マルウェア挙動解析のためのシステムコール実行結果取得法

大月 勇人　　瀧本 栄二　　樫山 武浩　　毛利 公一

立命館大学
525-8577 滋賀県草津市野路東 1-1-1
{yotuki, takimoto, kashiyama, mouri}@asl.cs.ritsumei.ac.jp

あらまし 次々に出現するマルウェアを短時間で解析するには動的解析が有効である. しかし, 近年のマルウェアの多くは動的解析を防ぐ機能を持つ. そこで, OS よりも下位層で動作する仮想計算機モニタ BitVisor 内へ解析のための拡張機能 Alkanet を開発している. Alkanet は, ゲスト OS 上のプロセスやスレッドから発行されるシステムコールをフックし, システムコールの種類と引数に加え, その処理結果の取得を可能とする. これによって, マルウェアの挙動をより詳細に解析可能になった. また, 取得したシステムコール履歴から, さらに具体的なマルウェアの挙動の抽出し, 解析レポートの出力を試みた結果について報告する.

A Method to Get Result of System Calls for Malware Analysis

Yuto Otsuki　　Eiji Takimoto　　Takehiro Kashiyama　　Koichi Mouri

Ritsumeikan University
1-1-1 Nojihigashi, Kusatsu, Shiga 525-8577 Japan
{yotuki, takimoto, kashiyama, mouri}@asl.cs.ritsumei.ac.jp

Abstract Recent malwares are applied anti-debugging techniques not to be analyzed by dynamic analysis tools. We are developing “Alkanet” that is an extension for malware analysis in virtual machine monitor. Virtual machine monitor runs higher privilege level than malware. Therefore, malware’s anti-debugging techniques are ineffective against Alkanet. Alkanet monitors behavior of malwares by a system call invoked by processes or threads on guest OS. The behavior of malwares is analyzed by getting result and arguments of the system calls. Furthermore, Alkanet extracts details of malware behavior from the system call log.

1　背景

近年, マルウェアの脅威が問題となっている. マルウェア対策には, マルウェアを解析し, どのような挙動をするかを調査する必要がある. しかし, マルウェアは, 新種や亜種が次々に出現するため, 1 体のマルウェアの解析に時間を費やすことができない. このような場合, 実際にマルウェアを実行し, その挙動を追跡することにより, 比較的短時間で解析できる動的解析が有効である. しかし, 最近のマルウェアの多くは, アンチデバッグと呼ばれる機能を持つ [1, 2]. これは, マルウェア自身が動的解析されていることを検知し, 実行の停止や解析の妨害などを行うものである. アンチデバッグにはさまざまな手法があり, 全てを回避して解析することは一般に困難である. そこで, OS よりも下位のレイヤで動作する仮想計算機モニタ (VMM) を利用したマルウェア解析機構 Alkanet を開発している [3]. 本論文では, 特に Alkanet のシステムコール情報取得機能に, システムコールの処理結果

取得機能を追加したので, その詳細について述べる. また, 得られた情報を基にマルウェアを解析した結果についても述べる.

2 挙動解析に必要な情報

マルウェアの挙動を把握するためには, その挙動の意図の理解しやすさの点から, 機械語レベルよりも API レベルのトレースが有効である. また, ユーザモードのマルウェアがシステムに影響を及ぼす動作を行うためには, システムコールを発行する必要がある. そこで, システムコールのトレースを行うことで, マルウェアの挙動を追跡する.

システムコールの発行元は, スレッドレベルで区別する必要がある. これは, 本論文で対象とする Windows において, 実行単位はスレッドであり, プロセスはスレッドのコンテナであるためである. また, マルウェアによるコードインジェクションによって, 通常のプロセスの中にも「悪意あるスレッド」が存在する場合がある. したがって, システムコール発行元をスレッドレベルで区別する必要がある. 具体的には, プロセス ID(PID), スレッド ID(TID), イメージ名を取得する. なお, Windows では PID と TID の組を Cid と呼ぶ.

さらに, マルウェアの挙動を調査するためには, マルウェアがアクセスしたファイルやレジストリなどの情報が必要になる. これらの情報は, システムコールの引数およびその実行結果から取得できる. しかし, 引数や戻り値には, ポインタや OS 固有のデータ構造が用いられることが多く, その値だけでは不十分な場合がある. したがって, これらのデータ構造を解釈し, 必要な情報を補う必要がある. 例えば, Windows ではファイルやレジストリなどのリソースにアクセスする場合, ハンドルというデータ構造を用いてリソースを指定する. ハンドルは, 各プロセス固有のものであるため, その値だけ取得してもどのリソースか識別できない. したがって, ハンドルが示すリソースの情報も合わせて取得する.

以上から, システムコールのトレースを実現するには, システムコールを発行したスレッドやシステムコールに与えられた引数などの情報が必要である. 具体的には以下の情報になる.

- システムコール発行元の Cid とイメージ名
- システムコール番号と引数
- システムコールの戻り値
- 固有のデータ構造に対する補足情報

3 Alkanet

3.1 概要

Alkanet では, マルウェアの動的解析を行うために, VMM を利用する. これは, アンチデバッグを回避するためには, マルウェアよりも高い権限が必要なためである. 技術的には, OS 内部からの監視・解析でも実現可能であるが, マルウェアの主なターゲットである Windows は, プロプライエタリソフトウェアであるため, このような手段を取ることが難しい. また, Windows 内部でプロセスやドライバを動作させると, これらをマルウェアに検知される恐れがある. よって, OS よりも下位層で動作する VMM で実現するのが適切である.

システムコールのトレースを実現するには, システムコールが発行される毎にフックし, システムコールの引数や OS 内部のデータ構造を解釈する必要がある. しかし, VMM から OS の API を使用できない. すなわち, OS レベルの情報が取得できないという問題がある [4]. そこで, Alkanet は, Windows が使用するメモリ領域を参照し, 独自に Windows のデータ構造の解釈を行う.

Alkanet は, Windows 上で発行されたすべてのシステムコールのトレースを行う. そして, トレースログを取得後に分析し, マルウェアの特徴的な動作を抽出したレポートを出力する.

3.2 構成

Alkanet の全体構成を図 1 に示す. Alkanet は, VMM である BitVisor[5] の拡張機能として実装している. BitVisor は, ホスト OS を必

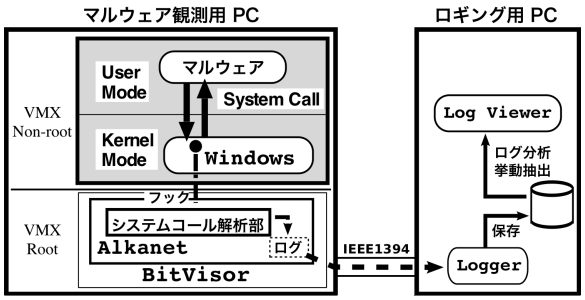

図 1: Alkanet の全体構成

要とせず, ハードウェア上で直接動作するハイパーバイザ型の VMM である. Intel 製 CPU の仮想化支援機能である Intel VT を利用しており, Windows を修正なしで実行することができる.

マルウェアの実行環境であるゲスト OS には, 32bit 版 Windows XP SP3 を用いる. この環境におけるシステムコールは, 通常 sysenter によってカーネルモードへ入り, sysexit によってユーザモードへ復帰する. システムコールのフックは, これらの命令にハードウェアブレイクポイントを設定することで実現している. システムコールをフックすることで, その種類や引数を取得し, ログに保存する. 詳細は 4 章で述べる.

Alkanet のログをマルウェアに検知・妨害されずに保存するために, IEEE 1394 として標準化されている高速シリアルバスを用いる. この IEEE 1394 は, 接続先 PC の物理メモリを DMA(Direct Memory Access) で読み書きできる. これを利用し, ロギング用 PC から Alkanet のメモリ領域内に保持されたログを取得する. また, 取得後, 取得したログを分析し, マルウェアの挙動を示すレポートを出力する.

4 システムコールの結果の取得

4.1 sysexit のフックの必要性と方法

前回の Alkanet の研究報告 [3] では, sysenter のフックのみ行っていた. しかし, sysenter のフックだけでは, システムコールの実行結果や戻り値が取得できない. そのため, そのシステムコールが成功したかどうかや, 新たに生成されたオブジェクトの情報が取得できないといった問題がある. したがって, システムコールの結果や戻り値を取得するために, sysexit のフックが必要である.

sysexit を実行する nt!KiSystemCallExit2 に, ハードウェアブレイクポイントを設定し, フックを行う. このシンボルは公開されているため, アドレスを取得できる. ハードウェアブレイクポイントは, デバッグレジスタで指定したアドレスの命令実行などによって, デバッグ例外を発生させる. これにより VMM へ処理を移すことができる. VMM から制御を戻す際には, EFLAGS の RF(Resume flag) をセットする. これにより, ブレイクポイントが 1 回無効になるため, 処理が継続される.

4.2 システムコールの特定

sysenter と sysexit のフックは個別に行う. また, システムコールの処理結果は, 引数に渡されたポインタによって返される. そのため, 戻り値や処理結果の情報を取得するために, sysexit 時でも何のシステムコールが実行されたか特定する必要がある.

Windows では, システムコール発行時にシステムコールの番号を EAX に格納する. したがって, sysenter のフックでは, EAX からシステムコールの番号が取得できる. 一方で, sysexit のフックでは, どのシステムコールに対応したものかわからない. よって, 通常システムコールは ntdll.dll に実装されているスタブを用いることを利用して, 区別することにした. スタブは, システムコールと同名のラベルが付けられているため, ユーザモードスタックに格納された戻りアドレスを調べることで, 発行されたシステムコールが特定できる.

4.3 戻り値と引数の取得

システムコールの戻り値は NTSTATUS と呼ばれる状態を示す値である. この値から, システムコールが成功か失敗か, あるいは失敗した

場合の理由がわかる．また, 引数にポインタを取り, そのポインタの示すアドレスに結果を返すものも存在する．したがって, システムコールの処理結果を得るためには, 戻り値や引数を取得する必要がある．

Windows の API では, stdcall 呼出規約 [6] が用いられる．stdcall では, 戻り値は EAX に, 引数はスタックに格納される．したがって, 戻り値は EAX から, 引数はユーザモード時のスタックから取得できる．また, sysexit は, 実行時に ECX の値を ESP にロードする．したがって, sysexit のフックの時点では, ユーザモードスタックの位置は ECX に保持されているため, ここから取得できる．

5 評価と考察

sysexit のフックがマルウェア解析に有効であることを確認するために, 実際に Alkanet を用いてマルウェアの挙動解析を行った．論文 [7] を参考にマルウェア起動から 2 分程度実行し, 動作している全てのプロセスのシステムコールのトレースを行った．なお, ネットワークには接続していない．検体として, CCC DATAset 2011[8] の中で活動が記録されているマルウェアを用いた．ここでは, Polipos.exe と呼称する．

図 2, 3, 4 に Polipos.exe を実行して得たシステムコールのログの一部を示す．また, 図 5, 6 は, ログを分析した結果の一部を示す．

図 2 では, PID 54c の Polipos.exe プロセスが, NtCreateProcessEx を用いてもう一つ Polipos.exe プロセスを起動している．sysexit のログから, このシステムコールは成功し, このとき作成されたプロセスの PID は bc であることがわかる．

図 3 では, Polipos.exe が, NtCreateThread を発行し, 別のプロセスの explorer.exe に対してスレッドを作成している．これはコードインジェクションの挙動である．sysexit のログから, explorer.exe 内に TID 1e8 のスレッドが作成されたことがわかる．この後, このスレッドは Polipos.exe を複製する挙動やネットワークへの接続を試みる挙動などを示した．sysenter

```
No. :    5786
Time:    677232506
Type:    sysenter
Ret :    - (-)
SNo.:    30 (NtCreateProcessEx)
Cid :    54c.6cc
Name:    Polipos.exe
Note:    \...\My Documents\Polipos.exe

No. :    5787
Time:    677233114
Type:    sysexit
Ret :    0 (STATUS_SUCCESS)
SNo.:    30 (NtCreateProcessEx)
Cid :    54c.6cc
Name:    Polipos.exe
Note:    PID: bc, ProcessName: Polipos.exe
```

図 2: プロセス生成

```
No. :    6339
Time:    689820849
Type:    sysenter
Ret :    - (-)
SNo.:    35 (NtCreateThread)
Cid :    bc.304
Name:    Polipos.exe
Note:    PID: b0, ProcessName: explorer.exe

No. :    6340
Time:    689820959
Type:    sysexit
Ret :    0 (STATUS_SUCCESS)
SNo.:    35 (NtCreateThread)
Cid :    bc.304
Name:    Polipos.exe
Note:    Cid: b0.1e8, ProcessName: explorer.exe
```

図 3: explorer.exe へのコードインジェクション

のフックのみでは, 作成されたプロセスやスレッドの ID がわからず, その後の挙動の正確な追跡が困難であった．sysexit のフックの実装により, こういった情報の取得が可能となった．

図 4 では, Polipos.exe は, NtCreateThread を用いて, System プロセスに対してスレッドの作成を試みている．しかし, sysexit のログから, 戻り値が STATUS_INVALID_HANDLE となっており, このシステムコールは失敗していることがわかる．この場合, 実際にスレッドは作成されていないため, この後の System プロセスの挙動を分析する必要はない．このように, システムコールの成功・失敗の区別が解析に活

```
No. :    6383
Time:    691816271
Type:    sysenter
Ret :    - (-)
SNo.:    35 (NtCreateThread)
Cid :    bc.304
Name:    Polipos.exe
Note:    PID: 4, ProcessName: System

No. :    6384
Time:    691816285
Type:    sysexit
Ret :    c0000008 (STATUS_INVALID_HANDLE)
SNo.:    35 (NtCreateThread)
Cid :    bc.304
Name:    Polipos.exe
Note:    PID: 4, ProcessName: System
```

図 4: System へのコードインジェクション

用できる.

Polipos.exe は, 他にも svchost.exe, service.exe, winlogon.exe, alg.exe, rundll32.exe, sqlservr.exe, lsass.exe などに対して, スレッドを作成している. 図 5 は, システムコールのログを分析することで, コードインジェクションによって作成されたスレッドから派生するスレッドを追跡し, 階層構造状に表わしたものである. ここでは, Polipos.exe の Cid 54c.18c スレッド によって, PID 480 の svchost.exe に Cid 480.2c4 のスレッドが作成されている. この Cid 480.2c4 スレッドは, Cid 480.22c のスレッドを作成する. この Cid 480.22c のスレッドは, Cid 480.38c, Cid 480.360, Cid 480.720, Cid 480.24c など, 多くのスレッドを作成している.

Cid 480.720 のスレッドを見ると, rundll32.exe に対してコードインジェクションを行い, Cid 220.7f8 のスレッドを作成している. さらに, この Cid 220.7f8 のスレッドから派生するスレッドも追跡できている. 同様に, svchost.exe の Cid 480.22c スレッドは, explorer.exe や alg.exe に対してスレッドを作成する挙動を示した.

このように, sysexit のフックにより, システムコールの結果として生成されたスレッドの情報の取得が可能となった. これにより, コードインジェクションによって作成されたスレッドや, そのスレッドから派生するスレッドやプロセスまで正確に追跡できる. したがって, 「悪意あるスレッド」と正常なスレッドの挙動を明確に区別できる.

さらに, 図 5 内で作成されている Cid 480.41c のスレッドついて分析すると, 図 6 のように drwebase.vdb, avg.avi, vs.vsn, anti-vir.dat といったファイルのオープンを試みている. しかし, 戻り値から実際にはファイルが存在しないことがわかる. また, コードインジェクションによって作成された他のスレッドも, これらを開こうとしていた. これらはアンチウィルスソフトが使用するファイルである. これは, マルウェアが自身を検出・駆除されないように, これらのファイルを削除する挙動の一部であると考えられる. 戻り値の取得が可能になったことで, このような挙動も解析できている.

6 まとめ

本論文では, 仮想計算機モニタを用いて, システムコールトレースを実現する “Alkanet” に sysexit のフックを追加した結果について述べた. sysexit のフックにより, システムコールが成功したか失敗したかということや, ファイルの有無, 生成されたプロセスやスレッドの ID などが取得できる. これらの情報はマルウェアの挙動の分析・抽出に有用である. 実際に, マルウェアが正常なプロセスに対して, コードインジェクションを行った場合でも, 通常のスレッド区別し, 悪意あるスレッドを正確に追跡できることを示した.

今後の課題として, カーネルモードマルウェアへの対応が挙げられる. このようなマルウェアは, カーネルの関数を直接利用できることや, カーネルのメモリ領域にもアクセスできることなどから, 現在の実装では解析が困難である.

参考文献

[1] N. Falliere: “Windows Anti-Debug Reference,” http://www.symantec.com/connect/articles/windows-anti-debug-reference (2007).

```
No. [5212, 5213]: Polipos.exe (Cid: 54c.18c) -> svchost.exe (Cid: 480.2c4) (Code Injection)
   No. [5288, 5289]: svchost.exe (Cid: 480.2c4) -> svchost.exe (Cid: 480.22c)
     No. [5959, 5960]: svchost.exe (Cid: 480.22c) -> svchost.exe (Cid: 480.38c)
     No. [6392, 6393]: svchost.exe (Cid: 480.22c) -> svchost.exe (Cid: 480.360)
     No. [11340, 11341]: svchost.exe (Cid: 480.22c) -> svchost.exe (Cid: 480.720)
       No. [14368, 14369]: svchost.exe (Cid: 480.720) -> rundll32.exe (Cid: 220.7f8) (Code Injection)
         No. [14546, 14547]: rundll32.exe (Cid: 220.7f8) -> rundll32.exe (Cid: 220.488)
         ...
     No. [11844, 11845]: svchost.exe (Cid: 480.22c) -> svchost.exe (Cid: 480.24c)
       No. [15080, 15081]: svchost.exe (Cid: 480.24c) -> alg.exe (Cid: 34c.1c8) (Code Injection)
         No. [15240, 15241]: alg.exe (Cid: 34c.1c8) -> alg.exe (Cid: 34c.5ac)
         ...
     No. [13214, 13215]: svchost.exe (Cid: 480.22c) -> svchost.exe (Cid: 480.7e0)
       No. [16586, 16587]: svchost.exe (Cid: 480.7e0) -> explorer.exe (Cid: 538.510) (Code Injection)
         No. [16744, 16745]: explorer.exe (Cid: 538.510) -> explorer.exe (Cid: 538.6ac)
         ...
     No. [13802, 13803]: svchost.exe (Cid: 480.22c) -> svchost.exe (Cid: 480.308)
     No. [14422, 14423]: svchost.exe (Cid: 480.22c) -> svchost.exe (Cid: 480.2d0)
     No. [14424, 14425]: svchost.exe (Cid: 480.22c) -> svchost.exe (Cid: 480.a0)
     No. [15144, 15145]: svchost.exe (Cid: 480.22c) -> svchost.exe (Cid: 480.41c)
     ...
```

図 5: コードインジェクションされた svchost.exe から派生するスレッド

```
svchost.exe (Cid: 480.41c)
  ...
  [NOT FOUND] No. [15246, 15247]: NtOpenFile \??\c:\program files\netmeeting\drwebase.vdb
  [NOT FOUND] No. [15248, 15249]: NtOpenFile \??\c:\program files\netmeeting\avg.avi
  [NOT FOUND] No. [15250, 15251]: NtOpenFile \??\c:\program files\netmeeting\vs.vsn
  [NOT FOUND] No. [15252, 15253]: NtOpenFile \??\c:\program files\netmeeting\anti-vir.dat
  ...
  [NOT FOUND] No. [15270, 15271]: NtOpenFile \??\c:\program files\netmeeting\avgqt.dat
  [NOT FOUND] No. [15272, 15273]: NtOpenFile \??\c:\program files\netmeeting\lguard.vps
```

図 6: アンチウィルスの使用するファイルの存在確認

[2] M. V. Yason: “The Art of Unpacking,” In Black Hat USA 2007 (2007).

[3] 大月, 毛利: “仮想計算機モニタによるマルウェアの監視,” コンピュータセキュリティシンポジウム 2010(CSS2010) 論文集, pp. 225–230 (2010).

[4] K. Nance, M. Bishop and B. Hay: “Virtual Machine Introspection: Observation or Interference?,” Security & Privacy, IEEE, Vol. 6, No. 5, pp. 32–37 (2008).

[5] 品川 他: “準パススルー型仮想マシンモニタ BitVisor の設計と実装,” IPSJ SIG Notes, Vol. 2008, No. 77, pp. 69–76 (2008).

[6] Microsoft Corporation: “__stdcall,” http://msdn.microsoft.com/en-us/library/zxk0tw93.aspx (2011).

[7] 青木 他: “動的解析における検体動作時間に関する検討,” コンピュータセキュリティシンポジウム 2010(CSS2010) 論文集, pp. 543–548 (2010).

[8] 畑田 他: “マルウェア対策のための研究用データセット ～MWS 2011 Datasets～,” マルウェア対策研究人材育成ワークショップ 2011 (MWS 2011) (2010).