情報処理学会論文誌 Vol.54 No.8 2079–2089 (Aug. 2013)

推薦論文

テイント伝搬に基づく解析対象コードの追跡方法

川古谷 裕平[1,a)] 岩村 誠[1,b)] 針生 剛男[1,c)]

受付日 2012年12月18日, 採録日 2013年5月18日

概要：マルウェアの動的解析を行う際，プロセス ID やスレッド ID などの識別子を使って解析対象コードとそれ以外のコードとを区別する場合が多い．しかし，これら識別子に基づく方法では，マルウェアの解析妨害機能により正確に区別ができない状況が生まれている．この問題を解決するため，本論文ではテイントタグを用いた解析対象コードの識別方法を提案する．提案手法の有効性を示すため，マルウェアの動作を模倣した各種テストコードと CCC Dataset 2012 を用いて実験を行った．この実験の結果，提案手法が様々な解析妨害機能に有効であり，実際のマルウェアにも適用可能であることを示した．本提案手法を利用することで，既存の各種マルウェア解析環境やマルウェア対策技術の精度を向上させることが可能になる．

キーワード：マルウェア，テイント解析，解析妨害，動的解析

Tracing Malicious Code with Taint Propagation

Yuhei Kawakoya[1,a)] Makoto Iwamura[1,b)] Takeo Hariu[1,c)]

Received: December 18, 2012, Accepted: May 18, 2013

Abstract: Dynamic malware analysis environments commonly distinguish their target code from benign code based on its process ID or thread ID. However, the distinction based on these IDs does not correctly handle malware which has anti-analysis functions. To solve this problem, we propose an approach for identifying the to-be-analyzed code based on taint tags. To prove the effectiveness of our proposal, we have conducted experiments with a set of test code which behaves like malware and also with CCC Dataset 2012. The results of these experiments indicated that our approach is effective against various anti-analysis functions, and that it is applicable to real-world malware. Our proposal will allow existing malware analysis environments and anti-malware research to be more precise and effective.

Keywords: malware, taint analysis, anti-analysis, dynamic analysis

1. はじめに

マルウェアの動的解析をする場合，解析対象コードと，それ以外のコードとを正確に識別する必要がある．この識別にはプロセス ID（以下，PID）やスレッド ID（以下，TID）といった OS のセマンティックスを用いるのが一般的である．

しかしながら，近年のマルウェアは解析環境の監視から逃れるため，自身のコードの一部を他のプロセスに注入するコード注入や，自身のコードの一部を他の実行ファイルに付け加えるファイル感染といった解析妨害手法を利用する [22]．これにより，PID や TID を基にした識別では，検知漏れや誤検知が発生してしまう．このような問題が発生する原因として，PID や TID に基づく識別方法には以下の問題があると考える．

(1) PID や TID では識別の粒度が粗い．

(2) コード注入方法に依存した追跡しかできない．

(1) に関して，同一プロセス，スレッド内に悪意のあるコードと正規のコードが混在する場合，両者のコードを正

[1] NTT セキュアプラットフォーム研究所
Musashino, Tokyo 180–8585, Japan
[a)] kawakoya.yuhei@lab.ntt.co.jp
[b)] iwamura.makoto@lab.ntt.co.jp
[c)] hariu.takeo@lab.ntt.co.jp

本論文の内容は 2012 年 10 月のコンピュータセキュリティシンポジウム 2012 にて報告され，マルウェア対策研究人材育成ワークショップ 2012 プログラム委員長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．

確に識別することができない．そのため，正規のコードによる挙動を，誤って解析対象としてしまう可能性がある．一方，(2) に関して，通常 PID，TID で解析対象を指定している場合，他のプロセスにコード注入されると，そのコード注入先プロセスの PID，TID を把握し，それらも解析対象として設定する必要がある．この際，既知の方法によるコード注入に関しては，あらかじめポイントとなる API などを監視しておくことで，注入の動作を検知し，注入先のプロセスやスレッドを追跡できる．しかし，未知の方法でコード注入が行われた場合，その注入の動作を見逃してしまい，注入先のプロセスを解析対象にすることができない．結果，その注入先のプロセス内で実行される悪意のあるコードの挙動を見逃してしまう．

そこで本論文では，これら 2 つの問題を解決するテイントタグに基づく解析対象コードの識別方法を提案する．テイントタグとは，動的なデータフロー解析手法の 1 つであるテイント解析で利用され，メモリ上の値に対して設定される属性情報である．本提案手法は，コンピュータ全体をエミュレートする仮想マシンモニタ（以下，VMM）を利用し，その VMM 上のゲスト OS 内で解析対象のマルウェアを実行し，挙動を監視する動的解析に適用することを想定している．解析を開始する前に解析対象の実行ファイルに対して解析対象を示すテイントタグを設定する．VMM 内の仮想 CPU で命令フェッチした際に，その命令に解析対象を示すテイントタグが設定されているか否かで解析対象コードを識別する．これにより，命令単位で解析対象コードを識別することができる．また，解析対象コードがゲスト OS 内で移動された場合，その移動を，そのコードに設定したテイントタグを伝搬させることで追跡することができる．

さらに，通常のテイントタグの伝搬に加えて，マルウェアによるファイル書き出しや動的生成コードなどを考慮し，ファイルへのテイントタグ伝搬，強制的なテイントタグ伝搬の 2 つの機能を実装する．ファイルへのテイントタグ伝搬機能は，テイントタグが設定されたメモリ上の値がファイルに書き出された場合，そのテイントタグをディスク上のファイルへ伝搬させる仕組みである．これにより，マルウェアが自身の一部をファイルとして書き出し，他のプロセスへそのファイルを注入するといった行為を行った場合，書き出されたファイル，そのファイルが注入されたメモリへとテイントタグを伝搬させることで，注入されたプロセス内での解析対象コードの実行をとらえることができる．また，強制的なテイントタグ伝搬機能は，解析対象コードが行ったすべてのメモリ書き込み操作に対して，その書き込み先の値に対して解析対象を示すテイントタグを設定するものである．これにより，マルウェアがテイントの伝搬が途切れるような処理を通して動的にコードを生成した場合でも，そのコードに対してテイントタグを設定し，解析対象とすることができる．

上記の提案手法を，テイントタグ伝搬機能付きのハードウェアエミュレータである Argos [16] 上に実装した．この Argos に対して，仮想 CPU 上のテイントタグ伝搬機構を改造し，強制的なテイントタグ伝搬機構の追加を行った．また，ディスク上のファイルに対してもテイントタグを保持できるよう，ファイルに関するテイントタグ保持用のデータ構造体，シャドウディスクを実装した．さらに，仮想 DMA（Direct Memory Access）コントローラを改造し，シャドウメモリとシャドウディスク間でテイントタグの転送を行える機能を追加した．

提案手法の有効性を示すため，上記の提案手法を実装したシステム（以下，提案システム）上で，一般的なパッカでパッキングした実行ファイル，マルウェアの挙動を模倣する複数のテストコード，CCC Dataset 2012 のマルウェア [12]，を用いて実験を行った．この実験の結果，提案手法が従来の PID や TID に基づく解析対象コード識別方法の問題点を解決でき，実際のマルウェアに対しても適用可能であることを示した．

本提案手法を利用することで，マルウェアの動的解析の際の挙動の見逃しや誤った挙動の取得を減少させることができる．これにより，様々なマルウェア対策技術の正確性を向上させることができる．

2. 問題定義

本章では，本論文が対象としている既存の動的解析システムの解析対象コード識別方法の問題について明らかにする．

本論文では，既存の動的解析システムの PID や TID による解析対象コードの識別方法に関して，以下の 2 つの問題があると考える．

(1) PID や TID では識別の粒度が粗い．
(2) コード注入方法に依存した追跡しかできない．

(1) に関して，同一プロセス，スレッド内に悪意のあるコードと正規のコードが混在する状況において，両者を正確に区別することができない．たとえば，ファイル感染型マルウェアが，感染先の実行ファイルに自身のコードを追加し，その実行ファイルのエントリポイントを追加したコードを指すように改ざんした場合を考える．この場合，このファイルが実行されると，同じ PID，TID で，まず悪意のあるコードが実行され，続いて，正規のコードが実行される．PID，TID に基づく解析対象コードの識別方法では，このような同一プロセス，スレッド上で行われる挙動の場合，どこまで悪意のあるコードでどこからが正規のコードかの区別ができない．

(2) に関して，通常 PID，TID で解析対象を指定している場合，他のプロセスにコード注入されると，そのコード注入先プロセスの PID，TID を把握し，それらも解析対象と

して設定する必要がある．この際，CreateRemoteThread のような多くのマルウェアに利用されるコード注入方法，いわゆる既知の手法の場合，この API をあらかじめフックし，監視しておくことで，コード注入先の PID，TID を取得することができる．しかし，あらかじめ把握していない方法でコード注入が行われた場合，このような追跡をすることができない．特に Windows のようなクローズドソースの環境では，あらかじめすべてのコード注入が行われる方法を把握することは現実的には難しい．たとえば，特定のレジストリにファイルを登録し，そのレジストリを参照したプロセスに，そこに登録されているファイルがロードされるような場合が Windows では多数存在している．このようなレジストリをあらかじめすべて列挙するのは現実的には困難である．

3. 提案手法

本章では，2 章で述べた従来手法の問題を解決するため，テイントタグに基づく解析対象コードの識別，追跡方法を提案する．ここでは，まずテイント解析について説明し，続いて提案手法の基本概念を説明する．次に例を用いて動作シーケンスを説明する．さらに，マルウェア解析に適用する際の問題点とその解決方法について述べる．

3.1 テイント解析

テイント解析とは，動的なデータフロー解析手法の 1 つである．テイントタグと呼ばれる属性情報を解析対象のメモリ上の値に設定し，CPU が演算命令やデータ遷移命令を実行する際に参照元のメモリの値にテイントタグが付与されていた場合は，その演算結果や値の遷移先にテイントタグを伝搬させる．これにより，あるメモリ上の値を見たとき，その値に設定されているテイントタグを確認することで，その値の起源を知ることができる．

3.2 基本設計

本提案手法は，ハードウェア全体をエミュレートする VMM を用いて，その VMM 上で動作するゲスト OS 内で解析対象コードを実行する環境で利用することを前提としている．本提案手法は，解析対象コードを実行する前に，解析対象を示すテイントタグを，その解析対象コードに対して設定する．そして，VMM の仮想 CPU において，命令フェッチの際にその命令に設定されているテイントタグを確認し，解析対象を示すテイントタグが設定されていた場合は，その命令を解析対象コードとして実行する．設定されていない場合は通常のコードとして実行する．そして，解析対象コードに対して，移動，コピー，または演算といった処理が行われた場合は，その移動，コピー先，演算結果の保存先に，その解析対象コードに設定されているテイントタグを伝搬させて，追跡する．

3.3 動作シーケンス

図 1 に提案システムの動作シーケンスを示す．最初に，解析対象であるマルウェアの実行ファイルのディスク上の位置を特定する（図 1 の 1）．次に，その実行ファイルの位置情報を利用し，実行ファイルに対して解析対象を示すテイントタグを設定する（図 1 の 2）．この際，テイントタグはディスク上のファイルに設定されたテイントタグを管理する機構であるシャドウディスクに保存される．続いて，マルウェアの実行ファイルが実行され，物理メモリ上にロードされると，それに応じてファイルに設定されていたテイントタグもシャドウディスクから，メモリ上の値のテイントタグを管理する機構であるシャドウメモリに伝搬され，保存される（図 1 の 3）．CPU が命令をフェッチす

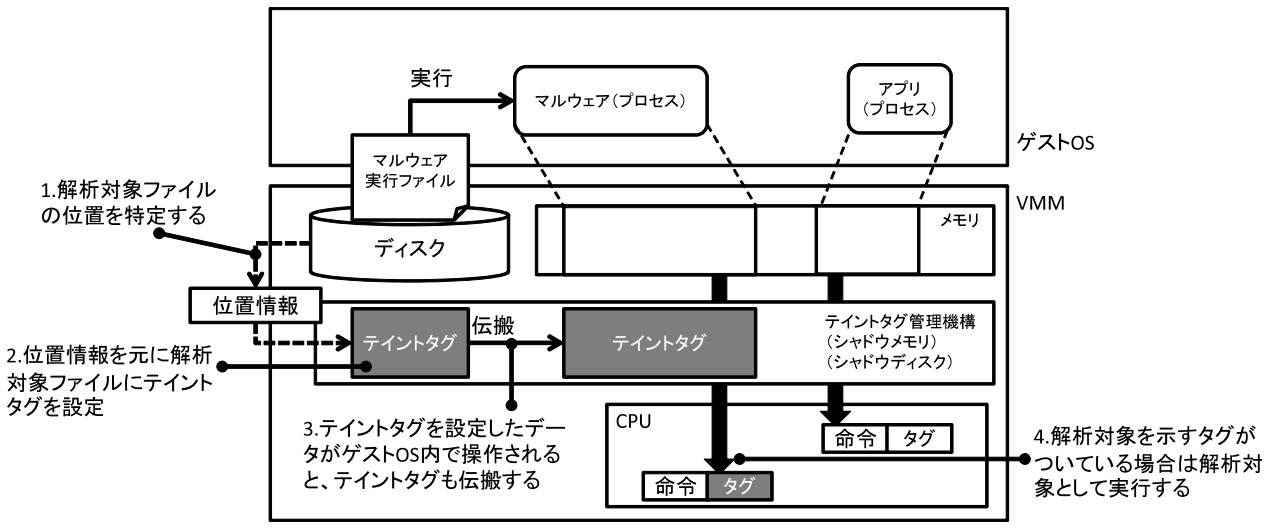

図 1 提案システムの概要図と動作例
Fig. 1 Overview of system in action.

る際に，同時にシャドウメモリも参照し，フェッチした命令に設定されているテイントタグも取得する．この際，解析対象を示すテイントタグが設定されていた場合，その命令を解析対象として実行する（図 1 の 4）．

以降では，上記の基本設計をマルウェアの解析に適用する場合の問題について述べ，これらの問題を解決するためのファイルへのテイントタグ伝搬機構と強制的なテイントタグ伝搬機構について述べる．

3.4 ファイルへのテイントタグ伝搬

ここでは，マルウェアがファイルへ実行コードの一部を書き出し，それを他プロセスへ注入する場合などを考慮し，メモリ上のテイントタグをディスク上のファイルへ伝搬，保持させる仕組みについて述べる．

マルウェアの中には，コード注入を行う際，注入するコードをいったんファイルとして保存し，そのファイルを直接，または間接的に別プロセスに注入するものがある．このようなマルウェアに対処するため，メモリ上のデータがファイルに書き出される場合，メモリ上のデータに設定されているテイントタグをディスク上のファイルへも伝搬させる機能を追加する．具体的には，ディスク上のデータに対するテイントタグを保持するデータ構造体であるシャドウディスクと，メモリ上のデータがディスクに Direct Memory Access（DMA）転送された場合，シャドウメモリ上のテイントタグをシャドウディスクへ転送させるシャドウディスク，シャドウメモリ間のテイントタグ伝搬機構とを提案システムに追加する．

これにより，マルウェアがコード注入のために書き出したファイルに対してもテイントタグを伝搬させ，そのファイルが別のプロセス上で実行された場合に，その実行も解析対象とすることができる．

またこの機構を逆向きの転送，つまりディスクからメモリへ転送されるデータに対しても適用する．この機能を利用することで，解析対象マルウェアの実行ファイルを実行する前に，このファイルに対してテイントタグを設定することができる．

3.5 強制的なテイントタグ伝搬

ここでは，テイントタグの伝搬が途切れてしまうという問題について述べ，その問題を回避するため，提案システム上で利用している強制的なテイントタグ伝搬機構について述べる．

パッキングされたマルウェアがオリジナルコードをメモリ上に展開する際に行う復号処理の最中に，解析対象コードであるマルウェアのコードに設定したテイントタグの伝搬が失敗する可能性がある．これは，文献 [7] でも指摘されているように，既存システムのテイントタグ伝搬機構は，意味上は依存関係のあるデータ間でも命令単位で見たとき直接の代入関係がない場合はテイントタグを伝搬させることができない，といった問題があるためである．

そこで，コードの動的生成によるテイントタグの消失を防ぐため，提案システムでは解析対象コードが何かしらの書き込みを行った場合，その書き込み先に対して，解析対象を示すテイントタグを強制的に伝搬させる方法をとる．これにより，仮にマルウェアが上記のようなコードを利用し，実行コードを生成したとしても，マルウェアが書き込んだメモリ上の値には解析対象を示すテイントタグが強制的に設定されるので，生成されたコードが解析対象から外れることを防止できる．

一方，解析対象コードがメモリ書き込みをともなう API やライブラリを呼び出した場合，テイントタグの伝搬は通常の伝搬機構に任せ，強制的なテイントタグ伝搬は行わない．これは，OS が提供している API やライブラリの中には上記のように意図的にテイントタグの伝搬を切断する動作をするものは含まれていないと想定しているためである．この件に関しては，7 章で詳しく述べる．

4. 実装

本章では，提案システムの実装について述べる．提案システムの主要な構成要素である仮想 CPU，テイントタグを保存するシャドウメモリ，シャドウディスク，シャドウメモリとシャドウディスクのテイントタグ伝搬を行う仮想 DMA コントローラについて述べる．

4.1 仮想 CPU

本提案手法は，Argos-0.5.0 [16] 上に実装を行った．Argos は Qemu [4] 上に実装されたテイント解析機能付きのハードウェアエミュレータであり，ハニーポットとしてゼロデイ攻撃を検知するために設計，実装されている．

Qemu の仮想 CPU には仮想化を実現するための 2 つの機能がある．ゲスト OS で実行される命令（以下，ゲスト命令）を，その挙動の整合性を保ったままホスト OS で実行可能な命令列（以下，ホスト命令列）に変換する動的バイナリ変換機能と，そのホスト命令列を実行する機能である．Argos では，動的バイナリ変換の際に，ゲスト命令として値を読み書きする命令が入力されると，その命令のホスト命令列を生成するとともに，その命令で読み書きされる値に設定されているテイントタグの伝搬を行うホスト命令列を生成する．この仕組みにより，テイントタグ伝搬を可能にしている．

提案システムでは，この Argos の仮想 CPU に以下の 2 つの機能を新たに実装した．

- 解析対象コードの判定・処理
- 強制的なテイントタグ伝搬

解析対象コードの判定・処理機能は，動的バイナリ変換でゲスト命令を読み込んだ際に，その命令に設定されてい

るテイントタグの値をチェックし，その値に応じた処理コードをホスト命令列として生成する．提案システムでは，ゲスト命令に解析対象を示すテイントタグが設定されていた場合，その命令とレジスタ値などの CPU のコンテキスト情報をログに出力する処理コードを生成する．これにより，解析対象コードの詳細な実行トレースを取得することができる．

強制的なテイントタグ伝搬機能は，Argos のメモリ上の値に対するテイントタグ伝搬を行うための処理部分を改造している．テイントタグ伝搬を行う関数の中で，実行している命令が解析対象であり，かつ，メモリに対してテイントタグの設定されていない値の書き込みを行うかを確認する．これらの条件が満たされるとき，書き込み先のメモリ領域に対して解析対象を意味するテイントタグを設定する．

4.2 シャドウメモリ

シャドウメモリは，Argos の実装をそのまま利用した．提案システムでは，物理メモリ上の 1 バイトの値に対して，1 バイトのシャドウメモリのエントリを割り当てている．つまり，ゲスト OS の物理メモリを 256 M バイトとすると，シャドウメモリは 1 バイトのエントリが 256×2^{20} 個並んだ配列になる．たとえば，物理アドレス 0xdeadbeef のシャドウメモリにアクセスする際は，シャドウメモリの配列を shdmem とすると，shdmem[0xdeadbeef] でアクセスできる．

4.3 シャドウディスク

シャドウディスクは，ディスク上のデータに対して設定されたテイントタグを保持するデータ構造体である．提案システムでは，テイントタグが設定されたデータ領域に関して，その領域のセクタごとに，セクタ番号，テイントタグを保存するバッファで構成されたデータ構造体（以下，SHD 構造体）を定義し，その構造体を 1 つのノードとして，セクタ番号をキーとする 2 分木（以下，SHD 木）で管理している．また，シャドウメモリ同様，ディスク上の 1 バイトのデータに対して 1 バイトのメモリ領域を割り当てている．1 セクタのサイズを 512 バイトだとすると，SHD 構造体のテイントタグを保存するバッファも 512 バイトになる．

通常ディスクは物理メモリに比べてサイズが大きいため，シャドウメモリのようにディスク上の全データに対応するエントリをあらかじめ確保するのはメモリ消費量が膨大になり困難である．そこで，提案システムではテイントタグを保持しているデータのみメモリを割り当てている．このシャドウディスク（SHD 木）に対しては，ノードの追加，削除，変更の操作を行うことができる．追加を行う際は，開始セクタ番号，オフセット，サイズ，テイントタグ，を引数とするノードの追加関数を呼び出す．この関数は，引数で渡された情報をもとにノードを作成し，セクタ番号をキーにして SHD 木を走査し，ノードを追加する．また，ノードを削除する場合は，セクタ番号を引数とするノードの削除処理を行う関数を呼び出す．ノードのテイントタグの値を変更する場合は，変更したいセクタ番号を引数として走査関数を呼び出し，目的の SHD 構造体が取得できた際はこの構造体のメンバを直接変更する．

ディスク上に保存されているファイルのセクタ番号，オフセット，サイズは，ディスクフォレンジックツールである The Sleuth Kit-3.2.3 [6] を用いて取得している．

4.4 仮想 DMA コントローラ

提案システムでは，DMA 要求を処理する仮想 DMA コントローラ内でデータ転送要求が来るのを監視し，要求が来た場合，その要求のデータ転送先と転送元のディスク，物理メモリの位置を特定する．そして，これらの位置情報に対応するシャドウメモリ，シャドウディスク間でテイントタグの転送を行う．

シャドウメモリ，シャドウディスクともに 1 バイトのデータに対して 1 バイトのシャドウ領域を割り当てているため，データ転送でテイントタグが失われることや，拡散することはない．

また，この仮想 DMA コントローラは，マルウェアが行ったファイル書き込みに関する DMA 要求だけでなく，ゲスト OS 全体からの DMA 要求を対象としている．そのため，たとえば OS がメモリページをスワップアウトした場合でも，適切にテイントタグがシャドウディスクに転送される．このメモリページがスワップインするときには，シャドウメモリへテイントタグが転送される．

5. 実験

本章では，提案手法の有効性を示すため行った 3 つの実験とその結果について述べる．まず 3 つの実験の目的について述べ，続いて，個々の実験の詳細と実験結果を述べる．

5.1 実験の目的

1 つ目の実験の目的は，3.5 節で提案している強制的なテイントタグ伝搬の有効性を示すことである．パッカにより，パッキングされた実行ファイルは，オリジナルコードを動的生成する．本実験では，動的生成されたコードにもテイントタグが伝搬しているか確認する．

2 つ目の実験の目的は，提案システムが 2 章であげた問題を解決可能であることを示すことである．テイントタグにより粒度を細かく監視することで正規のコード部分を誤って解析対象としないことを示す．また，様々なコード注入方法において，提案システムがそれらの注入方法に依存せずに解析対象コードを追跡可能であり，挙動の見逃しが発生しないことを示す．

3つ目の実験では提案手法が実際のマルウェアに対しても適用可能なことを示す．

すべての実験は，Intel Xeon CPU X5670，12 G バイトのメモリ，512 G の SSD 上に Ubuntu Linux 10.10 をインストールした環境で行った．また，VMM 上で動作させるゲスト OS は Windows XP SP2 を利用した．

5.2 実験 1

5.2.1 実験方法

実験 1 では，**表 1** に示す 9 つのパッカを用いて calc.exe をパッキングし，実験用の実行ファイルを用意した．ここで利用した 9 つのパッカは一般的なマルウェアによく利用されているものであり，実行時にオリジナルコードを動的生成する機能を持っている．

これらの実行ファイルを，強制的なテイントタグ伝搬の機能がない環境と，ある環境，それぞれで動作させ，動的生成されたオリジナルコードの実行をとらえることができるかを実験した．具体的には，動的生成されるオリジナルコードのうち，オリジナルエントリポイント（以下，OEP）に着目し，calc.exe の OEP の実行を正確にとらえることができるかを実験した．解析対象を示すテイントタグが正確に伝搬されている場合は，OEP を含む動的生成されたコードにもテイントタグが設定されており，それらのコードの実行を解析対象コードの実行としてとらえることができる．一方で，テイントタグの伝搬が切れていた場合は，OEP を含む動的生成されたコードの実行を見逃してしまう．

この実験では，正確に OEP の実行をとらえた場合を検知とし，OEP の実行を見逃した場合を検知漏れとする．

5.2.2 実験結果

実験 1 の結果を表 1 に示す．強制的なテイントタグ伝搬を用いた場合は正確に OEP の実行を検知できたが，強制的なテイントタグ伝搬を用いずに通常のテイントタグ伝搬ルールを利用した場合は，多くの場合で OEP の実行を検知できなかった．本実験の結果より，強制的なテイントタグ伝搬がパッキングされたマルウェアを解析する際，有効だということができる．

表 1 実験 1 の結果
Table 1 Result of 1st experiment.

パッカ	強制伝搬なし	強制伝搬あり
UPX	検知漏れ	検知
WinUpack	検知漏れ	検知
Yoda's Protector	検知漏れ	検知
aspack	検知漏れ	検知
execryptor	検知	検知
fsg	検知	検知
PE2Compact	検知漏れ	検知
npack	検知	検知
mew	検知漏れ	検知

5.3 実験 2

5.3.1 実験方法

実験 2 では，同一プロセス内や，同一スレッド内で動作し，様々な方法でコード注入を行う 5 種類のコードを用意し，これらを用いて実験を行った．この 5 種類のコードは以下の考察に基づき選定した．

まずは注入対象がファイルか，プロセスか，で分類する．ファイルに対してコードを注入する手法は，コードを付与する位置に違いはあるものの，おおむね同じような動作をする．具体的には，実行ファイルの前後にコードを追加し，プログラムのエントリポイントから注入したコードに到達できるようにファイルの一部を書き換えることで行われる．次にプロセスに対するコード注入方法を考えると，コード注入は基本的には次の 2 ステップから構成される．

(1) 対象のメモリ空間にコードを配置する．
(2) 配置したコードを実行する．

(1) に関しては，次の方法が考えられる．

A 直接メモリ空間にコードを書き込む（直接）．
B 注入するコードを DLL（Dynamic Link Library）としてファイルに書き出し，その DLL をレジストリに登録する（ファイル経由）．

A に関しては，API を用いて，注入対象のプロセス空間にメモリ領域を確保し，そのメモリ領域にコードを直接書き込む．B に関しては，コードを DLL ファイルに書き出し，特定のプロセス，スレッドが登録されている DLL を実行するレジストリに，その DLL を登録する．また (2) に関しては，次の方法が考えられる．

a API で実行する（能動的）．
b 実行されるのを待つ（受動的）．

a に関しては，API を用いて，注入したコードを能動的に実行する．b に関しては，注入したコードが実行されるイベントが発生するのを待つ．以上より，A または B，かつ，a または b の組合せにより，プロセスに対するコード注入には 4 種類のパターンが存在する．

以上の考察に基づき，以下の 5 種類のテストコードを用意した（**表 2**）．

コード 1 ファイル感染
実行ファイル（calc.exe）にコードを付加し，PE ヘッダを修正し，そのファイル実行する．

コード 2 CreateRemoteThread
WriteProcessMemory でコードを対象プロセス（calc.exe）に直接書き込み，CreateRemoteThread で対象プロセス内に新たなスレッドを生成し，そのコードを実行する．

コード 3 SetWindowsHookEx
注入コードを DLL ファイルに書き出し，SetWindowsHookEx を利用してその DLL を対象プロセス（calc.exe）に注入し，実行させる．正確には注入先の

表 2 実験 2 の結果
Table 2 Result of 2nd experiment.

注入方法	注入対象	注入パターン		誤検知	検知漏れ
ファイル感染	実行ファイル	—		なし	なし
CreateRemoteThread	プロセス	直接	能動的	なし	なし
SetWindowsHookEx	プロセス	ファイル経由	能動，受動的	なし	なし
AppInit_DLLs	プロセス	ファイル経由	受動的	なし	なし
CreateFile フック	プロセス	直接	受動的	なし	なし

スレッドに対してウィンドウメッセージが配信された際に，注入したコードがそのスレッドのコンテキスト内で動作する．

コード 4 AppInit_DLLs
注入コードを DLL ファイルに書き出し，AppInit_DLLs レジストリキーにその DLL を登録する．user32.dll をロードしたプロセス（スレッド）にその DLL が注入され，そのスレッドのコンテキスト内で動作する．

コード 5 CreateFile フック
WriteProcessMemory でコードを対象プロセスに直接書き込み，対象プロセスの CreateFile API をフックする．対象プロセス（スレッド）（calc.exe）内で CreateFile が呼ばれると注入したコードにジャンプし，そのスレッドのコンテキスト内でそのコードが動作する．

コード 1，3，4，5 では，2 章であげた問題 (1) のように，同一プロセス，スレッド内で注入したコードと元のコードが動作する．この状況において，提案システムが注入したコードと元のコードの実行を正確に区別できるかを実験する．

コード 1，2，3，4，5 では，他プロセスやファイルに対してコード注入が行われる．2 章の問題 (2) で述べているように，従来手法でこれらのコード注入を追跡するためには，注入方法ごとに監視ポイントを設ける必要がある．たとえば，コード 2 の場合は，CreateRemoteThread を監視し，注入が行われた PID と TID を取得する必要がある．コード 4 の場合は，AppInit_DLLs レジストリへのアクセスを監視し，そこに登録されるファイル名を取得し，そのファイルをロードするプロセスとスレッドをシステム全体から見張り，それらの PID と TID を取得する必要がある．提案手法では，これらの注入方法に依存せずにコードの追跡が可能であることを示す．既存のマルウェアのコード注入方法は，ここで用意したテストコード以外にも存在するが，利用する API やレジストリが異なる程度で，上記 5 パターンのどれかと類似する．そのため，この 5 パターンを正確に対処可能であれば，コード注入を行う多くのマルウェアに対応できると考える．

この実験では，解析対象コード以外の実行トレースを取得してしまった場合を誤検知とし，解析対象コードの一部の実行トレースを取り逃した場合を検知漏れとする．これら，誤検知，検知漏れの有無を調べ，正確に実行トレースが取得できたかを評価する

5.3.2 実験結果

実験 2 の結果を表 2 に示す．すべてのテストコードにおいて，提案手法では正確に解析対象コードの特定，追跡ができた．同一プロセス，スレッド内に解析対象コードと元のコードとが混在した状態でも，正確に解析対象コードの実行のみを抽出できた．また，各種コード注入方法に依存せずに，コード注入を追跡し，注入先のプロセスでの解析対象コードの実行を追跡できた．

以上により，提案システムは，2 章であげた 2 つの問題を解決できているといえる．

5.4 実験 3

5.4.1 実験方法

実験 3 では，CCC Dataset 2012 [12] のマルウェア検体 10,538 のうちアンチウィルス名でユニークなものに絞り込んだ 18 検体と，D3M で提供されているマルウェア 14 検体の合計 32 検体を対象にアプリケーションレベルで動作するコード部分の正確なトレースが取得できるかを実験した．この実験では，実際のマルウェアを利用しているため，取得した実行トレースが本当にマルウェアのものかを知るのは難しい．そこで，ここでは提案手法と PID，TID による識別方法の比較実験を行った．それぞれの手法で取得した実行トレースを比較し，実行トレースに違いが出た場合，その原因を調査した．その原因が，提案システムが解析対象コード以外の実行トレースを取得していることに起因する場合，これを誤検知とする．また，原因が，提案システムが解析対象コードとすべきコードの実行トレースの見逃したことに起因する場合，これを検知漏れとする．

なお，実験はすべてインターネットから隔離された環境で行い，1 検体あたりの実行時間は 5 分とした．

5.4.2 実験結果

この 32 検体のうち，22 検体はコード注入のような監視から逃れるための解析妨害を行わなかった[*1]．残りの 10 検体は，何かしらの監視から逃れるための動作を行った．この結果を**表 3** に示す．

493e3...，7aaed... の検体に関しては提案システムで検知漏れが発生した．この原因については以下のとおりである．

[*1] 単純な子プロセスの生成は除外した．

表 3 実験 3 の結果
Table 3 Result of 3rd experiment.

ハッシュ値	解析妨害	誤検知	検知漏れ
4cfab...	svchost.exe を改ざんして実行	なし	なし
84114...	CreateRemoteThread で他プロセスにコード注入	なし	なし
493e3...	cmd.exe でバッチファイルを実行	なし	あり
7aaed...	cmd.exe でバッチファイルを実行	なし	あり
9e366...	子プロセスをサービスとして起動	なし	なし
e95f7...	子プロセスをサービスとして起動	なし	なし
28d7c...	子プロセスをサービスとして起動	なし	なし
4ead4...	子プロセスをサービスとして起動	なし	なし
52a6d...	CreateRemoteThread で他プロセスにコード注入	なし	なし
545e2...	既存サービスを改ざんして実行	なし	なし

これらの検体は，実行の途中でバッチファイルを生成し cmd.exe 上でそのバッチファイルを実行する．バッチファイルは cmd.exe 上で実行されるコマンドを文字列として保持したファイルである．cmd.exe 上でコマンドが実行されると，そのコマンドに応じた機能や外部プログラムが呼び出される．cmd.exe は渡されたコマンド文字列を参照するのみであり，文字列を直接実行するわけではない．そのため，たとえ文字列に解析対象を示すテイントタグが設定されていた場合であっても，この文字列は cmd.exe の挙動を決定するために参照されるのみであり，cmd.exe によって実行される命令列とは直接の代入関係がない．そのため，コマンドによって cmd.exe 内で実行される命令列を解析対象とすることはできず，検知漏れが発生した．

その他の検体に関しては，誤検知，検知漏れなく正確にマルウェアの実行トレースを取得できた．

6. 関連研究

ここでは提案手法に関連する先行研究について述べる．動的解析における解析対象の識別について述べている研究は多くはない．しかし，PID や TID を利用して解析対象の識別を行う研究は複数行われている．

TTAnalyze [3]（Anubis [1]）はマルウェアの動的解析を行うためのシステムであり，Qemu 上に実装されている．TTAnalyze では，ゲスト OS 内にインストールした probe モジュールを通して CR3 レジスタの値と PID を結び付け，この CR3 を使って VMM 内で解析対象コードの識別を行っている．

Panorama [23] は TEMU [20] をベースにして実装されたシステムで，マルウェアの検知と解析を行う．TTAnalyze と同様に，ゲスト OS 内にモジュールをインストールし，そこからゲスト OS の PID，TID やモジュールのロードアドレスなどのセマンティックス情報を取得している．

Ether [10] は，Xen [2] 上に構築されたマルウェアを動的解析するための環境であり，Intel VT [21] を利用して実装されている．解析対象の識別は PID，TID に基づいて行われている．これら PID，TID の情報は，あらかじめゲスト OS を解析し，PID と TID の保存されている箇所を特定しておき，直接そこから読み取ることで取得している．

VMwatcher [13] は，あらかじめ Windows を解析し，各種データ構造体のシグネチャ [5] を作成する．そのシグネチャをもとに，各データ構造体のメモリ上の位置を特定し，そこから直接ゲスト OS のセマンティックス情報を取得している．

Virtuoso [11] は，Qemu 上に構築されたメモリフォレンジックツール生成用の環境である．作成したいツールと同等の動作をするプログラムを事前に実行し，その挙動から，その挙動を模倣するフォレンジックツールを生成する．

上記の関連研究はいずれも PID，TID を取得し，それに基づいて解析対象コードを識別するものである．本論文で述べてきたとおり，この方法に基づく識別には 2 章であげた問題が存在する．本研究は，この問題を解決するために行われたものであり，本提案手法を適用することで上記すべての解析環境の精度を向上させることができると考える．

7. 考察

本章では，提案手法の制限について，テイントタグ伝搬の正確性，Return Oriented Programming（以下，ROP），仮想マシン検知，提案システムのパフォーマンスの観点から議論する．

7.1 テイントタグ伝搬の正確性

テイントタグ伝搬の正確性に関して，テイントの伝搬漏れと誤り伝搬の観点から議論する．

7.1.1 テイントの伝搬漏れ

ここでは，ライブラリ関数とスクリプトファイルのテイントの伝搬漏れについて議論する．

3.5 節でも述べたとおり，既存のテイント解析手法ではテイントタグの伝搬漏れの問題がある．提案システム上では，強制的なテイントタグ伝搬を用いて，解析対象コード内での伝搬漏れを防いでいる．しかし，Windows が提供す

る API 中で伝搬漏れが発生した場合，強制的なテイントタグ伝搬では対処できない．実際，暗号や文字コード変換の API 中で，伝搬漏れが発生するという報告がある [9], [23]．仮に，攻撃者がこれらの API を使って，コード生成を行った場合，解析対象を示すテイントタグが消える可能性がある．その結果，正確に解析対象コードの識別ができなくなる可能性がある．この問題を解決するため，たとえば問題のある API を特別扱いし，API の入口と出口でテイントの伝搬をチェックする仕組みを作ることで対処できると考える．

5.4 節でも述べたとおり，解析対象がスクリプトの場合，提案手法を適用することが難しい．スクリプトの文字列は実行するコードを決めるために参照されるだけで，スクリプト自体が直接 CPU で実行されるわけでない．そのため，スクリプトにテイントタグを設定したとしても，それが CPU 上で実行されるコードにまで伝搬されない．これを回避するために，文献 [19] で提案されている，テイントを操作するコードが，ある構造的な特徴を持っていた場合，参照したデータのテイントタグを伝搬させる方法を検討する必要がある．

7.1.2 テイントの誤伝搬

攻撃者は本提案手法を回避するために意図的にテイントの誤り伝搬を引き起こすことができる．たとえば，解析対象コードが正常なコードの一部を読み取り，同じ値で上書きした場合，値は変化ないため，挙動に変化はないが，書き込まれた箇所が解析対象になってしまう．これにより，本来解析対象でないコードを解析対象として実行してしまう可能性がある．これらの問題を解決するためには，文献 [14], [19] のような，テイントの伝搬を強固にする手法を取り入れる必要がある．

7.2 ROP

ROP [18] で悪意のある処理を記述されると，テイントタグが設定されていないシステム上の実行ファイル内の命令列が，マルウェアの実行に利用されるため，提案手法ではこのコードの実行を見逃してしまう．しかし，ROP は通常のプログラムに比べ，ret の数が極端に多いなどの特徴的な挙動が多いため，その検知を行うのは難しくない [8]．ROP を利用するマルウェアを検知した場合，その検体に限り，PID，TID に基づく手法に切り替えるなどで，回避することが可能だと考える．

7.3 仮想マシン検知

マルウェアによる仮想マシン検知は，仮想マシンの実装に依存する手法が主である．そのため，この問題は提案手法を実装するプラットフォームの問題であり，本提案手法の本質的な制限事項ではない．しかし，解析システムを構築する場合に，考慮する必要があるため，ここでは提案システム上で利用している手法について述べる．

提案システムでは，エミュレータ型の仮想マシン上に構築している．そのため，マルウェアに仮想マシンの存在を検知される恐れがある [17]．そこで提案システム上では，以下の 2 つの方法で仮想マシン検知機能に対処している．

a　マルウェア実行時に特定の命令パターンを書き換える．
b　仮想マシン固有の文字列をソースコードから取り除く．

a に関して，提案システムでは，既知の仮想マシン検知手法の命令パターンをシグネチャとして持たせ，このシグネチャにマッチする命令列が実行時に現れた場合，その命令列を無効な命令に書き換える方法をとっている．

b に関して，提案システムでは，あらかじめ，仮想マシンのソースコードを修正し，仮想ハードウェア固有の文字列をソースコードから消すことで対処している．

7.4 提案システムのパフォーマンス

提案システムでは通常のテイント解析に加え，仮想 CPU での解析対象コードの判断・処理，仮想 DMA コントローラでのテイントタグ伝搬により，実行速度の低下がある．簡易的な実験を行ったところ，デフォルトの Qemu と比べ 2 倍から 10 倍程度の実行速度の低下がみられた．この速度低下は実用には大きな問題にはならないが，副作用として次のような問題が考えらえる．

マルウェアの中には特定の実行パスの処理にかかる平均的な時間を計測し，その時間が大きく異なる際，解析されていると判断するものがある．そのため，提案システムでの解析時の処理低下がマルウェアに検知される恐れがある．このようなシステム内の時計を利用して時間を計測しているマルウェアに対しては，ゲスト OS 内の時間を制御する方法 [15] で対処可能だと考える．仮想 CPU や DMA コントローラでの処理中は，ゲスト OS 内の時計の進み具合を制御することでマルウェアに見せる時間を誤魔化し，時間検知を回避することが可能である．

8. まとめ

マルウェアの動的解析には PID や TID を利用した解析対象の識別が多く利用されている．しかしながら，この方法では不正なコードと正規なコードが同一プロセス，スレッド内に混在した状況で正しく両者を識別することができない．さらに，未知の方法でコード注入が行われた場合，その注入を追跡することができず，注入された先でのコード実行を見逃してしまう．本論文はこれらの問題を解決するため，テイントタグに基づいた解析対象コードの識別，追跡方法を提案した．マルウェアの動作を模倣したテストコードと実際のマルウェアを利用して実験を行った結果，提案手法が実際のマルウェア解析にも適用可能であることを示した．本提案手法を利用することで，既存の多くのマルウェア解析システムや種々のマルウェア対策技術の精度

と効果を向上させることができる．

参考文献

[1] Anubis: Analyzing Unknown Binaries, International Security Lab (online), available from ⟨http://anubis.iseclab.org/⟩ (accessed 2012-12-13).

[2] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A.: Xen and the art of virtualization, *Proc. 19th ACM Symposium on Operating Systems Principles, SOSP '03*, New York, NY, USA, ACM, pp.164–177 (2003).

[3] Bayer, U., Kruegel, C. and Kirda, E.: TTAnalyze: A Tool for Analyzing Malware, *Proc. European Institute for Computer Antivirus Research Annual Conference* (2006).

[4] Bellard, F.: QEMU, a Fast and Portable Dynamic Translator, *USENIX Annual Technical Conference, FREENIX Track, USENIX*, pp.41–46 (2005).

[5] bugcheck: GREPEXEC: Grepping Executive Objects from Pool Memory, Uninformed (online), available from ⟨http://uninformed.org/⟩ (accessed 2012-12-13).

[6] Carrier, B.: The Slueth Kit (TSK), available from ⟨http://www.sleuthkit.org/⟩ (accessed 2012-12-13).

[7] Cavallaro, L., Saxena, P. and Sekar, R.: On the Limits of Information Flow Techniques for Malware Analysis and Containment, *Proc. 5th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, DIMVA '08*, Berlin, Heidelberg, pp.143–163, Springer-Verlag (2008).

[8] Chen, P., Xiao, H., Shen, X., Yin, X., Mao, B. and Xie, L.: DROP: Detecting Return-Oriented Programming Malicious Code, *Proc. 5th International Conference on Information Systems Security, ICISS '09*, Berlin, Heidelberg, pp.163–177, Springer-Verlag (2009).

[9] Chow, J., Pfaff, B., Garfinkel, T., Christopher, K. and Rosenblum, M.: Understanding data lifetime via whole system simulation, *USENIX Security Symposium*, pp.321–336 (2004).

[10] Dinaburg, A., Royal, P., Sharif, M. and Lee, W.: Ether: Malware analysis via hardware virtualization extensions, *Proc. 15th ACM Conference on Computer and Communications Security, CCS '08*, New York, NY, USA, ACM, pp.51–62 (2008).

[11] Dolan-Gavitt, B., Leek, T., Zhivich, M., Giffin, J.T. and Lee, W.: Virtuoso: Narrowing the Semantic Gap in Virtual Machine Introspection, *IEEE Symposium on Security and Privacy*, pp.297–312 (2011).

[12] 畑田充弘，中津留勇，秋山満昭：マルウェア対策のための研究用データセット～MWS 2011 Datasets～，情報処理学会シンポジウムシリーズ，Vol.2011 (2011).

[13] Jiang, X., Wang, X. and Xu, D.: Stealthy malware detection and monitoring through VMM-based "out-of-the-box" semantic view reconstruction, *ACM Trans. Inf. Syst. Secur.*, Vol.13, No.2, pp.12:1–12:28 (2010).

[14] Kang, M.G., McCamant, S., Poosankam, P. and Song, D.: DTA++: Dynamic Taint Analysis with Targeted Control-Flow Propagation, *NDSS* (2011).

[15] Kawakoya, Y., Iwamura, M. and Itoh, M.: Memory Behavior-Based Automatic Malware Unpacking in Stealth Debugging Environment, *Proc. 5th IEEE International Conference on Malicious and Unwanted Software* (2010).

[16] Portokalidis, G., Slowinska, A. and Bos, H.: Argos: An emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation, *Proc. 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006, EuroSys '06*, New York, NY, USA, ACM, pp.15–27 (2006).

[17] Raffetseder, T., Krügel, C. and Kirda, E.: Detecting System Emulators, *ISC*, pp.1–18 (2007).

[18] Roemer, R., Buchanan, E., Shacham, H. and Savage, S.: Return-Oriented Programming: Systems, Languages, and Applications, *ACM Trans. Inf. Syst. Secur.*, Vol.15, No.1, p.2 (2012).

[19] Slowinska, A. and Bos, H.: Pointless tainting?: Evaluating the practicality of pointer tainting, *Proc. 4th ACM European Conference on Computer Systems, EuroSys '09*, New York, NY, USA, ACM, pp.61–74 (2009).

[20] Song, D., Brumley, D., Yin, H., Caballero, J., Jager, I., Kang, M.G., Liang, Z., Newsome, J., Poosankam, P. and Saxena, P.: BitBlaze: A New Approach to Computer Security via Binary Analysis, *Proc. 4th International Conference on Information Systems Security, ICISS '08*, Berlin, Heidelberg, pp.1–25, Springer-Verlag (2008).

[21] Uhlig, R., Neiger, G., Rodgers, D., Santoni, A.L., Martins, F.C.M., Anderson, A.V., Bennett, S.M., Kagi, A., Leung, F.H. and Smith, L.: Intel Virtualization Technology, *Computer*, Vol.38, No.5, pp.48–56 (2005).

[22] Yason, M.V.: The Art of Unpacking, *Black Hat USA Briefings* (2007).

[23] Yin, H., Song, D., Egele, M., Kruegel, C. and Kirda, E.: Panorama: Capturing system-wide information flow for malware detection and analysis, *Proc. 14th ACM Conference on Computer and Communications Security, CCS '07*, New York, NY, USA, ACM, pp.116–127 (2007).

推薦文

従来は専門家の手動解析を要する課題に対して，強制テイント伝搬ならびにファイルへのテイント伝搬という提案手法による新規性の高い論文となっている．国内におけるマルウェアのテイント解析の研究は海外に比べて少なく，先進的である．また，評価目的に応じたテストコードおよび実マルウェアによる評価から信頼性の高い結果を得ている．テイント解析の手法，実装および評価方法が丁寧に論じられており，今後の同研究分野の発展に有用であるため，推薦したい．

（マルウェア対策研究人材育成ワークショップ 2012
プログラム委員長　畑田充弘）

川古谷 裕平　（正会員）

2005 年早稲田大学大学院理工学研究科修士課程修了．同年日本電信電話株式会社入社．現在，NTT セキュアプラットフォーム研究所勤務．

岩村 誠 （正会員）

2002 年早稲田大学大学院理工学研究科修士課程修了．同年日本電信電話株式会社入社．2012 年早稲田大学大学院基幹理工学研究科博士課程修了．現在，NTT セキュアプラットフォーム研究所勤務．博士（工学）

針生 剛男

1991 年電気通信大学大学院電気通信学研究科修士課程修了．同年日本電信電話株式会社入社．現在，NTT セキュアプラットフォーム研究所勤務．