Windows 10 x64 環境を対象とするシステムコールトレーサの実現手法

大月 勇人† 中野 進† 明田 修平† 瀧本 栄二†
齋藤 彰一‡ 毛利 公一†

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

‡名古屋工業大学
466-8555 愛知県名古屋市昭和区御器所町
shoichi@nitech.ac.jp

あらまし Windows XP は多くのマルウェア解析システムで用いられてきたが，現在では新しい Windows へシェアが移っている．新たに出現するマルウェアの動作を解析するためには，まずはマルウェア解析環境を新しい Windows へ対応させることが必要となる．我々が開発しているマルウェア解析システム Alkanet では，Windows 7 x64 への移行が進められている．さらに，最新の Windows 10 x64 でもシステムコールトレースを実現可能か検証を行った．本論文では，Windows 10 x64 環境を対象するシステムコールトレーサの実装手法について述べる．

Implementation of System Call Tracer for Windows 10 x64

Yuto Otsuki† Susumu Nakano† Shuhei Aketa† Eiji Takimoto†
Shoichi Saito‡ Koichi Mouri†

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

‡Nagoya Institute of Technology
Gokiso-cho, Showa-ku, Nagoya, Aichi, 466-8555 Japan
shoichi@nitech.ac.jp

Abstract Windows XP has been used by many dynamic malware analysis systems. However, today, new versions of Windows are grabbing market share from Windows XP. To observe malware newly emerged, we first need to develop new dynamic malware analysis system which uses new Windows as the malware execution environment. We has developed Alkanet, a system call tracer for malware analysis. Currentlly, we are developing new Alkanet for Windows 7 x64. In addition, we confirmed that our analysis method can be used for Windows 10 x64. In this paper, we describe the implementation of system call tracer for Windows 10 x64.

1 はじめに

近年，マルウェアの脅威が問題となっており，その対策が急務である．マルウェア対策のためには，マルウェアが持つ機能や挙動を正確に解析することが重要である．我々は，マルウェアを実際に実行して挙動を観察する動的解析をベー

スとして，短時間でより精度の高い解析が可能なマルウェア解析技術の確立を目指している．また，実際に仮想計算機モニタ (VMM) をベースとするシステムコールトレーサ Alkanet [1] を開発し，動的解析を妨害するアンチデバッグ機能を持つマルウェアをスレッド単位で短時間に解析可能とするなどの成果を得た．

マルウェア解析システムの多くでマルウェア動作環境として用いられてきた Windows XP は，日本時間の 2014 年 4 月 9 日にサポートが終了した．それに伴い，Windows 7，Windows 8.1 といった新しい Windows へシェアが移っている [2]．また，2014 年に攻撃に用いられた RAT (Remote Administration Tools) の内訳では，2010 年前後から存在する Poison Ivy が前年と比べて減少し，比較的新しい PlugX や Emdivi が増加している [3]．Poison Ivy が減少した理由の 1 つに，Windows 7 以降の OS を攻撃対象とする場合に修正が必要なことが挙げられる．このように OS のシェアの変化に伴い，使用されるマルウェアも変化している．

以上から，新しいマルウェアを解析するためには，マルウェアの動的環境も新しい Windows にあわせて更新する必要がある．そこで，2015 年 7 月 29 日にリリースされた Windows 10 をマルウェア動作環境とするシステムコールトレーサの実現を目指し，x64 版の Windows 10 Pro Insider Preview (build 10074) 環境に対応する Alkanet のプロトタイプを実現した．本プロトタイプでは，基本機能として，想定環境で動作する 64 ビットネイティブアプリケーションが発行するシステムコールをトレースする機能を持つ．また，x64 版 Windows では，WOW64 と呼ばれるエミュレーションレイヤにより，32 ビットアプリケーションも動作可能である．そこで，本プロトタイプでは，WOW64 上で動作する 32 ビットアプリケーションが発行するシステムコールをトレース可能にする拡張も施している．以下，本論文では，WOW64 上で動作するアプリケーションから発行されるシステムコールを WOW64 システムコールと呼称する．

以下，本論文では，2 章で Alkanet の概要と構成について述べる．3 章では，Alkanet の機能のうち本プロトタイプ実装時に再実装が必要となった機能について述べ，4 章で 64 ビットアプリケーションが発行するシステムコールをトレース可能とする基本機能について述べ，5 章ではさらに WOW64 システムコールをトレース可能とする拡張機能について述べる．6 章では，プロトタイプを用いて行った基礎評価の結果について述べる．さらに，7 章で関連研究について述べ，8 章で本稿をまとめる．

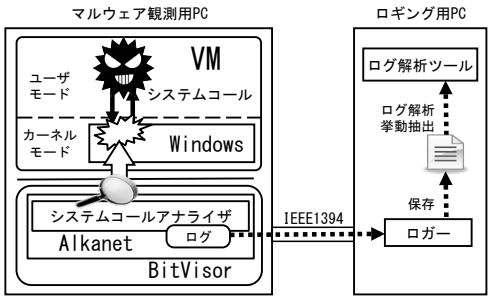

図 1: Alkanet の構成

2 Alkanet

Alkanet [1] は，VMM である BitVisor [4] をベースとするマルウェア動的解析システムである (図 1 参照)．Alkanet は，マルウェアの多くが攻撃対象としている Windows のシステムコールをトレースすることができる．システムコールを観測することにより，マルウェアの挙動を機能単位で抽出し，挙動の理解を容易にしている．システムコールのフックは，システムコールの出入口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．フック後は，レジスタやメモリの内容を解析してシステムコールの種類や引数を取得し，ログに保存する．Alkanet が生成したログはロギング用 PC から IEEE 1394 を用いて取得し，ログ解析ツールを用いて特徴的な挙動を抽出したレポートを得ることができる．また，Alkanet は，VMM ベースとすることで，多くのマルウェアに搭載されているアンチデバッグ機能を回避できるという特徴を有している．

3 他の Windows への移行時に再実装が必要な機能

Alkanet では，マルウェアを動作させる環境として x86 版 Windows XP Service Pack 3 (以下，Windows XP) を用いている．また，我々は，これまでに x64 版 Windows 7 Service Pack 1 (以下，Windows 7 x64) をマルウェア動作環境とするバージョンを作成してきた．本論文では，さらに x64 版 Windows 10 (以下，Windows 10 x64) 環境を対象とする Alkanet を実現する．本章では，Alkanet の機能のうち，マルウェア解析環境を他の Windows に移行する場合に，移行先となる Windows に合わせて再実装が必要となる機能について述べる．以下，本論文では，Alkanet の Windows XP，Windows 7 x64，Windows 10 x64 向けの実装をそれぞれ AlkanetXP，Alkanet7，Alkanet10 と呼称する．

Alkanet を異なるバージョンの Windows に対応させる際に追加や変更が必要な機能は，以下の 4 つである [1].

(1) システムコールのフック
(2) システムコールの特定
(3) プロセスとスレッドの特定
(4) 引数と戻り値の取得

上記 (1)，(3)，(4) については，Windows のバージョンではなく，x86 や x64 といったアーキテクチャに依存する点が多い [5]．したがって，これらについては，x86 版 Windows では AlkanetXP，x64 版 Windows では Alkanet7 の実装がそれぞれ流用できる．

上記 (2) については，Windows ではサービスパック単位でシステムコールの追加や変更などが行われるため，Windows のバージョンごとにシステムコールの番号やスタブの情報などが必要である．システムコールの番号は，文献 [5] に調査方法が記述されており，スタブについてはシンボル情報から取得することが可能である．また，(3) については，Windows のデータ構造である PCR （Processor Control Region）やスレッドオブジェクトなどを利用する．これらについては，Windows のバージョンごとに各種データ構造を解釈する機能の実装が必要である．(4) についても同様に，監視対象のシステムコールに対してそれぞれ個別の処理と，引数に応じたデータ構造解釈機能が必要となる．ただし，Alkanet のデータ構造解釈機能の一部はシンボル情報を基に生成しているため，軽微な差があれば吸収が可能である．

また，x64 版 Windows は，32 ビットアプリケーションを動作させるために，WOW64 と呼ばれるエミュレーションレイヤを持つ．WOW64 上で動作するアプリケーションから発行されるシステムコールは，通常のシステムコールとは発行に至るまでの処理の流れが異なる場合がある．WOW64 上で動作するマルウェアを解析可能とするためには，WOW64 システムコールもトレース可能とする必要がある．したがって，WOW64 レイヤの実装を Windows のバージョンごとに確認し，それにあわせて (1)，(2)，(4) を拡張する必要がある．

4 Alkanet10 の基本機能

本章では，Alkanet10 の機能のうち，64 ビットアプリケーションが発行するシステムコールをトレースする基本機能の実装について述べる．Alkanet10 では， Windows 10 x64 環境を対象とするため，Alkanet7 で用いた手法をベースに 3 章で述べた 4 つの機能を実現している．

4.1 システムコールのフック

x64 版 Windows のシステムコールは，syscall 命令を用いて KiSystemCall64 に遷移し，KiSystemServiceExit 内の sysret 命令により復帰する．Alkanet10 では，それぞれ syscall フック， sysret フックとしてブレイクポイントを設定する．図 2 に Alkanet10 における 2 つのフックポイントを示す．KiSystemCall64 のアドレスは，syscall 命令のジャンプ先を設定する IA32_LSTAR MSR から取得可能である．KiSystemServiceExit についても，図 2 のように KiSystemCall64 のアドレスとシンボル情報から得られる相対アドレス (RVA) から計算できる．

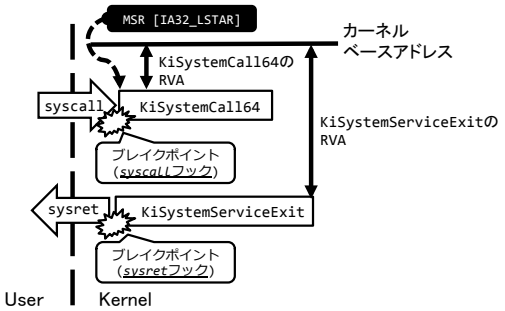

図 2: syscall フックと sysret フック

4.2 システムコールの特定

Windows では，EAX レジスタ [1] にシステムコール番号を設定することで，呼び出すシステムコールを指定する．システムコール番号は，Windows のバージョンによって異なることがあるため，予め Windows 7 x64 と Windows 10 x64 のシステムコールテーブルの比較調査を行った．その結果，両者のテーブルのデータ構造に差異は見られなかった．したがって，システムコール番号とシステムコールの対応付けは，文献 [5] と同じ方法で行った．

上記の通り syscall フックでは，EAX レジスタからシステムコール番号を取得することができる．一方，sysret 実行時には，RAX レジスタはシステムコールの戻り値を保持しているため，sysret フックでは，他の方法でシステムコール番号を取得する必要がある．そこで，sysret フックでは，(a) スレッドオブジェクト内の SystemCallNumber メンバから取得する方法と，(b) 戻り先スタブ内の命令を読み取って取得する方法の 2 つを併用し，システムコール番号を取得する．方法 (a) では，Windows 10 x64 のスレッドオブジェクトが，Windows 7 x64 のものと同様に，直前に発行されたシステムコール番号を保持するメンバを持つことを用いる．したがって，sysret フックでは当該メンバからシステムコール番号を取得することが可能である．ただし，カーネルからユーザモードへのコールバックが発生した場合，上記のメンバが示すシステムコールは，実際の戻り先に対応するシステムコールと異なる場合がある．

方法 (b) では，システムコールは通常 Windows が提供するシステムコールのスタブから発行されることと，当該スタブの構造が一定であることを利用し，戻り先のスタブ内の命令からシステムコール番号を取得する．具体的には，スタブ内には， EAX レジスタにシステムコール番号を設定する mov 命令が存在する．したがって，戻り先をスキャンし，上記の命令を検出できれば，システムコール番号が取得できる．しかし，未知のスタブが用いられた場合やマルウェアが直接システムコールを実行した場合，正しいシステムコール番号を取得できない．

方法 (a) と方法 (b) で取得した番号が異なる場合，その原因として前述のコールバックと未知のスタブが考えられる．コールバックは，方法 (a) で取得した番号や戻り先アドレスから検出でき，未知のスタブは戻り先の命令列を検査することで検出できる．Alkanet では，前者の場合は方法 (b)，後者の場合は方法 (a) の番号を採用する．前述の原因が両方同時に発生する場合については，今後の検討課題としている．

4.3 プロセスとスレッドの特定

Alkanet では，システムコールを発行したスレッドの情報として，プロセス ID とスレッド ID の組である Cid と，実行ファイル名を取得する．Alkanet は，それらの取得にあたって Windows の PCR と呼ばれるデータ構造を用いる．PCR は，現在動作中のスレッドの情報を示すスレッドオブジェクトへのポインタを保持しており，スレッドオブジェクトは，Cid と自身が属するプロセスオブジェクトへのポインタを保持している．実行ファイル名は，そのプロセスオブジェクトから取得できる．

x64 版 Windows における PCR は，GS セグメントの先頭に存在する．x64 アーキテクチャでは，GS セグメントは IA32_GS_BASE MSR によって示される．また，カーネルモードとユーザモードの両者における GS セグメントを保持するため，IA32_KERNEL_GS_BASE という退避用の MSR を持つ．これらの MSR の値を交換する swapgs 命令も用意されている．x64 で

[1] x86，x64 ともに，システムコール番号は 32 ビットの値であるため，EAX レジスタを用いる．

動作する Windows は，カーネルモードの出入り口で swapgs 命令を実行し，PCR の存在するアドレスを IA32_GS_BASE にロードする．

調査の結果，Windows 10 x64 でも上記の仕様や動作に変更はないことを確認した．また，PCR そのものやスレッドオブジェクト，プロセスオブジェクトなどのデータ構造を解釈する機能についても，これらのデータ構造に Windows 10 x64 と Windows 7 x64 との間で大きな差異はなく，シンボル情報を元に再定義することで対応可能である．

4.4 引数と戻り値の取得

x64 版 Windows では，関数呼出しに x64 呼出規約 [6] を用いる．x64 呼出規約では，第 1 引数から第 4 引数は RCX，RDX，R8，R9 レジスタに，第 5 引数以降はスタックに値が格納される．戻り値は， RAX レジスタに保持される．ただし，syscall 命令が RCX に戻りアドレスを保存するため，RCX に与えられた第 1 引数は，システムコールのスタブ内で R10 レジスタに退避される．syscall 命令および sysret 命令は RSP レジスタを変更しないため，システムコールのエントリポイントにおける RSP レジスタはユーザモードのスタックを示している．以上から，syscall フックでは，システムコールの第 1 引数から第 4 引数は，R10，RDX，R8，R9 レジスタ，第 5 引数以降はスタックより取得が可能である．

sysret フックでは，上記レジスタの値は既に別の値となっているため，syscall フック時に上記レジスタの値を保存しておくことで，第 1 引数から第 4 引数を補完する．第 5 引数以降は syscall フック時と同様にスタックより取得が可能である．また，sysret フック時には，RAX レジスタからシステムコールの戻り値も取得する．

ただし，引数には，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，OS 領域やプロセス領域内にあるそれらのデータ構造を解釈し，必要な情報を取得する．当該機能については，監視対象とするシステムコール個別の処理や引数に与えられるデータ構造ごとに Windows 10 x64 向けの実装が必要となる．

5 WOW64 向けの拡張機能

Windows 10 x64 と Windows 7 x64 では，WOW64 システムコールにおいて，システムコール番号や処理に差異が確認された．したがって，Alkanet10 ではこの変更点に対応するために一部の機能を拡張した．以下，本論文では，WOW64 内の ntdll.dll を，通常の 64 ビットアプリケーションが利用する ntdll.dll と区別するため，シンボル情報で示されている wntdll.dll と呼称する．また，通常の 64 ビットアプリケーションの動作環境を WOW64 と区別し，Win64 と呼称する．本章では，WOW64 システムコールの処理の流れと，発行されたシステムコールをトレースする方法を述べる．

5.1 WOW64 システムコールの流れ

WOW64 は，x64 版 Windows において 32 ビットアプリケーションを動作可能にする．32 ビットアプリケーションを動作させるプロセス (以下，WOW64 プロセス) は，はじめに wow64.dll，wow64cpu.dll，wow64win.dll，wntdll.dll などの DLL をロードする．WOW64 プロセスがシステムコールを発行するとき，上記の DLL が CPU の動作モードを x86 モードから x64 モードへ切り替え，システムコールをエミュレーションする関数を実行する．Windows 7 x64 では，wntdll.dll 内のスタブが ECX レジスタにセットした値を基にエミュレーション用関数が決定されていた．一方，Windows 10 x64 では，システムコール番号の 32 ビットのうち，使用されていなかった上位 16 ビットの値が利用される．

エミュレーション用の関数は多数存在するが，WOW64 レイヤからシステムコールの発行に至るまでの流れは，図 3 に示すように，Win64 側のスタブを経由するケース (1) と経由しないケース (2) の 2 通りに大別される．ケース (1) の場合，エミュレーション用の関数内でシステムコール番号に対応した Win64 側のスタブが呼出され，64

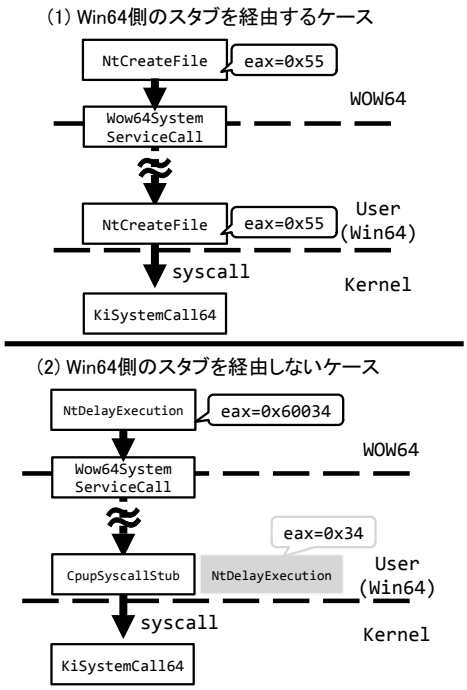

図 3: WOW64 からのシステムコール

ビットアプリケーションと同様にシステムコールに至る．図 3 のケース (1) に，NtCreateFile システムコールの例を示す．WOW64 上にて当該システムコールが発行されると，最終的に ntdll.dll 内の NtCreateFile スタブが実行され，通常と同様にシステムコールが発行される．

一方，ケース (2) は，Win64 側のスタブが呼出されず，wow64cpu.dll 内の CpupSyscallStub を経由してシステムコールに至る．図 3 のケース (2) は， NtDelayExecution システムコールの例を示す．wntdll.dll の NtDelayExecution スタブにより，0x60034 が EAX に セットされる．WOW64 レイヤでは，0x6 に対応するエミュレーション関数が実行され，その後，CpupSyscallStub から syscall が実行される．ntdll.dll 内の NtDelayExecution スタブは実行されない．

5.2 WOW64 システムコールのトレース

前述の 2 つのケースは，ともに syscall 命令でシステムコールを発行する．そのため，WOW64 システムコールのフックは，通常のシステムコールと同様に行える．また，システムコールの発行元プロセスが WOW64 プロセスであることは，プロセスオブジェクトを確認することで判定できる．これにより，WOW64 システムコールと通常のシステムコールを区別できる．

前述のケースのうち，ケース (1) のものは，通常のシステムコールと同様に情報の取得が可能である．一方，ケース (2) では，Win64 側のスタブを経由しないため，システムコール番号の上位 16 ビットが設定されたままシステムコールに至る．そこで，Alkanet10 でも取得したシステムコール番号の上位 16 ビットは WOW64 で行われたエミュレーション方法の特定に利用し，下位 16 ビットを本来のシステムコール番号として取り扱う．

また，sysret 命令の戻り先である CpupSyscallStub には，システムコール番号を EAX レジスタに入れる mov 命令がない．そのため，sysret フック時において，4.2 節で述べた方法 (b) ではシステムコール番号を得ることができない．そこで，スタックに積まれている WOW64 への戻りアドレスを取得し，WOW64 側のスタブから番号を取得する．ただし，エミュレーションの過程で複数回システムコールが呼ばれる場合もあるため，4.2 節の方法 (a) により得られたシステムコール番号を優先して用いる．なお，引数は，エミュレーション関数によって通常のシステムコールと同様にカーネルに受け渡される．したがって，通常のシステムコールと同様に取得が可能である．

6 機能評価

6.1 評価目的と検体

本手法の実現可能性および有効性を検証するために，Windows 10 x64 環境で発行されるシステムコールをトレースするプロトタイプを作成して評価を行った．評価用検体として，MWS Datasets 2015 [7] に含まれる D3M2014 に活動が記録されているマルウェアを用いた．当該検体は，複数のアンチウイルスで Fareit として検出された．事前に当該検体を AlkanetXP で解析したところ，Temp フォルダに bat ファイルをドロップし，cmd.exe を起動して読み込ませる挙動が観測された．本評価では， Windows

10 x64 上で当該検体を実行し，プロトタイプで上記の挙動が観測できることを確認する．これにより，Windows 10 x64 環境を対象としたシステムコールトレーサが実現できていることを確認する．なお，当該検体は，32 ビットアプリケーションであるため，WOW64 上で動作する．

6.2 ログエントリ

図 4 は，前述の検体を malware.exe という名前で実行し，プロトタイプにより取得したログの一部である．ログの最初に記載されている番号は記録されたログの通し番号である．`[ENTER]`，`[EXIT]` は，それぞれ syscall フック時のログ，sysret フック時のログであることを示す．`(fcc.f24) malware.exe` は，プロセス ID，スレッド ID，プロセス名を示す．

その後には，システムコールの名前と番号が続く．図 6 中では，NtCreateFile，NtWriteFile，NtCreateUserProcess の 3 つのシステムコールについて示している．それぞれ，ファイルの作成，指定したファイルへの書込み，プロセスと初期スレッドの生成を行うシステムコールである．

`Args` は，取得した引数の値を示す．`RetVal` は，戻り値を示し，sysret フック時のログのみに記載される．上記の 3 つのシステムコールの戻り値は，システムコールの成否を示す NTSTATUS である．図 6 中のログの戻り値は，全て `RetVal=0`，すなわち STATUS_SUCCESS であり，システムコールが成功したことを示している．なお，ログの通し番号を除き，ログ中の数値はすべて 16 進数で表記されている．

本評価で用いたプロトタイプには，NtCreateFile，NtCreateUserProcess システムコールの引数を解析する機能を追加している．NtCreateFile システムコールでは，第 3 引数 ObjectAttributes に含まれる ObjectName からファイルのパスを取得し，ログに追記している．また，第 1 引数 FileHandle が示す作成されたファイルハンドルを sysret 時のログに追記している．NtCreateUserProcess システムコールでは，第 9 引数である ProcessParameters に含まれる ImagePathName と CommandLine を追記している．これらは，実行されるファイルパスとコマンドライン引数を示す．また，第 1 引数 ProcessHandle および第 2 引数 ThreadHandle から，それぞれ生成されたプロセスオブジェクトとスレッドオブジェクトのハンドルの値を取得し，sysret 時のログに追記している．

6.3 評価結果

図 4 の (1) のログ番号 295423 と (2) の 295429 の NtCreateFile システムコールのログより，malware.exe は，Temp フォルダに 315421.bat を作成し，そのファイルハンドルとして，0x374 という値を得たことを示している．続く (3) のログ番号 295432 および (4) の 295436 の NtWriteFile システムコールのログでは，第 1 引数に 0x374 が与えられている．NtWriteFile システムコールの第 1 引数は，書き込むファイルを示すファイルハンドルである．すなわち，当該ログは，malware.exe が Temp フォルダに作成した 315421.bat に書込みを行ったことを示している．さらに，(5) のログ番号 312002 ，(6) の 312011 のログより，malware.exe は， NtCreateUserProcess システムコールを用いて cmd.exe を起動したことがわかる．また， CommandLine から 315421.bat が引数として与えられていることも確認できる．ProcessHandle や ThreadHandle も取得できているため，ハンドルを解析してオブジェクトを取得する機能を追加すれば，生成されたプロセスやスレッドの情報も収集することが可能となる．

以上から，確認項目であったマルウェアの動作をログより確認することができた．したがって，本プロトタイプは，Windows 10 x64 のシステムコールトレーサの機能を有しているといえる．今後，ハンドルを解析する機能や他のシステムコールについても追加情報を取得する機能を実現することで，実用的なマルウェア解析システムが実現できると考えられる．

7 関連研究

メモリフォレンジックフレームワークである Rekall[8] は，メモリダンプを解析し，マルウェ

```
295423: [ENTER] (fcc.f24)      malware.exe NtCreateFile(55):
(1)   Args=[9e6b8, c0100080, 9ef80, 9e6d0, 0, 0, 3, 5, 60, 0, 0]
      ObjectName="\??\C:\Users\yotuki\AppData\Local\Temp\315421.bat"
295429: [ EXIT] (fcc.f24)      malware.exe NtCreateFile(55):
(2)   Args=[9e6b8, c0100080, 9ef80, 9e6d0, 0, 0, 3, 5, 60, 0, 0], RetVal=0
      ObjectName="\??\C:\Users\yotuki\AppData\Local\Temp\315421.bat", FileHandle=374
295432: [ENTER] (fcc.f24)      malware.exe NtWriteFile(1a0008):
(3)   Args=[374, 0, 0, 0, 9f028, 4174c7, 5e, 0, 0]
295436: [ EXIT] (fcc.f24)      malware.exe NtWriteFile(1a0008):
(4)   Args=[374, 0, 0, 0, 9f028, 4174c7, 5e, 0, 0], RetVal=0
```

```
312002: [ENTER] (fcc.f24)      malware.exe NtCreateUserProcess(bc):
(5)   Args=[9e568, 9e570, 2000000, 2000000, 0, 0, 47004500000000, 1, 1e2620, 9e680, 9ee80]
      ImagePathName="C:\Windows\SysWOW64\cmd.exe",
      CommandLine="C:\Windows\system32\cmd.exe /c ""C:\Users\yotuki\AppData\Local\Temp\315421.bat" (略)""
312011: [ EXIT] (fcc.f24)      malware.exe NtCreateUserProcess(bc):
      Args=[9e568, 9e570, 2000000, 2000000, 0, 0, 47004500000000, 1, 1e2620, 9e680, 9ee80], RetVal=0
(6)   ImagePathName="C:\Windows\SysWOW64\cmd.exe",
      CommandLine="C:\Windows\system32\cmd.exe /c ""C:\Users\yotuki\AppData\Local\Temp\315421.bat" (略)"",
      ProcessHandle=468, ThreadHandle=474
```

図 4: bat ファイルのドロップと cmd.exe の起動のログ

アが感染した痕跡などを明らかにするツールである．Rekall は，Alkanet と同様に，事前にシンボル情報を解析してプロファイルを作成することで，多様な OS に対応している．バージョン 1.4.0 では，多くのプラグインが Windows 10 にも対応済みであると発表している [9]．Rekall は，あくまでメモリフォレンジックを目的としており，マルウェアの挙動を観測し，記録することは目的としていない．Alkanet は，システムコールによりマルウェアの挙動を観測する動的解析システムを実現することを目的としている．

8 おわりに

本論文では，Windows 10 x64 環境を対象とするシステムコールトレーサを実現する方法を述べた．システムコールの処理の多くは，x64 版 Windows 共通であるため，Windows 7 x64 向けの実装と同様の手法が Windows 10 x64 でも利用できた．Windows 10 x64 で発行されるシステムコールをトレースするプロトタイプを実装し，実際に発行されたシステムコールの情報が取得できることを確認した．今後は，リリース版の Windows 10 x64 での検証や，システムコールの引数を解析する機能を追加し，より実用性の高いマルウェア解析システムを目指す．

参考文献

[1] 大月 他：マルウェア観測のための仮想計算機モニタを用いたシステムコールトレース手法，情報処理学会論文誌，Vol. 55, No. 9, pp. 2034–2046 (2014).

[2] Net Applications.com: Operating system market share, Net Applications.com, http://www.netmarketshare.com/operating-system-market-share.aspx?qprid=11&qpcustomb=0&qpsp=183&qpnp=14&qptimeframe=M (accessed 2015-06-03).

[3] トレンドマイクロ株式会社：標的型サイバー攻撃分析レポート 2015 年版〜「気付けない攻撃」の高度化が進む〜 (2015).

[4] Shinagawa, T. et al.: BitVisor: a thin hypervisor for enforcing i/o device security, Proc. VEE '09, ACM, pp. 121–130 (2009).

[5] Russinovich, M. E. et al.: インサイド Microsoft Windows 第 6 版 上，日経 BP 社 (2012).

[6] Microsoft: x64 Software Conventions, Microsoft, https://msdn.microsoft.com/en-us/library/7kcdt6fy.aspx (accessed 2015-06-03).

[7] 神薗 他：マルウェア対策のための研究用データセット〜MWS Datasets 2015〜，研究報告コンピュータセキュリティ（CSEC），Vol. 2015-CSEC-70, No. 6, pp. 1–8 (2015).

[8] Google Inc.: Rekall Memory Forensic Framework, Google Inc., http://www.rekall-forensic.com/ (accessed 2015-08-15).

[9] Google Inc.: Release Release 1.4.0 Etzel google/rekall GitHub, GitHub Inc., https://github.com/google/rekall/releases/tag/v1.4.0 (accessed 2015-08-15).