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 †　　塩治榮太朗 †　　岩村誠 †　　針生剛男 †

†NTT セキュアプラットフォーム研究所
180-8585 東京都武蔵野市緑町 3-9-11
{kawakoya.yuhei,shioji.eitaro,iwamura.makoto,hariu.takeo}@lab.ntt.co.jp

あらまし　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監視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。

Tracing Malicious Code with Taint Propagation

Yuhei Kawakoya†　　Eitaro Shioji†　　Makoto Iwamura†　　Takeo Hariu†

†NTT Secure Platform Laboratories
3-9-11 Midori-cho, Musashino-shi, Tokyo, 180-8585, JAPAN
{kawakoya.yuhei,shioji.eitaro,iwamura.makoto,hariu.takeo}@lab.ntt.co.jp

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 Dataset2012. The results of these experiments indicated that our approach is effective against various anti-analysis functions, and that it is applicable to real malware. Our proposal will allow existing malware analysis environments and antimalware research to be more precise and effective.

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区別する必要がある。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、自身のコードを実行中の他のプロセスに注入するコード注入や、実行ファイルを書き換えて、自身のコードを付け加えるファイル感染といった解析妨害機能を利用する。これにより、PID や TID を基にした監視では、検知漏れや誤り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 に基づく方法には以下の問題があると考える。

1. PID や TID では監視の粒度が粗い
2. 未知のコード注入を追跡できない

1 に関して、同一 PID、TID 上に悪意のあるコードと正規のコードが混在する場合、両者のコードを区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、誤って監視対象としてしまう可能性がある。一方、2 に関して、マルウェアが他の実行プロセスに対して未知の方法でコード注入を行った場合、この注入を追跡できず、注入先のプロセスやスレッドを監視対象にすることができない。既知のコード注入に関しては、API フックなどで予めポイントとなる API を監視しておくことで、追跡が可能である。しかし、未知の方法でコード注入が行われた場合、その注入経路を捉えることができないため、注入先で実行されるコードの挙動を見逃してしまう。

そこで本研究では、これら二つの問題を解決するテイントタグに基づた解析対象コードの識別、追跡方法を提案する。本提案手法では、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。また、監視対象コードがゲスト OS 内で移動や改変された場合、その操作をテイントタグを伝搬させることで追跡する。さらに、通常のテイントタグの伝搬に加えて、マルウェアによる動的生成コードやファイル書き出しなどを考慮し、強制的なテイントの伝搬、ファイルへのテイント伝搬の二つの機能を実装する。

強制的なテイント伝搬機能は、監視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も監視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、マルウェアが動的生成したコードに対してもテイントタグを設定し、監視対象とすることができる。

また、ファイルへのテイント伝搬機能は、監視対象コードがテイントタグが設定されたメモリ上の値をファイルに書き出した場合、そのテイントタグをディスク上のファイルへも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などの行為を行った場合、注入されたファイルが実行されれば、その実行を監視対象とすることができる。

上記の提案方式を、テイント伝搬機能付きのハードウェアエミュレータである Argos-0.5.0 上に実装した。Argos は本来ハニーポットとして設計、実装されており、テイント伝搬機能を脆弱性を狙った攻撃を検知するため利用している。本提案手法では Argos の仮想 CPU 上のテイント伝搬機構を改造し、強制的なテイント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。

提案方式の有効性を示すため、提案方式を実装したシステム上で、マルウェアの挙動を模倣する複数のテストコードと CCC Dataset2012 のマルウェアで実験を行った。この実験の結果、提案手法が従来の PID や TID に基づく監視対象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。

本提案方式を利用することで、マルウェアの動的解析の際の検知漏れや誤り検知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の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、以下の二つの問題があると考える。

1. PID や TID では監視の粒度が粗い
2. 未知のコード注入を追跡できない

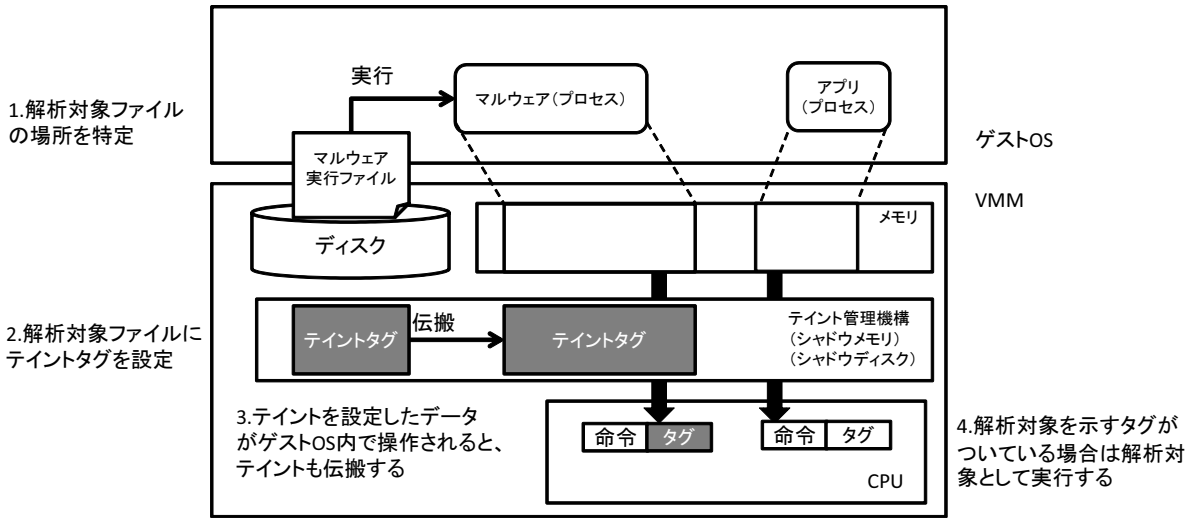

図 1: 概要図と実行例

1 に関して、同一 PID、TID 上に悪意のあるコードと正規のコードが混在するような状況においては両者を正確に区別することができない。例えば、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 に基づく監視対象コードの識別方法では、このような同一 PID、TID 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、CreateRemoteThread のような多くのマルウェアに頻繁に利用される方法でコード注入が行われた場合、この API を予めフックし、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、その PID、TID を更なる解析対象とすることができる。しかしながら、予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、予めすべてのコード注入が行われる経路を把握することは現実的には難しい。例えば、特定のレジストリ経由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予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くため、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。

テイントタグとは、動的なデータフロー解析手法の一つである、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。このテイント解析では、CPU において、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も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。

図 1 に提案手法の概要を示す。本提案手法では、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ことを想定している。まず解析対象であるマルウェア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。この際テイントタグはディスク上に位置する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。このテイント管理機構はVMMの中だけから操作が可能であり、ゲストOS内からは操作することはできない。またその存在もゲストOS内からは確認することはできない。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。CPUが命令をフェッチする際に、同時にシャドウメモリも参照し、フェッチした命令に該当するテイントタグも取得する。この際、フェッチした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、その命令を解析対象として実行する。

以降では、提案手法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強制的なテイント伝搬機構とファイルへのテイント伝搬機構について述べる。

3.1 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問題について述べ、その問題を回避するため、本提案手法で採用している強制的なテイント伝搬機構について述べる。

テイント解析において、テイントタグが付与されているデータに対して以下のコードのような特定の処理を行うとテイントタグが消えてしまう問題が指摘されている [4]。

```
array[0] = 0;
array[1] = 1;
...
array[255] = 255;
...
out = array[in];
```

変数 in に対してテイントタグが設定されていた場合、array[in] といった形で out 変数に値を代入すると、in と out で値に変化はないが、in と out で命令レベルでの直接的な代入関係がないため out 変数へテイントタグが伝搬しない。仮に攻撃者が上記のようなコードを利用して、実行コードを動的生成した場合、コード生成中にテイントタグが消えてしまい、生成されたコードにはテイントタグが設定されていない状況が生まれる。これはつまり、この動的生成されたコードが本提案手法の監視対象から外れてしまう可能性があることを意味している。

そこで、このような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、本提案では監視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、監視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、マルウェアが書きこんだメモリ上の値には監視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監視対象から外れることを防止できる。

一方、監視対象コードがメモリ書き込みを伴うAPIやライブラリを呼び出した場合、タグの伝搬は通常の伝搬機構に任せ、強制的な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。

3.2 ファイルへのテイント伝搬

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、注入するコードを一旦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。本提案手法では、このようなマルウェアを対処するため、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データ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伝搬機構を追加する。

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、その実行も監視対象とすることができる。

またこの機構は逆向きの転送、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、監視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。

4 実装

本章では、提案手法の実装方法について述べる。

本提案手法は、Argos-0.5.0[8] 上に実装を行った。Argos-0.5.0 は Qemu[3] 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る。Argos は本来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。Argos は、仮想 CPU におけるテイントの伝搬機構と、シャドウメモリを持っているため、これらを本提案手法ではそのまま利用した。一方で、Argos はシャドウディスクは実装されていないため、シャドウディスクと、シャドウディスク、シャドウメモリ間のデータ転送部分を新たに実装し、Argos に追加した。

ディスク上のファイルの位置の特定は、ディスクフォレンジックツールである The Sleuth Kit-3.2.3[2] を用いて行った。

5 実験

本章では、提案手法の有効性を示すため行った二つの実験とその結果について述べる。

提案手法の有効性を示すため、次の二つの実験を行った。一つ目は、マルウェアの挙動を模倣したテストコードを作成し、提案手法を実装したシステム上で動かし、正確に実行トレースが取得できるかを実験した。二つ目は CCC Dataset2012 のマルウェア検体のうちアンチウィルス名でユニークな 18 検体と D3M のハッシュ値でユニークな 14 検体の合計 32 検体を、提案手法を実装したシステム上で動かし、正確な命令トレースが取得できるかを実験した。尚、二つ目の実験は正確性を確認するため、コード注入が行われたマルウェアに関してはその結果を手動で調査し、確認した。

これらの実験の目的について述べる。一つ目の実験の目的は提案手法が 2 章で挙げた問題を解決でき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監視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、それら注入方法に依存せずに監視対象コードを追跡可能であり、検知漏れが発生しないことを示す。二つ目の実験では提案手法は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、Dell Precision T7500、Intel Xeon CPU X5670 × 6、12G メモリ、SSD ハードディスク 512G、ホスト OS は Ubuntu Linux 10.10、ゲスト OS は Widnows XP SP2、の環境で行った。。

5.1 実験 1

ここでは、実験 1 の概要とその結果について説明する。実験 1 で使用したテストコードは次に示す五種類のものを利用した。それぞれ異なる方法でコード注入、ファイル感染を行うものである。

1. CreateRemoteThread:WriteProcessMemory、CreateRemoteThread を利用して他プロセスに直接コードを注入する。
2. SetWindowsHookEx:SetWindowsHookEx を利用して他プロセスに DLL を注入する
3. AppInit_DLLs レジストリ:AppInit_DLLs レジストリキーに DLL を登録し、user32.dll をロードしたプロセスに DLL を注入する。
4. BHO:Browser Helper Object として、Internet Explorer に DLL を注入する。
5. ファイル感染:実行ファイルに悪意のあるコードを付加し、PE ヘッダを修正する。

実験 1 の結果を表 1 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監視対象コードの特定、追跡ができた。

5.2 実験 2

ここでは実験 2 の概要とその結果について説明する。ここで使用したマルウェアは、CCC Dataset 2012 のマルウェア検体 10,538 のアンチウィルス名でユニークなものに絞りこんだ 18 検体と、D3M で提供されているマルウェア 14

表 1: 実験 1 の結果

注入方法	注入先	誤検知	検知漏れ
CreateRemoteThread	calc.exe	なし	なし
SetWindowsHookEx	calc.exe	なし	なし
AppInit_DLLs	user32.dll をロードするプロセス	なし	なし
BHO	IEXPLORER.EXE	なし	なし
ファイル感染	calc.exe	なし	なし

検体の合計 32 検体を対象に実験を行った。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。

この 32 検体のうち、22 検体はコード注入のような監視対象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、何かしらの監視対象から逃れるための動作を行った。この結果を表 2 に示す。

493e3...、7aaed... の検体に関しては提案手法で検知漏れが発生した。この原因については以下の通りである。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、例え文字列に監視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 ないで実行される命令列を監視対象とすることはできなかったため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。

[1] 単純な子プロセスの生成は除外した

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[7](Anubis[6]) 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 を結びつけ、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[5] は TEMU[15] 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。TTAanlzye 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[11] は、Xen[17] 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[18] を利用して実装が行われている。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、予めゲスト 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[12] は、予め Windows を解析し、各種データ構造体のシグネチャ [9] を作成する。そのシグネチャを元に、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[10] 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

表 2: 実験 2 の結果

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
5452e...	既存サービスを改ざんして実行	なし	なし

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 章で挙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、テイントの伝搬漏れと誤り伝搬の観点から議論する。

7.1 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についてテイントの伝搬漏れについて議論する。

提案手法は、テイント解析を元に解析対象コードの識別を行っている。[4] では、3.1 章でも述べたとおり、テイント伝搬の伝搬漏れの問題が指摘されている。本提案手法では、強制的なテイント伝搬機構を実装することで伝搬漏れを防いでいる。しかしながら、Windows が提供している API の中で伝搬漏れが発生した場合、強制的なテイント伝搬機構では対処することができない。[5]、[13] でも述べられている通り、暗号や文字コード変換のライブラリの中で、この伝搬漏れが発生するとの報告がある。仮に、攻撃者がこれらのライブラリ関数を使って、動的コード生成を行った場合、監視対象を示すテイントタグが消えてしまう可能性がある。その結果、正確に不正なコードと正規のコードの見分けがつかなくなる可能性がある。この問題を解決するため、例えば問題のあるライブラリ関数を特別扱いし、関数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りこむことで回避できるものと考える。

5.2 でも述べたとおり、解析対象がスクリプトの場合、直接は提案手法を適用することが難しい。これは、基本的にスクリプトは実行コードを生成するため、または実行するコードを決めるために参照されるだけで、このスクリプト自体が直接 CPU で実行されるわけでないため、スクリプトにテイントタグを設定したとしてもそれが実行されるコードにまで伝搬しない。これを回避するために、[16] で提案されている switch-case 文中の参照関係でもテイントタグが伝搬する方法などを検討する必要がある。

7.2 テイントの誤り伝搬

攻撃者が、本提案手法を悪用すると、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。例えば、監視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、値はそのままのため、コードとしては挙動に変化はないが、書き込まれた箇所が監視対象になってしまう。これにより、本来監視対象でないコードを監視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、[14]、[16] 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と考える。

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PID、TID上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] MV Yason, "The Art of Unpacking" in Black Hat Briefings USA, Aug 2007

[2] The Sleuth Kit(TSK) http://www.sleuthkit.org/

[3] F. Bellard, "QEMU, a Fast and Portable Dynamic Translator," in Proceeding of 2005 USENIX Annual Technical Conference, 2005

[4] L.Cavallaro et al., "On the Limits of Information Flow Techniques for Malware Analysis and Containment" in the Proceedings of the GI SIG SIDAR Conference on Detection of Intrusions and Malware & Vulnerability Assessment,2008

[5] H. Yin et al., "Panorama: capturing system-wide information flow for malware detection and analysis," in Proceedings of the 14th ACM conference on Computer and communications security, 2007

[6] Anubis: Analyzing Unknown Binaries, http://anubis.iseclab.org/

[7] U. Bayer et al., "TTAnalyze: A Tool for Analyzing Malware," in Proceedings of the European Institute for Computer Antivirus Research Annual Conference,2006

[8] G. Portokalidis et al., "Argos: an emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation," in Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006, 2006

[9] bugcheck. "GREPEXEC: Grepping Executive Objects from Pool Memory", http://uninformed.org/?v=4&a=2&t=pdf

[10] B. Dolan-Gavitt et al., "Virtuoso: Narrowing the Semantic Gap in Virtual Machine Introspection", in Proceedings of the 2011 IEEE Symposium on Security and Privacy 2011, 2011

[11] A. Dinabug et al., "Ether: Malware Analysis via Hardware Virtualization Extensions", in Proceedings of the 15th ACM conference on Computer and communications security 2008, 2008

[12] X.Jiang et al., "Stealthy Malware Detection Through VMM-Based "Out-of-the-Box" Semantic View Reconstruction", in Journal ACM Transactions on Information and System Security (TISSEC),Volume 13 Issue 2, February 2010

[13] J. Chow et al., "Understanding Data Lifetime via Whole System Simulation", in Proceedings of the 13the conference on USENIX Security Symposium, 2004.

[14] M. G. Kang et al., "DTA++:Dynamic Taint Analysis with Targeted Control-Flow Propagation", in Proceedings of the 18th Annual Network and Distributed System Security Symposium, 2011

[15] D. Song et al., "BitBlaze: A New Approach to Computer Security via Binary Analysis", in Proceedings of the 4th International Conference on Information Systems Security, 2008

[16] A. Slowinska et al., "Pointless Tainting? Evaluating the Practicality of Ponter Tainting", in Proceeding of the 4th ACM European conference on Computer systems, 2009

[17] P. Barham et al., "Xen and the art of virtualization", in Proceedings of 19th ACM Symposium on Operating system principles, 2003

[18] G.Neiger et al., "Intel Virtualization Technology", in Intel Technology Journal(August 2006), vol. 10