仮想マシンモニタにおけるテイント伝搬を用いた不正プログラムのステルス解析

黒米 祐馬 †　　武田 圭史 ‡

† 慶應義塾大学 環境情報学部
252-0882 神奈川県藤沢市遠藤 5322
gomachan@sfc.wide.ad.jp, keiji@sfc.keio.ac.jp

あらまし 仮想マシンモニタからゲスト OS のメモリやデバイスを監視する手法を VM Introspection といい，マルウェアの解析に広く用いられている．しかし，既存の手法にはマルウェアから仮想マシンモニタが検出されうるという問題がある．また，ルートキットを用いてカーネル構造体が改竄された場合やコードインジェクションによって他のプログラムに実行状態が遷移した場合に正しくマルウェアを解析できないことがある．本論文ではゲスト OS を書き換えることなくマルウェアを解析する手法を用い，カーネル構造体とマルウェア本体に紐付けたテイントタグの伝搬からマルウェアの実行状態を正確に取得するシステムを提案する．

Stealth malware analysis by using taint propagation on virtual machine monitor

Yuma Kurogome†　　Keiji Takeda‡

†Faculty of Environment and Information Studies, Keio University
5322 Endo, Fujisawa-shi, Kanagawa 252-0882 Japan
gomachan@sfc.wide.ad.jp, keiji@sfc.keio.ac.jp

Abstract A method of monitoring devices or memory of guest OS from a virtual machine monitor is called VM Introspection, and it is widely used for analysis of malware. However, the existing methods are detectable from the malware. Also, it may not be possible to correctly analyze malware if the execution state transitions to other programs by code injection or if a kernel structure has been tampered with by rootkits. In this paper, we use a malware analysis method that do not rewrite a guest OS and proposes a system to obtain exact execution status of malware from propagation of taint tags associated with files and kernel structures.

1　はじめに

仮想マシンモニタからゲスト OS のメモリやデバイスを監視する VM Introspection は，クラウドコンピューティングサービスにおけるイベントの監視や，IDS やファイアウォールのほか，マルウェアの自動解析に用いられている．急増するマルウェアに人力のみで対処することは不可能であり，マルウェアの自動解析技術はマルウェアへの迅速な対応に欠かせない要素である．

VM Introspection においては，セマンティックギャップが重要な問題となる．ここでのセマンティックギャップとは，ゲスト OS から取得で

きる情報と，仮想マシンモニタから取得できる情報との隔たりを指す語である．例えば，APIやカーネル中のデータ構造をはじめとするゲストOSのセマンティックス情報を直接的に仮想マシンモニタから参照することはできない．

研究の趨勢としては，厳密なセマンティックス情報を取得する必要のない領域に関心が向けられつつある．マルウェアの検出に限ればシステムコールのみの監視で十分である場合が多く，セマンティックスギャップを解消するためのオーバーヘッドを抑えたいPaaSなどのクラウドコンピューティングサービスにおける利用が検討されている．

だが，クライアント環境を対象としたマルウェアを解析するにあたって，セマンティックギャップは依然として看過することのできない問題である．厳密なセマンティックス情報を取得しないVM Introspectionは，コードインジェクションやルートキットといった手法によって改竄された実行結果を取得してしまうためである．

コードインジェクションは他のプロセスに実行状態を遷移させることで，デバッガなどからマルウェアの解析を妨害する．また，近年急増しているMan-in-the-Browser攻撃において主に用いられる．Man-in-the-Browserではマルウェアのコードがブラウザのメモリ空間に挿入され，オンラインバンキングなどに関連した情報の窃取や悪意のウェブサイトへのリダイレクトなどが行われる．これは，DLLの読み込み順序を悪用したり，動的リンクを悪用したりすることによって実現される．

ルートキットは主にドライバを用いてカーネル構造体を改竄し，悪意のあるシステムコールへのリダイレクトやプロセスの隠蔽を行う．悪意のあるシステムコールへのリダイレクトは関数の先頭部分やシステムコールが呼び出される際にが参照される関数ポインタテーブルを書き換えることで実現され，プロセスの隠蔽はプロセス情報を管理する双方向リンクリストから対象のプロセス構造体を除外することで実現される．こうしたルートキットはMicrosoftによるカーネル構造体の保護機能であるPatchGuardの登場によって弱体化した．PatchGuardは，定期的にカーネル空間の完全性を検査し，ルートキットによる改竄が疑われる場合，KeBugCheckExを呼び出すことでマルウェアを抑制する．しかしながら，ルートキットは依然として自動での解析が困難な領域である．

これらの手法は，マルウェアの存在を隠蔽し，セキュリティソフトベンダや研究者によるシグネチャの作成を遅らせるという意図のもと用いられてきたものだが，近年では標的型攻撃における悪用が問題となっている．

いずれもVMMから直接的に取得できない極めてOSに依存したセマンティックス情報を改竄しているため，コードインジェクションやルートキットに対処しうるVM Introspectionを実現するためには，セマンティックギャップを解消しなければならない．

また，既存のVM Introspectionの多くには，コードインジェクションやルートキットを考慮していないほか，マルウェアによって解析環境が検出されうるという問題がある．解析環境を検出したマルウェアは，動作を停止したり，無害であるかのように振る舞うことがある．そのため，マルウェアに検出されないVM Introspectionについても検討する必要がある．

そこで本論文では，VM Introspectionのためのデータフローをゲスト OSの外部で完結させることでマルウェアからの検出を防ぎつつ，テイント解析によってVM Introspectionにおけるセマンティックギャップを解消し，コードインジェクションやルートキットを自動で解析する手法を提案する．

テイント解析とは，タグを結び付けた悪性データの伝搬を追跡する手法である．代入命令などにともなうメモリの参照に応じて，テイントタグを伝搬させることで，あるデータがどのデータによって操作されたかを明らかにすることができる．

2 関連研究

2.1 概要

ここでは，既存のVM Introspectionおよびテイント伝搬にもとづく解析手法について述べ，

その問題点について考察する．

2.2 VM Introspection による解析

TTAnalyze[1] は QEMU 上に実装されたシステムで，ゲスト OS 内に挿入したモジュールを通してプロセスの作成時にセットされる CR3 レジスタの値と PID を結び付け，VMM 内で解析対象コードの識別を行っている．だが，ゲスト OS のユーザー空間とカーネル空間の両方にモジュールを挿入していることから，マルウェアから PsGetCurrentProces などの挙動をもとに検出されうるという問題がある．

Ether[2] は Xen をベースにして実装されたシステムで，ゲストから不可視の trap flag を用いて命令単位でマルウェアを解析する．Ether は事前にゲスト OS を検索することで得た PID，TID をもとに解析対象を識別する．これによって Ether は，ゲスト OS にエージェントを挿入することなく VM Introspection を実現しているが，ルートキットによってシステムコールテーブルが改竄された場合に改竄された実行結果を受け取ってしまう．

EagleEye[3] は Xen と QEMU をベースにして実装されたシステムで，データセンターにおけるセキュリティ監査を目的としている．VM のメモリに CPUID 命令を挿入することで，ゲスト OS にエージェントを挿入することなくシステムコールのフックなどを実現している．また，ページテーブルの read，write ビットを操作することで PatchGuard から検出されるのを防いでいる．EagleEye はゲスト OS に修正を加えることなく，Ether よりも洗練された形で VM Introspection を実現しているものの，マルウェアの解析を目的としていないため，コードインジェクションに対する機能が備わっていない．また，EPTViolation による VM Exit について考慮されていない．

2.3 テイント伝搬にもとづく解析

Renovo[4] は TEMU をベースにして実装されたシステムで，マルウェアを命令単位で解析し，コードインジェクションによってプロセスを越えて動作するマルウェアをテイントの伝搬にもとづいて解析する機能を備えている．だが，セマンティックギャップ解消のためにゲスト OS 内にカーネルモジュールを挿入しているため，マルウェアに検出されうる問題がある．Renovo はエージェントをゲスト OS に挿入している点で VM Introspection としては不完全だが，テイントの伝搬によってコードインジェクションを解析することができることを明らかにした点で優れている．

3 提案手法

3.1 概要

ここでは，前述した従来手法の問題を解決するマルウェアの解析方法を提案する．

VM Introspection のためのエージェントをゲスト OS 内に挿入することは，マルウェアからシステムが検出されうる設計であるということを示している．そのため，VM Introspection のデータフローをゲスト OS 外部で完結させる必要がある．そして，ゲスト OS を書き換えずとも，カーネル構造体などにまつわるセマンティックギャップを解消できなければ，コードインジェクションによって他のプロセスに実行状態を移すマルウェアや，ルートキットによって改竄されたシステムコールを用いるマルウェアを解析するのは困難である．

そこで，テイント解析を導入する．Renovo におけるテイント解析では，解析対象のメモリ領域にしかテイントタグを結び付けていなかった．これのみでもコードインジェクションに対処することは可能だが，本研究ではさらなるセマンティックギャップを解消すべく，これを拡張して OS のデータ構造体にもテイントタグを付加する．本手法では，x86 のページング機構において物理アドレスと仮想アドレスを結びつけるページテーブルエントリ (PTE) および変換ルックアサイドバッファ(TLB) を基準としてテイントを伝搬させる．PTE のビットを操作することでページへの書き込みが検出可能となり，CR3

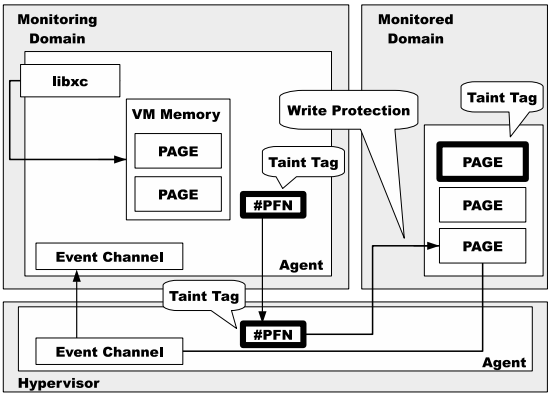

図 1: セマンティックス情報を動的に監視する機構

レジスタと連携して TLB ミスを監視することでコンテキストスイッチが検出可能となる．

3.2 システムの構成

本手法は，Xen の拡張である Ether をベースにしている．Xen は，Intel VT のもとで動作するハイパーバイザと特権ドメインからそれ以外の非特権ドメインを管理する構造となっている．

図 1 にセマンティックス情報を取り扱う仕組みを示す．システムは以下の順序で実行される．

1. libxc というインターフェイスを経由して，特権ドメインから監視する VM のメモリページを読み取るためのリクエストを発行する．
2. ReactOS から作成したシグネチャを参照しつつ，マッピングされたページから非特権ドメインのデータ構造体を検索する．
3. 検索した構造体にテイントタグを結び付ける．
4. ページフレーム番号およびテイントタグを特権ドメインに通知する．
5. PTE の R/W ビットを操作し，ページに書き込み禁止属性を設定する．
6. ページの操作を Event Channnel に通知し，必要ならテイントタグを伝搬させる．
7. テイントの伝搬に応じて，特権ドメインで管理しているデータ構造体の情報を更新する．

これによって，テイントの伝搬にもとづいたセマンティックス情報の動的な監視が実現される．

本手法は識別したセマンティックス情報について，メモリに対応するページとそのオフセットをもとに良性のテイントタグを付加したのち，物理ページフレーム番号単位で管理しており，非ページプールに存在するカーネル構造体についても監視対象である．

これらは，ハイパーコールによる Xen のハイパーバイザと特権ドメインの連携にもとづいている．ゲスト OS 内にエージェントを挿入することがないため，マルウェアに検出されにくい VM Introspection となっている．

本手法では，マルウェアがファイルとして存在し，独自のメモリ空間を使用することを前提としている．プロセスの生成は，CR3 と TLB およびセマンティックス情報として取得したプロセスリストを監視することで行い，メモリのマッピングに応じてページにタグを伝搬させる．解析対象を識別する基準としてテイントの伝搬を用いることによって，コードインジェクションへの対処が可能となる．

ルートキットの検出にあたっても同様に TLB を用いるが，本手法ではセマンティックス情報としてカーネル空間に付加した良性のテイントタグを監視し，マルウェアから伝搬してきた悪性のテイントタグとの衝突を監視することで，ルートキットによるゲスト OS の改竄を確認する．これによって，システムコールやプロセス情報が改竄された場合であっても，正しい実行結果を取得することができる．実行結果は Ether の trap flag を用いて命令単位で取得しているが，VM のメモリに CPUID を埋め込むことでシステムコールについても取得している．

良性のテイントタグは，プロセスを管理する EPROCESS，スレッドを管理する ETHREAD，システムコールへのポインタが格納された配列テーブルである SSDT，割り込みハンドラへの

ポインタが格納されている IDT など，PatchGuard に保護される領域に割り振られる．また，PatchGuard に保護されないオブジェクトのマネージャやヘッダに対しても適用される．これは，ObTypeIndexTable を書き換えることで，PatchGuard を迂回せずプロセスを隠蔽するルートキット手法 [6] を考慮したものである．

4 提案手法の評価

4.1 概要

プロトタイプを実装し，収集したマルウェア検体およびマルウェア対策のための研究用データセット [7] を用いて速度面と機能面での性能評価を行った．プロトタイプでは，非特権ドメインから実行されたマルウェアを 300,000 ミリ秒に渡って監視する．なお，プロトタイプはシングルプロセッサのみをサポートの対象としたが，これは簡略化のためであり，提案手法がプロセッサ数に依存することを意味しない．

4.2 オーバーヘッド

Windows では，ユーザーモードからカーネルモードへと遷移する際にモデル固有レジスタが参照され，SYSENTER 命令が実行される．ここでは，以下のコードを用いて SYSENTER 命令を 100 万回実行することで，ユーザーモードからカーネルモードへのスイッチにかかるオーバーヘッドを検証した．

```
void test(long count)
{
    for (long i=0; i<count; i++)
        FlushProcessWriteBuffers();
}
```

検証にあたっては，以下の環境でプロトタイプを動作させた．なお，シングルプロセッサのみ稼働させている．

解析用ドメイン	Windows 7 Professional
プロセッサ	Core i5-2540M
RAM	4GB

この場合，SYSENTER 命令についてネイティブ環境の 43 倍のオーバーヘッドが確認された．一方で，プロセスの作成やコードインジェクションにともなうコンテキストスイッチでは，実機の 2081 から 4642 倍ものオーバーヘッドが発生する．これは，コードインジェクションによって操作されたメモリ空間に対応する大量のページを処理するためである．

4.3 Win32/PlugX

Win32/PlugX は Man-in-the-Browser 機能を実装した RAT であり，2012 年 3 月に発見された．PlugX はプラグイン機能によって高い拡張性を獲得しており，標的型攻撃での悪用が広く確認されている．PlugX は DLL の読み込み順序を悪用したコードインジェクションをはじめに，svchost や msiexec などのプロセスに対して次々にコードインジェクションを行う．PlugX はコードインジェクションに際して，エンコードされた設定と自身のコードへのポインタなどが含まれる改竄されたヘッダを用いるため，メモリを PE ヘッダのシグネチャで検索してコードインジェクションを検出するといった手法は効果を持たない．本システムは，テイントの伝搬をもとに他のプロセスへのコードインジェクションを監視している．

本システム上で PlugX を実行したところ，複数回のコードインジェクションが確認され，PE ヘッダの改竄に関わらず，解析対象を見失うことなく追跡することができた．一方で，暗号処理については解析することができなかった．これは，タグの伝搬途切れが原因であると推察される．

4.4 WinNT/Turla

WinNT/Turla は標的型攻撃のために開発されたマルウェアとして 2014 年 3 月に発見された．WinNT/Turla はコードインジェクション機能に加えて高度なルートキットを備えており，現状では 64bit 版 Windows8 の PatchGuard を迂回して動作する唯一のマルウェアとして知られ

る．Turla は，KeBugCheckEx をフックし，割り込み要求レベルに応じて RtlCaptureContext を操作することで，PatchGuard によってカーネルの完全性を検証する遅延プロシージャ呼び出しを行わせない．

本システムは，PatchGuard が監視しているシステムコールの関数ポインタを管理するテーブルなどに良性のテイントタグを付加することで，ルートキットによるシステムコール改竄の検出と解析を可能にしている．プロトタイプ上で Turla を実行したところ，非ページプールへコードを隠蔽したり，0xC3 によるフックを挿入したりといった挙動が確認された．一方で，コマンドの受信部や独自に実装された仮想ファイルシステムの操作については解析することができなかった．これについても，タグの伝搬途切れが原因であると考えられる．

4.5 D3M

データセットのうち，D3M に含まれる検体を解析した．なかでも，検体 7a6850... では，コードインジェクションが確認された．検体 7a6850... は，コードインジェクションにおける典型的な API である SetWindowsHookEx や CreateRemoteThread が内部で呼び出している LdrLoadDll と，GetProcAddress が内部で呼び出している LdrGetProcedureAddress を操作する処理や，NtOpenSection で作成したセクションを ZwMapViewOfSection を用いてメモリにマッピングする処理を行う．本検体はこうした機能によって解析を妨害している．また，ブラウザから情報を窃取したり，レジストリを操作して Windows の起動時に自身が起動するよう設定する．これは，PlugX などの RAT に備えられた機能である．挙動から，検体 7a6850... は Zbot 系の亜種であると思われる．

4.6 考察

大幅な速度低下は，非特権ドメインからハイパーバイザを経由して特権ドメインに大量のテイントの伝搬を通知しているためであり，これを改善するためにはテイントタグ管理の効率化について検討しなければならない．

さらに，複数の検体において，タグの伝搬途切れによって正しい解析結果を取得できないという問題が確認された．プロトタイプでは，伝搬のルールとしてデータのコピーと演算を設定していたが，タグと結び付けられたメモリ上のデータをもとに条件分岐を行う場合などに伝搬途切れが発生してしまう．この問題については，タグの強制伝搬 [8] や分岐命令の監視 [9] の面で検討が加えられている．これらの導入にあたっては，テイントの誤伝搬について一層の考慮が必要となる．テイントが関係のない領域に伝搬してしまうと，マルウェアの影響を受けていない領域までマルウェアとして解釈されてしまうためである．

また，プロトタイプはマルウェアを一定時間のみ監視する設計となっているため，ネットワーク越しにコマンドを受け取って動作する RAT などについては，コードカバレッジの面で問題が見受けられる．コマンドを受信しなければ実行されないコードブロックが含まれるマルウェアの機能を単なる自動解析によって網羅することは難しい．プロトタイプでは，300,000 ミリ秒のみ検体を実行していたが，このような場合，攻撃者がコマンドを送信してくるまでの継続的な監視が必要であると考えられる．データセットには検体の通信パケットが含まれているため，これをリプレイすることによって，コマンドと対応するコードブロックを抽出するといった方法が有効ではないかと思われる．

加えて，記号的実行 [10] を適用することで，コマンドと対応するコードブロックを抽出する方法が考えられる．記号的実行とは，代数シンボルでプログラムを式で表現し，その操作を通じてプログラムの挙動を観察する手法である．記号的実行はモデルベーステストの分野で発達してきた技術であり，条件分岐の度に分岐が実行可能か制約充足ソルバに解かせることでテストケースを生成するものが一般的である．つまり，原理的にはプログラムのあらゆる実行パスを通ることができるため，どの入力値でどの実行パスを通るかという制約を抽出する手法とし

て期待されている．動的解析と組み合わせた手法としてCute[11]やKLEE[12]が知られており，ループ文の解釈などで課題が見受けられるものの，記号的実行はコマンドによって挙動を変化させるマルウェアを解析する手法となりうる．

5 まとめ

本論文では，テイントの伝搬を用いてセマンティックス情報を動的に解釈するとともに，コードインジェクションやルートキットを用いるマルウェアを自動で解析する手法を提案した．また，評価用のプロトタイプを作成し，マルウェア対策のための研究用データセットを用いて評価を行った．本手法は，標的型攻撃に用いられるような高度なマルウェアであっても正しく解析することができるため，迅速な挙動の把握および削除に貢献しうる．

一方で，プロトタイプを用いた実験から，実装上の問題が露呈した．現状では，一部のコンテキストスイッチにかかるオーバーヘッド，テイントの伝搬が途切れてしまうこと，コマンドによって挙動を変化させるマルウェアへの対策が不十分であることといった複数の問題を解決できていない．

だが，セマンティックギャップにまつわる問題については，ページにもとづいたテイント解析によってデータ構造体を監視することで解決することができたと言える．また，マルウェアに検出される要因となるエージェントをゲストOSに挿入することなくVM Introspectionを実現できている．

今後は考察で述べた問題に取り組み，さらなる自動解析機能について検討する必要がある．

6 謝辞

This material is partially based upon work supported by Asian Office of Aerospace Research and Development, U.S. Air Force Office of Scientific Research under Award No. FA2386-14-1-4059.

参考文献

[1] Ulrich Bayer, Christopher Kruegel, and Engin Kirda, ”TTAnalyze: A Tool for Analyzing Malware”, 15th European Institute for Computer Antivirus Research Annual Conference, 2006.

[2] A. Dinaburg, P. Royal, M. Sharif, and W. Lee, ”Ether: malware analysis via hardware virtualization extensions,” in ACM CCS, pp. 51-62, 2008.

[3] Wu, Y. S., Sun, P. K., Huang, C. C., Lu, S. J., Lai, S. F., and Chen, Y. Y., ”EagleEye: Towards mandatory security monitoring in virtualized datacenter environment”, Proceedings of the International Conference on Dependable Systems and Networks, 2013.

[4] Min Gyung Kang, Pongsin Poosankam, and Heng Yin, ”Renovo: A Hidden Code Extractor for Packed Executables”, Proceedings of the 5th ACM Workshop on Recurring Malcode, 2007.

[5] ReactOS
http://www.reactos.org

[6] Nikita Tarakanov, ”Exploiting Hardcore Pool Corruptions in Microsoft Windows Kernel”, HackInTheBoxSecConf2013 - Amsterdam , 2013.

[7] 秋山満昭, 神薗雅紀, 松木隆宏, 畑田充弘, ”マルウェア対策のための研究用データセット MWS Datasets 2014 ”, 情報処理学会 研究報告コンピュータセキュリティ（CSEC）Vol. 2014-CSEC-66, No. 19, pp. 1-7, 2014.

[8] 川古谷裕平, 岩村誠, 針生剛男, ”テイント伝搬に基づく解析対象コードの追跡方法”, 情報処理学会論文誌 Vol.54, No.8, pp. 2079-2089, 2013.

[9] 幾世知範, 青木一生, 針生剛男, ”制御フローと通信の関連性分析に基づく C&C サーバ

特定手法の提案”, 電子情報通信学会, (情報通信システムセキュリティ研究会 (ICSS) Vol. ICSS2013-81, No. 502, pp.137-142, 2014.

[10] J. C. King., ”Symbolic Execution and Program Testing”, Communications of the ACM, Vol. 19, Issue. 7,pp. 385-394, 1976.

[11] K. Sen, D. Marinov, and G. Agha., ”Cute: a concolic unit testing engine for c”, Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering, pp. 263-272, 2005.

[12] C. Cadar, D. Dunbar, and D. Engler, ”KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs”, Proceedings of the 8th USENIX conference on Operating systems design and implementation, pp. 209-224, 2008.