Matsumoto†

†Yokohama National University

79-7 Tokiwadai, Hodogaya-ku, Yokohama, 240-8501, JAPAN

{shiotani-masaharu-rg, mori-hiroshi-pg, yoshioka, tsutomu}@ynu.ac.jp

Abstract Most of modern malware are obfuscated by a packer. A packer encrypts a malware code while it is transmitted as an executable file and adds a decoder to decrypt the code upon its execution. It is essential to unpack the malware executable to read and analyze its code. However, full automation of an unpacking procedure is difficult with variety of packers especially with a multi-layered decoding and a highly complex decoding. In this paper, we propose a method to support human analyst to understand malware's self-modifying behavior by visualizing execution of instructions and their memory accesses as a movie. We show some examples in which characteristic self-modifying behavior of sample program and in-the-wild malware are visualized.

1 はじめに

近年、情報漏えいやサービス妨害など悪意のある活動を行うマルウェアと呼ばれるソフトウェアが数多く流通している。マルウェアに対して有効な対策を行うにはマルウェアの挙動やコードを解析する必要がある。マルウェアの解析方法は解析対象となる検体のコードを解析して内容を把握する静的解析とマルウェアを実行し、挙動を観察する動的解析に大別される。動的解析はマルウェアの実行した命令しか

観測できないため、マルウェアの動作を完全に把握するには静的解析によってコードを解析する必要がある。しかしながら、マルウェアはパッカーと呼ばれるソフトウェアによって、自身のコードを暗号化・圧縮し、実行時に復号しながら動作することが多い。パッキングされている場合には、マルウェア本来のコード(オリジナルコードと呼ぶ)を抽出(アンパック)しつつ解析を行う必要がある。アンパックはパッカー毎にカスタマイズされたアンパッカーを用いるか、

OllyDbg[4]や IDA Pro[5]等のデバッカ、逆アセンブラを用いて手動で行う。また、復号を行うコードとオリジナルコードの差異に注目してオリジナルコードを抽出する方法[2]やマルウェア実行時のメモリダンプに着目してオリジナルコードを抽出する方法[3]などアンパッキングの自動化が検討されている。しかし、多段で復号を行う手法や、分割デコーダにより復号を行う手法など、様々なパッキング手法があり、全てのパッキング手法に対応した完全な自動化は困難であるといえる。そこで、命令実行と書き込みデータの関係性を可視化することによって解析者のアンパッキング作業を支援する方法[1]が提案されている。文献[1]による可視化手法ではメモリ上に書き込まれるデータを階層的に表現することによって、自己書き換えが多段で実行されている様子の把握やオリジナルコードの推測を容易にしている。しかしながら、自己書き換えの様子を1枚の静止画で表現するため、複数箇所が書き換えられている場合ではその順序が把握しにくいことや、書き換え命令部(デコーダ部)の入出力関係が把握しにくいという課題があった。

そこで、本研究ではマルウェアによる命令実行の様子と命令が読み書きを行うデータの実行ステップ毎に可視化し、これを連続的に描画することで自己書き換え動作を動画として表現し、アンパッキング処理の概要把握や、デコーダ部の判定を支援する手法を提案する。また、パッキングを行ったサンプルプログラムと実際のマルウェア検体を用いて提案手法による可視化を行い、自己書き換え動作の様子と自己書き換え動作後にオリジナルコードが実行されている様子が分かりやすく表現される例を示す。本論文の構成は以下となる。まず、第2章では関連研究を説明し、第3章では提案手法の説明と自己書き換え動作可視化システムについて、第4章では12種類のパッカーを用いてパッキングを行ったサンプルプログラム、実マルウェア検体を用いて可視化を行った評価実験を示し、第5章では考察を述べ、第6章をまとめと今後の課題とする。

2 関連研究

マルウェアの難読化に用いられるパッカーはフリーソフトとして広く頒布されているものや、ソフトウェアの保護を目的として販売されるものなど既知のパッカーに対しては専用のアンパッカーを用いてアン

パッキングを行うことが出来る。しかし、未知のパッカー、特定の困難なパッカーに対しては専用のアンパッカーを用いてアンパッキングを行うことは難しい。そこで、専用のアンパッカーではなく汎用的なアンパッキングを行う手法や、自己書き換え動作を可視化することで解析者のアンパッキング作業を支援する方法が提案されている。

汎用的なアンパッカーとしては OllyBonE[6]、Renovo[7]、Saffron[8]、手法としては文献[2,3]等がある。OllyBonE や文献[3]ではデコーダによる復号が終わった後にオリジナルコードの展開が行われることが予想されるメモリ領域を出力することによってアンパッキングを行っている。OllyBonE ではオリジナルコード展開が予想されるメモリ領域にブレークポイントを設定し、メモリ領域内の命令が実行されるとメモリダンプを行いオリジナルコードとして抽出している。文献[3]ではマルウェアがDLLをメモリ上にロード、アンロードする際にメモリダンプを行い、PEヘッダの書き換えによる逆アセンブルなどの解析を可能とするシステムを提案している。Renovo、Saffronではメモリアクセスを監視することにより、データが書き込まれた後に実行されたコードをオリジナルコードとし抽出している。Renovo、Saffronでは多段にパッキングされていた場合にデコード中のコードをオリジナルコードとして判定することがある。そこで文献[2]ではオリジナルコードとデコーダの機械語命令を比較し、コンパイラが出力したコードとして尤度の高いコードをオリジナルコードとして抽出している。これらの手法は多くのパッカーに対して有効だが、オリジナルコードの抽出を回避する自己書き換え動作を組み込むことは理論的には可能のため、アンパッキングの完全な自動化は困難となっている。

一方、文献[1]では、アンパッキングの自動化ではなく、解析者が手動で行うアンパッキング等の作業を支援する可視化手法を提案している。まず、マルウェアが実行開始時にメモリに展開するデータを階層0とし、階層0上の命令によって書き込まれたデータを階層1とする。同様に階層 $n-1$ 上の命令によって書き込まれたデータを階層 n 上に描画することでデータの書き込みを階層的に表現している。これにより命令自己書き換え動作が多段に行われている様子の把握、オリジナルコードの抽出がしやすくなっている。しか

し、複数箇所が書き込まれている場合に書き込み順序が把握できない、デコーダ部によるメモリアクセスの様子が把握しにくいという課題がある。

3 提案手法

本章では、マルウェアによる命令実行の様子と各命令によるメモリ上のデータの読み書きを可視化し、動画として実行順に連続表示することにより、解析者によるアンパック処理の概要把握、デコーダ部の判定を支援する可視化手法を提案する。

3.1 メモリ上のデータ読み書きを可視化する 3層表現

提案手法は、マルウェアが実行した命令と、それらの命令により読み書きされたデータの位置を描画する。そのため、メモリ空間を表現する2次元平面を3枚用意し、読み込まれたデータの位置、実行された命令の位置、書き込まれたデータの位置を、それぞれの平面上に描画する。この3平面を、読み込み層、実行命令層、書き込み層と呼ぶ。これら3層の位置関係を図1に示す。

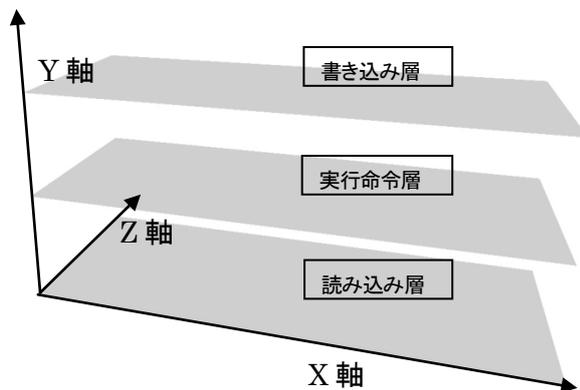


図1 各層の位置関係

マルウェアが実行する命令のメモリ領域は広大であり、ステップ数も多いため、単純に全ての命令を可視化することは困難である。そこでメモリ空間の省略と軸の取り方を工夫することにより広大なメモリ空間の表示を行う。具体的には、図2の様に16進数8桁で表現されるアドレス空間のうち、下3桁をX軸、上5桁をZ軸で表現することで、アドレスの微小な変化もX軸上で反映されるようにする。また、Z軸上では未使用のアドレス空間は省略している。

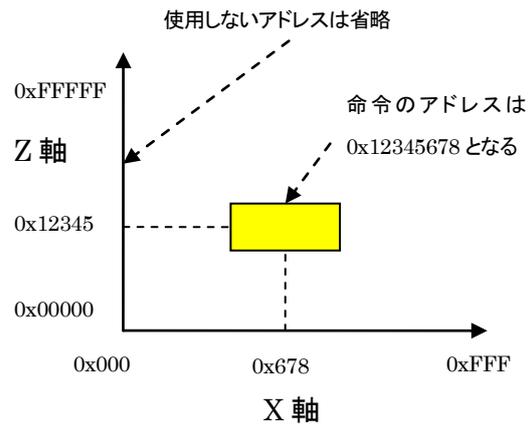


図2 メモリ空間の表示方法

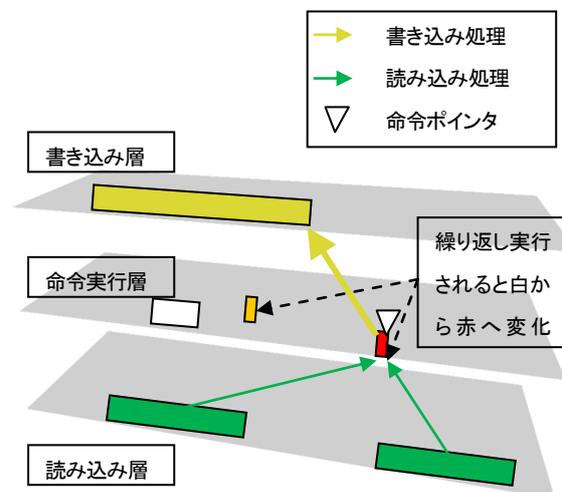


図3 命令実行と読み書きデータの可視化例

提案手法では、マルウェアの実行した命令のアドレスとサイズ、命令が読み書きをしたデータのアドレスとサイズを基にして図3の様に命令実行と命令によって読み書きされるデータの関係を可視化する。まず、命令が実行されると実行命令層に命令サイズ分の白色ブロックと、実行されている場所を示す命令ポインタを表示する。この命令が書き込み命令であった場合には書き込みデータサイズ分の黄色ブロックを書き込み層に表示し、実行命令層から書き込み層のブロックに矢印を描画することでデータの書き込みを表現する。読み込み命令の場合には読み込みデータサイズ分の緑色ブロックを読み込み層に表示し、当該ブロックから実行命令層に矢印を描画する。同じ命令が繰り返し実行された場合には、ブロッ

クの色を変化させることで繰り返し実行された命令や繰り返し読み書きされたアドレスを表現する。例えば、マルウェアがオリジナルコードなどを実行時に復号する場合は、デコーダ部分は通常ループ構造になっており、繰り返し実行されるため、上記の色付けにより、デコーダ部分が強調表示されることになる。また、実行時に書き込まれたデータが後に命令として実行された際は、命令ポインタを赤く表示することで、オリジナルコード候補や、暗号化されていたデコードの候補として表現を行う。

3.2 動画化

3.1 節で説明した方法により命令実行によるデータの読み書きの様子を1枚の静止画で表現できるが、マルウェアが実行する命令の数は膨大であるため、全ての命令を同一画像として描画すると処理の流れやデータの読み書きの対応関係が把握しにくくなってしまふ。そこで、実行順にデータの読み書きの様子を連続的に描画し、古い命令については順次消去する動画化を行うことで、処理の流れを把握しやすくする。

本稿では、2種類の動画化法として領域区分法とサンプリング法を提案する。領域区分法では、メモリ空間を単位領域に区分し、同一領域で命令が実行される間は過去のデータの読み書きの可視化結果の消去を行わない手法である。この手法では、各単位領域におけるデータの読み書きの様子を詳細に捉える事ができる。実際に、多くのパッカーにおいて、デコーダは比較的小さなメモリ領域内で動作するため、領域区分法によって復号の様子が把握できる。一方、サンプリング法は、一定の割合でしかデータ読み書きを可視化しない代わりに、過去の読み書きの描画結果を消去しない方法である。サンプリング法では、特定箇所の詳細な読み書き動作については把握が難しくなるが、読み書き動作全体の概略や分割デコーダの様子を把握しやすい。

3.3 ユーザーインターフェイス

上記のようなデータ読み書きの可視化および動画化に加えて、可視化結果においてメモリ領域を指定することで当該領域のバイナリコードをファイルとして取得や、可視化に用いるブロックの色合いやサンプリング法におけるサンプリング率を設定するためのユーザーインターフェイスが必要である。

4 可視化手法の実装と評価実験

提案手法を実装した、自己書き換え動作可視化システムについて説明する。当該システムにおける実行トレース取得は文献[1]の可視化システムを再利用した。概要を図4にて示す。システムは3つのモジュールからなっており、メモリアクセス監視モジュールでは CentOS 上に VMware(ゲスト OS Windows XP Professional SP1)を解析環境として用意し、解析環境上においてマルウェア検体を実行し、PIN[9]を用いて解析することによって実行トレースを取得する。マルウェアの実行が終わらない場合には設定した時間でタイムアウトし、タイムアウト時のログを取得する。DB作成モジュールでは、大量のログを整理、解析を行い、可視化モジュール内でデータ処理を行う処理が少なくなるようにDBを作成する。可視化モジュールではDB作成モジュールによって作成されたDBからデータを読み込んで可視化を行う。可視化モジュールは VPython を用いて実装を行い、解析者は設定を設定ファイルまたはインターフェイス上で変更できるようにした。

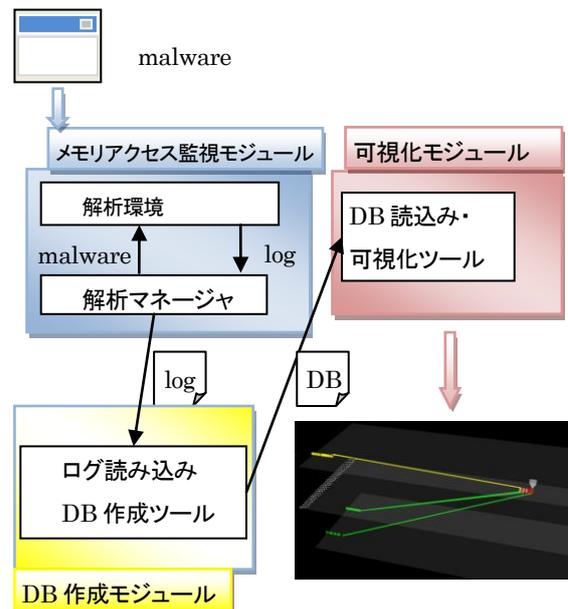


図4 自己書き換え動作可視化システム

評価実験

以降では自己書き換え動作可視化システムを用いて行った実験を説明する。実験は、12種類のパッカーをサンプルプログラム(コンソール上に Hello, World と出力するプログラム)に適用して得た検体と

実マルウェア検体 5 種を用いた。以下に実験で用いた検体の情報を示す。

表 1 実験 1 で用いたパッカー

パッカー名	パッカーのバージョン
ASPack	2.2
ASProtect	1.5
FSG	2.0
MEW	SE v1.1
Packman	1.0
PECompact	3.03.18 beta
PEDiminisher	0.1
PESpin	1.33
Telock	t0.98
UPX	3.03w
WinUpack	0.39final
WWPack	1.20.3.236 demo

表 2 実験 2 で用いたマルウェア検体

検体名	マルウェア名(Symantec[10])/ MD5 ハッシュ値
malware1.exe	WS.Malware.1/ fd9e6951781811b0c284a8db354b21a2
malware2.exe	WS.Malware.1/ d28a3d1c291aa7a5bfde08eb86c50e5b
malware3.exe	WS.Malware1/ 8e8aaff0418a51a8c91f02d61c50b306
malware4.exe	Trojan.Pandex/ 58153596d7e2a2dbd479776082f06967
malware5.exe	W32.SillyP2P/ 36dd47e81442ace3934537b35f5fe349

実験内容

実験 1 コンソール上に“Hello World.”と出力するプログラムを Microsoft Visual Studio 2010 Premium を用いて作成し、表 1 に示した 12 種類のパッカーを用いてパッキングして 12 種類の検体を取得した。これらの検体に対して提案手法により自己書き換え動作の可視化を試みた。

実験 2 Hispasec 社より提供を受けたマルウェア検体 5 体に対して提案手法により自己書き換え動作の可視化を試みた。

実験結果

実験 1 の結果として、図 5 に UPX と MEW によりパッキングされた Hello, world プログラムの自己書き換え動作の可視化結果を示す。なお、UPX については領域区分法、MEW についてはサンプリング法で動画化している。図 5 では、「パッカー名:数字」という形式でパッカー名と実行順を示す。白枠で囲まれている部分がデコーダ部であり、UPX は 1 つのデコーダ部、MEW では 2 つに分割されたデコーダによって

復号されたデコーダによって多段に復号処理を行っている。実験 2 については malware2, malware4 を可視化した例を図 6 にて示す。動画化には領域区分法を用いた。「検体名:数字」という形式で検体名と実行順を示す。malware2 では多段に復号される様子を、malware4 では復号後にオリジナルコード候補が実行される様子を示す。また上記を含む全ての検体に関する可視化結果の動画ファイルは文献[11]の URL にて公開されている。

5 考察

まず、実験 1 で用いたパッカーについて、特徴をまとめ、デコーダ抽出の判定、自己書き換え動作の概要把握のしやすさについて考察する。

UPXタイプ (ASPack, FSG, UPX, PEDiminisher, WinUpack, WWPack) プログラム実行直後に自己書き換え動作を行う。また、デコーダが連続したアドレスまたは互いに非常に近い範囲内にあり、復号は 1 回しか行わないという特徴がある。比較的単純なパッカーであり、デコーダ抽出、自己書き換え動作の概要把握はしやすい。

MEWタイプ (MEW) デコーダがメモリ上の離れたアドレスに分割配置されている。さらに多段の自己書き換え動作を行う。デコーダが異なる領域に分かれて可視化されるため、領域区分法では、デコーダの抽出は困難である。但し、多段の自己書き換え動作がプログラム実行直後に連続して行われるため、サンプリング法により概要把握はしやすい。

PECompactタイプ (PECompact) MEWタイプと同様に多段の自己書き換えを行うがデコーダ分割は無いタイプである。実行開始直後に多段の自己書き換えを連続して行うため、デコーダ抽出、自己書き換え動作の把握はしやすい。

ASProtectタイプ (ASProtect, Packman) 実行開始直後に大量のデータを書き込むループ処理など様々な命令が実行された後に自己書き換え動作を行う。大量のデータが書き込まれるが命令に使用されないことが多い。多くのループ処理や、多段の自己書き換えがあり、デコーダの抽出、自己書き換え動作の把握が困難なタイプである。実行ステップ数も多く、可視化自体が難しいタイプである。

telockタイプ (telock) ループ処理を繰り返すのみでオリジナルコードが実行されなかった。パッカーに

よる解析環境の検知，解析環境上での互換性の問題等が考えられる。

PESpin タイプ (PESpin) 解析環境で実行は出来たが，有効なログを取得することが出来なかった。

実験 2 で用いた実マルウェアは全て自己書き換え動作を示した。実験 1 で用いたタイプに当てはめて分類を行った結果を示す。

UPX タイプ (malware4) UPX タイプと同様にマルウェア実行開始直後に自己書換え動作が観測された。自己書換え後に書き換えられた命令(オリジナルコードと思われるコード)が実行されたことから，1 回の復号のみを行う単純なパッカーが使用されていたと考えられる。

PECompact タイプ (malware1, malware2, malware3) 多段の自己書き換えを連続して実行されていることから，PECompact タイプと同様の仕組みのパッカーによりパックされたと考えられるが，実行開始直後にループ処理がいくつか入り，PECompact タイプとは異なるパッカーを用いていると考えられる。

ASProtect タイプ (malware5) 実行開始直後に大量のデータを書き込み，その後，自己書き換え動作と考えられる挙動を示したが，ASProtect タイプと同様に多段に自己書き換えを行っており，複雑な復号処理が行われているようであった。

6 まとめと今後の課題

本稿では，自己書き換え動作をメモリアクセスに基づいて可視化を行うことにより，解析者によるデコーダの抽出，自己書き換え動作の概要把握を支援する手法の提案を行った。さらに提案手法を実装し，サンプルプログラム，実マルウェア検体を用いてデコーダ抽出，自己書き換え動作の可視化を行った。今後の課題として以下の案を検討している。

文献[1]にて提案された階層の概念の導入 多段で自己書き換えが行われる場合，書き込まれたデータが後にデコーダとして実行されるなど，書き込まれたデータとその後実行されるデータには強い関係があるが，現在の可視化手法では，書き込み層と実行命令層が分離しており，この関係が把握しにくい。そこで，今回の提案手法のような固定の 3 層ではなく，文献[1]で提案された多階層の概念を取り入れることで書き込まれたデータと，実行される命令の関係が把握しやすくなると期待される。

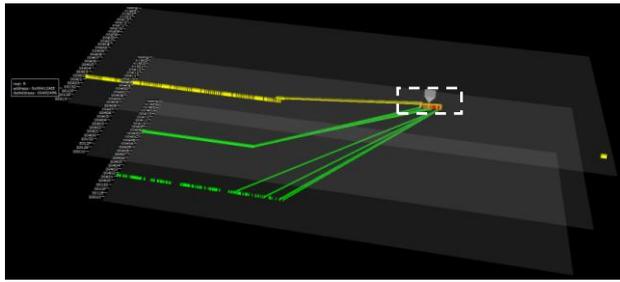
領域区分法の改良 領域区分法の現在の実装ではある領域から別の領域に実行が移動した際には，過去の可視化結果を消去しているが，条件設定などにより，解析者のニーズに応じて可視化結果を適宜残す機能を設けることで，より細やかな解析ができることが期待される。

謝辞

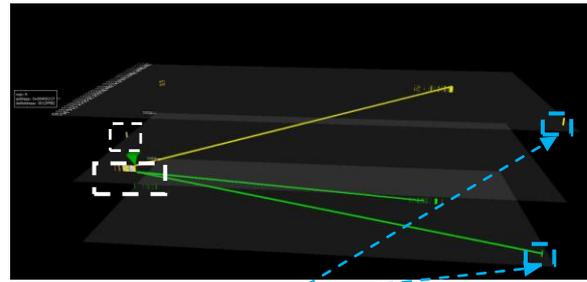
本研究の一部は，総務省情報通信分野における研究開発委託／国際連携によるサイバー攻撃の予知技術の研究開発／サイバー攻撃情報とマルウェア実体の突合分析技術／類似判定に関する研究開発により行われた。

参考文献

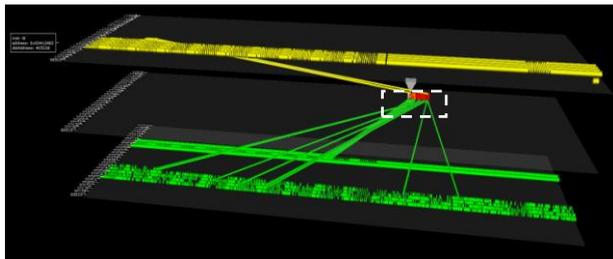
- [1] 織井達憲，吉岡克成，四方順司，松本勉，“マルウェア解析の効率化を目指した自己書き換え動作の可視化方法”，暗号と情報セキュリティシンポジウム (SCIS2011)，3B3-5，2011.
- [2] 岩村誠，伊藤光恭，村岡洋一，“コンパイラ出力コードの尤度に基づくアンパッキング手法”，MWS2008，pp.103-108，2008.
- [3] 岡田隼人，伊沢亮一，森井昌克，中尾康二，“ウイルスコード自動解析システムの開発”，暗号と情報セキュリティシンポジウム (SCIS2007)，pp.1-6，2007
- [4] “OllyDbg”，<http://www.ollydbg.de/>
- [5] “IDA Pro”，<http://www.hex-rays.com/products/ida/>
- [6] “OllyBonE”，<http://www.joestewart.org/ollybone/>
- [7] D. Quist, V. Smith, “Covert Debugging Circumventing Software Armoring,” Blackhat USA 2007 / Defcon 15, Las Vegas, NV.
- [8] M. G. Kang, P. Poosankam, and H. Yin, “Renovo: a hidden code extractor for packed executables,” In Proc. WORM '07: Proceedings of the 2007 ACM workshop on Recurring malware, pp. 46-53, 2007.
- [9] “Pin - A Dynamic Binary Instrumentation Tool,” <http://www.pintool.org/>.
- [10] “Symantec”,<http://www.symantec.com/>
- [11] 情報物理セキュリティ研究拠点: マルウェア等のプログラムの自己書き換え動作の可視化 <http://ipsr.ynu.ac.jp/self-modifying-malware.html>



UPX:1 デコーダが復号をはじめた様子

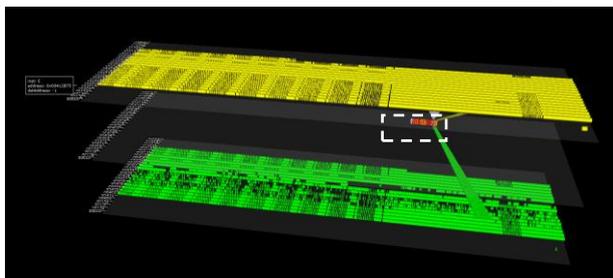


分割されたデコーダによって直前に書き込まれたデータが、読み込まれている様子



UPX:2 デコーダがデータを展開している様子

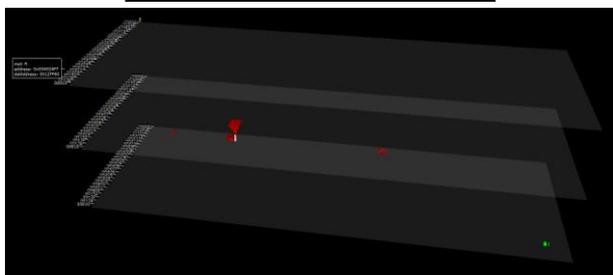
MEW:2 分割されたデコーダによって書き込まれたデータを読み込んで、データを復号している様子



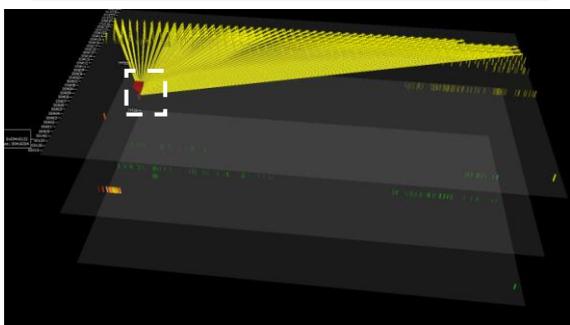
UPX:3 デコーダが復号を終えた様子



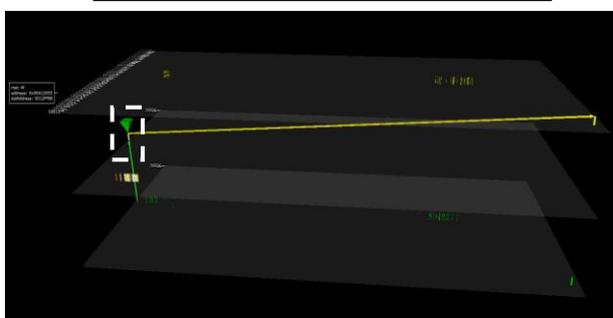
MEW:3 1つ目のデコーダによる復号が終わった様子



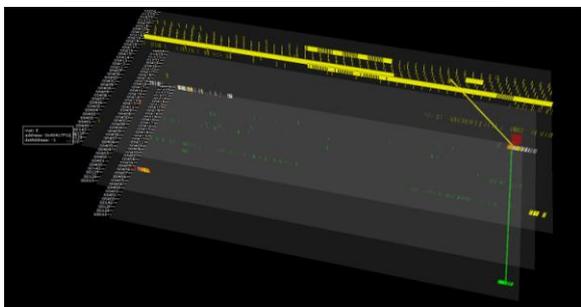
UPX:4 オリジナルコードが実行されている様子



MEW:4 暗号化されていたデコーダによる復号が行われている様子

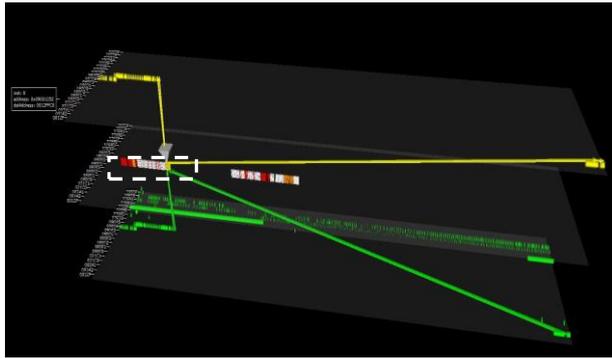


MEW:1 分割されたデコーダがデータを書き込む様子

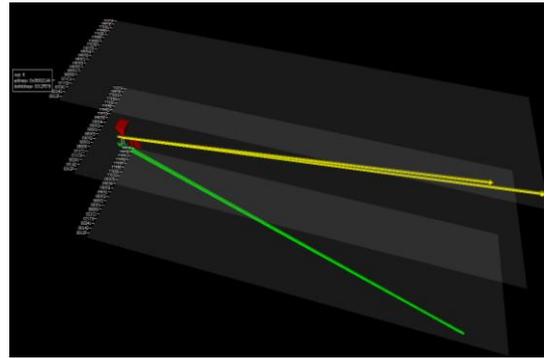


MEW:5 オリジナルコードが実行されている様子

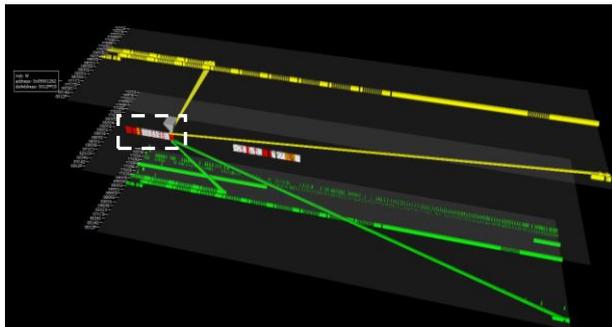
図 5 UPX, MEW を用いたサンプルプログラムの可視化例



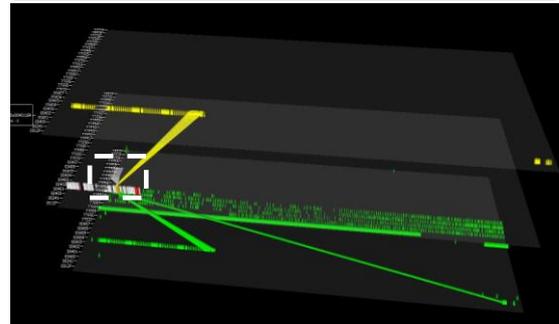
malware2:1 最初のデコーダが復号をはじめた様子



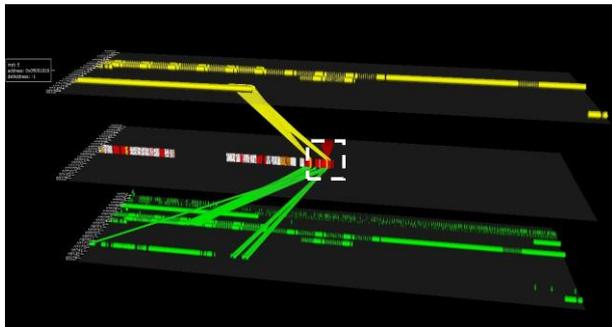
malware2:4 復号が終了した後に書き換えられた命令が実行されている様子



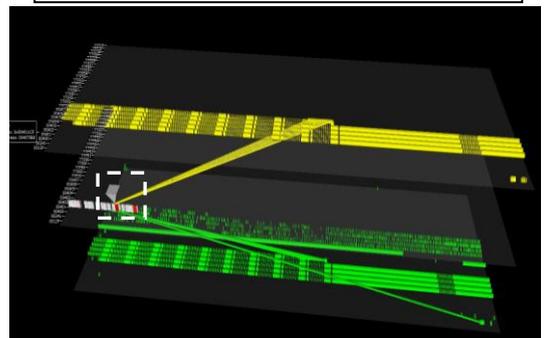
malware2:2 最初のデコーダが復号を終えた様子



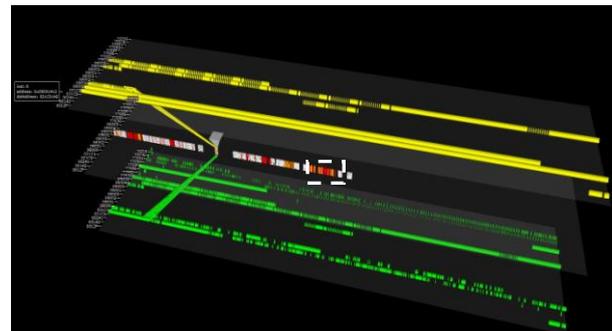
malware4:1 デコーダが復号をはじめた様子



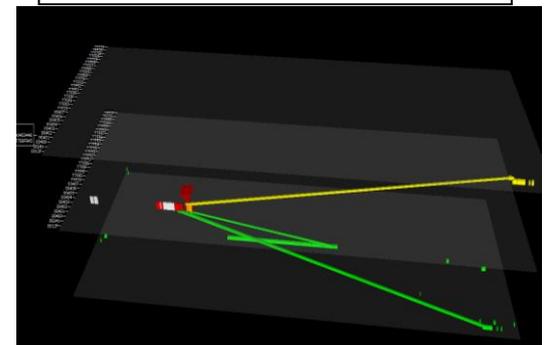
Malware2:3 暗号化されていたデコーダが復号をはじめた様子



malware4:2 デコーダが復号を終えた様子



malware2:4 暗号化されていたデコーダが復号を終え、次の命令へジャンプした様子



malware4:3 復号が終了した後に書き換えられた命令が実行されている様子

図6 マルウェア検体の自己書き換え動作の可視化例